**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**

**ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ**

**ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΑΤΡΙΚΗ**

**Κατεύθυνση: «Πληροφορική με Εφαρμογές στην Ασφάλεια, Διαχείριση Μεγάλου Όγκου Δεδομένων και Προσομοίωση»**

# “Επιτάχυνση σε Υλικό Αλγορίθμων Βελτιστοποίησης Λεπτομερής Τοποθέτησης Υλικού με Υλοποίηση σε FPGA”

**Καριώτης Κοσμάς**

## ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**Επιβλέπων Καθηγητής**

**Δαδαλιάρης Αντώνιος**

**Λαμία, 2023**


**UNIVERSITY OF THESSALY**

**SCHOOL OF SCIENCE**

**INTERDEPARTMENTAL POSTGRADUATE PROGRAM**

**INFORMATICS AND COMPUTATIONAL BIOMEDICINE**

**Direction: «Informatics with Applications in Security, Big Data Management and Simulation»**

# "Hardware Acceleration of Detailed Placement Optimization Algorithms with FPGA implementation"

**Kariotis Kosmas**

## MASTER THESIS

**Supervisor**

**Dadaliaris Antonios**

**Lamia, 2023**

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ.), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(Υπογραφή)

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

## Σύνοψη

Η παρούσα διατριβή περιγράφει την εφαρμογή της βελτιστοποίησης της τοποθέτησης στο στάδιο της λεπτομερούς τοποθέτησης της φάσης φυσικού σχεδιασμού στη διαδικασία σχεδίασης υλικού, εξηγώντας πρώτα το απαραίτητο θεωρητικό υπόβαθρο της θεωρίας σχεδίασης υλικού.

Στη συνέχεια καλύπτονται οι μετρικές βελτιστοποίησης και οι αλγόριθμοι βελτιστοποίησης με ιδιαίτερη έμφαση να δίνετε στην μετρική προσέγγισης μήκους καλωδίου HPWL, ενώ ακολουθεί μια συνοπτική παρουσίαση του τρόπου απεικόνισης και αποθήκευσης των τοποθετήσεων με τη χρήση της μορφής Bookshelf.

Η βελτιστοποίηση της τοποθέτησης πραγματοποιείται και υλοποιείται έπειτα αρχικά σε μοντέλο λογισμικού το οποίο έχει σχεδιαστεί με τρόπο τέτοιο που επιτρέπει τη πλήρη συνεργασία με τη σχεδίαση του υλικού.

Στο τέλος της εργασίας καλύπτεται εκτενώς η υλοποίηση υλικού, συμπεριλαμβανομένης της προσομοίωσης της και της σύνθεσης και επαλήθευσης της λειτουργικότητας της σχεδίασης με χρήση πλακετών FPGA.

## Abstract

This thesis describes the application of placement optimization to the detailed placement stage of the physical design phase in the hardware design process, by first explaining the necessary theoretical background of hardware design theory.

The optimization metrics and optimization algorithms are then covered with particular emphasis on the HPWL wire length approximation metric, followed by a brief presentation of how to visualize and store placements using the Bookshelf format.

The placement optimization is then first performed and implemented in a software model that is designed in such a way that it allows full cooperation with the hardware design.

At the end of the work, the hardware implementation is extensively covered, including its simulation and the synthesis and verification of the functionality of the design using FPGA boards.

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**

**ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ**

**ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΑΤΡΙΚΗ**

# Επιτάχυνση σε Υλικό Αλγορίθμων Βελτιστοποίησης Λεπτομερής Τοποθέτησης Υλικού με Υλοποίηση σε FPGA

**Καριώτης Κοσμάς**

## Περιεχόμενα

# Κατάλογος Σχημάτων

## Κατάλογος Εικόνων



## Κατάλογος Πινάκων

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**

**ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ**

**ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΑΤΡΙΚΗ**

# Επιτάχυνση σε Υλικό Αλγορίθμων Βελτιστοποίησης Λεπτομερής Τοποθέτησης Υλικού με Υλοποίηση σε FPGA

**Καριώτης Κοσμάς**

## 1. Εισαγωγή

Τις τελευταίες δεκαετίες, ο σύγχρονος σχεδιασμός τσιπ έχει καταστεί τόσο περίπλοκος που σε μεγάλο βαθμό η συνολική διαδικασία σχεδιασμού υλικού γίνεται με τη βοήθεια εξειδικευμένου λογισμικού. Το σύνολο των λογισμικών αυτών υλοποιείται από τη λεγόμενη βιομηχανία αυτοματοποίησης ηλεκτρονικού σχεδιασμού (*electronic design automation - EDA*).

Στη σχεδίαση ψηφιακών κυκλωμάτων πολύ μεγάλης κλίμακας ολοκλήρωσης (very large-scale integration - VLSI) η ανάγκη για αυτοματοποίηση είναι φυσικά κατά πολύ μεγαλύτερη από ότι στα αναλογικά κυκλώματα [1].

Η παρούσα διατριβή καλύπτει την αυτοματοποίηση της τοποθέτησης και πιο συγκεκριμένα τη βελτίωση της προκύπτουσας τοποθέτησης με κύριο στόχο τη μείωση του συνολικού μήκους καλωδίων.

Προκειμένου να καταστεί δυνατή η κατανόηση της διαδικασίας βελτιστοποίησης μιας τοποθέτησης, είναι απαραίτητο να καλυφθεί πρώτα η σχετική θεωρία σχεδιασμού και τοποθέτησης υλικού.

Καλύπτοντας τη θεωρητική βάση, θα μπορεί στη συνέχεια να πραγματοποιηθεί συζήτηση σχετικά με τις μετρικές που χρησιμοποιούνται για τον υπολογισμό της βελτίωσης, καθώς και οι αλγόριθμοι που εφαρμόζονται για τον σκοπό της βελτιστοποίησης.

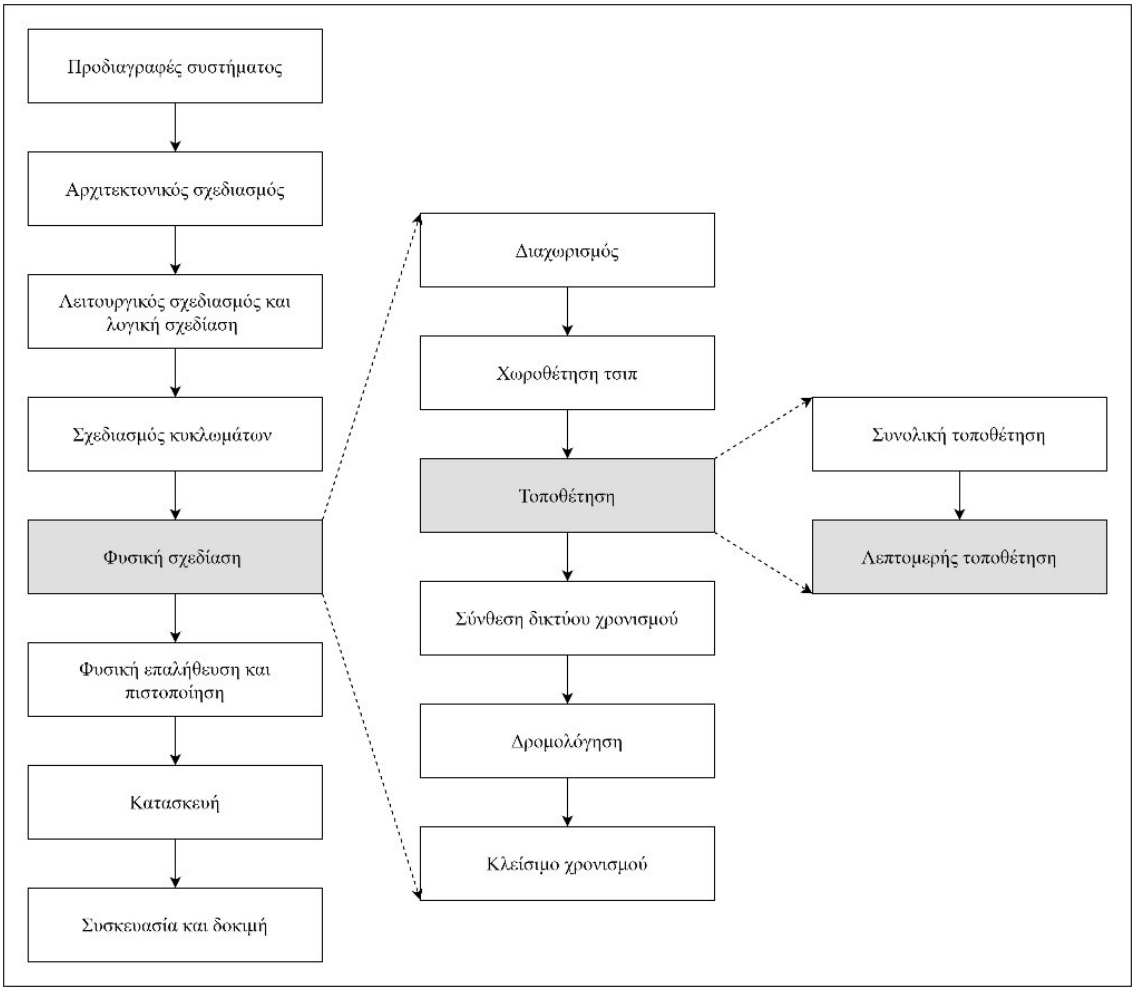
Ασφαλώς οι τοποθετήσεις αποτελούνται από πολλά δομικά στοιχεία και έχει μεγάλη σημασία η κάλυψη του τρόπου αναπαράστασης και αποθήκευσης των σημαντικών για τον σκοπό της βελτιστοποίησης δεδομένων σε κατάλληλα δομημένα αρχεία, δομές δεδομένων λογισμικού, καθώς και σε προσπελάσιμες μνήμες στο υλικό.

Παρόλο που ο κύριος στόχος της διατριβής είναι η υλοποίηση σε υλικό, είναι απαραίτητο να εφαρμοστεί και να υλοποιηθεί η βελτιστοποίηση και σε λογισμικό. Το μοντέλο λογισμικού συμπληρώνει το μοντέλο υλικού και χρησιμοποιείται πάντοτε παράλληλα με αυτό.

Έτσι, η επαλήθευση της σχεδίασης υλικού μπορεί φυσικά να γίνει μέσω εργαλείων προσομοίωσης, αλλά με βάση τη σχεδίαση λογισμικού είναι επίσης δυνατή η πραγματοποίηση της εκτέλεσης και της επαλήθευσης της λειτουργικότητας με τη χρήση πλακετών FPGA και κατάλληλη σειριακή επικοινωνία μεταξύ του λογισμικού και του FPGA.

## 2. Θεωρία Σχεδίασης Υλικού

Η διαδικασία σχεδιασμού υλικού ξεκινά από ένα επίπεδο υψηλότερης αφαιρετικότητας, ενώ στα μεταγενέστερα στάδια τα εργαλεία και οι αλγόριθμοι λειτουργούν με περισσότερο αναλυτικότερες πληροφορίες για κάθε κύκλωμα, οι οποίες περιλαμβάνουν το ακριβές γεωμετρικό σχήμα και τις ηλεκτρικές ιδιότητες που διαθέτει το κάθε στοιχείο [1].




*Σχήμα 2-1 Βήματα ροής σχεδιασμού υλικού με έμφαση στο στάδιο φυσικής σχεδίασης και τα βήματα τους σταδίου τοποθέτησης*

Τα βήματα της ροής σχεδιασμού VLSI έχουν ως εξής [1]:

1. **Προδιαγραφές συστήματος** (*System specification*): Καθορισμός των απαιτήσεων υψηλού επιπέδου του συστήματος, οι οποίες περιλαμβάνουν τη λειτουργικότητα, τις επιδόσεις, τις φυσικές διαστάσεις καθώς και την τεχνολογία παραγωγής.

2. **Αρχιτεκτονικός σχεδιασμός** (*Architectural design*): Καθορισμός βασικής αρχιτεκτονικής που πληροί τις προδιαγραφές του συστήματος

3. **Λειτουργικός σχεδιασμός και λογική σχεδίαση** (*Functional design and logic design*): Ορισμός της λειτουργικότητας και της συνδεσμολογίας (σύνολο εισόδων, εξόδων) κάθε δομικού στοιχείου της αρχιτεκτονικής.

   Η λογική σχεδίαση πραγματοποιείται σε επίπεδο μεταφοράς καταχωρητών (*register-transfer level - RTL*) με τη χρήση γλώσσας περιγραφής υλικού (*hardware description language - HDL*) μέσω προγραμμάτων που καθορίζουν τη λειτουργική και χρονική συμπεριφορά ενός τσιπ. Οι κοινές HDL περιλαμβάνουν τις γλώσσες Verilog και VHDL.

   Τα εργαλεία λογικής σύνθεσης μετατρέπουν την HDL και την αντιστοιχούν σε λίστα καλωδίων σημάτων (*netlist*) και συγκεκριμένα στοιχεία κυκλώματος, όπως τυποποιημένα κελιά και τρανζίστορ.

4. **Σχεδιασμός κυκλωμάτων** (*Circuit design*): Μέσω των εργαλείων λογικής σύνθεσης το μεγαλύτερο μέρος της ψηφιακής λογικής μετατρέπεται από εκφράσεις Boole σε μια λίστα καλωδίων (*netlist*) σε επίπεδο πύλης. Ωστόσο, υπάρχουν και ορισμένα στοιχεία χαμηλού επιπέδου τα οποία πρέπει να σχεδιαστούν απευθείας σε επίπεδο τρανζίστορ.

   Σε αυτό το σημείο υπεισέρχεται ο τομέας της σχεδίασης υλικού, γνωστός ως σχεδιασμός κυκλωμάτων, ο οποίος εξασφαλίζει την ορθότητα και επαληθευσιμότητα μιας σχεδίασης υλικού σε επίπεδο κυκλώματος από εργαλεία προσομοίωσης κυκλωμάτων όπως το SPICE.

5. **Φυσική σχεδίαση** (*Physical design*): Σε αυτό το στάδιο, όλα τα δομικά στοιχεία του σχεδιασμού διαμορφώνονται με γεωμετρικές αναπαραστάσεις σταθερών σχημάτων και μεγεθών, ανατίθενται σε θέσεις στο χώρο, ενώ πραγματοποιούνται και οι κατάλληλες διασυνδέσεις δρομολόγησης.

   Ο φυσικός σχεδιασμός επηρεάζει άμεσα την απόδοση, την επιφάνεια, την αξιοπιστία, την κατανάλωση ενέργειας και την αποδοτικότητα της κατασκευής ενός ψηφιακού κυκλώματος.

   Εφόσον αποτελεί ένα ιδιαίτερα σημαντικό και εξίσου πολύπλοκο βήμα του σχεδιασμού, είναι απολύτως λογικό πως το στάδιο αυτό χωρίζεται σε επιμέρους στάδια, τα οποία θα εξεταστούν αναλυτικά στην επόμενη ενότητα.

6. **Φυσική επαλήθευση και πιστοποίηση** (*Physical verification and signoff*): Αφού ολοκληρωθεί ο φυσικός σχεδιασμός, η διάταξη πρέπει να επαληθευτεί πλήρως για να εξασφαλιστεί η σωστή λειτουργία σε ηλεκτρικό και λογικό επίπεδο.

7. **Κατασκευή** (*Fabrication*): Ο σχεδιασμός αναπαρίσταται σε μορφή κατάλληλη για κατασκευή και αποστέλλεται σε ειδικό χυτήριο πυριτίου, όπου το ολοκληρωμένο κύκλωμα κατασκευάζεται σε στρογγυλά πλακίδια πυριτίου, τα οποία στη συνέχεια ελέγχονται και επισημαίνονται ως λειτουργικά ή ελαττωματικά, ενώ χωρίζονται τελικώς σε μικρότερα κομμάτια.

8. **Συσκευασία και δοκιμή** (*Packaging and testing*): Αφού διαχωριστούν, τα λειτουργικά τσιπ συσκευάζονται, σφραγίζονται και δοκιμάζονται εκτενώς για να διασφαλιστεί ότι πληρούν τις απαιτήσεις του σχεδιασμού.

## 2.1 Φυσική σχεδίαση

Το στάδιο φυσικής σχεδίασης της ροής σχεδιασμού υλικού διακρίνεται περαιτέρω στα ακόλουθα βήματα [1]:

1. **Διαχωρισμός** (*Partitioning*): Σε αυτό το στάδιο, το κύκλωμα διασπάται σε μικρότερα υποκυκλώματα ή μονάδες, τα οποία σχεδιάζονται ή αναλύονται ξεχωριστά.

2. **Χωροθέτηση τσιπ** (*Chip Planning*): Προσδιορισμός του σχήματος και της διάταξης των υποκυκλωμάτων καθώς και των θέσεων των εξωτερικών θυρών. Αυτό το στάδιο είναι επίσης συχνά συνυφασμένο με τη δρομολόγηση ισχύος και γείωσης, κατά την οποία οι καλωδιώσεις ισχύος και γείωσης κατανέμονται σε όλο το τσιπ.

3. **Τοποθέτηση** (*Placement*): Εύρεση των χωρικών θέσεων όλων των κελιών εντός του κάθε μπλοκ της σχεδίασης. Αυτό το βήμα διαχωρίζεται περαιτέρω στις φάσεις της συνολικής και της λεπτομερούς τοποθέτησης, οι οποίες θα αναλυθούν περαιτέρω σε επόμενη ενότητα.

4. **Σύνθεση δικτύου χρονισμού** (*Clock network synthesis*): Καθορισμός της προσωρινής αποθήκευσης (*buffering*) και δρομολόγησης του σήματος ρολογιού, με κεντρικό γνώμονα την ικανοποίηση των απαιτήσεων καθυστέρησης. Σε αυτό το βήμα περιλαμβάνεται επίσης και η διαχείριση ισχύος.

5. **Δρομολόγηση** (*Routing*): Κατανομή πόρων δρομολόγησης που προορίζονται για συνδέσεις, συμπεριλαμβανομένων των γραμμών δρομολόγησης στα κανάλια και τα πλαίσια μεταγωγής, ακολουθούμενη από την ανάθεση δρομολόγησης σε συγκεκριμένα μεταλλικά στρώματα και γραμμές δρομολόγησης. Το πρώτο μέρος της διαδικασίας δρομολόγησης είναι γνωστό ως συνολική δρομολόγηση (*global routing*), ενώ το δεύτερο ως λεπτομερής δρομολόγηση (*detailed routing*).

6. **Κλείσιμο χρονισμού** (*Timing closure*): Βελτιστοποίηση της απόδοσης του κυκλώματος με την εφαρμογή εξειδικευμένων τεχνικών τοποθέτησης και δρομολόγησης.

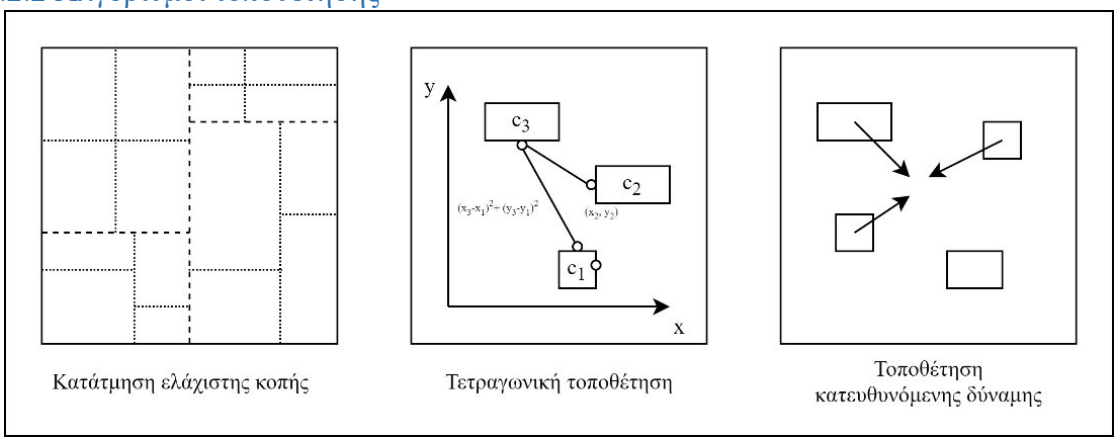
## 2.2 Τοποθέτηση υλικού

Αφού το κύκλωμα διαμεριστεί σε μικρότερα κυκλώματα και σχεδιασθεί η χωροθέτηση του τσιπ, η διαδικασία τοποθέτησης επιδιώκει να καθορίσει τις θέσεις των τυπικών κελιών μέσα σε κάθε μπλοκ, ικανοποιώντας παράλληλα στόχους βελτιστοποίησης. Καθώς τα περιγράμματα των μπλοκ και οι θέσεις των ακροδεκτών έχουν προσδιοριστεί κατά τα προηγούμενα στάδια, η τοποθέτηση ξεκινά με την ανάθεση γενικών θέσεων στα μετακινήσιμα κελιά, ενώ στη συνέχεια οι θέσεις αυτές επαναπροσδιορίζονται σε νόμιμες τοποθεσίες κελιών και επιβάλλονται περιορισμοί μην επικάλυψης μεταξύ των κελιών [1].

### 2.2.1 Συνολική τοποθέτηση

Κατά τη διάρκεια της συνολικής τοποθέτησης συχνά αμελούνται τα συγκεκριμένα σχήματα και μεγέθη των τοποθετούμενων κελιών. Η συνολική τοποθέτηση δεν επιχειρεί επίσης να παρατάξει τις θέσεις των κελιών σε ένα έγκυρο πλέγμα γραμμών και στηλών. Επιτρέπεται και ορισμένη επικάλυψη μεταξύ των τοποθετημένων κελιών, καθώς η έμφαση αυτού του βήματος σχεδιασμού δίνεται στον συνολικό προσδιορισμό θέσης και στη συνολική κατανομή της πυκνότητας [1].

### 2.2.2 Αλγόριθμοι τοποθέτησης




*Σχήμα 2-2 Παραδείγματα αλγορίθμων συνολικής τοποθέτησης*

Η συνολική τοποθέτηση μπορεί να υλοποιηθεί με ποικίλες τεχνικές. Στις κοινές τεχνικές και τους κοινούς αλγορίθμους συνολικής τοποθέτησης συγκαταλέγονται [1]:

1. **Κατάτμηση ελάχιστης κοπής** (*Min-Cut Partitioning*): Η κατάτμηση ελάχιστης περικοπής αποτελεί ένα εξαιρετικό παράδειγμα αλγορίθμου τοποθέτησης με βάση την κατάτμηση (*partitioning-based placement*).

   Οι εν λόγω αλγόριθμοι καταμερίζουν την διάταξη σε υπο-περιοχές σύμφωνα με μια συνάρτηση κόστους που βασίζεται στην κοπή. Η διαδικασία αυτή επαναλαμβάνεται έως ότου κάθε υπο-περιοχή είναι αρκετά μικρή ώστε να μπορεί να διαχειριστεί με τον πιο βέλτιστο τρόπο. Αξίζει να σημειωθεί πως κάθε υπο-περιοχή διαθέτει επίσης την δικιά της επιμέρους λίστα καλωδίων (*netlist*).

2. **Αναλυτικές τεχνικές** (*Analytic techniques*): Οι αλγόριθμοι τοποθέτησης που βασίζονται σε τεχνικές ανάλυσης βασίζονται σε μια αντικειμενική συνάρτηση κόστους, η οποία μπορεί να μεγιστοποιηθεί ή να ελαχιστοποιηθεί μέσω μαθηματικής ανάλυσης. Ο αντικειμενικός στόχος μπορεί να είναι τετραγωνικός ή μη κυρτός, επιτρέποντας την εφαρμογή αλγορίθμων όπως αυτοί που αναφέρονται παρακάτω.
   a. **Τετραγωνική τοποθέτηση** (*Quadratic placement*): Κατά το πρώτο της στάδιο, το οποίο περιλαμβάνεται στη συνολική τοποθέτηση, η τετραγωνική τοποθέτηση στοχεύει στην ελαχιστοποίηση του τετραγώνου της Ευκλείδειας απόστασης μεταξύ όλων των συνδεμένων κελιών:

   $$\sum (x_i - y_j)^2 + (y_i - y_j)^2$$

   b. **Τοποθέτηση κατευθυνόμενης δύναμης** (*Force-directed placement*): Η τοποθέτηση που βασίζεται σε δυνάμεις ορίζει μια δύναμη έλξης η οποία ασκείται από κάθε κελί της σχεδίασης σε κάθε άλλο κελί με το οποίο αυτό συνδέεται μέσω καλωδίου.

      Ο κύριος στόχος του αλγορίθμου είναι η ελαχιστοποίηση του αθροίσματος αυτών των δυνάμεων για κάθε κελί:

      $$\sum \vec{F}_{ij}$$

3. **Προσομοιωμένη ανόπτηση** (*Simulated annealing*): Οι στοχαστικοί αλγόριθμοι (*stochastic algorithms*), όπως η προσομοιωμένη ανόπτηση εφαρμόζουν τυχαίες κινήσεις προκειμένου να βελτιστοποιήσουν μια συνάρτηση κόστους.

   Μετά την εκτέλεση των τυχαίων κινήσεων η νέα τοποθέτηση συγκρίνεται με την αρχική και γίνετε αποδεκτή εάν υπήρξε βελτίωση στη συνάρτηση κόστους.

   Η διαδικασία ανόπτησης σταματά μόλις επιτευχθεί η λεγόμενη ισορροπία. Ισορροπία επιτυγχάνεται όταν πραγματοποιηθεί ένας προκαθορισμένος αριθμός προσπαθειών μετακίνησης ή η συνάρτηση κόστους είναι πλέον χαμηλότερη από μια συγκεκριμένη προκαθορισμένη ελάχιστη τιμή στόχο.

### 2.2.3 Νομιμοποίηση σχεδιασμού

Μετά τη συνολική τοποθέτηση και πριν ή κατά τη διάρκεια της λεπτομερούς τοποθέτησης επέρχεται νομιμοποίηση. Η νομιμοποίηση επιδιώκει την ευθυγράμμιση των τοποθετούμενων κελιών σε γραμμές και στήλες και αφαιρεί την επικάλυψη, ενώ παράλληλα προσπαθεί να ελαχιστοποιήσει τις μετατοπίσεις από τις συνολικές θέσεις τοποθέτησης και τις επιπτώσεις στο μήκος των διασυνδέσεων και την καθυστέρηση του κυκλώματος. Η νομιμοποίηση θα πρέπει να εφαρμόζεται μετά από κάθε βαθμιαία αλλαγή κατά τη διάρκεια της φυσικής σύνθεσης, όπως είναι η τροποποίηση των διαστάσεων των κελιών [1].

### 2.2.4 Λεπτομερής τοποθέτηση

Αφού νομιμοποιηθεί η τοποθέτηση, η νομιμοποιημένη πλέον τοποθέτηση βελτιστοποιείται σε σχέση με συγκεκριμένους στόχους μέσω τεχνικών λεπτομερούς τοποθέτησης. Στόχος της βελτιστοποίησης είναι συνήθως η μείωση του συνολικού μήκους καλωδίων μέσω της ανταλλαγής κελιών ή της μετατόπισης των κελιών κατά μήκος των σειρών, όταν υπάρχει διαθέσιμος αχρησιμοποίητος χώρος, αλλά υπάρχουν επίσης και αλγόριθμοι λεπτομερούς τοποθέτησης που στοχεύουν κατά κύριο λόγο στη δρομολογησιμότητα της σχεδίασης. Οι διαδικασίες της νομιμοποίησης και της λεπτομερούς τοποθέτησης συνδυάζονται συνήθως κάτω από μια ενιαία υλοποίηση λογισμικού [1].

# 3. Βελτιστοποίηση Τοποθέτησης

Οι σχεδιάσεις υλικού δεν πρέπει φυσικά να είναι μόνο νομιμοποιημένες και δρομολογήσιμες. Ένας από τους πρωταρχικούς στόχους της λεπτομερής τοποθέτησης υλικού είναι και η δημιουργία βελτιστοποιημένων τοποθετήσεων. Βελτιστοποιήσεις στην τοποθέτηση του υλικού εφαρμόζονται σε μια πληθώρα σχεδιαστικών παραμέτρων.

Η βελτιστοποίηση τοποθέτησης μπορεί να πραγματοποιηθεί σε όλα τα στάδια της τοποθέτησης. Δεδομένου όμως ότι κατά τα πρώιμα στάδια της συνολικής τοποθέτησης η εκτίμηση του χρονισμού και της καθυστέρησης του σήματος είναι εξαιρετικά ανακριβής, οι βελτιστοποιήσεις εφαρμόζονται κατά κανόνα στα μεταγενέστερα στάδια ή μετά την ολοκλήρωση της διαδικασίας συνολικής τοποθέτησης [1].

Ο κύριος λόγος για τον οποίο η εκτίμηση της καθυστέρησης, καθώς και άλλων μετρικών βελτιστοποίησης της σχεδίασης, είναι ανακριβής κατά τη διαδικασία της συνολικής τοποθέτησης είναι η ανάθεση γενικών θέσεων στα κελιά της σχεδίασης. Όπως αναφέρθηκε και στην προηγουμένη ενότητα, με τη λεπτομερή τοποθέτηση, τα κελιά της σχεδίασης τοποθετούνται σε νόμιμες θέσεις κελιών και υπόκεινται σε αυστηρούς περιορισμούς σχεδίασης [1].

## 3.1 Μετρικές και στόχοι βελτιστοποίησης

Οι τοποθετήσεις υλικού είναι σκόπιμο να βελτιστοποιούνται όσον αφορά τις καθυστερήσεις των σημάτων κατά τη μετάδοσή τους. Εφόσον η καθυστέρηση διάδοσης ενός σήματος μέσω ενός καλωδίου συσχετίζεται άμεσα με το μήκος του καθαυτού καλωδίου, ένας από τους βασικούς στόχους των αλγορίθμων τοποθέτησης είναι πάντοτε η ελαχιστοποίηση του συνολικού μήκους καλωδίων (*total wirelength*).

Ο τρόπος με τον οποίο τοποθετείται μια σχεδίαση υλικού επηρεάζει σε μεγάλο βαθμό τα μήκη των καλωδίων, καθώς και την πυκνότητά τους. Η συμφόρηση στα καλώδια που οφείλεται στην υψηλή πυκνότητα σε ορισμένες περιοχές ενός σχεδιασμού είναι ανεπιθύμητη, καθώς αποτελεί σημαντική αιτία πρόκλησης διαφόρων αρνητικών φαινομένων, όπως ηλεκτρομαγνητικές παρεμβολές μεταξύ γειτονικών καλωδίων, ανομοιόμορφη θερμοκρασία στην επιφάνεια του τελικού κυκλώματος, καθώς και μείωση της αξιοπιστίας του [1].

### 3.1.1 Μήκος καλωδίου και απόσταση Μανχάταν

Το μήκος του καλωδίου προσεγγίζεται και δεν υπολογίζεται λεπτομερώς στους αλγορίθμους βελτιστοποίησης τοποθέτησης με στόχο τη μείωση του χρόνου εκτέλεσης και της πολυπλοκότητας των αλγορίθμων αυτών [1].

Σε ένα καλώδιο με δύο ακροδέκτες $P_1 = (x_1, y_1)$ και $P_2 = (x_2, y_2)$ το μήκος του καλωδίου προσεγγίζεται μέσω της λεγόμενης απόστασης Μανχάταν (*Manhattan distance*) $d_M$, η οποία υπολογίζεται ως εξής [2]:

$$d_M(P_1, P_2) = |x_2 - x_1| + |y_2 - y_1|$$

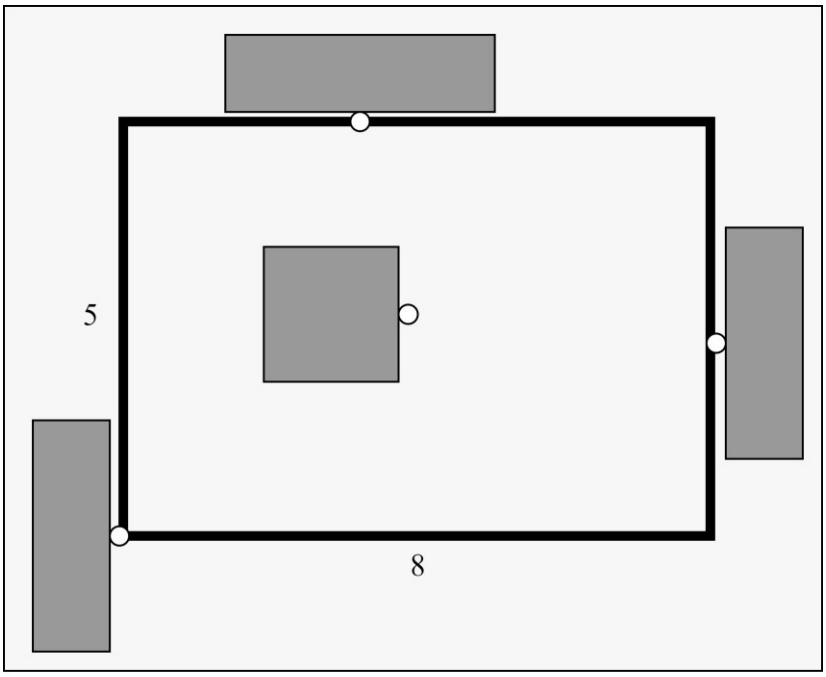
Η απόσταση Μανχάταν υπολογίζει την απόσταση μεταξύ των δύο σημείων (ακροδεκτών) κατά μήκος αξόνων με ορθή γωνία. Ως εκ τούτου, χρησιμοποιείται συχνά στους υπολογισμούς που αφορούν ολοκληρωμένα κυκλώματα, καθώς τα καλώδια τέτοιων σχεδιάσεων υλικού διατρέχουν μόνο παράλληλα τους άξονες $x$ ή $y$.

Με τη γενίκευση αυτού του ορισμού είναι δυνατόν να διατυπωθεί και να υπολογιστεί ένα προσεγγιστικό μήκος καλωδίου για καλώδια με οποιονδήποτε αριθμό ακροδεκτών.

### 3.1.2 Μήκος καλωδίου μισής περιμέτρου (HPWL)

Σε καλώδια πολλαπλών ακροδεκτών η πιο συχνά χρησιμοποιούμενη τεχνική προσέγγισης είναι το λεγόμενο μήκος καλωδίου μισής περιμέτρου (*half-perimeter wirelength - HPWL*).

Για ένα καλώδιο με πολλαπλούς ακροδέκτες ορίζουμε ένα νοητό πλαίσιο οριοθέτησης, το οποίο αποτελεί το μικρότερο ορθογώνιο, το οποίο περικλείει τις θέσεις των ακροδεκτών του καλωδίου. Το μήκος του καλωδίου προσεγγίζεται έπειτα ως το ήμισυ της περιμέτρου αυτού του πλαισίου οριοθέτησης [1].




*Σχήμα 3-1 Αναπαράσταση νοητού πλαισίου οριοθέτησης σε καλώδιο πολλαπλών ακροδεκτών και υπολογισμός μετρικής HPWL*

Στην πράξη, για να υπολογιστεί αυτή η μετρική, για κάθε καλώδιο μιας σχεδίασης πρέπει να υπολογιστεί μόνο η ελάχιστη και μέγιστη θέση των ακροδεκτών ανά άξονα. Έτσι, ο υπολογισμός του HPWL ενός καλωδίου απλοποιείται στον εξής υπολογισμό [3]:

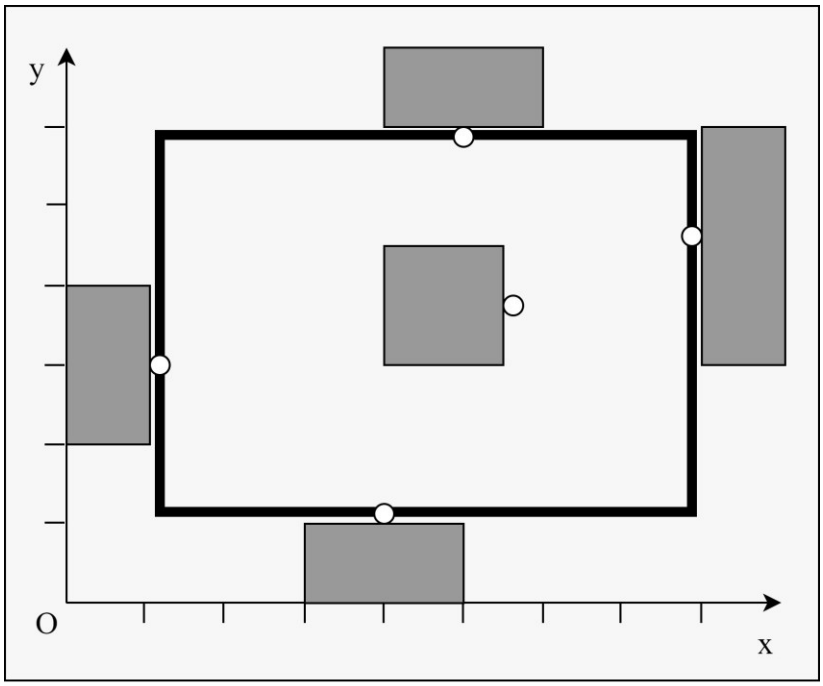
$$HPWL_{net} = (x_{max} - x_{min}) + (y_{max} - y_{min})$$

Αξίζει να σημειωθεί σε αυτό το σημείο, πως η μαθηματική εξίσωση που παρατέθηκε παραπάνω μοιάζει κατά πολύ με την εξίσωση απόστασης Μανχάταν που περιγράφηκε πρωτύτερα.

Το συνολικό μήκος των καλωδίων μιας τοποθέτησης υλικού προσεγγίζεται εύκολα με υπολογισμό του αθροίσματος της μετρικής HPWL κάθε καλωδίου.

Αν θεωρήσουμε πως μια σχεδίαση εμπεριέχει $N$ καλώδια, τότε το συνολικό HPWL της σχεδίασης, το οποίο θέλουμε να ελαχιστοποιήσουμε κατά τη διαδικασία της βελτιστοποίησης της τοποθέτησης υπολογίζεται ως εξής [3]:

$$HPWL_{total} = \sum^{N} HPWL_{net}$$




HPWL = (8 - 1) + (6 - 1) = 13

*Σχήμα 3-2 Παράδειγμα υπολογισμού της μετρικής HPWL με χρήση τεχνικής ελαχίστων και μεγίστων συντεταγμένων x και y των ακροδεκτών.*

Οι αλγόριθμοι βελτιστοποίησης τοποθέτησης, οι οποίοι υλοποιήθηκαν κατά τη διάρκεια της σύνταξης αυτής της μεταπτυχιακής διατριβής εφαρμόζουν την μετρική HPWL για την προσέγγιση του συνολικού μήκους του καλωδίου της τοποθέτησης. Η υλοποίηση των αλγορίθμων σε λογισμικό και υλικό θα αναλυθεί με λεπτομέρεια σε επόμενες ενότητες.

### 3.1.3 Άλλα γνωστά μοντέλα προσέγγισης μήκους καλωδίου

Η προσέγγιση του συνολικού μήκους των καλωδίων μέσω της μετρικής HPWL αποτελεί την προτιμώμενη μέθοδο και το προτιμότερο μοντέλο υπολογισμού και προσέγγισης του μήκους καλωδίου, καθώς είναι ένα εξαιρετικά απλό στην υλοποίησή του και γρήγορο στην εκτέλεσή του μοντέλο, ενώ επιτυγχάνει και ένα εξαιρετικά μικρό περιθώριο λάθους σε πραγματικά κυκλώματα.

Άλλα μοντέλα που αξίζει να αναφερθούν περιλαμβάνουν το μοντέλο πλήρους γραφήματος (*complete graph*), το μοντέλο μονοτονικής αλυσίδας (*monotone chain*), το μοντέλο αστεριού (*star*), το μοντέλο ευθύγραμμων ελάχιστων δέντρων (*rectilinear minimum spanning tree - RMST*), το μοντέλο ευθύγραμμου ελαχίστου δέντρου Steiner (*rectilinear Steiner minimum tree - RSMT*), το μοντέλο ευθύγραμμης δενδροστοιχίας Steiner (*rectilinear Steiner arborescence - RSA*), και τελευταίο αλλά εξίσου σημαντικό το μοντέλο δέντρου Steiner με έναν κορμό (*single-trunk Steiner tree - STST*). [1].

## 3.2 Τεχνικές βελτιστοποίησης

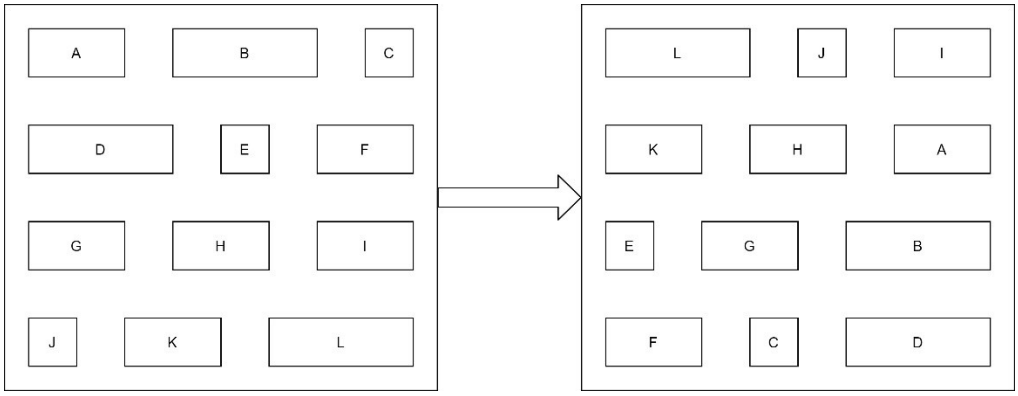
Έχοντας πλέον ορίσει μια βάση για τον υπολογισμό της βελτιστοποίησης μιας σχεδίασης σε σχέση με το συνολικό μήκος καλωδίων, είναι δυνατόν να προχωρήσουμε στη βελτίωση της τοποθέτησης. Δεδομένου ότι υπάρχουν πολλαπλοί σχεδιαστικοί περιορισμοί κατά την λεπτομερή τοποθέτηση, οι βελτιστοποιήσεις με τις οποίες μπορούμε να προχωρήσουμε σε μια τοποθέτηση είναι και αυτές με την σειρά τους εξαιρετικά περιορισμένες.

Οι βελτιστοποιήσεις περιορίζονται κυρίως στην εναλλαγή μεταξύ κελιών, στην περιστροφή κελιών ανά ομάδες των τριών, στην ολίσθηση κελιών κατά μήκος των γραμμών τους ή άλλες μικρότερες τοπικές αλλαγές [1].

### 3.2.1 Εναλλαγή μεταξύ κελιών

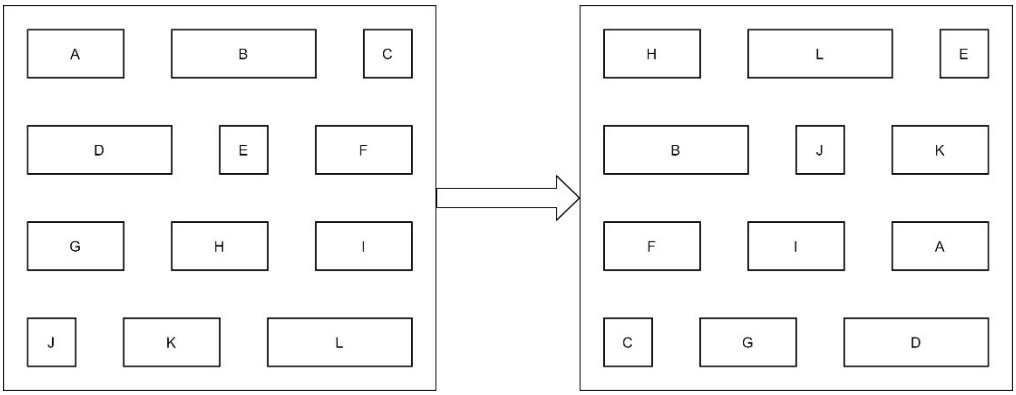
Με τη λεπτομερή τοποθέτηση τα κελιά της σχεδίασης τοποθετούνται σε ακριβείς θέσεις και οργανώνονται σε σειρές. Ανάμεσα σε αυτά τα κελιά υπάρχουν αρκετά κελιά τα οποία απαγορεύεται να μετατοπιστούν, καθώς βρίσκονται σε θέσεις ακροδεκτών. Τα κελιά που δεν υποβάλλονται στον εν λόγω περιορισμό μετακίνησης μπορούν και πρέπει να μετακινούνται και να εναλλάσσονται εντός των νόμιμων θέσεων κελιών, προκειμένου να βελτιστοποιηθεί η λεπτομερής τοποθέτηση ως προς τις μετρικές τις οποίες αναλύσαμε στην προηγούμενη ενότητα.

Επομένως, ένας αλγόριθμος βελτιστοποίησης της λεπτομερούς τοποθέτησης μπορεί να είναι τόσο απλός όσο η μετακίνηση κελιών μέσω της ανταλλαγής τους με άλλα κελιά μόνο όταν υπάρχει βελτίωση σε κάποια μετρική βελτιστοποίησης της τοποθέτησης.




*Σχήμα 3-3 Πραγματοποίηση εναλλαγής μεταξύ κελιών τοποθέτησης*

Δεδομένου ότι όλα τα κελιά της τοποθέτησης έχουν το ίδιο ύψος, για να κάνουμε τον αλγόριθμο βελτιστοποίησης ακόμη απλούστερο, είναι δυνατόν να περιοριστούμε στην εναλλαγή μεταξύ κελιών που έχουν και το ίδιο πλάτος, πράγμα που σημαίνει ότι μόνο τα κελιά του ίδιου μεγέθους θα θεωρούνται έγκυροι υποψήφιοι για εναλλαγή.




*Σχήμα 3-4 Πραγματοποίηση εναλλαγής μεταξύ κελιών του ιδίου μεγέθους*

Στην παρούσα διατριβή έχουν υλοποιηθεί πολλαπλές παραλλαγές της τεχνικής της εναλλαγής μεταξύ κελιών του ιδίου μεγέθους, οι οποίες θα αναλυθούν εκτενώς παρακάτω.

### 3.2.2 Εναλλαγή με πρώτο διαθέσιμο ιδίου μεγέθους

Ο πιο απλός αλγόριθμος εναλλαγής μεταξύ κελιών είναι εκείνος στον οποίο πραγματοποιείται εναλλαγή μεταξύ των πρώτων εγκύρων υποψηφίων εναλλαγής, με την εναλλαγή των οποίων υπάρχει βελτίωση στο συνολικό μήκος του καλωδίου. Δεν έχει σημασία πόσο έχει βελτιωθεί η μετρική, αυτό που έχει σημασία είναι αν υπήρξε βελτίωση ή όχι.

Στην πράξη, για κάθε κελί $c_1$ (*cell*), ελέγχεται πρώτα η δυνατότητα εναλλαγής με ένα άλλο κελί $c_2$ σύμφωνα με τον περιορισμό ιδίου μεγέθους. Όταν ικανοποιείται ο περιορισμός, πραγματοποιείται εναλλαγή μεταξύ των κελιών και υπολογίζεται εκ νέου το συνολικό μήκος του καλωδίου. Όταν υπάρχει βελτίωση στην εκτίμηση του μήκους καλωδίου, ο αλγόριθμος συνεχίζει με το επόμενο ζεύγος υποψήφιων κελιών, ενώ εάν δεν υπάρχει βελτίωση, ο αλγόριθμος αναιρεί την εναλλαγή και συνεχίζει με το επόμενο υποψήφιο κελί $c_2'$, εκτός εάν έχουν δοκιμαστεί όλα τα πιθανά υποψήφια κελιά $c_2$ και δεν υπήρξε βελτίωση. Σε αυτήν την περίπτωση η διαδικασία βελτιστοποίησης συνεχίζει με το επόμενο υποψήφιο κελί $c_1'$ εκτός αν έχουν δοκιμαστεί όλα τα κελιά της σχεδίασης, όπου ο αλγόριθμος ξεκινά την εκτέλεσή του από την αρχή.

Στη χειρότερη περίπτωση, θα εναλλαχθούν μεταξύ τους όλα τα έγκυρα κελιά με αναίρεση της εναλλαγής τους λόγω της μη βελτιστοποίησης της μετρικής. Αυτή η περίπτωση αποτελεί και την τελευταία επανάληψη του αλγορίθμου βελτιστοποίησης και σηματοδοτεί την ολοκλήρωση της εκτέλεσής του.

Algorithm-1: Swap First Available Same Size

**Input**: set of moveable cells $V$, cell width $w_c$, $c = 0, \cdots, N-1$, $N$ = number of cells

$hpwlTotal$ = initial HPWL total

```
foreach (cell c1 ∈ V)
  foreach (cell c2 ∈ V)
    if (c1 ≠ c2 AND w_c1 = w_c2)
      swapCells (c1, c2)
      hpwlAfterSwap = recalculateHPWL ()
      if (hpwlAfterSwap < hpwlTotal)
        hpwlTotal = hpwlAfterSwap
        break    // continue with next c1
      else
        swapCells (c1, c2)
```

Αν θεωρήσουμε πως μια σχεδίαση εμπεριέχει $N$ κελιά τότε για κάθε κελί ελέγχεται πρώτα ο περιορισμός ως πρός τα υπόλοιπα $N-1$ κελιά, ενώ στην συνέχεια πραγματοποείται εναλλαγή μεταξύ των κελιών.

Για κάθε κελί $c_i$ υπάρχουν επομένως τα εξής ενδεχόμενα ως πρός την εναλλαγή με κάποιο άλλο κελί $c_j$:

1. Δεν τηρείται ο περιορισμός ιδίου μεγέθους ($w_{c_i} \neq w_{c_j}$) και προχωρά η εκτέλεση του αλγορίθμου με τον επόμενο δυνατό υποψήφιο $c_{j+1}$.
2. Τηρείται ο περιορισμός ιδίου μεγέθους και υπάρχει βελτιστοποίηση στην εναλλαγή, επομένως προχωρά ο αλγόριθμος με το επόμενο ζεύγος υποψηφίων κελιών $c_{i+1}, c_0$.

3. Τηρείται ο περιορισμός ιδίου μεγέθους, αλλά δεν υπάρχει βελτιστοποίηση στην εναλλαγή και άρα αναιρείται η εναλλαγή και προχωρά ο αλγόριθμος με τον επόμενο δυνατό υποψήφιο $c_{j+1}$.
4. Δεν τηρείται ο περιορισμός ιδίου μεγέθους ή δεν υπήρξε βελτίωση κατά την εναλλαγή, αλλά δεν υπάρχει επόμενος υποψήφιος $c_j$, άρα ο αλγόριθμος συνεχίζει με το επόμενο ζεύγος υποψηφίων κελιών $c_{i+1}, c_0$, εκτός αν έχουν δοκιμαστεί και όλα τα υποψήφια κελιά $c_i$, όπου ο αλγόριθμος ξεκινά την εκτέλεσή του από την αρχή με δοκιμή μεταξύ των αρχικών υποψήφιων κελιών $c_0, c_1$.

### 3.2.3 Εναλλαγή με τελευταίο διαθέσιμο ιδίου μεγέθους

Στην προηγούμενη και πρώτη υλοποίηση του αλγορίθμου τα κελιά ελέγχονταν ένα προς ένα με αύξουσα σειρά, $c_0, c_1, c_2, \cdots$. Αν διατρέξουμε όλους τους δυνατούς υποψήφιους $c_i$ στον πρώτο βρόχο διέλευσης της δομής (ή των δομών) όπου αποθηκεύονται οι πληροφορίες των κελιών με αύξουσα σειρά, ενώ διατρέξουμε σε έναν βρόχο το υποψήφιο κελί $c_j$ με φθίνουσα σειρά, $c_{N-1}, c_{N-2}, c_{N-3}, \cdots$, τότε προκύπτει εύκολα μια δεύτερη παραλλαγή του αλγορίθμου.

Algorithm-2: Swap Last Available Same Size

**Input**: set of moveable cells $V$, cell width $w_c$, $c = 0, \cdots, N-1$, $N$ = number of cells

$V'$ = inverted $V$
$hpwlTotal$ = initial HPWL total

**foreach** (cell $c_1 \in V$)
  **foreach** (cell $c_2 \in V'$)
    **if** ($c_1 \neq c_2$ AND $w_{c_1} = w_{c_2}$)
      swapCells ($c_1$, $c_2$)
      $hpwlAfterSwap$ = recalculateHPWL ()
      if ($hpwlAfterSwap < hpwlTotal$)
        $hpwlTotal = hpwlAfterSwap$
        **break** // continue with next $c_1$
      else
        swapCells ($c_1$, $c_2$)

Στην πρώτη παραλλαγή γινόταν εναλλαγή με το πρώτο διαθέσιμο ιδίου μεγέθους. Σε αυτήν την παραλλαγή θα γίνετε εναλλαγή με το τελευταίο διαθέσιμο ιδίου μεγέθους, εφόσον η δομή θα διαπερνάτε με την αντίθετη κατεύθυνση.

Αν θεωρήσουμε σχεδίαση με $N$ κελιά, για κάθε κελί $c_i$ θα υπάρχουν τα εξής ενδεχόμενα ως πρός την εναλλαγή με κάποιο άλλο κελί $c_j$:

1. Δεν τηρείται ο περιορισμός ιδίου μεγέθους ($w_{c_i} \neq w_{c_j}$) και προχωρά η εκτέλεση του αλγορίθμου με τον επόμενο δυνατό υποψήφιο $c_{j-1}$.
2. Τηρείται ο περιορισμός ιδίου μεγέθους και υπάρχει βελτιστοποίηση στην εναλλαγή, επομένως προχωρά ο αλγόριθμος με το επόμενο ζεύγος υποψηφίων κελιών $c_{i+1}, c_{N-1}$.
3. Τηρείται ο περιορισμός ιδίου μεγέθους, αλλά δεν υπάρχει βελτιστοποίηση στην εναλλαγή και άρα αναιρείται η εναλλαγή και προχωρά ο αλγόριθμος με τον επόμενο δυνατό υποψήφιο $c_{j-1}$.

4. Δεν τηρείται ο περιορισμός ιδίου μεγέθους ή δεν υπήρξε βελτίωση κατά την εναλλαγή, αλλά δεν υπάρχει επόμενος υποψήφιος $c_j$, άρα ο αλγόριθμος συνεχίζει με το επόμενο ζεύγος υποψηφίων κελιών $c_{i+1}, c_{N-1}$, εκτός αν έχουν δοκιμαστεί και όλα τα υποψήφια κελιά $c_i$, όπου ο αλγόριθμος ξεκινά την εκτέλεσή του από την αρχή με δοκιμή μεταξύ των αρχικών υποψήφιων κελιών $c_0, c_{N-1}$.

### 3.2.4 Εναλλαγή με καλύτερο διαθέσιμο ιδίου μεγέθους

Στις προηγούμενες δύο υλοποιήσεις πραγματοποιούνταν εναλλαγή με την πρώτη βελτίωση που παρουσιάζονταν στην μετρική και ο αλγόριθμος συνέχιζε με τη δοκιμή του επόμενου ζεύγους κελιών.

Μια περαιτέρω παραλλαγή θα μπορούσε να εναλλάσσει κάθε κελί $c_i$ με όλα τα έγκυρα κελιά $c_j$ και να καταγράφει το καλύτερο υποψήφιο $c_{j_{best}}$ και την βελτίωση που συνεπάγεται αν πραγματοποιηθεί εναλλαγή με εκείνο το κελί $HPWL_{best}$, πραγματοποιώντας την εναλλαγή και αναιρώντας την, ανεξάρτητα από το αν υπήρχε βελτίωση ή όχι. Ένας τέτοιος αλγόριθμος θα προχωρήσει με την εναλλαγή με το καλύτερο δυνατό υποψήφιο κελί, αφότου έχουν δοκιμαστεί όλα τα δυνατά υποψήφια κελιά.

Για κάθε κελί θα πραγματοποιηθεί επομένως εναλλαγή αν τηρείται ο περιορισμός ιδίου μεγέθους και θα ενημερωθεί το καλύτερο υποψήφιο κελί μόνο εάν η βελτίωση που προκύπτει από την εναλλαγή ήταν καλύτερη από την υφιστάμενη καλύτερη. Στην συνέχεια, ο αλγόριθμος θα προχωρήσει με αναίρεση της εναλλαγής και δοκιμή του επόμενου υποψήφιου κελιού, ωσότου δοκιμαστούν όλα τα δυνατά υποψήφια κελιά. Τελικώς, ο αλγόριθμος θα ελέγξει αν υπάρχει έγκυρο υποψήφιο για εναλλαγή κελί και θα πραγματοποιήσει εναλλαγή με εκείνο.

Algorithm-3: Swap Best Available Same Size

**Input**: set of moveable cells $V$, cell width $w_c$, $c = 0, \cdots, N-1$, $N$ = number of cells

$hpwlTotal$ = initial HPWL total

**foreach** (cell $c_1 \in V$)
  $hpwlBest = hpwlTotal$
  $c2Best = None$

  **foreach** (cell $c_2 \in V$)
    **if** ($c_1 \neq c_2$ AND $w_{c_1} = w_{c_2}$)
      swapCells ($c_1$, $c_2$)
      $hpwlAfterSwap$ = recalculateHPWL ()

      if ($hpwlAfterSwap < hpwlBest$)
        $hpwlBest = hpwlAfterSwap$
        $c2Best = c_2$
      swapCells ($c_1$, $c_2$)

  **if** ($c2Best \neq None$)
    swapCells ($c_1$, $c2Best$)
    $hpwlTotal = hpwlBest$

### 3.2.5 Εναλλαγή μεταξύ γειτόνων ιδίου μεγέθους

Φυσικά, όσο αυξάνεται ο αριθμός των κελιών, άλλο τόσο αυξάνεται η πολυπλοκότητα του αλγορίθμου εναλλαγής κελιών.

Μια λύση για τη βελτίωση του χρόνου εκτέλεσης θα ήταν να μην εναλλάσσονται κελιά με το ίδιο μέγεθος, αλλά κελιά με το ίδιο μέγεθος τα οποία γειτνιάζουν, τα οποία συνδέονται δηλαδή μεταξύ τους με καλώδια. Έτσι, οι πιθανοί υποψήφιοι για εναλλαγή θα είναι ακόμα λιγότεροι, καθώς η ικανοποίηση και των δύο περιορισμών θα έχει σαφώς χαμηλότερη πιθανότητα.

Στην περίπτωση αυτή δεν έχει νόημα να συζητεί φυσικά κανείς για το καλύτερο δυνατό υποψήφιο κελί. Αρκεί η εναλλαγή με το πρώτο έγκυρο κελί που οδηγεί σε βελτίωση στη μετρική.

Algorithm-4: Swap Same Size Neighbors

**Input**: set of moveable cells $V$, adj list $adj_c$, width $w_c$, $c = 0, \cdots, N-1$, $N$ = number of cells

$hpwlTotal$ = initial HPWL total

**foreach** (cell $c_1 \in V$)
  **foreach** (cell $c_2 \in adj_{c_1}$)
    **if** ($c_1 \neq c_2$ AND $w_{c_1} = w_{c_2}$)
      swapCells ($c_1$, $c_2$)
      $hpwlAfterSwap$ = recalculateHPWL ()

      **if** ($hpwlAfterSwap < hpwlTotal$)
        $hpwlTotal = hpwlAfterSwap$
        **break**
      **else**
        swapCells ($c_1$, $c_2$)

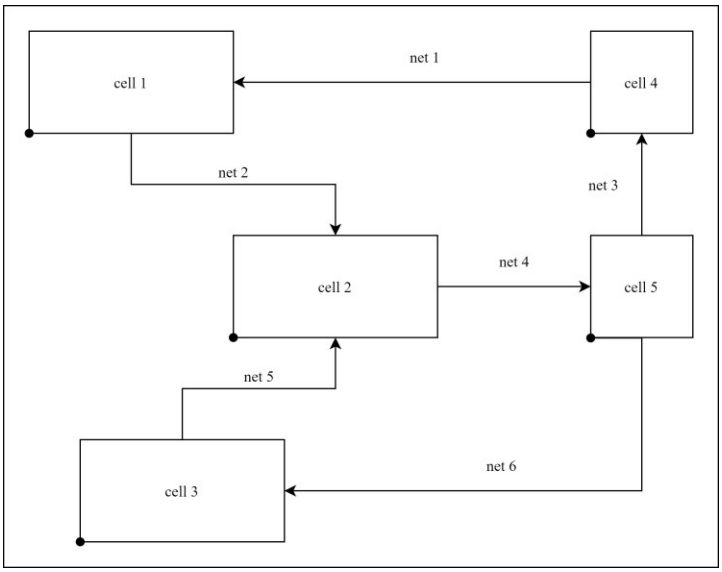
### 3.2.6 Τυχαία επιλογή υποψηφίου εναλλαγής

Μια ακόμα παραλλαγή του αλγορίθμου εναλλαγής θα μπορούσε να είναι αυτή στην οποία δε δοκιμάζονται τα κελιά γραμμικά ένα προς ένα, αλλά προσπερνιόνται ή ακυρώνονται υποψήφια κελιά με τυχαίο τρόπο, κάτι το οποίο οδηγεί στο να γίνετε τελείως τυχαία η επιλογή του υποψηφίου κελιού εναλλαγής.

Καθώς ο παράγοντας παράλειψης αυξάνεται, ο αλγόριθμος θα έχει φυσικά μικρότερη πιθανότητα εναλλαγής κόμβων. Με σωστή διαχείριση της ισορροπίας πραγματοποιείται καλύτερη βελτιστοποίηση της τοποθέτησης από ότι με οποιαδήποτε άλλη τεχνική, καθώς αυτός ο αλγόριθμος είναι ουσιαστικά μια εφαρμογή στοχαστικής τεχνικής παρόμοιας με την προσομοιωμένη ανόπτηση. Το μόνο πρόβλημα είναι ο χρόνος εκτέλεσης, ο οποίος είναι πολύ χειρότερος από οποιαδήποτε άλλη παραλλαγή των αλγορίθμων εναλλαγής.

# 4. Αναπαράσταση τοποθέτησης

Μια σχεδίαση υλικού απαρτίζεται συνήθως από πολλαπλά μικρότερα κυκλώματα, τα οποία συνδέονται μεταξύ τους μέσω καλωδίων. Συνεπώς, μια διάταξη υλικού μπορεί να εκφραστεί πολύ εύκολα με την χρήση γραφήματος ή γράφου. Κάθε ένα από τα υπο-κυκλώματα αποτελεί κόμβο του γράφου, ενώ οι συνδέσεις, τα καλώδια που συνδέουν τα υπο-κυκλώματα αποτελούν τις ακμές.




*Σχήμα 4-1 Σχηματική αναπαράσταση μιας τοποθέτησης*

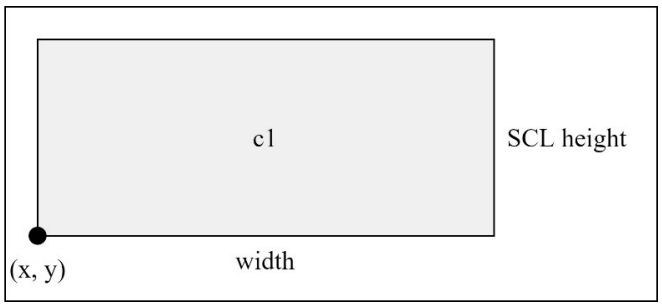
## 4.1 Τοποθέτηση σε κελιά

Τα υπο-κυκλώματα αναπαρίστανται με κελιά ή κόμβους (*nodes*) με συγκεκριμένες διαστάσεις και θέσεις. Κάθε κελί έχει συγκεκριμένο ύψος, πλάτος και θέση ανά τους άξονες που μπορεί να εκφραστεί με την χρήση συντεταγμένων $(x, y)$.

Καθώς τα κελιά διαθέτουν το ίδιο ύψος στις υλοποιήσεις που αναλύθηκαν στην παρούσα διατριβή (*Standard Cell Row - SCL*) χρειάζεται να αποθηκευτεί μόνο το πλάτος κάθε κελιού.

Κάθε κελί μπορεί να είναι και είτε μετακινήσιμο είτε όχι, πληροφορία που είναι πολύ σημαντικό να έχει στη διάθεσή του κατά την εκτέλεσή της βελτιστοποίησης ο αλγόριθμος εναλλαγής κελιών.

Για να καταστεί ακόμη πιο εύκολος ο υπολογισμός της μετρικής HPWL, αμελείται και ο προσανατολισμός των κελιών, και για κάθε κελί αποθηκεύονται μόνο η αριστερότερη (left $x$) και η χαμηλότερη (low $y$) θέση και ο υπολογισμός πραγματοποιείται αναφορικά με αυτήν την θέση.



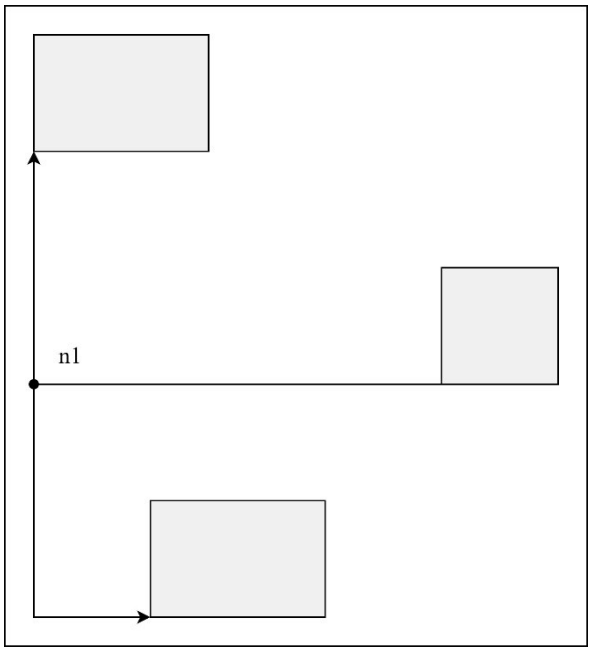

*Σχήμα 4-2 Σχηματική αναπαράσταση κελιού*

Τέλος, για κάθε κελί είναι απαραίτητο να αποθηκεύονται και τα καλώδια (*nets*) που συνδέονται με αυτό. Αν αυτό δεν γινόταν, τότε κατά την εναλλαγή δύο κόμβων θα ήταν αναγκαίο να υπολογιστεί η μετρική HPWL για όλα τα καλώδια και όχι μόνο για τα καλώδια που επηρεάζονται από την εναλλαγή.

Είναι φυσικά εφικτό ο αλγόριθμος βελτιστοποίησης να έχει αυτήν την γνώση και με διαφορετικό τρόπο που δεν απαιτεί μνήμη, ωστόσο μια τέτοια διαφορετική διαδικασία εύρεσης των καλωδίων θα απαιτούσε ακόμη περισσότερο χρόνο και θα ήταν ιδιαίτερης πολυπλοκότητας. Όσο αυξάνεται ο αριθμός των κελιών και καλωδίων, άλλο τόσο θα αυξάνονταν και ο χρόνος που απαιτείται για την εύρεση ενός κόμβου μέσω των ακροδεκτών των καλωδίων. Επομένως, μια τέτοια υλοποίηση δεν προτιμήθηκε κατά την σύνταξη της παρούσας εργασίας.

## 4.2 Κελιά και καλώδια

Για να μπορέσουμε να υπολογίσουμε τη μετρική HPWL κάθε καλωδίου πρέπει να αποθηκεύσουμε τα κελιά που συνδέονται μέσω των καλωδίων αυτών. Κάθε καλώδιο συνδέεται πρακτικώς με έναν αριθμό ακροδεκτών (*pins*), οι οποίοι μπορούν να συσχετιστούν κάθε φορά με δεδομένο κελί ή κόμβο της τοποθέτησης.




*Σχήμα 4-3 Σχηματική αναπαράσταση καλωδίου*

Στην πράξη απαιτείται έστω μια δομή ή ακόμη καλύτερα απαιτούνται δύο δομές που συνιστούν τη συναρτησιακή σχέση μεταξύ κελιών και καλωδίων. Καθώς ο υπολογισμός της μετρικής HPWL αποτελεί το πιο κρίσιμο μέρος της όλης διαδικασίας βελτιστοποίησης είναι εξίσου σημαντική τόσο η αποθήκευση των ακροδεκτών ανά καλώδιο, όσο και των καλωδίων με τα οποία συνδέεται το κάθε κελί.

Η μεν πρώτη δομή απαιτείται κατά τον υπολογισμό του HPWL ενός καλωδίου και δε δύναται να απαλειφθεί. Η μεν δεύτερη δομή μας επιτρέπει να προχωρήσουμε σε υπολογισμό της μετρικής μόνο σε καλώδια τα οποία επηρεάζονται κατά την εναλλαγή κελιών, παρακάμπτοντας έτσι τους περιττούς υπολογισμούς με τους οποίους θα έπρεπε να προχωρήσουμε σε διαφορετική περίπτωση και για όλα τα υπόλοιπα καλώδια της σχεδίασης.

## 4.3 Μορφοποίηση Bookshelf

Οι τυπικές τοποθετήσεις κελιών (*standard cell placements)* αναπαρίστανται με μορφοποίηση που είναι γνωστή ως μορφοποίηση βιβλιοθήκης (*Bookshelf format*). Η μορφοποίηση αυτή είναι σχεδιασμένη με τρόπο τέτοιο που επιτρέπει να είναι απλή και ευανάγνωστη στον άνθρωπο, καθώς και πρακτική για χρήση με προγράμματα ανάλυσης (*parsers*). Προς επίτευξη του εν λόγω σκοπού, η μορφοποίηση δεν περιλαμβάνει διάφορες αναλυτικές πληροφορίες που δεν απαιτούνται στο πλαίσιο της βελτιστοποίησης του σχεδιασμού [4].

Μια τοποθέτηση συναρμολογείτε από μια σειρά επιμέρους αρχείων, τα οποία αποθηκεύονται συνήθως σε ένα αρχείο *.aux*, το οποίο είναι της μορφής:

---

Όνομα μορφοποίησης : Όνομα αρχείου 1 Όνομα αρχείου 2 … Όνομα αρχείου N

---

Τα επιμέρους αρχεία που αποτελούν μια τοποθέτηση Bookshelf είναι τα εξής:

- Επέκταση *.nodes*: διαστάσεις και δυνατότητα μετακίνησης των κελιών της σχεδίασης.
- Επέκταση *.nets, wts*: ακροδέκτες ανά καλώδιο της σχεδίασης.
- Επέκταση *.pl*: λεπτομερής θέση και προσανατολισμός των κελιών της σχεδίασης.
- Επέκταση *.scl*: τυπική διάταξη κελιών της σχεδίασης.
- Επέκταση *.wts*: συντελεστές βάρους της σχεδίασης.

Για παράδειγμα μια σχεδίαση υλικού με το όνομα *test*, στην οποία τα κελιά τοποθετούνται σε σειρές θα μπορούσε να έχει το εξής αρχείο *test.aux*:

---

RowBasedPlacement : test.nodes test.nets test.wts test.scl test.pl

---

Τα ακριβές περιεχόμενα και η μορφή των αρχείων αυτών θα περιγραφεί με ιδιαίτερη λεπτομέρεια στις υπο-ενότητες που ακολουθούν.

### 4.3.1 Αρχείο κελιών ή κόμβων (.nodes)

Ένα αρχείο κόμβων αρχίζει με γραμμή που υποδεικνύει την έκδοση του πρότυπου, ενώ ακολουθούν γραμμές στις οποίες δηλώνεται η ημερομηνία δημιουργίας του αρχείου, ο χρήστης ή δημιουργός του αρχείο, καθώς και η πλατφόρμα πάνω στην οποία συντάθηκε το αρχείο.

Κατά συνέπεια, ένα αρχείο της επέκτασης *.nodes* δύναται να αρχίζει με τον εξής τρόπο:

---

UCLA nodes 1.0

# Created : Thu Jan 1 7:12:34 UTC 1970

# User : Bell Labs

# Platform : UNIX

---

Φυσικά τα περιττά κενά αγνοούνται, αλλά είναι σημαντικό να διαχωρίζεται κάθε ξεχωριστό στοιχείο είτε με κενά είτε με tabs, ώστε ο αναλυτής να μπορεί να τα διαχωρίσει.

Ακολουθούν δύο γραμμές, οι οποίες προορίζονται για τη δήλωση του συνολικού αριθμού κελιών, καθώς και του αριθμού των κελιών που είναι τερματικά ή μη μετακινήσιμα, και οι οποίες διαμορφώνονται ως εξής:

```
NumNodes     :   1204
NumTerminals :     64
```

Κατόπιν, σε κάθε γραμμή που περιλαμβάνει πληροφορίες και όχι κενά, αυτό που ορίζεται συχνά είναι ένα όνομα για τον κάθε κόμβο/κελί, το πλάτος του, το ύψος του, και ένα προαιρετικό 4ο στοιχείο που δηλώνει αν το κελί είναι τερματικό ή όχι.

Οπότε, το υπόλοιπο τμήμα του αρχείου *.nodes* θα έχει την εξής μορφή:

```
     a0      12      16
     a1       8      16
     a2      16      16
     a3      12      16

      ⋮

  a1138      18      16
  a1139       8      16
     p1       1       1      terminal
     p2       1       1      terminal
     p3       1       1      terminal

      ⋮

    p64       1       1      terminal
```

Ενδεικτικά, για να εξηγηθεί καλύτερα το περιεχόμενο του αρχείου, ο κόμβος a1 αποτελεί ένα κόμβο πλάτους 8 και ύψους 16, ο οποίος δεν είναι τερματικός και άρα μπορεί να μετακινηθεί ελεύθερα σε οποιαδήποτε επιθυμητή θέση κατά τη διαδικασία βελτιστοποίησης της τοποθέτησης, ενώ ο κόμβος p3 αποτελεί ένα τερματικό κόμβο με αυθαίρετο πλάτος 1 και ύψος 1.

Έχοντας υπόψη την διαδικασία βελτιστοποίησης, η εναλλαγή κελιών με σκοπό τη βελτιστοποίηση αυτής της συγκεκριμένης τοποθέτησης θα ήταν για παράδειγμα εφικτή μόνο μεταξύ των a0 και a3 ή μεταξύ των a1 και a1139, καθώς μόνο αυτοί οι κόμβοι έχουν τις ίδιες διαστάσεις.

Όπως ίσως έχει ήδη καταστεί αντιληπτό, το αρχείο αυτό δεν περιλαμβάνει καμία πληροφορία σχετικά με τη θέση των αντίστοιχων κόμβων ή κελιών, όπως επίσης δεν περιέχει και πληροφορίες σχετικά με τον τρόπο με τον οποίο συνδέονται μεταξύ τους και μέσω ποιων καλωδίων. Οι πληροφορίες αυτές παρέχονται με την σειρά τους από τα άλλα αρχεία που συνθέτουν το πρότυπο.

### 4.3.2 Αρχείο καλωδίων ή ακμών (.nets)

Ακολουθώντας την ίδια ακριβώς μορφή, ένα αρχείο που εμπεριέχει τις πληροφορίες των καλωδίων *.nets* αρχίζει με τον εξής τρόπο:

```
UCLA nets 1.0

# Created
# User
# Platform
```

Ακολουθούν δύο γραμμές που προορίζονται για τη δήλωση του συνολικού αριθμού καλωδίων και ακροδεκτών οι οποίες διαμορφώνονται ως εξής:

```
NumNets      :     20
NumPins      :     49
```

Στη συνέχεια, αυτό που ακολουθεί είναι οι πληροφορίες κάθε καλωδίου, οι οποίες ορίζονται κατά ομάδες γραμμών. Στην πρώτη γραμμή κάθε τέτοιας ομάδας αναγράφεται ο βαθμός του καλωδίου, που αποτελεί ουσιαστικά τον αριθμό των ακροδεκτών του. Κάθε γραμμή που ακολουθεί τη δήλωση του αριθμού ακροδεκτών αναφέρει το όνομα του ακροδέκτη, δηλαδή του αντίστοιχου κόμβου, ενώ ακολουθεί ένα γράμμα που υποδεικνύει αν είναι είσοδος, έξοδος ή και τα δύο.

Μπορεί να εξηγηθεί ευκολότερα η μορφή αυτών των ομάδων μέσω ενός παραδείγματος. Ο τρόπος με τον οποίο διαμορφώνεται το υπόλοιπο τμήμα ενός αρχείου *.nets* είναι ουσιαστικά ο εξής:

```
NetDegree   :   2
      p1     B
      a5     B
NetDegree   :   2
      p2     B
      a9     B
       ⋮
NetDegree : 5
      a5     O
      a3     I
      a9     I
      a6     I
     a14     I
```

Οι πρώτες δύο καταχωρήσεις του παραδείγματος ορίζουν καλώδια τα οποία συνδέουν τους ακροδέκτες των κελιών p1, a5, και p2, a9 αντίστοιχα, προς και τις δύο κατευθύνσεις. Η τελευταία καταχώρηση ορίζει καλώδιο μιας κατευθύνσεως και ουσιαστικά ορίζει τη διασύνδεση του συγκεκριμένου ακροδέκτη εξόδου του κελιού a5 με ακροδέκτη εισόδου από κάθε ένα από τα υπόλοιπα κελιά της σχεδίασης: a3, a9, a6 και a14.

### 4.3.3 Αρχείο τοποθέτησης κελιών (.pl)

Σε πλήρη εγκυρότητα με το πρότυπο το αρχείο τοποθέτησης των κελιών *.pl* ξεκινάει και αυτό με τη σειρά του με παρόμοια δήλωση όπως και αυτή που περιγράφηκε για τα αρχεία *.nodes* και *.nets*:

```
UCLA pl 1.0

# Created
# User
# Platform
```

Μετέπειτα, σε ξεχωριστές γραμμές διατυπώνονται το όνομα του κόμβου, οι συντεταγμένες θέσης $x$ και $y$, καθώς και ο προσανατολισμός του κελιού, οριστικοποιώντας τις πληροφορίες που απαιτούνται για την τοποθέτηση.

Το υπόλοιπο τμήμα ενός αρχείου *.pl* διαμορφώνεται επομένως ως εξής:

```
a0    20    112   :    S
a1    71     48   :    N
a2    33     64   :    N
a3    14     32   :    W
a4    45     96   :    S
a5    56     80   :    S
a6    68     64   :    S

 ⋮

p1    64    129   :   FS
p2    64    -33   :    N
p3   -33     64   :    E
p4   129     64   :   FW
```

Για παράδειγμα, η πρώτη γραμμή προσδιορίζει ότι ο κόμβος a0 βρίσκεται στη θέση $(20, 112)$ και είναι προσανατολισμένος προς το νότο. Ομοίως, όσον αφορά τον τερματικό κόμβο p3, ο κόμβος βρίσκεται στη θέση $(-33, 64)$ και είναι προσανατολισμένος προς τα ανατολικά.

Στο πλαίσιο της παρούσας διατριβής, η συντεταγμένη $x$ συσχετίζεται με την αριστερότερη εκ των συντεταγμένων $x$, του κελιού, ενώ η συντεταγμένη $y$ συσχετίζεται με την χαμηλότερη εκ των συντεταγμένων $y$ του κελιού. Επίσης, για λόγους απλότητας, ο προσανατολισμός του κελιού αμελείται και δε λαμβάνεται υπόψιν κατά την διαδικασία βελτιστοποίησης.

Τέλος, αξίζει να σημειωθεί σε αυτό το σημείο, πώς το αρχείο *.pl* αποτελεί το μόνο στοιχείο το οποίο μπορεί και πρέπει να τροποποιήσει ένας αλγόριθμος βελτιστοποίησης τοποθέτησης. Στην πράξη το αποτέλεσμα της όλης διαδικασίας βελτιστοποίησης θα είναι ένα νέο αρχείο τοποθέτησης των κελιών με επέκταση *.pl*, ενώ όλα τα υπόλοιπα αρχεία της μορφοποίησης Bookshelf μιας οποιασδήποτε τοποθέτησης θα παραμείνουν αμετάβλητα κατά τη βελτιστοποίησή της.

### 4.3.4 Λοιπά τυπικά αρχεία

Οι τρεις επεκτάσεις αρχείων *.nodes*, *.nets* και *.pl*, οι οποίες εξετάστηκαν έως το σημείο αυτό, αποτελούν τις τρεις πιο σημαντικότερες όσον αφορά την παρούσα διατριβή.

Στα πλαίσια της διπλωματικής εργασίας, δεν είναι αναγκαίο να αναλυθεί το αρχείο *.scl*, το οποίο εμφανίζει πληροφορίες σχετικά με την τυπική διάταξη των κελιών, καθώς όλες οι τοποθετήσεις που θα εξεταστούν έχουν κελιά τα οποία είναι τοποθετημένα σε νόμιμες θέσεις. Μέσω της βελτιστοποίησης τοποθέτησης δεν αλλάζουν και ποτέ αυτές οι καθιερωμένες θέσεις. Η μόνη αλλαγή που προκύπτει μετά τη βελτιστοποίηση είναι πως τα κελιά θα βρίσκονται σε διαφορετικές από τις αρχικές τους θέσεις, εφόσον θα πραγματοποιηθεί εναλλαγή μεταξύ τους.

Προκειμένου να γίνει έστω μια μικρή αναφορά, το αρχείο *.scl* έχει την εξής μορφή [5]:

```
UCLA scl 1.0
# Created
# User
# Platform

NumRows  :  96

CoreRow Horizontal
 Coordinate    :     0
 Height        :    16
 Sitewidth     :     1
 Sitespacing   :     1
 Siteorient    :     N
 Sitesymmetry  :     Y
 SubrowOrigin  :     0  Numsites  :  1550
End

⋮
```

Ομοίως, υπάρχουν και άλλες επεκτάσεις αρχείων, όπως η επέκταση *.wts*, η οποία χρησιμοποιείται για την αποθήκευση των συντελεστών βάρους για κόμβους ή καλώδια. Ένα αρχείο *.wts* που εμπεριέχει ένα συντελεστή βάρους ανά κόμβο ή κελί έχει την εξής μορφή:

```
UCLA wts 1.0

# Created
# User
# Platform

  a0      256
  a1      224

   ⋮
```

# 5. Υλοποίηση σε Λογισμικό

Η άμεση υλοποίηση της εναλλαγής σε υλικό δεν είναι εφικτή. Ως εκ τούτου, στα πλαίσια της παρούσας διατριβής έχει υλοποιηθεί λογισμικό, το οποίο συμπληρώνει την υλοποίηση υλικού.

Η γλώσσα προγραμματισμού της επιλογής δεν ήταν άλλη από την Python, καθώς η γλώσσα αυτή περιλαμβάνει πολλές χρήσιμες βιβλιοθήκες και είναι γρήγορη για τους σκοπούς της πρωτοτυποποίησης [6].

Φυσικά, ακόμη και ο σχεδιασμός της βελτιστοποίησης τοποθετήσεων σε λογισμικό αποτελείται από διάφορα συστατικά στοιχεία, τα οποία πρέπει να εξεταστούν σε βάθος. Αυτό το κεφάλαιο αποσκοπεί στο να εξηγήσει τι ακριβώς πρέπει να υλοποιηθεί σε λογισμικό.

Το κεφάλαιο εκκινεί με τις απαραίτητες δομές που έχουν οριστεί για την αποθήκευση μιας τοποθέτησης στο λογισμικό και συνεχίζει με τον τρόπο με τον οποίο οι τοποθετήσεις αναλύονται από τη μορφοποίηση Bookshelf προκειμένου να αποθηκευτούν στις εν λόγω δομές.

Στο πλαίσιο αυτής της διατριβής, υλοποιήθηκε επίσης μια γεννήτρια τυχαίας τοποθέτησης, για την οποία φυσικά απαιτήθηκε και η δημιουργία συναρτήσεων για την αποθήκευσή της στην αντίστοιχη μορφοποίηση Bookshelf.

Έχοντας αποθηκεύσει την τοποθέτηση είτε αναλύοντάς την από τη μορφοποίηση Bookshelf είτε δημιουργώντας τη μέσω της γεννήτριας στις αντίστοιχες δομές λογισμικού, η τοποθέτηση μπορεί πλέον να βελτιστοποιηθεί.

Για τον σκοπό αυτό είναι απαραίτητο να καλυφθεί πως υπολογίζεται το HPWL ενός συγκεκριμένου καλωδίου, πως υπολογίζονται εκ νέου οι τιμές της μετρικής μόνο για τα επηρεαζόμενα από μια εναλλαγή καλώδια, πως γίνεται η εναλλαγή των κόμβων και πως έχουν υλοποιηθεί οι παραλλαγές των αλγορίθμων εναλλαγής κελιών ίδιου μεγέθους. Στη συνέχεια, θα είναι δυνατό να ελεγχθεί το αποτέλεσμα της εκτέλεσης της βελτιστοποίησης μέσω λογισμικού.

Για να είναι δυνατή η προσομοίωση της σχεδίασης υλικού με συγκεκριμένη τοποθέτηση ή για να συμπληρωθούν οι μνήμες με αρχικές τιμές, είναι απαραίτητο να οριστούν και συναρτήσεις που αποθηκεύουν τις τιμές των κόμβων και των καλωδίων σε αρχεία μνήμης τα οποία μπορούν να χρησιμοποιηθούν από τη σχεδίαση σε γλώσσα HDL.

Τέλος, στην παρούσα διατριβή τα δεδομένα τοποθέτησης μεταφέρονται επίσης από το πρόγραμμα λογισμικού μέσω διασύνδεσης UART στο FPGA, με το FPGA να στέλνει πίσω τη νέα και βελτιστοποιημένη τοποθέτηση μετά την ολοκλήρωση της εκτέλεσής της.

## 5.1 Δομές αποθήκευσης πληροφοριών τοποθέτησης

Προκειμένου να είναι δυνατή η εκτέλεση βελτιστοποιήσεων σε μια τοποθέτηση, απαιτείται πρώτα η αποθήκευση των δεδομένων να γίνετε σε μια μορφή που καθιστά δυνατή τη χρήση τους σε επίπεδο λογισμικού.

Η γλώσσα προγραμματισμού Python παρέχει ποικίλες έτοιμες δομές δεδομένων που είναι ιδανικές για την υλοποίηση των στόχων της εργασίας. Στο πλαίσιο της παρούσας διατριβής προτιμήθηκαν οι τύποι δεδομένων λεξικού (*dictionary*) και λίστας (*list*), οι οποίοι περιλαμβάνονται στην μονάδα *typing*.

Μια λίστα στην Python αποτελεί ουσιαστικά τον ορισμό μιας αμετάβλητης ακολουθίας στοιχείων και συνήθως χρησιμοποιείται όπως ένας δυναμικός πίνακας. Ένα λεξικό στην πράξη αποτελεί έναν

πίνακα κατακερματισμού που αντιστοιχίζει συγκεκριμένα κλειδιά στις αντίστοιχες τιμές τους. Τα στοιχεία που θα αποθηκεύονται σε αυτές τις δομές δεδομένων αποτελούν νέους τύπους ή κλάσεις δεδομένων, οι οποίοι έχουν ορισθεί συγκεκριμένα για τον σκοπό της αποθήκευσης μιας τοποθέτησης και περιλαμβάνουν μόνο ακέραιους αριθμούς (*integers*) και αλφαριθμητικά (*strings*).

### 5.1.1 Κλάση αντικειμένων κόμβου

Αρχικά θα εξετασθεί η πιο περίπλοκη από τις καθορισμένες κλάσεις αντικειμένων, η οποία έχει οριστεί ειδικά για την αποθήκευση των πληροφοριών που απαιτούνται για κάθε κόμβο ή κελί της τοποθέτησης.

Κάθε ένα από τα αρχεία της μορφής Bookshelf παρέχει πληροφορίες σχετικά με τους κόμβους της σχεδίασης. Το αρχείο *.nodes* παρέχει στον αναλυτή το όνομα, το πλάτος, το ύψος και τη δυνατότητα μετακίνησης ή τον τύπο κίνησης ενός δεδομένου κόμβου. Κατά την ανάλυση του αρχείου *.nets* είναι δυνατή η αποθήκευση των δικτύων που σχετίζονται με έναν δεδομένο κόμβο, καθώς και των κόμβων που σχετίζονται με κάθε μεμονωμένο κόμβο, τη λίστα γειτνίασης του. Κατά την ανάλυση του αρχείου *.wts* είναι δυνατή η αποθήκευση του συντελεστή βάρους κάθε κόμβου. Τέλος, κατά την ανάλυση του αρχείου τοποθέτησης *.pl* είναι δυνατό να αποθηκευτούν οι συντεταγμένες x και y στο χώρο, καθώς και ο προσανατολισμός ενός δεδομένου κελιού.

Η τελική κλάση αντικειμένων τύπου κόμβου είναι η εξής:

```
class Node:

    def __init__(self, name: str, width: int, height: int, movetype: str, id: int):

        # .nodes
        self.name: str = name
        self.width: int = width
        self.height: int = height
        self.movetype: str = movetype
        self.id: int = id

        # .nets
        self.adjList: List[str] = []
        self.nets: List[str] = []

        # .wts
        self.weight: int = 0

        # .pl
        self.lowY: int = 0
        self.leftX: int = 0
        self.orientation: str = "N"
```

Καθώς το όνομα των κόμβων αποτελεί αλφαριθμητικό, όπως φαίνεται και παραπάνω, ορίζεται επίσης και ένα πρόσθετο πεδίο αναγνωριστικού (*id*), το οποίο αποτελεί στην ουσία τον αύξοντα αριθμό ευρετηρίου κάθε δεδομένου κόμβου και χρησιμοποιείται για την προσπέλαση των μνημών με τις πληροφορίες κόμβων στην υλοποίηση υλικού.

### 5.1.2 Κλάση αντικειμένων καλωδίου

Η αποθήκευση των πληροφοριών ενός δεδομένου καλωδίου είναι μια πολύ πιο εύκολη υπόθεση, καθώς όλες οι πληροφορίες που απαιτούνται παρέχονται από ένα και μόνο αρχείο της μορφοποίησης Bookshelf, το αρχείο *.nets*. Έτσι, κάθε δεδομένο καλώδιο αποτελείται πρακτικώς μόνο από μια λίστα με ακροδέκτες.

Καθώς δε δίνεται κάποιο όνομα από το αρχείο *.nets*, αυτό μπορεί να δημιουργηθεί εύκολα διατηρώντας σε κάθε στιγμή τον αριθμό των καλωδίων που επεξεργάστηκαν και επισυνάπτοντας τον αριθμό αυτό μετά από ένα γράμμα όπως για παράδειγμα το αγγλικό "*n*". Παρόμοια με την υλοποίηση κόμβων, κάθε καλώδιο έχει επίσης ένα πεδίο αναγνωριστικού (*id*) το οποίο συνδέεται άμεσα με τη δημιουργία του ονόματος. Τέλος, για τους σκοπούς της βελτιστοποίησης, κάθε καλώδιο διαθέτει επίσης ένα πεδίο για την αποθήκευση της τιμής της μετρικής HPWL.

Η τελική μορφή μιας κλάσης αντικειμένων τύπου καλωδίου είναι η ακόλουθη:

```
class Net:

    def __init__(self, name: str, id: int):

        # .nets
        self.name: str = name
        self.id: int = id
        self.pins: List[str] = []

        self.hpwl: int = 0
```

### 5.1.3 Κλάση αντικειμένων ακροδέκτη

Κατά την ανάλυση του αρχείου *.nets* είναι απαραίτητη η αποθήκευση των πληροφοριών για τους ακροδέκτες σε μια προσωρινή δομή δεδομένων, η οποία αποτελεί στην ουσία μια λίστα αντικειμένων που περιέχει το όνομα και την κατεύθυνση κάθε ακροδέκτη που λαμβάνεται κατά την ανάλυση των πληροφοριών για ένα δεδομένο καλώδιο.

Αυτή η δομή δεν απαιτείται για τα πεδία $pins$ και $nets$, αλλά για τη λίστα γειτνίασης ενός συγκεκριμένου κόμβου. Το αντικείμενο κάθε καλωδίου που καλύφθηκε στην προηγούμενη ενότητα αποθηκεύει όλα τα ονόματα ακροδεκτών μέσα στο πεδίο $pins$, ενώ το αντικείμενο κάθε κόμβου που είναι ακροδέκτης στο πλαίσιο του συγκεκριμένου καλωδίου, αποθηκεύει το όνομα του καλωδίου κάτω από το πεδίο $nets$.

Η λίστα αντικειμένων ακροδέκτη απαιτείται για την ευκολότερη ανάθεση των κόμβων που πρέπει να συμπεριληφθούν στη λίστα γειτνίασης. Στη λίστα αυτή αποθηκεύονται τα ονόματα των ακροδεκτών που αποτελούν ακροδέκτη εξόδου ("O") στη λίστα γειτνίασης του κόμβου/ακροδέκτη που αποτελεί ακροδέκτη εισόδου ("I"). Φυσικά, εάν ένας δεδομένος ακροδέκτης είναι και τα δύο ("B") θα συμπεριληφθεί δύο φορές στη διαδικασία.

Αυτή η διαδικασία είναι απαραίτητη για τον σκοπό της ενσωμάτωσης πληροφορίας σχετικά με την κατεύθυνση των καλωδίων, καθώς διαφορετικά όλα τα καλώδια θα θεωρούνταν αμφίδρομα και η λίστα γειτνίασης θα περιείχε αρκετές πρόσθετες μην υπαρκτές διασυνδέσεις.

Η τελική κλάση είναι η εξής:

```
class Pin:

    def __init__(self, name: str, direction: str):

        self.name = name
        self.direction = direction
```

### 5.1.4 Κλάση αντικειμένων τοποθέτησης

Για τον σκοπό της αποθήκευσης των πληροφοριών μιας τοποθέτησης ορίστηκε μια ακόμη κλάση, που αποτελεί και την τελευταία που θα αναλυθεί.

Κάθε τοποθέτηση έχει ένα όνομα και συγκεκριμένες σταθερές τιμές, οι οποίες περιλαμβάνουν τον αριθμό των κόμβων, τον αριθμό των τερματικών κόμβων, τον αριθμό των καλωδίων, καθώς και τον αριθμό των ακροδεκτών. Για το σκοπό της αποθήκευσης των πληροφοριών των κόμβων και των καλωδίων ορίζονται δύο λεξικά, με το κλειδί κάθε λεξικού να είναι το όνομα του κόμβου και του καλωδίου αντίστοιχα και την τιμή να είναι αντικείμενο των κλάσεων που καλύφθηκαν προηγουμένως. Τέλος, το αντικείμενο θα περιλαμβάνει επίσης ένα πεδίο για την αποθήκευση της συνολικής τιμής HPWL.

Η τελική μορφή της κλάσης αντικειμένων τύπου τοποθέτησης είναι η εξής:

```
class Placement:

    def __init__(self, name: str):

        self.name: str = name

        # constants
        self.nodeCount: int = 0
        self.terminalCount: int = 0
        self.netCount: int = 0
        self.pinCount: int = 0

        # nodes dictionary
        self.nodes: Dict[str, Node] = {}

        # nets dictionary
        self.nets: Dict[str, Net] = {}

        self.hpwl: int = 0
```

## 5.2 Ανάγνωση μορφοποίησης Bookshelf

Όλα τα αρχεία μιας δεδομένης τοποθέτησης σε μορφή Bookshelf αποθηκεύονται σε έναν ενιαίο κατάλογο και συνήθως έχουν το ίδιο όνομα. Ως εκ τούτου, για μια τοποθέτηση που ονομάζεται "*test*" η δομή του καταλόγου θα είναι η εξής:

```
test
├── test.nets
├── test.nodes
├── test.pl
└── test.wts
```

Μια τοποθέτηση δημιουργείται καθορίζοντας ένα όνομα για την τοποθέτηση και στη συνέχεια σε αυτή τη διατριβή η τοποθέτηση μπορεί είτε να αναλυθεί από μια δεδομένη μορφοποίηση Bookshelf ή να δημιουργηθεί τυχαία από μια γεννήτρια τυχαίων τοποθετήσεων, η οποία θα εξεταστεί περαιτέρω σε επόμενες ενότητες.

Η ανάγνωση των αρχείων της μορφοποίησης Bookshelf πραγματοποιείται από μια μεμονωμένη συνάρτηση που καλεί υπο-συναρτήσεις για κάθε επιμέρους αρχείο, όπως φαίνεται παρακάτω:

```
def parseFiles(placement: Placement, directoryName: str, fileName: str):
    parseNodesFile(directoryName + "/" + fileName + ".nodes", placement)
    parseNetsFile(directoryName + "/" + fileName + ".nets", placement)
    parseWTSFile(directoryName + "/" + fileName + ".wts", placement)
    parsePLFile(directoryName + "/" + fileName + ".pl", placement)
```

Έτσι, η ανάγνωση μιας δεδομένης τοποθέτησης σε μορφοποίηση Bookshelf απαιτεί στο βασικό πρόγραμμα μόνο την χρήση των παρακάτω γραμμών κώδικα:

```
directoryName = "test"
fileName = "test"

placement = Placement(fileName)

parseFiles(placement, directoryName, fileName)
```

### 5.2.1 Ανάγνωση αρχείου κόμβων (.nodes)

Η ανάλυση του αρχείου κόμβων γίνεται γραμμή προς γραμμή, με τις γραμμές να χωρίζονται σε μέρη (parts):

```
nodesFile = open(nodesFileName, 'r')

for line in nodesFile:
    parts = line.split()
        ⋮
nodesFile.close()
```

Γραμμές που δε διαχωρίζονται σε μέρη ή που αρχίζουν με τον ειδικό χαρακτήρα "#", ο οποίος προσδιορίζει σχόλια, παραλείπονται, ενώ η γραμμή που αρχίζει με "*UCLA*" εκτυπώνεται στην οθόνη τερματικού για να εμφανιστεί η μορφοποίηση και η αναθεώρησή της:

```
if len(parts) == 0:
    continue

if parts[0][0] == "#":
    continue

if parts[0] == "UCLA":
    print(line)
    continue
```

Οι γραμμές που αρχίζουν με "*NumNodes*" και "*NumTerminals*" εκτυπώνονται στην οθόνη τερματικού, ενώ οι αντίστοιχες τιμές τους, οι οποίες αποτελούν τον αριθμό κόμβων και αριθμό

τερματικών κόμβων αντίστοιχα αποθηκεύονται στα κατάλληλα πεδία σταθερών τιμών της τοποθέτησης μετά από μετατροπή τους σε ακέραια τιμή:

```
if parts[0] == "NumNodes":
    placement.nodeCount = int(parts[2])
    print(line)
    continue

if parts[0] == "NumTerminals":
    placement.terminalCount = int(parts[2])
    print(line)
    continue
```

Οι γραμμές που δεν εμπίπτουν σε καμία από τις προηγούμενες κατηγορίες αποτελούν γραμμές που περιλαμβάνουν τις πληροφορίες του εκάστοτε κόμβου. Για κάθε κόμβο δημιουργείται ένα καινούργιο αντικείμενο κόμβου και προστίθεται στο αντίστοιχο λεξικό κόμβων *nodes* του αντικειμένου τοποθέτησης. Ένας μετρητής αυξάνεται κάθε φορά που προστίθεται ένας νέος κόμβος, έτσι ώστε να μπορεί να αποθηκευτεί κατά τη δημιουργία του αντικειμένου στο πεδίο *id*. Ο μετρητής ξεκινά από το 0. Η τελική τιμή του μετρητή θα είναι ίση με τον συνολικό αριθμό των κόμβων μείον 1.

```
nodeCounter: int = 0

for line in nodesFile:
        ⋮

    name: str = parts[0]
    width: int = int(parts[1])
    height: int = int(parts[2])
    movetype: str = None
    id: int = nodeCounter

    if len(parts) > 3:
        movetype = parts[3]

    node = Node(name, width, height, movetype, id)
    placement.nodes[name] = node

    nodeCounter += 1
```

Φυσικά, μπορεί πολύ εύκολα να διαπιστωθεί ότι πολλά πεδία του αντικειμένου κόμβων δεν έχουν ακόμη συμπληρωθεί. Τα υπόλοιπα πεδία συμπληρώνονται φυσικά κατά την ανάλυση των άλλων αρχείων της μορφοποίησης Bookshelf.

### 5.2.2 Ανάγνωση αρχείου καλωδίων (.nets)

Η ανάλυση του αρχείου *.nets* αποτελεί σαφώς δυσκολότερη διαδικασία. Η ανάλυση γίνεται και πάλι γραμμή προς γραμμή, αλλά με διαφορετική δομή βρόχου, καθώς οι πληροφορίες ενός συγκεκριμένου καλωδίου δεν αποθηκεύονται σε μεμονωμένη γραμμή αλλά σε περισσότερες γραμμές.

Ως συνθήκη τερματισμού θεωρείται η ανάγνωση μιας γραμμής που δεν αποτελείται από τίποτα, ενώ η ανίχνευση των κενών διαστημάτων, των σχολίων, της αναθεώρησης και η αποθήκευση των σταθερών τιμών ακολουθεί την ίδια ακριβώς μέθοδο με την ανάλυση των κόμβων:

```
netsFile = open(netsFileName, 'r')

while True:

    line = netsFile.readline()

    if line == "":
        break

    parts = line.split()

    if len(parts) == 0:
        continue

    if parts[0][0] == "#":
        continue

    if parts[0] == "UCLA":
        print(line)
        continue

    if parts[0] == "NumNets":
        placement.netCount = int(parts[2])
        print(line)
        continue

    if parts[0] == "NumPins":
        placement.pinCount = int(parts[2])
        print(line)
        continue

        ⋮

netsFile.close()
```

Όταν μια γραμμή δεν εμπίπτει σε καμία από τις προηγούμενες κατηγορίες τότε ξεκινά η ανάλυση των πληροφοριών ενός καλωδίου. Πρώτο βήμα της διαδικασίας ανάλυσης αποτελεί σύμφωνα με τη μορφοποίηση Bookshelf η ανάγνωση του αριθμού ακροδεκτών του καλωδίου, δηλαδή του βαθμού του καλωδίου (*net degree*), που υλοποιείται ως εξής:

```
netDegree: int = 0
if parts[0] == "NetDegree":
    netDegree = int(parts[2])
```

Στη συνέχεια, σε έναν μικρό βρόχο δημιουργείται μια προσωρινή λίστα ακροδεκτών, η οποία αποθηκεύει το όνομα και την κατεύθυνση κάθε ακροδέκτη.

```
pins: list[Pin] = []

i: int
for i in range(0, netDegree):
    line2 = netsFile.readline()
    parts2 = line2.split()
    name = parts2[0]
    direction = parts2[1]

    pin = Pin(name, direction)

    pins.append(pin)
```

Ο πίνακας γειτνίασης κάθε κόμβου ενημερώνετε κατόπιν προσεκτικού ελέγχου των κατευθύνσεων, έτσι ώστε να μη συμπεριλαμβάνονται ανύπαρκτες διασυνδέσεις μεταξύ κόμβων:

```
for pin1 in pins:
   if pin1.direction != "O":
      for pin2 in pins:
         if pin1.name == pin2.name:
            continue
         if pin2.direction != "I":
            placement.nodes[pin1.name].adjList.append(pin2.name)
```

Στη συνέχεια, πραγματοποιείται η δημιουργία του αντικειμένου καλωδίου. Παράγεται πρώτα ένα αντίστοιχο όνομα για το καλώδιο, καθώς και ένα αναγνωριστικό (*id*), ενώ ακολουθεί η προσθήκη των ονομάτων των ακροδεκτών στην αντίστοιχη λίστα ακροδεκτών *pins* του αντικειμένου. Φυσικά το αντικείμενο προστίθεται και κατάλληλα στο λεξικό καλωδίων της τοποθέτησης.

```
netName = "n" + str(netCounter)
id = netCounter

net = Net(netName, id)

for pin in pins:
   net.pins.append(pin.name)

placement.nets[net.name] = net
```

Ο μετρητής αρχικοποιείται στην τιμή μηδέν και αυξάνεται στο τέλος του εξωτερικού *while* βρόχου. Παρόμοια με την περίπτωση του μετρητή κόμβων, η τελική τιμή του μετρητή θα είναι ίση με τον αριθμό των συνολικών καλωδίων μείον 1.

Τέλος, στη λίστα των καλωδίων *nets* κάθε κόμβου προστίθεται και το όνομα του καλωδίου:

```
for pin in pins:
   placement.nodes[pin.name].nets.append(netName)
```

### 5.2.3 Ανάγνωση αρχείου συντελεστών βάρους (.wts)

Η ανάλυση του αρχείου συντελεστών βάρους *.wts* είναι μια πολύ απλή διαδικασία. Κάθε γραμμή χωρίζεται σε μέρη, τα κενά και τα σχόλια αγνοούνται, η αναθεώρηση εκτυπώνεται στην οθόνη τερματικού, ενώ ο συντελεστής βάρους αποδίδεται στον αντίστοιχο κόμβο με μία απλή γραμμή μέσω πρόσβασης του αντίστοιχου πεδίου μέσω του λεξικού κόμβων που είναι πλέον αποθηκευμένο στο αντικείμενο τοποθέτησης.

Το κύριο σώμα της συνάρτησης έχει ως εξής:

```
wtsFile = open(wtsFileName, 'r')

for line in wtsFile:

   parts = line.split()

   if len(parts) == 0:
      continue
```

```
    if parts[0][0] == "#":
        continue

    if parts[0] == "UCLA":
        print(line)
        continue

    name = parts[0]
    weight = int(parts[1])

    placement.nodes[name].weight = weight

wtsFile.close()
```

### 5.2.4 Ανάγνωση αρχείου τοποθέτησης κελιών (.pl)

Παρόμοια με το αρχείο συντελεστών βάρους, η ανάλυση του αρχείου τοποθέτησης *.pl* είναι επίσης μια πολύ απλή διαδικασία, καθώς οι συντεταγμένες *x* και *y* και ο προσανατολισμός του κόμβου με το αντίστοιχο όνομα μπορούν να εκχωρηθούν απευθείας στην κατάλληλη εγγραφή του λεξικού κόμβων της τοποθέτησης.

Παραλείποντας, τον παρόμοιο κώδικα, ο κώδικας που προσθέτει νέες πληροφορίες έχει ως εξής:

```
plFile = open(plFileName, 'r')

for line in plFile:
    parts = line.split()

    ⋮

    name = parts[0]
    leftX: int = int(float(parts[1]))
    lowY: int = int(float(parts[2]))
    orientation = parts[4]

    placement.nodes[name].leftX = leftX
    placement.nodes[name].lowY = lowY
    placement.nodes[name].orientation = orientation

plFile.close()
```

## 5.3 Γεννήτρια τυχαίας τοποθέτησης

Καθώς οι διαθέσιμες σχεδιάσεις και οι αντίστοιχες τοποθετήσεις τους είναι συνήθως εξαιρετικά μεγάλες, καθίστατο ιδιαίτερα δύσκολη η δοκιμή πολλαπλών τοποθετήσεων σε FPGA. Για το λόγο αυτό, κρίθηκε απαραίτητη η δημιουργία μιας γεννήτριας τυχαίων τοποθετήσεων.

Η τοποθέτηση δημιουργείται μετά την απόδοση σειράς παραμέτρων και αρχικά δημιουργείται ως αντικείμενο της κλάσης, ενώ στη συνέχεια εγγράφεται σε μορφή Bookshelf δημιουργώντας έναν αντίστοιχο φάκελο και ένα προς ένα τα αρχεία που απαιτούνται.

### 5.3.1 Σύνθεση αρχείων μορφοποίησης Bookshelf

Η κατασκευή της μορφοποίησης Bookshelf έχει χωριστεί σε ξεχωριστές συναρτήσεις, οι οποίες λειτουργούν η καθεμία πάνω σε μία μόνο επέκταση αρχείου.

#### 5.3.1.1 Σύνθεση αρχείου κόμβων (.nodes)

Η δημιουργία και η εγγραφή πάνω στο αρχείο κόμβων είναι τόσο απλή όσο και η ανάγνωσή του. Πρώτα απ' όλα, προστίθενται γραμμές που σχετίζονται με τη μορφοποίηση και καθορίζουν στοιχεία όπως την αναθεώρηση, το δημιουργό και τη πλατφόρμα δημιουργίας:

```
nodes = placement.nodes

nodesFile = open(nodesFileName + ".nodes", 'w')

nodesFile.write("UCLA nodes 1.0\n")
nodesFile.write("# Created\n")
nodesFile.write("# Platform\n\n")

⋮

nodesFile.close()
```

Στη συνέχεια, είναι απαραίτητο να γραφτεί ο αριθμός των κόμβων και των τερματικών κόμβων ακολουθώντας τον αντίστοιχο ορισμό της μορφοποίησης και πραγματοποιώντας κατάλληλες προσαρμογές για καλύτερη αναγνωσιμότητα:

```
nodesFile.write("NumNodes :\t\t" + format(str(placement.nodeCount), '>6s') + "\n")
nodesFile.write("NumTerminals :\t" + format(str(placement.terminalCount), '>6s') + "\n")
```

Τέλος, με επαναληπτική διαδικασία γράφονται γραμμή προς γραμμή το όνομα, το πλάτος και το ύψος του κάθε κόμβου, καθώς και το αν είναι τερματικός ή όχι (*terminal*):

```
node: Node
for node in nodes.values():
    line: str = "\t" + format(node.name, '>6s') + \
        "\t" + format(str(node.width), '>6s') + \
        format(str(node.height), '>6s')

    if node.movetype == "terminal":
        line += "\t terminal \t"

    line += "\n"

    nodesFile.write(line)
```

### 5.3.1.2 Σύνθεση αρχείου καλωδίων (.nets)

Η δημιουργία και η εγγραφή πάνω στο αρχείο καλωδίων αποτελεί πιο πολύπλοκη διαδικασία. Αλλά, καθώς η μόνη χρήση της συνάρτησης θα είναι κατά τη διάρκεια της δημιουργίας μιας νέας τυχαίας τοποθέτησης, και καθώς η γεννήτρια τοποθέτησης είναι επίσης απλοποιημένη, η συμπλήρωση του αρχείου διευκολύνεται και αυτή με τη σειρά της.

Όπως και στο αρχείο *.nodes*, οι πρώτες γραμμές αφορούν την αναθεώρηση της μορφοποίησης, διάφορα σχόλια και τις τιμές των σταθερών που αφορούν το συνολικό αριθμό των καλωδίων και των ακροδεκτών:

```
nets = placement.nets

netsFile = open(netsFileName + ".nets", 'w')

netsFile.write("UCLA nets 1.0\n")
netsFile.write("# Created\n")
netsFile.write("# Platform\n\n")
```

```
netsFile.write("NumNets : " + format(str(placement.netCount), '>6s') + "\n")
netsFile.write("NumPins : " + format(str(placement.pinCount), '>6s') + "\n")

⋮

netsFile.close()
```

Εν συνεχεία, για κάθε καλώδιο, αυτό που γράφεται πρώτα είναι ο βαθμός του καλωδίου, ακολουθούμενος από ξεχωριστές γραμμές που περιλαμβάνουν το όνομα του κάθε ακροδέκτη και την κατεύθυνσή του. Στη γεννήτρια τοποθετήσεων αυτό που παράγεται είναι πάντοτε ένα καλώδιο για κάθε τερματικό κόμβο με κάθε τέτοιου είδους καλώδιο να έχει βαθμό 2, πράγμα που σημαίνει ότι ο δεύτερος ακροδέκτης θα είναι πάντα τυχαίος μην τερματικός κόμβος. Επιπλέον, η σύνδεση αυτή ορίζεται πάντοτε ως αμφίδρομη ("B"). Για τα υπόλοιπα καλώδια, ο πρώτος ακροδέκτης είναι πάντα ακροδέκτης εξόδου ("O"), ενώ όλοι οι υπόλοιποι ακροδέκτες είναι ακροδέκτες εισόδου ("I"). Κατά συνέπεια, ο κώδικας για τη δημιουργία των υπόλοιπων γραμμών έχει ως ακολούθως:

```
net: Net
netCounter: int = 0
for net in nets.values():
    netDegree = len(net.pins)
    netsFile.write("NetDegree : " + str(netDegree) + "\n")

    pin: str
    pinCounter: int = 0
    for pin in net.pins:
        line: str = "\t" + format(pin, '>6s')

        if netCounter < placement.terminalCount:
            line += " B\n"
        elif pinCounter == 0:
            line += " O\n"
        else:
            line += " I\n"

        netsFile.write(line)

        pinCounter += 1

    netCounter += 1
```

### 5.3.1.3 Σύνθεση αρχείου συντελεστών βάρους (.wts)

Η δημιουργία του αρχείου συντελεστών βάρους *.wts* είναι μια πολύ απλή διαδικασία, καθώς μετά τις τυπικές γραμμές, ακολουθούν μόνο γραμμές που περιλαμβάνουν κάθε φορά το όνομα του κόμβου και τον συντελεστή βάρους του, όπως φαίνεται παρακάτω:

```
wtsFile = open(wtsFileName + ".wts", 'w')

wtsFile.write("UCLA wts 1.0\n")
wtsFile.write("# Created\n")
wtsFile.write("# Platform\n\n")

node: Node
for node in nodes.values():
    line: str = "\t" + format(node.name, '>6s') + \
        "\t" + format(str(node.weight), '>6s') + "\n"

    wtsFile.write(line)

wtsFile.close()
```

### 5.3.1.4 Σύνθεση αρχείου τοποθέτησης κελιών (.pl)

Ομοίως, η δημιουργία του αρχείου τοποθέτησης *.pl* είναι επίσης μία πολύ απλή διαδικασία, καθώς μετά τις τυπικές γραμμές, αυτό που ακολουθεί είναι ξεχωριστές γραμμές, οι οποίες περιλαμβάνουν κάθε φορά το όνομα, τις συντεταγμένες *x* και *y* και τον προσανατολισμό του κάθε κόμβου, όπως φαίνεται παρακάτω:

```
plFile = open(plFileName + ".pl", 'w')

plFile.write("UCLA pl 1.0\n")
plFile.write("# Created\n")
plFile.write("# Platform\n\n")

node: Node
for node in nodes.values():
    line: str = node.name + " " + \
        str(node.leftX) + " " + str(node.lowY) + \
        " : " + node.orientation + "\n"

    plFile.write(line)

plFile.close()
```

## 5.3.2 Παράμετροι γεννήτριας

Φυσικά, σε μια γεννήτρια τοποθέτησης μπορούν να εφαρμοστούν ποικίλες παραμετροποιήσεις, ωστόσο στο πλαίσιο αυτής της διατριβής, αυτό που προτιμήθηκε ήταν η δημιουργία μιας απλής γεννήτριας, με βασικές δυνατότητες και στοιχειώδη δυνατότητα προσαρμογής της. Οι παράμετροι της γεννήτριας παρατίθενται στον παρακάτω πίνακα.

| | |
|---|---|
| nodeCount, terminalCount, netCount | Συνολικός αριθμός κόμβων, τερματικών κόμβων και καλωδίων |
| minNodeWidth, maxNodeWidth, nodeWidthStep | Ελάχιστη και μέγιστη τιμή πλάτους κόμβου με δεδομένο βήμα |
| sclHeight | Ύψος τυπικής διάταξης κελιών |
| minNodeWeight, maxNodeWeight, nodeWeightStep | Ελάχιστη και μέγιστη τιμή συντελεστή βάρους κόμβου με δεδομένο βήμα |
| minNetDegree, maxNetDegree | Ελάχιστος και μέγιστος βαθμός καλωδίου |
| numRows | Συνολικός αριθμός σειρών |
| performOptimization | Καθορισμός εκτέλεσης βελτιστοποίησης μετά τη δημιουργία |
| randomSeed | Σπόρος των τυχαίων συναρτήσεων |

*Πίνακας 5-1 Παράμετροι γεννήτριας τυχαίας τοποθέτησης*

## 5.3.3 Δημιουργία τυχαίων κόμβων

Ο αριθμός των κόμβων, των τερματικών κόμβων και των καλωδίων ορίζεται στη συνάρτηση $generatePlacement()$ η οποία καλεί τις υπόλοιπες συναρτήσεις και υπολογίζει την αρχική τιμή HPWL, ακολουθούμενη από προαιρετική βελτιστοποίηση σύμφωνα με την παραμετροποίηση.

Καθορίζοντας όλες τις ανώτερες παραμέτρους, η δημιουργία των κόμβων μπορεί να διαχωριστεί σε δύο μέρη:

1. Δημιουργία μην τερματικών κόμβων.
2. Δημιουργία τερματικών κόμβων.

Για τους μην τερματικούς κόμβους, το όνομα είναι πάντα ο χαρακτήρας "a" ακολουθούμενος από την τρέχουσα τιμή του μετρητή κόμβων, ο οποίος στην περίπτωση των μην τερματικών κόμβων μετράει από το 0 έως το συνολικό αριθμό κόμβων μείον τον αριθμό των τερματικών κόμβων μείον 1. Ομοίως, το πεδίου αναγνωριστικού (*id*) του κόμβου ισούται με την τιμή του μετρητή κόμβων.

```
nodeCounter: int
for nodeCounter in range(0, nodeCount - terminalCount):

    name: str = "a" + str(nodeCounter)
    id: int = nodeCounter

    ⋮
```

Το πλάτος δημιουργείται τυχαία λαμβάνοντας υπόψη τις τρείς παραμέτρους που ορίστηκαν προηγουμένως, το ύψος ισούται με το ύψος SCL και το είδος κίνησης (*movetype*) είναι "*None*".

```
width: int = nodeWidthStep * randint(minNodeWidth/nodeWidthStep, maxNodeWidth/nodeWidthStep)
height: int = sclHeight
movetype: str = None
```

Στη συνέχεια, δημιουργείται το αντικείμενο κόμβου, ακολουθεί ο καθορισμός ενός τυχαίου συντελεστή βάρους ο οποίος βασίζεται και πάλι στις τρείς παραμέτρους που ορίστηκαν προηγουμένως, και τέλος ο τυχαίος αυτός κόμβος προστίθεται στο λεξικό κόμβων *nodes* του αντικειμένου τοποθέτησης.

```
node = Node(name, width, height, movetype, id)

node.weight = nodeWeightStep*randint(minNodeWeight/nodeWeightStep, maxNodeWeight/nodeWeightStep)

placement.nodes[name] = node
```

Για τους τερματικούς κόμβους, το όνομα αρχίζει με τον χαρακτήρα "p" και ακολουθείται από τον τρέχοντα μετρητή τερματικών κόμβων, ο οποίος ξεκινά από το 1 και σταματά στο συνολικό αριθμό τερματικών κόμβων.

Το αναγνωριστικό (id) ισούται με τον αριθμό των συνολικών κόμβων μείον τον αριθμό των τερματικών κόμβων συν την τιμή του τερματικού μετρητή μείον 1, έτσι ώστε η τιμή του αναγνωριστικού να συνεχίζει μετά την τελευταία τιμή που χρησιμοποιήθηκε στον τελευταίο μην τερματικό κόμβο.

```
terminalCounter: int
for terminalCounter in range(1, terminalCount + 1):

    name: str = "p" + str(terminalCounter)
    id: int = nodeCount - terminalCount + terminalCounter – 1

    ⋮
```

Το πλάτος και το ύψος ορίζονται ως 1, ο συντελεστής βάρους ως 0 (η προεπιλεγμένη τιμή είναι μηδέν και δεν χρειάζεται να αλλάξει) και ο τύπος κίνησης (*movetype*) ορίζεται ως "*terminal*".

```
width: int = 1
height: int = 1
movetype: str = None
```

Στη συνέχεια δημιουργείται και πάλι το αντικείμενο κόμβου και προστίθεται στο ίδιο λεξικό κόμβων nodes του αντικειμένου τοποθέτησης.

```
node = Node(name, width, height, movetype, id)

placement.nodes[name] = node
```

### 5.3.4 Δημιουργία τυχαίων καλωδίων

Κατά τη δημιουργία των καλωδίων υπάρχουν και πάλι δύο περιπτώσεις, οι οποίες πρέπει να αντιμετωπιστούν ξεχωριστά:

1. Το καλώδιο περιλαμβάνει έναν τερματικό κόμβο.
2. Το καλώδιο δεν περιλαμβάνει τερματικό κόμβο.

Καθώς στη γεννήτρια αυτήν θεωρείται πώς υπάρχει μόνο ένα καλώδιο που συνδέεται με κάθε τερματικό κόμβο, η συνθήκη είναι τόσο απλή όσο ο έλεγχος εάν η τιμή του μετρητή καλωδίων είναι μικρότερη από τον αριθμό των τερματικών κόμβων ή όχι, όπως φαίνεται παρακάτω:

```
netCounter: int
for netCounter in range(0, netCount):
    if netCounter < terminalCount:
        ⋮

    else:
        ⋮
```

Ο συνολικός αριθμός ακροδεκτών της σχεδίασης αρχικοποιείται στην τιμή μηδέν, και στην πρώτη περίπτωση προστίθεται σε αυτόν η τιμή 2, ενώ στη δεύτερη περίπτωση προστίθεται μια τυχαία τιμή που βασίζεται στις παραμέτρους, δίνοντας όμως αρκετά μεγαλύτερη πιθανότητα στην περίπτωση να δοθεί η ελάχιστη τιμή στον βαθμό καλωδίου.

```
if netCounter < terminalCount:
    placement.pinCount += 2

    ⋮

else:
    netDegree: int
    if (random() ** 4 > 0.5):
        netDegree = randint(minNetDegree + 1, maxNetDegree)
    else:
        netDegree = minNetDegree

    placement.pinCount += netDegree

        ⋮
```

Στην περίπτωση του καλωδίου που περιλαμβάνει ακροδέκτη τερματικού κόμβου, ο κώδικας για τη συμπλήρωση της λίστας ακροδεκτών του καλωδίου είναι εξαιρετικά απλώς:

```
pins: list[Pin] = []

nodeID = netCounter + 1
name = "p" + str(nodeID)
direction = "B"

pin = Pin(name, direction)

pins.append(pin)

nodeID = randint(0, nodeCount - terminalCount - 1)
name = "a" + str(nodeID)
direction = "B"

pin = Pin(name, direction)

pins.append(pin)
```

Καθώς το όνομα των τερματικών κόμβων ξεκινά από το όνομα "a1", το όνομα του τερματικού ακροδέκτη θα περιλαμβάνει πάντα τον τρέχοντα αριθμό του μετρητή καλωδίων συν 1, ενώ ο μην τερματικός ακροδέκτης είναι οποιοσδήποτε τυχαίος κόμβος που δεν είναι τερματικός κόμβος.

Στην περίπτωση των άλλων καλωδίων, αυτό που προστίθεται είναι πάντα ένας μόνο τυχαίος μην τερματικός ακροδέκτης εξόδου και βαθμού καλωδίου μείον 1 μην τερματικοί ακροδέκτες εισόδου. Για τους ακροδέκτες εισόδου είναι πολύ σημαντικό να πραγματοποιηθεί κατάλληλος έλεγχος έτσι ώστε να μη συμπεριληφθεί εσφαλμένα δύο φορές κάποιο όνομα κόμβου.

```
pins: list[Pin] = []

nodeID = randint(0, nodeCount - terminalCount - 1)
name = "a" + str(nodeID)
direction = "O"

pin = Pin(name, direction)

pins.append(pin)

pinCounter: int
for pinCounter in range(1, netDegree):
    nodeID = randint(0, nodeCount - terminalCount - 1)
    name = "a" + str(nodeID)

    flag: int = 0
    for pin2 in pins:
        if pin2.name == name:
            flag = 1

    while flag:
        nodeID = randint(0, nodeCount - terminalCount - 1)
        name = "a" + str(nodeID)

        flag = 0

        for pin2 in pins:
        if pin2.name == name:
            flag = 1

    direction = "I"
```

```
    pin = Pin(name, direction)

    pins.append(pin)
```

Αυτό που ακολουθεί στο τέλος είναι παρόμοιος κώδικας με αυτόν που εφαρμόζεται κατά την ανάλυση του αρχείου *.nets*. Έτσι, για κάθε ακροδέκτη εισόδου, οι ακροδέκτες εξόδου προστίθενται στη λίστα γειτνίασης του αντίστοιχου κόμβου. Ένα αντικείμενο καλωδίου προστίθεται στην τοποθέτηση στην αντίστοιχη δομή λεξικού *nets* μετά τη δημιουργία του αντίστοιχου ονόματος, του αναγνωριστικού και της προσθήκης των ακροδεκτών του στη λίστα ακροδεκτών του *pins*. Τέλος, το όνομα του κάθε καλωδίου προστίθεται σε κάθε έναν από τους κόμβους που συμπεριλήφθηκαν στη λίστα ακροδεκτών εντός της λίστας καλωδίων *nets* τους.

```
for pin1 in pins:
    if pin1.direction != "O":
        for pin2 in pins:

            if pin1.name == pin2.name:
                continue

            if pin2.direction != "I":
                placement.nodes[pin1.name].adjList.append(pin2.name)

netName = "n" + str(netCounter)
id = netCounter

net = Net(netName, id)

for pin in pins:
    net.pins.append(pin.name)

placement.nets[net.name] = net

for pin in pins:
    placement.nodes[pin.name].nets.append(netName)
```

### 5.3.5 Δημιουργία νομιμοποιημένης τυχαίας τοποθέτησης

Το τελευταίο μέρος της γεννήτριας δημιουργεί την τοποθέτηση, δηλαδή ότι απαιτείται και αποθηκεύεται στο αντίστοιχο αρχείο *.pl*.

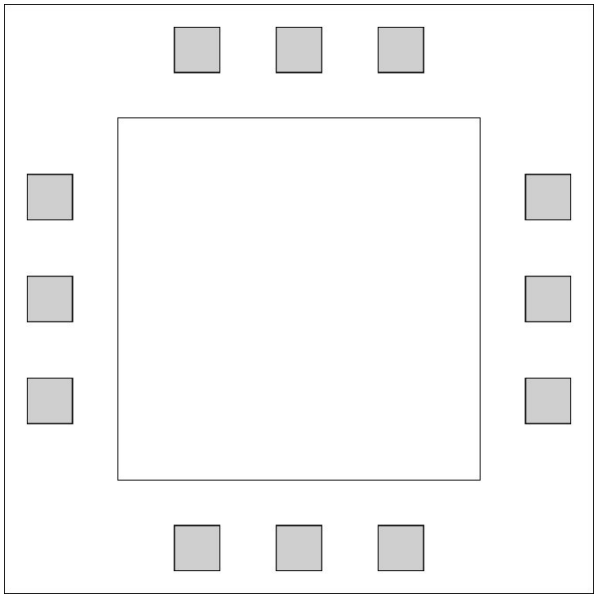
Κρίνεται εξαιρετικά σημαντικό να διαχωριστούν και να χειριστούν αναλόγως οι δύο ξεχωριστές περιπτώσεις κόμβων, οι οποίες είναι οι μην τερματικοί και οι τερματικοί κόμβοι.

Οι μην τερματικοί κόμβοι δημιουργούνται εύκολα, καθώς ο προσανατολισμός αμελείται στο πλαίσιο αυτής της διατριβής και επομένως ορίζεται απλώς ως "*N*", ενώ οι συντεταγμένες *x* και *y* μπορούν να δημιουργηθούν πολύ εύκολα χρησιμοποιώντας τις παραμέτρους που ορίζονται για την τοποθέτηση, και φροντίζοντας με κατάλληλο τρόπο έτσι επίσης ότι η τοποθέτηση είναι εκ των προτέρων νομιμοποιημένη.

```
nodeCounter: int
for nodeCounter in range(0, nodeCount - terminalCount):
    name = "a" + str(nodeCounter)

    orientation = "N"
    leftX: int = randint(0, sclHeight * numRows)
    lowY: int = sclHeight * randint(1, numRows)
```

```
    placement.nodes[name].orientation = orientation
    placement.nodes[name].leftX = leftX
    placement.nodes[name].lowY = lowY
```

Οι τερματικοί κόμβοι τοποθετούνται εκτός του νοητού τετραγώνου στο οποίο τοποθετήθηκαν οι υπόλοιποι κόμβοι. Για να επιτευχθεί αυτό οι τιμές του μετρητή τερματικών κόμβων διαχωρίζονται σε 4 ίσα διαστήματα. Η γεννήτρια διαχειρίζεται το κάθε τέτοιο διάστημα με κατάλληλο τρόπο, έτσι ώστε οι τερματικοί κόμβοι να έχουν ίσες αποστάσεις μεταξύ τους, ανά πλευρά που τοποθετούνται.



*Σχήμα 5-1 Τοποθεσίες των τερματικών κόμβων*

Παραλείποντας κώδικα που επαναλαμβάνεται, ο κώδικας που απαιτείται για την παραγωγή των συντεταγμένων των τερματικών κόμβων έχει τελικώς ως εξής:

```
terminalCounter: int
i: int

i = 1
for terminalCounter in range(1, int(terminalCount / 4) + 1):
    name = "p" + str(terminalCounter)

    orientation = "FS"
    leftX: int = int(sclHeight * i * numRows / (terminalCount / 4))
    lowY: int = sclHeight * (numRows + 2) + 1

    placement.nodes[name].orientation = orientation
    placement.nodes[name].leftX = leftX
    placement.nodes[name].lowY = lowY

    i += 1

i = 1
for terminalCounter in range(int(terminalCount / 4) + 1, int(terminalCount * 2 / 4) + 1):
    name = "p" + str(terminalCounter)

    orientation = "N"
```

```
    leftX: int = int(sclHeight * i * numRows / (terminalCount / 4))
    lowY: int = -sclHeight * 2 – 1

    ⋮

i = 1
for terminalCounter in range(int(terminalCount * 2 / 4) + 1, int(terminalCount * 3 / 4) + 1):
    name = "p" + str(terminalCounter)

    orientation = "E"
    leftX: int = -sclHeight * 2 - 1
    lowY: int = int(sclHeight * i * numRows / (terminalCount / 4))

    ⋮

i = 1
for terminalCounter in range(int(terminalCount * 3 / 4) + 1, terminalCount + 1):
    name = "p" + str(terminalCounter)

    orientation = "FW"
    leftX: int = sclHeight * (numRows + 2) + 1
    lowY: int = int(sclHeight * i * numRows / (terminalCount / 4))

    ⋮
```

Φυσικά, για να λειτουργήσει ο κώδικας χωρίς προβλήματα, ο αριθμός των τερματικών κόμβων πρέπει πάντα να είναι πολλαπλάσιο του 4, όπως συμβαίνει σε κάθε πραγματικό κύκλωμα, εφόσον οι ακροδέκτες μοιράζονται πάντοτε κατάλληλα στις τέσσερις πλευρές της τελικής τοποθέτησης.

Και με όλα αυτά, καλύφθηκε πλέον ο τρόπος αποθήκευσης των δεδομένων τοποθέτησης στο πρόγραμμα λογισμικού, ο τρόπος ανάλυσης και ανάγνωσης μιας τοποθέτησης η οποία είναι σε μορφοποίηση Bookshelf, καθώς και πώς παράγεται μια τυχαία εκ των προτέρων νομιμοποιημένη τοποθέτηση χρησιμοποιώντας μια γεννήτρια τοποθέτησης, της οποίας η δημιουργημένη τοποθέτηση μπορεί στη συνέχεια να αποθηκευτεί πολύ εύκολα και σε μορφή Bookshelf, για να διαβαστεί από τον αναλυτή αργότερα απευθείας.

## 5.4 Βελτιστοποίηση τοποθέτησης

Αφού μια τοποθέτηση έχει αποθηκευτεί στο αντίστοιχο αντικείμενο, είναι δυνατόν να καλυφθεί ο τρόπος βελτιστοποίησής της.

Πρώτα απ' όλα, είναι απαραίτητο να καλυφθεί ο τρόπος υπολογισμού του HPWL ενός συγκεκριμένου καλωδίου. Στη συνέχεια, θα είναι δυνατόν να καλυφθεί ο τρόπος επαναϋπολογισμού αυτής της τιμής κατά την εναλλαγή κόμβων μόνο για τα καλώδια που επηρεάζονται από την εναλλαγή. Ενώ τέλος θα μπορεί να αναλυθεί με λεπτομέρεια και ο τρόπος με τον οποίο γίνεται η εναλλαγή των κόμβων, καθώς και πως μπορούν να υλοποιηθούν εύκολα οι διάφορες παραλλαγές αλγορίθμων εναλλαγής κελιών ιδίου μεγέθους σε λογισμικό.

### 5.4.1 Υπολογισμός HPWL καλωδίου

Όπως έχει ήδη εξηγηθεί θεωρητικά σε προηγούμενο τμήμα της διατριβής, ο υπολογισμός του HPWL ενός συγκεκριμένου δικτύου είναι τόσο απλός όσο η εύρεση των ελάχιστων και μέγιστων τιμών $x$ και $y$ όλων των δεδομένων ακροδεκτών. Στην πράξη, ο αλγόριθμος περνάει από όλα τα στοιχεία της λίστας ακροδεκτών *pins* και συγκρίνει τις δεδομένες τιμές $x$ και $y$ του κόμβου αναζητώντας τον στο λεξικό των κόμβων *nodes* και ενημερώνοντας αναλόγως τις μέγιστες και ελάχιστες τιμές. Στο τέλος, η τιμή HPWL καθίσταται ένα απλό άθροισμα. Η αρχική τιμή των μεγίστων και ελαχίστων ορίζεται ίση με την τιμή του πρώτου κόμβου της λίστας ακροδεκτών.

```
minX: int = None
maxX: int = None
minY: int = None
maxY: int = None

for name in self.pins:
    node: Node = nodes[name]

    if minX == None:
        minX = node.leftX
        maxX = node.leftX
        minY = node.lowY
        maxY = node.lowY
        continue

    if node.leftX < minX:
        minX = node.leftX

    if node.leftX > maxX:
        maxX = node.leftX

    if node.lowY < minY:
        minY = node.lowY

    if node.lowY > maxY:
        maxY = node.lowY

self.hpwl = maxX - minX + maxY – minY
```

Σε αυτό το σημείο αξίζει σημειωθεί ότι αυτή η συνάρτηση έχει συμπεριληφθεί ως μέθοδος στην κλάση αντικειμένων τύπου καλωδίου που απαιτεί ως παράμετρο μόνο το αντίστοιχο λεξικό των κόμβων *nodes* από την τοποθέτηση.

Το συνολικό HPWL υπολογίζεται με παρόμοιο τρόπο μέσω μιας μεθόδου που περιλαμβάνεται στην κλάση τοποθέτησης, η οποία περνάει από όλα τα καλώδια, καλεί αυτήν τη συνάρτηση και στο τέλος υπολογίζει το άθροισμα των HPWL όλων των καλωδίων.

### 5.4.2 Επαναϋπολογισμός μόνο επηρεαζόμενων καλωδίων

Μπορεί να μοιάζει λίγο περίπλοκος, αλλά ο επαναϋπολογισμός μόνο του HPWL όσον αφορά τα καλώδια που επηρεάζονται από την εναλλαγή δύο κόμβων είναι στην πραγματικότητα πολύ απλός.

Καθώς κάθε κόμβος αποθηκεύει τα καλώδια στα οποία αναφέρεται, η συνάρτηση υπολογισμού HPWL καλείται στην ουσία μόνο για όλα αυτά τα καλώδια.

Για τους σκοπούς της χρήσης της συνάρτησης στο πλαίσιο των αλγορίθμων εναλλαγής, η συνάρτηση παρακολουθεί επίσης το συνολικό άθροισμα HPWL των επηρεαζόμενων καλωδίων πριν και μετά τον επαναϋπολογισμό και επιστρέφει τη διαφορά των αθροισμάτων αυτών. Θα υπάρχει βελτίωση μέσω της εναλλαγής όταν η τιμή της διαφοράς είναι αρνητική.

```
HPWL_before: int = 0
HPWL_after: int = 0

for netName in node.nets:
    HPWL_before += placement.nets[netName].hpwl
    placement.nets[netName].calcNetHPWL(placement.nodes)
    HPWL_after += placement.nets[netName].hpwl

return HPWL_after - HPWL_before
```

Φυσικά, αυτή η συνάρτηση πρέπει να χρησιμοποιηθεί δύο φορές, μία φορά για κάθε έναν από τους κόμβους που εναλλάσσονται, και έτσι η συνολική αλλαγή στο HPWL θα είναι τελικώς το άθροισμα των τιμών που επιστρέφονται.

### 5.4.3 Εναλλαγή κελιών

Η εναλλαγή δύο κόμβων έχει υλοποιηθεί σε δύο ξεχωριστές συναρτήσεις:

1. Η μία εναλλάσσει τους κόμβους και αναιρεί την εναλλαγή μόνο στην περίπτωση που δεν υπήρξε βελτίωση και στο τέλος επιστρέφει αν υπήρξε βελτίωση ή όχι μέσω της εναλλαγής.
2. Η άλλη εναλλάσσει τους κόμβους, υπολογίζει τη νέα τιμή του HPWL που προκλήθηκε από την εναλλαγή, αναιρεί την εναλλαγή και στο τέλος επιστρέφει τη νέα τιμή.

Η δεύτερη υλοποίηση χρησιμοποιείται φυσικά μόνο στην παραλλαγή του αλγορίθμου εναλλαγής που πραγματοποιεί εναλλαγή με το καλύτερο διαθέσιμο κελί ιδίου μεγέθους. Σε όλες τις άλλες περιπτώσεις αρκεί η χρήση της πρώτης υλοποίησης.

Προκειμένου να γίνει εναλλαγή δύο κόμβων, είναι απαραίτητο να αποθηκευτούν προσωρινά οι συντεταγμένες $x$ και $y$ κάθε κόμβου. Αφότου γίνει αυτό το βήμα μπορεί να πραγματοποιηθεί η εναλλαγή.

```
def setNodePos(node: Node, tmp: List[int]):
    node.leftX = tmp[0]
    node.lowY = tmp[1]
⋮
tmp1 = [node1.leftX, node1.lowY]
tmp2 = [node2.leftX, node2.lowY]

setNodePos(node1, tmp2)
setNodePos(node2, tmp1)
```

Ο επαναϋπολογισμός και ο έλεγχος της βελτίωσης στην πρώτη υλοποίηση γίνεται ως εξής:

```
change1: int = recalcHPWLAdj(placement, node1)
change2: int = recalcHPWLAdj(placement, node2)

hpwlPrev = placement.hpwl
placement.hpwl += change1 + change2

if placement.hpwl < hpwlPrev:
    return True
```

Όταν δεν υπάρχει βελτίωση η αναίρεση της εναλλαγής πραγματοποιείται καλώντας την ίδια βοηθητική συνάρτηση με πρωτύτερα, αλλά με διαφορετικές παραμέτρους, ενώ αγνοείται η τιμή που επιστρέφεται από τη συνάρτηση επαναϋπολογισμού. Τέλος, η τιμή του συνολικού HPWL τίθεται και πάλι στην αρχική της τιμή.

```
setNodePos(node1, tmp1)
setNodePos(node2, tmp2)

recalcHPWLAdj(placement, node1)
recalcHPWLAdj(placement, node2)

placement.hpwl = hpwlPrev

return False
```

Η μόνη αλλαγή που απαιτείται για τη δεύτερη υλοποίηση είναι πως δε θα υπάρχει έλεγχος για βελτίωση. Η νέα τιμή HPWL που προκύπτει από την πραγματοποίηση της εναλλαγής των κόμβων αποθηκεύεται σε μια μεταβλητή και επιστρέφεται μετά την αναίρεση της εναλλαγής.

```
⋮
hpwlNew: int = placement.hpwl + change1 + change2
⋮
return hpwlNew
```

Έχοντας υλοποιήσει όλα τα προηγούμενα μέρη της εναλλαγής σε ξεχωριστές συναρτήσεις ή μεθόδους, η υλοποίηση του αλγορίθμου εναλλαγής καθ' αυτού είναι πλέον εύκολη υπόθεση, επιτρέποντας φυσικά και τη δημιουργία των διαφόρων παραλλαγών.

### 5.4.4 Εναλλαγή με πρώτο διαθέσιμο ιδίου μεγέθους

Η εναλλαγή με τον πρώτο διαθέσιμο κόμβο ίδιου μεγέθους είναι τόσο απλή όσο και η διέλευση από όλους τους κόμβους της τοποθέτησης μέσω της δομής λεξικού τους. Φυσικά, θα πρέπει να τηρούνται και οι συνθήκες εναλλαγής και έτσι η παράλειψη τερματικών κόμβων είναι υποχρεωτική, ενώ οι κόμβοι εναλλάσσονται επίσης και μόνο όταν είναι διαφορετικοί και έχουν το ίδιο πλάτος.

Εάν η εναλλαγή οδηγήσει σε βελτιστοποιημένη τοποθέτηση, τότε ο αλγόριθμος συνεχίζει με το επόμενο ζεύγος κόμβων, ενώ όταν η εναλλαγή με τον αντίστοιχο δεύτερο υποψήφιο κόμβο δεν οδηγεί σε βελτίωση, η διαδικασία συνεχίζεται με τον επόμενο υποψήφιο κόμβο νούμερο 2.

Σε κώδικα Python η λειτουργικότητα που περιγράφηκε παραπάνω γράφεται ως εξής:

```
for nodeName1 in placement.nodes:
    node1: Node = placement.nodes[nodeName1]
    if node1.movetype == "terminal":
        continue
    for node2 in placement.nodes.values():
        if node1.name == node2.name or node1.width != node2.width or node2.movetype == "terminal":
            continue
        if swapNodes(placement, node1, node2):
            break
```

### 5.4.5 Εναλλαγή με τελευταίο διαθέσιμο ιδίου μεγέθους

Η αλλαγή του παραπάνω αλγορίθμου, έτσι ώστε το δεύτερο πέρασμα από τους κόμβους να γίνεται με τον αντίθετο τρόπο, είναι εύκολη υπόθεση.

Καθώς δεν μπορεί να πραγματοποιηθεί απευθείας αναστροφή ενός λεξικού, οι τιμές κλειδιού του λεξικού, δηλαδή τα ονόματα των κόμβων, αποθηκεύονται σε λίστα.

Στη συνέχεια, αυτή η λίστα αντιστρέφεται και ο δεύτερος βρόχος εφαρμόζεται πάνω σε αυτήν την λίστα.

Έτσι, οι μόνες αλλαγές στον κώδικα είναι οι παρακάτω:

```
nodesReverted: List[Node] = list(placement.nodes.values())
nodesReverted.reverse()

⋮

    for node2Reverted in nodesReverted:
        node2: Node = placement.nodes[node2Reverted.name]
```

### 5.4.6 Εναλλαγή με καλύτερο διαθέσιμο ιδίου μεγέθους

Για να είναι δυνατή η εναλλαγή με τον καλύτερο διαθέσιμο κόμβο του ίδιου μεγέθους πρέπει να ελεγχθούν εξαντλητικά όλοι οι πιθανοί υποψήφιοι κόμβοι 2, πράγμα που σημαίνει ότι η εναλλαγή αναιρείται και ενημερώνεται ο καλύτερος δυνατός υποψήφιος και ο HPWL όταν η βελτίωση από την εναλλαγή με τον επί του παρόντος υποψήφιου κόμβου 2 είναι καλύτερη από την προηγούμενη βελτίωση.

Αφού ελεγχθούν όλοι οι υποψήφιοι κόμβοι, αρχίζει η συνήθης εναλλαγή χρησιμοποιώντας την ίδια συνάρτηση με τις προηγούμενες παραλλαγές του αλγορίθμου.

Η διαφορά στον κώδικα έχει ως εξής:

```
⋮
    nodeName2: str = None
    bestHPWL: int = placement.hpwl

    for node2 in placement.nodes.values():

        if node1.name == node2.name or node1.width != node2.width or node2.movetype == "terminal":
            continue

        hpwlNew = swapNodesWithUndo(placement, node1, node2)

        if hpwlNew < bestHPWL:
            nodeName2 = node2.name
            bestHPWL = hpwlNew

    if nodeName2 != None:
        node2 = placement.nodes[nodeName2]

        swapNodes(placement, node1, node2)
```

### 5.4.7 Εναλλαγή μεταξύ γειτόνων ιδίου μεγέθους

Η παραλλαγή που εναλλάσσει μόνο τους γειτονικούς κόμβους είναι επίσης πολύ εύκολο να υλοποιηθεί. Η μόνη αλλαγή που απαιτείται είναι η πραγματοποίηση διέλευσης της λίστας γειτνίασης του πρώτου κόμβου κατά τη διάρκεια του δεύτερου βρόχου.

```
⋮
    for nodeName2 in node1.adjList:
        node2: Node = placement.nodes[nodeName2]
```

### 5.4.8 Εναλλαγή με τυχαίο ιδίου μεγέθους

Η τελική παραλλαγή που έχει υλοποιηθεί στο πλαίσιο αυτής της διατριβής, αλλά μόνο σε λογισμικό, είναι η εναλλαγή με έναν τυχαίο δεύτερο υποψήφιο κόμβο.

Ουσιαστικά, ακόμη και όταν πληρούνται οι προϋποθέσεις εναλλαγής, υπάρχει πάντοτε μια μικρή πιθανότητα να μην ελεγχθεί η εναλλαγή με τον εκάστοτε κόμβο, καθώς παράγεται μια τυχαία τιμή κινητής υποδιαστολής και συγκρίνεται με μια δεδομένη παράμετρο και όταν πληρείται η συγκεκριμένη συνθήκη ελέγχεται αντ' αυτού ο επόμενος πιθανός υποψήφιος.

```
⋮
    for node2 in placement.nodes.values():

        if node1.name == node2.name or node1.width != node2.width or node2.movetype == "terminal":
            continue

        if random() > skipChance:
            continue

        if swapNodes(placement, node1, node2):
            break
```

Με αυτόν τον τρόπο η εναλλαγή κόμβων πραγματοποιείται ακόμη πιο σπάνια, καθιστώντας τον αλγόριθμο πρακτικώς ένα είδος στοχαστικής τεχνικής.

## 5.5 Παράδειγμα εκτέλεσης

Ενδεικτικά, ακολουθεί ένα παράδειγμα εκτέλεσης. Αρχικά δηλώνεται μια τοποθέτηση με το όνομα "*test*" και δημιουργείται με τη χρήση της γεννήτριας τοποθέτησης:

```
placement = Placement("test")

generatePlacement(placement, nodeCount=1222, terminalCount=32, netCount=1403,
    minNodeWidth=8, maxNodeWidth=28, nodeWidthStep=4, sclHeight=16,
    minNodeWeight=96, maxNodeWeight=512, nodeWeightStep=16, minNetDegree=4, maxNetDegree=42,
    numRows=32, performOptimization=False, randomSeed=0)
```

Μετά την δημιουργία της τοποθέτησης υπολογίζεται η τιμή του HPWL:

```
placement.calcTotalHPWL()
```

Στη συνέχεια, εκτελείται ο αλγόριθμος εναλλαγής μέχρι να μην υπάρχει βελτίωση, υπολογίζοντας επίσης τον συνολικό χρόνο εκτέλεσης, και δημιουργώντας στο τέλος της εκτέλεσης και ένα νέο αρχείο στο οποίο αποθηκεύεται η νέα τοποθέτηση που προέκυψε σε μορφοποίηση Bookshelf:

```
timeBefore: float = time()

while True:

    hpwl_prev = placement.hpwl

    swapFirstAvailSameSize(placement)

    if placement.hpwl == hpwl_prev:
```

```
        break

    timeAfter: float = time()

    print("Total Runtime (in seconds):", timeAfter - timeBefore)

    serializePLFile(directoryName + "/" + fileName + "_sw", placement.nodes)
```

Φυσικά, έχει παραλειφθεί αρκετός κώδικας, αλλά εκτελώντας το πρόγραμμα σε έναν AMD Ryzen 9 5950x, η έξοδος στην οθόνη τερματικού έχει ως εξής:

```
⋮

Initial HPWL: 929267 or 9.293e+05

⋮


Swapped a476 with a755...
New HPWL: 580192 or 5.802e+05

Swapped a565 with a685...
New HPWL: 580181 or 5.802e+05

Total Runtime (in seconds): 182.19665718078613
```

## 5.6 Δημιουργία αρχείων μνήμης για χρήση στην προσομοίωση

Για να είναι δυνατή η προσομοίωση του υλικού, οι ποικίλες πληροφορίες για τους κόμβους και τα καλώδια πρέπει να αποθηκευτούν σε αρχεία μνήμης, κατά προτίμηση σε δεκαεξαδική μορφή. Με αυτόν τον τρόπο στον κώδικα Verilog μπορεί έπειτα να χρησιμοποιηθεί η συνάρτηση συστήματος *$readmemh()* για τον σκοπό της αρχικοποίησης των μνημών.

Οι συναρτήσεις που έχουν οριστεί είναι οι ακόλουθες:

1. *serializeNodesWidth()* : Με αυτή τη συνάρτηση γράφονται τα πλάτη των κόμβων σε ένα νέο αρχείο με όνομα *width.mem* σε ξεχωριστές γραμμές και δεκαεξαδική μορφή.
2. *serializeNodesMoveType()* : Με αυτή τη συνάρτηση γράφεται η δυνατότητα μετακίνησης των κόμβων σε ένα νέο αρχείο με όνομα *movetype.mem* σε ξεχωριστές γραμμές με την τιμή “1” να υποδεικνύει πως ο κόμβος είναι τερματικός.

3. *serializeNodesWidthAndMoveType()* : Συνδυασμός της εγγραφής του πλάτους και της δυνατότητα μετακίνησης κάθε κόμβου σε ένα νέο αρχείο με όνομα *width_movetype.mem*, όπου κάθε γραμμή είναι σε δεκαεξαδική μορφή και το τελευταίο bit αντιπροσωπεύει τη δυνατότητα μετακίνησης (βελτιστοποίηση της χρήσης του διαθέσιμου χώρου).

4. *serializeNodesNets()* : Αυτή η συνάρτηση γράφει δύο αρχεία, ένα αρχείο με όνομα *node_nets.mem* που αποθηκεύει τα αναγνωριστικά (*id*) όλων των καλωδίων της τοποθέτησης ομαδοποιημένα ανά κόμβο και ένα αρχείο με όνομα $node\_net\_addr.mem$ που αποθηκεύει τη διεύθυνση του πρώτου καλωδίου κάθε κόμβου στην πρώτη μνήμη.

5. *serializeNodesAdjList()* : Αυτή η συνάρτηση γράφει δύο αρχεία, ένα αρχείο με όνομα *nodes_adj_list.mem* που αποθηκεύει τα αναγνωριστικά (*id*) όλων των κόμβων στις λίστες

γειτνίασης της τοποθέτησης ομαδοποιημένα ανά κόμβο και ένα αρχείο με όνομα $node\_adj\_list\_addr.mem$ που αποθηκεύει τη διεύθυνση του πρώτου κόμβο της λίστας γειτνίασης του κάθε κόμβου στην πρώτη μνήμη.

6. *serializeNetsPins()* : Αυτή η συνάρτηση γράφει δύο αρχεία, ένα αρχείο με όνομα $net\_pins.mem$ που αποθηκεύει τα αναγνωριστικά (*id*) όλων των ακροδεκτών της τοποθέτησης ομαδοποιημένα ανά καλώδιο και ένα αρχείο με όνομα $net\_pin\_addr.mem$ που αποθηκεύει τη διεύθυνση του πρώτου ακροδέκτη κάθε καλωδίου στην πρώτη μνήμη.

7. *serializeNodesLowY()* : Με αυτή τη συνάρτηση γράφονται οι συντεταγμένες $y$ των κόμβων σε ένα νέο αρχείο με όνομα $nodes\_lowy.mem$ σε ξεχωριστές γραμμές και δεκαεξαδική μορφή.

8. *serializeNodesLeftX()* : Με αυτή τη συνάρτηση γράφονται οι συντεταγμένες $x$ των κόμβων σε ένα νέο αρχείο με όνομα $nodes\_leftx.mem$ σε ξεχωριστές γραμμές και δεκαεξαδική μορφή.

9. *serializeNetsHPWL()* : Με αυτή τη συνάρτηση γράφονται οι τιμές της μετρικής HPWL των καλωδίων σε ένα νέο αρχείο με όνομα $nets\_hpwl.mem$ σε ξεχωριστές γραμμές και δεκαεξαδική μορφή.

## 5.7 Μεταφορά και λήψη τοποθέτησης μέσω πρωτοκόλλου UART

Για τους σκοπούς της μεταφοράς και λήψης δεδομένων μέσω UART, έχει χρησιμοποιηθεί η βιβλιοθήκη *pyserial* [7].

Παραλείποντας τις αυτονόητες συναρτήσεις, οι οποίες έχουν οριστεί για το άνοιγμα και το κλείσιμο της σειριακής διασύνδεσης και τη μεταφορά και λήψη δεδομένων μέσω της διασύνδεσης, οι συναρτήσεις για τη μεταφορά των πληροφοριών τοποθέτησης από/προς το πρόγραμμα Python προς/από το FPGA είναι οι ακόλουθες:

1. *transferSerialParameters()* : Με αυτή τη συνάρτηση μεταφέρονται οι παράμετροι τοποθέτησης, δηλαδή ο αριθμός των κόμβων, ο αριθμός των καλωδίων και ο αριθμός των ακροδεκτών στο FPGA, έτσι ώστε το FPGA να μπορεί στη συνέχεια να λάβει τα στοιχεία των μνημών.

2. *transferSerialNodesWidthAndMoveType()* : Με αυτή τη συνάρτηση μεταφέρεται ο συνδυασμός του πλάτους και της δυνατότητας μετακίνησης κάθε κόμβου στο FPGA. Για να επιτευχθεί ο συνδυασμός τους, το πλάτος του κόμβου μετατοπίζεται αριστερά και το επόμενο bit λαμβάνει τιμή “1” για τερματικούς κόμβους και “0” για μην τερματικούς.

3. *transferSerialNodeNets()* : Με αυτή τη συνάρτηση μεταφέρονται όλα τα καλώδια της σχεδίασης ομαδοποιημένα ανά κόμβο στο FPGA. Η συνάρτηση ουσιαστικά διατρέχει τη λίστα των καλωδίων για όλους τους κόμβους και αποστέλλει το αναγνωριστικό (*id*) τους.

4. *transferSerialNodeNetAddr()* : Με αυτή τη συνάρτηση μεταφέρονται οι διευθύνσεις του πρώτου καλωδίου κάθε κόμβου στη μνήμη καλωδίων των κόμβων στο FPGA.

5. *transferSerialNetPins()* : Με αυτή τη συνάρτηση μεταφέρονται όλα οι ακροδέκτες της σχεδίασης ομαδοποιημένοι ανά καλώδιο στο FPGA. Η συνάρτηση ουσιαστικά διατρέχει τη λίστα των ακροδεκτών για όλα τα καλώδια και αποστέλλει το αναγνωριστικό (id) τους.

6. *transferSerialNetPinAddr()* : Με αυτή τη συνάρτηση μεταφέρονται οι διευθύνσεις του πρώτου ακροδέκτη κάθε καλωδίου στη μνήμη ακροδεκτών των καλωδίων στο FPGA.

7. *transferSerialNodesLowYLeftX()* : Με αυτή τη συνάρτηση μεταφέρονται οι συντεταγμένες $x$ και $y$ όλων τον κόμβων της τοποθέτησης στο FPGA.

8. *receiveSerialNodesLowYLeftX()* : Με αυτή τη συνάρτηση λαμβάνονται οι συντεταγμένες $x$ και y όλων τον κόμβων της προκύπτουσας τοποθέτησης από το FPGA.

Η μεταφορά της λίστας γειτνίασης και των αντίστοιχων διευθύνσεων προς τις τιμές της δεν έχει υλοποιηθεί, καθώς η υλοποίηση του αλγορίθμου εναλλαγής καλύτερων γειτόνων υλοποιήθηκε με χρήση ROM και όχι RAM.

Ομοίως, η μεταφορά ή η λήψη της μετρικής HPWL κάθε καλωδίου δεν κρίθηκε απαραίτητη, καθώς μετά τη λήψη των πληροφοριών τοποθέτησης η υλοποίηση υλικού προχωρά πρώτα με τον υπολογισμό της αρχικής μετρικής HPWL κάθε καλωδίου και της συνολικής τοποθέτησης, το οποίο αποτελεί μια πολύ γρήγορη διαδικασία. Η HPWL της προκύπτουσας σχεδίασης μπορεί επίσης να υπολογιστεί πολύ γρήγορα στο λογισμικό μετά τη λήψη των νέων συντεταγμένων $x$ και $y$ των κόμβων.

# 6. Υλοποίηση σε Υλικό

Σε αυτό το κεφάλαιο θα περιγραφεί ο τρόπος με τον οποίο η σχεδίαση έχει υλοποιηθεί σε κώδικα HDL με απώτερο σκοπό το ανέβασμα σε FPGA και την παράλληλη εκτέλεση και επαλήθευση της λειτουργικότητάς της με το μοντέλο λογισμικού.

Το κεφάλαιο αυτό καλύπτει αρχικά τον τρόπο με τον οποίο οι απαιτούμενες πληροφορίες τοποθέτησης αποθηκεύονται στο υλικό, ενώ στη συνέχεια, περιγράφεται εκτενώς η λειτουργικότητα κάθε επιμέρους μονάδας, συμπεριλαμβανομένου του τρόπου επικοινωνίας και συνεργασίας της με τις υπόλοιπες μονάδες. Έτσι, αφότου ολοκληρωθεί η αναφορά στον τρόπο αποθήκευσης της πληροφορίας, θα καλυφθούν και όλα τα παρακάτω θέματα:

1. Πώς πραγματοποιείται ο υπολογισμός της μετρικής HPWL για ένα καλώδιο.

2. Πώς πραγματοποιείται ο υπολογισμός του αρχικού συνολικού HPWL της τοποθέτησης.

3. Πώς επαναϋπολογίζεται η τιμή της μετρικής εκ νέου μόνο σε καλώδια που επηρεάζονται μετά την πραγματοποίηση της εναλλαγής κελιών.

4. Πώς πραγματοποιείται η εναλλαγή των κόμβων, καθώς και η αναίρεση της εναλλαγής τους σε περίπτωση μη βελτιστοποίησης της μετρικής ή με τη χρήση αντίστοιχου σήματος ελέγχου αναίρεσης, το οποίο σηματοδοτεί αναίρεση ανεξάρτητα από την αλλαγή στη μετρική HPWL που προκλήθηκε μέσω της εναλλαγής αυτής.

5. Πώς μεταφέρθηκαν και υλοποιήθηκαν οι διάφορες τεχνικές βελτιστοποίησης με χρήση εναλλαγής κελιών σε γλώσσα περιγραφή υλικού.

6. Πώς πραγματοποιείται η επικοινωνία μέσω της διεπαφής UART, η αποθήκευση των ληφθέντων πληροφοριών μιας τοποθέτησης στις μνήμες της σχεδίασης καθώς και η αποστολή της νέας και βελτιστοποιημένης τοποθέτησης μέσω UART μετά το πέρας της εκτέλεσης.

7. Τέλος, πραγματοποιείται και μια αναφορά στα διάφορα σχέδια ανώτατου επιπέδου που μπορούν να δημιουργηθούν χρησιμοποιώντας όλες αυτές τις υπο-μονάδες και τις διάφορες παραλλαγές τους.

## 6.1 Αποθήκευση τοποθέτησης

Δεν είναι δυνατόν να οριστούν ούτε να χρησιμοποιηθούν δομές όπως λίστες και λεξικά των γλωσσών λογισμικού ανώτερου επιπέδου σε γλώσσες περιγραφής υλικού HDL. Στο πλαίσιο της σχεδίασης υλικού, οι πληροφορίες τοποθέτησης αποθηκεύονται σε προσπελάσιμες μνήμες.

### 6.1.1 Αξιοποίηση διαφορετικών τύπων μνήμης

Εάν ο μόνος στόχος και λόγος σχεδιασμού του υλικού ήταν η βελτιστοποίηση μιας και μόνο τοποθέτησης, τότε οι μοναδικές πληροφορίες που θα έπρεπε να αποθηκευτούν σε μνήμη τυχαίας προσπέλασης RAM θα ήταν η θέσεις των κόμβων και η τιμή HPWL κάθε καλωδίου, καθώς αυτές είναι οι μοναδικές πληροφορίες που αλλάζουν κατά τη διάρκεια της βελτιστοποίησης μιας τοποθέτησης. Όλες οι υπόλοιπες πληροφορίες, δηλαδή το πλάτος των κόμβων, τα καλώδια των κόμβων και οι διευθύνσεις τους, οι ακροδέκτες των καλωδίων και οι διευθύνσεις τους, οι λίστες γειτνίασης των κόμβων και οι διευθύνσεις τους, θα μπορούσαν εύκολα να αποθηκευτούν σε μνήμες

μόνο για ανάγνωση τύπου ROM. Ο συγκεκριμένος τρόπος σκέψης προβλέπει τουλάχιστον δύο διαφορετικές υλοποιήσεις:

1. Όλη η μνήμη είναι και αναγνώσιμη και εγγράψιμη.
2. Μόνο οι μνήμες συντεταγμένων κόμβων x, y και HPWL καλωδίων είναι αναγνώσιμες και εγγράψιμες, ενώ όλες οι άλλες μνήμες είναι μνήμες μόνο για ανάγνωση.

Οι μνήμες από την πρώτη υλοποίηση είναι φυσικά ακόμα δυνατό να αρχικοποιηθούν με τις τιμές μιας συγκεκριμένης τοποθέτησης, επιτρέποντας στην επικοινωνία UART να παραλείπει τις πληροφορίες τοποθέτησης που δεν αλλάζουν.

### 6.1.2 Απαιτήσεις της σχεδίασης σε μνήμη

Ο υπολογισμός του ελάχιστου απαιτούμενου πλάτους δεδομένων και των απαιτούμενων διευθύνσεων κάθε μνήμης είναι μια ιδιαίτερα εύκολη διαδικασία από τη σκοπιά του λογισμικού. Πιο συγκεκριμένα, αρκεί να προσδιοριστεί η μέγιστη δυνατή τιμή κάθε πληροφορίας που θα αποθηκευτεί και να υπολογιστεί ο αριθμός των απαιτούμενων bits λαμβάνοντας τον λογάριθμο με βάση το 2 ($log2$). Καθώς οι συντεταγμένες $x$ και $y$ είναι προσημασμένοι ακέραιοι αριθμοί, είναι απαραίτητο να αυξηθεί το αποτέλεσμα κατά ένα. Τέλος, επειδή η τιμή του $log2$ είναι και πάντοτε τιμή κινητής υποδιαστολής, το αποτέλεσμα στρογγυλοποιείται και στον αμέσως επόμενο ακέραιο.

Οι μνήμες που χρησιμοποιούνται σε αυτή τη σχεδίαση παρατίθενται στον παρακάτω πίνακα.

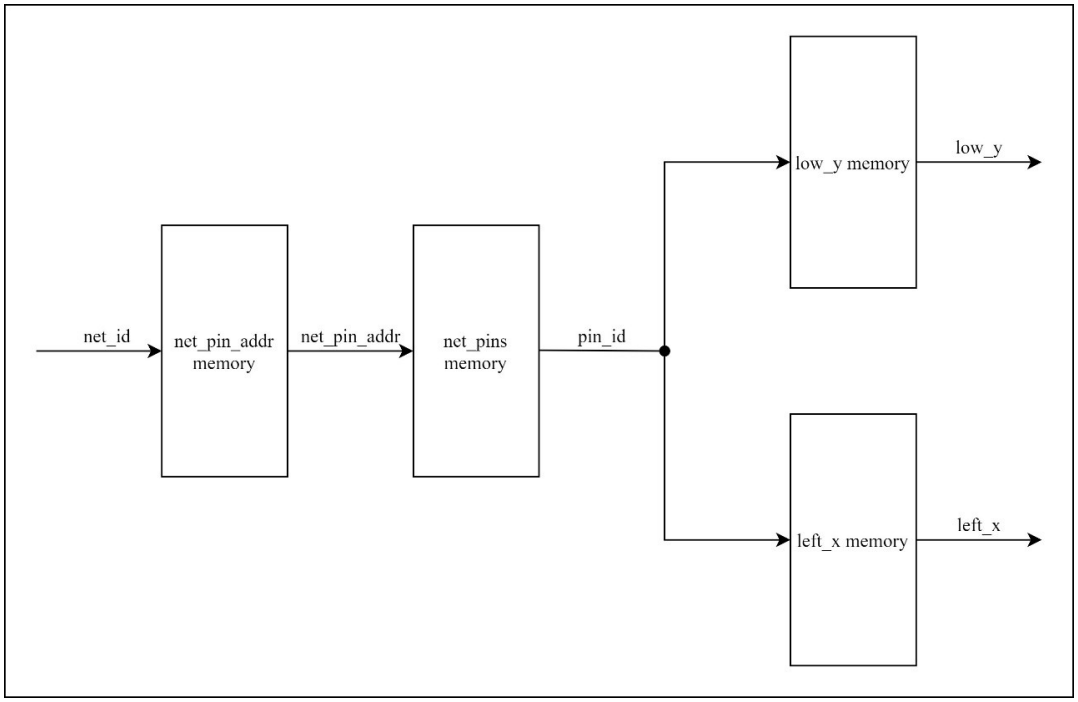
| Μνήμη | Στοιχεία | Πλάτος Διεύθυνσης | Πλάτος Δεδομένων |
|---|---|---|---|
| x_ram | $node\ count$ | $log2(node\ count)$ | $log2(max\ left\ x) + 1$ |
| y_ram | $node\ count$ | $log2(node\ count)$ | $log2(max\ low\ y) + 1$ |
| hpwl_net_ram | $net\ count$ | $log2(net\ count)$ | $log2(max\ hpwl\ net)$ |
| width_mt_ram (or rom) | $node\ count$ | $log2(node\ count)$ | $log2(max\ width)\ +\ 1$ |
| node_nets_ram (or rom) | $pin\ count$ | $log2(pin\ count)$ | $log2(node\ count)$ |
| node_net_addr_ram (or rom) | $node\ count\ +\ 1$ | $log2(node\ count + 1)$ | $log2(pin\ count)$ |
| net_pins_ram (or rom) | $pin\ count$ | $log2(pin\ count)$ | $log2(net\ count)$ |
| net_pin_addr_ram (or rom) | $net\ count\ +\ 1$ | $log2(net\ count + 1)$ | $log2(pin\ count)$ |
| node_adj_list_rom | $adj\_list\ entries$ | $log2(adj\_list\ entries)$ | $log2(node\ count)$ |
| node_adj_list_addr_rom | $node\ count\ +\ 1$ | $log2(node\ count + 1)$ | $log2(adj\_list\ entries)$ |

*Πίνακας 6-1 Οι μνήμες της σχεδίασης με αριθμό στοιχείων, πλάτος διευθύνσεων και δεδομένων.*

Ως εκ τούτου, ορίζεται μια μέγιστη τιμή για το αντίστοιχο πλάτος διεύθυνσης και δεδομένων κάθε μνήμης όταν πραγματοποιείται η σύνθεση, η υλοποίηση και ο προγραμματισμός ενός FPGA.

## 6.2 Υπολογισμός HPWL καλωδίου

Όπως έχει ήδη καλυφθεί στα θεωρητικά κεφάλαια της διατριβής, αλλά και όπως έχει ήδη παρουσιαστεί εφαρμοσμένο στο κεφάλαιο της υλοποίησης λογισμικού, ο υπολογισμός του HPWL ενός καλωδίου επιτυγχάνεται με την εύρεση των μεγίστων και ελαχίστων συντεταγμένων $x$ και $y$ των ακροδεκτών και τον υπολογισμό ενός συγκεκριμένου αθροίσματος των διαφορών τους. Η πρόσβαση στις αντίστοιχες τιμές $x$ και $y$ απαιτεί τρεις ξεχωριστές προσπελάσεις μνήμης, όπως απεικονίζεται στο σχήμα που ακολουθεί στην αρχή της επόμενης σελίδας.




*Σχήμα 6-1 Απεικόνιση των προσπελάσεων μνήμης που απαιτούνται για την ανάγνωση των συντεταγμένων κάθε ακροδέκτη ενός καλωδίου*

Συνεπώς, ο επαναϋπολογισμός (ή και γενικά ο υπολογισμός) της μετρικής HPWL ενός καλωδίου με αναγνωριστικό $net\_id$ διεξάγεται ως εξής:

1. Η διεύθυνση του πρώτου ακροδέκτη του καλωδίου, $net\_pin\_addr$, διαβάζεται από τη μνήμη διευθύνσεων ακροδεκτών καλωδίου παρέχοντας το αναγνωριστικό $net\_id$ ως διεύθυνση και ουσιαστικά διαβάζοντας την τιμή που είναι αποθηκευμένη στον καθορισμένο δείκτη μνήμης.

2. Καθώς ο αριθμός των ακροδεκτών ενός εκάστοτε καλωδίου είναι άγνωστος, το επόμενο βήμα αφορά στην ανάγνωση της διεύθυνσης του πρώτου ακροδέκτη από το καλώδιο που είναι αποθηκευμένο αμέσως μετά από αυτό για το οποίο υπολογίζεται εκ νέου το HPWL. Αυτό γίνεται με προσπέλαση της διεύθυνσης $net\_id + 1$. Με τον τρόπο αυτό επιτυγχάνεται η αποθήκευση της διεύθυνσης $last\_pin\_addr$, η οποία μπορεί να χρησιμοποιηθεί στη συνέχεια ως συνθήκη τερματισμού και αποτελεί επίσης και τη δεύτερη και τελευταία πρόσβαση ανάγνωσης που απαιτείται για την πρώτη μνήμη.

3. Κατόπιν, το αναγνωριστικό $pin\_id$ του ακροδέκτη και αντίστοιχα κόμβου μπορεί να διαβαστεί με προσπέλαση στη μνήμη ακροδεκτών μέσω της διεύθυνσης $net\_pin\_addr$.

4. Το $pin\_id$ αποτελεί ουσιαστικά ένα αναγνωριστικό κόμβου $node\_id$ το οποίο μπορεί να δοθεί επομένως απευθείας ως διεύθυνση στις μνήμες συντεταγμένων $x$ και $y$ για να διαβαστούν οι συντεταγμένες $low\_y$ και $left\_x$ του συγκεκριμένου κόμβου.

5. Στη συνέχεια, οι τιμές αυτές συγκρίνονται με τις τρέχουσες ελάχιστες και μέγιστες τιμές $x$, $y$, ενημερώνοντας τις τιμές $maxX$, $minX$, $maxY$ και $minY$ ανάλογα. Η αρχική τιμή

αυτών των καταχωρητών τίθεται στην ελάχιστη δυνατή και τη μέγιστη δυνατή τιμή που μπορεί να αποθηκευτεί αντίστοιχα, έτσι ώστε η πρώτη σύγκριση να οδηγεί στην αποθήκευση των συντεταγμένων του πρώτου ακροδέκτη.

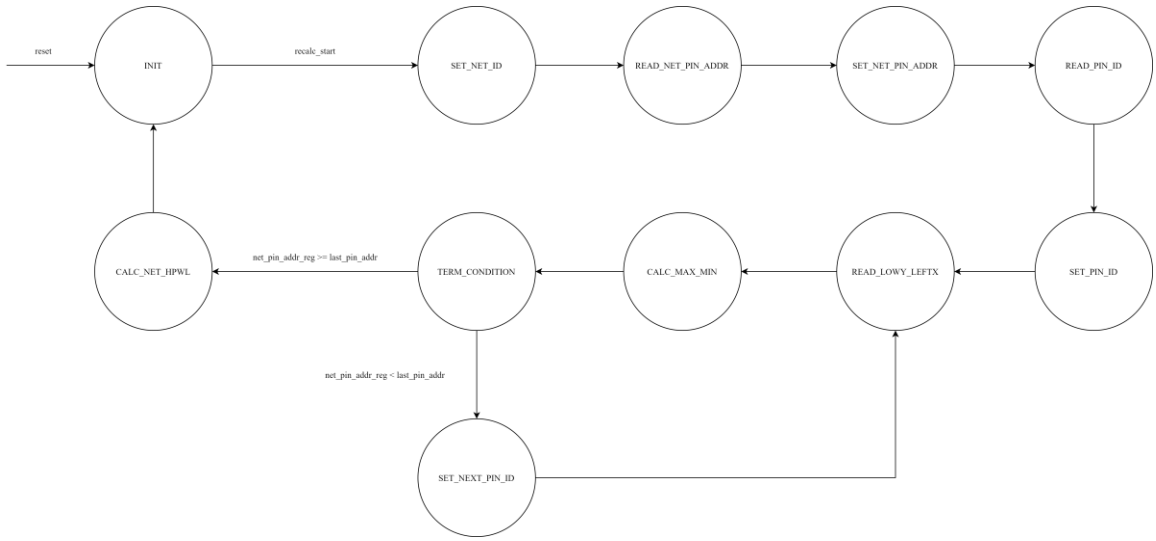
6. Εν συνεχεία, το $net\_pin\_addr$ αυξάνεται κατά 1 και όταν η τιμή είναι μικρότερη από την προηγουμένως αποθηκευμένη τιμή $last\_pin\_addr$, η πρόσβαση στη μνήμη αρχίζει εκ νέου από το βήμα 3. Διαφορετικά, όταν επιτευχθεί η συνθήκη τερματισμού, ο αλγόριθμος συνεχίζει με το έβδομο και τελευταίο βήμα.

7. Εφόσον έχουν ελεγχθεί πλέον οι τιμές $x$ και $y$ όλων των ακροδεκτών, ακολουθεί ο υπολογισμός του παρακάτω αθροίσματος:

$$maxX - minX + maxY - minY$$

που φυσικά ισούται με την τιμή της μετρικής HPWL του καλωδίου με αναγνωριστικό $net\_id$ και στη συνέχεια εξάγεται από τη μονάδα για περαιτέρω χρήση.

Αυτή ακριβής η διαδικασία διεκπεραιώνεται μέσω μιας μηχανής πεπερασμένων καταστάσεων (FSM - finite state machine), η οποία απεικονίζεται στο παρακάτω σχήμα.



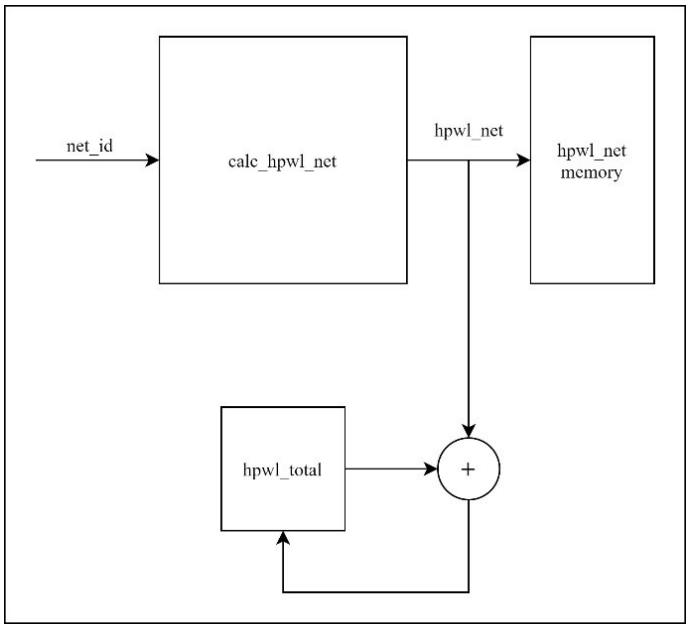

*Σχήμα 6-2 Μηχανή πεπερασμένων καταστάσεων (FSM) της μονάδας υπολογισμού HPWL*

## 6.3 Υπολογισμός αρχικού συνολικού HPWL

Μετά τη λήψη των πληροφοριών της τοποθέτησης μέσω της διεπαφής UART πρέπει να υπολογιστεί πρώτα η αρχική τιμή της μετρικής HPWL για κάθε καλώδιο της τοποθέτησης, καθώς και το συνολικό HPWL υπολογίζοντας το άθροισμά τους. Αυτό επιτυγχάνεται με τη μονάδα που περιγράφεται στην εν λόγω ενότητα.

Ο υπολογισμός της μετρικής πραγματοποιείται φυσικά από τη μονάδα που περιγράφηκε στην ακριβώς προηγούμενη ενότητα. Η μονάδα αρχικού υπολογισμού καλεί τη μονάδα υπολογισμού HPWL για κάθε δυνατή τιμή αναγνωριστικού καλωδίου $net\_id$, ξεκινώντας από την τιμή 0 και συνεχίζοντας ωσότου ο μετρητής καλωδίων έχει την τιμή $net\_count - 1$.

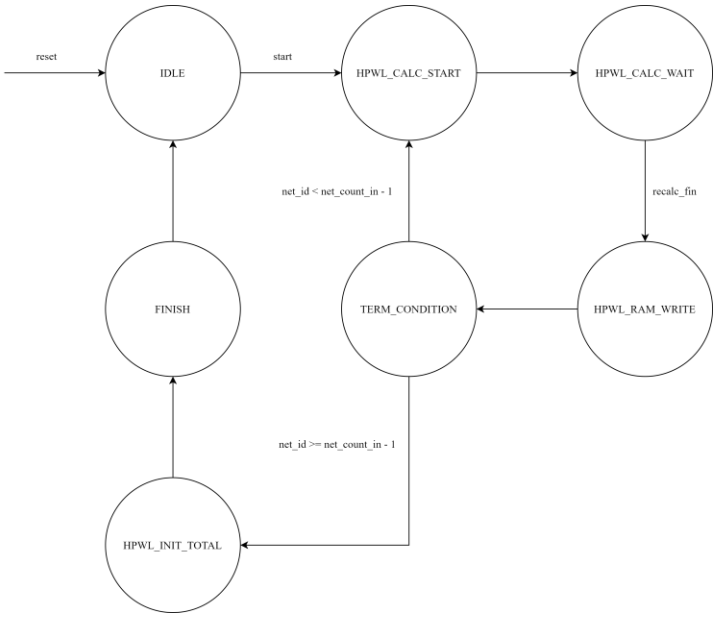
Η υπολογισμένη τιμή αποθηκεύεται στη μνήμη HPWL, ενώ προστίθεται και στην τρέχουσα τιμή του τοπικού καταχωρητή υπολογισμού συνολικού HPWL με ονομασία $hpwl\_total$, όπως απεικονίζεται και στο παρακάτω σχήμα.




*Σχήμα 6-3 Αναπαράσταση του τρόπου υπολογισμού της συνολικής μετρικής HPWL*

Μετά την ολοκλήρωση του υπολογισμού της μετρικής HPWL όλων των καλωδίων, η μονάδα αρχικού υπολογισμού ενημερώνει τον αντίστοιχο καταχωρητή συνολικού HPWL του ανώτατου επιπέδου ενεργοποιώντας το αντίστοιχο σήμα ελέγχου $hpwl\_total\_init$.

Η διαδικασία του αρχικού υπολογισμού διεξάγεται με τη βοήθεια μίας πολύ απλής μηχανής πεπερασμένων καταστάσεων, η οποία απεικονίζεται στο παρακάτω σχήμα.

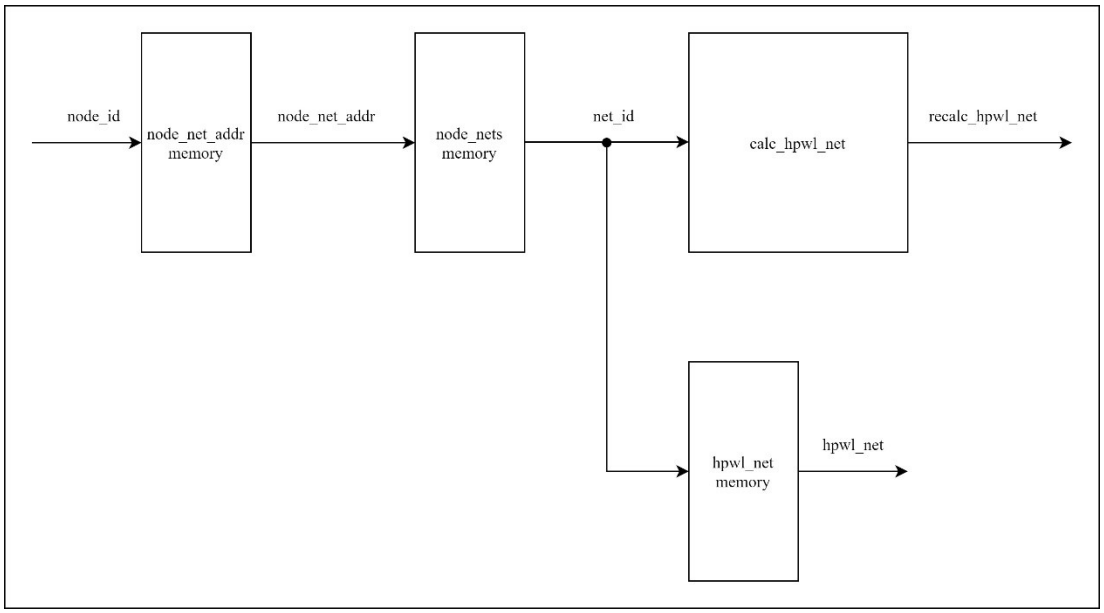



*Σχήμα 6-4 Μηχανή πεπερασμένων καταστάσεων (FSM) της μονάδας αρχικού υπολογισμού HPWL*

## 6.4 Επαναϋπολογισμός μόνο των επηρεαζόμενων καλωδίων

Ο επαναϋπολογισμός του HPWL μόνο για καλώδια που επηρεάζονται από την εναλλαγή δύο κόμβων γίνεται χρησιμοποιώντας τη μονάδα που περιγράφεται σε αυτήν την ενότητα. Φυσικά, χρησιμοποιεί την ίδια ακριβώς μονάδα υπολογισμού και προσπελάσει την ίδια μνήμη HPWL με τη μονάδα που περιγράφηκε στη προηγούμενη ενότητα, και επομένως πρέπει να δοθεί ιδιαίτερη προσοχή στην κοινή χρήση των πόρων.

Η μονάδα αυτή λαμβάνει ως είσοδο το αναγνωριστικό ενός κόμβου $node\_id$ και μετά από κατάλληλη πρόσβαση μνημών χρησιμοποιεί τη μονάδα υπολογισμού HPWL για τον υπολογισμό του νέου HPWL του κάθε καλωδίου, για το οποίο ο κόμβος αυτός αποτελεί ακροδέκτη στα πλαίσια της τοποθέτησης. Παράλληλα διαβάζεται και η τρέχον αποθηκευμένη τιμή του HPWL του ιδίου καλωδίου από την αντίστοιχη μνήμη και πραγματοποιείται και εξάγεται και το αποτέλεσμα της σύγκρισης, δηλαδή η θετική ή αρνητική αλλαγή της μετρικής. Στο τέλος, η νέα τιμή που υπολογίστηκε αποθηκεύεται στη μνήμη αντικαθιστώντας την προηγούμενη τιμή. Ο τρόπος με τον οποίο πραγματοποιείται η πρόσβαση των μνημών απεικονίζεται στο παρακάτω σχήμα.




*Σχήμα 6-5 Απεικόνιση των προσπελάσεων μνήμης που απαιτούνται για την ανάγνωση του αναγνωριστικού ενός καλωδίου του κόμβου και του τρόπου επαναϋπολογισμού του HPWL*

Καθώς μια εναλλαγή κόμβων απαιτεί δύο κόμβους, αυτή η μονάδα χρησιμοποιείται φυσικά δύο φορές, μία φορά για κάθε έναν από τους κόμβους που έχουν ανταλλάξει τις συντεταγμένες τους, έτσι ώστε ο επαναϋπολογισμός να πραγματοποιηθεί για το σύνολο των επηρεαζόμενων καλωδίων.

Για να κατανοηθεί καλύτερα ο τρόπος λειτουργίας της μονάδας, θα αναλυθούν παρακάτω τα βήματα της εκτέλεσης της μονάδας ένα προς ένα:

1. Η διεύθυνση του πρώτου καλωδίου για το οποίο ο δεδομένος κόμβος αποτελεί ακροδέκτη, $node\_net\_addr$, διαβάζεται από τη μνήμη διευθύνσεων καλωδίων πραγματοποιώντας προσπέλαση μνήμης με διεύθυνση το αναγνωριστικό του κόμβου $node\_id$.

2. Διότι ο αριθμός των καλωδίων είναι άγνωστος, ακολουθεί αντίστοιχη ανάγνωση και προσπέλαση και για τον αμέσως επόμενο κόμβο με προσπέλαση της διεύθυνσης

$node\_id + 1$. Έτσι αποθηκεύεται και η τιμή $last\_net\_addr$ η οποία στη συνέχεια χρησιμοποιείται ως συνθήκη τερματισμού.

3. Έπειτα, το αναγνωριστικό του καλωδίου $net\_id$ λαμβάνεται με προσπέλαση της μνήμης καλωδίων με διεύθυνση $node\_net\_addr$.

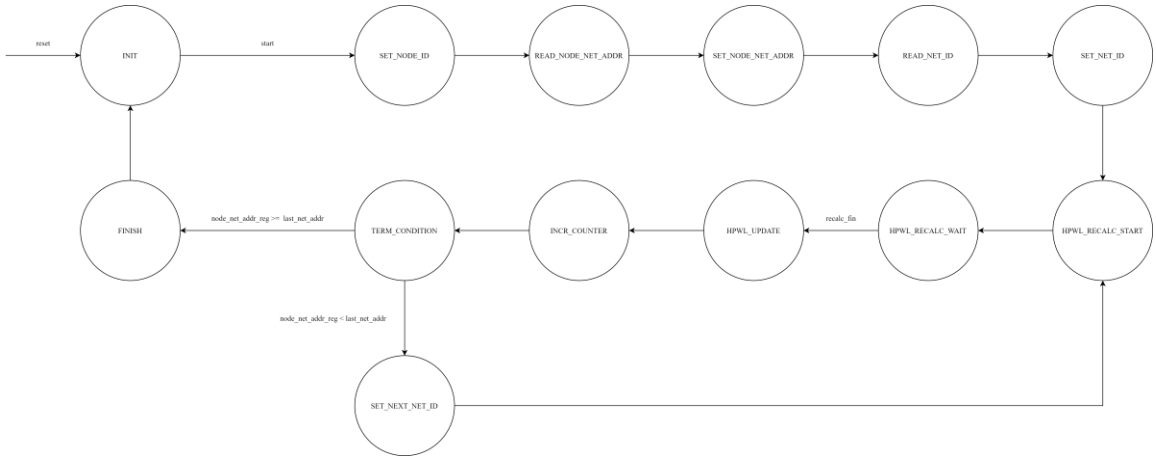
4. Έχοντας το αναγνωριστικό του καλωδίου είναι πλέον δυνατή η ανάγνωση της τρέχουσας τιμής της μετρικής HPWL $hpwl\_net$ του καλωδίου, ενώ στη συνέχεια το αναγνωριστικό παρέχεται και ως είσοδος στη μονάδα υπολογισμού HPWL, η οποία προχωρά στον επαναϋπολογισμό της μετρικής HPWL για το αντίστοιχο καλώδιο.

5. Αφού ολοκληρωθεί ο υπολογισμός, η νέα τιμή $recalc\_hpwl\_net$ αντικαθιστά την προηγουμένως αποθηκευμένη τιμή γράφοντάς τη στην αντίστοιχη διεύθυνση, ενώ υπολογίζεται και η διαφορά τους και προστίθεται έπειτα σε καταχωρητή που υπολογίζει τη συνολική αλλαγή στη μετρική, $hpwl\_change$.

6. Μετέπειτα, το $node\_net\_addr$ αυξάνεται κατά 1 και συνεχίζει η εκτέλεση από το βήμα 3 ωσότου λάβει την τιμή $last\_net\_addr$, όπου ο αλγόριθμος συνεχίζει με το έβδομο και τελευταίο βήμα.

7. Εξαγωγή της συνολικής μεταβολής στο HPWL που έχει υπολογιστεί για περαιτέρω χρήση από την μονάδα που πραγματοποιεί την εναλλαγή των κόμβων ή και την μονάδα που υλοποιεί τον αλγόριθμο βελτιστοποίησης .

Αυτή η λειτουργικότητα υλοποιείται φυσικά και πάλι με τη χρήση μιας FSM.




*Σχήμα 6-6 Μηχανή πεπερασμένων καταστάσεων (FSM) της μονάδας επαναϋπολογισμού επηρεαζόμενων καλωδίων.*

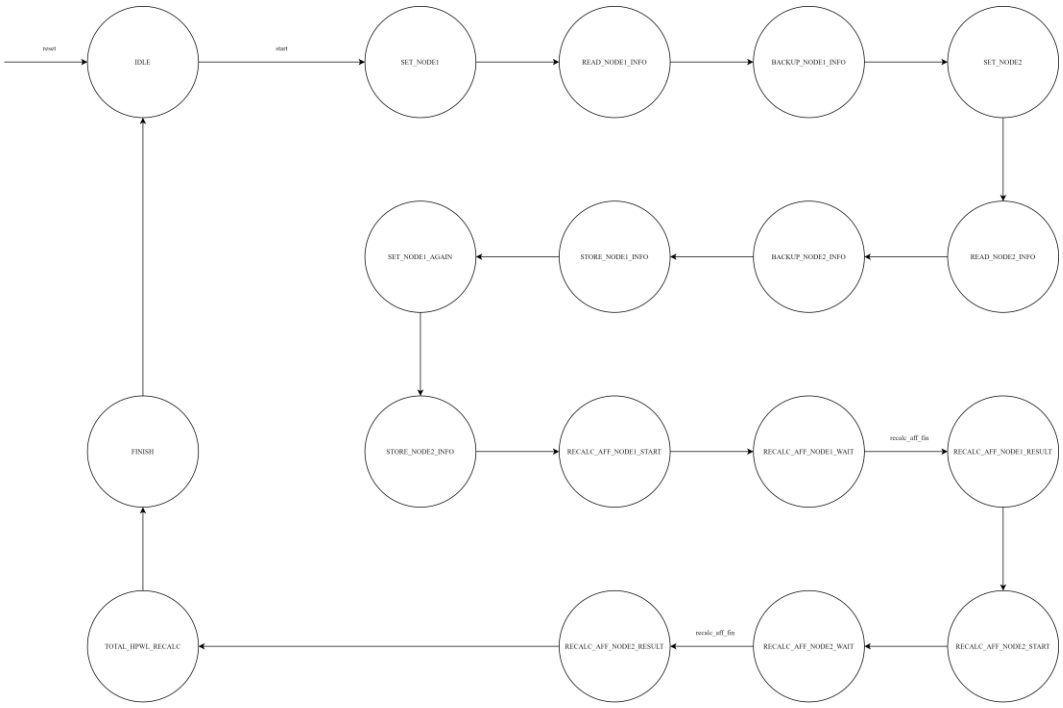
Για τον σκοπό της διαχείριση της πρόσβασης στη μνήμη HPWL και στην αντίστοιχη μονάδα επαναϋπολογισμού της μετρικής, ορίζεται στο ανώτερο επίπεδο της σχεδίασης ένα πρόσθετο FSM 2-καταστάσεων, το οποίο αρχικά (σε κατάσταση αδράνειας - *idle*) παρέχει πρόσβαση πόρων στη μονάδα αρχικού υπολογισμού, και αφότου αυτή ολοκληρώσει τον υπολογισμό και σηματοδοτήσει την ολοκλήρωση μέσω του αντίστοιχου σήματός της ($initial\_hpwl\_net\_calc\_fin$), η πρόσβαση μεταφέρεται στη συνέχεια στη μονάδα που επαναϋπολογίζει μόνο τα επηρεαζόμενα καλώδια.

## 6.5 Εναλλαγή κόμβων

Ο αλγόριθμος εναλλαγής διαχωρίζεται από τη λογική της εναλλαγής με δημιουργία μιας ξεχωριστής μονάδας που αφορά μόνο την εναλλαγή καθαυτή.

Πρακτικώς, δεν ελέγχονται οι προϋποθέσεις εναλλαγής, η μονάδα απλώς εναλλάσσει τους κόμβους και καλεί τις άλλες μονάδες που έχουν περιγραφεί πρωτύτερα για τον επαναϋπολογισμό του HPWL των επηρεαζόμενων καλωδίων. Οι προϋποθέσεις εναλλαγής κόμβων ελέγχονται επομένως από την μονάδα που υλοποιεί τον αλγόριθμο βελτιστοποίησης.

Εάν δε συμπεριληφθεί λογική αναίρεσης, τότε η μηχανή καταστάσεων για τη μονάδα εναλλαγής θα ήταν εξαιρετικά απλή, όπως απεικονίζεται και στο παρακάτω σχήμα.




*Σχήμα 6-7 Μηχανή πεπερασμένων καταστάσεων (FSM) της μονάδας εναλλαγής κόμβων*

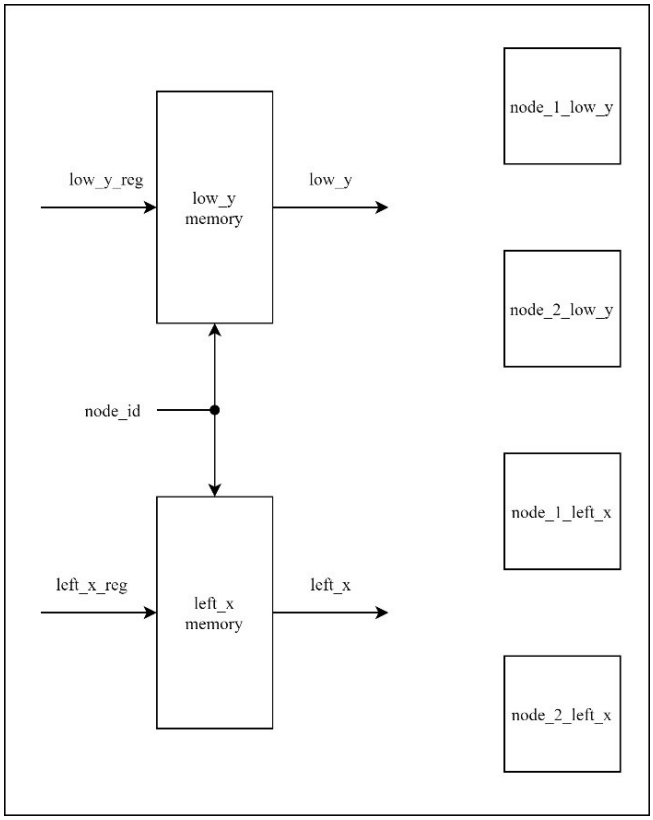
Φυσικά, χωρίς οποιαδήποτε λογική αναίρεσης, η χρήση της παραπάνω μονάδας είναι κατά βάση ανούσια στα πλαίσια της βελτιστοποίησης, ωστόσο η FSM χρησιμοποιείται ως βάση για τις FSM των μονάδων που προβαίνουν σε αναίρεση της εναλλαγής

Στη συνέχεια, διατυπώνονται δύο διαφορετικοί τρόποι αντιμετώπισης σχετικά με τον έλεγχο βελτίωσης της μετρικής:

1. Όταν δεν υπάρχει βελτίωση, η εναλλαγή των κόμβων αναιρείται και η μονάδα εξάγει ότι δεν υπήρξε βελτίωση, ενώ όταν υπάρχει βελτίωση η μονάδα εξάγει την αλλαγή που προκλήθηκε και ορίζει πως υπήρξε βελτίωση.

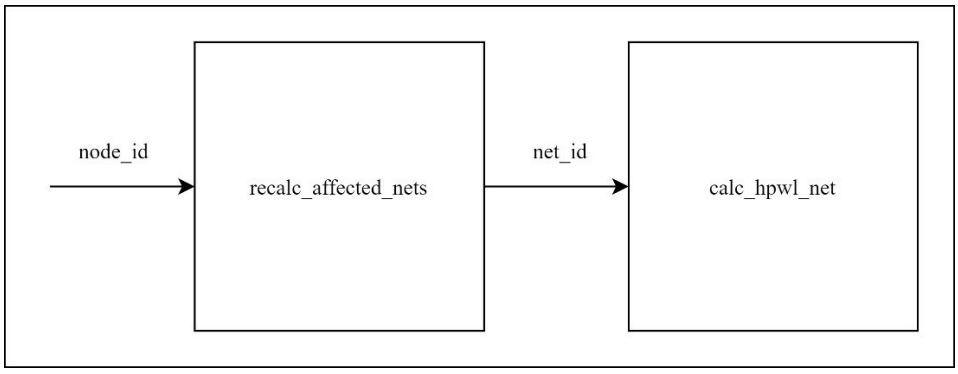
2. Η εναλλαγή αναιρείται όταν δεν υπάρχει βελτίωση ή όταν είναι ενεργό ένα συγκεκριμένο σήμα ελέγχου. Στη δεύτερη περίπτωση τοποθετείται η αλλαγή που προκλήθηκε στη μετρική στην έξοδο ενώ ακολουθεί αναίρεση ανεξάρτητα από το αποτέλεσμα της εναλλαγής. Όταν το αντίστοιχο σήμα δεν είναι ενεργό και υπάρχει βελτίωση, η δεδομένη βελτίωση που προκλήθηκε εξάγεται και σηματοδοτείται ανάλογα πως υπήρξε βελτίωση.

Προκειμένου να πραγματοποιηθεί η εναλλαγή και να είναι δυνατή η γρήγορη εκτέλεση της αναίρεσης, οι συντεταγμένες $x$ και $y$ των κόμβων που εναλλάσσονται αποθηκεύονται σε καταχωρητές πραγματοποιώντας αντίστοιχη προσπέλαση των μνημών $x$ και $y$. Στη συνέχεια, οι συντεταγμένες γράφονται ξανά στη μνήμη αποθηκεύοντας τις τιμές από τον κόμβο 1 στη θέση του κόμβου 2 και αντίστροφα, όπως φαίνεται στο παρακάτω σχήμα.




*Σχήμα 6-8 Τρόπος προσπέλασης μνημών x και y και καταχωρητές προσωρινής αποθήκευσης*

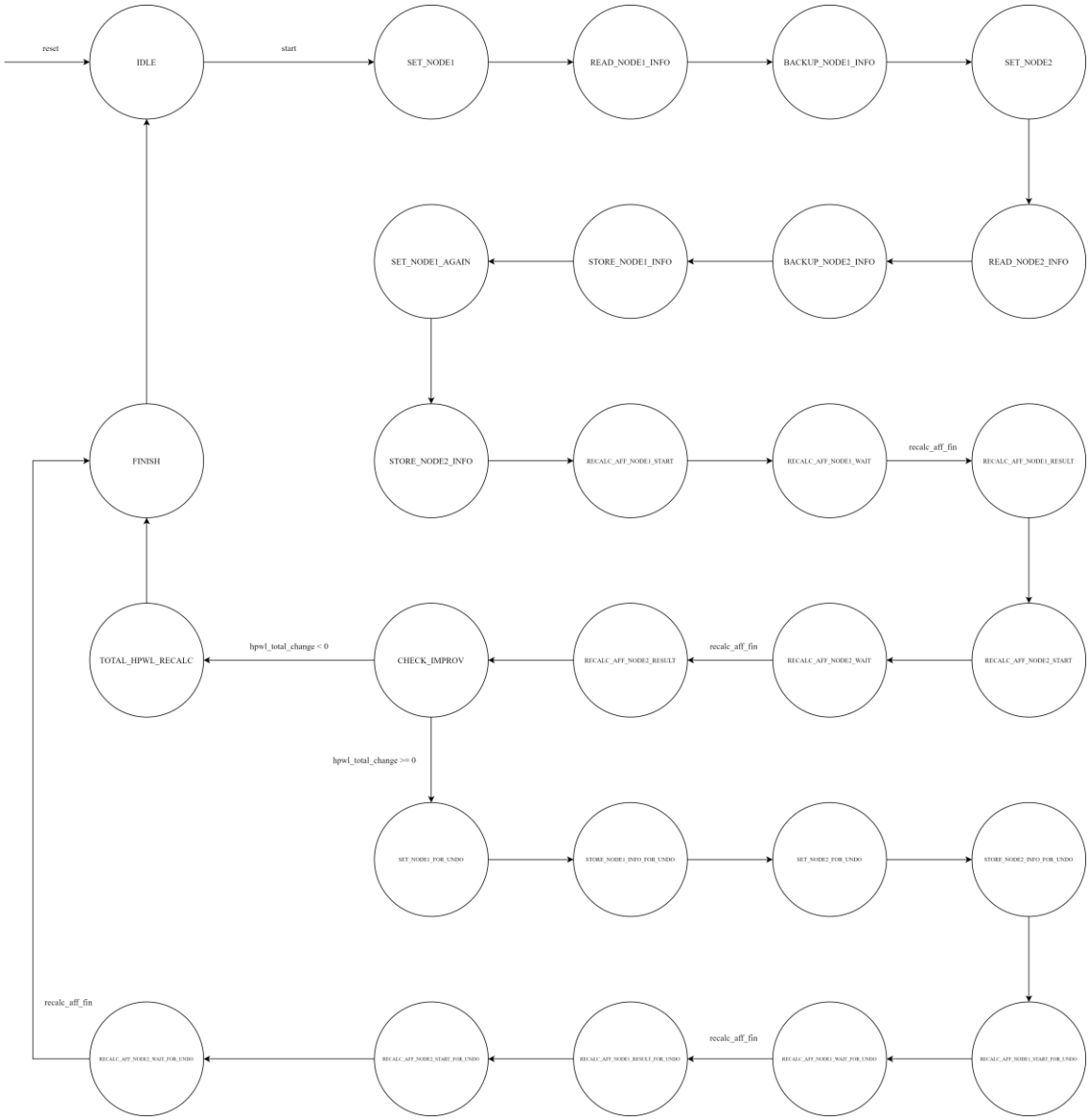
Μετά την εκτέλεση της εναλλαγής, καθένα από τα αναγνωριστικά κόμβου μεταβιβάζεται στη συνέχεια ως παράμετρος στη μονάδα επαναϋπολογισμού του HPWL των επηρεαζόμενων καλωδίων, η οποία με τη σειρά της χρησιμοποιεί τη μονάδα υπολογισμού, όπως φαίνεται παρακάτω.




*Σχήμα 6-9 Αναπαράσταση συνεργασίας των μονάδων επαναϋπολογισμού επηρεαζόμενων καλωδίων και υπολογισμού HPWL όπως αυτές χρησιμοποιούνται από την μονάδα εναλλαγής*

Προκειμένου να αναιρεθεί η εναλλαγή, οι τιμές για τον κόμβο 1 αποθηκεύονται και πάλι στην αρχική τους θέση. Ομοίως, και για τον κόμβο 2. Ενώ στη συνέχεια, ακολουθεί και πάλι επαναϋπολογισμός του HPWL, έτσι ώστε οι τιμές της μετρικής HPWL των καλωδίων να επαναφερθούν στις αρχικές τους τιμές.

Συμπεριλαμβάνοντας τη λογική αναίρεσης, η FSM διαμορφώνεται όπως φαίνεται παρακάτω.




*Σχήμα 6-10 Μηχανή πεπερασμένων καταστάσεων (FSM) μονάδας εναλλαγής κόμβων με αναίρεση*

Προκειμένου να υλοποιηθεί η λογική ελέγχου αναίρεσης, προστίθεται ο αντίστοιχος έλεγχος στην κατάσταση *CHECK_IMPROV*, το οποίο στην πράξη σημαίνει ότι οι συνθήκες για τη μετάβαση σε επόμενη κατάσταση τροποποιούνται ως εξής:

- $(hpwl\_total\_change < 0)\ AND\ (swap\_undo\_ctrl == 0)$.
- $(hpwl\_total\_change \geq 0)\ OR\ (swap\_undo\_ctrl == 1)$.

## 6.6 Υλοποίηση τεχνικών βελτιστοποίησης

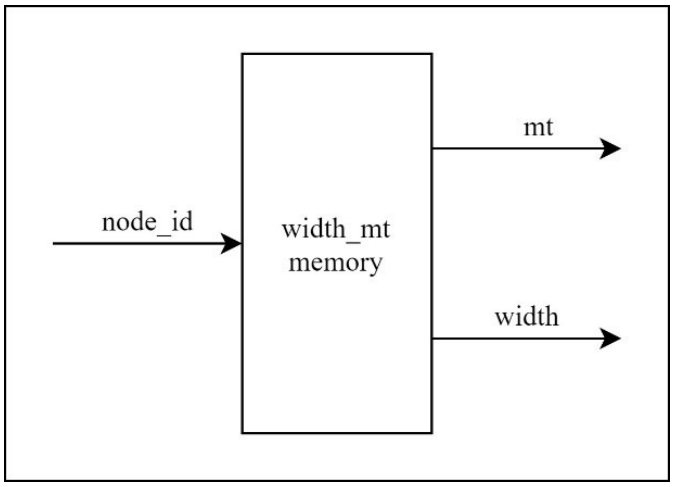
Έχοντας εξηγήσει με εκτενή τρόπο τη λειτουργικότητα των υπόλοιπων μονάδων της σχεδίασης, η υλοποίηση των αλγορίθμων βελτιστοποίησης της τοποθέτησης αποτελεί πλέον εύκολη υπόθεση.

Η μονάδα υλικού που υλοποιεί έναν αλγόριθμο βελτιστοποίησης στην πράξη χρησιμοποιεί τη μονάδα εναλλαγής κόμβων όταν πληρούνται οι συνθήκες εναλλαγής και ανάλογα το αποτέλεσμα της εναλλαγής προχωρά με τη δοκιμή του επόμενου υποψήφιου κελιού 2 ή το επόμενο ζεύγος υποψηφίων για εναλλαγή κελιών.

Για τις προϋποθέσεις εναλλαγής των κόμβων οι οποίες είναι οι εξής:

1. Οι κόμβοι δεν είναι τερματικοί.
2. Οι κόμβοι έχουν το ίδιο πλάτος.

η μονάδα προχωρά με την προσπέλαση της μνήμης που εμπεριέχει τον τύπο κίνησης και το πλάτος των κόμβων, όπως απεικονίζεται και στο παρακάτω σχήμα.




*Σχήμα 6-11 Προσπέλαση μνήμης πλάτους και τρόπου κίνησης*

Εάν δεν τηρείται η προϋπόθεση μην τερματικού κόμβου η μονάδα προχωρά απευθείας με τον έλεγχο του επόμενου κόμβου. Αν τα πλάτη των μην τερματικών κόμβων διαφέρουν τότε δοκιμάζεται ο επόμενος υποψήφιος κόμβος 2. Μόνο εάν πληρούνται και οι δύο προϋποθέσεις, η μονάδα προχωρά με την εναλλαγή τους με κατάλληλη χρήση της μονάδας εναλλαγής κόμβων.

Σαφώς ο καθένας από τους αλγορίθμους έχει τις ιδιαιτερότητες του και έτσι οι επόμενες ενότητες θα καλύψουν τον ακριβή τρόπο με τον οποίο υλοποιείται ο κάθε αλγόριθμος.

### 6.6.1 Πρώτο διαθέσιμο ιδίου μεγέθους

Ο αλγόριθμος εναλλαγής πρώτου διαθέσιμου κελιού ιδίους μεγέθους εκκινεί την διαδικασία βελτιστοποίησης με το πρώτο ζεύγος κόμβων, δηλαδή τους κόμβους με αναγνωριστικά 0 και 1. Οι τιμές αυτές αποθηκεύονται σε δύο αντίστοιχους μετρητές αναγνωριστικού κόμβου.

Το αναγνωριστικό κόμβου του πρώτου κόμβου χρησιμοποιείται έπειτα ως διεύθυνση για τη μνήμη πλάτους και τύπου κίνησης και αν ο κόμβος είναι τερματικός, η μονάδα προχωρά με το επόμενο ζεύγος κόμβων. Στην πράξη αυτό σημαίνει αύξηση του πρώτου μετρητή κατά 1 και επανεκκίνηση του δεύτερου μετρητή από την αρχή, δηλαδή την τιμή 0. Έτσι, το επόμενο ζεύγος κόμβων θα είναι εκείνο με αναγνωριστικά 1 και 0.

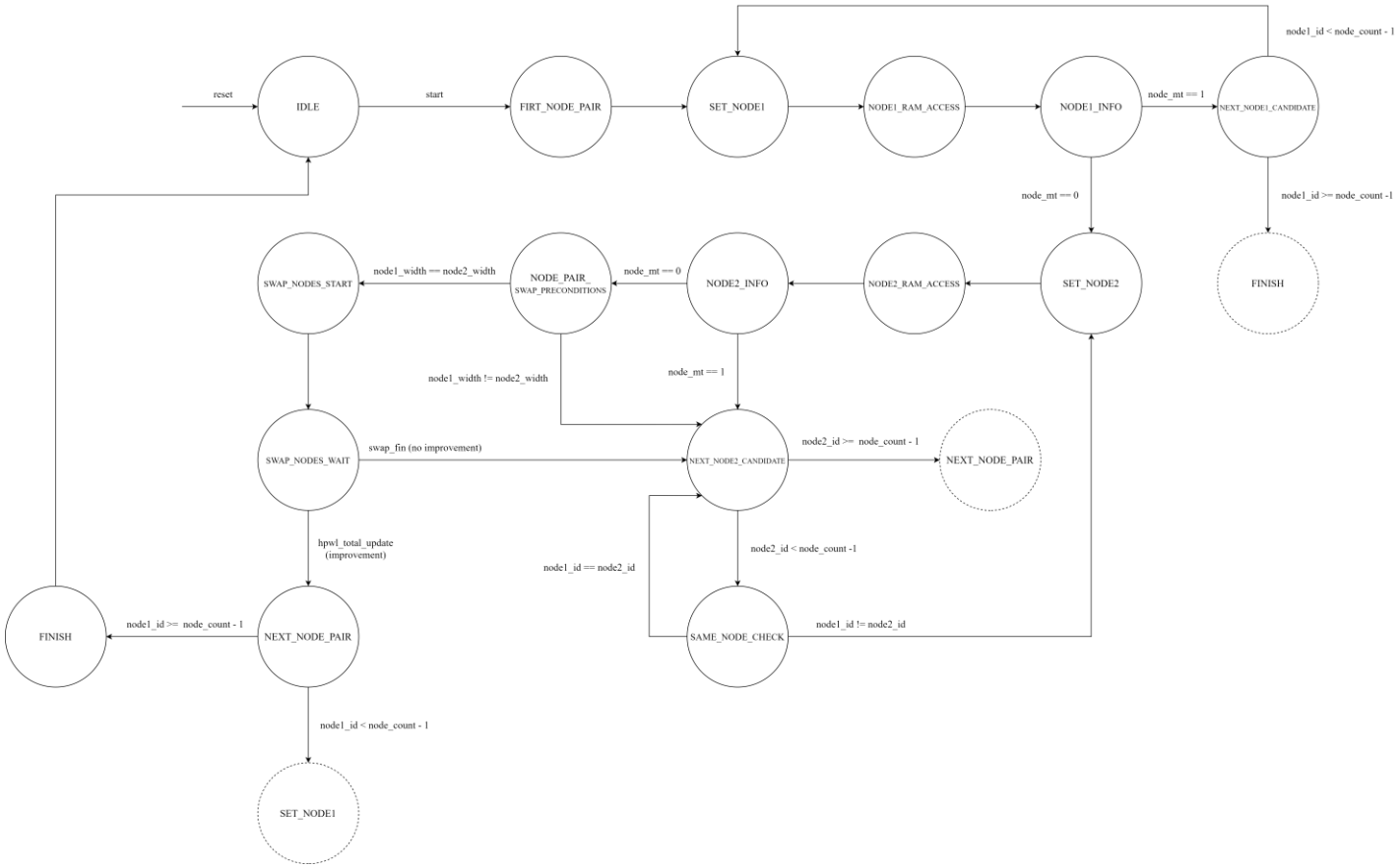
Εάν ο πρώτος κόμβος δεν είναι τερματικός (ο τύπος κίνησης είναι 0) τότε η μονάδα συνεχίζει με ανάγνωση των αντίστοιχων τιμών για τον δεύτερο κόμβο. Αν ο δεύτερος κόμβος δεν είναι τερματικός, έχει διαφορετικό αναγνωριστικό από τον πρώτο κόμβο και έχουν το ίδιο πλάτος μεταξύ τους τότε πραγματοποιείται εναλλαγή μεταξύ των κόμβων.

Αν οποιαδήποτε από τις προϋποθέσεις εναλλαγής δεν τηρείται για το δεύτερο κόμβο τότε η μονάδα προχωρά στην αύξηση του μετρητή του δεύτερου κόμβου και διαβάζει και πάλι τις τιμές επαναλαμβάνοντας τους ίδιους ελέγχους με πρωτύτερα. Το ίδιο συμβαίνει επίσης εάν η εναλλαγή των κόμβων δεν οδηγήσει σε βελτίωση της μετρικής HPWL. Μόνο όταν υπάρχει βελτίωση θα αυξηθεί η μετρητής του πρώτου κόμβου και θα πραγματοποιηθεί μεταφορά του δεύτερου μετρητή στην τιμή 0.

Όταν δεν υπάρχει επόμενος υποψήφιος για τον κόμβο 2, η μονάδα αυξάνει τον μετρητή του κόμβου 1 και επαναφέρει τον δεύτερο μετρητή στην τιμή 0. Αυτό μπορεί, για παράδειγμα, να συμβεί όταν δεν υπάρχει κάποιος υποψήφιος εναλλαγής για τον πρώτο κόμβο που να οδηγεί σε βελτίωση της μετρικής.

Η διαδικασία αυτή συνεχίζεται έως ότου δεν υπάρχει επόμενος υποψήφιος κόμβος 1, γεγονός που σηματοδοτεί την ολοκλήρωση της εκτέλεσης μιας πλήρους επανάληψης του αλγορίθμου.

Η λογική υλοποιείται μέσω μιας μηχανής πεπερασμένων καταστάσεων, η οποία απεικονίζεται στο παρακάτω σχήμα.




*Σχήμα 6-12 Μηχανή πεπερασμένων καταστάσεων (FSM) της μονάδας εναλλαγής πρώτου διαθέσιμου κελιού ιδίου μεγέθους*

### 6.6.2 Τελευταίο διαθέσιμο ιδίου μεγέθους

Για την υλοποίηση της παραλλαγής του αλγορίθμου εναλλαγής που εναλλάσσει με το τελευταίο κελί ιδίου μεγέθους, ο δεύτερος μετρητής θα πρέπει να μετρά αντίστροφα ξεκινώντας δηλαδή από μια τιμή αναγνωριστικού κόμβου που ισούται με τον αριθμό των κόμβων μείον 1.

Με αυτό συνεπάγεται επίσης και ότι το πρώτο ζεύγος κόμβων που ελέγχεται από τη μονάδα θα είναι το ζεύγος με αναγνωριστικά 0 και $node\_count - 1$.

Η υπόλοιπη λογική είναι κατά βάση η ίδια. Η μόνη διαφορά πρακτικά είναι η τιμή επανεκκίνησης του δεύτερου μετρητή που θα ισούται με $node\_count - 1$ αντί για 0, καθώς και το γεγονός πως έλεγχος του επόμενου υποψηφίου κελιού 2 θα σημαίνει μείωση του αντίστοιχου μετρητή κατά 1 αντί για αύξηση.

Στη μετάβαση καταστάσεων της αντίστοιχης FSM η διαφορά θα είναι ουσιαστικά ορατή μόνο στη λογική αλλαγής του μετρητή του δεύτερου κόμβου, όπου ο η συνθήκη ελέγχου ύπαρξης επόμενου υποψηφίου κόμβου εναλλαγής αλλάζει από $node2\_id \leq node\_count - 1$ σε $node2\_id > 0$.

Είναι δηλαδή εξαιρετικά εύκολη υπόθεση η συνένωση της εναλλαγής πρώτου και τελευταίου κελιού σε μία μονάδα, η οποία θα επιλέγει απλώς την κατάλληλη λογική του μετρητή του κόμβου 2 ανάλογα με την τιμή ενός σήματος ελέγχου. Περισσότερα για το συνδυασμό των παραλλαγών θα καλυφθούν σε επόμενη ενότητα.

### 6.6.3 Καλύτερο διαθέσιμο ιδίου μεγέθους

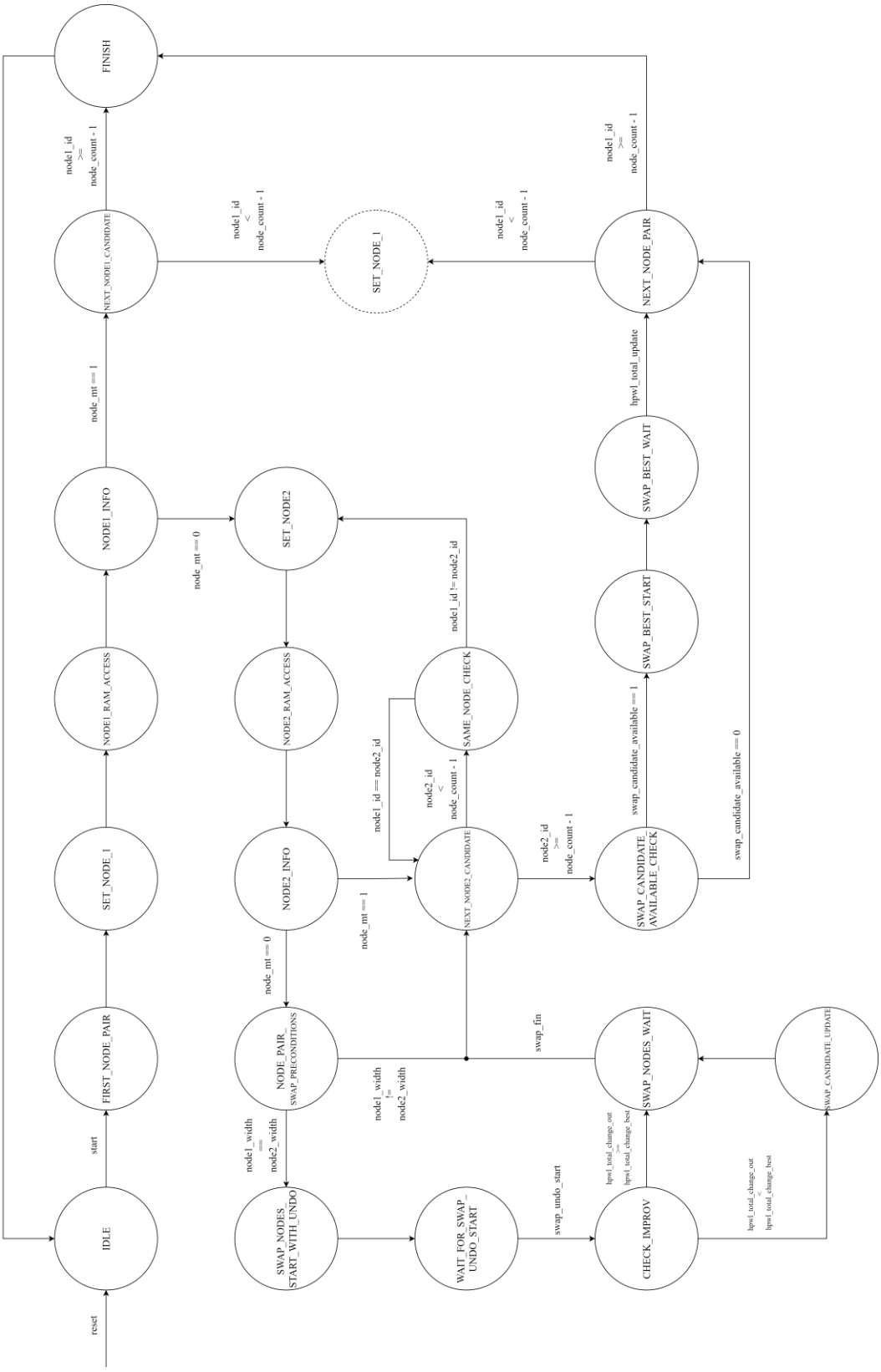
Ο αλγόριθμος εναλλαγής καλύτερου διαθέσιμου ίδιου μεγέθους λειτουργεί παρόμοια με τον αλγόριθμο πρώτου διαθέσιμου ίδιου μεγέθους από άποψη χρήσης των μετρητών κόμβων και τον τρόπο τροποποίησης των τιμών τους.

Η κύρια διαφορά της εκτέλεσης είναι πως η εναλλαγή αναιρείται πάντοτε και ενημερώνονται δύο αντίστοιχοι καταχωρητές όταν η βελτίωση που προκαλείται από την εναλλαγή με τον τρέχοντα υποψήφιο κόμβο είναι καλύτερη από αυτήν που προκλήθηκε τον τρέχον καλύτερο υποψήφιο κόμβο 2. Οι πληροφορίες που αποθηκεύεται είναι το αναγνωριστικό κόμβου του τρέχοντος καλύτερου κόμβου 2 καθώς και η συνολική αλλαγή/βελτίωση HPWL που προκλήθηκε από την εναλλαγή.

Αφού ελεγχθούν όλοι οι πιθανοί υποψήφιοι για τον κόμβο 2, η μονάδα ελέγχει αν υπάρχει γενικά υποψήφιος για εναλλαγή και πραγματοποιεί την εναλλαγή με τον συγκεκριμένο καλύτερο υποψήφιο κόμβο. Ο αλγόριθμος συνεχίζει με το επόμενο ζεύγος κόμβων αυξάνοντας τον μετρητή του πρώτου κόμβου κατά 1, έως ότου καλυφθούν όλοι οι δυνατοί υποψήφιοι κόμβοι 1, γεγονός που σηματοδοτεί το τέλος της επανάληψης του αλγορίθμου.

Στο αντίστοιχο FSM που υλοποιεί την εν λόγω λειτουργικότητα, η πρόσθετη αυτή λογική απαιτεί φυσικά την υλοποίηση πολλών νέων καταστάσεων. Πιο συγκεκριμένα, κατά τη διάρκεια του ελέγχου όλων των αντίστοιχων υποψηφίων κόμβων 2, η εναλλαγή γίνεται πάντα με αναίρεση, με αντίστοιχη ενημέρωση του καλύτερου υποψηφίου κόμβου, όταν η βελτίωση που προκαλείται από τον τρέχοντα ελεγχόμενο υποψήφιο είναι καλύτερη. Μόνο αφού ελεγχθούν όλοι οι πιθανοί υποψήφιοι, η μονάδα μεταβαίνει σε καταστάσεις στις οποίες εκτελείται η εναλλαγή με τον καλύτερο κόμβο και κατόπιν προχωρά με το επόμενο ζεύγος υποψηφίων κόμβων.

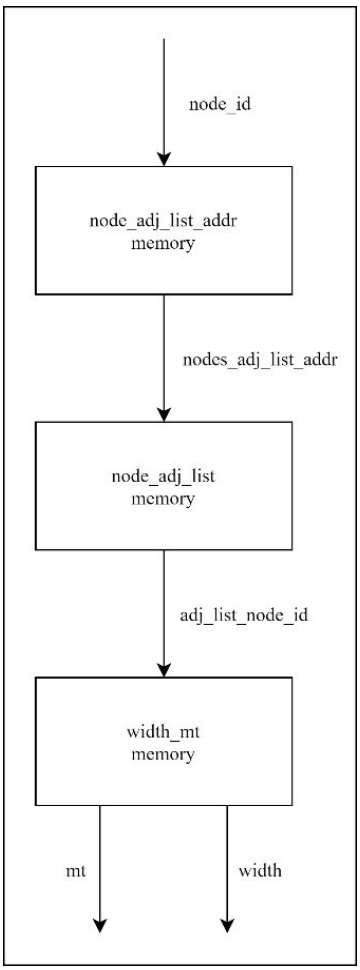
Η FSM που υλοποιεί τον αλγόριθμο εναλλαγής καλύτερου διαθέσιμου κελιού ίδιου μεγέθους εμφανίζεται συνολικά στην επόμενη σελίδα.




*Σχήμα 6-13 Μηχανή πεπερασμένων καταστάσεων (FSM) της μονάδας εναλλαγής καλύτερου διαθέσιμου κελιού ιδίου μεγέθους*

### 6.6.4 Γείτονες ιδίου μεγέθους

Προκειμένου να εφαρμοστεί ο αλγόριθμος εναλλαγής που λαμβάνει τον δεύτερο υποψήφιο κόμβο εναλλαγής από τη λίστα γειτνίασης του πρώτου υποψήφιου κόμβου πρέπει να πραγματοποιηθεί ανάλογη προσπέλαση μνήμης με αυτήν που εφαρμόστηκε και σε προηγούμενες μονάδες, όπως απεικονίζεται στο παρακάτω σχήμα.




*Σχήμα 6-14 Απεικόνιση των προσπελάσεων μνήμης που απαιτούνται για την ανάγνωση του αναγνωριστικού κόμβου της λίστας γειτνίασης και του αντίστοιχου πλάτους και είδους κίνησής του*

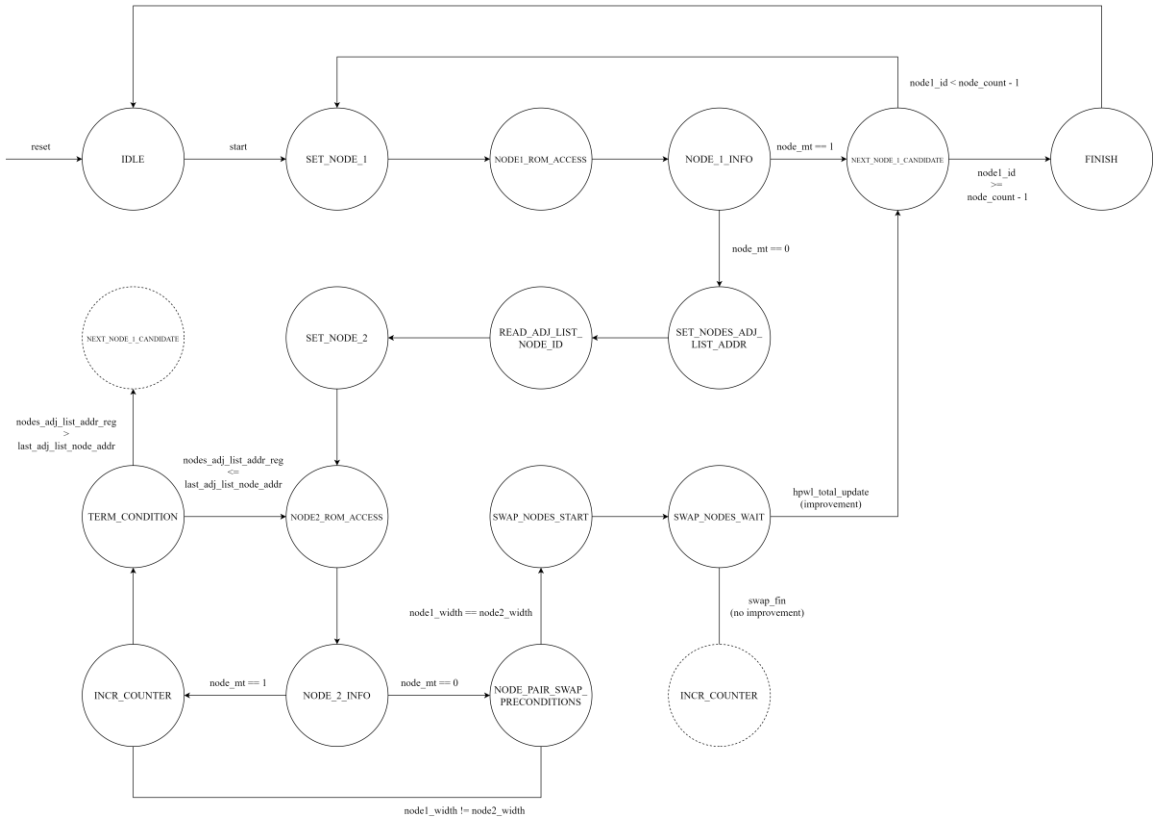
Εφόσον ελεγχθεί η συνθήκη μετακινησιμότητας του πρώτου κόμβου πραγματοποιείται επομένως προσπέλαση της μνήμης διευθύνσεων της αντίστοιχης μνήμης για να διαβαστούν η τιμή της διεύθυνσης του πρώτου κόμβου της λίστας γειτνίασης για τον πρώτο κόμβο, $nodes\_adj\_list\_addr$, καθώς και για τον κόμβο με αναγνωριστικό με τιμή 1 παραπάνω από εκείνον, $last\_adj\_list\_node\_addr$, η οποία θα χρησιμοποιηθεί έπειτα ως συνθήκη ελέγχου επόμενου υποψηφίου κόμβου 2.

Η διεύθυνση $nodes\_adj\_list\_addr$ χρησιμοποιείται έπειτα για προσπέλαση της μνήμης που εμπεριέχει τους κόμβους της λίστας γειτνίασης και έχοντας πλέον την τιμή του αναγνωριστικού του δεύτερου κόμβου μπορεί να πραγματοποιηθούν κανονικά οι έλεγχοι για την τήρηση των προϋποθέσεων εναλλαγής και να συνεχιστεί η διαδικασία με παρόμοιο τρόπο όπως και στις προηγούμενες παραλλαγές.

Φυσικά, υπάρχει πλέον μόνο ένας μετρητής κόμβων που σχετίζεται με τον πρώτο κόμβο και για έλεγχο του επόμενου υποψηφίου πρώτου κόμβου πραγματοποιείται κανονικά αύξηση του κατά 1.

Ο επόμενος υποψήφιος για τον δεύτερο κόμβο λαμβάνεται πάντοτε μέσω πρόσβαση μνήμης. Η αντίστοιχη διεύθυνση αυξάνεται επομένως κατά 1 μέχρις ότου λάβει την τιμή τερματισμού $last\_adj\_list\_node\_addr$ που σηματοδοτεί πως δεν υπάρχει άλλος διαθέσιμος κόμβος για έλεγχο.

Η FSM που υλοποιεί τον αλγόριθμο εναλλαγής μεταξύ γειτονικών κελιών ίδιου μεγέθους απεικονίζεται στο παρακάτω σχήμα.




*Σχήμα 6-15 Μηχανή πεπερασμένων καταστάσεων (FSM) της μονάδας εναλλαγής μεταξύ γειτόνων ιδίου μεγέθους*

### 6.6.5 Συνδυασμός τεχνικών βελτιστοποίησης

Λόγω των μικρών διαφορών που παρουσιάζουν μεταξύ τους, οι τρεις πρώτες τεχνικές μπορούν εύκολα να συνδυαστούν σε μια ενιαία μονάδα.

Με την εισαγωγή ενός σήματος ελέγχου αντιστροφής ($inverted$) είναι εύκολη η εναλλαγή μεταξύ των παραλλαγών της εναλλαγής μεταξύ του πρώτου και του τελευταίου διαθέσιμου ίδιου μεγέθους. Ομοίως, μπορεί επίσης να εισαχθεί ένα σήμα ελέγχου "καλύτερου" ($best$) για να επιτραπεί η εναλλαγή μεταξύ των παραλλαγών πρώτου και καλύτερου κόμβου εναλλαγής σε μια ενιαία μονάδα. Αυτή ακριβώς η λογική χρησιμοποιήθηκε κατά τη διάρκεια των δοκιμών, χρησιμοποιώντας μια παραλλαγή που επιτρέπει την εναλλαγή μεταξύ του πρώτου, του τελευταίου και του καλύτερου διαθέσιμου ίδιου μεγέθους, με απλή μεταβολή των διακοπτών στην πλακέτα του FPGA.

## 6.7 Διασύνδεση UART και μνημών

Στο πλαίσιο της παρούσας διατριβής, η διεπαφή UART γράφτηκε από το μηδέν με βάση ποικίλες υλοποιήσεις που διατίθενται στο διαδίκτυο. [8] [9].

Η λειτουργία αποστολής και λήψης μέσω UART έχει υλοποιηθεί σε μία μόνο μονάδα, η οποία λαμβάνει το αντίστοιχο σήμα χρονισμού (*tick*) από μια μονάδα-γεννήτρια ρυθμού μετάδοσης (*baud rate generator*).

Κατά την επικοινωνία μέσω διεπαφής UART με δειγματοληψία, ένα $tick$ αποστέλλεται μετά την πάροδο του ακόλουθου αριθμού κύκλων ρολογιού (*clock cycles*):

$$\frac{Συχνότητα\ ρολογιού}{Ρυθμός\ μετάδοσης\ \times\ Ρυθμός\ δειγματοληψίας} = \frac{Clock\ Frequency}{Baud\ Rate\ \times Sampling\ rate}$$

Στους συνήθεις ρυθμούς μετάδοσης περιλαμβάνονται οι τιμές 1200, 2400, 4800, 19200, 38400, 57600 και 115200. Στην παρούσα διατριβή χρησιμοποιήθηκε ο τελευταίος.

Καθώς τα FPGA που χρησιμοποιήθηκαν κατά τη διάρκεια των δοκιμών έχουν συχνότητα ρολογιού $100\ MHz$ και έχει επιλεγεί ρυθμός δειγματοληψίας 16, αποστέλλεται ένα $tick$ κάθε 54 κύκλους ρολογιού.

Έτσι, τα δεδομένα μεταφέρονται κάθε 16 $tick$ και η δειγματοληψία του πρώτου byte δεδομένων εισόδου πραγματοποιείται στο όγδοο $tick$ για λόγους "ευθυγράμμισης" της επικοινωνίας.

Εκτός από την επιλογή μεταξύ μεταφοράς και λήψης, η διεπαφή που κατασκευάστηκε έχει επίσης υλοποιηθεί και με τρόπο τέτοιο που επιτρέπει τη μεταφορά ενός ή δύο byte δεδομένων μέσω της αλλαγής ενός αντίστοιχου σήματος ελέγχου.

Η μονάδα αυτή μπορεί φυσικά να γενικευτεί εύκολα σε οποιοδήποτε αριθμό bytes, και στάθηκε ιδιαίτερα χρήσιμη στο πλαίσιο των τοποθετήσεων που δοκιμάστηκαν, καθώς η μεταφορά πλάτους και τύπου κίνησης απαιτούσε μεταφορά δεδομένων 1 byte, ενώ όλες οι άλλες μεταφορές 2 bytes.

### 6.7.1 Μόνο μνήμες X και Y

Η διεπαφή UART που περιγράφηκε στην προηγούμενη ενότητα χρησιμοποιείται από τις μονάδες διεπαφής μνήμης προς UART. Η απλούστερη μονάδα, η οποία θα περιγραφεί εκτενώς σε αυτήν την ενότητα μεταφέρει και λαμβάνει τις τιμές των συντεταγμένων $x$ και $y$. Για τη λήψη των δεδομένων ενός αντίστοιχου κόμβου μέσω UART η διαδικασία έχει ως εξής:

1. Το λογισμικό μεταφέρει τη συντεταγμένη $low\_y$.
2. Το FPGA λαμβάνει τη συντεταγμένη $low\_y$.
3. Το FPGA αποθηκεύει τη συντεταγμένη $low\_y$ σε έναν προσωρινό καταχωρητή.
4. Το λογισμικό μεταφέρει τη συντεταγμένη $left\_x$.
5. Το FPGA λαμβάνει τη συντεταγμένη $left\_x$.
6. Το FPGA αποθηκεύει τη συντεταγμένη $left\_x$ σε έναν προσωρινό καταχωρητή.
7. Το FPGA γράφει τις συντεταγμένες $low\_y$ και $left\_x$ στις μνήμες $x$ και $y$.

Η διαδικασία αυτή επαναλαμβάνεται φυσικά για όλους τους κόμβους με τελικό στόχο την λήψη των αντίστοιχων συντεταγμένων όλων τον κόμβων.

Για τη μεταφορά την τοποθέτησης από το FPGA προς το λογισμικό ακολουθείται η ακριβώς αντίθετη πορεία, δηλαδή πρώτα ανάγνωση των τιμών από τις αντίστοιχες μνήμες και έπειτα μεταφορά του $low\_y$ και του $left\_x$ για κάθε δεδομένο κόμβο μέσω της διεπαφής, όπως παρατίθεται επακριβώς παρακάτω:

1. Το FPGA διαβάζει τις συντεταγμένες $low\_y$ και $left\_x$ από τις μνήμες $x$ και $y$.
2. Το FPGA αποθηκεύει τις τιμές $low\_y$ και $left\_x$ σε προσωρινούς καταχωρητές.
3. Το FPGA μεταφέρει τη συντεταγμένη $low\_y$.
4. Το λογισμικό λαμβάνει τη συντεταγμένη $low\_y$.
5. Το FPGA μεταφέρει τη συντεταγμένη $left\_x$.
6. Το λογισμικό λαμβάνει τη συντεταγμένη $left\_x$.

### 6.7.2 Όλες οι μνήμες

Η μεταφορά όλων των πληροφοριών και η αποθήκευσή τους στην αντίστοιχη μνήμη κάθε φορά, πραγματοποιείται με την ίδια ακριβώς λογική. Η μονάδα θα είναι φυσικά πολύ μεγαλύτερη και με πολύ περισσότερες εισόδους και εξόδους.

Πρακτικώς έχουν οριστεί οχτώ διαφορετικοί τρόποι χρήσης ($mode$) της διεπαφής, οι οποίοι λειτουργού οι οποίες παρατίθεται στο παρακάτω πίνακα.

| **Σήμα "$mode$"** | **Λειτουργία διεπαφής** |
|---|---|
| 0 | Λήψη τιμών παραμέτρων (αριθμός κόμβων, καλωδίων και ακροδεκτών) |
| 1 | Λήψη του πλάτους και τύπου κίνησης κάθε κόμβου |
| 2 | Λήψη των καλωδίων των κόμβων |
| 3 | Λήψη των διευθύνσεων προς τα καλώδια των κόμβων |
| 4 | Λήψη των ακροδεκτών των καλωδίων |
| 5 | Λήψη των διευθύνσεων προς τους ακροδέκτες των καλωδίων |
| 6 | Λήψη των συντεταγμένων $x$ και $y$ των κόμβων |
| 7 | Μεταφορά των συντεταγμένων $x$ και $y$ των κόμβων |

*Πίνακας 6-2 Τρόποι λειτουργίας διεπαφής μνημών-UART*

Η σχεδίαση του ανώτερου επιπέδου προβαίνει σε ανάλογη χρήση της διεπαφής με σκοπό τη λήψη των πληροφοριών τοποθέτησης, την εκτέλεση της βελτιστοποίησης και στο τέλος τη μεταφορά της νέας τοποθέτησης μέσω της διεπαφής.

## 6.8. Συνολικό κύκλωμα

Με βάση τις μονάδες που έχουν υλοποιηθεί, δεν υπάρχει μία μόνο μονάδα ανώτατου επιπέδου. Έχει χρησιμοποιηθεί μια σύμβαση ονοματοδοσίας προκειμένου να προσδιοριστεί τι είναι δυνατό με κάθε υλοποίηση.

Καθώς υπάρχουν 4 κύριοι αλγόριθμοι βελτιστοποίησης, η ονομασία αρχίζει ως εξής:

- FASS : Εναλλαγή με πρώτο διαθέσιμο ιδίου μεγέθους (First Available Same Size).
- LASS : Εναλλαγή με τελευταίο διαθέσιμο ιδίου μεγέθους (Last Available Same Size).
- BASS : Εναλλαγή με καλύτερο διαθέσιμο ιδίου μεγέθους (Best Available Same Size).
- SNSS : Εναλλαγή μεταξύ γειτόνων ιδίου μεγέθους (Swap Neighbors of Same Size).

Εάν είναι δυνατή η εναλλαγή μεταξύ της παραλλαγής πρώτου και τελευταίου διαθέσιμου κελιού με τη χρήση του σήματος "$inverted$", προστίθεται στην ονομασία το αγγλικό γράμμα “I” και προκύπτει η ονομασία FASSI. Εάν είναι δυνατή η εναλλαγή μεταξύ της παραλλαγής πρώτου και καλύτερου διαθέσιμου κελιού με χρήση του σήματος "$best$", προστίθεται το αγγλικό γράμμα “B” και προκύπτει το FASSB. Τέλος, εάν διατίθενται και τα δύο αυτά σήματα επιτρέποντας την εναλλαγή μεταξύ των παραλλαγών πρώτου, τελευταίου και καλύτερου διαθέσιμου κελιού ιδίου μεγέθους η ονομασία που προκύπτει είναι FASSIB.

Περαιτέρω κατηγοριοποίηση γίνεται ανάλογα με την επικοινωνία UART. Εάν μεταφέρονται και λαμβάνονται μόνο οι τιμές $x$ και $y$ τότε ακολουθούν τα γράμματα “XY”. Εάν η σχεδίαση περιλαμβάνει πλήρη διασύνδεση μνήμης ακολουθεί το γράμμα “M”. Τέλος, αν περιλαμβάνεται πλήρης διεπαφή μνήμης και οι μνήμες είναι επίσης προ-συμπληρωμένες και αρχικοποιημένες με τις τιμές μιας δεδομένης τοποθέτησης και επιτρέπουν την παράλειψη της μεταφοράς όλων των αμετάβλητων πληροφοριών της τοποθέτησης με τη χρήση του αντίστοιχου σήματος “$skip\_placement\_info$” τότε το αγγλικό γράμμα “P” ακολουθεί το “M”.

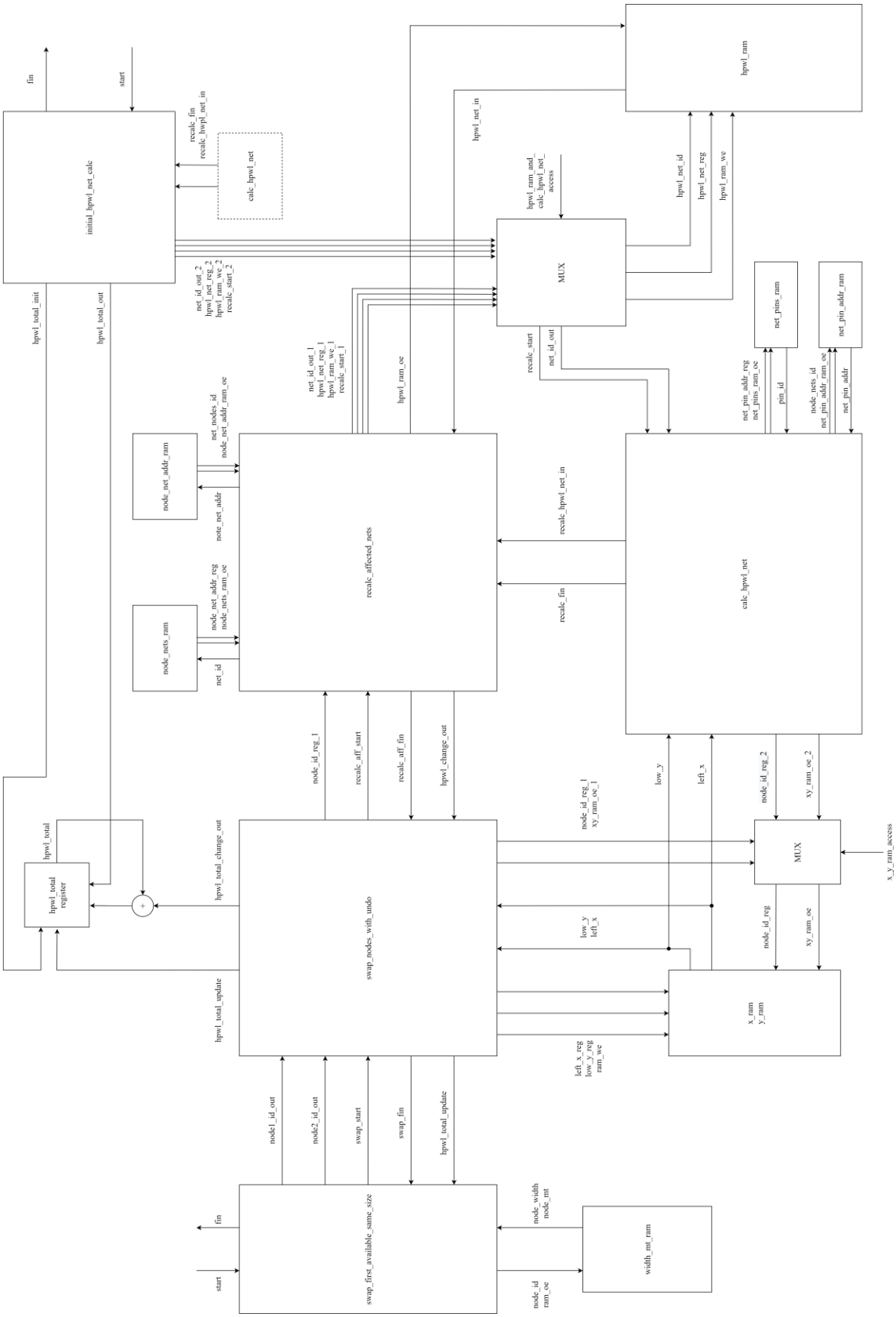
Ακολουθώντας αυτή τη σύμβαση ονοματοδοσίας, η ονομασία FASSIBMP αντιστοιχεί σε σχεδίαση που επιτρέπει την εναλλαγή μεταξύ των παραλλαγών πρώτου, τελευταίου και καλύτερου διαθέσιμου κελιού, έχει μνήμες που είναι αρχικοποιημένες με τις τιμές μιας δεδομένης τοποθέτησης και διαθέτει πλήρη διεπαφή μνήμης, με προαιρετική επιλογή λήψης μόνο των συντεταγμένων $x$ και $y$, παραλείποντας την λήψη των υπόλοιπων πληροφοριών της τοποθέτησης.

Αντίστοιχα η ονομασία SNSSXY περιγράφει την σχεδίαση που υλοποιεί τον αλγόριθμο εναλλαγής μεταξύ γειτόνων ιδίου μεγέθους και διαθέτει απλή διεπαφή UART για λήψη και μεταφορά των συντεταγμένων $x$ και $y$.

Αξίζει να σημειωθεί σε αυτό το σημείο πως κάθε σχεδίαση που περιλαμβάνει τα γράμματα XY διαθέτει αναγκαστικά μνήμες που είναι προ-συμπληρωμένες και αρχικοποιημένες με τις τιμές μιας δεδομένης τοποθέτησης καθώς διαφορετικά δε θα ήταν δυνατή η εκτέλεση οποιασδήποτε βελτιστοποίησης.

Προκειμένου να παρουσιαστεί ο ακριβής τρόπος διασύνδεσης των μονάδων, στην επόμενη σελίδα παρουσιάζεται η υλοποίηση του αλγορίθμου εναλλαγής πρώτου διαθέσιμου κελιού ίδιου μεγέθους, παραλείποντας οποιαδήποτε λογική διασύνδεσης UART.

Επίσης, παρουσιάζονται στο σχήμα και οι πολυπλέκτες πρόσβασης σε μνήμες RAM και μονάδες. Είναι σημαντικό να σημειωθεί σε αυτό το σημείο ότι χρειάζονται περισσότεροι τέτοιου είδους πολυπλέκτες όταν περιλαμβάνεται και η διεπαφή UART.




*Σχήμα 6-16 Σχηματική αναπαράσταση συνολικού κυκλώματος βελτιστοποίησης*

# 7. Προσομοίωση Υλικού

Κάθε μονάδα υλικού της υλοποίησης έχει δοκιμαστεί διεξοδικά με ειδικά σχεδιασμένα testbench χρησιμοποιώντας το εργαλείο προσομοίωσης Icarus Verilog [10]. Με αυτόν τον τρόπο, πραγματοποιήθηκε πλήρη επικύρωση της λειτουργικότητας των υπο-μονάδων της υλοποίησης ήδη προτού ανέβει αυτή σε κάποιο FPGA.

## 7.1 Υπολογισμός HPWL καλωδίου

Προκειμένου να δοκιμαστεί η μονάδα υπολογισμού HPWL των καλωδίων, ο υπολογισμός πραγματοποιείται για όλα τα καλώδια μιας τοποθέτησης που έχει μεταφερθεί στις αντίστοιχες μνήμες μέσω κατάλληλης χρήση της συνάρτησης συστήματος $\$readmemh()$.

Για παράδειγμα, για την ανάγνωση των εκ των προτέρων υπολογισμένων αρχικών τιμών της μετρικής HPWL κάθε καλωδίου, η οποία πραγματοποιήθηκε μέσω του μοντέλου λογισμικού, απαιτούνται οι ακόλουθες γραμμές HDL κώδικα Verilog:

```
reg [HPWL_NET_WIDTH - 1 : 0] hpwl_ram [2**CONST_WIDTH - 1 : 0];

initial
begin
   $readmemh("../mem/nets_hpwl.mem", hpwl_ram, 0, NET_COUNT - 1);
end
```

Έτσι, όποτε η μονάδα υπολογισμού του HPWL ενός καλωδίου ολοκληρώνει τον υπολογισμό η υπολογισθείσα τιμή εξόδου μπορεί να συγκρίνεται πολύ εύκολα με την τιμή που έχει υπολογιστεί μέσω του λογισμικού πρωτύτερα. Για να πραγματοποιηθεί έλεγχος της ορθής λειτουργίας του κυκλώματος, ο προσομοιωτής εκτυπώνει μήνυμα στο παράθυρο του τερματικού όταν η υπολογιζόμενη τιμή διαφέρει. Εάν δεν εκτυπωθεί τίποτα στο παράθυρο του τερματικού καθ' όλη τη διάρκεια της εκτέλεσης της προσομοίωσης, τότε η μονάδα υλικού έχει σχεδιαστεί σωστά.

Ακολουθεί απόσπασμα του αντίστοιχου testbench:

```
for (i = 0; i < NET_COUNT; i++)    // Βρόχος για όλα τα καλώδια
begin

   // Καθορισμός αναγνωριστικού καλωδίου
   net_id = i;

   // Σηματοδότηση έναρξης υπολογισμού
   recalc_start = 1;
   #2;
   recalc_start = 0;

   // Αναμονή μέχρι πέρας υπολογισμού
   while (recalc_fin == 0) #2;

   #2;

   // Σύγκριση τιμών
   if (hpwl_net_out != hpwl_net)
   begin
      // Εκτύπωση μηνύματος
   end
end

$finish;
```

Για τη μεταγλώττιση του testbench απαιτείται η ακόλουθη γραμμή τερματικού κώδικα:

```
iverilog -o calc_hpwl_net_tb calc_hpwl_net_tb.v calc_hpwl_net.v x_ram.v y_ram.v net_pins_rom.v net_pin_addr_rom.v hpwl_ram.v
```

Η εκτέλεση της προσομοίωσης γίνεται μέσω της γραμμής:

```
vvp calc_hpwl_net_tb
```

## 7.2 Υπολογισμός αρχικού συνολικού HPWL

Η μονάδα υπολογισμού συνολικού HPWL μπορεί να προσομοιωθεί με δύο τρόπους:

1. Κάθε φορά που η μονάδα αποστέλλει το σήμα ελέγχου για την εκκίνηση του υπολογισμού του HPWL του καλωδίου το testbench διαβάζει την τιμή απευθείας από μνήμη και την παρέχει στη μονάδα. Με αυτόν τον τρόπο ελέγχεται μόνο η ορθότητα του υπολογισμού του αθροίσματος των μετρικών HPWL και η ενημέρωση του αντίστοιχου καταχωρητή.

2. Η μονάδα προσομοιώνεται παράλληλα με την μονάδα υπολογισμού του HPWL και ελέγχεται η συνολική λειτουργικότητα και των δύο μονάδων.

Και στις δύο περιπτώσεις, δεδομένου ότι το βασικό ενδιαφέρον του testbench είναι η λειτουργικότητα της μονάδας συνολικού υπολογισμού HPWL, η έξοδος στο παράθυρο του τερματικού αρκεί να είναι η τελική τιμή του συνολικού HPWL που παρέχει η μονάδα ως έξοδο μετά τον υπολογισμό του αθροίσματος. Πιο συγκεκριμένα, αυτό που πρέπει να εκτυπωθεί είναι πρακτικώς η τιμή του καταχωρητή που ενημερώνεται όταν ανυψώνεται το αντίστοιχο σήμα αρχικοποίησης του συνολικού HPWL.

Η τιμή αυτή μπορεί στη συνέχεια να συγκριθεί με την τιμή που υπολογίζεται από το μοντέλο λογισμικού προκειμένου να επαληθευτεί η λειτουργικότητα της μονάδας.

Φυσικά, για τον σκοπό αυτό αρκεί η εκτύπωση μιας γραμμής της παρακάτω μορφής:

```
HPWL:  929267
```

Η παραπάνω γραμμή εκτυπώνεται μέσω της ακόλουθης γραμμής HDL κώδικα Verilog:

```
$display("HPWL: %d", hpwl_total);
```

## 7.3 Επαναϋπολογισμός μόνο των επηρεαζόμενων καλωδίων

Παρόμοια με την προηγούμενη μονάδα που ελέγχθηκε, η μονάδα επαναϋπολογισμού μόνο των επηρεαζόμενων καλωδίων μπορεί και αυτή με τη σειρά της να προσομοιωθεί με δύο τρόπους:

1. Κάθε φορά που η μονάδα ενεργοποιεί τον επαναϋπολογισμό για ένα συγκεκριμένο καλώδιο η τιμή διαβάζεται από μνήμη και παρέχεται απευθείας στην μονάδα.

2. Η μονάδα προσομοιώνεται παράλληλα με την μονάδα υπολογισμού του HPWL και ελέγχεται η συνολική λειτουργικότητα και των δύο μονάδων.

Μιας και δεν έχει γίνει ακόμη καμία αλλαγή στην τοποθέτηση, με εναλλαγή κόμβων για παράδειγμα, η αλλαγή στο HPWL θα πρέπει να είναι μηδενική. Για να ελεγχθεί λοιπόν η ορθή λειτουργίας της μονάδας θα πρέπει σε και τις δύο παραλλαγές του testbench να επαληθευθεί πως όλες οι μεταβολές είναι 0.

Όπως και κατά του προηγούμενους ελέγχους ορθότητας, ο συγκεκριμένος έλεγχος μπορεί να υλοποιηθεί πολύ εύκολα με την εκτύπωση της συνολικής μεταβολής HPWL για τα επηρεαζόμενα καλώδια ενός κόμβου στο παράθυρο του τερματικού όταν η μεταβολή δεν είναι μηδενική.

## 7.4 Εναλλαγή κόμβων

Όπως εξηγήθηκε προηγουμένως, στο πλαίσιο της παρούσας διατριβής έχουν σχεδιαστεί τρεις διαφορετικές μονάδες εναλλαγής κόμβων:

1. Εναλλαγή των κόμβων και εκ νέου υπολογισμός του HPWL των επηρεαζόμενων καλωδίων τους χωρίς περαιτέρω ελέγχους.

2. Εναλλαγή των κόμβων, υπολογισμός του νέου HPWL και αναίρεση της εναλλαγής στην περίπτωση που δεν υπάρχει βελτίωση.

3. Εναλλαγή των κόμβων, υπολογισμός του νέου HPWL και αναίρεση της εναλλαγής στην περίπτωση που δεν υπάρχει βελτίωση ή εάν ένα συγκεκριμένο σήμα ελέγχου είναι υψηλό.

Η πρώτη μονάδα δε χρησιμοποιείται κάπου στη σχεδίαση, αλλά ο σχεδιασμός ενός testbench που ελέγχει τη λειτουργικότητά της αποτελεί εύκολη υπόθεση και επεκτείνεται έπειτα εύκολα στις άλλες μονάδες εναλλαγής.

Για τον έλεγχο της σωστής εναλλαγής μπορούν τυπωθούν στο παράθυρο του τερματικού οι συντεταγμένες $x$ και $y$ δύο κόμβων, να γίνει η εναλλαγή τους με χρήση της μονάδας και στη συνέχεια να ελεγχθεί αν οι συντεταγμένες $x$ και $y$ έχουν τροποποιηθεί σωστά.

Για παράδειγμα η εκτέλεση της προσομοίωσης με παράμετρο τους κόμβους με αναγνωριστικά 3 και 168, και αντίστοιχες συντεταγμένες $(314, 112)$, $(333, 256)$, η έξοδος της προσομοίωσης θα είναι της μορφής:

```
NodeID:   3   Low Y:  112   Left X:  314
NodeID: 168   Low Y:  256   Left X:  333

NodeID:   3   Low Y:  256   Left X:  333
NodeID: 168   Low Y:  112   Left X:  314
```

Δίχως να χρειάζεται να πραγματοποιηθεί επαναϋπολογισμός του HPWL, ο οποίος θα απαιτούσε τη διασύνδεση και όλων των άλλων μονάδων, είναι δυνατόν να ελεγχθεί η λειτουργικότητα των άλλων μονάδων εναλλαγής με το πέρασμα διαφορετικών τιμών αλλαγής HPWL $hpwl\_change$ και τιμών σήματος ελέγχου $swap\_undo\_ctrl$ αντίστοιχα.

Εάν έχει σχεδιαστεί σωστά, η δεύτερη μονάδα εναλλαγής θα αναιρεί την εναλλαγή όταν η αλλαγή στο HPWL είναι μεγαλύτερη ή ίση με μηδέν. Στην οθόνη του τερματικού θα πρέπει να τυπωθούν αντίστοιχα δύο πανομοιότυπες έξοδοι.

Ομοίως, η τρίτη και τελευταία μονάδα εναλλαγής θα πρέπει να αναιρεί την εναλλαγή εάν δεν υπάρχει βελτίωση ή εάν υπάρχει βελτίωση αλλά το σήμα $swap\_undo\_ctrl$ έχει τιμή 1.

## 7.5 Εναλλαγή με πρώτο διαθέσιμο ιδίου μεγέθους

Η δοκιμή των αλγορίθμων βελτιστοποίησης, όπως ο πρώτος διαθέσιμος του ίδιου μεγέθους, μπορεί φυσικά να υλοποιηθεί με ποικίλους τρόπους.

Καθώς ο αλγόριθμος βελτιστοποίησης βρίσκεται ουσιαστικά στο ανώτατο επίπεδο της σχεδίασης, δεν έχει ιδιαίτερη αξία να πραγματοποιηθεί μια προσομοίωση που να περιλαμβάνει όλες τις μονάδες, καθώς αυτό θα πραγματοποιηθεί ήδη σε ένα testbench της συνολικής σχεδίασης.

Επομένως, για τον έλεγχο της λειτουργικότητας της μονάδας εναλλαγής με πρώτο διαθέσιμο κόμβο ιδίου μεγέθους, το testbench διατρέχει ουσιαστικά όλους τους μην τερματικούς κόμβους και εναλλάσσει με τον πρώτο διαθέσιμο κόμβο που πληροί τις προϋποθέσεις εναλλαγής, που είναι δηλαδή ίσου πλάτους, δεν είναι τερματικός και έχει διαφορετικό αναγνωριστικό. Με αυτόν τον τρόπο το testbench ελέγχει αν ο αλγόριθμος ελέγχει σωστά τις προϋποθέσεις εναλλαγής.

Αυτό σημαίνει πως το testbench θα πρέπει να ενεργοποιεί πάντοτε το σήμα ελέγχου $hpwl\_total\_update$, το οποίο ενεργοποιείται κανονικά μόνο όταν υπάρχει βελτίωση και παρέχεται αντίστοιχα τότε μόνο από τη μονάδα εναλλαγής κόμβων μετά την ολοκλήρωση της εναλλαγής και τον επαναϋπολογισμό του HPWL.

Στο παράθυρο του τερματικού είναι δυνατόν να εκτυπωθούν οι εναλλαγές που πραγματοποιήθηκαν και να ελεγχθεί στη συνέχεια αν οι κόμβοι που εναλλάχθηκαν πληρούσαν τις προϋποθέσεις της εναλλαγής.

## 7.6 Εναλλαγή με τελευταίο διαθέσιμο ιδίου μεγέθους

Η δοκιμή αυτής της μονάδας δεν απαιτεί διαφορετικό χειρισμό από την προηγούμενη μονάδα. Η μόνη διαφορά θα είναι ουσιαστικά το τι εκτυπώνεται στο παράθυρο του τερματικού όταν εκτελείται η προσομοίωση. Αν αυτό που εκτυπώνει η μονάδα εναλλαγής πρώτου διαθέσιμου κόμβου ίδιου μεγέθους κατά την προσομοίωση της είναι γραμμές της μορφής:

```
⋮
swapped  175 (0000010101111) and    6 (0000000000110)
swapped  176 (0000010110000) and   11 (0000000001011)
swapped  177 (0000010110001) and    9 (0000000001001)
⋮
```

τότε η μονάδα εναλλαγής τελευταίου διαθέσιμου κόμβου ιδίου μεγέθους θα εξάγει με τη σειρά της αντίστοιχες γραμμές της μορφής:

```
⋮
swapped  175 (0000010101111) and 1184 (0010010100000)
swapped  176 (0000010110000) and 1188 (0010010100100)
swapped  177 (0000010110001) and 1185 (0010010100001)
⋮
```

## 7.7 Εναλλαγή με καλύτερο διαθέσιμο ιδίου μεγέθους

Αυτή η μονάδα απαιτεί ειδικό χειρισμό, καθώς δοκιμάζονται για κάθε κόμβο όλοι οι δυνατοί υποψήφιοι κόμβοι εναλλαγής, προτού πραγματοποιηθεί η τελική εναλλαγή με εκείνον που οδηγεί στην καλύτερη βελτίωση του HPWL.

Για να επιτευχθεί προσομοίωση του βέλτιστου κόμβου εναλλαγής, η συνολική μεταβολή του HPWL, η οποία κανονικά παρέχεται κάθε φορά από τη μονάδα εναλλαγής μετά την ολοκλήρωση της εναλλαγής και τον αντίστοιχο επαναϋπολογισμό του HPWL των καλωδίων, λαμβάνει τυχαίες τιμές βελτίωσης μέσω της συνάρτησης συστήματος $\$random()$.

Εκτελώντας την προσομοίωση με παράμετρο την ίδια ακριβώς τοποθέτηση με τις προηγούμενες μονάδες, η έξοδος στο τερματικό θα είναι της μορφής:

```
⋮
swapped   175 (00000010101111) and   396 (00000110001100)
swapped   176 (00000010110000) and  1066 (00010000101010)
swapped   177 (00000010110001) and   602 (00001001011010)
⋮
```

## 7.8 Εναλλαγή μεταξύ γειτόνων ιδίου μεγέθους

Η δοκιμή αυτής της μονάδας υλικού δεν απαιτεί ιδιαίτερες διαφορές στον χειρισμό. Η μόνη διαφορά που παρουσιάζει αυτή η μονάδα με τις προηγούμενες μονάδες είναι ότι οι υποψήφιοι κόμβοι για εναλλαγή λαμβάνονται από τη λίστα γειτνίασης που είναι αποθηκευμένη για κάθε κόμβο στις αντίστοιχες μνήμες, όπως περιγράφηκε ήδη και στην αντίστοιχη ενότητα.

Με αυτόν τον τρόπο, η εναλλαγή θα πραγματοποιείται μόνο όταν οι υποψήφιοι κόμβοι έχουν το ίδιο πλάτος, είναι γείτονες, δεν αντιπροσωπεύουν τον ίδιο κόμβο και η εναλλαγή τους οδηγεί σε βελτίωση του συνολικού HPWL της τοποθέτησης.

Προσομοιώνοντας ότι κάθε έγκυρος υποψήφιος οδηγεί επίσης σε βελτίωση, η έξοδος στο τερματικό θα πρέπει να είναι κόμβοι όπου ο δεύτερος κόμβος είναι πάντα στη λίστα γειτνίασης του πρώτου για να επαληθευτεί η λειτουργικότητα του κυκλώματος.

Για παράδειγμα, εάν ένας δεδομένος κόμβος με αναγνωριστικό 5 έχει μια λίστα γειτνίασης που περιλαμβάνει τα αναγνωριστικά κόμβων $65,\ 267,\ 499,\ 1024$ και $1115$, τότε εάν η έξοδος του τερματικού υποδεικνύει την εναλλαγή του 5 με οποιονδήποτε κόμβο που δε συμπεριλαμβάνεται στη λίστα γειτνίασης τους, το κύκλωμα θα θεωρείται πως δε λειτουργεί ορθά.

Αν το κύκλωμα λειτουργεί σωστά τότε αν θα υπάρχει έξοδος στην οθόνη του τερματικού με εναλλαγή του κόμβου με αναγνωριστικό 5 στην πρώτη θέση τότε αυτή η εναλλαγή θα πρέπει να πραγματοποιείται μεταξύ του κόμβου 5 και του πρώτου κόμβου της λίστας γειτνίασης του που έχει το ίδιο πλάτος με αυτόν.

Ως εκ τούτου, εάν για παράδειγμα ο κόμβος με αναγνωριστικό 499 είναι ο πρώτος σε αύξουσα σειρά κόμβος που έχει το ίδιο πλάτος με τον κόμβο 5, η έξοδος θα ήταν η ακόλουθη:

```
⋮
swapped   5 (00000000000101) and   499 (00000111110011)
⋮
```

## 7.9 Διασύνδεση UART

Για την επαλήθευση της λειτουργικότητας της μονάδας UART, το πρώτο testbench που σχεδιάστηκε ελέγχει αν η γεννήτρια ρυθμού baud της UART βγάζει πραγματικά ένα tick μόνο όταν πρέπει με βάση το ρολόι του συστήματος, τον ρυθμό baud και τον ρυθμό δειγματοληψίας.

Για παράδειγμα, εάν το ρολόι του συστήματος είναι 100 MHz, ο ρυθμός baud είναι 115200 και ο ρυθμός δειγματοληψίας είναι 16, τότε ένα tick θα πρέπει να εξάγεται κάθε (100 * 1000000) / (115200 * 16) = 54 κύκλους ρολογιού.

Ως εκ τούτου, εάν το testbench περιμένει 54, 53, 1, 1, 53 περιόδους ρολογιού αντίστοιχα, πριν από την εμφάνιση της τιμής του tick, η ακολουθία που θα πρέπει να λαμβάνεται στο παράθυρο του τερματικού είναι η ακόλουθη: 10101.

Προκειμένου να επαληθευτεί η λειτουργικότητα της διασύνδεσης UART, η οποία είναι μια σύνθετη διασύνδεση που μπορεί να χρησιμοποιηθεί για τη λήψη και τη μετάδοση είτε ενός είτε δύο bytes ανάλογα με τις παραμέτρους εισόδου, η μονάδα υποβλήθηκε σε βαριές δοκιμές τόσο για τη λήψη όσο και για τη μετάδοση, συμπεριλαμβανομένων των μεταφορών ενός και δύο bytes, προκειμένου να διασφαλιστεί ότι ο χρονισμός της διασύνδεσης είναι σωστός.

Καθώς οι διεπαφές μνήμης προς UART αξιοποιούν απευθείας τη διεπαφή UART για τη μεταφορά δεδομένων, δεν είναι απαραίτητο να πραγματοποιηθεί οποιαδήποτε επίπονη δοκιμή ούτε για τη διεπαφή των μνημών x και y προς UART, ούτε και για την πλήρη διεπαφή όλων των μνημών προς UART. Η λειτουργικότητα αυτών των διεπαφών επαληθεύτηκε φυσικά κατά τη δοκιμή της σχεδίασης σε FPGA.

## 7.10 Συνολικό κύκλωμα

Για την επαλήθευση της λειτουργικότητας του πλήρους κυκλώματος είναι απαραίτητο να συνδεθούν όλες οι υπο-μονάδες της συνολικής σχεδίασης υλικού, και καθώς υπάρχουν τόσες πολλές διαφορετικές παραλλαγές (19 συνολικά) στο πλαίσιο της αναφοράς, θα εξεταστεί μόνο η επαλήθευση της παραλλαγής FASSIB. Η παραλλαγή αυτή επιτρέπει την επιλογή μεταξύ των αλγορίθμων που εναλλάσσουν το πρώτο, τελευταίο και καλύτερο διαθέσιμο κελί ιδίου μεγέθους.

Προκειμένου να απλοποιηθεί το testbench, η επικοινωνία UART δεν έχει προσομοιωθεί και ως εκ τούτου, όλες οι μνήμες, εκτός από τη μνήμη RAM για αποθήκευση της μετρικής HPWL των καλωδίων, είναι προ-συμπληρωμένες με τις τιμές μιας δεδομένης τοποθέτησης.

Μετά την επαναφορά του κυκλώματος μέσω του σήματος επαναφοράς, το αμέσως επόμενο βήμα είναι η εκκίνηση και η αναμονή για τον αρχικό υπολογισμό του HPWL, ενώ ακολουθεί η εκτέλεση μιας επανάληψης του αλγορίθμου βελτιστοποίησης. Στο τέλος της εκτέλεσης της προσομοίωσης, η οποία για την περίπτωση του αλγορίθμου εναλλαγής καλύτερου διαθέσιμου κελιού έχει απαγορευτικό χρόνο εκτέλεσης τυπώνεται στην οθόνη του τερματικού η υπολογισμένη τελική τιμή του HPWL. Επίσης, εκτυπώνεται και η αρχική τιμή του HPWL που προκύπτει μετά την ολοκλήρωση του αρχικού υπολογισμού. Επιπλέον, καταγράφονται και όλες οι εναλλαγές που πραγματοποιούνται για λόγους σύγκρισης με την αντίστοιχη έξοδο του λογισμικού.

Παρακάτω θα εξεταστεί και προσομοιωθεί η βελτιστοποίηση της ίδιας ακριβώς τοποθέτησης που εξετάστηκε και στην αντίστοιχη ενότητα 5.5 του λογισμικού.

Η αλλαγή μεταξύ της παραλλαγής εναλλαγής μεταξύ πρώτου, τελευταίου και καλύτερου κελιού ιδίους μεγέθους πραγματοποιείται αλλάζοντας την τιμή των εισόδων $best\_reg$ και $inverted\_reg$, όπου τιμή ‘1’ στο πρώτο σήμα υπονοεί ανεξάρτητα από την τιμή του άλλου σήματος χρήση της

παραλλαγής καλύτερου διαθέσιμου, ενώ με κατάλληλη εναλλαγή μεταξύ του ‘0’ και του ‘1’ επιτρέπεται η παραλλαγή μεταξύ του πρώτου και τελευταίου διαθέσιμου.

Ακολουθεί ένα απόσπασμα του αντίστοιχου κώδικα Verilog του testbench:

```
initial
   begin
      clk = 0;
      rst = 0;
      best_reg = 1;
      inverted_reg = 1;
      swap_first_last_best_avail_same_size_start = 0;
      initial_hpwl_net_calc_start = 0;

      #1;
      rst = 1;
      #2;
      rst = 0;
      #1;

      initial_hpwl_net_calc_start = 1;
      #2;
      initial_hpwl_net_calc_start = 0;

      while (initial_hpwl_net_calc_fin == 0) #2;

      $display("HPWL total:", hpwl_total);

      swap_first_last_best_avail_same_size_start = 1;
      #2;
      swap_first_last_best_avail_same_size_start = 0;

      while (swap_first_last_best_avail_same_size_fin == 0) #2;

      $display("HPWL total:", hpwl_total);

      $finish;
   end
```

Ύστερα από ορισμένα λεπτά, η άνω προσομοίωση δίνει τις παρακάτω τελικές γραμμές εξόδου για τον καλύτερο διαθέσιμο αλγόριθμο ίδιου μεγέθους:

```
⋮
swapped 1188 (10010100100) and  728 (01011011000)
swapped 1189 (10010100101) and  788 (01100010100)
HPWL total: 636480
```

οι οποίες είναι φυσικά όμοιες με αυτές που εκτυπώνονται από το αντίστοιχο μοντέλο λογισμικού:

```
⋮
Swapped a1188 with a728...
New HPWL: 636522 or 6.365e+05

Swapped a1189 with a788...
New HPWL: 636480 or 6.365e+05

Total Runtime (in seconds): 27.458499431610107
New HPWL: 636480 or 6.365e+0
```

# 8. Σύνθεση και Επαλήθευση σε FPGA

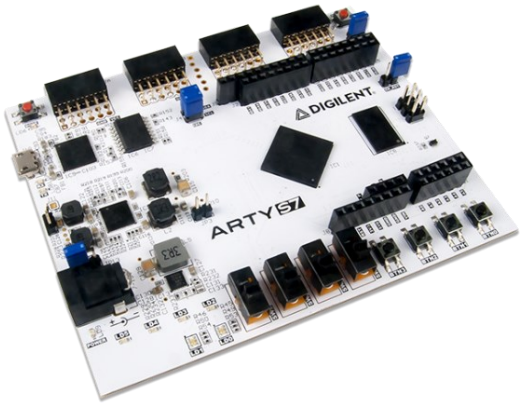
Ασφαλώς η προσομοίωση υλικού δεν είναι πάντα σωστή. Επομένως, ο σχεδιασμός υλικού έχει μεταφορτωθεί και σε FPGA προκειμένου να επαληθευτεί με ακρίβεια η λειτουργικότητά του. Το μοντέλο λογισμικού επικοινωνεί με το προγραμματισμένο FPGA μέσω του πρωτοκόλλου UART και υπολογίζει τον χρόνο εκτέλεσης του καθώς και δημιουργεί ένα αρχείο της βελτιστοποιημένης τοποθέτησης το οποίο μεταφέρει το FPGA μέσω UART.

Αυτό το κεφάλαιο καλύπτει ποιες FPGA χρησιμοποιήθηκαν κατά τη διαδικασία δοκιμής, πώς διαμορφώθηκε το αντίστοιχο πρότζεκτ στο κατάλληλο συμβατό λογισμικό σύνθεσης και προγραμματισμού, πόσοι πόροι του FPGA χρησιμοποιούνται, τι επηρεάζει τη χρήση τους και ποιες τοποθετήσεις μπορούν να βελτιστοποιηθούν ανάλογα τις παραμέτρους σύνθεσης και υλοποίησης, καθώς και πώς επαληθεύεται η λειτουργικότητα με εκτενή παραδείγματα εκτέλεσης και σύγκρισης.

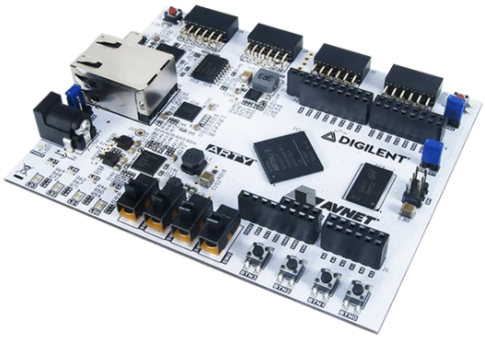
## 8.1 AMD Xilinx FPGAs

Στον κόσμο των συστοιχιών επιτόπια προγραμματιζόμενων πυλών (field programmable gate arrays - FPGA) υπάρχουν κατά τη σύνταξη της παρούσας εργασίας δύο κύριοι ανταγωνιστές, οι οποίοι είναι η Intel μετά την εξαγορά της Altera το 2015, καθώς και η AMD που έχει προχωρήσει και αυτή με τη σειρά της στην ολική εξαγορά της Xilinx που ολοκληρώθηκε το 2022.

Στα πλαίσια της παρούσας διπλωματικής εργασίας, χρησιμοποιήθηκαν FPGA της AMD Xilinx για τις δοκιμές, λόγω της ιδιοκτησίας διαφόρων πλακετών FPGA του εν λόγω κατασκευαστή.




*Εικόνα 8-1 Digilent Arty S7-50 FPGA*




*Εικόνα 8-2 Digilent Arty A7-35 FPGA*

Η σχεδίαση υλικού έχει συντεθεί και δοκιμαστεί εκτενώς σε δύο Digilent Arty FPGA, με το πρώτο να περιλαμβάνεται στην οικογένεια Spartan-7 με ακριβή ονομασία Arty S7-50 και το δεύτερο να περιλαμβάνεται στην οικογένεια Artix-7 με ακριβή ονομασία Arty A7-35. [11] [12].

Η Spartan FPGA διαθέτει λίγα περισσότερα λογικά κύτταρα και κυρίως περισσότερη BRAM επιτρέποντας τη δοκιμή λίγο μεγαλύτερων τοποθετήσεων, ενώ η Artix FPGA χρησιμοποιήθηκε κατά κύριο λόγο στην αρχή της σύνταξης της εργασίας λόγω του ότι διαθέτει λυχνίες LED τύπου RGB που χρησιμοποιήθηκαν αρχικά για τον έλεγχο της εξόδου των μονάδων.

## 8.2 Vivado πρότζεκτ

Καθώς ο σχεδιασμός βασίζεται σε AMD Xilinx FPGA, η σύνθεση και ο προγραμματισμός γίνονται μέσω του προγράμματος Vivado της AMD Xilinx [13].

Έχουν δημιουργηθεί επομένως δύο πρότζεκτ στο Vivado, με τα ονόματα *project_arty_a7_35* και *project_arty_s7_50* αντίστοιχα, τα οποία έχουν διαμορφωθεί κατάλληλα για χρήση με κάθε ένα από τα χρησιμοποιούμενα FPGA.

### 8.2.1 Ιεραρχία αρχείων

Στην ιεραρχία αρχείων περιλαμβάνονται και οι 19 παραλλαγές του σχεδιασμού ανώτατου επιπέδου που περιεγράφηκαν στο αντίστοιχο κεφάλαιο και μπορεί να πραγματοποιηθεί εύκολα εναλλαγή μεταξύ των σχεδιάσεων ορίζοντας νέο ανώτατο επίπεδο (μέσω “δεξί κλικ” > “set as top“) και τοποθετώντας πιθανόν αχρησιμοποίητες εισόδους στο αρχείο περιορισμών σε σχόλια (#).

Ενδεικτικά, παρακάτω παρατίθεται η ιεραρχία αρχείων για το FASSIB, δηλαδή την υλοποίηση του αλγορίθμου εναλλαγής κελιών ίδιου μεγέθους που επιτρέπει την επιλογή μεταξύ των παραλλαγών πρώτου (F), τελευταίου (I) και καλύτερου (B) κόμβου εναλλαγής.

Η παραλλαγή αυτή παρατίθεται με και τις δύο δυνατές διασυνδέσεις UART, οι οποίες είναι η διασύνδεση με όλες τις μνήμες (M) και η διασύνδεση μόνο με τις μνήμες $x$ και $y$ (XY). Το γράμμα P μετά το M υποδηλώνει πως έχουν αρχικοποιηθεί οι μνήμες με τις τιμές μιας τοποθέτησης και επιτρέπει την επιλογή μετάδοσης μόνο του $x$ και του $y$, μέσω κατάλληλης τιμής εισόδου. Το FASSIBM δεν θα παρείχε αυτήν την δυνατότητα, αλλά παρουσιάζει φυσικά παρόμοια ιεραρχία.

```
top_level_FASSIBMP.v
├── y_ram.v
├── x_ram.v
├── width_mt_ram.v
├── swap_first_last_best_available_same_size.v
├── swap_nodes_with_undo_ctrl.v
├── node_nets_ram.v
├── node_net_addr_ram.v
├── hpwl_net_ram.v
├── initial_hpwl_net_calc.v
├── recalc_affected_nets.v
├── net_pins_ram.v
├── net_pin_addr_ram.v
├── calc_hpwl_net.v
└── uart_memory_interface.v
    └── uart_interface.v
        └── uart_baud_rate_generator.v

top_level_FASSIBXY.v
├── y_ram.v
├── x_ram.v
├── width_mt_rom.v
├── swap_first_last_best_available_same_size.v
├── swap_nodes_with_undo_ctrl.v
├── node_nets_rom.v
├── node_net_addr_rom.v
├── hpwl_net_ram.v
├── initial_hpwl_net_calc.v
├── net_pins_rom.v
├── net_pin_addr_rom.v
├── calc_hpwl_net.v
└── uart_xy_ram_interface.v
```

### 8.2.2 Αρχείο περιορισμών

Για τη δημιουργία του απαραίτητου αρχείου περιορισμών για την αντιστοίχιση των εισόδων και εξόδων του FPGA διατίθεται ένα γενικό αρχείο περιορισμών σε αντίστοιχο αποθετήριο στο GitHub που προσφέρεται απευθείας από την Digilent [14]. Φυσικά, οι ακροδέκτες της συσκευασίας για κάθε στοιχείο εισόδου/εξόδου αντιστοιχούν σε διαφορετικές θέσεις για τα A7-35 και S7-50.

Στο πλαίσιο αυτής της σχεδίασης υλικού, οι είσοδοι και έξοδοι της σχεδίασης αντιστοιχίζονται στο σήμα ρολογιού, στα κουμπιά, στους διακόπτες, στις λυχνίες LED και στη διεπαφή USB-UART που διαθέτει η πλακέτα FPGA.

Προκειμένου να αντιμετωπιστεί ένα συγκεκριμένο προειδοποιητικό μήνυμα που εμφανίζεται σε αντίθετη περίπτωση, εκτός από το I/O, είναι επίσης απαραίτητο να οριστούν δύο πρόσθετοι περιορισμοί $set\_property$ που σχετίζονται με τον καθορισμό της τάσης.

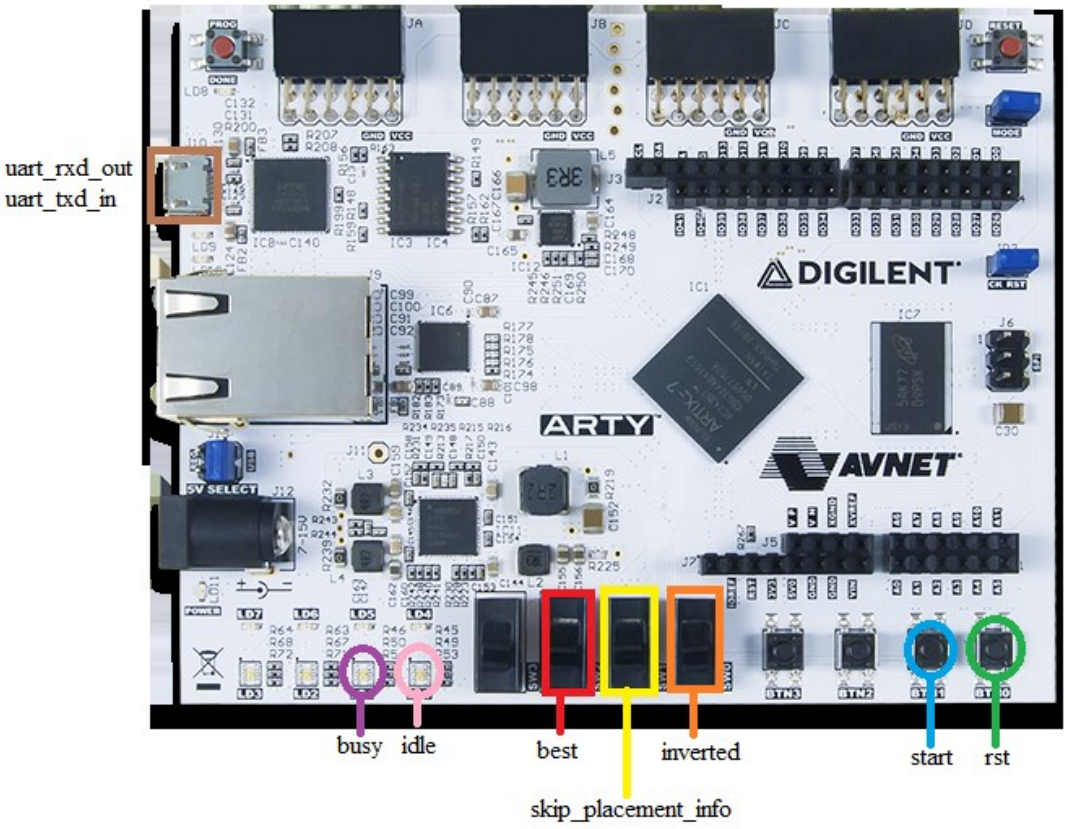
Η τελική μορφή του αρχείο περιορισμών του Arty S7-50 για το *FASSIBMP* είναι η εξής:

```
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## Clock signal
set_property -dict { PACKAGE_PIN R2 IOSTANDARD SSTL135 } [get_ports { clk }];
create_clock -add -name sys_clk_pin -period 10.000 -waveform {0 5.000}  [get_ports { clk }];

## Buttons
set_property -dict { PACKAGE_PIN G15 IOSTANDARD LVCMOS33 } [get_ports { rst }];
set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { start }];
#set_property -dict { PACKAGE_PIN J16 IOSTANDARD LVCMOS33 } [get_ports { btn[2] }];
#set_property -dict { PACKAGE_PIN H13 IOSTANDARD LVCMOS33 } [get_ports { btn[3] }];

## Switches
set_property -dict { PACKAGE_PIN H14 IOSTANDARD LVCMOS33 } [get_ports { inverted }];
set_property -dict { PACKAGE_PIN H18 IOSTANDARD LVCMOS33 } [get_ports { skip_placement_info }];
set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 } [get_ports { best }];
#set_property -dict { PACKAGE_PIN M5  IOSTANDARD SSTL135 } [get_ports { sw[3] }];

## LEDs
set_property -dict  { PACKAGE_PIN E18 IOSTANDARD LVCMOS33 } [get_ports { idle   }];
set_property -dict  { PACKAGE_PIN F13 IOSTANDARD LVCMOS33 } [get_ports { busy   }];
#set_property -dict { PACKAGE_PIN E13 IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
#set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 } [get_ports { led[3] }];

## USB-UART Interface
set_property -dict { PACKAGE_PIN R12 IOSTANDARD LVCMOS33 } [get_ports { uart_rxd_out }];
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { uart_txd_in }];
```

Για σχέδια ανώτατου επιπέδου που δεν επιτρέπουν την εναλλαγή μεταξύ των αλγορίθμων πρώτου και τελευταίου διαθέσιμου κελιού ιδίου μεγέθους, η γραμμή με το σήμα εισόδου $inverted$ που βρίσκεται μέσα στους διακόπτες (*switches*) πρέπει να τοποθετηθεί μέσα σε σχόλια. Ομοίως, εάν ο σχεδιασμός δεν αποθηκεύει τοποθέτηση και δεν επιτρέπει επομένως την παράλειψη της μεταφοράς των μη μεταβαλλόμενων στοιχείων, η γραμμή που περιέχει το αντίστοιχο σήμα $skip\_placement\_info$ θα πρέπει επίσης να τοποθετηθεί μέσα σε σχόλια. Τέλος, εάν δεν προσφέρεται η δυνατότητα επιλογής μεταξύ των αλγορίθμων πρώτου και καλύτερου διαθέσιμου κελιού ιδίου μεγέθους η γραμμή με το σήμα εισόδου $best$ θα πρέπει και αυτή με την σειρά της να τοποθετηθεί σε σχόλια.

### 8.2.2.1 Οπτική αναπαράσταση εισόδων και εξόδων σε πλακέτα FPGA

Παρακάτω μια εικόνα του Arty-A7 FPGA που χρησιμοποιήθηκε στη διπλωματική με επισυναπτόμενες πληροφορίες σχετικά με τις γενικές εισόδους και εξόδους του και πως αυτές αντιστοιχίζονται με τα σήματα εισόδου και εξόδου της υλοποίησης.




*Σχήμα 8-1 Αναπαράσταση χρήσης γενικών εισόδων της πλακέτας Arty A7-35*

### 8.2.3 Μέγιστη χρησιμοποίηση πόρων

Φυσικά, η χρησιμοποίηση και η εκμετάλλευση των διαθέσιμων πόρων της συσκευής FPGA εξαρτάται άμεσα από την τοποθέτηση που αυτή βελτιστοποιεί, και πιο συγκεκριμένα από τη μέγιστη δυνατή τοποθέτηση που μπορεί να αποθηκευτεί στις μνήμες της.

Η χρησιμοποίηση των πόρων της συσκευής, καθώς και η συμβατότητα της σχεδίασης με τις διάφορές τοποθετήσεις ορίζεται και επηρεάζεται άμεσα από τις παρακάτω γραμμές Verilog, οι οποίες βρίσκονται εντός του ορισμού της αντίστοιχης μονάδας υλικού:

```
⋮
parameter CONST_WIDTH = 13,
parameter PIN_ADDR_WIDTH = 15,
parameter HPWL_WIDTH = 32,
parameter HPWL_NET_WIDTH = 16,
parameter MAX_WIDTH_WIDTH = 7,
parameter MAX_LOWY_WIDTH = 16,
parameter MAX_LEFTX_WIDTH = 16,
⋮
```

Η παράμετρος $CONST\_WIDTH$ καθορίζει τον μέγιστο αριθμό bit για τα αναγνωριστικά κόμβων και καλωδίων. Ως εκ τούτου με τιμή 13 ο μέγιστος αριθμός κελιών ή καλωδίων που υποστηρίζει η συγκεκριμένη σύνθεση και υλοποίηση θα είναι $2^{13} = 8192$.

Είναι σημαντικό να σημειωθεί σε αυτό το σημείο ότι αυτή η τιμή σχετίζεται άμεσα με το μέγεθος των μνημών που χρησιμοποιούνται για την αποθήκευση των αντίστοιχων πληροφοριών για τους κόμβους και τα καλώδια. Ως εκ τούτου, εάν μια δεδομένη τοποθέτηση έχει έστω και λίγο περισσότερους από 8192 κόμβους ή καλώδια, η τιμή 13 δε θα αρκεί για την αποθήκευση της και έτσι δεν μπορεί να πραγματοποιηθεί και βελτιστοποίηση της από το προγραμματισμένο FPGA.

Η υψηλότερη χρήση πόρων μνήμης δεν προέρχεται ωστόσο από τον αριθμό των bits των αναγνωριστικών κόμβων ή καλωδίων, αλλά από τα bits που απαιτούνται για την αποθήκευση των διευθύνσεων των ακροδεκτών, δηλαδή την τιμή της παραμέτρου $PIN\_ADDR\_WIDTH$. Η τιμή 15 υποδηλώνει ότι μπορούν να αποθηκευτούν έως $2^{15} = 32768$ διαφορετικοί ακροδέκτες. Φυσικά, ο αριθμός των ακροδεκτών και των απαιτούμενων bits εξαρτάται πάντοτε από την τοποθέτηση που βελτιστοποιείται.

Αν θεωρηθεί ότι ο μέσος αριθμός ακροδεκτών ανά καλώδιο για μια δεδομένη τοποθέτηση είναι 3, τότε το $PIN\_ADDR\_WIDTH$ θα πρέπει να είναι τουλάχιστον ίσο με το $CONST\_WIDTH + 2$. Με τις παραπάνω αναφερόμενες παραμέτρους, διαπιστώνεται η παρακάτω χρησιμοποίηση πόρων συσκευής μετά την υλοποίηση (*post-implementation*) στο FASSIBMP για το Arty A7-35:

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 1146 | 20800 | 5.51 |
| FF | 1265 | 41600 | 3.04 |
| BRAM | 48 | 50 | 96.00 |
| IO | 9 | 210 | 4.29 |
| BUFG | 2 | 32 | 6.25 |

*Πίνακας 8-1 Χρησιμοποίηση πόρων Arty A7-35 με παραμέτρους $CONST\_WIDTH = 13$ και $PIN\_ADDR\_WIDTH = 15$*

Η αντίστοιχη χρησιμοποίηση πόρων για το Arty S7-50 είναι:

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 1151 | 32600 | 3.53 |
| FF | 1265 | 65200 | 1.94 |
| BRAM | 48 | 75 | 64.00 |
| IO | 9 | 210 | 4.29 |
| BUFG | 2 | 32 | 6.25 |

*Πίνακας 8-2 Χρησιμοποίηση πόρων Arty S7-50 με παραμέτρους $CONST\_WIDTH = 13$ και $PIN\_ADDR\_WIDTH = 15$*

Δυστυχώς, η παράμετρος $PIN\_ADDR\_WIDTH$ δεν μπορεί να αυξηθεί περαιτέρω, καθώς θα οδηγούσε σε υπερεκμετάλλευση των πόρων σε και τα δύο FPGA, ωστόσο είναι δυνατή η περαιτέρω αύξηση της παραμέτρου $CONST\_WIDTH$ κατά 1 για το Spartan-7 FPGA. Η αύξηση αυτή οδηγεί στην παρακάτω εκμετάλλευση πόρων:

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 1287 | 32600 | 3.95 |
| FF | 1288 | 65200 | 1.98 |
| BRAM | 71 | 75 | 94.67 |
| IO | 9 | 210 | 4.29 |
| BUFG | 2 | 32 | 6.25 |

*Πίνακας 8-3 Χρησιμοποίηση πόρων Arty S7-50 με παράμετρο $CONST\_WIDTH = 14$*

## 8.3 Επαλήθευση λειτουργικότητας

Για τους σκοπούς της επαλήθευσης της υλοποίησης μέσω FPGA, το μοντέλο λογισμικού παρέχει στο FPGA μια δεδομένη τοποθέτηση μέσω της επικοινωνίας UART και στη συνέχεια αναμένει την ολοκλήρωση της βελτιστοποίησης εκ μέρους του FPGA. Η επικοινωνία UART έχει επιτευχθεί και υλοποιηθεί με δύο τρόπους, οι οποίοι είναι η μεταφορά και λήψη μόνο των τιμών των συντεταγμένων $x$ και $y$ των κόμβων και η μεταφορά της πλήρους πληροφορίας σχετικά με μια τοποθέτηση.

Μετά τη λήψη των πληροφοριών μιας τοποθέτησης, το FPGA προχωρά στον υπολογισμό του αρχικού συνολικού HPWL των καλωδίων και στη συνέχεια εφαρμόζει τον αντίστοιχο αλγόριθμο βελτιστοποίησης εναλλαγής ωσότου να μην υπάρξει καμία βελτίωση κατά τη διάρκεια μιας πλήρους επανάληψης του αλγορίθμου. Στην προκειμένη περίπτωση, το FPGA εκκινεί τη μεταφορά της νέας βελτιστοποιημένης τοποθέτησης μέσω UART στο μοντέλο λογισμικού, το οποίο στη συνέχεια προβαίνει στην αποθήκευση της βελτιστοποιημένης τοποθέτησης σε ένα νέο αρχείο.

Προφανώς, ο ίδιος υπολογισμός εκτελείται και μέσω του μοντέλου λογισμικού και συνεπώς το τελικό βήμα της επαλήθευσης είναι ουσιαστικά ο έλεγχος του κατά πόσον οι δύο βελτιστοποιημένες τοποθετήσεις που προκύπτουν είναι πανομοιότυπες ή όχι. Η διαδικασία αυτή διεξήχθη και για τις 19 παραλλαγές του σχεδιασμού ανώτατου επιπέδου, εφαρμόζοντας βεβαίως το σύνολο των διαθέσιμων σημάτων ελέγχου, και το συμπέρασμα των δοκιμών ήταν ιδιαίτερα ευνοϊκό

### 8.3.1 Επικοινωνία λογισμικού και FPGA μέσω UART

Όπως έχει ήδη εξηγηθεί λεπτομερώς στο αντίστοιχο κεφάλαιο, διάφορες συναρτήσεις έχουν υλοποιηθεί στο λογισμικό για τη χρήση της επικοινωνίας UART.

Παρομοίως, η ανεβασμένη στο FPGA σχεδίαση υλικού απαρτίζεται επίσης από μια προηγμένη διασύνδεση UART-μνήμης.

Μετά την αποθήκευση της τοποθέτησης στην αντίστοιχη δομή δεδομένων, το μοντέλο λογισμικού ανοίγει τη σειριακή επικοινωνία με την αντίστοιχη θύρα σειριακής επικοινωνίας (COMX) με ρυθμό μετάδοσης (*baudrate*) 115200, ο οποίος είναι και ο μέγιστος δυνατός για τα FPGA που έχουν εξεταστεί και επιτρέπει φυσικά την ταχύτερη μεταφορά δεδομένων.

```
port = "COM6"
baudrate = 115200
serial: Serial = openSerialConnection(port, baudrate)
```

Το μοντέλο λογισμικού λαμβάνει πάντοτε τα ίδια δεδομένα από το FPGA, τα οποία απαρτίζονται από τις συντεταγμένες $x$ και $y$ των κόμβων, αλλά το τι μεταφέρεται στο FPGA εξαρτάται από τη σχεδίαση του υλικού και την παραμετροποίησή της με τη χρήση των διακοπτών ελέγχου.

#### 8.3.1.1 Μεταφορά και λήψη για μνήμες X και Y

Εάν η σχεδίαση ανώτερου επιπέδου έχει αποκλειστικά τη διασύνδεση μνήμης X και Y ή εάν το σήμα $skip\_placement\_info$ είναι ενεργό σε μια σχεδίαση ανώτερου επιπέδου η οποίο έχει αρχικοποιηθεί με τις πληροφορίες μιας δεδομένης τοποθέτησης (P), τότε το μοντέλο λογισμικού μεταφέρει και λαμβάνει μόνο τις συντεταγμένες $x$ και $y$ όπως φαίνεται στο απόσπασμα κώδικα Python που παρατίθεται στην αρχή της επόμενης σελίδας.

```
transferSerialNodesLowYLeftX(serial, placement.nodes, size=2)

print("Waiting for FPGA...")

receiveSerialNodesLowYLeftX(serial, placement.nodes, size=2)
```

#### 8.3.1.2 Μεταφορά και λήψη όλων των μνημών

Εάν η σχεδίαση διαθέτει πλήρη διασύνδεση μνημών-UART και δεν είναι ενεργοποιημένο φυσικά το σήμα παράλειψης πληροφοριών $skip\_placement\_info$, το μοντέλο λογισμικού θα μεταφέρει όλες τις πληροφορίες σχετικά με μια δεδομένη τοποθέτηση μέσω UART όπως φαίνεται στο παρακάτω απόσπασμα κώδικα Python.

```
transferSerialParameters(serial, placement, size=2)
transferSerialNodesWidthAndMoveType(serial, placement.nodes, size=1)
transferSerialNodeNets(serial, placement, size=2)
transferSerialNodeNetAddr(serial, placement.nodes, size=2)
transferSerialNetPins(serial, placement, size=2)
transferSerialNetPinAddr(serial, placement.nets, size=2)
transferSerialNodesLowYLeftX(serial, placement.nodes, size=2)

print("Waiting for FPGA...")

receiveSerialNodesLowYLeftX(serial, placement.nodes, size=2)
```

#### 8.3.1.3 Αποθήκευση νέας τοποθέτησης και χρόνος εκτέλεσης

Στη συνέχεια, το μοντέλο λογισμικού υπολογίζει το συνολικό HPWL, εκτυπώνει τη νέα τιμή, δημιουργεί και γράφει ένα νέο αρχείο με σκοπό την αποθήκευση της νέας βελτιστοποιημένης τοποθέτησης, και κλείνει την επικοινωνία UART.

```
placement.calcTotalHPWL()
print("New HPWL: " + str(placement.hpwl) +
    " or {:.3e}".format(placement.hpwl))

serializePLFile(directoryName + "/" + fileName + "_fpga", placement.nodes)

closeSerialConnection(serial)
```

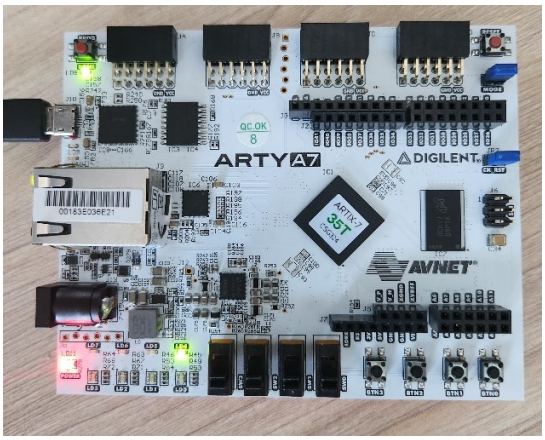
Τέλος, αξίζει να σημειωθεί πως το πρόγραμμα λογισμικού υπολογίζει και τον χρόνο εκτέλεσης που περιλαμβάνει τον χρόνο μεταφοράς UART και τον εκτυπώνει και αυτόν στην οθόνη του τερματικού.

```
timeBefore: float = time()
⋮
timeAfter: float = time()

print("Total Runtime (in seconds):", timeAfter - timeBefore)
```

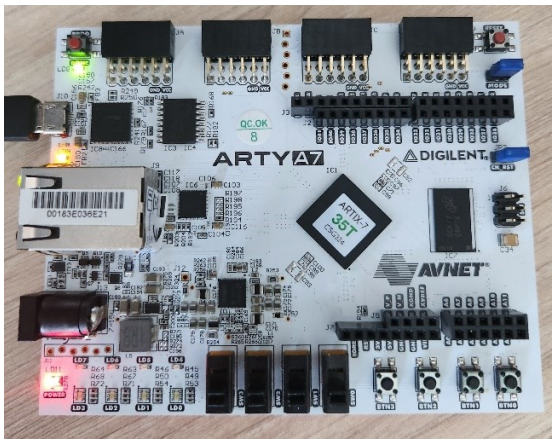
### 8.3.2 Οπτική απεικόνιση των σταδίων λειτουργίας σε πλακέτα FPGA

Τα στάδια λειτουργίας του FPGA είναι τα παρακάτω:

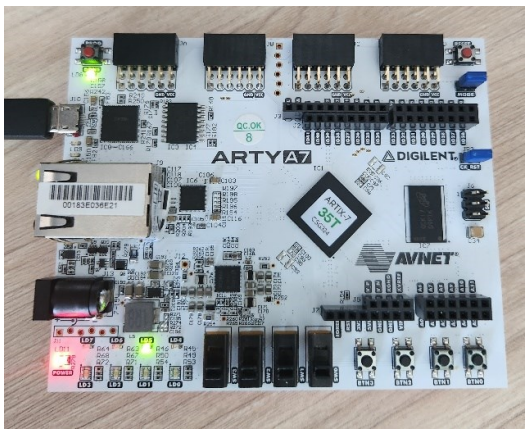
1. Αδρανής (*idle*).
2. Λήψη πληροφοριών μέσω UART.
3. "Απασχολημένο" (*busy*) με βελτιστοποίηση τοποθέτησης.
4. Μεταφορά της τοποθέτησης μέσω UART.



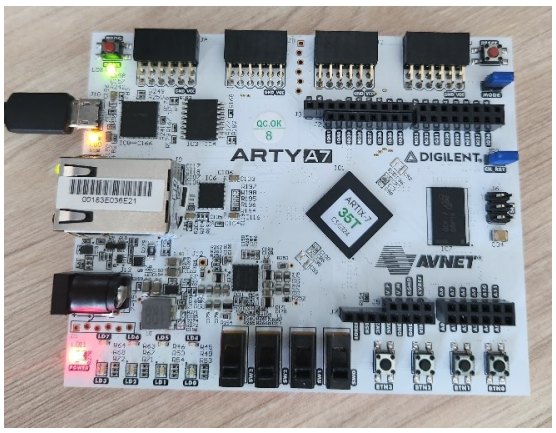

*Εικόνα 8-3 FPGA σε αδρανή λειτουργία*




*Εικόνα 8-4 Το FPGA λαμβάνει τις πληροφορίες της τοποθέτησης μέσω UART*




*Εικόνα 8-5 Το FPGA είναι «απασχολημένο» με τη βελτιστοποίηση της τοποθέτησης*




*Εικόνα 8-6 Το FPGA μεταφέρει την τοποθέτηση μέσω της διαπαφής UART*

Κάθε φορά που το FPGA μεταφέρει ή λαμβάνει δεδομένα μέσω UART ανάβουν δύο αντίστοιχες λυχνίες LED της πλακέτας, οι οποίες βρίσκονται δίπλα στη θύρα Micro-USB. Για τις καταστάσεις αδράνειας και βελτιστοποίησης ανάβουν δύο αντίστοιχες λυχνίες LED γενικής χρήσης, τις οποίες διαχειρίζεται κατάλληλα η υλοποίηση υλικού. Ο κύκλος και τα στάδια εκτέλεσης του FPGA, απεικονίζονται στις παρακάτω εικόνες, οι οποίες λήφθηκαν κατά την διάρκεια εκτέλεσης της μεταφοράς δεδομένων, βελτιστοποίησης και λήψης δεδομένων από την πλακέτα FPGA μέσω της αντίστοιχης συνεργασίας με το κομμάτι λογισμικού.

Φυσικά, η ίδια ακριβώς συμπεριφορά λυχνιών LED παρατηρείται και κατά την εκτέλεση των ιδίων σταδίων βελτιστοποίησης με την πλακέτα FPGA Arty S7.

Στις επόμενες ενότητες θα χρησιμοποιηθεί το ίδιο παράδειγμα τοποθέτησης όπως στο κεφάλαιο 5.5, προκειμένου να παρουσιαστεί η έξοδος του προγράμματος λογισμικού και του FPGA και να επαληθευτεί η λειτουργικότητά του.

### 8.3.3 Πρώτο διαθέσιμο ιδίου μεγέθους

Αφήνοντας το λογισμικό να προχωρήσει με περαιτέρω επαναλήψεις του αλγορίθμου μέχρι να μην υπάρχει επιπλέον βελτιστοποίηση, η έξοδος στο παράθυρο του τερματικού έχει ως εξής:

```
⋮
NumNodes :              1222
NumTerminals :     32
NumNets :   1403
NumPins :   9460
Initial HPWL: 929267 or 9.293e+05
Swapped a0 with a1...
New HPWL: 928784 or 9.288e+05
Swapped a1 with a18...
New HPWL: 928722 or 9.287e+05
⋮
Swapped a565 with a685...
New HPWL: 580181 or 5.802e+05
Total Runtime (in seconds): 228.42587447166443
New HPWL: 580181 or 5.802e+05
```

Αξίζει να σημειωθεί ότι αυτό είναι το αποτέλεσμα της εκτέλεσης σε ένα γρήγορο επεξεργαστή, τον AMD Ryzen 9 5950x.

Ο χρόνος εκτέλεσης σε φορητό υπολογιστή με Intel i7 9ης γενιάς είναι αρκετά υψηλότερος, με αποτέλεσμα η έξοδος να εμφανίζει τις ακόλουθες γραμμές:

```
⋮
Total Runtime (in seconds): 364.39416337013245
New HPWL: 580181 or 5.802e+05
```

Με μεταφορά όλων των πληροφοριών της τοποθέτησης και εκτέλεση της βελτιστοποίησης σε FPGA η έξοδος στο παράθυρο του τερματικού είναι η ακόλουθη:

```
Waiting for FPGA...
Total Runtime (in seconds): 53.996814250946045
New HPWL: 580181 or 5.802e+05
```

Με μεταφορά μόνο των συντεταγμένων $x$ και $y$ και παραλείποντας επομένως τη μεταφορά των υπόλοιπων πληροφοριών της τοποθέτησης, ο χρόνος εκτέλεσης βελτιώνεται με τον παρακάτω τρόπο:

```
Waiting for FPGA...
Total Runtime (in seconds): 50.15680456161499
New HPWL: 580181 or 5.802e+05
```

Έτσι, η μεταφορά δεδομένων μέσω UART απαιτεί μόνο ένα μικρό μέρος του συνολικού χρόνου που απαιτείται για την εκτέλεση της βελτιστοποίησης σε ένα FPGA.

Συγκρίνοντας τον χρόνο εκτέλεσης του FPGA με έναν αρκετά γρήγορο επεξεργαστή, η βελτίωση είναι λίγο μεγαλύτερη από 3 φορές (3,3 φορές επακριβώς), ενώ η σύγκριση με έναν λίγο πιο αργό επεξεργαστή οδηγεί σε σχεδόν 6-πλάσια (5.87 φορές επακριβώς) βελτίωση. Με μεγαλύτερες τοποθετήσεις, ο χρόνος εκτέλεσης θα ήταν στην ουσία απαγορευτικός αν χρησιμοποιούνταν για την εκτέλεση του κώδικα αργός επεξεργαστής.

### 8.3.4 Τελευταίο διαθέσιμο ιδίου μεγέθους

Η εκτέλεση αυτού του αλγορίθμου οδηγεί σε λίγο καλύτερη βελτιστοποίηση της τοποθέτησης με αύξηση του χρόνου εκτέλεσης. Η έξοδος στο παράθυρο του τερματικού είναι η ακόλουθη:

```
⋮

Swapped a0 with a1181...
New HPWL: 928962 or 9.290e+05

Swapped a1 with a1181...
New HPWL: 928527 or 9.285e+05

⋮

Swapped a797 with a1029...
New HPWL: 576860 or 5.769e+05

Total Runtime (in seconds): 303.9474606513977       (462.56678223609924)
New HPWL: 576860 or 5.769e+05
```

Η εκτέλεση σε FPGA με μεταφορά του συνόλου των πληροφοριών μέσω UART οδηγεί στο ακόλουθο αποτέλεσμα:

```
Waiting for FPGA...
Total Runtime (in seconds): 71.43525099754333
New HPWL: 576860 or 5.769e+05
```

Αντίστοιχα, αν μεταφερθούν και ληφθούν μόνο οι συντεταγμένες των κόμβων:

```
Waiting for FPGA...
Total Runtime (in seconds): 67.57989764213562
New HPWL: 576860 or 5.769e+05
```

Η βελτίωση είναι και πάλι λίγο μεγαλύτερη από 3-πλάσια και λίγο μικρότερη από 6-πλάσια, ανάλογα με τη CPU με την οποία πραγματοποιείται η σύγκριση.

Η βελτίωση της τιμής HPWL ανέρχεται σε περίπου 0,5% σε σχέση με τον αλγόριθμο εναλλαγής πρώτου διαθέσιμου κελιού ιδίου μεγέθους, ενώ ο χρόνος εκτέλεσης έχει αυξηθεί κατά 34%.

### 8.3.5 Καλύτερο διαθέσιμο ιδίου μεγέθους

Η εκτέλεση αυτού του αλγορίθμου οδηγεί σε χειρότερη βελτιστοποίηση της τοποθέτησης με χειρότερο χρόνο εκτέλεσης από τον αλγόριθμο εναλλαγής πρώτου διαθέσιμου, αλλά καλύτερο από τον αλγόριθμο εναλλαγής τελευταίου διαθέσιμου κελιού ιδίους μεγέθους.

Η έξοδος στο παράθυρο του τερματικού έχει ως εξής:

```
⋮

Swapped a0 with a204...
New HPWL: 928123 or 9.281e+05

⋮

Swapped a1029 with a378...
New HPWL: 584067 or 5.841e+05

Total Runtime (in seconds): 270.76687359809875        (413.2294566631317)
New HPWL: 584067 or 5.841e+05
```

Η εκτέλεση σε FPGA με μεταφορά του συνόλου των πληροφοριών μέσω UART οδηγεί στο ακόλουθο αποτέλεσμα:

```
Waiting for FPGA...
Total Runtime (in seconds): 64.20371747016907
New HPWL: 584067 or 5.841e+05
```

Αντίστοιχα, αν μεταφερθούν και ληφθούν μόνο οι συντεταγμένες των κόμβων ο χρόνος εκτέλεσης βελτιώνεται ως εξής:

```
Waiting for FPGA...
Total Runtime (in seconds): 60.36366653442383
New HPWL: 584067 or 5.841e+05
```

Η τιμή της μετρικής HPWL είναι κατά 1,2% χειρότερη από την καλύτερη τιμή, παρόλο που ο χρόνος εκτέλεσης είναι κατά 18% μεγαλύτερος από τον πρώτο αλγόριθμο και κατά 10% λιγότερος από τον αλγόριθμο εναλλαγής τελευταίο διαθέσιμου κελιού.

Στο πλαίσιο της βελτίωσης της τοποθέτησης μέσω εναλλαγής έχει διαπιστωθεί ότι οι τυχαίες εναλλαγές οδηγούν σε καλύτερη βελτίωση σε σχέση με την αναζήτηση του καλύτερου για εναλλαγή.

Φυσικά αυτό δεν σημαίνει απαραίτητα ότι ο αλγόριθμος είναι κακός, καθώς απαιτούνται λιγότερες επαναλήψεις του για την επίτευξη ισορροπίας.

### 8.3.6 Γείτονες ιδίου μεγέθους

Λόγω του μεγάλου αριθμού περιορισμών, ο χρόνος εκτέλεσης και η βελτίωση της τιμής HPWL δεν είναι τόσο μεγάλη.

Η έξοδος του μοντέλου λογισμικού διαμορφώνεται ως εξής:

```
⋮

Swapped a81 with a853...
New HPWL: 800397 or 8.004e+05

Total Runtime (in seconds): 2.5981087684631348       (3.606753587727783)
New HPWL: 800397 or 8.004e+0
```

Καθώς η μεταφορά της πλήρους τοποθέτησης μέσω UART θα απαιτούσε λίγο περισσότερα από 4 δευτερόλεπτα, όπως φάνηκε στις προηγούμενες εκτελέσεις, ο αλγόριθμος αυτός υλοποιήθηκε σε υλικό μόνο με τη χρήση της διασύνδεσης XY-UART.

Η έξοδος στην οθόνη του τερματικού με εκτέλεση σε FPGA έχει ως εξής:

```
Waiting for FPGA...
Total Runtime (in seconds): 1.3267061710357666
New HPWL: 800397 or 8.004e+05
```

Έτσι, το FPGA χρειάζεται περίπου το μισό χρόνο για να προχωρήσει στη βελτιστοποίηση της συγκεκριμένης τοποθέτησης με την χρήση αυτού το αλγορίθμου.

# 9. Συμπεράσματα

Η μεταφορά των αλγορίθμων βελτιστοποίησης τοποθέτησης σε υλικό επέφερε μεγάλη βελτίωση στο χρόνο εκτέλεσης των αντίστοιχων αλγορίθμων που έχουν δοκιμαστεί.

Φυσικά, με τον τρόπο που έχει ρυθμιστεί η σχεδίαση το μοντέλο υλικού δεν μπορεί να λειτουργήσει ανεξάρτητα και προϋποθέτει πάντοτε την ύπαρξη ενός αντίστοιχου μοντέλου λογισμικού το οποίο θα είναι ενεργό κατά τη διάρκεια της εκτέλεσής του. Αυτό βέβαια δεν αποτελεί πρόβλημα καθώς ο κύριος σκοπός της εργασίας ήταν η δημιουργία ενός επιταχυντή υλικού και όχι ενός επεξεργαστή.

Καθώς αυξάνεται ο αριθμός των κόμβων, των καλωδίων και των ακροδεκτών, αυξάνονται ωστόσο και οι απαιτήσεις σε μνήμη, γεγονός που σημαίνει ότι απαιτούνται πιο κοστοβόρες FPGA προκειμένου να χωρέσει η τοποθέτηση. Αυτό το συμπέρασμα ισχύει βέβαια μόνο εάν μεταφερθεί η πλήρης τοποθέτηση και δεν πραγματοποιηθεί άλλη τεχνική για την αντιμετώπιση αυτού του ζητήματος.

Ο συνολικός αριθμός των στοιχείων που πρέπει να αποθηκευτούν σε μνήμη για κάθε δεδομένη τοποθέτηση δίνεται από την ακόλουθη εξίσωση:

$$2\,x\,PIN\_COUNT\ +\ 4\,x\,NODE\_COUNT\ +\ 2\,x\,NET\_COUNT\ +\ 2$$

Αν θεωρηθεί ότι όλα τα δεδομένα αποθηκεύονται σε ενιαία μνήμη, τότε το πλάτος δεδομένων πρέπει να ισούται με το μέγιστο πλάτος δεδομένων, το οποίο στην πράξη είναι ίσο με το πλάτος διεύθυνσης της μνήμης διευθύνσεων ακροδεκτών ή καλωδίων που περιεγράφηκαν στο κεφάλαιο υλοποίησης υλικού.

Αν θεωρηθεί ο αριθμός των κόμβων και των καλωδίων ίσος και κάθε καλώδιο έχει κατά μέσο όρο 2 ακροδέκτες τότε η απαιτήσεις σε μνήμη αυξάνονται ως εξής:

| $node\_count = net\_count$ | $pin\_count$ | $max\_data\_width$ | **Memory (in kB)** |
|---|---|---|---|
| 2000 | 4000 | 12 | 240.024 |
| 4000 | 8000 | 13 | 520.026 |
| 6000 | 12000 | 14 | 840.028 |
| 8000 | 16000 | 14 | 1120.028 |
| 10000 | 20000 | 15 | 1500.03 |
| 12000 | 24000 | 15 | 1800.03 |
| 14000 | 28000 | 15 | 2100.03 |
| 16000 | 32000 | 15 | 2400.03 |
| 18000 | 36000 | 16 | 2880.032 |
| 20000 | 40000 | 16 | 3200.032 |

*Πίνακας 9-1 Απαιτήσεις μνήμης (σε kB) της υλοποίησης για τοποθετήσεις με ίσους κόμβους και καλώδια και κατά μέσο όρο 2 ακροδέκτες ανά καλώδιο*

Οι απαιτήσεις σε μνήμη αυξάνονται δηλαδή σχεδόν γραμμικά, με κάθε επιπλέον $2000$ κόμβους, $2000$ καλώδια και $4000$ ακροδέκτες να αντιστοιχούν σε περίπου $300kB$ μνήμης.

Συγκριτικά το Arty A7-25 διαθέτει $225\ kB$ μνήμης Block RAM, ενώ το Arty S7-50 $337{,}5\ kB$ [12] [11].

Με κατά μέσο όρο 4 ακροδέκτες ανά καλώδιο προκύπτει αύξηση κατά περίπου $500\ kB$, ενώ σε μια πιο πυκνή τοποθέτηση με $10$ ακροδέκτες ανά καλώδιο απαιτείται σχεδόν $1MB$ για κάθε επιπλέον προσθήκη $2000$ κόμβων και καλωδίων στην τοποθέτηση.

Όποτε με το κλείσιμο της διατριβής συμπεραίνεται πώς πρέπει και μπορεί να διεξαχθεί ακόμα πολύ έρευνα στο κομμάτι μεταφοράς αλγορίθμων που σχετίζονται με οτιδήποτε αφορά τη σχεδίαση υλικού με σκοπό την χρήση υλικού ως επιταχυντή των διαδικασιών, όπως η βελτιστοποίηση τοποθέτησης που έχει πρωτοτυποποιηθεί σε υλικό στην παρούσα διατριβή.

Λόγω της φύσης των αλγορίθμων λεπτομερής τοποθέτησης, μπορούν να υλοποιηθούν φυσικά αρκετές ακόμα βελτιστοποιήσεις, οι οποίες δεν κατέστη δυνατόν να ολοκληρωθούν στο πλαίσιο της μεταπτυχιακής διατριβής.

Ενδεικτικά, παρακάτω παρατίθεται βελτιστοποιήσεις και τροποποιήσεις που θα μπορούσαν να εφαρμοστούν.

Η συνεργασία του μοντέλου λογισμικού και του μοντέλου υλικού θα μπορούσε να είναι διαφορετική από τη μεταφορά της πλήρους τοποθέτησης και της λήψης της πλήρους τοποθέτησης το οποίο περιορίζει την επικοινωνία σε πριν και μετά τη βελτιστοποίηση. Το FPGA και το αντίστοιχο πρόγραμμα λογισμικού θα μπορούσαν να μεταφέρουν τα δεδομένα τοποθέτησης σύμφωνα με τις αντίστοιχες απαιτήσεις δεδομένων που απαιτούνται κάθε εκάστοτε στιγμή της εκτέλεσης της διαδικασίας βελτιστοποίησης. Η FPGA θα διέθετε επομένως μνήμη η οποία έχει τη μορφή κρυφής μνήμης επεξεργαστή.

Για μια τέτοια υλοποίηση, η εφαρμογή θεωρίας σχεδιασμού λειτουργικών συστημάτων ενδέχεται να αποτελεί πολύ χρήσιμη λύση. Θα χρειαστεί λοιπόν κάποιος διαχωρισμός της μνήμης σε φυσικές και εικονικές διευθύνσεις μνήμης.

Θα μπορούσε επίσης να είναι ενδιαφέρουσα μια υλοποίηση όπου η FPGA δεν έχει καθόλου μνήμη ή έχει πολύ λίγη μνήμη, δηλαδή μόνο την ελάχιστη απαιτούμενη μνήμη, πράγμα που σηματοδοτεί ότι η FPGA βασίζεται στο μοντέλο λογισμικού για τα δεδομένα τοποθέτησης.

Η παραλληλοποίηση διαφόρων σταδίων του υπολογισμού καθώς και η εκτέλεση της βελτιστοποίησης σε πολλαπλές πλακέτες FPGA θα μπορούσαν επίσης να οδηγήσουν σε εξαιρετικά ενδιαφέροντα αποτελέσματα.

# 10.Πηγές και Βιβλιογραφία


[1] J. L. I. L. M. J. H. Andrew B. Kahng, VLSI Physical Design: From Graph Partitioning to Timing Closure, New York: Springer, 2011.

[2] NIST, "Manhattan distance," 11 February 2019. [Online]. Available: https://xlinux.nist.gov/dads/HTML/manhattanDistance.html.

[3] P. S. A. R. T. M. D. P. M. B. N. B. Ray, "Half-Perimeter Wirelength Model for VLSI Analytical Placement," *International Conference on Information Technology,* p. 288, 2014.

[4] I. M. Andrew Caldwell Andrew B. Kahng, "Placement Formats, rev. 1.2," 20 November 1999. [Online]. Available: https://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement/plFormats.html.

[5] I. M. Saurabh Adya, "ISPD02 IBM-MS Mixed-size Placement Benchmarks," - - 2004. [Online]. Available: http://vlsicad.eecs.umich.edu/BK/ISPD02bench/.

[6] Python Software Foundation, "python," - - 2023. [Online]. Available: https://www.python.org/.

[7] C. Liechti, "pySerial Documentation," - - 2020. [Online]. Available: https://pyserial.readthedocs.io/en/latest/.

[8] NAND LAND, "UART, Serial Port, RS-232 Interface," - - 2022. [Online]. Available: https://nandland.com/uart-serial-port-module/.

[9] ELECTRONOOBS, "UART COMMUNICATION ON FPGA," - - 2023. [Online]. Available: https://electronoobs.com/eng_circuitos_tut26.php.

[10] S. Williams, "GitHub," - - 2019. [Online]. Available: https://github.com/steveicarus/iverilog.

[11] Digilent, "Arty S7 Reference Manual," - - 2023. [Online]. Available: https://digilent.com/reference/programmable-logic/arty-s7/reference-manual.

[12] Digilent, "Arty A7 Reference Manual," - - 2023. [Online]. Available: https://digilent.com/reference/programmable-logic/arty-a7/reference-manual.

[13] AMD Xilinx, "AMD Viado," - - 2023. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html.

[14] Digilent, "digilent-xdc," - - 2022. [Online]. Available: https://github.com/Digilent/digilent-xdc.

# 11. Παραρτήματα

## 11.1 Παράδειγμα απλής τοποθέτησης σε μορφοποίηση Bookshelf

Ακολουθούν τα αρχεία της μορφοποίησης Bookshelf μιας απλής τοποθέτησης με 20 κελιά, από τα οποία 4 είναι τερματικά, 20 καλώδια και 49 ακροδέκτες.

### 11.1.1 Αρχείο κόμβων απλής τοποθέτησης

```
UCLA nodes 1.0
# Created
# User
# Platform

NumNodes : 20
NumTerminals :  4
      a0        8       16
      a1        8       16
      a2        8       16
      a3       12       16
      a4        8       16
      a5        8       16
      a6       12       16
      a7        8       16
      a8        8       16
      a9       12       16
     a10        8       16
     a11        8       16
     a12        8       16
     a13        8       16
     a14       12       16
     a15       12       16
      p1        1       1 terminal
      p2        1       1 terminal
      p3        1       1 terminal
      p4        1       1 terminal
```

### 11.1.2 Αρχείο καλωδίων απλής τοποθέτησης

```
UCLA nets 1.0
# Created
# User
# Platform

NumNets : 20
NumPins : 49
NetDegree : 2
   p1 B
   a5 B
```

NetDegree : 2
p2 B
a9 B
NetDegree : 2
p3 B
a3 B
NetDegree : 2
p4 B
a15 B
NetDegree : 3
a0 O
a7 I
a11 I
NetDegree : 2
a1 O
a3 I
NetDegree : 4
a2 O
a5 I
a9 I
a10 I
NetDegree : 2
a3 O
a13 I
NetDegree : 2
a4 O
a8 I
NetDegree : 5
a5 O
a3 I
a9 I
a6 I
a14 I
NetDegree : 2
a6 O
a15 I
NetDegree : 2
a7 O
a10 I
NetDegree : 2
a8 O
a1 I
NetDegree : 3
a9 O
a4 I
a5 I
NetDegree : 2
a10 O
a2 I

```
NetDegree : 3
        a11 O
        a6 I
        a14 I
NetDegree : 2
        a12 O
        a1 I
NetDegree : 2
        a13 O
        a9 I
NetDegree : 2
        a14 O
        a2 I
NetDegree : 3
        a15 O
        a5 I
        a8 I
```

### 11.1.3 Αρχείο συντελεστών βάρους απλής τοποθέτησης

```
UCLA wts 1.0
# Created
# User
# Platform

    p1      0
    p2      0
    p3      0
    p4      0
    a0    256
    a1    224
    a2    128
    a3     96
    a4     96
    a5    128
    a6    128
    a7    128
    a8    128
    a9    192
   a10    672
   a11    128
   a12    192
   a13    128
   a14     96
   a15    224
```

### 11.1.4 Αρχείο τοποθέτησης κελιών απλής τοποθέτησης

```
UCLA pl 1.0
# Created
# User
# Platform

a0 20 112 : S
a1 71 48 : S
a2 33 64 : S
a3 14 32 : S
a4 45 96 : N
a5 56 112 : N
a6 68 64 : S
a7 92 32 : S
a8 12 16 : S
a9 44 16 : S
a10 74 16 : N
a11 46 32 : N
a12 27 80 : N
a13 90 96 : N
a14 26 48 : S
a15 70 80 : S
p1 64 129 : FS
p2 64 -33 : N
p3 -33 64 : E
p4 129 64 : FW
```

## 11.2 Δηλώσεις μονάδων υλικού σε γλώσσα περιγραφής υλικού Verilog

Ακολουθούν οι δηλώσεις των μονάδων της σχεδίασης σε γλώσσα περιγραφής υλικού Verilog.

### 11.2.1 Μονάδα υπολογισμού HPWL καλωδίου

```
module calc_hpwl_net
   #(
      parameter CONST_WIDTH = 14,
      parameter PIN_ADDR_WIDTH = 16,
      parameter HPWL_NET_WIDTH = 12,
      parameter MAX_LOWY_WIDTH = 12,
      parameter MAX_LEFTX_WIDTH = 12
   )
   (
      output reg [HPWL_NET_WIDTH - 1 : 0] hpwl_net,
      output reg [CONST_WIDTH - 1 : 0] node_id_reg,
      output reg xy_ram_oe,
      output reg recalc_fin,
      output reg [PIN_ADDR_WIDTH - 1 : 0] net_pin_addr_reg,
```

```
    output reg net_pins_ram_oe,
    output reg [CONST_WIDTH - 1 : 0] node_nets_id,
    output reg net_pin_addr_ram_oe,
    input [CONST_WIDTH - 1 : 0] net_id,
    input signed [MAX_LOWY_WIDTH - 1 : 0] low_y,
    input signed [MAX_LEFTX_WIDTH - 1 : 0] left_x,
    input [CONST_WIDTH - 1 : 0] pin_id,
    input [PIN_ADDR_WIDTH - 1 : 0] net_pin_addr,
    input recalc_start,
    input clk, rst
);
```

### 11.2.2 Μονάδα αρχικού υπολογισμού συνολικού HPWL

```
module initial_hpwl_net_calc
  #(
    parameter CONST_WIDTH = 14,
    parameter HPWL_WIDTH = 21,
    parameter HPWL_NET_WIDTH = 12
  )
  (
    output reg fin,
    output reg recalc_start,
    output reg [CONST_WIDTH - 1 : 0] net_id_out,
    output reg [HPWL_NET_WIDTH - 1 : 0] hpwl_net_reg,
    output reg hpwl_ram_we,
    output reg [HPWL_WIDTH - 1 : 0] hpwl_total_out,
    output reg hpwl_total_init,
    input [HPWL_NET_WIDTH - 1 : 0] recalc_hpwl_net_in,
    input recalc_fin,
    input [CONST_WIDTH - 1 : 0] net_count_in,
    input start,
    input clk, rst
  );
```

### 11.2.3 Μονάδα επαναϋπολογισμού μόνο των επηρεαζόμενων καλωδίων

```
module recalc_affected_nets
  #(
    parameter CONST_WIDTH = 14,
    parameter PIN_ADDR_WIDTH = 16,
    parameter HPWL_WIDTH = 21,
    parameter HPWL_NET_WIDTH = 12
  )
  (
    output reg [PIN_ADDR_WIDTH - 1 : 0] node_net_addr_reg,
    output reg node_nets_ram_oe,
```

```
    output reg [CONST_WIDTH - 1 : 0] net_nodes_id,
    output reg node_net_addr_ram_oe,
    output reg [CONST_WIDTH - 1 : 0] net_id_out,
    output reg [HPWL_NET_WIDTH - 1 : 0] hpwl_net_reg,
    output reg signed [HPWL_WIDTH - 1 : 0] hpwl_change_out,
    output reg hpwl_ram_we,
    output reg hpwl_ram_oe,
    output reg recalc_start,
    output reg fin,
    input [CONST_WIDTH - 1 : 0] node_id,
    input [HPWL_NET_WIDTH - 1 : 0] hpwl_net_in,
    input [HPWL_NET_WIDTH - 1 : 0] recalc_hpwl_net_in,
    input [PIN_ADDR_WIDTH - 1 : 0] node_net_addr,
    input [CONST_WIDTH - 1 : 0] net_id,
    input start, recalc_fin,
    input clk, rst
);
```

### 11.2.4 Μονάδα εναλλαγής κόμβων με αναίρεση

```
module swap_nodes_with_undo
  #(
    parameter CONST_WIDTH = 14,
    parameter HPWL_WIDTH = 21,
    parameter MAX_LOWY_WIDTH = 12,
    parameter MAX_LEFTX_WIDTH = 12
  )
  (
    output reg [CONST_WIDTH - 1 : 0] node_id_reg,
    output reg signed [MAX_LOWY_WIDTH - 1 : 0] low_y_reg,
    output reg signed [MAX_LEFTX_WIDTH - 1 : 0] left_x_reg,
    output reg ram_we,
    output reg ram_oe,
    output reg recalc_aff_start,
    output reg swap_fin,
    output reg signed [HPWL_WIDTH - 1 : 0] hpwl_total_change_out,
    output reg hpwl_total_update,
    input [CONST_WIDTH - 1 : 0] node1_id,
    input [CONST_WIDTH - 1 : 0] node2_id,
    input signed [MAX_LOWY_WIDTH - 1 : 0] low_y,
    input signed [MAX_LEFTX_WIDTH - 1 : 0] left_x,
    input signed [HPWL_WIDTH - 1 : 0] hpwl_change,
    input start,
    input recalc_aff_fin,
    input clk, rst
  );
```

### 11.2.5 Μονάδα εναλλαγής κόμβων με αναίρεση και έλεγχο αναίρεσης

```
module swap_nodes_with_undo_ctrl
    #(
        parameter CONST_WIDTH = 14,
        parameter HPWL_WIDTH = 21,
        parameter MAX_LOWY_WIDTH = 12,
        parameter MAX_LEFTX_WIDTH = 12
    )
    (
        output reg [CONST_WIDTH - 1 : 0] node_id_reg,
        output reg signed [MAX_LOWY_WIDTH - 1 : 0] low_y_reg,
        output reg signed [MAX_LEFTX_WIDTH - 1 : 0] left_x_reg,
        output reg ram_we,
        output reg ram_oe,
        output reg recalc_aff_start,
        output reg swap_fin,
        output reg signed [HPWL_WIDTH - 1 : 0] hpwl_total_change_out,
        output reg hpwl_total_update,
        output reg swap_undo_start,
        input [CONST_WIDTH - 1 : 0] node1_id,
        input [CONST_WIDTH - 1 : 0] node2_id,
        input signed [MAX_LOWY_WIDTH - 1 : 0] low_y,
        input signed [MAX_LEFTX_WIDTH - 1 : 0] left_x,
        input signed [HPWL_WIDTH - 1 : 0] hpwl_change,
        input swap_undo_ctrl,
        input start,
        input recalc_aff_fin,
        input clk, rst
    );
```

### 11.2.6 Μονάδα εναλλαγής με πρώτο διαθέσιμο κελί ιδίου μεγέθους

```
module swap_first_available_same_size
    #(
        parameter CONST_WIDTH = 14,
        parameter MAX_WIDTH_WIDTH = 6
    )
    (
        output reg [CONST_WIDTH - 1 : 0] node1_id_out,
        output reg [CONST_WIDTH - 1 : 0] node2_id_out,
        output reg swap_start,
        output reg fin,
        output reg [CONST_WIDTH - 1 : 0] node_id,
        output reg ram_oe,
        input start,
        input swap_fin,
```

```
        input hpwl_total_update,
        input [MAX_WIDTH_WIDTH - 1 : 0] node_width,
        input node_mt,
        input [CONST_WIDTH - 1 : 0] node_count,
        input clk, rst
    );
```

### 11.2.7 Μονάδα εναλλαγής με καλύτερο διαθέσιμο κελί ιδίου μεγέθους

```
module swap_best_available_same_size
    #(
        parameter CONST_WIDTH = 14,
        parameter HPWL_WIDTH = 20,
        parameter MAX_WIDTH_WIDTH = 6
    )
    (
        output reg [CONST_WIDTH - 1 : 0] node1_id_out,
        output reg [CONST_WIDTH - 1 : 0] node2_id_out,
        output reg swap_start,
        output reg swap_undo_ctrl,
        output reg fin,
        output reg [CONST_WIDTH - 1 : 0] node_id,
        output reg ram_oe,
        input start,
        input swap_fin,
        input hpwl_total_update,
        input signed [HPWL_WIDTH - 1 : 0] hpwl_total_change_out,
        input swap_undo_start,
        input [MAX_WIDTH_WIDTH - 1 : 0] node_width,
        input node_mt,
        input [CONST_WIDTH - 1 : 0] node_count,
        input clk, rst
    );
```

### 11.2.8 Μονάδα εναλλαγής μεταξύ γειτόνων ιδίου μεγέθους

```
module swap_same_size_neighbours
    #(
        parameter CONST_WIDTH = 11,
        parameter MAX_WIDTH_WIDTH = 5,
        parameter ADJ_LIST_ADDR_WIDTH = 14
    )
    (
        output reg [CONST_WIDTH - 1 : 0] node1_id_out,
        output reg [CONST_WIDTH - 1 : 0] node2_id_out,
        output reg swap_start,
        output reg fin,
```

```
        output reg [CONST_WIDTH - 1 : 0] node_id,
        output reg width_mt_ram_oe,
        output reg [CONST_WIDTH - 1 : 0] adj_list_node_id_reg,
        output reg nodes_adj_list_addr_oe,
        output reg [ADJ_LIST_ADDR_WIDTH - 1 : 0] nodes_adj_list_addr_reg,
        output reg nodes_adj_list_oe,
        input start,
        input swap_fin,
        input hpwl_total_update,
        input [MAX_WIDTH_WIDTH - 1 : 0] node_width,
        input node_mt,
        input [ADJ_LIST_ADDR_WIDTH - 1 : 0] nodes_adj_list_addr,
        input [CONST_WIDTH - 1 : 0] adj_list_node_id,
        input [CONST_WIDTH - 1 : 0] node_count,
        input clk, rst
    );
```

### 11.2.9 Μονάδα διεπαφής UART

```
module uart_interface
    #(
        parameter SYSTEM_CLK_MHZ = 100,
        parameter BAUD_RATE = 115200,
        parameter SAMPLING_RATE = 16,
        parameter COUNTER_WIDTH = 8,
        parameter MAX_DATA_WIDTH = 16 // 1 or 2 bytes
    )
    (
        output reg uart_rxd_out,
        input wire uart_txd_in,

        output reg [MAX_DATA_WIDTH - 1 : 0] data_out,
        input wire [MAX_DATA_WIDTH - 1 : 0] data_in,

        output reg done,
        input wire rtxd,    // 0 : RX,   1 : TX
        input wire bytes,   // 0 : 1 byte, 1 : 2 bytes
        input wire start,

        input wire clk, rst
    );
```

### 11.2.10 Μονάδα διασύνδεσης μνημών x, y και UART

```
module uart_xy_ram_interface
   #(
      parameter NODE_COUNT = 1222,
      parameter SYSTEM_CLK_MHZ = 100,
      parameter BAUD_RATE = 115200,
      parameter SAMPLING_RATE = 16,
      parameter COUNTER_WIDTH = 8,
      parameter MAX_DATA_WIDTH = 16, // 1 or 2 bytes
      parameter CONST_WIDTH = 14,
      parameter MAX_LOWY_WIDTH = 11,
      parameter MAX_LEFTX_WIDTH = 11
   )
   (
      output reg fin,

      output reg signed [MAX_LOWY_WIDTH - 1 : 0] low_y_out,
      output reg signed [MAX_LEFTX_WIDTH - 1 : 0] left_x_out,
      output reg [CONST_WIDTH - 1 : 0] node_id_out,
      output reg xy_ram_oe,
      output reg xy_ram_we,

      output wire uart_rxd_out,
      input uart_txd_in,

      input signed [MAX_LOWY_WIDTH - 1 : 0] low_y_in,
      input signed [MAX_LEFTX_WIDTH - 1 : 0] left_x_in,

      input mode,
      input start,
      input clk, rst
   );
```

### 11.2.11 Μονάδα διασύνδεσης όλων των μνημών και UART

```
module uart_memory_interface
   #(
      parameter SYSTEM_CLK_MHZ = 100,
      parameter BAUD_RATE = 115200,
      parameter SAMPLING_RATE = 16,
      parameter COUNTER_WIDTH = 8,
      parameter MAX_DATA_WIDTH = 16, // 1 or 2 bytes
      parameter CONST_WIDTH = 14,
      parameter PIN_ADDR_WIDTH = 14,
      parameter MAX_WIDTH_WIDTH = 5,
      parameter MAX_LOWY_WIDTH = 11,
```

```
    parameter MAX_LEFTX_WIDTH = 11
)
(
    output reg fin,

    output reg [CONST_WIDTH - 1 : 0] node_id_out,
    output reg [CONST_WIDTH - 1 : 0] net_id_out,
    output reg [PIN_ADDR_WIDTH - 1 : 0] pin_addr_id_out,

    output reg [CONST_WIDTH - 1 : 0] node_count_out,
    output reg [CONST_WIDTH - 1 : 0] net_count_out,
    output reg [PIN_ADDR_WIDTH - 1 : 0] pin_count_out,
    output reg parameter_reg_we,

    output reg [MAX_WIDTH_WIDTH : 0] node_width_movetype_out,
    output reg node_width_movetype_ram_we,

    output reg [CONST_WIDTH - 1 : 0] node_nets_id_out,
    output reg node_nets_ram_we,

    output reg [PIN_ADDR_WIDTH - 1 : 0] node_net_addr_out,
    output reg node_net_addr_ram_we,

    output reg [CONST_WIDTH - 1 : 0] net_pins_id_out,
    output reg net_pins_ram_we,

    output reg [PIN_ADDR_WIDTH - 1 : 0] net_pin_addr_out,
    output reg net_pin_addr_ram_we,

    output reg signed [MAX_LOWY_WIDTH - 1 : 0] low_y_out,
    output reg signed [MAX_LEFTX_WIDTH - 1 : 0] left_x_out,
    output reg xy_ram_oe,
    output reg xy_ram_we,

    output wire uart_rxd_out,
    input uart_txd_in,

    input [CONST_WIDTH - 1 : 0] node_count_in,
    input [CONST_WIDTH - 1 : 0] net_count_in,
    input [PIN_ADDR_WIDTH - 1 : 0] pin_count_in,

    input signed [MAX_LOWY_WIDTH - 1 : 0] low_y_in,
    input signed [MAX_LEFTX_WIDTH - 1 : 0] left_x_in,

    input [2 : 0] mode,

    input start,
    input clk, rst
);
```

### 11.2.12 Μονάδα ανώτατου επιπέδου της παραλλαγής FASSIBMP

```
module top_level_FASSIBMP
   #(
      parameter PLACEMENT_NODE_COUNT = 1222,
      parameter PLACEMENT_NET_COUNT = 1403,
      parameter PLACEMENT_PIN_COUNT = 9460,
      parameter CONST_WIDTH = 13,
      parameter PIN_ADDR_WIDTH = 15,
      parameter HPWL_WIDTH = 32,
      parameter HPWL_NET_WIDTH = 16,
      parameter MAX_WIDTH_WIDTH = 7,
      parameter MAX_LOWY_WIDTH = 16,
      parameter MAX_LEFTX_WIDTH = 16,
      parameter SYSTEM_CLK_MHZ = 100,
      parameter BAUD_RATE = 115200,
      parameter SAMPLING_RATE = 16,
      parameter COUNTER_WIDTH = 8,
      parameter MAX_DATA_WIDTH = 16
   )
   (
      output reg idle,
      output reg busy,
      output wire uart_rxd_out,
      input uart_txd_in,
      input best,
      input inverted,
      input skip_placement_info,
      input start,
      input clk, rst
   );
```