
**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ**
**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**
**ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ**
**ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ**
**«ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ**
**ΒΙΟΙΑΤΡΙΚΗ»**


# ΑΝΑΠΤΥΞΗ ΡΟΗΣ ΛΕΠΤΟΜΕΡΟΥΣ ΧΩΡΟΘΕΤΗΣΗΣ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**


**Υπεύθυνος**

**ΣΤΑΜΟΥΛΗΣ ΓΕΩΡΓΙΟΣ**


**Λαμία, Ιούλιος, 2016**


**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ**
**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**
**ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ**
**«ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ»**


# ΑΝΑΠΤΥΞΗ ΡΟΗΣ ΛΕΠΤΟΜΕΡΟΥΣ ΧΩΡΟΘΕΤΗΣΗΣ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

**Επιβλέπων**

**ΣΤΑΜΟΥΛΗΣ ΓΕΩΡΓΙΟΣ**

**Λαμία, Ιούλιος, 2016**

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο [«ΑΝΑΠΤΥΞΗ ΡΟΗΣ ΛΕΠΤΟΜΕΡΟΥΣ ΧΩΡΟΘΕΤΗΣΗΣ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ»] αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Η ΔΗΛΟΥΣΑ

20/07/2016

**ΑΝΑΠΤΥΞΗ ΡΟΗΣ ΛΕΠΤΟΜΕΡΟΥΣ ΧΩΡΟΘΕΤΗΣΗΣ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ**

**ΚΟΚΚΑΡΗ ΣΤΑΜΑΤΙΑ**

## Τριμελής Επιτροπή:

Σταμούλης Γεώργιος(επιβλέπων)

Δαδαλιάρης Αντώνιος

Κοζύρη Μαρία

# Abstract

Legalization is a very important step in the physical design for very large scale integration integrated circuits (VLSI). Most placement techniques produce a layout with some violations. Legalization eliminates these violations. The chip components should be aligned with the lines in which the chip is divided, so that they can be connected to the cable, there will be no overlapping among the items and finally the last will be within the limits of the chip.

The legalization process should be done in such a way that the initial disposition of the elements is changed, the least possible. For this purpose, there are different algorithms in the bibliography, where each one approaches the problem differently. However, the objective is the same; to have the least possible displacement and the cable length not to be greatly increased, in due time.

For the implementation in C language, "Abacus: Fast legalization of standard cell circuits with minimal movement" was chosen. The reason why it was chosen is that many algorithms are based on Abacus, or refer to it. Apart from Abacus, which is analyzed and the results of its execution are presented, other algorithms, that have a different approach to the problem, are presented too, after a brief description of the physical design and placement of integrated circuits has been done.

# Περίληψη

Η νομιμοποίηση της θέσης των στοιχείων ενός chip είναι ένα πολύ σημαντικό βήμα της φυσικής σχεδίασης για τα πολύ μεγάλης κλίμακας ολοκλήρωσης, ολοκληρωμένα κυκλώματα, VLSI (Very Large Scale Integration). Οι περισσότερες τεχνικές χωροθέτησης παράγουν ένα layout με κάποιες παραβιάσεις. Η νομιμοποίηση της θέσης των στοιχείων εξαλείφει αυτές τις παραβιάσεις. Τα στοιχεία του chip πρέπει να ευθυγραμμιστούν με τις γραμμές που είναι χωρισμένο το chip, ώστε να μπορούν να συνδεθούν με το καλώδιο σύνδεσης, να μην υπάρχουν επικαλύψεις μεταξύ των στοιχείων και τέλος τα τελευταία να είναι εντός των ορίων του chip.

Η διαδικασία νομιμοποίησης θα πρέπει να γίνει με τέτοιο τρόπο ώστε να αλλάξει η αρχική τοποθέτηση των στοιχείων, όσο το δυνατόν λιγότερο. Για αυτό το σκοπό στην βιβλιογραφία υπάρχουν διαφορετικοί αλγόριθμοι, όπου ο καθένας προσεγγίζει το πρόβλημα διαφορετικά. Ο αντικειμενικός σκοπός όμως είναι ο ίδιος. Να υπάρχει όσο το δυνατόν λιγότερο displacement και να μην αυξάνεται κατά πολύ το μήκος καλωδίου, σε εύθετο χρόνο.

Για την υλοποίηση σε γλώσσα C επιλέχθηκε ο "*Abacus: Fast legalization of standard cell circuits with minimal movement*" . Ο λόγος που επιλέχθηκε, είναι ότι πολλοί αλγόριθμοι έχουν βάση τον Abacus ή αναφέρονται σε αυτόν. Εκτός από τον *Abacus* ο οποίος αναλύεται και παρουσιάζονται τα αποτελέσματα εκτέλεσης του, παρουσιάζονται και άλλοι αλγόριθμοι, που έχουν μια διαφορετική προσέγγιση του προβλήματος, αφού γίνει πρώτα μια σύντομη περιγραφή της φυσικής σχεδίασης και της χωροθέτησης ολοκληρωμένων κυκλωμάτων.

# *ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ*

# *ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ*

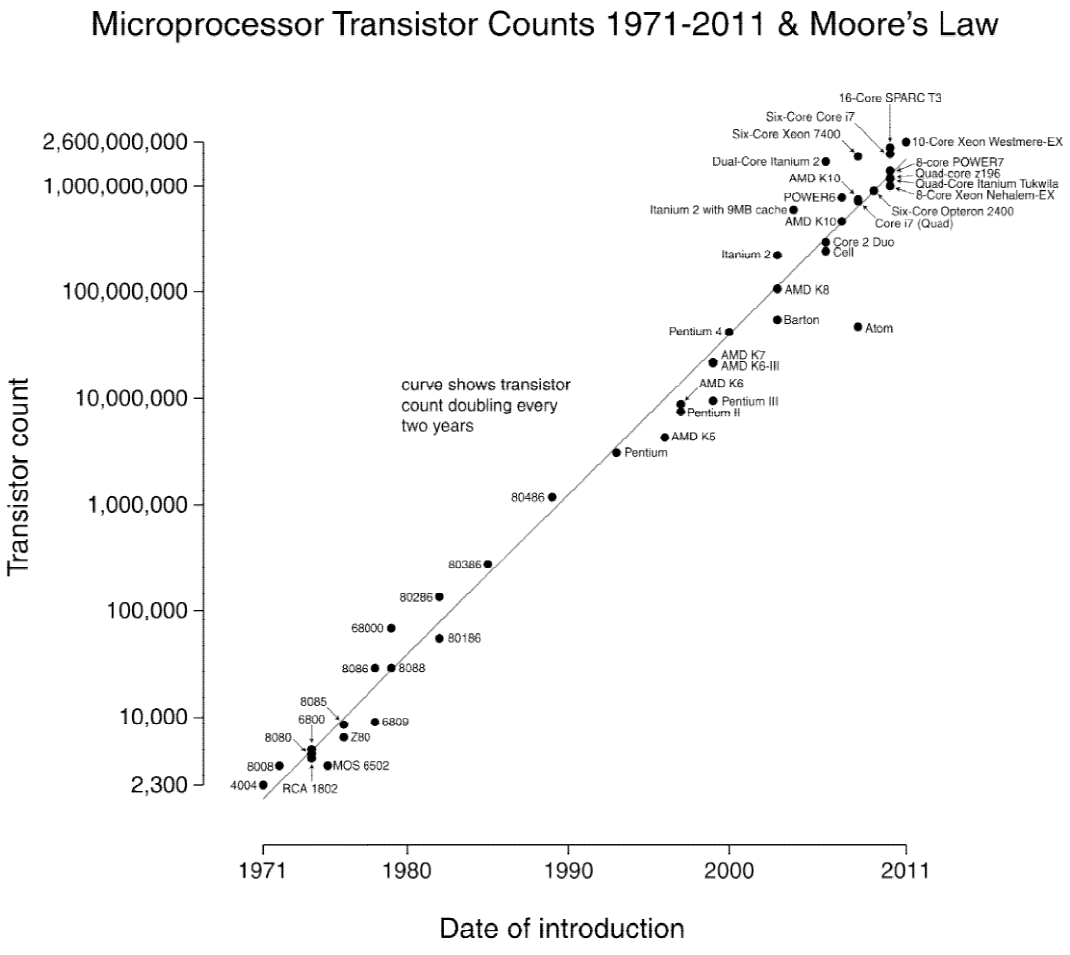
# *ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ*

# *ΚΑΤΑΛΟΓΟΣ ΑΛΓΟΡΙΘΜΩΝ*

# *ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ*

# 1 Εισαγωγή.

Η τεχνογνωσία για τη μαζική παραγωγή ολοκληρωμένων κυκλωμάτων ξεκίνησε την δεκαετία του 70 με διάφορα πρακτικά προβλήματα, όπως η σύνδεση των στοιχείων του ολοκληρωμένου. Με την εφεύρεση μιας μεθόδου για τη μηχανοποιημένη προσθήκη μικρών μεταλλικών καλωδίων, ξεκίνησε η μαζική παραγωγή των ολοκληρωμένων κυκλωμάτων. Η εξέλιξη των ολοκληρωμένων κυκλωμάτων ακολουθεί τον νόμο του Moore : το πλήθος των κρυσταλλοτριόδων ανά τετραγωνική ίντσα ενός ολοκληρωμένου κυκλώματος διπλασιάζεται κάθε 18 μήνες *(Εικόνα 1.1)* . Έτσι ο πρώτος επεξεργαστής της Intel ο 4004 είχε 2300 τρανζίστορ και σήμερα ο i7 έχει 1,170,000,000.




***Εικόνα 1.1 : Νόμος του Moore***

*Πηγή: wikipedia*

Το πλήθος των τρανζίστορ που χρησιμοποιούνται σήμερα στα πολύ μεγάλης κλίμακας ολοκλήρωσης, ολοκληρωμένα κυκλώματα, VLSI (Very Large Scale Integration), είναι τρομακτικά μεγάλος. Αυτό συμβαίνει κυρίως διότι τα τρανζίστορ, που είναι ο πυρήνας των ολοκληρωμένων, με την εξέλιξη της τεχνολογίας έχουν γίνει γρηγορότερα κι αυτό επηρεάζει σημαντικά, την απόδοση των ολοκληρωμένων κυκλωμάτων. Με αποτέλεσμα την τελευταία δεκαετία ακόμα και προσωπικοί υπολογιστές να δουλεύουν με παραλληλισμό, αφού επεξεργαστές για προσωπικούς υπολογιστές όπως ο I7 της INTEL, έχει 10 πυρήνες και ο INTEL XEON , 64 πυρήνες.

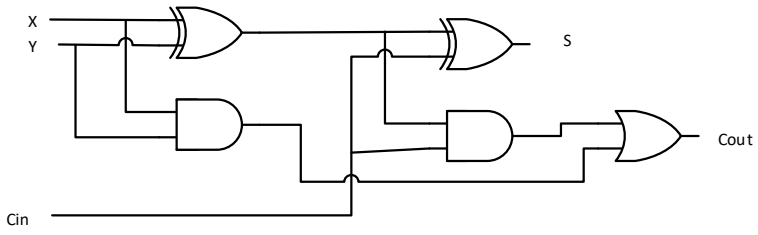
Ο σχεδιασμός λοιπόν των VLSI είναι πολύπλοκος και για την παραγωγή οικονομικά βιώσιμων προϊόντων VLSI, σε λογικά χρονικά περιθώρια, έχει αναπτυχθεί μια καλά σχεδιασμένη μεθοδολογία. Εκατοντάδες άτομα, μέλη μιας ομάδας που δραστηριοποιούνται σε ένα στάδιο της μεθοδολογίας, μπορεί να συμμετέχουν στην παραγωγή προϊόντων VLSI. Κάποια άτομα ασχολούνται με το front-end design και κάποια άλλα με τα back-end design. Αυτή η διπλωματική ασχολείται με την νομιμοποίηση (legalization) της θέσης των στοιχείων του chip. Ένα στοιχείο μπορεί να είναι τρανζίστορ, buffers, ALU κτλ. Η διαδικασία της νομιμοποίησης προϋποθέτει την τοποθέτηση των στοιχείων, σε κάποιες θέσεις με κάποια κριτήρια, από κάποιον αλγόριθμο τοποθέτησης. Και η νομιμοποίηση και η χωροθέτηση των στοιχείων του chip είναι δύο πολύ σημαντικά βήματα στην φυσική σχεδίαση των VLSI, που ανήκει στο κομμάτι του back- end design.

Στην χωροθέτηση των στοιχείων του chip, οι θέσεις, αυτών μπορεί να παραβαίνουν τα όρια του chip και το πιο συνηθισμένο είναι να υπάρχουν επικαλύψεις αυτών των στοιχείων, με αποτέλεσμα η κατασκευή του chip να είναι αδύνατη. Το κάθε στοιχείο πρέπει να καταλαμβάνει μια ευθυγραμμισμένη διακριτή, θέση πάνω στις γραμμές που θα περάσει το καλώδιο σύνδεσης των στοιχείων.

Αυτή είναι η νομιμοποίηση της θέσης των στοιχείων, Legalization και ένας από του αλγόριθμους που χρησιμοποιείται κατά κόρον, αλλά στην βιβλιογραφία υπάρχουν και πολλές παραλλαγές αυτού είναι ο *“Abacus: Fast legalization of standard cell circuits with minimal movement”*. Ο οποίος και υλοποιήθηκε σε γλώσσα C και συγκρίνεται με τον αλγόριθμο Tetris : “*Method and system for high speed detailed placement of cells within an integrated circuit design” [23]*, ο οποίος και αυτός αναφέρεται πολύ συχνά στην βιβλιογραφία.

## 1.1 Ροή σχεδίασης ολοκληρωμένων κυκλωμάτων.

Μια πράξη πρόσθεσης με δύο bits απαιτεί το παρακάτω λογικό κύκλωμα του πλήρη αθροιστή. Ένας 4-bit πλήρης αθροιστής απαιτεί να επαναληφθεί 4 φορές ένας πλήρης αθροιστής.



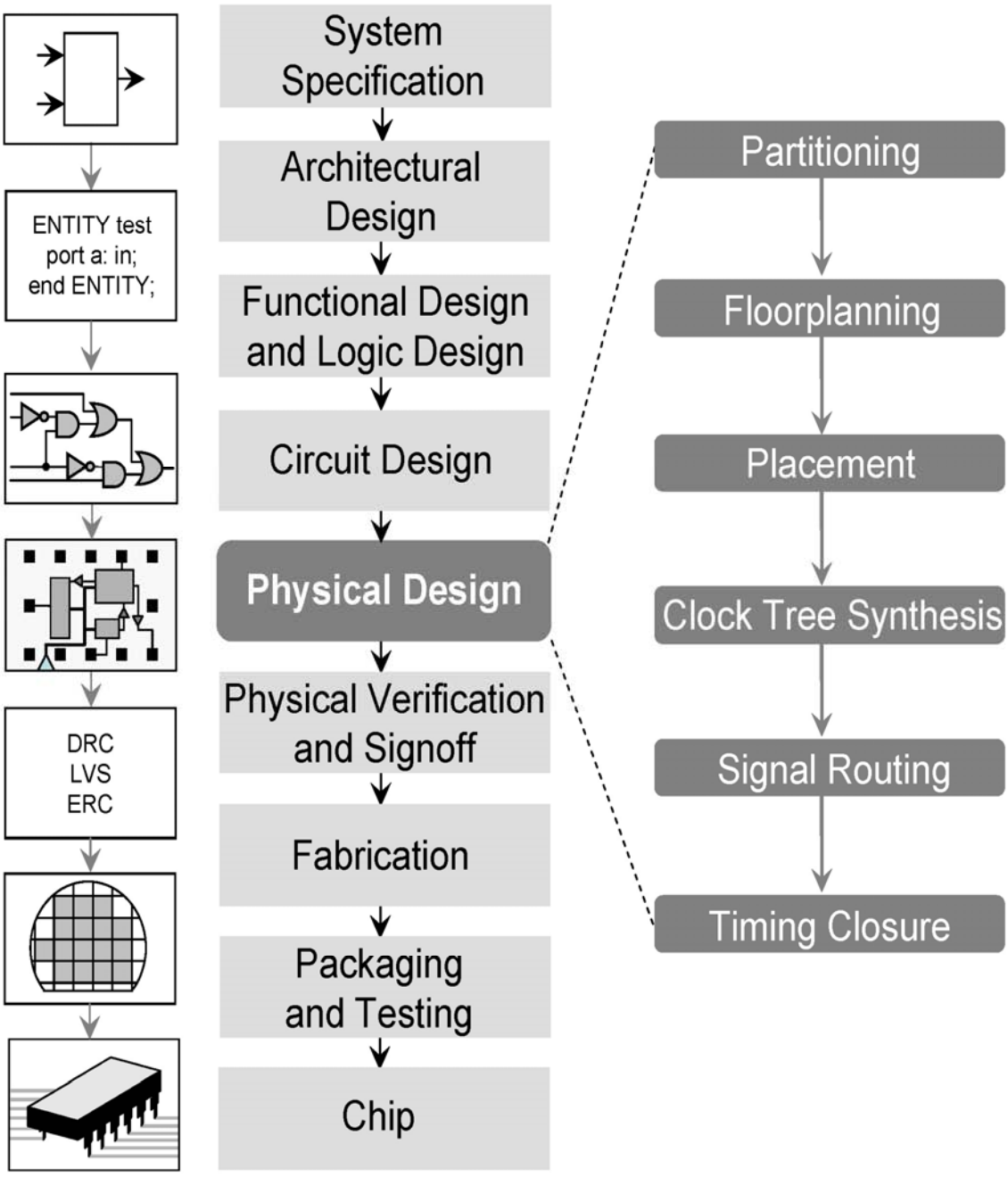

***Σχήμα 1.1 :*** Full adder

Αν τώρα αναλογιστούμε ότι κάθε πύλη είναι ένα τρανζίστορ σε ένα ολοκληρωμένο κύκλωμα, είναι φανερό ότι για πιο πολύπλοκα συστήματα όπως είναι μια μνήμη ένας επεξεργαστής, θα χρειαστούν εκατοντάδες, χιλιάδες ακόμα και εκατομμύρια τρανζίστορ.

Ο αριθμός είναι πάρα πολύ μεγάλος και είναι δύσκολα διαχειρίσιμος. Για αυτό κι έχουν αναπτυχθεί τεχνικές, για την κατασκευή των VLSI. Επειδή η κατασκευή ενός ολοκληρωμένου κοστίζει, όταν πια το ολοκληρωμένο είναι στη τελική φάση της κατασκευής του, οι σχεδιαστές θα πρέπει να έχουν κάνει το καλύτερο δυνατόν. Για αυτό ακολουθείται μια ροή σχεδίασης ολοκληρωμένων κυκλωμάτων, που χονδρικά χωρίζεται στο front-end design και στο back-end design.

Στο front-end design η λύση του προβλήματος για ένα δεδομένο πρόβλημα, που εξαρτάται από το τι θέλουμε να φτιάξουμε, μετατρέπεται σε μια περιγραφή κυκλώματος. Στο back-end design μεταμορφώνεται η περιγραφή του κυκλώματος σε ένα φυσικό σχεδιασμό, που αποτελείται από τις πύλες και τις διασυνδέσεις τους. Μια από τις κύριες διαδικασίες του back-end design είναι η φυσική σχεδίαση.

Η διαδικασία σχεδίασης VLSI είναι αρκετά πολύπλοκη και μπορεί να χωριστεί σε κάποια βήματα *(εικόνα 2)*.




***Σχήμα 1.2 :*** Ροή σχεδίασης ολοκληρωμένων κυκλωμάτων
*Πηγή: wikipedia*

**System specification:** Για να κατασκευαστεί ένα ολοκληρωμένο κύκλωμα το πρώτο πράγμα που πρέπει να έχουμε είναι η έμπνευση για το τι θέλουμε να φτιάξουμε. Αρχικά αποφασίζεται ποια θα είναι ακριβώς η λειτουργία του chip, ποια η επίδοσή

του, ποιες οι φυσικές διαστάσεις του, ποια θα είναι η τεχνολογία παραγωγής και έτσι δημιουργούνται οι προδιαγραφές του ολοκληρωμένου.

**Architectural design:** Καθορίζεται η βασική αρχιτεκτονική στηριζόμενη στις προδιαγραφές συστήματος. Σε αυτό το βήμα θα πρέπει να καθοριστεί τι είδους μνήμη θα χρησιμοποιηθεί, ποια πρωτόκολλα θα χρησιμοποιήσει για την εσωτερική και εξωτερική, επικοινωνία , ποια συστατικά θα το απαρτίζουν, ποιες θα είναι οι ανάγκες του σε ρεύμα. Φτιάχνοντας απλά μοντέλα γίνεται μια πρώτη προσομοίωση χρησιμοποιώντας γλώσσες υψηλού επιπέδου όπως η C++ ή ακόμα και το MATLAB.

**Functional and logic design:** Στο Functional design ορίζεται η λειτουργικότητα και η συνδεσιμότητα, η συμπεριφορά, η είσοδοι οι έξοδοι και οι χρονισμοί των συστατικών του ολοκληρωμένου. Το logic design εκτελείται στο επίπεδο register-transfer (RTL) χρησιμοποιώντας, μια γλώσσα περιγραφής υλικού όπως είναι η VHDL και η Verilog. Αφού λοιπόν έχουμε την προσομοίωση του κυκλώματος από τις περιγραφικές γλώσσες το επόμενο βήμα είναι η σύνθεση. Εργαλεία logic synthesis μετατρέπουν το κύκλωμα από την περιγραφική γλώσσα σε επίπεδο κυκλώματος, εξάγοντας το netlist το οποίο είναι μια λίστα με τα cells που θα χρησιμοποιηθούν και τα nets που ανήκουν αυτά.

**Circuit design:** Κάποια από τα συστατικά του κυκλώματος σε επίπεδο τρανζίστορ, δημιουργούνται σε αυτό το βήμα, όπως για παράδειγμα είναι τα στατικά block μνήμης, πολλαπλασιαστές και κυκλώματα αποφόρτισης για την προστασία του ολοκληρωμένου. Η ορθότητα της σχεδίασης σε επίπεδο κυκλώματος, επαληθεύεται από εργαλεία προσομοίωσης όπως είναι το SPICE.

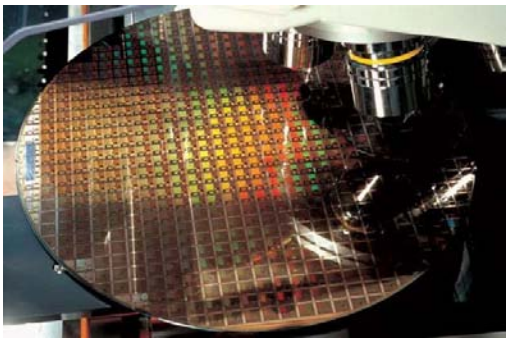
**Physical design** Το επίπεδο φυσικής σχεδίασης ανήκει στο κομμάτι back-end της ροής, είναι το βήμα όπου όλα τα συστατικά του chip παίρνουν μια φυσική, χωρική υπόσταση με τις κατάλληλες προϋποθέσεις για τις απαιτούμενες συνδέσεις δρομολόγησης. Το επίπεδο της φυσικής σχεδίασης αναλύεται σε επόμενο κεφάλαιο.

**Physical verification and Sign off:** Η επαλήθευση ορθότητας είναι αναγκαία πριν την κατασκευή του chip. Δημιουργούνται test-bench, για να εκτελεστούν προσομοιώσεις, ώστε να επαληθευτεί η λειτουργικότητα του. Έπειτα ακολουθεί το Static Timing Analysis στο οποίο γίνεται προσομοίωση του chip σε επίπεδο τρανσίστορ και στο Power Analysis καθορίζεται η κατανάλωση ισχύος του chip που σχεδιάστηκε.

**Fabrication:** Έχουμε την πρώτη κατασκευή wafer από πυρίτιο *(Εικόνα 1.2)*

**Packaging and Testing:** Γίνονται δοκιμές για να ελεγχθεί η αν το chip λειτουργεί σωστά και δεν υπάρχουν λάθη στη σχεδίαση ή στην κατασκευή του.

**Chip:** Αφού ολοκληρωθούν όλες οι δοκιμές και το chip δουλεύει σωστά και αποδοτικά, τότε είναι έτοιμο για την μαζική παραγωγή και την διάθεσή του στον πελάτη.



***Εικόνα 1.2 :*** wafer

***Πηγή:*** http://www.engadget.com/2013/01/18/tsmc-28nm-process-2013/

# 2 Φυσική σχεδίαση

Η φυσική σχεδίαση ανήκει στο back-end design και είναι ένα πολύ σημαντικό κομμάτι της όλης διαδικασίας. Τα δεδομένα που δέχεται σαν είσοδο είναι μια γενική λογική περιγραφή του κυκλώματος, με τη μορφή μιας λίστας από τα nets που απαρτίζουν το κύκλωμα. Κύριος στόχος της φυσικής σχεδίασης είναι η παραγωγή ενός layout, στο οποίο έχουν δοθεί γεωμετρικές συντεταγμένες στα στοιχεία του ολοκληρωμένου.

Η φυσική σχεδίαση είναι αρκετά πολύπλοκη διαδικασία, διότι πρέπει να χειριστεί έναν πολύ μεγάλο αριθμό από στοιχεία, αφού ένα ολοκληρωμένο κύκλωμα μπορεί να αναλυθεί σε πάνω από ένα εκατομμύριο πύλες. Λόγω της πολυπλοκότητάς της διαδικασίας, χωρίζεται στα παρακάτω προβλήματα τα οποία λύνονται σειριακά.

Αυτό το στάδιο περιλαμβάνει:

a) Logic partitioning

b) Floorplanning

c) Placement

d) Routing

e) Timing closure

**Logic partitioning**

Με την διαδικασία του partitioning, το κύκλωμα χωρίζεται σε μικρότερα κομμάτια, τα οποία είναι εύκολα διαχειρίσιμα, βελτιώνεται σημαντικά η απόδοση του chip, μειώνεται σημαντικά το κόστος του layout. Ο στόχος του partitioning είναι να πετύχει τον μικρότερο δυνατό αριθμό συνδέσεων μεταξύ των διαχωρισμένων στοιχείων (partitions).

**Floorplanning**

Το floorplaning είναι η διαδικασία εντοπισμού των δομών που θα πρέπει να τοποθετούνται κοντά μεταξύ τους, και η κατανομή χώρου θα πρέπει να γίνεται με τέτοιο τρόπο ώστε να πληρούνται αντικρουόμενοι στόχοι του διαθέσιμου χώρου

(κόστος του chip), τις απαιτούμενες επιδόσεις, και την επιθυμία να είναι όλα πάντα κοντά σε όλα τα άλλα.

Σε πολλές μεθοδολογίες σχεδίασης, γίνεται προσπάθεια να επιτευχθεί μια ισορροπία μεταξύ του μεγέθους και της ταχύτητας, τα οποία όμως είναι μεταξύ τους ασυμβίβαστα. Αυτό συμβαίνει διότι υπάρχουν περιορισμένοι πόροι δρομολόγησης και όσο πιο πολλοί πόροι υπάρχουν τόσο πιο αργό θα γίνεται.

Γίνεται προσπάθεια ελαχιστοποίησης του εμβαδού, η οποία επιτρέπει, να χρησιμοποιούνται λιγότεροι πόροι, και τα τμήματα του σχεδίου να είναι πιο κοντά. Αυτό οδηγεί σε μικρότερες αποστάσεις διασύνδεσης, λιγότερους πόρους δρομολόγησης.

**Placement**

Είναι η διαδικασία χωροθέτησης των κελιών στις βέλτιστες θέσεις σε προκαθορισμένο χώρο. Αναλύεται σε επόμενο κεφάλαιο

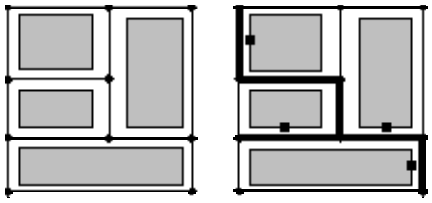
**Clock tree synthesis**

Σε αυτό το βήμα μειώνεται το insertion delay και το clock skew , η μέγιστη διαφορά στο χρόνο άφιξης ενός σήματος ρολογιού σε δύο διαφορετικά συστατικά, με αποτέλεσμα να μην χρησιμοποιείται το μεγάλο χρονικό διάστημα μεταξύ των παλμών ρολογιού, οπότε το σύστημα γίνεται πιο γρήγορο.

**Routing**

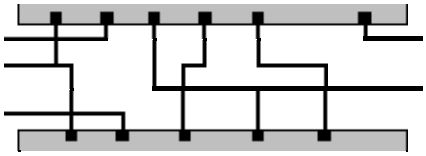
Είναι η διαδικασία τοποθέτησης καλωδίων τα οποία, συνδέουν τα τοποθετημένα κελιά. Το routing λόγω πολυπλοκότητας, χωρίζεται σε δύο φάσεις, το global routing και το detailed routing.

**Global routing:** Γίνεται μια επιφανειακή διασύνδεση των κελιών χωρίς να συγκεκριμενοποιείται το layout των καλωδίων.



***Σχήμα 2.1 :*** Global routing

**Detailed routing:** Είναι το τελικό στάδιο στο οποίο, βρίσκονται οι πραγματικές γεωμετρικές θέσεις του κάθε καλωδίου.



***Σχήμα 2.2 :*** Detailed routing

**Timing closure**

Είναι η διαδικασία όπου γίνονται κάποιες βελτιστοποιήσεις στο layout και κάποιες τροποποιήσεις στο netlist ώστε να ικανοποιηθούν οι χρονικοί περιορισμοί.

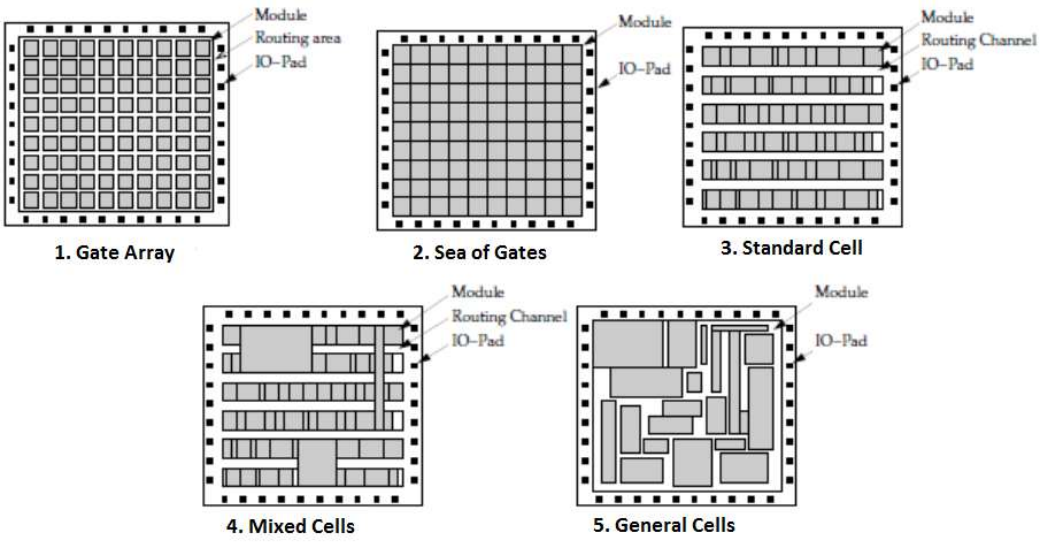
# 3 Χωροθέτηση ολοκληρωμένων κυκλωμάτων

Στο βήμα αυτό καθορίζεται η φυσική θέση των λογικών κυττάρων, πάνω στην επιφάνεια του chip έχοντας υπόψη το συνολικό μήκος καλωδίου που χρησιμοποιείται για τη σύνδεση των κελιών και την συνολική επιφάνεια που αυτά καταλαμβάνουν.

Τα κυκλώματα λογικής (πύλες και flip-flops) μετασχηματίζονται σε φυσικό σχέδιο (layout). Φυσικό σχέδιο (layout) είναι η γεωμετρία κάθε μάσκας κατασκευής του ολοκληρωμένου κυκλώματος

Το layout του chip χωρίζεται σε τρεις κατηγορίες.

1. Gate Array
2. Sea Of Gates
3. Standard Cell
4. Mixed Cell
5. General Cell ( Macros )




***Εικόνα 3.1 :*** Κατηγορίες layout

Στο placement τα κελιά θα πρέπει να τοποθετούνται έτσι ώστε, να μην υπάρχουν επικαλύψεις μεταξύ των κελιών, τα κελιά να μπορούν να συνδεθούν μέσω του καλωδίου με όλα τα nets, να είναι μέσα στα προκαθορισμένα όρια του chip και φυσικά να ελαχιστοποιεί το μήκος του καλωδίου.

Το πρόβλημα του placement είναι αρκετά πολύπλοκο και γι αυτό και χωρίζεται στα παρακάτω στάδια:

1. Global placement:
2. legalization:
3. Detailed placement:

**Global placement**: Η διαδικασία επαναλαμβάνεται μέχρι να παραχθεί ένα καλό αποτέλεσμα τοποθέτησης των κελιών, στο οποίο όμως υπάρχουν επικαλύψεις μεταξύ των κελιών.

**legalization:** (Νομιμοποίηση) : Το στάδιο του legalization παίρνει ως είσοδο το αποτέλεσμα του Global Placement και απαλείφει τις επικαλύψεις μεταξύ των ενοτήτων για να δημιουργήσει ένα «νόμιμο» (χωρίς επικαλύψεις) Placement.

**Detailed placement:** Σε αυτό το στάδιο γίνεται προσπάθεια βελτιστοποίησης τις θέσεις των κελιών, με γνώμονα τον κύριο στόχο του Placement, που τις περισσότερες φορές είναι το μήκος καλωδίου, διατηρώντας όμως τη νομιμοποίηση των κελιών.

## 3.1 Τεχνικές Χωροθέτησης ολοκληρωμένων κυκλωμάτων

Στο global placement τα κελιά τοποθετούνται έχοντας υπόψη το συνολικό μήκος καλωδίου που χρησιμοποιείται για τη σύνδεση των κελιών και την συνολική επιφάνεια που αυτά καταλαμβάνουν. Οι βασικές τεχνικές του global placement είναι:

**Simulated Annealing:** Το κύριο πλεονέκτημα είναι ότι οι αλγόριθμοι που βασίζονται σε αυτήν την τεχνική είναι heuristic, οπότε είναι πιο εύκολο να έχει σε συνδυασμό με τον στόχο του μικρού μήκους καλωδίου και άλλους στόχους όπως είναι για παράδειγμα, η κατανάλωση ρεύματος. Όμως αυτοί οι αλγόριθμοι είναι υπερβολικά αργοί, με αποτέλεσμα να μην μπορούν να χρησιμοποιηθούν για κυκλώματα με εκατομμύρια στοιχεία. Ένας γνωστός αλγόριθμος αυτής της κατηγορίας είναι ο TimberWolf [27].

**Αλγόριθμος 3.1.1**: TimberWolf

```
Algorithm Structure (j0, T0){
  /*
  * Given an initial state j0 and an
  * initial value for the parameter T,
  * T0,
  */
  T = T0;
  X = j0;
    while("stopping criterion" is not satisfied){
        while("inner loop criterion" is not satisified){
            j = generate(X);
            /*
            * generate is a function which
            * returns a new state j generated
            * incrementally from the previous state
            * X by a weighted random selection.
            */
            if (accept(c(j), c(X), T){
            X = j;
            }
        }
        T = update(T);
    }
}
```

**Top-down Partitioning:** Χρησιμοποιώντας την αναδρομή, το κύκλωμα διαιρείται σε μικρότερα κυκλώματα χρησιμοποιώντας ένα min-cut. Αλγόριθμος που χρησιμοποιεί αυτήν την τεχνική είναι ο Capo [28].

**Αλγόριθμος 3.1.2**: Capo

```
  Variables: queue of placement bins
  Initialize queue with top-level placement bin
1  While (queue not empty)
2    Dequeue a bin
3    If (bin has large/many macros or is marked as merged)
4     Cluster std-cells into soft macros
5     Use fixed-outline floorplanner to pack
       all macros (soft+hard)
6     If fixed-outline floorplanning succeeds
7      Fix macros and remove sites underneath the macros
8     Else
9      Undo one partition decision. Merge bin with sibling
10     Mark new bin as merged and enqueue
11   Else if (bin small enough)
12     Process end case
13   Else
14     Bi-partition the bin into smaller bins
15     Enqueue each child bin
```

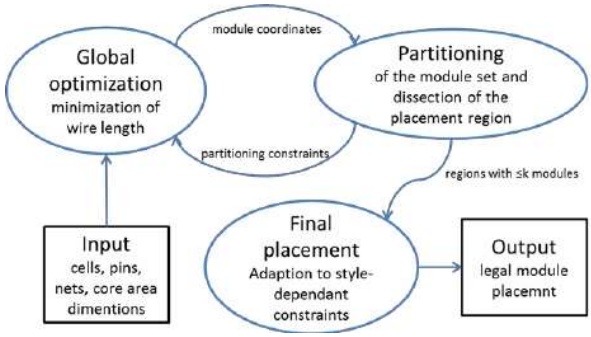
**Analytical Placement:** Και σε αυτήν την τεχνική το κύκλωμα διαιρείται σε μικρότερα κυκλώματα, όμως η τοποθέτηση βασίζεται σε μια συνάρτηση βελτιστοποίησης και δε χρησιμοποιείται το min-cut. Ανάλογα με την συνάρτηση η τεχνική αυτή χωρίζεται σε δύο υποκατηγορίες. Την μη γραμμική (Nonlinear) και την τετραγωνική (Quadratic) η οποία μπορεί να ελαχιστοποιηθεί με την επίλυση ενός συστήματος γραμμικών εξισώσεων. Σε αυτήν την κατηγορία ανήκει και ο Αλγόριθμος Gordian [26].

## 3.2 Gordian *"VLSI placement by quadratic programming and slicing optimization"* [26]

Το global placement που βασίστηκε η διπλωματική παράχθηκε με τον αλγόριθμο Gordian. Ο αλγόριθμος Gordian δημιουργεί αποκλειστικά ορθογώνιες περιοχές και πραγματοποιεί global optimization σε κάθε βήμα. Τα βήματα του αλγορίθμου είναι τα εξής:

1. Όλα τα στοιχεία του κυκλώματος τοποθετούνται, εντός της προβλεπόμενης περιοχής.
2. Δημιουργούνται ομάδες στοιχείων, με τις κατάλληλες μεθόδους partitioning, ενώ ακολουθούνται ρητά οι κανόνες καθολικής βελτιστοποίησης που έχουν τεθεί.
3. Όταν οι ομάδες στοιχείων που έχουν σχηματιστεί έχουν πλήθος στοιχείων μικρότερο από μια μεταβλητή, η τιμή της οποίας έχει προαποφασιστεί στην αρχή της εκτέλεσης του αλγορίθμου, τοποθετούνται καταλλήλως τα στοιχεία στον διαθέσιμο χώρο.

***Αλγόριθμος 3.2.1***: Gordian



# 4 Νομιμοποίηση χωροθετημένων κυκλωμάτων

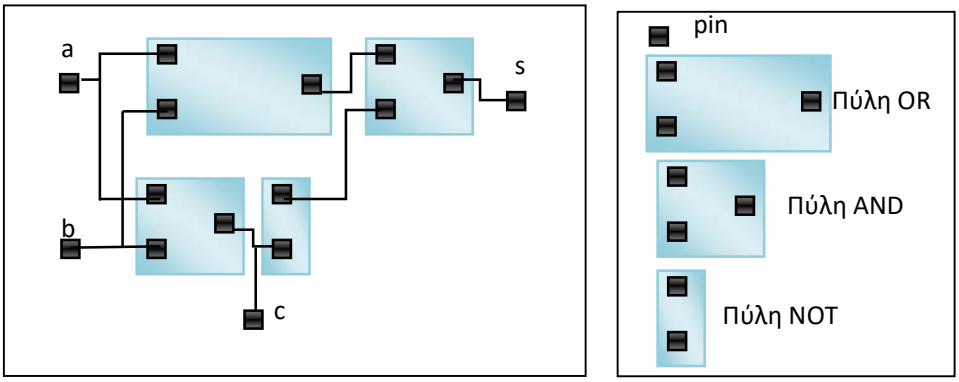
## 4.1 Νομιμοποίηση

Στο global placement έχουν ανατεθεί στα κελιά τύπου standard cells, αλλά και στα υπόλοιπα στοιχεία του κυκλώματος, οι γεωμετρικές θέσεις τους πάνω στο chip. Αυτές οι γεωμετρικές θέσεις δεν είναι ευθυγραμμισμένες με τις προκαθορισμένες γραμμές του chip και δεν είναι διακριτές με αποτέλεσμα να υπάρχουν επικαλύψεις μεταξύ των κελιών. Για αυτό το global placement, θα πρέπει να περάσει από την διαδικασία του legalization, για να νομιμοποιηθούν οι γεωμετρικές θέσεις ή όπως θα χρησιμοποιήσουμε παρακάτω, οι συντεταγμένες των κελιών.

Στο legalization λοιπόν, αναζητούνται νόμιμες συντεταγμένες για τα κελιά, χωρίς επικαλύψεις για όλα τα στοιχεία και η συνολική διαφορά των αρχικών συντεταγμένων των κελιών από τις τελικές συντεταγμένες, καλείται displacement.

Νόμιμες συντεταγμένες είναι αυτές που επιτρέπουν στο κελί να είναι ακριβώς στη γραμμή ούτος ώστε να μπορεί αργότερα στο routing να περάσει το καλώδιο σύνδεσης, να είναι εντός των ορίων του chip και φυσικά να μην υπάρχουν επικαλύψεις μεταξύ των κελιών.

Τα κελιά συνδέονται μεταξύ τους, με καλώδιο, μέσω των pins. Τα pins τοποθετούνται μέσα στα όρια του κελιού. Σε μερικές τεχνικές νομιμοποίησης, χρησιμοποιούνται ως κριτήριο. Η θέση τους μέσα στο κελί, θεωρείται δεδομένη και δεν αλλάζει από την διαδικασία του legalization. Στην εικόνα αναπαρίσταται το layout ενός half-adder, όπου φαίνονται καθαρά τα pins των κελιών, το διαφορετικό μέγεθος των κελιών και θα μπορούσαμε να πούμε ότι αυτό είναι ένα net, με είσοδο τα a,b και έξοδο τα s, c.




***Σχήμα 4.1 :*** Περιγραφή του layout

Το κάθε κελί ανήκει σε ένα ή περισσότερα nets. Οι τεχνικές χωροθέτησης έχουν στόχο, τα κελιά που ανήκουν στο ίδιο net, να τοποθετούνται το ένα δίπλα στο άλλο. Έτσι το καλώδιο που συνδέει τα pins των κελιών με τα κελιά και τα nets, να είναι όσο πιο μικρό επιτρέπεται, για να μην υπάρχουν καθυστερήσεις.

Μετά το Global placement, αφού εκτελεστεί η διαδικασία του legalization, τα κελιά έχουν μετακινηθεί από της αρχικές τους θέσεις. Που σημαίνει ότι το μήκος του καλωδίου, wirelength όπως αναφέρεται παρακάτω, έχει μεγαλώσει. Αυτό είναι ένα πρόβλημα που δεν μπορεί να αποφευχθεί, αλλά μπορεί να ελαχιστοποιηθεί με τις διάφορες τεχνικές νομιμοποίησης που έχουν αναπτυχθεί.

## 4.2 Ορισμός του προβλήματος

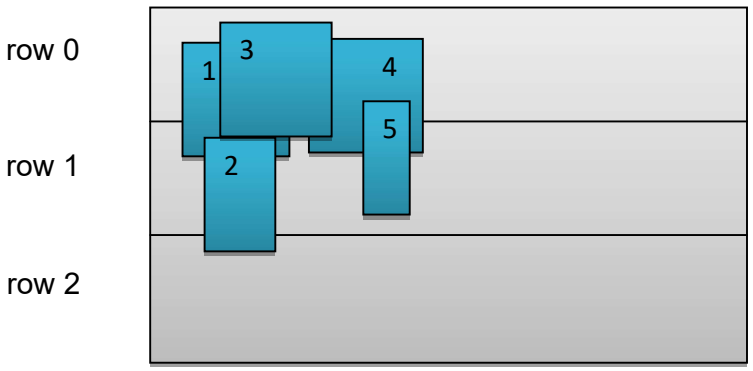
Η συνολική μετακίνηση των κελιών από την αρχική τους θέση στην νέα νόμιμη θέση τους, το λεγόμενο displacement, είναι το κόστος μετακίνησης και υπολογίζεται με το άθροισμα της απόστασης Manhattan, μεταξύ της αρχικής θέσης και της τελικής θέσης, για το κάθε κελί.

$$\sum_{\forall cell} (|x^{cell}_{original} - x^{cell}_{legalized}| + |y^{cell}_{original} - y^{cell}_{legalized}|)$$

Η περιοχή του chip χωρίζεται σε γραμμές ίδιου μήκους και ύψους. Τα κελιά προς νομιμοποίηση θέσης, έχουν όλα σταθερό ίδιο ύψος και στην περίπτωση των Standard Cell, έχουν μεταβλητό μήκος. Ένα κελί λοιπόν θα πρέπει να τοποθετηθεί σε μια κενή θέση, αλλά θα πρέπει να τοποθετηθεί με τέτοιο τρόπο ώστε να μην μεγαλώσει το συνολικό displacement.

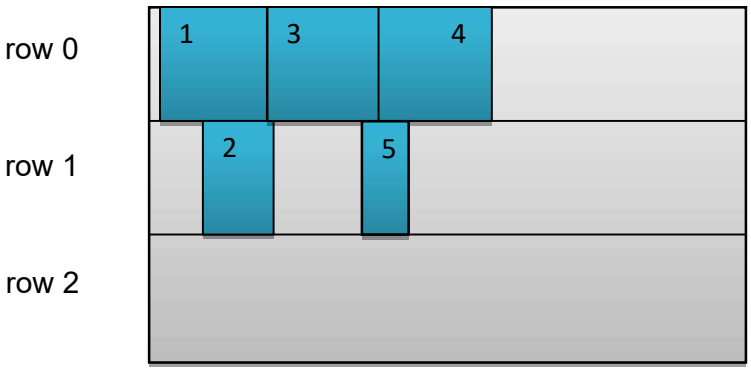
Ένα κελί είναι ένα ορθογώνιο ή τετράγωνο πάνω στο chip. Το chip χωρίζεται σε γραμμές και η κάθε γραμμή μπορεί να έχει πολλά κελιά. Ένα κελί μπορεί να είναι ένα τρανζίστορ ένας αθροιστής ή ακόμα και ένα υποκύκλωμα. Όλα τα κελιά έχουν το ίδιο προκαθορισμένο ύψος, αλλά το μήκος είναι διαφορετικό για το κάθε κελί. Το τελικό layout είναι η περίπτωση των Standard Cell.

Στην διαδικασία του legalization, τα δεδομένα του προβλήματος είναι οι συντεταγμένες X και Y των κελιών. Και για να το οπτικοποιήσουμε ας δούμε την εικόνα 4.2.1, όπου βλέπουμε ότι τα κελιά, δεν είναι ευθυγραμμισμένα στις γραμμές και υπάρχουν επικαλύψεις.




***Σχήμα 4.2.1 :*** Global Placement

Μετά το legalization, το layout θα πρέπει να είναι αυτό, που φαίνεται στην εικόνα 4.2.2, χωρίς επικαλύψεις και ευθυγραμμισμένα στις γραμμές του chip, με όσο το δυνατόν μικρότερο displacement, τοποθετώντας τα κελιά στην πιο κοντινή τους γραμμή.

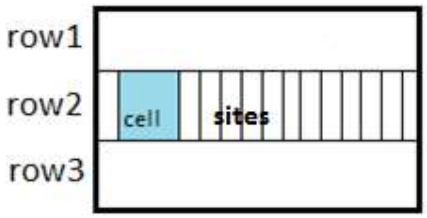



***Σχήμα 4.2.2 :*** Νομιμοποίηση (legalization)

## 4.3 Τεχνικές νομιμοποίησης

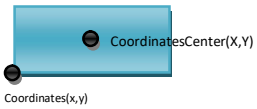
Όποια στρατηγική κι αν ακολουθηθεί για το Global placement πάντα θα υπάρχουν overlaps που θα πρέπει να εξαλειφθούν.

Όλες οι τεχνικές "νομιμοποίησης" legalizers, έχουν τον ίδιο αντικειμενικό στόχο, να απαλείψουν τις επικαλύψεις με όσο το δυνατόν μικρότερη διαταραχή του αρχικού Global placement. Ένα chip χωρίζεται σε γραμμές και οι γραμμές χωρίζονται σε sites, όπως φαίνεται στην εικόνα 4.3.1.




***Σχήμα 4.3.1 :*** Απεικόνιση του chip το οποίο χωρίζεται σε γραμμές και sites.

Δεν χρησιμοποιούν όλοι οι legalizers τα sites. Αλλά αυτοί που στηρίζονται στα sites προτιμούν οι συντεταγμένες των κελιών να είναι ακέραιοι αριθμοί.




***Σχήμα 4.3.2 :*** Συντεταγμένες κέντρου και κάτω αριστερής γωνίας του chip.

Κάποιοι χρησιμοποιούν τις συντεταγμένες κέντρου και άλλοι τις συντεταγμένες της κάτω αριστερής γωνίας.

Στη βιβλιογραφία μπορεί κανείς να βρει αλγορίθμους που βασίζονται σε διάφορες τεχνικές. Στη συνέχεια αναφέρονται κάποιοι αυτούς, ενώ αναλύονται ο Jezz [24] και φυσικά ο Abacus[14] ο οποίος και υλοποιήθηκε.

### 4.3.1 A VLSI Artwork Legalization Technique Based on a New Criterion of Minimum Layout Perturbation [13]

Σε αυτήν την δημοσίευση, διερευνάται το πρόβλημα τροποποίησης ενός layout, που προέκυψε από το Global placement, με σκοπό να μην υπάρχουν παραβάσεις όπως τα overlaps, με όσο το δυνατόν μικρότερη διατάραξη του αρχικού placement.

Ένα κελί θα πρέπει να μετακινηθεί μέσα στις γραμμές του chip, για να βρεθεί η βέλτιστη θέση του. Για να γίνει αυτό θα πρέπει πρώτα να βρεθεί η δοκιμαστική θέση και να ελεγχθεί αν η απόσταση από την αρχική είναι η καλύτερη. Φυσικά στην μετακίνηση αλλάζει και η συντεταγμένη X αλλά και η Y. Το ιδανικό θα ήταν να υπολογίζεται το κόστος και για τις δύο συντεταγμένες όμως αυτό δεν γίνεται. Γι αυτό πρώτα θα υπολογίζεται το ένα κόστος και μετά το άλλο

Μπορεί να χρησιμοποιηθούν δύο μετρικές για τον υπολογισμό του κόστους, η απόσταση Manhattan αλλά και η ευκλείδεια, όπου και στις δύο υπολογίζεται και το βάρος του κελιού.

$$\left\|x_i - x_i^{old}\right\| = w_i \cdot \left|x_i - x_i^{old}\right| \quad \text{μετρική L1, απόσταση Manhattan}$$

$$\left\|x_i - x_i^{old}\right\| = w_i \cdot \left|x_i - x_i^{old}\right|^2 \quad \text{μετρική L2, ευκλείδεια απόσταση}$$

Το layout είναι ένα σετ από πολύγωνα. Χρησιμοποιώντας την μετρική L1, απόσταση Manhattan, το κάθε πολύγωνο αποτελείται μια διατεταγμένη λίστα οριζόντιων ακμών, horizontal edges και κάθετων ακμών, vertical edges. Οι οριζόντιες και κάθετες, ακμές εκφράζουν την μετακίνηση του κελιού, βάση της συντεταγμένης X και της συντεταγμένης Y.

Δημιουργείται ένας γράφος, G(V,A). Κόμβοι V είναι οι ακμές του πολυγώνου, edges Ei. Η διαφορά μεταξύ δύο ακμών Ei και Ej είναι η ακμή του γράφου A. Λαμβάνοντας υπόψη τον γράφο βέλτιστες θέσεις υπολογίζονται από την αντικειμενική συνάρτηση :

$$\min \sum_{vi \in v} w_i \cdot (R_i - L_i) + \lambda \cdot \sum_{Am \in Ar} (M_i - X_j + L_{iJ})$$

Όπου R και L είναι μεταβλητές των κόμβων V του γράφου και M και X μεταβλητές των ακμών του γράφου A. Εφαρμόζοντας τον Graph – Based simplex αλγόριθμο λύνεται το πρόβλημα ικανοποιητικά και πάρα πολύ γρήγορα. Όπως αναφέρεται 40 φορές γρηγορότερα από ένα solver που χρησιμοποιείται στην βιομηχανία.

### 4.3.2 An Effective Legalization Approach Based on Multiple Ordering [17]

Η διαδικασία του legalization χωρίζεται σε τρία διαφορετικά στάδια, με κύριο στόχο την μείωση του μήκους του καλωδίου και μικρότερο displacement. Τα τρία στάδια είναι τα εξής (αλγόριθμος 4.3.2.1):

***Αλγόριθμος 4.3.2.1***

**Στάδιο1**

1. Ταξινόμηση των κελιών σε φθίνουσα σειρά βάση του μήκους των κελιών
2. Ευθυγράμμιση των κελιών στα sites κοντά στην αρχική τους θέση
3. Τα κελιά που τοποθετήθηκαν σε sites κατειλημμένα από άλλα κελιά δεν θεωρούνται έγκυρα είναι invalid

**Στάδιο 2**

1. Ταξινόμηση των invalid κελιών σε φθίνουσα σειρά βάση των pins των κελιών
2. Ορισμός ορίων της περιοχής αναζήτησης (bounding box) για το κάθε invalid κελί
3. Εύρεση βέλτιστης θέσης του κελιού με το μικρότερο displacement και μήκος καλωδίου

**Στάδιο 3**

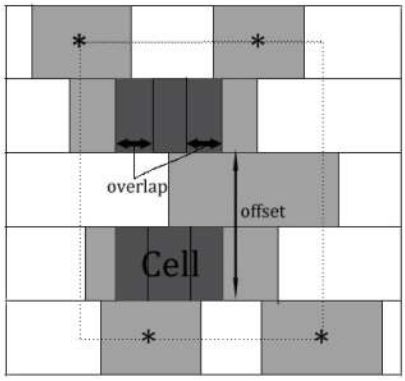
1. Ταξινόμηση των κελιών βάση της συντεταγμένης κέντρου σε αύξουσα σειρά
2. **Επανέλαβε**
   Επαναπροσδιόρισε τις θέσεις των κελιών για αφαιρεθούν τα overlaps
   **Μέχρι** να μην υπάρχει κανένα overlap

Στο πρώτο στάδιο ταξινομούνται τα κελιά σε φθίνουσα σειρά βάση του μήκους τους. Με αυτόν τον τρόπο τα κελιά τα οποία είναι μεγάλα και καταλαμβάνουν πολύ χώρο και θα είναι δύσκολο να τοποθετηθούν αργότερα, τοποθετούνται πρώτα. Τα κελιά ευθυγραμμίζονται με τα sites που είναι πιο κοντά στην θέση που πήραν στο global placement. Έτσι, για τα κελιά που τοποθετήθηκαν, το displacement και το wirelength είναι μικρά. Όμως σίγουρα θα υπάρχουν overlaps μεταξύ των κελιών και τα κελιά αυτά παίρνουν τον τίτλο invalid για τοποθετηθούν σε επόμενο στάδιο.

Στο δεύτερο στάδιο ταξινομούνται μόνο τα invalid κελιά σε φθίνουσα σειρά βάση των pins που έχουν. Όσα περισσότερα pins έχει ένα κελί με τόσα nets συνδέεται άρα ανάλογα αυξάνεται και το wirelength. Για το κάθε invalid κελί ορίζεται ένα bounding box το οποίο είναι ορισμός ορίων της περιοχής αναζήτησης (σχήμα *4.3.2.1*). Η βέλτιστη θέση υπολογίζεται από την συνάρτηση:

$$cost = \quad * offset + b * overlap$$

Όπου offset είναι η απόσταση της αρχικής και τελικής θέσης όπως φαίνεται στο σχήμα *4.3.2.1* και overlap είναι το μήκος των κοινών περιοχών των κελιών




***Σχήμα 4.3.2.1 :*** Υπολογισμός Κόστους

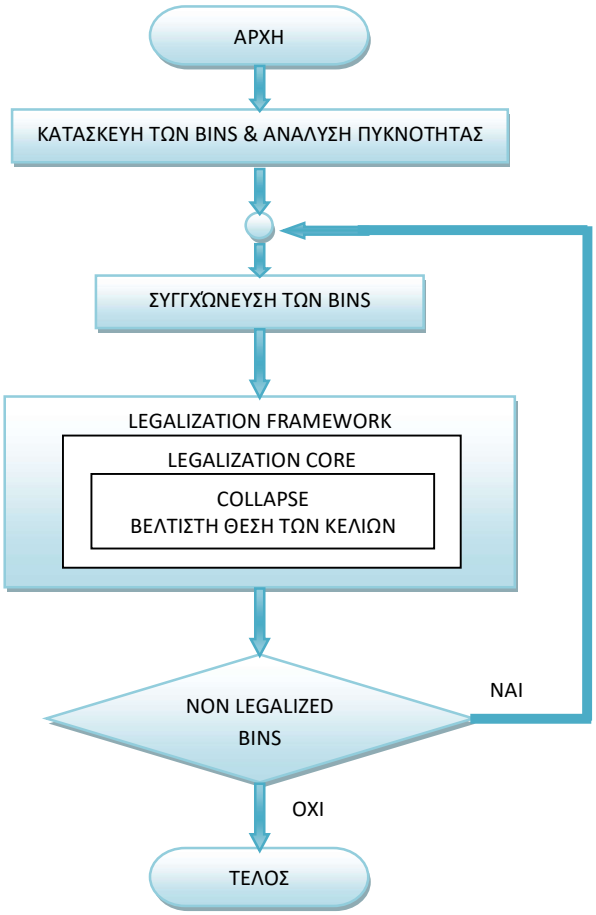
Το σχήμα *4.3.2.1* είναι ένα παράδειγμα από ένα κελί το οποίο φεύγει από ένα overlap πάει σε μια άλλη γραμμή όπου και εκεί δημιουργεί νέο overlap. Γι αυτό είναι αναγκαίο και το στάδιο τρία, στο οποίο τα κελιά που ανήκουν σε μια γραμμή ταξινομούνται σε αύξουσα σειρά βάση των συντεταγμένων κέντρου των κελιών. Μέσα σε μια επανάληψη η οποία εκτελείται όσο υπάρχουν overlaps, επανατοποθετούνται τα κελιά εξαλείφοντας τα overlaps.

### 4.3.3 HiBinLegalizer [2]

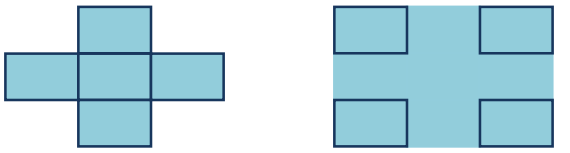
Ο HiBinLegalizer είναι μια ιεραρχική προσέγγιση νομιμοποίησης. Έχει αναπτυχθεί για να νομιμοποιήσει τα κελιά με την ελάχιστη δυνατή κίνηση βασιζόμενος στον υπολογισμό του κόστους με την απόσταση Manhattan.

Παρακάτω βλέπουμε το λογικό διάγραμμα του αλγορίθμου.

***Αλγόριθμος 4.3.3.1***




Πρώτον, ένα chip χωρίζεται σε διάφορες περιοχές, bins όπως αναφέρονται, ίσου μεγέθους. Γίνεται ανάλυση πυκνότητας των bins και αυτό με την μεγαλύτερη πυκνότητα, δηλαδή αυτό με τα περισσότερα κελιά συγχωνεύεται με τα bins που γειτνιάζει και αν είναι δυνατόν φτιάχνουν μια νοητή περιοχή σε σχήμα σταυρού (σχήμα*4.3.3.1 α*) και αν αυτό δεν είναι δυνατόν αντί σταυρού δημιουργείται ένα τετραγωνικό σχήμα (σχήμα *4.3.3.1β*).



α. Σταυρός β. Τετραγωνικό σχήμα

***Σχήμα 4.3.3.1 :*** Διαχωρισμός του chip σε Bins

Το νέο σχήμα που δημιουργήθηκε, είτε σε σταυρό είτε σε τετραγωνικό σχήμα, μπορεί να περιέχει ένα συγκεκριμένο αριθμό χωρητικότητας κελιών. Όταν φτάσει η χωρητικότητα σε αυτό το όριο, ολοκληρώνεται η διαδικασία συγχώνευσης και ψάχνει το επόμενο πολυπληθή Bin, για να ξεκινήσει και πάλι η διαδικασία συγχώνευσης. Αυτό συνεχίζεται μέχρι να μην υπάρχουν Bins που να μην έχουν συγχωνευτεί με τους γείτονές του. Με αυτήν την μέθοδο ο HiBinLegalizer επιτυγχάνει καλύτερο Displacement και καλύτερο wirelength.

Για κάθε συγχωνευμένο bin καλεί την legalization framework πριν αναζητήσει το επόμενο πολυπληθή Bin. Στην legalization framework τα κελιά, ταξινομούνται βάση των συντεταγμένων X σε αύξουσα σειρά. Η επανάληψη του legalization, ξεκινάει με το πρώτο κελί, της ταξινομημένης σειράς. Καλείται η διαδικασία legalization core πολλές φορές χωρίς να αποθηκεύεται το αποτέλεσμα. Η διαδικασία legalization core όταν καλείται δοκιμαστικά τοποθετεί το κελί σε μια θέση και υπολογίζει το κόστος. Η γραμμή με το μικρότερο κόστος επιλέγεται και καλείται πάλι η διαδικασία legalization core και αυτή τη φορά αποθηκεύει το αποτέλεσμα. Η διαδικασία legalization core χρησιμοποιεί την τεχνική του clustering. Σε κάθε γραμμή τα κελιά που γειτνιάζουν μπαίνουν σε ένα cluster. Αυτό το cluster κάθε φορά που μπαίνει ένα κελί αλλάζει θέση οπότε αλλάζουν θέση και τα κελιά που περιέχονται σε αυτό. Έτσι κάθε φορά θα πρέπει να γίνεται έλεγχος για το αν υπάρχει overlap με το προηγούμενο cluster. Αν υπάρχει τότε αυτό εξαλείφεται καλώντας αναδρομικά την συνάρτηση Collapse. Η διαδικασία legalization core είναι παρόμοια με τον αλγόριθμο Abacus [14] ο οποίος αναλύεται στην παράγραφο 4.3.9.

### 4.3.4 On Legalization of Row-Based Placements [22]

Δεδομένου ενός row-based global placement, σκοπός του legalizer είναι, να μην υπάρχουν overlaps, με όσο το δυνατό μικρότερη αύξηση του wirelength, και όσο το δυνατόν μικρότερο displacement. Πετυχαίνοντας αυτούς τους δύο στόχους, ο legalizer πετυχαίνει ακόμα και τον μη αποκλεισμό των pins των κελιών κάνοντας μια μικρή αλλαγή στον γράφο που αναπαριστά τα κελιά σε μια γραμμή. Ένα row-based global placement διαφέρει από της άλλες στρατηγικές στο ότι τα κελιά είναι ήδη ευθυγραμμισμένα σε γραμμές αλλά τα κελιά αυτά μπορεί να μην χωράνε όλα στην γραμμή που ευθυγραμμίστηκαν και σίγουρα υπάρχουν overlaps

Ο αλγόριθμος *4.3.4.1* χωρίζεται σε δύο φάσεις. Στην πρώτη φάση το πρώτο βήμα είναι να βρεθεί αν η κάθε γραμμή έχει κάποια κελιά που δεν χωράει. Το πλεόνασμα όπως αναφέρεται στον αλγόριθμο surplus. Αφού βρεθεί το surplus για την κάθε γραμμή οι γραμμές ταξινομούνται σε φθίνουσα σειρά βάση του surplus.

Σε μια επαναληπτική διαδικασία η οποία ολοκληρώνεται όταν σαρωθούν όλες οι ταξινομημένες γραμμές του chip τα κελιά που πλεονάζουν τοποθετούνται σε άλλες γραμμές. Πιο Συγκεκριμένα το πλεόνασμα, surplus, υπολογίζεται ως εξής:

$$S(\rho i) = \sum_{j=}^{|Ci|} w(cj) - \omega(\rho i)$$

Όπου $S(\rho i)$ είναι το πλεόνασμα της γραμμής, $\sum_{j=}^{|Ci|} w(cj)$ είναι το συνολικό μήκος των κελιών που ανήκουν στην γραμμή και $\omega(\rho i)$ είναι η χωρητικότητα της γραμμής.

***Αλγόριθμος 4.3.4.1***

**Input:** An overlapped placement where row capacitances are not met.
**Output:** An overlapped placement where row capacitances are met.

1. **for** each $i = 1$ to $r$: calculate the surplus of row $\rho_i$ as follows $S(\rho_i) = \sum_{j=1}^{|C_i|} w(c_j) - \omega(\rho_i)$.
2. Rank the rows in a non-increasing order according to the surplus.
3. **for** each row $\rho_i$ where the $S(\rho_i) > 0$
4. **while** $S(\rho_i) > 0$
5. Initialize $\delta_{best} = \infty$.
6. **for** each cell $c_j \in C_i$:
7. **for** each $k = 1$ to $r$:
8. **if** $S(k) < 0$ and $|S(k)| > w(c_j)$ **then**
9. measure the increase in HPWL $\delta_{HPWL}$ if $c_j$ is juggled to row $\rho_k$.
10. **if** $\delta_{HPWL} < \delta_{best}$ **then** $\delta_{best} = \delta_{HPWL}$, $\rho_{best} = \rho_k$, and $c_{best} = c_j$.
11. Move the cell $c_{best}$ to row $\rho_{best}$, and update the surplus values of rows $\rho_k$ and $\rho_i$.

Για κάθε κελί που περισσεύει αναζητά ελεύθερο χώρο σε γειτονικές γραμμές, που θα μπορέσει να ενσωματωθεί το κελί που περισσεύει. Σε αυτήν την φάση δεν ψάχνει αν υπάρχει overlap, αλλά μόνο αν χωράει το κελί στην γραμμή.

Στην δεύτερη φάση (αλγόριθμος 4.3.4.2 ) εξαλείφονται όλα τα overlaps. Μια γραμμή έχει ένα πλήθος από sites. Ο σκοπός εδώ είναι τα τοποθετηθούν τα κελιά σε sites ώστε να μην υπάρχουν overlaps. Για να επιτευχθεί αυτό δημιουργείται ένας μη κυκλικός γράφος G=(V,E). Όπου V τα κελιά και E, η ένωση των οριζόντιων και διαγώνιων ακμών.

***Αλγόριθμος 4.3.4.2***

**Input:** Row $\rho_i$, Set of cells $C_i = \{c_1, c_2, \ldots, c_m\}$ ordered from left to right.Placement objective.
**Output:** An non-overlapped placement of $C_i$ attaining the input objective.

1. Initialize $cost(0,0) = 0$. **for** $j = 1$ to $m$: $cost(j,0) = \infty$. **for** $k = 1$ to $n$: $cost(0,k) = 0$
2. **for** $j = 1$ to $m$
3. **for** $k = 1$ to $n - w(c_j)$
4. **if** objective is minPERB, minHPWL **then** $cost(j,k) = \min(cost(j,k-1), cost(j-1,k-w(c_j)))$
5. **if** objective is minMAX **then** $cost(j,k) = \min(cost(j,k-1), max(cost(j-1,k-w(c_j)), |k-w(c_j) - x_{c_j}|)$
5. **return** $cost(m,n)$

Ο γράφος δημιουργείται ανάλογα με το τι θέλουμε να βελτιστοποιήσουμε. Αν ο αντικειμενικός σκοπός είναι η μικρότερη αναστάτωση των κελιών δηλαδή το μικρότερο συνολικό displacement, *minPERB* όπως αναφέρεται, ή η μείωση του displacement μιας γραμμής, *minMAX*, τότε η ταμπέλα που μπαίνει, στην διαγώνια ακμή είναι η διαφορά της αρχικής θέσης του κελιού με την νέα.

Στην περίπτωση που ο αντικειμενικός σκοπός είναι η μείωση του wirelength, *minHPWL*, τότε δίνεται η ταμπέλα, στην διαγώνια ακμή, η διαφορά της αρχικής θέσης του κελιού με μια διαφορετική θέση από την τρέχουσα.

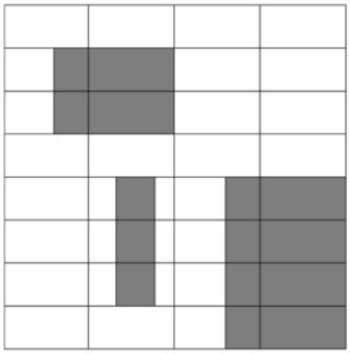
Η διαδικασία αναζήτησης καλύτερου wirelength χρησιμοποιώντας την διαδικασία *minHPWL* του αλγορίθμου, μπορεί να εκτελείται επαναληπτικά μέχρι το wirelength να γίνει μικρότερο από 0,1%. Στις μετρήσεις που έκαναν οι συγγραφείς, το wirelength βελτιώθηκε κατά 17% αλλά με πολύ μεγαλύτερο χρόνο εκτέλεσης.

### 4.3.5 Legalizing a Placement with Minimum Total Movement

[20]

Εδώ προτείνεται μια προσέγγιση δύο φάσεων. Στην πρώτη φάση μετακινούνται τα κελιά ανάμεσα στις περιοχές που έχει χωριστεί το chip. Στην δεύτερη φάση γίνεται αναζήτηση της βέλτιστης θέσης των κελιών πετυχαίνοντας καλύτερο displacement. Βέβαια αυτός ο αλγόριθμος, δουλεύει καλύτερα με τους αλγόριθμους του Global placement, που χωρίζουν το chip σε πίνακες μικρότερων περιοχών, όπου δεν υπάρχουν κελιά που να μην χωράνε σε αυτές τις περιοχές. Εάν ο αλγόριθμος που χρησιμοποιήθηκε για το global placement είναι διαφορετικός, τότε θα πρέπει να γίνει μια προεπεξεργασία. Θα πρέπει να οριστούν περιοχές, στις οποίες θα τοποθετηθούν τα κελιά βάση της συντεταγμένης του κέντρου. Σε οποιαδήποτε περίπτωση λοιπόν, θεωρείται ότι το chip είναι χωρισμένο σε γραμμές με ίδιο ύψος και όλα τα κελιά έχουν το ίδιο ύψος.

Στην πρώτη φάση το chip χωρίζεται σε περιοχές, regions όπως αναφέρονται οι οποίες γειτνιάζουν. Είναι δυνατόν ένα κελί (object) να ανήκει σε δύο περιοχές που γειτνιάζουν. Το κάθε κελί έχει ανατεθεί σε μια περιοχή βάση της συντεταγμένης κέντρου. Μια περιοχή είναι είτε ελεύθερη είτε κατειλημμένη από blocks όπως φαίνεται στο σχήμα *4.3.5.1.*



***Σχήμα 4.3.5.1 :*** Διαχωρισμός του chip σε περιοχές

Οι περιοχές που περιέχουν πάρα πολλά κελιά ονομάζονται sources και οι περιοχές που έχουν ελεύθερο χώρο sinks. Το συνολικό μήκος των κελιών που πρέπει να αποχωρίσουν από το source λέγονται supply και το μήκος των κελιών που μπορεί να δεχτεί ένα sink λέγεται demand. Αυτό λοιπόν είναι το πρόβλημα που πρέπει να λυθεί στην πρώτη φάση. Δημιουργείται ένας γράφος, όπου οι κόμβοι του

είναι οι περιοχές. Οι περιοχές που γειτνιάζουν ενώνονται με κατευθυνόμενες ακμές, άπειρης χωρητικότητας, για να λυθεί το minimum-cost flow πρόβλημα.

Στη δεύτερη φάση σε ένα κελί μπορεί να αλλάξει μόνο η θέση μέσα στην περιοχή που βρίσκεται. Δεν μπορεί να αλλάξει περιοχή αλλά ούτε και γραμμή. Ο αλγόριθμος Single Row Optimization Problem (SROP) βρίσκει την κατάλληλη θέση του κελιού μέσα σε μια περιοχή.

Οι συναρτήσεις f1…, fn είναι quadratic και υπολογίζουν το displacement για το κάθε κελί.

Τα μήκη των κελιών w1,…,wn>0 και οι συντεταγμένες X θα πρέπει: xmin - xmax > w1+…+wn

***Αλγόριθμος 4.3.5.1***

SROP ALGORITHM

*Input:* $n \in \mathbb{N}$. Convex functions $f_1, \ldots, f_n : \mathbb{R} \to \mathbb{R}$. Widths $w_1, \ldots, w_n > 0$ and $x_{\min}, x_{\max} \in \mathbb{R}$ with $x_{\max} - x_{\min} \geq w_1 + \cdots + w_n$.

*Output:* $x_1, \ldots, x_n$ with $x_{\min} \leq x_1, x_i + w_i \leq x_{i+1}$ for $i = 1, \ldots, n-1$, $x_n \leq x_{\max}$, and $\sum_{i=1}^{n} f_i(x_i)$ minimum.

① Set $x_0 := x_{\min}$ and $W_0 := 0$ and $W_i := w_i$ for $i = 1, \ldots, n$. Let $\mathcal{L}$ be the list consisting of $0, 1, \ldots, n$. Set $i := 1$.

② Let $h$ be the predecessor of $i$ in $\mathcal{L}$. IF $h = 0$ or $x_h + W_h \leq \min\{x_{\max} - W_i, \max\{x : f_i(x) \text{ minimum}\}\}$, THEN GO TO ③, ELSE GO TO ④.

③ Set $x_i := \max\{x_h + W_h, \min\{x_{\max} - W_i, \min\{x : f_i(x) \text{ minimum}\}\}\}$ IF there is a successor $j$ of $i$ in $\mathcal{L}$ THEN set $i := j$ and GO TO ② ELSE GO TO ⑤.

④ Redefine $f_h$ by $f_h : x \mapsto f_h(x) + f_i(x + W_h)$. Set $W_h := W_h + W_i$. Remove $i$ from $\mathcal{L}$. Set $i := h$ and GO TO ②.

⑤ For $i \in \{1, \ldots, n\} \setminus \mathcal{L}$ set $x_i := x_h + \sum_{j=h}^{i-1} w_j$ where $h$ is the maximum index smaller than $i$ that belongs to $\mathcal{L}$.

Δημιουργώντας μια διπλά συνδεδεμένη λίστα από τα κελιά που υπάρχουν στην περιοχή αλλάζει τις θέσεις των κελιών σύμφωνα με την σχέση:

$$\min\{x_{\max} - W_j, \min\{x : f_j(x) \text{ minimum}\}\} \leq x_j \leq \max\{x_{\min}, \max\{x : f_j(x) \text{ minimum}\}\}$$

Τέλος χρησιμοποιεί την postoptimization, στην οποία μετακινεί μεμονωμένα κελιά που έχουν μεγαλύτερο displacement για να βελτιωθεί ακόμα περισσότερο το συνολικό displacement.

### 4.3.6 Fast Legalization for Standard Cell Placement with Simultaneous Wirelength and Displacement Minimization [18]

Σε αυτόν τον αλγόριθμο συναντάμε μια γρήγορη τεχνική σάρωσης των γραμμών του chip. Η τεχνική αυτή είναι η top-down clustering. Οι γραμμές ομαδοποιούνται σε clusters σύμφωνα με την ομοιότητά τους. Η ομοιότητά τους βασίζεται στην απόσταση που έχουν οι γραμμές μεταξύ τους. Αν η απόσταση είναι μεγάλη τότε η ομοιότητά τους είναι μικρή, αλλιώς αν η απόστασή τους είναι μικρή η ομοιότητά τους είναι μεγάλη. Αρχικά όλες οι γραμμές ανήκουν σε ένα cluster το root cluster. Χρησιμοποιώντας τον αλγόριθμο k-medoid [29] όλες οι γραμμές ομαδοποιούνται σε clusters. Τελικά κατασκευάζεται ένα δυαδικό δέντρο αναζήτησης, το οποίο προσφέρει γρήγορη σάρωση των γραμμών, ώστε να βρεθεί η γραμμή με το μικρότερο κόστος. Το cost edge στο δυαδικό δέντρο είναι η απόσταση των κελιών από το row cluster. Η απόσταση των κελιών υπολογίζεται από την *Dmin(rowi,rowj)*.

***Αλγόριθμος 4.3.6.1***

```
1   Construct a binary index tree for rows;
2   Sort cells according to (x-position + width / 2);
3   foreach cell do
4   |     row_best = null;
5   |     row_best = DFS(root_cluster,cell);
6   |     Insert cell to row_best;
7   end
```

Όπως φαίνεται στον αλγόριθμο αφού δημιουργηθεί το δυαδικό δέντρο, τα κελιά ταξινομούνται βάση της συντεταγμένης του κέντρου. Στην επανάληψη που ακολουθεί υπολογίζει την βέλτιστη θέση στην βέλτιστη γραμμή, για το κάθε κελί. Αφού για το κάθε κελί βρεθεί η κατάλληλη γραμμή και θέση, τότε το κελί εισάγεται στην γραμμή.

Στην γραμμή 5 του αλγόριθμου καλείται η συνάρτηση DFS (αλγόριθμος *4.3.6.2*) με παραμέτρους τη ρίζα του δυαδικού δέντρου, root_cluster και το κελί, cell. Η τεχνική διάσχισης του δυαδικού δέντρου για την αναζήτηση της καλύτερης γραμμής, που χρησιμοποιείται είναι η depth-first search. Αφού βρεθεί σε ποιο row cluster θα εστιαστεί η αναζήτηση βέλτιστης θέσης, σε μια επανάληψη που ολοκληρώνεται όταν σαρωθούν όλες οι γραμμές που ανήκουν στο row cluster, για κάθε γραμμή καλείται η TrialPlace, με παραμέτρους την γραμμή και το κελί.

***Αλγόριθμος 4.3.6.2***

```
1  Function DFS(cell, row_cluster)
2  if row_cluster is not a leaf then
3      D_R = D_min(row_cluster.RightChild, cell);
4      D_L = D_min(row_cluster.LeftChild, cell);
5      if (D_R < D_L) then
6          if row_best is null or displacement_best > D_R then
7              DFS(cell, row_cluster.RightChild);
8              if row_best is null or displacement_best > D_L then
9                  DFS(cell, row_cluster.LeftChild);
10             end
11         end
12     else
13         if row_best is null or displacement_best > D_L then
14             DFS(cell, row_cluster.LeftChild);
15             if row_best is null or displacement_best > D_R then
16                 DFS(cell, row_cluster.RightChild);
17             end
18         end
19     end
20 else
21     foreach row in row_cluster do
22         displacement_candidate = TrialPlace(row, cell);
23         if displacement_candidate < displacement_best then
24             displacement_best = displacement_candidate;
25         end
26     end
27 end
```

Μέσα στην TrialPlace υπολογίζεται το κόστος, για την κάθε γραμμή που ελέγχεται. Μια γραμμή έχει πολλά clusters τα οποία περιέχουν τα κελιά που γειτνιάζουν. Αν τα clusters γειτνιάζουν ή υπάρχει overlap μεταξύ τους, τότε συγχωνεύονται. Για την βέλτιστη θέση χρησιμοποιείται η συντεταγμένη X του κέντρου του κελιού. Η αντικειμενική συνάρτηση που χρησιμοποιεί είναι η

$$min \sum_{i=1}^{n} |x_i^c - x'^c_i|$$

με $x_i^c$ την θέση στο global placement και $x'^c_i$ την προσωρινή θέση.
Η συνάρτηση υπολογισμού βέλτιστης θέσης του κελιού μέσα στο cluster είναι η

$$min \sum_{i=1}^{n} |x_1^c - \underbrace{[x_i^c - \sum_{k=1}^{i-1} w_k^c]}_{d_i^c}|$$

Όπου $d_i^c$ η δοκιμαστική θέση του cluster και όλα τα $d_i^c$ είναι ένα σετ από $D_c$. Η βέλτιστη θέση του cluster, $x_1^c$, θα προέρθει υπολογίζοντας τον μέσο των $D_c$, που είναι τελικά η βέλτιστη λύση.

Για να βρεθεί ο μέσος χρησιμοποιείται ένα red-black tree, επειδή έχει καλύτερο χρόνο εκτέλεσης από έναν αλγόριθμο ταξινόμησης. Θα πρέπει εδώ να προστεθεί, ότι αν το πλήθος των κελιών του cluster είναι ζυγός αριθμός, τότε το κελί τοποθετείται στην δεξιότερη θέση του cluster και το cluster δεν μετακινείται. Έτσι δεν γίνεται έλεγχος αν υπάρχει overlap με το προηγούμενο, cluster αφού δεν μετακινήθηκε. Αλλιώς αν το πλήθος των κελιών του cluster είναι μονός αριθμός, τότε το κελί τοποθετείται στην δεξιότερη θέση του cluster και το cluster μετακινείται αριστερά, οπότε ο έλεγχος για overlap με τον προηγούμενο είναι απαραίτητος.

### 4.3.7 Tetris [23]

Ο Tetris είναι ένας γρήγορος, greedy heuristic αλγόριθμος, ο οποίος χρησιμοποιείται ευρεία στην βιομηχανία. Το πρώτο βήμα του είναι να ταξινομήσει ως προς τις συντεταγμένες X τα κελιά του chip. Για κάθε κελί υπολογίζει το κόστος μετακίνησής του, για την κάθε πιθανή γραμμή που θα μπορούσε να τοποθετηθεί. Το κόστος μετακίνησης, υπολογίζεται με την απόσταση Manhattan, μεταξύ της αρχικής θέσης και της θέσης μέσα στην γραμμή την οποία δοκιμάζει. Μέσα από μια επανάληψη βρίσκει το μικρότερο κόστος τοποθέτησης αφού δοκιμάσει όλες τις γραμμές του chip. Τελικά τοποθετεί το κελί στη γραμμή με το μικρότερο κόστος και προσθέτει το μήκος του κελιού στην πιο αριστερή καταλαμβανόμενη θέση της γραμμής. Στον αλγόριθμο *4.3.7.1* φαίνεται ο αλγόριθμος του Tetris.

***Αλγόριθμος 4.3.7.1***

```
1  C = all cells to be legalized;
2  l_j = left-most position of each row j;
3  for each cell i in C sorted by x-position do
4  |   best_cost = ∞
5  |   for each row j in the circuit do
6  |   |   x = max{x_i, l_j}
7  |   |   cost = |x − x_i| + |y_j − y_i|;
8  |   |   if cost < best_cost then
9  |   |   |   best_cost = cost
10 |   |   |   best_row = j
11 |   |   end
12 |   end
13 |   x_i = max{x_i, l_best_row}
14 |   l_best_row = x_i + width_i
15 end
```

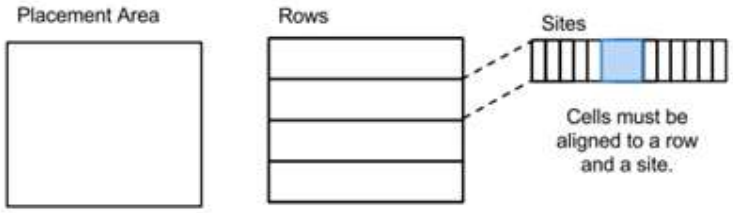
Τα κελιά ταξινομούνται ως προς την συντεταγμένη X δίνοντας μεγαλύτερο βάρος στα μεγαλύτερα σε μήκος κελιά. Μέσα σε μια επανάληψη που ολοκληρώνεται όταν τοποθετηθούν όλα τα κελιά, σαρώνει όλες τις γραμμές για κάθε ένα ταξινομημένο κελί. Για το κάθε κελί αναζητά ελεύθερα sites, που θα μπορούσε να χωρέσει και τοποθετηθεί το κελί, στην πιο αριστερή θέση της γραμμής για όλες τις γραμμές. Ελεύθερο site σημαίνει ότι δεν καταλαμβάνεται από άλλο κελί. Τα ελεύθερα sites αποθηκεύονται σε μια δομή, που μπορεί να είναι και ένας μονοδιάστατος πίνακας. Αφού ολοκληρωθεί η διαδικασία και βρεθούν τα ελεύθερα sites σε όλες τις γραμμές, αναζητείται η βέλτιστη θέση. Η βέλτιστη θέση είναι αυτή με το μικρότερο κόστος. Το κόστος υπολογίζεται από :

$$\text{cost} = |x_{or} - x_{tr}| + |Y_{or} - Y_{tr}|$$

Όπου $x_{or}, Y_{or}$ είναι οι συντεταγμένες από το Global placement και $x_{tr}, Y_{tr}$ είναι οι συντεταγμένες ενός ελεύθερου site. Για το κάθε ελεύθερο site υπολογίζεται το κόστος και το site με το μικρότερο κόστος επιλέγεται για να τοποθετηθεί το κελί το οποίο όταν τοποθετηθεί δεν μπορεί μετακινηθεί ξανά.

### 4.3.8. Jezz [24]

Είναι ένας legalizer ο οποίος χρησιμοποιεί την κατηγορία των standard cell, και μειώνει το συνολικό Displacement. Για τον Jezz το chip είναι χωρισμένο σε γραμμές και sites όπως φαίνεται στην εικόνα. Τα sites, τα οποία έχουν ίδιο μήκος και ίδιο ύψος τα χρησιμοποιεί για να βρει την καλύτερη θέση του κελιού, υπολογίζοντας τις επιπτώσεις (impact).




***Σχήμα 4.3.8.1 :*** Διαχωρισμός του chip σε γραμμές και sites

Ο Jezz μπορεί να χρησιμοποιηθεί σαν legalizer, δηλαδή σαν την διαδικασία εξάλειψης των overlaps που δημιουργήθηκαν στο Global Placement όπως είναι ο Abacus[14] και ο Tetris [23] αλλά μπορεί να χρησιμοποιηθεί και στο detailed placement, στο οποίο έχει γίνει ήδη legalization στα κελιά, και γίνεται προσπάθεια βελτιστοποίησης του legalization, μετακινώντας κάποια κελιά με σκοπό να μειωθεί το displacement.

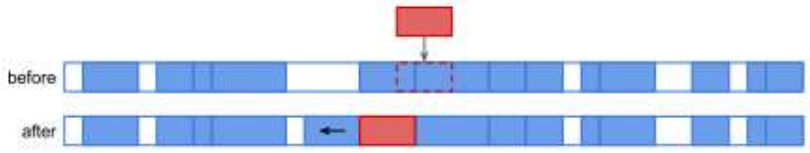
Το displacement υπολογίζεται με την απόσταση Manhattan:

$$\sum_{\forall cell} (\left|x_{original}^{cell} - x_{legalized}^{cell}\right| + \left|y_{original}^{cell} - y_{legalized}^{cell}\right|)$$

Η απόσταση Manhattan είναι πιο πρακτική στο υπολογισμό της μετακίνησης από την quadratic.

Όλες οι αποστάσεις στον Jezz είναι ακέραιοι αριθμοί και όχι double floating point όπως είναι στον Abacus [14].

Ο κόμβος που θα εισαχθεί θα πρέπει να εισαχθεί, μετά τον προηγούμενο από αυτόν κόμβο και ο προηγούμενος κόμβος έχει μικρότερη συντεταγμένη X από τον νέο κόμβο. Αυτό φαίνεται καλύτερα στην εικόνα 4.3.8.2.




***Σχήμα 4.3.8.2 :*** Εισαγωγή κελιού σε γραμμή

Υπάρχουν τρείς εκδόσεις του Jezz, εδώ θα αναλύσουμε την τρίτη και τελευταία έκδοση. Η πιο απλή έκδοση είναι φυσικά η πρώτη. Η μια έκδοση από την άλλη διαφέρει στον τρόπο με τον οποίο υπολογίζεται, το κόστος μετακίνησης των κελιών όπου, στον Jezz τα κελιά αναφέρονται ως κόμβοι (nodes). Το κόστος μετακίνησης των κόμβων, που είναι ήδη τοποθετημένοι στη γραμμή, για να εισαχθεί ένας νέος κόμβος, αναφέρεται στον αλγόριθμο ως impact και το optimum cost όπως επίσης αναφέρεται στον αλγόριθμο, είναι το κόστος επιλογής της βέλτιστης θέσης του νέου κόμβου σε μια γραμμή. Η διαδικασία εισαγωγής των κόμβων σε μια γραμμή είναι ίδια σε όλες τις εκδόσεις.

Στον πίνακα αναφέρονται οι τρείς διαφορετικοί τύποι κόμβων που χειρίζεται ο Jezz.

**Πίνακας 4.3.8.1**

| **Τύπος κόμβου** | Εξήγηση |
|---|---|
| **Whitespace** | Κενό στο οποίο θα μπορούσε να εισαχθεί νέος κόμβος |
| **Blockage** | Κόμβος ο οποίος δεν μπορεί να μετακινηθεί |
| **Cell** | Κελί προς legalization |

Μια γραμμή είναι μια διπλά συνδεδεμένη λίστα και αποτελείται από κόμβους. Κάθε λίστα έχει τουλάχιστον ένα κόμβο και το άθροισμα του μήκους των κόμβων δεν ξεπερνάει το μήκος της γραμμής. Αυτό είναι απαραίτητο για να μην βγαίνουν οι κόμβοι εκτός ορίων του chip.

Μια κενή γραμμή έχει έναν και μόνο κόμβο τύπου whitespace με μήκος το μήκος της γραμμής. Δύο κόμβοι τύπου whitespace δεν πρέπει να γειτνιάζουν. Σε μια τέτοια περίπτωση οι κόμβοι συγχωνεύονται προσθέτοντας τα μήκη τους. Δεν επιτρέπονται whitespaces με μηδενικό μήκος. Αν βρεθούν τέτοιοι κόμβοι διαγράφονται. Ο Jezz τοποθετεί τα κελιά βασιζόμενος στο impact, που θα έχει η

εισαγωγή του κόμβου στη γραμμή, στους άλλους κόμβους που είναι ήδη τοποθετημένοι στη γραμμή αλλά και στο κόστος μετακίνησης του νέου κόμβου. Ο αλγόριθμος *4.3.8.1* είναι ο αλγόριθμος του Jezz.

***Αλγόριθμος 4.3.8.1***

```
1  for each cell c sorted by increasing x-position do
2  |  r_0 = nearestRow(c)
3  |  best_impact = computeImpact(c, r_0)
4  |  best_row = r_0
5  |  for each row r above r_0 do
6  |  |  impact = computeImpact(c, r);
7  |  |  if impact < best_impact then
8  |  |  |  best_impact = impact
9  |  |  |  best_row = r
10 |  |  end
11 |  |  else if impact = best_impact then
12 |  |  |  if curr_max_disp < smallest_max_disp then
13 |  |  |  |  best_impact = impact
14 |  |  |  |  best_row = r smallest_max_disp = curr_max_disp
15 |  |  |  end
16 |  |  end
17 |  end
18 |  else
19 |  |  break;
20 |  end
21 end
22 for each row r below r_0 do
23 |  // see lines 6-20
24 end
25 insertCell(cell, best_row);
```

Το πρώτο βήμα είναι να ταξινομηθούν τα κελιά σε αύξουσα σειρά σύμφωνα με την συντεταγμένη Χ. Υπολογίζεται ποια είναι η πιο κοντινή γραμμή σύμφωνα με την συντεταγμένη Υ. Η πιο κοντινή γραμμή είναι η αφετηρία της αναζήτησης. Σε μια επανάληψη υπολογίζει το impact, με την function Impact Computation, για την κάθε γραμμή που θα μπορούσε να εισαχθεί ο νέος κόμβος για τις γραμμές πάνω από την πιο κοντινή γραμμή (αλγόριθμος *4.3.8.1,* γραμμές 5 - 20) και μετά υπολογίζει το impact για της επόμενες γραμμές από την πιο κοντινή. (αλγόριθμος *4.3.8.1,* γραμμές 21 - 23).

Για παράδειγμα αν το chip έχει 10 γραμμές και η πιο κοντινή είναι η γραμμή 4, θα υπολογίσει το impact πρώτα για την γραμμή τέσσερα και μετά σε μια επανάληψη θα υπολογίσει το impact για τις γραμμές 1,2,3. Θα κρατήσει τη γραμμή με το μικρότερο impact και μετά μέσα σε μια επανάληψη θα υπολογίσει και θα

κρατήσει το μικρότερο impact και από τις γραμμές 5,6,7,8,9,10. Τέλος αφού έχει κρατήσει την γραμμή με το μικρότερο impact και το μικρότερο κόστος εισάγει στην λίστα τον νέο κόμβο. Θα πρέπει να αναφερθεί εδώ ότι η τρίτη έκδοση του Jezz δεν υπολογίζει το μικρότερο impact, σε όλες τις γραμμές του chip μέχρι να βρει το καλύτερο αλλά όπως φαίνεται στις γραμμές 11-15 η επανάληψη σταματάει βάση ενός κριτηρίου.

Εάν το impact της γραμμής είναι ίδιο με το καλύτερο impact που έχει αποθηκεύσει τότε η επανάληψη θα σταματήσει και η γραμμή που θα επιλεγεί θα είναι αυτή που δίνει το μικρότερο displacement του κελιού. Το displacement υπολογίζεται στην function optimum shift.

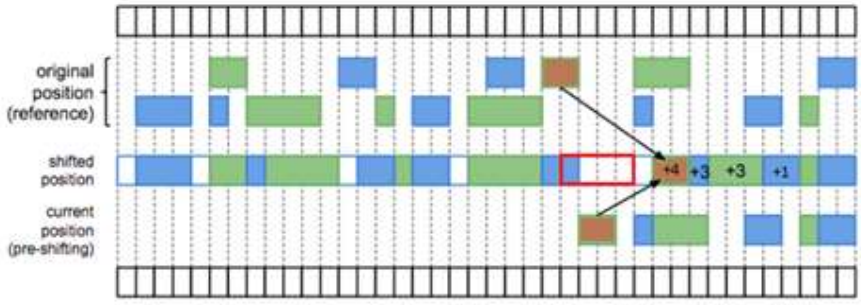
Ακολουθεί ο αλγόριθμος της function Impact Computation, για την αριστερή μετατόπιση, swift. Για να υπολογιστεί το impact για το δεξί swift ο αλγόριθμος είναι περίπου ο ίδιος, όμως αντί του impactl που χρησιμοποιείται για το αριστερό swift, χρησιμοποιείται το impactr.

***Αλγόριθμος 4.3.8.2***

```
1  node = reference node;
2  impact_r[0..w] = {0, ··· , 0};
3  k = 0;
4  while (node and k < w) do
5  |  if node is blockage then
6  |  |  break;
7  |  end
8  |  if node is whitespace then
9  |  |  k += min{width(node), w − k}
10 |  else
11 |  |  for i = k + 1, d = 1; i ≤ w; i++, d++ do
12 |  |  |  impact_r[i] +=
   |  |  |  weight^node × (|x^node_original − (x^node_current + d)| − |x^node_current − x^node_original|)
13 |  |  end
14 |  end
15 |  node = next(node);
16 end
17 overflow_r = w − k
```

Μέσα σε μια επανάληψη σαρώνονται οι γειτονικοί κόμβοι (αλγόριθμος *4.3.8.2,* γραμμή 15) του κόμβου προς εισαγωγή, από τα δεξιά του αφού σε αυτόν τον αλγόριθμο υπολογίζεται το impact.

Η μεταβλητή *k* αναφέρεται στον αριθμό των sites που θα πρέπει να γίνει το shift και η w είναι το μήκος του κελιού. Το σχήμα 4.3.8.3 δείχνει ένα παράδειγμα του υπολογισμού του κόστους. Ο κόμβος προς εισαγωγή είναι ο κόμβος με το κόκκινο περίγραμμα. Στην πάνω σειρά είναι οι θέσεις των συντεταγμένων X από το Global Placement, στη μέση οι θέσεις μετά το shift και στην τελευταία πριν το shift.




***Σχήμα 4.3.8.3 :*** Μετακίνηση κελιού από τη αρχική του θέση

Για να μπει ο κόκκινος κόμβος στη θέση που πρέπει, θα πρέπει να μετακινηθεί ο καφέ κόμβος 4 sites, όπως επίσης και οι επόμενοι από αυτόν όπως φαίνεται στο σχήμα *4.3.8.3*. Το impact για τον κάθε κόμβο που επηρεάζεται αποθηκεύεται στον πίνακα impact, με μέγεθος πίνακα το μήκος του κόμβου προς εισαγωγή. Ο υπολογισμός των impacts τερματίζεται όταν το μήκος των whitespaces είναι μεγαλύτερο ή ίσο με το μήκος του νέου κόμβου ή όταν δεν υπάρχει αρκετός χώρος για να εισαχθεί το νέο κελί.

Αφού βρεθούν τα impacts από το δεξί και αριστερό shift, γίνεται η επιλογή του βέλτιστου από την function optimum shift (αλγόριθμος*4.3.8.3* ). Στο Full legalization δεν γίνεται υπολογισμός του δεξί shift διότι η δεξιά μεριά του τελευταίου κόμβου είναι φυσικά κενή.

***Αλγόριθμος 4.3.8.3***

```
1  max_l = w − overflow_l // max available sites to left
2  max_r = w − overflow_r // max available space to right
3  if max_l + max_r < w then
4  |  exit(overflow);
5  end
6  best_cost = ∞;
7  smallest_max_disp = ∞;
8  for i = max_l; i ≥ w − max_r; i- - do
9  |  curr_cost = |(x_original + w) − i|;
10 |  cost = impact_l[i].cost + weight × curr_cost + impact_r[w − i].cost;
11 |  curr_max_disp = max(max(impact_l[i].disp, impact_r[w −
   |  i].disp, |node_y_disp + curr_cost × step|);
12 |  if cost < best_cost then
13 |  |  best_cost = cost;
14 |  |  shifts_left = i
15 |  end
16 |  if cost = best_cost then
17 |  |  if curr_max_disp = smallest_max_disp then
18 |  |  |  shifts_left = i smallest_max_disp = curr_max_disp;
19 |  |  end
20 |  end
```

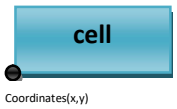
Στην optimum shift, υπολογίζεται το κόστος σαρώνοντας τους δύο πίνακες με αντίθετη κατεύθυνση. Επίσης υπολογίζει και επιλέγει το μικρότερο displacement. Υπολογίζει το maximum displacement το οποίο χρησιμοποιείται ως κριτήριο τερματισμού. Αφού τελειώσουν οι υπολογισμοί, αποφασίζει που θα τοποθετηθεί ο κόμβος. Επιλέγει τη θέση με το μικρότερο κόστος. Αυτή η θέση δεν είναι απαραίτητο ότι θα έχει μόνο αριστερά ή δεξιά shifts, μπορεί να έχει έναν συνδυασμό αριστερών και δεξιών shifts.

## 4.3.9 Abacus [14]

Στην περίπτωση του Abacus χρησιμοποιείται το layout των Standard Cell, όπου τα κελιά μπαίνουν πάνω σε διατάξεις οριζόντιων γραμμών και έχουν σταθερό ύψος και μεταβλητό μήκος και δεν χρησιμοποιούνται τα sites. Για αυτό και οι συντεταγμένες είναι τύπου double. Πρώτα γίνεται ταξινόμηση των κελιών και μετά ακολουθεί η νομιμοποίηση της θέσης τους, το legalization. Κατά τη διάρκεια του legalization τα κελιά που έχουν ήδη νομιμοποιηθεί μπορούν να μετακινηθούν πάλι, με στόχο να επιτευχθεί καλύτερο displacement. Συγκεκριμένα τα κελιά που έχουν ήδη νομιμοποιηθεί σε μια γραμμή μετακινούνται, έτσι ώστε η συνολική μετακίνηση των κελιών μέσα στη γραμμή να είναι η βέλτιστη.

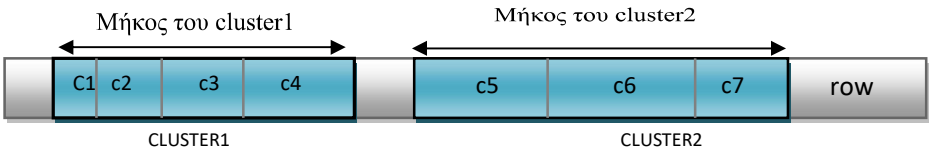
Η βέλτιστη θέση βρίσκεται επιλύοντας ένα quadratic program το οποίο μπορεί να μεταμορφωθεί, σε γρήγορο δυναμικό προγραμματισμό. Η ταξινόμηση των κελιών διατηρείται, δηλαδή αν μετά το global placement, δύο κελιά α και β που βρίσκονται στην ίδια γραμμή και το α είναι αριστερά του β, τότε και μετά το legalization με τον Abacus , τα α θα είναι αριστερά του β.

Οι συντεταγμένες που χρησιμοποιεί ο αλγόριθμος, για να υπολογίσει την βέλτιστη θέση μέσα σε μια γραμμή είναι οι συντεταγμένες της κάτω αριστερής γωνίας, *σχήμα 4.3.9.1*.




***Σχήμα 4.3.9.1 :*** Συντεταγμένες κάτω αριστερής γωνίας

Ο Abacus χρησιμοποιεί την τεχνική clustering. Στο clustering τα κελιά που γειτνιάζουν, ομαδοποιούνται σε μια συστάδα από κελιά τα που ονομάζονται clusters. Η συντεταγμένη X του cluster είναι η συντεταγμένη X του πρώτου κελιού στο cluster. Το μήκος του cluster (*Σχήμα 4.3.9.2*) είναι το συνολικό μήκος των κελιών που ανήκουν στο cluster. Οπότε η συντεταγμένη X της κάτω δεξιάς γωνίας του cluster, είναι το άθροισμα της συντεταγμένης X του cluster και του συνολικού μήκους του cluster. Δύο clusters δεν πρέπει να γειτνιάζουν. Αν αυτό συμβεί τότε τα δύο clusters θα πρέπει να γίνουν ένα.




***Σχήμα 4.3.9.2 :*** Απεικόνιση των cluster

***Αλγόριθμος 4.3.9.1***

1. sort cells according to x-position
2. **foreach** cell i**do**
3. cbest←∞
4. **foreach** row r **do**
5. Insert cell I into row r
6. Placerow r(trial)
7. Determine cost c
8. **if** c<cbest**then**cbest=c, rbest=r
9. Remove cell I from row r
10. **end**
11. Insert cell I to row rbest
12. Placerowrbest(final)
13. **end**

Τα κελιά ταξινομούνται ως προς τη συντεταγμένη X. Ακολουθεί ένα for loop με τόσες επαναλήψεις, όσα είναι τα κελιά του κυκλώματος που πρέπει να νομιμοποιηθούν. Μέσα σε αυτήν την επανάληψη, υπάρχει άλλη μια επανάληψη η οποία ψάχνει μια-μια τις γραμμές του chip, για να βρει ποια είναι η βέλτιστη γραμμή, για να τοποθετηθεί ένα κελί καλώντας την Placerow.

Δύο σημεία πρέπει να επισημανθούν εδώ.

1. Το ένα είναι ότι υπολογίζεται το οριζόντιο κόστος για την κάθε γραμμή. Το οριζόντιο κόστος υπολογίζεται, αφαιρώντας την αρχική συντεταγμένη X, με αυτήν που επέστρεψε από την Placerow.

$$Xcost = \left| x_{original}^{cell} - x_{legalized}^{cell} \right|$$

Ο Abacus δεν ψάχνει όλες τις γραμμές για να βρει την καλύτερη θέση. Υπολογίζει και το κάθετο κόστος, αφαιρώντας την αρχική συντεταγμένη Υ με την συντεταγμένη Υ της γραμμής που ελέγχει.

$$Ycost = |y_{original}^{cell} - y_{legalized}^{cell}|$$

Η δεύτερη επανάληψη σταματάει όταν το κάθετο κόστος γίνει μεγαλύτερο από το οριζόντιο κόστος.

2. Το δεύτερο σημείο είναι όταν η Placerow καλείται, όπως φαίνεται στον αλγόριθμο *4.3.9.1,* ως Placerow(trial). Η Placerow(trial) καλείται δοκιμαστικά, ώστε να ελέγξει τις γραμμές, που θα μπορούσε να τοποθετηθεί το κελί για να επιλεγεί τελικά, η γραμμή με το μικρότερο κόστος. Γι αυτό ότι αλλαγές και υπολογισμοί γίνονται στην Placerow(trial), είναι προσωρινοί και δεν πρέπει να αποθηκεύονται.

Τελικά αφού ολοκληρωθεί η εμφωλευμένη επανάληψη και είναι γνωστό ποια γραμμή έχει το μικρότερο κόστος, καλείται η Placerow(final) και αυτή τη φορά, αποθηκεύονται όλοι οι υπολογισμοί και φυσικά οι νέες συντεταγμένες των κελιών, που βρίσκονται στην γραμμή που επιλέχθηκε, υπολογίζοντας το μικρότερο κόστος.

***Πίνακας 4.3.9.1 :*** Ιδιότητες του κελιού

| **Ιδιότητα** | Εξήγηση |
|---|---|
| **Gx(i),Gy(i)** | Global placement |
| **x(i), y(i)** | Legal placement |
| **W(i)** | Μήκος κελιού |
| **e(i)** | Βάρος κελιού |

Η κάθε γραμμή έχει Nr standard cells. Ο πίνακας *4.3.9.1* δείχνει τις ιδιότητες ενός κελιού. Τα δεδομένα είναι οι συντεταγμένες X(i) και Y(i) της κάτω αριστερής γωνίας, που είναι γνωστές από το Global Placement, το μήκος του κελιού w(i) και το βάρος του κελιού e(i). Το βάρος του κελιού μπορεί να είναι 1 ή το εμβαδόν του κελιού. Η Placerow καθορίζει της νόμιμες συντεταγμένες X(i) και Y(i) του κελιού. Η νόμιμη συντεταγμένη Y(i) καθορίζεται πριν το κάλεσμα της Placerow μετακινώντας το κελί στις γραμμές. Τα κελιά έχουν ταξινομηθεί με την συντεταγμένη X που πήρε στο Global Placement με:

$$x'(i) \geq x'(i-1)$$

Η Placerow καθορίζεται από τον τετραγωνικό προγραμματισμό:

$$min \sum_{i=1}^{Nr} e(i)[x(i) - x'(i)]^2 (\mathbf{1})$$

με τους περιορισμούς:

$$x(i) - x(i-1) \geq w(i-1) \qquad i = 2, \ldots, Nr \quad (\mathbf{2})$$

Η αντικειμενική συνάρτηση (1) περιγράφει τη συνολική τετραγωνική κίνηση, στην οποία έχει συνυπολογιστεί και το βάρος του κάθε κελιού, μεταξύ των συντεταγμένων X' του Global Placement και τις νόμιμες συντεταγμένες X.

Ο περιορισμός (2) εξασφαλίζει ότι δεν υπάρχουν επικαλύψεις μεταξύ των κελιών και διατηρεί την σειρά των κελιών. Το να λυθεί ένα τετραγωνικό πρόγραμμα, με τον περιορισμό ≥ είναι γενικά χρονοβόρο. Αν χρησιμοποιηθεί μόνο η ισότητα σαν περιορισμός, τότε το τετραγωνικό πρόγραμμα λύνεται γρήγορα, επιλύοντας μόνο μια γραμμική εξίσωση.

Αν χρησιμοποιηθεί μόνο η ισότητα τότε οι περιορισμοί μεταμορφώνονται σε :

$$x(i) = x(1) + \sum_{k=1}^{i=1} w(k) i = 2, \ldots, Nr \, (\mathbf{3})$$

Τέλος αν ενώσουμε τις σχέσεις **(1)** και **(3)** τότε έχουμε:

$$\underbrace{\sum_{i=1}^{Nr} e(i)x(1)}_{ec} - \underbrace{[e(1)x'(1) + \sum_{i=2}^{Nr} e(i)[x'(i) - \sum_{k=1}^{i-1} w(k)]]}_{qc} = 0 \qquad (\mathbf{4})$$

Στον πίνακα *4.3.9.2* φαίνονται οι υπολογισμοί που γίνονται κατ' επανάληψη οι οποίοι εξαρτώνται από τις ιδιότητες των κελιών x(i), w(i), e(i).

| ***Πίνακας 4.3.9.2 :*** Συναρτήσεις υπολογισμού βέλτιστης θέσης του cluster | |
|---|---|
| **Υπολογισμοί** | Επεξηγήσεις |
| $\boldsymbol{ec \leftarrow ec + e(i)}$ | βάρος κελιών που ανήκουν στο cluster |
| $\boldsymbol{wc \leftarrow wc + w(i)}$ | συνολικό πλάτος κελιών |
| $\boldsymbol{qc \leftarrow qc + e(i)[x'(i) - wc]}$ | βέλτιστη θέση |
| $\boldsymbol{x(1) = \frac{qc}{ec}}$ | βέλτιστη θέση |

Τελικά η βέλτιστη θέση του cluster υπολογίζεται από την :

$$x(1) = \frac{qc}{ec}$$

***Αλγόριθμος 4.3.9.2***

1. **for** i=1,…Nr **do**
2. c←last cluster
3. **if** i=1 or xc(c)+w(c) ≤Gx(i) **then**
4. create new cluster
5. initec(c), wc(c), qc(c) to zero
6. xc(c)←Gx(i)
7. nfirst (c)←i
8. AddCell(c,i)
9. **else**
10. AddCell(c,i)
11. Collapse(c)
12. **end**

```
//Μετατροπή της θέσης των clusters σε θέσεις κελιών

13. i←1
14. for all clusters c do
15.    x=xc(c)
16.    fori≤nlast(c) do
17.      x(i)←x
18.      x←x+w(i)
19.   end
20. end
21. FunctionAddCell(c,i)
22.    nlast(c)←i
23.    ec(c) ←ec(c)+ec(i)
24.    qc(c) ←qc(c)+e(i)*(Gx(i)-wc(c))
25.    wc(c) ←wc(c)+w(i)
26. FunctionAddCluster(Pc,c)
27.    nlast(c)←nlast(c)
28.    ec(Pc)←ec(Pc) + ec(c)
29.    qc(Pc) ←qc(Pc)+qc(c)-(ec(Pc)*wc(c))
30.    wc(Pc) ←wc(Pc)+wc(c)
31. Function Collapse(c)

// Τοποθέτηση του cluster

32.   xc(c)←qc/ec(c)

/ / Το cluster δεν πρέπει να βγει εκτός των ορίων του chip

33.   if xc(c)<min then xc(c)=xmin
34.   if xc(c)>max-wc(c) then xc(c)=xmax-wc(c)

/ / Επικάλυψη του clusterc με το προηγούμενο clusterPc

35.   if Pc exists and  xc(Pc)+wc(Pc)> xc(c) then
36.      AddCluster(Pc, c)
37.      Remove cluster c
38.      Collapse(Pc)
39.   end
```

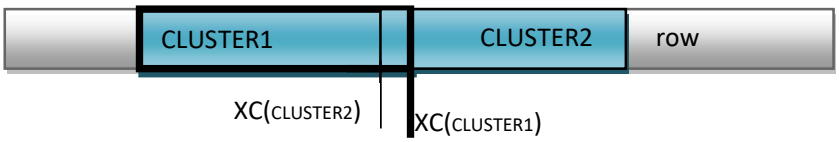
| ***Πίνακας 4.3.9.3 :*** Ιδιότητες του cluster | |
|---|---|
| **Ιδιότητες** | Επεξηγήσεις |
| nfirst(c) | Πρώτο κελί του cluster |
| nlast(c) | Τελευταίο κελί του cluster |
| Nc(c) | Πλήθος των κελιών στο cluster |
| xc(c) | Βέλτιστηθέση (συντεταγμένη χτου cluster) |
| ec(c) | Συνολικό βάρος |
| wc(c) | Συνολικό μήκος |
| qc(c)/ec(c) | Βέλτιστη θέση |

Τα κελιά έχουν ταξινομηθεί σε αύξουσα σειρά. Μέσα σε μια επανάληψη η οποία ολοκληρώνεται όταν προσπελαστούν όλα τα κελιά που ανήκουν στο ίδιο cluster αναζητείται η βέλτιστη θέση του cluster η οποία εξαρτάται από τα βάρη, τα μήκη και τις συντεταγμένες X, των κελιών που ανήκουν σε αυτά. Αν το κελί *i* είναι το πρώτο κελί του cluster ή αν δεν υπάρχει overlap με το τελευταίο cluster της γραμμής, τότε δημιουργείται νέο cluster ( αλγόριθμος *4.3.9.2,* γραμμή 4) και η συντεταγμένη X του cluster παίρνει την τιμή της συντεταγμένης X του κελιού *i* και καλείται η συνάρτηση AddCell η οποία προσθέτει το κελί *i στο* cluster. Έτσι έχει δημιουργηθεί ένα cluster που περιέχει το κελί *i*. Σε αντίθετη περίπτωση, δηλαδή αν δεν είναι το πρώτο κελί του cluster και υπάρχει overlap, του τελευταίου cluster με το κελί *i* τότε, καλείται η *AddCell* για να προστεθεί το κελί στο cluster και η Collapse στην οποία απαλείφονται τα overlaps μεταξύ των clusters.

Στην Function *Addcell* προστίθεται το βάρος του κελιού (αλγόριθμος *4.3.9.2,* γραμμή 24), στο συνολικό βάρος του cluster, το μήκος του κελιού στο συνολικό μήκος του cluster (αλγόριθμος *4.3.9.2,* γραμμή 26), και το αποτέλεσμα της πράξης qc, που θα οδηγήσει στην βέλτιστη θέση του cluster (αλγόριθμος *4.3.9.2,* γραμμή 25).

Στην function *Collapse* απαλείφονται τα overlaps μεταξύ των clusters με dynamic programming. Πρώτα τοποθετείται το cluster στην γραμμή, δηλαδή η ιδιότητα xc(c) παίρνει την τιμή qc(c)/ec(c), όπως εξηγήθηκε παραπάνω. Πρώτα ελέγχεται αν η συντεταγμένη του cluster xc(c) είναι εντός ορίων του chip (αλγόριθμος *4.3.9.2,* γραμμή 34 και 35). Υπάρχει περίπτωση η qc(c) να έχει πάρει αρνητική τιμή, αλλά και να έχει πάρει τιμή μεγαλύτερη από τις διαστάσεις του chip. Γι αυτό ο έλεγχος που γίνεται είναι, αν στην συντεταγμένη του cluster προστεθεί το συνολικό μήκος του cluster, το αποτέλεσμα δεν πρέπει να ξεπερνάει τα όρια του chip. Στην περίπτωση λοιπόν που η xc(c) είναι αρνητική, τότε ορίζεται ίση με το 0. Αν η xc(c) προστεθεί με το συνολικό μήκος του cluster και είναι εκτός ορίων τότε επαναφέρεται μέσα στα όρια του chip.

Σε κάθε περίπτωση πρέπει να ελεγχθεί αν υπάρχει overlap με προηγούμενο cluster όπως φαίνεται στο σχήμα *4.3.9.3* .




***Σχήμα 4.3.9.3 :*** Επικάλυψη των cluster

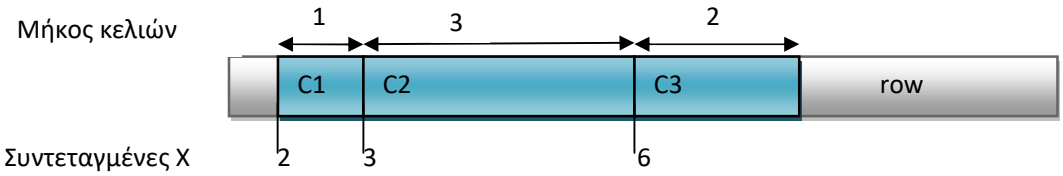
Σε αυτήν την περίπτωση τα cluster, που έχουμε το overlap, θα πρέπει να ενσωματωθούν, υπολογίζοντας πάλι την βέλτιστη θέση του νέου cluster.

Στην function *AddCluster* ενσωματώνεται το cluster c, με το προηγούμενο cluster Pc. Προστίθεται το βάρος του cluster c στο βάρος του προηγούμενου cluster Pc και στο μήκος του προηγούμενου cluster Pc, προστίθεται το μήκος του cluster c. Η qc(c) υπολογίζεται και πάλι, για να βρεθεί η βέλτιστη θέση του cluster, αυτή τη φορά όμως προστίθεται στην qc(PC) του προηγούμενου cluster , η qc(C) του cluster αφαιρώντας το γινόμενο του βάρους του cluster c και του μήκους του προηγούμενου cluster Pc (αλγόριθμος *4.3.9.2,* γραμμή 30).

Αφού η *AddCluster* ολοκληρωθεί, καλείται αναδρομικά η *collapse* και θα καλείται όσο υπάρχει προηγούμενο cluster και όσο αυτό έχει overlap με το cluster c.

Αφού λοιπόν ολοκληρωθεί η διαδικασία του clustering, μέσα σε μια επανάληψη, η οποία ολοκληρώνεται όταν προσπελαστούν όλα τα clusters της γραμμής, υπολογίζεται η συντεταγμένη του κάθε κελιού *i,* που βρίσκεται μέσα σε ένα cluster, αφαιρώντας από την συντεταγμένη X του cluster το μήκος του κάθε

κελιού. Δηλαδή αν η συντεταγμένη X του cluster είναι 2 και έχει 3 κελιά, το κελί C1 έχει μήκος 1 το C2 έχει μήκος 3 και το C3 έχει μήκος 2, τότε η συντεταγμένη του κελιού C1 είναι 2. Η συντεταγμένη του C2 είναι 2+1=3 και τέλος η συντεταγμένη του C3 είναι 3+3=6 (σχήμα 4.3.9.4).




***Σχήμα 4.3.9.4 :*** Συντεταγμένες των κελιών που βρίσκονται μέσα στο cluster

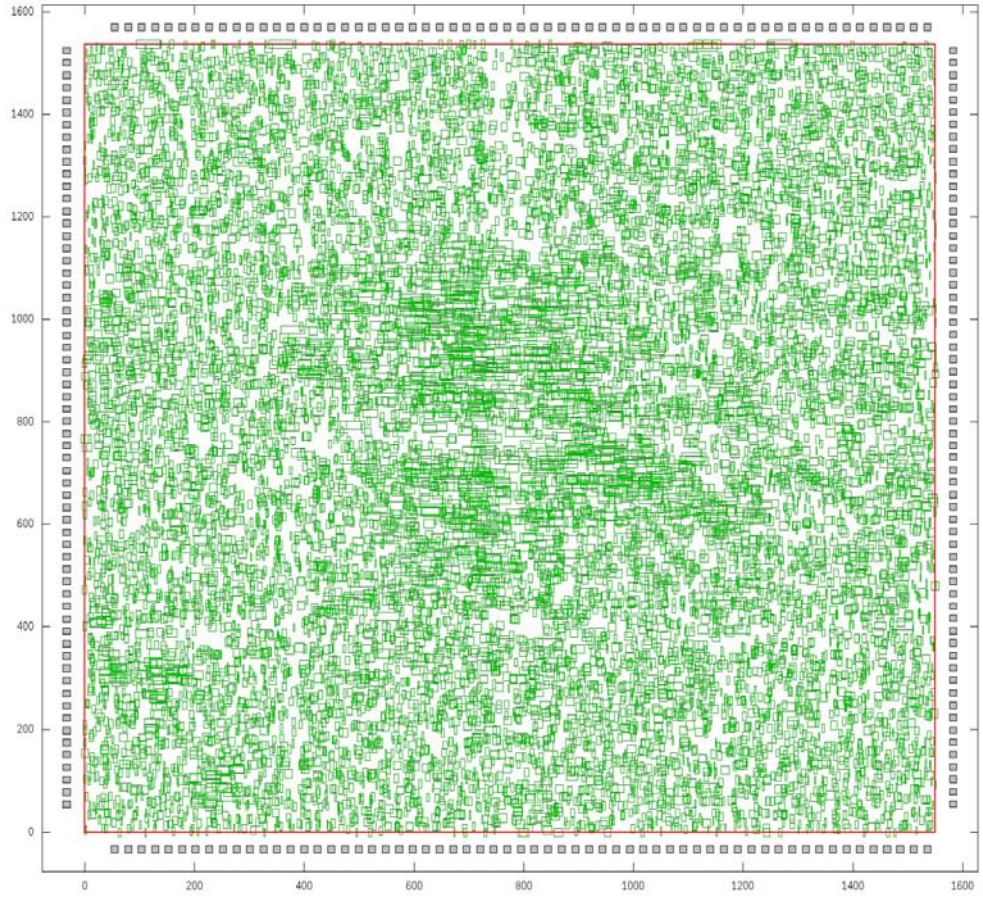
Έτσι ολοκληρώνεται η Placerow και αν εκτελέστηκε ως trial δεν αποθηκεύεται το αποτέλεσμα, αλλά συνεχίζεται να καλείται μέχρι το κόστος Y να γίνει μεγαλύτερο του κόστους X, κρατώντας το μικρότερο κόστος το οποίο σηματοδοτεί και την καλύτερη γραμμή.

Αν η Placerow εκτελέστηκε ως final, τότε τοποθετεί το κελί στην βέλτιστη θέση στην καλύτερη γραμμή και αποθηκεύει τις νέες θέσεις των κελιών τις γραμμής.

# 5 Πειραματικές μετρήσεις

## 5.1 Υλοποίηση

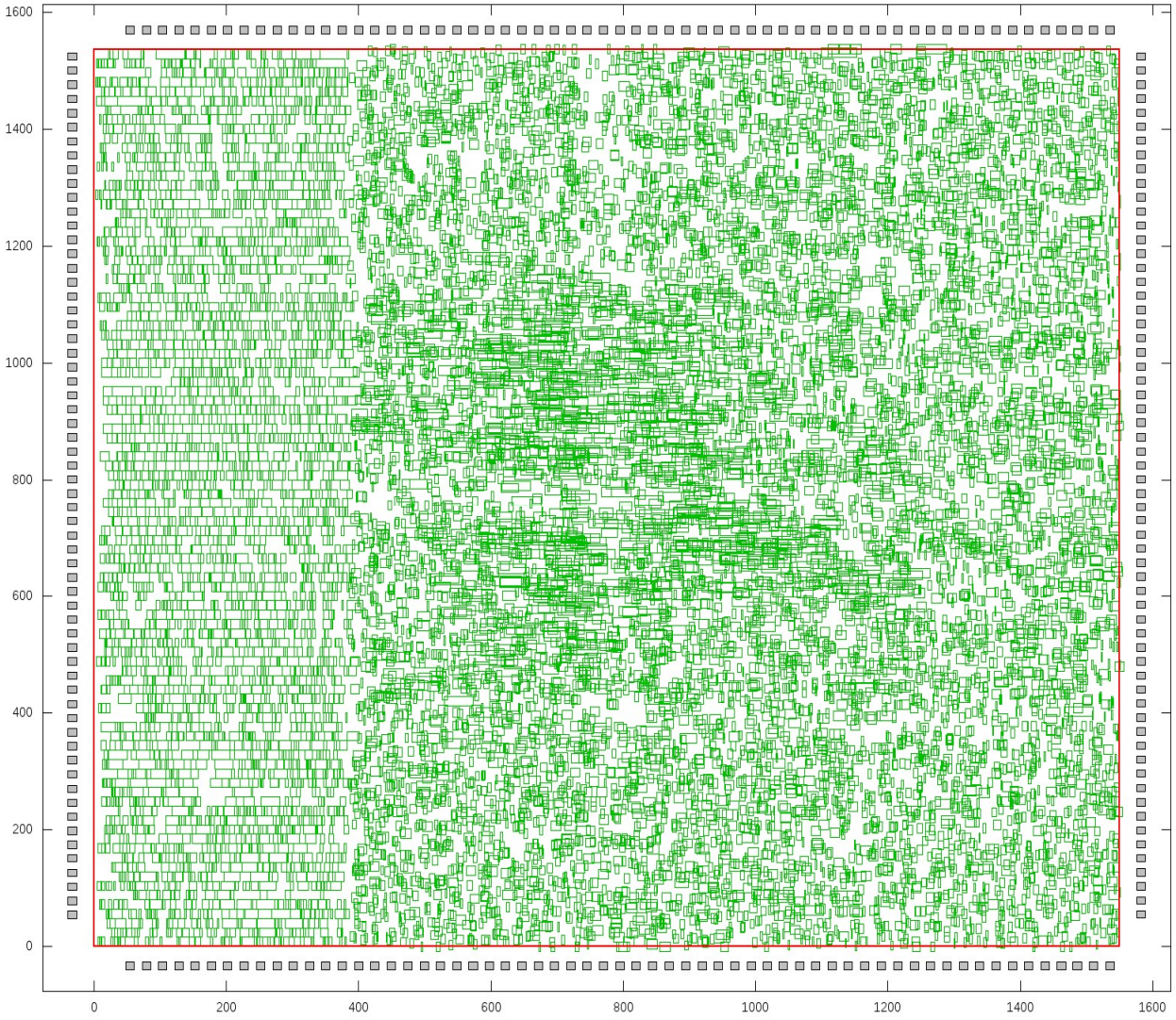
Ο Abacus υλοποιήθηκε σε γλώσσα C, παίρνοντας ως input ένα global Placement που υλοποιήθηκε με τον αλγόριθμο Gordian. Οι μετρήσεις έγιναν σε υπολογιστή με επεξεργαστή INTEL I5 μνήμη 8GB και λειτουργικό Ubuntu. Στην εικόνα *5.1.1* βλέπουμε οπτικοποιημένη, την αρχική κατάσταση του benchmark IBM01, όπως το πήρε ο Abacus από το Global placement. Παρατηρείται μια αταξία στα κελιά, υπάρχουν πάρα πολλά overlaps και κάποια από αυτά είναι και εκτός ορίων του chip. Κατά την εκτέλεση του, κρατήθηκε μια αποτύπωση του layout, όταν είχε ολοκληρωθεί το ¼ των κελιών προς νομιμοποίηση, μια στο ½ , μια στα ¾ και στην τελική του μορφή.




***Εικόνα 5.1.1 :*** Αρχική κατάσταση του ibm01

### *Αποτύπωση στο ¼ του πλήθους των κελιών*

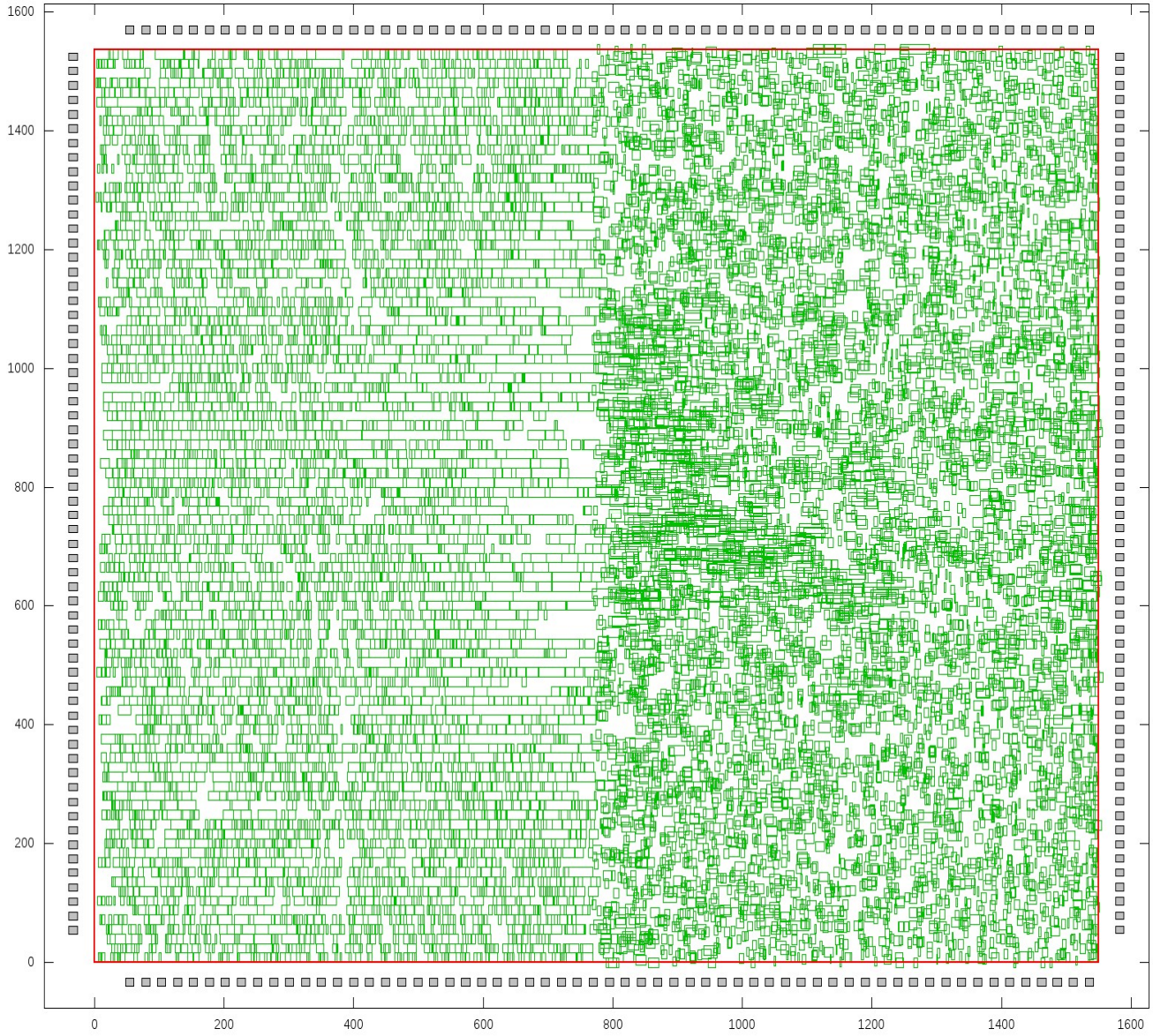
Τα κελιά είναι ταξινομημένα σύμφωνα με την συντεταγμένη X της κάτω αριστερής γωνίας. Η εικόνα *5.1.2* δείχνει το layout του chip όταν έχει σαρωθεί το 1/4 του πλήθους των κελιών. Φαίνεται καθαρά στην εικόνα ότι πόσο τακτοποιημένα είναι τα πρώτα κελιά. Επίσης ίσως μερικές γραμμές ακόμα δεν έχουν κανένα κελί, ή ελάχιστα κελιά, μέχρι το σημείο που αποτυπώθηκε το layout.




***Εικόνα 5.1.2 :*** Αποτύπωση στο ¼ του πλήθους των κελιών του ibm01

### *Αποτύπωση στο ½ του πλήθους των κελιών*

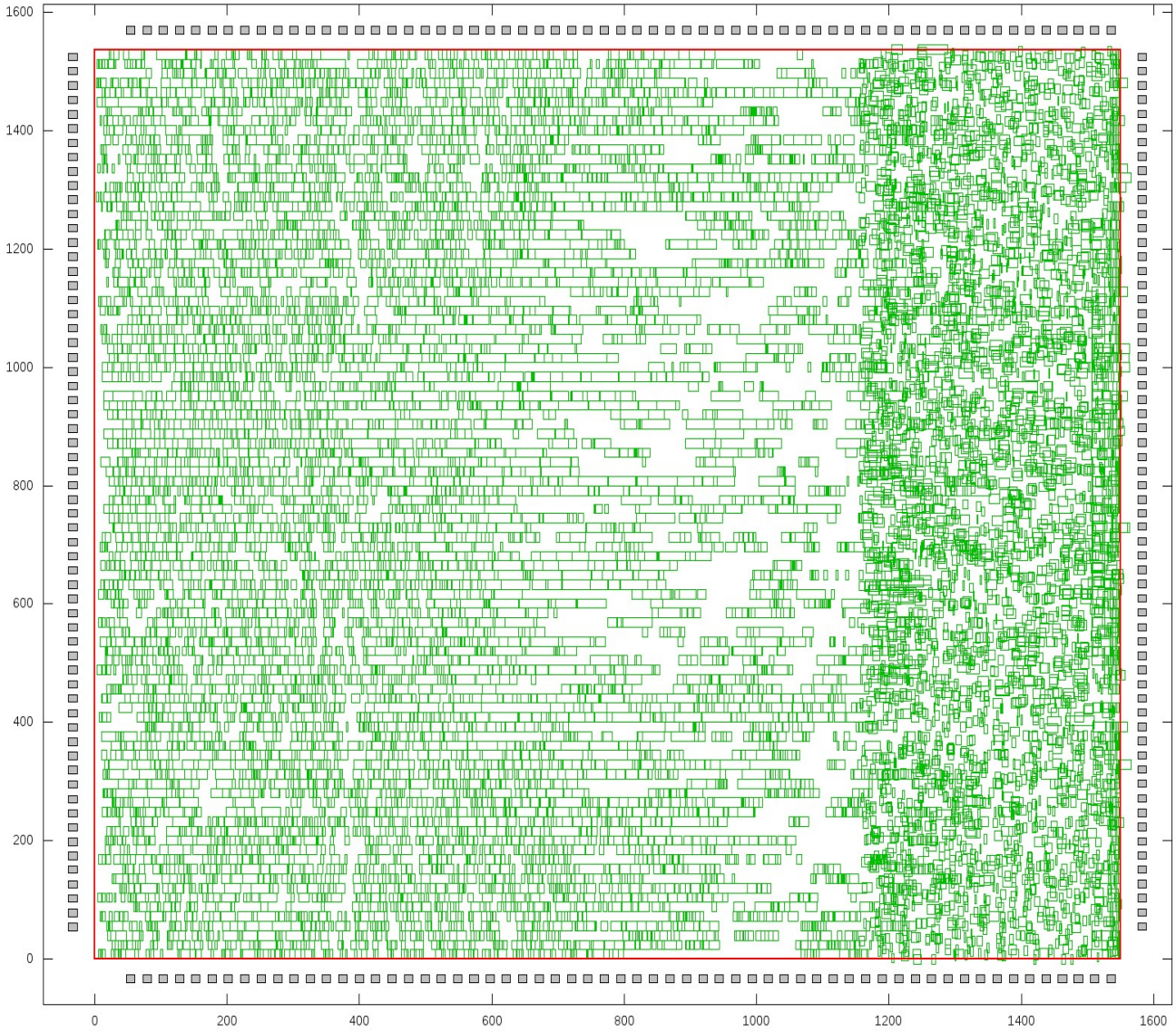
Η επόμενη εικόνα *5.1.3* δείχνει το layout όταν έχει σαρώσει τα μισά κελιά. Μπορεί κανείς να παρατηρήσει τα clusters που δημιουργούνται σε κάθε γραμμή. Ακόμα φαίνεται ότι κάποιες γραμμές που είχαν ελάχιστα κελιά, στην προηγούμενη αποτύπωση, τώρα προς την μέση του chip αρχίζουν σιγά-σιγά να γεμίζουν.




***Εικόνα 5.1.3 :*** Αποτύπωση στο ½ του πλήθους των κελιών του ibm01

### *Αποτύπωση στο ¾ του πλήθους των κελιών*

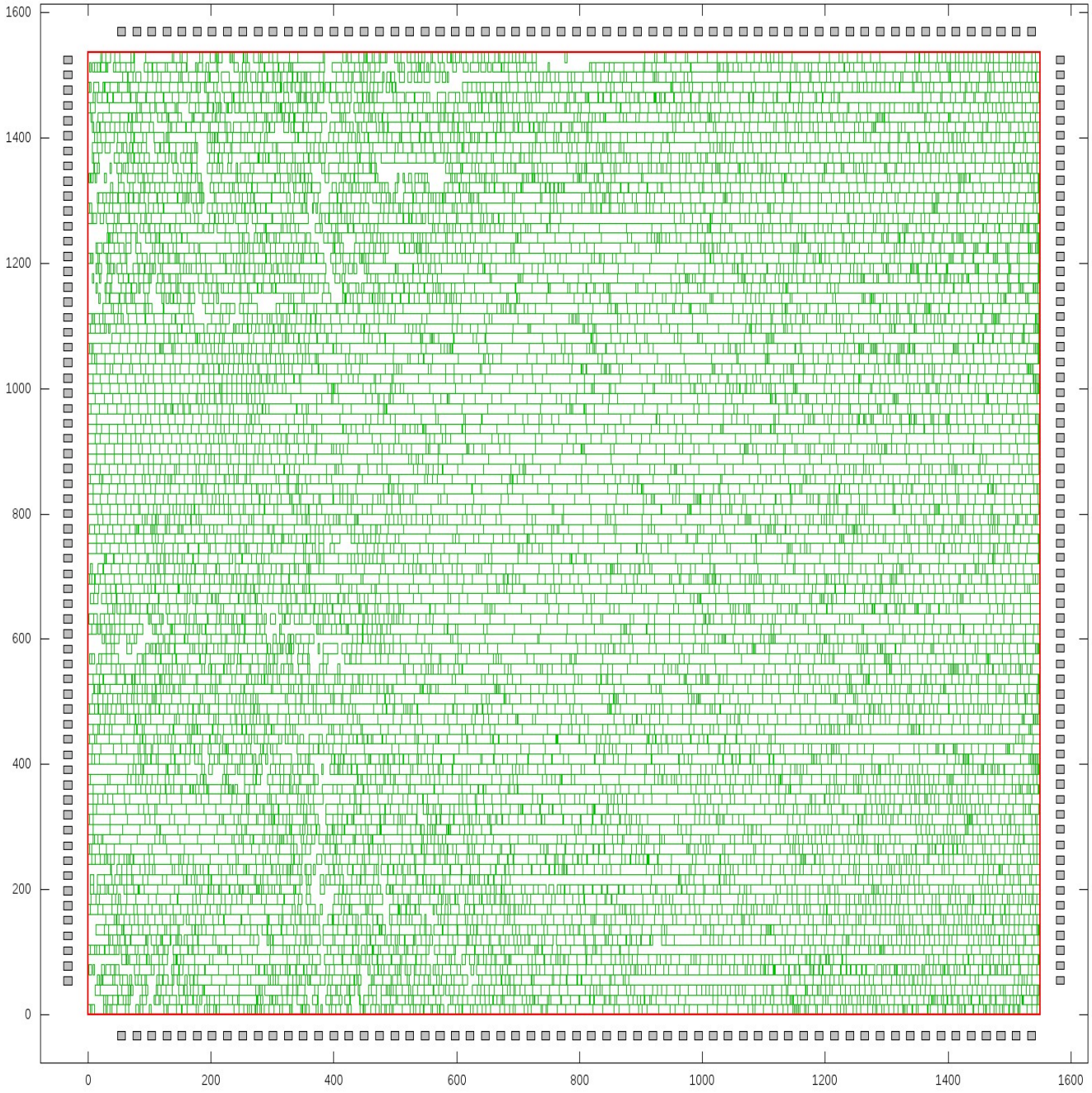
Πλησιάζουμε προς το τέλος και παρατηρούμε στην εικόνα *5.1.4* ότι σε κάποιες γραμμές, υπάρχουν πολλά κελιά τοποθετημένα και άλλες γραμμές έχουν πολύ ελεύθερο χώρο. Παρατηρούμε επίσης ότι τα κελιά που είναι δεξιά συσσωρευμένα, δεν έχουν σαρωθεί ακόμα και είναι αρκετά. Τα κελιά λοιπόν που πρόλαβαν και πήραν θέση θα μείνουν στην γραμμή αυτή αλλά ίσως αλλάξει λίγο η θέση τους, μέσα στη γραμμή, για να χωρέσει και κάποιο άλλο κελί. Τα κελιά που δεν χωράνε στην πιο κοντινή γραμμή τους ή στις πιο κοντινές γραμμές τους θα αναζητήσουν θέση σε πιο μακρινή γραμμή.




***Εικόνα 5.1.4 :*** Αποτύπωση στα ¾ του πλήθους των κελιών του ibm01

### *Αποτύπωση στο 100%  του πλήθους των κελιών*

Κι έτσι φτάνοντας στο τέλος (εικόνα *5.1.5*) βλέπουμε ότι όλες γραμμές, έχουν γεμίσει. Αυτό λοιπόν είναι και το τελικό layout  του chip.




***Εικόνα 5.1.5 :*** Αποτύπωση στο 100%  του πλήθους των κελιών του ibm01

## 5.2 Πειραματικές Μετρήσεις

Χρησιμοποιήθηκαν διπλά συνδεδεμένες λίστες για την αποθήκευση των clusters, τις κάθε γραμμής, και δυναμικοί πίνακες για την δομή των κελιών και για την δομή των γραμμών.

Η εφαρμογή εκτελέστηκε για τα benchmarks της IBM. Στον πίνακα παρουσιάζονται τα χαρακτηριστικά των συγκεκριμένων, benchmarks.

| ***Πίνακας 5.2.1 :*** Χαρακτηριστικά των Benchmarks | | | | |
|---|---|---|---|---|
| **Όνομα benchmark** | Αριθμός κελιών | Πλήθος Γραμμών | Πλήθος των nets | Πλήθος Των Pins |
| **ibm01** | 12506 | 96 | 14111 | 50566 |
| **ibm02** | 19342 | 109 | 19584 | 81199 |
| **ibm03** | 22853 | 121 | 27401 | 93573 |
| **ibm04** | 27220 | 136 | 31970 | 105859 |
| **ibm05** | 28146 | 139 | 28446 | 126308 |
| **ibm06** | 32332 | 126 | 34826 | 128182 |
| **ibm07** | 45639 | 166 | 48117 | 175639 |
| **ibm08** | 51023 | 170 | 50513 | 204890 |
| **ibm09** | 53110 | 183 | 60902 | 222088 |
| **ibm10** | 68685 | 234 | 75196 | 297567 |

Το συνολικό displacement μετρήθηκε με την απόσταση Manhattan

$$\sum_{\forall cell} (|x^{cell}_{original} - x^{cell}_{legalized}| + |y^{cell}_{original} - y^{cell}_{legalized}|)$$

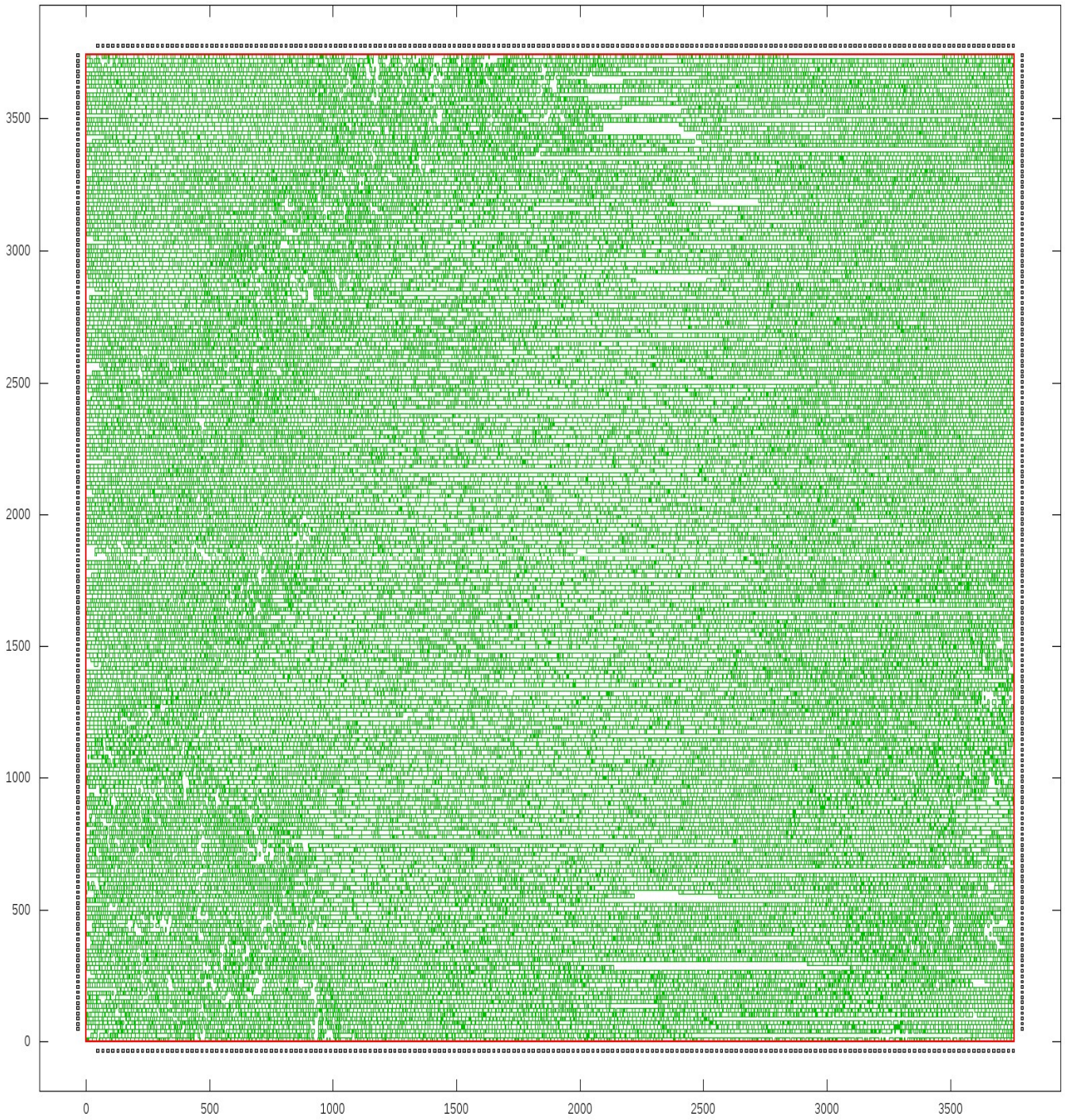
Στην αντικειμενική συνάρτηση (4) του Abacus, που υπολογίζει την βέλτιστη θέση του cluster, ως βάρος *e*, χρησιμοποιήθηκε το εμβαδόν του κελιού. Χρησιμοποιώντας ως βάρος το εμβαδόν, τα αποτελέσματα είναι πολύ καλύτερα, από ότι, αν δεν υπολογιστεί καθόλου το βάρος όπως αναφέρεται στο [14].

Στον πίνακα παρουσιάζονται οι μετρήσεις της εκτέλεσης του Abacus και για τα 10 benchmarks της IBM.

***Πίνακας 5.2.2 :*** *Μετρήσεις Abacus*

| bench mark | κελιά | overlaps | Displacement | Αρχικό wirelength | Τελικό wirelength | Χρόνος εκτέλεσης | Γραμμές | Διαστάσεις |
|---|---|---|---|---|---|---|---|---|
| **Ibm01** | 12506 | 21084 | 1346452 | 2679e6 | 4,767 | 635 | 96 | 1550x1536 |
| **Ibm02** | 19342 | 33921 | 1490592 | 6693e6 | 8,441 | 1051 | 109 | 1757x1744 |
| **Ibm03** | 22853 | 39852 | 2567584 | 8742e6 | 12,408 | 1412 | 121 | 1938x1936 |
| **Ibm04** | 27220 | 48460 | 1716651 | 10156e6 | 11,835 | 1169 | 136 | 2198x2176 |
| **Ibm05** | 28146 | 50437 | 3620578 | 18294e6 | 22,271 | 2716 | 139 | 2234x2224 |
| **Ibm06** | 32332 | 57154 | 3205475 | 1722e6 | 11,837 | 2788 | 126 | 2037x2016 |
| **Ibm07** | 45639 | 79742 | 3586937 | 15965e6 | 19,637 | 3234 | 166 | 2687x2656 |
| **Ibm08** | 51023 | 88148 | 4289280 | 14616e6 | 19,794 | 5028 | 170 | 2722x2720 |
| **Ibm09** | 53110 | 93497 | 5477573 | 15759e6 | 23,559 | 6786 | 183 | 2943x2920 |
| **Ibm10** | 68685 | 122227 | 12465182 | 32677e6 | 51,723 | 13102 | 234 | 3759x3744 |

Το ibm01 είναι το πιο μικρό από όλα τα benchmark και εκτελέστηκε σε έναν πολύ καλό χρόνο με ένα ικανοποιητικό displacement. Ομοίως ακολουθούν και τα υπόλοιπα benchmarks. Το ibm10 είναι το μεγαλύτερο από όλα. Στην εικόνα *5.2.1* φαίνεται το τελικό layout του ibm10.




***Εικόνα 5.2.1 :*** Αποτύπωση στο 100% του πλήθους των κελιών του ibm10

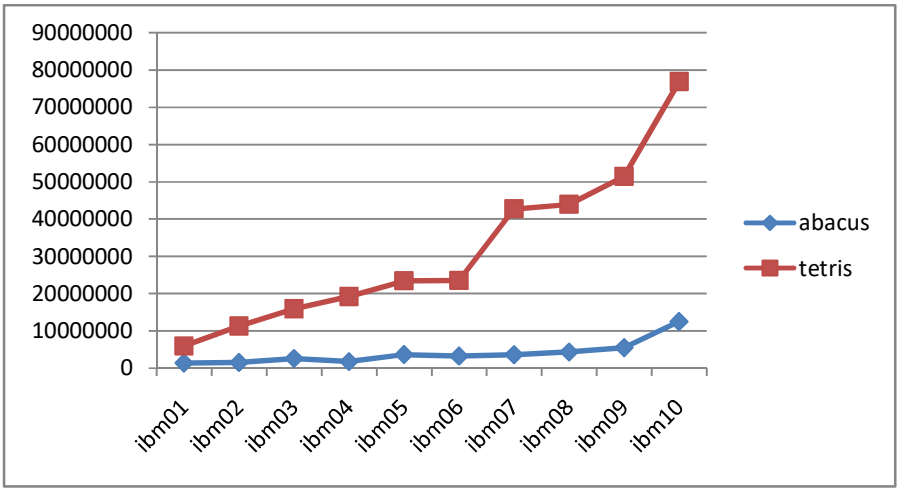
## 5.3 Σύγκριση Tetris - Abacus

Στον πίνακα γίνεται σύγκριση του Abacus που υλοποιήθηκε σε αυτήν την διπλωματική και του Tetris. Είναι γνωστό από την βιβλιογραφία ότι ο Abacus πετυχαίνει καλύτερο Displacement, αλλά έχει μεγαλύτερο χρόνο εκτέλεσης ενώ αντίθετα ο Tetris πετυχαίνει μικρότερο χρόνο εκτέλεσης αλλά μεγαλύτερο Displacement.

***Πίνακας 5.3.1 :*** *Συγκριτικά αποτελέσματα*

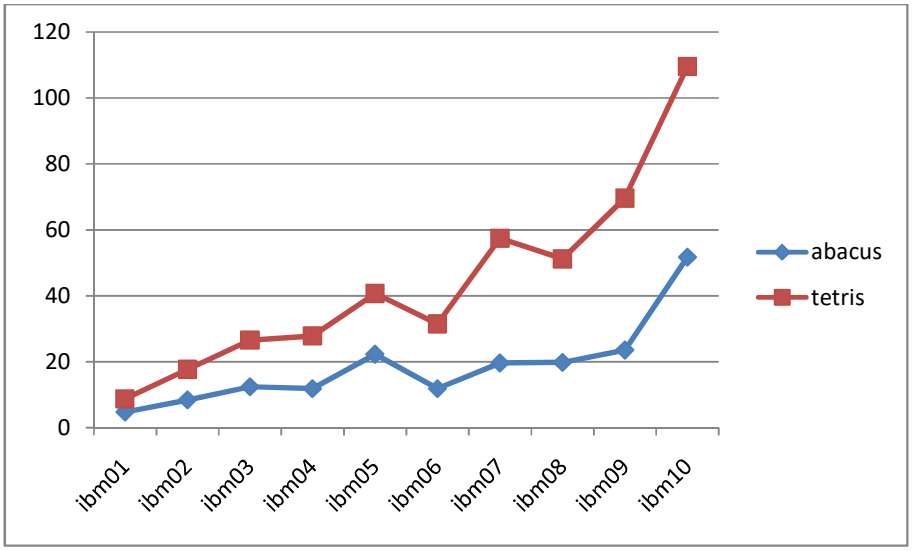
| Benchmark | ABACUS | | | TETRIS | | |
|---|---|---|---|---|---|---|
| | Execution Time (ms) | Displacement | HPWL (e6) | Execution Time (ms) | Displacement | HPWL (e6) |
| **ibm01** | 635 | 1346452 | 4,767 | 1552 | 5954671 | 8,758 |
| **ibm02** | 1051 | 1490592 | 8,441 | 3750 | 11308402 | 17,679 |
| **ibm03** | 1412 | 2567584 | 12,408 | 5189 | 15900396 | 26,524 |
| **ibm04** | 1169 | 1716651 | 11,835 | 8123 | 19213494 | 27,809 |
| **ibm05** | 2716 | 3620578 | 22,271 | 8078 | 23427376 | 40,673 |
| **ibm06** | 2788 | 3205475 | 11,837 | 10993 | 23507953 | 31,451 |
| **ibm07** | 3234 | 3586937 | 19,637 | 27293 | 42702124 | 57,456 |
| **ibm08** | 5028 | 4289280 | 19,794 | 39537 | 43915373 | 51,182 |
| **ibm09** | 6786 | 5477573 | 23,559 | 45259 | 51426880 | 69,617 |
| **ibm10** | 13102 | 12465182 | 51,723 | 104524 | 76839074 | 109,581 |

Ο Abacus είναι καλύτερος από τον Tetris στο Displacement σε όλα τα benchmarks (εικόνα 5.3.1). Αυτό είναι και το σημείο που υπερτερεί ο Abacus από τον Tetris και το συναντάμε σε όλη την βιβλιογραφία.



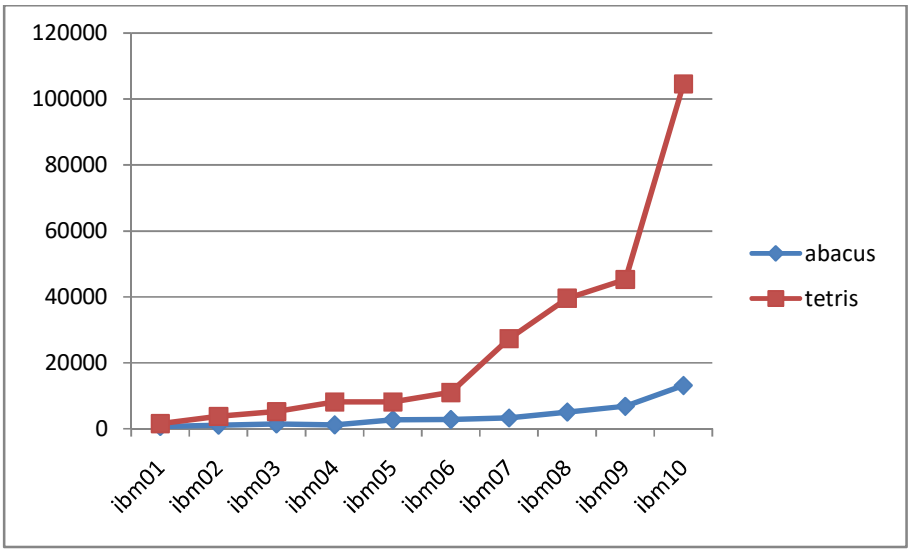

***Εικόνα 5.3.1 :*** Displacement

Μεγάλη διαφορά υπάρχει και στο wirelength, με τον Abacus να υπερτερεί. (εικόνα 5.3.2).




**Εικόνα *5.3.2* :** Wirelength

Η τελευταία παρατήρηση είναι ο χρόνος εκτέλεσης των δύο αλγορίθμων (εικόνα 5.3.3). Γενικά ο Abacus είναι πιο αργός από τον Tetris. Στην υλοποίηση όμως αυτή ο Abacus είναι πιο γρήγορος από τον Tetris.




**Εικόνα *5.3.3* :** Χρόνος εκτέλεσης

# 6 Επίλογος

## 6.1 Συμπεράσματα

Σε αυτήν τη διπλωματική παρουσιάστηκε η διαδικασία σχεδίασης ολοκληρωμένων κυκλωμάτων. Περιγράφηκε η φυσική σχεδίαση κάνοντας μια αναφορά στον αλγόριθμο Gordian [26] ο οποίος χρησιμοποιήθηκε στην εφαρμογή για την παραγωγή του Global placement. Εστίασε στην νομιμοποίηση ολοκληρωμένων κυκλωμάτων περιγράφοντας την διαδικασία. Ακόμα περιγράφηκαν αλγόριθμοι νομιμοποίησης ολοκληρωμένων κυκλωμάτων.

Ο κύριος στόχος της Διπλωματικής ήταν από ένα Global Placement με πολλά overlaps, η εφαρμογή που θα υλοποιούνταν σε γλώσσα C, να παραγάγει ένα layout χωρίς overlaps και όλα τα κελιά να είναι εντός των ορίων του chip. Επιπρόσθετα θα έπρεπε η εφαρμογή να πετυχαίνει καλύτερο Displacement από αυτό που πετυχαίνει ο Tetris [23]. Ο αλγόριθμος που επιλέχθηκε για την υλοποίηση της εφαρμογής είναι ο Abacus [14].

Παράχθηκε ένα layout, χωρίς overlaps, όλα τα κελιά είναι εντός ορίων του chip και φυσικά μικρότερο displacement από τον Tetris[23]. Επίσης πέτυχε καλύτερο χρόνο εκτέλεσης από τον Tetris, παρά το ότι στην βιβλιογραφία το μειονέκτημα του Abacus έναντι του Tetris είναι ο χρόνος εκτέλεσης.

## 6.2 Μελλοντικές επεκτάσεις

Θα μπορούσε να δοθεί μεγαλύτερο βάρος στο, σε πόσα nets ανήκει το κελί. Σε όσα περισσότερα nets ανήκει τόσο μεγαλύτερο θα είναι και το wirelength. Άρα τα κελιά που ανήκουν σε πολλά nets, όταν μετακινούνται αυξάνουν το wirelength. Θα μπορούσε να συνδυάσει ακόμα ένα κριτήριο. Να βάλει σε προτεραιότητα τα κελιά που έχουν τα περισσότερα pins. Όσα περισσότερα pins τόσες περισσότερες συνδέσεις απαιτούνται.

Η πιο σημαντική βελτίωση που θα μπορούσε να γίνει, είναι να εκτελεστεί το πρόγραμμα παράλληλα. Δηλαδή την ίδια στιγμή να μπορεί να ελέγχει 2 και 3 κελιά. Όμως υπάρχουν κάποιοι περιορισμοί. Δεν μπορεί τα κελιά να είναι γειτονικά γιατί αυτό σημαίνει ότι οι γραμμές που θα χρειαστεί να σαρώσουν για να βρουν την

βέλτιστη θέση θα είναι ίδιες. Με αυτόν τον τρόπο το πρόγραμμα θα εκτελούνταν πολύ γρηγορότερα, όπου ο χρόνος εκτέλεσης είναι ένα θέμα που μπορεί να βελτιωθεί, αφού για ένα VLSI τα κελιά που θα πρέπει να νομιμοποιηθούν μπορεί να είναι εκατομμύρια. Στην συγκεκριμένη διπλωματική στα benchmarks που χρησιμοποιήθηκαν έχουν μερικές χιλιάδες κελιά.

# ΒΙΒΛΙΟΓΡΑΦΙΑ


[1] https://en.wikipedia.org/wiki/Transistor_count

[2] https://www.researchgate.net/publication/221154679_A_hierarchical_bin-based_legalizer_for_standard-cell_designs_with_minimal_disturbancehttp://tvxs.gr/news/san-simera/radiofono-tesla-kai-oxi-markoni-toy-giorgoy-stamkoy

[3] https://physicsgg.me

[4] https://en.wikipedia.org/wiki/Physical_design_(electronics)

[5] http://www.vlsijunction.com/2015/08/floor-planning.html

[6] http://www2.inf.uos.de/papers_html

[7] http://dl.ifip.org/db/conf/vlsi/vlsisoc2010s/HoL10a.pdf

*[8] Δαδαλιάρης Αντώνιος Πανεπιστήμιο Θεσσαλίας "Χωροθέτηση ολοκληρωμένων κυκλωμάτων με παραμέτρους αξιοπιστίας"*

[9] Michael Gertz and Stephen Wright. *"Object-orientedsoftware for quadratic programming."* ACMTransactions on Mathematical Software, 29(1):58–81,March 2003.

[10] Ameya Agnihorti, Mehmet Can Yildiz, AteenKhatkhate, AjitaMathur, Satoshi Ono, and Patrick H. Madden, *"Fractional cut: Improved recursive bisection placement",* Proc. ICCAD, pp. 307-310, 2003.

[11] Andrew B. Kahng, Igor L. Markov, and SheriefReda, *"On legalization of rowbased placements,"* Proc. GLS-VLSI, pp. 214-219, 2004.

[12] Heng et al., "*A VLSI Artwork Legalization Technique Based on a New Criterion of Minimum Layout Perturbation*" IBM Corporation, T. J. Watson Research Center, pp. 116-121.

[13] Peter Spindler, Ulf Schlichtmann, and Frank M. Johannes, *"Abacus: Fast legalization of standard cell circuits with minimal movement,"* Proc. ISPD, pp. 47-53, 2008.

[14] http://www.ispd.cc/

[15] https://en.wikipedia.org/wiki/Simplex_algorithm

*[16]* Junying Hu Sch. of Mech. Electron. & Inf. Eng., China Univ. of Min. & Technol., Beijing, China Quan Zhou ; Wenchao Gao ; Xu Qian ; Qiang Zhou *"An effective legalization approach based on multiple ordering"*

*[17]* Tsung-Yi Ho and Sheng-Hung Liu Department of Computer Science and Information Engineering National Cheng Kung University *"Fast Legalization for Standard Cell Placement with Simultaneous Wirelength and Displacement Minimization"*

[18] Yu-Min Lee ; National Chiao Tung University, Hsinchu, Taiwan ; Tsung-You Wu ; Po-Yi Chiang *"A hierarchical bin-based legalizer for standard-cell designs with minimal disturbance"*

*[19]* U. Brenner ; Res. Inst. for Discrete Math., Univ. of Bonn, Germany ; J. Vygen *"Legalizing a placement with minimum total movement"*

[20] Andrew B. Kahng, Paul Tucker and Alex Zelikovsky, *"Optimization of Linear Placements for Wirelength Minimization with Free sites,"*Proc. ASPDAC99, pp. 241-244, 1999.

[21] Andrew B. Kahng, Igor L. Markov, and Sherief Reda, *"On legalization of row-based placements,"*Proc. GLSVLSI, pp. 214-219, 2004.

[22] D. Hill. "*Method and system for high speed detailed placement of cells within an integrated circuit design"*, 04 2002. Tetris

[23] Julia Casarin Puget, Guilherme Flach, Marcelo Johann, Ricardo Reis, PGMicro - Graduate Program on Microelectronics, Federal University of Rio, Grande do Sul, Porto Alegre, Brazil *"Jezz: An Effective Legalization Algorithm for Minimum Displacement"*

[24] Ulrich Brenner, *"BonnPlace Legalization: Minimizing Movement by Iterative Augmentation"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v.32 n.8, p.1215-1227, August 2013

[25] Kleinhans, J.M.; Sigl, G.; Johannes, F.M.; Antreich, K.J.; , *"GORDIAN: VLSI placement by quadratic programming and slicing optimization"* ComputerAided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.10, no.3, pp.356-365, Mar 1991

[26] CARL SECHEN AND ALBERTO SANGIOVANNI-VINCENTELLI, FELLOW, "The TimberWolf Placement and Routing Package "

[27] Jarrod A. Roy, David A, Igor L. Markov. Papa a Capo: Congestion-driven Placement for Standard-cell and RTL Netlists with Incremental Capability

[28] https://en.wikipedia.org/wiki/K-medoids