Standard cells placement algorithm for integrated circuits.

Διπλωματική Εργασία

Οικονόμου Παναγιώτης
Οκτώβριος, 2008

Standard cells placement algorithm for integrated circuits.

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον καθηγητή και πρόεδρο του τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων και βασικό επιβλέποντα της πτυχιακής αυτής εργασίας κ.Γεώργιο Σταμούλη που μου έδωσε την ευκαιρία να πραγματοποιήσω αυτή την μελέτη. Η υποστήριξή του, η αμέριστη συμπαράστασή του, αλλά και οι διαρκείς και εύστοχες υποδείξεις του βοήθησαν στην έγκαιρη ολοκλήρωση αυτής της μελέτης.

Επιπρόσθετα, θα ήθελα να ευχαριστήσω τον έταιρο επιβλέποντα καθηγητή και επισκέπτη καθηγητή στο τμήμα, κ.Νέστορα Ευμορφόπουλο για τις συμβουλές του στο μαθηματικό υπόβαθρο που χρησιμοποιήθηκε στις μετρήσεις που παρουσιάζονται και τον διδακτορικό φοιτητή του τμήματος Αντώνη Δαδαλιάρη για την συμπαράστασή του και τη διευκόλυνση στη κατανόηση βασικών εννοιών του Placement .

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου που μου συμπαραστάθηκε σε όλη την διάρκεια της εκπόνησης αυτής της εργασίας.

Standard cells placement algorithm for integrated circuits.

ΠΕΡΙΕΧΟΜΕΝΑ

1. Εισαγωγή

1.1 Περιγραφή του προβλήματος.....**6**

1.2 Στόχος της εργασίας & Σημασια Placement.....**9**

2. Έννοιες

2.1 Ολοκληρωμένα κυκλώματα.....**10**

2.2 Cell.....**10**

2.3 Blocks.....**10**

2.4 Standard Cell.....**10**

2.5 Mapping.....**11**

2.6 Place and Route.....**11**

3. Υλοποίηση του αλγορίθμου.

3.1 Γλώσσες προγραμματισμού που χρησιμοποιήθηκαν καθώς και τρόποι προσομοίωσης, βιβλιοθήκες γλώσσας.....**14**

3.2 Βασικά βήματα υλοποίησης του αλγορίθμου.....**15**

4. Χρήση C++

4.1 Βασικές Βιβλιοθήκες και συναρτήσεις.....**25.**

4.2 Γραφική απεικόνιση του αλγορίθμου.....**26**

4.3 Γραφική απεικόνιση του αλγορίθμου σε standard cell τεχνολογία...**26**

5. ***Πειραματικές Μετρήσεις και Αποτελέσματα***

5.1 Συγκριτικά Αποτελέσματα και Συμπεράσματα.....**27**

***Βιβλιογραφία*.....43**

***Κώδικας αλγορίθμου Placement*.....44**

1.Εισαγωγή

1.1 Περιγραφή του προβλήματος

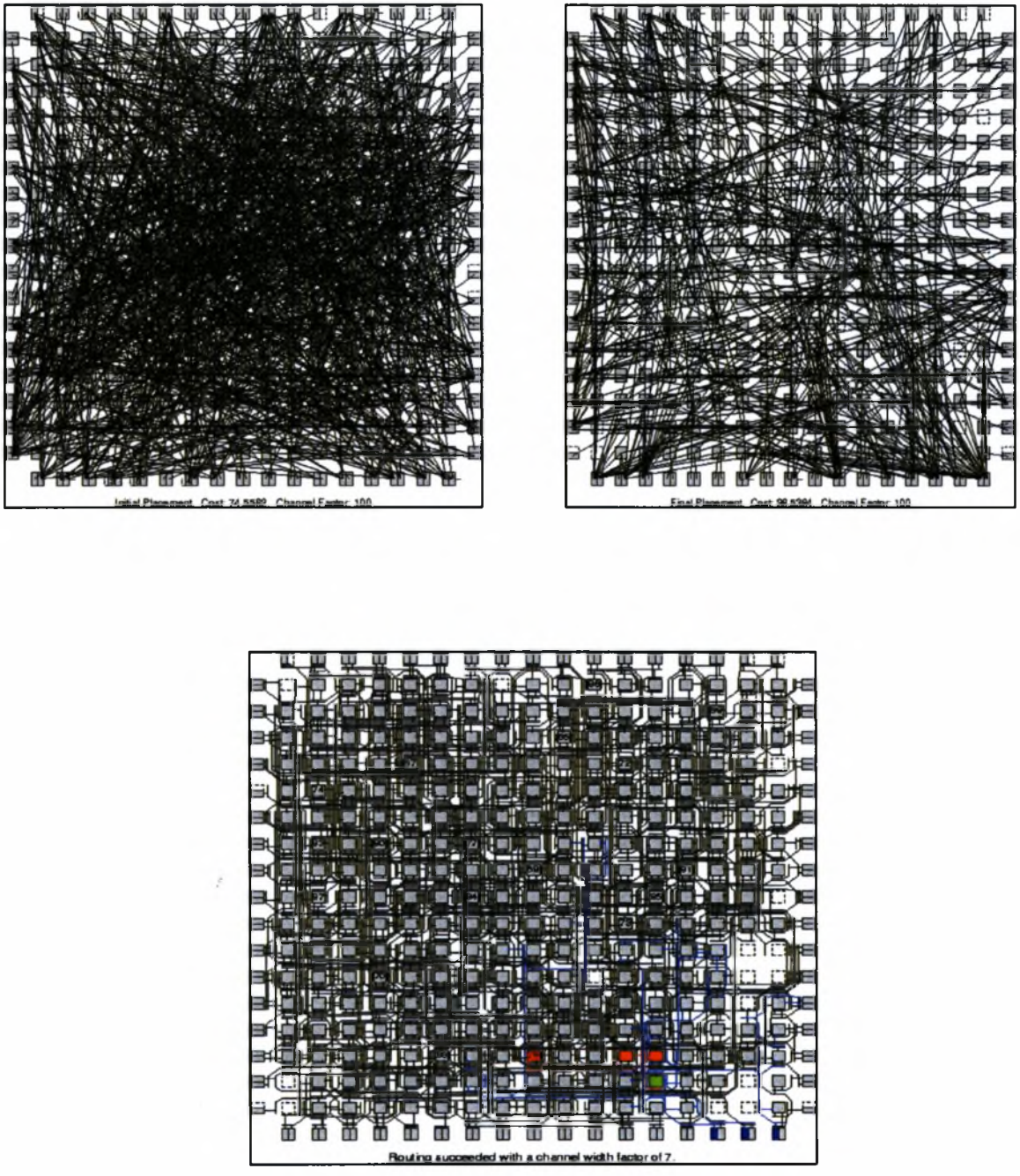
Η έρευνα πάνω στον τομέα των ψηφιακών κυκλωμάτων μέσα στο πέρασμα των δεκαετιών, από την κατασκευή του πρώτου ηλεκτρονικού υπολογιστή μέχρι και σήμερα, έχει αναθεωρήσει τους βασικούς στόχους και σκοπούς της ουκ ολίγες φορές.

Αρχικά, βασικό μέλημα των επιστημόνων ήταν η λειτουργικότητα των κυκλωμάτων, μεταγενέστερα τέθηκε όμως ως θέμα το υλικό κατασκευής των κυκλωμάτων και η «έκταση» τους.

Την τελευταία εικοσαετία, με την κατακόρυφη αύξηση της χρήσης των υπολογιστων σε ποικιλες εφαρμογές στην καθημερινότητά μας, έχει παρατηρηθεί μια αυξανόμενη απαίτηση για σχεδιάσεις μεγάλης περιεκτικότητας σε αριθμό τρανζιστορ οι οποίες θα καταλαμβάνουν τον μικρότερο δυνατό χώρο.

Γίνεται, επομένως εύκολα αντιληπτό το πόσο μεγάλης σημασίας είναι ο βέλτιστος τρόπος κατανομής των πυλών που αποτελούν ένα κύκλωμα, πάνω στο ολοκληρωμένο.

Η διαδικασία του placement, λοιπόν, θεωρείται ως ένα από τα πλέον ενεργά ζητήματα στην βιομηχανία κατασκευής ημιαγωγών. Ένα απλό παράδειγμα της βαρύνουσας σημασίας της διαδικασίας του placement στη συνολική ροή σχεδίασης (design flow) φαίνεται στο ακόλουθο σχήμα όπου βλέπουμε σταδιακά μια τυχαία αρχικοποίηση του κυκλώματος, το τελικό αποτέλεσμα του placement και ακολούθως το συνολικό αποτέλεσμα μετά το πέρας της διαδικασίας του routing.



ΣΧΗΜΑ 1.1: Συνολική ροή της ακολουθούμενης διαδικασίας Place & Route

Standard cells placement algorithm for integrated circuits.

Με βάση, λοιπόν, αυτή την σύγχρονη απαίτηση και παραδοχή έγινε η ακόλουθη προσπάθεια υλοποίησης ενός αλγορίθμου τοποθέτησης κελιών με αποτελέσματα που προσεγγίζουν τον βέλτιστο δυνατό τρόπο.

Standard cells placement algorithm for integrated circuits.

1.2 Στόχος της εργασίας

Βασικός στόχος αυτής της εργασίας είναι η υλοποίηση ενός αλγορίθμου ο οποίος θα πραγματοποιεί την βέλτιστη τοποθέτηση των εκάστοτε κυκλωματικών στοιχείων ώστε αυτά να καταλαμβάνουν το μικρότερο δυνατό χώρο πάνω στο ολοκληρωμένο.

Η υλοποίηση του αλγορίθμου έγινε με χρήση της γλώσσας προγραμματισμού C και κάποιων επιμέρους στοιχείων και συναρτήσεων της C++. Η ανάπτυξη του κώδικα και η αποσφαλμάτωσή του πραγματοποιήθηκε σε περιβάλλον UNIX προκείμένου να βελτιστοποιηθεί η κατανάλωση μνήμης και η δυνατότητα μεταφερσιμότητας (mobility) του κώδικα.

2.Βασικές Έννοιες

Στην ακόλουθη ενότητα ακολουθεί μια συνοπτική περιγραφή των εννοιών που θα μας απασχολήσουν.

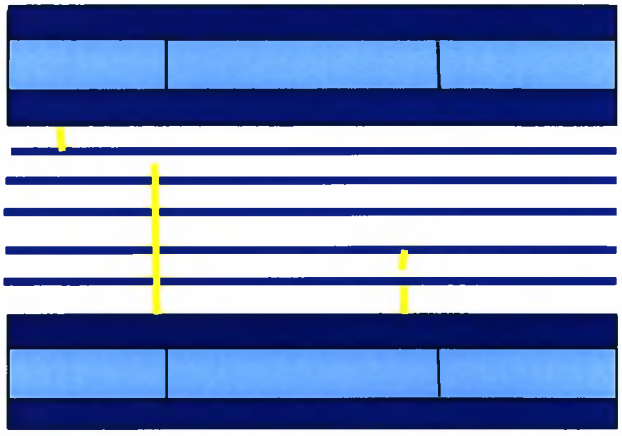
- **Ολοκληρωμένα Κυκλώματα (Integrated Circuits):** Η πλειοψηφία των κλασσικών κυκλωματικών στοιχείων υλοποιείται με την χρήση ενός μικρού αριθμού τρανζίστορ, η σύγχρονη, όμως τεχνολογία απαιτεί σχεδιάσεις που περιέχουν εκατομμύρια τρανζίστορ (όπως για παράδειγμα ένας σύγχρονος επεξεργαστής). Η πολυπλοκότητα των ολοκλήρωμένων αυτών κυκλωμάτων αποτελεί την μεγαλύτερη τροχοπέδη στην σχεδίασή τους και ο μόνος τρόπος αντιμετώπισης του προκείμενου προβλήματος είναι η χρήση ρητών και άρτια οργανωνμένων τεχνικών.

- **Κελιά (Cells):** Στην παρούσα μελέτη θεωρήσαμε ως κελί κάθε πύλη που χρησιμοποιήθηκε γνωρίζοντας επακριβώς τα βασικά χαρακτηριστικά της (μήκος και ύψος).

- **Μπλοκ (Blocks):** Με τον όρο μπλοκ περιγράφουμε κάθε ομαδοποιημένη συστοιχεία κελιών.

- **Τυποποιημένα Κελιά (Standard Cells):** Στα τυποποιημένα κελιά οι γραμμές των λογικών πυλών συνδέονται με καλώδια που δημιουργούνται στα κανάλια δρομολόγησης μεταξύ των γραμμών των πυλών. Μας παρέχεται, επιπρόσθετα η δυνατότητα χρησιμοποίησης

πολλών ειδών πυλών σε οποιδήποτε ολοκληρωμένο κύκλωμα. Τέλος, οι διαθέσιμες πύλες δομούνται εκ των προτέτρων και αποθηκεύονται σε μια βιβλιοθήκη η οποία μπορεί να ανακληθεί ανά πάσα ώρα και στιγμή προκειμένου να χρησιμοποιηθούν τα στοιχεία της. Στο ακόλουθο σχήμα παρουσιάζεται η τυπική μορφή ενός τυποποιημένου κελιού.

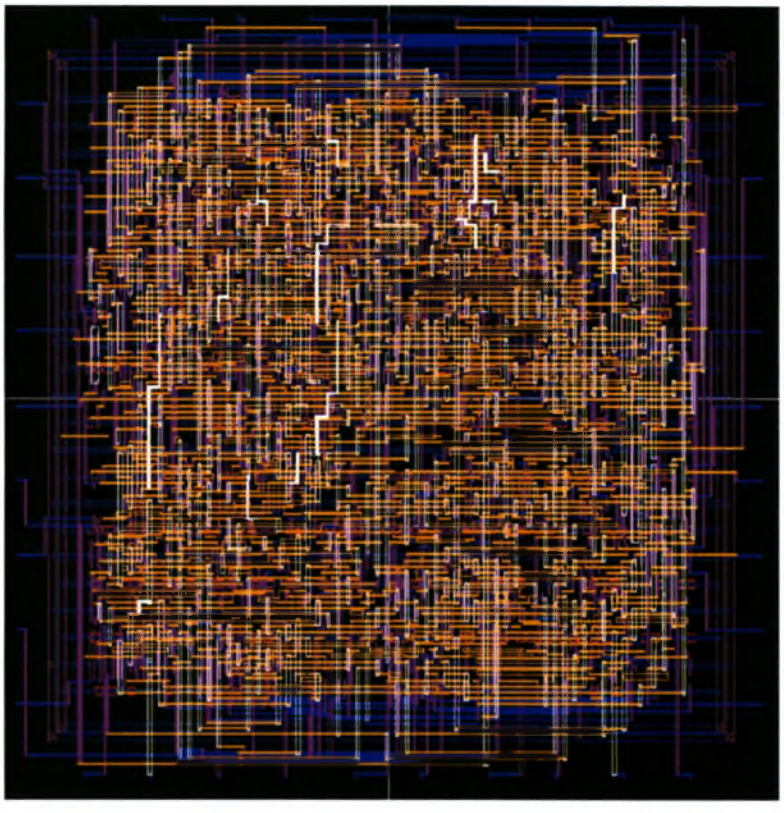


ΣΧΗΜΑ 2.1: Τυποποιημένο Κελί (Standard Cell)

Οι μπλε και οι κίτρινες γραμμές αντιπροσωπεύουν δυο διαφορετικά επίπεδα μετάλλοου. Αυτή η οργάνωση καθιστά δυνατή την τοποθέτηση γραμμών σε πολλαπλα επίπεδα στο χώρο χωρίς να συγκρούονται τα επίπεδα μεταλλικών γραμμών του ενός επιπέδου με το άλλο.

Η υλοποίηση των standard cells για μικρές κυκλωματικές σχεδιάσεις έχει σημαντικό κόστος και γι□υτόν το λόγο βρίσκει εφαρμογή σε μεγάλα κυκλώματα όπου ο αριθμός των cells ξεπερνάει τα χίλια. Στο ακόλουθο σχήμα παρουσιάζεται μια ολοκληρωμένη σχεδίαση σε φυσικό επίπεδο με χρήση standard cells.

Standard cells placement algorithm for integrated circuits.



ΣΧΗΜΑ 2.2: Πλήρης σχεδίαση σε φυσικό επίπεδο με χρήση standard cells

- **Αντιστοίχηση (Mapping):** Με τον όρο mapping αναφερόμαστε στην μεταφορά/αντιστοίχιση της λογικής σύνθεσης του κυκλώματος σε λογικά στοιχεία (CLBs, Cells κ.λ.π.) ανάλογα με την ακολουθούμενη ή προτεινόμενη τεχνολογία σχεδίασης (FPGA, ASIC κ.λ.π.). Από την διαδικασία αυτή προκύπτουν «γεωμετρικές» συντεταγμένες που αφορούν την τοποθέτηση όλων των στοιχείων του κυκλώματος όπως και η κατάλληλη καλωδίωση για την διασύνδεσή τους.

- **Τοποθέτηση και Δρομολόγηση (Place and Route):** Με την έννοια αυτή περιγράφουμε την χωροθέτηση και την διασύνδεση των στοιχείων της σχεδίασής μας στην φυσική τους τοπολογία, λαμβάνοντας υπόψη θέματα που άπτονται της συνολικής κατανάλωσης ισχύος του

κυκλώματος και της καθυστέρησης που παρουσιάζεται λόγω της διασύνδεσης των χρησιμοποιούμενων κελιών.

3.Αλγοριθμική Επίλυση του Προβλήματος

Στο παρόν κεφάλαιο θα γίνει εκτενής αναφορά στα «εργαλεία» που χρησιμοποιήθηκαν για την υλοποίηση του αλγορίθμου όπως και στην αναλυτικά διαδικασία που ακολουθεί ο αλγόριθμος για να ολοκληρωθεί το αποτέλεσμά του.

3.1 Εργαλεία

Για την υλοποίηση του αλγορίθμου χρησιμοποιήθηκε η γλώσσα προγραμματισμού C (και κάποιες επεκτάσεις της C++) σε UNIX περιβάλλον (Ubuntu Linux 8.04) ενώ το compilation του κώδικα έγινε με την χρήση του μεταγλωττιστή gcc και των κατάλληλων παραμέτρων του.

Οι βιβλιοθήκες που χρησιμοποιηθήκαν είναι οι ακόλουθες:

- **stdlib.h**
- **stdio.h**
- **string.h**
- **time.h**

Επιπρόσθετα, έγινε χρήση του εργαλείου SPICE. Το SPICE είναι ένα γενικού σκοπού πρόγραμμα εξομοίωσης κυκλώματος για μη γραμμικές (dc) και γραμμικές αναλύσεις (ac). Τα κυκλώματα μπορεί να περιλαμβάνουν αντιστάσεις, πυκνωτές, πηνία, αμοιβαίους επαγωγείς, ανεξάρτητες πηγές τάσης και ρεύματος, διακόπτες και τα πέντε πιο γνωστά ημιαγωγά στοιχεία (δίοδοι, BJTs, JFETs, MESFETs και MOSFETs).

3.2 Βασικα βηματα υλοποιησης του αλγοριθμου

Ο προτεινόμενος αλγόριθμος αποτελείται από οκτώ (8) διακριτά βήματα τα οποία ακολουθούνται κατά την σειρά που περιγράφεται ακολούθως:

1. Στο πρώτο βήμα καλούμαστε να διαβάσουμε στοιχεία που αφορούν τις χρησιμοποιούμενες πύλες από ένα αρχείο SPICE. Σε ένα αρχείο SPICE είναι αποθηκευμένες όλες οι πύλες που πρόκειται να χρησιμοποιηθούν στην συνέχεια από τον αλγόριθμο. Οι πύλες αυτές θα τοποθετηθούν κατά βέλτιστο τρόπο πάνω στο chip.
   Η τυπική σύνταξη μιας πύλης σε ένα αρχείο SPICE είναι η ακόλουθη:

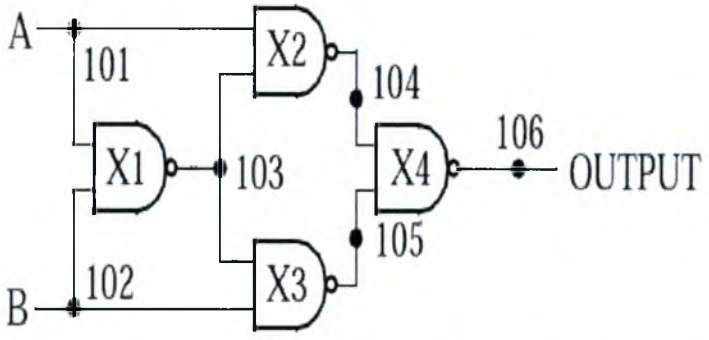
   Χ-ΑΡΙΘΜΟΣ ΕΙΣΟΔΟΙ ΕΞΟΔΟΣ ΤΥΠΟΣ ΠΥΛΗΣ

   Για παράδειγμα η ακόλουθη έκφραση:

   X1 100 101 200 NOR2

   δηλώνει ότι έχουμε μια πύλη η οποία αντιστοιχίζεται στον αριθμό 1, έχει 2 εισόδους (τις 100 και 101), μια έξοδο (την 200) και είναι τύπου NOR2 (είναι δηλαδή μια πύλη NOR 2 εισόδων).
   Τέλος, ένα ακόμη παράδειγμα της σύνταξης ενός αρχείου SPICE μιας μεγαλύτερης αυτή τη φορά σχεδίασης είναι το κύκλωμα που απεικονίζεται στο ακόλουθο σχήμα:

Standard cells placement algorithm for integrated circuits.



ΣΧΗΜΑ 3.1: Κυκλωματικό παράδειγμα της σύνταξης του SPICE

Ο κώδικας για την παραπάνω σχεδίαση είναι:

X1 101 102 103 NAND

X2 101 103 104 NAND

X3 103 102 105 NAND

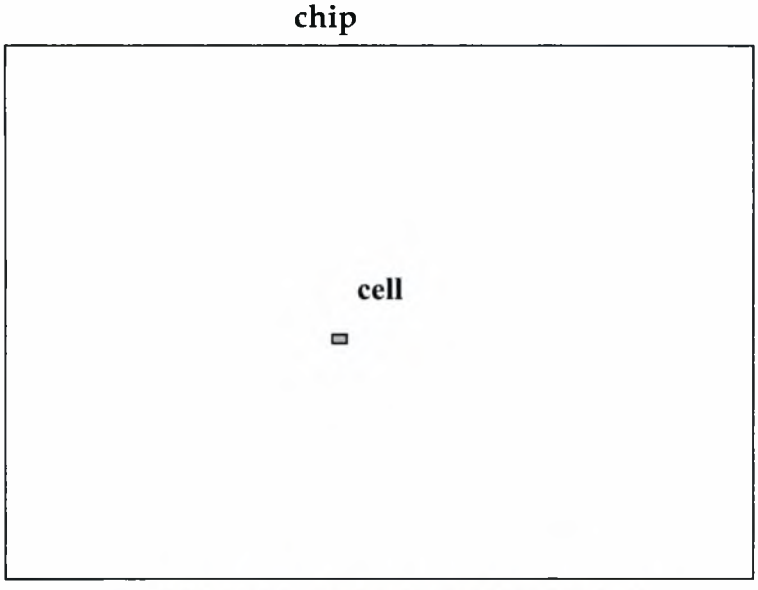
X4 104 105 106 NAND

2. Στο δεύτερο βήμα εξάγουμε τα γεωμετρικά χαρακτηριστικά που θα χρειαστούν στα επόμενα βήματα του αλγορίθμου. Το μέγεθος ενός cell μπορεί να ποικίλλει στην περίπτωσή μας όμως χρησιμοποιήσαμε ως default τιμές τα 40u για το πλάτος της κάθε πύλης και τα 25u για το ύψος της. Σε ορισμένα από τα αποτελέσματα που θα παρουσιαστούν παρακάτω χρησιμοποιήσαμε τις τιμές 8 και 5 αντίστοιχα προκειμένου να διευκολυνθεί η ανάγνωση και η επαλήθευση των αποτελεσμάτων.
   Όσον αφορά, τώρα, το chip όπου θα τοποθετηθούν τα προκείμενα cells θεωρήσαμε πως αναπαριστάται σχηματικά από ένα τετράγωνο του οποίου οι διαστάσεις δίνονται από τον ακόλουθο τύπο:

Μεγεθος_chip=πλατος_cell * υψος_cell * αριθμο_πυλων.

Τελικο_μεγεθος_chip=πλατος_cell * υψος_cell * αριθμο_πυλων + 20%Μεγεθος_chip

Στην παρακάτω εικόνα δίνεται σχηματικά η αναλογία των μεγεθών των κελιών που θα χρησιμοποιήσουμε ως προς το συνολικό chip.



ΣΧΗΜΑ 3.2: Αναλογία μεγεθών chip/cell

3. Στο επόμενο βήμα πραγματοποιείται η δυναμική αποθήκευση των εξαγόμενων στοιχείων του αρχείου εισόδου. Προκειμένου να επεξεργαστούμε δυναμικά τα χαρακτηριστικά της κάθε πύλης δημιουργήσαμε μια δυναμική λίστα όπου το σύνολο των κόμβων είναι ίσο με το συνολικό αριθμό των πυλών που διαβάσαμε από το αρχείο SPICE. Η αποθήκευση των στοιχείων αυτών γίνεται σε τρεις διαδοχικές φάσεις που περιγράφονται ακολούθως:
    - 1η φάση. Με χρήση της συνάρτησης int fgetc(FILE *stream) διαβάζουμε τον χαρακτήρα X από το αρχείο SPICE. Η συνάρτηση αυτή διαβάζει τον επόμενο χαρακτήρα από ένα αρχείο και τον αποθηκεύει σε μια μεταβλητή ανάθεσης, επιστρέφει 1 εάν ο χαρακτήρας έχει διαβαστεί σωστά και 0 εάν υπάρχει κάποιο σφάλμα.

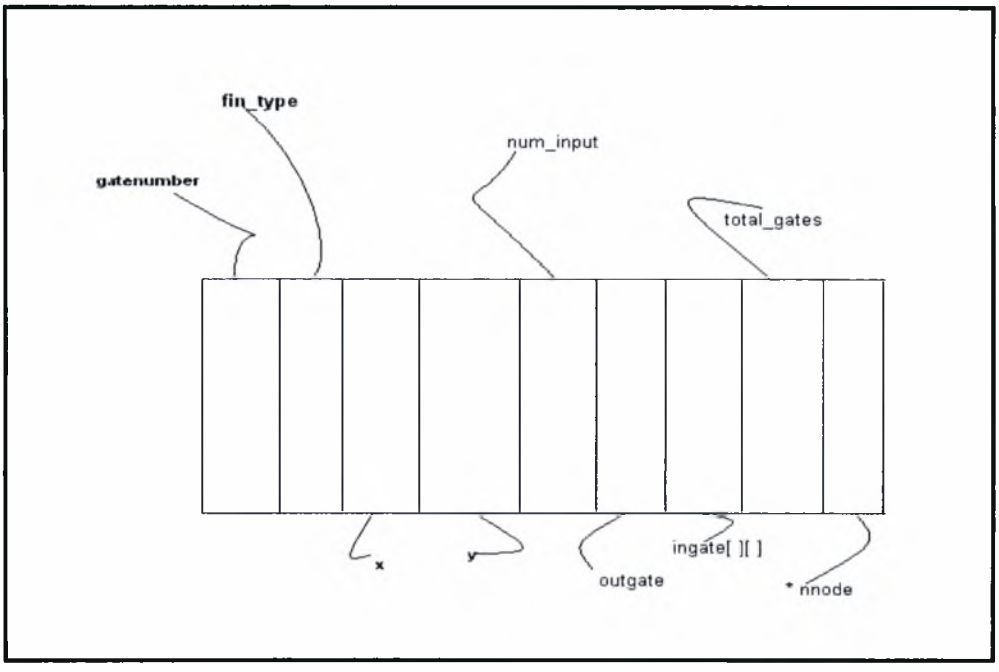
- 2η φάση. Χρησιμοποιώντας την συνάρτηση int fscanf(FILE *stream, const char * char *format,....) διαβάζουμε το στοιχείο. Στην παραπάνω συνάρτηση θα θεωρήσουμε όπου format το %d. Η συνάρτηση αυτή διαβάζει τον επόμενο ακέραιο αριθμό μέχρι να βρει ένα κενό και στην συνέχεια τον αποθηκεύει σε μια μεταβλητή ανάθεσης. Η τιμή που επιστρέφει είναι 1 εάν ο χαρακτήρας έχει διαβαστεί σωστά και 0 σε αντίθετη περίπτωση. Χρησιμοποιώντας μια while καλούμαι την fscanf όσες φορές χρειαστεί προκείνου να διαβάσουμε όλες τις εισόδους και την εκάστοτε έξοδο. Η παραπάνω διαδικασία θα τερματιστεί όταν η συνάρτηση μας επιστρέψει 0, γεγονός που σηματοδοτεί το διάβασμα ενός στοιχείου το οποίο δεν είναι αριθμός, δηλαδή το στοιχείο "N". Καθώς έχουμε διαβάσει τις εισόδους και τις εξόδους της κάθε πύλης είμαστε σε θέση να αποθηκεύσουμε τα αποτελέσματα σε προσωρινές μεταβλητές και χρησιμοποιώντας παράλληλα κάποιες αλγοριθμικές τεχνικές να διαχωρίσουμε τις εξόδους από τις εισόδους.

- 3η φάση. Κατά την ολοκλήρωση της προηγούμενης φάσης αναφέραμε πως διαβάζουμε τον χαρακτήρα "N". Επομένως σε κάθε μια επανάληψη διαβάζοντας τον επόμενο χαρακτήρα κάθε φορά μέχρι να βρούμε τον τερματικό

  χαρακτήρα θα έχουμε διαβάσει και τον τύπο της προκείμενης πύλης.

Οι παραπάνω φάσεις θα επαναληφθούν εώς ότου διαβάσουμε τον χαρακτήρα EOF. Έτσι, ανάλογα με τον συγκεκριμένο αριθμό επαναλήψεων μπορούμε να υπολογίσουμε πόσες πύλες θα περιέχει το

εκάστοτε αρχείο SPICE. Παράλληλα, σε κάθε επανάληψη θα δεσμεύουμε μνήμη για έναν καινούριο κόμβο ο οποίος θα περιέχει τα στοιχεία που παρουσιάζονται στο ακόλουθο σχήμα:



ΣΧΗΜΑ 3.3: Δέσμευση μνήμης

4. Στο επόμενο βήμα καλούμαστε να δημιουργήσουμε τον πίνακα που θα περιγράφει τις συνδέσεις μεταξύ των cells. Ο πίνακας αποτελεί ένα από τα πλέον βασικά στοιχεία του αλγορίθμου που παρουσιάζεται καθώς χρησιμοποιείται τόσο για τον υπολογισμό της απόστασης μεταξύ δυο κελιών όσο και για την εύρεση των συντεταγμένων για την τοποθέτηση των κελιών πα'νω στο chip. Επιπρόσθετα, χρησιμοποιείται ως κριτήριο για την μεταφορά των cells πάνω στο chip για την εύρεση της βέλτιστης τοπολογίας.

   Στην παρούσα μελέτη ο πίνακας συνδέσεων δεν δίνεται από κάποιον κατασκευαστή, επομένως πρέπει να δημιουργηθεί από την αρχή. Για την δημιουργία του, λοιπόν, χρειαζόμαστε αρχικά τις εισόδους κάθε πύλης σε μια εύκολα προσπελάσιμη μορφή και τέλος μια έξυπνη αλγοριθμική

προσέγγιση για την αναγνώριση της σύνδεσης μεταξύ δυο κελιών.
Η τελική απεικόνιση της υπάρχουσας σύνδεσης μεταξύ δυο κελιών θα είναι η εξής: ένας πίνακας του οποίου οι στήλες και οι γραμμές θα αντιστοιχούν στις πύλες που έχουν διαβαστεί μέσω SPICE και τα στοιχεία του θα αποτελούνται από 0 και 1. Στην θέση του πίνακα i,j όπου θα εμφανίζεται 0 θα συνεπάγεται ότι δεν υπάρχει σύνδεση μεταξύ των δυο κελιών ενώ στη θέση όπου θα εμφανίζεται 1 θα εξάγουμε το συμπέρασμα πως τα δυο αυτά κελιά συνδέονται. Τέλος, χρησιμοποιώντας μια βοηθητική μεταβλητή θα αποθηκεύσουμε τον συνολικό αριθμό συνδέσεων που θα υπολογίσουμε.

5. Στο βήμα αυτό καλούμαστε να αρχικοποιήσουμε τις συντεταγμένες του κάθε κελιού. Συνδυάζοντας τη δέσμευση μνήμης για κάθε χαρακτηριστικό μιας πύλης, την δημιουργία pointers για να δείχνουμε σε αυτό και την δημιουργία του πίνακα συνδέσεων θα οδηγηθούμε σε ένα νέο επίπεδο, την απεικόνιση των cells πάνω στο chip. Αφήνοντας, τώρα, πίσω το λογικό επίπεδο θα ασχοληθούμε με το φυσικό δείχνοντας τους περιορισμούς που ξεπερνά ο αλγόριθμος αλλά και τα αποτελέσματα από αυτό το βήμα μέχρι το τέλος.
   Για την αρχικοποίηση των συντεταγμένων ενός κελιού έχουμε δημιουργήσει την συνάρτηση init(….) η οποία δέχεται σαν είσοδο τον αριθμό των πυλών και τις συντεταγμένες που ορίζουν το κέντρο του chip. Το κέντρο υπολογίζεται ως εξής:

   συντεταγμενη x=συντεταγμενη y= Μεγεθος_chip/2

   Ακολούθως, οι επιμέρους συντεταγμένες του κάθε κελιού θα προκύψουν μέσω της τυχαίας τοποθέτησής τους γύρω από το κέντρο κατά μια ποσότητα ίση με **μεγεθος_chip/8**.

6. Στη συνέχεια καλούμαστε να υπολογίσουμε την απόσταση μεταξύ δυο κελιών με χρήση της κατάλληλης γεωμετρικής εξίσωσης. Έχοντας τοποθετήσει όλα τα κελιά γύρω από το νοητό κέντρο πρέπει να υπολογίσουμε την απόσταση μεταξύ τους. Για να επιτευχθεί αυτό έχουμε δημιουργήσει μια συνάρτηση distance_min(....). Η συνάρτηση αυτή δέχεται ως ορίσματα τον συνολικό αριθμό των πυλών, των πίνακα συνδέσεων, τον αριθμό των συνδέσεων καθώς και τις διαστάσεις του chip.
   Για να υπολογίσουμε, τώρα, την απόσταση μεταξύ δυο κελιών θα πρέπει αρχικά να ελέγξουμε κατά πόσο τα κελιά αυτά συνδέονται. Αυτό θα γίνει ψάχνοντας επαναληπτικά τον πίνακα συνδέσεων. Κάθε φορά που βρίσκουμε την τιμή 1 θα υπολογίζουμε την απόσταση που προκύπτει ανάμεσα στη στήλη και στη γραμμή όπου παρατηρήθηκε ο άσσος.
   Για παράδειγμα εάν έχουμε 1 στην θέση του πίνακα i.j τότε η απόσταση μεταξύ των κελιών θα είναι η ακόλουθη:

   (Yj-Yi) + (Xj-Xi)

   όπου Yi,Xi είναι οι συντεταγμένες της πύλης i. Κάθε υπολογισμός αυτής της μορφής αποθηκεύεται σε έναν πίνακα που έχουμε ορίσει.

7. Στο προκείμενο βήμα του αλγορίθμου που παρουσιάζεται καλούμαστε να υπολογίσουμε την ελάχιστη απόσταση μεταξύ δυο κελιών τα οποία έχουμε «ανακαλύψει» πως εννώνονται. Έχοντας αποθηκεύσει την τιμή της κάθε απόστασης σε έναν πίνακα μπορούμε να τον αναδιατάξουμε με βάση τη μικρότερη απόσταση χρησιμοποιώντας έναν απλό αλγόριθμο εύρεσης και ταξινόμησης ελαχίστου. Αυτό πραγματοποιείται θεωρώντας αρχικά ως ελάχιστη απόσταση την πρώτη που διαβάζουμε από τον πίνακα και συγκρίνοντάς την με τις υπόλοιπες διατάξουμε αναδρομικά τον πίνακα. Η διαδικασία αυτή τερματίζεται όταν έχουν ελεγχθεί όλες οι πύλες μεταξύ τους.

8. Πριν να εκκινήσουμε το επόμενο βήμα του αλγορίθμου παρατηρούμε πως έχουμε στην διάθεσή μας τα ακόλουθα δεδομένα: έναν πίνακα ταξινομημένο ως προς την ελάχιστη απόσταση καθώς και την ελάχιστη απόστασημεταξύ δυο κελιών από το σύνολο των αποστάσεων που έχουν προ□πολογιστεί στο έκτο βήμα του αλγορίθμου.

   Στο σημείο αυτό αντιμετωπίζουμε το εξής πρόβλημα, γνωρίζουμε την βέλτιστη απόσταση μεταξύ των κελιών, δεν γνωρίζουμε, όμως, ποια είναι τα κελιά που συνδέονται ώστε να προκύψει η κατάλληλη απόσταση. Η λύση του προκείμενου προβλήματος είναι αλγοριθμικά εύκολη. Αρκεί να δημιουργήσουμε δυο πίνακες τους cord_distx και cord_disty με μέγεθος ίσο με τον αριθμό των συνδέσεων. Ακολούθως διατρέχουμε κατ'επανάληψη τον πίνακα διαστάσεων και κάθε φορά που εμφανίζεται 1 υπολογίζουμε την απόσταση μεταξύ των κελιών. Αν η απόσταση που βρήκαμε είναι η πρώτη ελάχιστη τότε αντιλαμβανόμαστε ότι τα κελιά που συνδέονταιμεταξύ τους προκύπτουν από τον αριθμό της γραμμής και της στήλης. Η παραπάνω διαδικασία συνεχίζεται μέχρι να βρούμε την εκάστοτε i-οστή ταξινομημένη απόσταση βρίσκοντας ταυτόχρονα και τα κελιά που συνδέονται.

   Έχοντας τώρα αποθηκεύσει έναν πίνακα με ταξινομημένες τις ελάχιστες αποστάσεις και τα κελιά που εννώνονται για να προκύψει η κάθε μια μπορούμε να περάσουμε στο στάδιο της μετακίνησης κελιών.

9. Στο τρέχων βήμα του αλγορίθμου καλούμαστε να μετακινήσουμε τα κελία αλλάζοντας θέση στις συντεταγμένες τους σε σχέση με την αρχική τους τοποθέτηση

   Δημιουργούμε τη συνάρτηση partition(...) με τα εξής ορίσματα : συνολικό αριθμό πυλών, πίνακα συνδέσεων, αρίθμος των συνδέσεων, διαστάσεις του chip, μικροτερη αποσταση , και σαν τελευταίο όρισμα τους πίνακες cord_distx , int cord_disty.

   Άρχικα θα πρέπει να εισάγουμε την έννοια του σταθερού cell και του μετακινούμενου . Ένα κελί θεωρείτε σταθερό όταν η σύνδεση του με ένα άλλο συνεπάγεται τη μικρότερη που υπάρχει στο chip. Ένα κελί θεωρείτε μετακινούμενο όταν από συνολικά n συνδέσεις η απόσταση της σύνδεσης του με κάποιο άλλο ξεπερνά τη θέση n/2 των ταξινομημένων αποστάσεων.

   Στη παρούσα μελέτη η συνάρτηση partition(...) αρχίζει με την εύρεση των κελίων , η απόσταση των οποίων είναι η μικρότερη. Έστω ότι έχουν εξαάχθει τα κελία i , j . Στο επόμενο βήμα αλλάζει η τιμή στη θέση i , j του πίνακα συνδέσεων από 1 σε 0 . Πλεον η συγκεκριμένη σύνδεση δεν θα ξαναλειθφεί υπ'οψην. Τα cells i, j είναι σταθερά.

   Στη συνέχεια θα προβολυμε στη μετακινήσή κάποιων κελίων μέσα στο chip. Τα κελία θα μετακινηθούν κατά το εμβαδόν ενός κελλού. Η λογική είναι η εξης : Θα μετακινήσουμε εκείνα τα κελία που υποδεικνύουν οι πίνακες cord_distx , cord_disty απο τη μέση των πινάκων και μετα. Οι πίνακες cord_distx , cord_disty αντιπροσωπεύουν τον αριθμό των κελίων γνωρίζοντας την απόσταση μεταξύ 2 κελίων. Είναι κατανοητό ότι είναι προτιμότερο να μετακινήσουμε εκείνα τα κελία που οι απόσταση μεταξύ

τους είναι μεγάλη ώστε να αυξήσουμε τη δυνατότητα ελαχιστοποίησης της. Παράλληλα καθώς τα κελία αρχικοποιούνται στο κέντρο θα πρέπει να απομακρύνονται από αυτό ώστε να δώσουμε στο chip την κατάλληλη "ελευθερία" . Η τοποθέτηση των κέλιων όσο πιο κοντά γίνεται το ένα στο άλλο ναι μεν συνεπάγεται μικρότερη απόσταση μεταξύ τους αλλά η τελική υλοποίηση δε σε επίπεδο ολοκληρωμένου κυκλώματος είναι αδύνατη γίατι θα υπάρξουν συγκρούσεις τόσο σε επίπεδο επικάλυψης των κελίων όσο και σε επίπεδο καλωδίωσης.

Στη συνέχεια θα καλέσουμε ξανά τη συνάρτηση distance_min με διαφορετκή παράμετρο τον αριθμό των συνδέσεων μειωμένο κατά 1. Η distance_min με τη σείρα της θα καλέσει τη partition με αρίθμο συνδέσεων μικρότερο κατά 1. Τελικά η παραπάνω αλγοριθμική διαδικασία θα ολοκληρωθεί μόνο όταν ο πίνακας συνδέσεων έχει μόνο ένα 1 , γεγονός που μας ενημερώνει ότι υπάρχει μόνο μια σύνδεση που δεν εχει γίνει σταθερή, η οποία στο τρέχων βήμα θα γίνει.

Η συνάρτηση οκ() καλείται όταν εχει ολοκληρωθεί η τοποθέτησει των cells στο chip ώστε να εγγυηθεί ότι οι διαστάσεις του ενός δεν θα επιρεάζουν τη τοποθέτηση κάποιου άλλου. Αν εντοπιστεί μια σύγκρουση αυτής της μορφής τότε θα μετακινήσουμε το cell κατά το εμβαδό του σε μια μη προκαθορισμένη θέση.

Η συνάρτηση go_to δέχεται σαν όρισμα ένα ακέραιο αριθμό . Έπειτα ο τρέχων δείκτης πρόχωραει προς τα "δεξια" σύμφωνα τη τιμη του αριθμου μείων 1 .

Standard cells placement algorithm for integrated circuits.

4 Χρήση C++

4.1 *Βασικές Βιβλιοθήκες και συναρτήσεις.*

Η **C++** είναι μια γενικού σκοπού και υψηλού επιπέδου γλώσσα προγραμματισμου, η οποία ανήκει στην κατηγορία των αντικειμενοστρεφων γλωσσών. Σε αυτό το σημείο της μελέτης θα παρουσιάσουμε τη γραφική απεικόνιση και επίλυση του προβλήματος.

Οι βιβλιοθήκες που χρησιμοποιηθήκαν είναι οι ακόλουθες:

- #include <GL/gl.h>
- #include <GL/glu.h>
- #include <GL/glut.h>
- #include <iostream>
- #include <fstream>
- #include <stdlib.h>
- #include <stdio.h>
- //#include <windows.h>
- #include <time.h>
- #include <stdio.h>
- #include <pthread.h>

Οι συναρτήσεις που χρησιμοποιήθηκαν είναι οι ακόλουθες:

- glutInit (&argc, argv)

Είναι δυνατό να περαστούν οι επιλογές γραμμών εντολής στη βιβλιοθήκη Glut.

Standard cells placement algorithm for integrated circuits.

- glutInitDisplayMode (GLUT_RGBA)

Τα χρώματα επίδειξής μας θα διευκρινιστούν όπως κόκκινα, πράσινα και μπλε συστατικά μεγέθη (GLUT_RGBA).

- glutInitWindowSize (chipcord, chipcord)

Καθορίστε το αρχικά ύψος και το πλάτος οποιουδήποτε παραθύρου που δημιουργείται.

- glutInitWindowPosition (0, 0)

Αρχικά το παράθυρο θα τοποθετηθεί στον κορυφαίο, αφημένος τη γωνία της οθόνης.

- glutCreateWindow («Basic_OpenGL_GLUT»)

Τέλος, δημιουργήστε ένα παράθυρο με το τιτλο Basic_OpenGL_GLUT.

4.2 Γραφική απεικόνιση του αλγορίθμου

Μοντελοποιούμε το κάθε κελί ως ένα κύβο με χρώμα από μαύρο εως κόκκινο. Ένα ορθογώνιο θα δημιουργηθεί γύρω από κάθε κελί δείχνοντας τις διαστάσεις του κελιού. Η σύνδεση μεταξύ 2 κελιών μοντελοποιείται ως μια ευθεία γραμμή ανάμεσα σε 2 κύβους (κελιά) . Το chip απεικονίζεται ως ενα μεγάλο τετράγωνο.

4.3 Γραφική απεικόνιση του αλγορίθμου σε standard cell τεχνολογία

Όμοια απεικόνιση με τη παραπάνω. Ο χώρος (μεγάλο τετράγωνο) διαιρείται σε τόσα μέρη όσα ζητήσει ο κατασκευαστής.

5. *Πειραματικές Μετρήσεις και Αποτελέσματα*

5.1 Συγκριτικά Αποτελέσματα και Συμπεράσματα.

Στο κεφάλαιο αυτό θα παρουσιάσουμε μια σειρά αποτελεσμάτων τα οποία προέκυψαν μέσω των διεργασιών που αναφέρθηκαν σε προηγούμενες ενότητες της παρούσας μελέτης. Τα προκείμενα αποτελέσματα αναφέρονται κατά κύριο λόγο στον αριθμό των πυλών.
Τα «μετρικά» στοιχεία που χρησιμοποιούνται για τις συγκρίσεις είναι :
ο χρόνος υπολογισμού του αλγορίθμου placement , ο αριθμός των πυλών, ο αριθμός των συνδέσεων κ.τ.λ.

Για τον υπολογισμό του χρόνου ολοκλήρωσης του αλγορίθμου θα χρησιμοποιήσουμε τη βιβλιοθήκη time.h
Στην αρχή της συνάρτησης main () δημιουργούμε 2 μεταβλητές start και stop τύπου time_t και μια double με όνομα diff . Στο επόμενο βήμα καλούμε τη συνάρτηση time (&start) και αποθηκεύουμε στη μεταβλητή start τον τρέχων χρόνο. Στο τέλος της main καλούμε τη time (&stop) και βρίσκουμε το συνολικό χρόνο που ο αλγόριθμος παραμένει ενεργός μέσω της συνάρτησης difftime(stop,start) .

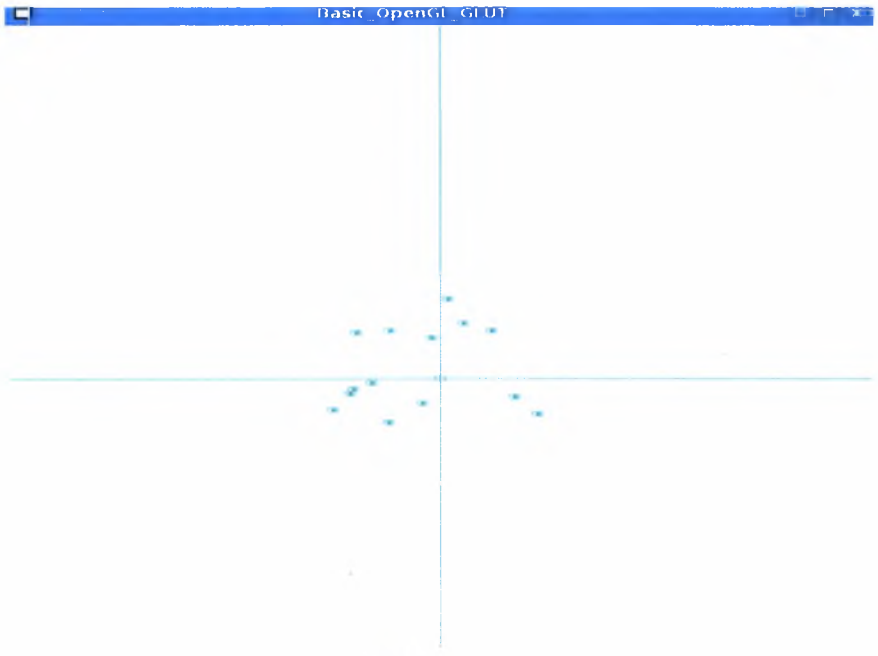
Τα αποτελέσματα καθώς και οι σημαντικές παράμετροι υλοποίησης του αλγορίθμου θα παρουσιαστούν με τη χρήση γραφικών επιπέδου C++ καθώς και πινάκων απεικόνισης μέσω του Microsoft Office.
Στο παράρτημα Α θα απεικονίσουμε τα στοιχεία μας με τη χρήση C++ και της βιβλιοθήκης Opengl - Glut ενώ στο παράρτημα Β με τη χρήση του προγράμματος Office .

Παράρτημα Α

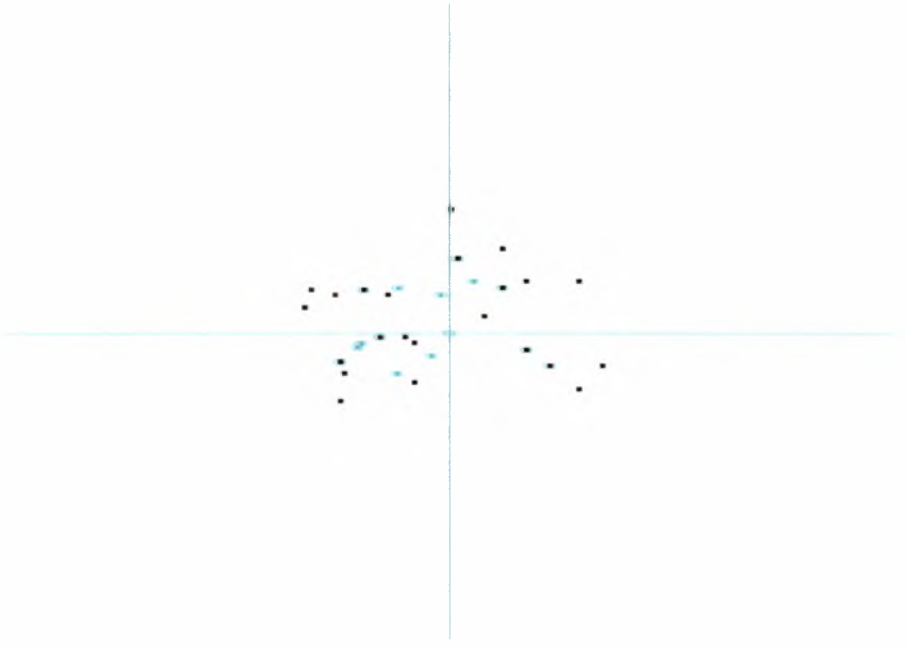
Α1 : Αριθμός πυλών 14.

- Ο χρόνος που χρειάζεται για να "τρέξει" ο αλγόριθμος είναι 672 μsec .
- Οι διαστάσεις του chip είναι 672*672.
- 308 υπολογισμοί για τις συντεταγμένες των σταθερών και των μετακινούμενων κελιών.
- Ο συνολικός αριθμός των κελιών που συνδέονται είναι 22.
- Τα κελιά θα τοποθετηθούν ως εξής : 13-12-5-7-8-11-2-4-1-6-3-9-10-14

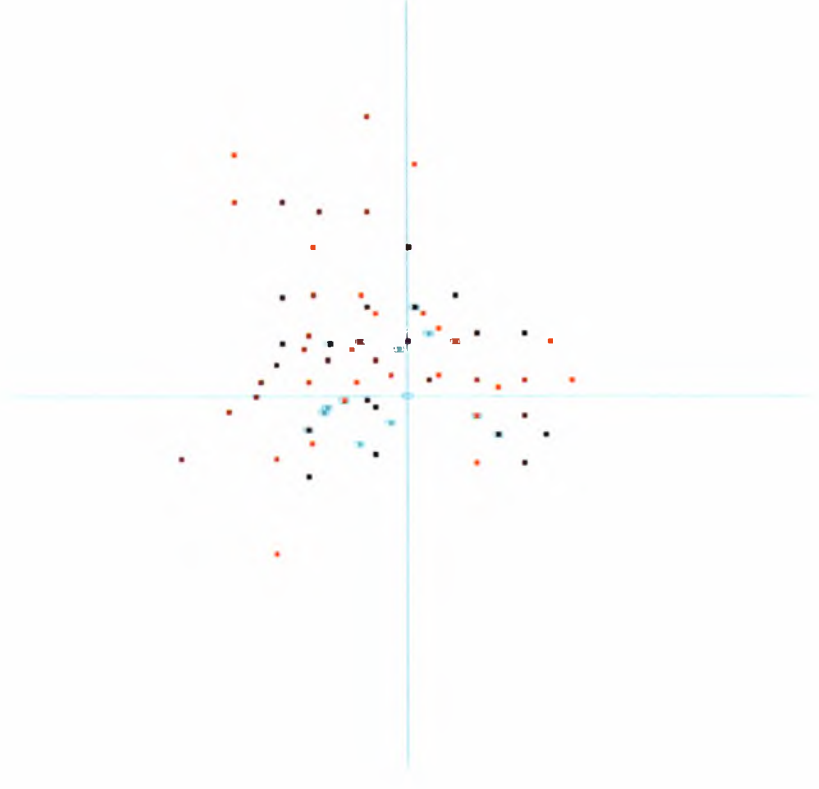
Αρχικοποίηση κελιών.



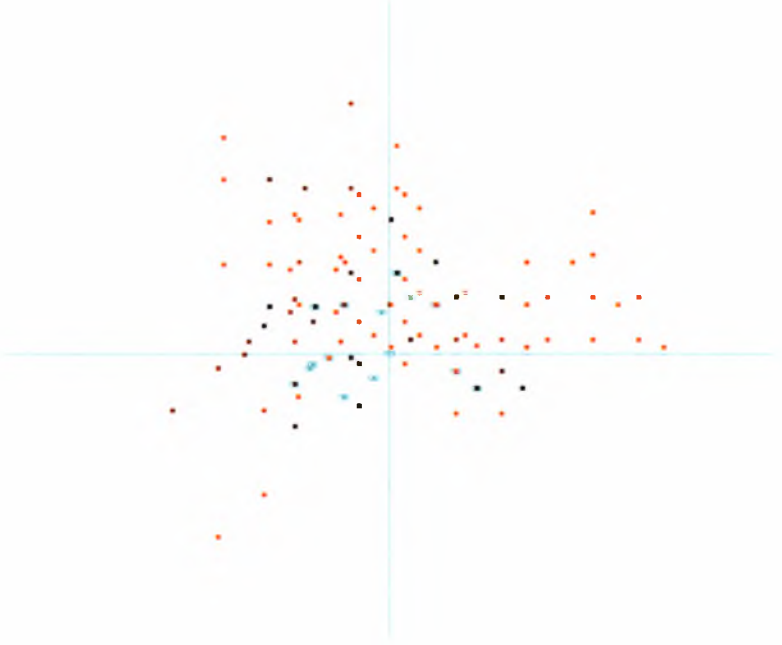
Μετακίνηση των κελιών . 1



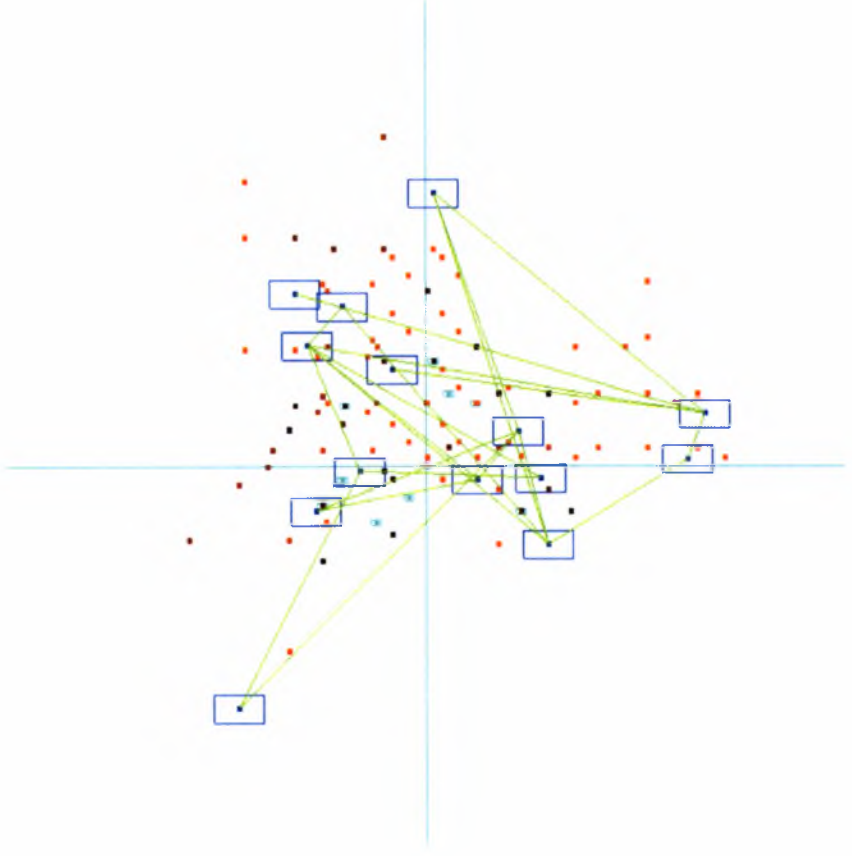
Μετακίνηση των κελιών . 2



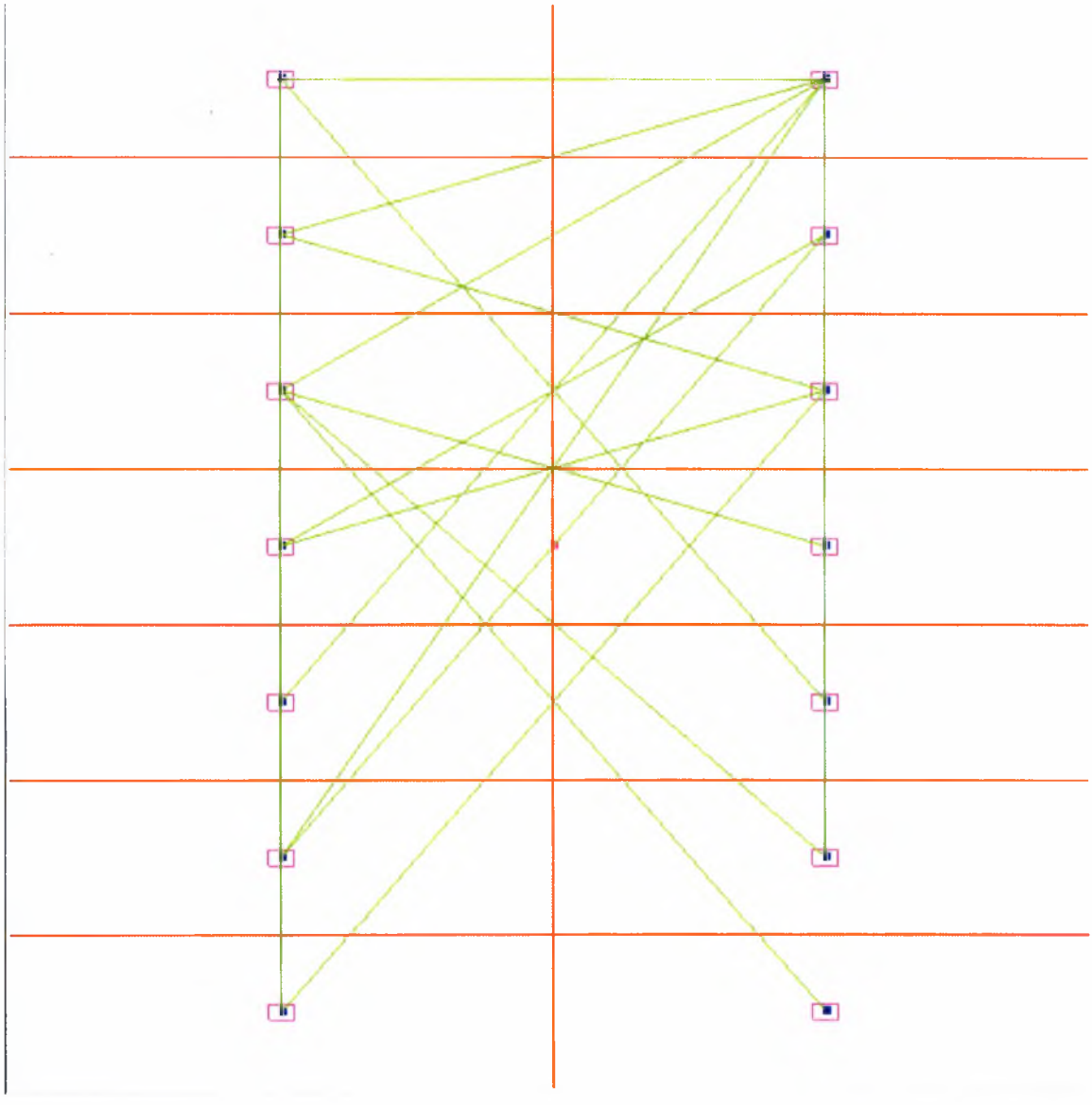
Μετακίνηση των κελιών .3



Τελική τοποθέτηση κελιών(χρώμα μπλε).



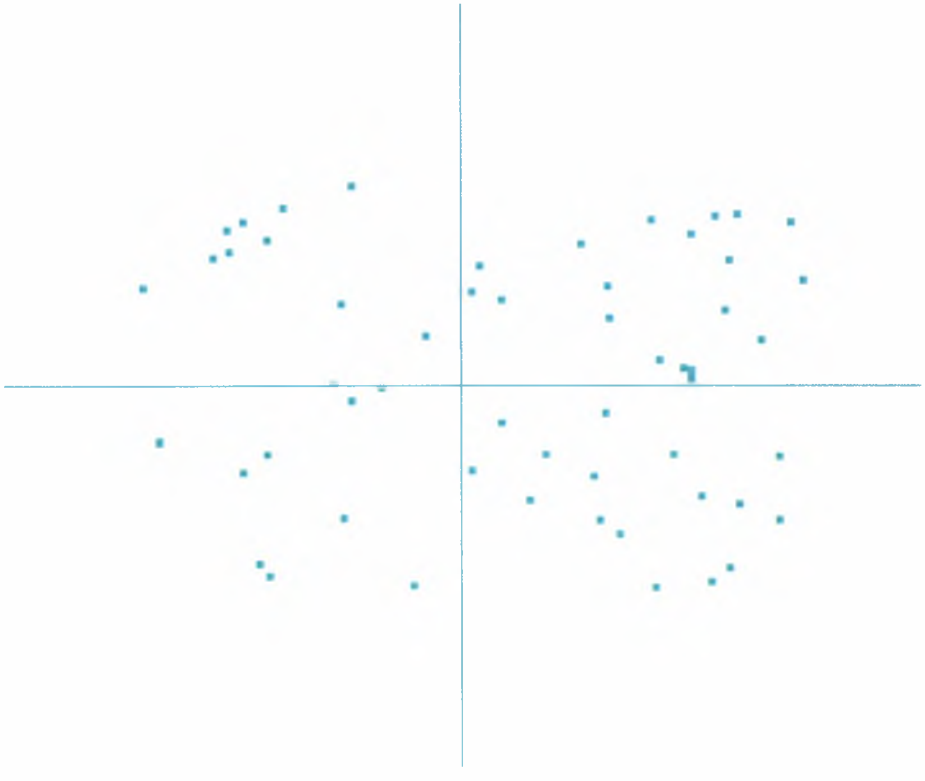
Standard Cell διάταξη.



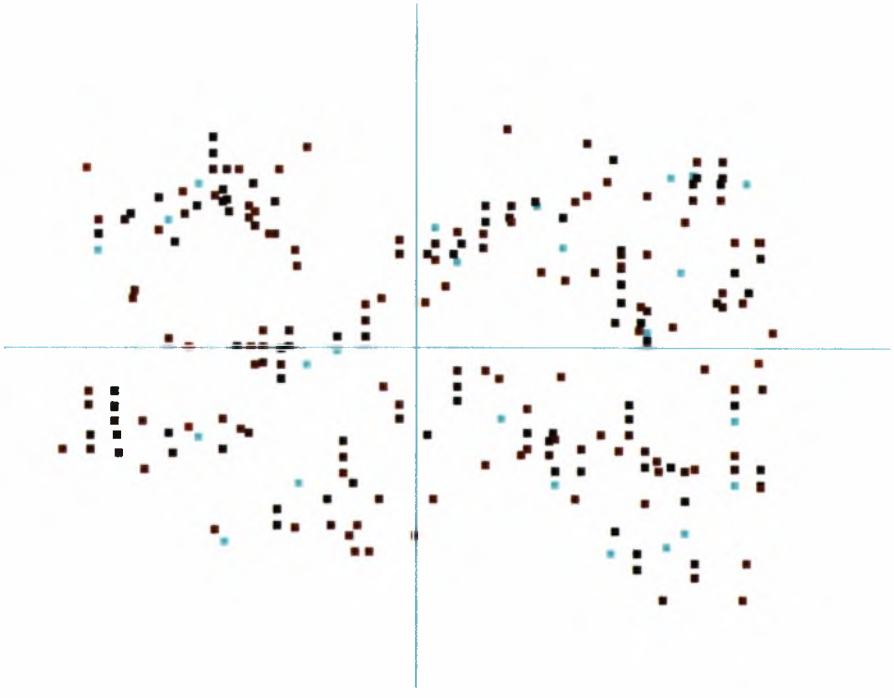
A2 : Αριθμός πυλών 90.

- Ο χρόνος που χρειάζεται για να "τρέξει" ο αλγόριθμος είναι 4320 μsec .
- Οι διαστάσεις του chip είναι 4320*4320.
- 8190 υπολογισμοί για τις συντεταγμένες των σταθερών και των μετακινούμενων κελιών.
- Ο συνολικός αριθμός των κελιών που συνδέονται είναι 91.
- Τα κελιά θα τοποθετηθούν ως εξής : 81-82-83-84-85-86-87-88-89-57-51-52 κτλ.

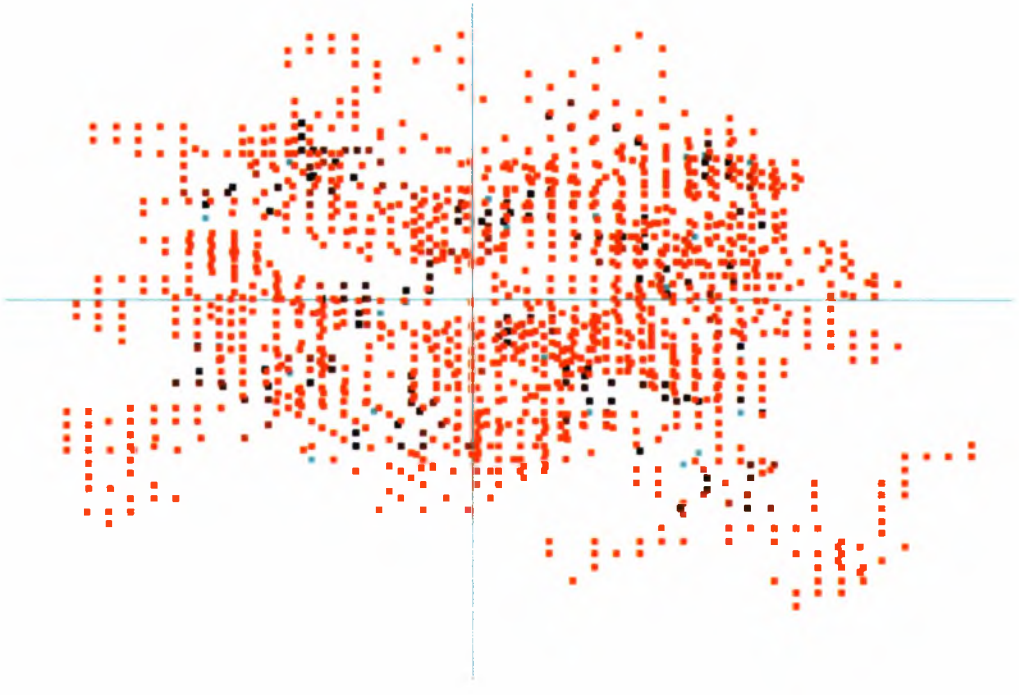
Αρχικοποίηση κελιών.



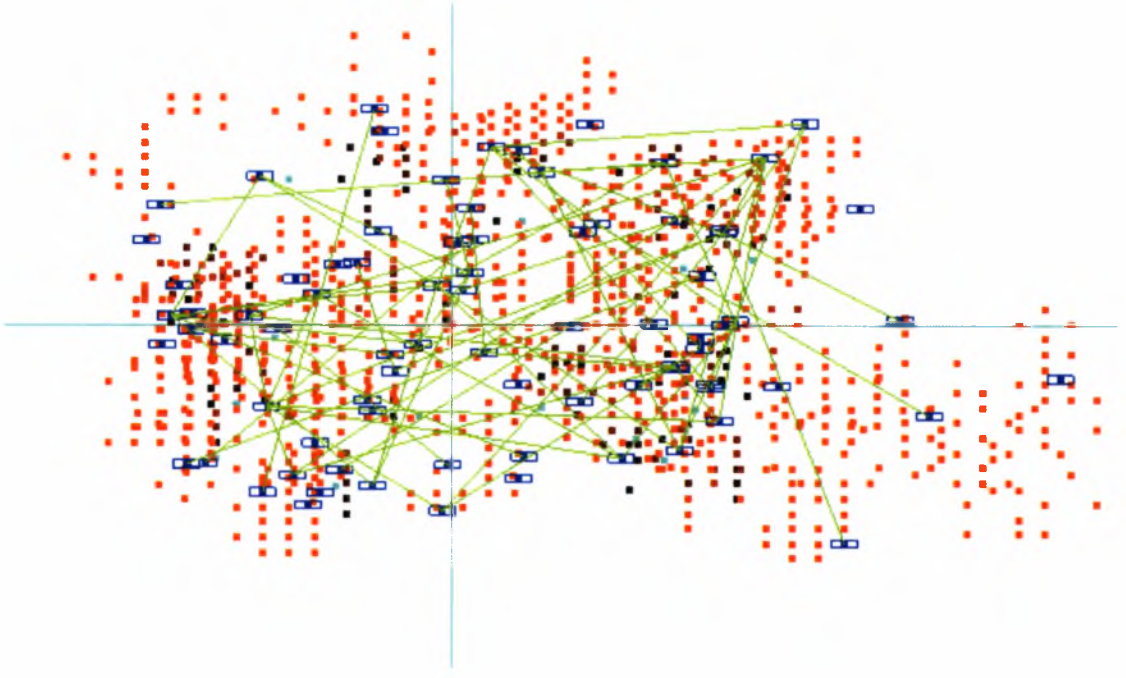
Μετακίνηση των κελιών . 1



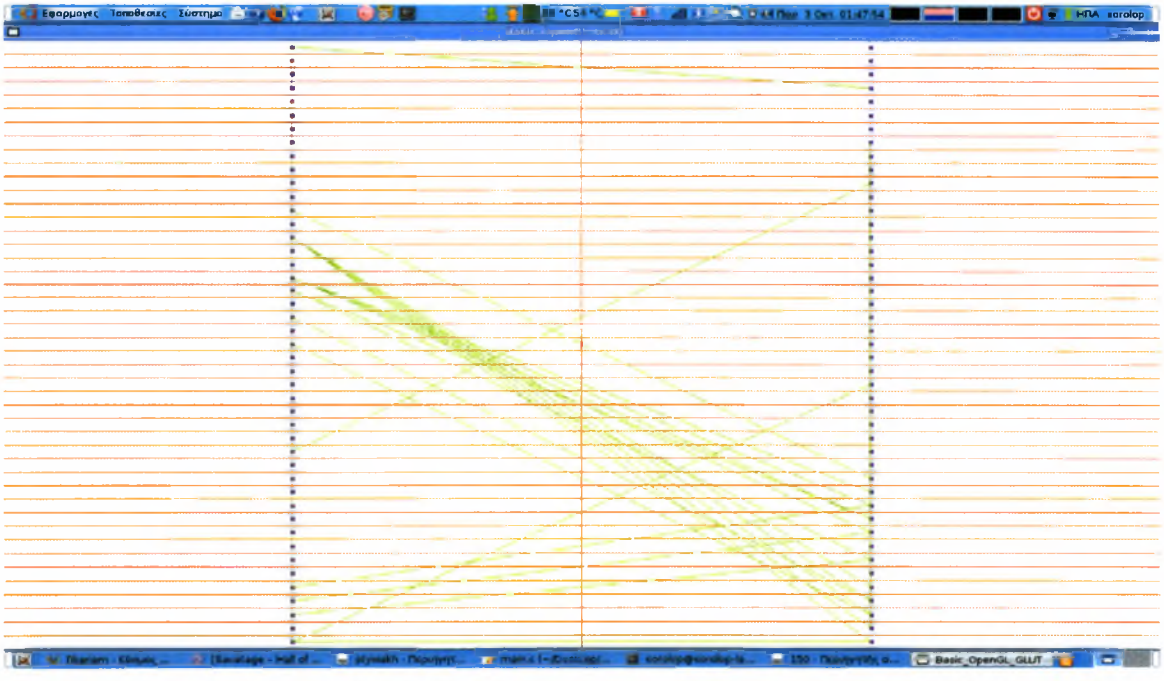
Μετακίνηση των κελιών .2



Τελική τοποθέτηση κελιών(χρώμα μπλε).



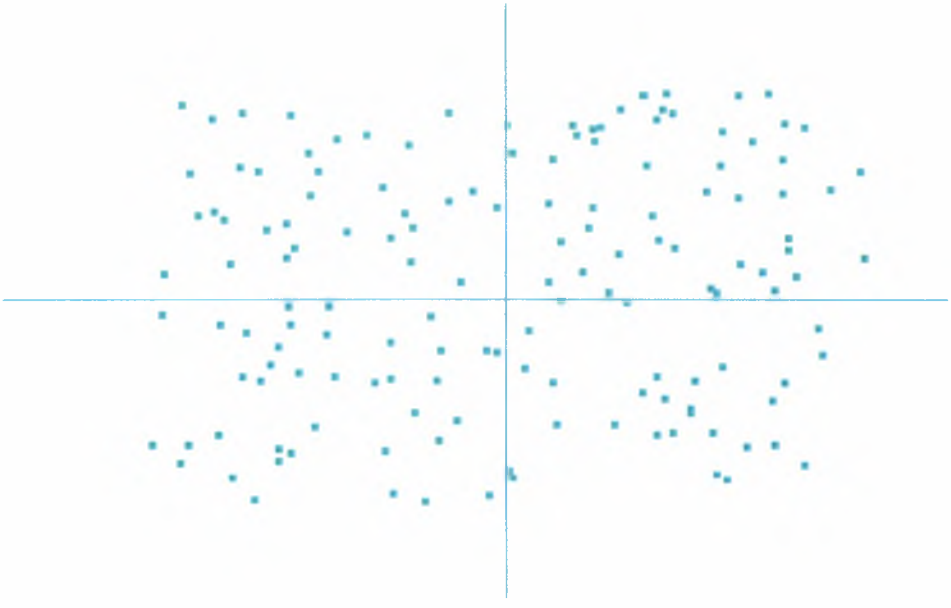
Standard Cell διάταξη.



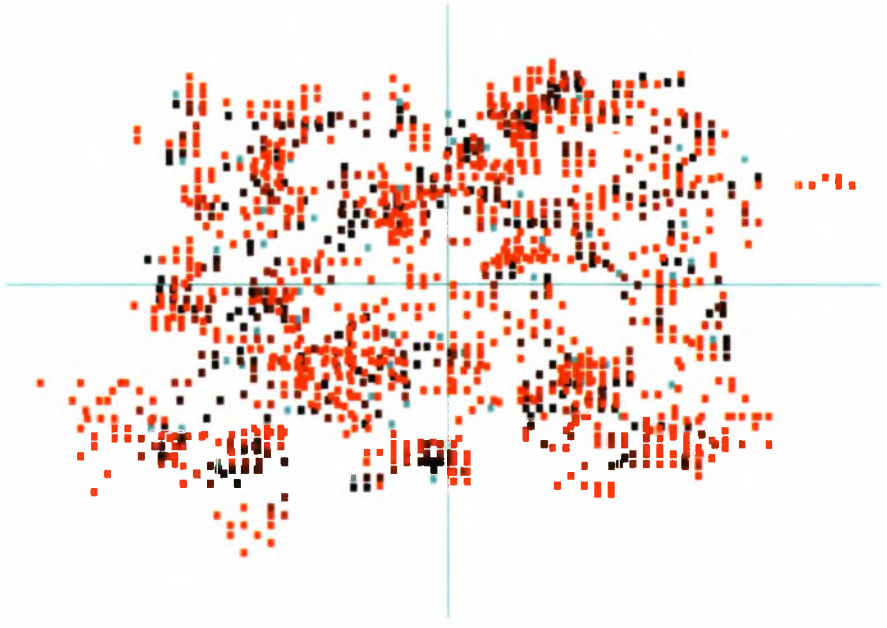
Α3 : Αριθμός πυλών 150.

- Ο χρόνος που χρειάζεται για να "τρέξει" ο αλγόριθμος είναι 7200 μsec .
- Οι διαστάσεις του chip είναι 7200*7200.
- 33.000 υπολογισμοί για τις συντεταγμένες των σταθερών και των μετακινούμενων κελιών.
- Ο συνολικός αριθμός των κελιών που συνδέονται είναι 220.
- Τα κελιά θα τοποθετηθούν ως εξής : 135-136-137-138-139-140-...-11-63-24 κτλ

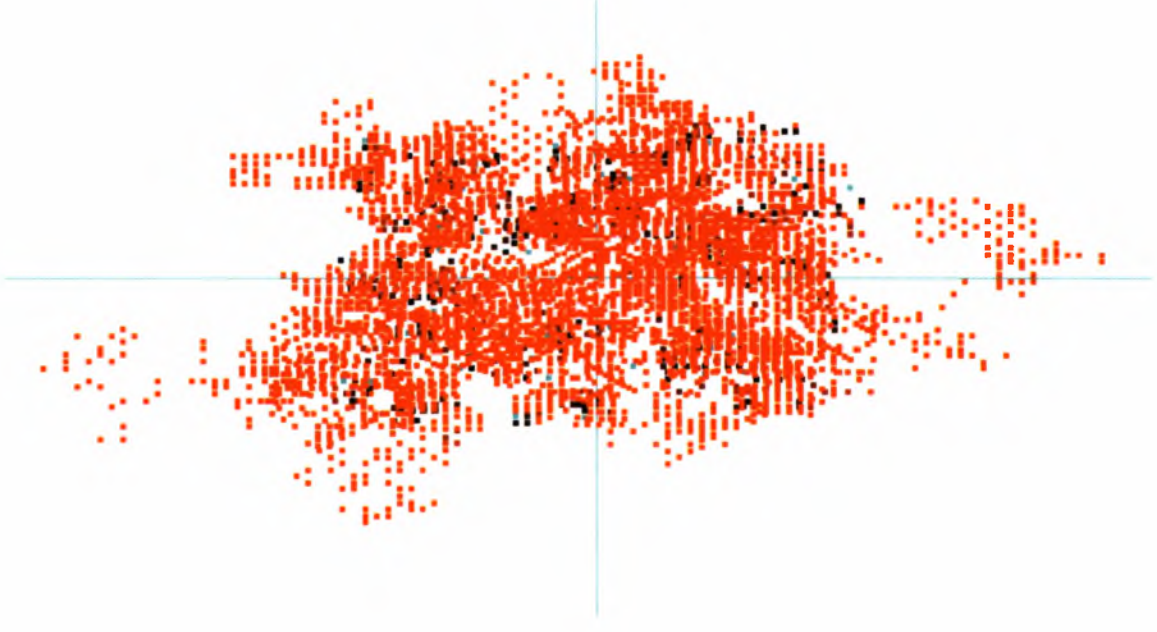
Αρχικοποίηση κελιών.



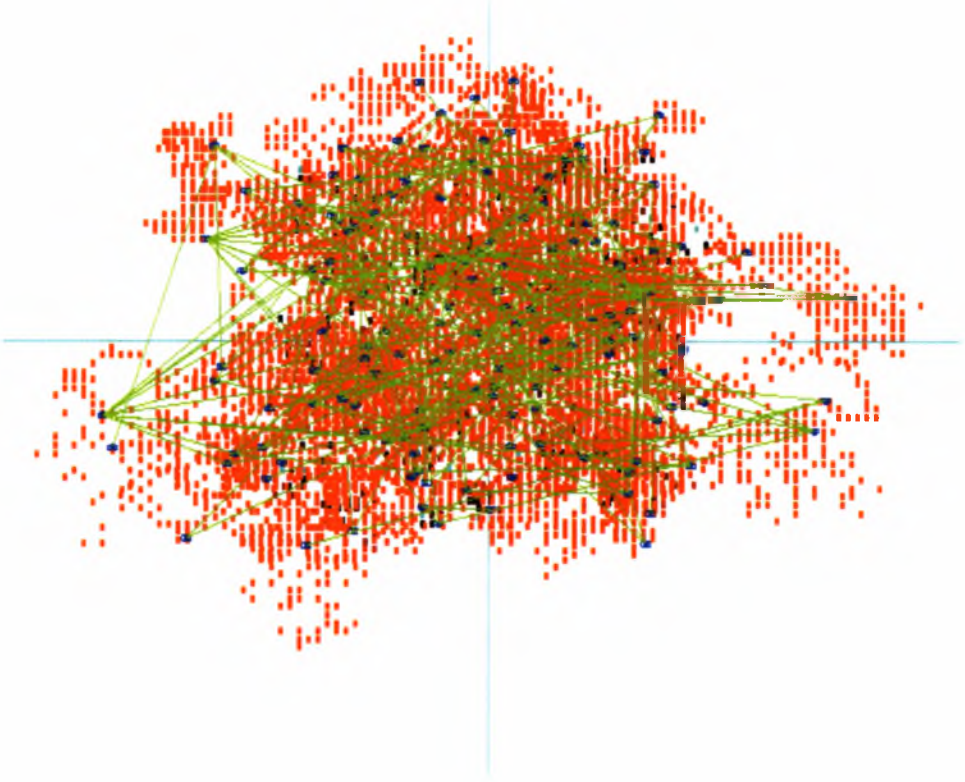
Μετακίνηση των κελιών . 1



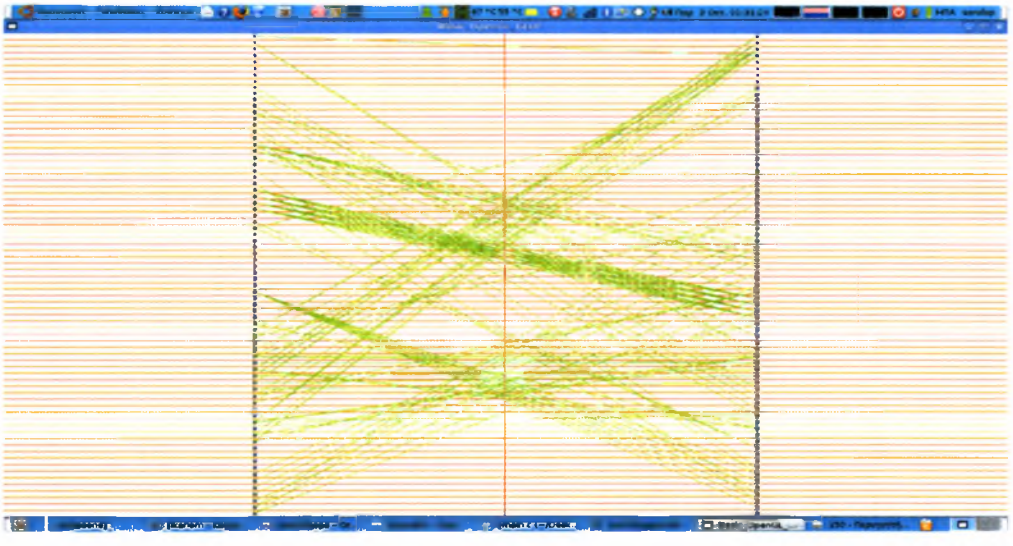
Μετακίνηση των κελιών . 2



Τελική τοποθέτηση κελιών(χρώμα μπλε).



Standard Cell διάταξη.

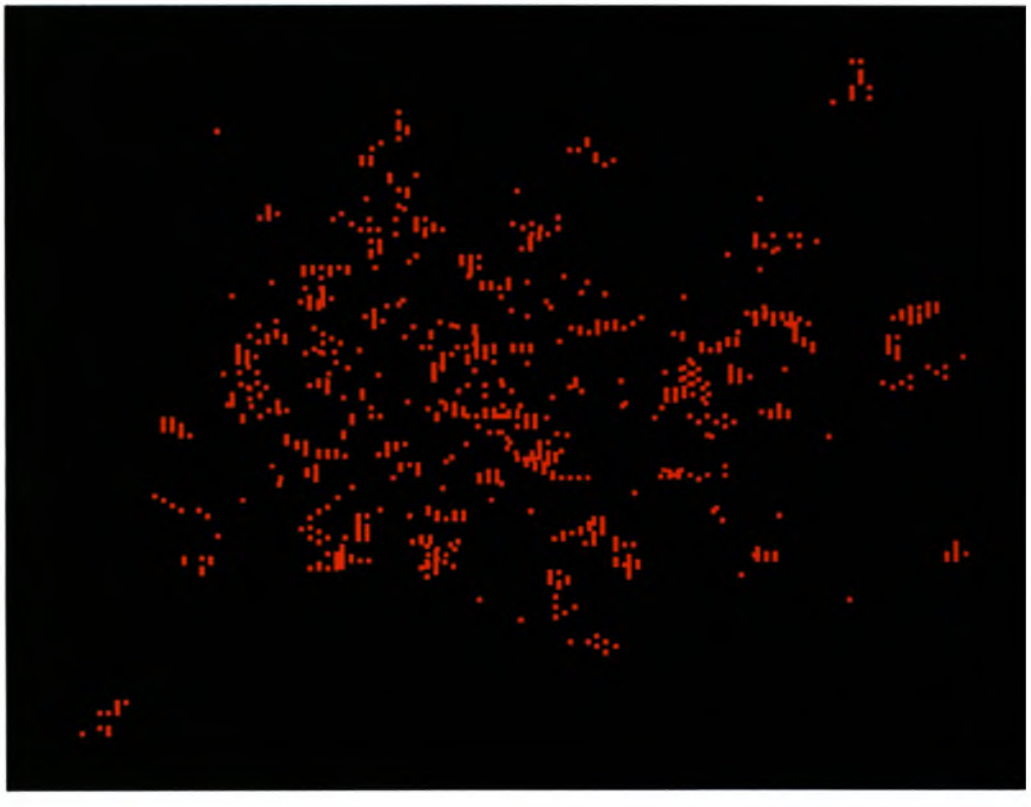


Standard cells placement algorithm for integrated circuits.

A4 Αριθμός πυλών 198

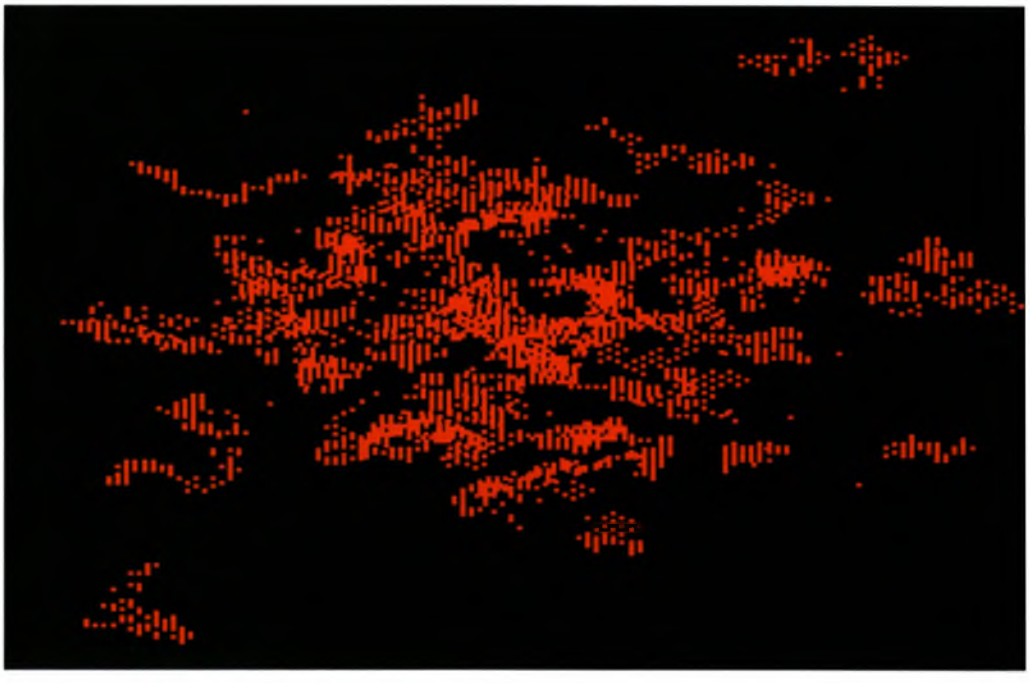
- Ο χρόνος που χρειάζεται για να "τρέξει" ο αλγόριθμος είναι 9504 μsec .
- Οι διαστάσεις του chip είναι 9504 *9504.
- 50.000 υπολογισμοί για τις συντεταγμένες των σταθερών και των μετακινούμενων κελιών.
- Ο συνολικός αριθμός των κελιών που συνδέονται είναι 350.
- Τα κελιά θα τοποθετηθούν ως εξής : 91-92-93-94-95-96-97-98-99-77-31-32 κτλ.

Αρχικοποιηση κελιων.

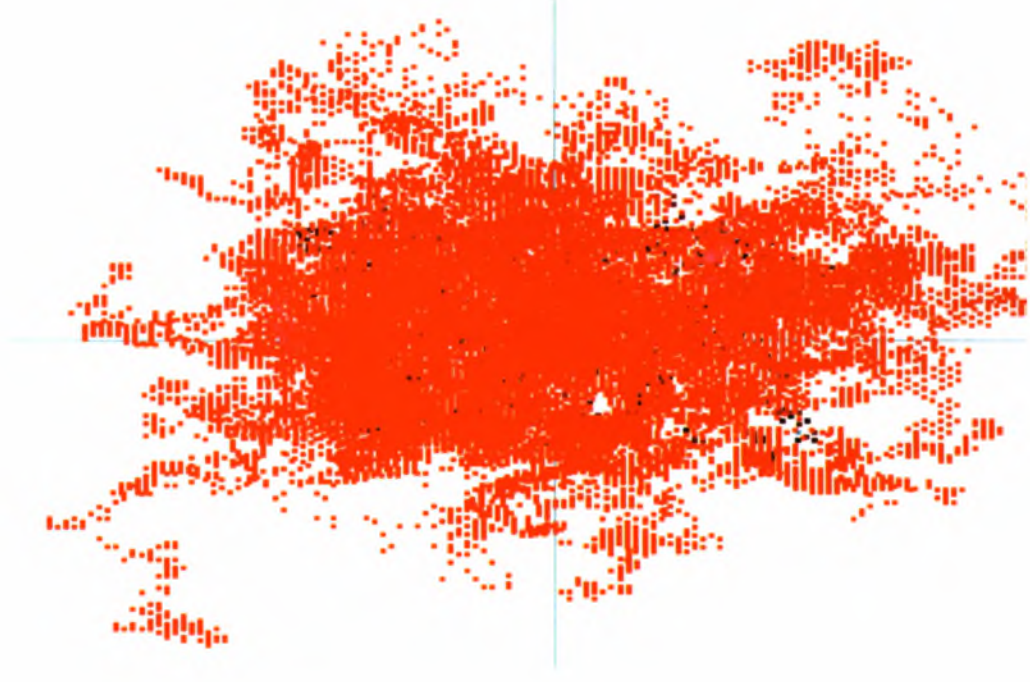


Standard cells placement algorithm for integrated circuits.

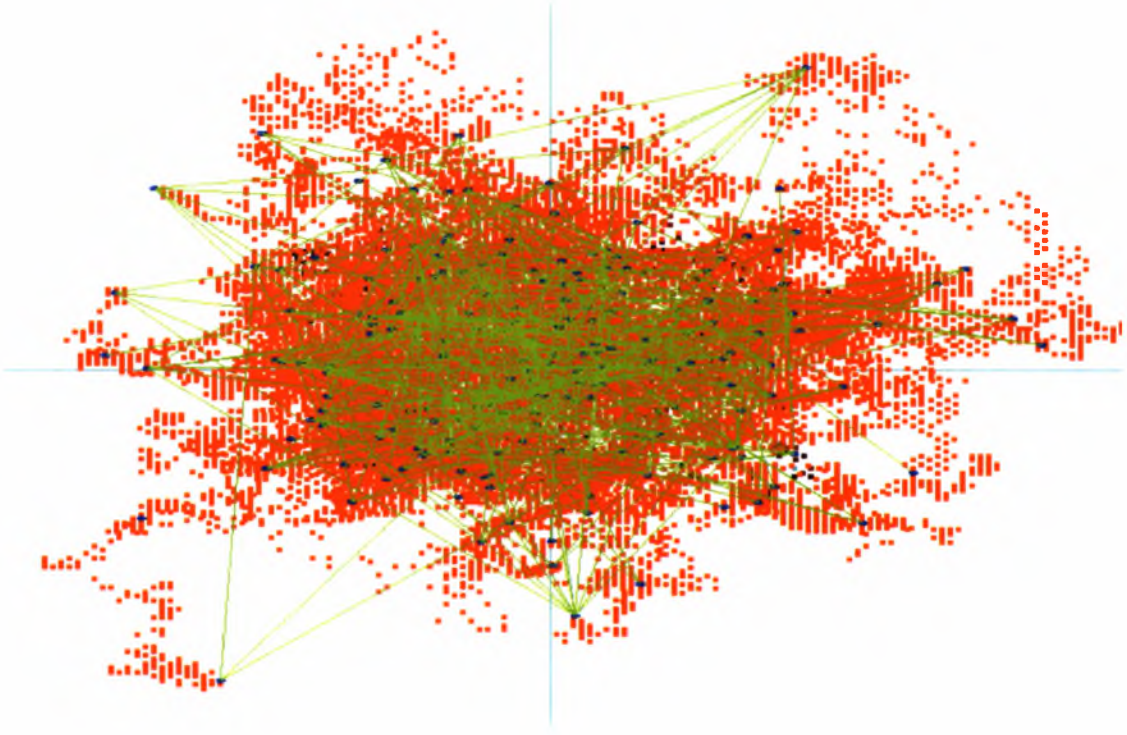
Μετακίνηση των κελιών . 1



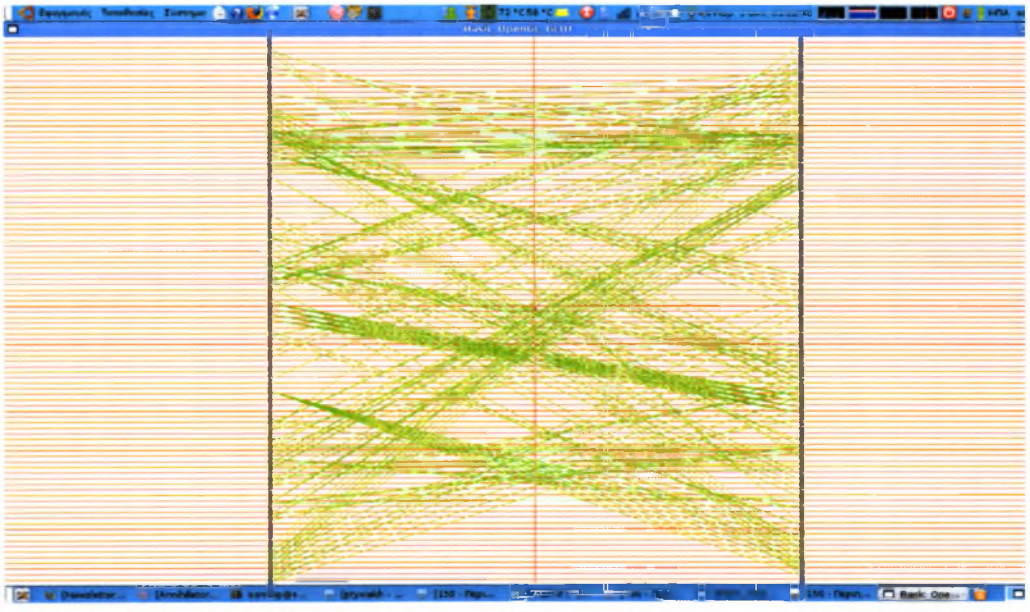
Μετακίνηση των κελιών .2



Τελική τοποθέτηση κελιών (χρώμα μπλε).



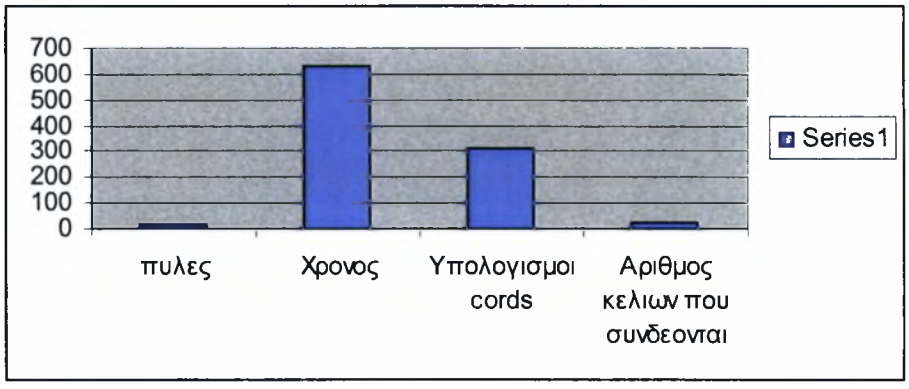
Standard Cell διάταξη.



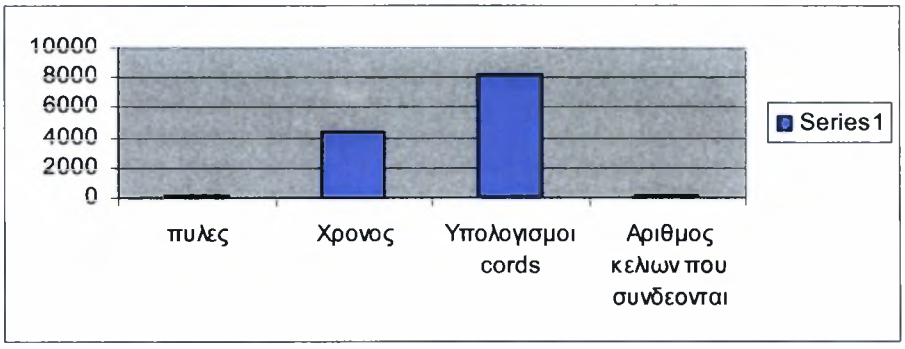
Παράρτημα Β

Θα εισάγουμε τις εισόδους και τις εξόδους του αλγορίθμου σε στατιστικά αρχεία excel.

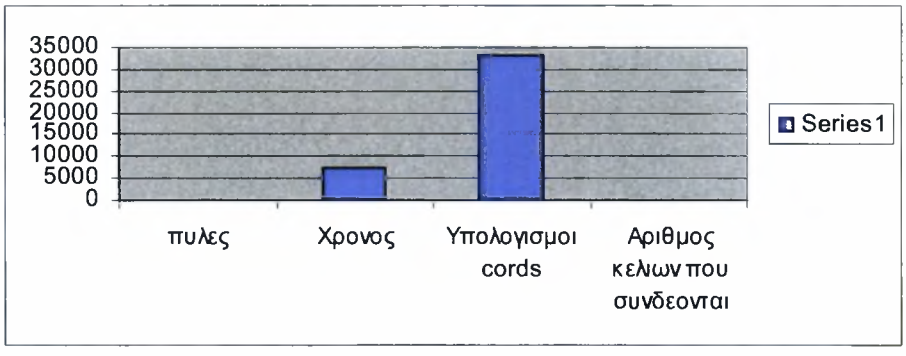
Β1. Αριθμός πυλών 14.



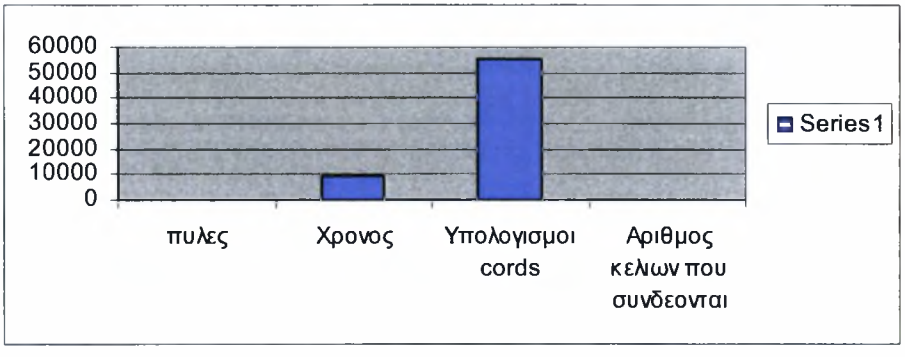
Β2. Αριθμός πυλών 90.



Β3. Αριθμός πυλών 150.



Β4. Αριθμός πυλών 198.



Παρατηρούμε οτι καθώς αυξάνεται ο αριθμός των πυλών ο χρόνος εκτέλεσης αυξάνεται αθροιστικά , ενώ ο συνολικός αριθμός υπολογισμού των συντεταγμένων αυξάνεται κατακόρυφα. Ο αριθμός των κελιών που συνδέονται ακολουθεί μια αυξητική πορεία παράλληλη με την αύξηση των πυλών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλια:

[1]. **Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL ,** Stephen Brown, Zvonko Vranesic.

[2]. **ΣΧΕΔΙΑΣΗ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ CMOS VLSI** , N. H. WEST, K.ESHRAGHIAN.

[3]. **Ψηφιακή Σχεδίαση** , M MORIS MANO.

[4]. **C για μηχανικούς** , H.H.TAN, T/B/ D'ORAZIO

[5]. **Η γλώσσα προγραματισμού C++ ,** BJARNE STROUSTRUP

[6]. **ΓΡΑΦΙΚΑ**

E-books :

[7] GORDIAN: VLSI Placement by Quadratic Programming and Slicing Opimization

Υλοποίηση του αλγορίθμου Placement σε γλώσσα C.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#define  FAKEINOUT 198
//#define  fakenumofgates 800

struct gates
{

        char fin_type[58];
        int gatenumber;
        int x;
        int y;
        float dist;
        int ingate[FAKEINOUT][9];
        int outgate;
        int num_input;
        int totalgates;
        //connect to...
        struct gates * nnode;
};

struct gates *current,*head;
int num_gates;
int num_connection;
int cellwidth = 8;
int cellheight = 5;
int chipdim;
FILE *fdw;
FILE *fdinit;
FILE *fdchip;
FILE *fdmincord;
FILE *fdfinal;

int init(int num_gates,int centerx,int centery);
int distance_min(int num_gates,int w[num_gates][num_gates],int yoff,int
num_connection,int chipdim);
void go_to(int temp);
int partition(int num_gates,int i1,int j1,int halffor_tempmin,int cord_distx[],int
cord_disty[],int w[num_gates][num_gates],int min,int num_connection,int chipdim);
void compute_cords(int for_mintemp[],int w[num_gates][num_gates],int num_1,int
*cell1,int *cell2,int num_connection);
```

```
int arraysize(int array[]);
void ok();

int main (int argc,char * argv[])
{
        time_t start, stop;
        double  diff;
        time(&start);
        FILE *fd;


        char c,tempc;

        fd=fopen("/home/sorolop/Desktop/ptyxiakh/spicesum","r+");
        fdw=fopen("/home/sorolop/Desktop/ptyxiakh/forcpp1","w");
        fdinit=fopen("/home/sorolop/Desktop/ptyxiakh/init","w");
        fdchip=fopen("/home/sorolop/Desktop/ptyxiakh/chip","w");
        fdmincord=fopen("/home/sorolop/Desktop/ptyxiakh/mincord","w");
        fdfinal=fopen("/home/sorolop/Desktop/ptyxiakh/final","w");

        if(fd!=0) printf("THE FILE HAS BEEN SUCCESSFULLY OPEN\n");
                int num_gates=0;
        while ((tempc=fgetc(fd))!=EOF)
        {
                if(tempc=='\n') num_gates++;

        }
        printf("Synolika exoume %d pules\n",num_gates);

        if(fseek(fd,0,SEEK_SET)!=0) printf("fseek error");//epanafora deikth sthn arxh

        head=(struct gates *)calloc(1,sizeof(struct gates));
        char tempc2;
        char tempc3;
        char number_c[58][58];
        char name[58][58];
        char character[58];
        char in_out[58][58];
        int number=0;
        int z1=0,z2=0,i1=0,z3=0;
        int temp=0;
        int number_i1=0,number_i2=0,number_i3=0;
        char temp_tempc2,temp_tempc3;
        current=head;
        int i=0;
        char input_c[80];
        int input_i1=0,input_i2=0;
        int input_i2_temp=0;
```

```
int i_temp1=1;
int arrin_1[FAKEINOUT][9];
int arrout_1[FAKEINOUT];
int i_tempinput1=0,i_tempinput2=0;
int i_rw1;
int i_temp11=0,i_temp12=1,i_temp13=1,i_temp14=0;
int i_temp121,i_temp131;
int i_1;
int i_temp2=1;
int i_rw2;
int i_temp21=0,i_temp22=0,i_temp23=0,i_temp24=0;

for(i1=0;i1<num_gates;i1++)
{
        do
        {
                tempc2=fgetc(fd);

                if(tempc2=='X')
                {
                        while(i_temp1==1)
                        {

                                if(i_temp11==0)
                                {
                                        i_temp1=fscanf(fd,"%d",&i_rw1);

                                        current->gatenumber=i_rw1;

                                }
                                else
                                {
                                        i_tempinput1++;
                                        i_temp1=fscanf(fd,"%d",&i_rw1);
                                        arrin_1[i_temp12][i_temp13]=i_rw1;
                                        i_temp13++;
                                }
                                i_temp11=1;


                        }

                        i_tempinput2=i_tempinput1-2;

                        current->num_input=i_tempinput2;
                        i_temp121=i_temp12;
                        i_temp131=i_temp13;
```

```
                                current->outgate=i_rw1;

                                i_temp1=1;
                                i_temp11=0;
                                i_temp13=1;
                                i_temp12++;
                                i_tempinput1=0;
                                for(i_1=1;i_1<i_temp131-2;i_1++)        //APO TO
[1][1]!!
                                {
                                        current-
>ingate[i_temp121][i_1]=arrin_1[i_temp121][i_1];
                                }

                        }

                        tempc2=fgetc(fd);


                        if(tempc2=='N')
                        {
                                name[i_temp21][i_temp22]=tempc2;
                                while((tempc2=fgetc(fd))&&(tempc2!='\n'))
                                {
                                        i_temp22++;
                                        name[i_temp21][i_temp22]=tempc2;

                                }
                                strcpy(current->fin_type,name[i_temp21]);
                        }
                        i_temp21++;
                        i_temp22=0;
                } while (tempc2!='\n'||tempc2==EOF);
                current->totalgates=num_gates;
                current->nnode=(struct gates *)calloc(1,sizeof(struct gates));
                current=current->nnode;
        }

        puts(" ");
        current=head;
        int j1;
        for (i1=1;i1<=num_gates;i1++)
        {
                printf("|Pulh No : %d \n",current->gatenumber);
                printf("|Typos pulhs :%s\n ",current->fin_type);
                for(j1=1;j1<=current->num_input;j1++)
                {
                        printf("Eisodos No %d : %d \n",j1,current->ingate[i1][j1]);
```

```
        }
        printf("Typos eksodoy %d\n",current->outgate);
        printf("|Eisodoi No :%d \n",current->num_input);
        current=current->nnode;

}
current=head;
//net list
j1=1;
i1=1;
int in[num_gates][num_gates];
int w[num_gates][num_gates];
int wtotal[num_gates];
int out[num_gates];
for (i1=0;i1<num_gates;i1++)
{
for (j1=0;j1<num_gates;j1++)
{
        in[i1][j1]=0;
        w[i1][j1]=0;
}
}

for (i1=0;i1<num_gates;i1++)
{
for (j1=0;j1<num_gates;j1++)
{
        in[i1][j1]=current->ingate[i1+1][j1+1];

}
out[i1]=current->outgate;
current=current->nnode;
}
int y1=0;
int num_connection=0;
for (i1=0;i1<num_gates;i1++)
{
for (j1=0;j1<num_gates;j1++)
{
        for (y1=0;y1<num_gates;y1++)
        {
                if(in[i1][j1]==out[y1]){w[i1][y1]=1;num_connection++;}
        }
}

}
fprintf(fdw,"%d\n",head->totalgates);
fprintf(fdw,"%d\n",num_connection);
```

```
        int wtemp=0;
        int i11=0;
        for (i1=0;i1<num_gates;i1++)
        {
        for (j1=0;j1<num_gates;j1++)
        {
                if(w[j1][i1]==1){wtemp++;}
        }
        wtotal[i1]=wtemp;

        wtemp=0;
        }

        //GORDIAN!
        chipdim=cellwidth*cellheight*num_gates;
        chipdim=(chipdim*20)/100 + chipdim;

        fprintf(fdchip,"%d\n",chipdim);


        int centerx=chipdim/2;
        int centery=chipdim/2;

        init(head->totalgates,centerx,centery);

        distance_min(head->totalgates,w,0,num_connection+1,chipdim);
        for (i1=0;i1<num_gates;i1++)
        {
        for (j1=0;j1<num_gates;j1++)
        {

                }

        }
        ok();
        fflush(fd);
        close(fd);
        time(&stop);
        diff=difftime(stop,start);
        printf("Diff is %.*fe%d\n",diff);
        return 0;
}
int init(int num_gate,int centerx,int centery)
{
   current=head;
   fprintf(fdinit,"%d\n",num_gate);
   int i,tempx,tempy;
```

```
  for(i=1;i<=num_gate;i++)
  {
    tempx=rand()%2;
    if(tempx==0)
    {
        current->x=centerx+rand()%centerx/4;

        fprintf(fdinit,"%d ",current->x);
    }
    else if(tempx==1)
    {
        current->x=centerx-rand()%centerx/4;

        fprintf(fdinit,"%d ",current->x);
    }
    tempy=rand()%2;
    if(tempy==0)
    {
        current->y=centery+rand()%centerx/4;

        fprintf(fdinit,"%d\n",current->y);
    }
    else if(tempy==1)
    {
        current->y=centery-rand()%centerx/4;

        fprintf(fdinit,"%d\n",current->y);
    }
    current=current->nnode;
  }
  return 0;
}

int distance_min(int num_gates,int w[num_gates][num_gates],int yoff,int
num_connection,int chipdim)
{
  current=head;
  int i1,j1;
  int dist_x1,dist_y1,dist_x2,dist_y2;
  int tmpp;
  int chipdimtemp=chipdim;
  int for_min[num_connection];
  int cord_distx[num_connection];
  int cord_disty[num_connection];
  int num_1=0;
  for(i1=0;i1<num_gates;i1++)//gia na vrw thn apostash kathe fora
  {
    for(j1=0;j1<num_gates;j1++)
```

```
    {
      if(w[i1][j1]==1)//prepei na to thewrhsoume logika +1
      {

            go_to(i1+1);
            dist_x1=current->x;
            dist_y1=current->y;
            go_to(j1+1);
            dist_x2=current->x;
            dist_y2=current->y;
            tmpp=(dist_y2-dist_y1)*(dist_y2-dist_y1) + (dist_x2-dist_x1)*(dist_x2-
dist_x1);
            cord_distx[num_1]=i1+1;
            cord_disty[num_1]=j1+1;
            for_min[num_1]=tmpp;

            num_1++;
      }
    }
  }

  //find min
  int for_tempmin[num_connection];
  for(i1=0;i1<num_connection;i1++)
  {
    for_tempmin[i1]=for_min[i1];
  }
  int mincount[num_1];

  int min;
  int i_min=0;
  for(i1=0;i1<num_connection;i1++)
  {  min=for_tempmin[i1];
    for(j1=i1;j1<num_connection;j1++)
    {
        if(min>=for_tempmin[j1])
        {
            min=for_tempmin[j1];
            i_min=j1;

        }

    }
    for_tempmin[i_min]=for_tempmin[i1];
    for_tempmin[i1]=min;

  }
 // puts("");
```

```
for(i1=0;i1<num_connection;i1++)
{


}
min=for_tempmin[0];

//Gia na vrw poia cells syndeontai
int cell1,cell2;

int y1=0;
num_1=0;
for(i1=0;i1<num_gates;i1++)
{
   for(j1=0;j1<num_gates;j1++)
   {
      if(w[i1][j1]==1)//prepei na to thewrhsoume logika +1
      {

            go_to(i1+1);
            dist_x1=current->x;
            dist_y1=current->y;
            go_to(j1+1);
            dist_x2=current->x;
            dist_y2=current->y;
            tmpp=(dist_y2-dist_y1)*(dist_y2-dist_y1) + (dist_x2-dist_x1)*(dist_x2-
dist_x1);
            while(tmpp!=for_tempmin[y1])
            {
                  y1++;
            }
            cord_distx[y1]=i1+1;
            cord_disty[y1]=j1+1;
            num_1++;

      }
      y1=0;
   }

}
for(j1=0;j1<num_1;j1++)
{

         if(num_1==1)
         {
               fprintf(fdmincord,"%d %d\n",cord_distx[j1],cord_disty[j1]);
         }
```

```
   }


   partition(head-
>totalgates,cord_distx[0],cord_disty[0],num_connection/2,cord_distx,cord_disty,w,min,n
um_connection,chipdimtemp);


 num_1=0;//giati to num_1 kathe fora tha mikranei kata 1;

}

int partition(int num_gates,int i1,int j1,int halffor_tempmin,int cord_distx[],int
cord_disty[],int w[num_gates][num_gates],int min,int num_connection,int chipdim)
{
   int dist_x1,dist_y1,dist_x2,dist_y2;
   int tmpp;
   int p1=0;
   int p2=0;
   int zeros=0;

   for(p1=0;p1<num_gates;p1++)
   {
      for(p2=0;p2<num_gates;p2++)
      {
          if(w[p1][p2]==1)
          {
                    zeros++;
          }
      }
   }
   if(zeros==1)
   {
      return 0;
   }
   else
   {
   for(p1=0;p1<num_gates;p1++)                                    //DELETE MIN
   {
      for(p2=0;p2<num_gates;p2++)
      {
         if(w[p1][p2]==1)//prepei na to thewrhsoume logika +1
         {

                  go_to(p1+1);
                  dist_x1=current->x;
                  dist_y1=current->y;
                  go_to(p2+1);
```

```
                dist_x2=current->x;
                dist_y2=current->y;
                tmpp=(dist_y2-dist_y1)*(dist_y2-dist_y1) + (dist_x2-dist_x1)*(dist_x2-
dist_x1);
                if(tmpp==min)
                {

                                fprintf(fdmincord,"%d %d\n",p1+1,p2+1);
                    w[p1][p2]=0;
                    break;
                }
        }
    }
  }
                                                //DELETE MIN


  current=head;
  int temp;

  for(p1=halffor_tempmin;p1<num_connection;p1++)
//Cell moved
  {
    temp=rand()%4;
    if(temp==0)
    {
    if(cord_distx[p1-1]!=cord_distx[0]&&cord_distx[p1-1]!=cord_disty[0])//prepei    na
to thewrhsoume logika +1
    {
       go_to(cord_distx[p1-1]+1);
       current->x=abs(current->x);
       current->y=abs(cellheight*cellwidth + current->y);// up

    }
    }
    if(temp==1)
    {
    if(cord_distx[p1-1]!=cord_distx[0]&&cord_distx[p1-1]!=cord_disty[0])//prepei    na
to thewrhsoume logika +1
    {
       go_to(cord_distx[p1-1]+1);
       current->x=abs(cellheight*cellwidth + current->x);
       current->y=abs(current->y);//right

    }
    }
    if(temp==2)
    {
```

```
    if(cord_distx[p1-1]!=cord_distx[0]&&cord_distx[p1-1]!=cord_disty[0])//prepei   na
to thewrhsoume logika +1
    {
       go_to(cord_distx[p1-1]+1);
       current->x=abs(cellheight*cellwidth - current->x);
       current->y=abs(current->y);//left

    }
    }
    if(temp==3)
    {
    if(cord_distx[p1-1]!=cord_distx[0]&&cord_distx[p1-1]!=cord_disty[0])//prepei   na
to thewrhsoume logika +1
    {
       go_to(cord_distx[p1-1]+1);
       current->x=abs(current->x);
       current->y=abs(cellheight*cellwidth - current->y);//down

    }
    }
    if(temp==3)
    {
    if(cord_disty[p1-1]!=cord_disty[0]&&cord_disty[p1-1]!=cord_distx[0])//prepei   na
to thewrhsoume logika +1
    {
        go_to(cord_disty[p1-1]+1);
        current->x=abs(current->x);
        current->y=abs(cellheight*cellwidth - current->y);//down

    }
    }
    if(temp==2)
    {
    if(cord_disty[p1-1]!=cord_disty[0]&&cord_disty[p1-1]!=cord_distx[0])//prepei   na
to thewrhsoume logika +1
    {
        go_to(cord_disty[p1-1]+1);
        current->x=abs(cellheight*cellwidth - current->x);
        current->y=abs(current->y);//left

    }
    }
    if(temp==1)
    {
    if(cord_disty[p1-1]!=cord_disty[0]&&cord_disty[p1-1]!=cord_distx[0])//prepei   na
to thewrhsoume logika +1
    {
        go_to(cord_disty[p1-1]+1);
```

```
            current->x=abs(cellheight*cellwidth + current->x);
            current->y=abs(current->y);//right

        }
        }
        if(temp==0)
        {
        if(cord_disty[p1-1]!=cord_disty[0]&&cord_disty[p1-1]!=cord_distx[0])//prepei    na
to thewrhsoume logika +1
        {
            go_to(cord_disty[p1-1]+1);
            current->x=abs(current->x);
            current->y=abs(cellheight*cellwidth + current->y);//up

        }
        }
    }

    current=head;
    for(i1=0;i1<num_gates;i1++)
    {

        fprintf(fdw,"%d %d \n",current->x,current->y);
        current=current->nnode;

    }


        distance_min(num_gates,w,0,num_connection-1,chipdim);

    }                                                    //Cell moved
}

void go_to(int temp)
{
    int i;
    current=head;
    for(i=1;i<=temp;i++)
    {
        if(i!=1)
        {
            current=current->nnode;
        }
    }
}

void ok()
{
```

```
int oktemp;
int okx[head->totalgates];
int oky[head->totalgates];
int p1,p2,i1;
int num_connection=1;
current=head;
if(num_connection==1)
{

        for(p1=0;p1<head->totalgates;p1++)
        {

                okx[p1]=current->x;
                oky[p1]=current->y;
        current=current->nnode;

        }
        current=head;
        for(p1=0;p1<head->totalgates;p1++)
        {
                for(p2=0;p2<head->totalgates;p2++)
                {
                        if(abs(okx[p1]-okx[p2])<cellwidth||abs(oky[p1]-
oky[p2])<cellwidth)
                        {


                                oktemp=rand()%4;
                                if(oktemp==0)
                                {
                                        go_to(p1+1);
                                        current->x=abs(current->x);
                                current->y=abs(cellwidth + current->y);//up


                                }
                                if(oktemp==1)
                                {
                                        go_to(p1+1);
                                        current->x=abs(cellwidth + current->x);
                                current->y=abs(current->y);//right


                                }
                                if(oktemp==2)
                                {
                                        go_to(p1+1);
                                        current->x=abs(cellwidth - current->x);
```

```
                    current->y=abs(current->y);//left


                    }
                    if(oktemp==3)
                    {
                            go_to(p1+1);
                            current->x=abs(current->x);
                    current->y=abs(cellwidth - current->y);//down


                    }

                    if(oktemp==0)
                    {
                            go_to(p2+1);
                            current->x=abs(current->x);
                    current->y=abs(cellwidth - current->y);//down


                    }
                    if(oktemp==1)
                    {
                            go_to(p2+1);
                            current->x=abs(cellwidth - current->x);
                    current->y=abs(current->y);//left


                    }
                    if(oktemp==2)
                    {
                            go_to(p2+1);
                            current->x=abs(cellwidth + current->x);
                    current->y=abs(current->y);//right


                    }
                    if(oktemp==3)
                    {
                            go_to(p2+1);
                            current->x=abs(current->x);
                    current->y=abs(cellwidth + current->y);//up


                    }
            }
    }
}
```

```
            current=head;
            for(p1=0;p1<head->totalgates;p1++)
            {
            fprintf(fdfinal,"%d %d \n",current->x,current->y);
            current=current->nnode;

            }
        }
}
```

Παρουσίαση αποτελεσματων με τη χρήση της C++,openGL

```
#pragma comment( linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"" )

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <stdio.h>
//#include <windows.h>
#include <time.h>
#include <stdio.h>

#include <pthread.h>
using std::ifstream;
using std::ofstream;
using std::cout;
time_t start_time, cur_time;

struct cord{
int x;
int y;
int gates;
int connection;
int initx;
int inity;
int center;
int chip;
int finalx;
int finaly;

struct cord *nnode;
};
struct temp{
int xt;
int yt;
struct temp *nextnode;
};
struct temp2{
int cordx;
int cordy;
struct temp2 *nextnode2;
};
struct cord *current,*head;
struct temp *currentt,*headt;
```

```
struct temp2 *currenttt,*headtt;

void go_to(int temp);

void myInit (void)
{
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(1.0, 1.0, 1.0);
        glPointSize(4);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,current->chip, 0.0,current->chip);
}

void myDisplay(void)
{
  double color=0.1;
  int i;

  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_LINES);
  glColor3f(0.0,1.0,1.0);
  glVertex2f(current->chip/2,0);
  glVertex2f(current->chip/2,current->chip);

  glEnd();

  glBegin(GL_LINES);
  glColor3f(0.0,1.0,1.0);
  glVertex2f(0,current->chip/2);
  glVertex2f(current->chip,current->chip/2);

  glEnd();
       glBegin(GL_LINE_LOOP);
  glColor3f(0.0,1.0,1.0);

  glVertex2f(current->chip/2-4,current->chip/2+2);
  glVertex2f(current->chip/2+4,current->chip/2+2);
  glVertex2f(current->chip/2+4,current->chip/2-2);
  glVertex2f(current->chip/2-4,current->chip/2-2);

  glEnd();

  for(i=1;i<=current->gates;i++)
  {


    glBegin(GL_POINTS);
    glVertex2i(current->initx,current->inity);

    glColor3f(0.0,1.0,0.0);

    glEnd();
```

```
    glBegin(GL_LINE_LOOP);
    glColor3f(0.0,1.0,1.0);

    glVertex2f(current->initx-4,current->inity+2);
    glVertex2f(current->initx+4,current->inity+2);
    glVertex2f(current->initx+4,current->inity-2);
    glVertex2f(current->initx-4,current->inity-2);

    glEnd();
current=current->nnode;


    }

    double colorx;
    double colory=0.0;
    double colorz=0.0;
    current=head;
    int i1,y1;
    for(i=1;i<=current->gates*(current->connection);i++)
    {


            glBegin(GL_POINTS);

            glVertex2i(current->x,current->y);

            if(i%current->gates==0||i==1)//axxristo
    {
            colorx=color;

        glColor3f(colorx,colory,colorz);
        color=color+0.1;

    }


            glEnd();
            int o,p;
            for(o=1;o<=200;o++)
            {
                    for(p=1;p<200;p++)
                    {
                    }
            }

            current=current->nnode;
            glFlush();
    }

    current=head;
```

```
    for(i=1;i<=current->gates;i++)
    {
            glBegin(GL_POINTS);
      colorx=color;
      colory=(double)(color*2);
      colorz=(double)(color/2);
      glColor3f(0.0,0.0,1.0);
      color=color+0.1;

            glVertex2i(current->finalx,current->finaly);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glColor3f(0.0,0.0,1.0);

            glVertex2f(current->finalx-20,current->finaly+10);
            glVertex2f(current->finalx+20,current->finaly+10);
            glVertex2f(current->finalx+20,current->finaly-10);
            glVertex2f(current->finalx-20,current->finaly-10);

            glEnd();

            current=current->nnode;

    }
    current=head;
    glBegin(GL_POINTS);

  glColor3f(1.0,0.0,0.5);
    glVertex2i(current->chip/2,current->chip/2);

  glEnd();
    currenttt=headtt;
    for(i=1;i<=current->connection;i++)
    {


            glBegin(GL_LINES);
            go_to(currenttt->cordx);
            glColor3f(0.0,1.0,0.0);
            glVertex2f(current->finalx,current->finaly);
            go_to(currenttt->cordy);
            glVertex2f(current->finalx,current->finaly);

            currenttt=currenttt->nextnode2;

            //sleep(0.8);
            glEnd();
            //glFlush();

    }
    cout<<current->connection<<"A";
            glFlush();
}
```

```
void keyEvent(unsigned char key, int x, int y)
{
	switch (key)
	{

		case 'Q':
		case 'q':

		exit(-1);
		case 'a': myDisplay();
		break;
		default:
		break;
	}
}


int main(int argc, char **argv)
{

	head=(struct cord *)calloc(1,sizeof(struct cord));
	current=head;
	headt=(struct temp *)calloc(1,sizeof(struct temp));
	currentt=headt;
	headtt=(struct temp2 *)calloc(1,sizeof(struct temp2));
	currenttt=headtt;
	ifstream inStream;
	inStream.open("/home/sorolop/Desktop/ptyxiakh/forcpp1");
	ifstream inStream3;
	inStream3.open("/home/sorolop/Desktop/ptyxiakh/chip");
	ifstream inStream4;
	inStream4.open("/home/sorolop/Desktop/ptyxiakh/mincord");
	ifstream inStream5;
	inStream5.open("/home/sorolop/Desktop/ptyxiakh/final");
	int num_gates;
	int num_connection;
	int chipcord;
	inStream3>>chipcord;
	inStream>>num_gates;

	inStream>>num_connection;
	int i,j;


	for(i=1;i<=num_gates*(num_connection);i++)
	{
	inStream>>current->x>>current->y;
		currentt->xt=current->x;
		currentt->yt=current->y;

	current->gates=num_gates;
	current->connection=num_connection;
```

```
    current->chip=chipcord;

    current->nnode=(struct cord *)calloc(1,sizeof(struct cord));
            currentt->nextnode=(struct temp *)calloc(1,sizeof(struct temp));
    current=current->nnode;
            currentt=currentt->nextnode;
    }

                                    //init ...
    current=head;
    ifstream inStream2;
    inStream2.open("/home/sorolop/Desktop/ptyxiakh/init");
    int initgates;
    inStream2>>initgates;

    for(i=1;i<=initgates;i++)
    {
    inStream2>>current->initx>>current->inity;
            inStream5>>current->finalx>>current->finaly;

current=current->nnode;

    }
    current=head;
    currentt=headt;
    for(i=1;i<=num_connection;i++)
    {
    inStream4>>currenttt->cordx>>currenttt->cordy;
    cout << i <<" ! "<< currenttt->cordx <<" "<< currenttt->cordy <<"\n";
            currenttt->nextnode2=(struct temp2 *)calloc(1,sizeof(struct temp2));
    currenttt=currenttt->nextnode2;

    }
    currentt=headt;
    currenttt=headtt;

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize (chipcord, chipcord);

    glutInitWindowPosition(1,1);

    glutCreateWindow ("Basic_OpenGL_GLUT");

    glutKeyboardFunc(keyEvent);

    glutDisplayFunc(myDisplay);

    myInit();
```

```
        glutMainLoop();



        return 0;
}

void go_to(int temp)
{
   int i;
   current=head;
   for(i=1;i<=temp;i++)
   {
      if(i!=1)
      {
            current=current->nnode;
      }
   }

}
```

Προσωμιωση κατασκεύης του ολοκληρωμένου κυκλώματος σε standard cell τεχνολογία με τη χρήση C++ και openGl .

```
#pragma comment( linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"" )


#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <stdio.h>
//#include <windows.h>
#include <time.h>
#include <stdio.h>

#include <pthread.h>
using std::ifstream;
using std::ofstream;
using std::cout;
time_t start_time, cur_time;

struct cord{
int x;
int y;
int gates;
int connection;
int initx;
int inity;
int center;
int chip;
int finalx;
int finaly;
int scellx;
int scelly;

struct cord *nnode;
};
struct temp{
```

```
int xt;
int yt;
struct temp *nextnode;
};
struct temp2{
int cordx;
int cordy;
int gateorder;
struct temp2 *nextnode2;
};
 struct cord *current,*head;
struct temp *currentt,*headt;
struct temp2 *currenttt,*headtt;
void go_to(int temp);
void myInit (void)
{
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(1.0, 1.0, 1.0);
        glPointSize(5);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,current->chip, 0.0,current->chip);
}

void myDisplay(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        double color=0.1;
        int i;

        int axonasx2,axonasy2;
        axonasx2=current->gates/2;
        axonasy2=2;
        int linex2=current->chip/(axonasx2);
        int linextemp2=linex2;
        linex2=0;
        for(i=1;i<=axonasx2;i++)
        {
                glBegin(GL_LINES);
                glColor3f(1.0,0.0,0.0);
                glVertex2f(0,linex2);
                glVertex2f(current->chip,linex2);
                glEnd();
                linex2=linex2+linextemp2;
        }

        for(i=1;i<=axonasy2;i++)
        {
```

```
            glBegin(GL_LINES);
            glColor3f(1.0,0.0,0.0);
            glVertex2f(current->chip/2,0);
            glVertex2f(current->chip/2,current->chip);
            glEnd();
    }


    double colorx;
    double colory=0.0;
    double colorz=0.0;
    current=head;
    int i1,y1;

    current=head;
    for(i=1;i<=current->gates;i++)
    {
            glBegin(GL_POINTS);
       glColor3f(0.0,0.0,1.0);
            glVertex2i(current->scellx,current->scelly);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glColor3f(1.0,0.0,1.0);

            glVertex2f(current->scellx-8,current->scelly+5);
            glVertex2f(current->scellx+8,current->scelly+5);
            glVertex2f(current->scellx+8,current->scelly-5);
            glVertex2f(current->scellx-8,current->scelly-5);

            glEnd();

            current=current->nnode;
      // glFlush();
    }
    current=head;
    glBegin(GL_POINTS);

glColor3f(1.0,0.0,0.5);
    glVertex2i(current->chip/2,current->chip/2);

glEnd();

    currenttt=headtt;
    for(i=1;i<=current->connection;i++)
    {


            glBegin(GL_LINES);
```

```
                go_to(currenttt->cordx);
                glColor3f(0.0,1.0,0.0);
                glVertex2f(current->scellx,current->scelly);
                go_to(currenttt->cordy);
                glVertex2f(current->scellx,current->scelly);

                currenttt=currenttt->nextnode2;

                //sleep(2);
                glEnd();
                //glFlush();

        }

                glFlush();
}

void keyEvent(unsigned char key, int x, int y)
{
        switch (key)
        {

                case 'Q':
                case 'q':

                exit(-1);
                case 'a': myDisplay();
                break;
                default:
                break;
        }
}


int main(int argc, char **argv)
{

        head=(struct cord *)calloc(1,sizeof(struct cord));
        current=head;
        headt=(struct temp *)calloc(1,sizeof(struct temp));
        currentt=headt;
        headtt=(struct temp2 *)calloc(1,sizeof(struct temp2));
        currenttt=headtt;
        ifstream inStream;
        inStream.open("/home/sorolop/Desktop/ptyxiakh/forcpp1");
        ifstream inStream3;
        inStream3.open("/home/sorolop/Desktop/ptyxiakh/chip");
        ifstream inStream4;
```

```
inStream4.open("/home/sorolop/Desktop/ptyxiakh/mincord");
ifstream inStream5;
inStream5.open("/home/sorolop/Desktop/ptyxiakh/final");
int num_gates;
int num_connection;
int chipcord;
inStream3>>chipcord;
inStream>>num_gates;

inStream>>num_connection;
int i,j;

for(i=1;i<=num_gates*(num_connection);i++)
{
inStream>>current->x>>current->y;
        currentt->xt=current->x;
        currentt->yt=current->y;

current->gates=num_gates;
current->connection=num_connection;
current->chip=chipcord;

current->nnode=(struct cord *)calloc(1,sizeof(struct cord));
        currentt->nextnode=(struct temp *)calloc(1,sizeof(struct temp));
current=current->nnode;
        currentt=currentt->nextnode;
 }

                                //init ...
current=head;
ifstream inStream2;
inStream2.open("/home/sorolop/Desktop/ptyxiakh/init");
int initgates;
int finalgates1[num_connection];
int finalgates2[num_connection];
int finaltemp[num_connection*2];
inStream2>>initgates;
for(i=1;i<=initgates;i++)
{
inStream2>>current->initx>>current->inity;
inStream5>>current->finalx>>current->finaly;
current=current->nnode;

}



current=head;
```

```
currentt=headt;
for(i=0;i<num_connection;i++)
{
inStream4>>currenttt->cordx>>currenttt->cordy;
        finalgates1[i]=currenttt->cordx;
        finalgates2[i]=currenttt->cordy;

        currenttt->nextnode2=(struct temp2 *)calloc(1,sizeof(struct temp2));
currenttt=currenttt->nextnode2;

}
currentt=headt;
currenttt=headtt;

int final[num_gates];
int fin1=0;
int fin2=0;
for(i=0;i<num_connection*2;i++)
{
        if(i%2==0)
        {
                finaltemp[i]=finalgates1[fin1];
                fin1++;
        }
        if(i%2==1)
        {
                finaltemp[i]=finalgates2[fin2];
                fin2++;
        }

}

int temptemp[num_connection*2];
for(i=0;i<num_connection*2;i++)
{
                temptemp[i]=finaltemp[i];
}
for(i=0;i<num_connection*2;i++)
{
        for(j=i;j<num_connection*2;j++)
        {

                if(finaltemp[i]==temptemp[j+1])
                {
                        temptemp[j+1]=0;

                }
```

```
        }
}
int fin_temp=0;
for(i=0;i<num_connection*2;i++)
{
        if(temptemp[i]!=0)
        {
                final[fin_temp]=temptemp[i];
                fin_temp++;
        }
}


int axonasx,axonasy;
axonasy=2;
axonasx=current->gates/axonasy;

int linex=current->chip/(axonasx);
int linextemp=linex;
int linex2temp=linex/axonasy;
linex=0;
int linexstatic=current->chip/4;
int for_cell1;
int for_cell2;

for(i=1;i<=num_gates;i++)
{
        if(i%2==1)
        {
                if(i==1)
                {
                        current->scellx=current->chip/axonasy - linexstatic;
                        current->scelly=current->chip - linex2temp;
                        for_cell1=current->scelly;
                        current=current->nnode;
                }
                else
                {
                        current->scellx=current->chip/axonasy - linexstatic;
                        current->scelly=for_cell1 - linextemp;
                        for_cell1=current->scelly;

                        current=current->nnode;


                }
        }
        if(i%2==0)
```

```
        {
                if(i==2)
                {
                        current->scellx=current->chip/axonasy + linexstatic;
                        current->scelly=current->chip - linex2temp;
                        for_cell2=current->scelly;
                        current=current->nnode;
                }
                else
                {
                        current->scellx=current->chip/axonasy + linexstatic;
                        current->scelly=for_cell2 - linextemp;
                        for_cell2=current->scelly;


                        current=current->nnode;

                }

        }

}

current=head;

currenttt=headtt;

for(i=0;i<num_gates;i++)
{

        currenttt->gateorder=final[i];
        cout << i <<" : "<< final[i]<<"\n";
        currenttt=currenttt->nextnode2;
}


glutInit(&argc, argv);

glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

glutInitWindowSize (chipcord, chipcord);

glutInitWindowPosition(1,1);

glutCreateWindow ("Basic_OpenGL_GLUT");

glutKeyboardFunc(keyEvent);
```

```
        glutDisplayFunc(myDisplay);



        myInit();


        glutMainLoop();




        return 0;
}

void go_to(int temp)
{
   int i;
   current=head;
   for(i=1;i<=temp;i++)
   {
      if(i!=1)
      {
            current=current->nnode;
      }
   }

}
```