ΤΕΧΝΙΚΕΣ ΤΟΠΟΘΕΤΗΣΗΣ ΚΑΙ ΔΙΑΣΥΝΔΕΣΗΣ ΠΡΟΤΥΠΩΝ ΛΟΓΙΚΩΝ ΚΥΤΤΑΡΩΝ VLSI ΚΥΚΛΩΜΑΤΩΝ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Δημήτρη Μαρκούζη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Νοέμβριος 2007

ΑΦΙΕΡΩΣΗ

Στον πατέρα μου

ΕΥΧΑΡΙΣΤΙΕΣ

Πρώτα από όλα θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Ιωάννη Φούντο για τη πολύτιμη βοήθεια του, τις σημαντικές παρατηρήσεις του και την υπομονή που έδειξε σε όλη τη διάρκεια της παρούσας εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω το Χ. Καλογήρου, το Γ. Ρήγα, το Ι. Καραμήτσιο και όλους τους συναδέλφους και φίλους μου που με βοήθησαν στην διεκπεραίωση της διατριβής αυτής.

Ακόμη, δε θα μπορούσα να εξαιρέσω την Κατερίνα Βιτουλαδίτη, οι φιλολογικές παρατηρήσεις της οποίας και η ηθική υποστήριξη που μου παρείχε ήταν ανεκτίμητες.

Τέλος, το μεγαλύτερο ευχαριστώ το οφείλω στη μητέρα μου και την αδερφή μου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΠΕΡΙΛΗΨΗ

Δημήτριος Μαρκούζης του Παύλου και της Σιδερής.
MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Νοέμβριος 2007.
ΤΕΧΝΙΚΕΣ ΤΟΠΟΘΕΤΗΣΗΣ ΚΑΙ ΔΙΑΣΥΝΔΕΣΗΣ ΠΡΟΤΥΠΩΝ ΛΟΓΙΚΩΝ ΚΥΤΤΑΡΩΝ VLSI ΚΥΚΛΩΜΑΤΩΝ.
Επιβλέπων: Ιωάννης Φούντος.

Στην εργασία αυτή προτείναμε την τοποθέτηση και τη διασύνδεση λογικών πυλών VLSI κυκλωμάτων για τεχνολογίες Standard Cells. Αρχικά, υλοποιήθηκαν δύο ξεχωριστοί αλγόριθμοι για την τοποθέτηση των λογικών κυττάρων στην επιφάνεια του τσιπ. Οι αλγόριθμοι αυτοί είναι βασισμένοι στη θεωρία γράφων καθώς το κύκλωμα αναπαραστάθηκε με τη μορφή γράφου. Ο πρώτος χρησιμοποιεί βασικούς αλγόριθμους διάσχισης γράφων όπως είναι ο dfs, ενώ ο δεύτερος υπολογίζει πυκνούς υπογράφους. Κάθε πυκνός υπογράφος αναπαραστά και ένα κομάτι του λογικού κυκλώματος το οποίο έχει πολλές συνδέσεις μεταξύ των λογικών στοιχείων του. Έτσι λοιπόν στοιχεία που ανήκουν στον ίδιο πυκνό υπογράφο τοποθετούνται κοντά το ένα στο άλλο. Τέλος, χρησιμοποιήθηκε ένας maze αλγόριθμος που με τη βοήθεια μιας ευριστικής συνάρτησης κατάφερε να διασυνδέσει τις πύλες μεταξύ τους.

Οι επιδόσεις των αλγορίθμων εξετάστηκαν ως προς το χρόνο εκτέλεσης τους, την ελάχιστη επιφάνεια του chip, που μπορούσαν να χρησιμοποιήσουν και το μήκος των διασυνδέσεων που απαιτείται για τη διασύνδεση των λογικών πυλών.

EXTENDED ABSTRACT IN ENGLISH

Markouzis Dimitris, Msc, Computer Science Department, University of Ioannina, Greece. November 2007. “Placement and Routing Methods of VLSI Circuits using Standard Cell Technologies.”
Thesis Supervisor: Ioannis Fudos.

Standard cell placement and routing is an important open problem in current CAD VLSI research. We present a novel approach to placement and routing in standard cell arrays. Placement is performed by an algorithm that places the standard cells in a spiral manner around the center of the cell array driven by a DFS on the interconnection graph. We provide an improvement of this technique by first detecting dense graphs in the interconnection graph and then placing cells that belong to denser graphs closer to the center of the spiral. This will minimize the wirelength required for routing. Routing is performed by a variation of the maze algorithm enhanced by a set of heuristics that have been tuned to maximize performance. Finally we present a visualization tool and an extensive experimental performance evaluation of our approach.

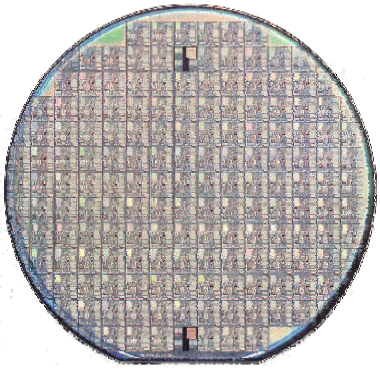
ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

1.1. Ιστορική Αναδρομή – Εφαρμογές VLSI

Η διατριβή αυτή πραγματεύεται την τοποθέτηση (placement) και τη διασύνδεση (routing) λογικών πυλών για τη δημιουργία λογικών κυκλωμάτων υψηλής ολοκλήρωσης. Τα κυκλώματα αυτά αποτελούν το κύριο λειτουργικό στοιχείο των ηλεκτρονικών υπολογιστών και των περισσοτέρων σύγχρονων ηλεκτρονικών εξαρτημάτων. Τα συναντάμε σε καθημερινά προϊόντα όπως είναι τα ψηφιακά ρολόγια, διάφορες οικιακές συσκευές, CD-DVD players, ηλεκτρονικά παιχνίδια, κινητά τηλέφωνα καθώς και σε μεγάλα συστήματα όπως είναι ο εξοπλισμός των δικτύων τηλεφωνίας και τηλεόρασης.

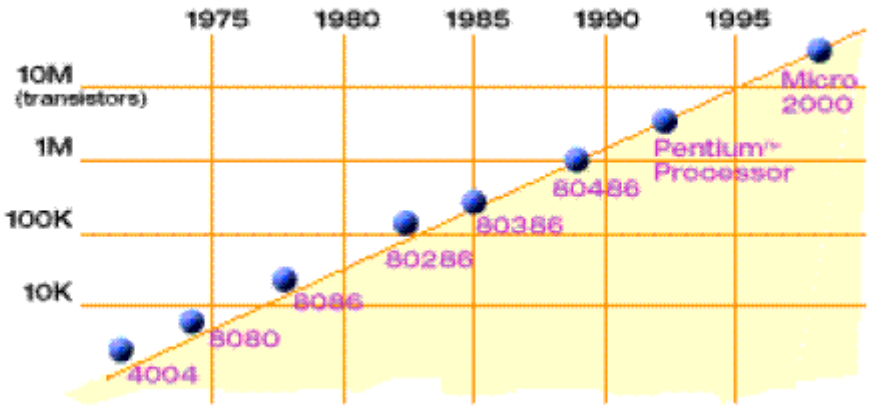
Η τεχνολογία που χρησιμοποιείται για τη δόμηση των παραπάνω συστημάτων έχει εξελιχθεί θεαματικά τις τελευταίες δεκαετίες. Έως το 1960 τα λογικά κυκλώματα κατασκευάζονταν από ογκώδη τεμάχια, όπως είναι τα τρανζίστορ και οι αντιστάσεις, τα οποία συνδέονταν ως χωριστά κομμάτια. Η ανάπτυξή των ολοκληρωμένων κυκλωμάτων κατέστησε δυνατή την τοποθέτηση πολλών τρανζίστορ μέσα σε ένα τεμάχιο που ονομάζεται τσιπ (chip). Στην αρχή τα κυκλώματα αυτά περιείχαν λίγα τρανζίστορ αλλά με την εξέλιξη της τεχνολογίας μπορούμε πλέον να τοποθετήσουμε στην ίδια μικρή επιφάνεια εκατοντάδες χιλιάδες από αυτά. Τα ολοκληρωμένα

κυκλώματα κατασκευάζονταν και συνεχίζουν να κατασκευάζονται πάνω σε δίσκους πυριτίου (Σχήμα 1.1) [17].



Σχήμα 1.1 Δίσκος Πυριτίου (wafer)

Από το 1970 ήταν δυνατή η τοποθέτηση όλων των στοιχείων κυκλώματος που ήταν αναγκαία για τη δημιουργία ενός ολοκληρωμένου κυκλώματος μικροεπεξεργαστή σε ένα ενιαίο τεμάχιο. Πριν από περίπου 40 χρόνια ο Gordon Moore, πρόεδρος της εταιρίας Intel παρατήρησε ότι η τεχνολογία των κυκλωμάτων εξελισσόταν με καταπληκτικούς ρυθμούς, διπλασιάζοντας των αριθμό των τρανζίστορ που μπορούσαν να χωρέσουν μέσα σε ένα ολοκληρωμένο κύκλωμα κάθε 1,5 με 2 χρόνια.



Σχήμα 1.2 Γραφική Παράσταση του Νόμου του Moore

Το φαινόμενο αυτό που είναι γνωστό ως νόμος του Moore (Σχήμα 1.2)[26] επιβεβαιώνεται μέχρι και σήμερα με αποτέλεσμα η τοποθέτηση περισσοτέρων από 10

εκατομμύρια τρανζίστορ να είναι εφικτή στα τέλη του 2007. Έτσι λοιπόν, στο πέρασμα του χρόνου τα κυκλώματα χωρίστηκαν σε κατηγορίες ανάλογα με τον αριθμό των λογικών πυλών που περιέχει κάθε chip, όπως φαίνεται στον Πίνακα 1.1

Πίνακάς 1.1 Ιστορική Εξέλιξη των Λογικών Κυκλωμάτων

Μικρής Κλίμακας Ολοκλήρωσης (Small Scale Integration, SSI)	**Περιέχουν λίγες (5-10) μεμονωμένες πύλες**
Μέσης Κλίμακας Ολοκλήρωσης (Medium Scale Integration, MSI)	**Περιέχουν 10-100 πύλες κατάλληλα συνδεδεμένες, ώστε να σχηματίζουν ένα ψηφιακό κύκλωμα**
Μεγάλης Κλίμακας Ολοκλήρωσης (Large Scale Integration, LSI)	**Περιέχουν περισσότερες από 100 πύλες μέχρι μερικές χιλιάδες πύλες (όπως οι απλοί μικροεπεξεργαστές)**
Πολύ Μεγάλης Κλίμακας Ολοκλήρωσης (Very Large Scale Integration, VLSI)	**Περιέχουν κυκλώματα τα οποία σχηματίζονται από πολλά εκατομμύρια πύλες (π.χ. σύγχρονοι μικροεπεξεργαστές)**

1.2. Ορισμός του Προβλήματος

Όπως αναφέρθηκε προηγούμενα, το πλήθος των λογικών πυλών που μπορούν να τοποθετηθούν σε ένα τσιπ μπορεί να φτάσει τα μερικά εκατομμύρια. Αυτό έχει ως αποτέλεσμα, η τοποθέτηση των πυλών πάνω στο chip αλλά και η μεταξύ τους διασύνδεση να παίζουν βασικό ρόλο στις επιδόσεις και τη συμπεριφορά του τσιπ.

Αρχικά -όπως αναφέρεται στο [1]- οι πύλες τοποθετούνται πάνω στο chip με τέτοιο τρόπο ώστε να μειωθεί όσο γίνεται περισσότερο το μήκος των ηλεκτρικών καλωδίων (*wires*), που θα τις διασυνδέσουν στα επόμενα βήματα. Εξαιτίας του μεγάλου αριθμού των πυλών (*gates*) συνηθίζεται το πρόβλημα της τοποθέτησης να διασπάται σε μικρότερα υποπροβλήματα με τη χρήση κάποιων λογικών κριτηρίων. Έτσι λοιπόν, το αρχικό κύκλωμα διαμερίζεται σε μικρότερα κυκλώματα. Στη συνέχεια, λύνεται το κάθε υποπρόβλημα χωριστά και τέλος τοποθετούνται στην επιφάνεια του τσιπ τα

επιμέρους κυκλώματα. Σε πολλές περιπτώσεις, γίνεται μια αρχική τοποθέτηση των πυλών, η οποία βελτιώνεται επαναληπτικά μέχρι να επιτευχθούν ορισμένοι περιορισμοί. Συνεπώς, μπορούμε να πούμε ότι το πρόβλημα της τοποθέτησης είναι ένα πρόβλημα βελτιστοποίησης (constrained optimization problem).

Μετά την τοποθέτηση ακολουθεί η διασύνδεση των πυλών (routing). Αυτή υλοποιείται με τέτοιο τρόπο ώστε να έχουμε τη μικρότερη δυνατή συμφόρηση από wires, τα λιγότερα δυνατά επίπεδα διασύνδεσης (όσο περισσότερα είναι τόσο αυξάνει το κόστος κατασκευής του chip), τις λιγότερες δυνατές στροφές (bends) των wires κτλ. Η διαδικασία που συνήθως ακολουθείται υπολογίζει σε πρώτη φάση τις σχετικές θέσεις των wires και σε δεύτερη φάση την οριστική θέση τους πάνω στο chip.

Στην εργασία αυτή, αρχικά προσπαθούμε να τοποθετήσουμε τις πύλες χρησιμοποιώντας διάφορους τρόπους προσέγγισης, αναπαριστώντας το κύκλωμα με γράφο. Στη συνέχεια, έχοντας υπολογίσει τις ακριβείς θέσεις των πυλών στο chip, προσπαθούμε να τις διασυνδέσουμε με τη βοήθεια ενός greedy αλγόριθμου.

1.3. Σχετικές Εργασίες

Πολλές είναι οι εργασίες που υπάρχουν μέχρι σήμερα και εξετάζουν από διαφορετική οπτική γωνία το πρόβλημα της τοποθέτησης και της διασύνδεσης στην περιοχή των VLSI κυκλωμάτων.

Οι Z. Yang και S. Areibi [2] παραθέτουν και συγκρίνουν δύο αλγόρίθμους σχετικούς με την τοποθέτηση λογικών πυλών στο chip. Ο πρώτος είναι ένας γενετικός αλγόριθμός, όπου ξεκινώντας από μια αρχική τυχαία τοποθέτηση προσπαθεί να τη βελτιώσει σε κάθε βήμα χρησιμοποιώντας μια συνάρτηση πρόβλεψης του μήκους των wires που θα χρησιμοποιηθούν. Ο δεύτερος αναπαριστά το κύκλωμα με έναν υπεργράφο, όπου οι πύλες του κυκλώματος είναι οι κόμβοι του και οι ακμές του είναι οι συνδέσεις μεταξύ των πυλών, τον οποίο διαμερίζει σε υπογράφους, ανάλογα με τη συνδεσιμότητα μεταξύ των πυλών. Στη συνέχεια, τοποθετεί με συγκεκριμένη σειρά έναν – έναν υπογράφο (επιμέρους κύκλωμα) πάνω στην επιφάνεια του chip. Η

λογική προσέγγιση αυτή μοιάζει αρκετά με τη δικιά μας, αφού και εμείς διαμερίζουμε το κύκλωμα ανάλογα με τον τρόπο σύνδεσης των λογικών πυλών.

Οι Bo Hu και Malgorzata Marek-Sadowska [3] προτείνουν μια ευριστική συνάρτηση εκτίμησης του μήκους των wires, η οποία υπολογίζεται κατά τη διάρκεια του placement. Όπως και οι προηγούμενοι, έτσι και αυτοί αναπαριστούν το κύκλωμα με έναν υπεργράφο, σε κάθε υπερακμή του οποίου αναθέτουν ένα βάρος με βάση την ευριστική συνάρτηση. Η τιμή της συνάρτησης εξαρτάται από τον αριθμό των κόμβων που μοιράζονται μια ακμή (λογική σύνδεση του κυκλώματος) και είναι τόσο μεγαλύτερη όσο μεγαλύτερος είναι ο αριθμός των κόμβων αυτών. Οι συνδέσεις λοιπόν τοποθετούνται ταξινομημένα σε μια ουρά ξεκινώντας από αυτή με το μεγαλύτερο βάρος. Σε κάθε βήμα του αλγορίθμου της τοποθέτησης εξάγεται η σύνδεση που βρίσκεται στην κορυφή της ουράς και οι κόμβοι που συμμετέχουν σε αυτή τοποθετούνται στην επιφάνεια του chip. Η διαφορά αυτής με τη δικιά μας μέθοδο είναι στο γεγονός ότι τα βάρη των ακμών και των κόμβων που αναθέτουμε είναι τα ίδια για όλους τους κόμβους και τις ακμές, ενώ η «ευριστική» συνάρτηση που χρησιμοποιούμε είναι η πυκνότητα του γράφου.

Οι X. Yang, M. Wang, R. Kastner, S. Ghiasi και M. Sarrafzadeh [4] παρουσιάζουν μια τεχνική βασισμένη σε Integer Linear Programming (ILP) για να προβλέψουν τη συμφόρηση των wires. Η μέθοδος αυτή εκτελείται κατά τη διάρκεια της τοποθέτησης ως ένα βήμα περαιτέρω επεξεργασίας αυτής. Το πρόβλημα της συμφόρησης μετατρέπεται σε ένα ILP πρόβλημα και παραθέτονται προσεγγιστικοί αλγόριθμοι για να το επιλύσουν. Σκοπός τους είναι να καταλήξουν σε μια τελική τοποθέτηση των λογικών πυλών που προκύπτει από συνεχείς αλλαγές της αρχικής τέτοια ώστε η προβλεπόμενη συμφόρηση να είναι η μικρότερη δυνατή.

Οι Chris Chu και Yiu-Chung Wong [5] χρησιμοποιούν έναν αλγόριθμο βασισμένο σε Rectilinear Steiner Minimal Trees (RSMT) για τον υπολογισμό του μήκους των διασυνδέσεων, ο οποίος ονομάζεται FLUTE. Αυτός έχει καλά αποτελέσματα για κυκλώματα με μικρό αριθμό πυλών και για το λόγο αυτό προτείνουν και τρεις ευριστικές συναρτήσεις για τη διάσπαση του αρχικού κυκλώματος. Στη συνέχεια για

κάθε υποκύκλωμα κατασκευάζεται ένα RSMT για τον υπολογισμό του μήκους των wires.

Στο [6] εξηγείται μια στρατηγική που έχει να κάνει με τη διασύνδεση των λογικών πυλών. Σύμφωνα με αυτή εκτελείται ένας «διαίρει και βασίλευε» αλγόριθμος σε δύο στάδια για να μπορεί να χειριστεί σχετικά εύκολα μεγάλα chips. Το πρώτο στάδιο περιέχει ένα βρόχο ελαχιστοποίησης και διαμέρισης της επιφάνειας, έτσι ώστε να προκύψουν όσο το δυνατό περισσότερα switchboxes. Τέλος, στο δεύτερο στάδιο επιχειρείται η ακριβής διασύνδεση αυτών.

Επίσης, στο [7] επιχειρείται ένας αλγόριθμος διασύνδεσης, οποίος λαμβάνει κυρίως υπόψη του τις καθυστερήσεις των σημάτων στο λογικό κύκλωμα. Επειδή πολλές φορές, τα σήματα των διαύλων είναι αναγκαίο να φτάνουν στους αντίστοιχους προορισμούς τους περίπου στην ίδια ώρα υιοθετείται μια ιδέα σύμφωνα με την οποία όλες οι λογικές πύλες που έχουν άμεση σχέση μεταξύ τους διασυνδέονται με wires προκαθορισμένου μήκους *l*. Η προσέγγιση αυτή έχει επιτυχία σε συγκεκριμένες περιπτώσεις και μια από αυτές είναι και το channel routing, όπου και παρουσιάζεται στη συγκεκριμένη εργασία.

Οι Degughi, Koide και Wakabayashi [8] προτείνουν έναν αλγόριθμο διασύνδεσης, ο οποίος χρησιμοποιεί πολλαπλά επίπεδα χρησιμοποιώντας χρονικούς περιορισμούς. Σύμφωνα με αυτόν, η περιοχή του chip χωρίζεται σε υποπεριοχές που ονομάζονται blocks. Αρχικά, η περιοχή αποτελείται από 4x4 blocks. Κάθε block διασπάται σε 4x4 σε κάθε επίπεδο. Στη συνέχεια, διασυνδέονται οι πύλες μεταξύ τους σε κάθε block επιλύνοντας το αντίστοιχο πρόβλημα ακεραίου προγραμματισμού (integer programming), στο οποίο κάθε πιθανή διασύνδεση αναπαριστάται ως ένα πρότυπο, που εξαρτάται από χρονικούς περιορισμούς. Ακόλουθα, ξεκινώντας από το πρώτο επίπεδο, συνδέουμε τις πύλες που βρίσκονται σε κάθε υποπεριοχή μεταξύ τους μέχρι να φτάσουμε στο τελευταίο.

Μια ακόμα εργασία σχετική με τη διασύνδεση των λογικών πυλών είναι και η [9]. Σε αυτήν παρουσιάζεται μια ευριστική συνάρτηση σε non-Manhattan Steiner δέντρα, η οποία καταφέρνει να μειώσει το μέγεθος των wires περίπου σε 17% κατά μέσο όρο

συγκρινόμενη με ευθύγραμμες τοπολογίες (rectilinear topologies). Επιπρόσθετα, παρατίθεται ένας αλγόριθμος βελτιστοποίησης των διασυνδέσεων, ο οποίος είναι βασισμένος σε γράφους (graph based) και ονομάζεται GRATS-tree, πετυχαίνοντας παραπέρα βελτιστοποίηση. Τέλος, το μειονέκτημα της προσέγγισης αυτής είναι ότι απαιτεί περισσότερα σημεία Steiner, περισσότερες στροφές των wires (bends) και παραπάνω vias.

Συνάμα, στο [10] συνδυάζονται το διαγώνιο μοντέλο με το Manhattan κατά τη διάρκεια της διασύνδεσης. Θεωρούνται δύο επίπεδα διασύνδεσης όπου για παράδειγμα στο πρώτο οι γραμμές διασύνδεσης μπορούν να είναι οριζόντιες ή διαγώνιες (να έχουν δηλαδή κλίση ± 45^o) και στο άλλο να είναι κάθετες. Επίσης, παρουσιάζονται αλγόριθμοι βασισμένοι σε αυτή τη μέθοδο σε συγκεκριμένες περιπτώσεις όπως είναι τα channels και τα switchboxes.

Ακόμα, πολλοί είναι αυτοί που έχουν ασχοληθεί με τη χωροθέτηση (floorplanning) ομάδων από λογικές πύλες πάνω στην επιφάνεια του chip. Έτσι λοιπόν, στο [11] αναφέρεται μια ευφυής δενδρική αναπαράσταση των ομάδων αυτών, που ονομάζεται slicing tree. Αυτή, μας δίνει το πλεονέκτημα να χειριστούμε με ευκολία το σχήμα και τον προσανατολισμό τον στοιχείων αυτών καθώς επίσης και τη δυνατότητα να τα συμπτύξουμε ώστε να ελαττώσουμε την επιφάνεια του chip.

Τέλος, στα [12], [13] και [14] παρουσιάζονται διάφοροι τρόποι σύμφωνα με τους οποίους επιτυγχάνεται μια καλή χωροθέτηση, η οποία αφενός μεν θα προσπαθεί να μειώσει το μέγεθος του chip και αφετέρου δε να μειώσει το μήκος των μεταξύ τους συνδέσεων. Και τρεις παραπάνω εργασίες αναφέρουν μεθόδους βασισμένες σε simulated annealing. Εμείς στην παρούσα εργασία αναφερθήκαμε με ξεχωριστό τρόπο στο συγκεκριμένο ζήτημα.

1.4. Δομή της Εργασίας

Η συγκεκριμένη εργασία είναι δομημένη ως εξής. Στο Κεφάλαιο 2 αναφέρονται ορισμένες βασικές έννοιες από τη θεωρία των VLSI κυκλωμάτων και από τη θεωρία των γράφων ώστε να γίνει καλύτερα αντιληπτή η συνέχεια. Στο Κεφάλαιο 3

παρουσιάζουμε τη μέθοδο που ακολουθήσαμε τόσο για την τοποθέτηση των λογικών πυλών στο κύκλωμα όσο και την χωροθέτηση αυτών. Ακολούθως, στο Κεφάλαιο 4 παραθέτουμε τον αλγόριθμο διασύνδεσης που εφαρμόσαμε. Στο επόμενο Κεφάλαιο 5 παρουσιάζουμε τα αποτελέσματα της μελέτης μας, εξάγουμε ορισμένα συμπεράσματα από τη μέθοδο μας και εξηγούμε κάποια γενικά θέματα υλοποίησης. Τέλος, στο Κεφάλαιο 6 αναφερόμαστε σε μελλοντική δουλειά που πιθανόν μπορεί να γίνει για τη βελτίωσή της.

ΚΕΦΑΛΑΙΟ 2. ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ

Στο κεφάλαιο αυτό θα εξηγήσουμε ορισμένες βασικές έννοιες της σχεδίασης ολοκληρωμένων κυκλωμάτων και της θεωρίας των γραφών, τις οποίες θα συναντήσουμε παρακάτω με σκοπό να γίνει περισσότερο κατανοητή η ανάλυση της εργασίας μας.

2.1. Βασικά Στοιχεία Λογικών Κυκλωμάτων

Τα δομικά στοιχεία ενός ψηφιακού λογικού κυκλώματος είναι οι λογικές πύλες. Αποτελούνται από βασικά ηλεκτρονικά κυκλώματα (πχ τρανζίστορ, πυκνωτές κλπ) και έχουν μια ή περισσότερες εισόδους, αλλά μόνο μια έξοδο. Οι τιμές εισόδου και εξόδου είναι 0 ή 1 («true» ή «false» αντίστοιχα). Η τιμή της εξόδου εξαρτάται μόνο από τις τιμές των εισόδων γεγονός που σημαίνει ότι οι πύλες δεν έχουν μνήμη. Η συμπεριφορά ή η λειτουργία μιας πύλης περιγράφεται από ένα πίνακα που ονομάζεται πίνακας αλήθειας.

Ο πίνακας αλήθειας περιγράφει την συμπεριφορά οποιασδήποτε λογικής συνάρτησης και σχηματίζεται από τον συνδυασμό όλων των δυνατών τιμών των εισόδων οι οποίες οδηγούν σε ένα συνδυασμό τιμών εξόδου. Για παράδειγμα ο Πίνακας 2.1 περιγράφει την συμπεριφορά ενός λογικού κυκλώματος με τρεις εισόδου (x,y,z) και δύο εξόδους (a,b)

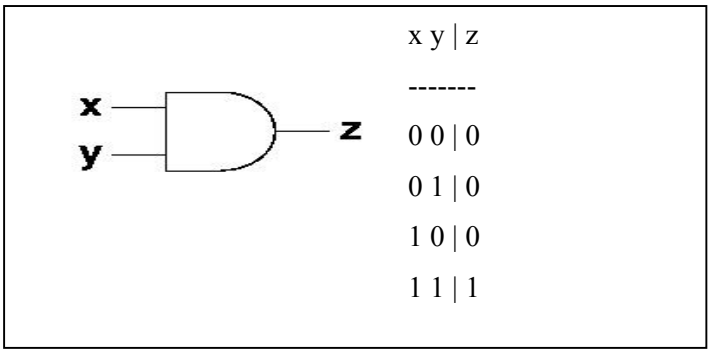
Πίνακας 2.1 Πίνακας Αλήθεια ενός Λογικού Κυκλώματος με Τρεις Εισόδους και Δύο Εξόδους

ΕΙΣΟΔΟΙ			**ΕΞΟΔΟΙ**	
w	**x**	**y**	**a**	**b**
0	0	0	0	1
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

Οι βασικές πύλες που συναντάμε στα ολοκληρωμένα κυκλώματα είναι οι παρακάτω:

- Η πύλη AND

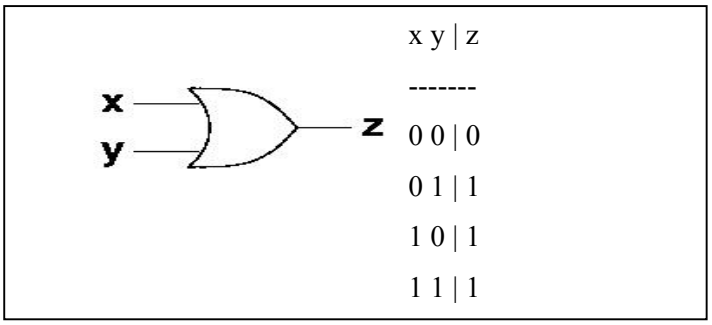
  Η πύλη αυτή μπορεί να έχει πολλές εισόδους αλλά μία έξοδο. Η έξοδος έχει την τιμή 1, αν και μόνον αν, όλες οι είσοδοι έχουν την τιμή 1. Διαφορετικά η τιμή της είναι μηδέν. Στο Σχήμα 2.1 φαίνεται η αναπαράσταση μιας πύλης AND και ο πίνακας αληθείας αυτής.



Σχήμα 2.1 Γραφική Αναπαράσταση μιας Πύλης AND και ο Πίνακας Αληθείας

- Η πύλη OR

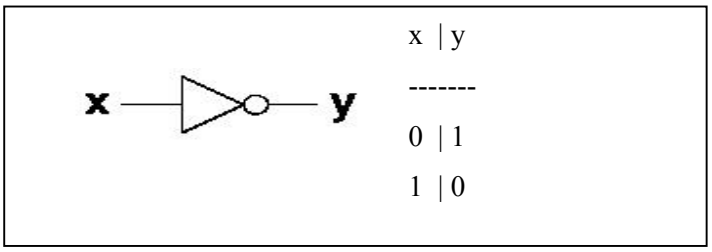
  Η πύλη αυτή μπορεί να έχει πολλές εισόδους αλλά μία έξοδο. Η έξοδος έχει την τιμή 1 αν τουλάχιστον μια από τις εισόδους έχει την τιμή 1, διαφορετικά έχει την τιμή 0. Στο Σχήμα 2.2 παρουσιάζεται η αναπαράσταση μιας πύλης OR και ο πίνακας αληθείας αυτής.



Σχήμα 2.2 Γραφική Αναπαράσταση μιας Πύλης OR και ο Πίνακας Αληθείας

- Η πύλη NOT

  Η πύλη NOT ή πύλη αντιστροφής όπως συνηθίζεται να λέγεται έχει μια είσοδο και μια έξοδο. Η τιμή εξόδου είναι 1 αν η τιμή εισόδου είναι 0 και το αντίστροφο. Στο Σχήμα 2.3 φαίνεται η αναπαράσταση μιας πύλης NOT και ο πίνακας αληθείας αυτής

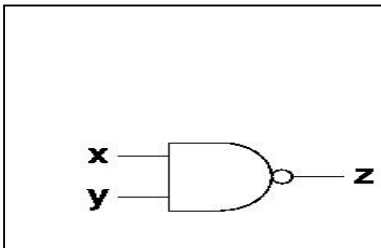


Σχήμα 2.3 Γραφική Αναπαράσταση μιας Πύλης NOT και ο Πίνακας Αληθείας

Εκτός από τις βασικές πύλες στα λογικά κυκλώματα χρησιμοποιούνται κατά κόρον και σύνθετες πύλες, οι οποίες είναι συνδυασμοί των παραπάνω και σκοπό έχουν την απλοποίηση των διαγραμμάτων αυτών.

- Η πύλη NAND

  Η πύλη αυτή κατασκευάζεται από μια AND και μια NOT. Μπορεί να έχει και αυτή πολλές εισόδους αλλά μόνο μια έξοδο. Η αναπαράσταση της και ο πίνακας αληθείας της φαίνονται στο Σχήμα 2.4

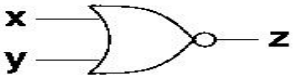


x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

Σχήμα 2.4 Γραφική Αναπαράσταση μιας Πύλης NAND και ο Πίνακας Αληθείας

- Η πύλη NOR

  Η πύλη αυτή κατασκευάζεται από μια OR την οποία ακολουθεί μια NOT. Όπως και οι προηγούμενες μπορεί να έχει πολλές εισόδους και μια μόνο έξοδο. Το Σχήμα 2.5 παρουσιάζει αυτή και τον πίνακα αληθείας της.



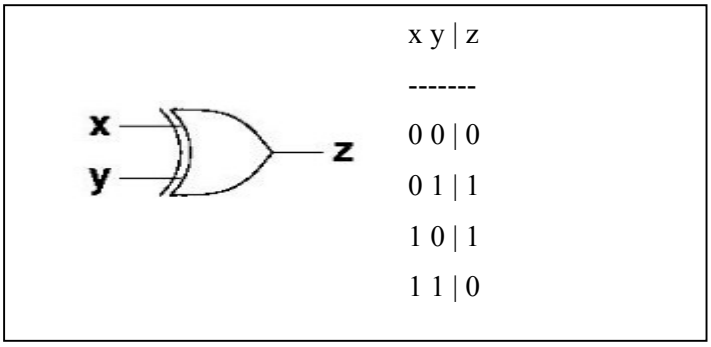
x	y	z
0	0	1
0	1	0
1	0	0
1	1	0

Σχήμα 2.5 Γραφική Αναπαράσταση μιας Πύλης NOR και ο Πίνακας Αληθείας

- Η πύλη XOR

  Το ολοκληρωμένο όνομα αυτής είναι eXclusive OR και μπορεί να έχει περισσότερες από μια εισόδους, ενώ η μοναδική έξοδος της παίρνει την τιμή

ένα αν και μόνο αν υπάρχει μόνο ένας άσσος στις εισόδους της πύλης, διαφορετικά παίρνει την τιμή μηδέν.



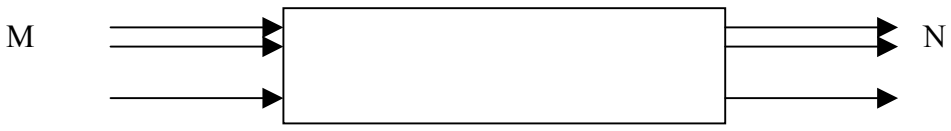
Σχήμα 2.6 Γραφική Αναπαράσταση μιας Πύλης XOR και ο Πίνακας Αληθείας

2.2. Κατηγορίες Λογικών Κυκλωμάτων

Τα λογικά κυκλώματα τα χωρίζουμε σε δύο κατηγορίες: τα συνδυαστικά και τα ακολουθιακά.

2.2.1. Συνδυαστικά Κυκλώματα

Τα συνδυαστικά λογικά κυκλώματα αποτελούνται από ένα συνδυασμό από πύλες και είναι τα κυκλώματα που η έξοδος τους (ή οι έξοδοι) εξαρτάται μόνο από την λογική κατάσταση της εισόδου (ή των εισόδων) του. Στο Σχήμα 2.7 περιγράφουμε ένα γενικευμένο συνδυαστικό κύκλωμα το οποίο αποτελείται από M εισόδους και N εξόδους



Σχήμα 2.7 Γενικευμένο Συνδυαστικό Κύκλωμα

Ένα συνδυαστικό κύκλωμα είναι μια γενικευμένη πύλη με M εισόδους και N εξόδους. Δεν έχει στοιχεία μνήμης, δηλαδή οι τιμές στις εξόδους εξαρτώνται

αποκλειστικά από τις τιμές των εισόδων. Για την κατασκευή ενός συνδυαστικού κυκλώματος που εκτελεί μια επιθυμητή λειτουργία πρέπει να συνδυάσουμε έναν αριθμό από πύλες. Στόχος του κατασκευαστή είναι να πετύχει τον σκοπό του με τον ελάχιστό αριθμό πυλών. Ο ελάχιστος αριθμός πυλών μας εξασφαλίζει μεγαλύτερη ταχύτητα και μικρότερη κατανάλωση ηλεκτρικής ισχύος. Η ελαχιστοποίηση είναι ένα δύσκολο πρόβλημα και επιτυγχάνεται με την βοήθεια ευριστικών αλγορίθμων.

2.2.2. Ακολουθιακά Κυκλώματα

Στα ακολουθιακά κυκλώματα η έξοδος (οι έξοδοι) είναι συνάρτηση της εισόδου (των εισόδων) και της κατάστασης του κυκλώματος πριν από την χρονική στιγμή της εφαρμογής της εισόδου. Για το λόγο αυτό διαθέτουν στοιχεία μνήμης στα οποία αποθηκεύεται κάθε φορά η κατάσταση του κυκλώματος. Στο Σχήμα 2.8 παρουσιάζουμε ένα ακολουθιακό κύκλωμα με τρεις εισόδους X1, X2, X3 , δύο εξόδους Z1, Z2 και ένα στοιχείο μνήμης.

Σχήμα 2.8 Γενικευμένο Ακολουθιακό Κύκλωμα

Αν οι ενέργειες ενός ακολουθιακού κυκλώματος συντονίζονται από ένα ρολόι (Clock), το οποίο στέλνει τετραγωνικούς παλμούς με σταθερή συχνότητα, τότε τα κυκλώματα αυτά ονομάζονται σύγχρονα. Αν οι ενέργειες γίνονται σε χρονικές στιγμές που καθορίζει το ίδιο το κύκλωμα τότε λέγονται ασύγχρονα.

Όπως οι πύλες αποτελούν τα βασικά δομικά υλικά των συνδυαστικών κυκλωμάτων έτσι τα latches και τα flip-flops αποτελούν τα βασικά δομικά υλικά των ακολουθιακών κυκλωμάτων. Οι πύλες κατασκευάζονται από τρανζίστορ ενώ τα

latches κατασκευάζονται από πύλες και τα flip-flops από latches. Οι έξοδοι στα latches και τα flip-flops εξαρτώνται από όλες τις προηγούμενες τιμές των εισόδων τους και όχι μόνο από την τελευταία , όπως στις πύλες. Τα flip-flop χρονίζονται από το σήμα ενός ρολογιού ενός τα latches όχι [16].

2.3. Σχεδίαση Λογικών Κυκλωμάτων

Στην προηγούμενη παράγραφο εξηγήσαμε τι είναι λογικό κύκλωμα, πόσα είδη υπάρχουν και ποια είναι τα δομικά στοιχεία αυτών. Στο σημείο αυτό θα δούμε τη διαδικασία που ακολουθείται για το σχεδιασμό τους από την αρχή μέχρι να βγουν στην παραγωγή και στην αγορά. Τα κυκλώματα αυτά κατασκευάζονται συνήθως για συγκεκριμένους σκοπούς και για το λόγο αυτό ονομάζονται *ASIC* (Application Specific Integrated Circuits). Η σχεδίαση αυτών χωρίζεται σε τρεις κατηγορίες

- *Full Custom*, στην οποία ο σχεδιαστής πρέπει να σχεδιάσει όλο το κύκλωμα από την αρχή
- *Semi Custom*, στην οποία υπάρχουν έτοιμες κάποιες λογικές μονάδες που ο σχεδιαστής μπορεί να χρησιμοποιήσει. Στην κατηγορία αυτή ανήκουν τα gate-arrays και τα standard-cells.
- *Programmable,* στην οποία τα chips είναι έτοιμα και το μόνο που χρειάζεται είναι ο προγραμματισμός αυτών.

Πριν προχωρήσουμε στην ανάλυση του σχεδιασμού των λογικών κυκλωμάτων θα αναφέρουμε ορισμένα στοιχεία για τα Standard Cells, αφού αυτή είναι η τεχνική που χρησιμοποιήσαμε στην εργασία μας. Το Standard Cell είναι ένα σύνολο από τρανζίστορ και δομών διασύνδεσης που είτε εκτελεί λογικές συναρτήσεις όπως είναι η AND, OR, NOR κτλ είτε χρησιμοποιείται ως δομή αποθήκευσης όπως flip-flops και latches. Συνήθως, ο αρχικός σχεδιασμός αυτών επιτυγχάνεται σε επίπεδο τρανζίστορ με τη μορφή τρανζίστορ netlist. Αυτή είναι μια ιδιαίτερα σημαντική αναπαράσταση των τρανζίστορ, των μεταξύ τους συνδέσεων και των τερματικών που τα συνδέουν με το εξωτερικό περιβάλλον. Παρά το γεγονός, ότι ένα Standard Cell είναι επαρκές για να αναπαραστήσει μια λογική συνάρτηση, σήμερα ο μοντέρνος σχεδιασμός των ASIC έχει επιβάλλει τη δημιουργία βιβλιοθηκών από αυτά. Κάθε

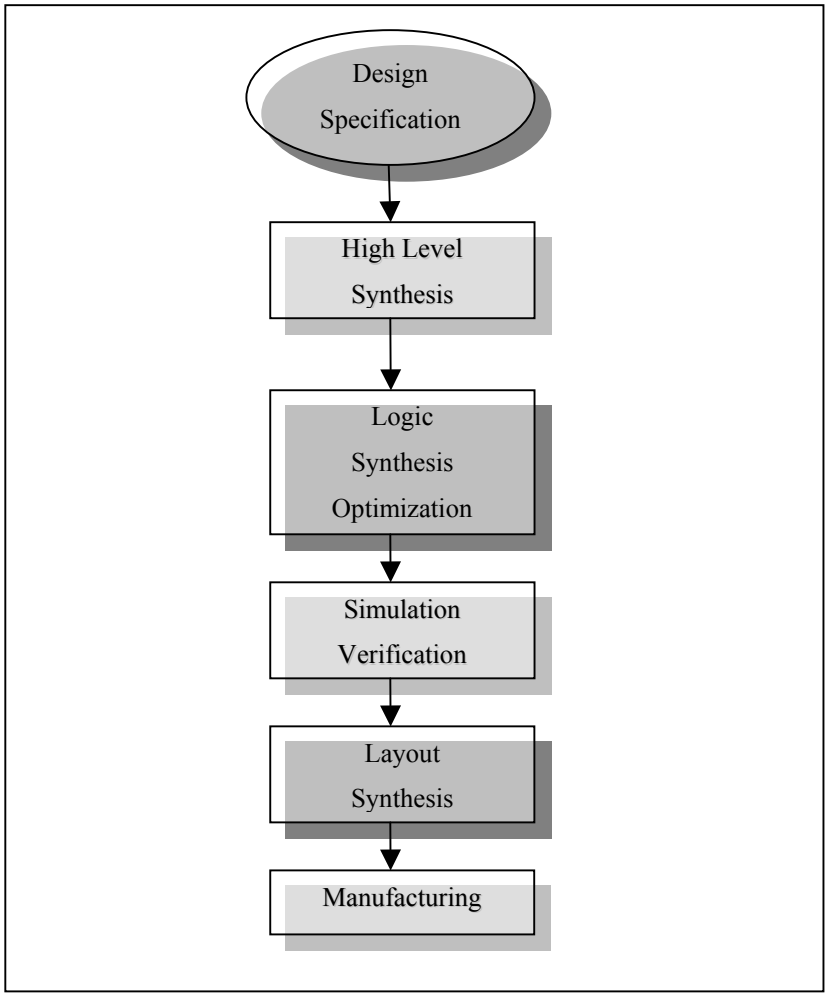
βιβλιοθήκη υλοποιεί το ίδιο Standard Cell με διαφορετικό τρόπο ως προς την ταχύτητα και το μέγεθος του. Έτσι λοιπόν, ο σχεδιαστής μπορεί να επιλέξει τη βιβλιοθήκη που ταιριάζει καλύτερα στη δουλεία του. Τελειώνοντας, τα δύο βασικά πλεονεκτήματα του Semi Custom σχεδιασμού με τη βοήθεια των Standard Cells είναι ότι

1. Μειώνεται πολύ ο χρόνος σχεδίασης του ολοκληρωμένου αφού ο σχεδιαστής δεν είναι ανάγκη να σχεδιάσει όλο το κύκλωμα από την αρχή.
2. Τα Standard Cells μπορούν να εγγυηθούν για τη λειτουργία τους, με αποτέλεσμα το ρίσκο σωστής λειτουργίας να είναι μικρό.

Επίσης, είναι χρήσιμο να αναφερθούν συνοπτικά ορισμένα στοιχεία για το σχεδιασμό ολοκληρωμένων κυκλωμάτων βασισμένο σε gate arrays. Στην περίπτωση αυτή, τα τρανζίστορ είναι προσχεδιασμένα και προκατασκευασμένα στο wafer με την μορφή σειρών ενός cell που επαναλαμβάνεται. Έτσι λοιπόν, όπως και παραπάνω, ο σχεδιαστής καλείται να διασυνδέσει μόνο τα τρανζίστορ μεταξύ τους σε ένα υψηλότερο επίπεδο, καθώς τα χαμηλότερα είναι προκατασκευασμένα. Αυτό έχει ως αποτέλεσμα ο χρόνος αλλά και το κόστος κατασκευής να μειώνεται αρκετά. Υπάρχουν τριών ειδών gate arrays

1. Channeled gate arrays: Εδώ αφήνουμε συγκεκριμένο σταθερό χώρο μεταξύ των γραμμών που είναι τοποθετημένα τα τρανζίστορ και μόνο σε αυτό μπορούν να τοποθετηθούν οι διασυνδέσεις.
2. Channeless gate arrays: Εδώ δεν υπάρχει χώρος μεταξύ των γραμμών που είναι τοποθετημένα τα τρανζίστορ και ο σχεδιαστής είναι να χρησιμοποιήσει περισσότερα επίπεδα μετάλλου για την υλοποίηση των διασυνδέσεων.
3. Structured gate arrays: Εδώ αφήνουμε ελεύθερη μια περιοχή του τσιπ στην οποία αναθέτουμε μια συγκεκριμένη λειτουργία, τοποθετώντας πάνω της κάποιο μπλοκ λογικής (π.χ μικροεπεξεργαστής), το οποίο όμως δεν είναι gate array [17].

Η διαδικασία σχεδιασμού (design flow) των ολοκληρωμένων κυκλωμάτων (ASIC) ανεξάρτητα σε ποια από τις παραπάνω κατηγορίες ανήκει τους φαίνεται στο Σχήμα 2.9



Σχήμα 2.9 Design Flow ASIC

Η σύνθεση υψηλού επιπέδου (high level synthesis) είναι συνήθως το πρώτο στάδιο στη σχεδίαση ενός ολοκληρωμένου για να αποφύγουμε με αυτόν τον τρόπο την ανάγκη της αναπαράστασης του κυκλώματος με τη βοήθεια χιλιάδων πυλών. Βασικός σκοπός της είναι να ορίσει βασικές συναρτήσεις που θα υλοποιηθούν σε αυτό καθώς επίσης και τα επιμέρους κομμάτια του κυκλώματος που τις υλοποιήσουν. Έτσι λοιπόν, για μια αφηρημένη περιγραφή ενός συστήματος η φάση αυτή μας δίνει ως έξοδο μια δομή που ονομάζεται register-transfer level (RTL), που πραγματοποιεί συγκεκριμένες συμπεριφορές του κυκλώματος. Αυτό επιτυγχάνεται με γλώσσες περιγραφής όπως είναι η VHDL.

Η λογική σύνθεση (logic synthesis) δέχεται ως είσοδο την RTL περιγραφή του κυκλώματος και τις βιβλιοθήκες που καθορίζουν τον τρόπο λειτουργίας και

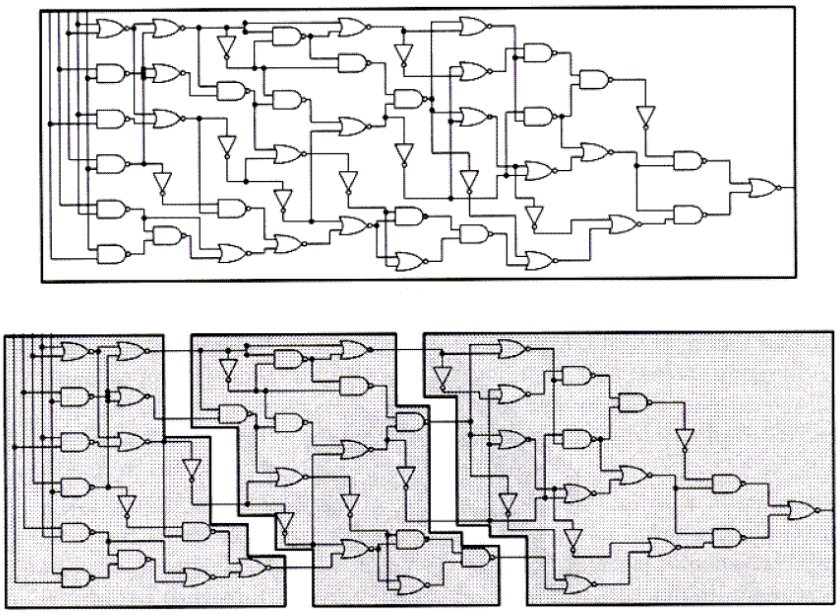
κατασκευής των λογικών μονάδων αυτού. Κάθε λογικό κομμάτι του κυκλώματος που περιγράφηκε στην προηγούμενη φάση μεταφέρεται τώρα σε μια δομή διασυνδεδεμένων λογικών κυττάρων με σκοπό είτε την ελαχιστοποίηση του μεγέθους του είτε την ελαχιστοποίηση της χρονικής καθυστέρησης του είτε την ελαχιστοποίηση και των δύο μαζί. Η βελτιστοποίηση (optimization) του κυκλώματος γίνεται συνήθως σε αυτό το στάδιο και αφορά παράγοντες όπως είναι η επιφάνεια, η ισχύς, η ταχύτητα κτλ.

Στη συνέχεια προσομοιώνεται (simulation) και επαληθεύεται (verification) η συμπεριφορά στοιχείων του κυκλώματος πάνω σε διάφορες λογικές συναρτήσεις

Η επόμενη φάση της διάταξης του κυκλώματος (layout synthesis) είναι ιδιαίτερα σημαντική. Στο σημείο αυτό τοποθετούνται τα λογικά κύτταρα στις κατάλληλες πραγματικές τους θέσεις πάνω στην επιφάνεια του chip και κατόπιν γίνεται οι πραγματική τους διασύνδεση. Έτσι λοιπόν, ενώ στα προηγούμενα στάδια οι διασυνδέσεις δεν ήταν πραγματικές αλλά λογικές αφού περιέγραφαν ποια στοιχεία του κυκλώματος συνδέονται μεταξύ τους, τώρα αυτές μας δίνουν πληροφορίες για την πραγματική τους διασύνδεση. Για να επιτευχθούν όλα αυτά το συγκεκριμένο στάδιο χωρίζεται στα εξής βήματα:

a) <u>Διαμέριση του συστήματος (System Partioning)</u>

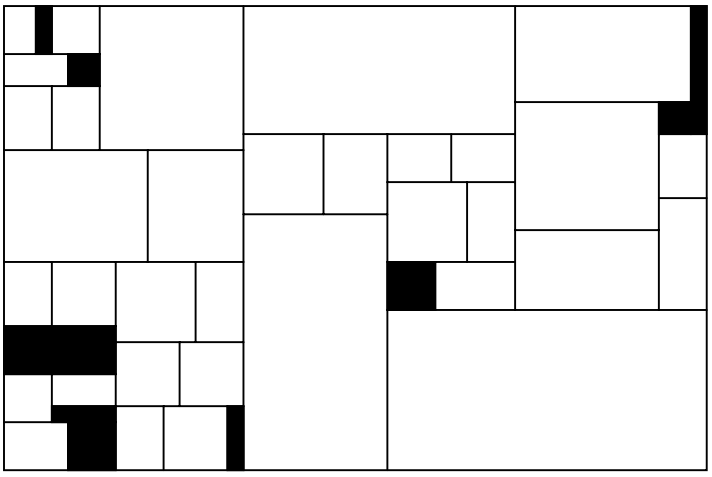
Με τον τρόπο αυτό το ολοκληρωμένο διαιρείται σε μικρότερα υποκυκλώματα τα οποία ονομάζονται blocks. Αυτό συμβαίνει για να μικραίνουμε το μέγεθος του προβλήματός μας και να το αντιμετωπίσουμε έτσι αποτελεσματικότερα. Οι παράγοντες που λαμβάνουμε υπόψη στο βήμα αυτό είναι ο αριθμός των block, το μέγεθος τους και οι διασυνδέσεις μεταξύ τους. Πολλές φορές το στάδιο αυτό γίνεται από την αρχή οπότε το αρχικό μας σύστημα χωρίζεται σε υποσυστήματα καθένα από τα οποία θα τοποθετηθεί σε ξεχωριστό chip. Στο Σχήμα 2.10 φαίνεται το αρχικό κύκλωμα και η διαμέριση αυτού



Σχήμα 2.10 Αρχικό Κύκλωμα και η Διαμέριση του

b) Χωροθέτηση (Floor planning)

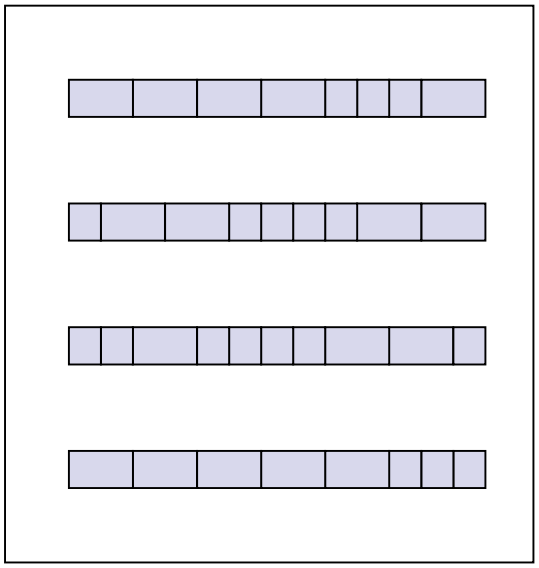
Στο βήμα αυτό τοποθετούνται το blocks που βρήκαμε από παραπάνω στην επιφάνεια του chip. Τη διαδικασία αυτή θα μπορούσαμε να την παρομοιάσουμε με την τοποθέτηση επίπλων σε ένα διαμέρισμα. Αυτό που ενδιαφέρει το σχεδιαστεί στο σημείο αυτό είναι τα blocks που συνδέονται μεταξύ τους να τοποθετηθούν κοντά το ένα στο άλλο και τα blocks που συνδέονται με το εξωτερικό περιβάλλον να μπούνε στην περιφέρεια του chip. Στο Σχήμα 2.11 φαίνεται μια τυπική χωροθέτηση όπου τα τετράγωνα με μαύρο χρώμα είναι ο χώρος που δεν έχει χρησιμοποιηθεί (dead space)



Σχήμα 2.11 Χωροθέτηση ενός Κυκλώματος. Κάθε Τετράγωνο Είναι και Ένα block. Τα Μαύρα Τετράγωνα Αποτελούν το Dead Space.

c) Τοποθέτηση (Placement)

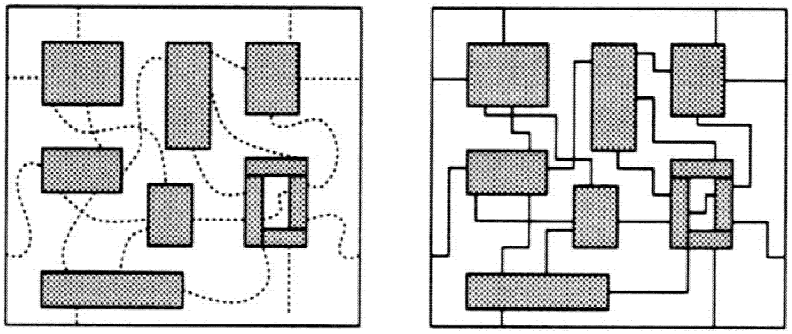
Στο βήμα αυτό καθορίζεται η ακριβής φυσική θέση των λογικών κυττάρων μέσα στα blocks όπου ανήκουν και κατ' επέκταση πάνω στην επιφάνεια του chip. Στην περίπτωση των Standard Cell, την οποία και χρησιμοποιήσαμε μπαίνουν πάνω σε διατάξεις οριζόντιων γραμμών. Όπως και στην προηγούμενη περίπτωση έτσι και εδώ σκοπός είναι λογικά κύτταρα που συνδέονται μεταξύ τους να μπουν κοντά το ένα στο άλλο ώστε να ελαχιστοποιηθεί το μήκος των διασυνδέσεων. Στο Σχήμα 2.12 φαίνεται μια τοποθέτηση Standard Cells, τα οποία αναπαραστώνται με ορθογώνια. Το διαφορετικό μέγεθος του δηλώνει και το τύπο τους.



Σχήμα 2.12 Τοποθέτηση Standard Cells σε Οριζόντια Διάταξη Γραμμών

d) Διασύνδεση (Routing)

Είναι το τελευταίο βήμα της παρούσας φάσης και σε αυτό επιχειρείται τόσο η διασύνδεση των στοιχείων του κυκλώματος μεταξύ τους όσο και των blocks του κυκλώματος. Συνήθως, χωρίζεται σε 2 κομμάτια. Στο πρώτο όπου υπολογίζεται από πού θα περάσει η κάθε διασύνδεση και το δεύτερο όπου η καθορίζεται η ακριβής της θέση. Σε μια καλή διασύνδεση το μήκος και η επιφάνεια των διασυνδέσεων είναι μικρά. Στο Σχήμα 2.13 παρουσιάζονται τα 2 στάδια τις διασύνδεσης μεταξύ block του κυκλώματος [17].



Σχήμα 2.13 Αριστερά: Πρώτη Φάση της Διασύνδεσης Δεξιά: Δεύτερη Φάση της Διασύνδεσης

Στην τελευταία φάση (manufacture) το chip προγραμματίζεται και βγαίνει στην αγορά προς πώληση

2.4. Βασικά Στοιχεία Αλγορίθμων σε Γράφους

Στο κεφάλαιο αυτό όπως προαναφέρθηκε θα αναφέρουμε ορισμένες από τις βασικές έννοιες των λογικών κυκλωμάτων και της θεωρίας των γράφων. Το πρώτο αναλύθηκε εκτενώς παραπάνω, ενώ το τελευταίο θα το αναλύσουμε στη συνέχεια. Η ανάλυση αυτού θεωρείται αναγκαία εξαιτίας του ότι στην εργασία μας αναπαραστήσαμε τα κυκλώματα με γράφους και δουλέψαμε με βασικές λειτουργίες αυτών.

2.4.1. Αναπαράσταση Γράφων

Δύο είναι οι τρόποι που χρησιμοποιούνται συνήθως για την αναπαράσταση των γράφων. Ο πρώτος είναι ως λίστα γειτνίασης και ο δεύτερος ως πίνακας γειτνίασης. Και οι δύο είναι κατάλληλοι για αναπαράσταση τόσο κατευθυνόμενων όσο και μη-κατευθυνόμενων γράφων.

Η αναπαράσταση με λίστα γειτνίασης (adjacency list representation) ενός γράφου G=(V, E), όπου V το σύνολο των κόμβων του γράφου και E το σύνολο των ακμών του, αποτελείται από έναν πίνακα Adj που έχει |V| λίστες, μια για κάθε κόμβο που ανήκει στο V. Η λίστα γειτνίασης Adj[u] περιέχει όλους τους κόμβους υ, τέτοιους ώστε υπάρχει ακμή (u,υ)∈ E. Αυτό σημαίνει ότι η λίστα Adj[u] αποτελείται από

όλους τους κόμβους που είναι γειτονικοί του u στον G. Οι κόμβοι σε κάθε λίστα γειτνίασης αποθηκεύονται συνήθως με τυχαίο τρόπο. Εάν ο γράφος είναι κατευθυνόμενος, τότε το συνολικό μήκος των λιστών γειτνίασης είναι ίσο με |E|, αφού για κάθε ακμή της μορφής (u, υ) το υ ανήκει στην Adj[u]. Από την άλλη πλευρά αν ο G είναι μη κατευθυνόμενος τότε το συνολικό μήκος των λιστών γειτνίασης είναι ίσο με 2*|E|, αφού για κάθε ακμή (u,υ) τόσο το υ βρίσκεται στην Adj[u] όσο και το αντίστροφο. Πάντως και για τις δύο περιπτώσεις, το μεγάλο πλεονέκτημα της αναπαράστασης αυτής είναι ότι απαιτεί Θ(V+E) μνήμη.

Επίσης, οι λίστες γειτνίασης είναι κατάλληλες και για την αναπαράσταση γράφων που οι ακμές τους έχουν βάρη (weighted graphs). Σε αυτή την περίπτωση σε κάθε ακμή του γράφου αναθέτουμε και ένα βάρος w, το οποίο προκύπτει από μια συνάρτηση βάρους w:E→R. Για παράδειγμα, έστω ένας γράφος G=(V, E) και w μια συνάρτηση βάρους αυτού. Τότε το βάρος w(u,υ) μιας ακμής (u, υ) αποθηκεύεται μαζί με τον κόμβο υ στη λίστα γειτνίασης.

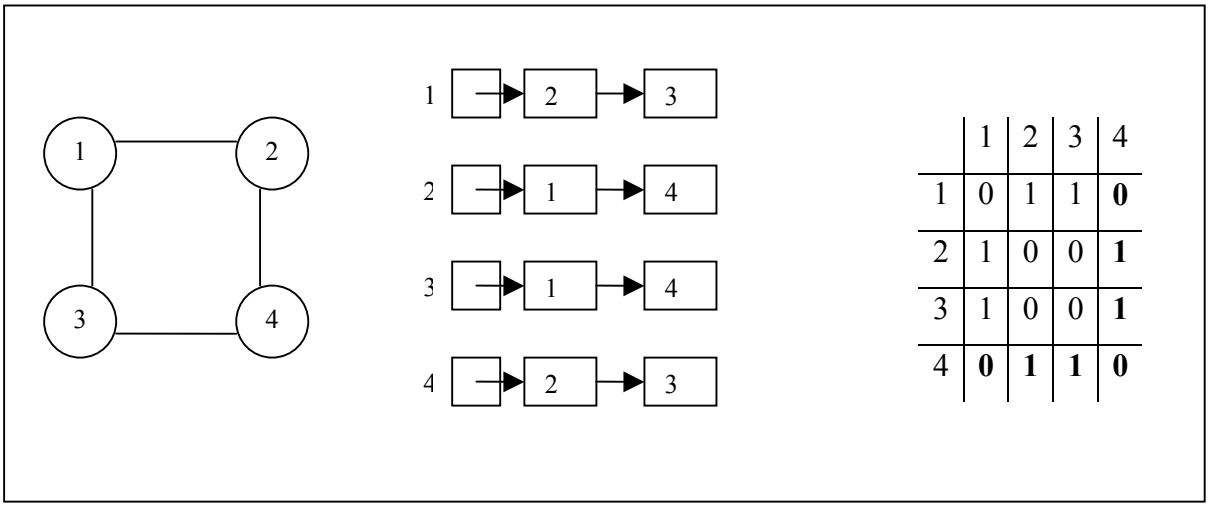
Το μεγάλο μειονέκτημα της αναπαράστασης αυτής είναι ο χρόνος που χρειαζόμαστε για να βρούμε μια ακμή (u, υ), αφού χρειάζεται να διασχίσουμε την Adj[u]. Το πρόβλημα αυτό μας το λύνει η αναπαράσταση με τον πίνακα γειτνίασης.

Η αναπαράσταση με πίνακα γειτνίασης (adjacency matrix representation) ενός γράφου G=(V, E), όπου V είναι το σύνολο των κόμβων και E το σύνολο των ακμών, θεωρεί ότι οι κόμβοι είναι αριθμημένοι ως 1,2,…|V| κατά ένα τυχαίο τρόπο. Τότε ο πίνακας γειτνίασης του G είναι ένας |V|x|V| πίνακας $A = (a_{ij})$ τέτοια ώστε

$$a_{ij} = \begin{cases} 1 \text{ , αν } (i,j) \in E \\ 0, \text{ διαφορετικά} \end{cases}$$

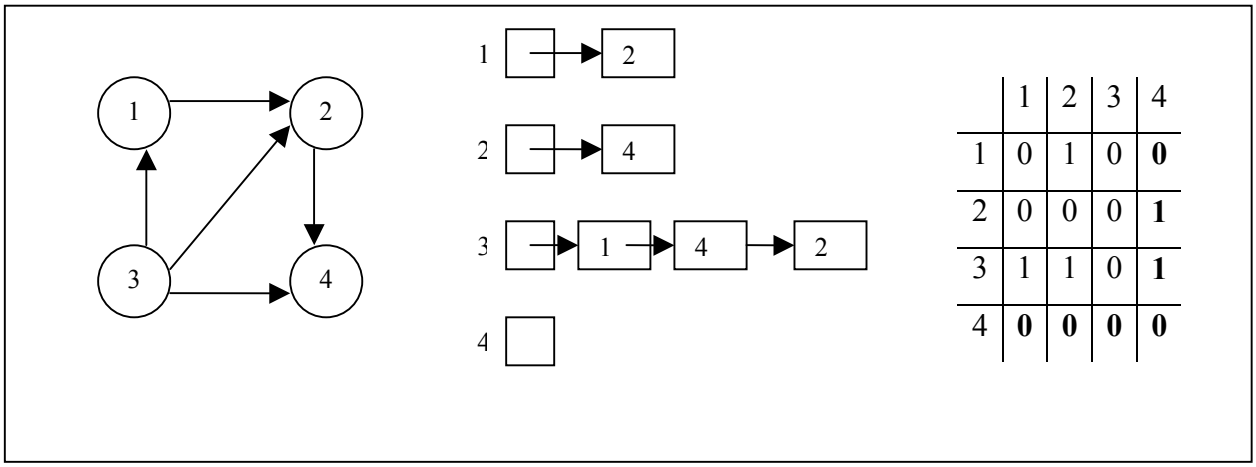
Δε θα αναφέρουμε περισσότερα πράγματα για την περίπτωση αυτή, επειδή επιλέξαμε την αναπαράσταση με λίστα γειτνίασης στην εργασία μας. Στο Σχήμα 2.14 φαίνεται ένας μη κατευθυνόμενος γράφος η αντίστοιχη αναπαράσταση με λίστα και πίνακα

γειτνίασης, ενώ στο Σχήμα 2.15 φαίνεται οι παραπάνω αναπαραστάσεις για να έναν κατευθυνόμενο γράφο.



	1	2	3	4
1	0	1	1	**0**
2	1	0	0	**1**
3	1	0	0	**1**
4	**0**	**1**	**1**	**0**

Σχήμα 2.14 Λίστα και Πίνακας Γειτνίασης Μη Κατευθυνόμενου Γράφου



	1	2	3	4
1	0	1	0	**0**
2	0	0	0	**1**
3	1	1	0	**1**
4	**0**	**0**	**0**	**0**

Σχήμα 2.15 Λίστα και Πίνακας Γειτνίασης Κατευθυνόμενου Γράφου

2.4.2. Διάσχιση Γράφων

Δύο είναι οι περισσότερο διαδεδομένοι τρόποι με τους οποίους μπορούμε να διασχίσουμε ένα γράφο. Ο πρώτος είναι ο λεγόμενος Breadth First Search (BFS) και ο δεύτερος είναι ο Depth First Search (DFS).

Breadth First Search (BFS)

Δεδομένου ενός γράφου G = (V,E) και ενός διακριτού κόμβου s, που ονομάζουμε source ο αλγόριθμος αυτός εξετάζει τις ακμές του G, για να ανακαλύψει όλους τους γειτονικούς κόμβους του s. Μόλις το πετύχει αυτό, ένας από τους γειτονικούς κόμβους επιλέγεται με τυχαίο τρόπο να είναι ο source και επαναλαμβάνεται η προηγούμενη διαδικασία μέχρι να ανακαλυφθούν όλοι ο κόμβοι. Ο χρόνος εκτέλεσης του αλγόριθμου είναι O(V+E). Αυτός προκύπτει ως εξής:

Κάθε κόμβος του γράφου εξετάζεται το πολύ μια φορά. Επειδή το πλήθος των κόμβων είναι V αυτό μας κοστίζει σε χρόνο O(V). Επίσης, για κάθε κόμβο εξετάζουμε τη λίστα γειτνίασης του το πολύ μια φορά. Το άθροισμα του μήκους όλων των λιστών γειτνίασης είναι Θ(E). Άρα λοιπόν το συνολικό κόστος εκτέλεσης είναι O(V+E).

Δεν θα γίνει μεγαλύτερη αναφορά στο συγκεκριμένο τρόπο αναζήτησης αφού εμείς επιλέξαμε τον DFS.

Depth First Search (DFS)

Στην περίπτωση αυτή εξετάζονται οι ακμές του κόμβου υ, ο οποίος έχει ανακαλυφθεί τελευταίος από τον αλγόριθμο και ενώνεται με ακμές που δεν έχουν εξεταστεί ακόμα. Όταν όλες οι ακμές του υ έχουν ανακαλυφθεί η αναζήτηση επιστρέφει (backtracks) για να εξετάσει τις ακμές του κόμβου μετά από τον οποίο ανακαλύφθηκε ο υ. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να ανακαλυφθούν όλοι οι κόμβοι που είναι προσιτοί από έναν αρχικό, ο οποίος επιλέγεται τυχαία και ονομάζεται source. Αν παραμένουν κόμβοι, που δεν έχουν εξεταστεί τότε επιλέγεται ένας από αυτός ως source και η διαδικασία επαναλαμβάνεται.

Στον DFS οι κόμβοι «χρωματίζονται» για να γίνεται αντιληπτή η κατάσταση τους κατά τη διάρκεια της αναζήτησης. Έτσι λοιπόν, αρχικά έχουν όλοι χρώμα άσπρο, ενώ όταν ανακαλύπτονται από την αναζήτηση γίνονται γκρι. Τέλος, όταν ολόκληρη η λίστα γειτνίασης του έχει ανακαλυφθεί «χρωματίζεται» μαύροι. Ο αλγόριθμος DFS για ένα γράφο G = (V, E) φαίνεται αναλυτικά στο Σχήμα 2.16

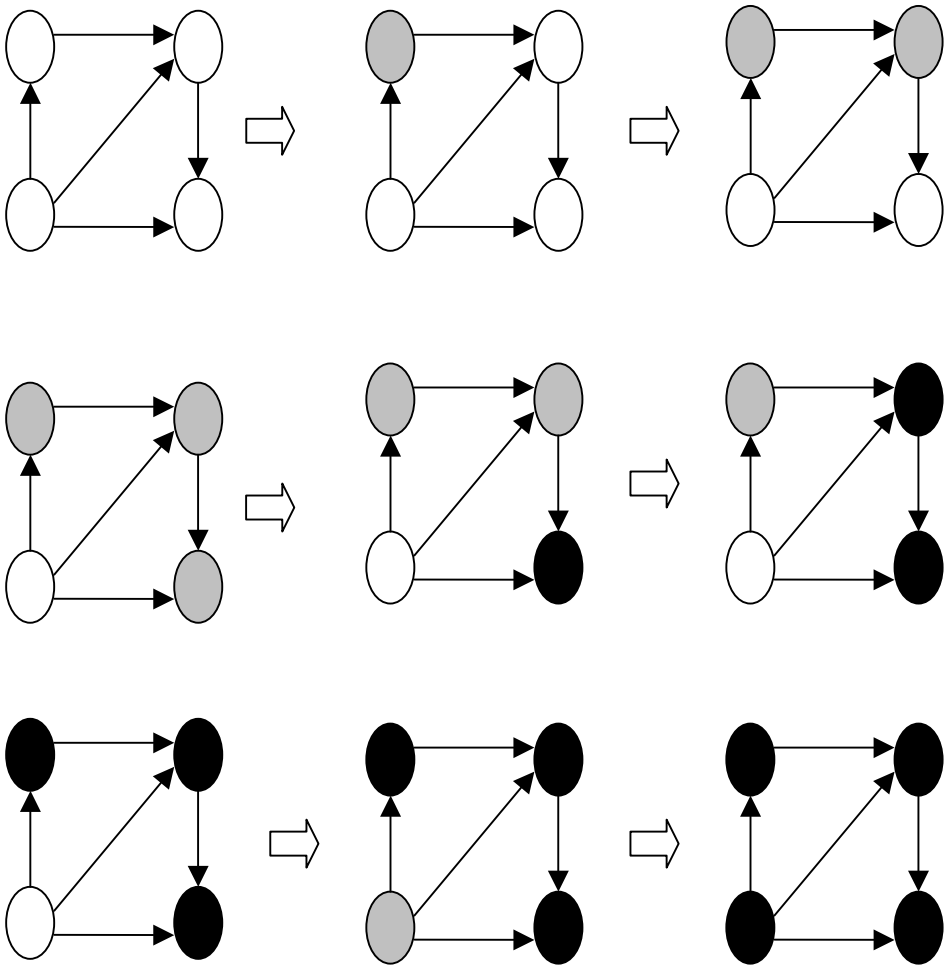
```
DFS(G)
1.  for each vertex u∈ V[G]
2.      do color[u] ← WHITE
3.  for each vertex u∈ V[G]
4.      do if color[u] = WHITE
5.          then DFS-VISIT(u)

DFS-VISIT(u)
6.  color[u] ← GRAY
7.  for each υ ∈ Adj[u]
8.      do if color[υ] = WHITE
9.          DFS-VISIT(υ)
10. color[u] = BLACK
```

Σχήμα 2.16 Αλγόριθμος DFS

Ο χρόνος εκτέλεσης του DFS είναι ίσος με Θ(V+E). Αυτός προκύπτεί αν λάβουμε υπό όψη ότι οι γραμμές 1-2 και 3-4 (Σχήμα 2.15) θα εκτελεστούν ακριβώς Θ(V) φορές αν αμελήσουμε το χρόνο εκτέλεσης της συνάρτησης DFS-VISIT. Πρέπει να σημειωθεί ότι αυτή εκτελείται ακριβώς μια φορά για κάθε κόμβο. Οι γραμμές 7-9 θα εκτελεστούν | Adj[u] | φορές. Συνεπώς το συνολικό κόστος της DFS-VISIT είναι το μήκος όλων των λιστών γειτνίασης | Adj[u] |, το οποίο είναι ίσο με Θ(E). Επομένως ο χρόνος εκτέλεσης του DFS είναι Θ(V+E).

Τέλος, στο Σχήμα 2.17 παρουσιάζεται μια εκτέλεση του DFS σε ένα κατευθυνόμενο γράφο. Κάθε γράφος παρουσιάζει και ένα βήμα στην εκτέλεση του αλγόριθμου. Αρχικά όλοι οι κόμβοι είναι άσπροι, ενώ όποιος εξετάζεται κάθε φορά χρωματίζεται γκρι. Όταν όλοι οι κόμβοι της λίστας γειτνίασης ενός κόμβου έχουν επισκεφθεί αυτός γίνεται μαύρος.



Σχήμα 2.17 Εκτέλεση Αλγόριθμου DFS

2.4.3. Δίκτυα Ροής.(Flow Networks)

Στην παράγραφο αυτή θα αναφέρουμε ορισμένα βασικά θεωρητικά στοιχεία των flow networks γιατί τα χρησιμοποιήσαμε σε μεγάλη κλίμακα στην εργασία μας.

Ένα δίκτυο ροής G=(V, E) είναι ένας κατευθυνόμενος γράφος με V το σύνολο των κόμβων του και E το σύνολο των ακμών του, στον οποίο κάθε ακμή (u, υ) ∈ E έχει μια μη αρνητική χωρητικότητα (capacity) c, δηλαδή c(u,υ) ≥ 0. Αν δεν υπάρχει μια ακμή τότε η χωρητικότητα αυτής είναι ίση με μηδέν. Επίσης, υπάρχουν δύο ξεχωριστοί κόμβοι, η πηγή (source) s και η καταβόθρα (sink) t. Κάθε κόμβος του γράφου βρίσκεται πάνω σε ένα μονοπάτι που ενώνει την πηγή με την καταβόθρα. Έτσι λοιπόν, για κάθε κόμβο υ ∈ V, υπάρχει ένα μονοπάτι της μορφής s → υ → v.

Από αυτό προκύπτει ότι το δίκτυο ροής είναι ένας συνδεδεμένος γράφος για τον οποίο ισχύει |E| ≥ |V| -1.

Στο σημείο αυτό θα ορίσουμε τη ροή (flow) ενός δικτύου ροής G=(V, E). Η ροή λοιπόν του G είναι μια πραγματική συνάρτηση f: VxV → **R** που ικανοποιεί τις τρεις παρακάτω συνθήκες.

1. Capacity Constraint: Για κάθε u, υ ∈ V ισχύει c(u, υ) ≥ f(u, υ)

   Η ιδιότητα αυτή μας λέει ότι η ροή από ένα κόμβο του γράφου σε έναν άλλο δε μπορεί να ξεπερνά τη δεδομένη χωρητικότητα.

2. Skew Symmetry: Για κάθε u, υ ∈ V ισχύει f(u, υ) = -f(υ, v)

   Η ιδιότητα αυτή μας εξηγεί πως η ροή από έναν κόμβο σε έναν άλλο γίνεται ίση με την αρνητική της όταν η κατεύθυνση αλλάζει

3. Flow conservation: Για κάθε u∈V – {s, t} ισχύει

   $$\sum_{\upsilon \in V} f(u,\upsilon) = 0$$

   Η ιδιότητα αυτή μας εξηγεί πως η συνολική ροή που φεύγει από ένα κόμβο εκτός του κουβά και της πηγής είναι ίση με 0. Αυτό συμβαίνει όταν η ροή μπορεί να πάρει και αρνητικές τιμές. Στην περίπτωση μας όμως η ροή παίρνει μόνο θετικές τιμές.

Έτσι λοιπόν η συνολική θετική ροή (total positive flow) που εισέρχεται σε έναν κόμβο ορίζεται από τη σχέση

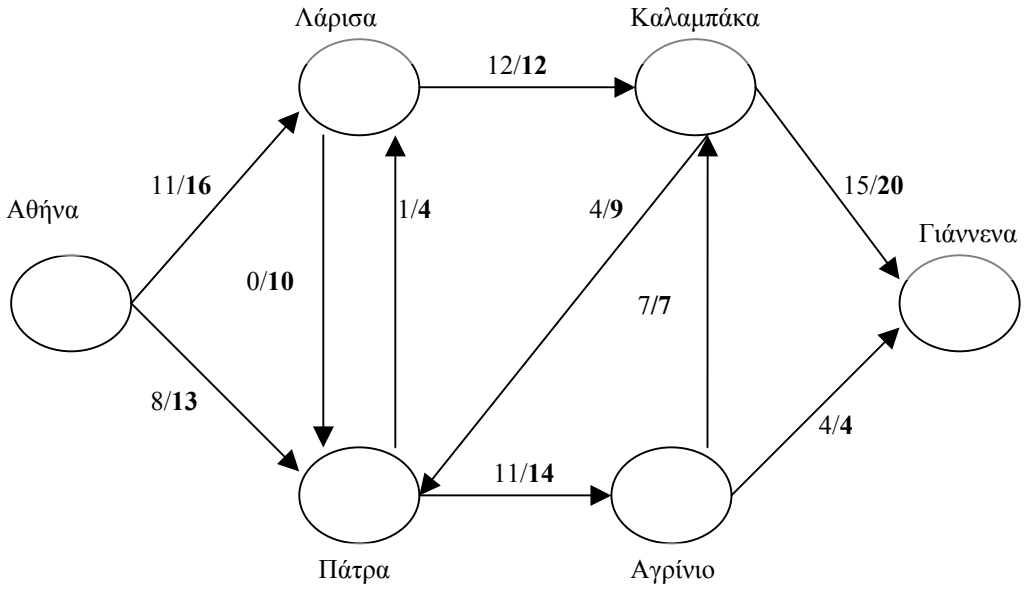
$$\sum_{\substack{u \in V \\ f(u,\upsilon) \geq 0}} f(u,\upsilon)$$

Η συνολική θετική ροή που εξέρχεται από έναν κόμβο ορίζεται συμμετρικά. Συνεπώς, μια άλλη ερμηνεία της ιδιότητας (3) είναι ότι η συνολική θετική ροή που εισέρχεται σε έναν κόμβο είναι ίση με την συνολική θετική ροή που εξέρχεται από αυτόν [18].

Τέλος, η τιμή της ροής f ενός δικτύου ροής ορίζεται από την ακόλουθη σχέση, όπου ο συμβολισμός | | δηλώνει απλά την τιμή της συνάρτηση και όχι απόλυτη τιμή ή πλήθος

$$| f | = \sum_{u \in V} f(s,u)$$

Για να γίνουν περισσότερα κατανοητά τα δίκτυα ροής παρουσιάζεται παρακάτω ένα παράδειγμα μεταφοράς φορτιών από την Αθήνα στα Γιάννενα (Σχήμα 2.18). Ο κόμβος πηγή είναι η Αθήνα και ο κόμβος καταβόθρα είναι τα Γιάννενα. Οι υπόλοιποι κόμβοι αναπαριστούν τους ενδιάμεσους σταθμούς. Κάθε ακμή έχει και μια χωρητικότητα που υποδηλώνει πόσα φορτία μπορούν να μεταφερθούν και είναι με έντονα μαύρα γράμματα. Επίσης, φαίνεται και η ροή του δικτύου με κανονικά γράμματα που στη συγκεκριμένη περίπτωση είναι ίση με 19, δηλαδή |f|=19.



Σχήμα 2.18 Δίκτυο Ροής

Περισσότερες πληροφορίες για πιο εξειδικευμένους αλγόριθμους δικτύων ροής θα αναφέρουμε στο επόμενο κεφάλαιο.

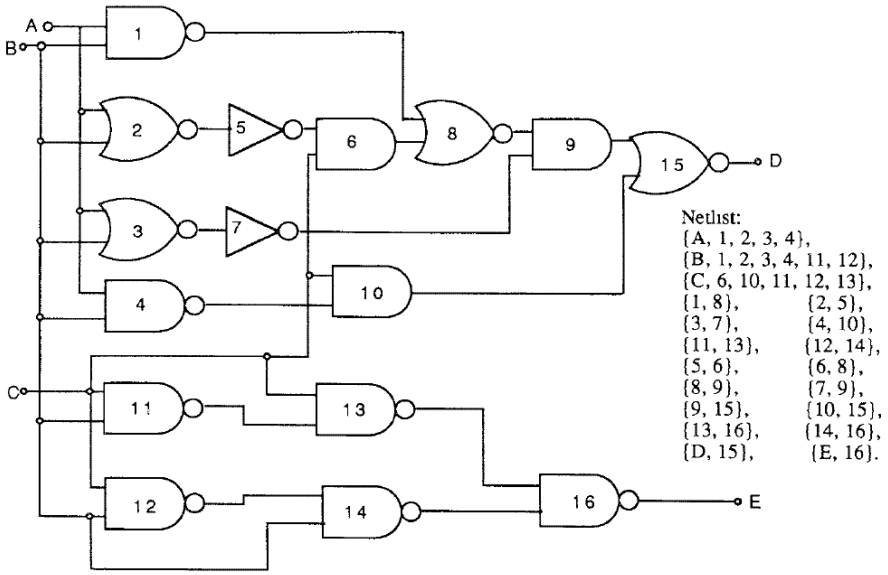
ΚΕΦΑΛΑΙΟ 3. ΤΟΠΟΘΕΤΗΣΗ ΛΟΓΙΚΩΝ ΚΥΤΤΑΡΩΝ ΤΟΥ ΚΥΚΛΩΜΑΤΟΣ

3.1. Εισαγωγή

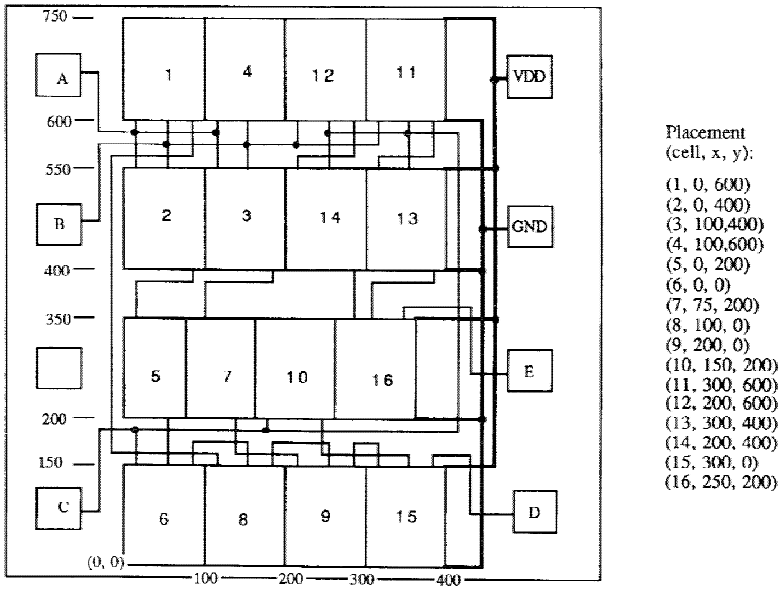
Στα προηγούμενα κεφάλαια είδαμε πως στα στάδια της φυσικής διασύνδεσης, η τοποθέτηση ακολουθεί το διαμερισμό του κυκλώματος και τη χωροθέτηση των επιμέρους κομματιών που προκύπτουν από αυτόν. Στην εργασία μας ακολουθήσαμε δύο τρόπους τοποθέτησης των λογικών κυττάρων του κυκλώματος, που στην περίπτωση μας είναι οι λογικές πύλες. Ο πρώτος τοποθετεί απευθείας τις πύλες πάνω στην επιφάνεια του chip, χωρίς να χωρίσει το κύκλωμα σε μικρότερα υποκυκλώματα και να τα χωροθετήσει. Αντίθετα, ο τελευταίος πρώτα διαμερίζει το κύκλωμα, στη συνέχεια κάνει μια ιδιαίτερη χωροθέτηση και τέλος τοποθετεί τις πύλες πάνω στο chip.

Πριν προχωρήσουμε στην ανάλυση των δύο παραπάνω μεθόδων είναι αναγκαίο να ορίσουμε συγκεκριμένα το πρόβλημα της τοποθέτησης. Δεδομένου λοιπόν, ενός ηλεκτρικού κυκλώματος, το οποίο αποτελείται από λογικά στοιχεία (modules) με ορισμένο αριθμό εισόδων και εξόδων και τον τρόπο διασύνδεσης μεταξύ τους, κατασκευάζουμε μια διάταξη αυτών που απεικονίζει τις ακριβείς τους θέσεις πάνω στην επιφάνεια του chip, έτσι ώστε το αναμενόμενο μήκος των διασυνδέσεων και η συνολική επιφάνεια να ελαχιστοποιούνται.

Η είσοδος του προβλήματος είναι η περιγραφή των λογικών στοιχείων, δηλαδή το σχήμα τους, το μέγεθός τους, η θέση των εισόδων και των εξόδων τους καθώς επίσης η netlist, που δείχνει τον τρόπο με τον οποίο αυτά διασυνδέονται. Η έξοδος είναι μια λίστα (x, y) συντεταγμένων για κάθε στοιχείο. Στο Σχήμα 3.1 φαίνεται η είσοδος και στο Σχήμα 3.2 η έξοδος του προβλήματος της τοποθέτησης.



Σχήμα 3.1 Η Netlist ενός Κυκλώματος ως Είσοδος στο Πρόβλημα της Τοποθέτησης



Σχήμα 3.2 Η Έξοδος της Τοποθέτησης ενός Κυκλώματος με τις Συντεταγμένες της Κάτω Αριστερής Γωνίας Κάθε Λογικού Στοιχείου

Αρχικά, όλοι οι αλγόριθμοι τοποθέτησης προσπαθούν να ελαχιστοποιήσουν τη συνολική επιφάνεια το chip για να μπορέσει αυτό να γίνει περισσότερο λειτουργικό. Επίσης, ένας ακόμα στόχος τους είναι να ελαχιστοποιήσουν το αναμενόμενο μήκος των διασυνδέσεων ώστε να μικραίνουν οι καθυστερήσεις του και να γίνει έτσι ταχύτερο. Τα δύο αυτά συμβάντα είναι άμεσα συνυφασμένα μεταξύ τους στην περίπτωση των Standard Cells με την οποία και ασχοληθήκαμε. Αυτό συμβαίνει γιατί η συνολική επιφάνεια του chip είναι περίπου ίση με την επιφάνεια των λογικών στοιχείων και την επιφάνεια που καταλαμβάνουν οι διασυνδέσεις.

Άλλα κριτήρια, λιγότερα σημαντικά που χρησιμοποιούνται σε μια σωστή τοποθέτηση είναι τα ακόλουθα

- Τα λογικά στοιχεία δεν πρέπει να επικαλύπτονται μεταξύ τους
- Τα λογικά στοιχεία πρέπει να βρίσκονται εντός ορίων του chip
- Τα λογικά στοιχεία στην περίπτωση των Standard Cells πρέπει να τοποθετούνται πάνω σε διατεταγμένες γραμμές

Γενικά, το σύνηθες είναι η χρήση μιας συνάρτησης κόστους (cost function), που συμπεριλαμβάνει όλες τις παραπάνω προϋποθέσεις και ενός αλγόριθμου, που προσπαθεί να την ελαχιστοποιήσει.

Πρέπει να σημειωθεί ότι το πρόβλημα της τοποθέτησης είναι NP-complete, πράγμα που σημαίνει ότι δεν υπάρχει αλγόριθμος που να το επιλύει με ακρίβεια σε πολυωνυμικό χρόνο. Αυτό σημαίνει ότι αν προσπαθήσουμε να βρούμε όλες τις πιθανές τοποθετήσεις των λογικών στοιχείων ενός κυκλώματος και επιλέξουμε από αυτές τη βέλτιστη θα χρειαστούμε εκθετικό χρόνο ως προς τον αριθμό των στοιχείων, γεγονός που καθιστά τον υπολογισμό της πρακτικά αδύνατο, αν λάβουμε υπ' όψιν ότι σήμερα η τεχνολογία VLSI χρησιμοποιεί εκατομμύρια από λογικές πύλες. Έτσι λοιπόν, χρησιμοποιούνται ευριστικοί αλγόριθμοι, που χρησιμοποιούν ευριστικές συναρτήσεις κόστους και προσεγγίζουν τη βέλτιστη λύση.

3.2. Αλγόριθμοι Τοποθέτησης

Οι αλγόριθμοι τοποθέτησης μπορούν να κατηγοριοποιηθούν στους κατασκευαστικούς (constructive) και στους σταδιακής βελτίωσης (iterative improvement). Στην πρώτη περίπτωση η τοποθέτηση κατασκευάζεται βήμα – βήμα ξεκινώντας από μια αφετηρία, ενώ στη δεύτερη ο αλγόριθμος ξεκινά με μια αρχική τοποθέτηση, την οποία προσπαθεί να τη βελτιώσει σε κάθε επανάληψη της εκτέλεσης του με βάση την ελάττωση κάποιας συνάρτησης κόστους.

Πιο συγκεκριμένα, οι κατασκευαστικοί αλγόριθμοι αρχικά επιλέγουν μια πύλη την οποία τοποθετούν πάνω στην επιφάνεια του chip. Στη συνέχεια, οι υπόλοιπες πύλες επιλέγονται μια τη φορά και τοποθετούνται με βάση τον τρόπο σύνδεσης τους με τις ήδη τοποθετημένες στην επιφάνεια, έτσι ώστε το μήκος της διασύνδεσης που θα τις ενώσει να είναι το ελάχιστο δυνατό. Οι αλγόριθμοι αυτοί σε γενικές γραμμές είναι ιδιαίτερα γρήγοροι αλλά τα αποτελέσματά τους δεν είναι πολύ ενθαρρυντικά. Αυτοί χρησιμοποιούνται κυρίως για να κατασκευάσουν μια τοποθέτηση, η οποία αποτελεί την αρχική των αλγόριθμων σταδιακής βελτίωσης. Συνοψίζοντας, η βασική αιτία χρήσης τους είναι η ταχύτητά τους και το γεγονός ότι καταλαμβάνουν αμελητέο

υπολογιστικό χρόνο σε σχέση με την άλλη κατηγορία αλγορίθμων, αποτελώντας ένα καλό σημείο αρχής για τους τελευταίους.

Από την άλλη πλευρά, οι αλγόριθμοι σταδιακής βελτίωσης παράγουν καλές τοποθετήσεις αλλά είναι ιδιαίτερα χρονοβόροι. Η πιο απλή μέθοδος που χρησιμοποιείται είναι η τυχαία εναλλαγή δύο λογικών πυλών και η αποδοχή αυτής αν το συνολικό κόστος (που προκύπτει με βάση κάποια συγκεκριμένα κριτήρια) μειώνεται. Ο αλγόριθμός αυτός τερματίζει όταν δεν μπορεί να επιτευχθεί παραπέρα βελτίωση μετά από μεγάλο αριθμών εναλλαγών. Πολλές φόρες για να επιτευχθεί καλύτερο αποτέλεσμα δοκιμάζονται πολλές διαφορετικές αρχικές καταστάσεις. Οι πιο γνωστοί αλγόριθμοι της κατηγορίας αυτής είναι οι γενετικοί αλγόριθμοι και ο simulated annealing [19].

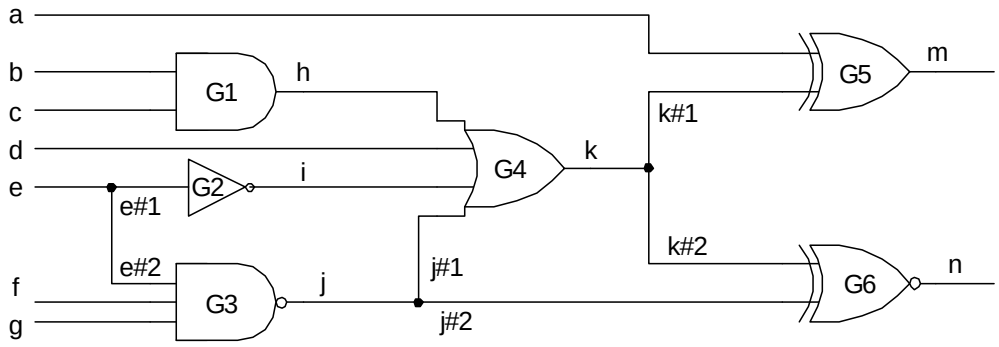
3.3. DFS Αλγόριθμος Τοποθέτησης

Στην παράγραφο αυτή θα εξετάσουμε την πρώτη προσέγγιση μας στην τοποθέτηση λογικών πυλών στην επιφάνεια του chip, η οποία βασίστηκε στη dfs διάσχιση ενός γράφου

3.3.1. Αναπαράσταση Κυκλώματος

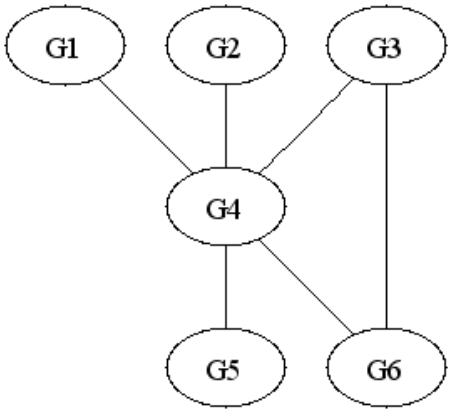
Πριν προχωρήσουμε στην ανάλυση είναι αναγκαίο να αναφερθεί ότι τόσο για την τοποθέτηση όσο και για τη διασύνδεση που θα τη δούμε στο άλλο κεφάλαιο αναπαραστήσαμε το ηλεκτρικό κύκλωμα με ένα γράφο. Κάθε κόμβος του γράφου αντιστοιχίζεται και σε μια λογική πύλη του κυκλώματος, ενώ κάθε ακμή σε μια σύνδεση μεταξύ δύο πυλών. Πρέπει να πούμε ότι οι συνδέσεις των λογικών πυλών με το εξωτερικό περιβάλλον δεν λήφθηκαν υπ’ όψιν. Οι συνδέσεις αυτές είναι οι είσοδοι και οι έξοδοι του chip. Για παράδειγμα αν μια λογική πύλη έχει μια είσοδο που δεν αποτελεί έξοδο μιας άλλης που ανήκει στο chip, αλλά προέρχεται από ένα εξωτερικό λογικό στοιχείο τότε αυτή δεν την υπολογίζουμε. Φορμαλιστικά, λέμε ότι ένα κύκλωμα αναπαριστάται με ένα γράφο G(V, E) όπου V είναι κόμβοι του γράφου και $u \in V$, αν u είναι λογικό κύτταρο και E είναι οι ακμές του γράφου και $(u, \upsilon) \in E$ αν $u,\upsilon \in V$ και υπάρχει λογική σύνδεση μεταξύ αυτών στο κύκλωμα.

Για την αναπαράσταση ενός γράφου αναφέραμε στο προηγούμενο κεφάλαιο υπάρχουν δύο τρόποι· η λίστα γειτνίασης και ο πίνακας γειτνίασης. Εμείς επιλέξαμε τον πρώτο γιατί είναι δυναμικός τρόπος αποθήκευσης και καταλαμβάνει μικρότερη μνήμη. Την άποψη αυτή ενισχύει το ότι το πρόγραμμα μας δοκιμάστηκε σε κυκλώματα με διαφορετικό αριθμό πυλών, πράγμα που σημαίνει αν επιλέγαμε τον πίνακα γειτνίασης θα καταλαμβάναμε πολλή μνήμη, την οποία δεν θα τη χρησιμοποιούσαμε ποτέ. Στα Σχήματα 3.3 και 3.4 φαίνεται ένα λογικό κύκλωμα και η αναπαράσταση αυτού σε γράφο



Σχήμα 3.3 Λογικό Κύκλωμα που Αποτελείται από 6 Πύλες

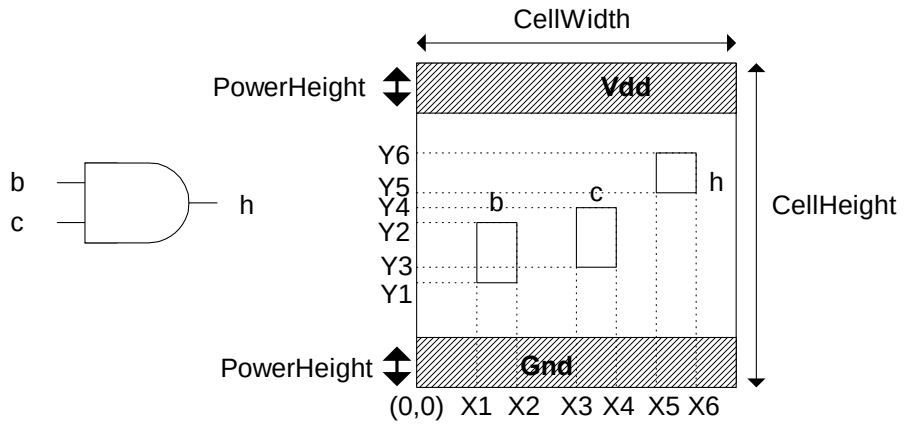
Το κύκλωμα του Σχήματος 3.3 αποτελείται από έξι πύλες, τις G1, G2, …, G6. Οι γραμμές που συνδέουν τις πύλες είναι οι λογικές διασυνδέσεις τους, οι οποίες μάλιστα είναι και ονοματισμένες. Παρατηρούμε ότι οι διασυνδέσεις a, b, c, d, e, f, και g είναι εξωτερικοί είσοδοι του κυκλώματος. Επίσης οι m και n είναι έξοδοι του κυκλώματος προς το εξωτερικό περιβάλλον. Άρα λοιπόν, οι παραπάνω είσοδοι και έξοδοι δε συμμετέχουν στην κατασκευή του γράφου του Σχήματος 3.4



Σχήμα 3.4 Γράφος που Αναπαριστά Λογικό Κύκλωμα

3.3.2. Αναπαράσταση Λογικών Πυλών

Ένα ακόμα στοιχείο που πρέπει να αναφερθεί είναι ο τρόπος αναπαράστασης των λογικών πυλών. Αναφέρθηκε προηγούμενα ότι ο τρόπος σχεδίασης που χρησιμοποιήσαμε είναι τα Standard Cells. Αυτό σημαίνει ότι οι πύλες του κυκλώματος αναπαριστώνται ως ορθογώνια παραλληλόγραμμα, τα οποία έχουν το ίδιο ύψος και διαφορετικό μήκος. Επίσης, κάθε πύλη έχει εισόδους και έξοδο, που και αυτές κατασκευάζονται ως μικρότερα ορθογώνια μέσα στο μεγάλο που αναπαριστά την πύλη. Αυτές δεν παίζουν απολύτως κανένα ρόλο στη διαδικασία της τοποθέτησης παρά μόνο στη διασύνδεση και απλά αναφέρονται στο σημείο αυτό. Οι διαστάσεις τόσο των πυλών όσο και των εισόδων και εξόδων τους βρίσκονται μέσα σε μια βιβλιοθήκη που δίνεται από τον κατασκευαστή. Από αυτήν λοιπόν εμείς αντλήσαμε τα δεδομένα για να δημιουργήσουμε το κύκλωμα. Στο Σχήμα 3.5 φαίνεται η αναπαράσταση της πύλης G1 του παραπάνω κυκλώματος.



Σχήμα 3.5 Αναπαράσταση Λογικής Πύλης

Αυτός που μας ενδιαφέρει είναι το ύψος της πύλης (Cell Height), που είναι ίδιο για όλες τις πύλες του κυκλώματος και το πλάτος της (Cell Width), που ποικίλει από πύλη σε πύλη. Τα (X1, Y1), (X2, Y2) είναι οι συντεταγμένες της κάτω αριστερής και πάνω δεξιάς γωνίας που αναπαριστά την είσοδο b, δεδομένου ότι η κάτω αριστερή γωνία της πύλης είναι τοποθετημένη στο σημείο (0, 0). Αντίστοιχα είναι και τα (X3,Y3), (X4, Y4) και (X5, Y5), (X6, Y6).

3.3.3. Βασική Ιδέα

Η βασική ιδέα του αλγόριθμου είναι πολύ απλή. Επιλέγουμε τον πρώτο κόμβο που είναι αποθηκευμένος στη λίστα γειτνίασης του γράφου που αναπαριστά το λογικό κύκλωμα και ξεκινάμε μια dfs διάσχιση του γράφου με αρχή αυτόν. Με τον τρόπο λοιπόν αυτό επισκεπτόμαστε όλους τους κόμβους. Κάθε κόμβος που συναντάται τοποθετείται στην επιφάνεια του chip με ένα συγκεκριμένο τρόπο. Για τον dfs ο αναγνώστης παραπέμπεται στην παράγραφο 2.4.2.

Το εμβαδό του chip που κατασκευάζεται δεν είναι δεδομένη αλλά εξαρτάται από το χρήστη. Οι διαστάσεις του chip, δηλαδή το ύψος και το μήκος του δίνονται από το χρήστη και μπορούν να είναι όσο μεγάλες επιθυμεί. Είναι προφανές πως αν το εμβαδόν της είναι μικρότερο από το συνολικό εμβαδόν των λογικών πυλών η τοποθέτηση θα είναι ανεπιτυχής. Αφού οριστούν οι διαστάσεις χωρίζουμε το chip με οριζόντιες γραμμές, οι οποίες έχουν μήκος ίσο με το μήκος του chip και απέχουν

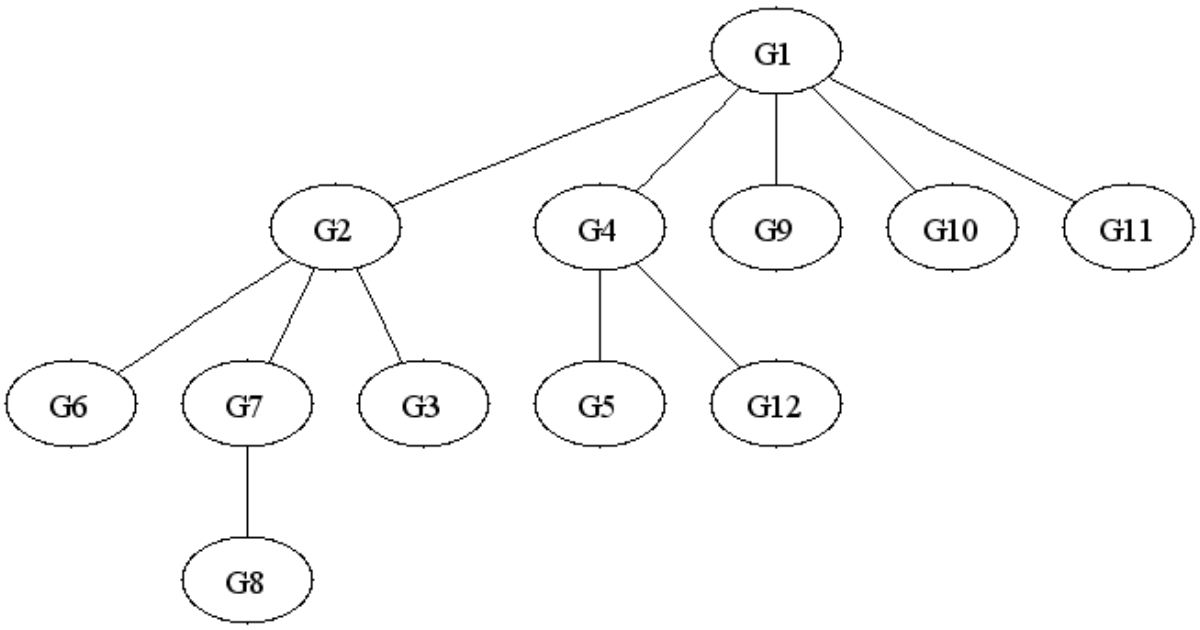
μεταξύ τους απόσταση ίση με το ύψος των πυλών (το οποίο είναι κοινό για όλες τις πύλες). Πάνω σε αυτές τις γραμμές τοποθετούνται οι πύλες του κυκλώματος και πουθενά αλλού. Έτσι λοιπόν, μετά την οριοθέτηση του χώρου ξεκινά η τοποθέτηση.

Ο πρώτος κόμβος που επιλέγεται και ονομάζεται source τοποθετείται στο κέντρο της επιφάνειας του chip για να μπορέσουμε να επεκταθούμε ομοιόμορφα προς όλες τις κατευθύνσεις. Στη συνέχεια, η dfs διάσχιση μας επιτρέπει να επισκεφθούμε όλους τους κόμβους που είναι προσιτοί από τον αρχικό. Έπειτα, επιλέγεται ένας νέος κόμβος ως source με τον τρόπο που αναφέρθηκε στον προηγούμενο κεφάλαιο και η διαδικασία συνεχίζεται μέχρι να τοποθετηθούν όλοι οι κόμβοι.

Ο τρόπος τοποθέτησης τους είναι ο εξής: Αρχικά, κάθε κόμβος προσπαθεί να τοποθετηθεί στην ίδια γραμμή με τον πατέρα του στη dfs διάσχιση σε θέση αμέσως δεξιότερα αυτού. Αν δεν μπορεί να γίνει αυτό, επειδή για παράδειγμα αυτός έχει παραπάνω από έναν γείτονες τότε τον τοποθετούμε στην ίδια γραμμή αμέσως αριστερότερα. Αν ούτε αυτό είναι εφικτό, τον τοποθετούμε στην από πάνω ή στην από κάτω γραμμή, όπου υπάρχει ελεύθερος χώρος. Σε περίπτωση που στις παραπάνω θέσεις υπάρχει κενός χώρος, που δεν είναι αρκετός ώστε να χωρέσει η δεδομένη πύλη αλλά ο συνολικός ελεύθερος χώρος που υπάρχει στη γραμμή είναι μεγαλύτερος από αυτόν πύλης τότε οι πύλες που βρίσκονται πάνω στη συγκεκριμένη γραμμή μετακινούνται είτε αριστερά είτε δεξιά ανάλογα με την περίσταση ώστε να δημιουργηθεί ο χώρος που χρειαζόμαστε.

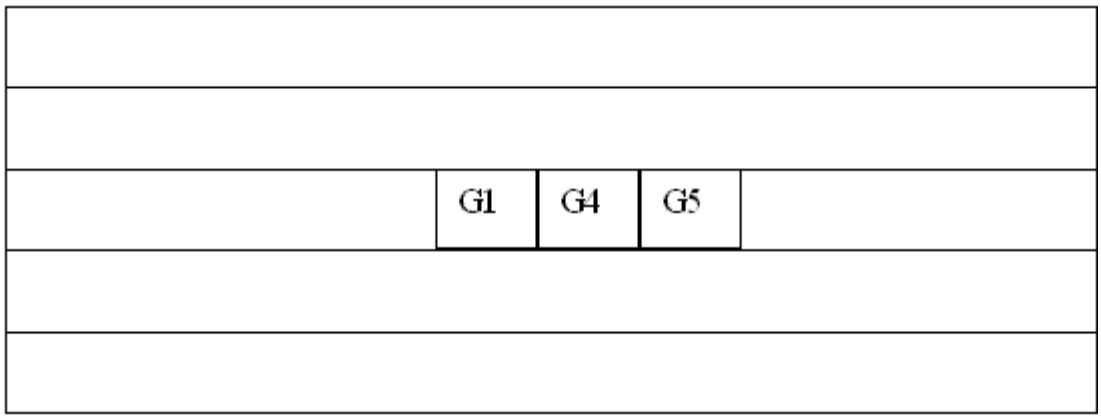
Αν τίποτα από όλα τα παραπάνω δε μπορεί να γίνει εξαιτίας του ότι οι θέσεις αυτές είναι κατειλημμένες από άλλες πύλες τότε ο αλγόριθμος ψάχνει να βρει μια ελεύθερη γραμμή ακριβώς από πάνω ή από κάτω από τον «πατέρα» για να κάνει την τοποθέτηση. Αν ούτε αυτό μπορεί να γίνει τότε ψάχνει να βρει οπουδήποτε υπάρχει ελεύθερος χώρος στην επιφάνεια για να κάνει την τοποθέτηση. Αν υπάρχουν πολλές τέτοιες θέσεις τότε επιλέγει αυτή που είναι κοντινότερα στον «πατέρα» (σύμφωνα με τη dfs διάσχιση) της πύλης. Αν δεν υπάρχει καμία τότε η συγκεκριμένη πύλη δε τοποθετείται και ο αλγόριθμος συνεχίζει με την επόμενη, που προκύπτει από την dfs αναζήτηση.

Για να γίνει περισσότερο κατανοητή η λειτουργία του αλγόριθμου της τοποθέτησης, ας θεωρήσουμε μια περίπτωση όπου θέλουμε να τοποθετήσουμε δώδεκα ίδιες λογικές πύλες πάνω στην επιφάνεια του τσιπ. Ο όρος ίδιες πύλες σημαίνει ότι είναι όλες του ίδιου τύπου και κατά συνέπεια έχουν τις ίδιες διαστάσεις. Αυτές αποτελούν κομμάτι ενός ολοκληρωμένου κυκλώματος, το οποίο το αναπαριστάμε ως γράφο όπως φαίνεται στο Σχήμα 3.6

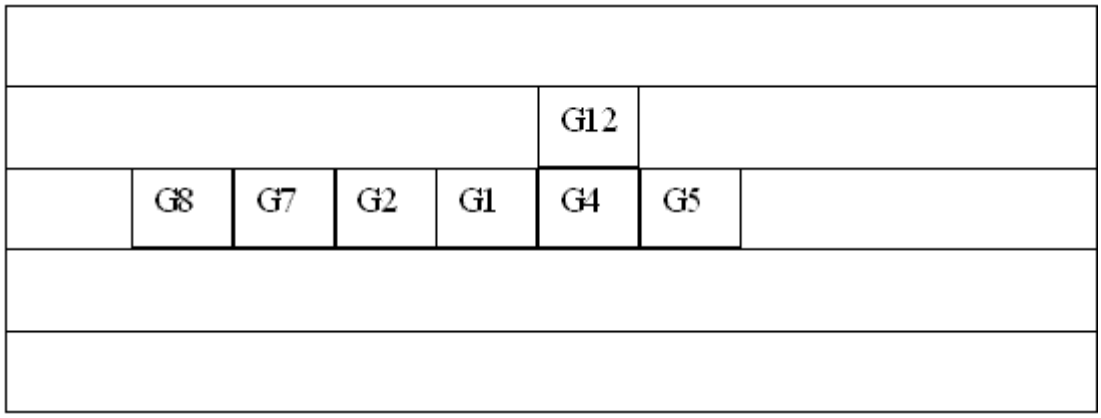


Σχήμα 3.6 Γραφική Αναπαράσταση Λογικού Κυκλώματος

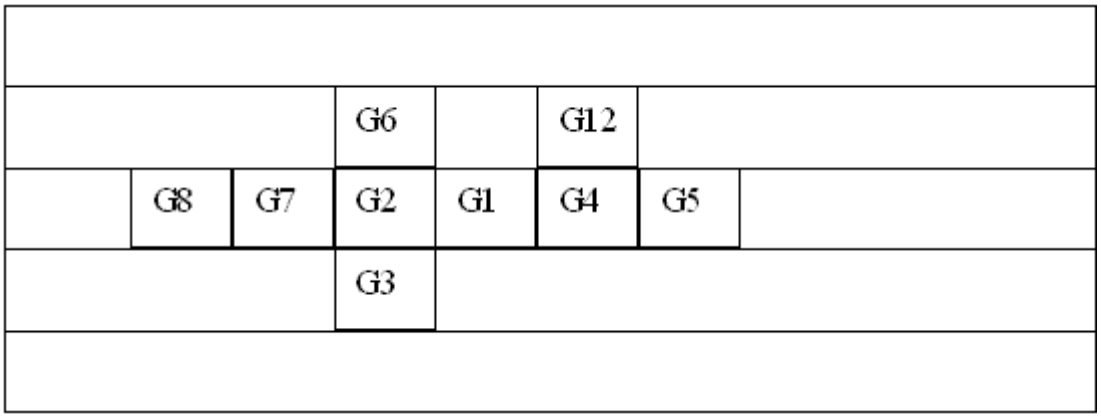
Αρχικά τοποθετείται η πύλη G1 στο κέντρο της επιφάνειας του chip. Στη συνέχεια, ο dfs επισκέπτεται την G4, η οποία τοποθετείται αμέσως δεξιότερα της G1. Η επόμενη πύλη που συναντάμε είναι η G5, που τοποθετείται δεξιότερα της G4 (Σχήμα 3.7). Έπειτα, η αναζήτηση συναντά την G12, η οποία τοποθετείται από επάνω από τη G4 γιατί δεν υπάρχει χώρος ούτε αριστερά ούτε δεξιά της τελευταίας. Στη συνέχεια, τοποθετούνται οι πύλες G2, G7, G8 όπως φαίνεται στο Σχήμα 3.8. Ο Dfs έπειτα συναντά τη G6 που τοποθετείται πάνω από τη G2 και μετά τη G3 που μπαίνει κάτω από αυτή (Σχήμα 3.9). Η G9 θα τοποθετηθεί πάνω από τη G1, ενώ η G10 κάτω από αυτή. Τέλος, η πύλη G11 θα τοποθετηθεί πάνω από την G9 (Σχήμα 3.10). Αυτό συμβαίνει γιατί δεν υπάρχει αρκετός χώρος σε γειτονικές θέσεις γύρω από τον πατέρα της (G1) σύμφωνα με τη dfs διάσχιση.



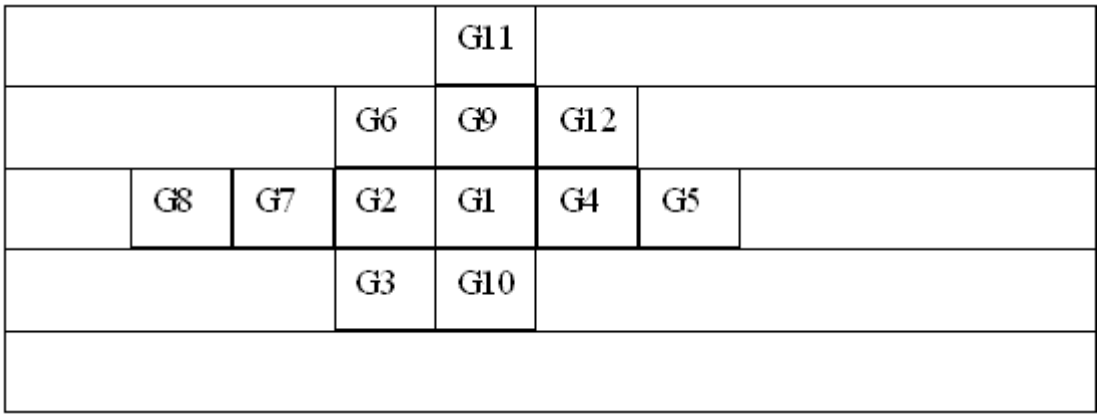
Σχήμα 3.7 Τοποθέτηση των Πυλών G1, G4 και G5



Σχήμα 3.8 Τοποθέτηση των Πυλών G12, G2, G7 και G8



Σχήμα 3.9 Τοποθέτηση των Πυλών G6 και G3



Σχήμα 3.10 Τοποθέτηση των πυλών G9, G10 και G11

Δύο είναι οι παρατηρήσεις που πρέπει να γίνουν. Η πρώτη είναι ότι η τοποθέτηση μπορεί να είναι διαφορετική, αν η αναζήτηση dfs επιλέξει διαφορετικούς κόμβους ως source και η δεύτερη είναι ότι το μήκος των πυλών δεν είναι το ίδιο, αλλά εξαρτάται από τον τύπο τους.

Συνοψίζοντας όλα τα παραπάνω, παραθέτουμε κωδικοποιημένο τον αλγόριθμο που αναπτύξαμε παραπάνω και στηρίζεται στη dfs αναζήτηση. Για την καλύτερη κατανόησή του θεωρούμε ότι το κύκλωμα αναπαριστάται με ένα γράφο G(V,E), όπου V οι κόμβοι του και E οι ακμές του, ο οποίος είναι αποθηκευμένος στη λίστα γειτνίασης Adj.

DFS (G)

1. *For each vertex u ∈G*
2. *u[visit] ← FALSE*
3. *endFor*
4. *For each vertex u ∈G*
5. *if u[visit] = FALSE then*
6. *u[father] ← NULL*
7. *DFS-VISIT(u)*
8. *endIf*
9. *endFor*

DFS-VISIT(u)

1. *u[visit] ← TRUE*
2. *PLACE(u)*
3. *For each v ∈Adj[u]*
4. *if v[visit] = FALSE*
5. *v[father] ← u*
6. *DFS-VISIT(v)*
7. *endIf*
8. *endFor*

PLACE(u)

1. *if Area is empty then*
2. *place u in the middle of the Area*
3. *else*
4. *if there is enough place* ***exact*** *right of u[father] then*
5. *place u right of u[father]*
6. *else if there is enough* ***exact*** *place left of u[father] then*
7. *place u left of u[father]*
8. *else if there is enough* ***exact*** *place above u[father] then*
9. *place u above u[father]*

*10. else if there is enough place **exact** below u[father] then*
11. place u below u[father]
12. else if there is enough place right of u[fahter]
13. move cells right of father and place u
14. else if there is enough place left of u[father]
15. move cells left of father and place u
16. else if there is enough place above u[father]
17. move cells left or right and place u exact above u[father]
18. else if there is enough place below u[father]
19. move cells left or right and place u exact below to u[father]
20. else if there is a line up to u[fahter] with free space exact above u[father]
21. place u
22. else if there is a line down to u[father] with free space exact below to u[father]
23. place u
24. else if there is free area space wherever on the Area
25. place u
26. else
27. placement FAILED.
28. endIf
29. endIf

Ο παραπάνω αλγόριθμος είναι η πρώτη προσέγγιση μας στο πρόβλημα για την τοποθέτηση των λογικών πυλών πάνω στο κύκλωμα. Αυτός λαμβάνει υπ' όψιν τη συνδεσιμότητα μεταξύ των πυλών, αφού ο dfs εξ ορισμού επισκέπτεται γειτονικούς κόμβους, οι οποίοι προσπαθούν να τοποθετηθούν κοντά ο ένας στον άλλο για να κάνουν πιο εύκολο το ρόλο της διασύνδεσης και να περιορίσουν το κόστος της. Επίσης, το γεγονός ότι μετακινούμε τις πύλες πάνω στην επιφάνεια για να δημιουργήσουμε χώρο έχει το μειονέκτημα ότι πύλες που πρέπει να βρίσκονται κοντά, πιθανόν απομακρύνονται με τη μετακίνηση αυτή αλλά επιτυγχάνουμε με αυτόν τον τρόπο να μικραίνουμε πολύ την επιφάνεια του chip. Αυτό σημαίνει ότι ο αλγόριθμος λειτουργεί με επιτυχία και κάνει σωστή τοποθέτηση ακόμα και σε περιπτώσεις που η επιφάνεια του chip είναι σχεδόν ίση με τη συνολική επιφάνεια των

πυλών. Είναι προφανές ότι η μέθοδος αυτή ανήκει στις κατασκευαστικές, αφού ξεκινάμε με μια αρχική πύλη και βήμα-βήμα κατασκευάζουμε ολόκληρη την τοποθέτηση της επιφάνειας, πράγμα που την καθιστά και ιδιαίτερα γρήγορη.

3.4. Τοποθέτηση Βασισμένη σε Πυκνούς Υπογράφους

Αυτή αποτελεί τη δεύτερη προσέγγιση μας στο πρόβλημα της τοποθέτησης λογικών πυλών. Σε αντίθεση με την πρώτη, εδώ προηγείται της τοποθέτησης ένας διαμερισμός του αρχικού γράφου που αναπαριστά το κύκλωμα. Σύμφωνα με το διαμερισμό αυτό, ψάχνουμε να βρούμε πυκνούς υπογράφους (dense sub graphs) μέσα στον αρχικό γράφο, δηλαδή να βρούμε κόμβους που έχουν πολλές ακμές (συνδέσεις) μεταξύ τους. Στη συνέχεια, τοποθετούμε έναν-έναν τους υπογράφους που έχουμε υπολογίσει, πράγμα που αποτελεί μια χωροθέτηση του κυκλώματος. Παράλληλα με αυτήν γίνεται και η τοποθέτηση των πυλών πάνω στην επιφάνεια του chip.

3.4.1. Υπολογισμός Πυκνού Υπογράφου

Βασικό στοιχείο της μεθόδου αυτής είναι ο υπολογισμός πυκνών υπογράφων. Αρχικά, θα ορίσουμε την πυκνότητα ενός γράφου με βάρη (weighted graph) και στη συνέχεια τι πούμε ονομάζουμε πυκνό υπογράφο.

Ο G(V, E, w) είναι ένας μη κατευθυνόμενος γράφος με θετικά βάρη (weighted undirected graph) με n κόμβους V και m ακμές E. Θεωρούμε w(u) το βάρος του κόμβου u και w(e) το βάρος της ακμής e. Το βάρος ενός γράφου ορίστηκε στο 2^{ο} Κεφάλαιο. Η πυκνότητα ενός τέτοιου γράφου δίνεται από τη Εξίσωση 3.1 και συμβολίζεται με d(G).

$$d(G) = \sum_{e \in G} w(e) - \sum_{u \in G} w(u) \qquad \text{Εξ 3.1}$$

Ένας υπογράφος A του G, η πυκνότητα του οποίου είναι μεγαλύτερη από μια σταθερά K, δηλαδή ισχύει η εξίσωση 3.2 ονομάζεται πυκνός.

$$d(A) = \sum_{e \in A} w(e) - \sum_{u \in A} w(u) > K \qquad \text{Εξ 3.3}$$

Η τιμή του K μπορεί να είναι οσοδήποτε μεγάλη ή μικρή (μπορεί να πάρει και αρνητικές τιμές) ανάλογα με το πόσο πυκνούς υπογράφους θέλουμε να βρίσκουμε. Επίσης, στον αρχικό γράφο αποδώσαμε βάρη τόσο στις ακμές όσο και στους κόμβους του. Οι τιμές αυτών ήταν ακέραιες και δόθηκαν με τέτοιο τρόπο ώστε ο αρχικός γράφος να είναι πυκνός. Αυτό έγινε με το σκεπτικό να υπάρχει τουλάχιστον ένας πυκνός υπογράφος – ο αρχικός – τον οποίο θα εντοπίσει σίγουρα ο αλγόριθμος. Έτσι λοιπόν, αν στις ακμές αποδώσουμε βάρος ίσο με 10, τότε το βάρος των κόμβων θα δίνεται από την εξίσωση 3.4 δεδομένου ότι K=0.

$$w(e) = 10 \qquad \text{Εξ 3.3}$$

$$w(u) = floor(\frac{|E|}{|V|} * 10) \qquad \text{Εξ 3.4}$$

Για παράδειγμα αν ο γράφος μας αποτελείται από 160 κόμβους και 255 ακμές τότε κάθε ακμή θα έχει βάρος 10 και κάθε κόμβος

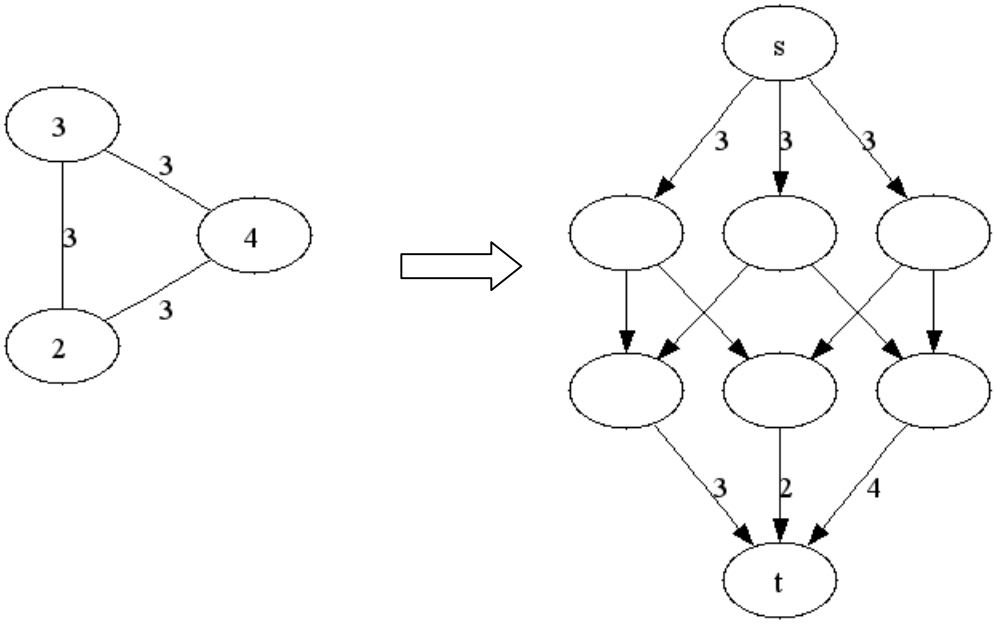
$$w(u) = floor(\frac{|E|}{|V|} * 10) = floor(\frac{255}{160} * 10) = floor(15.9375) = 15$$

Για τον υπολογισμό τέτοιων γράφων χρησιμοποιήσαμε έναν αλγόριθμο, ο οποίος βασίζεται σε δίκτυα ροής τα οποία αναλύθηκαν στο Κεφάλαιο 2. Για το λόγο λοιπόν αυτό μετατρέπουμε τον αρχικό γράφο που αναπαριστά το κύκλωμα σε ένα τέτοιο δίκτυο.

3.4.2. Κατασκευή Δικτύου Ροής

Από τον αρχικό γράφο G που αναπαριστά το κύκλωμα κατασκευάζουμε ένα διμερές κατευθυνόμενο δίκτυο ροής (bipartite directed network) G΄=(M, N, s, t, E΄, w), όπου N είναι οι κόμβοι του V και M οι ακμές του E του αρχικού γράφου G. Επίσης, οι E΄ είναι οι ακμές του G΄ για τις οποίες ισχύει E΄={(e, u), (e, v) | e = (u, v), e ∈ E}. Η πηγή (source) s ενώνεται με μια κατευθυνόμενη ακμή με κάθε κόμβο που ανήκει στο M και κάθε κόμβος που ανήκει στο N ενώνεται με μια κατευθυνόμενη ακμή με την καταβόθρα (sink) t. Η χωρητικότητα της ακμής του δικτύου (s, e), e ∈ M είναι ίση με το βάρος w(e) της ακμής e του G. Η χωρητικότητα της ακμής του δικτύου (u, t), u ∈ N, είναι ίση με το βάρος w(u) του κόμβου u στο G. Η χωρητικότητα της ακμής

δικτύου (e, u), e ∈ M και u ∈ N θεωρείται άπειρη. Τέλος, δεν υπάρχουν άλλες ακμές δικτύου. Στο Σχήμα 3.11 φαίνεται η μετατροπή από ένα γράφο σε ένα δίκτυο με την παραπάνω διαδικασία. Κάθε ακμή έχει βάρος 3, ενώ το βάρος κάθε κόμβου είναι γραμμένο μέσα στον κύκλο που τον αναπαριστά. Έτσι λοιπόν , ο ένας κόμβος έχει βάρος 2, ο άλλος 3 και ο τελευταίος 4. Πρέπει να σημειωθεί όμως ότι οι κόμβοι στην περίπτωση μας έχουν όλοι το ίδιο βάρος.

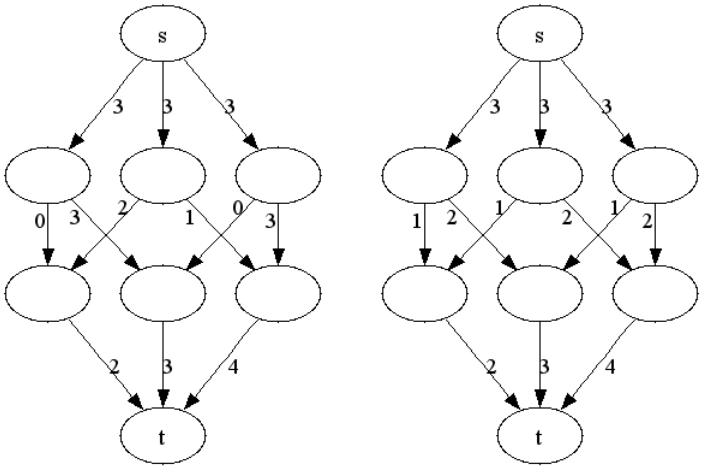


Σχήμα 3.11 Ο Αρχικός Γράφος και η Μετατροπή του σε Δίκτυο Ροής

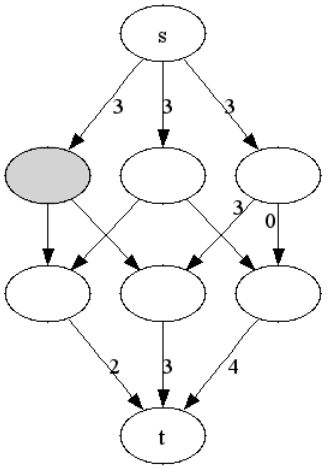
3.4.3. Βασική Ιδέα

Η βασική ιδέα του αλγόριθμου είναι η εξής: Ξεκινάμε με έναν άδειο γράφο G^* από το G και κάθε φορά προσθέτουμε σε αυτόν και από ένα κόμβο. Μόλις προσθέτουμε έναν κόμβο u, εξετάζουμε τις γειτονικές ακμές αυτού e με τον G^*. Για κάθε ακμή e, προσπαθούμε να κατανείμουμε το βάρος της w(e) είτε στον έναν είτε και στους δύο κόμβους που ενώνει, χωρίς να υπερβούμε τα βάρη τους. Ο τρόπος με τον οποίο θα κατανεμηθούν τα βάρη των ακμών δεν είναι πάντα ο ίδιος, έτσι λοιπόν στο Σχήμα 3.12 παρατηρούμε δύο διαφορετικές κατανομές των βαρών για το ίδιο δίκτυο ροής. Επίσης, πολλές φορές είναι αναγκαίο να ανακατανεμηθεί διαφορετικά το βάρος μιας

ακμής για να μπορέσουν να κατανεμηθούν αργότερα και τα βάρη άλλων ακμών όπως φαίνεται στο Σχήμα 3.13



Σχήμα 3.12 Δύο Διαφορετικές Κατανομές Βαρών για το Ίδιο Δίκτυο Ροής



Σχήμα 3.13 Δίκτυο Ροής, το οποίο θα Χρειαστεί Ανακατανομή

Στο Σχήμα 3.13 όταν θα επιχειρήσει ο αλγόριθμος να κατανείμει το βάρος της ακμής του γράφου G, που είναι χρωματισμένη γκρι δε θα μπορέσει γιατί ο κόμβος που έχει βάρος τρία έχει ήδη συμπληρωμένη τη χωρητικότητά του. Άρα λοιπόν, θα χρειαστεί να γίνει ανακατανομή της κατανεμημένης ακμής και να καταλήξει έτσι σε ένα δίκτυο ροής αντίστοιχο με αυτών του Σχήματος 3.12.

Εάν ο αλγόριθμος καταφέρει να κατανείμει το βάρος όλων των ακμών, τότε ο G^* ταυτίζεται με το G και κατά συνέπεια ο τελευταίος δεν είναι πυκνός. Στην περίπτωση που δεν υπάρχει κανένας πυκνός υπογράφος τότε ο αλγόριθμος τερματίζει μετά από O(n(m+n)) βήματα, όπου n είναι το πλήθος των κόμβων και m το πλήθος των ακμών του G. Ο χρόνος αυτός προκύπτει ως εξής: Για κάθε ένα κόμβο ο αλγόριθμος εκτελεί μια BFS αναζήτηση στο γράφο για να μπορέσει να κατανείμει το βάρος των γειτονικών ακμών του. Στη χειρότερη περίπτωση θα ελέγξει όλους του κόμβους και όλες τις ακμές, που είναι m+n. Συνεπώς, θα χρειαστεί τελικά χρόνο O(n(m+n)). Από την άλλη πλευρά, στην περίπτωση που υπάρχει κάποιος πυκνός υπογράφος, τότε θα υπάρχει μια ακμή, της οποίας το βάρος δε θα μπορεί να κατανεμηθεί ακόμα και μετά από ανακατανομή των βαρών. Ο κόμβος αυτός, οι ακμές του οποίου δεν μπορούν να κατανεμηθούν προστίθεται στον G^* και ο αλγόριθμος επιστρέφει τον πυκνό υπογράφο.

Κατανομή του βάρους μιας ακμής e του G αντιστοιχεί στο να προσθέσουμε ροή στο δίκτυο ροής G΄ ίση με τη χωρητικότητα της (s, e) από το s στο t του G΄. Αυτό μπορεί να επιτευχθεί είτε απευθείας με ένα μονοπάτι της μορφής $s \rightarrow e \rightarrow u \rightarrow t$, είτε να διαμοιραστεί σε δύο κόμβος διαμέσου ενός μονοπατιού της μορφής $s \rightarrow e_1 \rightarrow u_1 \rightarrow e_2 \rightarrow u_2 \rightarrow t$ είτε ακόμα να γίνει πρώτα ανακατανομή της ήδη υπάρχουσας ροής του δικτύου. Για να πετύχουμε το τελευταίο χρησιμοποιούμε τεχνικές παρόμοιες με αυτές του [20]. Το μεγάλο πλεονέκτημα που έχουμε σε αυτήν την περίπτωση είναι ότι το βάρος μιας ακμής και κατά συνέπεια η ροή κατανέμεται το πολύ σε δύο κόμβους.

Έτσι λοιπόν, αν καταφέρουμε να βρούμε ένα τέτοιο μονοπάτι, το οποίο ονομάζεται αυξητικό (augmenting path), τότε η ροή που έχει εισαχθεί στο G΄ παρουσιάζει μια κατανομή των βαρών κάθε ακμής e του συγκεκριμένου υπογράφου G^*, ο οποίος αποτελείται από κόμβους και τις ακμές που έχουν εξεταστεί και ανήκουν στο αρχικό γράφο G έτσι ώστε

το βάρος w(e) κάθε ακμής που ενώνει δύο κόμβους a και b χωρίζεται σε δύο μέρη τα f_e^a και f_e^b ώστε να ισχύει $f_e^a + f_e^b = w(e)$

και για κάθε κόμβο $u \in G', \sum_{e=(e,*)} f_e^u \leq w(u)$

Τέλος, εάν δε μπορέσουμε να βρούμε ένα αυξητικό μονοπάτι τότε το βάρος w(e) της ακμής e δεν μπορεί να κατανεμηθεί με αποτέλεσμα ο G^* να είναι ένας πυκνός υπογράφος. Αυτός αποτελείται από όλους εκείνους τους κόμβους και τις ακμές που εξέτασε ο αλγόριθμος στο δίκτυο ροής G΄ για να μπορέσει να κατανείμει το w(e) [21].

Με τον τρόπο λοιπόν αυτό έχουμε υπολογίσει έναν μόνο πυκνό υπογράφο. Το ζητούμενο όμως είναι να βρούμε όλους τους πυκνούς υπογράφους που πιθανόν υπάρχουν μέσα στον αρχικό γράφο. Για να το πετύχουμε αυτό, μόλις εντοπίσουμε ένα τέτοιο γράφο G^*, τον αντικαθιστούμε με έναν υπερκόμβο στον αρχικό γράφο G ως εξής.

Αρχικά, διαγράφουμε όλους τους κόμβους του G^* από τον G, καθώς επίσης και όλες τις εσωτερικές ακμές του G^*. Στη θέση αυτών κατασκευάζουμε έναν υπερκόμβο h, ο οποίος αντιπροσωπεύει τους κόμβους του G^* και συνδέεται με τους υπόλοιπους του G, έτσι ώστε η ακμή (h, v) υπάρχει αν $(u, v) \in E$ και $u \in G^*$ και $v \in G$. Στη συνέχεια διαγράφουμε και τις εσωτερικές ακμές του G^* δηλαδή όλες εκείνες που έχουν τη μορφή (u, v) και $u \in G^*$ και $v \in G^*$. Έπειτα, κατασκευάζουμε το νέο δίκτυο ροής, το οποίο περιλαμβάνει και τον υπερκόμβο, το βάρος του οποίου το θεωρούμε μηδέν και επαναεκτελούμε τον αλγόριθμο για να βρούμε έναν άλλο πυκνό υπογράφο, ο οποίος θα αντικατασταθεί από έναν υπερκόμβο κοκ μέχρις ότου ανακαλυφθούν όλοι οι πυκνοί υπογράφοι που υπάρχουν στον αρχικό γράφο. Παρακάτω παρουσιάζεται ο αλγόριθμος, που εντοπίζει πυκνούς υπογράφους καθώς επίσης και αυτός που κατανέμει το βάρος της ακμής στο διάγραμμα ροής.

Αλγόριθμος Dense

1. *$G^* = NULL$*
2. *for every vertex u do*
3. *for every edge incident to u do*
4. ***Distibute** w(e) of e*
5. *if not able to distribute all of w(e) then*
6. *A = set of vertices labeled during **Distribute***

7. goto 12

8. endif

9. endfor

10. add u to G^*

11. endfor

12. if A = NULL then no dense subgraph exists

13. else A is a dense subgraph

14. make A hyper node

15. goto 1

16. endIf

Αλγόριθμος Distribute (G΄, f, edge)

G΄ = (M, N, s, t, E΄, w)

f = set of flows f_e^u

edge = edge that be distributed

1. Initialize scan(u) = label(u) = scan(e) = label(e)=0 for all u∈N & e∈M

2. vert = capvert = 0

3. label(edge) = 1, pathcap(edge) = w(edge)

4. while(w(edge) > $\sum_u f_{edge}^u$) or not all labeled nodes have been scanned

5. for all labeled e∈M, with scan(e) = 0

6. label unlabeled neighbors of e (i.e. u∈N)

7. if pathcap(e) != 0 then pathcap(u) = pathcap(e)

8. else pathcap(u) = w(e) – f_e^u

9. endif

10. scan(e) = 1, pred(u) = e

11. endfor

12. for all labeled u∈N with scan(u)=0

13. if(min(w(u) - $\sum_e f_e^u$, pathcap(u), pathcap(edge))>capvert) then

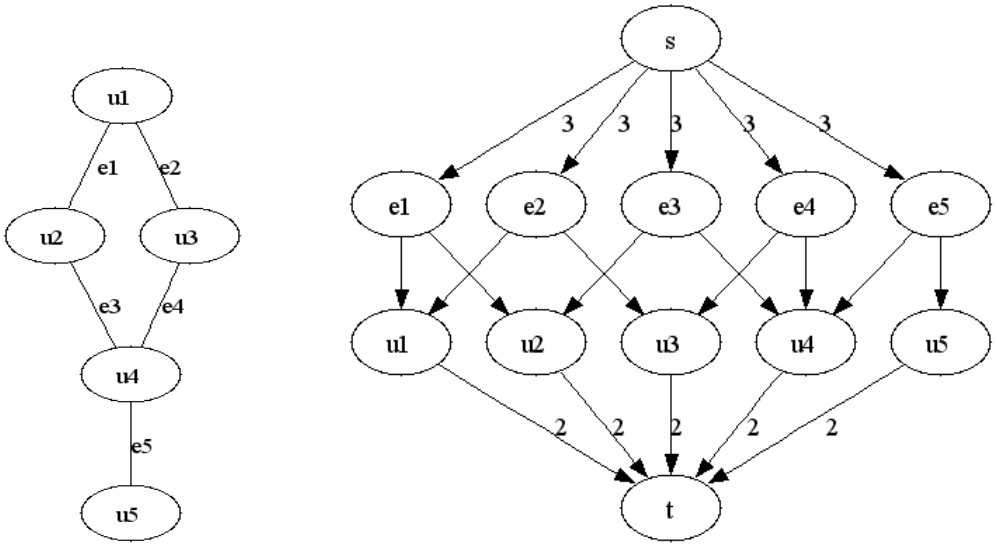
14. vert = u

15. capvert = min(w(u) - $\sum_e f_e^u$, pathcap(u), pathcap(edge))

16. else

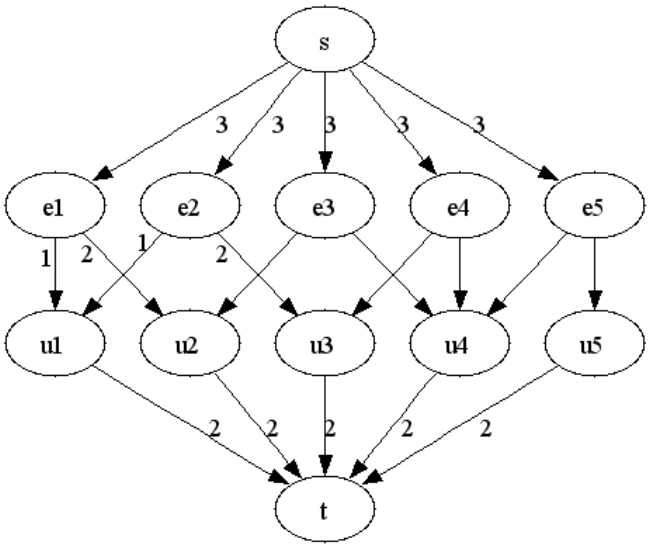
17. *label all unlabeled $e' \in M$ s.t. $f_{e'}^{u} > 0$*
18. *endif*
19. *scan(u)=1*
20. *endfor*
21. *if vert > 0 then*
22. *an augmenting path from s to t found: backtrace from vert using pred() and change values of f_e^u as required*
23. *for all $e \in M$ & $u \in N$*
24. *label(e) = 0, scan(e) = 0, label(u) = 0, scan(u) = 0*
25. *endfor*
26. *vert = 0, capvert = 0, label(edge) = 1*
27. *pathcap(edge) = w(edge) - $\sum_u f_{edge}^u$*
28. *endif*
29. *endwhile*

Για να γίνει περισσότερο κατανοητή η λειτουργία του παραπάνω αλγόριθμου παραθέτουμε στη συνέχεια ένα παράδειγμα εκτέλεσής του. Στο Σχήμα 3.14 φαίνεται ένας γράφος G=(V,E) με V={u1, u2, u3, u4, u5}, ο οποίος αναπαριστά ένα κομμάτι ενός λογικού κυκλώματος. Θεωρούμε ότι όλοι οι κόμβοι έχουν βάρος ίσο με δύο και όλες οι ακμές ίσο με 3. Στο ίδιο σχήμα φαίνεται και το δίκτυο ροής που αντιστοιχεί στο γράφο.

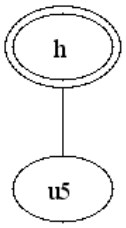


Σχήμα 3.14 Γράφος G που Αναπαριστά Λογικό Κύκλωμα και το Αντίστοιχο Δίκτυο Ροής.

Αρχικά, ο αλγόριθμος επιλέγει το κόμβο u1 και κατανέμει το βάρος των γειτονικών ακμών του (u1, u2) και (u1, u3) με επιτυχία (Σχήμα 3.15). Στη συνέχεια επιλέγεται ο κόμβος u2 και επιχειρείται να κατανεμηθεί μόνο το βάρος της (u2, u4), αφού το βάρος της (u1, u2) έχει ήδη κατανεμηθεί. Αυτό όμως είναι αδύνατο, οποιαδήποτε ανακατανομή βαρών και να γίνει. Συνεπώς, έχουμε ανακαλύψει ένα πυκνό υπογράφο, τον *H*. Ο *H* αποτελείται από τους κόμβους {u1, u2, u3, u4} και από τις ακμές {(u1, u2), (u1, u3), (u2, u4), (u3, u4)}. Έπειτα, μετατρέπουμε τον *H* σε έναν υπερκόμβο *h* και ο αρχικός γράφος G μετατρέπεται στον $G^*=(V^*,E^*)$ με $V^* = \{h, u5\}$ και $E^*=\{(h,u5\}\}$ (Σχήμα 3.16).



Σχήμα 3.15 Κατανομή των Βαρών των Ακμών των (u1, u2) και (u1, u3)



Σχήμα 3.16 Ο γράφος G^* που Προκύπτει από τον G με την Αντικατάσταση του Πυκνού Υπογράφου από τον Υπερκόμβο *h*

Στη συνέχεια, ο αλγόριθμος θα αναζητήσει στο G* έναν νέο πυκνό υπογράφο και η παραπάνω διαδικασία επαναλαμβάνεται μέχρις ότου όλος ο αρχικός γράφος G αντικατασταθεί από έναν μόνο υπερκόμβο ή να μην υπάρχει άλλος πυκνός υπογράφος. Στη συγκεκριμένη περίπτωση ο G^* δεν περιλαμβάνει άλλο πυκνό υπογράφο αλλά είναι ο ίδιος πυκνός. Έτσι λοιπόν, θα μετατραπεί σε έναν υπερκόμβο *h1* και ο αλγόριθμος θα τερματίσει.

Αφού έχουμε υπολογίσει όλους τους πυκνούς υπογράφους ξεκινάμε να τους τοποθετούμε με διαδικασία παρόμοια με αυτή που περιγράφηκε στον προηγούμενο αλγόριθμο, στην επιφάνεια του τσιπ ξεκινώντας από αυτόν που βρήκαμε πρώτα έως ότου να τοποθετηθούν όλοι οι υπογράφοι. Εδώ αρχικά θα τοποθετηθεί ο *h* και μετά ο *h1*.

Πρέπει να σημειώσουμε ότι αυτό που μας ενδιαφέρει δεν είναι να υπολογίζουμε απλά πυκνούς υπογράφους, αλλά να υπολογίζουμε αυτούς που περιέχουν τον ελάχιστο αριθμό κόμβων (minimal dense subgraphs).Για το λόγο αυτό χρησιμοποιούμε τεχνικές παρόμοιες με του [21] υλοποιώντας τον παρακάτω αλγόριθμο. Αυτός παίρνει ως είσοδο τον πυκνό υπογράφο που υπολογίσαμε με την παραπάνω μέθοδο και την κατανομή των βαρών όλων των ακμών. Αναφέρουμε ότι ο κόμβος u_{l+1} είναι ο τελευταίος κόμβος που προκάλεσε την ανακάλυψη ενός πυκνού υπογράφου. Επίσης, e' είναι η ακμή μεταξύ των u_{l+1} και u_k.

1. $B = \{u_{l+1}\}$
2. *while* $d(B) \leq 0$ *and d(A) > 1 do*
3. *choose* $u_k \in A - B$
4. $c = d(A) - w(e') + f_{e'}^{u_k} + f_{e'}^{u_{l+1}}$
5. *for all* $u \in N$ *(Removing* $\{u_k\}$ *form A)*
6. *Let* $e = (u, u_k)$
7. *remove e from M*
8. *end for*
9. *remove* u_k *from N*
10. *Distribute (in A) excess c from the edges of u*
11. *if there are some undistributed edges left then*
12. *set A = new labeled graph*
13. *else*
14. *set* $A = A \cup \{u_k\}$
15. *set* $B = B \cup \{u_k\}$
16. *end if*
17. *end while*
18. *output B, if d(B)>0, else output A*

ΚΕΦΑΛΑΙΟ 4. ΔΙΑΣΥΝΔΕΣΗ ΛΟΓΙΚΩΝ ΚΥΤΤΑΡΩΝ ΤΟΥ ΚΥΚΛΩΜΑΤΟΣ

4.1. Εισαγωγή

Στο φυσικό σχεδιασμό (physical design) των κυκλωμάτων VLSI, κατά τη φάση της διασύνδεσης (Routing phase) καλούμαστε να βρούμε τη διάταξη των ηλεκτρικών διασυνδέσεων (wires), τα οποία συνδέουν τις λογικές πύλες ή τα μπλοκ των λογικών πυλών στο κύκλωμα. Στο σημείο αυτό, γνωρίζουμε τις θέσεις των λογικών πυλών στο κύκλωμα, αφού αυτές έχουν υπολογιστεί στη φάση της τοποθέτησης και της χωροθέτησης (placement – floorplaning phase). Πριν προχωρήσουμε παρακάτω, θα πρέπει να ορίσουμε μερικά χαρακτηριστικά του routing που θα τα χρησιμοποιήσουμε στη συνέχεια (Σχήμα 4.1).

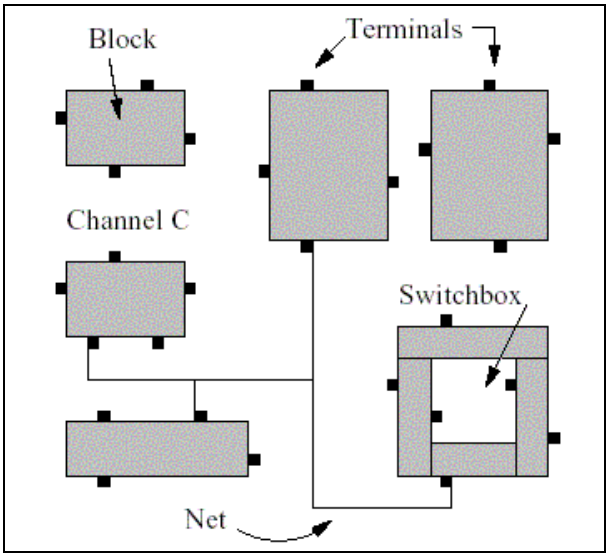
Terminal ονομάζουμε μια τοποθεσία στο μπλοκ, όπου τις περισσότερες φορές βρίσκεται σε μια ακμή του, η οποία πρέπει να διασυνδεθεί με άλλο μπλοκ.

Net ονομάζουμε σύνολο λογικών πυλών, οι οποίες πρέπει να συνδεθούν μεταξύ τους

Netlist ονομάζουμε το σύνολο από nets που πρέπει να συνδεθούν.

Routing space ονομάζουμε το χώρο που μπορούμε να τοποθετήσουμε wires πάνω στο τσιπ.

Channel ονομάζουμε το routing space μεταξύ δύο μπλοκ



Σχήμα 4.1

Επίσης, οι διασυνδέσεις γίνονται σε 2 επίπεδα, καθένα από τα οποία έχει διαφορετική κατεύθυνση. Στο πρώτο επίπεδο οι διασυνδέσεις γίνονται κάθετα και στο δεύτερο οριζόντια. Οι γραμμές διασύνδεσης δε μπορούν ποτέ να είναι διαγώνιες. Η διασύνδεση 2 γραμμών που βρίσκονται σε διαφορετικό επίπεδο γίνεται με τη χρήση vias.

Το γεγονός λοιπόν, ότι ένα κύκλωμα μπορεί να αποτελείται από εκατομμύρια τρανζίστορ και από χιλιάδες nets καθιστά το πρόβλημα της διασύνδεσης (routing problem) υπολογιστικά πολύ δύσκολο. Πιο συγκεκριμένα, ο υπολογισμός μιας βέλτιστης διάταξης (wire layout) ηλεκτρικών καλωδίων (είναι αυτή που έχει το μικρότερο μήκος) για ένα απλό net είναι NP-hard πρόβλημα. Για το λόγο αυτό χωρίζουμε το πρόβλημα σε 2 μέρη: το *Global Routing* και το *Detailed Routing*.

Στο global routing τα ακριβή γεωμετρικά χαρακτηριστικά παραβλέπονται και προσδιορίζονται τα channels που θα χρησιμοποιηθούν για τη διασύνδεση των μπλοκ των λογικών πυλών. Σκοπός του είναι η ελαχιστοποίηση της επιφάνειας των διασυνδέσεων. Στο detail routing καθορίζονται οι ακριβείς διαδρομές διασύνδεσης μεταξύ των μπλοκ (routes) και τα επίπεδα (layers) από τα οποία περνάει η κάθε

διασύνδεση. Σκοπός της φάσης αυτής είναι η ελαχιστοποίηση του συνολικού μήκους των διασυνδέσεων, της συνολικής επιφάνειας των διασυνδέσεων και του αριθμού των επιπέδων.

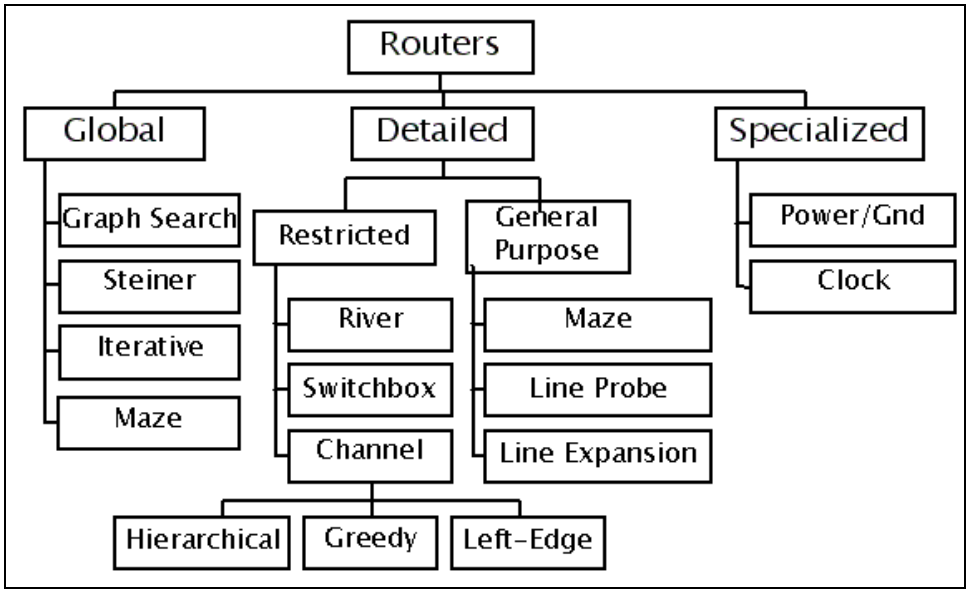
Παρά τον διαχωρισμό αυτόν του routing, το πρόβλημα παραμένει να είναι NP-hard και για το λόγο αυτό αλγόριθμοι που έχουν επινοηθεί λύνουν το πρόβλημα προσεγγιστικά

4.2. Αλγόριθμοι Routing – Διασυνδέτες VLSI

Σήμερα, υπάρχουν πολλοί αλγόριθμοι που προσπαθούν να επιλύσουν το πρόβλημα της διασύνδεσης σε VLSI κυκλώματα. Οι τεχνικές αυτών ποικίλουν ανάλογα το σκοπό για τον οποίο χρησιμοποιούνται (Σχήμα 4.2). Έτσι λοιπόν, στην περίπτωση του global routing, ευρέως διαδεδομένοι είναι οι Maze και οι Steiner αλγόριθμοι. Οι πρώτοι είναι greedy και τα πλεονεκτήματα αυτών είναι η απλότητα και το ότι μας οδηγούν σίγουρα στη βέλτιστη λύση. Το μεγάλο μειονέκτημα τους είναι ότι δε μπορούν να διαχειριστούν καλά την περίπτωση διασύνδεσης πολλών terminals μεταξύ τους (multi terminals). Αυτή συναντάται όταν η έξοδος μιας λογικής πύλης ενός λογικού κυκλώματος γίνεται είσοδος σε περισσότερες από μια λογικές πύλες. Ο πιο γνωστός ίσως αλγόριθμος της κατηγόρίας αυτής είναι ο Lee Algorithm [22]. Οι τελευταίοι χρησιμοποιούν Steiner Trees, είναι ιδιαίτερα γρήγοροι και βρίσκουν αρκετά καλές λύσεις. Το μειονέκτημά τους είναι ότι δυσκολεύονται να προβλέψουν ή να αποφύγουν καταστάσεις συμφόρησης γραμμών διασύνδεσης.

Από την άλλη πλευρά στο detail routing οι αλγόριθμοι είναι περισσότερο εξειδικευμένοι. Έτσι λοιπόν, υπάρχουν αυτοί που χρησιμοποιούνται για διασύνδεση μέσα σε switchboxes (Σχήμα 4.1), μέσα σε channels κτλ. Εκτός αυτών, υπάρχουν και αλγόριθμοι γενικού σκοπού, που χρησιμοποιούνται για όλα τα παραπάνω. Από αυτούς «ξεχωρίζουν» οι Maze (όμοιοι με το global routing) και οι line Probe. Τα πλεονεκτήματα των line Probe αλγορίθμων είναι ότι η ταχύτητα και οι μικρές απαιτήσεις τους σε μνήμη. Το κυριότερο μειονέκτημα αυτών είναι το γεγονός ότι μερικές φορές ο βασικός αλγόριθμος δε μπορεί να βρει μια διασύνδεση, ενώ αυτή υπάρχει, με αποτέλεσμα να επαναλαμβάνει πολλές φορές την αναζήτηση με διαφορετικά κριτήρια.

Τέλος, υπάρχει η ειδική διασύνδεση ρολογιού και τροφοδοσίας. Η περίπτωση αυτή απαιτεί ειδική διαχείριση, καθώς τόσο η τροφοδοσία όσο και το ρολόι είναι δύο σήματα με πολύ μεγάλες απαιτήσεις και πολλά προβλήματα. Για το λόγο αυτό ασχολούμαστε πρώτα με αυτές τις διασυνδέσεις και ύστερα με όλες τις υπόλοιπες [17].



Σχήμα 4.2 Κατηγοριοποίηση Αλγορίθμων Διασύνδεσης

4.3. Maze Router

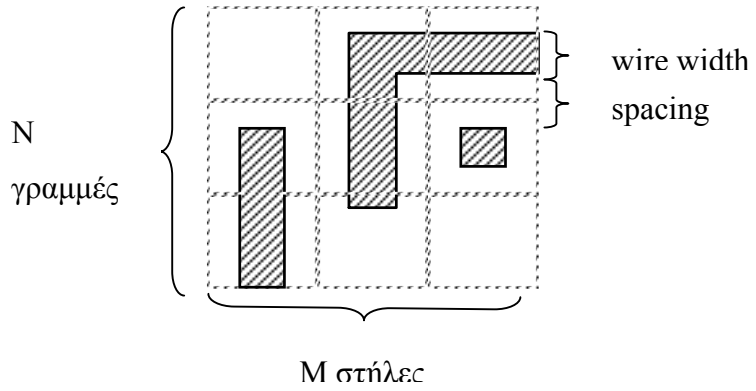
4.3.1. Εισαγωγή

Η δικιά μας προσέγγιση στο πρόβλημα βασίστηκε σε τεχνικές παρόμοιες με αυτές που υλοποιούνται στα [22], [23], [24] οι οποίες αν και είναι αρκετά παλιές εφαρμόζονται σήμερα με διάφορες παραλλαγές. Πριν προχωρήσουμε παρακάτω θα πρέπει να αναφέρουμε πως ως routing space θεωρήσαμε όλη την επιφάνεια του τσιπ και όχι μόνο αυτή που βρίσκεται μεταξύ των μπλοκ των λογικών πυλών (channels).

Αρχικά, κατασκευάσαμε στην επιφάνεια του τσιπ ένα πλέγμα (grid), πάνω στο οποίο τοποθετήσαμε αργότερα τις ηλεκτρικές διασυνδέσεις (wires) και τα υπόλοιπα συστατικά του κυκλώματος. Το grid αποτελείται από MxN τετράγωνα πλευράς ίσης

με το άθροισμα του πάχους των ηλεκτρικών καλωδίων και της απόστασης μεταξύ αυτών (Σχήμα 4.3). Δηλαδή

grid size = wire width + spacing



Σχήμα 4.3

Σκοπός μας είναι να ενώσουμε την έξοδο μιας λογικής πύλης με την είσοδο μιας άλλης με μια ηλεκτρική διασύνδεση (wire), το οποίο από εδώ και πέρα θα το ονομάζουμε γραμμή. Τόσο η είσοδος, όσο και η έξοδος μιας πύλης αντιστοιχούν σε τουλάχιστον ένα τετράγωνο του grid, αφού είναι από κατασκευής τους μεγαλύτερες από τις γραμμές. Συνεπώς, το πρόβλημα αντιστοιχίζεται στην αναζήτηση του ελάχιστου μονοπατιού από ένα τετράγωνο του grid σε ένα άλλο (η περίπτωση όπου η έξοδος μιας πύλης πηγαίνει σε πολλές εισόδους θα εξεταστεί παρακάτω). Η διαπίστωση αυτή αποτελεί και τη βασική ιδέα του αλγόριθμου μας.

4.3.2. Βασική Ιδέα

Αρχικά, θα εξετάσουμε την περίπτωση, όπου η έξοδος μιας πύλης συνδέεται με την είσοδο μιας άλλης. Επειδή η έξοδος και η είσοδος, όπως προαναφέρθηκε, αντιστοιχίζονται σε περισσότερα από ένα grid, βρίσκουμε πρώτα το κοντινότερο grid της εξόδου ως προς την είσοδο, το οποίο ονομάζουμε source και το κοντινότερο της εισόδου ως προς την έξοδο, το οποίο ονομάζουμε target. Η απόσταση του source και

του target είναι ίση με τη Manhattan απόσταση τους. Γενικά, η Manhattan απόσταση δύο σημείων (x_1,y_1) και (x_2,y_2) ορίζεται ως

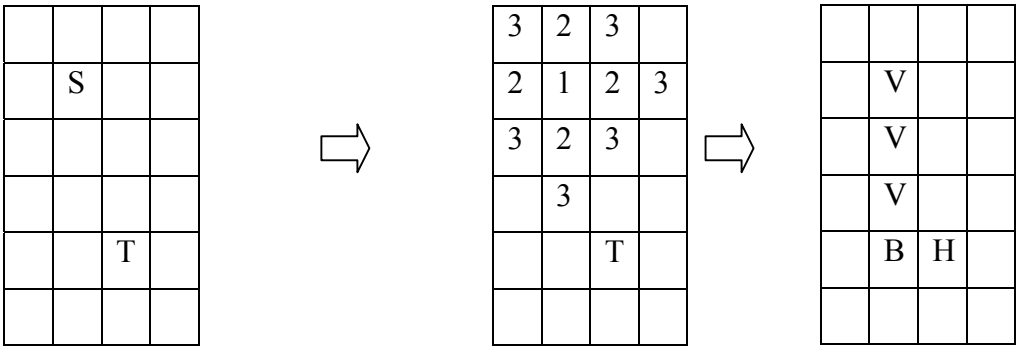
$$d = (y_2 - y_1) + (x_2 - x_1)$$

Έτσι λοιπόν, αναζητούμε το ελάχιστο μονοπάτι από το source στο target, δεδομένου ότι μπορούμε να κινούμαστε μόνο οριζόντια ή κάθετα πάνω στο grid.

Σε κάθε βήμα του αλγόριθμου, βρίσκουμε όλα τα τετράγωνα που έχουν απόσταση ίση με ένα από το τετράγωνο που ελέγχουμε και τα μαρκάρουμε ως ελεγμένα. Η επέκταση ενός τετραγώνου τη φορά δημιουργεί ένα σύνολο από μονοπάτια που αναζητούν το target, το οποίο θα το ονομάζουμε wavefront. Η διαδικασία αυτή επαναλαμβάνεται έως ότου φτάσουμε στο target. Μόλις επιλεχθεί το target διασχίζουμε το μονοπάτι προς τα πίσω και καταγράφουμε τα τετράγωνα ώστε να μη χρησιμοποιηθούν ξανά. Τέλος, διαγράφουμε όλα τα μονοπάτια που ανακαλύφθηκαν κατά τη διάρκεια της αναζήτησης και διατηρούμε μόνο το ελάχιστο, το οποίο αποτελεί και εμπόδιο (obstacle) για τις επόμενες διασυνδέσεις. Άρα τα βασικά βήματα του αλγόριθμου είναι:

1. Εκτελούμε BFS για να βρούμε όλα τα μονοπάτια από το source στο target
2. Διατρέχουμε το ελάχιστο μονοπάτι προς τα πίσω μαρκάροντας τα κελιά του ως χρησιμοποιημένα
3. Καθαρίζουμε το grid από όλα τα μονοπάτια που βρήκαμε εκτός από το ελάχιστο.

Σχηματικά ο αλγόριθμος έχει ως εξής:



Σχήμα 4.4 Εκτέλεση Αλγορίθμου Διασύνδεσης

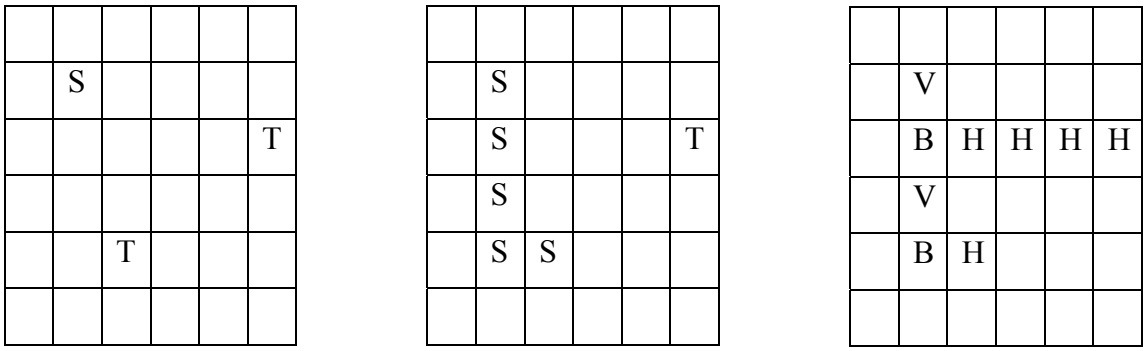
Στο Σχήμα 4.4 η έξοδος βρίσκεται στο τετράγωνο S (source) και η είσοδος στο τετράγωνο T (target). Στο δεύτερο πλέγμα του ίδιου Σχήματος παρατηρούμε την επέκταση των μονοπατιών που αναζητούν το target, ενώ στο τελευταίο βλέπουμε το συντομότερο μονοπάτι από το s στο t που βρήκε ο αλγόριθμός μας.

Πρέπει να αναφερθεί ότι κάθε τετράγωνο του grid μπορεί να το διατρέξει μια μόνο κάθετη και μια μόνο οριζόντια γραμμή. Για παράδειγμα, όταν ένα τετράγωνο καταγράφεται ως χρησιμοποιημένο από μια οριζόντια γραμμή (συμβολίζεται με H στο Σχήμα 4.4.), τότε το ίδιο τετράγωνο μπορεί να το διαπεράσει και μια κάθετη (συμβολίζεται με V στο Σχήμα 4.4), καθώς και το αντίστροφο. Αν από ένα τετράγωνο περνάει μια οριζόντια και μια κάθετη γραμμή (συμβολίζεται με B στο Σχήμα 4.4), τότε δε μπορεί να περάσει καμιά άλλη και προφανώς δε μπορεί να το χρησιμοποιήσει ο αλγόριθμος αργότερα.

Μέχρι τώρα, εξετάσαμε την περίπτωση όπου η έξοδος μιας λογικής πύλης οδηγείται στην είσοδο μιας άλλης. Στο σημείο αυτό, θα ελέγξουμε τη συμπεριφορά του αλγόριθμου όταν η έξοδος μιας πύλης γίνεται είσοδος πολλών πυλών. Στην κατάσταση αυτή έχουμε ένα source και πολλά targets (multi terminal net).

Η στρατηγική που ακολουθείται είναι η εξής: Ξεκινάμε από το source και εκτελούμε παρόμοια διαδικασία με την παραπάνω μέχρι να βρούμε το κοντινότερο target. Μόλις το target βρεθεί διατρέχουμε προς τα πίσω το ελάχιστο μονοπάτι και ονομάζουμε όλα τα τετράγωνα από τα οποία αποτελείται ως sources. Στη συνέχεια, εξαπλώνουμε με τον ίδιο τρόπο το σύνολο των sources ώσπου να βρούμε το επόμενο target και

διατρέχοντας προς τα πίσω το νέο ελάχιστο μονοπάτι μαρκάρουμε τα καινούργια τετράγωνα ως sources. Η διαδικασία αυτή επαναλαμβάνεται έως ότου ανακαλύψουμε όλα τα targets. Επακόλουθα, ορίζουμε όλα τα τετράγωνα που από αποτελούν τα ελάχιστα μονοπάτια σαν χρησιμοποιημένα και τέλος, καθαρίζουμε το gird από τα «άχρηστα» μονοπάτια που υπολόγισε ο αλγόριθμος. Η περίπτωση του multi terminal net φαίνεται στο Σχήμα 4.5



Σχήμα 4.5 Εκτέλεση Αλγορίθμου Διασύνδεσης στην Multi-terminal Διασύνδεση

Στο πρώτο πλέγμα του Σχήματος 4.5 παρατηρούμε μια έξοδο S, την οποία θέλουμε να την ενώσουμε με δύο εισόδους Τ. Στο δεύτερο υπολογίζουμε το μονοπάτι προς την κοντινότερη είσοδο και τέλος υπολογίζουμε όλα τα ελάχιστα μονοπάτια.

4.3.3. Επιτάχυνση του Maze Router

Το μεγάλο πρόβλημα του αλγόριθμου αυτού είναι η ταχύτητα του. Εξαιτίας του γεγονότος ότι είναι ένας greedy αλγόριθμος, πολλές φορές είναι αναγκασμένος να αναζητήσει όλο το grid για να βρει ένα μονοπάτι ελάχιστου μήκους. Για το λόγο αυτό είναι σημαντικό να βρεθούν τρόποι, ώστε ο maze router να γίνει γρηγορότερος. Στην κατεύθυνση αυτή βοηθούν τόσο ευριστικές συναρτήσεις, όσο και το wavefront.

4.3.4. Ευριστική Συνάρτηση

Όπως αναφέρθηκε προηγούμενα, σε κάθε βήμα επεκτείνουμε ένα τετράγωνο του grid προς 4 κατευθύνσεις. Αυτό έχει ως αποτέλεσμα να οδηγούμαστε σε πολλά μονοπάτια

που να μας απομακρύνουν από το target και μας κοστίζουν σε χρόνο και σε μνήμη (γιατί όλα αυτά πρέπει να αποθηκεύονται μέχρι να βρεθεί το ελάχιστο). Συνεπώς, για να αποφύγουμε αυτές τις ανεπιθύμητες καταστάσεις αντιστοιχίζουμε σε κάθε ένα τετράγωνο του grid μια πληροφορία, που σχετίζεται με την Manhattan απόσταση του συγκεκριμένου τετραγώνου από το target και την ονομάζουμε κόστος του τετραγώνου (cell). Στο κόστος αυτό προσθέτουμε και ένα παράγοντα penalty, στην περίπτωση που απομακρυνόμαστε από το target.

$$\text{cost} = \begin{cases} \text{Manhattan } (\text{cell}_{i,j} \text{ , target}), \text{ αν πλησιάζουμε στο target} \\ \text{Manhattan } (\text{cell}_{i,j} \text{ , target}) + \text{penalty}, \text{ αν απομακρυνόμαστε από target} \end{cases}$$

Αυτά συμβαίνουν όταν έχουμε ένα source και ένα target. Στην περίπτωση του multi terminal net η ευριστική συνάρτηση που χρησιμοποιούμε είναι διαφορετική. Το κόστος κάθε cell είναι ίσο με το άθροισμα των Manhattan αποστάσεων του από όλα τα targets. Σε αυτό προσθέτουμε για δεύτερη φορά την ελάχιστη Manhattan απόσταση του συγκεκριμένου cell από τα targets πολλαπλασιασμένη κατά 0.25. Αυτό γίνεται για να ενθαρρύνουμε την αναζήτηση προς τη κατεύθυνση που βρίσκεται το κοντινότερο target. Συνοψίζοντας, αν έχουμε t targets το κόστος ενός τετραγώνου $\text{cell}_{i,j}$ θα είναι ίσο με αυτό της εξίσωσης 4.1

$$\text{cost} = \sum_{k \le t} \text{dist}(\text{cell}_{i,j}, \text{target}_k), k \ge 0 \qquad \text{Εξ } 4.1$$

Έτσι, σε κάθε βήμα δεν επεκτείνουμε όλα τα τετράγωνα προς όλες τις κατευθύνσεις παρά μόνο αυτά που έχουν το μικρότερο κόστος.

4.3.5. Wavefront

Το wavefront είναι μια δομή που αποτελείται από τα cells εκείνα που πρόκειται να επεκταθούν για να ανακαλύψουμε νέα μονοπάτια. Με τον τρόπο λοιπόν αυτό, δεν είναι ανάγκη να ελέγχουμε κάθε φορά ποιο τετράγωνο θα επεκτείνουμε, αλλά επιλέγουμε μόνο αυτά που ανήκουν στο wavefront. Τα cells, που υπάρχουν σε αυτό

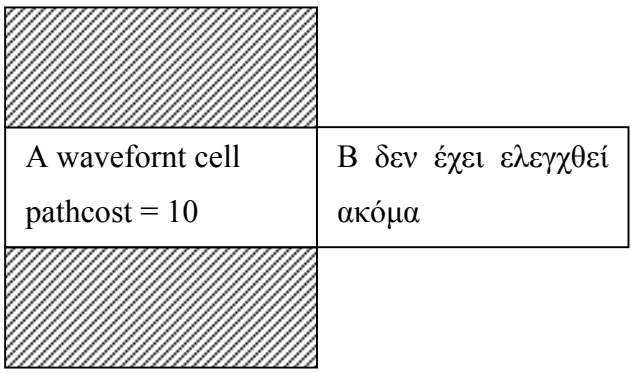
είναι ταξινομημένα με βάση το κόστος του μονοπατιού από το source στο συγκεκριμένο cell. Αυτό είναι ίσο με το άθροισμα του κόστους του μονοπατιού πριν φτάσουμε στο τετράγωνο που εξετάζουμε συν το κόστος του τετραγώνου. Για κάθε βήμα που γίνεται και ελέγχεται και ένας νέος γείτονας προσθέτουμε μια μονάδα στο κόστος του μονοπατιού. Δηλαδή,

$$pathcost = pathcost(predecessor) + 1 + cellcost \qquad \text{Εξ 4.2}$$

Επιγραμματικά, το wavefront αυξάνεται με βάση τον παρακάτω αλγόριθμο

1. Εξάγουμε από το wavefront, το cell που έχει το μικρότερο κόστος μονοπατιού, πχ C
2. Ελέγχουμε τους γείτονες του που δεν έχουν ελεγχθεί ακόμα., οι οποίοι μπορεί να είναι το πολύ τέσσερις.
3. Υπολογίζουμε τα νέα κόστη μονοπατιού για κάθε γείτονα
4. Προσθέτουμε τους γείτονες στο wavefront
5. Διαγράφουμε το C από το wavefront
6. Επαναλαμβάνουμε την παραπάνω διαδικασία

Στη συνέχεια φαίνεται ένα παράδειγμα, που εξηγεί την παραπάνω διαδικασία.

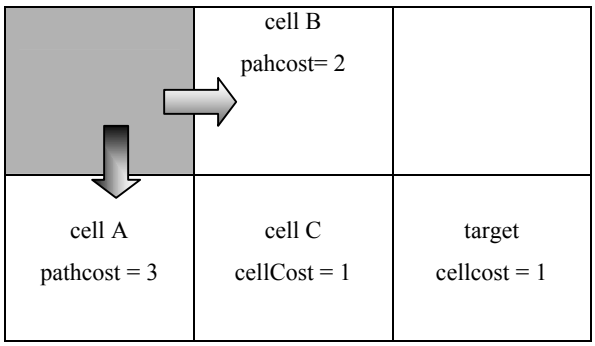


Σχήμα 4.6

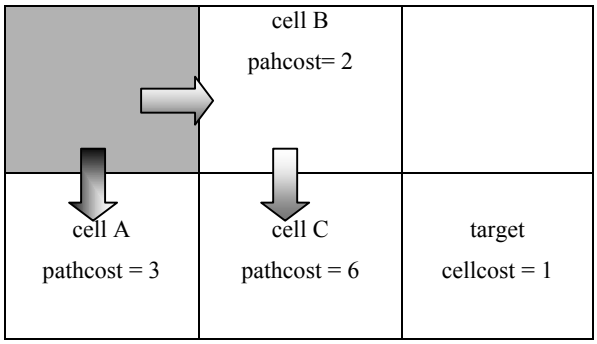
Επίσης, σκοπός μας στη διαδικασία του routing είναι να μειώσουμε όσο γίνεται περισσότερο τον αριθμό των στροφών (bends) των γραμμών διασύνδεσης. Για το λόγο αυτό εισάγουμε έναν όρο penalty bend στον υπολογισμό του κόστους του μονοπατιού, ο οποίος είναι στην περίπτωση μας είναι ίσος με δύο. Έτσι λοιπόν, η εξίσωση 4.2 γίνεται:

$$pathcost = pathcost(predecessor) + 1 + cellcost + penalty\ bend\ (2) \qquad \text{Εξ 4.2}$$

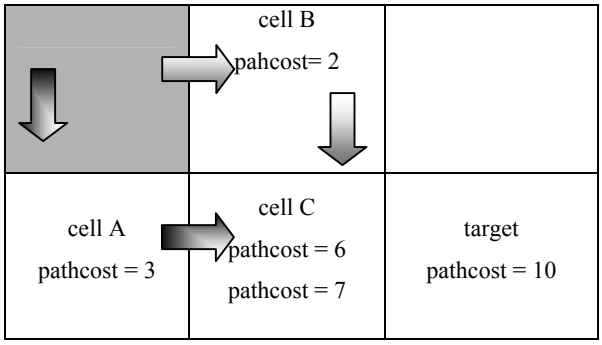
Συνεπώς, όταν ο αλγόριθμος μας έχει να επιλέξει μεταξύ 2 τετραγώνων με το ίδιο κόστος θα επιλέγει εκείνο, στο οποίο δε θα γίνεται στροφή. Αυτό όμως δημιούργησε την ανάγκη να αποθηκεύουμε στο wavefront πολλές φορές το ίδιο τετράγωνο με διαφορετικά κόστη. Για να γίνει περισσότερο κατανοητό ακολουθεί το παρακάτω παράδειγμα



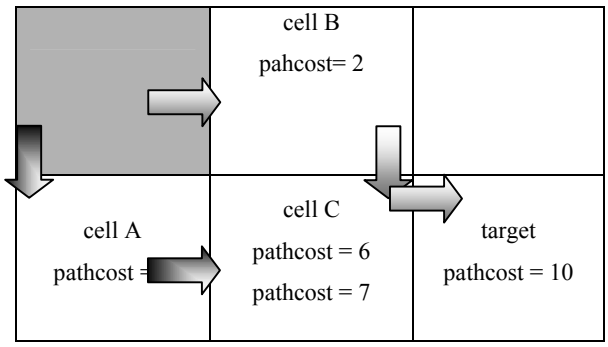
- Τα cells A και B βρίσκονται στο wavefront
- Στο επόμενο βήμα επεκτείνεται το cell B



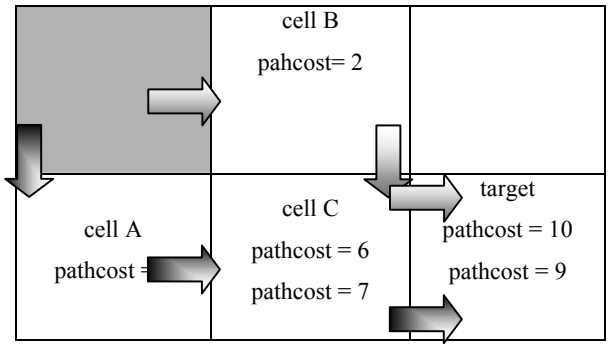
- Εξάγουμε το B από το wavefront
- Εξετάζεται το cell C και εισάγεται στο wavefront με pathcost = 2 + 1 + 1 + 2 = 6, λόγω του ότι κάνει στροφή η γραμμή
- Διαγράφεται το B από το wavefront



- Εξάγουμε το A από το wavefront
- Εξετάζεται το cell C και εισάγεται στο wavefront με pathcost = 3 + 1 + 1 + 2 = 7, λόγω του ότι η γραμμή κάνει στροφή
- Διαγράφεται το A από το wavefront



- Εξάγουμε το C με pathcost = 6 από το wavefront
- Εξετάζεται το target και εισάγεται στο wavefront με pathcost = 6 + 1 + 1 + 2 = 10, λόγω του ότι η γραμμή κάνει στροφή
- Διαγράφεται το C με pathcost = 6 από το wavefront



- Εξάγουμε το C με pathcost =7 από το wavefront
- Εξετάζεται το target και εισάγεται στο wavefront με pathcost = 7 + 1 + 1 = 9
- Διαγράφεται το C με pathcost = 6 από το wavefront

Παρατηρούμε ότι αν δεν εισάγουμε το τετράγωνο C με κόστος 7 στο wavefront, θα οδηγηθούμε σε λάθος αποτέλεσμα, δηλαδή θα υπολογίσουμε ένα μονοπάτι το οποίο δεν θα είναι το ελάχιστο. Αυτό συμβαίνει γιατί το κόστος του μονοπατιού είναι δυναμικό και εξαρτάται κάθε φορά από τη θέση του cell και των γειτόνων του με αποτέλεσμα να έχει πολλά διαφορετικά pathcosts, ανάλογα με τη κατεύθυνση της

γραμμής διασύνδεσης που το επισκέπτεται. Γίνεται λοιπόν κατανοητό, πώς αν δεν αποθηκευτούν όλες αυτές οι πιθανές καταστάσεις μπορεί να μη φτάσουμε σε σωστό αποτέλεσμα.

Τέλος ο αλγόριθμος τερματίζει όταν φτάσουμε στο target και όλα τα cells που βρίσκονται στο wavefront έχουν pathcost μεγαλύτερο από αυτό του target.

4.3.6. Ο αλγόριθμος

Αφού αναλύσαμε και εξηγήσαμε το πώς δουλεύει ο αλγόριθμος διασύνδεσης Maze router, θα αναπτύξουμε παρακάτω τον ψευδοκώδικα του.

```
1.  source  = FindSource (source_x, source_y)
2.  target = FindTarget(target_x, target_y)
3.  wavefront_structure = {source}
4.  while(have not hit target){
5.      if(wavefront = = empty)
6.          quit – no path found
7.      C = get lowest cost cell on wavefront_structure
8.          mark C as reached
9.      if( C = = target){
10.         if(pathcost( C ) < minimum pathcost cell on on wavefront_structure){
11.         backtrace path in grid
12.         cleanup
13.         return – we found a path
14.     }
15. }
16. for each (unreachable neighbor N of cell C){
17.     compute cost to reach it = pathcost of C + cellCost of C + 1+ penalty bend
18.     mark N cell in grid with predecessor direction back to cell C from N
19.     add this cell N to wavefront
20. }
21. delete cell C from wavefront
```

ΚΕΦΑΛΑΙΟ 5. ΘΕΜΑΤΑ ΥΛΟΠΟΙΗΣΗΣ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

5.1. Θέματα Υλοποίησης

Η παρούσα εργασία υλοποιήθηκε σε γλώσσα προγραμματισμού C με τη βοήθεια της openGL, σε λειτουργικό σύστημα SUSE LINUX 9.3. Οι κύριοι λόγοι που επιλέξαμε τα παραπάνω είναι οι εξής:

- Κρίθηκε ότι δεν είναι αναγκαία η χρήση αντικειμενοστρέφειας στην εργασία, οπότε αποκλείστηκε αυτόματα και η χρήση μιας αντικειμενοστραφής γλώσσας.
- Οι μαθηματικοί υπολογισμοί που γίνονται κατά την υλοποίηση των αλγορίθμων είναι σχετικά απλοί και η C μπορεί να τους χειριστεί με ιδιαίτερη επιτυχία.
- Ο κύριος σκοπός της εργασίας μας είναι να εξάγουμε ποιοτικά συμπεράσματα σχετικά με την τοποθέτηση και τη διασύνδεση των λογικών πυλών ενός κυκλώματος και η δημιουργία ενός υποτυπώδους interface, το οποίο θα οπτικοποίησει το κύκλωμα. Σε αυτό το κομμάτι μας βοηθάει αρκετά η openGL.
- Η C είναι μια εξαιρετικά γρήγορη γλώσσα.
- Το LINUX είναι open source λογισμικό

Το πρόγραμμα μας παίρνει ως είσοδο το λογικό κύκλωμα, δηλαδή τις λογικές πύλες από τις οποίες αποτελείται και τον τρόπο διασύνδεσης αυτών (netlist) και μια βιβλιοθήκη όπου υπάρχουν τα χαρακτηριστικά των λογικών πυλών. Τέτοια είναι το μέγεθος της πύλης, το πλήθος των εισόδων και των εξόδων της καθώς και οι θέσεις αυτών πάνω στην πύλη. Το λογικό κύκλωμα είναι αποθηκευμένο σε ένα xnf αρχείο, ενώ η βιβλιοθήκη σε ένα txt. Αυτό που κάνουμε εμείς είναι να διαβάζουμε τα αρχεία, να αποθηκεύουμε τα στοιχεία που περιέχουν σε συγκεκριμένες δομές δεδομένων και στη συνέχεια να εκτελούμε τους προαναφερθέντες αλγόριθμους για να εξάγουμε τα συμπεράσματα μας. Οι δομές δεδομένων που χρησιμοποιήσαμε είναι κυρίως η ουρά και ο σωρός. Σε ελάχιστες περιπτώσεις χρησιμοποιήθηκαν πίνακες. Αυτό έγινε για να εξοικονομήσουμε μνήμη αλλά και να επιταχύνουμε ορισμένες διαδικασίες.

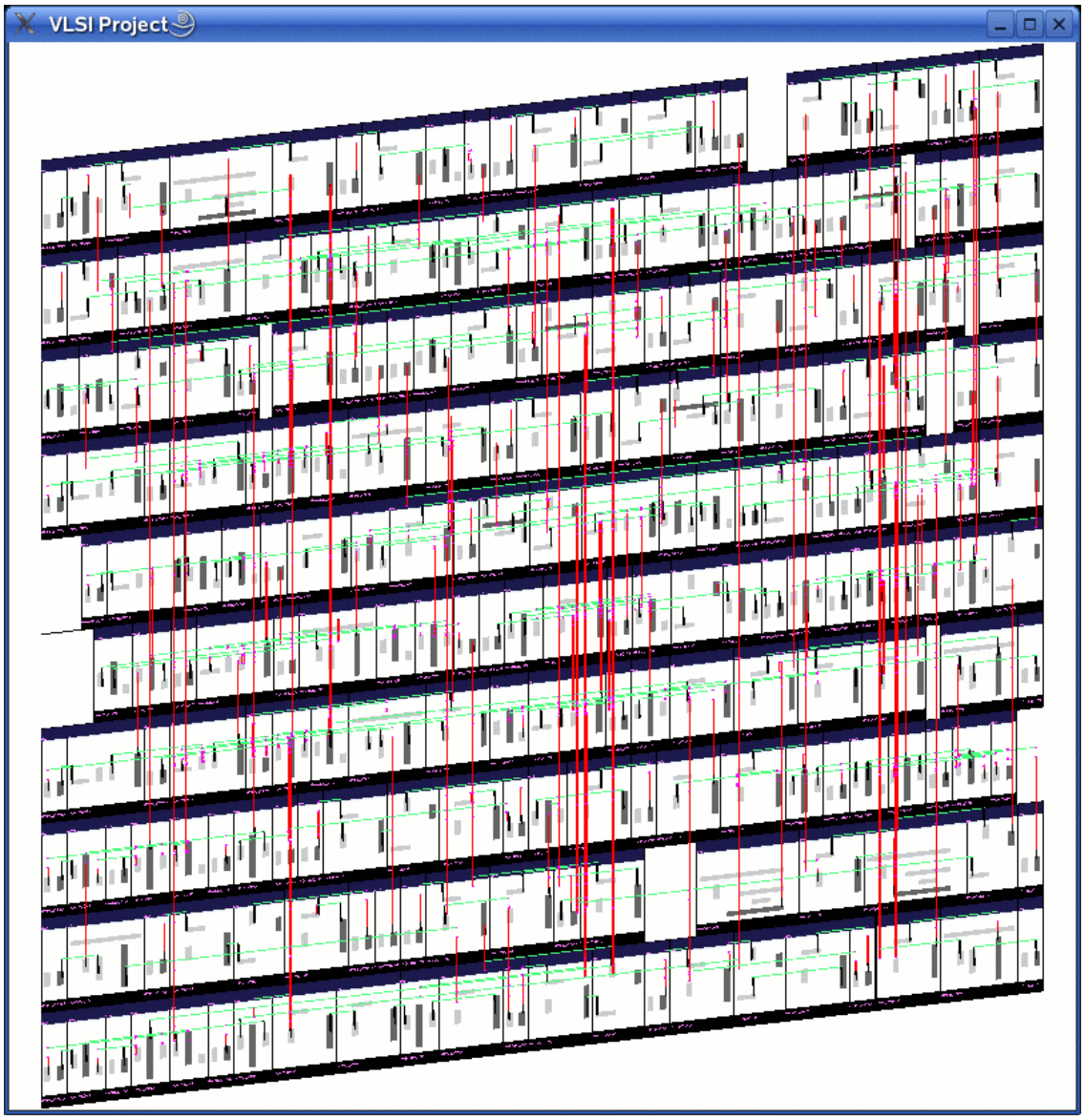
Τέλος, η υλοποίηση μας χρησιμοποιεί το πακέτο geosteiner 3.1 [25] για τον υπολογισμό του βέλτιστου μήκους των διασυνδέσεων. Αυτό χρησιμεύει μόνο στη εξαγωγή κάποιων χρήσιμων πειραματικών αποτελεσμάτων και δεν έχει καμιά άλλη πρακτική σημασία στον κώδικα.

5.1.1. Interface Εργασίας

Όπως αναφέρθηκε, το interface της εργασίας είναι πολύ απλό και βασικό σκοπό έχει να οπτικοποιήσει το λογικό κύκλωμα ώστε να είναι πιο κατανοητό στο χρήστη. Τόσο η επιφάνεια του τσιπ, όσο και οι λογικές πύλες σχεδιάζονται ως ορθογώνια παραλληλεπίπεδα με αμελητέο πλάτος. Με τον ίδιο ακριβώς τρόπο σχεδιάζονται και οι γραμμές διασύνδεσης. Αυτές δε βρίσκονται όλες στην ίδια απόσταση από την επιφάνεια του τσιπ, αλλά εξαρτάται από το είδος της γραμμής (κάθετη ή οριζόντια) και από το επίπεδο στο οποίο βρίσκεται.

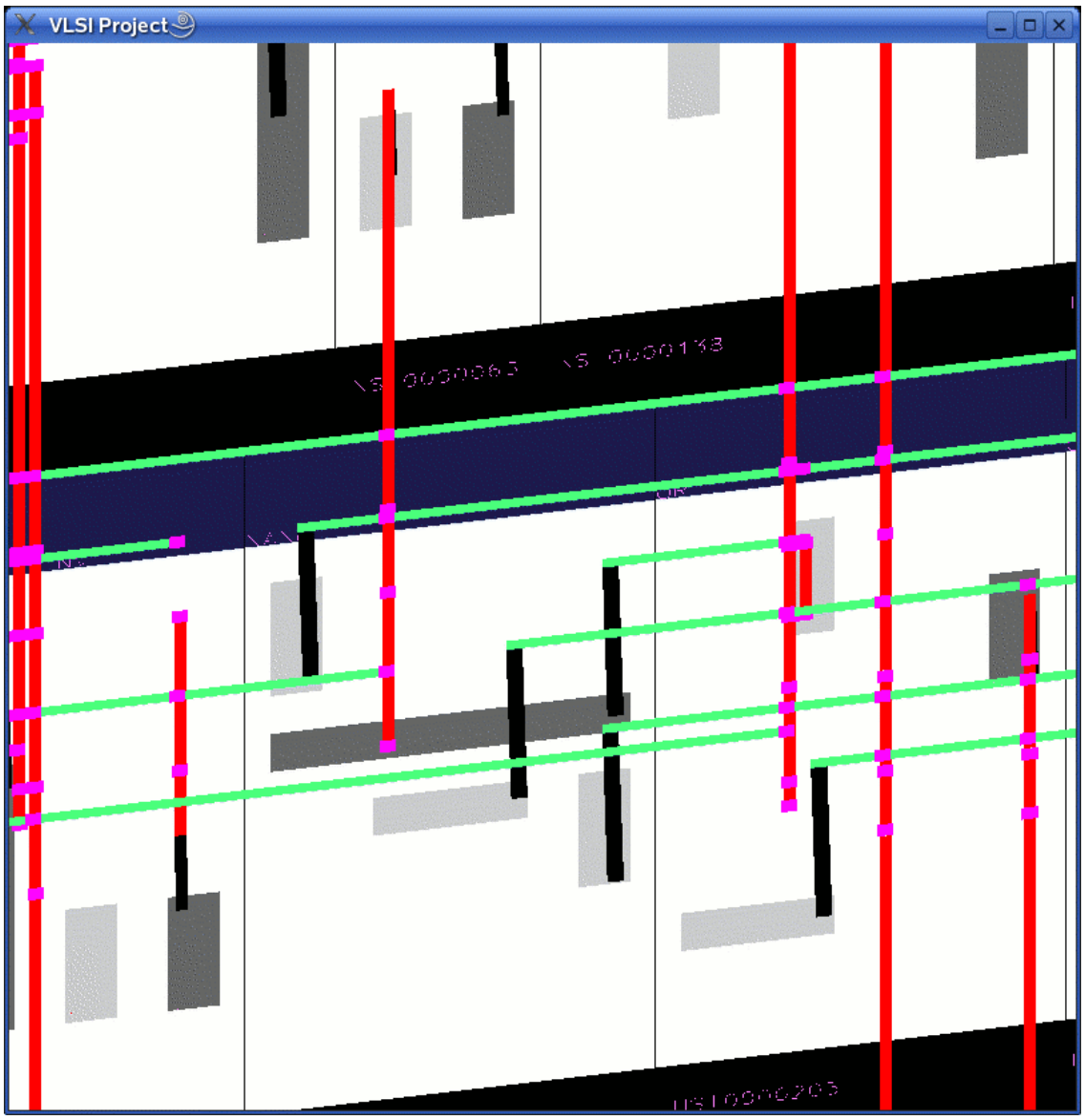
Το πρόγραμμα εκτελείται από γραμμή εντολών και ο χρήστης μπορεί να επιλέξει την τοποθέτηση των πυλών είτε με την απλή μέθοδο που χρησιμοποιεί τον αλγόριθμο dfs, είτε με τη μέθοδο που υπολογίζει πυκνούς υπογράφους. Αφού εκτελεστεί το πρόγραμμα, στο παράθυρο της openGL που ανοίγει, εμφανίζεται το λογικό κύκλωμα το οποίο ο χρήστης μπορεί να το περιστρέψει με τη βοήθεια του ποντικιού γύρω και από τους τρεις άξονες, καθώς επίσης και να το μετακινήσει προς όλες τις

κατευθύνσεις. Στα Σχήματα 5.1 και 5.2 φαίνεται η υλοποίηση του ίδιου λογικό κυκλώματος, με τη διαφορά ότι στο Σχήμα 5.2 έχουμε μετακινήσει το κύκλωμα πάνω στον z-άξονα.



Σχήμα 5.1 Τοποθέτηση και Διασύνδεση ενός Λογικού Κυκλώματος

Στο Σχήμα 5.1 τα παραλληλόγραμμα που φαίνονται είναι οι λογικές πύλες και τα μικρότερα που υπάρχουν μέσα σε αυτά είναι οι είσοδοι και οι έξοδοι των πυλών. Πάνω από αυτά παρατηρούνται ορισμένες γραμμές οι οποίες είναι οι γραμμές διασύνδεσης.



Σχήμα 5.2 Το Κύκλωμα του Σχήματος 5.1 όπως Έχει Μετακινηθεί στον z-Άξονα.

Στο Σχήμα 5.2 παρατηρούμε τις εισόδους και τις εξόδους των λογικών πυλών, οι οποίες είναι συνδεδεμένες μεταξύ τους. Επίσης, οι γραμμές που αναπαριστούν τις διασυνδέσεις είναι με διαφορετικό χρώμα και σε διαφορετικό επίπεδο. Έτσι οι οριζόντιες έχουν πράσινο χρώμα και βρίσκονται ψηλότερα από τις κάθετες, που έχουν κόκκινο χρώμα.

5.2. Εκτέλεση Πειραμάτων

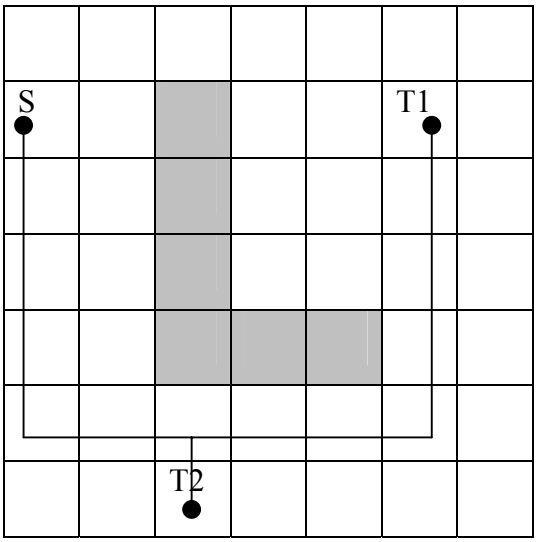
Οι αλγόριθμοι δοκιμάστηκαν σε έξι διαφορετικά αρχεία, καθένα από τα οποία αναπαριστά και ένα λογικό κύκλωμα. Τα στοιχεία που μας ενδιέφεραν κυρίως ήταν

- Η σύγκριση του μήκους των διασυνδέσεων μεταξύ των δύο μεθόδων που χρησιμοποιήθηκαν. Στο σημείο αυτό πρέπει να αναφέρουμε ορισμένα πράγματα για τις μεθόδους ώστε να γίνει πιο εύκολη η κατανόηση των πειραμάτων. Η πρώτη μέθοδος που χρησιμοποιήθηκε είναι αυτή που εκτελεί μια απλή dfs διάσχιση στο γράφο και τοποθετεί τις πύλες με τα ανάλογα κριτήρια. Η δεύτερη είναι αυτή που υπολογίζει πυκνούς υπογράφους ανάλογα με τα βάρη που δίνουμε στους κόμβους και τις ακμές. Έτσι λοιπόν, την τελευταία τη δοκιμάσαμε για διαφορετικά ζεύγη βαρών των ακμών και των κόμβων.

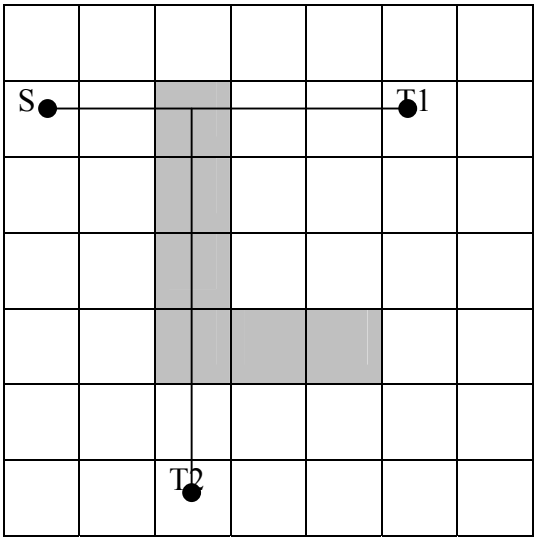
- Η απόκλιση από τη βέλτιστη λύση. Για κάθε διασύνδεση που υλοποιεί ο αλγόριθμός μας υπολογίζουμε δύο κόστη. Το πρώτο είναι το πραγματικό κόστος της διασύνδεσης (μήκος της διασύνδεσης), που υλοποιείται από τη μέθοδο μας, λαμβάνοντας υπό όψη και τα διάφορα εμπόδια (block cells) που πιθανόν να υπάρχουν. Το δεύτερο είναι ίσο με το μήκος της διασύνδεσης που υπολογίζεται με τη βοήθεια των manhattan steiner – trees χωρίς να λαμβάνουμε υπόψη μας τα πιθανά εμπόδια. Στο Σχήμα 5.3 φαίνεται μια πιθανή διασύνδεση τριών πυλών, των S, T1 και T2 που πραγματοποιεί ο αλγόριθμός μας και έχει μήκος 14. Τα τετράγωνα που είναι χρωματισμένα γκρι είναι block cells και δε μπορεί να περάσει από πάνω τους άλλη γραμμή διασύνδεσης. Στο Σχήμα 5.4 φαίνεται η διασύνδεση με το βέλτιστο μήκος, που είναι ίσο με 10 και πραγματοποιείται με τη βοήθεια των manhattan steiner – trees.

- Ο χρόνος εκτέλεσης τους.

- Η επιφάνεια που καταλαμβάνουν



Σχήμα 5.3 Διασύνδεση Τριών Πυλών Μεταξύ τους με Συνολικό Μήκος 14



Σχήμα 5.4 Διασύνδεση Τριών Πυλών Μεταξύ τους Μήκους 10 με Manhattan Steiner – Tree Χωρίς να Λαμβάνονται Υπό Όψη τα Block Cells

5.2.1. File c432o

Το αρχείο αυτό περιέχει το μικρότερο κύκλωμα στο οποίο δοκιμάστηκαν οι αλγόριθμοι και τα βασικά χαρακτηριστικά του είναι τα εξής:

Πλήθος Πυλών: 160

Πλήθος Διασυνδέσεων: 255

Συνολική Επιφάνεια των Πυλών: 2486,272

Στο κύκλωμα αυτό, όπως και σε όλα τα υπόλοιπα που ακολουθούν, δοκιμάστηκε τόσο ο αλγόριθμος τοποθέτησης που χρησιμοποιεί μια απλή διάσχιση dfs όσο και αυτός που χρησιμοποιεί πυκνούς υπογράφους. Ο τελευταίος δοκιμάστηκε για διαφορετικά ζεύγη βαρών κορυφών και ακμών. Στον Πίνακα 5.1 στην πρώτη γραμμή παρουσιάζεται ο λόγος του βάρους των κόμβων προς το βάρος των ακμών (από δω και στο εξής θα τον ονομάζουμε *λόγο βάρους*) που χρησιμοποιήθηκε σε κάθε πείραμα και στη δεύτερη γραμμή το πλήθος των πυκνών υπογράφων που βρέθηκαν. Έτσι λοιπόν όταν ο λόγος αυτός είναι 1.5, ο οποίος προκύπτει αν επιλέξουμε το βάρος των κόμβων να είναι 15 και το βάρος των ακμών να είναι 10 τότε ο αλγόριθμός μας θα υπολογίσει δέκα πυκνούς υπογράφους που υπάρχουν στον αρχικό γράφο.

Πίνακας 5.1 Πλήθος Πυκνών Υπογράφων του Κυκλώματος c432o

dfs	1.5	1.555	1.6	1.666	1.7	1.72	1.777
0	10	7	7	5	1	1	1

Επίσης, οι δύο αλγόριθμοι δοκιμάστηκαν σε εφτά διαφορετικές επιφάνειες του chip, ως προς το μέγεθός τους. Στον Πίνακα 5.2 φαίνεται το μέγεθος των επιφανειών.

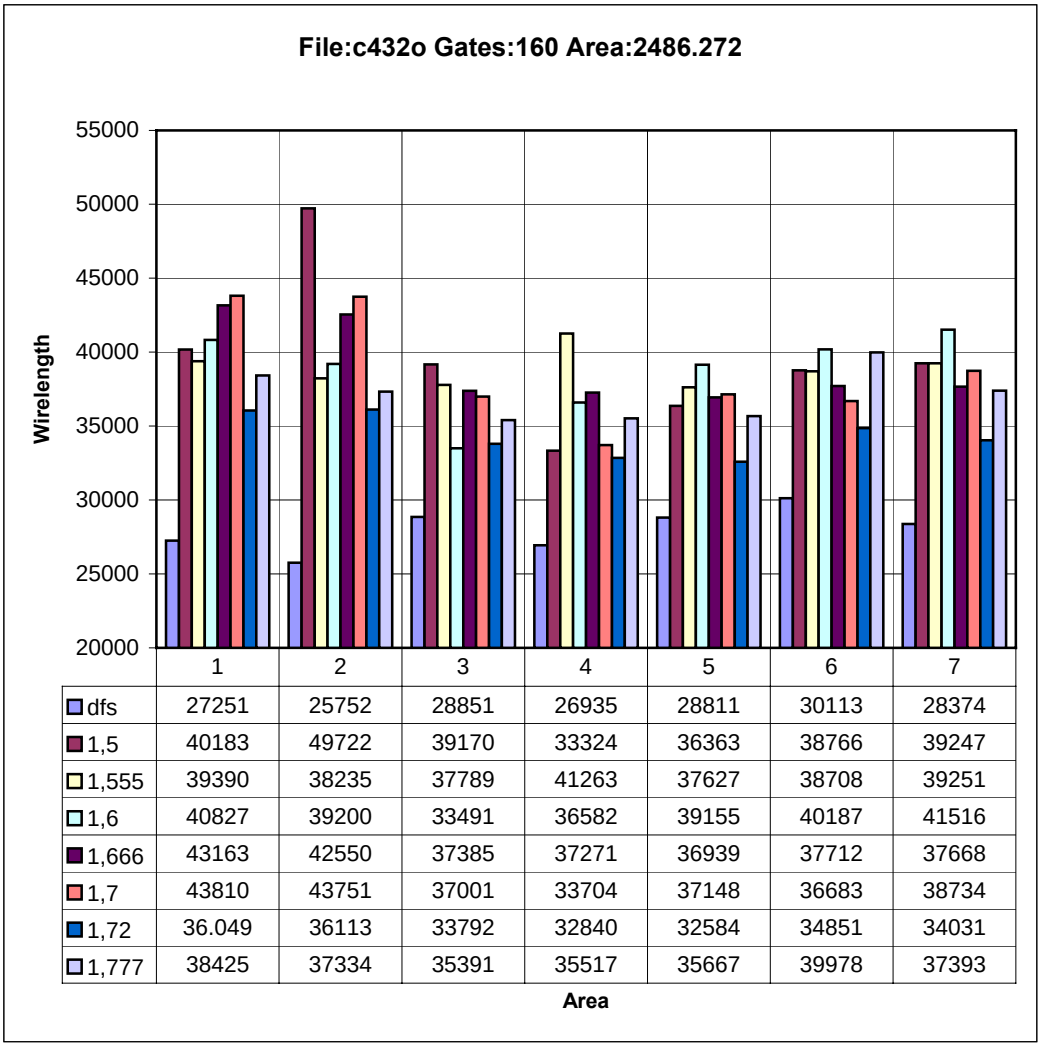
Πίνακας 5.2 Μέγεθος των Επιφανειών για το κύκλωμα c432o

Επιφ. 1	Επιφ. 2	Επιφ. 3	Επιφ. 4	Επιφ. 5	Επιφ. 6	Επιφ. 7
5760	4800	3200	2912	2880	2816	2624

Στη συνέχεια στον Πίνακα 5.3 παρουσιάζεται το μήκος των διασυνδέσεων που χρησιμοποιήθηκαν σε κάθε πείραμα. Στον άξονα των y βρίσκεται το συνολικό μήκος

των διασυνδέσεων και στον άξονα των x φαίνονται οι διαφορετικές επιφάνειες του chip. Πρέπει να σημειωθεί ότι οι επιφάνειες είναι ταξινομημένες κατά φθίνουσα σειρά δηλαδή η επιφάνεια 1 είναι η μεγαλύτερη και η επιφάνεια 7 είναι η μικρότερη. Στον Πίνακα 5.4 παρατίθεται ο λόγος του συνολικού μήκους των διασυνδέσεων που επιτυγχάνει ο αλγόριθμός μας προς το συνολικό βέλτιστο μήκος των διασυνδέσεων (απόκλιση από τη βέλτιστη λύση). Όμοια με παραπάνω στον άξονα των y βρίσκεται ο συγκεκριμένος λόγος, ενώ στον άξονα των x η επιφάνεια του chip.

Πίνακας 5.3 Συνολικό Μήκος Διασυνδέσεων Συναρτήσει της Επιφάνειας του Chip του Κυκλώματος c432o

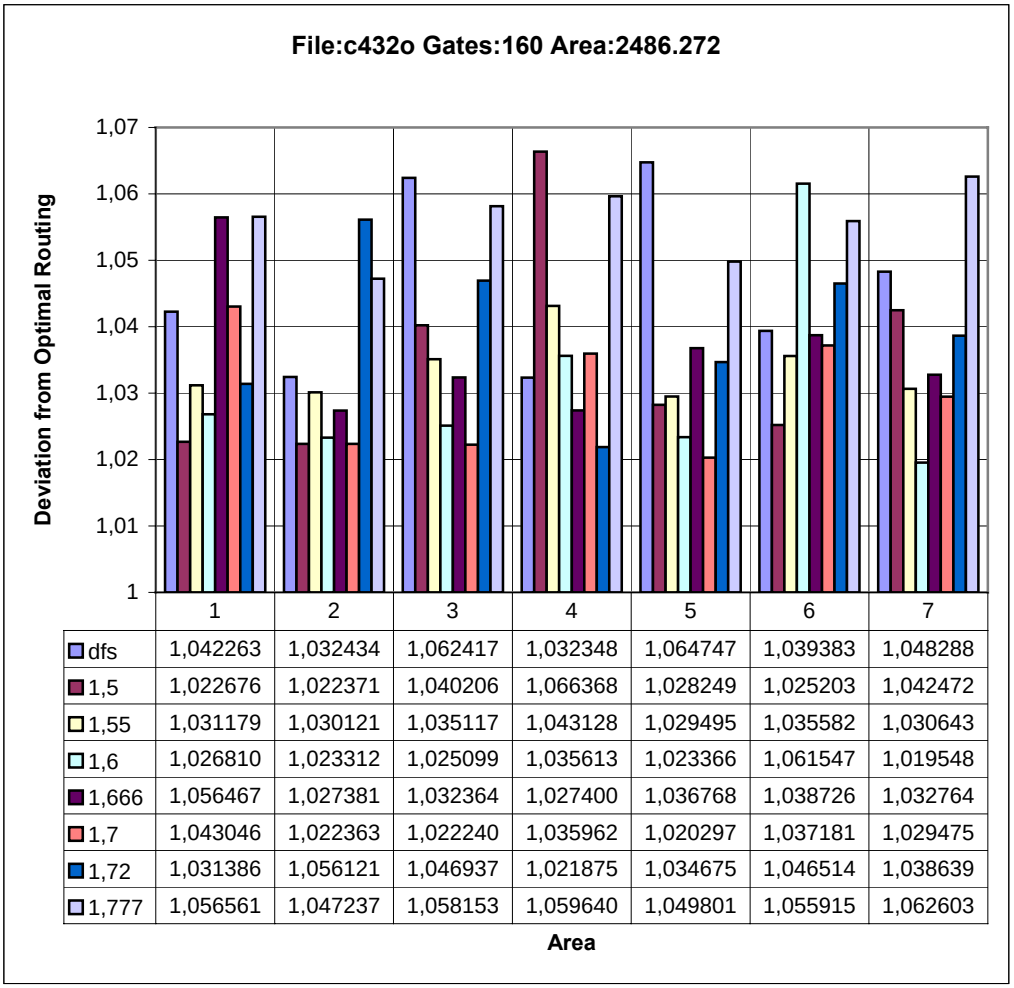


	1	2	3	4	5	6	7
dfs	27251	25752	28851	26935	28811	30113	28374
1,5	40183	49722	39170	33324	36363	38766	39247
1,555	39390	38235	37789	41263	37627	38708	39251
1,6	40827	39200	33491	36582	39155	40187	41516
1,666	43163	42550	37385	37271	36939	37712	37668
1,7	43810	43751	37001	33704	37148	36683	38734
1,72	36.049	36113	33792	32840	32584	34851	34031
1,777	38425	37334	35391	35517	35667	39978	37393

Από το γράφημα παρατηρούμε ότι στο συγκεκριμένο λογικό κύκλωμα η πρώτη προσέγγιση μας είχε πολύ καλύτερα αποτελέσματα από ότι είχε ο αλγόριθμος που

υπολογίζει πυκνούς υπογράφους ανεξάρτητα από το βάρος των ακμών και των κόμβων που χρησιμοποιήθηκε.

Πίνακας 5.4 Απόκλιση Από τη Βέλτιστη Λύση Συναρτήσει της Επιφάνειας του Chip του Κυκλώματος c432o



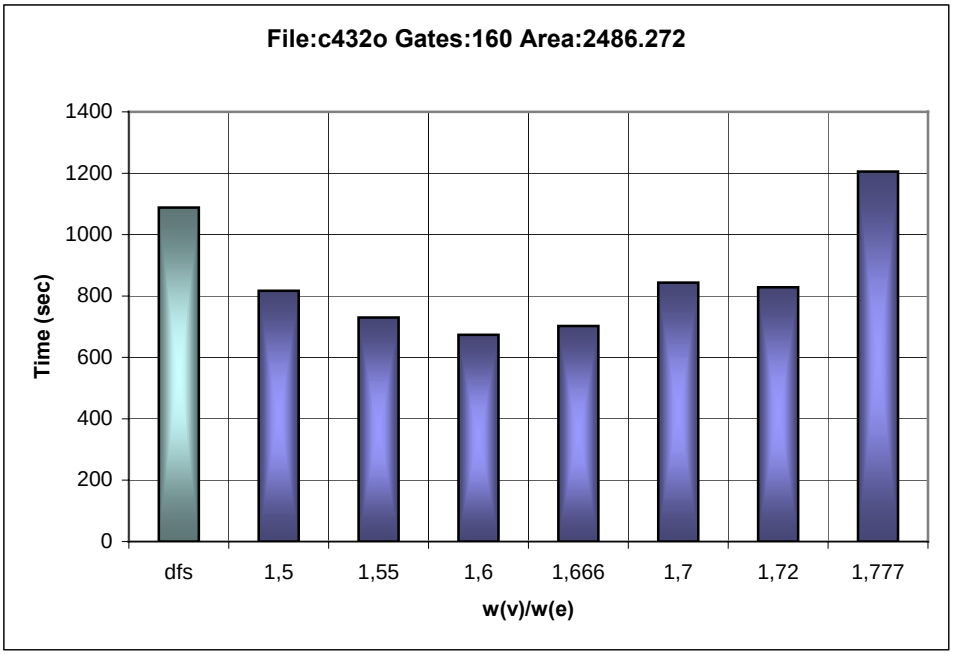
	1	2	3	4	5	6	7
dfs	1,042263	1,032434	1,062417	1,032348	1,064747	1,039383	1,048288
1,5	1,022676	1,022371	1,040206	1,066368	1,028249	1,025203	1,042472
1,55	1,031179	1,030121	1,035117	1,043128	1,029495	1,035582	1,030643
1,6	1,026810	1,023312	1,025099	1,035613	1,023366	1,061547	1,019548
1,666	1,056467	1,027381	1,032364	1,027400	1,036768	1,038726	1,032764
1,7	1,043046	1,022363	1,022240	1,035962	1,020297	1,037181	1,029475
1,72	1,031386	1,056121	1,046937	1,021875	1,034675	1,046514	1,038639
1,777	1,056561	1,047237	1,058153	1,059640	1,049801	1,055915	1,062603

Στον Πίνακα 5.4 παρατηρούμε ότι κατά μέσο όρο η απόκλιση από τη βέλτιστη λύση στην περίπτωση του αλγόριθμου που κάνει μια απλή dfs διάσχιση είναι μεγαλύτερη από αυτήν που υπολογίζονται πυκνοί υπογράφοι. Αυτό σημαίνει ότι η δεύτερη προσέγγιση μπορεί να χρειάζεται περισσότερο μήκος ηλεκτρικών καλωδίων αλλά οι λύση που μας δίνει είναι πιο κοντά στη βέλτιστη.

Επίσης, στον Πίνακα 5.5 παρατηρούμε το μέσο χρόνο που χρειάστηκε να γίνει η διασύνδεση των λογικών πυλών του κυκλώματος όταν αυτές τοποθετήθηκαν και στις εφτά διαφορετικές επιφάνειες του chip με τους 2 αλγόριθμους. Έτσι λοιπόν, ο y-

άξονας αναπαριστά το χρόνο σε δευτερόλεπτα, ενώ ο x-άξονας αναπαριστά τις διαφορετικές εκτελέσεις του αλγορίθμων. Για παράδειγμα, η μπάρα που έχει ετικέτα «dfs» αναφέρεται στο χρόνο που χρειάστηκε να γίνει η διασύνδεση όταν η τοποθέτηση έγινε με τον αλγόριθμο που εκτελεί μια απλή dfs διάσχιση, η μπάρα με ετικέτα «1.5» αναπαριστά το χρόνο που χρειάστηκε να γίνει η διασύνδεση όταν η τοποθέτηση έγινε με τον αλγόριθμο που υπολογίζει πυκνούς υπογράφους δεδομένου ότι ο λόγος βάρους είναι ίσος με 1.5 κοκ.

Πίνακας 5.5 Λόγος Βάρους του Αλγόρίθμου Συναρτήσει του Χρόνου για το Κύκλωμα c432o



Παρατηρούμε ότι όταν οι πύλες τοποθετούνται με τον ο αλγόριθμο που υπολογίζει πυκνούς υπογράφους ο χρόνος διασύνδεσης τους είναι μικρότερος από ότι όταν η τοποθέτηση γίνεται με την άλλη προσέγγιση.

Τέλος, και οι δύο αλγόριθμοι κατάφεραν να τοποθετήσουν τις λογικές πύλες σε επιφάνεια ίση με 2624 όταν η ελάχιστη που θα μπορούσαν να τοποθετηθούν είναι 2486. Αυτό μεταφράζεται στο ότι η μικρότερη επιφάνεια που μπορούν να χρησιμοποιήσουν οι προσεγγίσεις μας είναι περίπου 5.5% μεγαλύτερη της ελάχιστης.

5.2.2. File s420.1

Τα χαρακτηριστικά του κυκλώματος που περιγράφεται στο συγκεκριμένο αρχείο είναι

Πλήθος Πυλών:	218
Πλήθος Διασυνδέσεων:	269
Συνολική Επιφάνεια Πυλών:	3104,766

Στον Πίνακα 5.6, στην πρώτη γραμμή παρουσιάζεται ο λόγος βάρους που χρησιμοποιήθηκε σε κάθε πείραμα και στη δεύτερη γραμμή το πλήθος των πυκνών υπογράφων που βρέθηκαν.

Πίνακας 5.6 Πλήθος Πυκνών Υπογράφων του Κυκλώματος s420.1

Dfs	1.2	1.222	1.25	1.2727	1.3	1.333	1.375
0	14	11	7	2	2	4	1

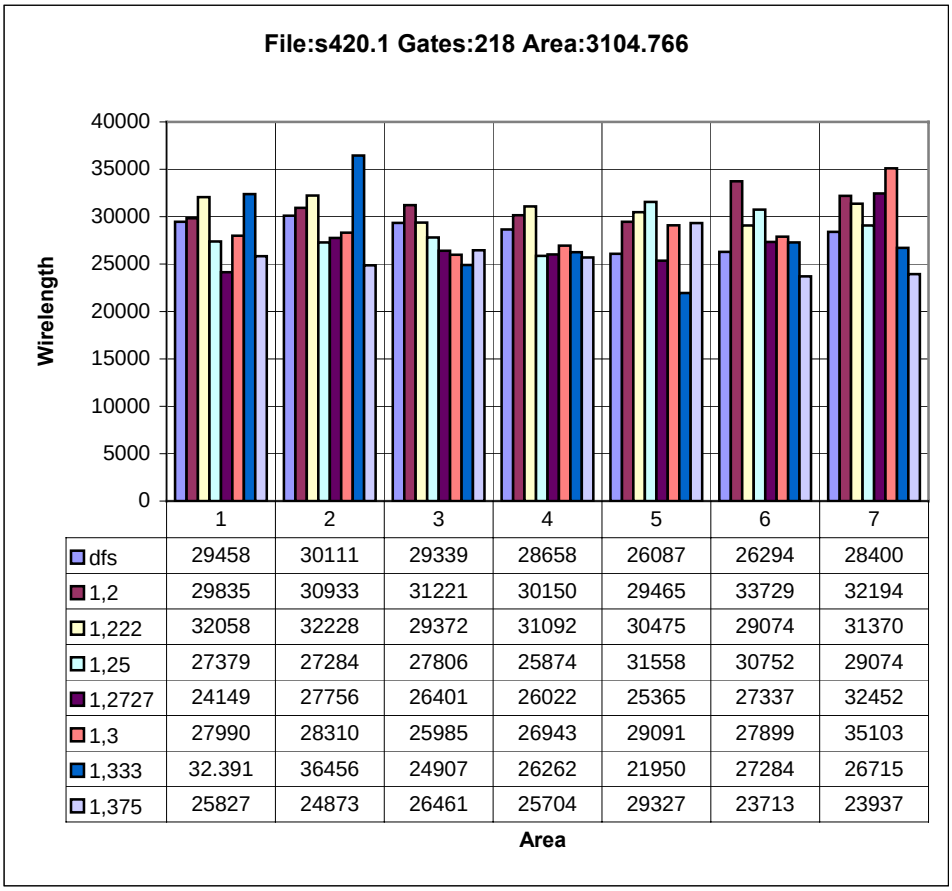
Στον Πίνακα 5.7 φαίνονται οι διαφορετικές επιφάνειες του chip στις οποίες δοκιμάστηκαν οι αλγόριθμοι.

Πίνακας 5.7 Μέγεθος των Επιφανειών για το κύκλωμα s420.1

Επιφ. 1	Επιφ. 2	Επιφ. 3	Επιφ. 4	Επιφ. 5	Επιφ. 6	Επιφ. 7
7680	7040	5760	5280	3840	3840	3520

Στη συνέχεια στον Πίνακα 5.8 παρουσιάζεται το μήκος των διασυνδέσεων που χρησιμοποιήθηκαν σε κάθε πείραμα.. Στον άξονα των y βρίσκεται το συνολικό μήκος των διασυνδέσεων και στον άξονα των x φαίνεται η επιφάνεια του chip. Στον Πίνακα 5.9 παρατίθεται ο λόγος του συνολικού μήκους των διασυνδέσεων που επιτυγχάνει ο αλγόριθμός μας προς το συνολικό βέλτιστο μήκος των διασυνδέσεων. Όμοια με παραπάνω στον άξονα των y βρίσκεται ο συγκεκριμένος λόγος, ενώ στον άξονα των x η επιφάνεια του chip.

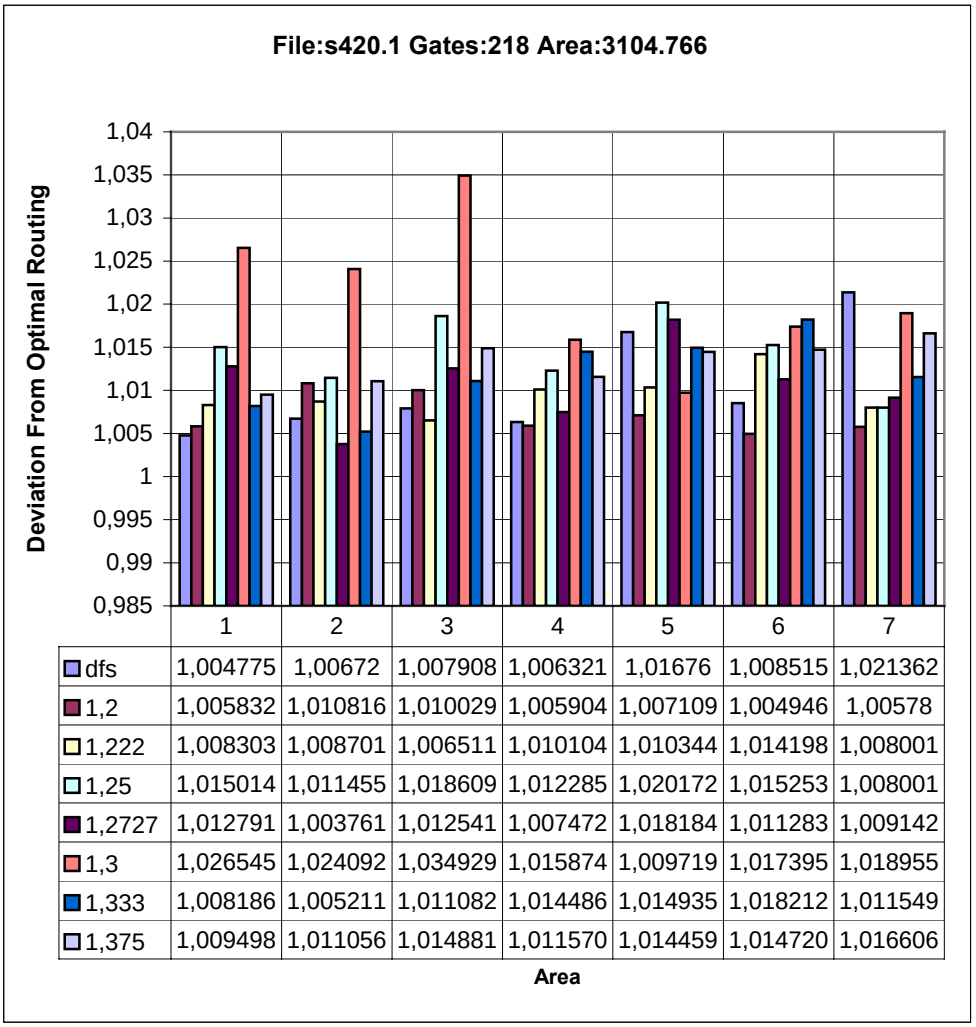
Πίνακας 5.8 Συνολικό Μήκος Διασυνδέσεων Συναρτήσει της Επιφάνειας του Chip για το Κύκλωμα s420.1



	1	2	3	4	5	6	7
dfs	29458	30111	29339	28658	26087	26294	28400
1,2	29835	30933	31221	30150	29465	33729	32194
1,222	32058	32228	29372	31092	30475	29074	31370
1,25	27379	27284	27806	25874	31558	30752	29074
1,2727	24149	27756	26401	26022	25365	27337	32452
1,3	27990	28310	25985	26943	29091	27899	35103
1,333	32.391	36456	24907	26262	21950	27284	26715
1,375	25827	24873	26461	25704	29327	23713	23937

Αντίθετα με το προηγούμενο κύκλωμα, εδώ το μήκος των διασυνδέσεων στις δύο τελευταίες περιπτώσεις, όπου ο λόγος βάρους είναι 1.333 και 1.375 είναι μικρότερο κατά μέσο όρο από ότι στις άλλες περιπτώσεις. Αξίζει να σημειωθεί ότι στην περίπτωση που ο λόγος είναι ίσος με 1.375 υπάρχει μόνο ένας πυκνός υπογράφος, που συνιστά και το μικρότερο πλήθος στο συγκεκριμένο πείραμα.

Πίνακας 5.9 Απόκλιση Από τη Βέλτιστη Λύση Συναρτήσει της Επιφάνειας του Chip για το Κύκλωμα s420.1

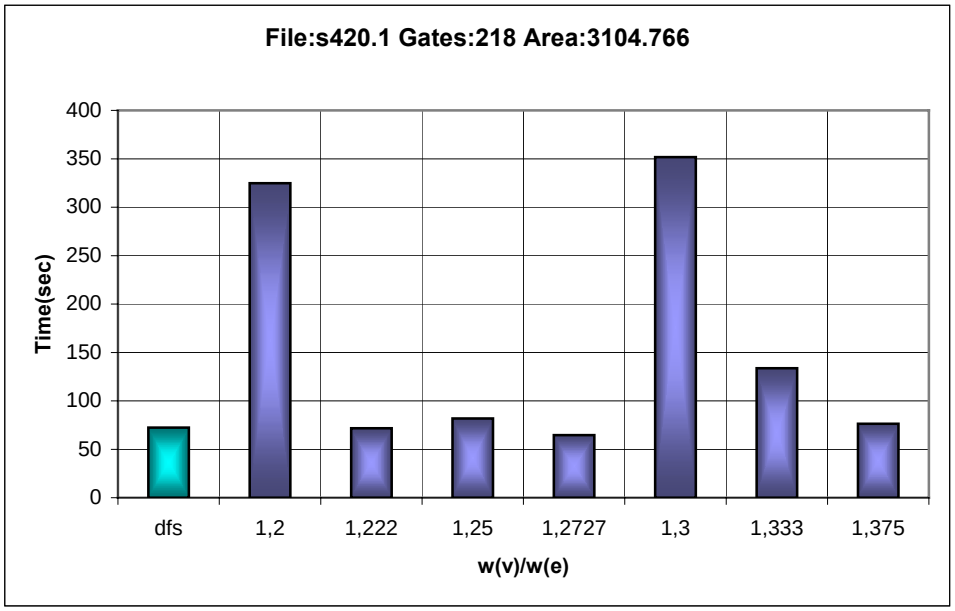


	1	2	3	4	5	6	7
dfs	1,004775	1,00672	1,007908	1,006321	1,01676	1,008515	1,021362
1,2	1,005832	1,010816	1,010029	1,005904	1,007109	1,004946	1,00578
1,222	1,008303	1,008701	1,006511	1,010104	1,010344	1,014198	1,008001
1,25	1,015014	1,011455	1,018609	1,012285	1,020172	1,015253	1,008001
1,2727	1,012791	1,003761	1,012541	1,007472	1,018184	1,011283	1,009142
1,3	1,026545	1,024092	1,034929	1,015874	1,009719	1,017395	1,018955
1,333	1,008186	1,005211	1,011082	1,014486	1,014935	1,018212	1,011549
1,375	1,009498	1,011056	1,014881	1,011570	1,014459	1,014720	1,016606

Στο πείραμα αυτό παρατηρούμε ότι μόνο στη δεύτερη και στην τρίτη περίπτωση (ο λόγος βάρους είναι ίσος με 1.2 και 1.222 αντίστοιχα) η απίκλιση από τη βέλτιστη λύση είναι μικρότερη κατά μέσο από την περίπτωση του αλγόριθμου της απλής dfs διάσχισης.Εν τούτοις, αυτή σε όλες τις περιπτώσεις είναι σχετικά μικρή και πουθενά δε ξεπερνά το 3.5%.

Ακόλουθα, παρατίθεται ο Πίνακας 5.10 όπου κανείς μπορεί να παρατηρήσει το χρόνο που χρειάστηκε να γίνει διασύνδεση των λογικών πυλών για τις διαφορετικές τοποθετήσεις. Εδώ, σε αντίθεση με παραπάνω η τοποθέτηση που επιτυγχάνεται με τον αλγόριθμο που εκτελεί μια απλή dfs διάσχιση έχει ως αποτέλεσμα μια συγκριτικά γρήγορη διασύνδεση των λογικών πυλών.

Πίνακας 5.10 Λόγος Βάρους του Αλγόρίθμου Συναρτήσει του Χρόνου για το Κύκλωμα s420.1



Τέλος, η μικρότερη επιφάνεια που μπορεί να χρησιμοποιηθεί είναι περίπου 3% μεγαλύτερη από την ελάχιστη.

5.2.3. File s713

Τα χαρακτηριστικά του κυκλώματος που περιγράφεται στο παρών αρχείο είναι τα παρακάτω

Πλήθος Πυλών:	393
Πλήθος Διασυνδέσεων:	537
Συνολική Επιφάνεια των Πυλών:	4911.091

Σε πλήρη αντιστοιχία με τον Πίνακα 5.1 και Πίνακα 5.6 παραθέτουμε τον Πίνακα 5.11

Πίνακας 5.11 Πλήθος Πυκνών Υπογράφων του Κυκλώματος s713

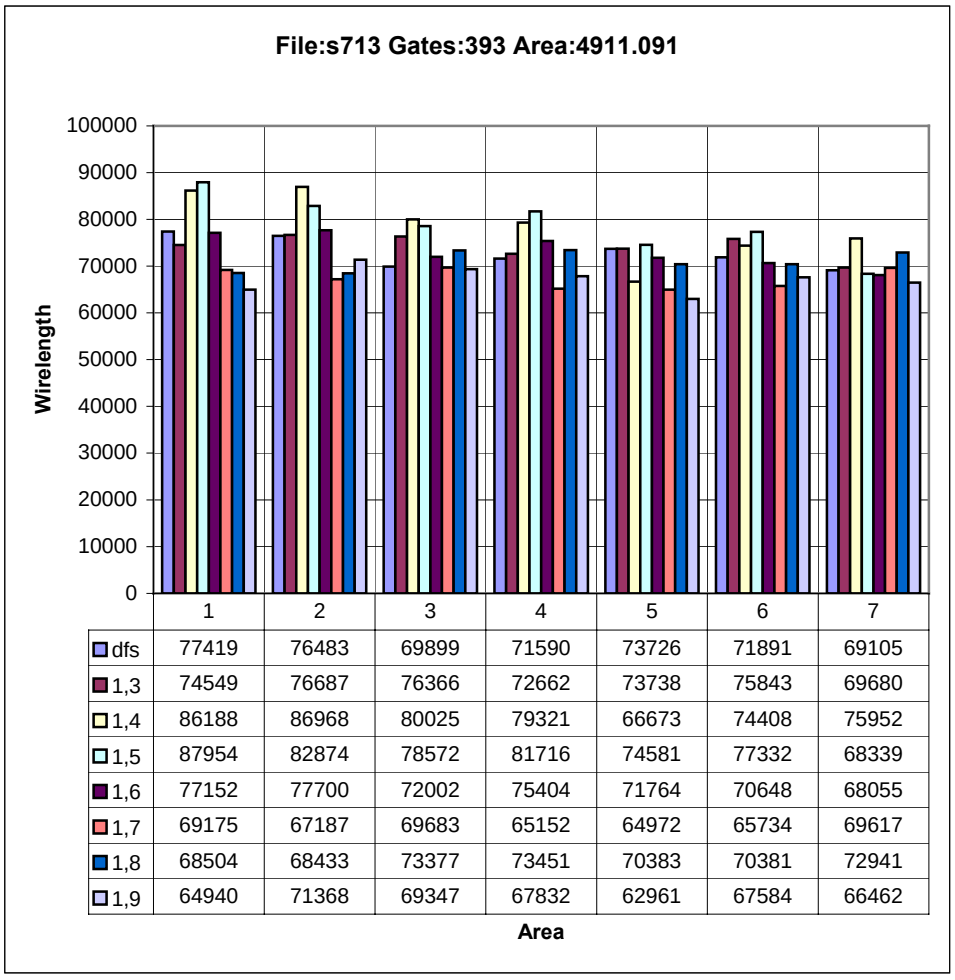
Dfs	1.3	1.4	1.5	1.6	1.7	1.8	1.9
0	45	32	22	15	6	3	1

Πίνακας 5.12 Μέγεθος των Επιφανειών για το κύκλωμα s713

Επιφ. 1	Επιφ. 2	Επιφ. 3	Επιφ. 4	Επιφ. 5	Επιφ. 6	Επιφ. 7
10400	9600	8320	7680	6240	5760	5683

Στον Πίνακα 5.13 παρατηρούμε πως στις τρεις τελευταίες περιπτώσεις το μήκος των διασυνδέσεων κατά μέσο όρο και το πλήθος των υπογράφων που υπολογίζονται είναι μικρότερα από ότι στις άλλες. Πράγματι, σύμφωνα με τον Πίνακα 5.11, όταν ο λόγος βάρους είναι ίσος με 1.7 το πλήθος των υπογράφων είναι ίσο με 6, όταν είναι ίσος με 1.8 είναι 3, ενώ όταν είναι ίσος με 1.9 είναι 1 ο υπογράφος που υπολογίζεται από τον αλγόριθμο.

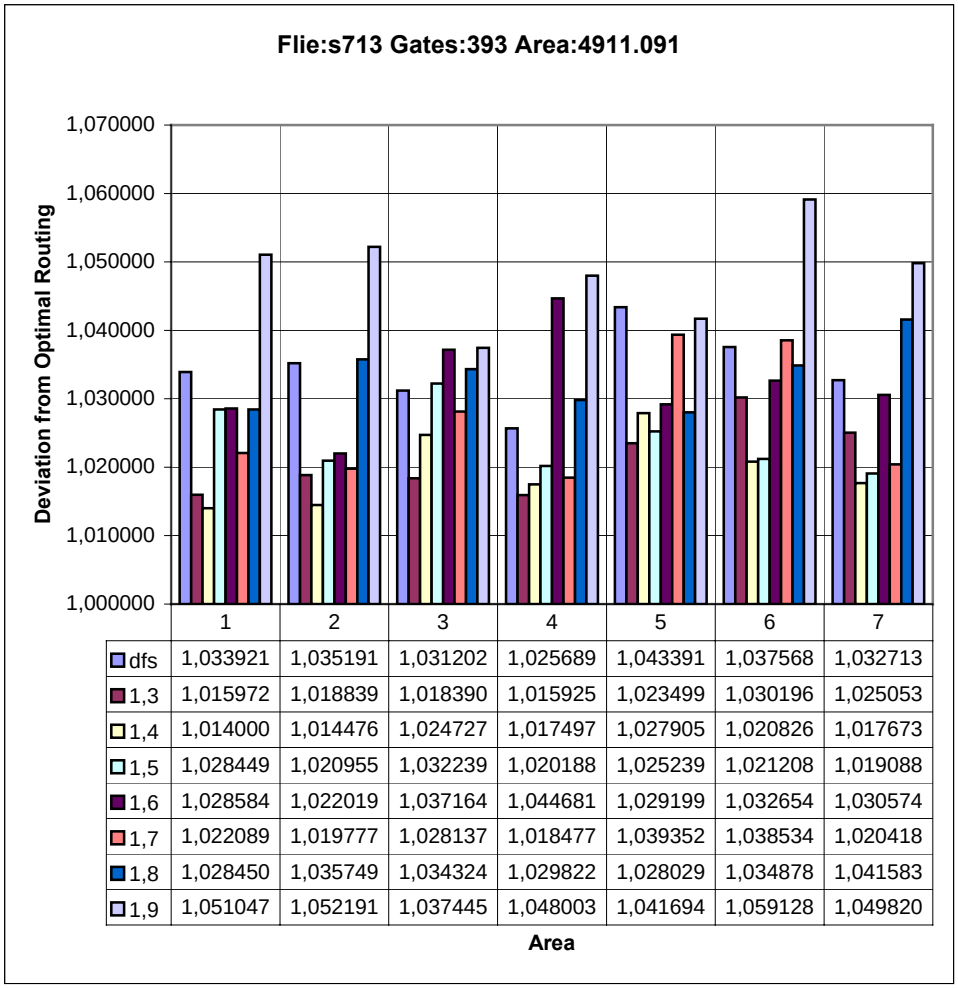
Πίνακας 5.13 Συνολικό Μήκος Διασυνδέσεων Συναρτήσει της Επιφάνειας του Chip για το Κύκλωμα s713



	1	2	3	4	5	6	7
dfs	77419	76483	69899	71590	73726	71891	69105
1,3	74549	76687	76366	72662	73738	75843	69680
1,4	86188	86968	80025	79321	66673	74408	75952
1,5	87954	82874	78572	81716	74581	77332	68339
1,6	77152	77700	72002	75404	71764	70648	68055
1,7	69175	67187	69683	65152	64972	65734	69617
1,8	68504	68433	73377	73451	70383	70381	72941
1,9	64940	71368	69347	67832	62961	67584	66462

Αντίστοιχα με τους Πίνακες 5.3 και 5.8 παρουσιάζουμε τον Πίνακα 5.14, όπου φαίνεται η απόκλιση από τη βέλτιστη λύση για το κύκλωμα s713.

Πίνακας 5.14 Απόκλιση Από τη Βέλτιστη Λύση Συναρτήσει της Επιφάνειας του Chip για το Κύκλωμα s713

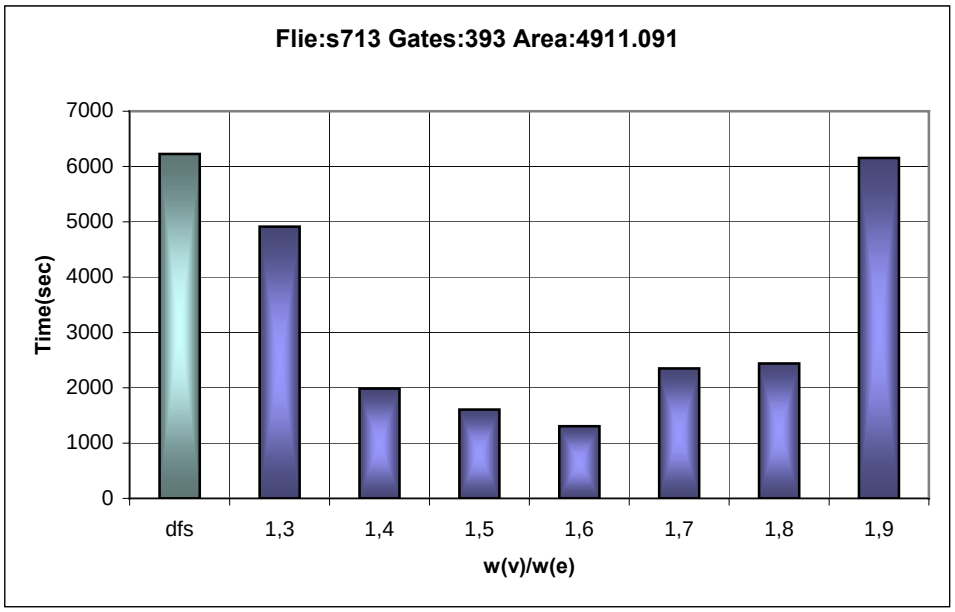


	1	2	3	4	5	6	7
dfs	1,033921	1,035191	1,031202	1,025689	1,043391	1,037568	1,032713
1,3	1,015972	1,018839	1,018390	1,015925	1,023499	1,030196	1,025053
1,4	1,014000	1,014476	1,024727	1,017497	1,027905	1,020826	1,017673
1,5	1,028449	1,020955	1,032239	1,020188	1,025239	1,021208	1,019088
1,6	1,028584	1,022019	1,037164	1,044681	1,029199	1,032654	1,030574
1,7	1,022089	1,019777	1,028137	1,018477	1,039352	1,038534	1,020418
1,8	1,028450	1,035749	1,034324	1,029822	1,028029	1,034878	1,041583
1,9	1,051047	1,052191	1,037445	1,048003	1,041694	1,059128	1,049820

Στο πείραμα αυτό, μόνο στην τελευταία περίπτωση όπου ο λόγος βάρους είναι ίσος με 1.9, η απόκλιση από τη βέλτιστη λύση είναι χειρότερη κατά μέσο όρο από την αντίστοιχη που προκύπτει από τον αλγόριθμο που εκτελεί μια απλή dfs διάσχιση του γράφου. Και εδώ η απόκλιση όλων των περιπτώσεων είναι σχετικά χαμηλή και δεν ξεπερνά το 6% της καλύτερης περίπτωσης.

Στη συνέχεια, βλέπουμε το λόγο βάρους συναρτήσει του χρόνου στον Πίνακα 5.15, όπου η τοποθέτηση των λογικών πυλών με τη βοήθεια του αλγόριθμου που εκτελεί μια απλή dfs διάσχιση έχει ως αποτέλεσμα την πιο αργή διασύνδεση αυτών.

Πίνακας 5.15 Λόγος Βάρους του Αλγόρίθμου Συναρτήσει του Χρόνου για το Κύκλωμα s713



Τέλος, η μικρότερη επιφάνεια που μπορεί να χρησιμοποιηθεί για την τοποθέτηση των λογικών πυλών και με τους δύο αλγόριθμους είναι 1.6% μεγαλύτερη της ελάχιστης.

5.2.4. File s953

Τα βασικά χαρακτηριστικά του κυκλώματος που περιγράφεται στο συγκεκριμένο αρχείο είναι

Πλήθος Πυλών:	395
Πλήθος Διασυνδέσεων:	628
Συνολική Επιφάνεια των Πυλών:	5356.76

Στον Πίνακα 5.16 φαίνεται το πλήθος των υπογράφων που υπολογίζει ο αλγόριθμος σε σχέση με το λόγο βάρους.

Πίνακας 5.16 Πλήθος Πυκνών Υπογράφων του Κυκλώματος s953

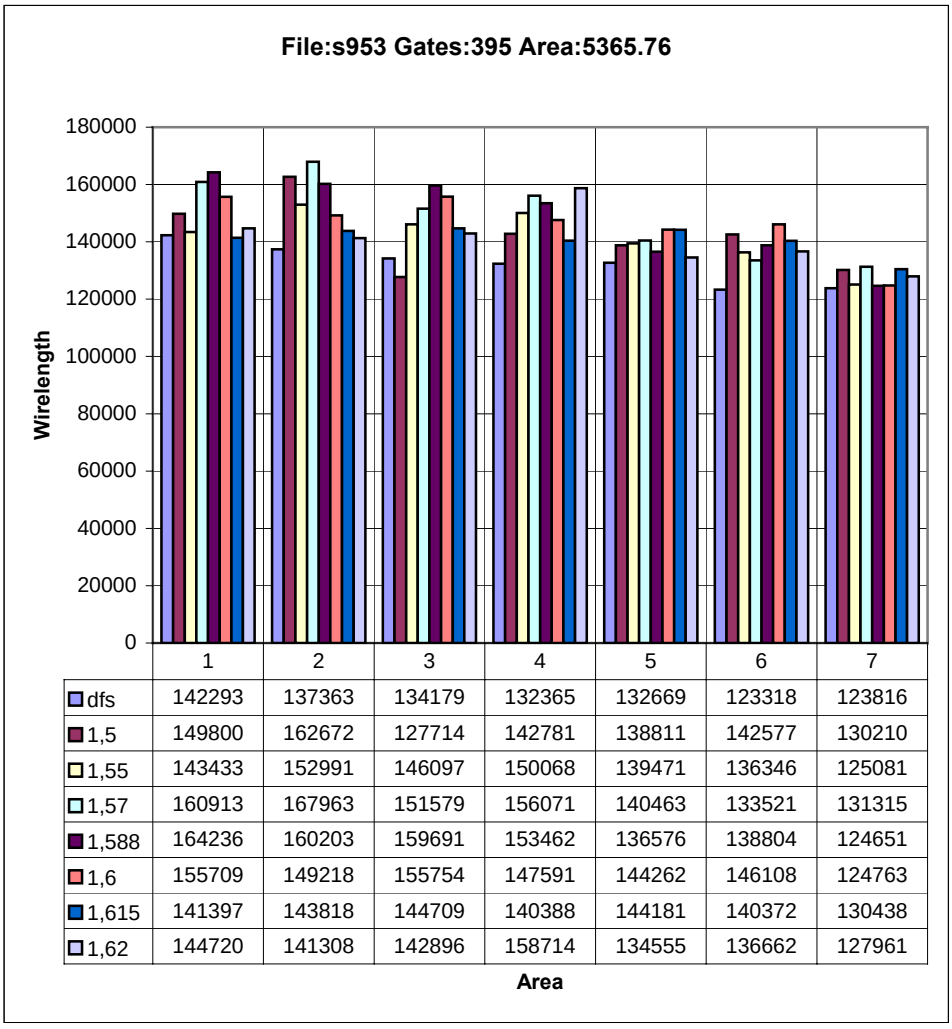
dfs	1.5	1.55	1.57	1.588	1.6	1.615	1.62
0	32	24	18	8	8	4	3

Πίνακας 5.17 Μέγεθος των Επιφανειών για το κύκλωμα s953

Επιφ. 1	Επιφ. 2	Επιφ. 3	Επιφ. 4	Επιφ. 5	Επιφ. 6	Επιφ. 7
12480	11520	10400	9600	8320	7680	7680

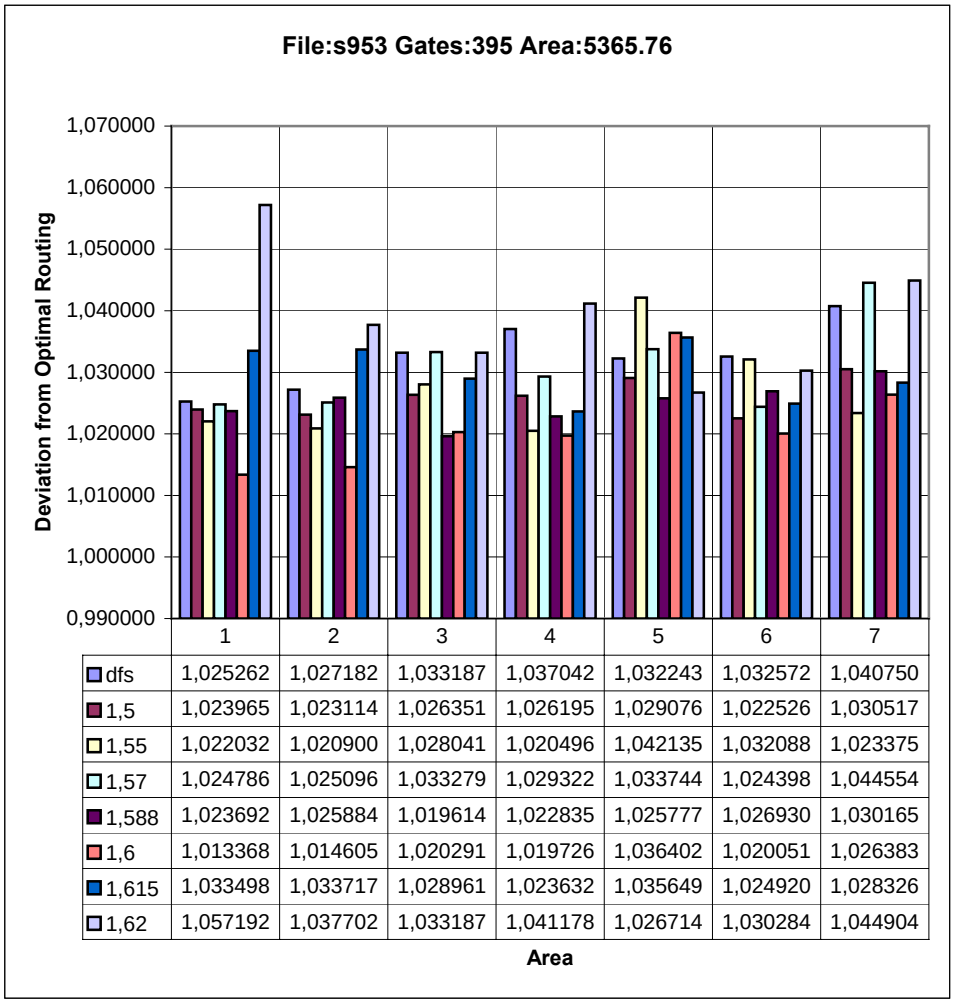
Στον Πίνακα 5.18 παρατηρούμε ότι ο το μήκος των διασυνδέσεων στην περίπτωση του αλγόριθμού με τους πυκνούς υπογράφους είναι κατά μέσο όρο μεγαλύτερο σε όλες τις περιπτώσεις συγκριτικά με τον αλγόριθμο που κάνει μια απλή dfs διάσχιση. Παρόλα αυτά το μέσο μήκος των διασυνδέσεων στις δύο τελευταίες περιπτώσεις (ο λόγος βάρους είναι ίσος με 1.615 και 1.62 αντίστοιχα) είναι σχεδόν το ίδιο και μικρότερο από ότι στις άλλες.

Πίνακας 5.18 Συνολικό Μήκος Διασυνδέσεων Συναρτήσει της Επιφάνειας του Chip για το Κύκλωμα s953



	1	2	3	4	5	6	7
dfs	142293	137363	134179	132365	132669	123318	123816
1,5	149800	162672	127714	142781	138811	142577	130210
1,55	143433	152991	146097	150068	139471	136346	125081
1,57	160913	167963	151579	156071	140463	133521	131315
1,588	164236	160203	159691	153462	136576	138804	124651
1,6	155709	149218	155754	147591	144262	146108	124763
1,615	141397	143818	144709	140388	144181	140372	130438
1,62	144720	141308	142896	158714	134555	136662	127961

Πίνακας 5.19 Απόκλιση Από τη Βέλτιστη Λύση Συναρτήσει της Επιφάνειας του Chip για το Κύκλωμα s953

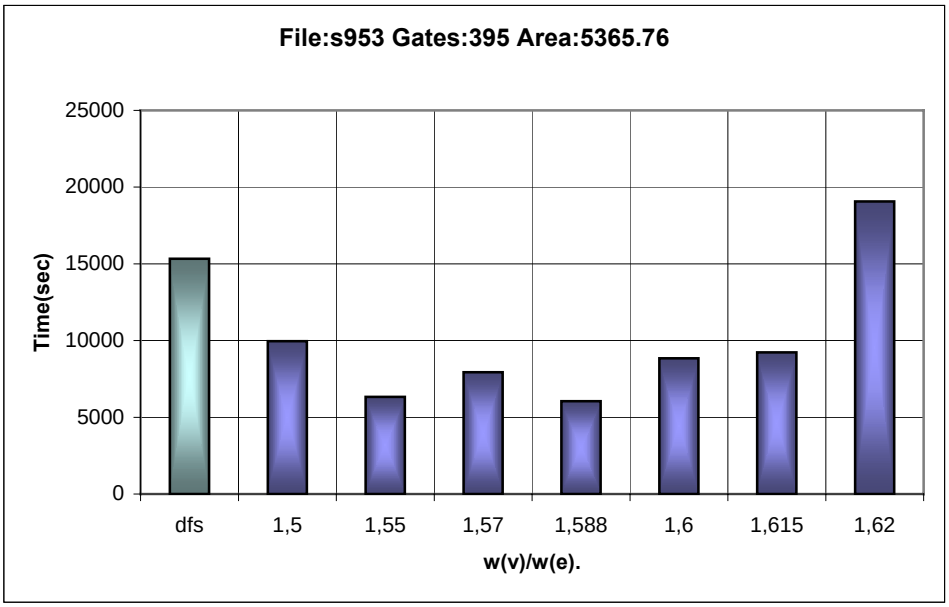


	1	2	3	4	5	6	7
dfs	1,025262	1,027182	1,033187	1,037042	1,032243	1,032572	1,040750
1,5	1,023965	1,023114	1,026351	1,026195	1,029076	1,022526	1,030517
1,55	1,022032	1,020900	1,028041	1,020496	1,042135	1,032088	1,023375
1,57	1,024786	1,025096	1,033279	1,029322	1,033744	1,024398	1,044554
1,588	1,023692	1,025884	1,019614	1,022835	1,025777	1,026930	1,030165
1,6	1,013368	1,014605	1,020291	1,019726	1,036402	1,020051	1,026383
1,615	1,033498	1,033717	1,028961	1,023632	1,035649	1,024920	1,028326
1,62	1,057192	1,037702	1,033187	1,041178	1,026714	1,030284	1,044904

Όπως και προηγούμενα, έτσι και σε αυτό το πείραμα (Πίνακας 5.19) παρατηρούμε ότι κατά μέσο όρο μόνο η απόκλιση από τη βέλτιστη λύση της τελευταίας περίπτωσης είναι χειρότερη από αυτήν που προκύπτει από την τοποθέτηση των λογικών πυλών με τη βοήθεια του αλγόριθμου που κάνει μια απλή dfs διάσχιση του γράφου.

Στον Πίνακα 5.20 που αναπαριστά το χρόνο διασύνδεσης των λογικών πυλών παρατηρούμε όπως και στα προηγούμενα κυκλώματα πως όταν η τοποθέτηση γίνει με τον πρώτο αλγόριθμο η διασύνδεση απαιτεί περισσότερο χρόνο.

Πίνακας 5.20 Λόγος Βάρους του Αλγόρίθμου Συναρτήσει του Χρόνου για το Κύκλωμα s953



Τέλος, η μικρότερη επιφάνεια που χρησιμοποιεί η τοποθέτηση μας, είναι περίπου 7% μεγαλύτερη από την ελάχιστη.

5.2.5. File s838.1

Τα κύρια χαρακτηριστικά του συγκεκριμένου κυκλώματος είναι τα εξής

Πλήθος Πυλών:	446
Πλήθος Διασυνδέσεων:	561
Συνολική Επιφάνεια των Πυλών:	6414.325

Η ερμηνεία του Πίνακα 5.21 είναι παρόμοια με αυτή των Πινάκων 5.13, 5.9, 5.5 και 5.1

Πίνακας 5.21 Πλήθος Πυκνών Υπογράφων του Κυκλώματος s838.1

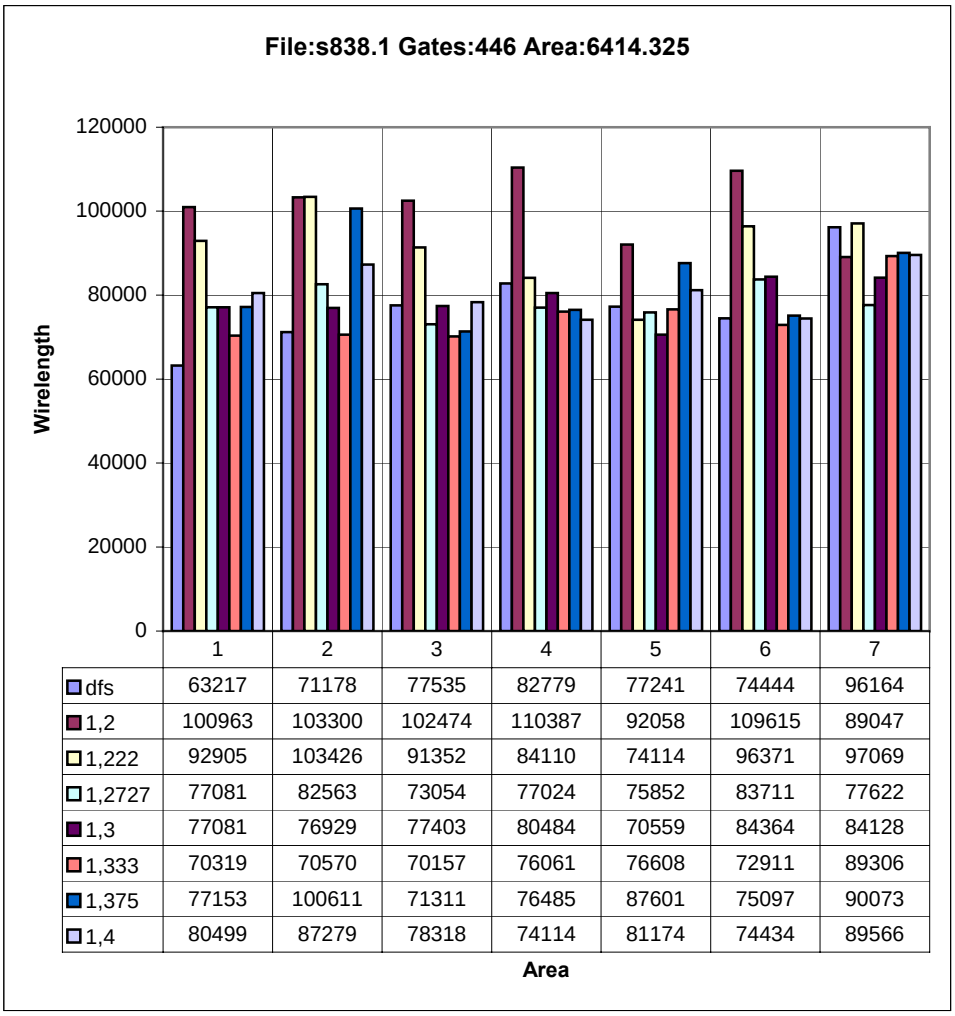
dfs	1.2	1.222	1.2727	1.3	1.333	1.375	1.4
0	32	27	7	4	16	13	9

Πίνακας 5.22 Μέγεθος των Επιφανειών για το κύκλωμα s838.1

Επιφ. 1	Επιφ. 2	Επιφ. 3	Επιφ. 4	Επιφ. 5	Επιφ. 6	Επιφ. 7
13440	12480	11520	10400	9600	8320	7680

Στη συνέχεια, στον Πίνακα 5.23 παρατηρούμε τη τιμές που παίρνει το συνολικό μήκος των διασυνδέσεων για τις ξεχωριστές εκτελέσεις του αλγόριθμου. Σε όλες τις περιπτώσεις το μήκος που προκύπτει από την εκτέλεση του αλγόριθμου με μια απλή dfs διάσχιση του αλγόριθμου είναι το μικρότερο.

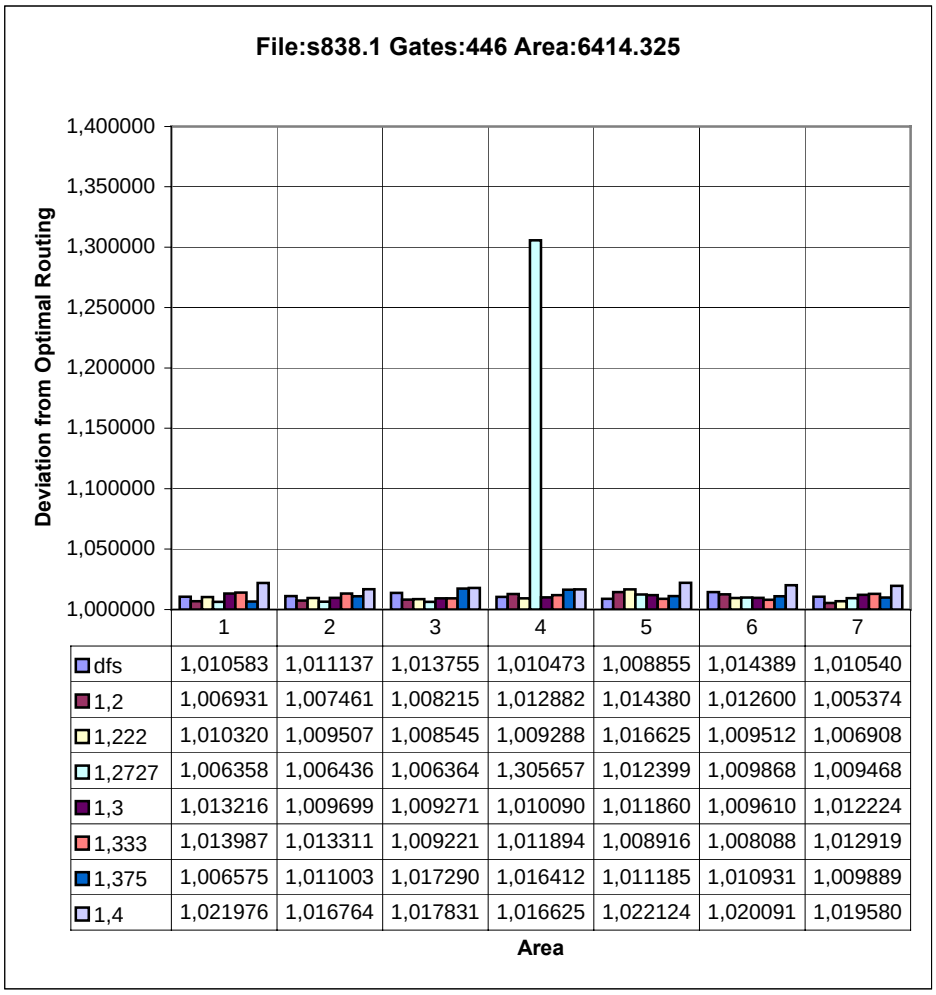
Πίνακας 5.23 Συνολικό Μήκος Διασυνδέσεων Συναρτήσει της Επιφάνειας του Chip για το Κύκλωμα s838.1



	1	2	3	4	5	6	7
dfs	63217	71178	77535	82779	77241	74444	96164
1,2	100963	103300	102474	110387	92058	109615	89047
1,222	92905	103426	91352	84110	74114	96371	97069
1,2727	77081	82563	73054	77024	75852	83711	77622
1,3	77081	76929	77403	80484	70559	84364	84128
1,333	70319	70570	70157	76061	76608	72911	89306
1,375	77153	100611	71311	76485	87601	75097	90073
1,4	80499	87279	78318	74114	81174	74434	89566

Στον Πίνακα 5.24, όπου παρουσιάζεται η απόκλιση από τη βέλτιστη λύση, το συμπέρασμα που βγάζουμε είναι παρόμοιο με τα προηγούμενα. Σε όλες τις περιπτώσεις εκτός της τελευταίας και της τρίτης (ο λόγος βάρους είναι 1,2727) η απόκλιση του αλγόριθμου που υπολογίζει πυκνούς υπογράφους είναι μικρότερη κατά μέσο όρο από την αντίστοιχη που υπολογίζει ο αλγόριθμος που κάνει μια απλή dfs διάσχιση το γράφου. Επίσης, ούτε εδώ η απόκλιση από τη βέλτιστη λύση είναι μεγάλη, αφού στη «χειρότερη» εκτέλεση το μήκος των διασυνδέσεων είναι μόλις 3% μεγαλύτερο από τη βέλτιστη περίπτωση.

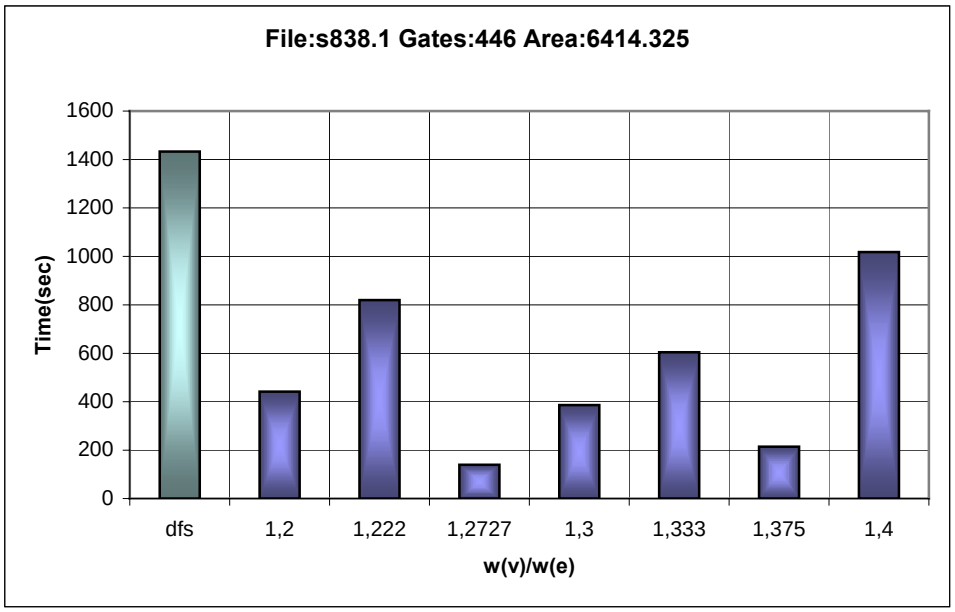
Πίνακας 5.24 Απόκλιση από τη Βέλτιστη Λύση Συναρτήσει της Επιφάνειας του Chip για το Κύκλωμα s838.1



	1	2	3	4	5	6	7
dfs	1,010583	1,011137	1,013755	1,010473	1,008855	1,014389	1,010540
1,2	1,006931	1,007461	1,008215	1,012882	1,014380	1,012600	1,005374
1,222	1,010320	1,009507	1,008545	1,009288	1,016625	1,009512	1,006908
1,2727	1,006358	1,006436	1,006364	1,305657	1,012399	1,009868	1,009468
1,3	1,013216	1,009699	1,009271	1,010090	1,011860	1,009610	1,012224
1,333	1,013987	1,013311	1,009221	1,011894	1,008916	1,008088	1,012919
1,375	1,006575	1,011003	1,017290	1,016412	1,011185	1,010931	1,009889
1,4	1,021976	1,016764	1,017831	1,016625	1,022124	1,020091	1,019580

Παρόμοια με τα προηγούμενα κυκλώματα, έτσι και σε αυτό όταν η τοποθέτηση επιτυγχάνεται με τον αλγόριθμο που εκτελεί μια απλή dfs διάσχιση, τότε ο χρόνος που απαιτείται για τη διασύνδεση είναι συγκριτικά ο μεγαλύτερος (Πίνακας 5.25).

Πίνακας 5.25 Λόγος Βάρους του Αλγόρίθμου Συναρτήσει του Χρόνου για το Κύκλωμα s953



Τέλος, η μικρότερη επιφάνεια που απαιτείται από τους αλγορίθμους μας είναι περίπου 4% μεγαλύτερη της ελάχιστης.

5.2.6. File c2670o

Τα χαρακτηριστικά του κυκκλώματος που εξετάζεται είναι

Πλήθος Πυλών:	1389
Πλήθος Διασυνδέσεων:	2046
Συνολική Επιφάνεια των Πυλών:	18190.402

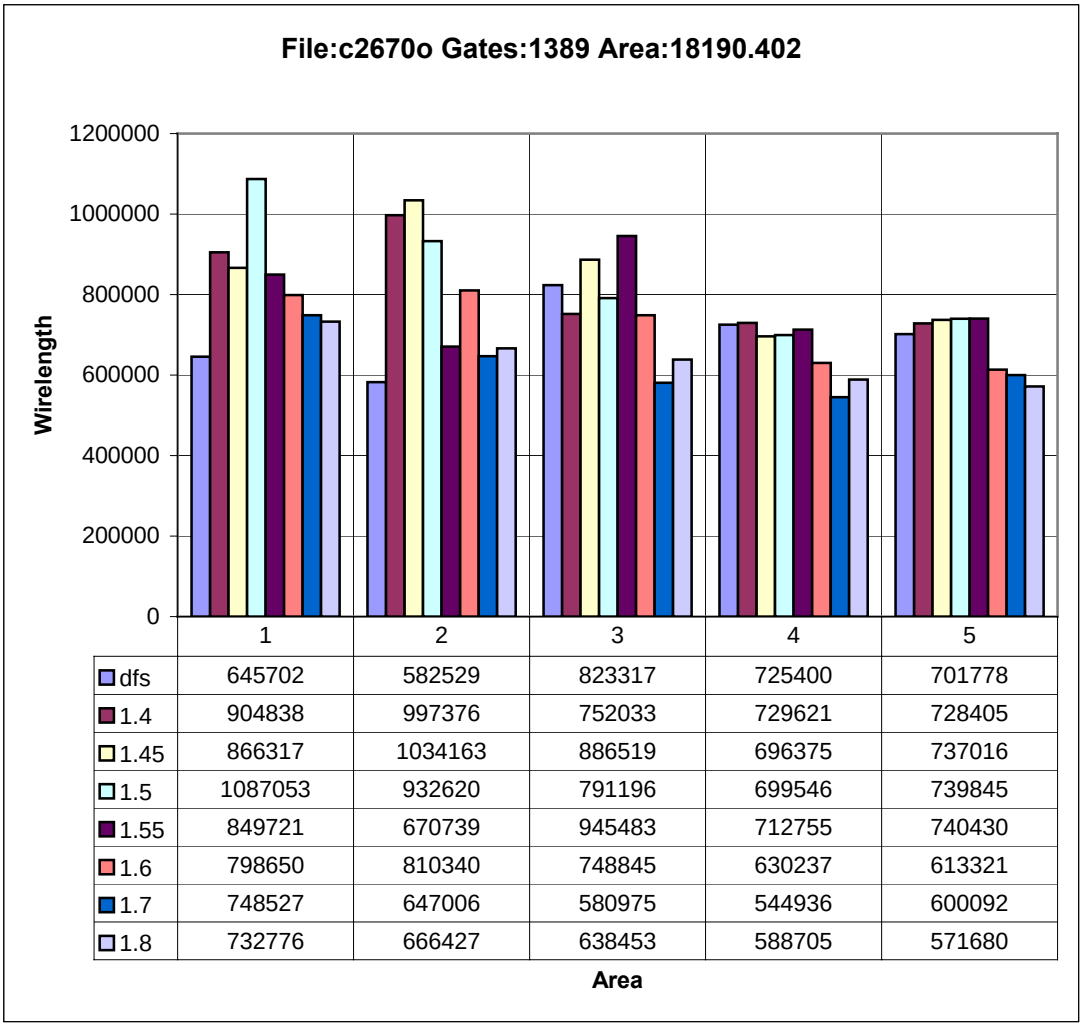
Πίνακας 5.26 Πλήθος Πυκνών Υπογράφων του Κυκλώματος c2670o

dfs	1.4	1.45	1.5	1.55	1.6	1.7	1.8
0	116	72	48	38	21	4	4

Πίνακας 5.27 Μέγεθος των Επιφανειών για το Κύκλωμα c2670o

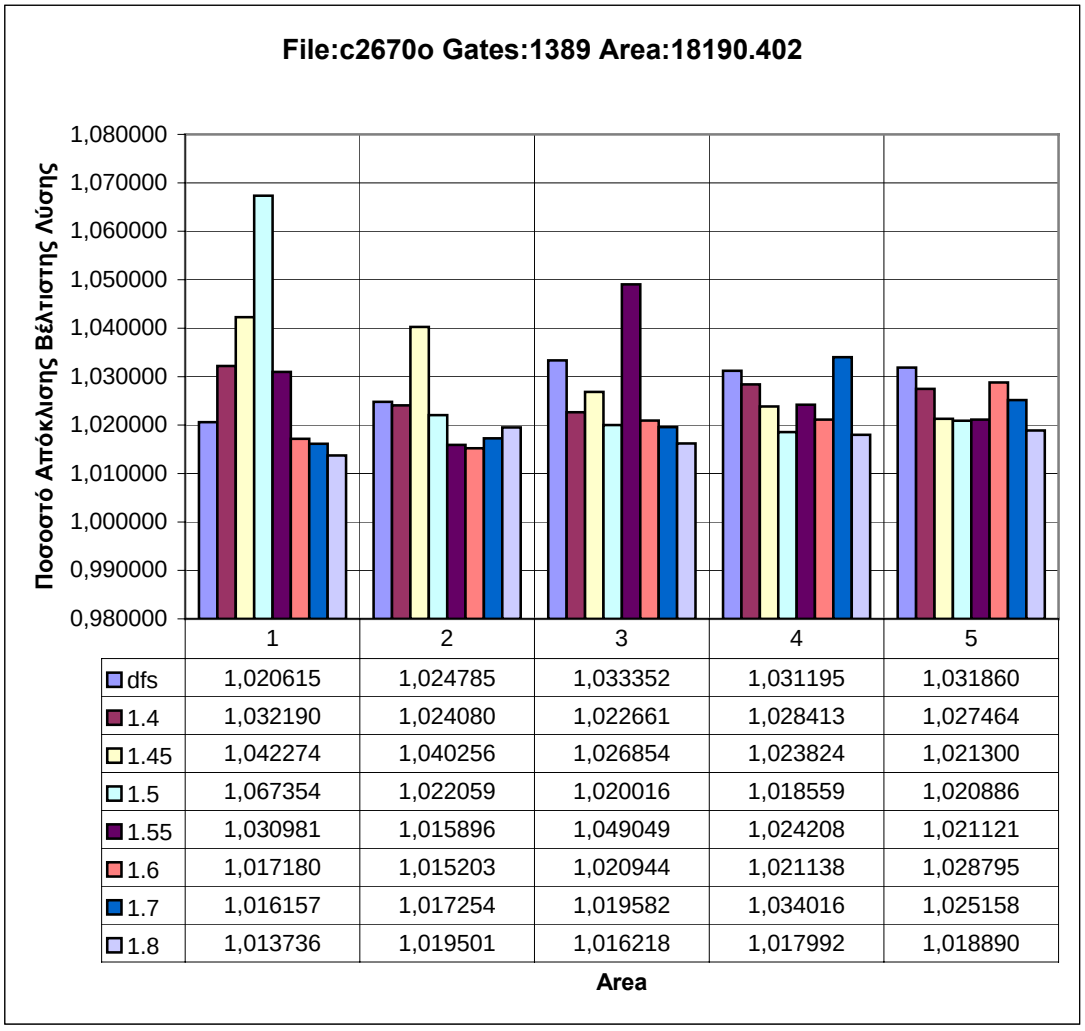
Επιφάνεια 1	Επιφάνεια 2	Επιφάνεια 3	Επιφάνεια 4	Επιφάνεια 5
20480	20160	19200	19200	18560

Πίναακας 5.28 Συνολικό Μήκος Διασυνδέσεων Συναρτήσει της Επιφάνειας του Chip για το Κύκλωμα c2670o



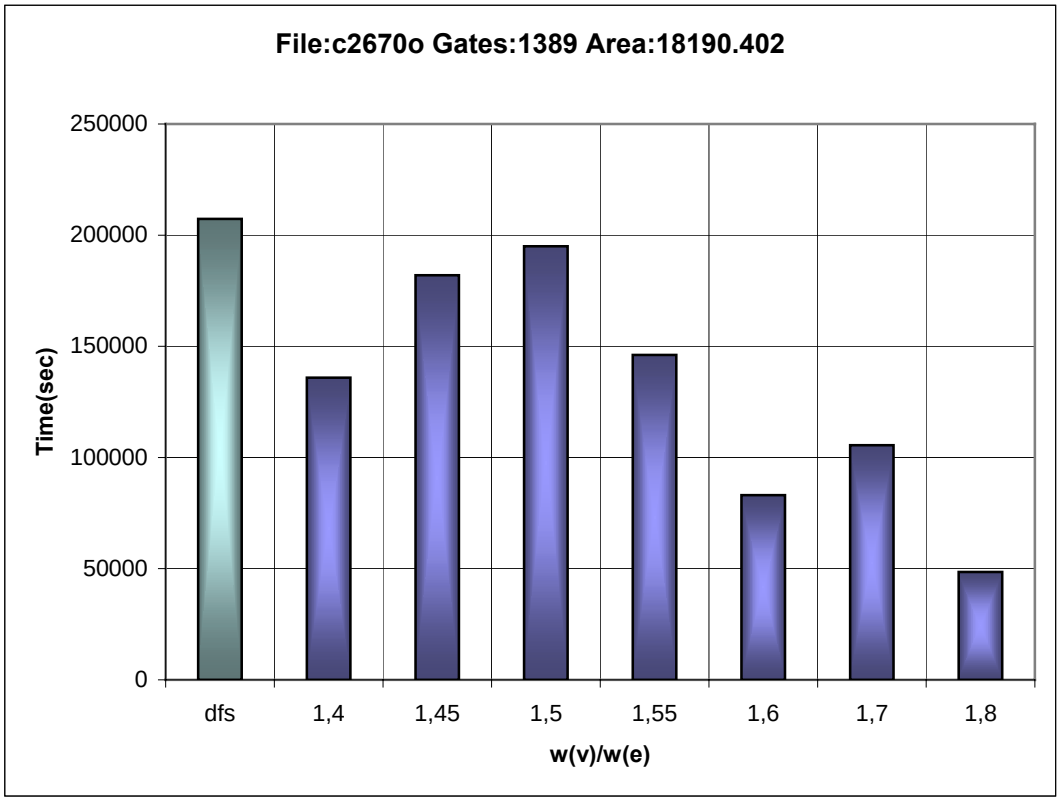
	1	2	3	4	5
dfs	645702	582529	823317	725400	701778
1.4	904838	997376	752033	729621	728405
1.45	866317	1034163	886519	696375	737016
1.5	1087053	932620	791196	699546	739845
1.55	849721	670739	945483	712755	740430
1.6	798650	810340	748845	630237	613321
1.7	748527	647006	580975	544936	600092
1.8	732776	666427	638453	588705	571680

Πίνακας 5.29 Απόκλιση από τη Βέλτιστη Λύση Συναρτήσει της Επιφάνειας του Chip για το κύκλωμα c2670o



	1	2	3	4	5
dfs	1,020615	1,024785	1,033352	1,031195	1,031860
1.4	1,032190	1,024080	1,022661	1,028413	1,027464
1.45	1,042274	1,040256	1,026854	1,023824	1,021300
1.5	1,067354	1,022059	1,020016	1,018559	1,020886
1.55	1,030981	1,015896	1,049049	1,024208	1,021121
1.6	1,017180	1,015203	1,020944	1,021138	1,028795
1.7	1,016157	1,017254	1,019582	1,034016	1,025158
1.8	1,013736	1,019501	1,016218	1,017992	1,018890

Πίνακας 5.30 Λόγος Βάρους του Αλγορίθμου Συναρτήσει του Χρόνου για το Κύκλωμα c2670o



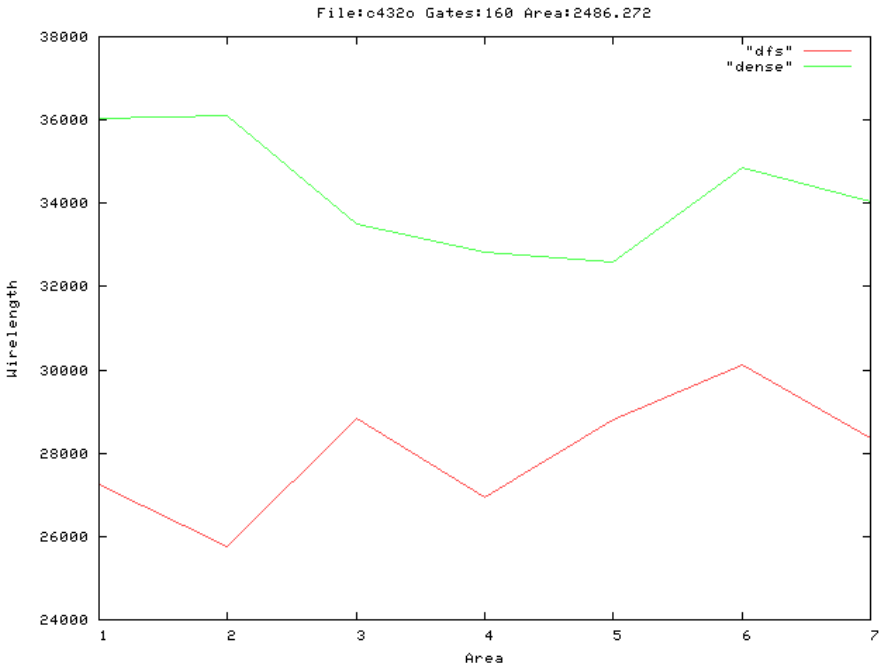
Στον Πίνακα 5.28 παρατηρούμε ότι το συνολικό μήκος των διασυνδέσεων, στην περίπτωση όπου το εμβαδό της περιοχής του chip είναι σχετικά μικρό, είναι κατά πολύ καλύτερο όταν η τοποθέτηση γίνεται με τον dense αλγόριθμο. Η ίδια προσέγγιση επιτυγχάνει και αρκετά καλύτερο χρόνο διασύνδεσης (Πίνακας 5.30). Τέλος, μικρότερη η επιφάνεια που απαιτείται για να γίνει η τοποθέτηση είναι περίπου 3% μεγάλυτερη της ελάχιστης.

5.3. Σύγκριση Αποτελεσμάτων

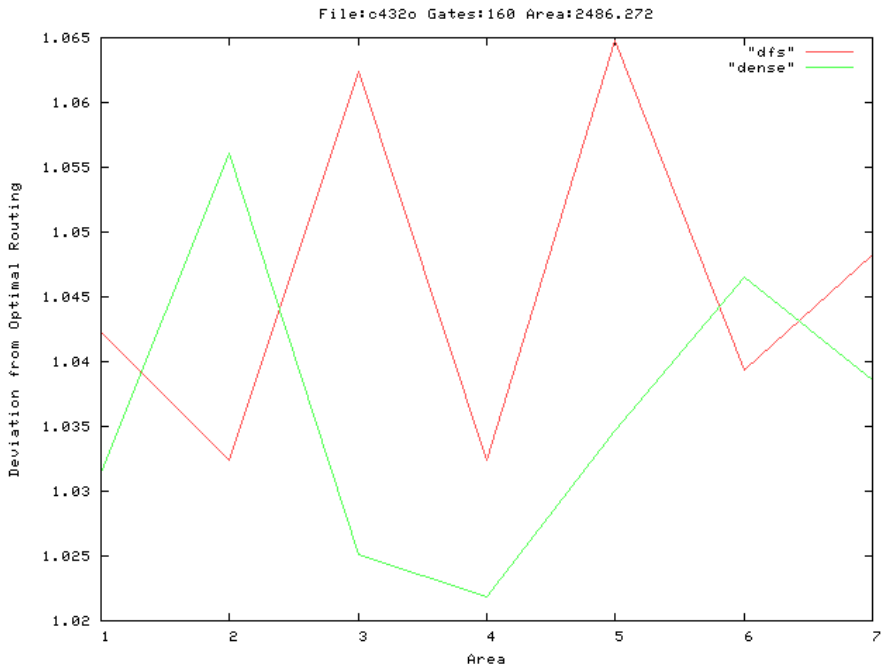
Στην προηγούμενη παράγραφο παραθέσαμε τα αποτελέσματα των πειραμάτων που σχετίζονται με το μήκος των διασυνδέσεων, με την απόκλιση από τη βέλτιστη λύση και με το χρόνο εκτέλεσης του αλγόριθμου διασύνδεσης για μια δεδομένη τοποθέτηση για κάθε κύκλωμα χωριστά.

Στο σημείο αυτό, επιλέξαμε το λόγο του βάρους που μας δίνει το μικρότερο μήκος διασύνδεσης για κάθε περιοχή που δοκιμάστηκε αλγόριθμος dense για ένα δεδομένο κύκλωμα. Έτσι λοιπόν, κατασκευάσαμε για κάθε κύκλωμα τρία διαγράμματα που αναπαριστούν τα αποτελέσματα του αλγόριθμου dense δεδομένου του συγκεκριμένου λόγου βάρους και του αλγόριθμου που εκτελεί μια απλή dfs αναζήτηση. Το πρώτο διάγραμμα αναπαριστά το μήκος των διασυνδέσεων συναρτήσει των περιοχών του chip, το δεύτερο την απόκλιση από τη βέλτιστη λύση συναρτήσει των περιοχών του chip και το τρίτο το χρόνο εκτέλεσης συναρτήσει των περιοχών του chip.

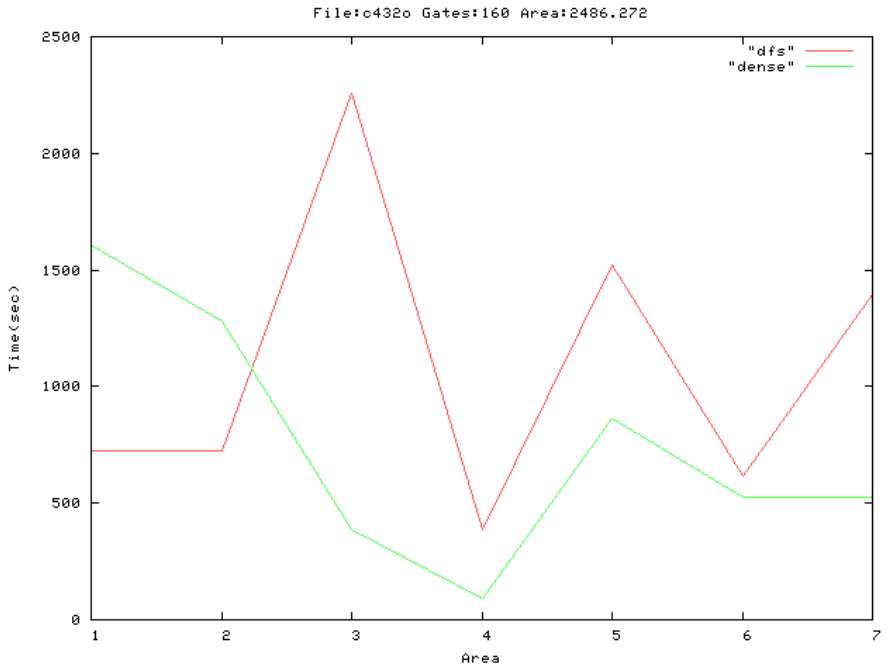
Έτσι λοιπόν, στα Σχήματα 5.5, 5.6 και 5.7 φαίνονται τα παραπάνω διαγράμματα για το κύκλωμα c432o, τα Σχήματα 5.8, 5.9 και 5.10 αντιστοιχούν στο κύκλωμα s420.1, τα Σχήματα 5.11, 5.12 και 5.13 στο κύκλωμα s713, τα Σχήματα 5.14, 5.15 και 5.16 στο κύκλωμα s953, τα Σχήματα 5.17, 5.18 και 5.19 στο κύκλωμα s838.1 και τα Σχήματα 5.20, 5.21 και 5.22 στο κύκλωμα c2670o



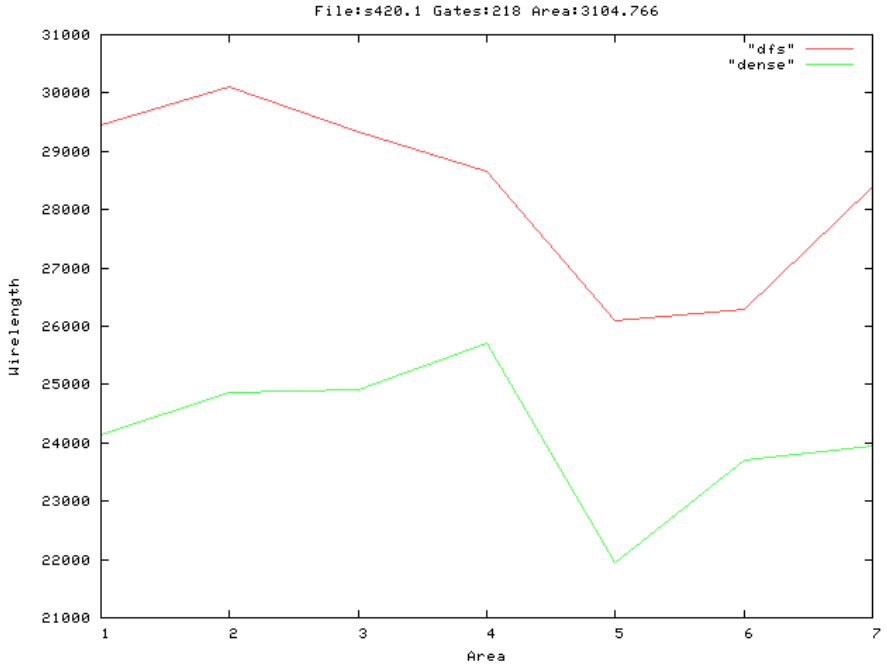
Σχήμα 5.5 Το Μήκος Διασύνδεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το λόγο Βάρους με το Καλύτερο Αποτέλεσμα για το Κύκλωμα c432o



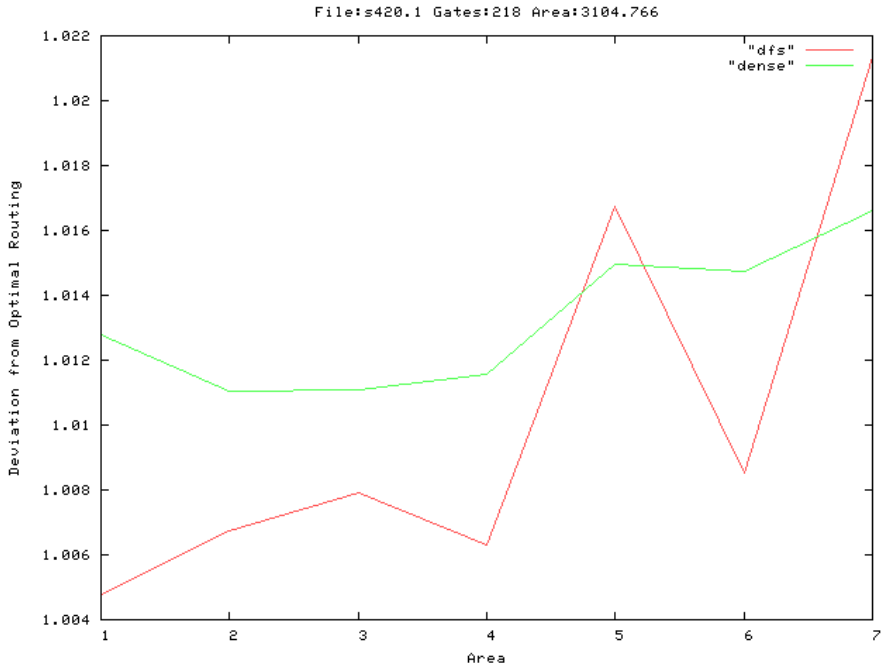
Σχήμα 5.6 Η Απόκλιση της Βέλτιστης Λύσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα c432o



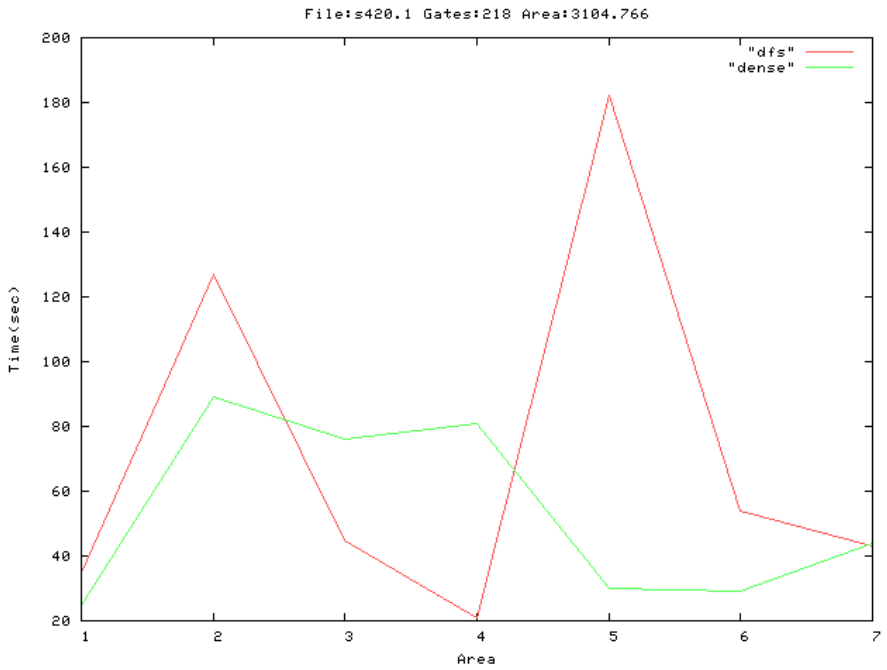
Σχήμα 5.7 Ο Χρόνος Εκτέλεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα c432o



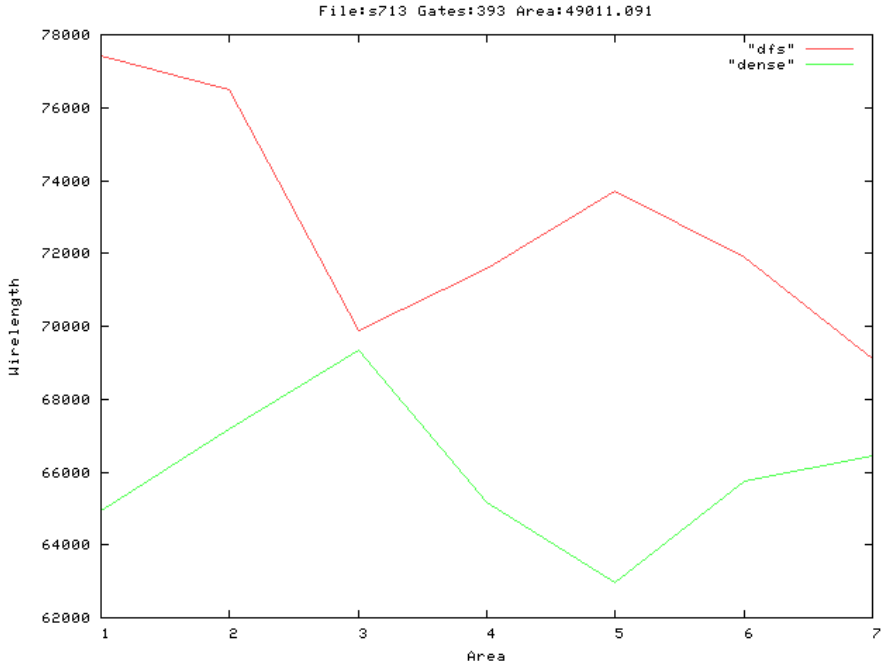
Σχήμα 5.8 Το Μήκος Διασύνδεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Καλύτερο Αποτέλεσμα για το Κύκλωμα s420.1



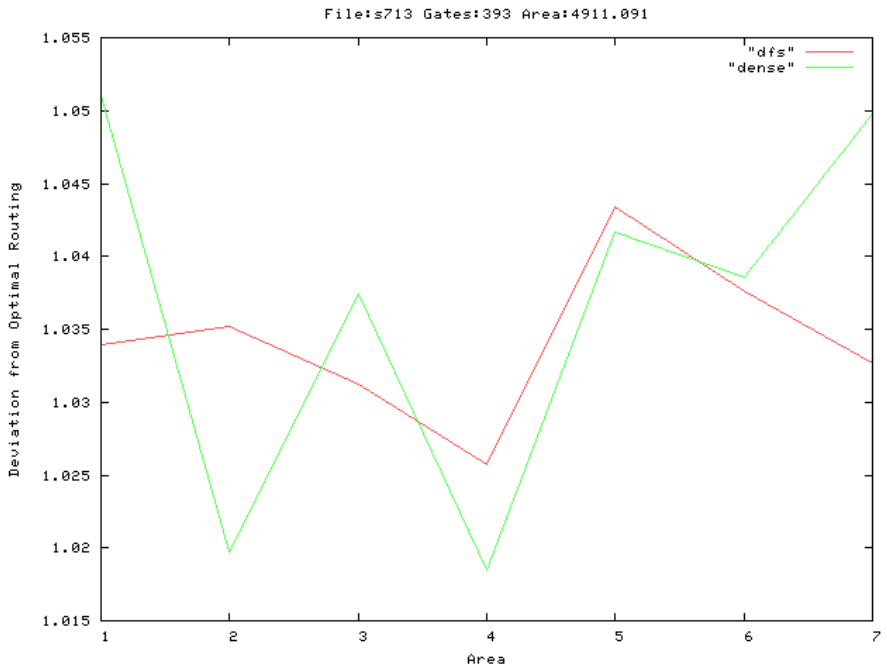
Σχήμα 5.9 Η Απόκλιση της Βέλτιστης Λύσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα s420.1



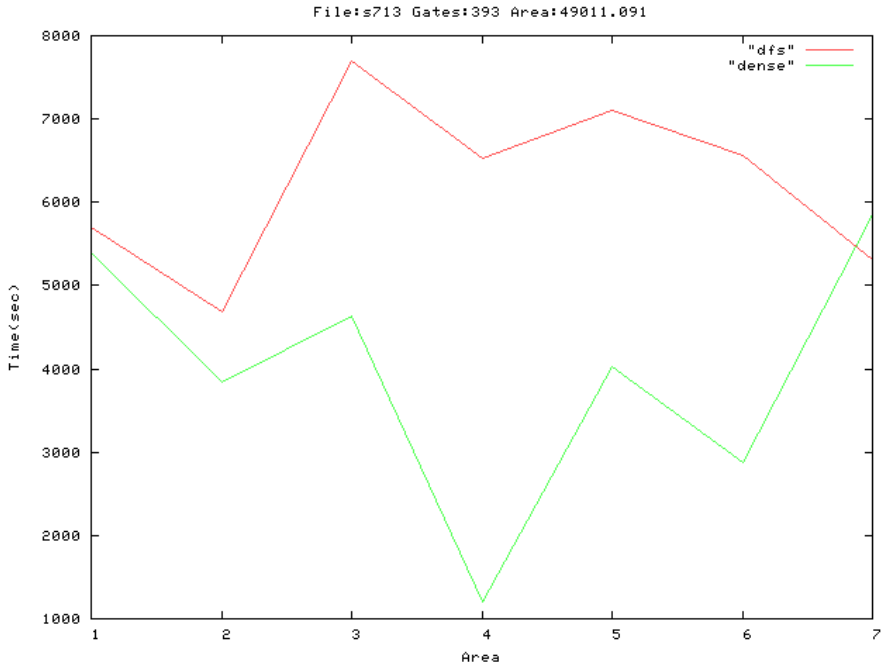
Σχήμα 5.10 Ο Χρόνος Εκτέλεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα s420.1



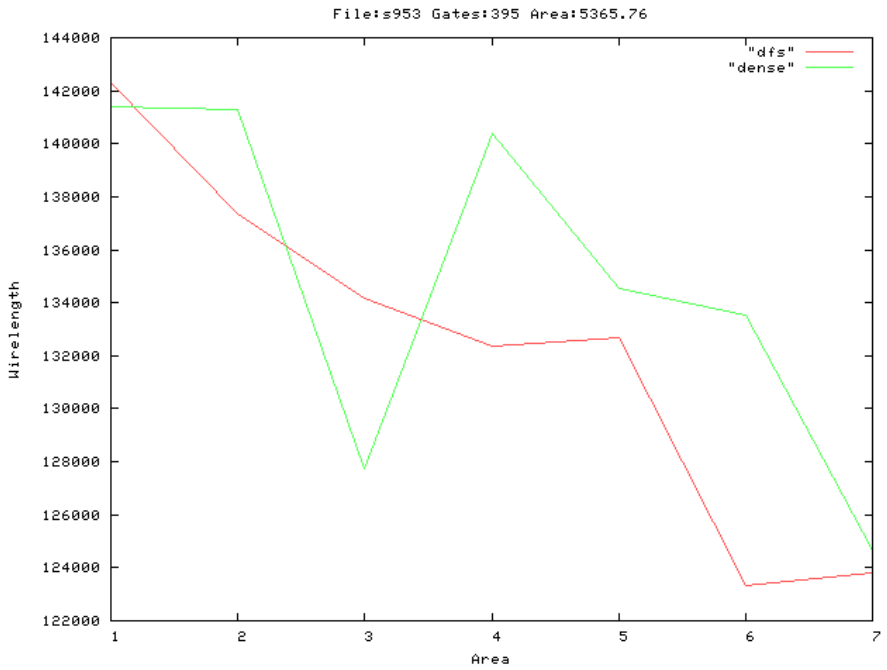
Σχήμα 5.11 Το Μήκος Διασύνδεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Καλύτερο Αποτέλεσμα για το Κύκλωμα s713



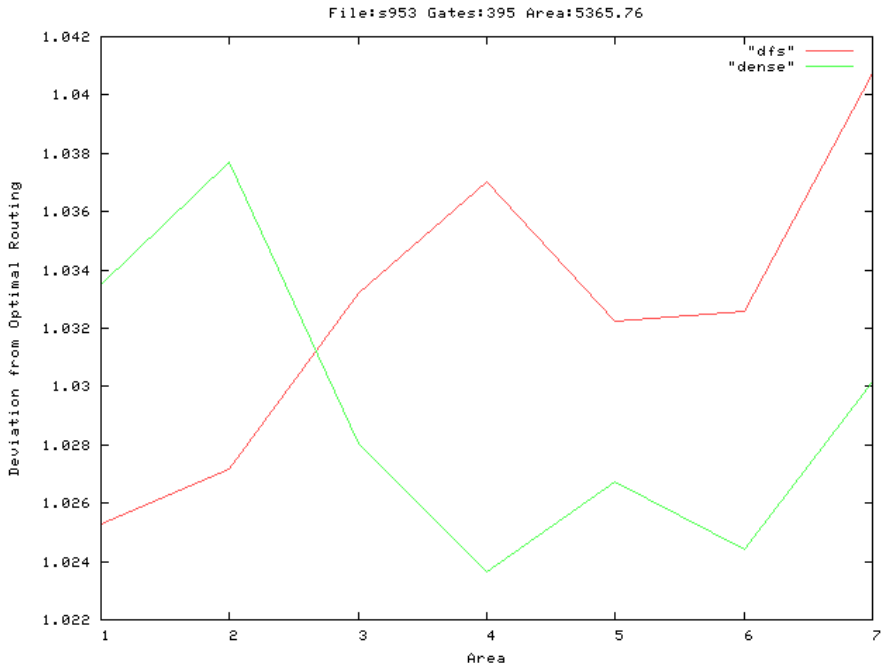
Σχήμα 5.12 Η Απόκλιση της Βέλτιστης Λύσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα s713



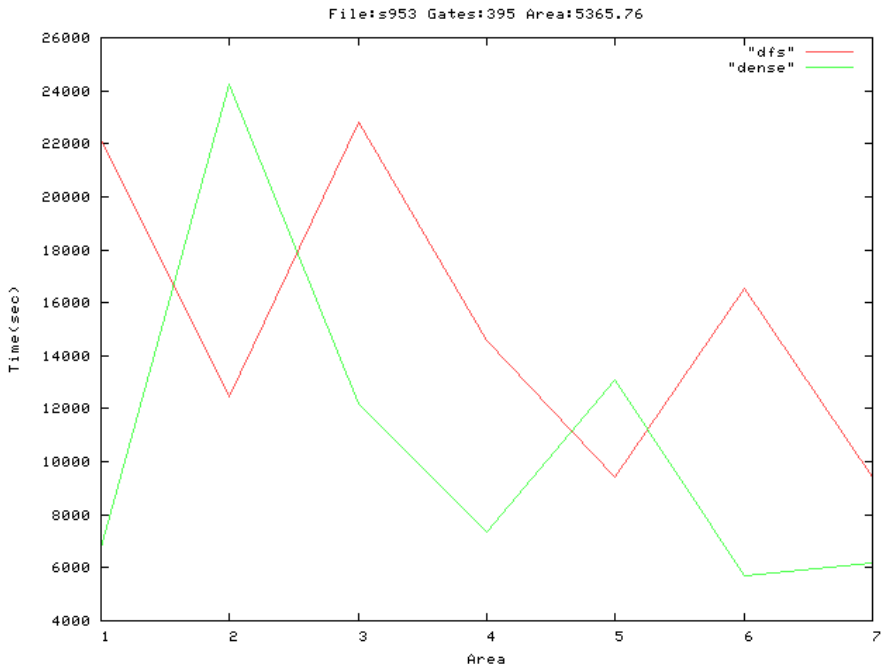
Σχήμα 5.13 Ο Χρόνος Εκτέλεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα s713



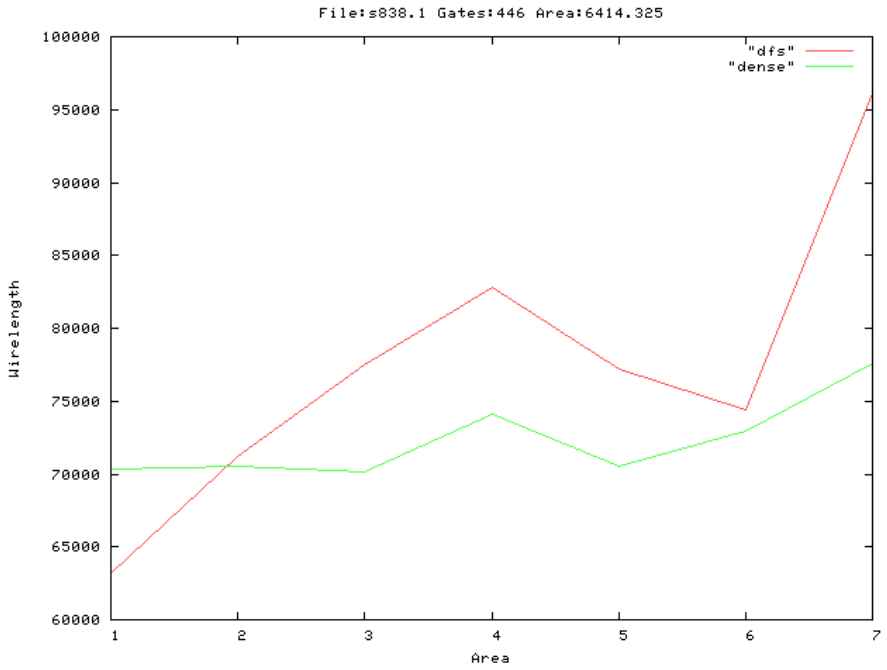
Σχήμα 5.14 Το Μήκος Διασύνδεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Καλύτερο Αποτέλεσμα για το Κύκλωμα s953



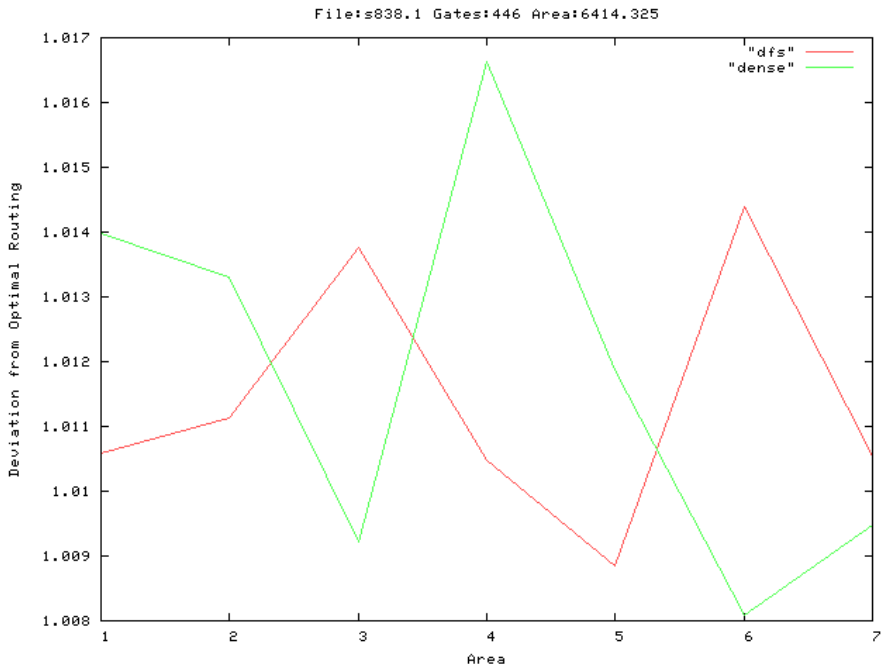
Σχήμα 5.15 Η Απόκλιση της Βέλτιστης Λύσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα s953



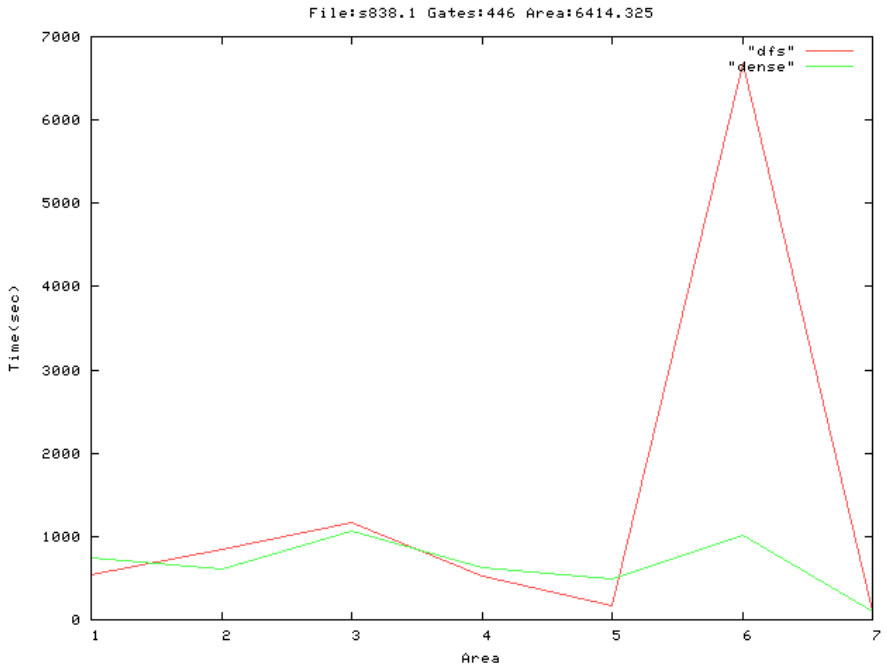
Σχήμα 5.16 Ο Χρόνος Εκτέλεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα s953



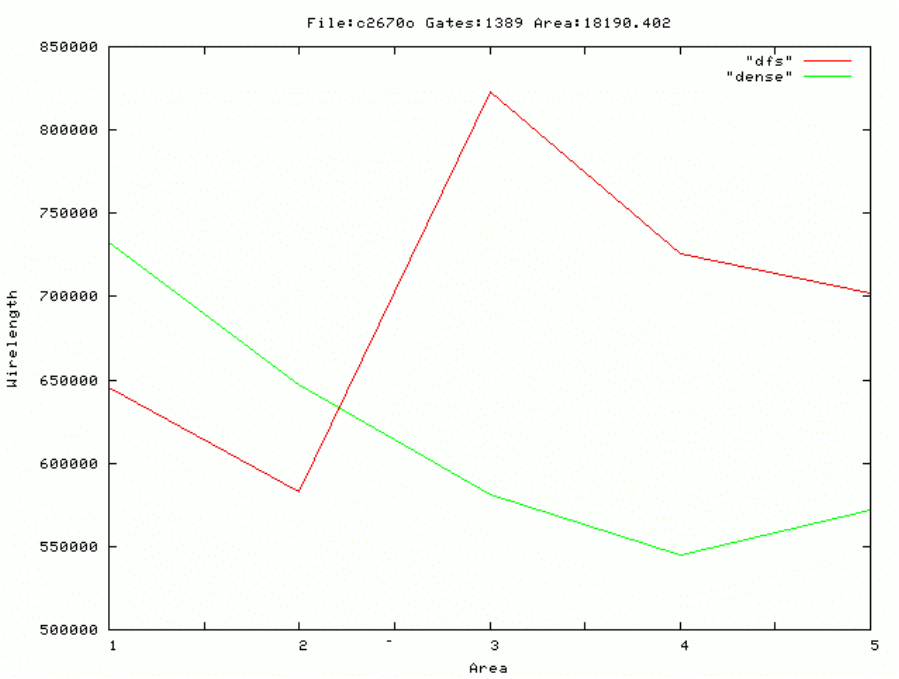
Σχήμα 5.17 Το Μήκος Διασύνδεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Καλύτερο Αποτέλεσμα για το Κύκλωμα s838.1



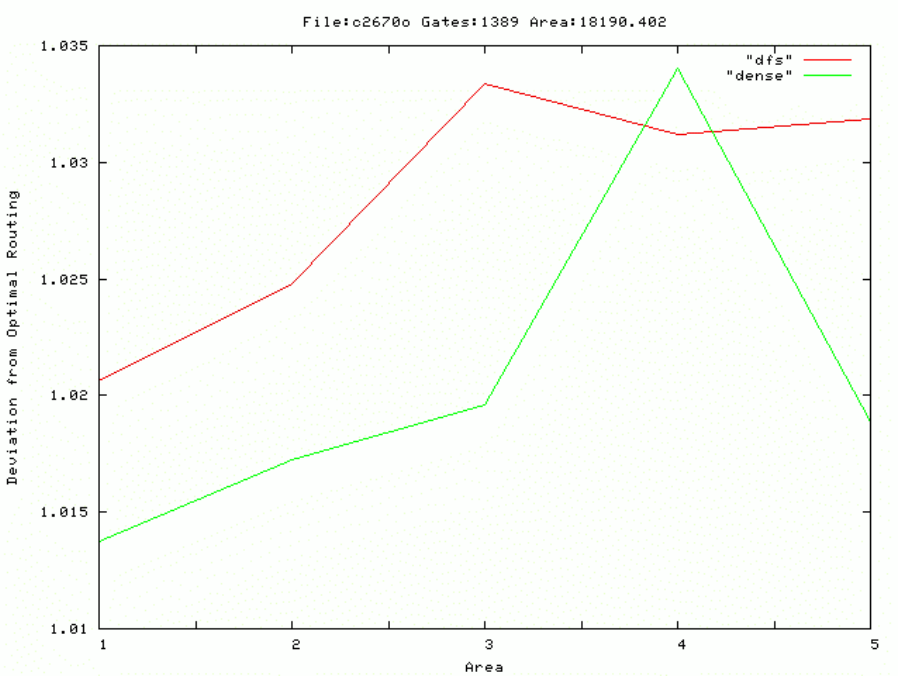
Σχήμα 5.18 Η Απόκλιση της Βέλτιστης Λύσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα s838.1



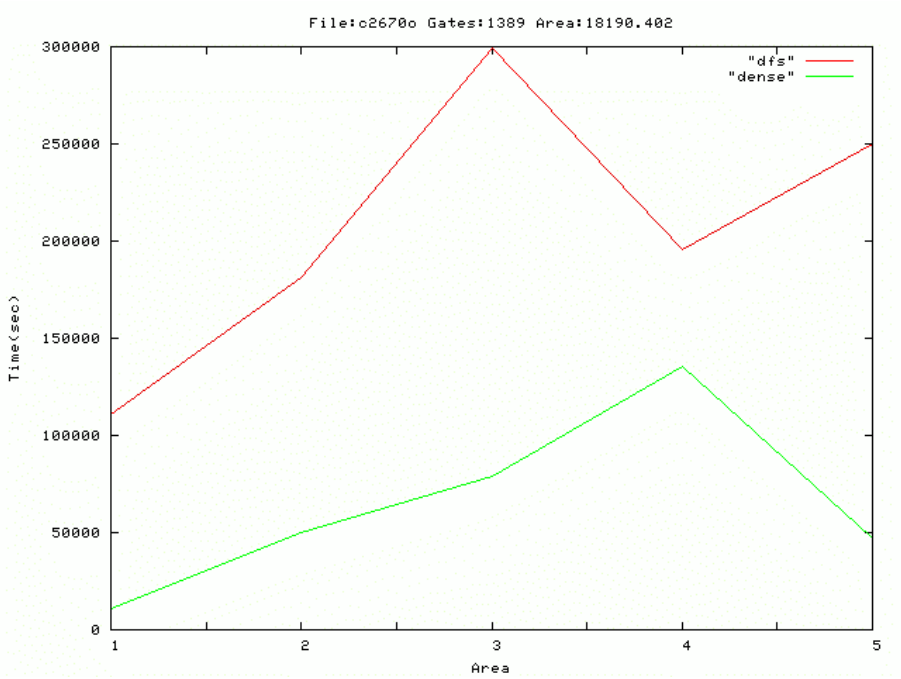
Σχήμα 5.19 Ο Χρόνος Εκτέλεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα s838.1



Σχήμα 5.20 Το Μήκος Διασύνδεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Καλύτερο Αποτέλεσμα για το Κύκλωμα c2670o



Σχήμα 5.21 Η Απόκλιση της Βέλτιστης Λύσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα c2670o



Σχήμα 5.22 Ο Χρόνος Εκτέλεσης Συναρτήσει της Περιοχής Δεδομένου ότι Επιλέξαμε το Λόγο Βάρους με το Μικρότερο Μήκος Διασύνδεσης για το Κύκλωμα c2670o

Συγκρίνοντας λοιπόν, τα παραπάνω αποτελέσματα παρατηρούμε ότι κατά μέσο όρο μόνο στο κύκλωμα c432o το μήκος των διασυνδέσεων είναι εμφανές μεγαλύτερο στον dense αλγόριθμο, περίπου 22%, ενώ στα υπόλοιπα κυκλώματα είναι είτε μικρότερο είτε σχεδόν ίσο.

Στον Πίνακα 5.31 φαίνεται η μέση ποσοστιαία διαφορά του μήκους των διασυνδέσεων που προκύπτεί από την εκτέλεση των δύο αλγόρίθμων για κάθε κύκλωμα. Στην πρώτη στήλη είναι το όνομα του κυκλώματος, στη δεύτερη η μέση ποσοστιαία διαφορά και τέλος υπάρχει μια κουκίδα στην τρίτη ή στην τέταρτη στήλη, ανάλογα με το ποιος αλγόριθμος έχει καλύτερα αποτελέσματα στη συγκεκριμένη περίπτωση. Με την ετικέτα dfs συμβολίζουμε τον αλγόριθμο που εκτελεί μια απλή dfs αναζήτηση, ενώ με την ετικέτα dense συμβολίζουμε τον αλγόριθμο που υπολογίζει dense γράφους

Πίνακας 5.31 Μέση Ποσοστιαία Διαφορά του Μήκους Διασύνδεσης για τα Κυκλώματα

Κύκλωμα	**Μέση Ποσοστιαία Διαφορά**	**Dfs**	**Dense**
c432o	22.37%	•	
s420.1	14.68%		•
s713	9.47%		•
s953	1.89%	•	
s838.1	6.69%		•
c2670o	10.97%		•

Αναφορικά με την απόκλιση από τη βέλτιστη λύση και στις δύο περιπτώσεις τα αποτελέσματα είναι πού μικρά, ενώ η μέση διαφορά τους για όλα τα κυκλώματα είναι σχεδόν αμελητέα. Τέλος, ο χρόνος εκτέλεσης του αλγορίθμου διασύνδεσης είναι πολύ καλύτερος στην περίπτωση του dense αλγορίθμου. Σε πλήρη αντιστοιχία με τον Πίνακα 5.31 παρουσιάζεται ο Πίνακας 5.32 μονό που στον τελευταίο φαίνεται η μέση ποσοστιαία διαφορά του χρόνο εκτέλεσης για κάθε κύκλωμα.

Πίνακας 5.32 Μέση Ποσοστιαία Διαφορά του Χρόνου Διασύνδεσης για τα Κυκλώματα

Κύκλωμα	**Μέση Ποσοστιαία Διαφορά**	**Dfs**	**Dense**
c432o	30.77%		•
s420.1	26.23%		•
s713	36.16%		•
s953	29.65%		•
s838.1	53.45%		•
c2670o	68.82%		•

5.4. Συμπεράσματα

Συγκρίνοντας λοιπόν τις δύο μεθόδους που χρησιμοποιήσαμε, παρατηρούμε ότι η μέθοδος που τοποθετεί τις πύλες κάνοντας μια απλή dfs διάσχιση του γράφου που αναπαριστά το κύκλωμα, έχει ως αποτέλεσμα το συνολικό μήκος των διασυνδέσεων στη γενική περίπτωση να είναι μικρότερο από αυτό που επιτυγχάνεται με τον άλλο αλγόριθμο, ανεξάρτητα από το λόγο βάρους που επιλέχθηκε. Παρόλα αυτά, η δεύτερη προσέγγιση μας δίνει επί μέρους καλύτερα αποτελέσματα όταν επιλέγουμε έναν τέτοιο λόγο βάρους, ο οποίος έχει ως αποτέλεσμα το πλήθος των πυκνών υπογράφων που θα προκύψει να είναι σχετικά μικρό. Συμπεραίνοντας, θα λέγαμε ότι ο υπολογισμός των πυκνών υπογράφων δεν έχει τα αναμενόμενα θετικά αποτελέσματα αναφορικά με το μήκος των διασυνδέσεων και αυτό γιατί δε λαμβάνει υπό όψη του τις διασυνδέσεις που υπάρχουν μεταξύ διαφορετικών πυκνών υπογράφων.

Επίσης, αναφορικά με το λόγο σφάλματος τα αποτελέσματα των πειραμάτων έδειξαν πως και οι δύο προσεγγίσεις αντιμετωπίζουν το πρόβλημα με επιτυχία. Αυτό γιατί η διασύνδεση που επιτυγχάνεται με τη δεδομένη τοποθέτηση δεν είναι ποτέ χειρότερη από το 7% της βέλτιστης περίπτωσης. Εδώ, ο αλγόριθμος που υπολογίζει πυκνούς υπογράφους τους τοποθετεί με καλύτερο τρόπο από ότι ο άλλος, ώστε να διευκολύνει τη δουλειά του διασυνδέτη (maze router) μικραίνοντας το ποσοστό σφάλματος.

Αυτό έχει άμεσο αντίκτυπο και στην ταχύτητα εκτέλεσης της διασύνδεσης με τη δεδομένη τοποθέτηση. Έτσι λοιπόν, η δεύτερη μέθοδος μας είχε ως αποτέλεσμα την ταχύτερη διασύνδεση. Μάλιστα, αν επιλέξουμε το λόγο βάρους να είναι τέτοιος ώστε να μας δίνει και το μικρότερο μήκος διασύνδεσης έχουμε ως αποτέλεσμα τις περισσότερες φορές ακόμα καλύτερα αποτελέσματα. Αυτό ως ένα βαθμό ήταν και το αναμενόμενο, αφού οι κόμβοι που είχαν αρκετές συνδέσεις μεταξύ τους τοποθετήθηκαν κοντά ο ένας στον άλλο.

Τέλος, η μικρότερη επιφάνεια του chip που μπόρεσαν να χρησιμοποιήσουν οι αλγόριθμοι για να τοποθετήσουν τις λογικές πύλες σε όλες τις περιπτώσεις δεν απείχε πολύ από την ελάχιστη. Πιο συγκεκριμένα ποτέ δεν ξεπέρασε το 8% αυτής.

ΚΕΦΑΛΑΙΟ 6. ΜΕΛΛΟΝΤΙΚΗ ΔΟΥΛΕΙΑ

Αρχικά, στην εργασία αυτή υλοποιήσαμε δυο διαφορετικούς αλγορίθμους για την τοποθέτηση των λογικών πυλών στην επιφάνεια ενός τσιπ. Ο δεύτερος εξ αυτών, υλοποιεί μια διαμέριση του κυκλώματος με μια μέθοδο ανεύρεσης πυκνών υπογράφων, η οποία δεν έχει ξαναδοκιμαστεί. Η τοποθέτηση που επιτυγχάνει έχει ως αποτέλεσμα μεγάλο μήκος ηλεκτρικών καλωδίων στη φάση της διασύνδεσης. Έτσι λοιπόν, μια διαφορετική χωροθέτηση των επιμέρους κομματιών του κυκλώματος πάνω στο τσιπ από αυτή που υλοποιήσαμε εμείς πιθανόν να δίνει καλύτερα αποτελέσματα. Για παράδειγμα, μια διαφορετική χωροθέτηση θα μπορούσε να χρησιμοποιήσει μια ευριστική συνάρτηση που θα λάμβανε σε μεγαλύτερο βαθμό υπό όψη της τις διασυνδέσεις μεταξύ των διαφορετικών υποκυκλωμάτων. Επίσης, μια άλλη χωροθέτηση θα μπορούσε να χρησιμοποιήσει channels ή switchboxes. Εκτός των άλλων, ένας αλγόριθμος που θα μπορούσε να «μαντέψει» το καλύτερο λόγο βάρους των κόμβων προς το βάρος των ακμών εκτιμάται ότι θα βελτίωνε πολύ τα αποτελέσματα. Γενικά, υπάρχουν πολλές διαφορετικές προσεγγίσεις στον τομέα αυτό, που μπορούν να δοκιμαστούν μελλοντικά και να συγκριθούν τα αποτελέσματά τους με αυτά της παρούσας μελέτης.

Εκτός όμως από τη τοποθέτηση των λογικών πυλών, σε αυτή την εργασία υλοποιήθηκε και ένας αλγόριθμος για τη διασύνδεση τους. Αυτός δίνει πάντα τη βέλτιστη διασύνδεση όταν δεν υπάρχουν εμπόδια, ενώ όταν υπάρχουν αυτή δεν απέχει πολύ από τη βέλτιστη. Εδώ, στο μέλλον για δεδομένες τοποθετήσεις μπορούν να χρησιμοποιηθούν διάφοροι αλγόριθμοι και να συγκριθούν ως προς το μήκος των ηλεκτρικών καλωδίων που χρησιμοποιούν, ως προς την απόκλισή τους από τη βέλτιστη λύση και ως προς την ταχύτητα εκτέλεσής τους.

Τέλος, μελλοντικά θα μπορούσε να κατασκευαστεί ένα περισσότερο λειτουργικό interface, ώστε η εφαρμογή να γίνει πιο φιλική στο χρήστη και να χρησιμοποιηθούν πολυπλοκότερα σχεδιαστικά πακέτα για την καλύτερη σχεδίαση του ολοκληρωμένου κυκλώματος.

ΑΝΑΦΟΡΕΣ

[1] N. Sherwani. “Algorithms for VLSI Physical Design Automation”, 3rd ed., Kluwer Academic Publishers, Boston , MA, 1995.

[2] Z. Yang and S. Areibi. “Global Placement Techniques for VLSI Physical Design Automation”, CAINE, pp 243-247, 2002.

[3] B. Hu and M. Marek-Sadowska. “Wire length prediction-based clustering and its application in placement”, ACM/IEEE Design Automation Conference, pp 800-805, June 2003.

[4] X. Yang, M. Wang, R. Kastner, S. Ghiasi and M. Sarrafzaseh. “Congestion Reduction during Placement with Provably Good Approximation Bound”, ACM Transactions on Design Automation of Electronic Systems, Vol. 8(3), pp 316-333, July 2003.

[5] C. Chu and Y.-C. Wong. “Congestion Reduction during Placement with Provably Good Approximation Bound”, ISPD’ 05, pp 28-35, April 2005.

[6] L.-C. E. Liu, H.-P. Tseng and C Sechen. “Chip-level area routing”, ISPD 1998, pp 197-104, April 1998.

[7] Y. Kubo, H. Miyashita, Y. Kajitani and K. Tateishi. “Equidistance routing in high-speed VLSI layout design”, Integration, the VLSI Journal, Vol. 38(3), pp 439-449, January 2005.

[8] T. Deguchi, T. Koide and S. Wakabayashi. “Timing-driven hierarchical global routing with wire-sizing andbuffer-insertion for VLSI with multi-routing-layer”, Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific, pp 99-104, 2000.

[9] C.-K. Koh and P. Madden. “Manhattan or non-Manhattan?: a study of alternative VLSI routing architectures”, Great Lakes Symposium on VLSI, pp 47-52, 2000.

[10] S. Das, S.-K Susmita and B Bhattacharya. “Manhattan-diagonal routing in channels and switchboxes”, ACM Transactions on Design Automation of Electronic Systems, Vol 9(1), pp 75-104, January 2004.

[11] M. Lai.and D.F. Wong. “Slicing tree is a complete floorplan representation”, Design, Automation, and Test in Europe, pp 228-232, 2001.

[12] D.F. Wong and C.L. Liu. “A New Algorithm for Floorplan Design”, Annual ACM IEEE Design Automation Conference, pp 101-107, 1986.

[13] S. Koakutsu, M. Kang and W. W. M. Dai. “Genetic simulated annealing and application to non-slicing floorplan design”, Proc. of the 5th ACM/SIGDA Physical Design Workshop, pp 134-141, April 1996.

[14] T.C. Chen and Y.W. Chang. “Modern floorplanning based on fast simulated annealing”, International Symposium on Physical Design, pp 104-112, April 2005.

[16] M.M. Mano. “Digital Design”, 2nd ed., Prenrice-Hall, Inc, 1991

[17] Χρ. Καβουσιανός. “Σχεδίαση Ψηφιακών Συστημάτων με χρήση Η/Υ”, Πανεπιστημιακές Εκδόσεις Ιωαννίνων, Ιούνιος 2004.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. “Introdustion To Algorithms”, 2nd ed., The MIT Press, 2001.

[19] K. Shahookar and P. Mazumder. “VLSI Cell Placement Techniques”, ACM Computing Surveys, Vol. 23(2), pp 143-220, June 1991.

[20] L.R. Ford and D.R. Fulkerson. “Flows in Networks”, Princeton Univ. Press, 1962.

[21] C. M. Hoffmann, A. Lomonosov and M. Sitharam. “Finding solvable subsets of constraint graphs”,In Principles and Practice of Constraint Programming CP'97, pp 463-477, 1977

[22] C.Y. Lee. “An Algorithm for Path Connection and Its Application”, IRE Trans. Electron. Comput., pp. 346-365, Sept. 1961.

[23] F. Rubin. “The Lee Path Connection Algorithm”, IEEE Trans. on Computers, Vol 23(9), pp 907-914, Sept 1974.

[24] D. Hightower. “The Lee Router Revisited”, ICAAD, pp 136-139, 1993.

[25] http://www.diku.dk/geosteiner/

[26] http://www.ece.umn.edu/users/kia/

ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ο Δημήτρης Μαρκούζης του Παύλου και της Σιδερής γεννήθηκε στη Θεσσαλονίκη το 1981. Αποφοίτησε από το 1^{ο} Γενικό Λύκειο Φλώρινας το 1999. Σπούδασε στο Τμήμα Πληροφορικής του Πανεπιστημίου Ιωαννίνων, από όπου αποφοίτησε και έλαβε πτυχίο με βαθμό 6.57 «λίαν καλώς». Στο διάστημα 2004-2007 παρακολούθησε το μεταπτυχιακό πρόγραμμα του ίδιου Τμήματος.