ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Τροποποίηση και Επεκτάσεις Πρακτικών Νομιμοποίησης κατά τη Φυσική Σχεδίαση Ολοκληρωμένων Κυκλωμάτων

Κυριαζής Ιωάννης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Δρ. Αντώνιος Δαδαλιάρης
Επίκουρος Καθηγητής Πανεπιστημίου Θεσσαλίας

Λαμία 24/02 έτος 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Τροποποίηση και Επεκτάσεις Πρακτικών Νομιμοποίησης κατά τη Φυσική Σχεδίαση Ολοκληρωμένων Κυκλωμάτων

Κυριαζής Ιωάννης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Δρ. Αντώνιος Δαδαλιάρης
Επίκουρος Καθηγητής Πανεπιστημίου Θεσσαλίας

Λαμία 24/02 έτος 2025

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Modification and Extensions of Legalization Practices in IC Physical Design

Kyriazis Ioannis

FINAL THESIS

ADVISOR

Dr. Antonios Dadaliaris
Assistant Professor at the University of Thessaly

Lamia 24/02 year 2025

«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: 24/02/2025

Ο Δηλών

Κυριαζής Ιωάννης

(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.»

ΠΕΡΙΛΗΨΗ

Ένα ASIC(Application Specific Integrated Circuit) πρόκειται για ένα ολοκληρωμένο ενσωματωμένο κύκλωμα που έχει σχεδιαστεί εξ αρχής για την επιτέλεση μιας πολύ συγκεκριμένης λειτουργίας. Η σχεδιαστική του πορεία αποτελείται από συνολικά δεκατρία βήματα, τα οποία αναλύονται στο πρώτο κεφάλαιο. Η παρούσα πτυχιακή μελετάει αρχικά τον αλγόριθμο FBM, έναν αλγόριθμο που σχεδιάστηκε για το στάδιο του Legalization, το οποίο είναι και μέρος της ροής σχεδίασης ενός ASIC. Πρώτα γίνεται μια ομαλή εισαγωγή και περιγραφή του βήματος του Legalization και έπειτα αναλύεται διεξοδικά και εις βάθος ο αλγόριθμος και η λειτουργικότητά του. Έχοντας τελειώσει με αυτό τον τρόπο με το δεύτερο κεφάλαιο, στη συνέχεια αναλύθηκαν ορισμένες τροποποιήσεις και προσθήκες που έγιναν πάνω στον αλγόριθμο με στόχο το να γίνει πιο αποδοτικός αλλά και να είναι σε θέση να εκτελεί επιπλέον βελτιστοποιήσεις πάνω στο ήδη υπάρχων κύκλωμα. Τέλος, οι υλοποιήσεις που περιγράφηκαν, αφού έγιναν πράξη, πραγματοποιήθηκε προσεκτική καταγραφή των αποτελεσμάτων με σκοπό την εξαγωγή συμπερασμάτων σχετικά με το πόσο αποτελεσματικές ήταν τελικά όλες οι αλλαγές που έγιναν, καθώς και ποιες ήταν οι βέλτιστες από αυτές, οι οποίες ήταν και οι μόνες που τελικά εφαρμόστηκαν.

ABSTRACT

An Application Specific Integrated Circuit (ASIC) is an integrated circuit that has been designed for a very specific application. It's design process consists of thirteen steps, which are examined in the first chapter. This thesis firstly investigates the FBM algorithm, which was designed for the Legalization step, which is part of the design process of an ASIC. The Legalization step is described before we dive deeper into the algorithm FBM and its functionality. With this the second chapter was finished, and then some modifications and additions were considered to be applied to the algorithm, in order to make it more efficient, and to make it able to implement some optimizations to the existing circuit. After all these were implemented, the results were written down, in order to come to a conclusion about how effective all the changes that were made were, and which of them were the most optimal, which at the end were the only ones that were applied.

Table of Contents

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

1.1 Φυσική σχεδίαση ολοκληρωμένων κυκλωμάτων

1.1.α Τι είναι ένα ενσωματωμένο σύστημα

Σήμερα με την εκρηκτική άνοδο και εξέλιξη της τεχνολογίας όλη η ανθρωπότητα έχει στα χέρια της μια τεράστια γκάμα ηλεκτρονικών συσκευών και μηχανημάτων τα οποία εξυπηρετούν τόσο την εργασία όσο και τις ανάγκες των ανθρώπων για επικοινωνία και διασκέδαση. Όλοι βλέπουν και χρησιμοποιούν όλα αυτά τα τεχνολογικά επιτεύγματα, πολλοί όμως δεν γνωρίζουν ότι πίσω από αυτά υπάρχει ένας «μικρόκοσμος» υπολογιστικών συστημάτων που είναι υπεύθυνα για όλες τις λειτουργίες τους. Αυτά τα υπολογιστικά συστήματα είναι τα ενσωματωμένα συστήματα. Τα ενσωματωμένα συστήματα αποτελούνται από μνήμη, διεπαφές εισόδου-εξόδου καθώς και από έναν ή και παραπάνω επεξεργαστές. Μπορούμε να αναλογιστούμε ένα ενσωματωμένο σύστημα ως μια «μικρογραφία» ενός «κανονικού» υπολογιστικού συστήματος. [1]

1.1.β Τι είναι ένα ASIC

Ένα ASIC (Application Specific Integrated Circuit) πρόκειται για ένα ολοκληρωμένο ενσωματωμένο κύκλωμα το οποίο έχει σχεδιαστεί με σκοπό να επιτελεί μια συγκεκριμένη και ξεκάθαρα ορισμένη λειτουργία, είναι αποτέλεσμα προσεκτικού σχεδιασμού και προσφέρει ταχύτητα εκτέλεσης και αποτελεσματικότητα. Χρησιμοποιούνται σε πολλά ηλεκτρονικά προϊόντα που παράγονται σήμερα και κυρίως εφαρμόζονται σε περιπτώσεις όπου υπάρχουν συγκεκριμένες προδιαγραφές επεξεργασίας, όπως παραδείγματος χάριν στις τηλεπικοινωνίες, ενώ το κόστος τους τείνει να είναι αυξημένο. Ένα ASIC μπορεί να συνδυάζει σε ένα chip τόσο αναλογικά, όσο και ψηφιακά δομικά στοιχεία. [2]

Για την κατασκευή των ASIC ακολουθούνται δύο μέθοδοι:

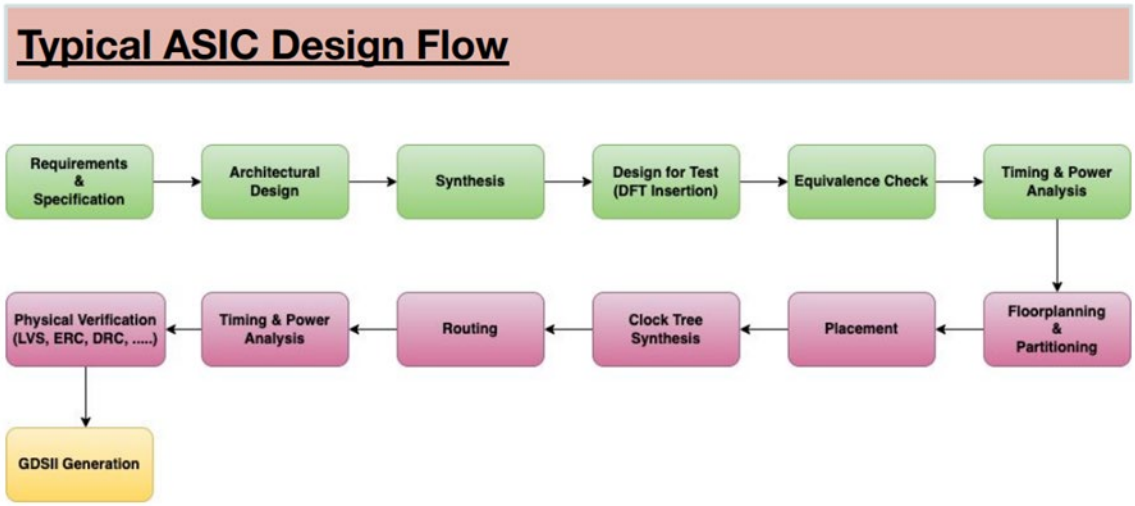
-Gate array: Κατασκευή ενός ASIC χρησιμοποιώντας ένα προκατασκευασμένο chip το οποίο στην πορεία συνδέεται με logic devices.

-Full custom: Πρόκειται για κατασκευές με περισσότερο κόστος από την προηγούμενη κατηγορία και αρκετά πιο πολύπλοκες. [3]

Η διαδικασία ανάπτυξης ενός τυπικού ASIC μπορεί να κρατήσει από μήνες μέχρι και χρόνια.

1.1.γ Μια τυπική διαδικασία σχεδίασης ολοκληρωμένου κυκλώματος

Μια διαδικασία σχεδίασης ολοκληρωμένου κυκλώματος αποτελείται συνολικά από 13 βήματα, τα οποία φαίνονται συγκεντρωτικά στην εικόνα 1.1.



Εικόνα 1.1 Μια σχηματική αναπαράσταση της διαδικασίας σχεδίασης ενός ολοκληρωμένου κυκλώματος.

- Requirements and specification

Πρόκειται για το 1ο βήμα της διαδικασίας σχεδίασης, στο οποίο ορίζονται ξεκάθαρα οι απαιτήσεις και οι προδιαγραφές του κυκλώματος που πρόκειται να αναπτυχθεί. Είναι η διαδικασία καταγραφής όλων των πληροφοριών που καθορίζουν τη σχεδίαση. Παραδείγματα τέτοιων πληροφοριών είναι τα αναμενόμενα αποτελέσματα μετά τη σωστή λειτουργία του κυκλώματος, η επιθυμητή απόδοση, τα ποσά ενέργειας που θα καταναλωθούν, τα υλικά που θα χρησιμοποιηθούν και άλλα. Πολλές φορές περιλαμβάνει και συζητήσεις με έμπειρους σχεδιαστές ώστε να γίνουν καλύτερα κατανοητές οι ανάγκες και να υπάρξει μια όσο το δυνατόν πιο σωστή καθοδήγηση γίνεται στο να οικοδομηθεί μια ολοκληρωμένη και διεξοδικά μελετημένη βάση πάνω στην οποία επρόκειτο να στηριχθούν όλα τα επόμενα βήματα της διαδικασίας.

- Architectural Design

Εδώ καθορίζεται η γενική δομή που θα έχει η σχεδίαση, η διάταξη που τα λειτουργικά blocks θα έχουν στο χώρο όπως και το κομμάτι της διασυνδεσιμότητάς τους. Είναι ένα βήμα στο οποίο απαιτούνται αρκετοί πειραματισμοί από τους κατασκευαστές μέχρι να βρεθεί μια προσέγγιση σχεδιασμού αρχιτεκτονικής που θα βελτιστοποιεί την απόδοση, λαμβάνοντας με προσοχή υπόψιν τους διαθέσιμους πόρους καθώς και μεριμνώντας για την διατήρηση των επιπέδων του κόστους στα αρχικά προσυμφωνημένα επίπεδα. Στο τέλος του βήματος έχει επιτευχθεί η

αναλυτική περιγραφή της αρχιτεκτονικής που επιλέχθηκε και έχει ετοιμαστεί μία αρκετά αναλυτική διαγραμματική αναπαράσταση των blocks που θα χρησιμοποιηθούν.

- Synthesis

Στο βήμα αυτό γίνεται μετατροπή της σχεδίασης που έχει υλοποιηθεί χρησιμοποιώντας μια γλώσσα περιγραφής υλικού (HDL) σε ένα netlist επιπέδου πυλών, κάτι που αποτελεί την αναπαράσταση μιας φυσικής (υλικής) υλοποίησης της σχεδίασης. [4]

- Logic Synthesis :Υλοποιεί ακριβώς αυτό.

-Physical Aware Synthesis: Ασχολείται με ότι ασχολείται και η διαδικασία του logic synthesis, με τη μόνη διαφορά ότι σαν επιπλέον βήμα προσθέτει και τα κελιά στο διαθέσιμο χώρο της σχεδίασης, πάντα σεβόμενη τους κανόνες και περιορισμούς που έχουν τεθεί.[5]

- Design for test (DFT)

Στο συγκεκριμένο βήμα με χρήση κατάλληλων τεχνικών και εργαλείων επισφραγίζεται η σωστή λειτουργία των κυκλωμάτων στην πράξη. Γίνονται εγκαίρως ορατές πιθανές ατέλειες και εντοπίζονται με επιτυχία πιθανά σφάλματα, τόσο λειτουργικά όσο και χρονισμού. Πρόκειται για ένα ιδιαίτερης σημασίας βήμα, με την έλλειψη του οποίου πολλά ψεγάδια και ατέλειες του κυκλώματος, με το να μην γίνονται εγκαίρως αντιληπτά, θα εμφανίζονταν σε αρκετά προχωρημένο στάδιο της σχεδίασης, όπου η αντιμετώπισή τους θα ήταν πολύ πιο δύσκολη αλλά και χρονοβόρα, κάτι που σίγουρα θα καθυστερούσε και την κυκλοφορία του προϊόντος στην αγορά και θα επέφερε οικονομικές απώλειες. Μία από τις πιο ευρέως διαδεδομένες τεχνικές που χρησιμοποιούνται για το DFT είναι η διάσπαση του κυκλώματος σε ένα σύνολο από μικρότερα κυκλώματα. Με αυτό τον τρόπο επικεντρωνόμαστε σε πολλές μικρότερες σχεδιάσεις, κάτι που συνεπάγεται πολύ πιο εύκολος και γρήγορος έλεγχος και διόρθωση πιθανών λαθών.[6]

- Equivalence check

Χρησιμοποιείται για την επαλήθευση ότι δύο διαφορετικές σχεδιάσεις έχουν τα ίδια αποτελέσματα. Πιο συγκεκριμένα, όταν μια σχεδίαση έχει επαληθευτεί λειτουργικά και υπάρχει σαφής κατανόηση της συμπεριφοράς της και θέλουμε να εξετάσουμε αν μια διαφορετικά δομημένη σχεδίαση έχει παρόμοια συμπεριφορά με την επαληθευμένη, τότε χρησιμοποιείται το συγκεκριμένο βήμα. Το equivalence checking αποτελείται από 3 μέρη:

-Setup: Εδώ πραγματοποιείται ανάγνωση των εισόδων που δίνονται και προετοιμάζονται οι δομές δεδομένων που θα είναι απαραίτητες στη συνέχεια.

-Mapping: Εντοπισμός των σημείων στα οποία θα γίνει η σύγκριση και σύγκριση των designs.

-Compare : Εδώ παράγονται τα τελικά αποτελέσματα. [7]

- Timing analysis

Σε αυτό το βήμα στόχος είναι η ανάλυση του χρονισμού. Μελετώνται τόσο η καθυστέρηση που παρατηρείται αθροιστικά σε όλη τη σχεδίαση όσο και επιμέρους περιπτώσεις καθυστερήσεων.

-Static time analysis: Η σχεδίαση διασπάται σε timing paths, και γίνεται έλεγχος για χρονικές παραβιάσεις υπολογίζοντας την καθυστέρηση διάδοσης του σήματος κατά μήκος κάθε ενός μονοπατιού. [8]

-Dynamic time analysis: Χρήση test vectors ως είσοδοι και έλεγχος για το αν τα output vectors που παράγονται έχουν τα επιθυμητά αποτελέσματα. [9]

- Power analysis

Πρόκειται για τον υπολογισμό της συνολικής καταναλισκόμενης ισχύος του κυκλώματος. Έχουμε συνολικά τρεις περιπτώσεις κατανάλωσης ισχύος:

-Dynamic Power: Είναι η απώλεια ισχύος σε ένα CMOS κατά τη διαφοροποίηση των εισόδων.[10]

-Short Circuit Power:Δημιουργείται όταν τα NMOS και PMOS τρανζίστορ είναι ταυτόχρονα ανοιχτά, δημιουργώντας κατά αυτόν τον τρόπο ένα αγώγιμο μονοπάτι.

-Static Power: Πρόκειται απλά για την περίπτωση διαρροής ισχύος

Το power analysis μπορεί να λάβει χώρα σε πολλά επίπεδα αφαίρεσης, όπως τα: Circuit Level Power Analysis, Static Power Analysis και Register Transfer Level Power Analysis.

- Partitioning

Partitioning ορίζεται η διαδικασία κατά την οποία το κύκλωμα που διαθέτουμε διασπάται σε μικρότερα τμήματα με στόχο να μειωθούν όσο το δυνατόν περισσότερο γίνεται οι συνδέσεις μεταξύ των υποκυκλωμάτων και κατά συνέπεια να έχουμε βελτιστοποιημένη απόδοση και μια πιο αυτόνομη σχεδίαση. Απαιτείται προσοχή βέβαια κατά τη διαδικασία μείωσης των συνολικών συνδέσεων μεταξύ των επιμέρους τμημάτων, καθώς η περίπτωση που στο τέλος του partitioning είναι είτε πολλές είτε αρκετά λίγες θα οδηγήσει σε μια προβληματική σχεδίαση [11]. Με τη διαδικασία του partitioning συμβάλλουμε και στην απλοποίηση του routing, ένα από τα επόμενα βήματα στη ροή σχεδίασης.

- Floorplanning

Το στάδιο αυτό ασχολείται με την χωροθέτηση των blocks της σχεδίασης σε ένα chip πυριτίου[12]. Σε πολύ αρχικό στάδιο πρέπει να εισαχθούν κάποιοι φάκελοι εισόδου στο κατάλληλο εργαλείο που θα υλοποιήσει το floorplan και να γίνουν κάποιοι απαραίτητοι έλεγχοι ως προς την ορθότητα και εγκυρότητα αυτών των φακέλων, βήμα ιδιαίτερης σημασίας καθώς αυτοί οι φάκελοι θα είναι η βάση για την σωστή υλοποίηση αυτού του βήματος.

Για την σωστή υλοποίηση του floorplanning οι σχεδιαστές πρέπει δώσουν ιδιαίτερη βαρύτητα σε δύο παραμέτρους, οι οποίες καθορίζουν σε σημαντικό βαθμό τη σχεδίαση. Πρόκειται για την παράμερο core area, η οποία δεν είναι τίποτε άλλο από τον χώρο μέσα στο chip στον οποίο θα τοποθετηθούν όλα τα cells της σχεδίασης και για το Aspect ratio, μια τιμή που αναπαριστά την διαφορά ανάμεσα στο ύψος και στο πλάτος της πρώτης παραμέτρου. Το aspect ratio ορίζεται ως ο λόγος του ύψους του core area προς το πλάτους του.

Υπάρχουν τρεις διαφορετικές μέθοδοι για το floorplanning:

-Abutted floorplan: Δεν υπάρχει καθόλου χώρος ανάμεσα στα στοιχεία του κυκλώματος κατά το τέλος της διαδικασίας.

-Non-Abutted floorplan: Σε αυτήν την περίπτωση υπάρχουν μικρά κενά χώρου στο τελικό κύκλωμα.

-Mixed floorplan: Κατάλληλος συνδυασμός των δύο παραπάνω μεθόδων.[13]

Με το πέρας της διαδικασίας αυτής και έχοντας οπτικοποιήσει πλέον τη συνολική σχεδίαση είμαστε σε θέση να υλοποιήσουμε βελτιώσεις σε πολλούς τομείς.

- Placement

Είναι το βήμα στο οποίο προσπαθούμε να βρούμε μια κατάλληλη φυσική θέση εντός των διαθέσιμων ορίων που διαθέτουμε για κάθε cell της σχεδίασης, πάντα τηρώντας τις προδιαγραφές και τους περιορισμούς που έχουν εκ των προτέρων τεθεί. Αναλυτικότερα, το placement μπορεί να περιγραφεί ως η διαδικασία κατά την οποία προσπαθούμε να δημιουργήσουμε μια συγκεκριμένη διάταξη των στοιχείων του κυκλώματος που έχουμε καθώς και των μεταξύ τους συνδέσεων έτσι ώστε στο τέλος της διαδικασίας να έχουμε στοιχεία τοποθετημένα στο χώρο με τέτοιο τρόπο ώστε να έχουμε μείωση του χώρου που καταλαμβάνει τώρα το κύκλωμα σε σχέση με πριν. Εκτός από αυτό, μετά την υλοποίηση του placement θα πρέπει το μήκος του καλωδίου που χρησιμοποιείται από το κύκλωμα να είναι μικρότερο από αυτό που χρησιμοποιούνταν πριν την εφαρμογή του βήματος. Σε αυτό το βήμα η είσοδος που δίνεται είναι η συνολική διάταξη του κυκλώματος και τα επιμέρους χαρακτηριστικά του κάθε ενός cell, όπως και όλες οι συνδέσεις που παρατηρούνται μεταξύ των κελιών. Στο τελικό αποτέλεσμα έχουμε στα χέρια μας τις νέες θέσεις των κελιών στο χώρο, δηλαδή τις νέες συντεταγμένες του κάθε ενός cell.[17]

Χωρίζεται σε τρία επιμέρους βήματα:

-Global placement: Είναι μια πρώτη τοποθέτηση των κελιών στο χώρο.

-Legalization: Μετακίνηση των κελιών στο χώρο με στόχο τα cells να βρίσκονται εντός των ορίων των διαθέσιμων γραμμών και εξάλειψη τυχόν περιπτώσεων επικάλυψης.[14]

-Detailed placement: Επιπλέον βελτιστοποιήσεις στο προηγούμενο αποτέλεσμα.[15]

- Clock tree synthesis

Διαδικασία που ασχολείται με το σωστό διαμοιρασμό του χρόνου του ρολογιού σε όλα τα στοιχεία του κυκλώματος. Δέχεται ως είσοδο τις θέσεις των στοιχείων της

σχεδίασης και στοχεύει με την ολοκλήρωση της εκτέλεσής του τόσο στο να φτάνει το σήμα του ρολογιού σε κάθε σημείο του κυκλώματος με επιτυχία όσο και με τη λιγότερη δυνατή καθυστέρηση. Για το σωστό χειρισμό της καθυστέρησης λοιπόν, στο κύκλωμα εισάγονται buffers και inverters. Η αρχή του δικτύου διαμοιρασμού του ρολογιού είναι το clock source, από το οποίο στέλνονται όλα τα επιμέρους σήματα του ρολογιού στο κύκλωμα και τα τερματικά του σημεία είναι τα clock pins που έχει το κάθε cell της σχεδίασης. Πρόκειται για ένα βήμα το οποίο πρέπει να γίνει με ιδιαίτερη προσοχή ώστε το τελικό δίκτυο διαμοιρασμού του ρολογιού να είναι όσο το δυνατόν πιο αποτελεσματικό ώστε να γίνει στο τέλος όσο το δυνατόν πιο συνετή κατανάλωση ισχύος από το δίκτυο, καθώς ήδη τα ποσά ενέργειας που απαιτεί ένα τέτοιο δίκτυο φτάνουν τις περισσότερες φορές τα επίπεδα του 30%-40% της συνολικής ισχύος του κυκλώματος [16]

- Routing

Σε αυτό το βήμα της σχεδίασης στόχος είναι η ολοκληρωμένη υλοποίηση των συνδέσεων μεταξύ των pins του κυκλώματος, δίνοντας έμφαση στον βέλτιστο σχεδιασμό των καλωδίων. Με την ολοκλήρωση των δύο προηγούμενων βημάτων που αναφέραμε(Clock tree synthesis και placement) πλέον είναι γνωστές οι θέσεις όλων των στοιχείων που περιλαμβάνονται στη σχεδίαση. Στο στάδιο του routing λοιπόν υλοποιούνται αυτές ακριβώς οι συνδέσεις, δίνεται ιδιαίτερη προσοχή στο να διατηρηθεί το τελικό μήκος του καλωδίου σε όσο το δυνατόν χαμηλότερα επίπεδα γίνεται, καθώς και στο να μην δημιουργούνται παραβιάσεις στα επίπεδα χρονισμού που έχουν τεθεί για την προκειμένη σχεδίαση, όπως και στους γενικότερους σχεδιαστικούς κανόνες που υπάρχουν. Όλη η διαδικασία πρέπει να γίνει με δεδομένο ότι στο τελικό αποτέλεσμα τα καλώδια πρέπει να χαραχτούν μόνο οριζόντια και κάθετα. Η διαδικασία του routing αποτελείται από συνολικά 4 βήματα. Το πρώτο βήμα είναι το Global routing, κατά τη διάρκεια του οποίου η υλοποίηση διασπάται σε μικρότερες υλοποιήσεις, με στόχο την εύρεση όλων των «μονοπατιών» των συνδέσεων.[22] Το δεύτερο βήμα είναι το Track Assignment(TA), το οποίο προσθέτει μεταλλικά layers στη σχεδίαση, ακολουθεί το τρίτο βήμα του Detailed Routing που ελέγχει αν έχει μέχρι στιγμής παραβιαστεί κάποιος από τους αρχικούς κανόνες και περιορισμούς της σχεδίασης και προσπαθεί να κάνει ανάλογες διορθώσεις. Τέλος έχουμε το τέταρτο βήμα (Search and Repair), κατά το οποίο υλοποιείται ένας τελικός έλεγχος για τυχόν παραβιάσεις που δεν επιλύθηκαν από το προηγούμενο βήμα.[23]

Κάποιοι από τους πιο γνωστούς τύπους δρομολόγησης είναι οι: Maze routing και Channel routing.

-Maze routing: Εύρεση του συντομότερου μονοπατιού μεταξύ δύο κόμβων. Χρονοβόρος αλγόριθμος που για την υλοποίηση του απαιτείται και κατανάλωση πολλής μνήμης.[18]

-Channel routing: Χρησιμοποιείται για μείωση του μεγέθους της περιοχής που το ενσωματωμένο κύκλωμα καταλαμβάνει. Είναι πιο αποτελεσματικός αλγόριθμος από τον Maze routing.

- Timing analysis

Στο σημείο που έχει φτάσει η σχεδίαση έχει ήδη πραγματοποιηθεί πλήθος αλλαγών και τροποποιήσεων από την τελευταία φορά που υπήρξε έλεγχος στον χρονισμό του κυκλώματος, οπότε για τη διασφάλιση της διατήρησης του χρονισμού στα επιτρεπτά επίπεδα είναι ιδιαίτερα σημαντική η επανεξέτασή του. Οι αλλαγές που θα πρέπει να γίνουν σε αυτό το βήμα στην πλειοψηφία των περιπτώσεων είναι εύκολα υλοποιήσιμες.

- Physical verification

Το κύκλωμα πλέον βρίσκεται πολύ κοντά στην τελική του μορφή, και σε αυτό το βήμα γίνεται ένας γενικός τελευταίος έλεγχος του κυκλώματος πάνω σε όλους τους τομείς ώστε να διασφαλιστεί ότι το τελικό αποτέλεσμα είναι άρτιο, πλήρως λειτουργικό και βασισμένο αυστηρά στους αρχικούς περιορισμούς και απαιτήσεις. Σε περίπτωση που το βήμα αυτό παραλειφθεί πιθανά λάθη και ατέλειες του κυκλώματος δεν θα γίνουν εγκαίρως ορατά και κατά συνέπεια δεν θα διορθωθούν, κάτι που θα οδηγήσει σίγουρα σε ένα από λίγο έως πολύ προβληματικό τελικό προϊόν, οδηγώντας όλη την κατασκευαστική πορεία και προσπάθεια στην αποτυχία. Η διαδικασία διευκολύνεται με τη χρήση κατάλληλων για αυτό το σκοπό εργαλείων, τα οποία σε πολύ σύντομο χρονικό διάστημα εντοπίζουν και επιστρέφουν ειδοποιήσεις σχετικά με τις πιθανές αλλαγές που πρέπει να γίνουν Οι έλεγχοι που πραγματοποιούνται είναι οι εξής:

-Design Rule Check (DRC) : Έλεγχος για το πόσο πιστά ακολουθήθηκαν οι αρχικοί σχεδιαστικοί όροι που είχαν τεθεί σε προηγούμενο στάδιο της σχεδίασης.

-Layout vs Schematic (LVS) : Επιβεβαίωση ότι το τελικό κύκλωμα ανταποκρίνεται σε σχέδια που είχαν παραχθεί σε προηγούμενα βήματα της σχεδιαστικής πορείας.

-Electrical Rule Checking (ERC) : Έλεγχος για το αν όλες οι υλοποιήσεις σχετικά με τον ηλεκτρισμό στο κύκλωμα είναι σύμφωνες με τους αρχικούς όρους. Το εργαλείο που υλοποιεί το ERC πραγματοποιεί έλεγχο στο τελικό αποτέλεσμα και επιστρέφει ειδοποιήσεις σχετικά με τυχόν διορθώσεις που πρέπει να γίνουν. [19]

-Design for Manufacturability (DFM): Εδώ πραγματοποιούνται αλλαγές-βελτιώσεις στο τελικό αποτέλεσμα της σχεδίασης ώστε να είναι πιο εύκολη και ομαλή η τελική κατασκευαστική του πορεία στο εργοστάσιο.[20]

- GDSII generation

Στο τελικό βήμα της σχεδίασης, παράγεται ένα αρχείο, και πιο συγκεκριμένα ένα binary file το οποίο έχει συγκεντρωμένα πολλά χαρακτηριστικά σχετικά με τη σχεδίαση. Περιέχει σχηματικές αναπαραστάσεις και γεωμετρικά σχήματα των στοιχείων του τελικού κυκλώματος που έχουμε δημιουργήσει. Τα στοιχεία είναι ταξινομημένα ανά layer και κάθε ένα χαρακτηρίζεται από τις τιμές layer number, η οποία υποδηλώνει το layer στο οποίο έχει τοποθετηθεί το στοιχείο που

εξετάζουμε, datatype και texttype. Το αρχείο αυτό χρησιμοποιείται σε περιπτώσεις που θέλουμε να μεταφέρουμε την αρχιτεκτονική που έχουμε σχεδιάσει από ένα εργαλείο σχεδίασης σε κάποιο άλλο, αποφεύγοντας πιθανά προβλήματα και δυσκολίες που μπορούν να προκύψουν σε περιπτώσεις που το ένα εργαλείο σχεδίασης λειτουργεί με διαφορετικά data formats σε σχέση με το άλλο, κάτι που παλαιότερα ήταν ιδιαίτερα σύνηθες. Έχοντας έτσι έναν καθολικό τύπο αρχείου που να αναγνωρίζεται από όλα τα εργαλεία σχεδίασης απλουστευόταν όλη η διαδικασία σε σημαντικό βαθμό. Ο συγκεκριμένος τύπος αρχείου είναι απαραίτητος για τις κατασκευαστικές εταιρίες που θα αναλάβουν την κατασκευή του κυκλώματος που έχει σχεδιαστεί. Σε αυτό μπορούν να δουν με λεπτομέρεια και ακρίβεια στοιχεία που έχουν να κάνουν με την τελική φυσική μορφή που θα έχει το κύκλωμα.[21]

ΚΕΦΑΛΑΙΟ 2. Αλγόριθμος FBM

2.1 Εισαγωγή

Όπως αναφέρθηκε και στην προηγούμενη ενότητα, το βήμα του placement χωρίζεται σε τρία μέρη: Στο global placement, στο legalization και στο detailed placement.

Αναλυτικότερα, στο global placement τα cells τοποθετούνται στο χώρο χωρίς να δοθεί ιδιαίτερη έμφαση στην τήρηση των περιοριστικών όρων που έχουν τεθεί για τη συγκεκριμένη σχεδίαση. Σαν μια πρώτη μέριμνα το βήμα αυτό απλά φροντίζει ώστε τα cells να είναι γενικά διασκορπισμένα στο διαθέσιμο χώρο. Στο τέλος του βήματος θα υπάρχουν ακόμα μερικές περιπτώσεις επικαλύψεων (overlaps) ανάμεσα στα cells και σε πολλές περιπτώσεις τα cells δεν θα βρίσκονται τοποθετημένα εντός των ορίων μιας γραμμής. Πρόκειται λοιπόν απλά για μια πρώτη τοποθέτηση των cell στο χώρο.[14][24]

Το βήμα του legalization ουσιαστικά παίρνει τα τελικά αποτελέσματα του global placement και εφαρμόζει συγκεκριμένες βελτιστοποιήσεις πάνω σε αυτά. Φροντίζει για την μετακίνηση των κελιών που επικαλύπτονται πλήρως ή μερικώς και για την τοποθέτηση των κελιών εξ' ολοκλήρου εντός των ορίων των διαθέσιμων γραμμών, προσπαθώντας ταυτόχρονα να αποκλίνει όσο το δυνατόν λιγότερο γίνεται από το τελικό αποτέλεσμα του global placement. Για να επιτευχθεί αυτό όσο το δυνατόν περισσότερο, στο βήμα αυτό το τα cells μετακινούνται όσο το δυνατόν λιγότερο γίνεται.[14]

Στο detailed placement, υλοποιούμε μια σειρά βελτιστοποιήσεων πάνω στο κύκλωμα που έχουμε αυτή τη στιγμή.[15] Μια από τις βελτιστοποιήσεις που η έξοδος του legalization δέχεται είναι ως προς το συνολικό μήκος του καλωδίου (προσπάθεια ανταλλαγής θέσεων ανάμεσα σε cells ίδιου πλάτους αλλά

διαφορετικού net με σκοπό την ελαχιστοποίηση του μήκους καλωδίου του τελικού κυκλώματος).

2.2 Legalization

Το δεύτερο βήμα του placement με την πάροδο του χρόνου τείνει να παρουσιάζει όλο και μεγαλύτερες προκλήσεις. Σε παλαιότερες τεχνολογίες οι σχεδιαστικοί κανόνες που θέτονταν για ένα κύκλωμα δεν χαρακτηρίζονταν από ιδιαίτερη πολυπλοκότητα. Κατά την ενασχόληση όμως με νέες και πιο προηγμένες τεχνολογίες, τα πράγματα είναι πολύ διαφορετικά. Οι legalizing αλγόριθμοι για να μπορούν να ανταποκριθούν στα νέα και ραγδαία εξελισσόμενα δεδομένα οφείλουν εκ φύσεως να είναι γρήγοροι και ισχυροί.[14] Ας μην ξεχνάμε κιόλας ότι η διαδικασία του legalization είναι μια διαδικασία που κατά την υλοποίηση της σχεδίασης πρέπει να εκτελεστεί πολλές φορές, οπότε είναι ιδιαίτερα σημαντικό να είναι γρήγορη.[25]

Κάποιοι από τους πιο διαδεδομένους αλγορίθμους για το legalization είναι οι: Tetris και Abacus.

-Tetris: Ξεκινά με την ταξινόμηση των cells με βάση την χ τους θέση από το μικρότερο στο μεγαλύτερο. Στη συνέχεια, αφού βρεθεί ο πρώτος διαθέσιμος χώρος σε κάθε γραμμή στον οποίο χωράει το cell (ξεκινώντας από τα αριστερά προς τα δεξιά) και έπειτα όλοι οι υπόλοιποι(εάν υπάρχουν), γίνεται επιλογή της θέσης που βρίσκεται πιο κοντά στην αρχική θέση. Ένα από τα σημαντικότερα χαρακτηριστικά της διαδικασίας του αλγορίθμου είναι ότι κάθε μία τοποθέτηση είναι οριστική και δεν υπάρχει δυνατότητα αλλαγής της θέσης του cell μετά από αυτό. Πρόκειται για έναν greedy αλγόριθμο.[26]

-Abacus: Το πρώτο βήμα του αλγορίθμου είναι ίδιο με το πρώτο βήμα του Tetris, δηλαδή ταξινόμηση των cells κατά τον άξονα χ σε αύξουσα σειρά. Εν συνεχεία κάνει legalize ένα-ένα τα cells τοποθετώντας τα σε συγκεκριμένες γραμμές, προσέχοντας να παρεκκλίνει όσο το δυνατόν λιγότερο γίνεται από τις αρχικές θέσεις που είχαν τα cells στο αποτέλεσμα της εξόδου του global placement, μέχρι να βρεθεί η βέλτιστη τοποθέτηση. Διαλέγοντας μια θέση χ για κάθε cell αφού τοποθετηθεί, αναλαμβάνει και τη μετακίνηση κελιών που ήδη έχουν περάσει από τη φάση του legalization, κάτι που όπως αναφέραμε ήδη ο Tetris δεν κάνει.[27] Ο αλγόριθμος χαρακτηρίζεται από αρκετά αυξημένο χρόνο εκτέλεσης, κάτι που τον κάνει να μην είναι η βέλτιστη επιλογή για το legalization σε περιπτώσεις που υπάρχουν αυστηρές προδιαγραφές ως προς το χρονισμό.[26]

Οι legalizing αλγόριθμοι δύναται να συγκριθούν ως προς το χρόνο εκτέλεσης, το HPWL και το Average Displacement.

-HPWL(Half Perimeter Wirelength): Χρησιμοποιείται για τον υπολογισμό του συνολικού μήκους καλωδίου που θα χρειαστεί για το κύκλωμα.

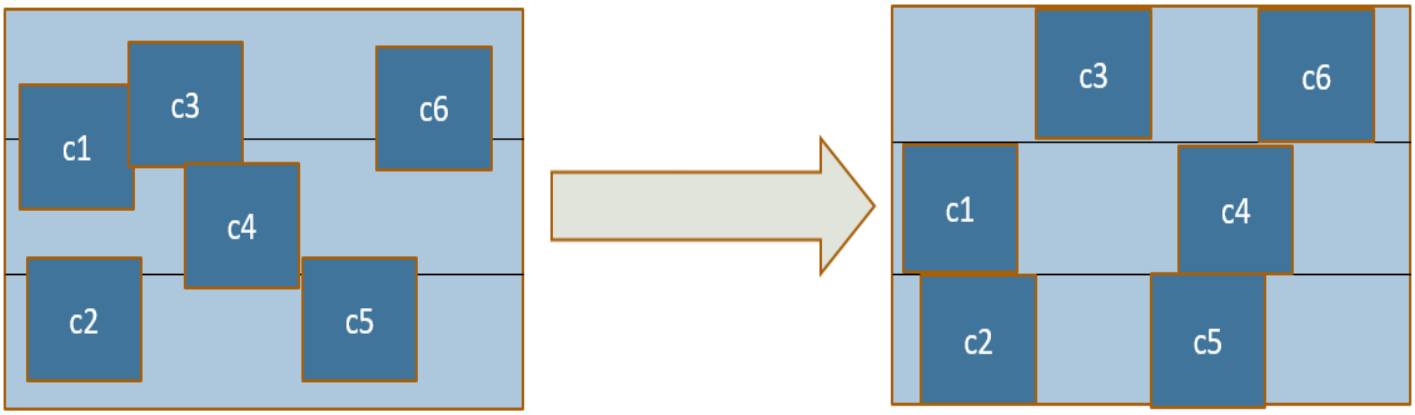
-Average Displacement: Υπολογίζει τις διαφορές στις θέσεις των κελιών που προέκυψαν στο κύκλωμα εξόδου του global placement μετά την εφαρμογή του legalization. [14]

2.3 Εισαγωγή στον FBM legalizer

Ο αλγόριθμος που επρόκειτο να παρουσιαστεί είναι ένας αλγόριθμος για το στάδιο του legalizing, ο FBM legalizer. Στηρίζεται στον Abacus που αναλύθηκε προηγουμένως. Ο Abacus λειτουργεί υπολογίζοντας πάντα για κάθε cell ποια είναι η βέλτιστη θέση τοποθέτησης για να πετύχει το legalizing. Μετά το πέρας της εκτέλεσής του είναι δυνατόν ένα cell να έχει αλλάξει θέση πολλές φορές μέσα στο κύκλωμα. Ο FBM legalizer εξαλείφει τα overlaps με το να ελέγχει κάθε ζεύγος από cells για περιπτώσεις μερικής ή πλήρης επικάλυψης. Μετακίνηση του δεξιότερου cell στα δεξιά κατά τόση απόσταση όση απαιτείται ώστε να εξαλειφθεί το overlap είναι το μέτρο που ακολουθείται σε περίπτωση που διαπιστωθεί επικάλυψη. Ελέγχοντας και πραγματοποιώντας μετακινήσεις μόνο για την περίπτωση που όντως διαπιστωθεί κατάσταση επικάλυψης ο FBM λειτουργεί με πιο αποδοτικό τρόπο από τον Abacus, καθώς πλέον αποφεύγονται οι συνεχείς και πολλές φορές άσκοπες μετακινήσεις κελιών.[14]

2.4 Ανάλυση του αλγορίθμου FBM

Η διαδικασία των βημάτων του αλγορίθμου FBM φαίνεται στην εικόνα 2.2 Αρχικά θεωρούμε ότι έχουμε ένα σύνολο από n cells, τα οποία έχουν το ίδιο ύψος (height) αλλά διαφέρουν ως προς τα πλάτη (widths). Ο αλγόριθμος ξεκινά τοποθετώντας τα διαθέσιμα cells της σχεδίασης στις γραμμές. Για κάθε ένα cell υπολογίζεται η απόστασή του από κάθε μία γραμμή του κυκλώματος και το cell στο τέλος τοποθετείται στη γραμμή από την οποία απείχε τη μικρότερη απόσταση. Με άλλα λόγια, το cell τοποθετείται στη γραμμή της οποίας η συντεταγμένη y είναι πιο κοντά αριθμητικά στην y συντεταγμένη του cell που πρόκειται να τοποθετηθεί. Έτσι επιλύεται το πρόβλημα όσων cell δεν έχουν τοποθετηθεί εντός των ορίων των διαθέσιμων γραμμών μετά το global placement.[14] Στην εικόνα 2.1 ακριβώς παρακάτω φαίνεται μια μικρή σχηματική αναπαράσταση του 1ου βήματος του αλγορίθμου χρησιμοποιώντας έναν πολύ μικρό αριθμό από cells και γραμμές για απλούστευση.



Εικόνα 2.1 Σχηματική αναπαράσταση του 1ου βήματος του αλγορίθμου FBM

Algorithm 1 FBM Legalizer

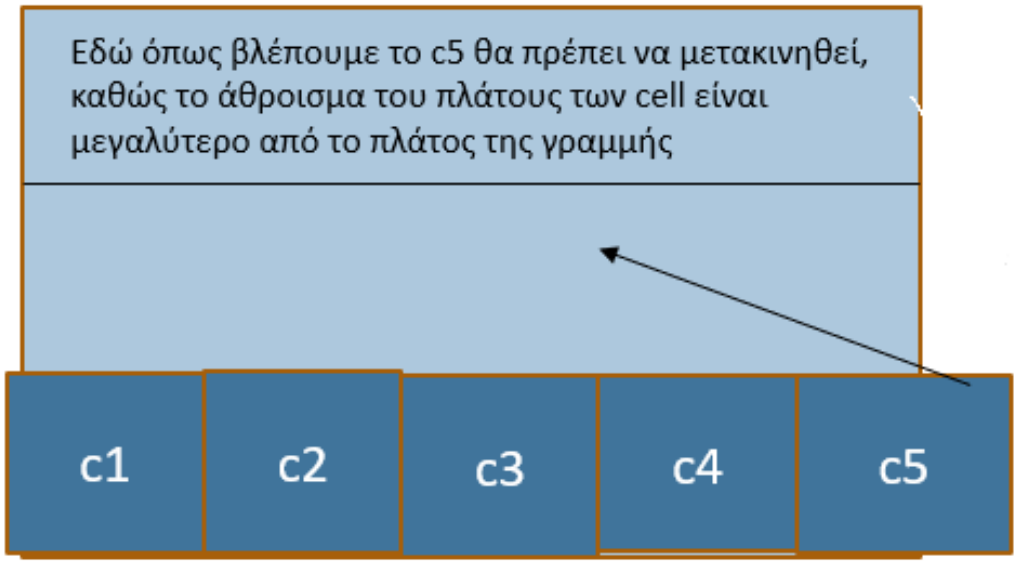
$\{C\} \leftarrow$ set of cells;
Place all cells in the nearest rows according to their positions y;
Sort cells according to width and y-position;
Checks if cells fits in the same row;
Sort cells according to x and y-position;
for i=1 **to** C **do**
  if $Cell_i$ and $Cell_{i+1}$ are overlapping **then**
    $Cell_{i+1}$ is moved to right
  end if
end for
for i=1 **to** C **do**
  if $Cell_i$ is not in a valid position **then**
    Move the $Cell_i$ to a valid position to left
    while $Cell_{i-1}$ not be in a valid position **do**
      Move the $Cell_{i-1}$ untill to a valid position to left
    end while
  end if
end for

Πηγή: : J. Ferreira, P. F. Butzen, C. Meinhardt and R. A. L. Reis, "FBM: A Simple and Fast Algorithm for Placement Legalization," *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genoa, Italy, 2019, pp. 209-212, doi: 10.1109/ICECS46596.2019.8965013

Εικόνα 2.2 Τα βήματα του αλγορίθμου FBM

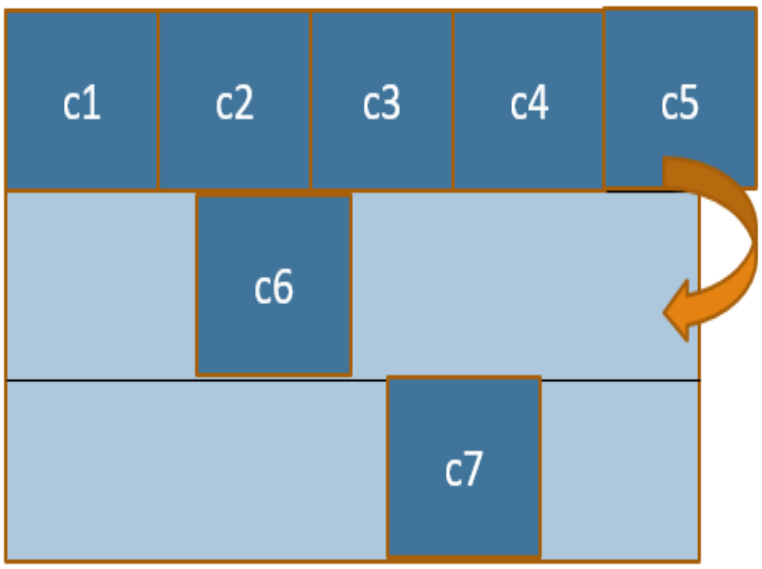
Στο 2ο βήμα του FBM legalizer, τα cells ταξινομούνται με βάση το πλάτος τους στις γραμμές. Δίνεται προτεραιότητα σε αυτά που έχουν μικρότερο width, δηλαδή ταξινομούνται από το μικρότερο στο μεγαλύτερο. Σε περίπτωση που δύο cells τύχει να έχουν το ίδιο πλάτος, ο αλγόριθμος δίνει προτεραιότητα στο cell της χαμηλότερης γραμμής. Ο αλγόριθμος εφαρμόζει την ταξινόμηση αυτή και πραγματοποιεί ενδεχόμενες μετακινήσεις σε όλα τα υπάρχοντα cells του κυκλώματος που δόθηκε ως αποτέλεσμα από το πρώτο βήμα, με εξαίρεση τα αρχικά μεγαλύτερα cells κάθε γραμμής αυτού του κυκλώματος. Αυτά παραμένουν στην ίδια αρχική γραμμή και μετά το πέρας του 2ου βήματος του αλγορίθμου, δηλαδή είναι σίγουρο ότι θα έχουν την ίδια θέση y.[14]

Το 3ο βήμα του FBM ασχολείται με τη διαχείριση των cell που βρέθηκαν εκτός των ορίων των γραμμών μετά την επιτυχή εκτέλεση του 2ου βήματος. Μετά το δεύτερο στάδιο υπάρχει πιθανότητα κατά τη διαδικασία της ταξινόμησης κάποια cells να τοποθετήθηκαν σε περιοχές εκτός του διαθέσιμου χώρου της σχεδίασης. Αυτά τα cell σε καμία περίπτωση δεν πρέπει να παραμείνουν εκεί, καθώς τότε θα είχαμε παραβίαση των ορίων και περιορισμών της σχεδίασης και έναν προβληματικό legalizer. Για κάθε ένα τέτοιο κελί ο αλγόριθμος ψάχνει προσεκτικά κάθε επόμενη γραμμή (δηλαδή κάθε γραμμή πάνω από τη γραμμή που παρουσιάζεται το πρόβλημα) για διαθέσιμο χώρο ώστε να τοποθετηθεί το cell. Στην πρώτη περίπτωση που θα τύχει να υπάρχει χώρος ίσος με το πλάτος του cell, εκεί θα τοποθετηθεί το cell, καταφέρνοντας να το επαναφέρουμε εντός του χώρου της σχεδίασης σε μια καινούργια έγκυρη θέση(εικόνα 2.3). Σε αυτό το βήμα ειδική περίπτωση αποτελεί η κατάσταση όπου το cell βρίσκεται εκτός ορίων στην τελευταία γραμμή(εικόνα 2.4). Όταν συμβεί κάτι τέτοιο, ο αλγόριθμος κάνει ότι έκανε και πριν αλλά αντίστροφα, δηλαδή ελέγχει κάθε γραμμή από πάνω προς τα κάτω. Με την ίδια λογική με πριν το cell θα τοποθετηθεί στον πρώτο διαθέσιμο ελεύθερο χώρο που θα βρεθεί ξεκινώντας από την τελευταία και φτάνοντας μέχρι και την πρώτη γραμμή.[14]



Εικόνα 2.3 Η περίπτωση που το cell πρέπει να μετακινηθεί προς τα πάνω

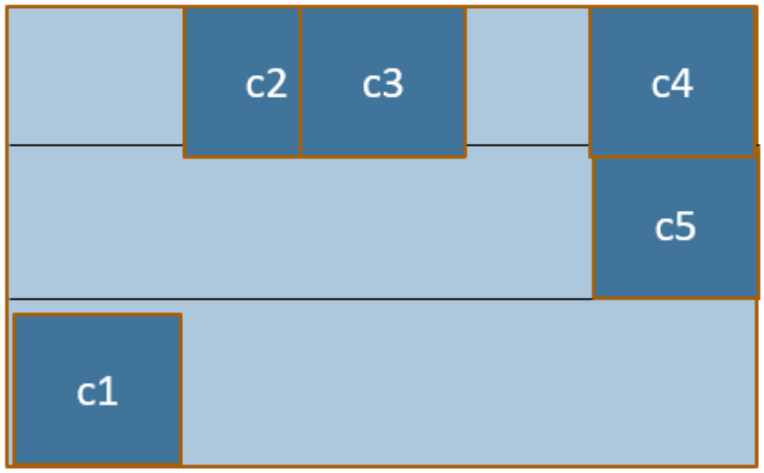
Εδώ είμαστε στην τελευταία γραμμή, οπότε το c5 θα μετακινηθεί μια γραμμή παρακάτω, όπου υπάρχει και διαθέσιμος ελεύθερος χώρος.



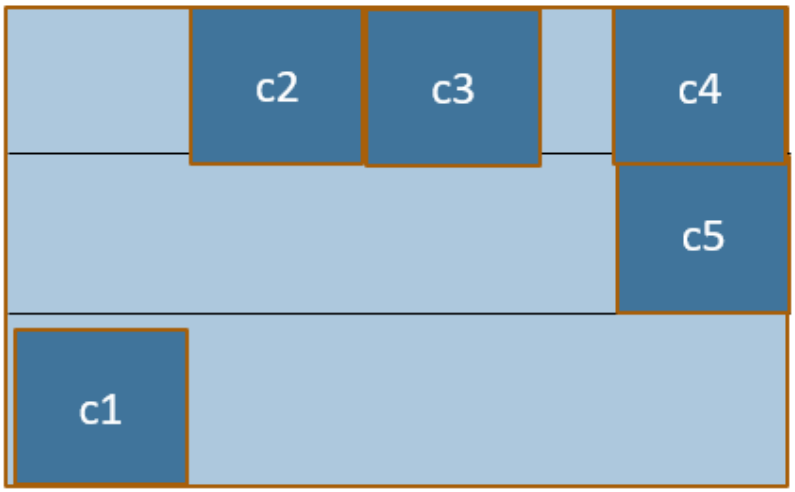
Εικόνα 2.4 Η περίπτωση που το cell πρέπει να μετακινηθεί προς τα κάτω

Σειρά τώρα έχει το 4ο βήμα του αλγορίθμου FBM. Εδώ, εφόσον έχουν πλέον εξαλειφθεί όλα τα cells εκτός σχεδίασης που προέκυψαν από το 2ο βήμα, πραγματοποιείται τοποθέτηση των cells στις διαθέσιμες γραμμές με βάση την χ θέση τους κατά αύξουσα σειρά. Σε περιπτώσεις που συμβαίνει κατά τη σύγκριση να εντοπίζεται η ίδια αριθμητική τιμή για την θέση χ και των δύο cells, προτεραιότητα δίνεται σε αυτό που βρίσκεται στη χαμηλότερη γραμμή, δηλαδή σε αυτό με τη μικρότερη y θέση. Μετά το τέλος του 4ου βήματος του αλγορίθμου υπάρχει πιθανότητα να δημιουργήθηκαν ξανά επικαλύψεις ανάμεσα στα cells. Μια από τις αρμοδιότητες του βήματος που ακολουθεί, που είναι και το τελευταίο του αλγορίθμου, είναι η εξάλειψή τους.[14]

Το 5ο βήμα του legalizer που μελετάμε εκτελείται πάντα, αλλά αλλάζει το αποτέλεσμα του προηγούμενου βήματος μόνο σε περιπτώσεις που η σχεδίαση που προέκυψε ήταν προβληματική. Ο αλγόριθμος ελέγχει όλα τα τοποθετημένα cells της σχεδίασης ανά ζευγάρι. Εάν παρατηρηθεί επικάλυψη, τότε ο αλγόριθμος μετακινεί το cell με τη μεγαλύτερη θέση x από το ζεύγος προς τα δεξιά μέχρι το καθένα να έχει το δικό του ξεχωριστό χώρο(εικόνα 2.5). Το cell που μετακινείται τοποθετείται στον πρώτο διαθέσιμο χώρο που θα βρεθεί κατά μήκος της ίδιας γραμμής.[14]



Στα c2 και c3 παρατηρείται overlapping

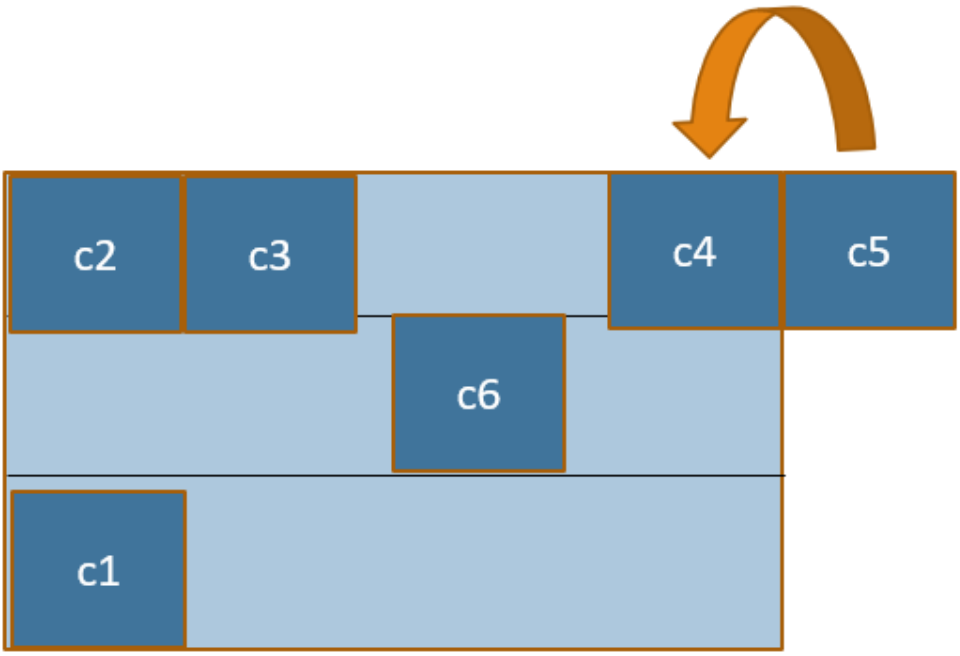


Το πρόβλημα λύθηκε με μετακίνηση του c3 προς τα δεξιά.

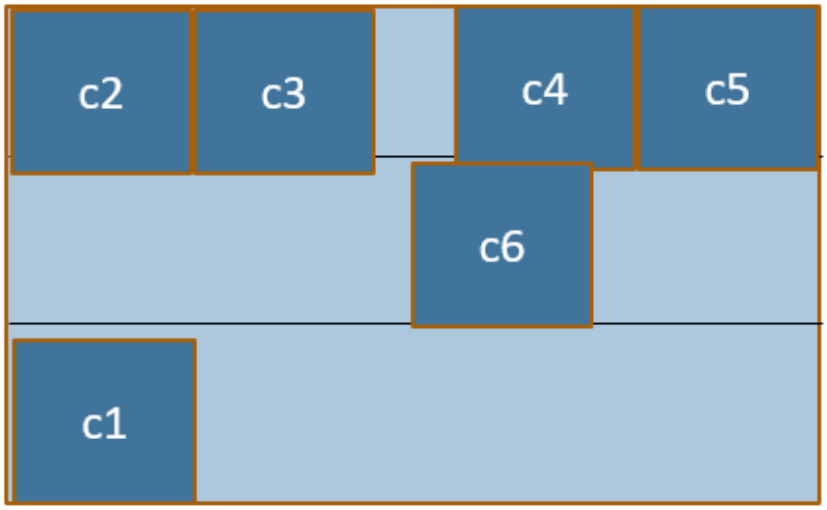
Εικόνα 2.5 Η αντιμετώπιση των overlaps στην περίπτωση του 5ου βήματος

Η δεύτερη «αρμοδιότητα» του τελευταίου βήματος είναι να διασφαλίσει ότι με τις μετακινήσεις που υλοποίησε μέχρι στιγμής προς τα δεξιά δεν μετακίνησε κάποιο cell σε σημείο που δεν ανήκει στη σχεδίαση. Πραγματοποιείται ξανά επαναληπτικός έλεγχος για όλα τα cells της σχεδίασης. Σε περίπτωση που βρεθεί τέτοιο cell, μεταφέρεται προς τα αριστερά αυτή τη φορά μέχρι να βρεθεί σε έγκυρη εντός των ορίων θέση. Εδώ βέβαια θα μπορούσε κανείς σωστά να παρατηρήσει ότι με τη μετακίνηση αυτή θα μπορούσε το cell να τοποθετούνταν από τον αλγόριθμο σε μια θέση που ναι μεν είναι έγκυρη από την άποψη ότι πλέον δεν βρίσκεται εκτός ορίων όπως πριν, αλλά όμως υπάρχει κίνδυνος η θέση που γίνεται η μετακίνηση να είναι ήδη πλήρως ή μερικώς «κατειλημμένη» από κάποιο άλλο cell, οπότε σε μία τέτοια κατάσταση παρουσιάζεται ξανά το πρόβλημα του overlapping. Για αυτή την περίπτωση μεριμνά ο αλγόριθμος κάνοντας κατάλληλο έλεγχο μετά τη μετακίνηση. Εάν βρεθεί όντως περίπτωση επικαλυπτόμενων cell, το «παλιό» cell μετακινείται προς την πρώτη έγκυρη διαθέσιμη θέση στα αριστερά και με αυτό τον τρόπο εξαλείφεται και το overlapping. Το κύκλωμα εξόδου του τελευταίου βήματος του FBM λοιπόν, είναι ένα ορθά legalized κύκλωμα, χωρίς overlapping και με cells που βρίσκονται σίγουρα μέσα στα όρια κάποιας γραμμής.[14]

Η έγκυρη επανατοποθέτηση των cells αναπαρίσταται σχηματικά στην εικόνα 2.6



Εδώ το c5 βρίσκεται εκτός ορίων, οπότε και μετακινείται προς τα αριστερά ώστε να είναι εντός ορίων. Τώρα όμως επικαλύπτεται με το c4.



Το c4 σύμφωνα με τον αλγόριθμο μετακινήθηκε στα αριστερά μέχρι που πλέον βρίσκεται εντός ορίων χωρίς να δημιουργείται επικάλυψη.

Εικόνα 2.6 Η έγκυρη επανατοποθέτηση των cells στην περίπτωση του 5ου βήματος

Ο συγκεκριμένος αλγόριθμος έχει αναλυθεί και ως προς την πολυπλοκότητα. Η πολυπλοκότητα του στην καλύτερη περίπτωση, όταν δηλαδή στο κύκλωμα που δέχεται ως είσοδο από το global placement τα cells τυχαίνει να είναι ήδη legalized, είναι O(N). Στην περίπτωση που όλα τα cells επικαλύπτονται και που υπάρχουν cells εκτός των ορίων του κυκλώματος που δίνεται ως είσοδος στο FBM η πολυπλοκότητα είναι $O(N^2)$. Αυτή είναι και η χειρότερη περίπτωση του αλγορίθμου. Σε κάθε περίπτωση το N αναπαριστά τον συνολικό αριθμό των cells που έχουμε στη διάθεσή μας.[14]

2.5 Συμπεράσματα από την εφαρμογή του αλγορίθμου FBM

Ο FBM εφαρμόστηκε πάνω σε συνολικά 24 κυκλώματα, εκ των οποίων τα 6 δέχτηκαν αλλαγές ως προς την πυκνότητα, ώστε να διερευνηθούν οι επιδόσεις του αλγορίθμου και σε περιπτώσεις κυκλωμάτων υψηλής πυκνότητας. Για την εκτέλεση του αλγορίθμου 0.9 και 0.8 ήταν τα επίπεδα πυκνότητας που χρησιμοποιήθηκαν.[14]

Παράλληλα με τον FBM, εκτελέστηκε και ο αλγόριθμος Jezz πάνω στα 24 κυκλώματα που επιλέχθηκαν. Πρόκειται για έναν άλλον αλγόριθμο για το στάδιο του legalization. Οι συγκρίσεις των αποτελεσμάτων μεταξύ των δυο αυτών αλγορίθμων είχαν ως στόχο την καλύτερη κατανόηση των δυνατών σημείων και των αδυναμιών του αλγορίθμου FBM, καθώς και την όσο το δυνατόν πιο έγκυρη εξαγωγή τελικών συμπερασμάτων για τον αλγόριθμο. Αξίζει να σημειωθεί ότι οι τομείς πάνω στους οποίους υπήρχε το ενδιαφέρον για την εξαγωγή συμπερασμάτων ήταν οι τρεις τομείς που αναφέρθηκαν και πιο πάνω: ο χρόνος εκτέλεσης, το πόσο ο κάθε ένας αλγόριθμος κατάφερε να μειώσει το συνολικό μήκος καλωδίου χρησιμοποιώντας τη μέθοδο του HPWL, καθώς και το πόσο κοντά το τελικό αποτέλεσμα και των δύο ήταν στο αρχικό global placement (Average Displacement).[14]

Στην εικόνα 2.7 φαίνεται ένας συγκεντρωτικός πίνακας με τα στοιχεία όλων των κυκλωμάτων που χρησιμοποιήθηκαν στη διαδικασία, καθώς και τα αποτελέσματα των αλγορίθμων πάνω στους τρεις παραπάνω τομείς.

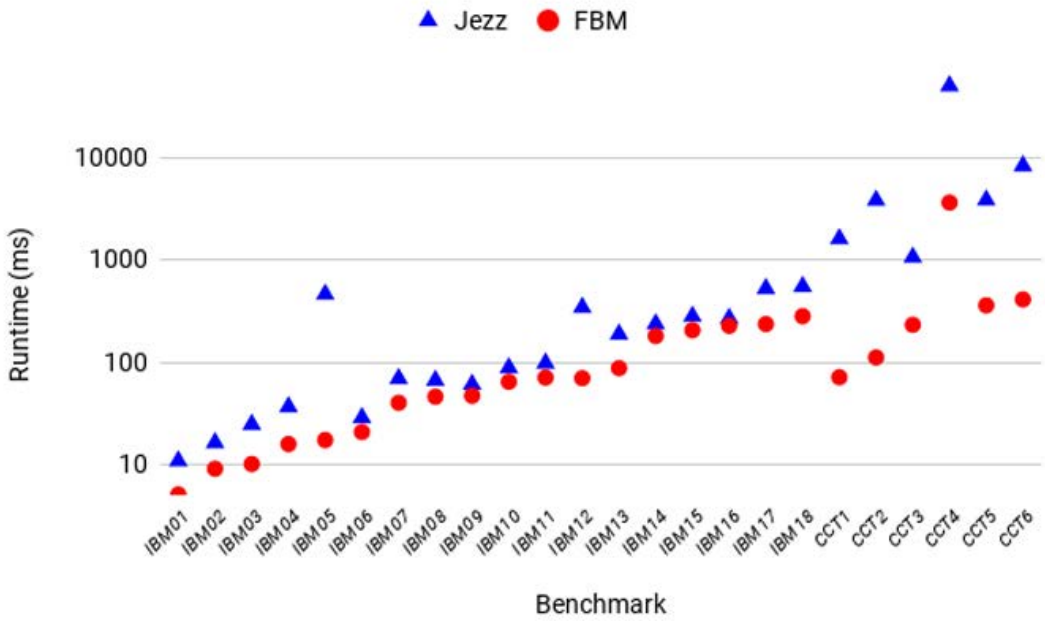
			Initial HPWL	FBM Legalizer			Jezz		
Circuits	Cells	Occupation	Wirelength (E+07)	Avg Displacement	Impact of HPWL	Runtime (ms)	Avg Displacement	Impact of HPWL	Runtime (ms)
IBM01	12506	43.65%	0.25	0.69	3.04%	5.19	0.41	1.58%	11.03
IBM02	19342	28.61%	0.53	0.67	1.77%	9.25	0.39	0.67%	16.62
IBM03	22853	33.72%	0.72	1.38	6.86%	10.30	0.42	0.82%	25.10
IBM04	27220	41.79%	0.77	1.36	7.54%	16.13	0.45	0.96%	37.27
IBM05	28146	80.02%	1.00	3.92	12.71%	17.66	2.53	26.47%	467.79
IBM06	32332	37.54%	0.71	0.68	3.33%	21.16	0.38	0.89%	29.18
IBM07	45639	47.07%	1.16	1.32	6.42%	40.77	4.75	1.06%	70.32
IBM08	51023	41.70%	1.14	1.03	9.75%	46.78	0.43	1.22%	67.55
IBM09	53110	42.31%	1.42	1.12	6.56%	47.80	0.42	1.01%	61.64
IBM10	68685	22.86%	2.65	1.22	6.25%	65.37	0.42	0.63%	89.54
IBM11	70152	44.72%	2.00	1.32	7.19%	71.91	0.45	-0.14%	99.38
IBM12	70439	30.96%	2.72	2.78	14.64%	70.95	0.65	2.90%	347.89
IBM13	83709	46.20%	2.29	2.62	17.58%	88.82	0.50	2.48%	190.89
IBM14	147088	61.33%	3.95	1.31	4.61%	183.05	0.51	1.73%	240.39
IBM15	161187	54.41%	5.20	2.04	10.60%	208.53	0.51	4.50%	284.77
IBM16	182980	43.91%	5.71	1.39	5.95%	229.76	0.49	1.47%	272.25
IBM17	184752	64.97%	7.25	2.47	7.73%	239.33	0.59	1.37%	531.78
IBM18	210341	72.31%	4.77	1.82	7.29%	285.90	0.56	2.48%	557.01
CCT1	51023	80.30%	0.98	3.90	27.47%	72.42	1.55	24.27%	1630.35
CCT2	51023	90.51%	0.96	6.16	46.42%	112.98	4.84	89.75%	3870.50
CCT3	147088	80.06%	3.71	2.81	12.11%	235.25	0.61	2.32%	1074.86
CCT4	147088	90.10%	3.36	12.43	47.33%	3675.15	13.91	180.13%	50736.70
CCT5	210341	80.20%	4.78	3.28	14.50%	364.52	0.94	9.74%	3891.41
CCT6	210341	90.04%	4.43	4.99	26.50%	416.49	1.25	17.55%	8406.31

Πηγή: J. Ferreira, P. F. Butzen, C. Meinhardt and R. A. L. Reis, "FBM: A Simple and Fast Algorithm for Placement Legalization," *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genoa, Italy, 2019, pp. 209-212, doi: 10.1109/ICECS46596.2019.8965013

Εικόνα 2.7 Στοιχεία κυκλωμάτων και αποτελέσματα της εφαρμογής των δύο αλγορίθμων πάνω στα κυκλώματα

Όσον αφορά τον τομέα του χρόνου εκτέλεσης, ο FBM υπερίσχυσε, καθώς παρατηρήθηκε ότι εκτελέστηκε σε 8 φορές περίπου μικρότερο χρονικό διάστημα από τον Jezz κατά μέσο όρο σε όλα τα κυκλώματα. Στην περίπτωση των κυκλωμάτων που δεν υπέστησαν κάποια τροποποίηση για την πυκνότητα ο FBM τελείωσε κατά μέσο όρο 3.27 φορές νωρίτερα από τον Jezz, ενώ στην περίπτωση των τροποποιημένων, ο FBM έφτασε στο επίπεδο να παράγει αποτελέσματα κατά μέσο όρο 18 φορές νωρίτερα από τον Jezz. Ένα από τα δυνατά σημεία του FBM που έγινε ξεκάθαρο στο τρέχον βήμα ήταν ότι είναι ιδιαίτερα αποδοτικός σε κυκλώματα με υψηλότερη πυκνότητα, καθώς σε τέτοιες περιπτώσεις τα αποτελέσματα εξάγονταν πολύ γρήγορα.[14]

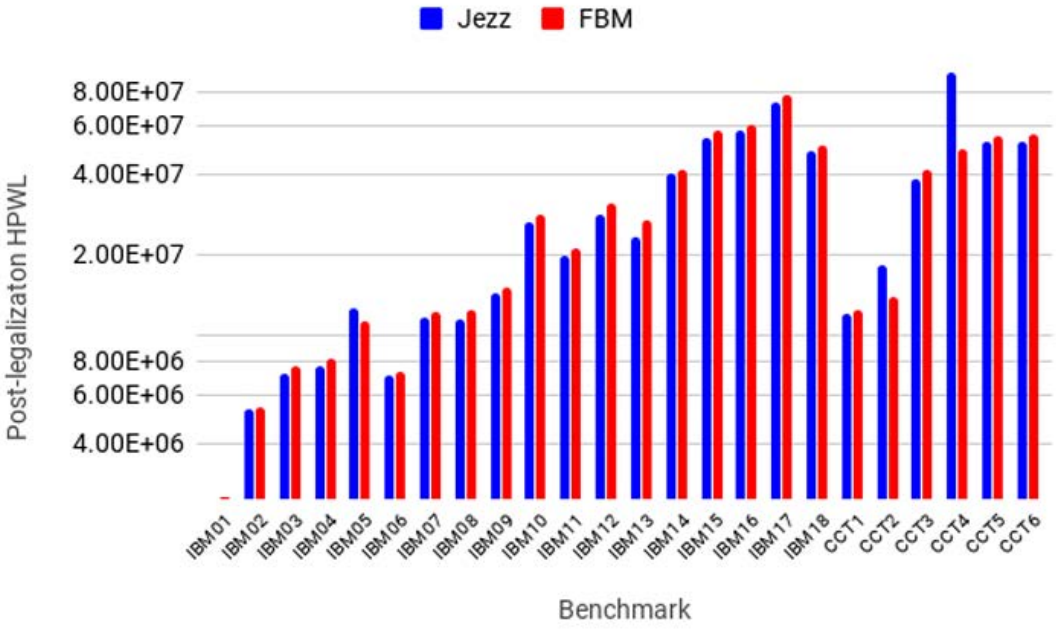
Στην εικόνα 2.8 φαίνεται ένα διάγραμμα της επίδοσης και των δύο αλγορίθμων ως προς το runtime.



Πηγή: J. Ferreira, P. F. Butzen, C. Meinhardt and R. A. L. Reis, "FBM: A Simple and Fast Algorithm for Placement Legalization," *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genoa, Italy, 2019, pp. 209-212, doi: 10.1109/ICECS46596.2019.8965013

Εικόνα 2.8 Επίδοση των αλγορίθμων FBM και Jezz ως προς το runtime.

Σχετικά με τον τομέα της μείωσης του συνολικού μήκους καλωδίου, με το πέρας της εκτέλεσης του FBM το καλώδιο είχε μειωθεί σε πολύ μεγαλύτερο βαθμό χρησιμοποιώντας τη μέθοδο HPWL για τα κυκλώματα που είχαν την υψηλότερη πυκνότητα από ότι μειώθηκε με τον Jezz. Ο Jezz από την άλλη υπερίσχυσε στις περιπτώσεις των κυκλωμάτων που δεν υπέστησαν τροποποίηση. Πριν την εφαρμογή του FBM το αρχικό μήκος του καλωδίου ήταν κατά περίπου 12% μεγαλύτερο από ότι μετά την εφαρμογή του.[14] Στην εικόνα 2.9 αναπαρίσταται η επίδοση των δύο αλγορίθμων ως προς το τελικό μήκος καλωδίου.

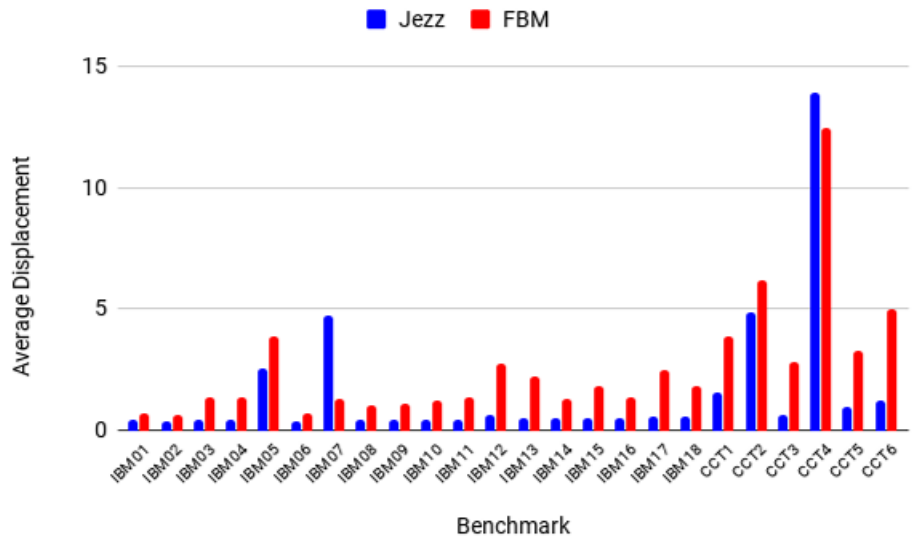


Πηγή: J. Ferreira, P. F. Butzen, C. Meinhardt and R. A. L. Reis, "FBM: A Simple and Fast Algorithm for Placement Legalization," *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genoa, Italy, 2019, pp. 209-212, doi: 10.1109/ICECS46596.2019.8965013

Εικόνα 2.9 Επίδοση των αλγορίθμων FBM και Jezz ως προς το τελικό μήκος καλωδίου .

Τέλος, ως προς τον τομέα του Average Displacement, ο FBM παρουσιάζει αδυναμία, κάτι που ίσως αποδίδεται στο γεγονός ότι ο αλγόριθμος είχε εκ φύσεως σχεδιαστεί για να είναι αρκετά γρήγορος ως προς την εκτέλεσή του. Η διαφορά των θέσεων των cells μεταξύ της εξόδου του global placement και της εξόδου του legalization για τον FBM ήταν κατά μέσο όρο περίπου 2.8 φορές μεγαλύτερη από ότι για τον Jezz για τα κυκλώματα με την υψηλότερη πυκνότητα. Στα κυκλώματα που δεν τροποποιήθηκαν ο FBM είχε πάλι υψηλότερη τιμή στον τομέα αυτό, και πιο συγκεκριμένα κατά περίπου 2.75 φορές κατά μέσο όρο σε σχέση με τον Jezz.[14]

Στην εικόνα 2.10 φαίνεται ένα γράφημα της επίδοσης των δυο αλγορίθμων ως προς το Average Displacement.



Πηγή: J. Ferreira, P. F. Butzen, C. Meinhardt and R. A. L. Reis, "FBM: A Simple and Fast Algorithm for Placement Legalization," *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genoa, Italy, 2019, pp. 209-212, doi: 10.1109/ICECS46596.2019.8965013

Εικόνα 2.10 Επίδοση των αλγορίθμων FBM και Jezz ως προς το Average Displacement.

ΚΕΦΑΛΑΙΟ 3. Τροποποιήσεις και επεκτάσεις

3.1 Εισαγωγή

Μετά από προσεκτική μελέτη και κατανόηση του αλγορίθμου FBM και της λειτουργίας του, στην παρούσα πτυχιακή υλοποιήθηκαν ορισμένες τροποποιήσεις που κύριο σκοπό είχαν την εφαρμογή ορισμένων βελτιστοποιήσεων πάνω στον ήδη υπάρχοντα αλγόριθμο, καθώς και ορισμένες προσθήκες, με στόχο την ανανέωσή του με νέους τομείς λειτουργικότητας. Στο παρών κεφάλαιο, αναλύονται λεπτομερώς και διεξοδικά όλα αυτά τα επιχειρήματα, καθώς και το κίνητρο πίσω από το καθένα.

3.2 Οι τροποποιήσεις

3.2.α Αλγόριθμοι ταξινόμησης

Όπως ήδη έχει γίνει σαφές από το προηγούμενο κεφάλαιο και πιο συγκεκριμένα από τη διαδικασία του αλγορίθμου FBM, πραγματοποιείται ταξινόμηση όλων των διαθέσιμων cells του κυκλώματος συνολικά 2 φορές ακριβώς, μία στο 2ο βήμα και μία στο 4ο βήμα του αλγορίθμου. Κάθε μία, όπως είδαμε, υλοποιεί την ταξινόμηση με διαφορετικά κριτήρια. Στο βήμα 2 ταξινομούνται με βάση το width τους από το μικρότερο στο μεγαλύτερο, ενώ στο βήμα 4 με βάση την αριθμητική τιμή της χ θέσης τους από την μικρότερη στη μεγαλύτερη. Και στις 2 περιπτώσεις υπάρχει κατάλληλη μέριμνα για την περίπτωση που δύο cells έχουν την ίδια τιμή για το χαρακτηριστικό ως προς το οποίο πραγματοποιείται η ταξινόμηση και κατά συνέπεια η σύγκριση. Όταν συμβεί αυτό, και τα 2 βήματα χρησιμοποιούν ως τεχνική την ταξινόμηση των κελιών αυτών και ως προς ένα άλλο κριτήριο, όπου στις περιπτώσεις και των δύο βημάτων τυχαίνει να είναι η αριθμητική τιμή της θέσης y τους.[14] Από τη στιγμή λοιπόν που η διαδικασία της ταξινόμησης των κελιών στο χώρο εντός των ορίων του κυκλώματος χρησιμοποιείται δύο φορές στον αλγόριθμο, εύκολα βγαίνει το συμπέρασμα ότι η ταξινόμηση είναι από τις πιο σημαντικές λειτουργίες του αλγορίθμου και άρα ένας από τους πιο βασικούς παράγοντες που θα καθορίσουν με σαφήνεια το συνολικό χρόνο εκτέλεσής του(runtime). Η επιλογή ενός αποδοτικού για τα συγκεκριμένα βήματα αλγορίθμου ταξινόμησης ήταν ένα ζήτημα ιδιαίτερης σημασίας και έπρεπε η επιλογή του να γίνει προσεκτικά.

Κατά την προσπάθεια υλοποίησης του αλγορίθμου μελετήθηκαν και δοκιμάστηκαν προσεκτικά τρεις διαφορετικοί αλγόριθμοι ταξινόμησης: ο Bubble sort, ο Insertion sort καθώς και ο Timsort, με τον τελευταίο να είναι αυτός που χρησιμοποιείται σε πραγματικά σενάρια και υπολογιστικά προβλήματα, καθώς έχει φτιαχτεί με αυτή την προοπτική, οπότε και είναι όπως θα δούμε και παρακάτω ο πιο αποδοτικός αλγόριθμος για τέτοιου είδους περιπτώσεις.[28] Ο Insertion sort είναι η βέλτιστη επιλογή για υλοποιήσεις πιο μικρού μεγέθους με χαρακτηριστικά που θα εξηγηθούν παρακάτω, ενώ όταν πρόκειται για ρεαλιστικές υλοποιήσεις ή δεν θα χρησιμοποιηθεί καθόλου εάν για τη συγκεκριμένη περίπτωση υπάρχει πιο αποδοτική λύση ή εναλλακτικά θα χρησιμοποιηθεί συνδυαστικά με κάποιον άλλον αλγόριθμο ταξινόμησης. Τέλος, ο Bubble sort αξιοποιείται κυρίως για διδασκαλία-εκμάθηση και για ακόμα πιο απλές και μικρές υλοποιήσεις από αυτές για τις οποίες χρησιμοποιείται ο Insertion sort, ενώ σε ρεαλιστικές υλοποιήσεις δεν χρησιμοποιείται ποτέ.

3.2.α.1 Bubble sort

Ο αλγόριθμος ταξινόμησης Bubble sort πρόκειται για έναν από τους πιο απλούς τόσο ως προς την κατανόηση όσο και ως προς την υλοποίηση αλγορίθμους για ταξινόμηση. Τα βήματα του αλγορίθμου έχουν ως εξής:

Αρχικά υποθέτουμε ότι έχουμε μια δομή με στοιχεία τα οποία θέλουμε να ταξινομήσουμε σε αύξουσα σειρά. Ξεκινάμε να ταξινομούμε τα στοιχεία από αριστερά προς τα δεξιά συγκρίνοντας τα ανά δύο και σε περίπτωση που διαπιστωθεί ότι το πρώτο είναι μεγαλύτερο από το δεύτερο τότε πραγματοποιούμε αντιμετάθεση των στοιχείων, αλλιώς τα αφήνουμε όπως ήταν εξ' αρχής. Η διαδικασία συνεχίζεται κατά τον ίδιο τρόπο μέχρι να φτάσουμε στο τέλος της δομής. Ακολουθώντας αυτή τη μεθοδολογία έχουμε πλέον τοποθετήσει αυτόματα το μεγαλύτερο στοιχείο της δομής στο τέλος της. Επαναλαμβάνουμε τη μεθοδολογία αυτή σταματώντας πάντα τη διαδικασία των συγκρίσεων πριν το στοιχείο που ταξινομήθηκε επιτυχώς από την προηγούμενη επανάληψη. Συνεχίζουμε μέχρι να μην υπάρχουν άλλα στοιχεία προς ταξινόμηση. Σε περίπτωση που έπρεπε να πραγματοποιήσουμε ταξινόμηση σε φθίνουσα σειρά θα συγκρίναμε ξανά κάθε στοιχείο με το αμέσως επόμενο, απλά θα αντιστρεφόταν η λογική των αντιμεταθέσεων, δηλαδή σε περίπτωση που το πρώτο στοιχείο από το ζεύγος σύγκρισης ήταν μεγαλύτερο από το δεύτερο, δεν θα γινόταν καμιά κίνηση και ο αλγόριθμος θα προχώραγε κανονικά, ενώ σε περίπτωση που το δεύτερο ήταν μεγαλύτερο από το πρώτο τότε θα γινόταν αντιμετάθεση. Κατά αυτό τον τρόπο στο τελικό αποτέλεσμα θα είχαμε τις μεγαλύτερες τιμές στην αρχή της δομής και όσο προχωράγαμε προς το τέλος της οι τιμές θα μίκραιναν.

Μια απλή υλοποίηση του αλγορίθμου σε γλώσσα Python φαίνεται στην εικόνα 3.1.

```python
def bubble_sort(cells, key_func):

    n = len(cells)
    for i in range(n):
        for j in range(0, n-i-1):
            if key_func(cells[j]) > key_func(cells[j+1]):

                cells[j], cells[j+1] = cells[j+1], cells[j]

    return cells
```

Εικόνα 3.1 Ο αλγόριθμος Bubble sort

Με την εξαγωγή των αποτελεσμάτων του αλγορίθμου βγαίνει το συμπέρασμα ότι ο αλγόριθμος αυτός είναι stable, δηλαδή σε περίπτωση που τύχει δύο από τα στοιχεία που ταξινομούνται να είναι ίσα, τότε η τελική σειρά που θα έχουν με το πέρας του αλγορίθμου θα είναι η ίδια με την αρχική σειρά που είχαν προτού εφαρμοστεί ο αλγόριθμος. Στην καλύτερη περίπτωση ο αλγόριθμος έχει πολυπλοκότητα O(n), όταν δηλαδή η δομή είναι ήδη ταξινομημένη κατά την

επιθυμητή διάταξη. Στη μέση περίπτωση ο Bubble sort έχει πολυπλοκότητα $O(n^2)$, όταν δηλαδή τα στοιχεία δεν είναι καθόλου ταξινομημένα. Στην χειρότερη περίπτωση ο αλγόριθμος έχει πολυπλοκότητα $O(n^2)$. Πρόκειται για την περίπτωση που θα τύχει τα στοιχεία να είναι τοποθετημένα σε αντίθετη διάταξη από αυτή που πρέπει να τοποθετηθούν με βάση τον Bubble sort.[29] Αυτό δείχνει ότι ο αλγόριθμος δεν είναι καθόλου αποδοτικός και δεν πρέπει να επιλέγεται για περιπτώσεις προβλημάτων που έχουν μεγάλα inputs. Αυτός είναι και ο λόγος που δεν αποτελεί καν επιλογή σε περιπτώσεις πιο πολύπλοκων προβλημάτων-προγραμμάτων που απαιτείται η χρήση κάποιου αλγορίθμου ταξινόμησης. Ο λόγος που χρησιμοποιήθηκε στην παρούσα υλοποίηση ήταν κυρίως για την απόκτηση μιας πιο ολοκληρωμένης εικόνας σχετικά με την αποδοτικότητα της χρήσης των άλλων δυο αλγορίθμων ταξινόμησης που θα αναλυθούν εις βάθος στην συνέχεια. Το σκεπτικό ήταν ότι από τη στιγμή που θα υπήρχε η λιγότερο αποδοτική μέθοδος ταξινόμησης και άλλες δύο αρκετά πιο αποδοτικές και συχνότερα χρησιμοποιούμενες μέθοδοι, θα υπήρχε η δυνατότητα να εξαχθούν ακόμα καλύτερα συμπεράσματα ως προς την εφαρμογή των άλλων δύο αλγορίθμων για την περίπτωση του αλγορίθμου FBM αλλά και των δεδομένων εισόδου που έχουμε στη διάθεσή μας.

3.2.α.2 Insertion sort

Πρόκειται για έναν αλγόριθμο με αρκετά απλή υλοποίηση, η οποία μπορεί να γίνει εύκολα κατανοητή, καθώς το ίδιο βήμα ακριβώς λαμβάνει χώρα σε κάθε μία επανάληψη, θυμίζοντας τον Bubble sort, στον οποίο συνέβαινε το ίδιο: σε κάθε μία επανάληψη εκτελούσε σύγκριση των στοιχείων ανά ζευγάρια. Ένα ακόμα κοινό χαρακτηριστικό που μπορούμε να πούμε πως έχει με τον Bubble sort είναι ότι και οι δύο δεν προκαλούν αλλαγές στη σειρά των στοιχείων που κατά τη σύγκριση προκύπτουν ίσα πριν και μετά την εξαγωγή των αποτελεσμάτων. Η όλη διαδικασία που ακολουθείται με λίγα λόγια είναι η εξής: σε κάθε επανάληψη, κάθε ένα από τα στοιχεία τοποθετείται στην τελική ταξινομημένη του θέση μετά τις απαραίτητες συγκρίσεις.

Η διαδικασία ξεκινά μη λαμβάνοντας υπόψιν το πρώτο στοιχείο, δηλαδή ταξινομούνται όλα από το δεύτερο και έπειτα. Αρχικά, τοποθετούμε το δεύτερο στοιχείο στη θέση του πρώτου και μετακινούμε το πρώτο στη θέση του δεύτερου αν από τη σύγκρισή τους προέκυψε ότι το πρώτο είναι μεγαλύτερο(δεδομένου ότι θέλουμε το τελικό αποτέλεσμα να είναι ταξινόμηση κατά αύξουσα σειρά). Αφού τακτοποιηθεί και το δεύτερο στοιχείο, για κάθε επόμενο γίνεται η εξής διαδικασία: το στοιχείο τοποθετείται πίσω από το τελευταίο στοιχείο που είναι μεγαλύτερο από αυτό, ελέγχοντας μόνο την αριστερή πλευρά της δομής, η οποία είναι και η ταξινομημένη, με τη δεξιά να είναι η μη ταξινομημένη. Η διαδικασία σταματά όταν δεν υπάρχει πια άλλο στοιχείο προς ταξινόμηση.

Αξίζει να σημειωθεί ότι η συγκεκριμένη μέθοδος μοιάζει ιδιαίτερα με μία άλλη μέθοδο ταξινόμησης, την selection sort. Η λειτουργικότητά τους είναι αρκετά παρόμοια, καθώς και οι δύο υλοποιούν μετακινήσεις μετά από συγκρίσεις. Η κύρια διαφορά τους έγκειται στο τμήμα της δομής όπου η καθεμία μέθοδος υλοποιεί τις συγκρίσεις. Η Insertion sort όπως είδαμε, υλοποιεί τις συγκρίσεις στην αριστερή

πλευρά της δομής, σε αντίθεση με την selection sort που εξετάζει την δεξιά πλευρά.

Στην εικόνα 3.2 φαίνεται η υλοποίηση του αλγορίθμου σε γλώσσα python:

```
def insertion_sort(arr, key=lambda x: x):

    for i in range(1, len(arr)):
        current = arr[i]
        j = i - 1
        while j >= 0 and key(arr[j]) > key(current):
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = current
    return arr
```

Εικόνα 3.2 Ο αλγόριθμος Insertion sort

Στην χειρότερη περίπτωσή του, ο Insertion sort έχει πολυπλοκότητα $O(n^2)$, στην καλύτερη $O(n)$ και στη μέση $O(n^2)$.[30] Κάθε μιας από αυτές τις περιπτώσεις είναι ακριβώς η ίδια με αυτές του αλγορίθμου Bubble sort. Το δυνατό του σημείο είναι όταν ως είσοδο δέχεται δομές που δεν περιέχουν πολλά στοιχεία ή που τα στοιχεία τους είναι πολύ κοντά στην πλήρη ταξινόμηση. Σε περιπτώσεις που ως είσοδο έχουμε μια δομή με πολλά στοιχεία, ο αλγόριθμος θα επιβραδύνει σημαντικά τη διαδικασία και θα αυξήσει ταυτόχρονα το runtime. Για μεγάλα inputs δεν είναι η βέλτιστη επιλογή. Ένα από τα πλεονεκτήματα του αλγορίθμου είναι ότι δεν απαιτεί επιπλέον μνήμη καθώς δεν αποθηκεύει τα στοιχεία σε κάποια επιπλέον δομή δεδομένων κατά την υλοποίηση του, αλλά τα ταξινομεί κατευθείαν. Τέλος αξίζει να σημειωθεί ότι ο Insertion sort χρησιμοποιείται για τη δημιουργία υβριδικών αλγορίθμων ταξινόμησης, όπως παραδείγματος χάριν ο Introsort και ο Timsort. Ο τελευταίος θα εξεταστεί στη συνέχεια.

Ο Insertion sort χρησιμοποιήθηκε με το σκεπτικό ότι θα είναι γενικά αποδοτικός όσον αφορά τα benchmark designs πάνω στα οποία υλοποιήθηκε ο αλγόριθμος, καθώς το input data σε όλα τα κυκλώματα δεν ήταν ιδιαίτερα μεγάλο, οπότε θεωρήθηκε πιθανό να είναι αυτός ο βέλτιστος αλγόριθμος για την υλοποίηση των βημάτων δύο και τέσσερα του αλγορίθμου FBM.

3.2.α.3 Timsort

Πρόκειται για έναν υβριδικό αλγόριθμο, ο οποίος χρησιμοποιεί τους αλγορίθμους Merge sort και Insertion sort. Ο κύριος σκοπός της δημιουργίας του ήταν να μπορεί να είναι αποδοτικός κατά τα μέγιστα σε προβλήματα και εφαρμογές πλήρως ρεαλιστικά τα οποία χρησιμοποιούν στοιχεία και δεδομένα που ανταποκρίνονται πλήρως σε πραγματικές συνθήκες. Ο συγκεκριμένος αλγόριθμος πετυχαίνει αποδοτικότητα και ταχύτητα με το να συνδυάζει τον αλγόριθμο Insertion sort και την λειτουργία συγχώνευσης του Merge sort τις κατάλληλες στιγμές της διαδικασίας ταξινόμησης που το κάθε ένα είναι πιο αποδοτικό.[28] Αναλυτικότερα, όσον αφορά την Insertion sort, όπως είδαμε και παραπάνω, το δυνατό της σημείο ήταν η ταξινόμηση δεδομένων μικρού συνολικού μεγέθους, καθώς και η ταξινόμηση δεδομένων τα οποία ήταν ήδη ταξινομημένα ως ένα συγκεκριμένο σημείο. Από τη άλλη, η Merge sort είναι ιδιαίτερα αποδοτική στην ταξινόμηση όταν σαν είσοδο έχουμε ένα μεγάλο πλήθος στοιχείων. Βλέπουμε λοιπόν ότι ένας υβριδικός αλγόριθμος μπορεί να χειριστεί με επιτυχία πληθώρα περιπτώσεων ταξινόμησης, κάνοντας έξυπνες και γρήγορες εναλλαγές μεταξύ των επιμέρους αλγορίθμων ταξινόμησης που αξιοποιεί, ανάλογα με το ποιος είναι ο ικανότερος να διαχειριστεί κάθε επιμέρους φάση. Ο Timsort λοιπόν, ανάλογα με το στάδιο της ταξινόμησης και τις ανάγκες χρησιμοποιεί είτε τον έναν είτε τον άλλον αλγόριθμο. Ο Insertion sort χρησιμοποιείται περισσότερο στα αρχικά στάδια του Timsort, ενώ ο Merge sort πιο μετά. [28]

Ο αλγόριθμος Timsort έχει τη δυνατότητα να εκμεταλλεύεται κατάλληλα ένα φαινόμενο που παρουσιάζεται σε όλους τους πίνακες αλλά και γενικά σε όλες τις δομές δεδομένων όταν πρόκειται για ένα πρόβλημα ταξινόμησης. Πρόκειται για την κατάσταση όπου από τα αρχικά στοιχεία που μας δίνονται και τα οποία πρέπει να ταξινομήσουμε υπάρχουν ορισμένα που είναι ήδη ταξινομημένα, που ήδη δηλαδή κατά τύχη βρίσκονται στη σωστή σειρά. Αυτό είναι ένα πολύ σύνηθες φαινόμενο και είναι πολύ δύσκολο μια τυχαία σειρά στοιχείων να μην περιλαμβάνει τουλάχιστον μία τέτοια ακολουθία.[31] Αυτό μπορεί πολύ εύκολα να διαπιστωθεί και στην πράξη πχ σε περίπτωση που κάποιος κάτσει και σκεφτεί και γράψει 10 τυχαίους αριθμούς σε σειρά. Είτε θελήσει να τους ταξινομήσει κατά αύξουσα σειρά είτε κατά φθίνουσα σχεδόν πάντα κάποια ψηφία θα είναι ήδη ταξινομημένα εξ' αρχής. Η σωστή διαχείριση αυτών των ακολουθιών μπορεί να μειώσει σε πολύ μεγάλο βαθμό το χρόνο εκτέλεσης ενός αλγορίθμου, γιατί μειώνονται κατά πολύ οι συνολικές ενέργειες που αυτός πρέπει να επιτελέσει ώστε να λύσει το πρόβλημα της ταξινόμησης, ειδικά σε σχέση με αλγορίθμους που ελέγχουν προσεκτικά κάθε ένα στοιχείο προς ταξινόμηση, πραγματοποιώντας κάθε δυνατή σύγκριση και αντιμετάθεση, όπως για παράδειγμα ο Bubble sort που προαναφέραμε. Τέτοιες τυχαία ταξινομημένες ακολουθίες γλυτώνουν χρόνο τόσο από το κομμάτι της σύγκρισης, όσο και από το κομμάτι της αντιμετάθεσης των στοιχείων. Πολλές φορές μάλιστα αν τύχει τέτοιου είδους ακολουθίες και είναι είτε συχνές είτε μεγάλες σε μήκος(είτε και τα δύο) συμβαίνει το φαινόμενο τα στοιχεία της δομής δεδομένων να είναι ήδη πολύ κοντά στην πλήρη ταξινόμηση. Η δυνατότητα σωστής διαχείρισης αυτών των ακολουθιών είναι και ένας από τους λόγους που ο αλγόριθμος Timsort θεωρείται ως ο πιο γρήγορος αλγόριθμος ταξινόμησης που υπάρχει μέχρι στιγμής.[32]

Στον αλγόριθμο Timsort πρώτο βήμα είναι ο καθορισμός του minrun. Πρόκειται για μία μεταβλητή η οποία εκφράζει ποιο θα είναι το ελάχιστο επιτρεπτό μέγεθος ενός run. Στον αλγόριθμο ο όρος run δεν είναι τίποτα άλλο από μια ήδη ταξινομημένη ακολουθία. Ορίζουμε λοιπόν με την τιμή minrun ποιο θα είναι το ελάχιστο μέγεθος που μια τέτοια ακολουθία δύναται να έχει. Εξαίρεση αποτελεί το τελευταίο run που θα σχηματιστεί στα διαθέσιμα δεδομένα, το οποίο σε περίπτωση που το μέγεθος του είναι μικρότερο από το minrun δεν το θεωρούμε ως λάθος. Συνήθως το minrun παίρνει ως αρχική τιμή του την τιμή 32, και όπως θα δούμε και στη συνέχεια, η τιμή κάθε φορά διπλασιάζεται(64,128 κλπ.).[28]

Αφού λοιπόν οριστεί στην αρχή η τιμή του minrun, ο αλγόριθμος ξεκινάει προσπελαύνοντας ένα-ένα τα αρχικά στοιχεία που μας δίνονται, προσπαθώντας να βρει τις προαναφερθείσες ακολουθίες. Σε περίπτωση που βρει κάποια ακολουθία με μήκος ίσο η παραπάνω από το ορισμένο minrun την αποθηκεύει και προχωράει προς εύρεση των επόμενων. Εάν βρει κάποια ακολουθία με μικρότερο αριθμό στοιχείων από το minrun που δίνεται, τότε ο αλγόριθμος χτίζει μια ακολουθία. Ο τρόπος που το κάνει αυτό είναι ο εξής: κάθε επόμενο στοιχείο που θα βρει μπροστά του το προσθέτει στην ήδη υπάρχουσα ακολουθία. Το νέο στοιχείο δεν προστίθεται απλώς στα ήδη υπάρχοντα στοιχεία και τοποθετείται σε μια τυχαία θέση στην ακολουθία, αλλά πραγματοποιείται ταξινόμηση του στοιχείου στην κατάλληλη θέση. Για το σκοπό αυτό, χρησιμοποιείται η μία από τις δύο συναρτήσεις ταξινόμησης που ο αλγόριθμος αξιοποιεί, και πιο συγκεκριμένα η Insertion sort.[28]

Η συγκεκριμένη μέθοδος ταξινόμησης ήταν από τις πιο αποδοτικές (αν όχι η πιο αποδοτική) που θα μπορούσε να χρησιμοποιηθεί για τη συγκεκριμένη περίπτωση, καθώς η προσθήκη νέων στοιχείων στα runs σημαίνει προσθήκη νέων στοιχείων σε ήδη ταξινομημένες ακολουθίες, καθώς τα runs ταξινομούνται πρώτα προτού εισαχθεί σε αυτά κάποιο νέο στοιχείο, αλλά είναι και σχετικά μικρά σε μέγεθος στις περισσότερες περιπτώσεις. Με την Insertion sort λοιπόν πετυχαίνουμε πολύ γρήγορη ταξινόμηση και κατά συνέπεια δημιουργία των runs, καθώς εδώ ισχύουν και οι δύο συνθήκες που αποτελούν τα δυνατά της σημεία. Εδώ πρέπει να τονιστεί ότι ανάλογα με το είδος της ταξινόμησης που απαιτείται να γίνει, γίνονται και οι αντίστοιχες αλλαγές στις ακολουθίες είτε προτού προστεθούν νέα στοιχεία είτε εάν δεν προστεθούν νέα στοιχεία. Για παράδειγμα, στις περιπτώσεις που το μέγεθος της ακολουθίας είναι ίσο με το μέγεθος του minrun, οπότε δεν εισάγονται νέα στοιχεία, ο αλγόριθμος Timsort ελέγχει για τον αν η σειρά ταξινόμησης είναι σωστή. Εάν κάτι τέτοιο δεν ισχύει, τότε πρώτα ταξινομεί την ακολουθία σε αύξουσα η φθίνουσα σειρά και μετά προχωράει στην εύρεση καινούργιων run. Η ίδια λογική ακολουθείται και για τις περιπτώσεις που πρέπει υποχρεωτικά να προστεθούν νέα στοιχεία. Και εκεί, ο Timsort πρώτα επιβεβαιώνει ότι η ακολουθία βρίσκεται στη σωστή σειρά, εάν όχι πράττει αναλόγως και έπειτα προχωράει στην οποιαδήποτε προσθήκη νέων στοιχείων, αν αυτά υπάρχουν.[28]

Τη στιγμή που η διαδικασία προχωρήσει ως ένα σημείο και φτάσουμε να έχουμε δύο ολοκληρωμένα runs ίδιου μεγέθους, τότε πια φτάνουμε στο βήμα της συγχώνευσης των runs. Ο Timsort λοιπόν σε αυτή την περίπτωση αξιοποιεί τον δεύτερο αλγόριθμο ταξινόμησης, τον Merge sort, και πιο συγκεκριμένα μόνο τη λειτουργία συγχώνευσης του. Αναλυτικότερα, η λειτουργία συγχώνευσης του Merge sort είναι η εξής: έχοντας δύο πίνακες στοιχείων ταξινομημένους με τον

ίδιο τρόπο(είτε και οι δύο να είναι σε αύξουσα σειρά είτε σε φθίνουσα) ξεκινάμε να συγκρίνουμε μεταξύ τους τα πρώτα στοιχεία των πινάκων(από τα αριστερά προς τα δεξιά). Σε περίπτωση αύξουσας διάταξης, διαλέγουμε το μικρότερο από τα δύο και το τοποθετούμε σε μια έξτρα δομή δεδομένων που έχει αρχικοποιηθεί για αυτόν ακριβώς τον σκοπό (σε περίπτωση φθίνουσας θα παίρναμε το μεγαλύτερο από τα δύο). Η ίδια διαδικασία των συγκρίσεων και των τοποθετήσεων των στοιχείων στη νέα δομή δεδομένων συνεχίζει μέχρις ότου και οι δύο υποπίνακες να αδειάσουν τελείως. Το τελικό αποτέλεσμα της λειτουργίας συγχώνευσης του Merge sort είναι ένας πλήρως ταξινομημένος πίνακας. Αυτή η λειτουργικότητα εφαρμόζεται τώρα και στα ταξινομημένα runs. Η προϋπόθεση των ταξινομημένων δομών πληρείται, καθώς τα runs όπως είδαμε είναι ήδη ταξινομημένα. Το τελικό αποτέλεσμα είναι ένα πλήρως ταξινομημένο run με προφανώς το διπλάσιο μέγεθος από τα αρχικά αφού συγχωνεύτηκαν.[28][33]

Μετά το βήμα της πρώτης συγχώνευσης, ο αλγόριθμος συνεχίζει επαναλαμβάνοντας το πρώτο αρχικό βήμα της εύρεσης νέων runs, δηλαδή δίνοντας έμφαση στην αρχική τιμή minrun ξαναδημιουργεί runs και τα ξανασυγχωνεύει και αυτά δημιουργώντας ένα μεγαλύτερο run με ίδιο μέγεθος και ίδια διάταξη με το μεγαλύτερο προηγούμενο. Ξανασυγχωνεύει τα δύο μεγάλα runs και ακριβώς η ίδια επαναληπτική διαδικασία συνεχίζεται μέχρις ότου να έχουν όλα τα στοιχεία ταξινομηθεί.[28]

Ο Timsort παρουσιάζει πολυπλοκότητα O(nlogn) στη χειρότερη περίπτωση. Η χειρότερη περίπτωση για τον Timsort είναι όταν τα στοιχεία της δομής είναι ταξινομημένα κατά αντίθετη διάταξη από αυτή που ο αλγόριθμος καλείται να υλοποιήσει. Στην καλύτερη περίπτωση ο αλγόριθμος έχει πολυπλοκότητα O(n). Πρόκειται για την περίπτωση όπου η δομή με τα στοιχεία τυχαίνει να είναι ήδη ταξινομημένη κατά την επιθυμητή διάταξη, οπότε ο αλγόριθμος γλυτώνει αρκετό χρόνο μη έχοντας αυτό το βήμα και γίνεται αποδοτικότερος κατά συνέπεια. Στο average case ο Timsort έχει πολυπλοκότητα O(nlogn), όπως και στη χειρότερη. Είναι όταν τα στοιχεία είναι γενικά μη ταξινομημένα και υπάρχουν απλά μερικές ταξινομημένες ακολουθίες. Απαιτείται έξτρα μνήμη για την υλοποίηση του, καθώς όπως είπαμε κατά την συγχώνευση των runs απαιτείται μια έξτρα δομή δεδομένων, σε αντίθεση με τους δύο προηγούμενους αλγορίθμους που είδαμε(Bubble και Insertion). Ο Timsort είναι και αυτός stable στα αποτελέσματά του, ακριβώς όπως και οι άλλοι δύο.[34] Ο Timsort επιλέχθηκε ως ένας από τους αλγορίθμους που θα χρησιμοποιηθούν, καθώς δεδομένου του γεγονότος ότι θεωρείται ο πιο γρήγορος αλγόριθμος ταξινόμησης σε σχέση με όλους τους άλλους που υπάρχουν και γνωρίζοντας ότι έχει φτιαχτεί με την προοπτική να είναι αποδοτικός σε ρεαλιστικές υλοποιήσεις, θεωρήθηκε ότι υπάρχει σοβαρή πιθανότητα να αποτελεί την πιο αποδοτική και γρήγορη λύση για τη συγκεκριμένη περίπτωση της ταξινόμησης κελιών. Μάλιστα, επειδή χρησιμοποιεί και την Insertion sort πιο έξυπνα και μόνο όταν η χρήση της είναι συμφέρουσα, θεωρήθηκε πολύ πιθανό να υπερισχύει και της λύσης της χρήσης μόνο του αλγορίθμου Insertion sort.

Αυτοί οι τρεις αλγόριθμοι λοιπόν χρησιμοποιήθηκαν για την υλοποίηση του βήματος της ταξινόμησης του αλγορίθμου. Δημιουργήθηκαν συνολικά τρεις διαφορετικές υλοποιήσεις του αλγορίθμου, οι οποίες συγκρίθηκαν μεταξύ τους ως προς το χρόνο εκτέλεσης χρησιμοποιώντας τη συνάρτηση time() της Python. Για τον Bubble sort και τον Insertion sort δημιουργήθηκαν ξεχωριστές υποσυναρτήσεις στον κώδικα ώστε σε κάθε ανάγκη ταξινόμησης να μπορούν να

κληθούν από οποιαδήποτε σημείο του χρειαστεί. Ωστόσο, όσον αφορά τον αλγόριθμο Timsort, δεν υπήρξε ανάγκη υλοποίησης επιπλέον εξωτερικής συνάρτησης καθώς η μέθοδος sorted() της Python χρησιμοποιούσε από μόνη της τον Timsort, οπότε στην περίπτωσή του απλά γινόταν κλήση της συνάρτησης sorted(), κάτι ιδιαίτερα βολικό καθώς η υλοποίηση του Timsort από την αρχή θα ήταν μια πιο χρονοβόρα και περίπλοκη διαδικασία σε σχέση με την υλοποίηση των δύο άλλων αλγορίθμων. Οι αλγόριθμοι ταξινόμησης και κατά επέκταση ο αλγόριθμος FBM εκτελέστηκαν πάνω σε 4 διαφορετικά benchmark design. Τα 2 από αυτά είχαν μεγάλο αριθμό από cells, το τρίτο είχε λίγα ενώ το τελευταίο βρισκόταν ανάμεσα στα δύο πρώτα και στο τρίτο ως προς το μέγεθος του input data. Αξίζει να σημειωθεί ότι κατά τα πολύ αρχικά στάδια της υλοποίησης του αλγορίθμου FBM, έγιναν δοκιμές του πάνω σε πολύ μικρά κυκλώματα με ελάχιστα cells, ώστε να επιβεβαιωθεί η ορθότητα του σε μια πολύ πρώτη φάση και να μπορεί κατά συνέχεια να εφαρμοστεί και σε πιο μεγάλα benchmark designs, όπως αυτά που δόθηκαν.

3.2.β Σειριακή και παράλληλη εκτέλεση

Μια δεύτερη τροποποίηση που υλοποιήθηκε στον FBM ήταν η εφαρμογή παράλληλης εκτέλεσης αντί για σειριακή.

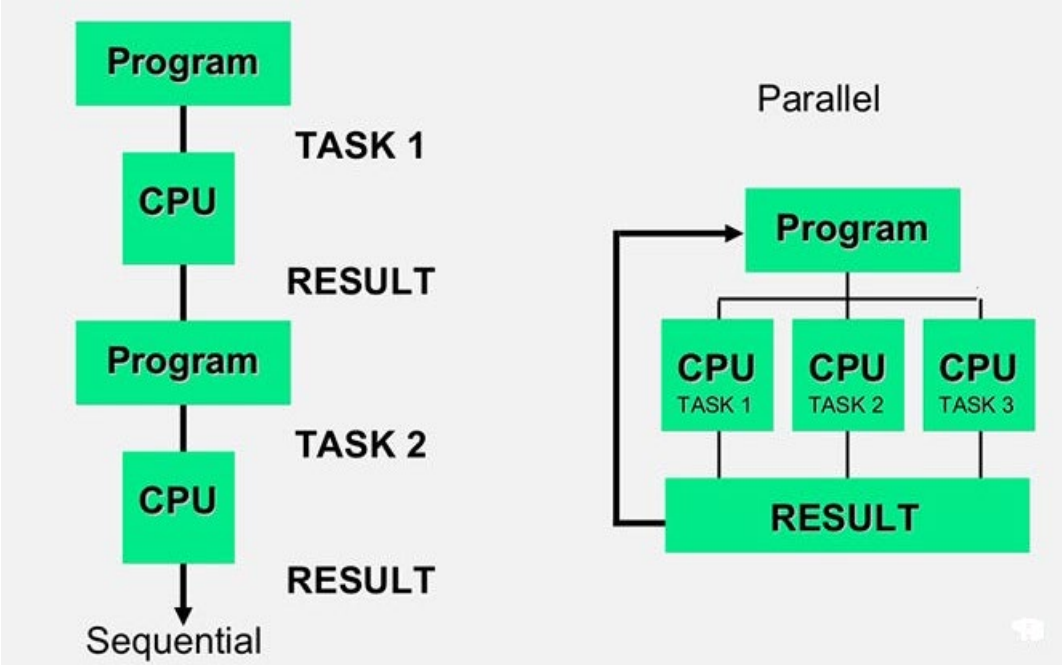
Η σειριακή εκτέλεση(sequential execution) είναι ο κλασσικός και πιο συνηθισμένος τρόπος εκτέλεσης εντολών σε ένα πρόγραμμα. Κάθε εντολή εκτελείται όταν έρθει η ώρα να εκτελεστεί, και με την ολοκλήρωσή της ξεκινάει η εκτέλεση της επόμενης εντολής, ακολουθώντας πιστά τη ροή των εντολών του προγράμματος. Μπορούμε να το σκεφτούμε και ως εξής: οι εντολές σε ένα πρόγραμμα βρίσκονται σε μια σειρά και κάθε μία αναμένει να έρθει πρώτα η σειρά της και μετά να εκτελεστεί, σεβόμενη την προτεραιότητα των εντολών που έχουν γραφεί νωρίτερα από αυτή. Είναι η ίδια κατάσταση με οποιαδήποτε περίπτωση που υπάρχει ουρά αναμονής στον πραγματικό κόσμο, πχ στο ταμείο του σούπερ μάρκετ. Σε περιπτώσεις που επιβάλλεται η χρήση αυτού του είδους της εκτέλεσης, το υπολογιστικό σύστημα αποτελείται από έναν επεξεργαστή με σχετικά χαμηλές ταχύτητες κα με συνεχή φόρτο επεξεργασίας, καθώς έτσι όπως συνεχώς εκτελείται η μία εντολή μετά την άλλη δύσκολα υπάρχει χρόνος αδράνειας για αυτόν.[35] Συνήθως για μικρά και πιο απλά tasks η σειριακή εκτέλεση είναι η βέλτιστη επιλογή, καθώς σε τέτοιες περιπτώσεις ο χρόνος εκτέλεσης είναι ήδη πολύ μειωμένος και η σειριακή εκτέλεση δεν προκαλεί κάποιο θέμα με το χρονισμό λόγω και του μικρού input data που τέτοιες υλοποιήσεις συνήθως έχουν. Όταν από την άλλη όμως ερχόμαστε αντιμέτωποι με προγράμματα που έχουν μεγάλο input data, κάτι που είναι σύνηθες σε ρεαλιστικές εφαρμογές και υλοποιήσεις, τότε στο προσκήνιο έρχεται η παράλληλη εκτέλεση.

Η παράλληλη εκτέλεση (parallel execution) λειτουργεί με το να εκτελεί ταυτόχρονα πολλές επιμέρους εντολές. Πλέον δεν υπάρχει η προϋπόθεση η προηγούμενη εντολή να έχει πρώτα εκτελεστεί ώστε να εκτελεστεί μετά η επόμενη, γιατί τώρα εκτελούνται την ίδια στιγμή και οι δύο. Πρόκειται για μια τεχνική που, όπως προκύπτει εύκολα και με τη λογική, συμβάλλει στη μείωση του συνολικού χρόνου εκτέλεσης, καθώς πλέον δεν υπάρχει στάδιο αναμονής-αδράνειας για μια εντολή. Είναι η πιο αποτελεσματική λύση για περιπτώσεις

μεγάλων και πολύπλοκων προγραμμάτων που υλοποιούν επεξεργασία πολλών δεδομένων. Εύκολα μπορεί να συμπεράνει κανείς ότι αν σε τέτοιου είδους υλοποιήσεις χρησιμοποιούνταν η σειριακή εκτέλεση, το αποτέλεσμα θα ήταν ένα δύσχρηστο πρόγραμμα με τεράστιο πρόβλημα χρονισμού. Σε υπολογιστικά συστήματα που χρησιμοποιούν παράλληλη εκτέλεση απαιτείται η χρήση πολλών επεξεργαστών, οι οποίοι για λόγους αποδοτικότητας είναι πιο γρήγοροι από τον έναν που έχουν τα υπολογιστικά συστήματα που χρησιμοποιούν σειριακή εκτέλεση, και ο καθένας τους έχει και λιγότερα δεδομένα να επεξεργαστεί, αφού η συνολική δουλειά μοιράζεται αποτελεσματικά σε όλους τους επεξεργαστές.[35]

Υπάρχουν περιπτώσεις που κάποιες υλοποιήσεις μπορούν να γίνουν μόνο σειριακά και εκεί χρειάζεται ιδιαίτερη προσοχή. Πρόκειται για τις περιπτώσεις όπου η εκτέλεση μιας εντολής εξαρτάται από την εκτέλεση κάποιας προηγούμενης.[36] Για παράδειγμα, όταν έχουμε μια εντολή η οποία χρησιμοποιεί το αποτέλεσμα της προηγούμενης εντολής από αυτήν για να υλοποιήσει κάποια λειτουργία. Σε τέτοιες περιπτώσεις εάν χρησιμοποιηθεί η παράλληλη εκτέλεση, οι δύο εντολές θα εκτελεστούν ταυτόχρονα και η δεύτερη δεν θα λάβει τη σωστή τιμή εισόδου ή δε θα λάβει καθόλου τιμή, κάτι που θα οδηγήσει είτε σε λάθος αποτέλεσμα στο τέλος της εκτέλεσης, ή ακόμα χειρότερα θα εμφανιστεί μήνυμα λάθους και το πρόγραμμα δεν θα μπορέσει καν να προχωρήσει τελικά. Σαν τελικό συμπέρασμα από όλα αυτά, θα μπορούσαμε να πούμε ότι σε περιπτώσεις όπου υπάρχει εξάρτηση μεταξύ εντολών, εφαρμόζουμε υποχρεωτικά σειριακή εκτέλεση.

Στην εικόνα 3.3 φαίνεται μια απλή σχηματική αναπαράσταση της σειριακής και της παράλληλης εκτέλεσης:



Πηγή: https://ozdemirhasan.medium.com/parallel-processing-with-dask-4647a58bd2a5

Εικόνα 3.3 Σειριακή και παράλληλη εκτέλεση

Κατά την υλοποίηση του αλγορίθμου FBM, έγινε αντιληπτή μια συγκεκριμένη περίπτωση κατά την οποία θα μπορούσε να εφαρμοζόταν η παράλληλη εκτέλεση. Αναλυτικότερα, στην αρχή του αλγορίθμου έπρεπε να διαβαστούν τα αρχεία που περιέχουν τα δεδομένα του καθενός benchmark design. Πρέπει να σημειώσουμε εδώ ότι κάθε ένα benchmark design αποτελούνταν από συνολικά τέσσερα διαφορετικά αρχεία. Το πρώτο αρχείο περιείχε πληροφορίες σχετικά με τα nets του κυκλώματος και με τα cells που το κάθε ένα από αυτά περιείχε. Είχε στο τέλος του ονόματος του την κατάληξη .nets. Το δεύτερο αρχείο περιείχε για κάθε cell τις τιμές του πλάτους και του ύψους του. Αποτελούνταν από τρεις στήλες: στην πρώτη αναγράφονταν τα ονόματα των cells, στη δεύτερη το πλάτος του καθενός και στην τελευταία το ύψος του καθενός, το οποίο είχε την ίδια τιμή για όλα τα cells, καθώς ο αλγόριθμος FBM υποθέτει ότι όλα τα cells που πρόκειται να γίνουν legalize έχουν ίδιο ύψος και διαφορετικά πλάτη. Το αρχείο αυτό είχε την κατάληξη .nodes στο όνομά του. Το τρίτο αρχείο είχε και αυτό τρεις στήλες με την πρώτη να είναι ξανά το όνομα του αρχείου, ενώ οι άλλες δύο ήταν οι αρχικές θέσεις x και y των cells στον χώρο. Το αρχείο αυτό τροποποιήθηκε δίνοντας τυχαίες συντεταγμένες για κάθε ένα cell, καθώς αρχικά ήταν όλες οι θέσεις αρχικοποιημένες στις θέσεις x=0 και y=0. Η κατάληξη του αρχείου ήταν .pl. Το τελευταίο αρχείο είχε κατάληξη .scl και περιείχε μέσα διάφορες πληροφορίες σχετικές με τις γραμμές της σχεδίασης(το ύψος, το μήκος της γραμμής και άλλα). Όλα αυτά τα αρχεία έπρεπε να διαβάζονται από το πρόγραμμα και να αποθηκεύονται σε κατάλληλες δομές, ώστε να μπορέσουν να χρησιμοποιηθούν και να τροποποιηθούν καταλλήλως στην πορεία της εκτέλεσης του αλγορίθμου FBM. Για τον σκοπό αυτό είχαν δημιουργηθεί τέσσερις συναρτήσεις, μία για κάθε ένα από τα τέσσερα αρχεία. Η ανάγνωση και αποθήκευση των δεδομένων του κάθε ενός αρχείου ήταν διαδικασία ανεξάρτητη η μία από την άλλη, καθώς τα αρχεία δεν συνδέονταν μεταξύ τους με κάποιον τρόπο. Το σκεπτικό στην αρχή ήταν να υλοποιηθεί η ανάγνωση των τεσσάρων αρχείων χρησιμοποιώντας την τεχνική της παράλληλης εκτέλεσης, όπως και έγινε, καθώς δεν υπήρχε λόγος η μια διαδικασία ανάγνωσης και αποθήκευσης να περιμένει πρώτα να τελειώσει η προηγούμενη διαδικασία ώστε να εκτελεστεί. Μια τέτοια κίνηση θεωρήθηκε πως θα βελτίωνε το συνολικό χρονισμό του αλγορίθμου.

Η τελική υλοποίηση της παράλληλης εκτέλεσης είχε ως εξής: Αρχικά στον χώρο του κυρίου προγράμματος κλήθηκε μια συνάρτηση που θα αναλάμβανε την παράλληλη εκτέλεση. Οι παράμετροί της ήταν τα ονόματα των τεσσάρων φακέλων και τα αποτελέσματα που αυτή θα επέστρεφε θα αποθηκεύονταν κατάλληλα σε δομές. Μέσα στην συγκεκριμένη συνάρτηση λοιπόν αρχικοποιήθηκε μια δεξαμενή νημάτων η οποία λειτουργεί ως εξής: Κάθε ένα νήμα από τη συγκεκριμένη δεξαμενή αναλαμβάνει την εκτέλεση κάποιας διαδικασίας, οπότε με την ανάθεση πολλών διαδικασιών σε πολλά νήματα ταυτόχρονα επιτυγχάνουμε την παράλληλη εκτέλεση στην πράξη. Ακολουθώντας αυτό το σκεπτικό, χρησιμοποιήθηκαν τέσσερα νήματα από την δεξαμενή, ένα για κάθε μια από τις συναρτήσεις μας.

Μετά το βήμα της ανάθεσης, υπήρξε μια σύντομη αναμονή των αποτελεσμάτων. Η συνάρτηση μπορούμε να πούμε πως «πάγωνε» κατά τη διάρκεια αυτού του διαστήματος μέχρι να επιβεβαιωθεί ότι όλα τα νήματα εκτέλεσαν με επιτυχία ότι τους ανατέθηκε και επέστρεψαν τα αντίστοιχα αποτελέσματα, τα οποία και επιστράφηκαν στο κύριο πρόγραμμα, όπου και αποθηκεύτηκαν κατάλληλα.

3.2.γ Τροποποιήσεις πυκνότητας.

Η τελευταία τροποποίηση που έγινε στα πλαίσια της υλοποίησης ήταν σχετική με την πυκνότητα. Η πυκνότητα ενός κυκλώματος είναι ο λόγος του αθροίσματος των εμβαδών όλων τον των κελιών προς το συνολικό εμβαδόν του chip. Το εμβαδόν του κάθε ενός cell υπολογίστηκε χρησιμοποιώντας τον απλό τύπο μήκος x πλάτος, μιας και θεωρούμε ότι στη σχεδίασή μας τα cells είναι μικρά παραλληλόγραμμα. Από την άλλη, το εμβαδόν του chip για κάθε ένα benchmark design υπολογίστηκε αρχικά πολλαπλασιάζοντας το σύνολο των γραμμών του συγκεκριμένου κυκλώματος με το ύψος κάθε μιας γραμμής. Το αποτέλεσμα αυτού του υπολογισμού πολλαπλασιάστηκε με το μήκος της κάθε μίας γραμμής και έτσι είχαμε το τελικό εμβαδόν του chip. Η πυκνότητα ενός κυκλώματος είναι ένα μέτρο που αναπαριστά θα μπορούσαμε να πούμε τι ποσοστό του συνολικού χώρου του chip είναι κατειλημμένο από cells. Η πυκνότητα ενός κυκλώματος πολλές φορές συμβαίνει να δέχεται τροποποιήσεις εάν παρουσιάστηκε κάποιο πρόβλημα κατά την εκτέλεση του αλγορίθμου πάνω στο κύκλωμα. Κάτι ανάλογο έγινε και στην περίπτωση της υλοποίησης του αλγορίθμου FBM.

Αναλυτικότερα, κατά την εκτέλεση του αλγορίθμου πάνω στα κυκλώματα χωρίς την εφαρμογή οποιασδήποτε τροποποίησης, παρατηρήθηκε το φαινόμενο κατά το τέταρτο βήμα του αλγορίθμου(και λίγες φορές κατά το δεύτερο) κάποια cells να βγαίνουν εκτός ορίων στο τέλος. Όπως έχει ήδη αναφερθεί, στο δεύτερο και στο τέταρτο βήμα γίνεται ταξινόμηση των cell με βάση διαφορετικά κριτήρια. Τα εκτός ορίων cells στην αρχή παρατηρήθηκαν σε όλα τα δοσμένα benchmark designs, το οποίο όμως δεν σήμαινε ότι υπάρχει λάθος με την υλοποίηση του αλγορίθμου FBM, καθώς ο ίδιος ο αλγόριθμος μεριμνά για τις περιπτώσεις που θα τύχει να υπάρχουν cells εκτός ορίων μετά τα βήματα των ταξινομήσεων. Όμως, για να εξασφαλιστεί η γενικότερη ορθότητα της όλης κατάστασης, έγινε ένας έλεγχος σχετικά με το αν τα δοσμένα cells χώραγαν στο διαθέσιμο χώρο που είχε δοθεί για κάθε ένα κύκλωμα, δηλαδή για το αν ο συνολικός διαθέσιμος χώρος ήταν μεγαλύτερος σε μέγεθος από το συνολικό άθροισμα του width όλων των cell. Έτσι θα διασφαλιζόταν το ότι το φαινόμενο αυτό ήταν καθαρά αποτέλεσμα εκτέλεσης του αλγορίθμου και δεν υπήρχε κάποιος άλλος παράγοντας που να ευθύνεται γι' αυτό, γιατί στην περίπτωση που δεν υπήρχε αρκετή χωρητικότητα για όλα τα cells στο κύκλωμα, τότε θα ήταν λογικό και αναμενόμενο αυτή να είναι η πιθανότερη αιτία που τα cells έβγαιναν εκτός των ορίων της σχεδίασης. Αφού υπολογίστηκαν οι τιμές του συνολικού χώρου και του αθροίσματος του width των cells για κάθε ένα κύκλωμα και έγιναν και οι απαραίτητες συγκρίσεις, τελικά βγήκε το συμπέρασμα ότι σε όλα τα κυκλώματα το άθροισμα του width των cells τύχαινε να είναι μεγαλύτερο από το συνολικό διαθέσιμο χώρο. Δηλαδή, για το πρώτο κύκλωμα με όνομα b01 το άθροισμα του width των cells ήταν 398.04 ενώ ο συνολικός διαθέσιμος χώρος ήταν 394,305. Για το κύκλωμα με όνομα b02 το άθροισμα του

width των cells ήταν 231,04, ενώ ο συνολικός διαθέσιμος χώρος 211,719. Για το κύκλωμα με όνομα s27 το συνολικό width των cells ήταν 121,8, ενώ ο συνολικός χώρος 90,49. Τέλος για το s208_1 το άθροισμα του width των cells ήταν 430,12 ενώ ο συνολικός χώρος 406,4823. Μετά από αυτή την παρατήρηση επόμενος στόχος ήταν η τροποποίηση της πυκνότητας όλων των κυκλωμάτων, ώστε να υπάρχει διαθέσιμη χωρητικότητα για όλα τα cells.

Αρχικά έγιναν μικρές αλλαγές ως προς τη χωρητικότητα των γραμμών της σχεδίασης. Το μέγεθος Numsites που υπήρχε στο αρχείο .scl των benchmark designs καθόριζε πόσο είναι το μήκος της κάθε γραμμής και άρα πόση είναι και η χωρητικότητά της. Αυξάνοντας τη χωρητικότητα των γραμμών κατά λίγο ώστε να χωράνε οριακά ή με λίγη άνεση όλα τα cells του κυκλώματος, παρατηρήθηκε το φαινόμενο πάλι κάποια cells να βρίσκονται εκτός των ορίων της σχεδίασης, αλλά μόνο στην τελευταία γραμμή, τόσο στο δεύτερο όσο και στο τέταρτο βήμα του αλγορίθμου FBM. Και πάλι λοιπόν όπως και πριν, αυτό είναι κάτι που θα λυνόταν από τον ίδιο τον αλγόριθμο. Στο δεύτερο βήμα τα cells κατά τη διαδικασία της τοποθέτησης τους στις γραμμές, πρώτα γέμιζαν μία γραμμή όσο πιο πολύ γινόταν και μετά πήγαιναν στην επόμενη, εάν το cell του οποίου ήταν τώρα η σειρά να τοποθετηθεί δεν χώραγε στον εναπομείναντα χώρο της προηγούμενης. Συνήθως ο χώρος που έμενε διαθέσιμος σε κάθε γραμμή ήταν πολύ λίγος και στην τελευταία γραμμή έμπαιναν πάντα τα cells με το μεγαλύτερο width, εφόσον ταξινομούνταν ως προς το width σε αύξουσα σειρά. Για όσα cells όμως έμεναν εκτός ορίων, δεν υπήρχε πουθενά αλλού διαθέσιμος χώρος για να επανατοποθετηθούν από το τρίτο βήμα, εφόσον όπως είπαμε η διαθέσιμη χωρητικότητα των υπόλοιπων γραμμών ήταν ήδη πολύ περιορισμένη και μόνο cells πολύ μικρού πλάτους θα ήταν εφικτό να χωρέσουν σε αυτές. Παρατηρήθηκε λοιπόν ότι το τρίτο βήμα του αλγορίθμου δεν θα μπορούσε να λειτουργήσει καν μη βρίσκοντας πουθενά χώρο για το cell, κάτι που θα δημιουργούσε σοβαρότατο πρόβλημα και στο τέταρτο βήμα, γιατί εκεί τα cells ταξινομούνταν με βάση τη χ θέση τους και όπως εύκολα μπορεί να συμπεράνει κανείς, η χ θέση των κελιών που βρίσκονταν έξω από τα όρια θα ήταν μια τιμή λανθασμένη, οδηγώντας στο τέλος όλο το legalization σε λανθασμένα αποτελέσματα.

Βλέποντας λοιπόν την ανάγκη για περαιτέρω τροποποίηση της πυκνότητας των κυκλωμάτων για να λυθεί αυτό το απροσδόκητο πρόβλημα, η χωρητικότητα των γραμμών πλέον αυξήθηκε αρκετά παραπάνω, ώστε με τον επιπλέον αυτό χώρο να επιλυθεί στα σίγουρα το ζήτημα που περιγράφηκε προηγουμένως. Η αύξηση του συνολικού μήκους των γραμμών ήταν αυθαίρετη στην αρχή με μοναδικό σκοπό την επιβεβαίωση ότι ο αλγόριθμος θα τρέχει σωστά επιτελώντας τις λειτουργίες που πρέπει για όλα τα βήματα. Το αποτέλεσμα της αύξησης της χωρητικότητας των γραμμών σε μεγαλύτερο βαθμό από ότι αρχικά ήταν το αναμενόμενο: ο αλγόριθμος έτρεχε σωστά για όλα τα βήματα.

Στο τέλος βρέθηκε και ποια είναι κατά μέσο όρο η μέγιστη πυκνότητα για την οποία ο αλγόριθμος είναι πλήρως λειτουργικός πάνω στα διαθέσιμα κυκλώματα. Όλα τα κυκλώματα σε τελική φάση τροποποιήθηκαν ώστε να βρίσκονται στα επίπεδα αυτής της πυκνότητας. Έγιναν αρκετοί πειραματισμοί για κάθε ένα κύκλωμα ξεχωριστά τροποποιώντας ξανά τη χωρητικότητα των γραμμών και υπολογίζοντας για κάθε μια τροποποίηση την τρέχουσα πυκνότητα. Για το κύκλωμα b01 έγιναν συνολικά τρεις δοκιμές, για το b02 έγιναν ξανά τρεις δοκιμές,

για το s27 έγιναν συνολικά τέσσερις και τέλος για το κύκλωμα s208_1 έγιναν συνολικά εφτά πειραματισμοί.

3.3 Οι επεκτάσεις

Ο αλγόριθμος όπως είπαμε ασχολείται καθαρά και μόνο με το στάδιο του legalization. Το detailed placement, όντας το επόμενο βήμα του legalization, είναι μια εντελώς διαφορετική διαδικασία που δεν συνδέεται με τον ίδιο τον αλγόριθμο και εκτελείται αφού ο αλγόριθμος ολοκληρώσει πλήρως τη λειτουργία του. Όπως τονίστηκε και σε προηγούμενο κεφάλαιο, το detailed placement υλοποιεί μια σειρά από βελτιστοποιήσεις πάνω στο τελικό αποτέλεσμα του legalization. Μια από αυτές είναι η προσπάθεια μείωσης του συνολικού μήκους του καλωδίου που χρησιμοποιείται στο κύκλωμα.

Μετά την επιτυχή εφαρμογή του αλγορίθμου στα δεδομένα και την εξαγωγή των τελικών αποτελεσμάτων, έγινε προσπάθεια υλοποίησης μιας συνάρτησης για το detailed placement, αλλά μόνο όσον άφορα το κομμάτι της ελαχιστοποίησης του καλωδίου, ενσωματώνοντας το και αυτό στη συνολική λειτουργία του αλγορίθμου. Στα πλαίσια αυτής της υλοποίησης λοιπόν, αρχικά δημιουργήθηκε μια συνάρτηση η οποία υπολόγιζε το μήκος του καλωδίου ακριβώς μετά την ολοκλήρωση του αλγορίθμου FBM, πριν δηλαδή από την εφαρμογή οποιασδήποτε βελτιστοποίησης. Η συνάρτηση κλήθηκε από το κύριο πρόγραμμα με παραμέτρους την τελική τοποθέτηση των cells στο κύκλωμα από τον legalizer και το κατάλληλα αποθηκευμένο αποτέλεσμα της συνάρτησης ανάγνωσης του .nets αρχείου, δηλαδή τα nets μαζί με τα cells που το κάθε ένα περιείχε. Στη συνέχεια η συνάρτηση αυτή υπολόγιζε τον τύπο "|minx-maxx| + |miny-maxy|" για κάθε ένα net και στο τέλος άθροιζε τα αποτελέσματα. Ο προηγούμενος τύπος υπολογίζει το μικρότερο παραλληλόγραμμο το οποίο χωράει όλα τα cells ενός net. Αυτό το άθροισμα των μικρότερων παραλληλογράμμων είναι και το ελάχιστο μήκος καλωδίου που απαιτείται.

Στη συνέχεια, καλείται από το κύριο πρόγραμμα μια άλλη συνάρτηση η οποία δέχεται ως ορίσματα πάλι τις τελικές θέσεις των cells που προκύπτουν από τον αλγόριθμο FBM και τα nets της σχεδίασης. Πρόκειται για τη συνάρτηση που υλοποιεί το detailed placement. Η συνάρτηση αυτή ξεκινά προσπελαύνοντας για κάθε ένα net όλα τα cells του. Έπειτα συγκρίνει κάθε cell με όλα τα υπόλοιπα cells όλων των υπολοίπων nets και σε περίπτωση που βρεθεί ότι έχουν το ίδιο width, πραγματοποιεί προσωρινή αντιμετάθεσή τους. Ο λόγος που γίνεται σύγκριση με βάση το width των cell είναι γιατί δεν θέλουμε στο τέλος να προκύψουν λάθη ως προς το legalization που υλοποιήθηκε νωρίτερα. Αν για παράδειγμα τύχει το νέο cell που θα τοποθετηθεί στη θέση του παλιού να έχει μεγαλύτερο width από το παλιό, τότε σε περίπτωση που η ανταλλαγή αυτή μονιμοποιηθεί, θα παρατηρηθούν πάλι φαινόμενα επικάλυψης ανάμεσα σε cells, κάτι που σημαίνει ότι θα πρέπει να ξανατρέξουμε τον αλγόριθμο FBM ώστε αυτά να εξαλειφθούν. Οπότε για να αποφευχθεί αυτή η επιπλέον διαδικασία, γίνεται πρώτα ο έλεγχος ως προς το width και μετά πραγματοποιείται οποιαδήποτε κίνηση αντιμετάθεσης. Η αντιμετάθεση είναι προσωρινή, γιατί δεν ξέρουμε αν ακόμα η αντιμετάθεση αυτή είναι η βέλτιστη που θα μπορούσε να είχε γίνει ως προς την ελαχιστοποίηση του συνολικού μήκους καλωδίου, ενώ στόχος μας είναι για κάθε ένα cell να βρεθεί η

βέλτιστη αντιμετάθεση ως προς το συγκεκριμένο κριτήριο. Αφού λοιπόν γίνει η αντιμετάθεση, υπολογίζουμε εκ νέου το συνολικό μήκος καλωδίου. Έπειτα κάθε cell επιστρέφει στην αρχική του θέση και η διαδικασία των συγκρίσεων συνεχίζεται επαναληπτικά μέχρις ότου να έχουν γίνει όλες οι προβλεπόμενες συγκρίσεις. Για κάθε ένα cell αποθηκεύουμε τη βέλτιστη αντιμετάθεση και την υλοποιούμε στο τέλος, γιατί μόνο τότε είμαστε σίγουροι ότι αυτή είναι όντως η βέλτιστη επιλογή. Η ίδια διαδικασία συνεχίζεται λοιπόν και για τα υπόλοιπα cells. Έτσι στο τέλος, παράλληλα με τις αντιμεταθέσεις, έχει βρεθεί και το νέο μήκος του καλωδίου.

Αφού λοιπόν η όλη παραπάνω υλοποίηση δοκιμάστηκε στην πράξη, σε πρώτη φάση παρατηρήθηκε ότι σε μερικά nets υπήρχαν κάποια cells παραπάνω από μία φορά, κάτι που δείχνει ότι έγινε λάθος στην υλοποίηση του detailed placement. Για να αποφευχθεί λοιπόν αυτό το φαινόμενο έγινε κατάλληλη προσθήκη που έλυνε το πρόβλημα, και έτσι σε περίπτωση που βρισκόντουσαν όντως cells με το ίδιο width, πραγματοποιούνταν έλεγχος για το αν το κάθε ένα από αυτά βρισκόταν ήδη στο net στο οποίο επρόκειτο να τοποθετηθεί για τους λόγους της αντιμετάθεσης. Εάν κάτι τέτοιο ήταν αληθές, η αντιμετάθεση δεν γινόταν, γιατί παρότι ήταν προσωρινή και προς το παρόν δεν θα δημιουργούσε πρόβλημα, εφόσον τα cells στο τέλος θα επέστρεφαν στις αρχικές τους θέσεις, υπήρχε πάντοτε πιθανότητα στο τέλος αυτή η αντιμετάθεση να ήταν και η βέλτιστη, οπότε και να υλοποιούνταν μόνιμα, με το τελικό αποτέλεσμα του detailed placement να έχει ξανά το ίδιο λάθος.

Στο τέλος της υλοποίησης επιστρέφονταν τα τροποποιημένα από τις αντιμεταθέσεις nets καθώς και το νέο ελαχιστοποιημένο μήκος καλωδίου.

ΚΕΦΑΛΑΙΟ 4. Πειραματικά αποτελέσματα και συμπεράσματα.

4.1 Εισαγωγή

Μετά τις τροποποιήσεις και προσθήκες που περιγράφηκαν λεπτομερώς στο προηγούμενο κεφάλαιο, υπήρξε το στάδιο της υλοποίησης τους. Αφού έγινε προσεκτική εκτέλεση του αλγορίθμου FBM μαζί με τα τέσσερα αυτά νέα στοιχεία, σειρά τώρα είχε το κομμάτι της αναλυτικής καταγραφής των αποτελεσμάτων της εκτέλεσής τους, καθώς και η προσπάθεια εξαγωγής τελικών συμπερασμάτων μέσα από αυτά τα αποτελέσματα. Τα κυκλώματα που δόθηκαν πάνω στα οποία έτρεξαν όλα τα προηγούμενα, όπως είχε αναφερθεί και σε προηγούμενο κεφάλαιο, ήταν τέσσερα, το καθένα με διαφορετικό αριθμό από cells και nets. Εδώ πρέπει να τονιστεί το γεγονός ότι χρησιμοποιήθηκαν μόνο τα τελικά τροποποιημένα ως προς την πυκνότητα κυκλώματα, αλλά αξιοποιώντας το μέγεθος της μέγιστης πυκνότητας για την οποία ο FBM ήταν πλήρως λειτουργικός για το καθένα, έτσι

ώστε να υπάρχει μια πιο σαφής και συγκεκριμένη υλοποίηση και να μπορούν να ερμηνευτούν και με πιο στοχευμένο τρόπο τα αποτελέσματα που θα προέκυπταν από τα παραπάνω κυκλώματα. Η υλοποίηση του αλγορίθμου FBM, τόσο η αρχική που περιείχε μόνο τα βήματα του αλγορίθμου όσο και η τροποποιημένη, έγινε σε γλώσσα Python σε μια μηχανή WSL Ubuntu 24.04.1 LTS αρχιτεκτονικής x86_64 με AMD Ryzen 5 4500U with Radeon Graphics και 7.44GB μνήμη.

4.2 Πρώτοι πειραματισμοί

4.2.α Αλγόριθμοι ταξινόμησης

Όσον αφορά τους αλγορίθμους ταξινόμησης, αρχικά δοκιμάστηκε η μεμονωμένη εκτέλεσή τους πάνω στα βήματα δύο και τέσσερα του αλγορίθμου προτού γίνει η υλοποίησή τους πάνω στον συνολικό αλγόριθμο. Κίνητρο αυτού του σκεπτικού ήταν να το να υπάρξει μια πρώτη ρεαλιστική εικόνα πάνω στις επιδράσεις του καθενός ως προς το χρόνο εκτέλεσης και να γίνει μια πρώτη έμπρακτη κατανόηση της αποδοτικότητας του κάθε αλγορίθμου. Για κάθε έναν αλγόριθμο ταξινόμησης επιλέχθηκε να γραφεί ξανά όλη η υπόλοιπη υλοποίηση του αλγορίθμου μαζί με τις υπόλοιπες τροποποιήσεις, έτσι ώστε στο τέλος να έχουμε τρία ξεχωριστά προγράμματα με μόνες διαφορές τον αλγόριθμο ο οποίος θα υλοποιούσε την ταξινόμηση σε κάθε περίπτωση, με τελικό στόχο να είναι όλες οι υλοποιήσεις πιο καθαρογραμμένες. Αφού λοιπόν είχαμε μια υλοποίηση τέτοιου είδους, επόμενο βήμα ήταν η μέτρηση του χρόνου εκτέλεσης των βημάτων που μας ενδιέφεραν για κάθε ένα από τα τέσσερα benchmark designs. Και σε αυτό το αρχικό στάδιο χρησιμοποιήθηκαν τα τροποποιημένα ως προς την πυκνότητα κυκλώματα για τους πειραματισμούς που έγιναν.

Αρχικά, δοκιμάστηκε η υλοποίηση του βήματος δύο χρησιμοποιώντας τον αλγόριθμο Timsort. Τρέχοντας το βήμα δύο για το κύκλωμα b01,ο χρόνος εκτέλεσης που σημειώθηκε ήταν 0.0002 seconds. Για το ίδιο κύκλωμα αλλά χρησιμοποιώντας τον αλγόριθμο Bubble sort το βήμα δύο ολοκληρώθηκε σε 0.0017 seconds, ενώ με τον Insertion sort τα αποτελέσματα εμφανίστηκαν σε 0.0008 seconds. Σε αυτό το σημείο έγινε μια πρώτη γρήγορη παρατήρηση σχετικά με τους χρόνους εκτέλεσης που προσέφεραν οι τρεις διαφορετικοί αλγόριθμοι ταξινόμησης, με τον Timsort να είναι με διαφορά ο πιο γρήγορος αλγόριθμος για τη συγκεκριμένη περίπτωση. Οι πειραματισμοί συνεχίστηκαν και για το τέταρτο βήμα του αλγορίθμου πάνω στο συγκεκριμένο κύκλωμα. Οι χρόνοι εκτέλεσης για τους Timsort, Bubble sort και Insertion sort ήταν αντιστοίχως: 0.0007 seconds, 0.0026 seconds και 0.0015 seconds.

Η ίδια ακριβώς διαδικασία ακολουθήθηκε και για τα υπόλοιπα τρία κυκλώματα. Στο b02 για το βήμα δύο οι αλγόριθμοι Timsort, Bubble sort και Insertion sort είχαν αποτελέσματα 0.0001 seconds, 0.0005 seconds και 0.0003 seconds, ενώ για το βήμα τέσσερα πάνω στο ίδιο κύκλωμα είχαν αντίστοιχα 0.0004 seconds, 0.0008 seconds και 0.0006 seconds. Για το τρίτο κύκλωμα με όνομα s27 οι επιδόσεις των τριών αλγορίθμων με την ίδια σειρά(Timsort, Bubble sort και Insertion sort) για το βήμα δύο ήταν: 0.0001 seconds, 0.0002 seconds και 0.0001

seconds, ενώ για το τέταρτο βήμα είχαμε τους εξής χρόνους: 0.0003 seconds, 0.0005 seconds και 0.0003 seconds. Τέλος, για το κύκλωμα s208_1 οι χρόνοι για το δεύτερο βήμα (Timsort, Bubble sort και Insertion sort) είναι: 0.0002 seconds, 0.0019 seconds και 0.0006 seconds, ενώ για το τέταρτο βήμα του αλγορίθμου σημειώθηκαν οι εξής χρόνοι: 0.0006 seconds, 0.0024 seconds και 0.0033 seconds.

Μετά την εκτέλεση όλων των βημάτων του αλγορίθμου στα οποία υπήρχε η ανάγκη για ταξινόμηση των κελιών, διαπιστώθηκε ότι το φαινόμενο που παρατηρήθηκε αρχικά στο πρώτο κύκλωμα (b01) παρατηρήθηκε τώρα και κατά την καταγραφή των αποτελεσμάτων των υπόλοιπων κυκλωμάτων ως προς το χρόνο εκτέλεσης. Πιο συγκεκριμένα, σε όλες τις περιπτώσεις, τόσο για το τέταρτο όσο και για το δεύτερο βήμα, ο πιο γρήγορος αλγόριθμος ταξινόμησης ήταν ξανά ο αλγόριθμος Timsort. Στη συνέχεια ο αμέσως πιο γρήγορος ήταν σε όλες τις περιπτώσεις (με εξαίρεση το τέταρτο βήμα του κυκλώματος s208_1) ο Insertion sort, ο οποίος σε γενικές γραμμές δεν είχε μεγάλη διαφορά στον χρόνο εκτέλεσης ως προς τον Timsort και μάλιστα στην περίπτωση του κυκλώματος s27 οι δύο αυτοί αλγόριθμοι είχαν ακριβώς το ίδιο runtime για το δεύτερο και για το τέταρτο βήμα του αλγορίθμου. Η πιθανότερη αιτία που παρατηρήθηκε αυτό το φαινόμενο ήταν ότι το s27, όπως είχε τονιστεί και σε προηγούμενο κεφάλαιο, ήταν το κύκλωμα με τα λιγότερα συνολικά cells, έχοντας μεγάλη διαφορά σε σχέση με τα υπόλοιπα τρία benchmark designs. Επειδή λοιπόν το input data της περίπτωσης του s27 ήταν αρκετά μικρό, ο Insertion sort υπήρξε ιδιαίτερα αποδοτικός, φτάνοντας τα επίπεδα του Timsort, επιβεβαιώνοντας στην πράξη το γεγονός ότι ο αλγόριθμος Insertion sort είναι πολύ γρήγορος και αποδοτικός σε περιπτώσεις που έχουμε υλοποίηση με μικρό input data. Από την άλλη, ο Timsort υπήρξε ιδιαίτερα αποδοτικός ακόμα και σε περιπτώσεις μεγαλύτερων κυκλωμάτων, που αναπαριστούσαν σχεδιάσεις με πολλά cells, μια κατάσταση που επαληθεύει το γεγονός ότι ο Timsort είναι ικανός να χειριστεί αποδοτικά μεγάλες και «πιο κοντά στην πραγματικότητα» υλοποιήσεις. Η αρχική υπόθεση που είχε γίνει κατά την επιλογή του αλγορίθμου Timsort ότι θα είναι πιο αποδοτικός αλγόριθμος για την περίπτωση της ταξινόμησης των cells από τον Insertion sort, καθώς ο Timsort, όντας ένας υβριδικός αλγόριθμος, έχει τη δυνατότητα να πραγματοποιεί κατάλληλες εναλλαγές ανάμεσα στον Insertion sort και στον Merge sort, εκμεταλλευόμενος μόνο τα δυνατά σημεία του καθενός, αποδείχθηκε τελικά σωστή, με τον Timsort να σημειώνει τις καλύτερες επιδόσεις και για τα μεγάλα κυκλώματα. Από την άλλη, ο Insertion sort για μεγαλύτερα κυκλώματα δεν κατάφερε να έχει τις βέλτιστες επιδόσεις, καθώς από μόνος του δεν έχει κατάλληλη υλοποίηση ώστε να καταφέρει να διαχειριστεί περιπτώσεις που δεν είναι το δυνατό του σημείο, καταλήγοντας να είναι πιο χρονοβόρος. Το σκεπτικό ότι ο Insertion sort θα ήταν ο πιο αποτελεσματικός αλγόριθμος για τη συγκεκριμένη περίπτωση της ταξινόμησης κελιών λόγω του όχι υπερβολικά μεγάλου αριθμού κελιών που είχαν τα κυκλώματα b01, b02 και s208_1 αποδείχθηκε λανθασμένο. Τα παραπάνω κυκλώματα είχαν μεν παραπάνω cells από το s27, αλλά σε καμιά περίπτωση ο αριθμός τους δεν έφτανε σε πολύ μεγάλα επίπεδα. Για να γίνει καλύτερη κατανόηση της κατάστασης, το b01 κύκλωμα είχε συνολικά 91 cells, το b02 48 και το s208_1 αποτελούνταν από συνολικά 85 κελιά (με το s27 να έχει μόλις 25). Θεωρήθηκε λοιπόν πριν την πραγματοποίηση οποιασδήποτε υλοποίησης ότι είναι πιθανό, λόγω του γεγονότος ότι τα δεδομένα προς ταξινόμηση σε γενικές γραμμές δεν ήταν πάρα πολλά σε κάθε benchmark

design, ο Insertion sort να είναι αρκετά αποδοτικός και ίσως ο αποδοτικότερος για όλες τις περιπτώσεις κυκλωμάτων που έχουμε στη διάθεσή μας. Ωστόσο, όπως τελικά αποδείχτηκε έμπρακτα(κύκλωμα s27), ο αλγόριθμος Insertion sort είναι η βέλτιστη περίπτωση για τον αλγόριθμο FBM μόνο σε περιπτώσεις όπου το input data είναι πολύ μικρό. Σε περιπτώσεις που είναι από λίγο έως πολύ ανεβασμένο το πλήθος των δεδομένων εισόδου, η επίδοση του αλγορίθμου αρχίζει σταδιακά να μειώνεται. Η αρχική εκτίμηση λοιπόν για αυτόν τον αλγόριθμο ότι θα ήταν η πιο αποδοτική επιλογή ήταν εσφαλμένη, γιατί δεν έγινε σωστή εκτίμηση της αποδοτικότητας του αλγορίθμου Insertion sort ως προς τα μεγέθη του input data που υπήρχαν στη συγκεκριμένη περίπτωση που είχαμε.

Όσον αφορά τον αλγόριθμο Bubble sort από την άλλη πλευρά, τα αποτελέσματά του ως προς το runtime ήταν πολύ χειρότερα από τους άλλους δύο αλγορίθμους, παρουσιάζοντας αρκετά μεγάλους χρόνους. Άλλωστε, όπως γίνεται αντιληπτό και από τα προηγούμενα αποτελέσματα που παράχθηκαν, η «ταξινόμηση» των αλγορίθμων ταξινόμησης ως προς το runtime σε αύξουσα διάταξη ήταν σχεδόν σε όλα τα τελικά καταγεγραμμένα αποτελέσματα η εξής: Timsort, Insertion sort και Bubble sort. Το γεγονός ότι ο Bubble sort θα αποτελούσε τη χειρότερη επιλογή αλγορίθμου ταξινόμησης για την περίπτωση της ταξινόμησης κελιών κατά τη διάρκεια της διαδικασίας του Legalization ήταν πλήρως αναμενόμενο, όπως θα ήταν αναμενόμενο και για οποιαδήποτε άλλη περίπτωση που έπρεπε να επιλέξουμε κάποιον αλγόριθμο για την υλοποίηση ενός σεναρίου ταξινόμησης. Μοναδική εξαίρεση σε αυτή την διαπίστωση ίσως αποτελεί η περίπτωση που έχουμε πολύ μικρό αριθμό στοιχείων. Εκεί πολύ πιθανό να αποδειχθεί μια αποδοτική λύση, ακριβώς λόγω του υπερβολικά μικρού αριθμού δεδομένων προς ταξινόμηση. Άλλωστε, όπως φαίνεται και από τα τελικά αποτελέσματα των εκτελέσεων, για την περίπτωση του s27, ο χρόνος εκτέλεσής του ήταν αρκετά κοντά στους χρόνους εκτέλεσης των άλλων δύο αλγορίθμων, φτάνοντας μάλιστα στην περίπτωση της εκτέλεσης του δεύτερου βήματος οι χρόνοι τους να είναι σχεδόν οι ίδιοι, με τον Bubble sort να είναι πιο αργός κατά μόνο 0.0001 seconds. Σε αυτό το κύκλωμα σημείωσε ο Bubble sort τους καλύτερους χρόνος και ήταν το κύκλωμα στο οποίο υπήρξε πιο αποδοτικός από οποιαδήποτε άλλο κύκλωμα, καθώς όπως προαναφέρθηκε, ήταν αυτό με τη λιγότερη «δουλειά» στο κομμάτι της ταξινόμησης από όλα τα υπόλοιπα. Εννοείται όμως ότι για περισσότερα στοιχεία, οι χρόνοι που σημείωσε ξεπερνούσαν κατά πολύ τους χρόνους των άλλων αλγορίθμων. Χαρακτηριστικό παράδειγμα αποτελεί η περίπτωση του δεύτερου βήματος του αλγορίθμου για το κύκλωμα s208_1, όπου ο Bubble sort σημείωσε χρόνο 0.0019 seconds, ο Timsort 0.0002 seconds και ο Insertion 0.0006 seconds, όπως και η περίπτωση του δεύτερου βήματος για το κύκλωμα b01, όπου ο Bubble sort σημείωσε runtime πάνω από το διπλάσιο του Insertion sort (Insertion sort 0.0008 seconds, Bubble sort 0.0017 seconds). Ο αλγόριθμος Bubble sort είναι χρήσιμος στην περίπτωσή μας, καθώς για τις περιπτώσεις των πιο μεγάλων κυκλωμάτων μας βοηθάει να έχουμε μια γενική εικόνα για ίσως έναν από τους χειρότερους χρόνους που θα μπορούσαμε γενικά να έχουμε για την συγκεκριμένη περίπτωση ταξινόμησης και αυτό μας βοηθά να βγάλουμε καλύτερα τελικά συμπεράσματα για τους δύο άλλους αλγορίθμους και κυρίως για τον Insertion sort. Καταρχήν, σε όλες σχεδόν τις περιπτώσεις, ακόμα και για τα κυκλώματα με μεγαλύτερο πλήθος από cells, βλέπουμε σε γενικές γραμμές ότι ο Insertion sort «πλησιάζει» τον Timsort, όπως για παράδειγμα στην περίπτωση του δεύτερου

βήματος του κυκλώματος b02, με τον Timsort να βρίσκεται στο πολύ καλό επίπεδο των 0.0001 seconds, ενώ ο Insertion να σημειώνει 0.0003 seconds, αλλά και στην περίπτωση του δεύτερου βήματος του κυκλώματος s208_1 οι επιδόσεις των δύο αυτών αλγορίθμων είναι σε γενικές γραμμές κοντά μεταξύ τους(Timsort 0.0002 seconds και Insertion 0.0006 seconds). Αν σε αυτές τις περιπτώσεις συμπεριλάβουμε και τις επιδόσεις του Bubble sort(0.0005 seconds στην πρώτη και 0.0019 seconds στην δεύτερη περίπτωση), βλέποντας τον Insertion sort όπως είπαμε να βρίσκεται σχετικά κοντά από άποψη επίδοσης στον Timsort και συγκρίνοντας τον Insertion με ίσως την χειρότερη περίπτωση που είναι ο Bubble sort, μπορεί κανείς εύκολα να συνειδητοποιήσει ότι τελικά ο Insertion sort, για την περίπτωση ταξινόμησης των κελιών, είναι σε γενικές γραμμές ένας σχετικά αποδοτικός αλγόριθμος, αλλά σαφώς όχι ο αποδοτικότερος.

4.2.β Σειριακή και παράλληλη εκτέλεση

Μετά από αυτούς τους πρώτους πειραματισμούς, σειρά είχαν κάποιοι πειραματισμοί σχετικοί με την υλοποίηση της παράλληλης εκτέλεσης αντί για τη σειριακή στις περιπτώσεις ανάγνωσης των τεσσάρων αρχείων εισόδου, όπως περιγράφηκε αναλυτικά στο βήμα τρία. Τόσο η σειριακή όσο και η παράλληλη εκτέλεση αυτής της λειτουργίας υλοποιήθηκαν με χρήση ξεχωριστών συναρτήσεων για την καθεμία. Η μία συνάρτηση απλά καλούσε τις συναρτήσεις που θα υλοποιούσαν τις αναγνώσεις των αρχείων τη μία μετά την άλλη, ενώ η δεύτερη υλοποιούσε την παράλληλη εκτέλεση με μία δεξαμενή νημάτων. Για κάθε μία συνάρτηση καταγράφηκε λοιπόν ο χρόνος εκτέλεσής της. Για το κύκλωμα b01 η συνάρτηση της σειριακής εκτέλεσης ολοκλήρωσε την λειτουργία της σε 0.0007 seconds, ενώ η συνάρτηση για την παράλληλη τελείωσε σε 0.0025 seconds. Για το κύκλωμα b02 η σειριακή παρήγαγε αποτελέσματα σε 0.0004 seconds, ενώ η παράλληλη σε 0.0022 seconds, για το κύκλωμα s27 η σειριακή εκτέλεση είχε runtime 0.0004 seconds και η παράλληλη σε 0.0022 seconds και τέλος για το κύκλωμα s208_1 η σειριακή εκτέλεση ολοκληρώθηκε σε 0.0005 seconds και η παράλληλη σε 0.0022 seconds.

Αφού έγινε λοιπόν καταγραφή των αποτελεσμάτων για όλα τα κυκλώματα, παρατηρήθηκε ότι η αρχική υπόθεση πως με την χρήση της παράλληλης εκτέλεσης αντί για την σειριακή θα βελτιωνόταν ο χρόνος εκτέλεσης ήταν πλήρως λανθασμένη. Όπως αποδείχθηκε, τόσα στα κυκλώματα που είχαν περισσότερα cells όσο και στο s27 που είχε αρκετά λιγότερα, η παράλληλη εκτέλεση καθυστερούσε σε πολύ σημαντικό βαθμό σε σχέση με την σειριακή. Παρά το σκεπτικό ότι η σειριακή εκτέλεση θα ήταν μια πιο χρονοβόρα προσέγγιση από τη στιγμή που κάθε μία συνάρτηση έπρεπε να περιμένει να εκτελεστεί η προηγούμενή της πρώτα ώστε να εκτελεστεί μετά αυτή, τελικά υπήρξε η πιο αποδοτική επιλογή για τη συγκεκριμένη περίπτωση, φτάνοντας σε σημεία να ολοκληρώνεται έως και πεντέμιση φορές νωρίτερα από την παράλληλη εκτέλεση(στην περίπτωση του κυκλώματος s27). Ενώ λοιπόν υπήρχαν όλες οι απαραίτητες προϋποθέσεις που θα οδηγούσαν την παράλληλη εκτέλεση να υπεριοχύσει ως προς την αποτελεσματικότητα, τόσο από άποψη έλλειψης εξάρτησης ανάμεσα στις εντολές, όσο και από άποψη άσκοπης αναμονής των εντολών, τελικά υπερίσχυσε η σειριακή εκτέλεση. Ο λόγος που υπήρξε αυτό το απροσδόκητο αποτέλεσμα θεωρήθηκε πως

ήταν η δομή των συναρτήσεων που κλήθηκαν χρησιμοποιώντας τα δύο είδη εκτέλεσης. Επρόκειτο για συναρτήσεις με πολύ μικρή πολυπλοκότητα. Και οι τέσσερις είχαν μόνο ένα for loop στον κώδικά τους, με όλα τα υπόλοιπα να είναι απλές εντολές, κάτι που οδηγεί στο συμπέρασμα ότι θα είχαν πολύ μικρό χρόνο εκτέλεσης. Στην περίπτωση της παράλληλης εκτέλεσης, με τη χρήση της δεξαμενής νημάτων, ο χρόνος που απαιτήθηκε για την αρχικοποίηση των νημάτων, την ανάθεση κάθε μίας συνάρτησης σε ένα νήμα και την επιστροφή των αποτελεσμάτων από το κάθε ένα, κατέληξε να είναι πολύ υψηλότερος από το χρόνο που απαιτήθηκε απλά για να εκτελεστούν οι συναρτήσεις η μία μετά την άλλη, ακολουθώντας την φυσική ροή των εντολών. Προφανώς, σε περίπτωση που είχαμε να κάνουμε με συναρτήσεις πιο πολύπλοκες στις οποίες επιτελούνταν πιο σύνθετες και περίπλοκες λειτουργίες, οπότε κατά συνέπεια είχαμε και σημαντικά αυξημένο χρόνο εκτέλεσης, θα ήταν πολύ πιθανό η παράλληλη εκτέλεση να ήταν η πιο αποτελεσματική λύση, καθώς σε μια τέτοια περίπτωση θα αυξανόταν σημαντικά και ο χρόνος αναμονής κάθε μίας συνάρτησης για να εκτελεστεί πρώτα η προηγούμενη(στην περίπτωση της σειριακής εκτέλεσης). Το τελικό συμπέρασμα λοιπόν ήταν ότι η σειριακή εκτέλεση ήταν με διαφορά η βέλτιστη επιλογή για την περίπτωσή μας.

4.3 Τελικά αποτελέσματα

4.3.α Πίνακας συγκεντρωτικών αποτελεσμάτων

Στη συνέχεια αφού ολοκληρώθηκε η διαδικασία αυτή των πρώτων πειραματισμών και βγήκαν κάποια πρώτα συμπεράσματα, έγινε μια πιο συνολική και συγκεντρωτική καταγραφή αποτελεσμάτων. Για κάθε ένα κύκλωμα, χρησιμοποιώντας κάθε φορά έναν από τους τρεις αλγορίθμους ταξινόμησης, καθώς και εξετάζοντας για κάθε έναν τόσο την περίπτωση της σειριακής εκτέλεσης όσο και την περίπτωση της παράλληλης, όπως αυτές περιγράφηκαν προηγουμένως, έχοντας δηλαδή 6 επιμέρους περιπτώσεις συνολικά για κάθε ένα, καταγράφηκε η τιμή του μήκους του καλωδίου της σχεδίασης πριν την εφαρμογή της συνάρτησης που υλοποιούσε το detailed placement, η αλλαγή στην τιμή του μήκους του καλωδίου μετά την εφαρμογή της συνάρτησης, ο χρόνος εκτέλεσης για τον αλγόριθμο FBM(χωρίς να συμπεριλαμβάνεται το detailed placement), το runtime μόνο για τη συνάρτηση του detailed placement, ο συνολικός χρόνος εκτέλεσης για όλη την υλοποίηση μαζί και τέλος η ποσοστιαία μεταβολή του συνολικού μήκους καλωδίου. Τα αναλυτικά αποτελέσματα φαίνονται στη εικόνα 4.1

b01	sorting algorithm	multiprocessing	THPWL after FBM	runtime for FBM	THPWL after Detailed Placemen	runtime for Detailed Placement	Total Runtime	% improvement in THPWL
scenario#1	BubbleSort	no	4383.31	0.0110 sec	1987.31	2.3310 sec	2,342	54.6%
scenario#2	BubbleSort	yes	4383.31	0.0177 sec	1987.31	2.2485 sec	2,2662	54.6%
scenario#3	TimSort	no	4383.31	0.0040 sec	1987.31	2.2279 sec	2,2319	54.6%
scenario#4	TimSort	yes	4383.31	0.0103 sec	1987.31	2.2060 sec	2,2163	54.6%
scenario#5	InsertionSort	no	4383.31	0.0074 sec	1987.31	2.2006 sec	2,208	54.6%
scenario#6	InsertionSort	yes	4383.31	0.0159 sec	1987.31	2.4129 sec	2,4288	54.6%
bo2								
scenario#1	BubbleSort	no	1945.09	0.0057 sec	1079.72	0.7070 sec	0,7127	44.4%
scenario#2	BubbleSort	yes	1945.09	0.0144 sec	1079.72	0.6900 sec	0,7044	44.4%
scenario#3	TimSort	no	1945.09	0.0031 sec	1079.72	0.6583 sec	0,6614	44.4%
scenario#4	TimSort	yes	1945.09	0.0125 sec	1079.72	0.6653 sec	0,6778	44.4%
scenario#5	InsertionSort	no	1945.09	0.0050 sec	1079.72	0.7184 sec	0,7234	44.4%
scenario#6	InsertionSort	yes	1945.09	0.0133 sec	1079.72	0.7138 sec	0,7271	44.4%
s27								
scenario#1	BubbleSort	no	741.32	0.0017 sec	450.38	0.0334 sec	0,0351	39.2%
scenario#2	BubbleSort	yes	741.32	0.0069 sec	450.38	0.0335 sec	0,0404	39.2%
scenario#3	TimSort	no	741.32	0.0015 sec	450.38	0.0323 sec	0,0338	39.2%
scenario#4	TimSort	yes	741.32	0.0061 sec	450.38	0.0340 sec	0,0401	39.2%
scenario#5	InsertionSort	no	741.32	0.0017 sec	450.38	0.0341 sec	0,0358	39.2%
scenario#6	InsertionSort	yes	741.32	0.0099 sec	450.38	0.0351 sec	0,045	39.2%
s208_1								
scenario#1	BubbleSort	no	4013.14	0.0071 sec	2003.05	1.0697 sec	1,0768	50%
scenario#2	BubbleSort	yes	4013.14	0.0070 sec	2003.05	1.0402 sec	1,0472	50%
scenario#3	TimSort	no	4013.14	0.0035 sec	2003.05	1.0352 sec	1,0387	50%
scenario#4	TimSort	yes	4013.14	0.0037 sec	2003.05	1.1141 sec	1,1178	50%
scenario#5	InsertionSort	no	4013.14	0.0040 sec	2003.05	1.0838 sec	1,0878	50%
scenario#6	InsertionSort	yes	4013.14	0.0048 sec	2003.05	1.0025 sec	1,0073	50%

Εικόνα 4.1 Ο πίνακας με τα τελικά αποτελέσματα

Στον παραπάνω πίνακα φαίνονται όλα όσα περιγράφηκαν προηγουμένως. Αρχικά, μπορούμε να δούμε την ξεκάθαρη μείωση στο συνολικό μήκος του καλωδίου της σχεδίασης, με το πρώτο κύκλωμα να σημειώνει τη μεγαλύτερη μείωση(54.6%). Τα τελικά ποσοστά αυτής της βελτίωσης ήταν σε γενικές γραμμές αρκετά καλά για όλα τα benchmark designs, αποδεικνύοντας ότι ο αλγόριθμος που χρησιμοποιήθηκε για το detailed placement υπήρξε αρκετά αποδοτικός. Προφανώς και το αποτέλεσμα του detailed placement δεν επηρεαζόταν από την χρήση διαφορετικών αλγορίθμων ταξινόμησης, ούτε από την χρήση της σειριακής ή της παράλληλης εκτέλεσης, οπότε ήταν λογικό και επόμενο για κάθε μία από τις έξι περιπτώσεις κάθε κυκλώματος να παραμένουν οι ίδιες τιμές για το μήκος του καλωδίου πριν και μετά το βήμα αυτό.

Όσον αφορά τώρα το runtime για τον αλγόριθμο FBM μόνο, τόσο για το πρώτο κύκλωμα όσο και για τα υπόλοιπα τρία, μπορούμε εύκολα να παρατηρήσουμε ότι ο αλγόριθμος Timsort είχε τα πιο γρήγορα αποτελέσματα ως προς τον τελικό χρόνο εκτέλεσης του αλγορίθμου σε σχέση με τους άλλους δύο αλγορίθμους, αμέσως μετά ακολουθούσε ο Insertion sort με το να δίνει αποτελέσματα πιο αργά από τον Timsort αλλά και πιο γρήγορα από τον Bubble sort, με τον τελευταίο να είναι και ο πιο αργός από όλους. Κάτι τέτοιο ήταν ήδη αναμενόμενο, έχοντας δει ακριβώς το ίδιο φαινόμενο και στους πρώτους

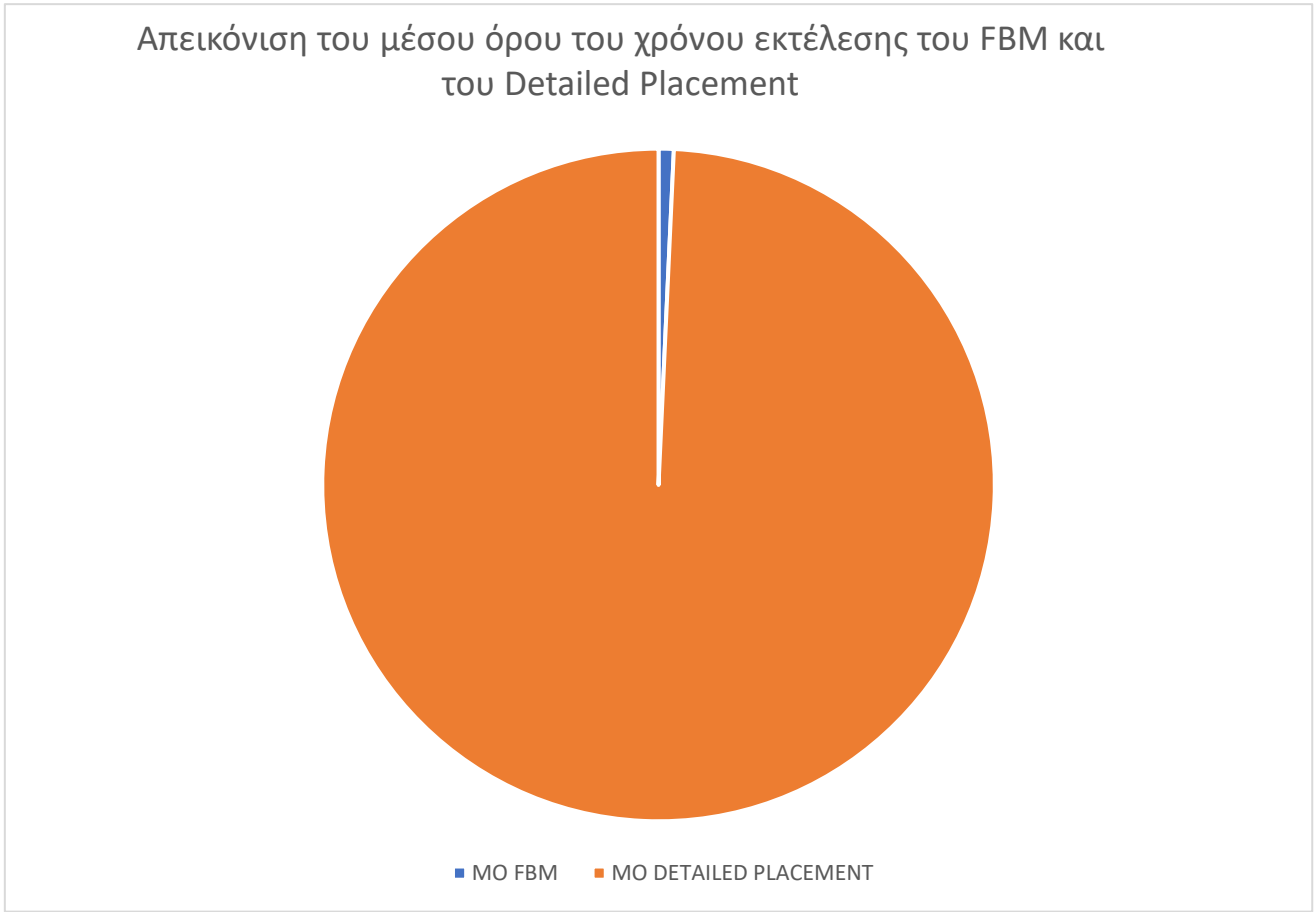
πειραματισμούς που έγιναν με αυτούς τους αλγορίθμους ταξινόμησης για όλα τα κυκλώματα. Επίσης για άλλη μια φορά έγινε φανερό ότι για κάθε έναν αλγόριθμο ταξινόμησης η χρήση της παράλληλης εκτέλεσης ήταν μια επιλογή που κόστισε πολύ περισσότερο χρονικά σε σχέση με την σειριακή εκτέλεση. Δεν ήταν λίγες οι φορές που η παράλληλη εκτέλεση οδήγησε σε εμφάνιση αποτελεσμάτων με πάνω από το διπλάσιο συνολικό χρόνο εκτέλεσης σε σχέση με την σειριακή, όπως για παράδειγμα στην περίπτωση του κυκλώματος s27 με τη χρήση της Insertion sort, όπου είχαμε καθυστέρηση σχεδόν 6 φορές περισσότερο από την σειριακή εκτέλεση, αλλά και στην περίπτωση της χρήσης του Bubble sort για το ίδιο κύκλωμα, με καθυστέρηση αυτή τη φορά περίπου 4 φορές περισσότερο από τη σειριακή. Οι διαφορές ανάμεσα στον χρόνο εκτέλεσης μεταξύ σειριακής και παράλληλης εκτέλεσης ελαχιστοποιήθηκαν στην περίπτωση του τέταρτου benchmark design(s208_1), ενώ μεγιστοποιήθηκαν στην περίπτωση του τρίτου κυκλώματος(s27). Το τελικό συμπέρασμα που μπορεί να εξαχθεί από όλα αυτά είναι ότι για τον αλγόριθμο FBM ο αλγόριθμος Timsort είναι αυτός που για κάθε κύκλωμα ελαχιστοποιεί τον χρόνο εκτέλεσης, ενώ ταυτόχρονα οι αρχικές συναρτήσεις που υλοποιούν την ανάγνωση των αρχείων εισόδου πρέπει να υλοποιηθούν με σειριακό τρόπο, καθώς με την παράλληλη εκτέλεση τους δεν υπάρχει κανένα επιπλέον όφελος, παρά μόνο επιβράδυνση του χρόνου εκτέλεσης, η οποία μπορεί όπως είδαμε να φτάσει σε αρκετά υψηλά επίπεδα.

Το runtime για το detailed placement τώρα δεν επηρεαζόταν χρονικά ούτε από τον εκάστοτε αλγόριθμο ταξινόμησης ούτε από την σειριακή-παράλληλη εκτέλεση, καθώς στο βήμα αυτό δεν υπήρχε η ανάγκη για ταξινόμηση από τη στιγμή που η κύρια λειτουργία του ήταν οι ανταλλαγές ανάμεσα σε cells, ενώ η σειριακή και η παράλληλη εκτέλεση υλοποιήθηκαν πολύ νωρίτερα και δεν ξαναχρησιμοποιήθηκαν έκτοτε για κάποια άλλη κατάσταση. Οι μόνοι παράγοντες από τους οποίους επηρεαζόταν το συγκεκριμένο βήμα ως προς το runtime ήταν ο αριθμός των nets αλλά και ο αριθμός των cells του κάθε κυκλώματος. Όσο πιο πολλά nets και πιο πολλά cells είχε το κύκλωμα, τόσο παραπάνω χρόνο έκανε αυτός ο αλγόριθμος να εκτελεστεί, καθώς τόσα παραπάνω στοιχεία είχε να συγκρίνει και να αντιμεταθέσει. Αυτό επαληθεύτηκε εμφανέστατα στην καταγραφή των τελικών αποτελεσμάτων, καθώς οι χαμηλότεροι χρόνοι σημειώθηκαν στο κύκλωμα s27 που ήταν και το μικρότερο, οι αμέσως μεγαλύτεροι στο b02, μετά στο s208_1 και τέλος στο b01. Αυτή είναι και η σειρά των κυκλωμάτων αν τα ταξινομήσουμε με βάση τα παραπάνω κριτήρια που περιγράψαμε από το μικρότερο στο μεγαλύτερο. Οι χρόνοι για το detailed placement ήταν οι ίδιοι και για τις έξι περιπτώσεις κάθε ενός κυκλώματος, χωρίς βέβαια να βγαίνουν ακριβώς οι ίδιες τιμές για όλες τις περιπτώσεις, καθώς ο χρόνος εκτέλεσης είχε μερικές αυξομειώσεις τρέχοντας επανειλημμένα τη συνάρτηση του detailed placement με τη χρήση της συνάρτησης time() της Python, αλλά γενικά όπως φαίνεται και στον πίνακα, οι αυξομειώσεις ήταν σε γενικές γραμμές αρκετά μικρές. Τέλος, η στήλη του Total runtime περιέχει το άθροισμα τόσο του runtime για τον αλγόριθμο FBM όσο και του runtime για το detailed placement.

4.3.β Γραφήματα

Μετά την επιτυχή ολοκλήρωση της διαδικασίας καταγραφής των τελικών αποτελεσμάτων και την δημιουργία του συγκεντρωτικού πίνακα ώστε να βρίσκονται σε μια πιο οργανωμένη δομή, μελετήθηκαν κάποιες επιπλέον τιμές-στατιστικά που προέκυπταν από όλα τα προηγούμενα συγκεντρωτικά στοιχεία, για ακόμα καλύτερη και βαθύτερη κατανόηση των προηγούμενων τιμών και την εξαγωγή επιπλέον συμπερασμάτων.

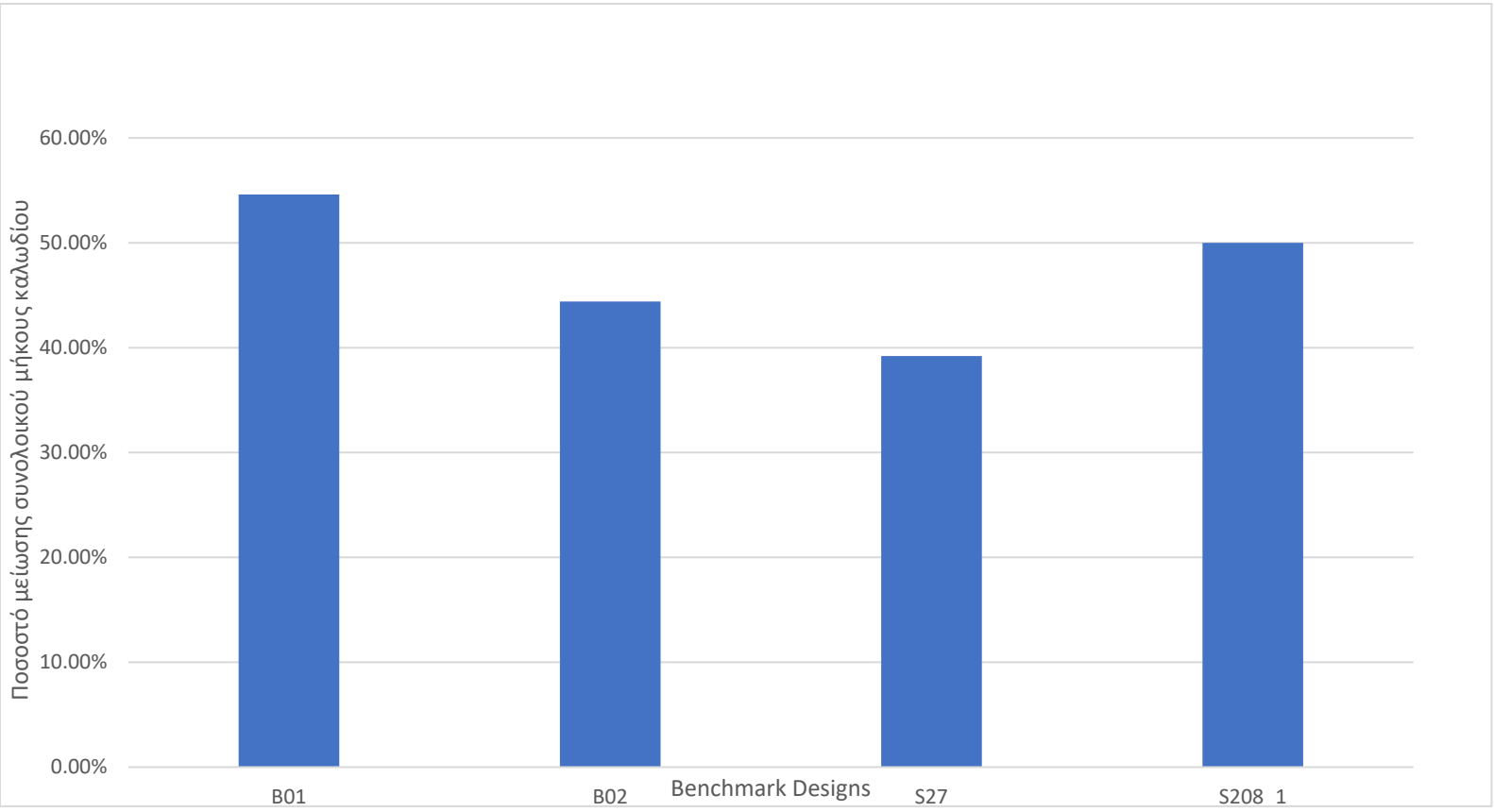
Το πρώτο γράφημα που δημιουργήθηκε σύγκρινε τους μέσους όρους των χρόνων εκτέλεσης του FBM και του detailed placement και για τα τέσσερα benchmark designs. Αφού λοιπόν υπολογίστηκαν με προσοχή οι αριθμητικές τιμές που απαιτούνταν για την δημιουργία του γραφήματος, και πιο συγκεκριμένα η τιμή 0,007425 για την περίπτωση του αλγορίθμου FBM και η τιμή 1,013 για την περίπτωση του detailed placement, παράχθηκε το διάγραμμα, το οποίο φαίνεται στην εικόνα 4.2.



Εικόνα 4.2 Το γράφημα των μέσων όρων των χρόνων εκτέλεσης του αλγορίθμων FBM και του detailed placement

Όπως είναι εμφανές, η διαδικασία του detailed placement ήταν μία πολύ πιο χρονοβόρα διαδικασία από ότι η εκτέλεση του αλγορίθμου FBM, κάτι που γίνεται ξεκάθαρο και παρατηρώντας τις τιμές του προηγούμενου συγκεντρωτικού πίνακα. Αντιπροσωπευτικά παραδείγματα αυτής της κατάστασης ήταν η περίπτωση του κυκλώματος b01, όπου οι χρόνοι εκτέλεσης του FBM και για τις έξι περιπτώσεις ήταν πολύ πιο κάτω από το 1 second, ενώ για την περίπτωση του detailed placement ήταν πάντα μεγαλύτεροι των 2 seconds, αλλά και η περίπτωση του s208_1, στην οποία το detailed placement ήταν πάνω από 1 second ως προς το χρόνο εκτέλεσης, με το runtime για τον FBM να κινείται ξανά αρκετά κάτω από το 1 second. Κυριότερος λόγος για τον οποίον το detailed placement αργούσε πολύ περισσότερο να ολοκληρωθεί σε σχέση με τον αλγόριθμο FBM, ήταν το ότι το detailed placement χρησιμοποιούσε τέσσερα εμφωλευμένα for loops κατά την υλοποίησή του, για να συγκρίνει δηλαδή κάθε ένα cell με όλα τα υπόλοιπα cells όλων των υπόλοιπων nets, ενώ ο FBM σε κάθε του συνάρτηση χρησιμοποιούσε συνήθως ένα ή δύο εμφωλευμένα for loops, με εξαίρεση μια φορά στο τέταρτο βήμα που χρησιμοποιήθηκαν τρία. Αν αυτό συνδυαστεί τώρα και με περιπτώσεις κυκλωμάτων που είχαν και μεγάλο αριθμό από cells αλλά και μεγάλο αριθμό από nets, τότε μπορεί κανείς εύκολα να καταλάβει ότι η εκτέλεση του detailed placement επιβραδύνεται ακόμα περισσότερο.

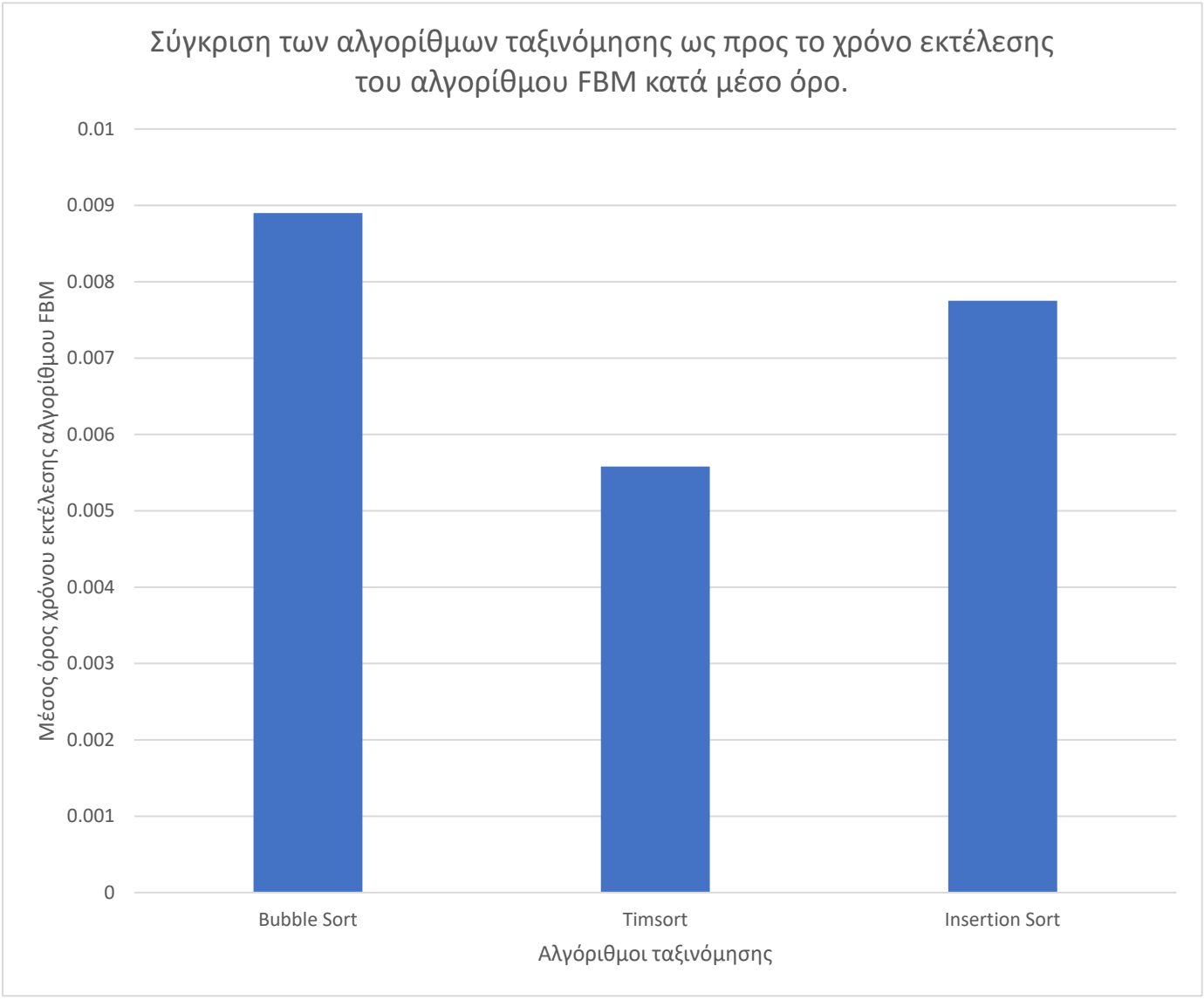
Στη συνέχεια, δημιουργήθηκε ένα διάγραμμα το οποίο απεικόνιζε την ποσοστιαία μεταβολή του συνολικού μήκους καλωδίου πριν και μετά την εφαρμογή του βήματος του detailed placement. Αφού λοιπόν πρώτα υπολογίστηκαν τα ποσοστά για κάθε ένα benchmark design και εκχωρήθηκαν στον πίνακα σε κατάλληλη στήλη, δημιουργήθηκε το παρακάτω διάγραμμα(εικόνα 4.3).



Εικόνα 4.3 Η ποσοστιαία μείωση του συνολικού μήκους καλωδίου πριν και μετά την εφαρμογή του detailed placement.

Εδώ παρατηρούμε, όπως σχολιάστηκε και νωρίτερα, ότι η μεγαλύτερη ελαχιστοποίηση ως προς το συνολικό μήκος του καλωδίου της σχεδίασης έγινε για την περίπτωση του κυκλώματος b01, καθώς όπως ήδη είδαμε για το συγκεκριμένο κύκλωμα το τελικό καλώδιο ήταν κατά 54,6% μικρότερο σε σχέση με πριν, για το κύκλωμα b02 είχαμε βελτίωση κατά 44,4%, για το s27 κατά 39,2% (η μικρότερη βελτιστοποίηση από όλες) και τέλος για το s208_1 κατά 50%.

Τέλος, υλοποιήθηκε ένα ακόμα διάγραμμα, το οποίο συγκρίνει τους τρεις αλγορίθμους ταξινόμησης που χρησιμοποιήθηκαν ως προς το μέσο όρο του χρόνου εκτέλεσης για τον αλγόριθμο FBM. Το διάγραμμα φαίνεται στην εικόνα 4.4.



Εικόνα 4.4 Σύγκριση των αλγορίθμων ταξινόμησης ως προς τον χρόνο εκτέλεσης του αλγορίθμου FBM κατά μέσο όρο.

Εδώ βλέπουμε ότι ο πιο γρήγορος αλγόριθμος για την εκτέλεση του FBM ήταν και πάλι με διαφορά ο Timsort με τιμή 0,00558. Ο δεύτερος πιο γρήγορος ήταν ο Insertion sort, με τιμή 0,00775 και τελευταίος ως προς το runtime κατά μέσο όρο είναι ο Bubble sort, με τιμή 0,0089. Ο Timsort, όντας ο πιο γρήγορος αλγόριθμος ταξινόμησης για κάθε κύκλωμα, τόσο για την παράλληλη όσο και για τη σειριακή εκτέλεση, ήταν αναμενόμενο να είναι ο πιο γρήγορος από τους τρεις και κατά μέσο όρο. Με παρόμοια σκεπτικό, ο Insertion και ο Bubble sort, με το να είναι ο δεύτερος και ο τρίτος πιο γρήγορος αλγόριθμος αντίστοιχα για κάθε κύκλωμα, παρέμειναν στις ίδιες «θέσεις» και κατά την αξιολόγησή τους ως προς το runtime κατά μέσο όρο.

ΚΕΦΑΛΑΙΟ 5 Συμπεράσματα

Η διαδικασία σχεδίασης ενός ASIC, όπως είδαμε, είναι μια ιδιαίτερα δύσκολη και χρονοβόρα διαδικασία που απαιτεί ιδιαίτερη προσοχή και έξυπνους χειρισμούς κατά την διάρκεια κάθε ενός βήματος, ούτως ώστε να διασφαλιστεί η ποιότητα και η αξιοπιστία του τελικού κυκλώματος που θα παραχθεί. Για την γρήγορη, σωστή και αποτελεσματική λειτουργία των βημάτων της σχεδιαστικής πορείας έχει αναπτυχθεί μια μεγάλη ποικιλία αλγορίθμων, και ένας από αυτούς είναι και ο αλγόριθμος FBM. Ήταν ένας αρκετά απλός και ταυτόχρονα γρήγορος αλγόριθμος, που φτιάχτηκε έχοντας ως βάση του τον αλγόριθμο Abacus, αλλά βελτιώνοντας τον σε πολύ σημαντικό βαθμό περιορίζοντας τις άσκοπες μετακινήσεις των κελιών. Ο FBM αποδείχθηκε ότι ήταν ιδιαίτερα αποδοτικός για περιπτώσεις κυκλωμάτων τα οποία είχαν γενικά υψηλότερη πυκνότητα, παράγοντας αποτελέσματα σε αρκετά γρήγορο χρονικό διάστημα, ενώ ένα από τα αδύναμα σημεία του ήταν το Average Displacement. Πάνω σε αυτό τον αλγόριθμο υλοποιήθηκαν ορισμένες τροποποιήσεις και μία προσθήκη, με τις τροποποιήσεις να έχουν να κάνουν με την επιλογή του αποδοτικότερου αλγορίθμου ταξινόμησης για τις περιπτώσεις που ο αλγόριθμος έπρεπε να ταξινομήσει τα cells του κυκλώματος, με την εφαρμογή παράλληλης εκτέλεσης αντί για σειριακή κατά την ανάγνωση των δεδομένων εισόδου, όπως και με την τροποποίηση της πυκνότητας των δοσμένων κυκλωμάτων για τις ανάγκες της υλοποίησης, με την τελική τιμή της μέγιστης πυκνότητας κατά μέσο όρο για την οποία ο αλγόριθμος ήταν πλήρως λειτουργικός να είναι η 0,81894, και με την προσθήκη να είναι η υλοποίηση μιας λειτουργίας βελτιστοποίησης του αλγορίθμου πάνω στο υπάρχων κύκλωμα. Αφού όλα τα παραπάνω υλοποιήθηκαν και υπήρξε καταγραφή και προσεκτική μελέτη των τελικών αποτελεσμάτων, έγινε αντιληπτό ότι ο βέλτιστος αλγόριθμος ταξινόμησης για τον συγκεκριμένο αλγόριθμο ήταν ο Timsort, καθώς ήταν με διαφορά ο πιο γρήγορος αλγόριθμος για κάθε κύκλωμα πάνω στο οποίο δοκιμάστηκε και ότι η σειριακή εκτέλεση ήταν πολύ γρηγορότερη σε σχέση με την παράλληλη, κυρίως γιατί οι συναρτήσεις στις οποίες αυτά τα δύο είδη εκτέλεσης εφαρμόστηκαν εκτελούσαν αρκετά απλές και γρήγορες λειτουργίες. Το συμπέρασμα που εξάχθηκε σχετικά με τον αλγόριθμο που χρησιμοποιήθηκε για την βελτιστοποίηση η οποία είχε να κάνει με τη μείωση του συνολικού μήκους καλωδίου ήταν ότι υπήρξε αρκετά αποτελεσματικός, καθώς τα τελικά ποσοστά μείωσης ήταν σε καλό επίπεδο, αλλά υπήρξε πολύ πιο χρονοβόρα διαδικασία σε σχέση με τον αλγόριθμο FBM.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Dasygenis, M., & Soudris, D. (2015). *Ενσωματωμένα συστήματα* [Undergraduate textbook]. Kallipos, Open Academic Editions. https://dx.doi.org/10.57713/kallipos-789

[2] https://www.chipverify.com/verilog/asic-soc-chip-design-flow

[3] https://www.arm.com/glossary/asic

[4] https://www.maven-silicon.com/blog/what-is-synthesis-in-vlsi/

[5] https://vlsitalks.com/physical-design/synthesis/

[6] https://www.maven-silicon.com/blog/what-is-dft-in-vlsi/

[7] https://www.synopsys.com/glossary/what-is-equivalence-checking.html

[8] https://www.synopsys.com/glossary/what-is-static-timing-analysis.html

[9] https://asic-soc.blogspot.com/2008/08/dynamic-vs-static-timing-analysis.html

[10] https://hardwarebee.com/WikiBee/dynamic-power

[11] https://www.techsimplifiedtv.in/2024/08/what-is-partitioning-in-physical-design.html

[12] https://www.vlsisystemdesign.com/floorplanning/

[13] https://www.physicaldesign4u.com/2019/12/floorplanning-floor-planning-is-art-of.html

[14] J. Ferreira, P. F. Butzen, C. Meinhardt and R. A. L. Reis, "FBM: A Simple and Fast Algorithm for Placement Legalization," *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genoa, Italy, 2019, pp. 209-212, doi: 10.1109/ICECS46596.2019.8965013.

[15] https://ivlsi.com/refine-placement-detailed-placement-vlsi-physical-design/

[16] https://chipedge.com/what-is-clock-tree-synthesis

[17] Shahookar, K. & Mazumder, P.. (1991). VLSI placement techniques. ACM Computing Surveys (CSUR). 23. 143-220. 10.1145/103724.103725.

[18] https://www.vlsisystemdesign.com/maze-routing-lees-algorithm/

[19] https://chipedge.com/the-importance-of-vlsi-physical-verification-in-chip-design/

[20] https://www.disher.com/blog/design-for-manufacturing/

[21] https://layouteditor.org/layout/file-formats/gdsii

[22] https://vlsiweb.com/global-routing-vs-detailed-routing/

[23] https://chipedge.com/what-is-routing-in-vlsi-physical-design/

[24] https://ivlsi.com/global-placement-vlsi-physical-design/

[25] U. Brenner, "VLSI legalization with minimum perturbation by iterative augmentation," *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2012, pp. 1385-1390, doi: 10.1109/DATE.2012.6176579.
keywords: {Algorithm design and analysis;Law;Marine vehicles;Timing;Benchmark testing;Optimization},

[26] P. Oikonomou, A. N. Dadaliaris, T. Loukopoulos, A. Kakarountas and G. I. Stamoulis, "A Tetris-based legalization heuristic for standard cell placement with obstacles," *2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, Thessaloniki, Greece, 2018, pp. 1-4, doi: 10.1109/MOCAST.2018.8376559.
keywords: {Standards;Circuits and systems;Measurement;Iterative algorithms;Heuristic algorithms;Dynamic programming;Computational modeling;legalization;obstacles;standard cell placement;Tetris;heuristic},

[27] J. C. Puget, G. Flach, R. Reis and M. Johann, "Jezz: An effective legalization algorithm for minimum displacement," *2015 28th Symposium on Integrated Circuits and Systems Design (SBCCI)*, Salvador, Brazil, 2015, pp. 1-5.
keywords: {Microelectronics;Algorithm design and analysis;Design automation;Physical design;Servers;Joining processes;Law;Legalization;Placement;Standard-cell;Microelectronics},

[28] https://www.chrislaux.com/timsort

[29] https://www.tpointtech.com/bubble-sort

[30] https://www.tpointtech.com/insertion-sort

[31] https://www.baeldung.com/cs/timsort

[32] https://www.tpointtech.com/tim-sort

[33] https://www.chrislaux.com/mergesort

[34] https://www.tpointtech.com/tim-sort

[35] https://www.geeksforgeeks.org/difference-between-sequential-and-parallel-computing/

[36] https://www.starburst.io/blog/parallel-vs-sequential-processing/