ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ
Π.Μ.Σ. «Επιστήμη και τεχνολογία Υπολογιστών Τηλ/νιων και Δικτύων»

«Αυτόματη Διαχείριση περιγραφής Φυσικού Επιπέδου, Επίπεδων και μη Επίπεδων Διατάξεων Τρανζίστορ»

"Automatic Layout Manipulation of planar and no planar devices"

Μεταπτυχιακή Εργασία

Ιακώβου Αριστείδης, Προδρόμου Μαρία,
Ρουμελιώτου Κωνσταντίνα, Χαλκιάς Παναγιώτης

Επιβλέποντες Καθηγητές: Σταμούλης Γεώργιος
Ευμορφόπουλος Νέστωρας
Σωτηρίου Χρήστος

Βόλος, Μάρτιος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ
Π.Μ.Σ. «Επιστήμη και τεχνολογία Υπολογιστών Τηλ/νιων και Δικτύων»

«Αυτόματη Διαχείριση περιγραφής Φυσικού Επιπέδου, Επίπεδων και μη Επίπεδων Διατάξεων Τρανζίστορ»

Μεταπτυχιακή Εργασία

Ιακώβου Αριστείδης, Προδρόμου Μαρία,
Ρουμελιώτου Κωνσταντίνα, Χαλκιάς Παναγιώτης

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

.....	.....	.....
Γ. Σταμούλης	Ν. Ευμορφόπουλος	Χ.Σωτηρίου

Ιακώβου Α., Προδρόμου Μ., Ρουμελιώτου Κ., Χαλκιάς Π.
Διπλωματούχοι Μηχανικοί Η/Υ, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Ευχαριστίες

Ολοκληρώνοντας τις μεταπτυχιακές μας σπουδές θα θέλαμε να ευχαριστήσουμε τα άτομα που στάθηκαν δίπλα μας και μας στήριξαν κατά την διάρκεια της φοίτησης μας και κατά την εκπόνηση της παρούσας εργασίας.

Αρχικά, θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα καθηγητή μας κ. Γεώργιο Σταμούλη, καθηγητή του τμήματος Μηχανικών Υπολογιστών Τηλεπικοινωνιών και Δικτύων για την ευκαιρία που μας έδωσε να πραγματοποιήσουμε αυτήν την μελέτη, καθώς και τους επιβλέποντες καθηγητές μας κ. Νέστορα Ευμορφόπουλο και κ. Σωτηρίου Χρήστο.

Επιπρόσθετα θα θέλαμε να ευχαριστήσουμε τις αγαπημένες μας οικογένειες για την αμέριστη συμπαράσταση τους καθ' όλη την ακαδημαϊκή μας πορεία, που στάθηκαν δίπλα μας σε κάθε δυσκολία που αντιμετωπίσαμε και αποτέλεσαν το κατ' εξοχήν βασικό στήριγμά μας.

Τέλος ένα μεγάλο ευχαριστώ σε όλους τους φίλους και τα κοντινά μας πρόσωπα που με την κατανόηση και εμψύχωσή τους στάθηκαν σαν δεύτερη οικογένεια σε όλη την διάρκεια των σπουδών μας.

Προδρόμου Μαρία
Ρουμελιώτου Κωνσταντίνα
Ιακώβου Αριστείδης
Χαλκιάς Παναγιώτης

ΣΥΝΟΨΗ

Οι αυξανόμενες απαιτήσεις για υλοποίηση ολοένα και περισσότερων λειτουργιών από ένα κύκλωμα, έχει οδηγήσει σε υψηλό βαθμό ολοκλήρωσης. Κάτι τέτοιο, οδηγεί σε αναζήτηση νέων μεθόδων κατασκευής και υλοποίησης ολοκληρωμένων κυκλωμάτων. Η ανάγκη αυτή προέκυψε από την ελάττωση των διαστάσεων της τεχνολογίας που απαιτείται σε κάθε φάση εξέλιξης και βασικό μέλημα των κατασκευαστών ήταν η ταχύτητα απόκρισης τους. Τα τελευταία χρόνια, όμως, έχει παρατηρηθεί μια αυξανόμενη απαίτηση για σχεδιάσεις χαμηλής κατανάλωσης ισχύος. Έτσι, αφού ο σχεδιαστής γράψει τον κατάλληλο κώδικα, και λαμβάνοντας υπόψη του όλες τις τεχνικές προδιαγραφές που του έχουν δοθεί, καλείται να υπολογίσει την ισχύ που καταναλώνεται, προκειμένου να επιβεβαιώσει ότι η εκάστοτε σχεδίαση ανταποκρίνεται στις δεδομένες απαιτήσεις. Αυτό γίνεται έτσι ώστε να μην έχουμε προβλήματα ορθότητας κατά την τελική κατασκευή του ολοκληρωμένου κυκλώματος.

Στην παρούσα εργασία, υλοποιούμε ένα πρόγραμμα για διαχείριση ενέργειας από τυποποιημένα κύτταρα. Στόχος του εργαλείου που υλοποιήσαμε είναι η ελαχιστοποίηση κατανάλωσης ισχύος μέσω μείωσης του πλάτους των τρανζίστορ. Ικανοποιώντας όλους τους απαιτούμενους σχεδιαστικούς κανόνες (DRC), και λαμβάνοντας αρχεία εισόδου σε GDSII μορφή από βιβλιοθήκες τυποποιημένων κυττάρων της εταιρείας NangateOpenCell Library, το εργαλείο που αναπτύχθηκε στην παρούσα εργασία, είναι ικανό να παράγει μια φυσική αναπαράσταση ενός κυκλώματος με μειωμένα πλάτη τρανζίστορ. Με την τεχνική αυτή επιτυγχάνεται χαμηλή καταναλισκόμενη ισχύς και αύξηση αντοχής των συσκευών με πολύ μικρές επιπτώσεις στην απόκριση ταχύτητας.

ABSTRACT

The growing demands for the achievement of more and more processes of one circuit have led to a high level of integration. Such a thing increases the need for research of new integrated circuits' construction and materialization methods. This need emerged from the decrease of technology's dimensions, which is demanded on every phase of evolution and t he basic concern of the manufacturers was the speed response. The last years though, a growing demand for low power consumption designs, has been observed. So when the designer produces the appropriate code, considering all the technical standards, he has to calculate the consumed power in order to confirm that the design meets the given needs. That is, for avoiding any accuracy issues during the construction of the integrated circuit.

In this project we implement a program for power management by standard cells. Our goal is to achieve low power consumption by decreasing the transistors' width. By having satisfied the required designing rules (DRC) and having received input files in GDSII form from standard cell libraries of NangateOpenCell Library company, the tool that we developed in this project is capable of yielding a physical layout of a circuit with decreased transistor widths. With this technique, a low consumed power and an increased resistance of the devices is achieved, with just a small effect on the speed response.

Περιεχόμενα

1 Εισαγωγή

Οι ραγδαίες αλλαγές που συντελέστηκαν στον τομέα των επικοινωνιών και των υπολογιστών, κατά την διάρκεια της παγκόσμιας ιστορίας, έγιναν στα πλαίσια της αποκαλούμενης «ψηφιακής επανάστασης». Ήταν αυτή, η οποία καθόρισε σε πολύ μεγάλο βαθμό της μετέπειτα τεχνολογικές συνθήκες και κατόρθωσε να θέσει νέες παραμέτρους στις σύγχρονες κοινωνίες. Οι περισσότερες ερευνητικές και βιομηχανικές προσπάθειες βασίστηκαν στην αύξηση της ταχύτητας και της πολυπλοκότητας των ψηφιακών κυκλωμάτων καθώς και στην παραγωγή τους με χαμηλό κόστος. Με το πέρας των δεκαετιών, ανοίχτηκαν οι δρόμοι για μια νέα εποχή που την χαρακτηρίζουν κατά βάση ασύρματα δίκτυα που επιτυγχάνουν τεράστιες ταχύτητες, προσωπικοί υπολογιστές με δυνατότητες απεικόνισης σύνθετων γραφικών και υπολογιστικών συστημάτων, και κινητά τρίτης γενιάς που μπορούν να αναπαράγουν video και διαθέτουν πλοηγό ιστοσελίδων. Η ψηφιακή τεχνολογία έχει ενταχθεί ποικιλοτρόπως στην καθημερινότητά μας κι όλα δείχνουν πως κινείται στην κατεύθυνση πολλών επεξεργαστών πάνω σε ένα μόνο chip.

Στον πυρήνα της ψηφιακής τεχνολογίας λαμβάνουν χώρα οι διατάξεις των ολοκληρωμένων κυκλωμάτων. Δημιουργείται λοιπόν η ανάγκη τα ολοκληρωμένα κυκλώματα να λαμβάνουν ολοένα και λιγότερο χώρο ενώ οι απαιτήσεις τους για ταχύτητα να αυξάνονται. Από τα δεδομένα αυτά ανακύπτει το ζήτημα της κατανάλωσης ισχύος, η οποία πρέπει να είναι χαμηλή για να ανταποκρίνεται στις αυστηρές απαιτήσεις φορητών συσκευών και να μεγαλώνουν την διάρκεια ζωής των μπαταριών τους.

Ο υπολογισμός κι η μείωση της καταναλισκόμενης ισχύος σε ένα κύκλωμα αποτελεί ένα θέμα καίριας σημασίας κι ένα πεδίο με περαιτέρω και ραγδαίες δυνατότητες ανάπτυξης. Για το λόγο αυτό, πολλοί σχεδιαστές αλλά και εταιρείες σχεδίασης ολοκληρωμένων κυκλωμάτων προχώρησαν σε δημιουργία εργαλείων που στόχο έχουν την ελαχιστοποίηση της καταναλισκόμενης ισχύος. Επιπλέον, για τον σκοπό αυτό, προχώρησαν στην βελτιστοποίηση άλλων παραμέτρων, όπως την αλλαγή μεγεθών στα στοιχεία του κυκλώματος. Με βάση αυτές τις απαιτήσεις, έγινε η προσπάθεια υλοποίησης αλγορίθμου ενός τέτοιου εργαλείου που περιγράφεται στην παρούσα εργασία.

1.1 Αντικείμενο πτυχιακής εργασίας

Βασικός στόχος της παρούσας μεταπτυχιακής εργασίας είναι η δημιουργία ενός εργαλείου CAD για power management από τυποποιημένα κύτταρα (standard cells). Συγκεκριμένα υλοποιήθηκε ένας parser GDSII αρχείων με σκοπό την παρέμβαση και εν συνεχεία την ελαχιστοποίηση του πλάτους των τρανζίστορ με τελικό στόχο την βελτίωση της κατανάλωσης ισχύος. Τα αρχεία εισόδου που χρησιμοποιήσαμε είναι σε μορφή GDSII και η φυσική αναπαράσταση της εξόδου (layout) των ολοκληρωμένων κυκλωμάτων σε GDSII stream format έγινε μέσω του λογισμικού (layout viewer) "OwlVision GDSII Viewer"

Η υλοποίηση του αλγορίθμου πραγματοποιήθηκε σε γλώσσα προγραμματισμού C. Η ανάπτυξη του κώδικα και η αποσφαλμάτωση του έγινε σε περιβάλλον UNIX προκειμένου να βελτιστοποιηθεί η κατανάλωση μνήμης και η δυνατότητα μεταφερσιμότητας του κώδικα.

1.2 Οργάνωση του τόμου

Στο πρώτο κεφάλαιο παρατίθεται μία σύντομη περιγραφή του αντικειμένου και της δομής που θα αναλύσουμε στην παρούσα διπλωματική εργασία.

Στο δεύτερο κεφάλαιο αναλύονται και περιγράφονται κάποιες βασικές έννοιες και ορισμοί που θα συναντήσουμε και θα μας απασχολήσουν κατά την μελέτη της παρούσας μεταπτυχιακής.

Στο τρίτο κεφάλαιο παρουσιάζονται οι βασικές πηγές κατανάλωσης ισχύος και οι τεχνικές που ακολουθούνται κατά τον σχεδιασμό χαμηλής κατανάλωσης.

Στο τέταρτο κεφάλαιο περιγράφεται η μορφή και δομή των αρχείων GDSII που χρησιμοποιήθηκαν ως αρχεία εισόδου και υπέστησαν επεξεργασία από το εργαλείο που υλοποιήθηκε.

Στο πέμπτο κεφάλαιο γίνεται μια σύντομη αναφορά στην τεχνολογία κατασκευής των τρανζίστορ, στους κανόνες σχεδίασης που πρέπει να τηρηθούν και στην περιγραφή της μεθοδολογίας που ακολουθήθηκε στην παρούσα μεταπτυχιακή για επίτευξη χαμηλής κατανάλωσης

Στο έκτο κεφάλαιο αναλύονται τα βήματα του προγράμματος που υλοποιήθηκε. Γίνεται αναφορά στα εργαλεία που χρησιμοποιήθηκαν και επεξηγούνται όλες οι βιβλιοθήκες και οι συναρτήσεις που δημιουργήσαμε.

Στο έβδομο κεφάλαιο παρατίθενται κάποια πειραματικά αποτελέσματα του εργαλείου που δημιουργήθηκε καθώς και κάποιες περιπτώσεις που δεν μπορούσε να παρέμβει στις διαστάσεις του κυκλώματος για λόγους ορθής λειτουργίας.

Στο όγδοο κεφάλαιο συνοψίζονται τα συμπεράσματα της εργασίας και οι μελλοντικές επεκτάσεις του εργαλείου.

Τέλος παρουσιάζονται οι πηγές από όπου αντλήσαμε τι πληροφορίες για την υλοποίηση και ανάπτυξη του αντικειμένου της παρούσας διπλωματικής.

.

2 Βασικές Έννοιες

Η ανάπτυξη κυκλωμάτων σε ολοκληρωμένη μορφή έφερε μία πραγματική επανάσταση στην Τεχνολογία της Ηλεκτρονικής στο τέλος της δεκαετίας του ΄60 με την κατασκευή πολύπλοκων κυκλωμάτων με παθητικά και ενεργητικά στοιχεία πάνω σε ένα chip. Επιπλέον τα τελευταία χρόνια είμαστε μάρτυρες μιας συνεχούς και εκτεταμένης ανάπτυξης της μικροηλεκτρονικής και συγκεκριμένα της τεχνολογίας CMOS με βασικό στοιχείο την σμίκρυνση των ολοκληρωμένων κυκλωμάτων με σκοπό την επίτευξη υψηλότερων επιδόσεων, χαμηλότερης κατανάλωσης και μειωμένου κόστους. Η διευκόλυνση στην σχεδίαση ψηφιακών κυκλωμάτων σε συνδυασμό με την εξοικονόμηση χρόνου αλλά και την εγγυημένη λειτουργία τους οδήγησε στην δημιουργία προσχεδιασμένων βιβλιοθηκών κυττάρων που απαιτούνται από όλα τα εργαλεία CAD κατά των σχεδιασμό των chip. Παρακάτω παρατίθενται αναλυτικά οι έννοιες που προαναφέρθηκαν και θα μας απασχολήσουν κατά την μελέτη της παρούσας εργασίας.

2.1 Ολοκληρωμένα κυκλώματα

Ολοκληρωμένο κύκλωμα (integrated circuit) ή απλά ολοκληρωμένο ονομάζεται ένα κύκλωμα συνδεδεμένων λογικών πυλών, δημιουργημένο πάνω σε ένα φύλλο. Η συντριπτική πλειοψηφία των ολοκληρωμένων κυκλωμάτων δημιουργούνται πάνω σε φύλλα ημιαγωγών, κατά κύριο λόγο πυριτίου. Οι λογικές πύλες με την παρούσα τεχνολογία υλοποιούνται με παθητικά στοιχεία, οπότε τα ολοκληρωμένα κυκλώματα είναι παθητικά. Το ολοκληρωμένο κύκλωμα δεν είναι τίποτα άλλο παρά ένα πολύ προηγμένο ηλεκτρικό κύκλωμα. Ένα ηλεκτρικό κύκλωμα αποτελείται από διάφορα ηλεκτρικά στοιχεία όπως τρανζίστορ, αντιστάσεις και διόδους, τα οποία ενώνονται μεταξύ τους με διάφορους τρόπους. Συγκεκριμένα τα στοιχεία που απαρτίζουν ένα ολοκληρωμένο κύκλωμα είναι τα εξής:

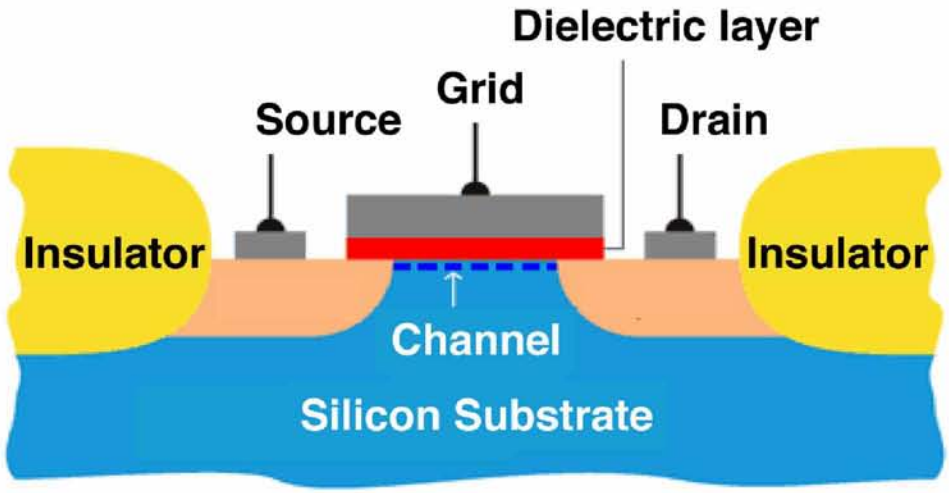
- Το τρανζίστορ δουλεύει σαν ένας διακόπτης. Μπορεί να ενεργοποιήσει (ή να απενεργοποιήσει) την ροή του ηλεκτρικού ρεύματος ή να αυξομειώσει την ένταση του ρεύματος.
- Η αντίσταση περιορίζει την ροή του ηλεκτρισμού και δίνει την ικανότητα ελέγχου της ποσότητας του ρεύματος που είναι επιτρεπτό να περάσει.

- Ο πυκνωτής συγκεντρώνει ρεύμα και το απελευθερώνει με μια γρήγορη έκρηξη.
- Η δίοδος σταματά τον ηλεκτρισμό κάτω από κάποιες συνθήκες και επιτρέπει τη διέλευση ρεύματος όταν οι συνθήκες αυτές αλλάξουν.

Τα ολοκληρωμένα κυκλώματα διακρίνονται σε τέσσερις κατηγορίες SSI (μικρή κλίμακας ολοκλήρωσης), MSI (μεσαίας κλίμακας ολοκλήρωσης), LSI (μεγάλης κλίμακας ολοκλήρωσης) και VLSI (πολύ μεγάλης κλίμακας ολοκλήρωσης), ανάλογα με τον αριθμό των λογικών πυλών από τα οποία αποτελούνται.

2.1.1 CMOS

Αναφερθήκαμε νωρίτερα στη χρήση της τεχνολογίας CMOS χωρίς να αναλύουμε σε τι συνίσταται αυτή. Το CMOS είναι αρκτικόλεξο του **C**omplementary **M**etal **O**xide **S**emiconductor δηλαδή συμπληρωματικός ημιαγωγός μετάλλου-οξειδίου. Η λέξη συμπληρωματικός αναφέρεται στην συμπληρωματική συμμετρία ζευγών MOSFET p και n τύπου, από τα οποία μόνο το ένα άγει κάθε φορά. Το κομμάτι του ονόματος «ημιαγωγός μετάλλου-οξειδίου» είναι κατάλοιπο των πρώτων διαδικασιών κατασκευής κυκλωμάτων CMOS, όπου τα FET κατασκευάζονταν υλοποιώντας το ηλεκτρόδιο πύλης με μέταλλο τοποθετημένο πάνω σε ένα οξείδιο που λειτουργούσε ως μονωτής, και το οποίο ήταν με τη σειρά του τοποθετημένο πάνω σε ημιαγωγό. Στις σημερινές διαδικασίες CMOS ως ηλεκτρόδιο πύλης χρησιμοποιείται πολυπυρίτιο, παρόλο που η ονομασία διατηρείται. Κύριοι λόγοι της ευρείας χρήσης της λογικής CMOS στις σύγχρονες σχεδιάσεις και υλοποιήσεις είναι η μεγάλη ανοχή στο θόρυβο, η χαμηλή κατανάλωση στατικής ισχύος, και η δυνατότητα που παρέχει για αρκετά «πυκνές» σχεδιάσεις. Ο όρος CMOS αναφέρεται τόσο στη λογική σχεδίαση, όσο και στη διαδικασία κατασκευής του συγκεκριμένου κυκλώματος.



Εικόνα: Τρανζίστορ MOS

2.1.2 Τεχνολογία κατασκευής ολοκληρωμένων κυκλωμάτων

Η διαδικασία κατασκευής ολοκληρωμένων κυκλωμάτων περιλαμβάνει ένα σύνολο χημικών διεργασιών και υλικών σε κατάλληλες συνθήκες, οι οποίες οδηγούν στη δημιουργία ημιαγώγιμων δομών που αποτελούν τα κυκλώματα. Σκοπός της παραγράφου είναι μια συνοπτική αναφορά της διαδικασίας και των χημικών διεργασιών που ακολουθούνται καθώς και μια εισαγωγή σε ορισμένους περιορισμούς σχεδίασης που υπαγορεύει η συγκεκριμένη τεχνολογία.

Προετοιμασία Δισκίου

Το βασικό ακατέργαστο υλικό που χρησιμοποιείται στα εργοστάσια κατασκευής ημιαγωγών είναι ένα δισκίο πυριτίου (wafer) με διάμετρο μεταξύ 75 και 230 χιλιοστών και πάχος μικρότερο από 1 χιλιοστό. Τα wafers προέρχονται από τον τεμαχισμό των ράβδων μονοκρυσταλλικού πυριτίου που παίρνουμε με τη μέθοδο Czochralski. Το κόψιμο των ράβδων σε δισκία πραγματοποιείται με τη χρήση λεπίδων διαμαντιού και το πάχος τους κυμαίνεται από 0.25 έως 1.0 χιλιοστών. Τα δισκία λειαίνονται και πάνω σε αυτά με κατάλληλες χημικές διεργασίες και μάσκες δημιουργούνται τα ολοκληρωμένα.

Οξείδωση

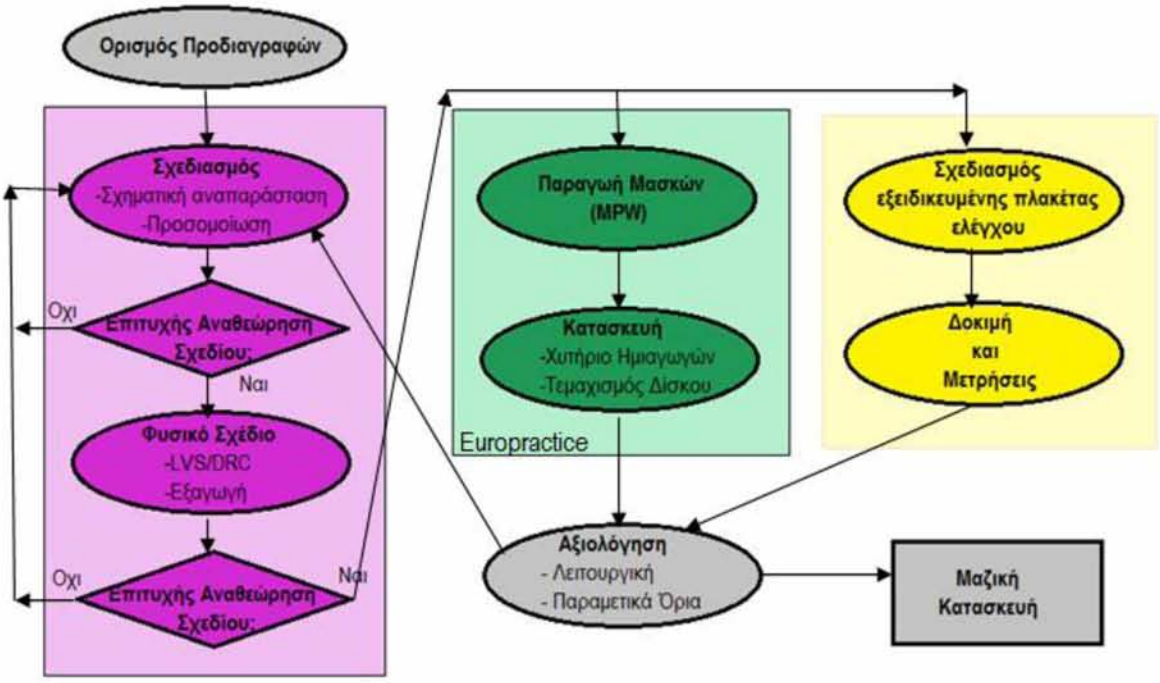
Η οξείδωση του πυριτίου επιτυγχάνεται με την θέρμανση των δισκίων του πυριτίου μέσα σε κατάλληλη ατμόσφαιρα οξείδωσης, για παράδειγμα σε οξυγόνο ή ατμούς νερού. Οι πιο κοινές μέθοδοι είναι η υγρή οξείδωση και η ξηρή οξείδωση. Η διαδικασία αυτή είναι πολύ σημαντική καθώς το οξείδιο του πυριτίου είναι μονωτικό και προστατεύει το υπόστρωμα από νοθεύσεις και από ακαθαρσίες.

Επίταξη, Απόθεση, Εμφύτευση Ιόντων και Διάχυση

Για την δημιουργία διαφόρων ημιαγώγιμων στοιχείων απαιτείται το πυρίτιο να περιέχει διάφορες αναλογίες προσμίξεων από δότες ή αποδέκτες. Η επίταξη περιλαμβάνει την ανάπτυξη μιας μονοκρυσταλικής μεμβράνης πάνω στην επιφάνεια του πυριτίου. Η απόθεση περιλαμβάνει την εξάτμιση υλικού νόθευσης πάνω στην επιφάνεια του πυριτίου ακολουθούμενη από ένα κύκλο θέρμανσης που σαν συνέπεια έχει να οδηγηθούν οι προσμίξεις από την επιφάνεια στο κύριο σώμα. Στην εμφύτευση ιόντων το υπόστρωμα του πυριτίου υπόκεινται στην επίδραση ατόμων υψηλής ενέργειας δοτών ή αποδεκτών. Διάχυση είναι η διαδικασία με την οποία τα άτομα κινούνται μέσα στο κρυσταλλικό πλέγμα.. Αφού δημιουργηθούν οι περιοχές νόθευσης είναι σημαντική η διατήρηση της θερμοκρασίας σε όσο το δυνατό χαμηλότερα επίπεδα.

Ο τύπος της νόθευσης εξαρτάται από το υλικό νόθευσης (p-τύπου βόριο, n-τύπου αρσενικό, φώσφορος), ενώ η συγκέντρωση των φορέων εξαρτάται από το χρόνο

έκθεσης του πυριτίου στο υλικό, τη θερμοκρασία και στην περίπτωση της εμφύτευσης ιόντων στην ενέργεια. Το που θα δημιουργηθούν οι περιοχές συγκέντρωσης φορέων καθορίζεται από τις μάσκες, οι οποίες με την ιδιότητά τους να φράσσουν τη νόθευση του υλικού (επιλεκτική νόθευση) μας επιτρέπουν να δημιουργούμε πάνω στο πυρίτιο περιοχές με τα γεωμετρικά χαρακτηριστικά της μάσκας, οι οποίες έχουν τις ηλεκτρικές ιδιότητες που θέλουμε. Τα κύρια υλικά που χρησιμοποιούνται ως μάσκες είναι το φωτοευαίσθητο υλικό, το πολυπυρίτιο, το διοξείδιο του πυριτίου (SiO_2) και το νιτρίδιο του πυριτίου (SiN). Με τη χρήση των διαδικασιών αυτών και των κατάλληλων μασκών είναι δυνατή η δημιουργία όλων των περιοχών που αποτελούν τα CMOS κυκλώματα.



Εικόνα: Διαδικασία κατασκευής ενός ολοκληρωμένου κυκλώματος

2.2 Τυποποιημένα κύτταρα (Standard Cell)

Στη σχεδίαση ημιαγωγών, η μέθοδος standard cell αποτελεί μια μέθοδο σχεδιασμού ASIC κυκλωμάτων κυρίως με ψηφιακά-λογικά χαρακτηριστικά. Αποτελεί ένα παράδειγμα αφαιρετικού σχεδιασμού (design abstraction), όπου ένα χαμηλού επιπέδου layout ενσωματώνεται σε μια αφαιρετική λογική αναπαράσταση, όπως η πύλη NAND. Η μέθοδος βασιζόμενη στα cells (cell based methodology) παρέχει τη δυνατότητα σε ένα σχεδιαστή να εστιάζει σε υψηλού επιπέδου πτυχή της ψηφιακής σχεδίασης (logical function) ενώ παράλληλα κάποιος άλλος μπορεί να εστιάζει στην υλοποίηση (physical). Παράλληλα με την πρόοδο στην κατασκευή ημιαγωγών, η

βασισμένη στα standard cells μέθοδος ήταν υπεύθυνη για την κλιμάκωση των κυκλωμάτων ASIC από συγκριτικά απλά ολοκληρωμένα κυκλώματα, τα οποία εκτελούν μια συγκεκριμένη λειτουργία (αποτελούμενα από αρκετές χιλιάδες πύλες), σε σύνθετες συσκευές αποτελούμενες από πολλά εκατομμύρια πύλες (SoC-System on chip).

2.2.1 Περιγραφή Standard Cell Design

Στο Standard Cell Design παρέχεται στον σχεδιαστή μια βιβλιοθήκη και προχωράει στον σχεδιασμό με την διασύνδεση των στοιχείων που περιέχονται σ' αυτή την βιβλιοθήκη, με όλα τα κομμάτια-σχέδια, που δημιουργούνται, να βασίζονται στα έτοιμα σχεδιαστικά κομμάτια αυτής. Η βιβλιοθήκη αυτή παρέχεται από τον κατασκευαστή του ολοκληρωμένου κυκλώματος και μπορεί να περιέχει από πολύ λίγα έως πάρα πολλά και πολύ σύνθετα σχεδιαστικά κομμάτια. Μια standard cell based βιβλιοθήκη θα μπορούσε να περιέχει μόνο τη λογική πύλη NAND δύο εισόδων μιας και κάθε λογική συνάρτηση μπορεί να εκφραστεί συναρτήσει αυτής της πύλης. Όμως οι βιβλιοθήκες με τις οποίες εφοδιάζονται οι σχεδιαστές σήμερα, χάριν διευκόλυνσής τους, περιέχουν όλες τις λογικές πύλες (σε διάφορες εκδόσεις ταχύτητας, εμβαδού και οδηγητικής ικανότητας), στοιχεία μνήμης, μικρά έως μεσαία συνδυαστικά κυκλώματα μικρά έως μεσαία ακολουθιακά κυκλώματα (καταχωρητές, ολισθητές, μετρητές κλπ).

Οι τυποποιημένες βιβλιοθήκες cell απαιτούνται από σχεδόν όλα τα εργαλεία CAD για το σχεδιασμό των chip. Πρότυπες βιβλιοθήκες cell περιέχουν πρωτόγονα κύτταρα που απαιτούνται για την ψηφιακή σχεδίαση. Ωστόσο, μπορούν επίσης να συμπεριληφθούν και πιο σύνθετα cells
Ο κύριος σκοπός των CAD εργαλείων είναι η εφαρμογή της λεγόμενης RTL-to-GDS ροή. Το τελικό αποτέλεσμα από τη διαδικασία του σχεδιασμού είναι η πλήρης διάταξη ενός τσιπ, ως επί το πλείστον στην μορφή GDSII. Για να παραχθεί ένας λειτουργικά σωστός σχεδιασμός που πληροί όλες τις προδιαγραφές και τους περιορισμούς, απαιτείται ένας συνδυασμός διαφορετικών εργαλείων στο σχεδιασμό των ροών. Τα εργαλεία αυτά απαιτούν συγκεκριμένες πληροφορίες σε διάφορες μορφές για κάθε ένα από τα cells των βιβλιοθηκών που τους παρέχονται για τον σχεδιασμό.

2.2.2 Σύνθεση Standard Cell

Ένα standard cell απαρτίζεται από ένα σύνολο τρανζίστορ και διασυνδεδεμένων δομών τα οποία αναπαριστούν μία Boolean συνάρτηση ή μια συνάρτηση αποθήκευσης. Τα απλά cell είναι αναπαραστάσεις στοιχειωδών Boolean συναρτήσεων ενώ χρησιμοποιούνται και πιο πολύπλοκα όπως ένας πλήρης αθροιστής. Μια λογική συνάρτηση cell ονομάζεται λογική αναπαράσταση (logical

view) και η λειτουργική συμπεριφορά τους περιγράφεται από έναν πίνακα αληθείας ή από μία εξίσωση ή από έναν πίνακα μετάβασης.

Το σχέδιο ενός standard cell αναπτύσσεται σε επίπεδο transistor και αποτελεί τη μορφή netlist του transistor που είναι μια περιγραφή transistors, που λαμβάνουν χώρα στο σχέδιο, των συνδέσεων μεταξύ τους και με το εξωτερικό περιβάλλον. Για την προσομοίωση της ηλεκτρονική συμπεριφορά της μορφής netlist του αρχικού σχεδίου, χρησιμοποιούνται CAD προγράμματα από σχεδιαστές δηλώνοντας διαφορές παραμέτρους εισαγωγής και υπολογίζοντας την απόκριση του κυκλώματος. Επιπλέον απαιτείται και η φυσική αναπαράσταση του standard cell η οποία ονομάζεται layout view και από κατασκευαστικής άποψης αποτελεί την πιο σημαντική αφού μοιάζει με ένα ακριβές «αποτύπωμα» του, οργανωμένο σε επίπεδα μετάλλων. Για κάθε μορφή netlist, μπορεί να υπάρξουν πολλά διαφορετικά layouts, τα οποία συμβαδίζουν με τις παραμέτρους απόδοσης της netlist. Στόχος κάθε σχεδιαστή είναι η ελαχιστοποίηση του κόστους κατασκευής του layout, λαμβάνοντας υπόψη τις διάφορες απαιτήσεις, σχετικές με την ταχύτητα και την απόδοση ισχύος του cell, μια αρκετά επίπονη διαδικασία.

Τέλος, τα στοιχεία ενός standard cell έχουν συνήθως όλα ένα σταθερό ύψος (height). Για τον λόγο αυτό, δίνεται η δυνατότητα να μπορούν να τοποθετηθούν το ένα δίπλα στο άλλο, ώστε να επιτυγχάνεται η μεταξύ τους διασύνδεση στα πλαίσια ενός πιο πολύπλοκου κυκλώματος. Συνεπώς, η έκταση του standard cell στο chip αποτελείται από ένα μεγάλο αριθμό cells τοποθετημένα στη σειρά με τη τροφοδοσία και τη γείωση συνδεδεμένες στο πάνω και στο κάτω μέρος του συνολικού χώρου. Η επεξεργασία στα επιμέρους στοιχεία γίνεται από ειδικά εργαλεία και εξαρτάται από την λογική του κυκλώματος που θα υλοποιηθεί.

2.3 Εργαλεία CAD

Με το όρο CAD (Computer Aided Design) αναφερόμαστε στην χρήση συστημάτων πληροφορικής με σκοπό να βοηθήσουν στην δημιουργία, τροποποίηση, ανάλυση ή βελτιστοποίηση μίας σχεδίασης.

Την δεκαετία του ΄80 η εξέλιξη των εργαλείων CAD συνέβαλλε στην μείωση της ανάγκης για εξειδικευμένο προσωπικό ειδικά στις μικρομεσαίες επιχειρήσεις. Η απόκτηση σε προσιτές τιμές και η ικανότητα χρήσης των

προγραμμάτων CAD στον προσωπικό υπολογιστή από τις εταιρείες ή από έναν μέσο χρήστη - εξαλείφοντας παραδοσιακές χειρωνακτικές μεθόδους σχεδίασης - βοήθησε σε μεγάλο βαθμό την εξέλιξη και την ανάπτυξη πολλών κλάδων της μηχανικής.

Σήμερα διατίθενται πολλές μορφές προγραμμάτων CAD, οι οποίες εκτείνονται από αυτές που βοηθούν την σχεδίαση επιμέρους τμημάτων έως αυτές που επιτρέπουν την παράσταση ολόκληρης της δομής του προϊόντος στον υπολογιστή. Όταν ολοκληρωθεί η αρχική σχεδίαση τα αποτελέσματα πρέπει να ελεγχτούν ως προς τις αρχικές προδιαγραφές. Παραδοσιακά προτού αναπτυχθούν τα προγράμματα σχεδίασης CAD, το βήμα αυτό περιελάμβανε την κατασκευή ενός φυσικού μοντέλου του σχεδιασμένου προϊόντος. Σήμερα κάτι τέτοιο είναι εξαιρετικά σπάνιο καθώς τα προγράμματα CAD επιτρέπουν στους σχεδιαστές να προσομοιώνουν την συμπεριφορά απίστευτα περίπλοκων προϊόντων καθώς και παρέχουν ελέγχους αν το σχέδιο πληροί τις καθορισμένες προδιαγραφές. Αν διαπιστωθεί η ύπαρξη σφαλμάτων, τότε εκτελούνται οι αναγκαίες αλλαγές και το νέο σχέδιο ελέγχεται εκ νέου. Όταν η προσομοίωση δείξει ότι το σχέδιο είναι σωστό, τότε κατασκευάζεται ένα πλήρες φυσικό πρότυπο του αντικειμένου.

2.4 Εργαλεία EDA

Τα εργαλεία αυτοματοποιημένης ηλεκτρονικής σχεδίασης EDA (Electronic Design Automation) αποτελούν μια κατηγορία εργαλείων λογισμικού τα οποία έχουν δημιουργηθεί για την σχεδίαση ηλεκτρονικών συστημάτων όπως τα ολοκληρωμένα κυκλώματα. Η εν σειρά χρησιμοποίηση των εργαλείων αυτών για την παραγωγή της τελικής σχεδίασης στοιχειοθετεί μια ροή σχεδιασμού.

2.4.1 Ιστορία του EDA

Πριν την ανάπτυξη των EDA, ο σχεδιασμός των ολοκληρωμένων κυκλωμάτων γινόταν με το χέρι. Τα κυκλώματα απεικονίζονταν γραφικά και πιο ειδικά στην μετατροπή της ηλεκτρονικής περιγραφής ενός κυκλώματος σε γραφική αναπαράσταση του. Η πιο γνωστή εταιρεία της εποχής ήταν η Calma η οποία δημιούργησε το GDSII format το οποίο χρησιμοποιείται μέχρι και σήμερα.

Στα μέσα του ΄70 οι προγραμματιστές ξεκίνησαν να αυτοματοποιούν και τον σχεδιασμό και όχι μόνο την σύνταξη και με αυτόν τον τρόπο δημιουργήθηκαν τα πρώτα εργαλεία χωροθέτησης. Η επόμενη εποχή ξεκίνησε με την δημοσίευση των Carver Mead και Lynn Conway το 1980 του "Introduction to VLSI Systems". Το πρωτοποριακό αυτό κείμενο υποστήριζε τον σχεδιασμό των chip με γλώσσες προγραμματισμού καταρτισμένες σε πυρίτιο. Το άμεσο αποτέλεσμα ήταν μία αύξηση

της πολυπλοκότητας των chip που θα μπορούσαν να σχεδιάζονται με βελτιωμένη πρόσβαση σε εργαλεία επαλήθευσης που χρησιμοποιούν λογική προσομοίωση.

Παρόλα αυτά τα πρώτα εργαλεία EDA εμφανίστηκαν σε ακαδημαϊκά πλαίσια. Το VLSI Tools Tarball που αναπτύχθηκε στο Πανεπιστήμιο του Berkley ήταν ένα σύνολο εργαλείων σε περιβάλλον Unix για τον σχεδιασμό VLSI συστημάτων. Η συστηματική ανάπτυξη των EDA συνεχίστηκε μέχρι που εταιρείες όπως η Daisy Systems, Mentor Graphics και Valid Logic Systems αποφάσισαν να αφοσιωθούν αποκλειστικά στην παραγωγή των EDA. Από τις αρχές, λοιπόν, της δεκαετίας του ΄80 ξεκινά η βιομηχανία των EDA και συνεχίστηκε με την εμφάνιση δύο υψηλού επιπέδου γλωσσών σχεδιασμού την VHDL και Verilog.

Οι σύγχρονες ροές σχεδίασης ολοκληρωμένων κυκλωμάτων αποτελούνται πλέον από πολλαπλά βήματα σε καθένα από το οποία χρησιμοποιείται διαφορετικό εργαλείο.

2.4.2 Τομείς εφαρμογής EDA

Οι σημαντικότεροι τομείς εφαρμογής των εργαλείων EDA για την σχεδίαση ενός ολοκληρωμένου κυκλώματος είναι οι ακόλουθοι:

- Σχεδιασμός
  -High Level Synthesis
  -Logic Synthesis
  -Schematic Capture
  -Layout

- Προσομοίωση
  -Transistor Simulation
  -Logic Simulation
  -Hardware Emulation
  -Technology CAD

- Ανάλυση και επαλήθευση
  -Functional Verification
  -Clock Domain Crossing Verification
  -Formal Verification
  -Equivalence Checking
  -Static Timing Analysis
  -Physical Verifications

- Κατασκευή
  -Mask Data Preparation

3 Κατανάλωση Ισχύος Στα Ολοκληρωμένα Κυκλώματα CMOS

Η ανάγκη για χαμηλότερη κατανάλωση ισχύος αποτελεί σήμερα μέρος του βασικού κορμού της σχεδίασης ψηφιακών ολοκληρωμένων κυκλωμάτων και επηρεάζει όλα τα στάδια σχεδίασης ενός συστήματος.

Ισχύς είναι ο ρυθμός με τον οποίο καταναλώνεται η ενέργεια. Σε ένα ολοκληρωμένο κύκλωμα η ηλεκτρική ενέργεια μετατρέπεται σε θερμότητα, η οποία πρέπει να απαχθεί προς αποφυγήν ανόδου της θερμοκρασίας του κυκλώματος που μπορεί να οδηγήσει σε βλάβες.

Κατά την διάρκεια σχεδίασης ενός ολοκληρωμένου κυκλώματος οι διαδικασίες που σχετίζονται με την κατανάλωση ισχύος είναι η ανάλυση και η βελτιστοποίηση. Η ανάλυση γίνεται σε κάθε φάση της σχεδίασης και στόχος της είναι η αύξηση εμπιστοσύνης στο σχεδιασμό και η εξασφάλιση ότι οι προδιαγραφές κατανάλωσης ισχύος δεν παραβιάζονται. Βελτιστοποίηση είναι η διαδικασία δημιουργίας του καλύτερου σχεδίου χωρίς να παραβιάζονται οι προδιαγραφές που έχουν τεθεί. Υπάρχουν διάφορες τεχνικές που μπορούν να εφαρμοστούν και ανάλογα με την περίπτωση παράγουν και διαφορετικά αποτελέσματα.

3.1 Ανάγκη για χαμηλή κατανάλωση

Η ανάγκη για χαμηλή κατανάλωση προκύπτει από την ολοένα αυξανόμενη ολοκλήρωση. Σύμφωνα με τον νόμο του Moore η χωρητικότητα (πλήθος στοιχείων) των ολοκληρωμένων κυκλωμάτων διπλασιάζεται κάθε δεκαοχτώ μήνες με αποτέλεσμα να αυξάνει η πυκνότητα ισχύος των συστημάτων και άρα η ανάγκη για απαγωγή θερμότητας και μείωσης της κατανάλωσης.

Ένας άλλος παράγοντας στην σχεδίαση κυκλωμάτων CMOS με χαμηλή κατανάλωση ισχύος αλλά με υψηλή απόδοση είναι η αυξανόμενη ζήτηση για φορητά συστήματα που λειτουργούν με μπαταρίες. Με δεδομένο ότι η διάρκεια ζωής της μπαταρίας είναι ο βασικότερος παράγοντας κατά το σχεδιασμό ενός φορητού συστήματος και ότι η πρόοδος στις μπαταρίες δεν είναι η αναμενόμενη σε σχέση με την ανάγκη που

προβλέπεται από τον νόμο του Moore, η χαμηλή κατανάλωση μετατρέπεται σε κρίσιμο στοιχείο της σχεδίασης.

Ένας τελευταίος λόγος είναι η αυξανόμενη απόδοση των υπολογιστικών συστημάτων μέσω της αύξησης της συχνότητας λειτουργίας. Η ισχύς είναι ανάλογη ης συχνότητας λειτουργίας και στους σημερινούς επεξεργαστές υψηλής απόδοσης μπορεί να φτάσει τα δεκάδες Watts.

3.2 Βασικές πηγές κατανάλωσης

Η ισχύς που καταναλώνεται σε ένα κύκλωμα CMOS αποτελείται από δύο συνιστώσες: την στατική κατανάλωση ισχύος λόγω ρεύματος διαρροής ή άλλων ρευμάτων που ρέουν συνεχώς από την τροφοδοσία ισχύος και την δυναμική κατανάλωση λόγω του ρεύματος μεταγωγής και της φόρτισης και εκφόρτισης των χωρητικών φορτίων.

3.2.1 Συνολική κατανάλωση

Υπάρχουν τρείς πηγές κατανάλωσης ισχύος στα κυκλώματα CMOS, η δυναμική κατανάλωση, η στατική κατανάλωση και η κατανάλωση βραχυκυκλώματος. Η συνολική κατανάλωση ισχύος προκύπτει από την άθροιση αυτών των τριών συνιστωσών :

$$P_{total} = P_{dynamic} + P_{static} + P_{sortcircuit}$$

Όπου $P_{dynamic}$ είναι η δυναμική κατανάλωση που οφείλεται στην δραστηριότητα του κυκλώματος κατά την οποία φορτίζονται και αποφορτίζονται οι παρασιτικές χωρητικότητες κατά την διάρκεια των μεταβάσεων σήματος στους εσωτερικούς κόμβους. Η P_{static} οφείλεται σε διαρροές ρεύματος και εξακολουθεί να υφίσταται ακόμη και αν δεν υπάρχει δραστηριότητα στο κύκλωμα (π.χ. είναι σε κατάσταση αναμονής). Η $P_{sortcircuit}$ είναι η ισχύς που καταναλώνεται κατά τη μετάβαση ενός σήματος εισόδου όπου και το Pmos και το Nmos δικτύωμα μιας πύλης CMOS μπορεί να άγουν ταυτόχρονα.

3.2.2 Στατική κατανάλωση

Ως στατική ισχύ χαρακτηρίζουμε την ισχύ που καταναλώνει μια πύλη όταν αυτή είναι αδρανής ή στατική. Στα CMOS «ιδανικά» καταναλώνεται μηδενική στατική ισχύ αφού στην κατάσταση ισορροπίας τους δεν υπάρχει μονοπάτι που να συνδέει την πηγή με την γείωση. Στην πραγματικότητα όμως, πάντοτε υπάρχουν έστω και πολύ μικρές τιμές ρευμάτων διαρροής. Το μεγαλύτερο ποσοστό στατικής ισχύος οφείλεται σε ένα δυναμικό που αναπτύσσεται ανάμεσα στην πηγή και την γείωση, το οποίο

προκαλείται από ελαττωμένα δυναμικά κατωφλίου τα οποία εμποδίζουν την εκάστοτε πύλη από το να κλείσει εντελώς.

Τα χαρακτηριστικά αυτά αποδίδονται στη λειτουργία των ημιαγωγών και την ιδιότητα των περιοχών διαχύσεων να σχηματίζουν ανάστροφα πολωμένες διόδους με το υπόστρωμα και άρα να μην άγουν σε ανάστροφη πόλωση. Γνωρίζουμε ότι στις διόδους με ανάστροφη πόλωση διαπερνούν μικρά ρεύματα διαρροής σύμφωνα με την εξίσωση :

$$i_o = i_s\,(e^{qV/kT}-1)$$

όπου

i_s : ανάστροφο ρεύμα κόρου

V : τάση διόδου

q : φορτίο ηλεκτρονίου ($1.602\text{x}10^{-19}$ C)

k: σταθερά του Boltzman ($1.38\text{x}10^{-23}$ J/K)

T : θερμοκρασία

Η στατική κατανάλωση ισχύος είναι το γινόμενο μεταξύ ρεύματος διαρροής του στοιχείου και τάσης τροφοδοσίας. Για κάθε στοιχείο το ρεύμα διαρροής θα πρέπει να κυμαίνεται μεταξύ 0.1 nA και 0.5 nA για θερμοκρασία δωματίου. Αν το κύκλωμα αποτελείται από n στοιχεία, η ολική κατανάλωση ισχύος του κυκλώματος δίνεται από τον τύπο:

$$\mathbf{P_s}= \sum_{1}^{n} i_o \times V$$

3.2.3 Δυναμική κατανάλωση

Δυναμική ονομάζεται η ισχύς που καταναλώνεται όταν μία πύλη είναι ενεργή. Ένα κύκλωμα είναι ενεργό όταν οι τάσεις του δικτύου εναλλάσσονται λόγω κάποιου εξωτερικού ερεθίσματος που εφαρμόζουμε στην έξοδο. Επειδή η τάση στην είσοδο μπορεί να αλλάξει, χωρίς αυτό να συνεπάγεται κάποια λογική μεταβολή στην έξοδο, δυναμική ισχύς καταναλώνεται κι όταν η έξοδος δεν αλλάζει την λογική της κατάσταση.

Η δυναμική ισχύς που καταναλώνεται σε ένα κύκλωμα δίνεται από τον τύπο:

$$P_d = C_L V_{DD}^2 f_p$$

Όπου:

C_L: χωρητικό φορτίο εξόδου

V_{DD} : τάση τροφοδοσίας

f_p : συχνότητα παλμού εισόδου

Εάν θέλουμε να αναφερθούμε σε μεγαλύτερο πλήθος κύκλων ρολογιού θα πρέπει να συμπεριλάβουμε και τον παράγοντα μεταβάσεων (**switching activity**) '**a**' ο οποίος υπολογίζει τον αριθμό μεταβάσεων ανά κύκλο ρολογιού. Κι ο τύπος που προκύπτει είναι ο εξής:

$$P_d = a\, C_L V_{DD}^2 f_p$$

Συνεπώς η μέση δυναμική ισχύς που καταναλώνεται είναι ανάλογη της ενέργειας που απαιτείται για την φόρτιση και την εκφόρτιση της χωρητικότητας του κυκλώματος. Επίσης, προκύπτει ότι η ισχύς είναι ανάλογη της συχνότητας μεταγωγής και ανεξάρτητη των παραμέτρων ενός στοιχείου.

3.2.4 Κατανάλωση βραχυκυκλώματος

Στα στατικά κυκλώματα CMOS, υπάρχει μια χρονική περίοδος κατά την μετάβαση των σημάτων εισόδου όπου και το πάνω και το κάτω δικτύωμα είναι συγχρόνως σε αγώγιμη κατάσταση οπότε σχηματίζουν ένα μονοπάτι ρεύματος DC μεταξύ τροφοδοσίας και γείωσης. Το ρεύμα DC που διαρρέει ένα CMOS κύκλωμα κατά την διάρκεια μετάβασης του σήματος εισόδου (κατά την διάρκεια μη-μηδενικών χρόνων ανόδου και καθόδου των σημάτων εισόδου) λέγεται ρεύμα βραχυκυκλώματος.

Η κατανάλωση ισχύος βραχυκυκλώματος δίδεται από την σχέση:

$$P_{sc} = I_{mean} * V_{DD}$$

Όπου:

I_{mean} : ρεύμα βραχυκυκλώματος

V_{DD} : τάση τροφοδοσίας

Το ρεύμα βραχυκυκλώματος παρατηρείται προσωρινά κατά τη διάρκεια μετάβασης του σήματος εισόδου και εξαρτάται τόσο από το χωρητικό φορτίο όσο και από τις γεωμετρικές διαστάσεις της πύλης.

3.3 Βασικές αρχές σχεδίασης

Οι τεχνικές που ακολουθούνται κατά τον σχεδιασμό χαμηλής ισχύος κατηγοριοποιούνται σε συντηρητικές και συμβιβαστικές. Η συντηρητική σχολή προσπαθεί να μειώσει την ισχύ, η οποία καταναλώνεται χωρίς σοβαρό λόγο. Κατά την σχεδίαση αναλύονται και ελαχιστοποιούνται οι απώλειες. Η άλλη σχολή εξετάζει εναλλακτικούς τρόπους λογικής σχεδίασης που μειώνουν την κατανάλωση. Πρέπει να τονισθεί ότι δεν υπάρχει μια τεχνική που να εφαρμόζεται αποτελεσματικά σε όλες τις εφαρμογές. Οι περιορισμοί στην σχεδίαση αυξάνονται με λεπτομέρεια στα πλαίσια των προδιαγραφών.

Κάποιες από τις τεχνικές που χρησιμοποιούνται σήμερα για χαμηλή κατανάλωση ισχύος φαίνονται στον παρακάτω πίνακα:

Traditional Techniques	Dynamic Power Reduction	Leakage Power Reduction	Other Power Reduction techniques
Clock Gating	Clock Gating	Minimize Usage of low V_t cells	Multi oxide devices
Power Gating	Power Efficient Circuits	Power Gating	Minimize capacitance by custom design
Variable Frequency	Variable Frequency	Back Biasing	Power Efficient Circuits
Variable Voltage Supply	Variable Voltage Supply	Reduce Oxide Thickness	
Variable Device Threshold	Voltage Islands	Use FinFET	

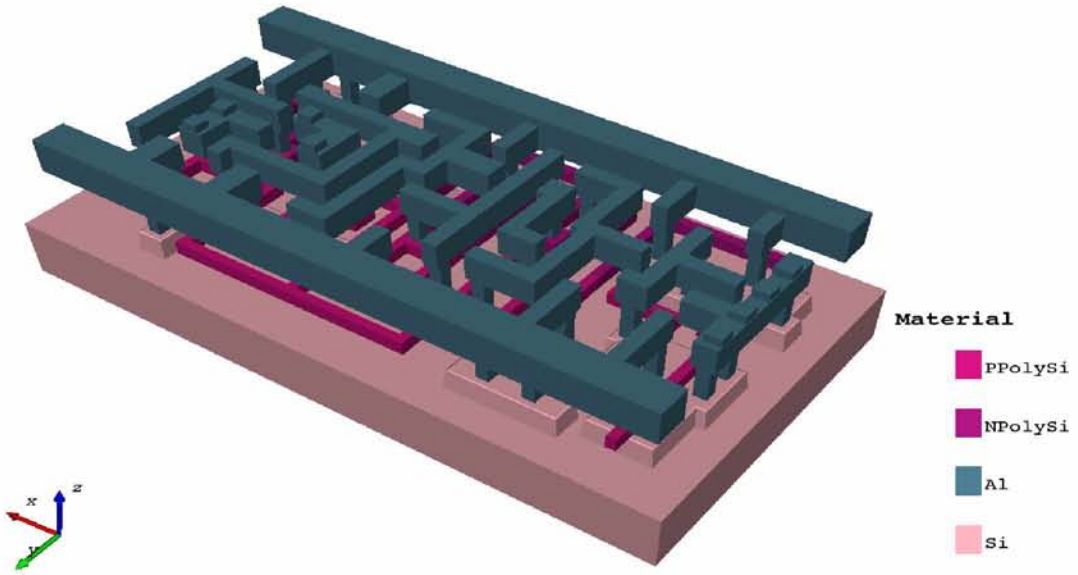
Σίγουρα αυτό που πρέπει να τονισθεί είναι ότι, χρησιμοποιώντας μια οποιαδήποτε τεχνική βελτίωσης ερχόμαστε αυτομάτως αντιμέτωποι και με κάποιο κόστος σε άλλες παραμέτρους του κυκλώματος όπως παραδείγματος χάριν στο χρονισμό του, στην περιοχή που αυτό καταλαμβάνει και στην πολυπλοκότητα υλοποίησής του.

4 Δομή Αρχείων GDSII

4.1 Εισαγωγικά

Για πολλά χρόνια στον σχεδιασμό ολοκληρωμένων κυκλωμάτων το δημοφιλέστερο σχήμα για ανταλλαγή ήταν το GDS II (Graphic Data System) stream format αναπτυγμένο από την Calma Company. Υπήρξε το μοναδικό στο είδος του και πολλοί προμηθευτές το χρησιμοποιούσαν στα συστήματα τους.

Στην παρούσα υλοποίηση του αλγορίθμου χρησιμοποιούμε όπως προαναφέρθηκε, αρχεία GDSII ως αρχεία εισόδου. Πρόκειται για αρχεία δυαδικής μορφής, που αντιπροσωπεύουν ιεραρχικά, τις επίπεδες γεωμετρικές μορφές, τις ετικέτες κειμένων, και άλλες πληροφορίες για το Layout. Τα στοιχεία είναι τοποθετημένα σε στρώματα (layers). Είναι ένα δυαδικό σχήμα που είναι ανεξάρτητο πλατφορμών, καθώς χρησιμοποιεί εσωτερικά καθορισμένα σχήματα για τους τύπους δεδομένων του. Διαβάζοντας αρχεία GDSII, οι εσωτερικοί τύποι δεδομένων GDSII (όπως reals, integers κ.λπ.) πρέπει να μετατραπούν σε platform/CAE package. Το σχήμα GDSII είναι ένας διαδοχικός κατάλογος αρχείων, όπου κάθε αρχείο χαρακτηρίζεται από έναν header (επικεφαλίδα) που περιέχει τις πληροφορίες που βρίσκονται στο αρχείο. Η δομή του αρχείου είναι βασισμένη στο GDSII BNF και λόγω αυτής της αυστηρής οργάνωσης είναι σχετικά εύκολο να διαβαστεί από τον υπολογιστή (parsing). Ο μέγιστος αριθμός vertixes επίσημα, είναι μόνο μέχρι 200 ζευγάρια X,Y παρόλα αυτά πολλά packages μπορούν να διαβάσουν μέχρι 64k/2=32k διότι αυτό είναι και το μέγιστο μήκος αρχείων που μπορεί να υπάρξει (2 bytes).



Εικόνα 4.1:Απόδοση ενός μικρού GDSII standard cell με τρία μεταλλικά στρώματα

Σε αντίθεση με τον υπολογιστή, ο απλός χρήστης-αναγνώστης, είναι δύσκολο να διαβάσει τα GDSII αρχεια, δεδομένης της δυαδικής μορφής τους, αφού τα περιεχόμενα τους εκφράζονται σε μορφή ASCII. Τέλος, έχει αναπτυχθεί μια μορφή ASCII (KEY format), η οποία είναι κάτι περισσότερο από μια αναπαράσταση κειμένου. Είναι δυνατή η μετατροπή GDSII format σε KEY format και ανάποδα και μπορεί να περιέχει κύκλους, τόξα, πολύγωνα με τμήματα τόξων. Ένα παράδειγμα αναπαράστασης GDSII αρχείου σε KEY format φαίνεται παρακάτω:

```
# KEY file for GDS-II
# File = example.key
# ====================================================================

HEADER 5; # version
BGNLIB;
LASTMOD {98-8-25  15:53:12}; # last modification time
LASTACC {98-8-25  15:53:12}; # last access time
LIBNAME TEMPEGS.DB;
UNITS;
USERUNITS 0.01; PHYSUNITS 1e-08;

BGNSTR; # Begin of structure
CREATION {98-8-25  15:53:12}; # creation time
LASTMOD  {98-8-25  15:53:12}; # last modification time
STRNAME AAP;

BOUNDARY; LAYER 1;  DATATYPE 0;
  XY 5;
  X    -920000.000; Y     452000.000;  X     656500.000; Y     765500.000;
```

```
  X     175000.000; Y    -174000.000;   X    -756000.000; Y    -198000.000;
  X    -920000.000; Y     452000.000;
ENDEL;

ENDSTR AAP;

BGNSTR; # Begin of structure
CREATION {98-8-25  15:53:12}; # creation time
LASTMOD  {98-8-25  15:53:12}; # last modification time
STRNAME LAYOUT;

BOUNDARY; LAYER 0;  DATATYPE 0;
  XY 5;
  X   -2032000.000; Y    1410000.000;   X    1427000.000; Y    1666000.000;
  X     502000.000; Y   -1580500.000;   X     502000.000; Y   -1523500.000;
  X   -2032000.000; Y    1410000.000;
ENDEL;

BOX; LAYER 2;  BOXTYPE 0;
  XY 5;
  X    1526500.000; Y   -1034500.000;   X    2623500.000; Y   -1034500.000;
  X    2623500.000; Y    1105500.000;   X    1526500.000; Y    1105500.000;
  X    1526500.000; Y   -1034500.000;
ENDEL;

SREF;
SNAME AAP;
STRANS 0,0,0;
  XY 1;

  X   -1112500.000; Y   -1267000.000;
ENDEL;

PATH; LAYER 3;  DATATYPE 0;WIDTH 100000;
  XY 4;
  X     891912.000; Y    2322024.000;   X     966537.000; Y    1854278.000;
  X    2599515.000; Y    2311647.000;   X    2626485.000; Y    2005353.000;
ENDEL;

TEXT; LAYER 3;
TEXTTYPE 0; PRESENTATION 0,2,0; PATHTYPE 1; STRANS 0,0,0; MAG 1875;
  XY 1;

  X   -2256500.000; Y    1539500.000;
STRING "Boundary";
ENDEL;
```

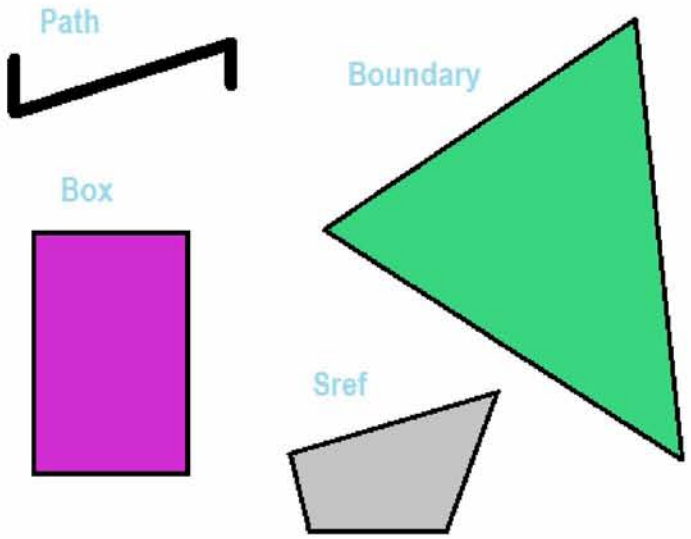
```
TEXT; LAYER 3;
TEXTTYPE 0; PRESENTATION 0,2,0; PATHTYPE 1; STRANS 0,0,0; MAG 1875;
  XY 1;

  X    -151500.000; Y    1924500.000;
STRING "Path";
ENDEL;

TEXT; LAYER 3;
TEXTTYPE 0; PRESENTATION 0,2,0; PATHTYPE 1; STRANS 0,0,0; MAG 1875;
  XY 1;

  X   -1740000.000; Y    -511500.000;
STRING "Sref";
ENDEL;

TEXT; LAYER 3;
TEXTTYPE 0; PRESENTATION 0,2,0; PATHTYPE 1; STRANS 0,0,0; MAG 1875;
  XY 1;

  X    1579000.000; Y    1301500.000;
STRING "Box";
ENDEL;

ENDSTR

LAYOUT;
ENDLIB;
```

Μια περιγραφή ενός GDS II κυκλώματος είναι μια συλλογή από κελιά (cells) που μπορεί να περιέχουν γεωμετρικές ή άλλες αναφορές κελιών. Αυτά τα κελιά, όπου στην GDSII γλώσσα καλούνται δομές (structures) έχουν αλφαριθμητικά ονόματα με μέγεθος μέχρι 32 χαρακτήρες . Μια βιβλιοθήκη από δομές (structures), περιλαμβάνεται σε ένα αρχείο που αποτελείται από μια επικεφαλίδα βιβλιοθήκες (library header), μια ακολουθία από δομές (structures) και μια ουρά βιβλοθήκης (library tail). Με τη σειρά της κάθε δομή (structure) στην ακολουθία αποτελείται από μια επικεφαλίδα δομής (structure header), μια ακολουθία στοιχείων, και μια ουρά δομής (structure tail).

Υπάρχουν επτά είδη στοιχείων:

- ***Boundary*** ,το οποίο καθορίζει ένα γεμισμένο πολύγωνο
- ***Path***, το οποίο καθορίζει ένα καλώδιο
- ***Structure reference***, η οποία επικαλείται ένα υπο-κελί (subcell)

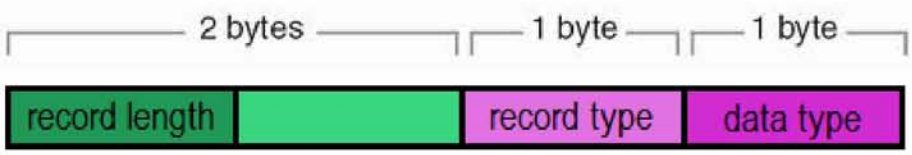
- ***Array reference***, η οποία επικαλείται έναν πίνακα από υπο-κελιά (subcells)
- ***Text***, το οποίο περιλαμβάνει την τεκμηρίωση (documentation)
- ***Node***, ο οποίος καθορίζει ένα μονοπάτι ηλεκτρονικού κυκλώματος,
- ***Box***, το οποίο τοποθετεί την ορθογώνια γεωμετρία



Εικόνα 4.2: Αναπαράσταση ακολουθίας στοιχείων σε ένα structure

4.2 Μορφή αρχείου

Για να γίνει κατανοητή η ακριβής μορφή ενός GDSII αρχείου, θα πρέπει πρώτα να περιγράψουμε την γενική μορφή του. Κάθε αρχείο GDSII αποτελείται από μία επικεφαλίδα (header) μεγέθους 4 Byte η οποία προσδιορίζει το μέγεθος της εγγραφής και της συνάρτησης. Τα πρώτα 2 Byte σχηματίζουν έναν ακέραιο αριθμό των 16 bit που δηλώνει το μήκος της εγγραφής σε bytes και το οποίο με την σειρά του περιλαμβάνει έναν header (επικεφαλίδα) που πρέπει να είναι πάντα ένα ζυγός αριθμός. Το τέλος μιας εγγραφής μπορεί να περιέχει ένα μηδενικό (null) byte όταν το περιεχόμενο της εγγραφής είναι περιττός αριθμός bytes. Το τρίτο byte της επικεφαλίδας περιέχει τον τύπο της εγγραφής και το τέταρτο byte περιέχει τον τύπο των δεδομένων της.



Εικόνα 4.3: Το μήκος εγγραφής χρησιμοποιείται για να βρεθεί ο αριθμός των στοιχείων για κάθε τύπο δεδομένων (data types).

Data Type	Value
No Data	0
Bit Array	1
Two Byte Signed Integer	2
Four Byte Signed Integer	3
Four Byte Real	4 (not used)
Eight Byte Real	5
ASCII string	6

Επειδή ο τύπος δεδομένων είναι σταθερός για κάθε τύπο εγγραφής, αυτό το πεδίο των 2-byte ορίζει και τα πιθανά αρχεία, όπως φαίνεται στα παρακάτω σχήματα.

File Header Records:	Bytes 3 and 4	Parameter Type
HEADER	0002	2-byte integer
BGNLIB	0102	12 2-byte integers
LIBNAME	0206	ASCII string
REFLIBS	1F06	2 45-character ASCII strings
FONTS	2006	4 44-character ASCII strings
ATTRTABLE	2306	44-character ASCII string
GENERATIONS	2202	2-byte integer
FORMAT	3602	2-byte integer
MASK	3706	ASCII string
ENDMASKS	3800	No data
UNITS	0305	2 8-byte floats
File Tail Records:	**Bytes 3 and 4**	**Parameter Type**
ENDLIB	0400	No data
Structure Header Records:	**Bytes 3 and 4**	**Parameter Type**
BGNSTR	0502	12 2-byte integers
STRNAME	0606	Up to 32-characters ASCII string
Structure Tail Records:	**Bytes 3 and 4**	**Parameter Type**
ENDSTR	0700	No data

Εικόνα 4.4: GDSII header records types

Element Header Records:	Bytes 3 and 4	Parameter Type
BOUNDARY	0800	No data
PATH	0900	No data
SREF	0A00	No data
AREF	0B00	No data
TEXT	0C00	No data
NODE	1500	No data
BOX	2D00	No data

Element Contents Records:	Bytes 3 and 4	Parameter Type
ELFLAGS	2601	2-byte integer
PLEX	2F03	4-byte integer
LAYER	0D02	2-byte integers
DATATYPE	0E02	2-byte integer
XY	1003	Up to 200 4-byte integer pairs
PATHTYPE	2102	2-byte integer
WIDTH	0F03	4-byte integer
SNAME	1206	Up to 32-character ASCII string
STRANS	1A01	2-byte integer
MAG	1B05	8-byte float
ANGLE	1C05	8-byte float
COLROW	1302	2 2-byte integers
TEXTTYPE	1602	2-byte integer
PRESENTATION	1701	2-byte integer

Εικόνα 4.5: GDSII elements record type

4.3 Βιβλιοθήκες HEAD και TAIL

Η επικεφαλίδα ενός GDSII αρχείου ξεκινά με την εγγραφή HEADER. Οι παράμετροι της περιλαμβάνουν πληροφορίες για τον αριθμό έκδοσης του αρχείου. Παραδείγματος χάριν, η σειρά των Bytes 0, 6, 0, 2, 0, 1 στην αρχή του αρχείου αποτελούν την επικεφαλίδα μιας εγγραφής ενός αρχείου αριθμού έκδοσης 1. Έπειτα ακολουθεί μια εγγραφή BGNLIB που περιέχει την ημερομηνία της τελευταίας τροποποίησης και την ημερομηνία της τελευταίας πρόσβασης στο αρχείο. Η τρίτη

εγγραφή ενός αρχείου είναι η LIBNAME, το οποίο προσδιορίζει το όνομα αυτού του αρχείου της βιβλιοθήκης. Για παράδειγμα, τα bytes 0, 8, 2, 6, "C", "H", "I", "P" ορίζουν μια βιβλιοθήκη που ονομάζεται “chip”. Μετά από αυτήν την εγγραφή μπορεί να υπάρχουν κάποιες επικεφαλίδες προαιρετικών εγγραφών όπως: REFLIB που περιέχει τα ονόματα των βιβλιοθηκών αναφοράς, FONTS όπου αναφέρονται έως και τέσσερις γραμματοσειρές και FORMAT που υποδεικνύει το είδος του αρχείου.

Η τελευταία εγγραφή είναι η UNITS, η οποία δεν είναι προαιρετική και οι παράμετροι τις οποίας περιέχουν τον αριθμό μονάδων, ανά μονάδα δεδομένων και τον αριθμό των μέτρων, ανά μονάδα βάσης δεδομένων.

Μετά τις εγγραφές του HEADER file ακολουθούν οι εγγραφές των δομών (structures). Αφού έχει οριστεί και η τελευταία δομή (structure) το αρχείο τερματίζει με μια εγγραφή ENDLIB.

4.4 Δομή HEAD και TAIL

Κάθε δομή (structure) περιέχει δύο επικεφαλίδες αρχείου (header records) και μια ουρά αρχείου (tail record). Η πρώτη από τις επικεφαλίδες αρχείου είναι η εγγραφή BGNSTR, η οποία περιέχει την ημερομηνία δημιουργίας και την ημερομηνία τελευταίας τροποποίησης και η δεύτερη είναι η εγγραφή STRNAME, που περιέχει το όνομα της δομής (structure) . Η τελευταία εγγραφή είναι η ENDSTR. Μετά θα πρέπει να ακολουθεί άλλο ένα BGNSTR ή το τέλος της βιβλιοθήκης, ENDLIB.

4.5 Στοιχείο Boundary

Το στοιχείο αυτό ορίζει ένα πολύγωνο και ξεκινά με την εγγραφή BOUNDARY (ενδεχομένως να περιέχει και τις προαιρετικές ELFLAGS και PLEX) και εν συνεχεία απαιτούνται οι εγγραφές LAYER, DATATYPE και XY.

Η εγγραφή LAYER είναι απαραίτητη για να προσδιορίσει ποιο στρώμα (layer) (αριθμημένο από 0 έως 63) χρησιμοποιείται από το boundary. Οι έννοιες των στρωμάτων (layers) δεν έχουν αυστηρούς ορισμούς και για τον λόγο αυτό, θα πρέπει να προσδιορίζονται για κάθε σχεδιαστικό περιβάλλον και βιβλιοθήκη.

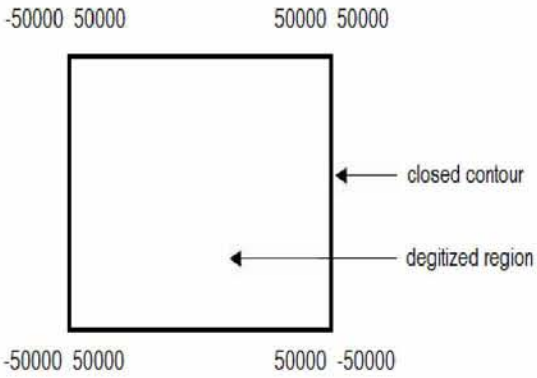
Η εγγραφή DATATYPE περιέχει πληροφορίες ήσσονος σημασίας γι’ αυτό και η τιμή της πρέπει να είναι μηδενική.

Η εγγραφή XY περιέχει από 4 έως 200 ζεύγη συντεταγμένων τα οποία καθορίζουν το περίγραμμα του πολυγώνου άρα το πρώτο ζεύγος συντεταγμένων πρέπει να

ταυτίζεται με το τελευταίο. Ο αριθμός των σημείων σε αυτό το αρχείο καθορίζεται από το μήκος της εγγραφής.

Ένα παράδειγμα φαίνεται παρακάτω:

```
BOUNDARY 5 0        <-- Boundary on layer 5, datatype 0
-50000  -50000      <-- First X,Y vertex
 50000  -50000      <-- Second X,Y vertex
 50000   50000      <-- Third X,Y vertex
-50000   50000      <-- Fourth X,Y vertex
-50000  -50000      <-- Back to First X,Y vertex
ENDEL
```



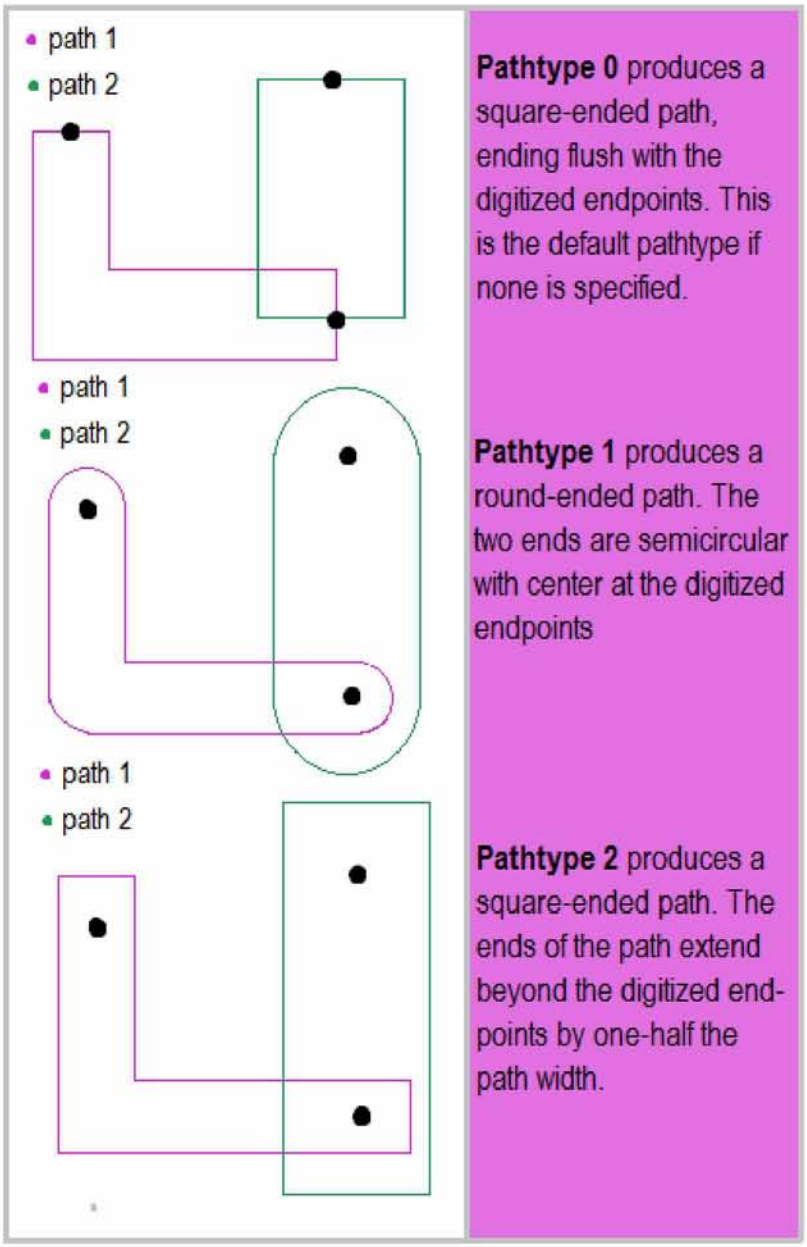
4.6 Στοιχείο Path

Ένα path είναι μία τεθλασμένη γραμμή, με μη μηδενικό πλάτος, που συνήθως χρησιμοποιείται για να τοποθετηθούν τα καλώδια. Το στοιχείο αυτό αρχικοποιείται με μία εγγραφή PATH και ακολουθείται από τις προαιρετικές εγγραφές ELFLAGS και PLEX. Έπεται η εγγραφή LAYER που είναι απαραίτητη για τον προσδιορισμό του υλικού του path, η εγγραφή DATATYPE και η XY που προσδιορίζει τις συντεταγμένες του (από 2 έως 200).

Πριν από την XY μπορεί να υπάρχουν δύο προαιρετικές εγγραφές που ονομάζονται PATHTYPE και WIDTH. Η PATHTYPE περιγράφει το είδος των άκρων του path, σύμφωνα με την τιμή της παραμέτρου του. Η τιμή αυτή είναι:

- Μηδέν, εάν έχουν τετραγωνικές άκρες και τερματίζουν στις κορυφές του μονοπατιού
- Ένα, εάν έχουν στρογγυλεμένες άκρες
- Δύο, εάν έχουν στρογγυλεμένες άκρες που επικαλύπτουν τις κορυφές τους κατά το ήμισυ του πλάτους του

Παρακάτω παρουσιάζονται σχηματικά:



Εικόνα 4.6: Το είδος των άκρων του PATHTYPE, σύμφωνα με την τιμή της παραμέτρου του.

4.7 Στοιχείο Structure Reference

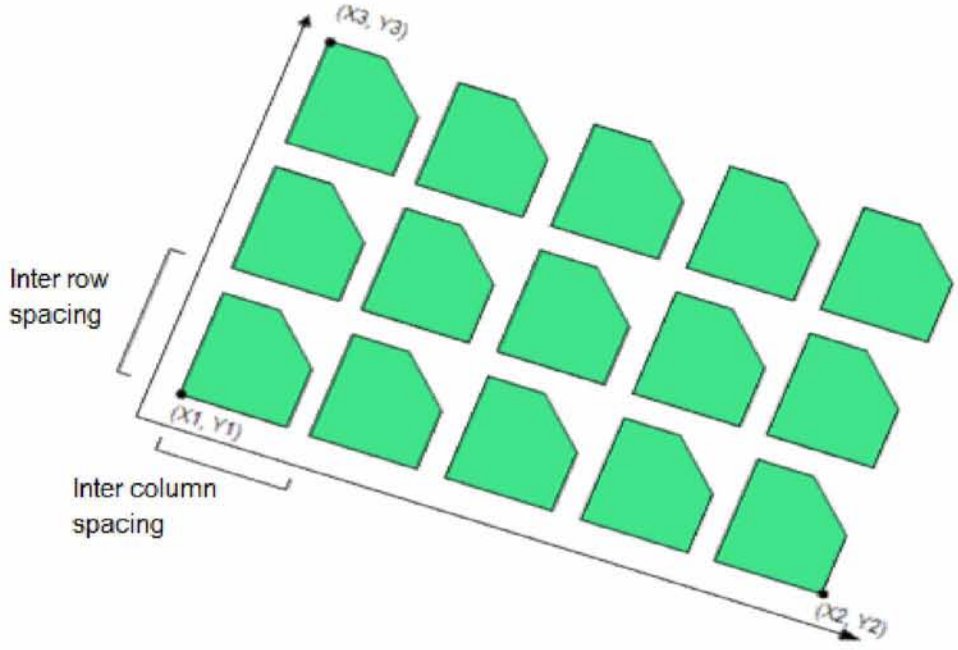
Η ιεραρχία επιτυγχάνεται επιτρέποντας αναφορές σε structures (στιγμιότυπα) να εμφανίζονται σε άλλα structures. Η εγγραφή SREF υποδηλώνει μια αναφορά σε structure και ακολουθείται από τη προαιρετική ELFLAGS και PLEX. Η εγγραφή SNAME κατονομάζει στη συνέχεια την επιθυμητή δομή και μια εγγραφή XY περιέχει μία ξεχωριστή συντεταγμένη όπου θα τοποθετηθεί το στιγμιότυπο. Είναι επίσης επιτρεπτό να γίνεται αναφορά σε δομές που δεν έχουν ακόμη καθορισθεί με STRNAME.

4.8 Στοιχείο Array Reference of Structures Element

Για λόγους ευκολίας, ένας πίνακας από structure στιγμιότυπα μπορεί να καθοριστεί με την εγγραφή AREF. Μετά την προαιρετική ELFLAGS και PLEX εγγραφή ακολουθεί η SNAME για τον προσδιορισμό του πίνακα. Στη συνέχεια, η προαιρετική μετατροπή των εγγραφών STRANS, MAG, και ANGLE μπορεί να δώσει τον προσανατολισμό των στιγμιότυπων.

Ακολουθεί η εγγραφή COLROW η οποία καθορίζει τον αριθμό των στηλών και τον αριθμό των γραμμών του πίνακα. Η τελική εγγραφή είναι η XY με τρία σημεία: μία συντεταγμένη για την γωνία, μία συντεταγμένη για το τελευταίο στιγμιότυπο στην κατεύθυνση στήλης, και μία συντεταγμένη για το τελευταίο στιγμιότυπο προς την κατεύθυνση σειράς. Από τις πληροφορίες αυτές, μπορεί να καθοριστεί το ποσό της επικάλυψης ή του διαχωρισμού των στιγμιότυπων.

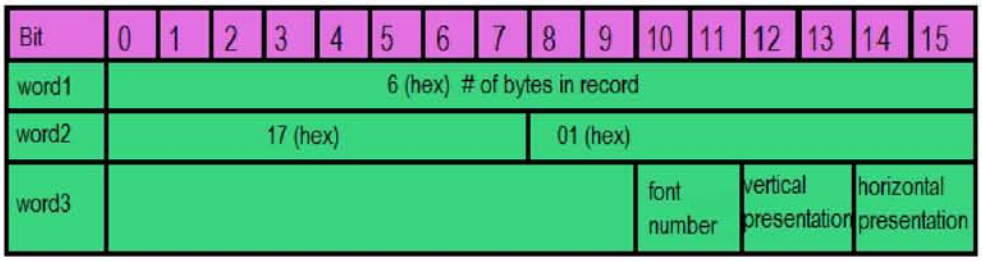
Ένα παράδειγμα φαίνεται παρακάτω:



Εικόνα 4.7: Σχηματική αναπαράσταση για τον καθορισμό σειρών και στηλών του πίνακα μέσω της εγγραφής COLROW

4.9 Στοιχείο Text Element

Με την εγγραφή TEXT μπορούμε να συμπεριλάβουμε μηνύματα μέσα σε ένα κύκλωμα. Μετά τις προαιρετικές ELFLAGS, PLEX και την υποχρεωτική LAYER ακολουθεί η TEXTTYPE με μηδενική τιμή. Εν συνεχεία ακολουθεί η προαιρετική εγγραφή PRESENTATION η οποία καθορίζει την γραμματοσειρά σε Bits όπως φαίνεται παρακάτω:



Τέλος, πέραν των προαιρετικών εγγραφών PATHTYPE, WIDTH, STRANS, MAG, και ANGLE που μπορεί να εμφανιστούν μετά το TEXT απαιτούνται άλλες δύο εγγραφές : η XY με μία συντεταγμένη για να τοποθετήσει το κείμενο και η STRING που το ορίζει πλήρως.

4.10 Στοιχείο Node

Τα ηλεκτρονικά δίκτυα μπορούν να καθοριστούν από μία εγγραφή NODE**.** Μετά τις προαιρετικές ELFLAGS, PLEX και την υποχρεωτική LAYER ακολουθεί η NODETYPE με μηδενική τιμή. Στην συνέχεια ακολουθεί η XY με ένα έως πενήντα σημεία που προσδιορίζουν τις συντεταγμένες για το ηλεκτρονικό δίκτυο. Οι πληροφορίες σε αυτό το στοιχείο δεν είναι σε γραφικό περιβάλλον και δεν επηρεάζει το κατασκευασμένο κύκλωμα.

4.11 Στοιχείο Box

Το τελευταίο στοιχείο ενός αρχείου GDSII είναι το box. Μετά τις προαιρετικές ELFLAGS, PLEX και την υποχρεωτική LAYER ακολουθεί η BOXTYPE με μηδενική τιμή και η XY. Η XY πρέπει να περιέχει πέντε σημεία που περιγράφουν ένα κλειστό, τετράπλευρο κουτί. Σε αντίθεση με το όριο, αυτό δεν αποτελεί ένα γεμάτο σχήμα.

5 Τεχνικές και Μεθοδολογία Σχεδίασης

Σκοπός του παρόντος κεφαλαίου είναι να περιγράψει την τεχνική που ακολουθήσαμε για την επίτευξη χαμηλής κατανάλωσης ισχύος σε standard cell. Οι μέθοδοι για την ελαχιστοποίηση της ενέργειας μπορεί να ποικίλουν και να πραγματοποιούνται με διάφορες τεχνικές όπως μέσω της μείωσης της τάσης τροφοδοσίας, της χωρητικότητας μεταγωγής και της συχνότητας λειτουργίας του ρολογιού. Στην παρούσα υλοποίηση η ελαχιστοποίηση ενέργειας επιτυγχάνεται με μείωση του πλάτους των τρανζίστορ. Η ανάμειξη και η τροποποίηση της σχεδίασης των τρανζίστορ απαιτεί έλεγχο κανόνων σχεδίαση για την ορθή λειτουργία του κυκλώματος. Παρακάτω θα γίνει μια σύντομη αναφορά στην τεχνολογία κατασκευής των τρανζίστορ και στους κανόνες σχεδίασης DRC (Design Rule Check) που εφαρμόστηκαν και στην περιγραφή της μεθοδολογίας για επίτευξη χαμηλής κατανάλωσης.

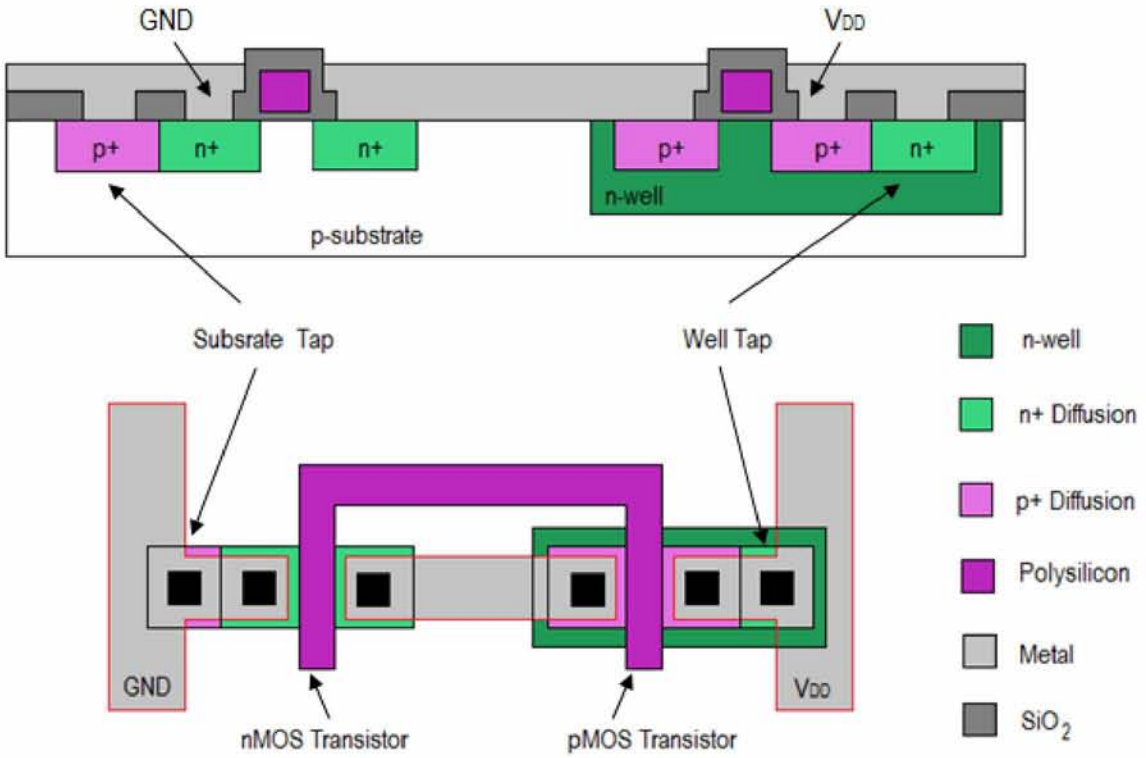
5.1 Τεχνολογία κατασκευής τρανζίστορ MOS

Για την ορθή λειτουργία του κυκλώματος πέραν των γεωμετρικών κανόνων που πρέπει να ακολουθήσουμε, πρέπει η αλληλοσυσχέτιση των μασκών μέσω των βημάτων επεξεργασίας της να παράγει σωστά κυκλώματα.

Όπως αναλύσαμε και στον δεύτερο κεφάλαιο τα τρανζίστορ κατασκευάζονται από πυρίτιο (ένας όγκος πυριτίου σε μορφή καρότου) το οποία τεμαχίζεται σε πολύ λεπτές φέτες τα wafers. Πάνω στην επιφάνεια του πυριτίου τοποθετούμε πολλαπλά επίπεδα αγώγιμων και μονωτικών υλικών. Είναι μια διαδικασία η οποία γίνεται σε βήματα που βασίζονται σε μια σειρά χημικών διεργασιών όπως οξείδωση πυριτίου, διάχυση προσμίξεων, απόθεση και χάραξη αλουμινίου. Παρακάτω φαίνεται η κατασκευή ενός CMOS αντιστροφέα. Η ακολουθία των βημάτων κατασκευής του είναι η εξής:

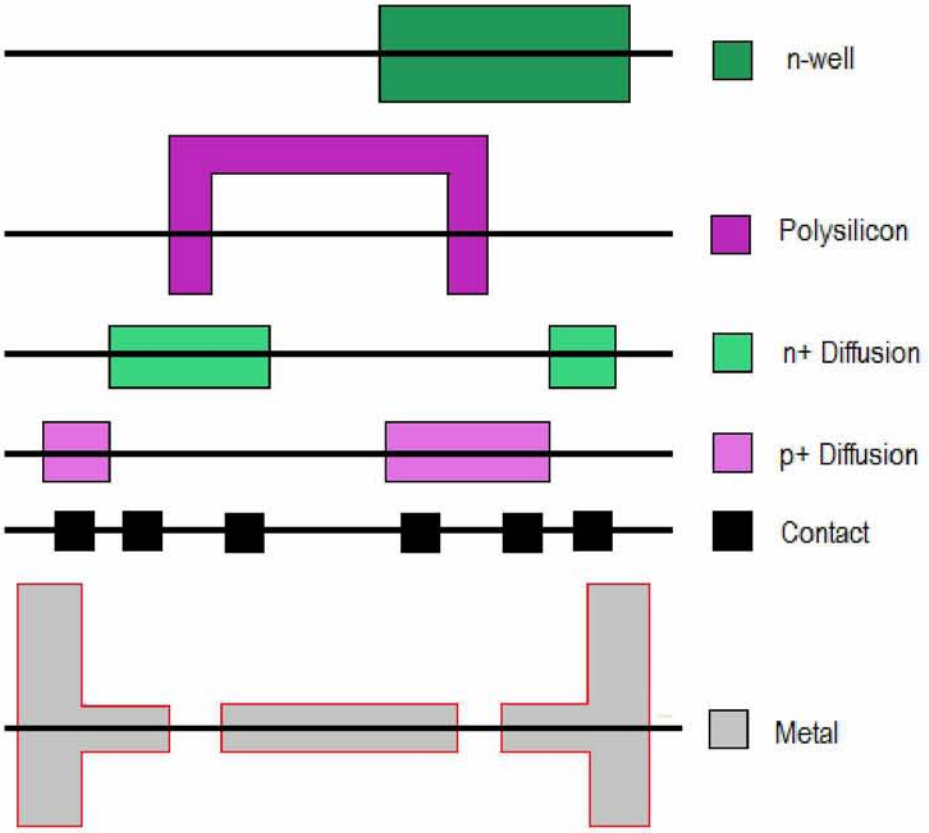
- Κατασκευή n-well
- Κατασκευή αγωγών πολυπυριτίου
- Κατασκευή των περιοχών n-diffusion
- Κατασκευή των περιοχών p-diffusion

- Κατασκευή αγωγών μετάλλου



Εικόνα: Κατασκευή ενός CMOS αντιστροφέα

Η διαδικασία κατασκευής είναι άμεσα συνδεδεμένη με την κατασκευή ειδικών μασκών. Οι μάσκες πρέπει να τηρούν κι αυτές τους κανόνες κατασκευής (DRC) κι όλες οι απαιτούμενες λεπτομέρειες κατασκευής είναι καταγεγραμμένες στις βιβλιοθήκες των κυττάρων. Παρακάτω, φαίνονται στις εικόνες τα βήματα σχεδιασμού μασκών για την κατασκευή. Παρακάτω φαίνεται το φυσικό σχέδιο ενός αντιστροφέα CMOS.



Εικόνα 5.1: Το εργαλείο σχεδίασης χωρίζει τον σχεδιασμό σε μάσκες για την κατασκευή.

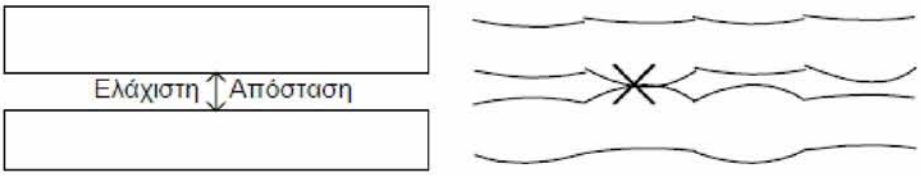
Συνεπώς η οποιαδήποτε αλλαγή που θα αφορά των πλάτος των τρανζίστορ θα πρέπει λοιπόν να είναι σύμφωνη με το DRC του κατασκευαστή. Εφόσον δεν επηρεάσουμε την κατασκευή των μασκών και λειτουργήσουμε βάσει των σχεδιαστικών κανόνων, η κατασκευή του Layout και η λειτουργία του κυκλώματος θα είναι ορθές.

5.2 Σχεδιαστικοί κανόνες

Οι κανόνες φυσικής σχεδίασης, αναφέρονται ακόμη και ως κανόνες σχεδίασης μπορούν να θεωρηθούν ως μια συνταγή για την κατασκευή φωτομασκών που χρησιμοποιούνται στην κατασκευή ολοκληρωμένων κυκλωμάτων. Ο κυριότερος σκοπός των κανόνων σχεδίασης είναι η λήψη ενός κυκλώματος με βέλτιστη απόδοση υλικού σε μια όσο το δυνατόν μικρότερη επιφάνεια, χωρίς όμως να γίνεται κανένας συμβιβασμός στην λειτουργία του κυκλώματος. Όσο πιο «συντηρητικοί» είναι οι κανόνες σχεδίασης τόσο πιο πιθανό είναι το κύκλωμα να λειτουργήσει σωστά ενώ όσο πιο «επιθετικοί» είναι οι κανόνες αυξάνουν την πιθανότητα βελτίωσης της απόδοστiης του κυκλώματος. Οι κανόνες σχεδίασης προσδιορίζουν ορισμένους γεωμετρικούς περιορισμούς κατά την σχεδίαση έτσι ώστε τα σχήματα κατά την

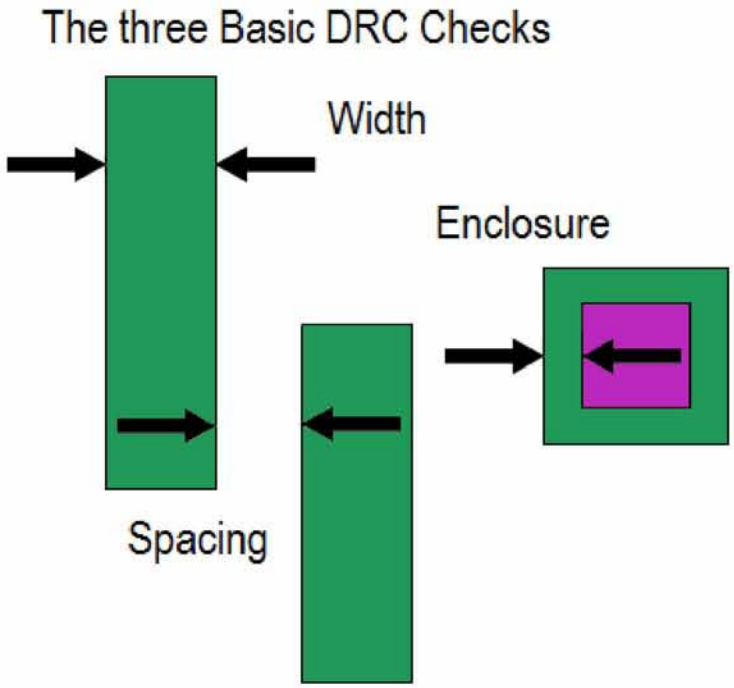
επεξεργασία να διατηρήσουν την τοπολογία και την γεωμετρία των σχεδιάσεων. Είναι σημαντικό να σημειωθεί ότι οι κανόνες σχεδίασης δεν αναπαριστούν κάποια σαφή όρια μεταξύ σωστής και λανθασμένης κατασκευής αλλά μια ανοχή που εξασφαλίζει σαφή όρια μεταξύ σωστής κατασκευής και αξιόπιστης λειτουργίας.

Σε μια τεχνολογία υπάρχουν δύο σύνολα περιορισμών κανόνων σχεδίασης που περιγράφονται από τα πάχη των γραμμών και την τοποθέτηση των επιπέδων. Αν τα πάχη γίνουν πολύ μικρά υπάρχει κίνδυνος ασυνέχειας, ενώ στην αντίθετη περίπτωση υπάρχει κίνδυνος βραχυκυκλώματος.



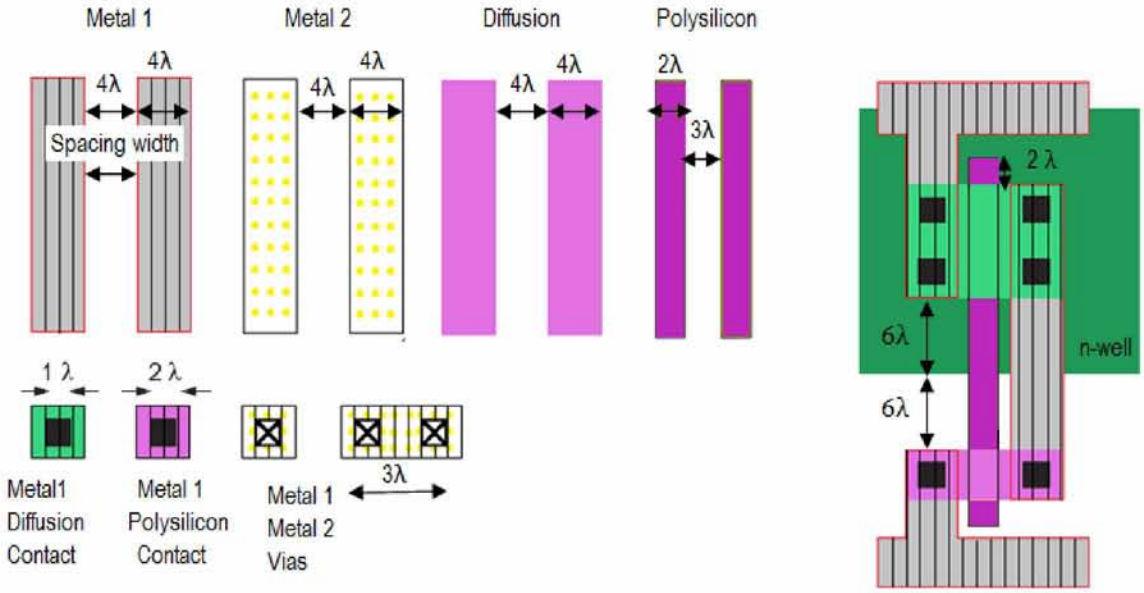
Οι κανόνες σχεδίασης συνήθως αφορούν δύο ζητήματα: (1) την γεωμετρική αναπαραγωγή των σχημάτων που μπορούν να αναπαραχθούν από την διαδικασία αναπαραγωγής των μασκών και της λιθογραφίας και (2) την αλληλεπίδραση μεταξύ των δύο διαφορετικών στρώσεων. Επίσης χρησιμοποιούνται αρκετές προσεγγίσεις. Αυτές περιλαμβάνουν τους κανόνες του «μικρόμετρου» και σε κανόνες βασισμένους στην παράμετρο λάμδα (λ). Το λ ορίζεται ως το ½ του ελάχιστου μήκους καναλιού του τρανζίστορ (π.χ. για ελάχιστο μήκος 180nm, λ=0.09nm) και χρησιμοποιείται έτσι ώστε όλες οι γεωμετρικές ανοχές σε έναν σχεδιασμό να μπορούν να ορίζονται ως ακέραια πολλαπλάσια του λ. Οι κανόνες σχεδίασης μικρόμετρου συνήθως δίνονται ως μία λίστα από ελάχιστες χαρακτηριστικές τιμές μεγεθών και αποστάσεων για όλες τις μάσκες που απαιτούνται σε μια δεδομένη τεχνολογία.

Οι πιο βασικοί κανόνες είναι αυτή του μεγέθους (τυπική τιμή 100nm από το 2007), της απόστασης (τυπική τιμή 400nm από το 2007) και της επικάλυψης (τυπική τιμή 10nm από το 2007) που φαίνονται παρακάτω.



Εικόνα 5.2: Οι τρεις βασικότεροι σχεδιαστικοί κανόνες: μέγεθος, απόσταση, επικάλυψη

Η βιομηχανία συνήθως χρησιμοποιεί τους πραγματικούς κανόνες σχεδίασης με βάση το μικρότερο και κωδικοποιεί τις σχεδιάσεις σαν συνάρτηση αυτών των διαστάσεων ή χρησιμοποιεί συστήματα συμβολικής φυσικής σχεδίασης που να στοχεύουν ακριβώς στους κανόνες σχεδίασης. Ένα DRC λογισμικό λαμβάνει σαν είσοδο μία αναπαράσταση σε GDSII μορφή και μία λίστα κανόνων που έχουν επιλεγεί για την κατασκευή και παράγει μια σειρά από παραβιάσεις κανόνων που ο ίδιος ο σχεδιαστής θα επιλέξει κατά πόσο θα διορθώσει ή όχι.



Εικόνα 5.3: Κανόνες Σχεδίασης (Design Rules)

Δημιουργία κανόνων σχεδίασης DRC

Οι κανόνες σχεδιασμού που εφαρμόστηκαν για την υλοποίηση του εργαλείου που περιγράφεται στην παρούσα μεταπτυχιακή παρουσιάζονται παρακάτω. Οι κανόνες σχεδίασης προσδιορίζουν ορισμένους γεωμετρικούς περιορισμούς που πρέπει να εφαρμοστούν κατά την διαδικασία σχεδίασης και επιπλέον εκφράζουν και μια ανοχή για την εξασφάλιση σαφών ορίων μεταξύ ορθής κατασκευής και λειτουργίας.

Λόγω του ότι στην παρούσα εργασία η επέμβαση στην γεωμετρία αφορά ενδεχομένη μείωση πλάτους των στοιχείων τρανζίστορ, όπως θα αναλυθεί εκτενέστερα και παρακάτω, οι κανόνες σχεδίασης που πρέπει να ελεγχτούν είναι συγκεκριμένοι. Γενικότερα οι λίστες με τους κανόνες σχεδίασης είναι μια διαδικασία αρκετά περίπλοκη για να πραγματοποιηθεί από έναν χρήστη. Είθισται ο ίδιος ο σχεδιαστής ή η βιομηχανία να παραθέτει τους κανόνες ή να τεθούν από συγκεκριμένο λογισμικό. Η απόπειρα για έλεγχο σχεδιαστικής εγκυρότητας στην παρούσα εργασία έγινε ελέγχοντας κατά πόσο τα layers που εμπλέκονται κατά την αλλαγή μεγέθους τηρούν μία λίστα από ελάχιστες χαρακτηριστικές τιμές μεγεθών και αποστάσεων η οποία φαίνεται παρακάτω

Συγκεκριμένα η ελάχιστη επικάλυψη μεταξύ layer 1 και layer 10 πρέπει να είναι περί τα 44nm, η ελάχιστη απόσταση μεταξύ layer 9 και layer 10 περί τα 354nm, το ελάχιστο πλάτος και μήκος για το layer 10 στα 650nm.

DRC	Layers	Minimum Value
Enclosure	*layer 1, layer 10*	*44nm*
Spacing	*layer 9, layer 10*	*354nm*
Width	*layer 10*	*650nm*
Height	*layer 10*	*650nm*

5.3 Σχεδίαση χαμηλής κατανάλωσης

Μετά την σύντομη αναφορά στην τεχνολογία κατασκευής των τρανζίστορ και τους σχεδιαστικούς περιορισμούς που απαιτούνται για την ορθή λειτουργία και την βέλτιστη απόδοση τους, θα αναφερθούμε στην τεχνική που ακολουθήθηκε στην παρούσα διατριβή για σχεδίαση χαμηλής κατανάλωσης. Η διαδικασία αυτή παρεμβαίνει κατά κάποιο τρόπο στην γεωμετρική κατασκευή των τρανζίστορ. Αυτό έχει ως αποτέλεσμα την αναθεώρηση των ήδη υπαρχόντων σχεδιαστικών κανόνων

και την δημιουργία λίστας γεωμετρικών περιορισμών που θα εξασφαλίσουν ανοχή και σαφή όρια μεταξύ σωστής κατασκευής και αξιόπιστης λειτουργίας.

Το εργαλείο που δημιουργήθηκε στην παρούσα εργασία λαμβάνει σαν είσοδο την αναπαράσταση μιας πύλης σε GDSII μορφή των οποίων η κωδικοποίηση, αποκωδικοποίηση, και αναζήτηση των επιπέδων από τα αρχεία αυτά είναι μια διαδικασία που αναλύεται στο επόμενο κεφάλαιο.

5.3.1 Μείωση πλάτους τρανζίστορ για ελαχιστοποίηση κατανάλωσης ισχύος

Οι μέθοδοι για την ελαχιστοποίηση της ενέργειας μπορεί να ποικίλουν και να πραγματοποιούνται με διάφορες τεχνικές. Μείωση της τάσης τροφοδοσίας, της χωρητικότητας μεταγωγής και της συχνότητας λειτουργίας του ρολογιού είναι ορισμένες από αυτές. Στην παρούσα υλοποίηση η ελαχιστοποίηση ενέργειας θα επιτευχθεί με επέμβαση στο ρεύμα διαρροής και κατ' επέκταση με μείωση του πλάτους των τρανζίστορ.

Όπως αναλύθηκε και σε προηγούμενο κεφάλαιο όπου έγινε αναφορά για την κατανάλωση ισχύος ο τύπος της στατικής ισχύος είναι το γινόμενο του ρεύματος διαρροής και της τάσης τροφοδοσίας : $P = I * V$

Εφόσον η ισχύς εξαρτάται από την τάση τροφοδοσίας και το ρεύμα διαρροής, αυτοί είναι και οι παράγοντες οι οποίοι θα πρέπει να μειωθούν για να προκύψει ελαχιστοποίηση της ενέργειας. Για την πραγματοποίηση του εργαλείο που υλοποιείται στην παρούσα διατριβή επιλέγεται να γίνει αλλαγή στο ρεύμα διαρροής και όχι στην τάση τροφοδοσίας. Η τάση τροφοδοσίας έχει τα χαρακτηριστικά παραμέτρου της σχεδίασης και οι σύγχρονες σχεδιάσεις χαμηλής κατανάλωσης ισχύος έχουν τάση τροφοδοσίας μεταξύ 1.5 και 3 V. Άρα δεν μπορεί να γίνει επέμβαση στην τάση.

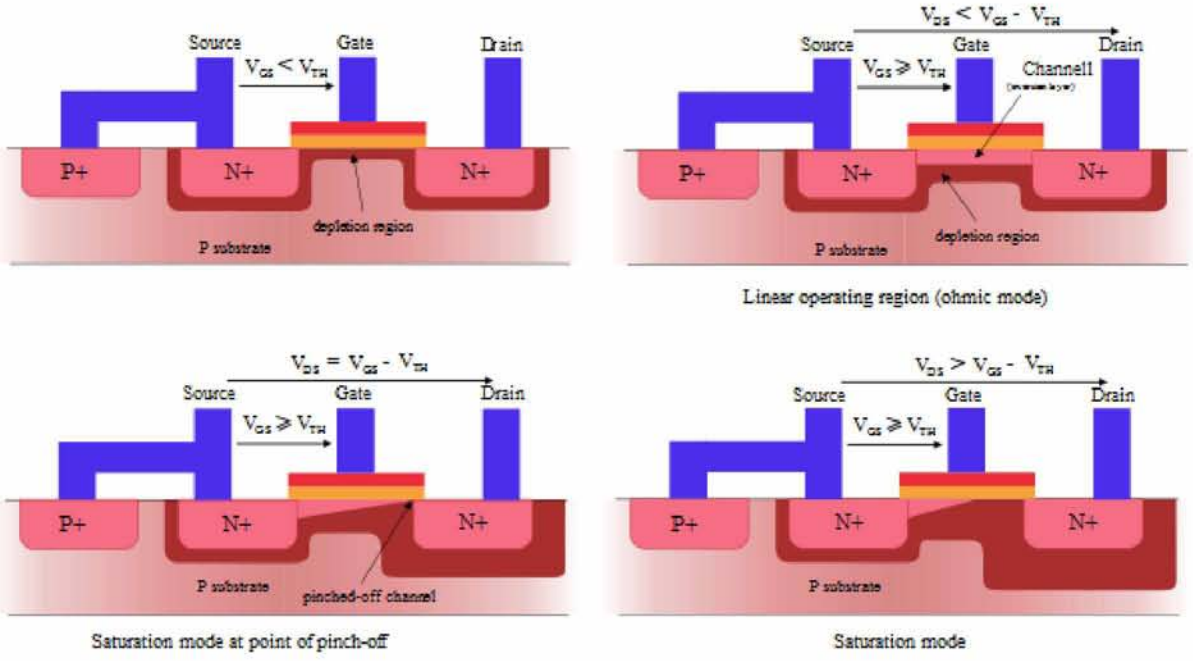
Η ελαχιστοποίηση λοιπόν της κατανάλωσης ισχύος άπτεται στην μείωση του ρεύματος διαρροής. Πρώτος τρόπος επίτευξης αυτού είναι η χρησιμοποίηση συμπληρωματικών πυλών, καθώς επίσης, ως γνωστόν, η διαρροή ρεύματος σε επαφές p-n είναι ανάλογες με την επιφάνεια.

Οι τύποι ρευμάτων για τα n-mos και p-mos αντίστοιχα είναι:

$$\text{Περιοχές λειτουργίας}\begin{cases}\text{Αποκοπής} & I_{ds}=0,\quad V_{gs}\le V_t\\ \text{Γραμμική} & I_{ds}=\beta\left[(V_{gs}-V_t)V_{ds}-\frac{V_{ds}^2}{2}\right],\quad 0<V_{ds}<V_{gs}-V_t\\ \text{Κόρου} & I_{ds}=\beta\frac{(V_{gs}-V_t)^2}{2},\quad 0<V_{gs}-V_t<V_{ds}\end{cases}$$

$$\beta=\frac{\mu\varepsilon}{t_{ox}}\left(\frac{W}{L}\right)$$

Με β συντελεστή κέρδους, μ κινητικότητα φορέων, ε επιδεκτικότητα μονωτή πύλης, t_{ox} πάχος μονωτή πύλης, πλάτος/μήκος (W/L) λόγος διαστάσεων, τα τρανζίστορ εργάζονται στην κατάσταση κόρου.



Εικόνα 5.4: Περιοχές λειτουργίας transistor MOS

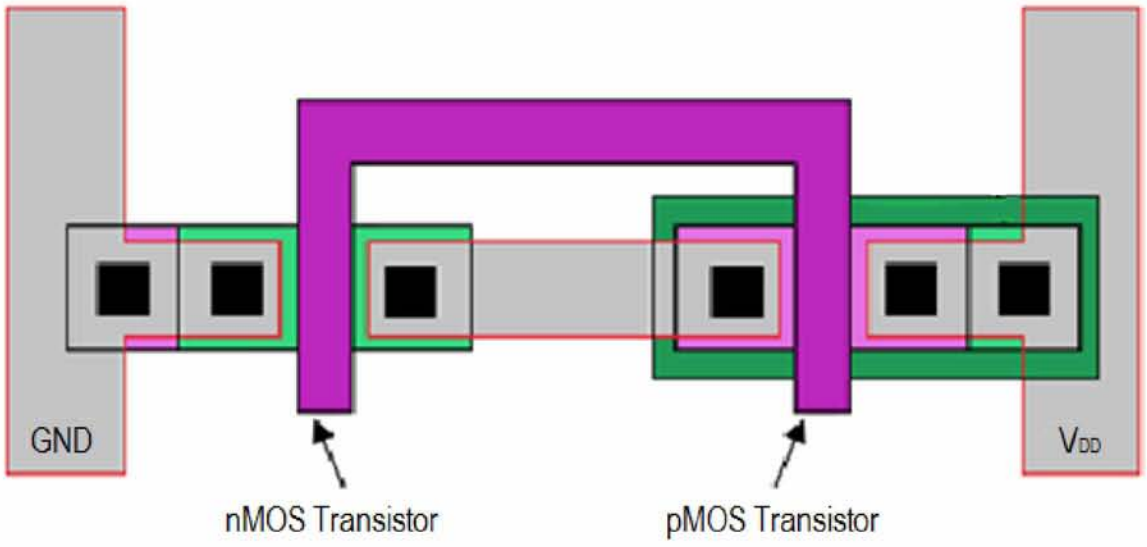
Εφόσον οι τάσεις δεν μπορεί, όπως είπαμε, να επηρεαστούν το ενδιαφέρον στρέφεται στους παράγοντες πλάτους W και μήκους L από τα οποία εξαρτάται, όπως ειπώθηκε και παραπάνω, το ρεύμα διαρροής.

Παράγοντας L

Τα μήκη L κατασκευάζονται συνήθως στην ελάχιστη δυνατή διάσταση που τους επιτρέπεται από την υπάρχουσα τεχνολογία, και είναι συγκεκριμένη στα standard cell εφόσον έχουν κανονικότητα στην γεωμετρία και το φυσικό τους ύψος είναι σταθερό. Τα μεγέθη L είναι σταθερά σε κάθε τεχνολογία CMOS. Το μέγεθος θα έπρεπε να διπλασιαστεί για να υποδιπλασιαστεί η κατανάλωση. Συνεπώς, οποιαδήποτε αλλαγή στο μέγεθός τους θα προκαλούσε και θέματα placement. Επίσης αν πραγματοποιούνταν αλλαγή στο μήκος L του τρανζίστορ θα είχε ως συνέπεια και επιρροή της συχνότητας προς το χειρότερο. Άρα γι' αυτό το λόγο η οποιαδήποτε απόπειρα αλλαγής στο μήκος L του τρανζίστορ απορρίπτεται.

Παράγοντας W

Ο μοναδικός ελεύθερος παράγοντας που απομένει για να για να επηρεάσει ο σχεδιαστής είναι ο παράγοντας πλάτους W. Εφόσον το πλάτος W είναι ανάλογο με το ρεύμα διαρροής, θα πρέπει να μειώσουμε το μέγεθος του για να προκαλέσουμε κατ' επέκταση ελαχιστοποίηση της ισχύς. Η μείωση του πλάτους των τρανζίστορ όμως οφείλει να
γίνει με προσοχή και συγκεκριμένο τρόπο γιατί υπάρχουν όρια και σχεδιαστικοί κανόνες που πρέπει να ακολουθηθούν και να εφαρμοστούν προκειμένου να επιτευχθεί ορθή λειτουργία του κυκλώματος . Παρακάτω φαίνεται το φυσικό σχέδιο ενός αντιστροφέα CMOS.



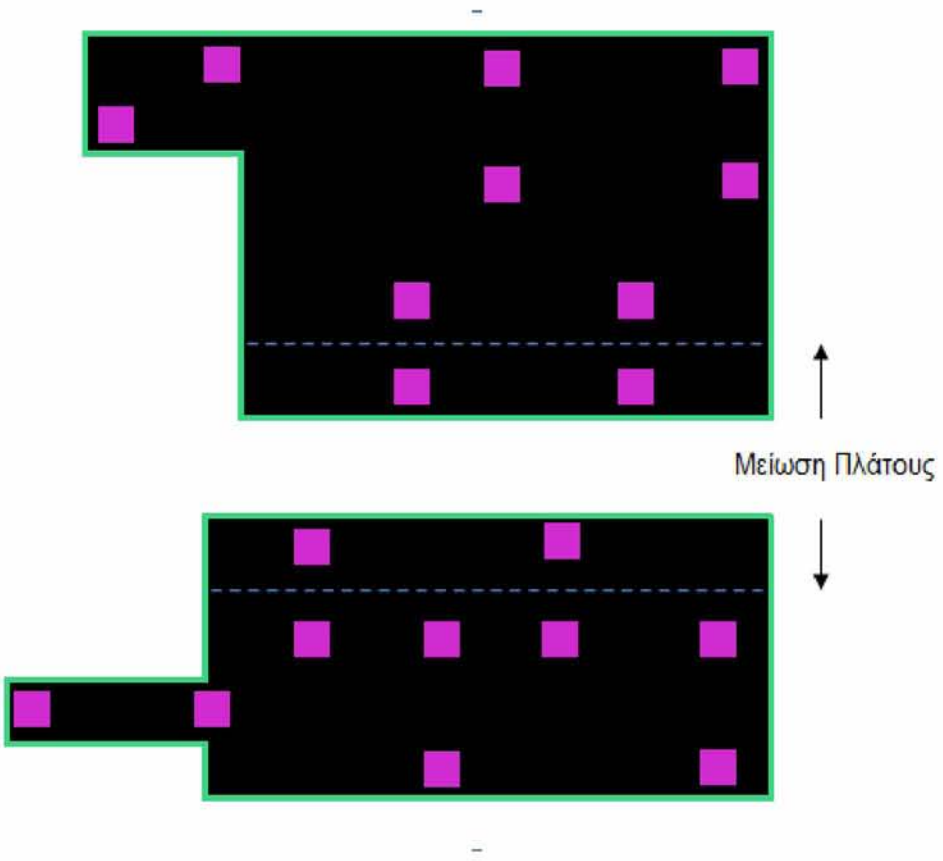
Εικόνα 5.5: Layout ενός αντιστροφέα CMOS

Το πλάτος των τρανζίστορ πρέπει να μειωθεί από μέσα προς τα έξω κι όχι ανάποδα, γιατί στην περίπτωση αυτή υπάρχει κίνδυνος οι τάσεις τροφοδοσίας και τα μέταλλα

να βγουν εκτός επιφάνειας. Κάτι τέτοιο θα προκαλούσε προβλήματα ορθής λειτουργίας του κυκλώματος.

Στον πηγάδι τύπου n μειώνουμε το πάνω μέρος από πάνω προς τα κάτω ενώ το πηγάδι τύπου p το κάτω μέρος από κάτω προς τα πάνω. Όπως αναφέρθηκε και στο κεφάλαιο 4, η περιγραφή κάθε πολυγώνου γίνεται αριστερόστροφα ξεκινώντας από το κάτω αριστερά σημείο (συντεταγμένες). Αναλόγως εάν η μείωση γίνει σε τρανζίστορ τύπου n-mos ή p-mos πρέπει να ελέγχουμε τα ευθύγραμμα τμήματα που σχηματίζουν το εκάστοτε πολύγωνο ορίζοντας αντίστοιχα τα κατώτερα και ανώτερα οριζόντια τμήματα προς μείωση με μια διαδικασία η οποία θα περιγραφεί αναλυτικά στο επόμενο κεφάλαιο.

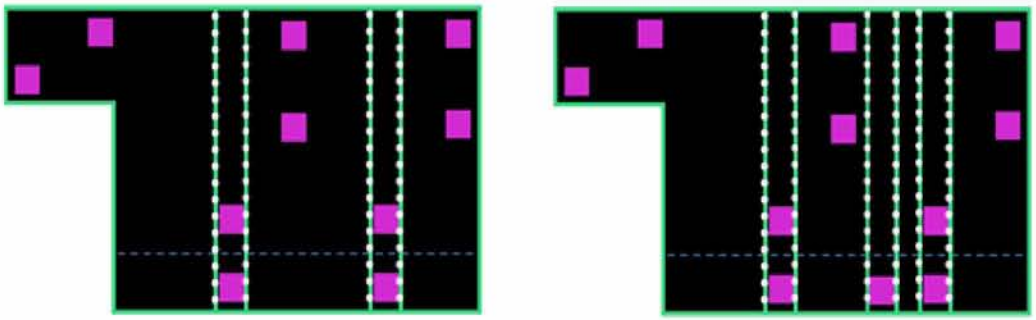
Συγκεκριμένα:



Εικόνα 5.6**:** Μείωση από πάνω προς τα κάτω στο τύπου n (πάνω σχήμα) και από κάτω προς τα πάνω στο τύπου p (κάτω σχήμα).

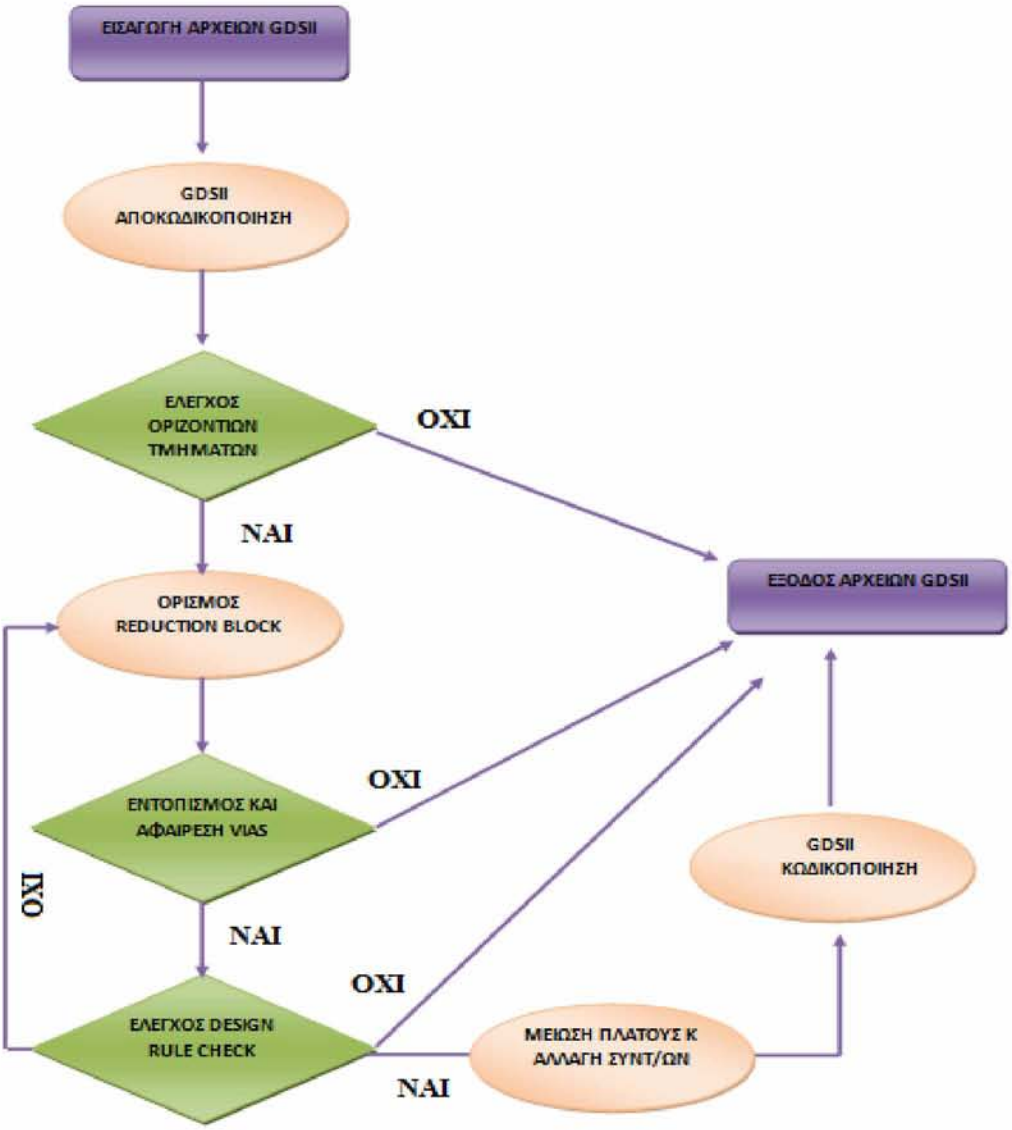
Με την επέμβαση και μείωση του πλάτους των τρανζίστορ πρέπει να λάβουμε υπ' όψιν μας και να γίνει έλεγχος για την ύπαρξη περασμάτων vias που ενδεχομένως λαμβάνουν χώρα στο αποκομμένο τμήμα. Αν εντοπιστούν θα πρέπει να γίνει αφαίρεση τους με την προϋπόθεση ότι στον χώρο του πολυγώνου υπάρχει vias με τις ίδιες τετμημένες. Συνεπώς θα πρέπει να υπάρχει μέριμνα για τουλάχιστον ένα vias για

κάθε είσοδο , έξοδο και τάση τροφοδοσίας για την εξασφάλιση λειτουργίας και τροφοδοσίας του κυκλώματος. Αν η προϋπόθεση αυτή δεν πληρείται η μείωση της διάστασης ακυρώνεται και το τρανζίστορ επανέρχεται στις αρχικές του διαστάσεις. Ένα ακόμη σημείο που λαμβάνεται υπ' όψιν είναι η μείωση των αντίστοιχων πηγαδιών και των επιπέδων (layers) που εμπλέκονται και επηρεάζονται από την αλλαγή του πλάτους καθώς και των μετάλλων.



Εικόνα 5.7: Έλεγχος ύπαρξης vias στο καθορισμένο προς μείωση μπλοκ ενός τρανζίστορ

Αλλάζοντας λοιπόν το τρανζίστορ με φορά από «μέσα προς τα έξω», θα πρέπει στην επόμενη φάση να καταφέρουμε να το προσαρμόσουμε σύμφωνα με προκαθορισμένους γεωμετρικούς κανόνες σχεδίασης που υπαγορεύει το DRC (Design Rule Check) πρόγραμμα, προκειμένου το κύκλωμα που κατασκευάζεται να λειτουργεί σωστά. Η διαδικασία του ελέγχου πάνω στους κανόνες σχεδιασμού γίνεται πριν την οριστικοποίηση της μείωσης. Αν στο μπλοκ που έχει καθοριστεί για μείωση πληρείται η προϋπόθεση των vias που αναφέρθηκε παραπάνω τότε το τρανζίστορ είναι έτοιμο αλλαγή. Στο σημείο αυτό γίνεται έλεγχος των σχεδιαστικών περιορισμών που έχουν τεθεί από τον σχεδιαστή. Αν πληρούνται οι κανόνες σχεδίασης και δεν υπάρχει παραβίαση οριστικοποιείται το νέο μειωμένο μέγεθος του τρανζίστορ. Ειδάλλως γίνεται επαναπροσδιορισμός της μείωσης σύμφωνα με τους σχεδιαστικούς περιορισμός ή επαναφορά στις αρχικές διαστάσεις. Στην κάτωθι εικόνα παρουσιάζεται το διάγραμμα ροής της μεθοδολογίας που ακολουθήθηκε για την υλοποίηση του εργαλείου που περιγράφεται στην παρούσα διατριβή.



Εικόνα 5.8: Διάγραμμα ροής της μεθοδολογίας που ακολουθήθηκε για ελαχιστοποίηση κατανάλωσης ισχύος

6 Υλοποίηση

Στο παρόν κεφάλαιο θα γίνει αναφορά στα εργαλεία που χρησιμοποιήθηκαν, περιγραφή του κώδικα που υλοποιήθηκε για την πραγματοποίηση του εργαλείου που περιγράφεται στην παρούσα εργασία καθώς και παρουσίαση της χρονικής πολυπλοκότητας του.

6.1 Εργαλεία

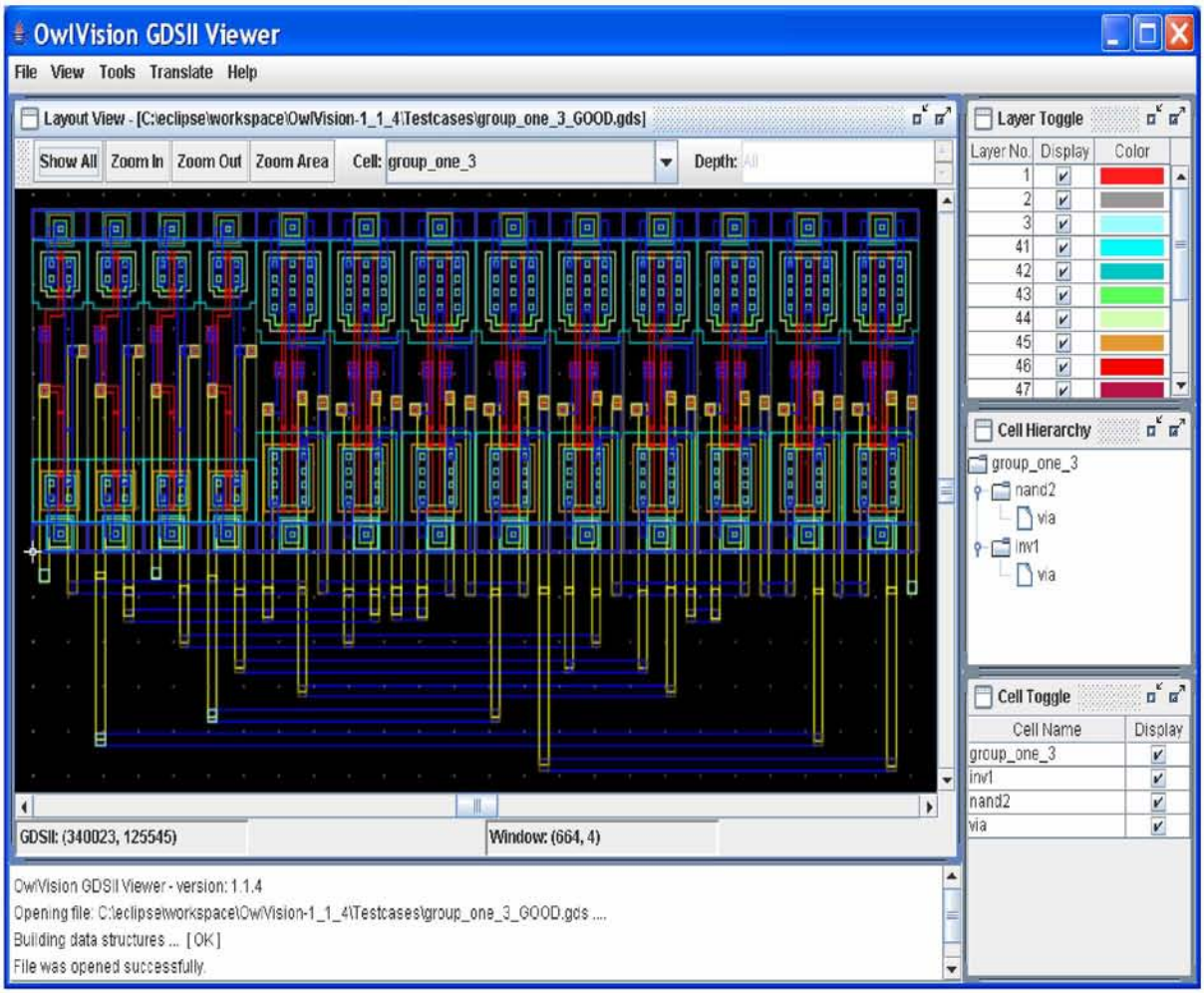
Για την υλοποίηση του εργαλείου χρησιμοποιήθηκαν τα παρακάτω εργαλεία:

Nangate Open Cell Library

Για τις ανάγκες κάλυψης της τεχνολογίας CMOS χρησιμοποιήθηκε η βιβλιοθήκη τυποποιημένων κελιών Nangate 45nm Open Cell Library και συγκεκριμένα η έκδοση NangateOpenCellLibrary_PDKv1_3_v2007_07. Η συγκεκριμένη βιβλιοθήκη είναι μία open source βιβλιοθήκη βασισμένη στο FreePDK45 που αναπτύχθηκε με σκοπό την έρευνα και την δοκιμή ροών αυτοματοποιημένης σχεδίασης κυκλωμάτων. Περιέχει όλα τα ευρέως χρησιμοποιούμενα τυποποιημένα κελιά, και κυρίως υλοποίηση κελιών σε GDSII.

Owl Vision GDSII viewer

Το λογισμικό “Owl Vision GDSII Viewer” αποτελεί ένα πρόγραμμα φυσικής αναπαράστασης εξόδου (Layout Viewer) ολοκληρωμένων κυκλωμάτων σε GDSII stream format. Το πρόγραμμα αυτό βοήθησε στην εκτίμηση και σύγκριση των πυλών που αναπαρίσταντο πριν και μετά την επεξεργασία τους από τον κώδικα που υλοποιήθηκε. Τα χαρακτηριστικά του εργαλείου περιλαμβάνουν συν τοις άλλοις δυνατότητα μετάφρασης ενός GDSII αρχείου σε ASCII και ανάποδα και να αναπαραστήσει συγκεκριμένα στοιχεία από κάθε δομή (Structure).



Εικόνα 6.1: Περιβάλλον λογισμικού "OwlVision GDSII Viewer"

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής είναι η γλώσσα προγραμματισμού C. Η επιλογή της συγκεκριμένης γλώσσας προγραμματισμού έγινε για την επίτευξη καλύτερων ταχυτήτων. Η ανάπτυξη του κώδικα και η αποσφαλμάτωση του έγινε σε περιβάλλον UNIX ενώ έγινε χρήση του μεταγλωττιστή gcc και των κατάλληλων παραμέτρων του. Επιπλέον, οι βιβλιοθήκες που χρησιμοποιήθηκαν είναι οι εξής:

- #include <stdio.h>
- #include <stdlib.h>
- #include <string.h>
- #include "lib/tran/transferfile.h"
- #include "lib/deco/decodefile.h"

6.2 Ανάλυση Μεθοδολογίας

Στην ενότητα αυτή περιγράφονται αναλυτικά τα βήματα του προγράμματος που υλοποιήθηκε, αναλύονται οι τεχνικές που χρησιμοποιήθηκαν, τους λόγους εφαρμογής τους καθώς και τα προβλήματα που προέκυψαν.

Το εργαλείο που δημιουργήθηκε, βασικό του στόχο έχει την ελαχιστοποίηση της κατανάλωσης ενέργειας σε standard cells. Οι μέθοδοι για την ελαχιστοποίηση της ενέργειας μπορεί να ποικίλουν και να πραγματοποιούνται με διάφορες τεχνικές όπως έχουμε προαναφέρει, ενώ στην παρούσα εργασία γίνεται με επέμβαση στο ρεύμα και κατ' επέκταση στο πλάτος των τρανζίστορ. Συγκεκριμένα, ο σχεδιαστής είναι αυτός που θα επιλέξει πόσο θέλει να μικρύνει το μέγεθος του πλάτους, εφόσον αυτό είναι εφικτό, καθώς σε συγκεκριμένες σχεδιάσεις υπάρχουν κάποια όρια ως προς το πόσο μειώνουν την κατανάλωση ισχύος. Σημαντικό βήμα για την πραγματοποίηση της εργασίας και ειδικότερα για την επέμβαση στις διαστάσεις των τρανζίστορ στοιχείων ήταν η αποκωδικοποίηση και ανάγνωση των αρχείων GDSII.

Η διεκπεραίωση της εργασίας χωρίστηκε σε έξι βασικά μέρη τα οποία είναι:

- Αποκωδικοποίηση των αρχείων GDSII
- Διάβασμα του αρχείου και των δεδομένων, διαχωρισμός των layer και εισαγωγή των μεγεθών τους σε λίστες
- Επεξεργασία του πλάτους των τρανζίστορ και τεχνικές προσαρμογής τους στο κύκλωμα
- Αλλαγή των vias και των υπολοίπων layers που εμπλέκονται- αν είναι απαραίτητο
- Έλεγχος τήρησης σχεδιαστικών κανόνων
- Κωδικοποίηση του νέου, επεξεργασμένου αρχείου με τα αλλαγμένα πλάτη σε αρχείο GDSII

6.2.1 Αποκωδικοποίηση αρχείων GDSII

Πρωταρχικό και καθοριστικό ρόλο για πραγματοποίηση της εργασίας έπαιξε η επίτευξη ανάγνωσης των αρχείων GDSII. Όπως αναλύθηκε και στο κεφάλαιο τέσσερα, πρόκειται για αρχεία σε δυαδική μορφή που αντιπροσωπεύουν τις επίπεδες γεωμετρικές μορφές των στοιχείων ενός cell, τις ετικέτες κειμένων και άλλες πληροφορίες για το layout με ιεραρχική δομή. Λόγω της δυαδικής αναπαράστασης του, τα GDSII αρχεία είναι δύσκολο να διαβαστούν γι αυτό και τα περιεχόμενα ενός αρχείου υπάρχουν εκφρασμένα σε μια πιο αναγνώσιμη μορφή, όπως σε μορφή ASCII.

Ένα παράδειγμα σε μορφή ASCII ενός GDSII αρχείου φαίνεται παρακάτω. Σύμφωνα με αυτό το format έγινε η μελέτη και αποκωδικοποίηση των αρχείων στην εργασία.

Επειδή το μέγεθός τους είναι αρκετά μεγάλο παρατίθεται η περιγραφή των δύο πρώτων layer (layer 2 και layer 3).

Αρχείο GDS της AND2_X2:

```
HEADER 3;  # version
BGNLIB;
 LASTMOD {109-7-17  17:21:30}; # last modification time
 LASTACC {109-7-17  17:21:30}; # last access time
LIBNAME NangateOpenCellLibrary;
UNITS;
 USERUNITS 1.0E-4;
 PHYSUNITS 1.0E-10;

BGNSTR;  # Begin of structure
 CREATION {109-7-17  17:21:30}; # creation time
 LASTMOD {109-7-17  17:21:30}; # last modification time
STRNAME AND2_X1;

BOUNDARY;
LAYER 3;
DATATYPE 0;
XY 5;
 X: -1150;                    Y: 5900;
 X: 8750;                     Y: 5900;
 X: 8750;                     Y: 15150;
 X: -1150;                    Y: 15150;
 X: -1150;                    Y: 5900;
ENDEL;

BOUNDARY;
LAYER 2;
DATATYPE 0;
XY 5;
 X: -1150;                    Y: -1150;
 X: 8750;                     Y: -1150;
 X: 8750;                     Y: 5900;
 X: -1150;                    Y: 5900;
 X: -1150;                    Y: -1150;
ENDEL;
```

Στόχος, συνεπώς, αποτελεί να αντληθούν οι πληροφορίες από τα αρχεία, να διαβάζονται και να επεξεργάζονται αυτόματα, καθώς η παραπάνω μορφή του ενδεικτικού μέρους βρίσκεται σε μορφή text. Η τεχνική που ακολουθήθηκε, ήταν η μετατροπή του GDSII αρχείου από δυαδική μορφή σε δεκαδική και εν συνεχεία η

αποθήκευσή του σε έναν πίνακα χαρακτήρων. Οι υπόλοιπες διαδικασίες και συναρτήσεις που υλοποιήθηκαν για τον διαχωρισμό των layers, την σμίκρυνση του πλάτους και της επεξεργασίας των δεδομένων έγιναν διαβάζοντας πλέον τον προαναφερθέντα πίνακα.

Βιβλιοθήκες "decodefile.h" και "transferfile.h"

Για την μεταφορά των αρχείων στην μνήμη δημιουργήθηκε η βιβλιοθήκη "transferfile.h"

- #include "lib/tran/transferfile.h"

Για την διαδικασία αποκωδικοποίησης συγκεντρώθηκαν όλες τις συναρτήσεις σε μια βιβλιοθήκη η οποία ονομάστηκε "decodefile.h"

- #include "lib/deco/decodefile.h"

Μεταφορά αρχείου στη μνήμη

Ξεκινώντας την διαδικασία αποκωδικοποίησης, πρωταρχικό μέλημα αποτελεί η μετατροπή κάθε αρχείου GDSII που λαμβάνεται σαν είσοδος στη μνήμη να επεξεργάζεται από εκεί. Συγκεκριμένα, δίνοντας το όνομα του αρχείου, επιστρέφεται το μέγεθός του και δεσμεύεται η κατάλληλη μνήμη. Κατόπιν, τα περιεχόμενα του αρχείου αποθηκεύονται σε κελιά των 32 bit σε έναν καθολικό πίνακα ("buffer") και στη συνέχεια μεταφέρονται στην μνήμη. Έπειτα, γίνεται έλεγχος της ορθής μεταφοράς τους εκτυπώνοντας τον πίνακα σε ένα αρχείο σε text μορφή ("buffer_note"). Οι συναρτήσεις που υλοποιήθηκαν για την διαδικασία μεταφοράς στην μνήμη περιγράφονται παρακάτω:

int file_length()	Λαμβάνει σαν όρισμα το όνομα ενός αρχείου και επιστρέφει το μέγεθος του
int *transfer_file()	Αποθήκευση όλων των δεδομένων του πίνακα που δίνεται σαν όρισμα σε κελιά των 32 Bit και μεταφορά τους στην μνήμη
int test_buffer()	Έλεγχος ορθής μεταφοράς του πίνακα στην μνήμη και εκτύπωσή του σε αρχείο μορφής κειμένου

Η εκτύπωση των περιεχομένων του "buffer" στο αρχείο "buffer_note" έγινε σε δεκαεξαδική αναπαράσταση χάριν ευκολίας (λόγω του χαρακτήρα του GDSII format) και ένα μέρος (20 πρώτων θέσεων) του αρχείου φαίνεται στην παρακάτω εικόνα.

ΑΡΧΕΙΟ "buffer_note"

1. 02000600
2. 1c000300
3. 6d000201
4. 11000700
5. 15001100
6. 6d001e00
7. 11000700
8. 15001100
9. 1a001e00
10. 614e0602
11. 7461676e
12. 65704f65
13. 6c65436e
14. 62694c6c
15. 79726172
16. 05031400
17. 8bdb683d
18. b40c71ac
19. 7ff36d38
20. ecf65e67

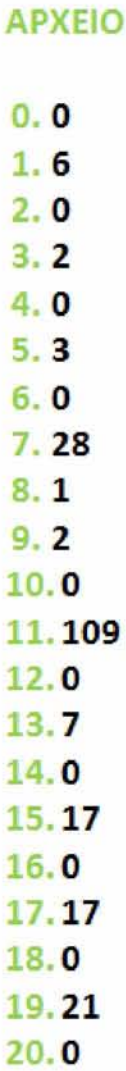
Εικόνα 6.2: Αναπαράσταση περιεχομένων του αρχείου "buffer_note". Με πράσινο φαίνονται οι θέσεις του πίνακα "buffer" ενώ με μαύρο τα περιεχόμενα του

Αποκωδικοποίηση αρχείου από την μνήμη

Στην φάση αυτή πραγματοποιείται η αποκωδικοποίηση των στοιχείων του αρχείου "buffer_note". Όπως ειπώθηκε και προηγουμένως, τα στοιχεία αυτά είναι αποθηκευμένα στην καθολική μεταβλητή "buffer". Η τεχνική που ακολουθήθηκε ήταν ο διαχωρισμός των binary αριθμών των 32 bit σε τέσσερα κελιά των οχτώ bit και η αποθήκευσή τους ένα-ένα σε byte σε έναν πίνακα χαρακτήρων σε μορφή ακεραίων ("data2use"). Κατόπιν, δεσμεύεται μνήμη για την μεταφορά του επεξεργασμένου πίνακα ("buffer") στον πίνακα χαρακτήρων ("data2use"). Η ορθότητα της μεταφοράς ελέγχεται με την εκτύπωσή του σε ένα αρχείο κειμένου ("char_note").
Οι συναρτήσεις που υλοποιήθηκαν είναι οι εξής:

int negativeNo()	Χειρισμός αρνητικών αριθμών σε δυαδική αναπαράσταση
void dec2bin()	Μετατροπή δεκαδικών αριθμών σε δυαδικούς
int bin2dec()	Χωρίζει έναν 32 bit δυαδικό αριθμό αυτό σε 4 μέρη των οχτώ bit και τα αποθηκεύει ανά οχτώ σε ένα byte. Κάθε byte αποθηκεύεται σε μία θέση πίνακα χαρακτήρων ("char_byte") μεγέθους τεσσάρων θέσεων. Κάθε πίνακας αναπαριστά έναν αριθμό 32 bit χωρισμένο σε τέσσερα κελιά, με κάθε κελί να περιέχει έναν χαρακτήρα σε ακέραια μορφή, μεγέθους ενός byte (8 bit).
char*decode_file()	Αποκωδικοποίηση του "buffer" πίνακα και δημιουργία πίνακα ("data2use") που περιλαμβάνει όλα τα στοιχεία του αποκωδικοποιημένου πίνακα σε χαρακτήρες και συγκεκριμένα με κάθε θέση του να περιέχει ένα χαρακτήρα, μεγέθους ενός byte, σε μορφή ακεραίου. Κατόπιν γίνεται μεταφορά του στην μνήμη για μετέπειτα επεξεργασία.
int test_data2use()	Έλεγχος της ορθής μεταφοράς του καθολικού ("data2use") πίνακα στην μνήμη και η εκτύπωση του σε ένα αρχείο μορφής κειμένου το ("char_note")

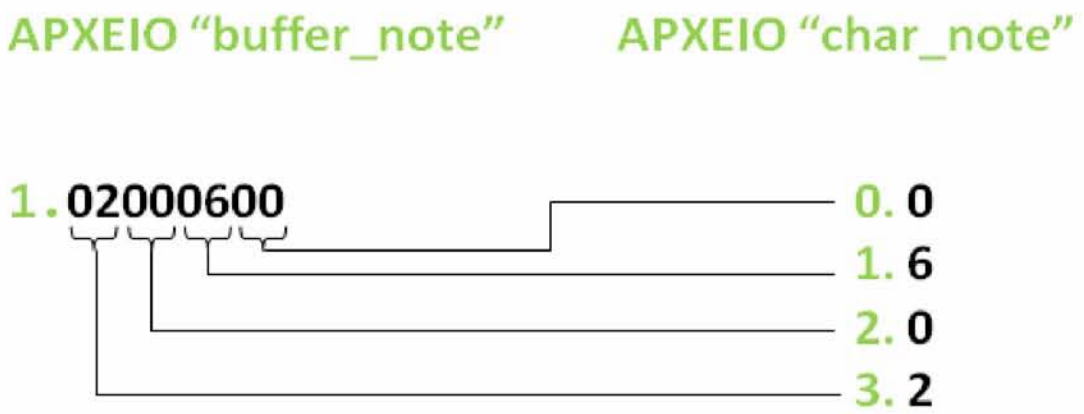
Το αποτέλεσμα που προέκυψε ήταν το αρχείου εισόδου GDSII να μετατραπεί σε μια πιο ευανάγνωστη μορφή όπως φαίνεται στην παρακάτω εικόνα, το οποίο πλέον επιτρέπει την αναγνώριση των header κάθε structure και συνεπώς την ανάγνωση και την επεξεργασία των δεδομένων του . Παρακάτω φαίνεται ένα μέρος (20 πρώτων θέσεων) των περιεχομένων του τελικού αρχείου "char_note"



Εικόνα 6.3: Αναπαράσταση περιεχομένων του αρχείου "char_note". Με πράσινο φαίνονται οι θέσεις του πίνακα "data2use" ενώ με μαύρο τα περιεχόμενα του

.

Η αναπαράσταση των περιεχομένων του αρχείου "char_note" που έχει αποθηκευμένο σε χαρακτήρες τα στοιχεία του "buffer" δεν γίνεται με αντιστοιχία όπως εύκολα διαπιστώνεται με την παρατήρηση των δύο τελευταίων εικόνων που παρουσιάστηκαν. Αυτό συμβαίνει διότι η αναπαράσταση από τα GDSII αρχεία των επίπεδων γεωμετρικών μορφών των στοιχείων ενός cell καθώς και όλων των άλλων πληροφοριών για το layout μπορεί να είναι σε δυαδική μορφή μεν, αλλά η ανάγνωσή τους γίνεται χιαστί και ανάποδα ανά τέσσερα bit, όπως φαίνεται στην κάτωθι εικόνα.



Εικόνα 6.4: Ανάγνωση ενός GDSII αρχείου μετά την μετατροπή του σε δεκαεξαδική μορφή και σε πίνακα χαρακτήρων χιαστί και με ανάποδη σειρά

6.2.2 Εντοπισμός θέσης και διαχωρισμός των layers

Το επόμενο βήμα της υλοποίησης είναι η δημιουργία συναρτήσεων που δίνουν την δυνατότητα ανάγνωσης του αρχείου, του εντοπισμού της θέσης κάθε εγγραφής καθώς και το πλήθος τους μέσα στο αρχείο, με την εισαγωγή μόνο του header της. Στην συνέχεια, γίνεται ο διαχωρισμός των layers που χρησιμοποιούνται από κάθε στοιχείο BOUNDARY και η εισαγωγή τους σε μία κύρια λίστα. Κάθε κόμβος (layer) της λίστας δείχνει με την σειρά του σε μια άλλη λίστα που περιέχει τις τιμές των δεδομένων (συντεταγμένες) που αντιστοιχούν σε κάθε layer.

Εντοπισμός θέσης

Σε αυτή την φάση γίνεται ο εντοπισμός της θέσης και του πλήθους των εγγραφών που περιγράφονται σε κάθε στοιχείο BOUNDARY (LAYER, DATATYPE, XY) μέσα στο αρχείο. Στο κεφάλαιο 4 (εικόνα 4.3) έγινε αναφορά στους header όλων των εγγραφών καθώς και στην αναπαράσταση των περιεχομένων ενός GDSII αρχείο σε μορφή ASCII (KEY format). Πιο αναλυτικά, υλοποιήθηκαν οι συναρτήσεις:

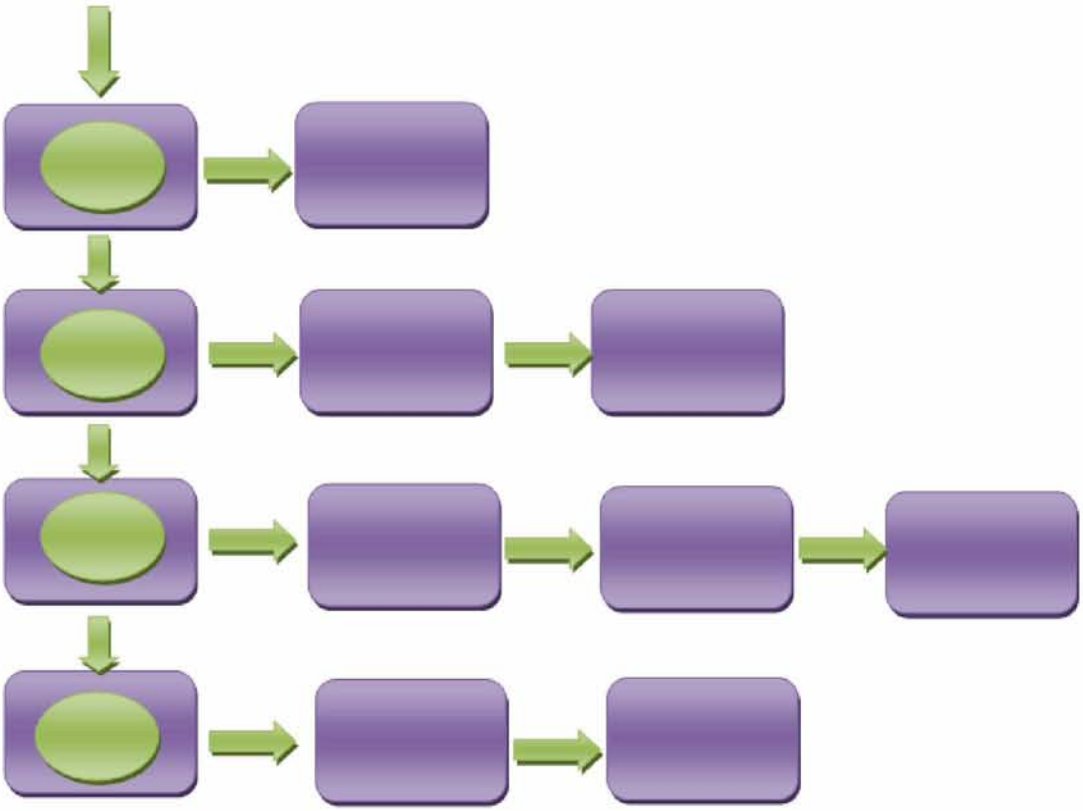
int layer_counter()	Επιστρέφει το πλήθος των layers που περιγράφονται από κάθε BOUNDARY στοιχείο. Συγκεκριμένα μετρά τις φορές που θα συναντήσει τον header του στοιχείου BOUNDARY μέσα στο αρχείο, έως ότου τελειώσει η περιγραφή των Layers
int *find()	Επιστρέφει έναν πίνακα με όλες τις θέσεις των εγγραφών που υπάρχουν στο αρχείο -σύμφωνα με το πλήθος αυτών- αλλά και τους header τους που εισάγουμε σαν όρισμα

Γνωρίζοντας, το πλήθος και τις θέσεις όλων των εγγραφών του BOUNDARY στοιχείου μπορεί να γίνει πλέον ο διαχωρισμός και η εισαγωγή τους σε λίστες.

Διαχωρισμός και εισαγωγή σε λίστες

Δημιουργείται μία κύρια λίστα όπου κάθε κόμβος της περιέχει τις εγγραφές κάθε BOUNDARY στοιχείου και δείχνει σε μια επόμενη λίστα όπου περιέχει τις συντεταγμένες που καθορίζουν το πολύγωνο που σχετίζεται με κάθε layer. Κάθε κόμβος περιέχει ένα ζεύγος συντεταγμένων X,Y.

Η δομή των λιστών φαίνεται παρακάτω:



Εικόνα 6.5: Δομή λιστών για τοποθέτηση των δεδομένων

Κάθε κουτί με κύκλο είναι δείκτης και περιέχει έναν σειριακό αριθμό, τον τύπο κάθε layer, την εγγραφή DATATYPE, την εγγραφή XY και των αριθμό ζεύγους των συντεταγμένων της. Κάθε άλλο κουτί περιέχει ένα ζεύγος συντεταγμένων που σχετίζεται με το περίγραμμα του πολυγώνου που περιγράφεται στο εκάστοτε BOUNDARY στοιχείο.

Οι συναρτήσεις που υλοποιήθηκαν για τον διαχωρισμό των layers και την εισαγωγή των δεδομένων τους σε λίστες είναι οι εξής:

int *x_values()	Αποθηκεύει σε ένα πίνακα όλες τις τιμές των συντεταγμένων X σε δεκαδικό, αφού πρώτα έχουν μετατραπεί από χαρακτήρες.
int *y_values()	Αποθηκεύει σε ένα πίνακα όλες τις τιμές των συντεταγμένων X σε δεκαδικό, αφού πρώτα έχουν μετατραπεί από χαρακτήρες.
int list_organisation()	Οργάνωση των λιστών. Μέσω των δομών της αρχικοποίησης και εισαγωγής σε λίστα δημιουργείται η κύρια λίστα ("layer_pointer_list") η οποία περιέχει έναν σειριακό αριθμό, τον τύπο του layer, του DATATYPE και την εγγραφή XY με τον αριθμό των ζεύγους συντεταγμένων που περιέχει. Στην συνέχεια υλοποιείται μία λίστα (**"**layer_data_list") που δείχνει κάθε κόμβος της κύριας και περιέχει όλες τις συντεταγμένες του πολυγώνου που περιγράφονται σε κάθε layer. Η δομή αυτή μεταφέρεται στην μνήμη για μετέπειτα επεξεργασία.
int list_organisation_print()	Έλεγχος της ορθής μεταφοράς στην μνήμη μέσω εκτύπωσης των λιστών σε ένα αρχείου κειμένου ("lista.txt").

Η μορφή των λιστών που λαμβάνουμε από το αρχείο "lista.txt" περιέχει όλες τις απαιτούμενες τιμές και τύπους εγγραφών. Ένας μέρος της (δύο πρώτα layer) φαίνεται παρακάτω:

ΑΡΧΕΙΟ "lista.txt"

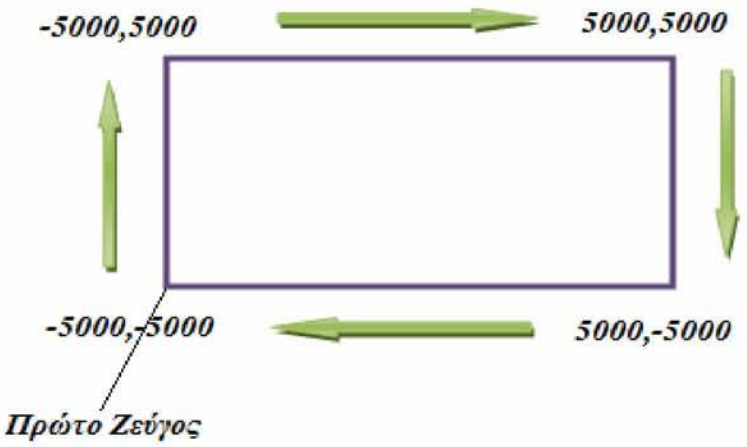
Serial: 0
Layer: 3
Datatype: 0
XY: 5
dataserial: 0 X: 400 Y: 1650
dataserial: 1 X: 3000 Y: 1650
dataserial: 2 X: 3000 Y: 2550
dataserial: 3 X: 400 Y: 2550
dataserial: 4 X: 400 Y: 1650

Serial: 1
Layer: 2
Datatype: 0
XY: 5
dataserial: 0 X: 400 Y:11450
dataserial: 1 X: 3000 Y:11450
dataserial: 2 X: 3000 Y:12800
dataserial: 3 X: 400 Y:12800
dataserial: 4 X: 400 Y:11450

Εικόνα 6.6: Αναπαράσταση περιεχομένων του αρχείου "lista.txt". Με πράσινο φαίνονται τα στοιχεία των κόμβων της λίστας ενώ με μαύρο τα στοιχεία της

6.2.3 Ελαχιστοποίηση του πλάτους των τρανζίστορ

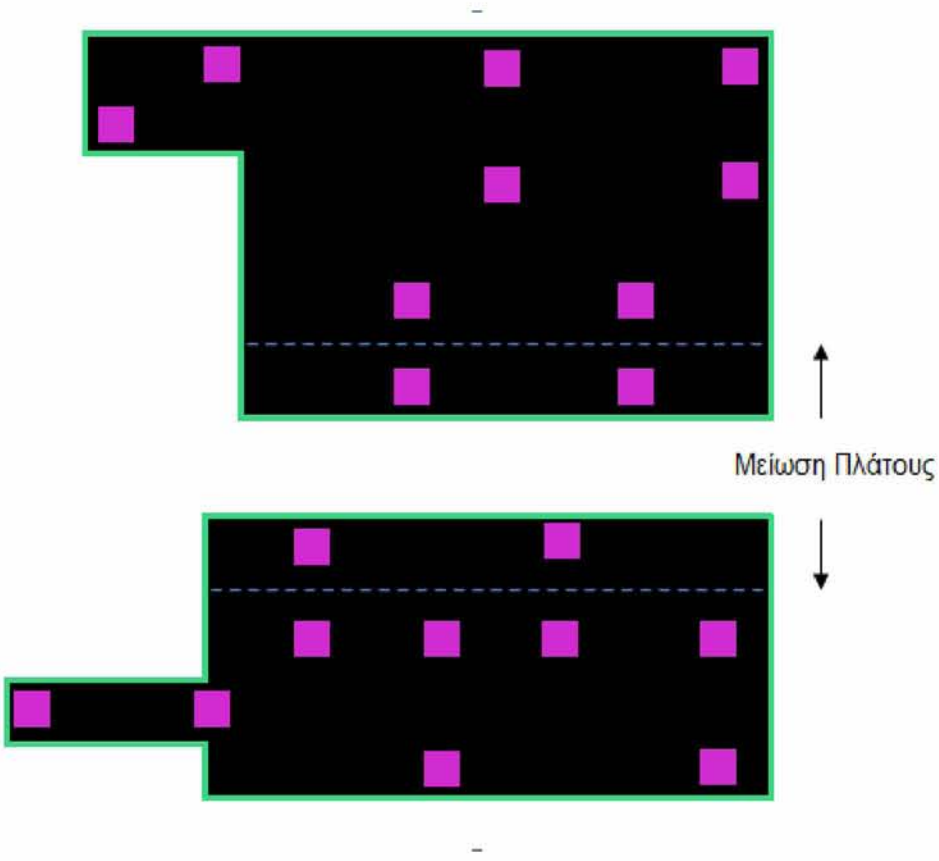
Στο τέταρτο κεφάλαιο αναφέρθηκε ότι η εγγραφή XY περιέχει το πλήθος ζευγών συντεταγμένων που περιγράφουν ένα πολύγωνο. Επιπλέον, το τελευταίο ζεύγος συντεταγμένων πρέπει να συμπίπτει με το πρώτο ζεύγος για να εγγυηθούμε τα κλειστά όρια του πολυγώνου. Η περιγραφή του πολυγώνου ξεκινάει από τα κάτω αριστερά και κινείται αριστερόστροφα όπως φαίνεται στο παρακάτω σχήμα:



Εικόνα 6.7: Αριστερόστροφη φορά κατά την περιγραφή του πολυγώνου, ξεκινώντας από το κάτω αριστερά ζεύγος

Στην ενότητα αυτή θα αναλυθεί η τεχνική που ακολουθήθηκε για την ελαχιστοποίηση του πλάτους των τρανζίστορ. Ο σχεδιαστής είναι αυτός που θα καθορίσει το μέγεθος της ελαχιστοποίησης, αναλόγως την περίπτωση και την πύλη που χρησιμοποιεί.

Στο κεφάλαιο πέντε αναφέρθηκε και επεξηγήθηκε ότι το πλάτος των τρανζίστορ πρέπει να μειώνεται με φορά από μέσα προς τα έξω για να αποφευχθεί ο κίνδυνος να βρεθούν εκτός επιφάνειας οι τάσεις τροφοδοσίας και τα μέταλλα. Στο πηγάδι τύπου n μειώνεται το πάνω μέρος από πάνω προς τα κάτω ενώ το πηγάδι τύπου p το κάτω μέρος από κάτω προς τα πάνω όπως φαίνεται στο παρακάτω σχήμα.

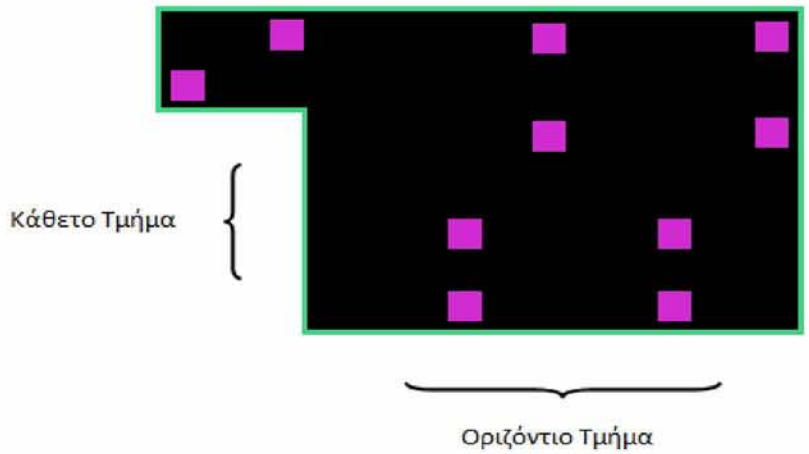


Εικόνα 6.8: Μείωση από πάνω προς τα κάτω στο τύπου n και από κάτω προς τα πάνω στο τύπου p.

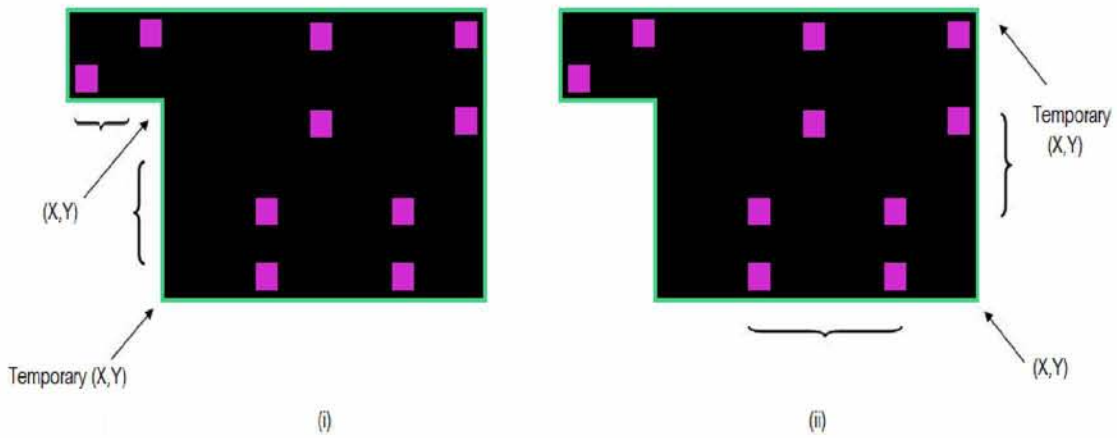
Σε κάθε layer , εξετάζονται οι συντεταγμένες που περιγράφουν το πολύγωνο με τον εξής τρόπο: Ένας δείκτης διατηρεί το πρώτο ζεύγος συντεταγμένων κι ένα άλλος temporary δείκτης διατηρεί το αμέσως επόμενο ζεύγος. Κάθε φορά και αφού οριστεί το block στο οποίο θα γίνει ελαχιστοποίηση πρέπει να γίνεται έλεγχος των Vias που λαμβάνουν χώρα και κατά πόσο είναι δυνατή η αφαίρεση τους από το block. Τα κριτήρια και οι συνθήκες για την διαδικασία αυτή περιγράφονται αναλυτικά στην αμέσως επόμενη ενότητα.

Για τρανζίστορ τύπου p

Κάθε φορά πρέπει να ελέγχονται τα ευθύγραμμα τμήματα που σχηματίζουν το πολύγωνο. Όταν από ένα οριζόντιο τμήμα ακολουθεί κάθετο τμήμα με φορά προς τα κάτω, δεν μπορεί να πραγματοποιηθεί μείωση πλάτους (εικόνα 6.10(i)). Η μείωση μπορεί να γίνει με αντικατάσταση των συντεταγμένων (στο μέγεθος που έχει ορίσει ο σχεδιαστής) σε ένα ή περισσότερα από τα κάτω οριζόντια τμήματα του πολυγώνου τα οποία ακολουθούνται από ένα κάθετο τμήμα με φορά προς τα πάνω (εικόνα 6.10(ii)).



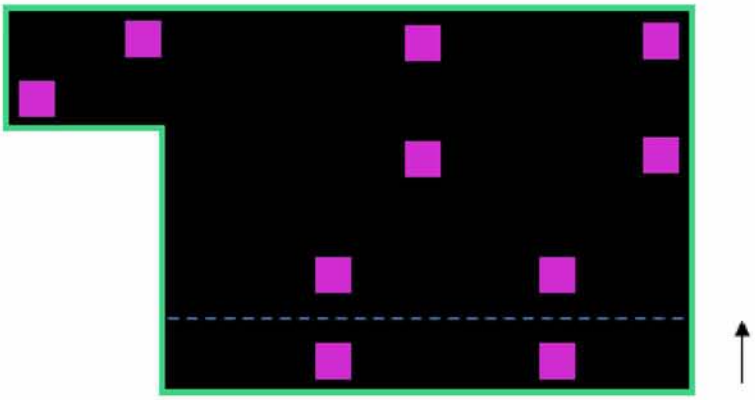
Εικόνα 6.9: Τμήματα ενός πολυγώνου. Αναπαράσταση τρανζίστορ με πράσινο, vias με μωβ



Εικόνα 6.10: (i) Στο οριζόντιο τμήμα της εικόνας δεν μπορούμε να παρέμβουμε εφόσον ακολουθεί κάθετο τμήμα με φορά προς τα κάτω
(ii) Στο οριζόντιο τμήμα της εικόνας μπορούμε να παρέμβουμε εφόσον ακολουθεί κάθετο τμήμα με φορά προς τα πάνω

Εάν η τιμή Y του δείκτη ταυτίζεται με την τιμή Y του δείκτη που έπεται και η τιμή του X αντίστοιχα μειώνεται ή αυξάνεται σημαίνει εντοπισμό σε οριζόντιο τμήμα του πολυγώνου. Επειδή όμως στα τρανζίστορ τύπου p η μείωση γίνεται από κάτω προς τα πάνω, αυτό που ενδιαφέρει είναι το οριζόντιο τμήμα του πολυγώνου όπου η τετμημένη αυξάνεται κατά μήκος της ευθείας από τα αριστερά προς τα δεξιά. Για να οριστεί το κατώτερο οριζόντιο τμήμα του πολυγώνου πρέπει να προσδιορίσουμε τα κάθετα τμήματα που προηγούνται και έπονται αυτού, ως εξής:

Εάν η τιμή X του δείκτη ταυτίζεται με την τιμή X του δείκτη που έπεται ενώ η τιμή Y αντίστοιχα μειώνεται σημαίνει εντοπισμό σε κάθετο τμήμα του πολυγώνου με φορά προς τα κάτω. Εάν η τιμή του Y αυξάνεται σημαίνει εντοπισμό σε κάθετο τμήμα πολυγώνου με φορά προς τα πάνω. Στο οριζόντιο τμήμα που παρεμβάλλεται σε δύο τέτοια ευθύγραμμα τμήματα (με το X να αυξάνεται κατά μήκος της ευθείας) επιτρέπεται στον σχεδιαστή να επέμβει. Στην συνέχεια, αφού γίνει ο έλεγχος και η πιθανή αφαίρεση των vias, αντικαθίστανται οι παλιές τιμές Y που ορίζουν την ευθεία με τις καινούργιες τιμές -που περιλαμβάνουν το μέγεθος για μείωση που επιθυμεί ο σχεδιαστής- και το πλάτος του τρανζίστορ μειώνεται.

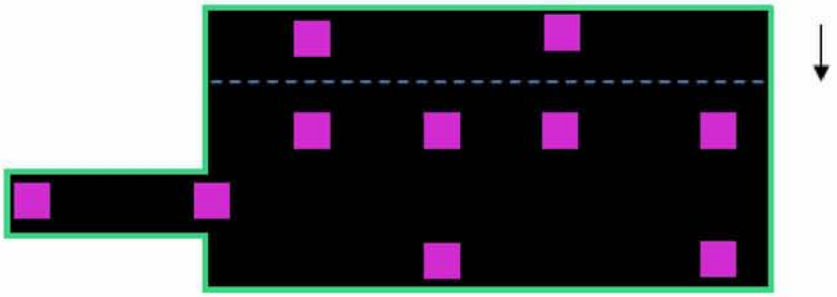


Εικόνα 6.11: Μείωση τρανζίστορ του κάθετου τμήματός του από κάτω προς τα πάνω

Για τρανζίστορ τύπου n

Ο προσδιορισμός των οριζόντιων και κάθετων τμημάτων, εδώ, γίνεται ομοίως με πριν με την διαφορά ότι τα τρανζίστορ τύπου n θα πρέπει να μειωθούν με φορά από πάνω προς τα κάτω. Συνεπώς, αυτό που προέχει είναι ο προσδιορισμός των ανώτερων οριζοντίων τμημάτων του πολυγώνου (με φθίνουσα τετμημένη κατά μήκος της ευθείας από τα δεξιά προς τα αριστερά).

Ένα τέτοιο ευθύγραμμο τμήμα θα πρέπει να παρεμβάλλεται ανάμεσα από ένα κάθετο τμήμα με φορά προς τα πάνω (X σταθερή, Y αύξουσα) και από ένα κάθετο τμήμα με φορά προς τα κάτω (X σταθερή, Y φθίνουσα). Στην συνέχεια ακολουθεί μείωση του πλάτους, εφόσον επιτρέπεται και έχει περάσει τον έλεγχο των vias, με την διαδικασία που περιγράψαμε παραπάνω και αντικαθιστώντας τις παλιές τιμές Y που ορίζουν την οριζόντια ευθεία με τις καινούργιες τιμές που περιλαμβάνουν το μέγεθος για μείωση που επιθυμεί ο χρήστης.



Εικόνα 6.12: Μείωση τρανζίστορ του κάθετου τμήματός του από κάτω προς τα πάνω

Η συνάρτηση που υλοποιήθηκε για την μείωση του πλάτους των τρανζίστορ είναι η εξής:

int reduce_transistor_width()	Η συνάρτηση αυτή παίρνει σαν όρισμα το μέγεθος της μείωσης του πλάτους. Χρησιμοποιείται ένας δείκτης που κρατά το πρώτο ζεύγος συντεταγμένων (X,Y) και ακόμα ένας προσωρινός δείκτης που κρατά το αμέσως επόμενο ζεύγος. Οι τιμές των X,Y λαμβάνονται από την δομή "data_layer_pointer_list". Στην συνέχεια, γίνεται έλεγχος των τιμών που διατηρούν οι δύο δείκτες, προσδιορίζονται τα κάθετα και οριζόντια τμήματα του τρανζίστορ και προχωρά στη μείωση του πλάτους εφόσον αυτό είναι επιτρεπτό

6.2.4 Αφαίρεση vias

Το τελευταίο βήμα αφορά την αφαίρεση των vias στον χώρο που ορίστηκε για ελαχιστοποίηση του πλάτους. Τα κριτήρια και η τεχνική που εφαρμόστηκε περιγράφονται στην παρούσα ενότητα.

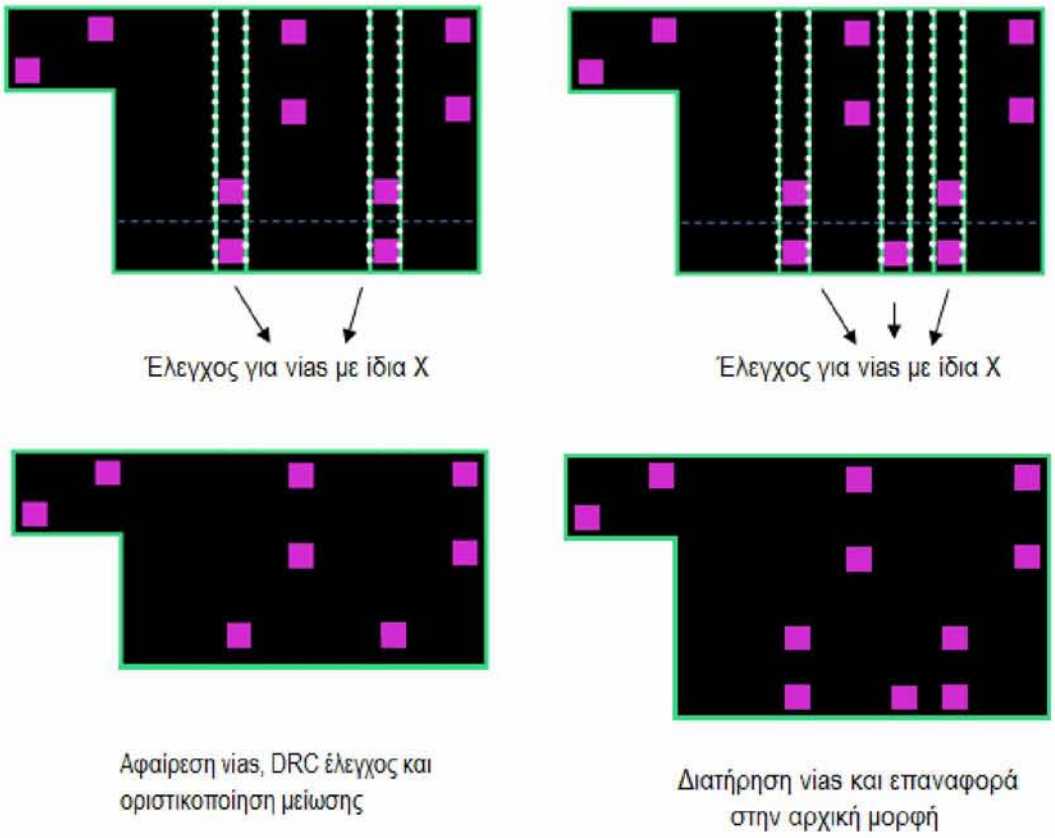
Όντας πλέον ορισμένο το block που επιτρέπει την ελαχιστοποίηση μέσα από την συνάρτηση "reduce_transistor_width" υλοποιείται μια άλλη συνάρτηση η οποία είναι υπεύθυνη για τον έλεγχο ύπαρξης vias στην περιοχή, την αφαίρεση τους -αν κρίνεται απαραίτητο- και αλλαγή των υπολοίπων layers που εμπλέκονται. Η διαδικασία εφαρμογής έχει ως εξής:

Σε πρώτη φάση διατηρούνται οι αρχικές συντεταγμένες που περιγράφουν το πολύγωνο για πιθανή επαναφορά του σε περίπτωση που δεν επιτραπεί μείωση του πλάτους μετά τον έλεγχο των vias. Ο έλεγχος γίνεται στο block που προέκυψε από την συνάρτηση "reduce_transistor_width" και που αφαιρέθηκε από το τρανζίστορ. Εάν εντοπιστούν vias μέσα στον χώρο του block, για να επιβεβαιωθεί η ορθότητα της μείωσης του πλάτους του τρανζίστορ πρέπει να ισχύει το εξής:

Για κάθε vias που λαμβάνει χώρα στο αποκομμένο τμήμα (block) θα πρέπει να υπάρχει τουλάχιστον ένα Vias με τις ίδιες τετμημένες (X, X') το οποίο να

περιλαμβάνεται μέσα στα όρια του πολυγώνου -που έχουν οριστεί μετά την μείωση του πλάτους του. Εφόσον κάτι τέτοιο ισχύει, τα vias αφαιρούνται και – αφού γίνει ο έλεγχος τήρησης σχεδιαστικών κανόνων- οριστικοποιούνται οι νέες διαστάσεις του μειωμένου τρανζίστορ. Σε αντίθετη περίπτωση, εάν εντοπιστεί έστω και ένα vias στο αποκομμένο τμήμα που δεν πληροί την παραπάνω προδιαγραφή δεν πραγματοποιείται μείωση του πλάτους και το τρανζίστορ επιστρέφει στην αρχική του μορφή (Εικόνα 6.12).

Ο λόγος που πρέπει να ισχύουν οι παραπάνω περιορισμοί, όπως επεξηγήθηκε και στο κεφάλαιο πέντε αναλυτικά, είναι επειδή πρέπει να υπάρχει ένα τουλάχιστον vias για κάθε τάση τροφοδοσίας, προκειμένου να επιτευχθεί ορθή λειτουργία του κυκλώματος.



Εικόνα 6.13: Έλεγχος για αφαίρεση vias και για οριστική μείωση του πλάτους των τρανζίστορ (μετά από επιτυχή DRC έλεγχο).

Η συνάρτηση που υλοποιήθηκε για τον έλεγχο και την αφαίρεση των vias είναι η εξής:

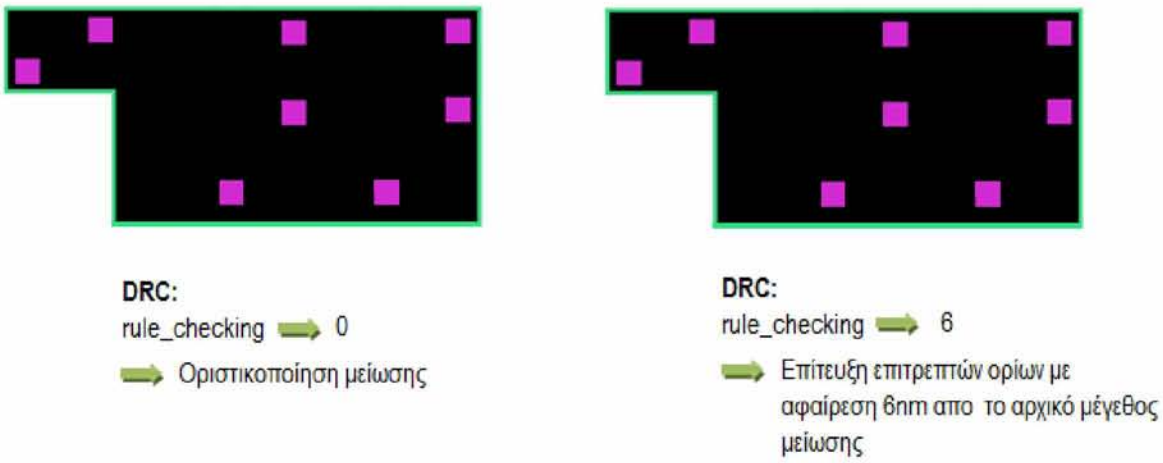
int remove_vias_transistor()	Διατηρεί τις συντεταγμένες (X,Y) από τα αρχικά όρια του πολυγώνου προς επεξεργασία και εν συνεχεία κάνει έλεγχο ύπαρξης Vias στο στα όρια του αποκομμένου τμήματος. Γνωρίζοντας τις συντεταγμένες του αποκομμένου τμήματος και των vias ελέγχεται κατά πόσο τα vias βρίσκονται μέσα στα όρια της αποκομμένης περιοχής. Για καθένα από αυτά, εξετάζεται εάν υπάρχει vias με τις ίδιες τιμές X μέσα στα όρια του μειωμένου τρανζίστορ και προβαίνει σε αφαίρεση των vias- εάν επιτρέπεται- και μείωση ή επαναφορά των διαστάσεων του πολυγώνου. Σε αυτή την φάση και πριν την οριστικοποίηση της μείωσης καλείται και η συνάρτηση ελέγχου τήρησης των σχεδιαστικών κανόνων που έχουν τεθεί από τον σχεδιαστή. Οι κόμβοι των layers που περιγράφουν τα vias που αφαιρέθηκαν αποκόπτονται από την δομή "layer_pointer_list" ενώ οι τελικές τιμές των συντεταγμένων του εκάστοτε πολυγώνου αντικαθίστανται στην δομή " layer_data_list".

6.2.5 Έλεγχος DRC

Στην επόμενη φάση πρέπει να γίνει έλεγχος και το προσαρμογή σύμφωνα με προκαθορισμένους γεωμετρικούς κανόνες σχεδίασης που υπαγορεύει το DRC (Design Rule Check) πρόγραμμα. Αν στο μπλοκ που έχει καθοριστεί για μείωση πληρείται η προϋπόθεση των vias που αναφέρθηκε παραπάνω τότε το τρανζίστορ είναι έτοιμο αλλαγή. Στο σημείο αυτό γίνεται έλεγχος τήρησης των σχεδιαστικών κανόνων που έχουν τεθεί από τον σχεδιαστή. Ο έλεγχος γίνεται ταυτόχρονα και μέσα στην συνάρτηση αφαίρεσης των vias για λόγους επίτευξης μεγαλύτερων ταχυτήτων. Αν πληρούνται οι κανόνες και δεν υπάρχει παραβίαση οριστικοποιείται το νέο μειωμένο μέγεθος του τρανζίστορ. Σε αντίθετη περίπτωση γίνεται επαναπροσδιορισμός του αποκομμένου τμήματος για μείωση σύμφωνα με τους σχεδιαστικούς περιορισμούς ή επαναφορά στις αρχικές διαστάσεις. Συγκεκριμένα υλοποιήθηκε η εξής συνάρτηση:

int rule_checking()	Η συνάρτηση παίρνει ως ορίσματα τα layer που αλληλεπιδρούν, το είδος αλληλεπίδρασης που έχουν (width, spacing, enclosure) και την τιμή μείωσης που έχει αποφασιστεί από τον χρήστη. Οι τιμές των σχεδιαστικών περιορισμών αποθηκεύονται σε έναν πίνακα από doubles. Κατά την κλήση της συνάρτησης οι τιμές αυτές συγκρίνονται με τις αποστάσεις που προκύπτουν, κατά την μείωση, ανάμεσα στα layers. Αν οι περιορισμοί τηρούνται επιστρέφεται μηδέν. Ειδάλλως, επιστρέφεται η επιτρεπόμενη τιμή μείωσης που μπορεί να θέσει ο

	χρήστης.

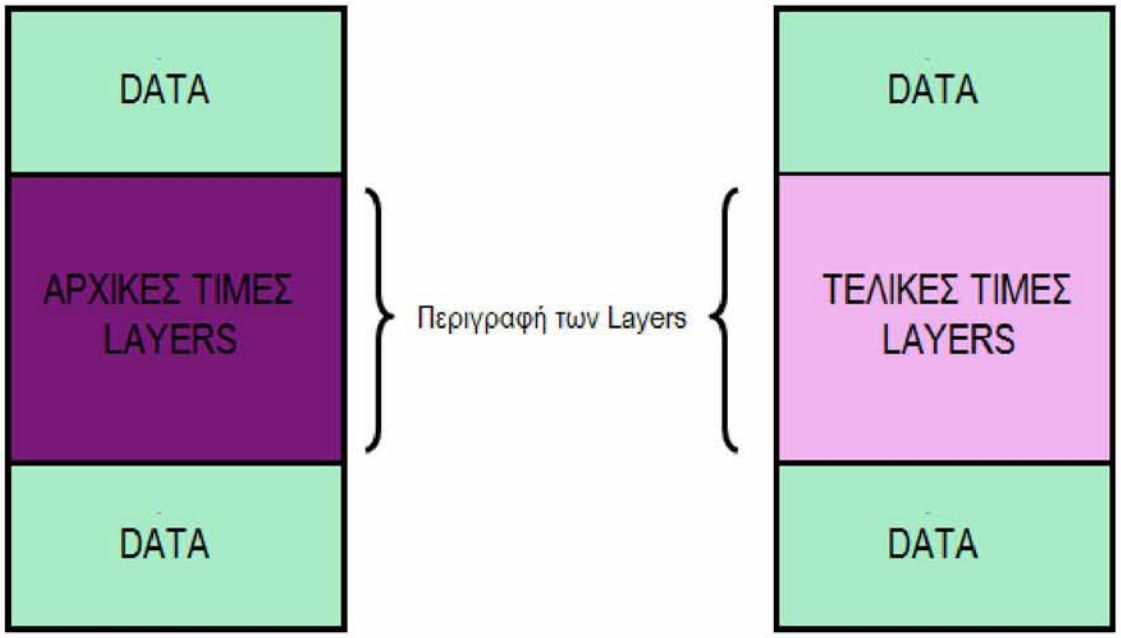


Εικόνα 6.14: Διαδικασία έλεγχου τήρησης σχεδιαστικών κανόνων. Στην αριστερή περίπτωση οι κανόνες τηρούνται και γίνεται η μείωση πλάτους. Στην δεξιά περίπτωση ο χρήστης πρέπει να επαναπροσδιορίσει το μέγεθος μείωσης κατά 6nm για να επιτύχει επιτρεπτά όρια

6.2.6 Κωδικοποίηση σε μορφή GDSII

Το τελευταίο βήμα της μεθοδολογίας είναι η δημιουργία ενός νέου αρχείου GDSII που θα περιέχει τις νέες τιμές συντεταγμένων και τα αλλαγμένα layers. Η διαδικασία έχει ως εξής:

Δημιουργείται ένα νέο αρχείο το οποίο περιέχει όλα τα δεδομένα του αρχικού πίνακας "buffer" μέχρι πριν την περιγραφή των layers (μέχρι τον πρώτο header της εγγραφής LAYER) και τα δεδομένα που υπάρχουν μετά την περιγραφή των layers. Στο νέο αρχείο και ανάμεσα σε αυτά τα δεδομένα αντιγράφονται οι νέες τιμές των layers που διατηρούνται στις δομές λιστών. Τα νέα δεδομένα μεταφέρονται στην μνήμη και ο έλεγχος ορθής μεταφοράς γίνεται από το αρχείο "fbuffer_note". Το νέο αρχείο "fbuffer_note" είναι της μορφής GDSII παρέχοντας την δυνατότητα στον χρήστη να το χρησιμοποιήσει σαν αρχείο εισόδου σε οποιοδήποτε πρόγραμμα φυσικής αναπαράστασης εξόδου (Layout Viewer) ολοκληρωμένων κυκλωμάτων σε GDSII stream format.



Εικόνα 6.15: Αναπαράσταση δημιουργίας τελικού αρχείου GDSII με τις νέες τιμές των layers. Αριστερά τα στοιχεία του "buffer" αρχείου και δεξιά τα στοιχεία του "fbuffer" με τις νέες τιμές του

Η συνάρτηση που υλοποιήσαμε για την πραγματοποίηση των παραπάνω είναι η εξής:

int encode_file()	Είναι υπεύθυνη για την αντιγραφή των δεδομένων που λαμβάνουν χώρα πριν και μετά την περιγραφή των layers από τον πίνακα "buffer", την αντικατάσταση των νέων αλλαγμένων layers και την μεταφορά τους στην μνήμη

6.3 Χρονική πολυπλοκότητα συναρτήσεων

Κατά την σχεδίαση ενός αλγορίθμου, είναι στοιχειωδώς απαραίτητο να ορισθεί το κριτήριο ή τα κριτήρια τα οποία πρέπει να εκπληρώνει ώστε να μπορεί να χαρακτηρισθεί το κατά πόσον είναι ή όχι αποδοτικός. Και απόδοση σημαίνει η χρήση υπολογιστικών πόρων που απαιτούνται για την επίλυση του προβλήματος. Οι υπολογιστικοί πόροι μπορεί να είναι η CPU, η μνήμη ή ακόμη και δικτυακοί πόροι. Ουσιαστικά όμως στα περισσότερα προβλήματα αυτό που εξετάζεται είναι η χρήση της CPU, δηλαδή, πόσος χρόνος απαιτείται για την εκτέλεση του αλγορίθμου.

Σαν πολυπλοκότητα ή απόδοση αλγορίθμου ορίζεται το κόστος χρήσης του αλγορίθμου για την βλήματος. Το πρόβλημα χαρακτηρίζεται από τα δεδομένα εισόδου ενώ η συνάρτηση πολυπλοκότητας εκφράζει την απαίτηση του αλγορίθμου σε χρόνο εκτέλεσης σε σχέση με τω δεδομένων εισόδου n. Ανάμεσα στις διάφορες μονάδες μέτρησης κόστους ο χρόνος εκτέλεσης θεωρείται σαν το πιο σημαντικό μέτρο χαρακτηρισμού.

Παρακάτω παρουσιάζονται οι πολυπλοκότητες και επεξήγησή των, των συναρτήσεων κάθε βιβλιοθήκης που υλοποιήθηκαν και που περιγράφηκαν στην προηγούμενη ενότητα καθώς επίσης και κάποιες από τις βοηθητικές συναρτήσεις που χρησιμοποιήθηκαν κατά την υλοποίηση του προγράμματος.

- *Deco library*

Συνάρτηση	**Πολυπλοκότητα**
int negativeNo(char *binary)	O(n) στο μέγεθος του πίνακα. Αλλάζει πρόσημο σε ένα αριθμό
char *dec2bin(long decimal)	O(1) αλλαγή ενός αριθμού
char *decode_file(int *input_matix_buffer,int input_buffer_length)	O(n) στο μέγεθος του πίνακα. Μεταφέρει το GDSII σε char μνήμη
int header_finder(char *input_matrix_data2use,int input_header1,int input_header2)	O(n) στο μέγεθος του πίνακα
int header_finder(char *input_matrix_data2use,int input_header1,int input_header2)	O(n) στο μέγεθος του πίνακα για να βρει τον header

- *Tran library*

Συνάρτηση	Πολυπλοκότητα
int file_legth(char *namegiven)	O(n) στο μέγεθος του αρχείου για να επιστρέψει το μέγεθος του αρχείου
int *transfer_file(char *namegiven)	O(n) στο μέγεθος του αρχείου για να το περάσει στην μνήμη
int test_buffer(int *input_matix_buffer,int input_buffer_length)	O(n) στο μέγεθος του πίνακα για ελέγξει τον buffer

- *Main*

Συνάρτηση	Πολυπλοκότητα
int char2int_convert(char input1,char input2,char input3,char input4)	O(1) μετατροπή 4 char σε int
void encode_file(char *data2use,int end_point)	O(n) επανατοποθετεί τη λίστα σε ένα νέο αρχείο
int layer_pointer_insert(int layer_serial, int layer, int datatype, int xy)	O(n) στο μέγεθος της λίστας. Εισαγωγή layer
int layer_pointer_list_insert_reverse(int layer_serial, int layer, int datatype, int xy)	O(n) στο μέγεθος της λίστας των layer. Εισαγωγή στοιχείων στο layer
int layer1_pointer_insert_reverse(int layer_serial, int layer, int datatype, int xy)	O(n) στο μέγεθος της λίστας. Εισαγωγή layer
int layer_pointer_list_remove(int layer_serial)	O(n) στο μέγεθος της λίστας. Εξαγωγή layer
struct layer_pointer_list *list_hasElement(int layer_serial, int layer)	O(n) στο μέγεθος της λίστας. Αναζήτηση για layer
void layer_data_list_remove(int layer, int layer_serial, int layer_data_serial)	$O(n^2)$ στο μέγεθος της λίστας και των στοιχείων των layer. Εξαγωγή των στοιχείων από ένα block
int layer_counter(char *input_matrix_data2use,int input_header1,int input_header2)	O(n) στο μέγεθος της λίστας. Μετρά τα layers

int last_layer_count(char * input_ matrix_ data2use,int input_header1,int input_header2)	O(n) στο μέγεθος της λίστας. Βρίσκει το τελευταίο layer
int layer_finder(char *input_matrix_data2use,int input_header1,int input_header2,int input_offset)	O(n) στο μέγεθος της λίστας. Αναζητά τη θέση στον πίνακα σε ένα συγκεκριμένο layer
int layer_data_list_insert(int layer_data_serial, int x, int y, struct layer_pointer_list *l_input)	O(n) στο μέγεθος της λίστας. Εισαγωγή των στοιχείων ενός block στο συγκεκριμένο layer της λίστας
int list_organisation(char *data2use)	O(n) στο μέγεθος του αρχείου. Τοποθετεί το αρχείο από ένα πίνακα στη λίστα
int list_organisation_print()	O(n) στο μέγεθος της λίστας. Τυπώνει τη λίστα σε ένα αρχείο
int reduce_transistor_width(int number_to_reduce)	$O(n^2)$ στο μέγεθος της λίστας και των στοιχείων των layer. Αναζητά και μειώνει το πλάτος των τρανζίστορ στη λίστα σε συγκεκριμένο layer και συγκεκριμένο block
int remove_vias_transistor()	$O(n^2)$ στο μέγεθος της λίστας και των στοιχείων των layer. Αναζητά και αφαιρεί τα vias των τρανζίστορ στη λίστα σε συγκεκριμένο layer και συγκεκριμένο block

Οι δύο τελευταίες συναρτήσεις είναι αυτές που ευθύνονται για την πολυπλοκότητα. Η reduce_transistor_width βλέπει το πλάτος που είναι προς μείωση και βάση αυτής της πληροφορίας κάνει αναζήτηση σε δύο διαστάσεις της λίστας, μήκος και πλάτος. Ομοίως, η remove_vias_transistor αναζήτα τα vias προς αφαίρεση σε συγκεκριμένο layer και μπλοκ εκτελώντας επίσης αναζήτηση της λίστας στις δύο διαστάσεις. Με αυτόν τον τρόπο ο χρόνος για αλλαγή μεγέθους ή διαγραφή είναι ο ίδιος , δηλαδή n^2.

Μια πολυπλοκότητα, όμως, της τάξης $O(n^2)$ είναι απαγορευτική για την ταχύτητα και την απόδοση του προγράμματος. Συνεπώς, προέκυψε και η ανάγκη για βελτιστοποίηση. Η βελτίωση που πραγματοποιήθηκε για τις συγκεκριμένες συναρτήσεις ήταν με πρόβλεψη θέσης και αυτόματη επέμβαση. Για την ακρίβεια, κάθε φορά που πρέπει να γίνει επέμβαση στην γεωμετρία, κατά την διαδικασία μείωσης ή κατά την αφαίρεση των vias , δεν γίνεται άσκοπη διάτρεξη της δισδιάστατης λίστας για να βρεθεί το layer στο οποίο θα γίνει επεξεργασία αλλά απευθείας μετάβαση στο επιθυμητό μπλοκ και επεξεργασία (μείωση πλάτους ή

αφαίρεση Vias). Με τον τρόπο αυτό η ο χρόνος γίνεται απευθείας n (πολυπλοκότητα O(n)) πετυχαίνοντας σαφώς καλύτερες ταχύτητες εκτέλεσης.

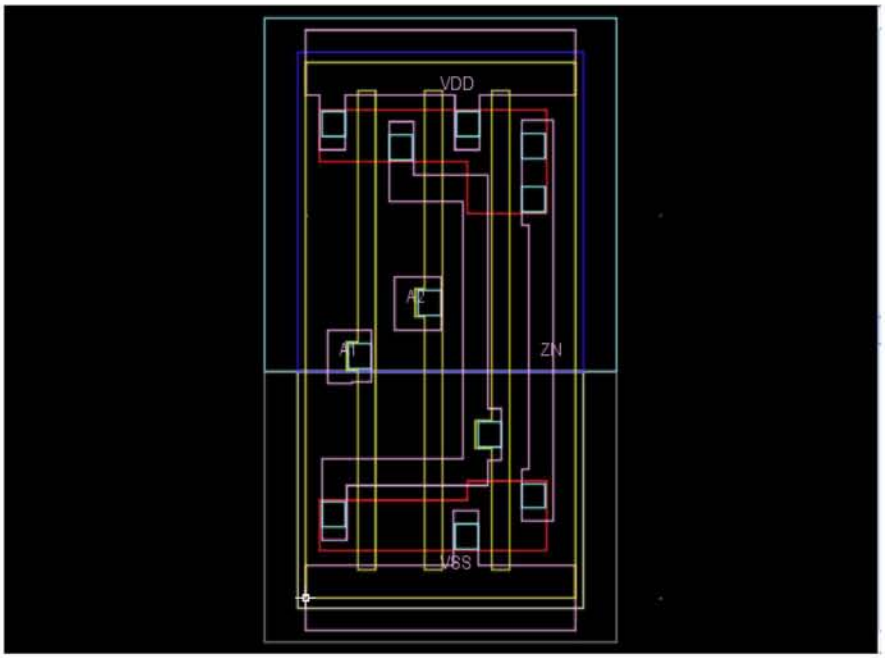
6.4 Ανακεφαλαίωση ανάλυσης μεθοδολογίας

Κλείνοντας την ανάλυση του προγράμματος, καταφέραμε να κάνουμε parse τα GDSII αρχεία που λάβαμε σαν είσοδο, να αλλάξουμε τα πλάτη των τρανζίστορ και να αφαιρέσουμε τα vias, μετά από κατάλληλους ελέγχους και εφόσον αυτό ήταν επιτρεπτό. Έπειτα, έγινε ο έλεγχος τήρησης των σχεδιαστικών κανόνων που είχε θέσει ο κατασκευαστής και αναλόγως το αποτέλεσμα έγινε οριστικοποίηση των αλλαγών η επαναφορά της αρχικής κατασκευής Τέλος μεταφέραμε τις αλλαγές σε ένα νέο GDSII αρχείο το οποίο μπορούμε πλέον να το χρησιμοποιήσουμε σαν νέο αρχείο εισόδου σε προγράμματα αναπαράστασης εξόδου για την πραγματοποίηση των σχετικών μετρήσεων.

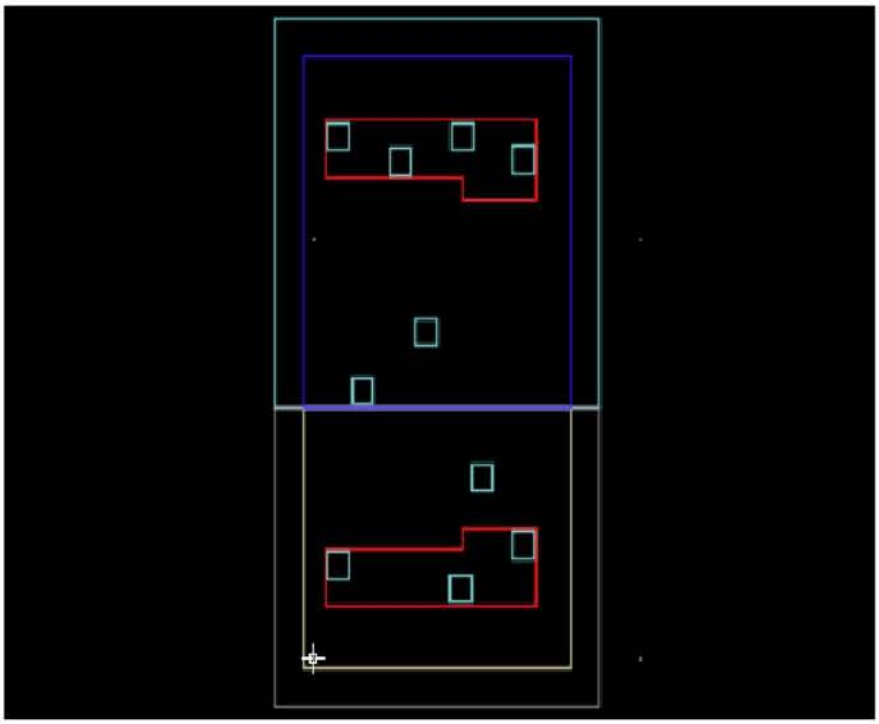
7 Πειραματικά Αποτελέσματα

Στον παρόν κεφάλαιο θα γίνει μια παρουσίαση χρήσης του εργαλείου που υλοποιήθηκε και συγκεκριμένα των αποτελεσμάτων που αυτό εξάγει.
Παρακάτω παρατίθενται κάποια παραδείγματα. Η εφαρμογή του προγράμματος έγινε εισάγοντας αρχεία που περιγράφουν πύλες σε GDSII μορφή. Τα GDSII αρχεία που χρησιμοποιήθηκαν είναι της βιβλιοθήκης NangateOpenCell. Το εργαλείο για την φυσική αναπαράσταση εξόδου είναι το Owl Vision GDSII Viewer.
Συγκεκριμένα, παρουσιάζεται το layout των πυλών πριν και μετά την μείωση πλάτους των τρανζίστορ. Επίσης γίνεται αναφορά στις «χειρότερες» περιπτώσεις πυλών όπου το εργαλείο δεν επιτρέπεται να επέμβει στις διαστάσεις του κυκλώματος λόγω περιορισμού σχεδιαστικών κανόνων ή μη ορθότητας λειτουργίας του κυκλώματος .
Τα αρχεία που χρησιμοποιήθηκαν ως είσοδοι περιγράφουν τις πύλες AND2_X2, BUF_X16, BUF_X32, INV_X1, AND2_X1.

Layout GDSII αναπαράστασης της πύλης "AND2_X2" πριν και μετά την μείωση

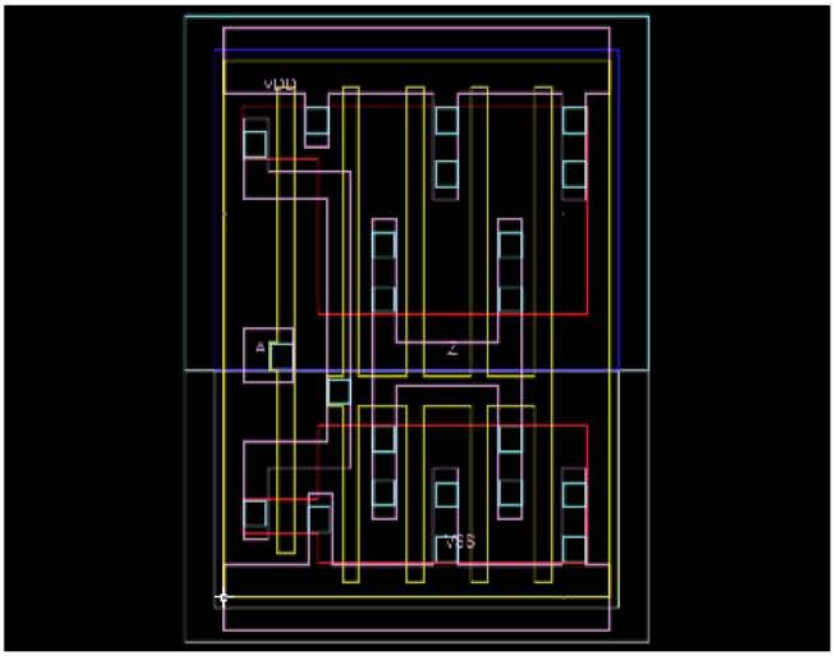


Layout με μειωμένα πλάτη

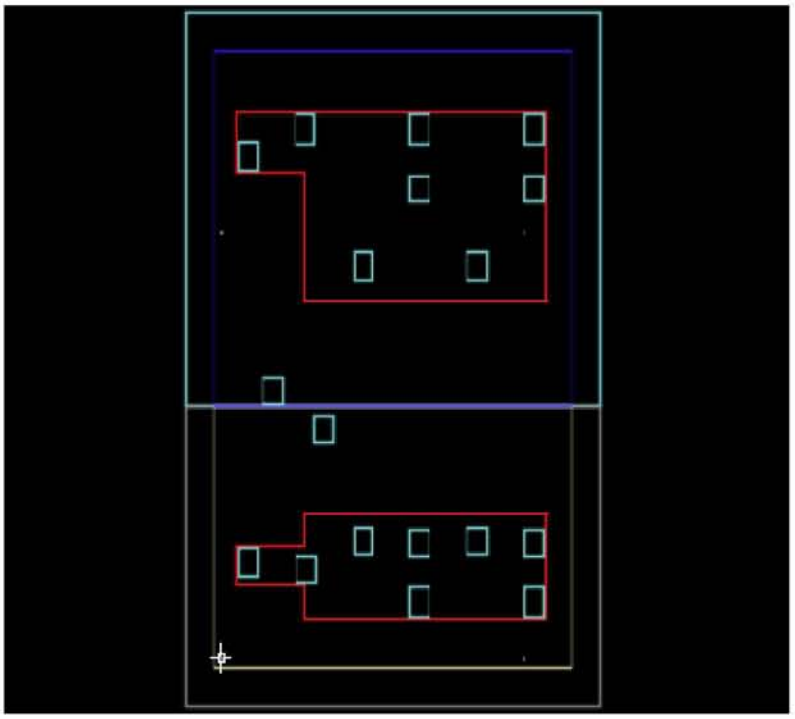


Εδώ παρουσιάζεται η πύλη AND2_X2 της βιβλιοθήκης σε φυσική αναπαράσταση μέσω του εργαλείου OwlVision GDSII Viewer.
Μετά την μείωση των τρανζίστορ που αναπαρίστανται με το κόκκινο (layer 1) το τελικό Layout είναι αυτό που φαίνεται στην τελευταία εικόνα

Layout GDSII αναπαράστασης της πύλης "BUF_X16" πριν και μετά την μείωση

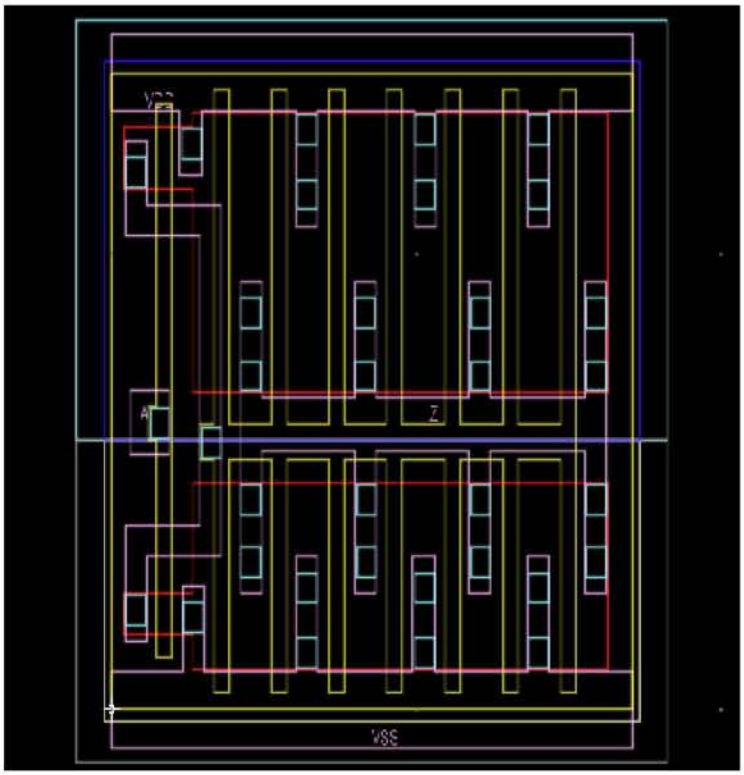


Layout με μειωμένα πλάτη

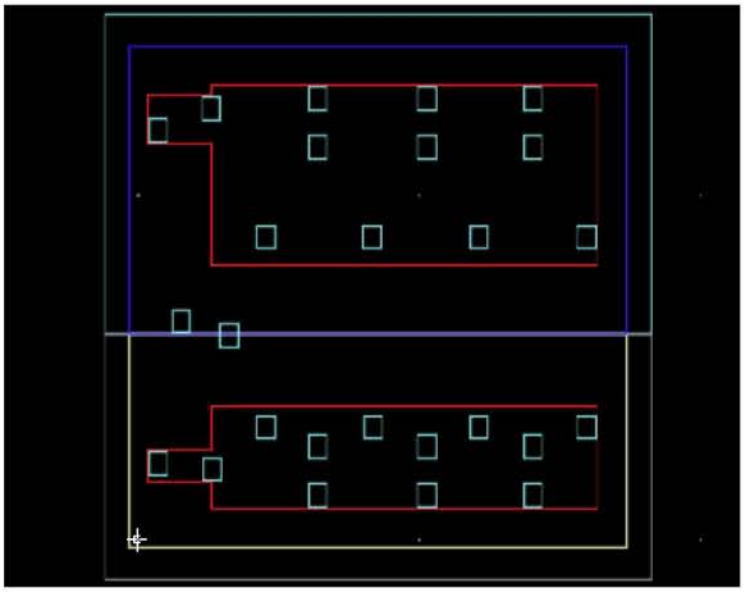


Εδώ παρουσιάζεται η πύλη BUF_X16 της βιβλιοθήκης Nangate σε φυσική αναπαράσταση μέσω του εργαλείου OwlVision GDSII Viewer.
Μετά την μείωση των τρανζίστορ που αναπαρίστανται με το κόκκινο (layer 1) το τελικό Layout είναι αυτό που φαίνεται στην τελευταία εικόνα

Layout GDSII αναπαράστασης της πύλης "BUF_X32" πριν και μετά την μείωση

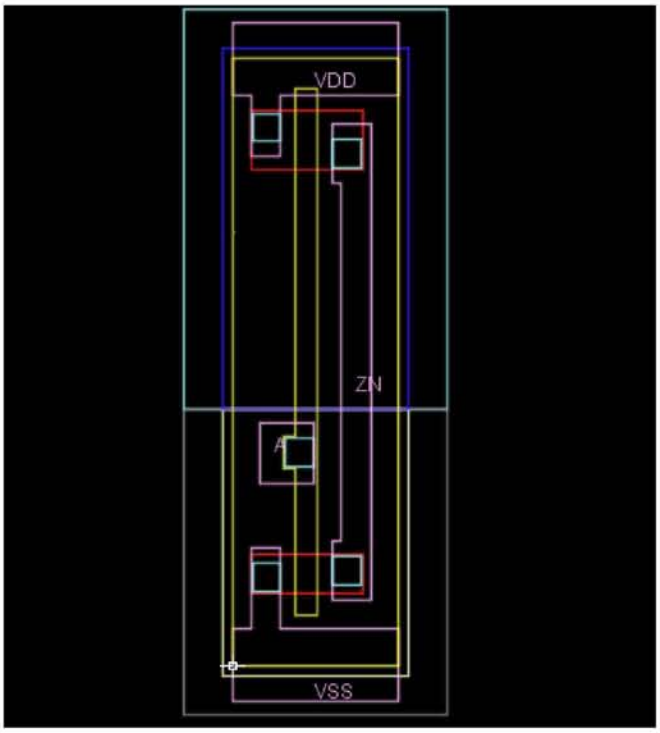


Layout με μειωμένα πλάτη

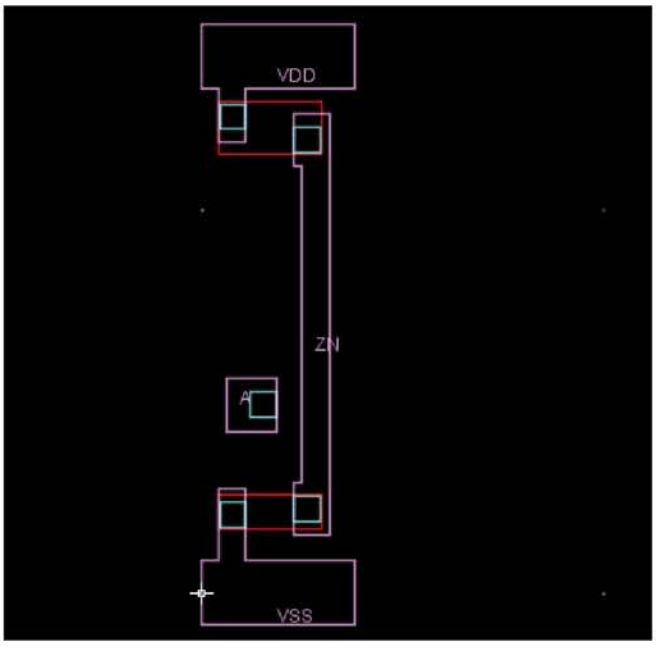


Εδώ παρουσιάζεται η πύλη BUF_X32 της βιβλιοθήκης Nangate σε φυσική αναπαράσταση μέσω του εργαλείου OwlVision GDSII Viewer.
Μετά την μείωση των τρανζίστορ που αναπαρίστανται με το κόκκινο (layer 1) το τελικό Layout είναι αυτό που φαίνεται στην τελευταία εικόνα

Layout GDSII αναπαράστασης της πύλης "INV_X1" χωρίς δυνατότητα μείωσης

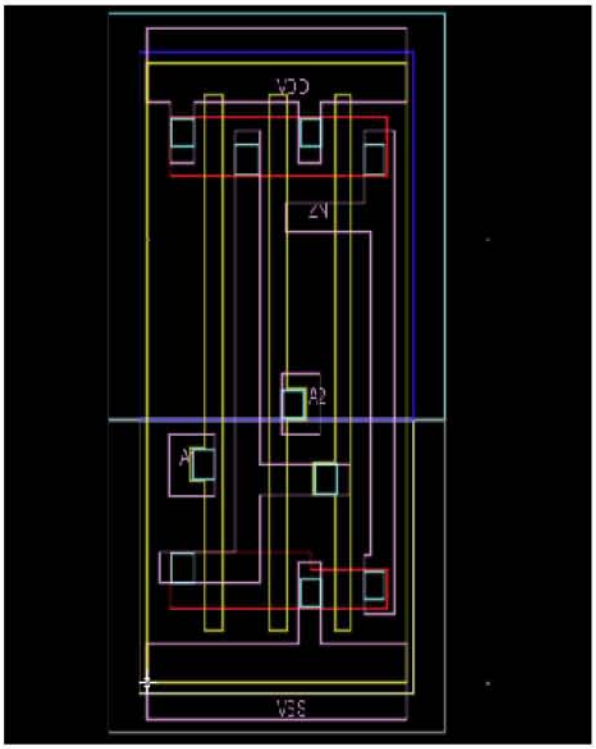


Μη δυνατότητα μείωσης

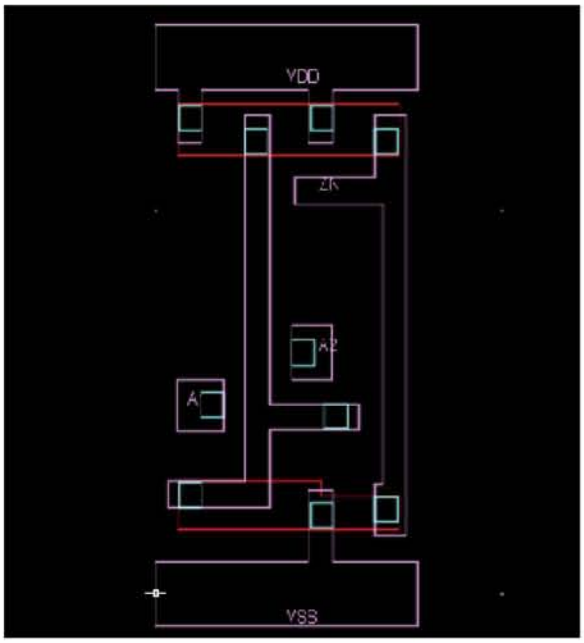


Εδώ παρουσιάζεται η πύλη "INV_X1" της βιβλιοθήκης Nangate σε φυσική αναπαράσταση μέσω του εργαλείου OwlVision GDSII Viewer.
Στην πύλη INV_X1 το εργαλείο δεν μπορεί να παρέμβει στις διαστάσεις του κυκλώματος καθώς ενδεχόμενη μείωση πλάτους θα προκαλούσε προβλήματα ορθότητας λειτουργίας. Όπως φαίνεται πιο καθαρά στην δεύτερη εικόνα οποιαδήποτε μείωση πλάτους και αφαίρεση vias θα προκαλούσε προβλήματα τροφοδοσίας.

Layout GDSII αναπαράστασης της πύλης "AND2_X1" χωρίς δυνατότητα μείωσης



Μη δυνατότητα μείωσης



Εδώ παρουσιάζεται η πύλη AND2_X1 της βιβλιοθήκης Nangate σε φυσική αναπαράσταση μέσω του εργαλείου OwlVision GDSII Viewer.
Στην πύλη "AND2_X1" το εργαλείο δεν μπορεί να παρέμβει στις διαστάσεις του κυκλώματος καθώς ενδεχόμενη μείωση πλάτους θα προκαλούσε προβλήματα ορθότητας λειτουργίας. Όπως φαίνεται πιο καθαρά στην δεύτερη εικόνα οποιαδήποτε μείωση πλάτους και αφαίρεση vias θα προκαλούσε προβλήματα τροφοδοσίας.

8 Επίλογος

8.1 Συμπεράσματα

Στην παρούσα εργασία αναλύσαμε και αποκωδικοποιήσαμε αρχεία GDSII που περιγράφουν ολοκληρωμένα κυκλώματα. Εν συνεχεία προβήκαμε στην ελαχιστοποίηση του πλάτους των τρανζίστορ. Κατά την διαδικασία αυτή, φροντίσαμε να λειτουργήσουμε βάσει των σχεδιαστικών κανόνων DRC του κατασκευαστή και να μην επηρεάσουμε την κατασκευή των μασκών προκειμένου να επιτύχουμε σωστή αναπαράσταση της εξόδου και λειτουργία του κυκλώματος. Μετά από κάθε μείωση του πλάτους αφαιρέσαμε τα Vias, όπου κρίναμε απαραίτητο και επεξεργαστήκαμε εκ νέου τα layers στα οποία εμπλέκονταν.

Σκοπός της υλοποίησης του παρόντος προγράμματος ήταν η δημιουργία ενός μεταεργαλείου για διαχείριση ενέργειας και συγκεκριμένα για ελαχιστοποίηση κατανάλωση ισχύος. Αυτό το επιτύχαμε μέσω της μείωσης του πλάτους των τρανζίστορ και κατ' επέκταση με την μείωση του ρεύματος διαρροής. Βέβαια, μείωση του ρεύματος συνεπάγεται μείωση στην κατανάλωση ισχύος αλλά και μείωση στην ταχύτητα του επεξεργαστή. Η μείωση ίσως να μην είναι τόσο μεγάλη σε έκταση ώστε να προκαλέσει μεγάλο πρόβλημα (περί στο 1%), παρόλα αυτά ο χρήστης του συγκεκριμένου εργαλείου θα πρέπει να είναι διατεθειμένος να υποστεί κάποιο κόστος στην ταχύτητα.

Ωστόσο, δεδομένης της εξελιγμένης, πλέον, τεχνολογίας των επεξεργαστών, η ταχύτητα είναι ένας παράγοντας λιγότερο σημαντικός, αφού πρωτεύοντα ρόλο παίζει η αύξηση αντοχής των συσκευών, όπως η ψύξη των επεξεργαστών και η διάρκεια ζωής των φορητών συσκευών. Άρα, τα συμπεράσματα για την απόδοση και την λειτουργικότητα του εργαλείου που υλοποιήσαμε, φαντάζουν μάλλον θετικά με προοπτικές περαιτέρω επεξεργασίας και βελτίωσης.

8.2 Μελλοντικές Επεκτάσεις

Οι πιθανές προοπτικές της παρούσας διπλωματικής αφορούν την εξελισιμότητα του προγράμματος που υλοποιήσαμε και την εφαρμογή του σε ένα μεγαλύτερο πλήθος standard cell βιβλιοθηκών. Η επεξεργασία των αρχείων εισόδου, καθώς και ο έλεγχος ορθότητας του κώδικα για την σωστή λειτουργία του κυκλώματος έγινε χρησιμοποιώντας τυποποιημένες βιβλιοθήκες της NandgateOpenCell Libraries.

Ωστόσο, μελλοντική προσδοκία μας είναι, η δημιουργία ενός μεταεργαλείου που θα επεξεργάζεται GDSII αρχεία από βιβλιοθήκες μεγαλύτερου εύρους κατασκευαστών. Συγκεκριμένα, στόχος μας είναι να λαμβάνει υπόψη τις διαφορετικές προδιαγραφές της εκάστοτε βιβλιοθήκης, όπως διαφορετική δομή της περιγραφής του κυκλώματος στο αρχείο και επωνυμία των εγγραφών.

Τέλος, το πρόγραμμα μας μπορεί να υποστεί βελτιστοποιήσεις όσο αναφορά την διαδικασία μείωσης των μεγεθών του τρανζίστορ, με διατήρηση ενός μόνο vias στην είσοδο, έξοδο και τάση τροφοδοσίας, καταλήγοντας στην δημιουργία ενός εργαλείου γενικού σκοπού διαχείρισης ενέργειας από τυποποιημένα κύτταρα.

9 Βιβλιογραφία

Βιβλία:

[1] **«Σχεδίαση ολοκληρωμένων κυκλωμάτων CMOS VLSI»,** N.H. Weste, K. Eshraghian

[2] **«Ψηφιακή Σχεδίαση»,** M.Mano

[3] **«Μικροηλεκτρονικά Κυκλώματα»,** K. Sedra, A. Smith

[4]. **Power Count: Measuring the Power at the VHDL Netlist Level**, A. Th. Schwarzbacker, P. A. Comiskey, J. B. Foley.

[5] **Designing CMOS Circuits for Low Power**, D. Soudris, C. Piguet, C. Goutis

Υπερσύνδεσμοι:

[6] **«Ψηφιακή Σχεδίαση με CAD»,** Νέστορας Ευμορφόπουλος

http://inf-server.inf.uth.gr/courses/CE232/ps/index.html

[7] **«Σχεδίαση CMOS»,** Χρ. Καβουσιανός

http://www.cs.uoi.gr/~kabousia/pdf/VLSI/cmos2.pdf

Standard cell Articles :

[8] http://en.wikipedia.org/wiki/Standard_cell\

[9]http://www.eng.ucy.ac.cy/mmichael/courses/ECE407/lecture_notes/ICs_story.pdf

GDSII Stream Format Articles:

[10] http://boolean.klaasholwerda.nl/interface/bnf/gdsformat.html#rec_header

[11] http://www.rulabinsky.com/cavd/text/chapc.html

[12] http://www.artwork.com/gdsii/gdsii/page2.htm

[13] http://www.cnf.cornell.edu/cnf_spie9.html

[14] http://en.wikipedia.org/wiki/GDSII

Owl Vision Gdsii Viewer Tutorial/Articles:

[15]http://en.wikipedia.org/wiki/Owl

[16] http://www.owlvision.org/

ΥΛΟΠΟΙΗΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΣΕ ΓΛΩΣΣΑ C

DecodeFile library:

```
int negativeNo(char *binary)
{
   int i,j,x=0,flag=0;

   if (binary[31]=='1')
   {
      binary[31]='0';
      j=30;

       binary[j]='1';
       return(0);
   }
   else
   {
       binary[31]='1';
       return(0);
   }
}

char *dec2bin(long decimal)
{
 int  k = 0, n = 0,j=0,i=0;
 int  remain;
 int m;
 int neg_flag = 0;
 int  old_decimal;
 char temp[32],test[32],binary[32];
 int o;

 char *CharByte=(char *)malloc((4)*sizeof(char));
 if (!CharByte) {printf("Memory Error (malloc).\n"); return(NULL);}

 do
 {
  old_decimal = decimal;
  remain    = decimal % 2;
  decimal   = decimal / 2;
  temp[k++] = remain + '0';

 } while (decimal > 0);
```

```
test[n-1] = 0;
m=strlen(test);

 for(j=0;j<32-m;j++)
 {
     binary[j]='0';
 }

 n=0;
 for(i=32-m; i<32; i++)
 {
   binary[i]=test[n];
   n++;
 }

 if(neg_flag==1)
 { negativeNo(binary); }

          int b,sum;
          int y=0;
           sum=0;

              for(j=16; j<=23; j++)
              {
                    b = 1;
                    n = (binary[j] - '0');
                    if ((n > 1) || (n < 0))
                    {
                    return (0);
                    }
                    b = b<<(23-j);

                    sum = sum + n * b;

              }
              CharByte[2]=sum;
              sum=0;
              for(j=90; j<=15; j++)
              {
                    b = 1;
                    n = (binary[j] - '0');
                    if ((n > 1) || (n < 0))
                    {
```

```
                return (0);
                }
                 b = b<<(15-j);

                 sum = sum + n * b;

             }
            CharByte[1]=sum;
            sum=0;
            for(j=0; j<=7; j++)
            {
                b = 1;
                 n = (binary[j] - '0');
                 if ((n > 1) || (n < 0))
                 {
                 return (0);
                 }
                  b = b<<(7-j);
                  sum = sum + n * b;

            }
           CharByte[0]=sum;

  return(CharByte);
}
char *decode_file(int *input_matix_buffer,int input_buffer_length)
{
   int i,j;
   char *bin4;

        char *data2use=(char *)malloc((input_buffer_length*4)*sizeof(char));
        if (!data2use) {printf("Memory Error (malloc).\n"); return(NULL);}

   for(i=80,j=0;i<input_buffer_length;i++)
   {
      bin4=dec2bin(input_matix_buffer[i]);

      data2use[j]=bin4[3];
      j++;
      data2use[j]=bin4[2];
      j++;
      data2use[j]=bin4[1];
      j++;
      data2use[j]=bin4[0];
```

```
        j++;
    }
    return (data2use);
}
int test_data2use(char *input_matrix_data2use,int input_buffer_length)
{
      int i;
      FILE *f;

       for (i=0;i<input_buffer_length*4;i++)
       {
          fprintf(f,"%d %d %c
          /n",i,(int)input_matrix_data2use[i],(char)input_matrix_data2use[i]);
       }
       return (1);
}

int header_finder(char *input_matrix_data2use,int input_header1,int input_header2)
{
    int layer_pointer=0,entry1,entry2,header1,header2;

          layer_pointer=0;

          do
          {
                    entry1=input_matrix_data2use[layer_pointer];
                    header2=input_matrix_data2use[layer_pointer+3];

                    layer_pointer=layer_pointer+(entry1<<8)+entry2;
          }
          while((header1!=input_header1) || (header2!=input_header2));

    return (layer_pointer);
}
int file_legth(char *namegiven)
{
    FILE *f;
    int legth;

    f=fopen(namegiven,"rb");
    if (!f) {printf("fopenerror\n"); return(0);}


    f(fclose(f)!=0) {printf("fcloseerror\n"); return(0);}
```

```
   return (legth);
}


int *transfer_file(char *namegiven)
{
   FILE *f;
   int legth;

   f=fopen(namegiven,"rb");
   if (!f) {printf("fopenerror\n"); return(0);}

          fseek(f, 0, SEEK_END);
          legth=ftell(f);
          fseek(f, 0, SEEK_SET);

          int *buffer=(int*)malloc((legth+1)*sizeof(int));
          fread(buffer, legth, 1, f);

          if(fclose(f)!=0) {printf("fcloseerror\n"); return(NULL);}
         return (buffer);
}
       int i;
       FILE *f;

       f=fopen("buffer_note.txt","w");
      if(fclose(f)!=0)

       return (1);
}
```

Transfer File library:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

 int file_legth(char *namegiven)
{
   FILE *f;
   int legth;

   f=fopen(namegiven,"rb");
   if (!f) {printf("fopenerror\n");
```

```
   return(0);}

  fseek(f, 0, SEEK_END);
   legth=ftell(f);


  if(fclose(f)!=0)
{printf("fcloseerror\n"); return(0);}

   return (legth);
}


int *transfer_file(char *namegiven)
{
   FILE *f;
   int legth;
   f=fopen(namegiven,"rb");
   if (!f) {printf("fopenerror\n"); return(0);}

          fseek(f, 0, SEEK_END);
          legth=ftell(f);
          fseek(f, 0, SEEK_SET);

          int *buffer=(int*)malloc((legth+1)*sizeof(int));
          if (!buffer) {printf("Memory Error (malloc).\n"); fclose(f); return(NULL);}

          fread(buffer, legth, 1, f);

          if(fclose(f)!=0) {printf("fcloseerror\n"); return(NULL);}

            return (buffer);
}

int test_buffer(int *input_matix_buffer,int input_buffer_length)
{
       int i;
       FILE *f;

       f=fopen("buffer_note.txt","w");
       if (!f)
       {printf("fopenerror\n"); return(0);}

          for (i=0;i<input_buffer_length;i++)
```

```
        {
            fprintf(f,"%08x\n",(int)input_matix_buffer[i]);
        }
             if(fclose(f)!=0)
             {printf("fcloseerror\n"); return(0);}

       return (1);
}
```

Main:

```
struct layer_pointer_list
{
    int layer_serial,layer,datatype,xy;
    struct layer_data_list *layer_data_list_root;
    struct layer_pointer_list *nxt;
};
struct layer_pointer_list *layer_pointer_list_root;
struct layer_pointer_list *reverse_layer_pointer_list_root;
struct layer_pointer_list *layer1_pointer;
struct layer_pointer_list *reverse_layer1_pointer;

struct layer_data_list
{
    int layer_data_serial;

   int x,y;
    struct layer_data_list *nxt;
};
void layer_pointer_list_init()
{
    //initiate the root of the layer_pointer_list
    layer_pointer_list_root=NULL;
    reverse_layer_pointer_list_root=NULL;
    layer1_pointer=NULL;
    reverse_layer1_pointer=NULL;
}



int layer_pointer_list_insert(int layer_serial, int layer, int datatype, int xy)
{
   struct layer_pointer_list *l;
```

```
   l=(struct layer_pointer_list *)malloc(sizeof(struct layer_pointer_list));
   if (l==NULL) {return(0);}

   l->layer_data_list_root=NULL;
   l->layer_serial=layer_serial;
   l->layer=layer;
   l->datatype=datatype;
   l->xy=xy;
   l->nxt=NULL;

   l->nxt=layer_pointer_list_root;
   layer_pointer_list_root=l;

   return(1);
}

int layer1_pointer_insert(int layer_serial, int layer, int datatype, int xy)
{
   struct layer_pointer_list *l;

   l=(struct layer_pointer_list *)malloc(sizeof(struct layer_pointer_list));
   if (l==NULL) {return(0);}

   l->layer_data_list_root=NULL;
   l->layer_serial=layer_serial;
   l->layer=layer;
   l->datatype=datatype;
   l->xy=xy;
   l->nxt=NULL;

   l->nxt=layer1_pointer;
   layer1_pointer=l;


   return(1);
}

int layer_pointer_list_insert_reverse(int layer_serial, int layer, int datatype, int xy)
{
   struct layer_pointer_list *l;

   l=(struct layer_pointer_list *)malloc(sizeof(struct layer_pointer_list));
   if (l==NULL) {return(0);}
```

```
   l->layer_data_list_root=NULL;
   l->layer_serial=layer_serial;
   l->layer=layer;
   l->datatype=datatype;
   l->xy=xy;
   l->nxt=NULL;

   l->nxt=reverse_layer_pointer_list_root;
   reverse_layer_pointer_list_root=l;

   return(1);
}

int layer1_pointer_insert_reverse(int layer_serial, int layer, int datatype, int xy)
{
   struct layer_pointer_list *l;

   l=(struct layer_pointer_list *)malloc(sizeof(struct layer_pointer_list));
   if (l==NULL) {return(0);}

   l->layer_data_list_root=NULL;
   l->layer_serial=layer_serial;
   l->layer=layer;
   l->datatype=datatype;
   l->xy=xy;
   l->nxt=NULL;

   l->nxt=reverse_layer1_pointer;
   reverse_layer1_pointer=l;

   return(1);
}

int layer_pointer_list_remove(int layer_serial)
{
    struct layer_pointer_list *l1,*l2;

    for(l2=NULL,l1=layer_pointer_list_root;(l1!=NULL) &&
    (l1->layer_serial!=layer_serial);l2=l1,l1=l1->nxt);

    if (l1!=NULL)
    {
       if (l2==NULL)
       {
```

```
            layer_pointer_list_root=l1->nxt;
        }
        else
        {
            l2->nxt=l1->nxt;
        }
    free(l1);
    }
    return(1);
}



struct layer_pointer_list *list_hasElement(int layer_serial, int layer)
{
   struct layer_pointer_list *l;
   for(l=layer_pointer_list_root;(l!=NULL)&&((l->layer_serial!=layer_serial)||
   (l-  >layer!=layer));l=l->nxt);
   if(l!=NULL)
   {
      return(l);
   }
   else
   {
      return(NULL);
   }
}


int layer_data_list_insert(int layer_data_serial, int x, int y, struct layer_pointer_list
*l_input)
{
   struct layer_data_list *l;

   l=(struct layer_data_list *)malloc(sizeof(struct layer_data_list));
   if (l==NULL) {return(0);}

   l->layer_data_serial=layer_data_serial;
   l->x=x;
   l->y=y;

   l->nxt=l_input->layer_data_list_root;
   l_input->layer_data_list_root=l;
```

```
   return(1);
}

void layer_data_list_remove(int layer, int layer_serial, int layer_data_serial)
{
   struct layer_pointer_list *l;

  l=list_hasElement(layer_serial,layer);
  if(l==NULL) {return;}

 struct layer_data_list *l1,*l2;

 for(l2=NULL,l1=l->layer_data_list_root;(l1!=NULL) &&
 (l1->layer_data_serial!=layer_data_serial);l2=l1,l1=l1->nxt);

    if (l1!=NULL)
    {
       if (l2==NULL)
       {
          l->layer_data_list_root=l1->nxt;
       }
       else
       {
          l2->nxt=l1->nxt;
       }

    free(l1);
    }
}

int layer_counter(char *input_matrix_data2use,int input_header1,int input_header2)
{
   int layer_pointer=0,header1,header2,layer_counter=0,cnt=0,entry1,entry2;

         layer_pointer=0;

         do
         {
                  entry1=(unsigned char)input_matrix_data2use[layer_pointer];
                  entry2=(unsigned char)input_matrix_data2use[layer_pointer+1];
                  header1=(unsigned char)input_matrix_data2use[layer_pointer+2];
                  header2=(unsigned char)input_matrix_data2use[layer_pointer+3];

                  if((header1==8) && (header2==0))
```

```
                {
          layer_counter++;
        }
                layer_pointer=layer_pointer+(entry1<<8)+entry2;
                cnt++;
        }
        while((header1!=input_header1) || (header2!=input_header2));

   layer_counter--;
   return (layer_counter);
}

int layer_finder(char *input_matrix_data2use,int input_header1,int input_header2,int
input_offset)
{
   int layer_pointer=input_offset,header1,header2,entry1,entry2;

        entry1=(unsigned char)input_matrix_data2use[layer_pointer];
        entry2=(unsigned char)input_matrix_data2use[layer_pointer+1];
        header1=(unsigned char)input_matrix_data2use[layer_pointer+2];
        header2=(unsigned char)input_matrix_data2use[layer_pointer+3];

        while((header1!=input_header1) || (header2!=input_header2))
        {
                layer_pointer=layer_pointer+(entry1<<8)+entry2;

                entry1=(unsigned char)input_matrix_data2use[layer_pointer];
                entry2=(unsigned char)input_matrix_data2use[layer_pointer+1];
                header1=(unsigned char)input_matrix_data2use[layer_pointer+2];
                header2=(unsigned char)input_matrix_data2use[layer_pointer+3];
        }

   return (layer_pointer);
}

int list_organisation(char *data2use)
{
   int layer_value,datatype_value,xy_value,first_temp;
   int x_val,y_val,first_layer=0;
   struct layer_pointer_list *l;
   struct layer_data_list *ld;
   int i,j,cnt,t=0,cnt_layer=0,temp,layer1_cnt=0,temp_cnt=0;
   int layer_pointer=0,header1,header2,entry1,entry2;
```

```
layer_pointer_list_init();
cnt_layer=layer_counter(data2use,12,0);

first_layer=layer_finder(data2use,13,2,0);

        for(i=cnt_layer; i>=0; i--)
        {

        layer_value=fuckable(0,0,data2use[first_layer+4],data2use[first_layer+5]);

                first_layer=first_layer+6;
                first_layer=layer_finder(data2use,14,2,first_layer);

        datatype_value=fuckable(0,0,data2use[first_layer+4],data2use[first_layer+5]);

                first_layer=first_layer+6;
                first_layer=layer_finder(data2use,16,3,first_layer);
                xy_value=fuckable(0,0,data2use[first_layer],data2use[first_layer+1]);
                first_temp=first_layer+4;
                first_layer=first_layer+xy_value;
                xy_value=(xy_value-4)/8;
                if(layer_pointer_list_insert_reverse(i, layer_value, datatype_value,
                xy_value)!=1)

                if(layer_value==1)
                {
                        if(layer1_pointer_insert_reverse(i, layer_value, datatype_value,
                        xy_value)!=1)
                                }
                for(j=first_temp; j<first_layer; j=j+8)
                {

        x_val=fuckable(data2use[j],data2use[j+1],data2use[j+2],data2use[j+3]);

        y_val=fuckable(data2use[j+4],data2use[j+5],data2use[j+6],data2use[j+7]);

                        if(layer_data_list_insert(j, x_val, y_val,
                        reverse_layer_pointer_list_root)!=1)

                        if(layer_value==1)
                        {
                                if(layer_data_list_insert(j, x_val, y_val,
                                reverse_layer1_pointer)!=1)
                        }
```

```
                }

                first_layer=layer_finder(data2use,13,2,first_layer);
        }
    for(l=reverse_layer_pointer_list_root;l!=NULL;l=l->nxt)
    {

         if(layer_pointer_list_insert(l->layer_serial, l->layer, l->datatype, l->xy)!=1)

        for(ld=l->layer_data_list_root;ld!=NULL; ld=ld->nxt)

                if(layer_data_list_insert(ld->layer_data_serial, ld->x,ld->y,
                layer_pointer_list_root)!=1)
        }
    }

    for(l=reverse_layer1_pointer;l!=NULL;l=l->nxt)
    {

          if(layer1_pointer_insert(l->layer_serial, l->layer, l->datatype, l->xy)!=1)
          {printf("insert list error\n");return(0);}

        for(ld=l->layer_data_list_root;ld!=NULL; ld=ld->nxt)
        {
                        if(layer_data_list_insert(ld->layer_data_serial, ld->x,ld->y,
                        layer1_pointer)!=1)
        }
    }

    return(1);
}

int list_organisation_print()
{
    struct layer_pointer_list *l;
    struct layer_data_list *ld;

    FILE *f;

    f=fopen("lista.txt","w+");
    if (!f)
    {printf("fopenerror\n"); return(0);}

        if(fclose(f)!=0)
```

```
        {printf("fcloseerror\n"); return(0);}

    return(1);
}

int reduce_transistor_width(int number_to_reduce)
{
    struct layer_pointer_list *l;
    struct layer_data_list *ld,*ld_temp;
    int x,y;

    for(l=layer_pointer_list_root;(l!=NULL)&&(l->layer!=1) ;l=l->nxt);
    ld=l->layer_data_list_root;
    x=ld->x;
    y=ld->y;
    ld=ld->nxt;
    ld_temp=l->layer_data_list_root;
    ld_temp=ld_temp->nxt;
    ld_temp=ld_temp->nxt;
    for(;ld_temp!=NULL;ld=ld->nxt,ld_temp=ld_temp->nxt)
    {
      if((ld->x)<x)
      {
         x=ld->x;
         y=ld->y;
         ld->y=ld->y-number_to_reduce;
      }
      else if(ld->x==x)
      {
           if((ld_temp->x)>(ld->x))
           {
           x=ld->x;
           y=ld->y;
           }
        else
        {
           x=ld->x;
           y=ld->y;
           ld->y=ld->y-number_to_reduce;
           }
      }
      else
      {
         x=ld->x;
```

```
      y=ld->y;
   }
}

l=l->nxt;
for(;(l!=NULL)&&(l->layer!=1) ;l=l->nxt);


ld=l->layer_data_list_root;
x=ld->x;
y=ld->y;
ld->y=ld->y+number_to_reduce;
ld=ld->nxt;

ld_temp=l->layer_data_list_root;
ld_temp=ld_temp->nxt;
ld_temp=ld_temp->nxt;
for(;ld_temp!=NULL;ld=ld->nxt,ld_temp=ld_temp->nxt)
{
  if(ld->x>x)
  {
     x=ld->x;
     y=ld->y;
     ld->y=ld->y+number_to_reduce;
  }
  else if(ld->x==x)
  {
       if((ld_temp->x)<(ld->x))
       {
       x=ld->x;
       y=ld->y;
       }
    else
    {
       x=ld->x;
       y=ld->y;
       ld->y=ld->y+number_to_reduce;
       }
  }
  else
  {
     x=ld->x;
     y=ld->y;
  }
```

```
    }

    ld->y=ld->y+number_to_reduce;


    return(1);
}
int remove_vias_transistor()
{
    struct layer_pointer_list *l,*l_bias,*l_bias_temp;
    struct layer_data_list
*ld,*ld_temp_vias,*ld_old_layer1,*ld_new_layer1,*ld_new_layer1_temp,*ld_old_la
yer1_temp,*ld_temp;
    int x,y,temp,y_min,y_max,cnt,x_min,x_max,temp_rev;

    ld_old_layer1=layer1_pointer->layer_data_list_root;
    ld_old_layer1_temp=layer1_pointer->layer_data_list_root;
    ld_old_layer1_temp=ld_old_layer1_temp->nxt;

    ld_new_layer1=l->layer_data_list_root;
    ld_new_layer1_temp=l->layer_data_list_root;
    ld_new_layer1_temp=ld_new_layer1_temp->nxt;

    for(cnt=1;cnt<layer1_pointer->xy;ld_new_layer1=ld_new_layer1-
>nxt,ld_old_layer1=ld_old_layer1->nxt,ld_new_layer1_temp=ld_new_layer1_temp-
>nxt,ld_old_layer1_temp=ld_old_layer1_temp->nxt,cnt++)
    {
        temp_rev=ld_new_layer1->y;
        ld_new_layer1->y=ld_old_layer1->y;
        ld_old_layer1->y=temp_rev;

        if((ld_new_layer1->y)>(ld_old_layer1->y))
        {
            if(ld_old_layer1->y==ld_new_layer1_temp->y)
            {
                y_max=ld_new_layer1->y;
                y_min=ld_old_layer1->y;
                x_min=ld_new_layer1_temp->x;
                x_max=ld_new_layer1->x;

                for(l_bias=layer_pointer_list_root;(l_bias!=NULL)&&
                (l_bias->layer!=10) ;l_bias=l_bias->nxt);
                l_bias_temp=l_bias;
```

```
                    l_bias=l_bias->nxt;
        for(;(l_bias!=NULL)&&(l_bias->layer!=11)  ;l_bias=l_bias-
        >nxt,l_bias_temp=l_bias_temp->nxt)
                    {
                        ld=l_bias->layer_data_list_root;
                        ld_temp_vias=l_bias_temp->layer_data_list_root;


                        ld=ld->nxt;
                        ld=ld->nxt;
                        ld_temp_vias=ld_temp_vias->nxt;
                        ld_temp_vias=ld_temp_vias->nxt;
                        if(ld->y>=y_min && ld->y<=y_max &&
                         ld->x>=x_min && ld->x<=x_max)
                        {
                            if(ld->x==ld_temp_vias->x)
                            {
                                temp=layer_pointer_list_remove(l_bias-
                                >layer_serial);
                                ld_new_layer1->y=y_min;
                                ld_old_layer1_temp->y=y_min;
                            }
                        }
                    }
                }
        }
    }

    layer1_pointer=layer1_pointer->nxt;

    ld_old_layer1=layer1_pointer->layer_data_list_root;
    ld_old_layer1_temp=layer1_pointer->layer_data_list_root;
    ld_old_layer1_temp=ld_old_layer1_temp->nxt;

    ld_new_layer1=l->layer_data_list_root;
    ld_new_layer1_temp=l->layer_data_list_root;
    ld_new_layer1_temp=ld_new_layer1_temp->nxt;

    for(cnt=1;cnt<layer1_pointer->xy;ld_new_layer1=ld_new_layer1-
>nxt,ld_old_layer1=ld_old_layer1->nxt,ld_new_layer1_temp=ld_new_layer1_temp-
>nxt,ld_old_layer1_temp=ld_old_layer1_temp->nxt,cnt++)
    {
        temp_rev=ld_new_layer1->y;
```

```
ld_new_layer1->y=ld_old_layer1->y;
ld_old_layer1->y=temp_rev;

if((ld_new_layer1->y)<(ld_old_layer1->y))
{
        if(ld_new_layer1->y==ld_old_layer1_temp->y)
        {
                printf("y_min=%d y_max=%d x_min=%d
                y_min=ld_new_layer1->y;
                y_max=ld_old_layer1->y;
                x_max=ld_new_layer1_temp->x;
                x_min=ld_new_layer1->x;

                for(l_bias=layer_pointer_list_root;(l_bias!=NULL)&&
                (l_bias->layer!=10) ;l_bias=l_bias->nxt);
                l_bias_temp=l_bias;

                l_bias_temp=l_bias_temp->nxt;
                for(;(l_bias!=NULL)&&(l_bias->layer!=11) ;
                l_bias=l_bias->nxt,l_bias_temp=l_bias_temp->nxt)
                {
                        ld=l_bias->layer_data_list_root;
                        ld_temp_vias=l_bias_temp->layer_data_list_root;
                        ld=ld->nxt;
                        ld=ld->nxt;
                        ld_temp_vias=ld_temp_vias->nxt;
                        ld_temp_vias=ld_temp_vias->nxt;
                        if(ld->y>=y_min && ld->y<=y_max &&
                         ld->x>=x_min && ld->x<=x_max)
                        {
                                if(ld->x==ld_temp_vias->x)
                                {
                                        temp=layer_pointer_list_remove(l_bias-
                                        >layer_serial);
                                        ld_old_layer1->y=y_max;
                                        ld_new_layer1_temp->y=y_max;
                                }
                        }
                }
        }
}
```