ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΤΗΛΕΠΙΚΟΙΚΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΑΝΑΛΥΣΗ ΧΡΟΝΙΣΜΟΥ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

Διπλωματική Εργασία :

Μήττας Λαζαρίδης Αλέξανδρος

Σεπτέμβριος 2011

Βόλος

Ευχαριστίες

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

Επιπρόσθετα, θα ήθελα να ευχαριστήσω τον έταιρο επιβλέποντα καθηγητή, κ. Παναγιώτη Μποζάνη για τις συμβουλές και τον διδακτορικό φοιτητή του τμήματος Αντώνη Δαδαλιάρη για την προγενέστερη δουλειά του στον τομέα.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου που μου συμπαραστάθηκαν σε όλη την διάρκεια της εκπόνησης αυτής της εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 1. ΕΙΣΑΓΩΓΗ

1.1 Στόχος της εργασίας

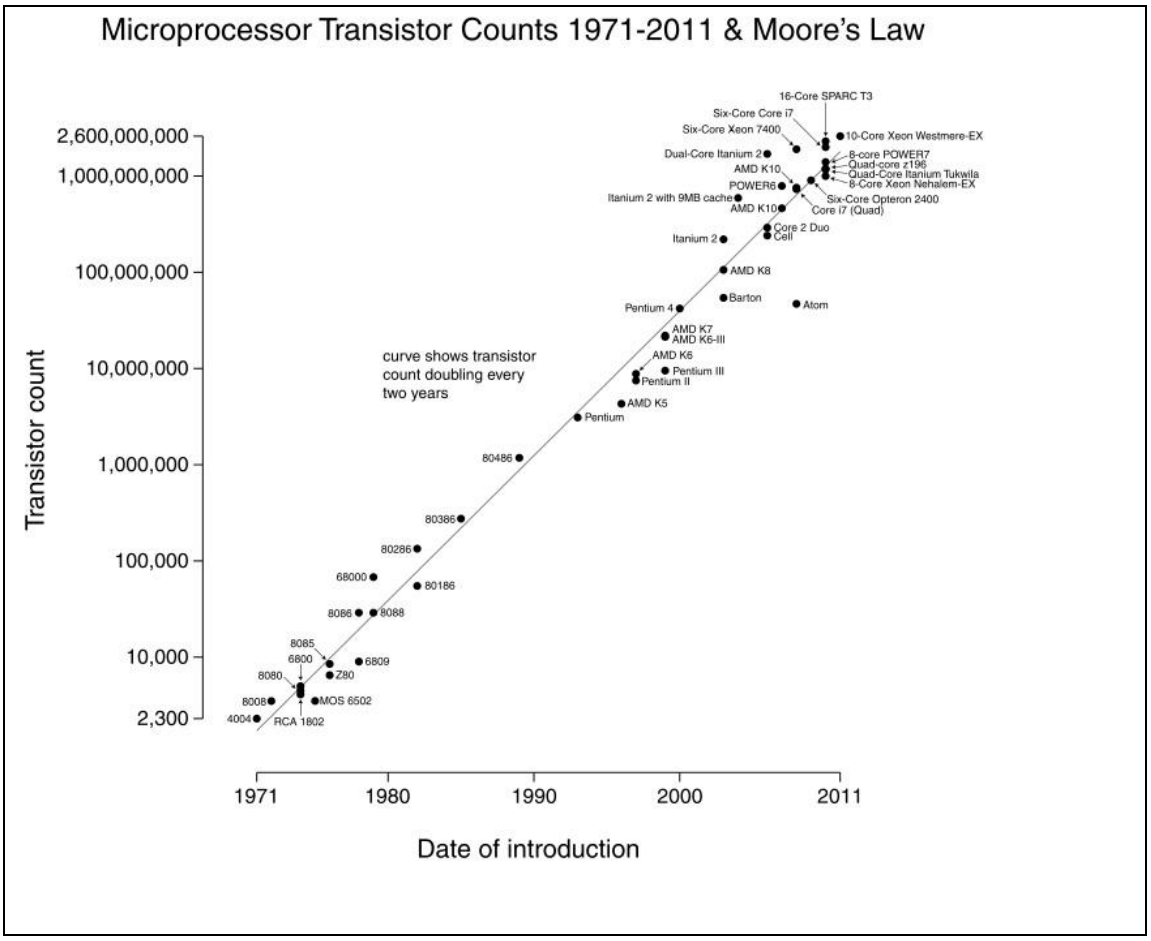
Δεδομένου ενός κυκλώματος αποτελούμενου από λογικές πύλες, **flip-flops**, πολυπλέκτες και λοιπά στοιχεία, καλούμαστε να υπολογίσουμε το κρίσιμο μονοπάτι (**critical path**), δηλαδή το μονοπάτι με τη μέγιστη καθυστέρηση (το χρόνο που απαιτείται για να φτάσει ένα σήμα εισόδου στην έξοδο προκειμένου να παρατηρήσουμε τα αποτελέσματα του. Υπάρχουν διάφορες τεχνικές για την εύρεση του εν λόγω μονοπατιού (λιγότερο ή περισσότερο ακριβείς), το οποίο μπορεί να επηρεάζεται από διάφορους παράγοντες τους οποίους θα εξηγήσουμε αναλυτικά παρακάτω.

1.2 Νόμος του Moore

Ο **νόμος του Moore** περιγράφει μια μακροχρόνια τάση στην ιστορία του υλικού υπολογιστών (**hardware**). Ο αριθμός των **transistor** που μπορούν να τοποθετηθούν σε ένα ολοκληρωμένο κύκλωμα, διπλασιάζεται σχεδόν κάθε δύο χρόνια. Αυτή η τάση επικρατεί περίπου 50 χρόνια τώρα και αναμένεται να συνεχιστεί μέχρι το 2015 – 2020.

Τα τεχνολογικά χαρακτηριστικά πολλών ψηφιακών ηλεκτρονικών συσκευών, όπως για παράδειγμα η ταχύτητα ενός επεξεργαστή, η χωρητικότητα της μνήμης ή ακόμη και ο αριθμός και το μέγεθος των **pixels** στις ψηφιακές φωτογραφικές μηχανές συνδέονται στενά με το **νόμο του Moore**. Ως αποτέλεσμα όλα αυτά αυξάνονται δραματικά (με εκθετικούς ρυθμούς) και ως εκ τούτου ενισχύεται το αντίκτυπο των ψηφιακών ηλεκτρονικών συστημάτων σε κάθε τομέα της παγκοσμίας οικονομίας.

Η αρχική πρόβλεψη του Moore το 1965 μιλούσε για τουλάχιστον 10ετή συνέχιση αυτής της τάσης. Σήμερα ο νόμος του χρησιμοποιείται στη βιομηχανία ως οδηγός μακρόπνοων προγραμμάτων και θέτει στόχους για έρευνα και ανάπτυξη.



Εικόνα 1.1 Νόμος του Moore

Απόρια λοιπόν του παραπάνω νόμου, είναι κυκλώματα με ακόμη περισσότερα transistor – λογικές πύλες. Έτσι βασικός σκοπός ενός σχεδιαστή ολοκληρωμένων κυκλωμάτων σήμερα είναι να βρει τη χρυσή τομή ανάμεσα σε ταχύτητα και επιφάνια που καταλαμβάνει το ολοκληρωμένο κύκλωμα καθώς το μέγεθος του transistor όλο και μειώνεται.

Η παρούσα εργασία, λοιπόν, συνδέεται στενά με τις παραπάνω εξελίξεις στον τομέα ηλεκτρονικών συστημάτων, καθώς πραγματεύεται τον υπολογισμό της μέγιστης καθυστέρησης σε ένα ολοκληρωμένο κύκλωμα.

Κεφάλαιο 2. ΑΝΑΛΥΣΗ ΧΡΟΝΙΣΜΟΥ (TIMING ANALYSIS)

2.1 Τι είναι Ανάλυση Χρονισμού

Ανάλυση χρονισμού είναι αναπόσπαστο κομμάτι της σχεδίασης συστημάτων ASIC/VLSI. Είναι μία μέθοδος για τον υπολογισμό του χρονισμού ενός ολοκληρωμένου κυκλώματος.

2.2 Είδη Ανάλυσης Χρονισμού

Η ανάλυση χρονισμού διαφοροποιείται σε τρία κύρια είδη και κάποια άλλα δευτερεύοντα:

1. **Στατική Ανάλυση Χρονισμού (Static Timing Analysis)**
2. **Δυναμική Ανάλυση Χρονισμού (Dynamic Timing Analysis)**
3. **Στατιστική Ανάλυση Χρονισμού (Statistical Timing Analysis)**

Από τα δευτερεύοντα είδη ανάλυσης χρονισμού το σημαντικότερο είναι:

- ο **Υβριδικός προσδιορισμός χρονισμού** (**hybrid timing verification):** ο οποίος συνδιάζει επιλεκτικά στοιχεία της στατικής και της δυναμικής ανάλυσης, σε μία προσπάθεια δημιουργίας του καλύτερου αποτελέσματος επί των δυο αυτών διαφορετικών προσεγγίσεων.

2.3 Στατική Ανάλυση Χρονισμού (Static Timing Analysis)

Η στατική ανάλυση χρονισμού είναι λοιπόν μία από τις διαθέσιμες τεχνικές για να προσδιορίσουμε το χρονισμού ενός ολοκληρωμένου κυκλώματος. Μία εναλλακτική τεχνική προσδιορισμού του χρόνου θα ήταν να προσομοιώσουμε το χρόνο (**timing simulation**). Ο όρος ανάλυση χρονισμού χρησιμοποιείται για οποιαδήποτε από τις δύο παραπάνω τεχνικές. Γενικότερα λοιπόν ο όρος ανάλυση χρονισμού μπορούμε να πούμε ότι αναφέρεται στην ανάλυση ενός ολοκληρωμένου κυκλώματος για την εξαγωγή των χρονισμών αυτού.

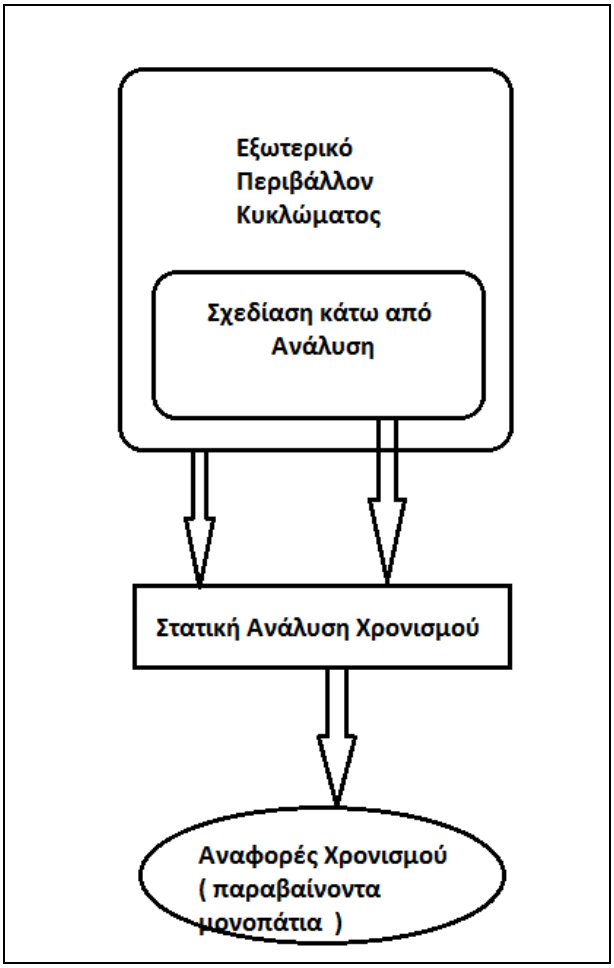
Ο προσδιορισμός στατική σημαίνει ότι η ανάλυση χρονισμού γίνεται στατικά και δεν εξαρτάται από τις τιμές που επεξεργάζονται τα λογικά στοιχεία στις εισόδους τους. Αυτό έρχεται σε αντίθεση με την τεχνική της προσομοίωσης του χρόνου, όπου μας απασχολούν οι τιμές των σημάτων εισόδου του κυκλώματος και προσδιορίζουμε τα αποτελέσματα στην έξοδο, για διάφορες τιμές στις εισόδους.

Δεδομένων λοιπόν κάποιων προδιαγραφών για το κύκλωμα μας, η στατική ανάλυση χρονισμού προσδιορίζει αν η συγκεκριμένη υλοποίηση αυτού, πληρεί αυτές τις προδιαγραφές χροσισμού. Δηλαδή προσδιορίζει αν το κύκλωμα μας λειτουργεί ομαλά, σε μία προκαθορισμένη συχνότητα παράγοντας ορθά αποτελέσματα. Στην εικόνα 2.1 φαίνεται η βασική λειτουργικότητα της στατικής ανάλυσης χρονισμού.

Για να επιτευχθουν τα παραπάνω πραγματοποιούμε μία σειρά από ελέγχους χρονισμού (**timing checks)** όπως για παράδειγμα έλεγχοι εγκατάστασης (**setup checks**) και έλεγχοι συγκράτησης σήματος (**hold checks**). Ένας έλεγχος εγκατάστασης επιβεβαιώνει ότι ένα σήμα θα φτάσει σε ένα **flip-flop**, εντός της περιόδου του ρολογιού. Από την άλλη, ένας έλεγχος συγκράτησης, επιβεβαιώνει ότι τα δεδομένα συγκρατούνται για τουλάχιστον ένα ελάχιστο χρονικό διάστημα, ούτως ώστε να μην υπάρχει μη προβλεπόμενη ροή δεδομένων από ένα **flip-flop**. Δηλαδή το **flip-flop** συλλέγει και επεξεργάζεται τα σωστά δεδομένα. Οι έλεγχοι αυτοί λοιπόν επαληθεύουν ότι τα σωστά δεδομένα επεξεργάζονται και τα σωστά αποτελέσματα μεταβιβάζονται για επεξεργασία στα επόμενα λογικά στοιχεία του κυκλώματος.

Το πιο σημαντικό κομμάτι στην στατική ανάλυση χρονισμού, είναι ότι ολοκληρο το κύκλωμα αναλύεται μόλις μία φορά και οι απαιτούμενοι έλεγχοι χρονισμού πραγματοποιούνται για όλα τα πιθανά μονοπάτια. Επομένως η στατική ανάλυση χρονισμού αποτελεί μία ολοκληρωμένη και εξοντωτική μέθοδο προσδιορισμού του χρονισμού ενός κυκλώματος.

Η «σχεδίαση κάτω από ανάλυση» πραγματοποιείται χρησιμοποιώντας μία γλώσσα περιγραφής υλικού όπως **VHDL** ή **Verilog**.



Εικόνα 2.1 Σχηματική Αναπαράσταση Στατικής Ανάλυσης Χρονισμού

2.4 Στατική Vs Δυναμική Ανάλυση χρονισμού

- Η δυναμική ανάλυση χρονισμού (**Δ.Α.Χ.**) προσδιορίζει τη λειτουργικότητα ενός ολοκληρωμένου κυκλώματος, εφαρμόζοντας διανύσματα εισόδου (**input vectors**) στις εισόδους αυτού και ελέγχοντας τα αντίστοιχα διανύσματα εξόδου (**output vectors**) στις αντίστοιχες εξόδους.

- Η στατική ανάλυση χρονισμού (**Σ.Α.Χ.**) υπολογίζει στατικά την καθυστέρηση και τη συγκρίνει με τις προδιαγραφές του αντίστοιχου κυκλώματος χωρίς καθόλου χρήση παρόμοιων διανυσμάτων

Η Δ.Α.Χ. πρέπει να έχει ολοκληρωθεί και η λειτουργικότητα του κυκλώματος πρέπει να έχει καθοριστεί πλήρως πριν αυτό υποστεί Σ.Α.Χ. Δεν αποτελούν λοιπόν δύο εναλλακτικές λύσεις μεταξύ τους. Η ποιότητα της Δ.Α.Χ. αυξάνει με την αύξηση των διανυσμάτων εισόδου. Η αύξηση των διανυσμάτων εισόδου έχει ως αποτέλεσμα την αύξηση του χρόνου προσομοίωσης. Η Δ.Α.Χ. μπορεί να χρησιμοποιηθεί τόσο για σύγχρονα όσο και για ασύγχρονα σχέδια σε αντίθεση με τη Σ.Α.Χ. που δεν μπορεί να σε ασύγχρονα σχέδια.

Παράδειγματα εργαλείων Δ.Α.Χ. αποτελούν το **Modelsim** της **Mentor Graphics** και το **VCS** της **Synopsys**. Δ.Α.Χ. εφαρμόζεται επίσης κατά τη μεταδιάταξη του κυκλώματος για να διαπιστωθεί ότι η λειτουργικότητα αυτού δεν έχει αλλάξει. Τα διανύσματα ελέγχου προφανώς παραμένουν τα ίδια.

2.5 Γιατί επιλέξαμε Στατική Ανάλυση Χρονισμού

Η δυναμική ανάλυση χρονισμού είναι τόσο εξοντωτική όσο και τα διανύσματα ελέγχου που χρησιμοποιούνται. Για να προσδιορίσουμε όλες τις πιθανκες καταστάσεις σε ένα κύκλωμα 10-100 εκατομυρίων πυλών, το πρόγραμμα μας θα γίνει πάρα πολύ αργό και πάλι μπορεί να μην προσδιοριστούν όλες. Από την άλλη η στατική ανάλυση χρονισμού παρέχει ένα πιο εύκολο και γρήγορο τρόπο για να ελέγξουμε και να αναλύσουμε τους χρονισμούς όλων των μονοπατιών σε ένα ολοκληρωμένο κύκλωμα.

Ακολουθούν ορισμένα πλεονεκτήματα της στατικής ανάλυσης χρονισμού:

- Όλα τα μονοπάτια λαμβάνονται υπόψη για την ανάλυση χονισμού
- Οι χρόνοι ανάλυσης είναι σχετικά μικροί, συγκρινόμενοι με τους χρόνους προσομοίωσης του κυκλώματος
- Μπορεί να γίναι ανάλυση χρονισμού για χειρότερη και καλύτερη περίπτωση ταυτόχρονα
- Αποτελεί μία απαισιόδοξη προσέγγιση του προβλήματος και γιαυτό υπολογίζει τη μέγιστη καθυστέρηση του κυκλώματος. Εφόσον η Δ.Α.Χ. πραγματοποιεί πλήρη προσομοίωση το μεγαλύτερο μειονέκτημα της είναι η υπολογιστική πολυπλοκότητα της εύρεσης των διανυσμάτων εισόδου που παράγουν τη μέγιστη καθυστέρηση στην έξοδο.

Κεφάλαιο 3. ΜΕΘΟΔΟΛΟΓΙΑ

3.1 Γενικά

3.1.1 Περιληπτική περιγραφή του προβλήματος

Για να πραγματοποιήσουμε το ζητούμενο της εν λόγω εργασίας, χρειαζόμαστε κάποιο ολοκληρωμένο κύκλωμα, ή καλύτερα την περιγραφή αυτού, το οποίο θα δώσουμε ως είσοδο σε κάποιο κώδικα που θα συγγράψουμε προκειμένου να υπολογίσουμε τις ζητούμενες ποσότητες.

Χρησιμοποιήσαμε ,λοιπόν, σαν είσοδο τα ακόλουθα:

1. Βασιστήκαμε σε κάποια ολοκληρωμένα κυκλώματα γραμμένα σε γλώσσα περιγραφής υλικού (**VHDL**). Φυσικά η δουλειά μας μπορεί να δώσει αποτελέσματα για οποιαδήποτε κύκλωμα-αρχείο σε VHDL, ή να επεκταθεί ούτως ώστε να πραγματοποιεί τους ίδιους υπολογισμούς για οποιοδήποτε αρχείο κάποιας άλλης γλώσσας περιγραφής υλικού (π.χ. **Verilog**). Τα κυκλώματα στα οποία στηρίχτηκε η μελέτη μας ανήκουν στα **ISCAS Benchmark Circuits '89** και ποικίλουν από απλά κυκλώματα 20 πυλών, μέχρι αρκετά πιο σύνθετα που αποτελούνται από περίπου 10.000 στοιχεία.

2. Χρησιμοποιήσαμε μια **Standard Cell Library**, δηλαδή μια βιβλιοθήκη που περιέχει δεδομένα χρονισμού για κάθε πύλη που μπορεί να συμμετέχει σε ένα από τα ολοκληρωμένα κυκλώματα που περιγράψαμε παραπάνω, τα οποία ανακτούσαμε κάθε φορά με διαφορετικά κριτήρια, τα οποία θα παρουσιαστούν λίγες σειρές παρακάτω.

Αρχικά, λοιπόν, διαβάζαμε τους 2 παραπάνω τύπους αρχείων και αποθηκεύαμε τα δεδομένα που θα χρειαστούν στη συνέχεια σε κατάλληλες δομές δεδομένων για μεταγενέστερη χρήση.

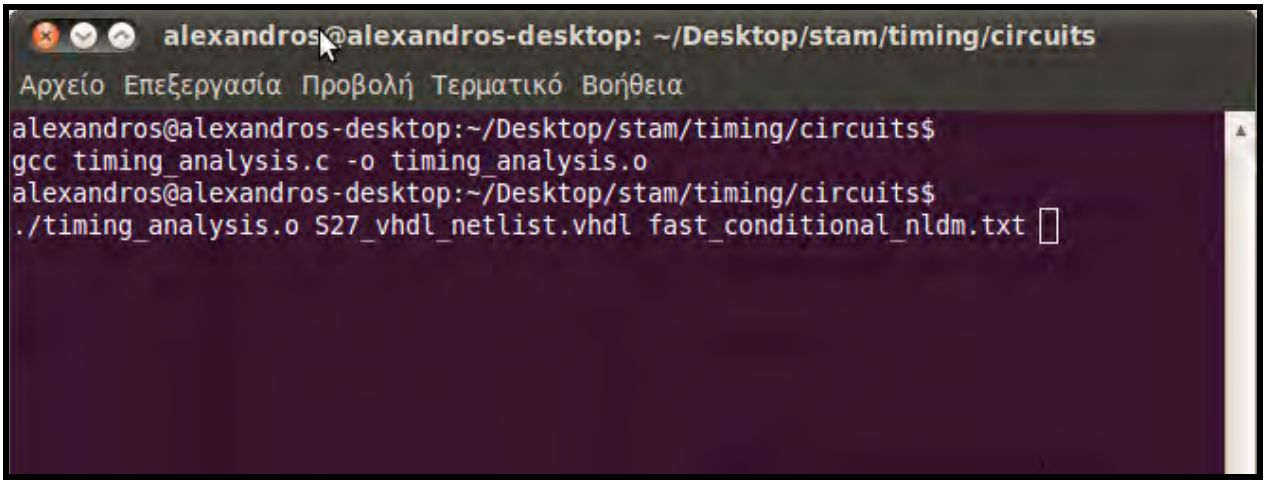
Εν συνεχεία, στο καθαρά υπολογιστικό κομμάτι της εργασίας, αφού είχαμε καθορίσει πλήρως τη συνδεσμολογία του κυκλώματος εισόδου, χωρίσαμε το κύκλωμα σε επίπεδα, καθορίσαμε τη χωρητικότητα, έπειτα την καθυστέρηση κάθε πύλης και τέλος βρήκαμε το κρίσιμο μονοπάτι.

3.1.2 Γλώσσα και περιβάλλον συγγραφής κώδικα

Πρίν παρουσιάσουμε πιο αναλυτικά τη διαδικασία που ακολουθήθηκε να σημειώσουμε πως το πρόγραμμα μας είναι γραμμένο σε γλώσσα προγραμματισμού **C**. Η επιλογή αυτή κάθε άλλο παρά τυχαία ήταν καθώς η συγκεκριμένη γλώσσα παρουσιάζει μια πληθώρα πλεονεκτημάτων που δεν εμφανίζουν άλλες γλώσσες προγραμματισμού.

Για παράδειγμα ένα πρόγραμμα σε **C** είναι ένα ελαφρύ πρόγραμμα χωρίς υπερβολικά μεγάλες αποθηκευτικές απαιτήσεις που μεταγλωττίζεται και τρέχει γρήγορα. Δεν είναι τυχαίο ότι πολλές βιομηχανικές εφαρμογές ακόμη και ολόκληρα λειτουργικα συστήματα προγραμματίζονται σε **C**. Επιπλέον επιδεικνύει τεράστια μεταφερσιμότητα και ο κώδικας μας μπορεί να εκτελεστεί και να παράγει αποτελέσματα σε οποιοδήποτε μηχάνημα. Απαιτείται απλώς η μεταγλώττιση και η εκτέλεση του κώδικα μας σε μία γραμμή εντολών. Τέλος αν και υψηλού επιπέδου γλώσσα η **C** είναι αρκετά κοντά στη γλώσσα του υπολογιστή, καθώς κάνει διαθέσιμο ένα σημαντικό εργαλείο στον προγραμματιστή , την άμεση διαχείριση μνήμης. Η **C** λοιπόν παρέχει δείκτες (**pointers**) στη μνήμη στον προγραμματιστή για να καθορίσει ο ίδιος πως θα την διαχειριστεί, να δεσμεύσει ή να απελευθερώσει **Bytes** και να αποθηκεύσει οποιαδήποτε τιμή σε οποιαδήποτε προσβάσιμη σε αυτόν θέση μνήμης επιθυμεί.

Ο κώδικας μας γράφτηκε σε **UNIX** και μεταγλωττίστηκε και εκτελέστηκε με τις ακόλουθες δύο εντολές.



Εικόνα 3.1 Μεταγλώττιση και εκτέλεση του κώδικα

Όπου timing_analysis.c, ο κώδικας που υλοποιήσαμε για τον υπολογισμό του ζητούμενου της παρούσας εργασίας και timing_analysis.o το αντίστοιχο εκτελέσιμο.

Για την εκτέλεση του κώδικα το όνομα του εκτελέσιμου ακολουθούν ένα λογικό κύκλωμα σε **VHDL** (το οποίο έχει υποστεί synthesis με χρήση του **Design Compiler** της **Synopsys**) ως αρχείο εισόδου συγκεκριμένα το (S27_vhdl_netlist.vhdl) και η **Standard Cell Library** fast_conditional_nldm.txt, από την οποία υπολογίζαμε την καθυστέρηση της εκάστοτε πύλης.

3.2 Ανάγνωση αρχείων εισόδου

3.2.1 Ανάγνωση ολοκληρωμένου κυκλώματος

Ένα λογικό κύκλωμα θα αποτελείται από εισόδους (**primary inputs**), εξόδους (**outputs**), λογικές πύλες (**components**) και ενδιάμεσους συνδέσμους (**signals**) μεταξύ αυτών των πυλών. Οι ακολουθες εντολές VHDL λοιπόν

```
port( S1, S2, S3, S4, S6, S8, S10 : in std_logic;
      S7, S11, S5, S9_out         : out std_logic);
```

Εικόνα 3.2 inputs και outputs κυκλώματος

μας πληροφορούσαν πως το συγκεκριμένο λογικό κύκλωμα αποτελούνταν από τις εισόδους

`S1, S2,..., S10` και τις εξόδους `S7, S11, S5, S9_out` ,

ενώ η ακόλουθη εντολή μας γνωστοποιούσε τα ονόματα των συνδέσμων-καλωδίων μεταξύ των λογικών στοιχείων.

```
signal net285, net286, net292, net305, net291, net282,
       net281, n6, n7, n8, n9, n10, n11, n12, n13 : std_logic;
```

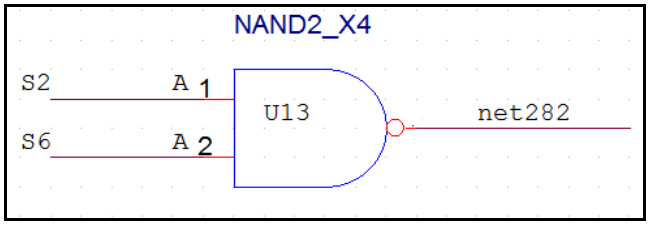
Εικόνα 3.3 signals κυκλώματος

Τέλος κατά την ανάγνωση μιας εντολής όπως

```
U13 : NAND2_X4 port map( A1 => S6, A2 => S2, ZN => net282);
```

Εικόνα 3.4 παράδειγμα σύνταξης εντολής *port map( )*

Καθορίσαμε πλήρως τη συνδεσμολογία μιας πύλης τον τύπο της, το όνομα της και την έξοδο της. Η εν λόγω πύλη ονομαζόταν U13, ήταν τύπου NAND2_X4 , δέχονταν στις εισόδους A1 και A2 τα primary inputs S6 και S2 αντίστοιχα και έξοδο της αποτελούσε το καλώδιο net282, το οποίο με τη σειρά του αποτελούσε είσοδο σε μία ή περισσότερες πύλες.



Εικόνα 3.5 Σχηματική αναπαράσταση δεδομένων μιας εντολής *port map( )*

Ορίσαμε λοιπόν κατάλληλες δομές δεδομένων για να αποθηκεύσουμε τις προαναφερθείσες πληροφορίες. Ενδεικτικά λοιπόν παραθέτουμε τις δομές δεδομένων που ορίσαμε για ένα **primary input** και για μια πύλη του κυκλώματος.

```
struct primary_inputs
{
    char *pr_in_id; //name of primary_input
    struct component *inp_in_comp[50]; //pointers to components
    int num_inp;   //number of gates in which is input
    struct primary_inputs *next;
};
```

Εικόνα 3.6 Υλοποίηση ενός primary_input

```
struct component
{
     char *id;
     char *type;
     int level;
     int num_inp;

     char *sig_pins[6];     // PINS
     char *pr_in_pins[6];   // PINS

     double cout;

     //--------------------------------//
     /*         timing fields          */
     //--------------------------------//
     struct cf_cr_pins *pin_r_f[6];    //
     double delay[6];                  //
     double transition[6];             //
                                       //
     double gate_delay;                //
     double gate_transition;           //
                                       //
     double total_delay;               //
     /* ////////////////////////////  */

     //-------------------------------------------------------------//
     /*       pointers and data to inputs and output                */
     //-------------------------------------------------------------//
     struct primary_inputs *pr_in_ptr[6];                           //
     int num_of_pr_in_ptrs;                                         //
                                                                    //
     struct signals *sig_ptr[6];                                    //
     int num_of_sig_ptrs;                                           //
                                                                    //
     struct outputs *output_ptr;                                    //
     struct signals *out_sig_ptr;                                   //
     char type_of_output; // 'n' for signal, 'o' for output         //
     //-------------------------------------------------------------//

     struct component *next;
};
```

Εικόνα 3.7 Υλοποίηση ενός λογικού στοιχείου

Ορίσαμε πεδία όπως

- **id** : μοναδικός κωδικός που χαρακτηρίζει την εκάστοτε πύλη
- **type :** τύπος της πύλης
- **level :** επίπεδο στο οποίο συγκαταλέγεται η εν λόγω πύλη
- **num_inp :** πλήθος εισόδων στην εν λόγω πύλη
- **cout :** εξωτερική χωρητικότητα της πύλης
- **gate_delay :** καθυστέρηση της πύλης
- **gate_transition :** καθυστέρηση μετάβασης της πύλης

- **total_delay :** συνολική καθυστέρηση κατά μήκος του μονοπατιού από την είσοδο έως και την εν λόγω πύλη
- **num_of_pr_in_ptrs** και **num_of_sig_ptrs :** πλήθος εισόδων στην πύλη τύπου **primary_input** και τύπου **signal** αντίστοιχα
- ***pr_in_ptr[6] , *sig_ptr[6] :** πίνακες από δείκτες στις θέσης μνήμης όπου αποθηκεύονται τα δεδόμενα των **primary inputs** και **signals**, που αποτελούν εισόδους για την εν λόγω πύλη αντίστοιχα. Οι πίνακες έχουν μέγεθος 6 που ισούται με το μέγιστο αριθμό εισόδων για όλες τις πύλες των κυκλωμάτων μας.
- **out_ptr :** δείκτης στη θέση μνήμης όπου αποθηκεύονται τα δεδομένα της εξόδου της πύλης, όταν αυτή αποτελεί έξοδο του κυκλώματος.
- **out_sig_ptr :** δείκτης στη θέση μνήμης όπου αποθηκεύονται τα δεδόμενα της εξόδου της πύλης, όταν αυτή αποτελεί είσοδο σε κάποια άλλη πύλη.
- **type_of_output :** προσδιορισμος του τύπου εξόδου της πύλης (**output** ή **signal**)
- ***next :** δείκτης στην επόμενη πύλη για την οργάνωση όλων των πυλών σε μία κατάλληλη δομή δεδομένων (στην υλοποίηση μας μια λίστα) .
- **Λοιπά πεδία :** παρατηρείται ο ορισμός αρκετών ακόμη πεδίων που αποτελούν κυρίως βοηθητικά για αυτό και δε χρίζουν επιπλέον ανάλυσης.

3.2.2 Ανάγνωση μιας Standard Cell Library

Standard Cell Library

Μία **Standard Cell Library** είναι μία συλλογή χαμηλού επιπέδου λογικών συναρτήσεων όπως **AND**, **OR**, **INVERTERS**, **flip-flops**, **latches**, και **buffers**. Σημαντικό πλεονέκτημα αποτελεί ότι αυτά τα cells είναι προκαθορισμένου ύψους, κάτι που επιτρέπει την τοποθέτηση τους σε σειρές, κάνοντας ευκολότερη τη διαδικασία της αυτόματης διάταξης αυτών. Μία τυπική **Standard Cell Library** αποτελείται κυρίως από:

1) Βάση Δεδιμένων Βιβλιοθήκης: Η οποία περιέχει διαφόρων ειδών διατάξεις σχήματα ή και προσομοιώσεις. Οι πληροφορίες αυτές μπορούν να εξαχθούν σε διάφορες μορφές ώστε να περιέχουν ουσιώδεις πληροφορίες σχετικά με τη διάταξη των **cells**, ώστε να χρησιμοποιηθούν σε εργαλεία τοποθέτησης λογικών στοιχείων (και root tools ???)

2) Αφηρημένους χρονισμούς όπου παρέχονται ορισμοί συναρτήσεων και πληροφορίες χρονισμού, ισχύος και θορύβου για κάθε πύλη.

Η βιβλιοθήκη που επεξεργαστήκαμε περιείχε μεταξύ άλλων δεδομένα χρονισμού για κάθε πύλη που θα μπορούσαμε να συναντήσουμε στο εκάστοτε ολοκληρωμένο κύκλωμα που επεξεργαζόμαστε.

NanGate 45nm Open Cell Library

Για την πραγματοποίηση της εργασίας αυτής χησιμοποιήσαμε μία open-source βιβλιοθήκη, την **NanGate 45nm Open Cell Library**. Η παραπάνω βιβλιοθήκη είναι ειδικά κατασκευασμένη για έλεγχο (**testing**) ολοκληρωμένων κυκλωμάτων.

Η **NanGate** ανέπτυξε και χάρισε αυτή τη βιβλιοθήκη στην **Si2.org** για ελεύθερη χρήση. Η βιβλιοθήκη αυτή βοηθάει στην έρευνα ακαδημαικών προγραμμάτων και οργανισμών όπως η **Si2**, στο σχεδιασμό κυκλωμάτων και στην υλοποιήση νέων αλγορίθμων επεξεργασίας αυτών. Στην πρώτη της έκδοση περιείχε 38 διαφορετικές συναρτήσεις, από **buffers** μέχρι **flip-flops** και αν αναλογιστούμε τις ποικίλες δυνατότητες οδήγησης αυτών των στοιχείων, συνολικά περιείχε πάνω από 100 διαφορετικά **Standard Cells**.

Η βιβλιοθήκη δημιουργήθηκε χρησιμοποιώντας τον **Nangate's Library Creator** και των 45nm FreePDK Kit από το πανεπιτήμιο της Β. Καρολίνα και χαρακτηρίστηκε με το NanSpice της

NanGate χρησιμοποιώντας το μοντέλο προγνωστικής τεχνολογίας (**Predictive Technology Model**) από το πανεπιστήμιο της Αριζόνα. Η **NanGate** ανακοίνωσε την κυκλοφορία της, στις 3 Μαρτίου 2008.

Η εν λόγω βιβλιοθήκη βελτιώνεται ανά τακτά χρονικά διαστήματαμε υποδείξεις χρηστών.

Η παραπάνω βιβλιοθήκη περιέχει τα ακόλουθα:

- Βιβλιοθήκες με **CCS Timing**, **ECSM Timing** and **NLDM/NLPM** δεδομένα (αργά, τυπικά γρήγορα, χαμηλής θερμοκρασίας και χείριστης χαμηλής θερμοκρασίας.
- Γεωμετρική βιβλιοθήκη σε μορφή **Library Exchange**
- Βιβλιοθήκες προσομοίωσης σε **Verilog** και **Spice**, πριν και μετά την εξαγωγή των παρασιτικών καθυστερήσεων.
- Διατάξεις των **Cells** σε **GDSII**
- Σχήματα σε μορφές **EDIF PNG**
- Βιβλιο δεδομένων βιβλιοθήκης σε μορφή **HTML**
- Ελέυθερης πρόσβασης Βάση Δεδομένων με διατάξεις και **netlists**

Ακολουθούν για παράδειγμα ενδεικτικά δεδομένα χρονισμού για μια πύλη τύπου **AND2_X1** .

Είσοδος στην πύλη

```
cell (AND2_X1) {
        pin (A1) {

                direction               : input;
                capacitance             : 0.000543;
                fall_capacitance        : 0.000540;
                rise_capacitance        : 0.000547;
                fall_capacitance_range          (0.000486, 0.000601);
                rise_capacitance_range          (0.000469, 0.000609);
                max_transition          : 0.600000;
        }

        pin (A2) {
                ......................
        }

        pin (ZN) {
                ......................
        }
}
```

Εικόνα 3.8 ενδεικτικά περιεχόμενα **Standard Cell Library** για πύλη **AND** 2 εισόδων

Δίδονται τα δεδομένα που περιγράφουν την A1 είσοδο της, όπως χωρητικότητα, που έπειτα διαχωρίζεται σε ανόδου και καθόδου, το εύρος των αντίστοιχων χωρητικοτήτων και ο μέγιστος χρόνος μετάδοσης ενός σήματος εισόδου. Ομοίως περιγράφεται και η A2 είσοδος.

Εξοδος από την πύλη

```
pin (ZN) {

        direction               : output;
        max_capacitance         : 0.025600;
        max_transition          : 0.600000;
        function                : "(A1 & A2)";

        timing () {

                related_pin        : "A1";
                timing_sense       : positive_unate;

                cell_fall(Timing_data_X1) {
                        values ("0.023009,0.025401,0.029763,0.037776,0.053083,0.083403,0.144080", \
                                "0.028061,0.030452,0.034814,0.042824,0.058114,0.088432,0.149083", \
                                "0.034724,0.037276,0.041841,0.050021,0.065334,0.095600,0.156223", \
                                "0.044981,0.047800,0.052758,0.061396,0.076998,0.107269,0.167779", \
                                "0.061197,0.064453,0.070166,0.079888,0.096625,0.127442,0.187951", \
                                "0.086875,0.090849,0.097847,0.109399,0.128647,0.162130,0.224500", \
                                "0.130507,0.135411,0.143863,0.158202,0.181444,0.220061,0.288321");
                }
                cell_rise(Timing_data_X1) {
                        ..................................
                }
                fall_transition(Timing_data_X1) {
                        ..................................
                }
                rise_transition(Timing_data_X1) {
                        ..................................
                }
        }
}
```

Εικόνα 3.9 ενδεικτικά περιεχόμενα **Standard Cell Library** για την έξοδο μιας πύλης

Για την επεξεργασία της αντίστοιχης εξόδου της πύλης η διαδικασία είναι αισθητά δυσκολότερη. Υπάρχουν πάλι αποθηκευμένες πληροφορίες όπως μέγιστος χρόνος μετάδοσης και χωρητικότητα αλλά και η λογική συνάρτηση που έχει υλοποιηθεί στην έξοδο της πύλης. Βασικό σημείο εδώ αποτελεί η συνάρτηση **timing()** η οποία αποτελείται από 4 διαφορετικές περιπτώσεις δεδομένων χρονισμού: **cell_fall**, **cell_rise**, **fall_transition**, **rise_transition**. Καθε συνάρτηση περιέχει ένα πινάκα n*n όπου χρήση παρεμβολής βρίσκουμε την καθυστέρηση της πύλης. Η καθυστέρηση λοιπόν μιας πύλης μπορεί να διαφοροποιείται κάθε φορά και υπολογίζεται παρεμβάλλοντας στην αντίστοιχη γραμμή και στήλη του πίνακα. Πιο συγκεκριμένα οι γραμμές του πίνακα αντιπροσωπεύουν το χρόνο εναλλαγής στην είσοδο από 0 σε 1 και οι στήλες την εξωτερική χωρητικότητα που βλέπει συνολικά η πύλη. Το εκαστοτε στοιχείο αντιστοιχίζεται με τη βοήθεια ενός **look_up_table** όπως ο επόμενος.

```
lu_table_template (Timing_data_X1) {
        variable_1 : input_net_transition;
        variable_2 : total_output_net_capacitance;
        index_1 ("0.007500,0.018750,0.037500,0.075000,0.150000,0.300000,0.600000")
        index_2 ("0.000400,0.000800,0.001600,0.003200,0.006400,0.012800,0.025600")
 }
```

Εικόνα 3.10 βοηθητικός πίνακας **look_up_table** για την εξαγωγή της καθυστέρησης κάθε πύλης

Όλες οι παραπάνω πληροφορίες αποθηκεύονται επίσης σε κατάλληλες δομές που έχουμε ορίσει στο πνεύμα τις προηγούμενης παραγράφου.

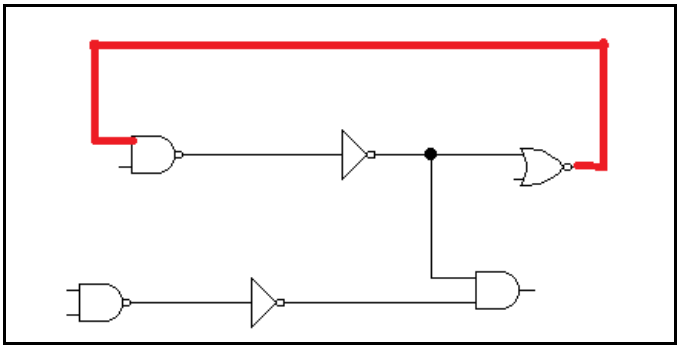
3.3 Υπολογισμοί

3.3.1 Χωρισμός του κυκλώματος σε επίπεδα.

Θα διαχωρίσουμε τις πύλες του κυκλώματος μας σε επίπεδα χρήση της εξής απλής παρατήρησης:

Σε κάθε πύλη επιπέδου Ν εισέρχονται σήματα, τα οποία αποτελούν εξόδους σε πύλες έως και επίπεδου Ν-1.

Εδώ πρέπει να αποσαφινίσουμε πως ο παραπάνω κανόνας ισχύει για κυκλώματα στα οποία δεν παρατηρείται ανάδραση (π.χ. όσα περιέχουν **flip-flops**). Δηλαδή κυκλώματα στα οποία η έξοδος μιας πύλης αποτελεί είσοδο σε πύλη, της οποίας η έξοδος άμεσα ή έμμεσα καταλήγει πάλι ως είσοδος της αρχικής.

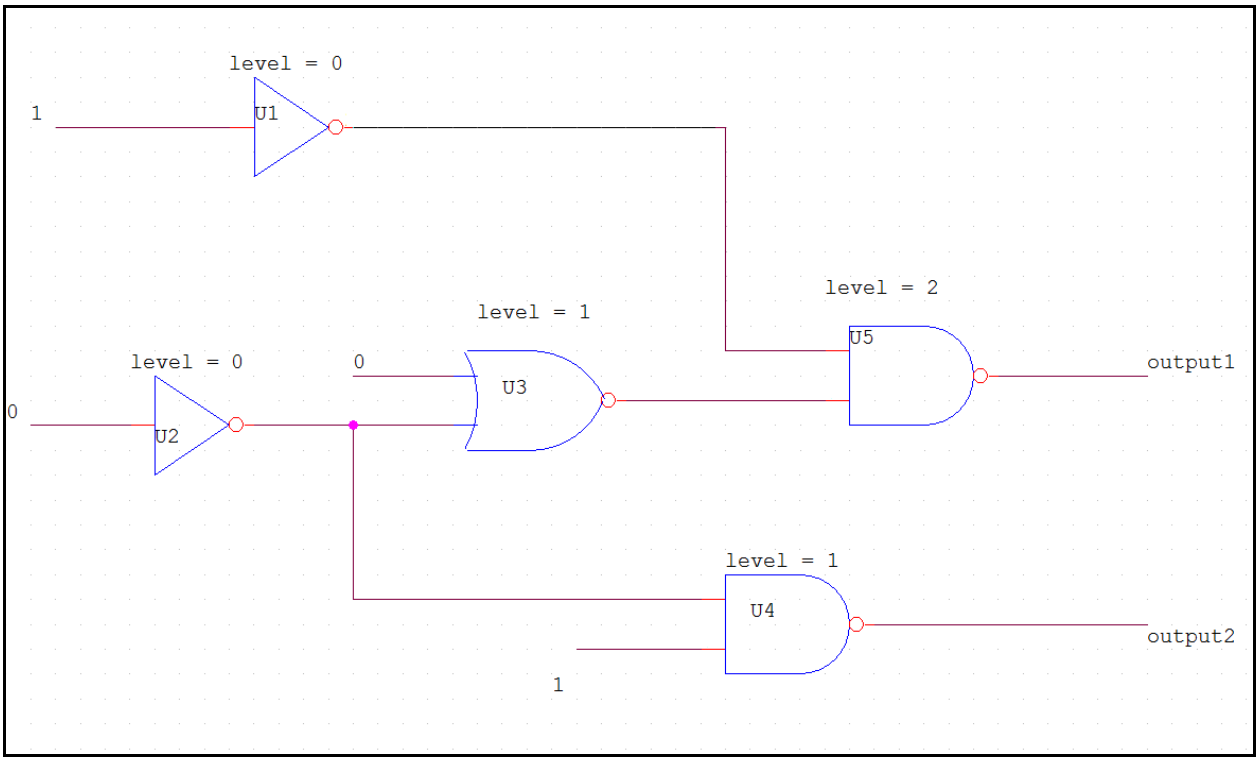


Εικόνα 3.11 ενδεικτικό κύκλωμα όπου παρατηρείται ανάδραση

Έτσι για παράδειγμα μια πύλη η οποία έχει εισόδους μόνο primary inputs εξορισμού θα είναι επιπέδου 0. Ενώ μια πύλη **NAND** που αποτελείται π.χ. από τρεις εισόδους εκ των οποίων η μία είναι **primary input**, η δεύτερη εξερχεται από πύλη επιπέδου 1 και η τρίτη εξέρχεται από πύλη επιπέδου 2 θα έχει επίπεδο κατά 1 μεγαλύτερο από το επίπεδο της μέγιστης εισόδου, δηλαδή 2+1=3.

Συνοψίζοντας για το επίπεδο μιας πύλης k, i εισόδων θα ισχύει:

$$\mathbf{level(k) = max\ (\ level(\ input_1\ ),\ level(\ input_2\ ),\ \dots\ ,\ level(\ input_i\ )\ ) + 1}$$



Εικόνα 3.12 επίπεδα πυλών λογικού κυκλώματος

Να επισημάνουμε επίσης ότι η έξοδος των **flip-flops** για τις πύλες στις οποίες ενδεχομένως αποτελεί είσοδο, ισοδυναμεί με σήμα εισόδου επιπέδου 0 κατά παράβαση του παραπάνω κανόνα.

Ακολουθεί μία ενδεικτική υλοποίηση της συνάρτησης **levelize()** η οποία πραγματώνει την παραπάνω λειτουργία για κάθε πύλη του κυκλώματος.

```
void levelize(struct component *head_comp)
{
    struct component *cur_comp=NULL, *prev_comp=NULL;
    struct signals *cur_sig=NULL;
    int cnt=0, level=0, j=0, i=0, ok=0, end=0;

    // search all gates. for each gate: if all its inputs are primary inputs then gate level=0
    for(cur_comp=head_comp;cur_comp->next!=NULL;cur_comp=cur_comp->next)
    {
        if(cur_comp->num_of_sig_ptrs==0)
        {
            if(cur_comp->type_of_output=='n')
            {
                cur_comp->out_sig_ptr->level=0;
            }
            else
            {
                        cur_comp->output_ptr->level=0;
            }
            cur_comp->level=0;
        }
    }

    do
    {
          level++; // will compute next level gates
          ok=0;
          end=1;
          //search all gates
          for( cur_comp = head_comp; cur_comp -> next != NULL; cur_comp = cur_comp -> next )
          {
                      if(cur_comp->level==-1) //if gate's level not found yet this is excecuted
                      {
                          ok=1;
                          for(i=0;i<cur_comp->num_of_sig_ptrs;i++) // check all gates' inputs
                          {
                              if(cur_comp->sig_ptr[i]->level==-1 || cur_comp->sig_ptr[i]->level==level)
                              {
                                ok=0; // if ok==0 can't compute gate's level so far
                              }
                          }
                          if( ok )
                          {
                              if( cur_comp -> type_of_output == 'n' ) // if output of gate isn't output of circuit
                              {
                                  cur_comp -> out_sig_ptr -> level = level;
                              }
                              else
                              {
                                  cur_comp -> output_ptr -> level = level;
                              }
                          }
                          end=0; // there are more gates whose level hasn't be computed yet
                      }
          }
    }
    while(!end);
```

Εικόνα 3.13 υλοποίηση συνάρτησης εύρεσης επιπέδου κάθε πύλης

3.3.2 Υπολογισμός χωρητικοτήτων εξόδου κάθε πύλης

Στην ανάλυση μας *η χωρητικότητα εξόδου μιας πύλης ισούται με το άθροισμα των χωρητικοτήτων όλων των πυλών που αυτή οδηγεί*. Ισχύει λοιπόν

κ : πύλη, τη χωρητικότητα εξόδου της οποίας θέλουμε να υπολογίσουμε

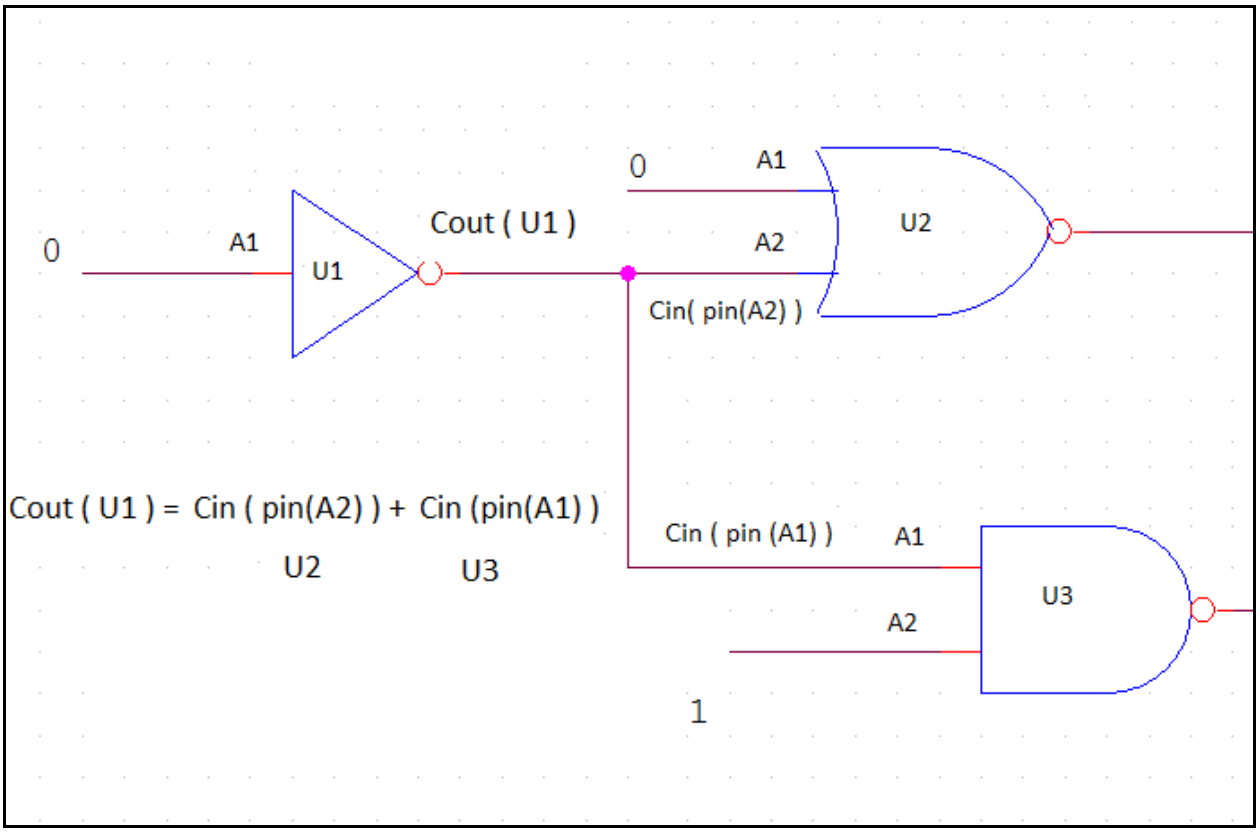
$\mathbf{i_1}$ **, … ,** $\mathbf{i_n}$: πύλες στις οποίες η έξοδος της κ αποτελεί είσοδο

$\mathbf{C_{out}}$: χωρητικότητα εξόδου

$\mathbf{C_{in}}$: Χωρητικότητα μιας συγκεκριμένης εισόδου στην αντίστοιχη πύλη

Οπότε γενικεύοντας

$$\mathbf{C_{out}(k) = C_{in} (i_1) + \dots + C_{in} (i_n)}$$



Εικόνα 3.14 υπολογισμός εξωτερικής χωρητικότητας μιας πύλης

3.3.3 Υπολογισμός καθυστέρησης κάθε πύλης

Επόμενο βήμα των υπολογισμών μας αποτελεί η εύρεση της καθυστέρησης μιας πύλης.

*Timing sense μιας πύλης (**positive, negative, non unate**)*

Κάθε είσοδος μιας πύλης μπορεί να χαρακτηριστεί **positive unate**, **negative unate** ή **non unate**.

Μια είσοδος θεωρείται **positive unate**, αν μία αυξανόμενη μετάβαση στην είσοδο (δηλαδή από 0 σε 1) επιδρά αυξανόμενα στην έξοδο, ή δεν της μεταβάλει την τιμή (δηλαδή η έξοδος παραμένει σταθερή στην προηγούμενη τιμή της). Επίσης μια πτωτική μετάβαση στην είσοδο (από 1 σε 0) προκαλεί πτωση στη έξοδο ή δεν της μεταβάλει την τιμή. Π.χ. μια πύλη **AND**.

	AND 2	
input	input	output
A1	A2	Z
0	0	0
0	1	0
1	0	0
1	1	1

αν κάποια είσοδος από 1 γίνει 0, η έξοδος είτε θα γίνει 0, είτε θα παραμείνει ίδια

αν κάποια είσοδος από 0 γίνει 1, η έξοδος είτε θα γίνει 1, είτε θα παραμείνει ίδια

Πίνακας 3.1 Πίνακας αληθείας **AND** 2 εισόδων και επεξήγηση του **timing sense** της

Αντίστροφα σε μία **negative unate** είσοδο, μία αυξανόμενη μετάβαση προκαλεί πτώση στη έξοδο ή δεν της μεταβάλει την τιμή και μία πτωτική μετάβαση στην είσοδο προκαλεί μια αυξανόμενη μετάβαση στην έξοδο ή δεν της μεταβάλει την τιμή. Π.χ. μια πύλη **NAND**.

	NAND 2	
input	input	output
A1	A2	Z
0	0	1
0	1	1
1	0	1
1	1	0

αν κάποια είσοδος από 1 γίνει 0, η έξοδος είτε θα γίνει 1, είτε θα παραμείνει ίδια

αν κάποια είσοδος από 0 γίνει 1, η έξοδος είτε θα γίνει 0, είτε θα παραμείνει ίδια

Πίνακας 3.2 Πίνακας αληθείας **NAND** 2 εισόδων και επεξήγηση του **timing sense** της

Μία **non unate** είσοδος εξαρτάται από την κατάσταση των υπόλοιπων εισόδων. Π.χ. μια πύλη **XOR**.

Παρεμβολή και υπολογισμός χρόνων

Δεδομένης της χωρητικότητας εξόδου της κάθε πύλης θα υπολογίσουμε την καθυστέρηση αυτής. Για παράδειγμα αν γνωρίζουμε ότι ο χρόνος μετάβασης στην είσοδο μιας πύλης ισούται με 0.22 ns και η χωρητικότητα εξόδου της πύλης ισούται με 0.49 pF χρήση του ακόλουθου πίνακα αρκεί να παρεμβάλουμε μεταξύ των τιμών 0.1937, 0.7280, 0.2327, 0.7676 και να βρούμε το ζητούμενο χρόνο. Η διαδικασία θα εξηγηθεί αναλυτικά σε επόμενη παράγραφο.

fall_transition (delay_template_3X3){				
index_1("0.1, 0.3, 0.7"); /*input transition */				
index_2("0.16, 0.35, 1.43"); /* output capacitance */				
values (/*	0.16	0.35	1.43	*/
/* 0.1 */	"0.0817,	**0.1937,**	**0.7280",**	
/* 0.3 */	"0.1018,	**0.2327,**	**0.7676",**	
/*0.7 */	"0.1334,	0.2937,	0.8452");	

Πίνακας 3.3 Πίνακας όπου θα παρεμβάλουμε δεδομένες τιμές για να βρούμε την καθυστέρηση μετάβασης κατά την πτώση της τιμής της εισόδου μιας πύλης

Υπολογισμός των τιμών χρονισμού κάθε πύλης

Υπολογίζουμε λοιπόν για κάθε ενα από τους πίνακες **cell_fall**, **cell_rise**, **fall_transition**, **rise_transition** την αντίστοιχη καθυστέρηση. Εν συνεχεία θα συγκρίνουμε αυτές τις τιμές σύμφωνα με τα παρακάτω κριτήρια:

- Για μία είσοδο που είναι **positive_unate** αν **cell_fall > cell_rise** η καθυστέρηση της πύλης για την παρούσα είσοδο θα ισούται με **cell_fall** και ο χρόνος μετάβασης με **fall_transition**. Διαφορετικά η καθυστέρηση της πύλης για την παρούσα είσοδο θα ισούται με **cell_rise** και ο χρόνος μετάβασης με **rise_transition**.

- Για μια **negative unate** είσοδο αν **cell_fall > cell_rise**, ο χρόνος μετάβασης θα προκύπτει από τον πίνακα **rise_transition** αλλιώς από τον **fall_transition**.

max delay	**timing sense**	**transition**
cell_fall	positive	*fall_transition*
cell_rise	positive	*rise_transition*
cell_fall	negative	*rise_transition*
cell_rise	negative	*fall_transition*

Πίνακας 3.4 πίνακας για τον καθορισμό της σωστής τιμής μετάβασης της πύλης

Αυτή διαδικασία επαναλαμβάνεται για όλες τις εισόδους μιας πύλης. Έτσι αφού έχουμε επεξεργαστεί όλες τις εισόδους η μέγιστη καθυστέρηση και ο μέγιστος χρόνος μετάβασης αποτελούν τα χαρακτηριστικά της πύλης. Αυτή η διαδικασία έπειτα επαναλαμβάνεται για κάθε πύλη του κυκλώματος.

Κάναμε την παραδοχή ότι ο χρόνος μετάβασης για κάποια είσοδο ισούται με 0 για τις πύλες επιπέδου 0 και με το χρόνο μετάβασης της πύλης από τη οποία εξέρχεται η εν λόγω είσοδος για κάθε άλλη πύλη.

3.3.4 Παρεμβολή για τον υπολογισμό των ζητούμενων χρόνων

Θα εξηγήσουμε τώρα πως προκύπτουν οι ζητούμενοι χρόνοι χρήση παρεμβολής. Θεωρούμε πως αναζητούμε το **fall_transition** δεδομένου πως ο χρόνος μετάβασης στην είσοδο είναι x_0=0.15ns και η συνολική χωρητικότητα εξόδου ισούται με y_0=1.16pF.

Εστω x_1, x_2 χρόνοι μετάβασης στην είσοδο του index_1 για τους οποίους ισχύει $x_1 < x_0 < x_2$ και

έστω y_1, y_2 συνολικές χωρητικότητες εξόδου του index_2 για τις οποιες ισχύει $y_1 < y_0 < y_2$.

Εντοπίζουμε τις αντίστοιχες τιμές στον πίνακα T_{11}, T_{12}, T_{21}, T_{22}

fall_transition (delay_template_3X3){			
index_1("0.1, 0.3, 0.7"); /*input transition */			
index_2("0.16, 0.35, 1.43"); /* output capacitance*/			
values(/*	0.16	0.35	1.43 */
/* 0.1 */	"0.0817,	0.1937,	0.7280",
/* 0.3 */	"0.1018,	0.2327,	0.7676",
/*0.7 */	"0.1334,	0.2937,	0.8452");

Πίνακας 3.5 Πίνακας όπου θα παρεμβάλουμε δεδομένες τιμές. Οι παρεμβάλλουσες τιμές δίνονται σε γαλάζιο φόντο

Παρεμβάλουμε λοιπόν τις αντίστοιχες τιμές και υπολογίζουμε το ζητούμενο T_{00} για τα δεδομένα x_0, y_0 ως εξής:

$$T_{00} = x_{20} * y_{20} * T_{11} + x_{20} * y_{01} * T_{12} + x_{01} * y_{20} * T_{21} + x_{01} * y_{01} * T_{22}$$

όπου

$$x_{01} = (x_0 - x_1) / (x_2 - x_1)$$

$$x_{20} = (x_2 - x_0) / (x_2 - x_1)$$

$$y_{01} = (y_0 - y_1) / (y_2 - y_1)$$

$$y_{20} = (y_2 - y_0) / (y_2 - y_1)$$

Εκτελώντας τους παραπάνω υπολογισμούς προκύπτει λοιπόν εύκολα ότι ο ζητούμενος χρόνος θα ισούται με $\mathbf{T_{00}=0.8516}$.

Σε περίπτωση που η ζητούμενη τιμή προκύπτει από x_0, y_0 τα οποία ξεφεύγουν εκτός των ορίων των **index_1** και **index_2**, τον ρόλο των παρεμβάλοντων τιμών παίρνουν τα δυο πλησιέστερα σε αυτή την τιμή στοιχεία.

Φυσικά όλη αυτή η διαδικασία μπορεί να παραληφθεί σε περίπτωση που τα δεδομένα μας ταυτίζονται ακριβώς στα **index_1** και **index_2**, οπότε ο ζητούμενος χρόνος θα προκύπτει εύκολα ανακτώντας απλώς ένα στοιχείο από τον πίνακα (δίχως χρήση παρεμβολής), κάτι το οποίο όμως είναι εξαιρετικά σπάνιο όπως παρατηρήθηκε και στα πειράματα μας.

3.3.5 Εύρεση κρίσιμου μονοπατιού

Έχοντας ακολουθήσει πιστά όλη την προαναφερθείσα διαδικασία διαθέτουμε όλα τα ενδιάμεσα αποτελέσματα για τον υπολογισμού του κρίσιμου μονοπατιού. Όπως έχει εξηγηθεί και προηγουμένως, κρίσιμο μονοπάτι είναι αυτό που επιφέρει τη μέγιστη καθυστέρηση από μία είσοδο σε μία έξοδο του κυκλώματος και ως αποτέλεσμα χαρακτηρίζει ολόκληρο το κύκλωμα.

Αρκεί λοιπόν για κάθε πύλη να υπολογίσουμε τη συνολική καθυστέρηση αυτής. Η συνολική καθυστέρηση μιας πύλης προκύπτει ως άθροισμα δυο παραγόντων:

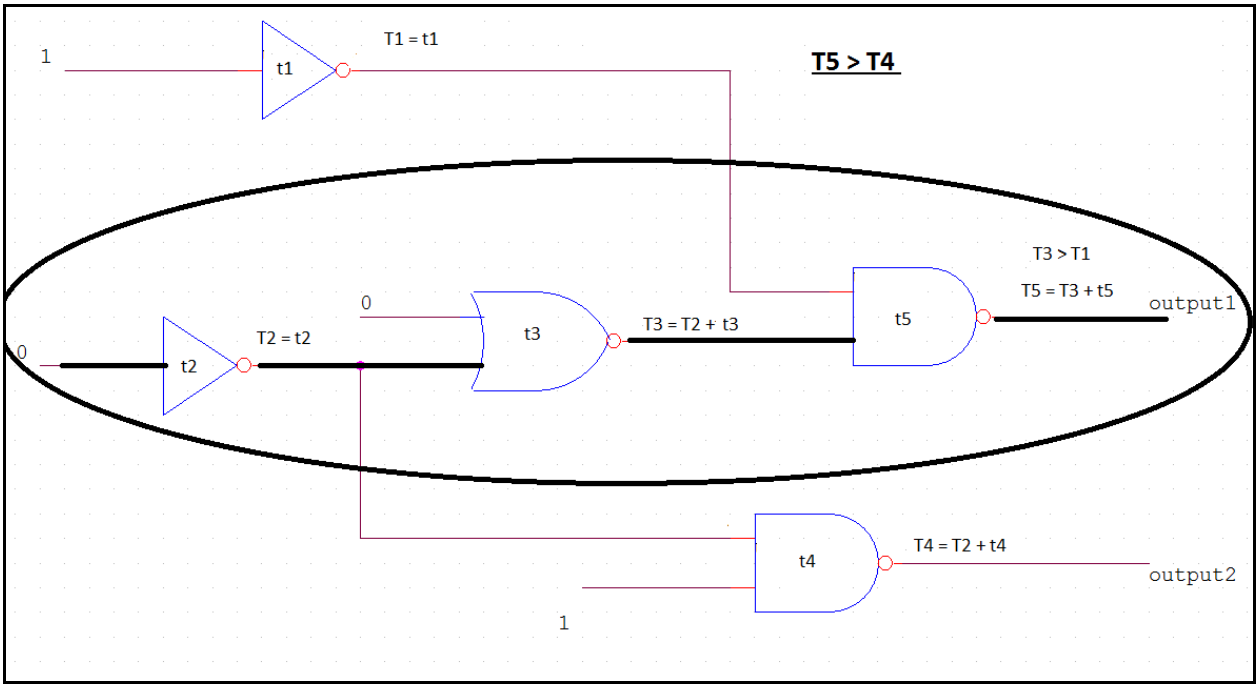
1) Την καθυστέρηση αυτής και
2) Της μέγιστης συνολικής καθυστέρησης από όλες τις πύλες των οποίων η έξοδος αποτελεί είσοδο στην εν λόγω πύλη.

Αν λοιπόν t_i καθυστέρηση της πύλης i, T_i συνολική καθυστέρηση μέχρι και την πύλη i και $k_o,\ldots,k_n$ πύλες των οποίων η έξοδος αποτελεί είσοδο στην πύλη i, για τη συνολική καθυστέρηση της πύλης i θα ισχύει:

$$\mathbf{T_i = t_i + max\ (\ T_{k0}, \ldots ,\ T_{kn}\ )}$$

Από τα παραπάνω καθίσταται προφανές ότι οι συνολική καθυστέρηση μιας πύλης επιπέδου 0 θα ισούται με την ίδια την καθυστέρηση της πύλης (δηλαδη $T_i = t_{i\ αφού}$ όλες οι είσοδοι στην πύλη θα αποτελούν **primary inputs**. Η σάρωση των πυλών πραγματοποιείται από τις εισόδους προς τις εξόδους του κυκλώματος, δηλαδή κατά αύξουσα τιμή επιπέδου πύλης. Πρώτα υπολογίζεται τετριμμένα η συνολική καθυστέρηση των πυλών επιπέδου 0, έπειτα η καθυστέρηση των πυλών επιπέδου 1, έπειτα του $2^{ου}$ επιπέδου κ.ο.κ.

Αφού λοιπόν υπολογίσουμε τη συνολική καθυστέρηση κάθε πύλης εντοπίζουμε την πύλη που η έξοδος της αποτελεί και έξοδο του κυκλώματος και η συνολική της καθυστέρηση είναι η μέγιστη. Από τις πύλες των οποίων οι έξοδοι αποτελούν εισόδους στην εν λόγω πύλη επιλέγουμε αυτή με τη μέγιστη συνολική καθυστέρηση και οπισθοχωρούμε εκ νέου σε μία πύλη ακόμη μικρότερου επιπέδου, πάλι αυτή με τη μέγιστη συνολική καθυστέρηση. Η διαδικασία επαναλαμβάνεται εώς ότου φτάσουμε σε μία πύλη επιπέδου 0. Και το μονοπάτι που ανακαλύψαμε κατά την προηγούμενη διαδικασία αποτελεί το κρίσιμο μονοπάτι.



Εικόνα 3.15 Εύρεση Κρίσιμου Μονοπατιού

Κεφάλαιο 4. Παρουσίαση Αποτελεσμάτων

4.1 Κρίσιμο Μονοπάτι Δεδομένου Κυκλώματος

Σε αυτό το σημείο θα επιδείξουμε τα αποτελέσματα των πειραμάτων μας για τα δεδομένα κυκλώματα εισόδου. Η παρουσίαση θα γίνει με το πιο απλό κύκλωμα που επεξεργαστήκαμε για πρακτικούς λόγους, π.χ. χρονικοί περιορισμοί της παρούσας εργασίας-παρουσίασης και έκτασης κυκλώματος. Φυσικά η ίδια μεθοδολογία εφαρμόστηκε και σε ακόμη μεγαλύτερα κυκλώματα όπου και πήραμε και τα ανάλογα αποτελέσματα.

Παραθέτουμε λοιπόν την περιγραφή του δεδομένου κυκλώματος εισόδου S27_vhdl_netlist.vhdl σε VHDL την οποία αρχικά επεξεργάστηκε ο κώδικας μας.

```
port( S1, S2, S3, S4, S6, S8, S10 : in std_logic;  S7, S11, S5, S9_out : out std_logic);

signal net285, net286, net292, net305, net291, net282, net281,
      | n6, n7, n8, n9, n10, n11, n12, n13 : std_logic;

begin
   U8 : INV_X32 port map( A => S6, ZN => net291);
   U9 : INV_X8 port map( A => S10, ZN => net286);
   U10 : INV_X16 port map( A => S4, ZN => n12);
   U11 : AOI21_X4 port map( B1 => net281, B2 => net282, A => n11, ZN => S7);
   U12 : NOR2_X4 port map( A1 => n12, A2 => S10, ZN => net281);
   U13 : NOR2_X4 port map( A1 => S6, A2 => S2, ZN => net282);
   U14 : INV_X16 port map( A => S6, ZN => n6);
   U15 : NOR3_X4 port map( A1 => n10, A2 => S2, A3 => S6, ZN => n13);
   U16 : INV_X16 port map( A => S8, ZN => n8);
   U17 : NOR3_X4 port map( A1 => n8, A2 => S6, A3 => S1, ZN => n7);
   U18 : NAND4_X4 port map( A1 => net286, A2 => net285, A3 => net291, A4 => S4,
                            ZN => net292);
   U19 : INV_X32 port map( A => S2, ZN => net285);
   U20 : INV_X32 port map( A => S1, ZN => n11);
   U21 : NAND2_X4 port map( A1 => n9, A2 => S4, ZN => n10);
   U22 : INV_X16 port map( A => S10, ZN => n9);
   U23 : NOR2_X2 port map( A1 => n13, A2 => n7, ZN => S5);
   U24 : NAND3_X2 port map( A1 => S8, A2 => n11, A3 => n6, ZN => net305);
   U25 : NAND2_X2 port map( A1 => net292, A2 => net305, ZN => S9_out);
   U26 : AOI21_X2 port map( B1 => net285, B2 => net286, A => S3, ZN => S11);
end SYN_S27_arch;
```

Εικόνα 4.1 περιγραφή S27_vhdl_netlist.vhdl

Αφού δημιουργήθηκαν οι αντίστοιχες δομές δεδομένων στη μνήμη του υπολογιστή, καθορίστηκε το επίπεδο κάθε πύλης, όπου η εν λόγω πληροφορία φαίνεται στα ακολουθα αποτελέσματα.

_gate	_type	level
U8	INV_X32	0
U9	INV_X8	0
U10	INV_X16	0
U11	AOI21_X4	2
U12	NOR2_X4	1
U13	NOR2_X4	0
U14	INV_X16	0
U15	NOR3_X4	2
U16	INV_X16	0
U17	NOR3_X4	1
U18	NAND4_X4	1
U19	INV_X32	0
U20	INV_X32	0
U21	NAND2_X4	1
U22	INV_X16	0
U23	NOR2_X2	3
U24	NAND3_X2	1
U25	NAND2_X2	2
U26	AOI21_X2	1

Πίνακας 4.1 αποτελέσματα επιπεδοποίσης S27_vhdl_netlist.vhdl

Έπειτα υπολογίσαμε τις εξωτερικές χωρητικότητες για κάθε πύλη και εν συνεχεία την καθυστέρηση κάθε πύλης. Τα αποτελέσματα παρουσιάζονται στον αριστερό και στον πίνακα στα δεξιά αντιστοίχως.

gate	type	Cout
U8	INV_X32	0.002471
U9	INV_X8	0.003273
U10	INV_X16	0.001892
U11	AOI21_X4	0
U12	NOR2_X4	0.002161
U13	NOR2_X4	0.002035
U14	INV_X16	0.001021
U15	NOR3_X4	0.000998
U16	INV_X16	0.00218
U17	NOR3_X4	0.000994
U18	NAND4_X4	0.000975
U19	INV_X32	0.003508
U20	INV_X32	0.002849
U21	NAND2_X4	0.00218
U22	INV_X16	0.001742
U23	NOR2_X2	0
U24	NAND3_X2	0.000936
U25	NAND2_X2	0
U26	AOI21_X2	0

gate	type	delay
U8	INV_X32	0.00521
U9	INV_X8	0.006426
U10	INV_X16	0.005047
U11	AOI21_X4	0.020161
U12	NOR2_X4	0.015461
U13	NOR2_X4	0.015209
U14	INV_X16	0.004755
U15	NOR3_X4	0.021819
U16	INV_X16	0.005143
U17	NOR3_X4	0.021809
U18	NAND4_X4	0.023397
U19	INV_X32	0.005389
U20	INV_X32	0.005276
U21	NAND2_X4	0.01172
U22	INV_X16	0.004996
U23	NOR2_X2	0.013547
U24	NAND3_X2	0.017875
U25	NAND2_X2	0.011527
U26	AOI21_X2	0.020704

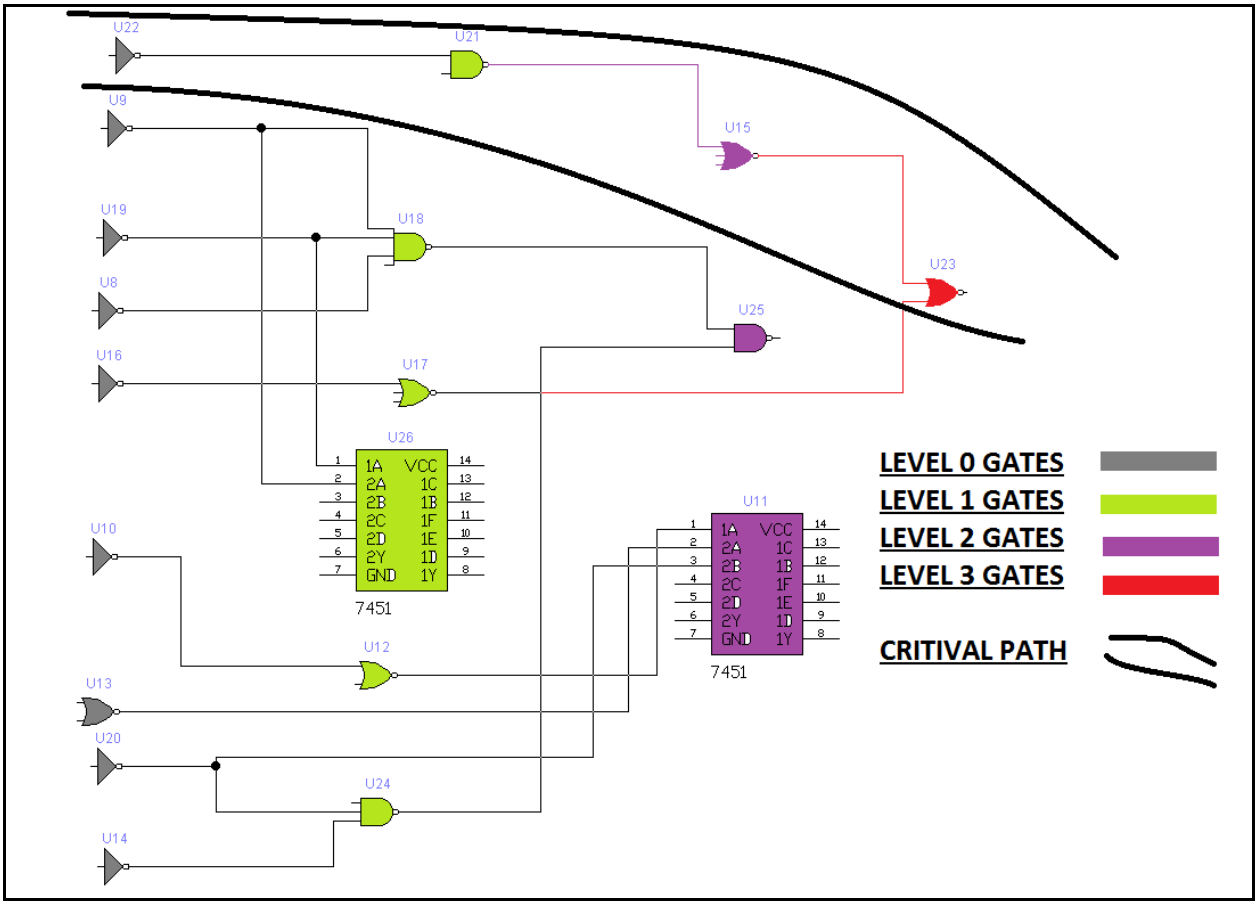
Πίνακες 4.2 , 4.3 εξωτερικές χωρητικότητες και καθυστέρηση κάθε πύλης αντίστοιχα στο αρχείο εισόδου S27_vhdl_netlist.vhdl

τέλος είχαμε υπολογίσει όλα τα απαραίτητα στοιχεία για να εντοπίσουμε το κρίσιμο μονοπάτι

gate_id	gate_type	gate_delay	total_path_delay	gate_level
U23	NOR2_X2	0.013547	0.052081	3
U15	NOR3_X4	0.021819	0.038535	2
U21	NAND2_X4	0.01172	0.016716	1
U22	INV_X16	0.004996	0.004996	0

Πίνακας 4.4 κρίσιμο μονοπάτι του S27_vhdl_netlist.vhdl

Το κρίσιμο μονοπάτι λοιπόν αποτελείται από τις πύλες U22, U21,U15 και U23 και στον παραπάνω πίνακα στην στήλη **total path delay** δίδεται η συνολική καθυστέρηση κατά μήκος του κρίσιμου μονοπατιού εώς και την αντίστοιχη πύλη. Ακολουθεί μία σχηματική αναπαράσταση του επεξεργαζόμενου κυκλώματος, όπου έχουμε σημειώσει το κρίσιμο μονοπάτι.



Εικόνα 4.2 σχηματική αναπαράσταση του S27_vhdl_netlist.vhdl και παρουσίαση του κρίσιμου μονοπατιού

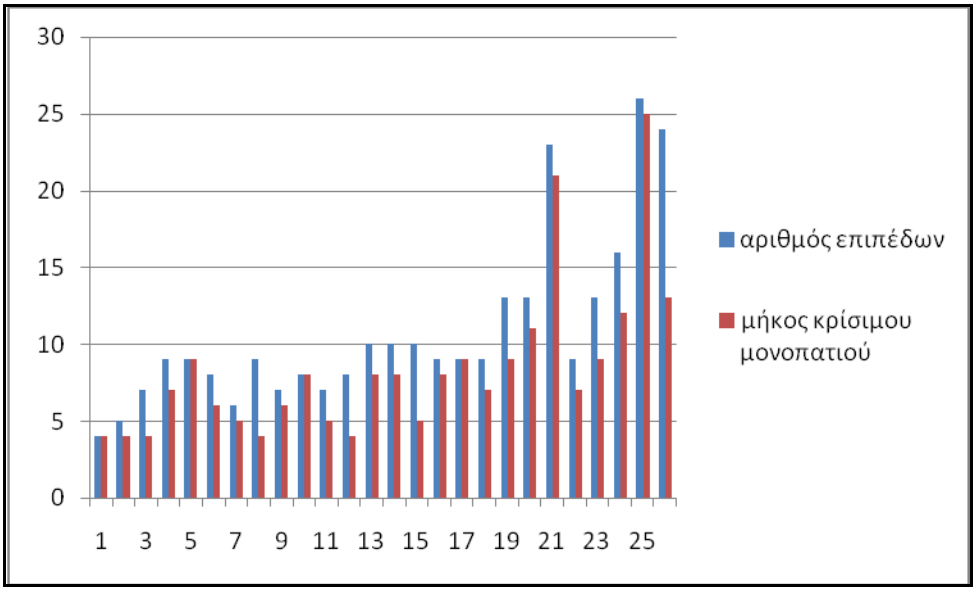
4.2 Παρατηρήσεις που ενισχύουν την ορθότητα των αποτελεσμάτων.

Θα ακολουθήσει η παρουσίαση ορισμένων παρατηρήσεων που βασίζονται κυρίως στη διαισθητική αντίληψη του προβλήματος της εργασίας μας και επαληθεύονται από τα αποτελέσματα των πειραμάτων που πραγματοποιήσαμε.

4.2.1 Μήκος Κρίσιμου Μονοπατιού

Η κοινή διαίσθηση μας προϊδεάζει ότι το κρίσιμο μονοπάτι σε ένα κύκλωμα επιπέδου N, θα αποτελείται από όσο το δυνατό περισσότερες πύλες κ, όπου θα ισχύει όμως **κ<=N**. Για παράδειγμα σε ένα κύκλωμα με 10 επίπεδα πυλών το κρίσιμο μονοπάτι είναι πιο πιθανό να απαρτίζεται από 7 ή 8 πύλες παρά από 3 καθώς μια πρώτη εύστοχη σκέψη είναι ότι περισσότερες πύλες επιφέρουν μεγαλύτερη καθυστέρηση. Περιπτώσεις όπου αυτή η παρατήρηση δεν ισχύει και το κρίσιμο μονοπάτι μπορεί να αποτελείται από λιγότερες πύλες θα εξηγήσουμε σε προσεχείς παραγραφους.

Το γράφημα που ακολουθεί συνοψίζει την παραπάνω παρατήρηση και ενισχύει την ορθότητα των αποτελεσμάτων μας για τα δεδομένα κυκλώματα εισόδου που επεξεργαστήκαμε.

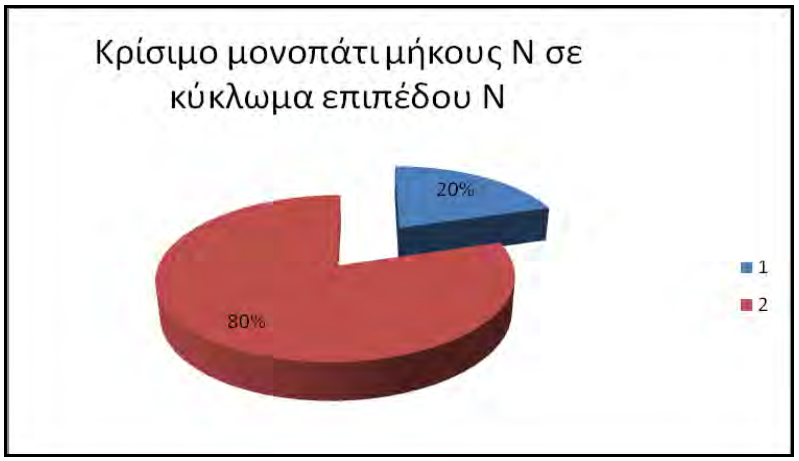


Εικόνα 4.3 Σύγκριση αριθμού επιπέδων με μήκους κρίσιμου μονοπατιού κάθε κυκλώματος

Από τις παραπάνω στατιστικές μετρήσεις προκύπτει λοιπόν πως για όλα τα κυκλώματα εισόδου σε σύνολο 257 επιπέδων, τα κρίσιμα μονοπάτια θα αποτελούνται από 205 συνολικά πύλες.

205 / 257 = 0,8

Μπορούμε να πούμε λοιπόν, ότι ο λόγος των πυλών του κρίσιμου μονοπατιού προς το αριθμό επιπέδων των κυκλωμάτων ισούται με 80% κάτι το οποίο διαισθητικά ήταν και αναμενόμενο. Δηλαδή **το κρίσιμο μονοπάτι σε ένα κύκλωμα Ν επιπέδων θα αποτελείται από Ν πύλες κατά 80%**.



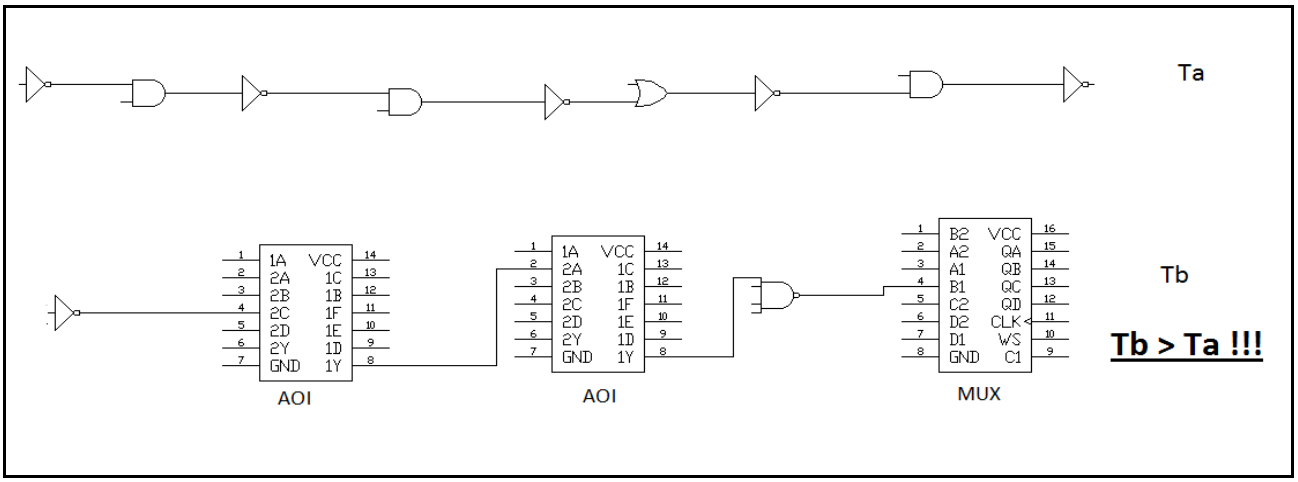
Εικόνα 4.4 πιθανότητα ένα σε κύκλωμα επιπέδου Ν το κρίσιμο μονοπάτι να διέρχεται από Ν πύλες.

Θεωρούμε πως ο αριθμός των αρχείων εισόδου που εκτελέσαμε αποτελεί σεβαστό-αντιπροσωπευτικό δείγμα και πως όσα περισσότερα κυκλώματα και αν δώσουμε στην είσοδο του προγράμματος μας, η πιθανότητα το κρίσιμο μονοπάτι σε κύκλωμα επίπεδου Ν να αποτελείται από Ν πύλες, θα **συγκλίνει στο 0,8**.

4.2.2 Πολυπλοκότητα πυλών

Είναι προφανές πως μία πύλη πολυπλοκότερης εσωτερικής υλοποίησης όπως ένας πολυπλέκτης (**mux**), ή μία πύλη που προκύπτει από το συνδυασμό απλούστερων πυλών όπως ένας **AND-OR-INVERTER (AOI)**, θα επιφέρει πολύ μεγαλύτερη καθυστέρηση στο κύκλωμα από έναν απλό αντιστροφέα η μία πύλη **AND**. Εξάλλου ο αριθμός των **transistor** που απαιτείται για την υλοποίση της εν λόγω συνάρτησης θα είναι αρκετά μεγαλύτερος από τα 4 **transistor** που απαιτεί για παράδειγμα μία πύλη **OR** για να υλοποιήσει τη λογική πράξη **ΚΑΙ**. Το ίδιο θα ισχύει και για μία πύλη αποτελούμενη από πολλές εισόδους.

Έτσι κατά παράβαση της παρατήρησης της προηγούμενης παραγράφου, το κρίσιμο μονοπάτι σε ένα κύκλωμα επιπέδου N αποτελείται από σημαντικά λιγότερες από N πύλες, εφόσον διέρχεται από τέτοιου είδους πολύπλοκα στοιχεία.



Εικόνα 4.5 μονοπάτια αποτελούμενα από απλά και πολυπλοκότερα στοιχεία αντίστοιχα

Ενδεικτικά λοιπόν των παραπάνω παραθέτουμε τα αποτελέσματα της επεξεργασίας του κυκλώματος S820_vhdl_netlist.vhdl, όπου παρόλου που το κύκλωμα μας αποτελείται από 10 διαφορετικών επιπέδων πύλες, το κρίσιμο μονοπάτι αποτελείται από μόλις 5, στις οποίες όμως συμπεριλαμβάνονται ένας πολυπλέκτης, ένας **OAI** και δυο πύλες τύπου **negative unate** για τις εισόδους τους (**NOR3**) , με βαθμό εισόδου μεγαλύτερου του 2.

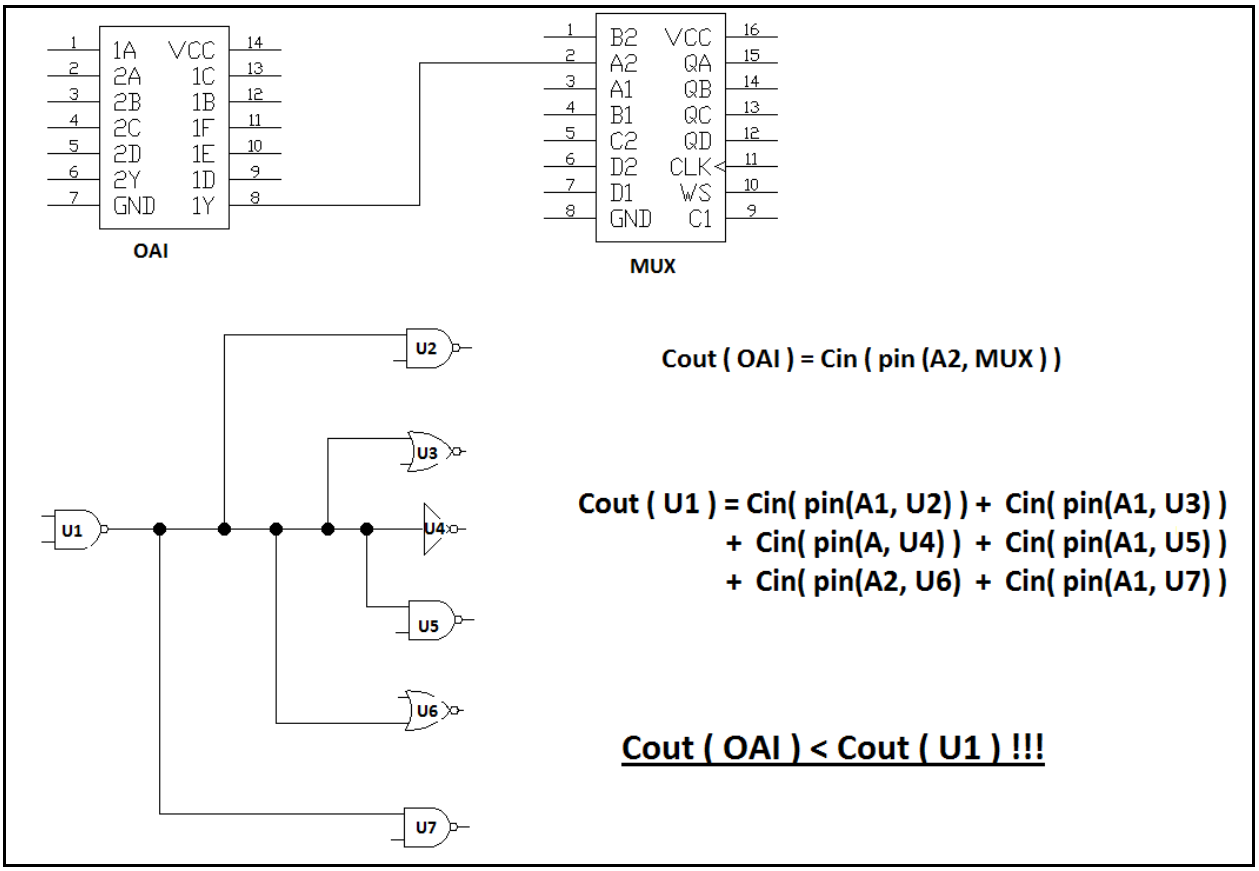
gate	gate_delay	total_path_delay	gate_level	input in
OAI22_X2	0.023879	0.161295	6	0
MUX2_X2	0.069535	0.137417	4	1
NOR3_X2	0.02623	0.067881	3	1
NOR3_X2	0.026356	0.041651	1	1
NAND2_X4	0.015295	0.015295	0	3

Πίνακας 4.5 κρίσιμο μονοπάτι του S820_vhdl_netlist.vhdl

Παρατηρήστε τη σημαντικά μεγαλύτερη καθυστέρηση που επιφέρει στο κύκλωμα μας ο πολυπλέκτης (**MUX2_X2**) και την αισθητά μικρότερη καθυστέρηση της **NAND2_X4**. Η **OAI22_X2** θα επέφερε ακόμη μεγαλύτερη καθυστέρηση στο κύκλωμα αν ήταν κάπου ενδιάμεσα στο μονοπάτι και η έξοδος της αποτελούσε είσοδο για κάποιες πύλες και όχι στην έξοδο του κυκλώματος. Αυτό εξηγεί εν μέρει και την καθυστέρηση της πύλης **NAND2_X4**, η οποία είναι μεν μικρότερη από την αντίστοιχη των **NOR** και της **OAI**, αλλά όχι σε μεγάλο βαθμό, καθώς η έξοδος της αποτελεί είσοδο σε 3 πύλες, ενώ όλων των υπολοίπων πυλών του κρίσιμου μονοπατιού η έξοδος αποτελεί είσοδο μόλις σε μία πύλη. Η τελευταία παρατήρηση αποτελεί καλή αφορμή για να εξετάσουμε άλλον έναν παράγοντα που επιφέρει σημαντική καθυστέρηση σε μία πύλη, όπως ο βαθμός εξόδου και έμμεσα η εξωτερική χωρητικότητα αυτής.

4.2.3 Εξωτερική χωρητικότητα πυλών

Όπως αναλύσαμε και στα προηγούμενα *η χωρητικότητα εξόδου μιας πύλης ισούται με το άθροισμα των χωρητικοτήτων όλων των πυλών που αυτή οδηγεί.* Μία πρώτη απλή παρατήρηση είναι πως μεγαλύτερος βαθμός εξόδου για μια πύλη, συνεπάγεται μεγαλύτερη χωρητικότητα εξόδου. Άρα και μεγαλύτερη καθυστέρηση για την πύλη καθώς για να υπολογίσουμε την καθυστέρηση της πύλης παρεμβάλαμε δυο μεγέθη, την καθυστέρηση μετάβασης και τη χωρητικότητα εξόδου.



Εικόνα 4.6 Πύλες με μικρό και μεγάλο βαθμό εξόδου αντίστοιχα

Κάπου εδώ θα ήταν φρόνιμο να επισημάνουμε πως οι παραπάνω παρατηρήσεις ισχύουν στη γενική περίπτωση δίχως να αποτελούν αναγκαία προϋπόθεση για την εισαγωγή μιας πύλης στο κρίσιμο μονοπάτι. Εξάλλου η εξωτερική χωρητικότητα μιας πύλης εξαρτάται και από τη συγκεκριμένη είσοδο στην οποία θα εισέλθει η έξοδος της πρώτης καθώς

διαφορετική χωρητικότητα επιδεικνύει για παράδειγμα η πρώτη είσοδος μιας πύλης **AND** 2 εισόδων και διαφορετική η δεύτερη.

Παραθέτουμε λοιπόν τα αποτελέσματα από την επεξεργασία του κυκλώματος S1488_vhdl_netlist.vhdl το οποίο αποτελείται από 513 στοιχεία και 9 διαφορετικά επίπεδα.

gate	gate_delay	gate_level	Cout	input in
AOI21_X2	0.023542	6	-	0
NOR2_X2	0.020353	5	0.001107	1
MUX2_X2	0.068949	4	0.000998	1
NOR2_X2	0.016833	3	0.000924	1
INV_X8	0.009919	2	0.005308	5
NOR2_X4	0.017406	1	0.002917	1
INV_X32	0.005316	0	0.003082	2

Πίνακας 4.6 κρίσιμο μονοπάτι του S1488_vhdl_netlist.vhdl

Παρατηρούμε ότι το κύκλωμα μας αποτελείται από μία πύλη τύπου **INV_X8** το οποίο εκ πρώτης όψεως δε δείχνει αναμενόμενο. Εν τούτοις η έξοδος της εν λόγω πύλης αποτελεί είσοδο σε 5 άλλα στοιχεία. Κάτι το οποίο αυξάνει αισθητά την εξωτερική της χωρητικότητα. Η εξωτερική χωρητικότητα της συγκεκριμένης πύλης είναι κατά πολύ μεγαλύτερη των υπολοίπων πυλών κάτι το οποίο έχει άμεση επίδραση στην αύξηση της καθυστέρησης αυτής, η οποία πλέον γίνεται αρκετά μεγάλη και πλησιάζει τους χρόνους όλων των υπολοίπων, κάτι το οποίο την καθιστά μέρος του κρίσιμου μονοπατιού. Αξίζει να παρατηρήσουμε και τον **INV_X32** ο οποίος αποτελεί είσοδο για δυο πύλες, σε αντίθεση με όλες τις υπόλοιπες πύλες του κρίσιμου μονοπατιού των οποίων οι έξοδοι συνδέονται με μία μόλις πύλη. Γεγονός που είναι αρκετό για να αποδώσει στη πύλη την δεύτερη υψηλότερη εξωτερική χωρητικότητα και ικανή καθυστέρηση ώστε να την συμπεριλάβει στο κρίσιμο μονοπάτι.

4.3 Χρονοι επεξεργασίας

4.3.1 Χρόνοι παραγωγής αποτελεσμάτων

Θα παραθέσουμε τους χρόνους που χρειάστηκαν για την επεξεργασία των δεδομένων αρχείων εισόδου και την παραγωγή των αντίστοιχων αποτελεσμάτων. Εδώ να σημειώσουμε πως η υλοποίηση του προγράμματος μας, η εκτέλεση και η παραγωγή των τελικών αποτελεσμάτων πραγματοποιήθηκαν εξ' ολοκλήρου σε σύστημα με επεξεργαστή **Intel Core 2 6400** με συχνότητα 2.13GHz και 1GB μνήμη **RAM**. Πρόκειται δηλαδή για ένα σύστημα όχι απαρχαιωμένο, αλλά σίγουρα αρκετά πίσω σε σχέση με τις εξελίξεις στον τομέα των Υπολογιστών. Ως αποτέλεσμα το πρόγραμμα μας μπορεί να επιδείξει ακόμη καλύτερους χρονισμούς σε ένα νεότερο σύστημα. Αν αναλογιστούμε φυσικά ότι η πλήρης μορφή του (θα αναφερθούμε παρακάτω αναλυτικά σχετικώς) αποτελεί εκτός από ερευνητικής φύσης ένα προϊόν επίσης εμπορικό και άρα οι χρήστες του θα έχουν τη δυνατότητα χρήσης τεχνολικά σύγχρονων συστημάτων, οι χρονισμοί βελτιώνονται ακόμη περισσότερο και κρίνονται αρκετά ικανοποιητικοί.

Για την επεξεργασία ενός δεδομένου ολοκληρωμένου κυκλώματος, την ανάγνωση της **Standard Cell Library** και την εύρεση του κρίσιμου μονοπατιού βάση της διαδικασίας που εξηγήθηκε στα προηγούμενα παρατίθενται ενδεικτικά οι παρακάτω χρόνοι.

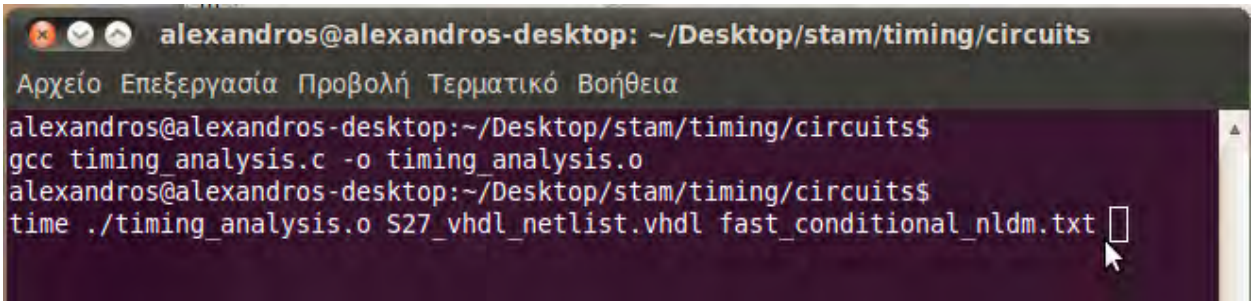
πλήθος πυλών	χρόνος σε sec
19	0.161
57	0.163
100	0.164
202	0.239
512	0.183
1500	0.286
2039	0.361
3874	0.878
5149	1.892
7164	4.634

Πίνακας 4.7 χρόνοι επεξεργασίας κυκλωμάτων ανάλογα με τον αντίστοιχο αριθμό πυλών

Εικόνα 4.7 αντίστοιχο γράφημα όπου παρουσιάζεται η εκθετική αύξηση του χρόνου επεξεργασίας

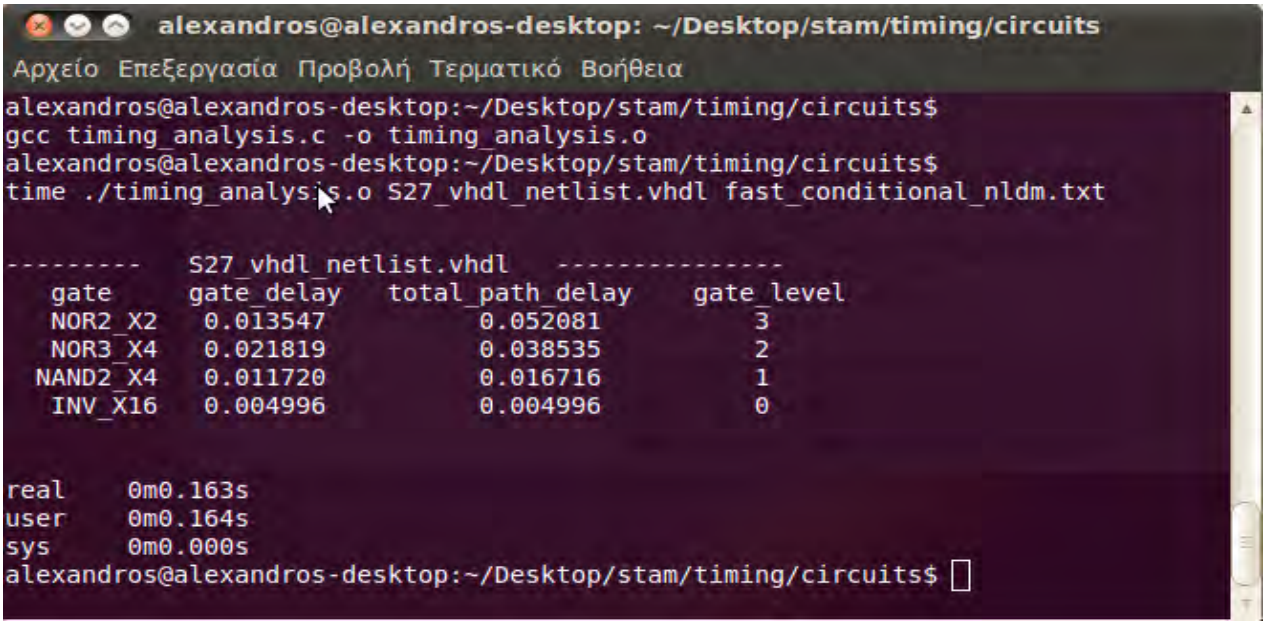
Παρατηρούμε λοιπόν μία εκθετική αύξηση του απαιτούμενου χρόνου της επεξεργασίας ενός κυκλώματος, με την αύξηση του αριθμού των πυλών του τελευταίου. Κάτι τέτοιο φυσικά είναι απολύτως αναμενόμενο αφού περισσότερες πύλες (εκτός από περισσότεροι υπολογισμοί χωρητικοτήτων και καθυστερήσεων), σημαίνει περισσότερα επίπεδα πυλών και συνέπεια αυτού, εκθετική αύξηση στο πλήθος των μονοπατιών για επεξεργασία, ώστε να εξαχθεί το κρίσιμο μονοπάτι.

Για την εξαγωγή των χρονικών απαιτήσεων μεταγλωττίσαμε κανονικά το πρόγραμμα μας και και κατά την εκτέλεση συμπεριλάβαμε κανονικά πριν την εντολή αυτής της εντολή **time** του **UNIX**.



Εικόνα 4.8 διαδικασία εμφάνισης χρονικής διάρκειας εκτέλεσης

Τα αποτελέσματα αυτής της εντολής ήταν ορατά στην κονσόλα μετά την ολοκλήρωση της εκτέλεσης του προγράμματος μας.



Εικόνα 4.9 εκτύπωση χρονικής διάρκειας εκτέλεσης στην κονσόλα

Το πρόγραμμα δύναται να βελτιωθεί ελαφρώς, αν η επεξεργασία της **Standard Cell Library** πραγματοποιούνταν μόλις μία φορά και τα δεδομένα αυτής αποθηκεύονταν στη μνήμη και για την επεξεργασία του επόμενου κυκλώματος, δίχως να απαιτείται η εκ νέου ανάγνωση αυτής. Εξάλλου η τελευταία αποτελεί ένα αρχείο εισόδου 80 000 γραμμών όπου περιγράφονται όλα τα στοιχεία από τα οποία μπορεί να αποτελείται το πρόγραμμα μας όπως **AND**, **NAND**, **OR**, **NOR**, **AOI**, **OAI** πύλες διαφορετικών ταχυτήτων αλλά και αριθμού εισόδων, πύλες **NOT**, **buffers**, **flip-flops** και αρκετά ακόμη στοιχεία. Απαιτείται λοιπόν η ανάγνωση εκατοντάδων χιλιάδων μεταβλήτων και η αποθήκευση αυτών.

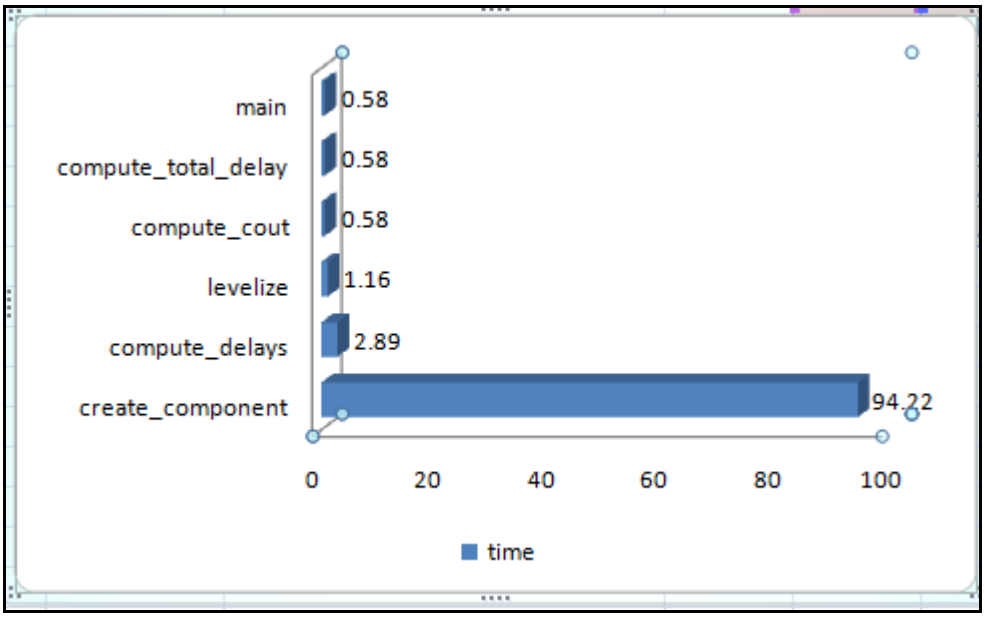
4.3.2 Στατιστικές Παρατηρήσεις επί των Χρονισμών

Για να εξάγουμε συμπεράσματα σχετικά με την κατανάλωση των πόρων του συστήματος μας (π.χ. δέσμευση του επεξεργαστή) χρησιμοποιήσαμε την εντολή **gprof** του **UNIX**. Η εν λόγω εντολή δημιούργησε ένα αρχείο με στατιστικές αναλύσεις επί των συναρτήσεων που επεξεργάστηκαν το κύκλωμα το οποίο δώσαμε ως είσοδο στο πρόγραμμα μας. Παραθέτουμε ενδεικτικά τις πληροφορίες που προέκυψαν από την από την επεξεργασία του αρχείου S38417_vhdl_netlist.vhdl.

time	cumulative seconds	self seconds	calls	self msec/call	total msec/call	name
94.22	1.63	1.63	7164	0.23	0.23	create_component
2.89	1.68	0.05	1	50	50	compute_delays
1.16	1.7	0.02	1	20	20	levelize
0.58	1.71	0.01	1	10	10	compute_cout
0.58	1.72	0.01	1	10	10	compute_total_delay
0.58	1.73	0.01				main
0	1.73	0	86072	0	0	interpolation
0	1.73	0	21518	0	0	find_elements_for_interpolation
0	1.73	0	5774	0	0	create_sig_list
0	1.73	0	3012	0	0	read_tim_arrays
0	1.73	0	2895	0	0	substring
0	1.73	0	1524	0	0	create_out_list
0	1.73	0	1524	0	0	create_pr_in_list
0	1.73	0	660	0	0	read_tim_data
0	1.73	0	388	0	0	read_inp_pin
0	1.73	0	186	0	0	read_tim_data_inp
0	1.73	0	128	0	0	read_module
0	1.73	0	127	0	0	read_pin_out
0	1.73	0	53	0	0	read_tablate

Πίνακας 4.8 αποτελέσματα στατιστικής ανάλυσης με την εντολή **gprof**

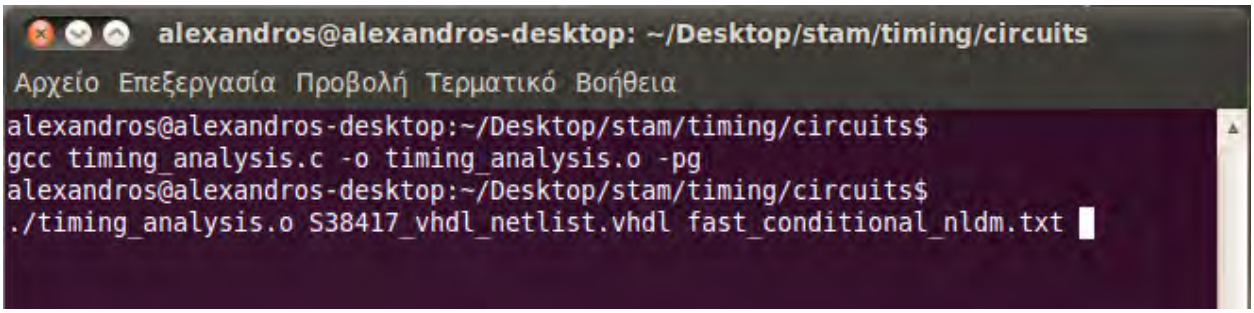
Παρατηρούμε ότι το συντριπτικό ποσοστό του χρόνου εκτέλεσης καταναλώθηκε από την **create_component( )** κάτι απολύτως φυσιολογικό για ένα τόσο μεγάλο αρχείου εισόδου, καθώς η παραπάνω συνάρτηση ήταν υπεύθυνη για τον ορισμό των κατάλληλων δομών για κάθε πύλη του κυκλώματος και την σύνδεση όλων αυτών μεταξύ τους. Ουσιώδη καθυστέρηση προσέθεσαν επίσης στο πρόγραμμα μας η συναρτήσεις **compute_delays( )** η οποία ήταν υπεύθυνη για τον προσδιορισμό της καθυστέρησης κάθε πύλης και η **levelize( )** η οποία ανακάλυπτε το επίπεδο στο οποίο συγκαταλέγονταν κάθε πύλη του κυκλώματος, επίσης αναμενόμενο αν αναλογιστούμε τις πολλές επαναλαμβανόμενες πράξεις επί κάθε πύλης εως ότου υπολογιστούν οι αντίστοιχες τιμές. Επίσης στοιχειωδώς επιβάρυναν το πρόγραμμα μας η **compute_cout ( )** και η **compute_total_delay( )** οι οποίες ήταν υπεύθυνες για τον υπολογισμό της χωρητικότητας της κάθε πύλης και την εύρεση του κρίσιμου μονοπατιού και την καθυστέρηση αυτού αντίστοιχα.



Εικόνα 4.10 δέσμευση του επεξεργαστή από τις συνάρτησης του προγράμματος

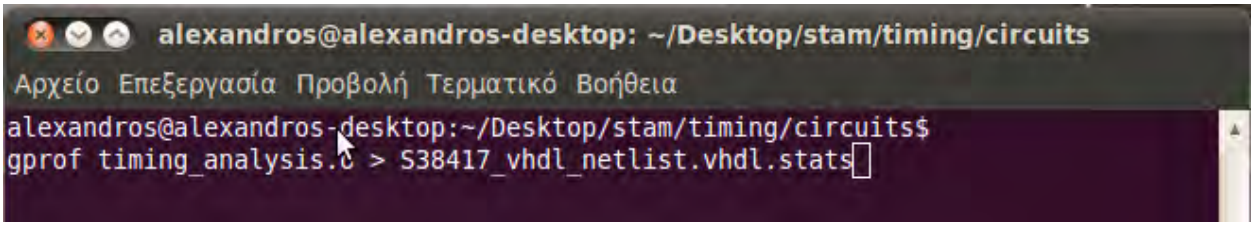
Τέλος οι συναρτήσεις οι οποίες ήταν υπεύθυνες για την επεξεργασία και αποθήκευση των τιμών της **Standard Cell Library** δεν επιβάρυναν σχεδόν καθόλου το πρόγραμμα μας, παρά το μεγάλο μέγεθος του συγκεκριμένου αρχείου. Φυσικά κάτι τέτοιο δεν ίσχυε για την επεξεργασία σχετικά μικρών αρχείων εισόδου, όπου εκεί η ανάγνωση της βιβλιοθήκης ήταν η απαιτητική δουλειά και όχι και τόσο η εκτέλεση των υπολογισμών.

Για την εξαγωγή των παραπάνω στοιχείων μεταγλωττίσαμε κανονικά το πρόγραμμα μας προσθέτοντας απλώς στο τέλος της αντίστοιχης εντολής το **flag** ***–pg*** , και εκτελέσαμε κανονικά τον κώδικα μας.



Εικόνα 4.11 διαδικασία μεταγλώττισης και εκτέλεσης για την εξαγωγή στατιστικών συμπερασμάτων

Έπειτα για τη δημιουργία ενός αρχείου στο οποίο αποθηκεύτηκε η παραπάνω στατιστική ανάλυση εκτελέσαμε την εντολή **gprof** ακολουθούμενη από το όνομα του εκτελέσιμου αρχείου, και το όνομα του αρχείου στο οποίο αποθηκευτήκαν οι παραπάνω πληροφορίες ως ακολούθως:



Εικόνα 4.12 διαδικασία δημιουργίας αρχείου στατιστικών συμπερασμάτων

Κεφάλαιο 5. Μελλοντικές Κατευθύνσεις

5.1 Μη Πλήρης Προσπάθεια

Η δουλειά η οποία πραγματοποιήθηκε δεν είναι ολοκληρωμένη. Αποτελεί απλώς μία προσπάθεια πάντα στα πλαίσια μίας πτυχιακής εργασίας εξάμηνης προβλεπόμενης διάρκειας. Πιο σωστά θα μπορούσαμε να υποστηρίξουμε πως η παρούσα εργασία αποτελεί ένα σωστό πρώτο βήμα στον τομέα της *αναλυσης χρονισμού* ολοκληρωμένων κυκλωμάτων. Η γρήγορη και εύστοχη εύρεση του κρίσιμου μονοπατιού, αποτελεί την πρώτη και βασικότερη προϋπόθεση για την επεξεργασία και βελτιστοποίηση κάθε ολοκληρωμένου κυκλώματος. Η παραπάνω δουλειά λοιπόν μπορεί να χρησιμοποιηθεί ως δευτερεύον βοηθητικό πρόγραμμα για κάποιον σχεδιαστή ολοκληρωμένων κυκλωμάτων, καθώς παρέχει υπολογισμούς καθυστερήσεων και ως αποτέλεσμα μαρτυρά τις αδυναμίες του εκαστοτε κυκλώματος και δείχνει πως μπορεί το τελευταίο να γίνει γρηγορότερο.

5.2 Επιπλέον Βελτιστοποιήσεις

Διάφορες βελτιστοποιήσεις θα μπορούσαν να γίνουν στην παρούσα εργασία. Για παράδειγμα κατά τη συγγραφή του κώδικα δε λάβαμε καθόλου υπόψη μας τις παρασιτικές χωρητικότητες των αγωγών διασύνδεσης τον λογικών πυλών δίχως αυτο να επιδρά σημαντικά στο τελικό αποτέλεσμα καθώς το κρίσιμο μονοπάτι στη συντριπτική πλειοψηφία των αρχείων εισόδου θα παράμενε το ίδιο. Επιπλέον διάφορες προσθήκες θα μπορούσαν να γίνουν ούτως ώστε τα αποτελέσματα μας να παράγονται ακόμη πιο γρήγορα. Για παράδειγμα διαφοροποιημένες ακόμη πιο εύστοχα ορισμένες δομές δεδομένων, ή διαφορετικοί αλγόριθμοι που θα καθόριζαν γρηγορότερα τη συνδεσμολογία του κυκλώματος ή θα υπολόγιζαν χωρητικότητες και καθυστερήσεις. Επίσης θα μπορούσαμε να εκτελέσουμε λιγότερους υπολογισμούς όπως να απορρίπταμε εξαρχής πολύ μικρά μονοπάτια καθώς η πιθανότητα κάποιο εξ αυτών να αποτελεί το κρίσιμο μονοπάτι ενός δεδομένου κυκλώματος είναι σχεδόν μηδενική.

Φυσικά μιλάμε για μια διαφορετική υλοποίηση ακόμη πιο γρήγορη και εξελιγμένη από την παρούσα, χωρίς όμως να υπονοούμε κάτι για την τρέχουσα δουλειά. Εξάλλου το αντικείμενο της παρούσας εργασίας είναι σε μεγάλο μέρος καθαρά προγραμματιστικό και ο προγραμματισμός δεν είναι παρά σχετικός-υποκειμενικός καθώς οποιοδήποτε πρόβλημα

δύναται να αντιμετωπιστεί από μία πληθώρα διαφορετικών προσεγγίσεων. Προφανώς και δεν μπορούμε να χαρακτηρίσουμε αυτή τη λύση βέλτιστη καθώς ενδέχεται να παρουσιαστεί κάποια άλλη εκδοχή επίλυσης του προβλήματος μας, καλύτερη της δικής μας σε απαιτήσεις σε χρόνο και χώρο. Για τον ίδιο όμως λόγο ούτε και η νέα λύση θα χαρακτηρίζεται βέλτιστη κ.ο.κ., καθώς πάντα μία δουλειά θα μπορεί να ανέχθεί επιπλέον βελτιστοποιήσεις.

5.3 Επεκτάσεις

Η υλοποίηση της παρούσας εργασίας μπορεί να εφαρμοστεί κατά το **Placement** ενός ολοκληρωμένου κυκλώματος δηλαδή κατά την τοποθέτηση των στοιχείων αυτού στην επιφάνεια του πυρήνα του αντίστοιχου **chip**. Η εν λόγω τοποθέτηση οφείλει να γίνει κατά τέτοιο τρόπο ώστε να ελαχιστοποιούνται διάφοροι παράγοντες όπως:

1) το συνολικό μήκος των αγωγών διασύνδεσης μεταξύ των στοιχείων
2) η συνολική καθυστέρηση του κυκλώματος, δηλαδή η καθυστέρηση κατά μήκος του κρίσιμου μονοπατιού να μην υπερβαίνει μια δεδομένη τιμή.
3) Η κατανάλωση ισχύος του κυκλώματος, δηλαδή να τοποθετηθούν τα στοιχεία κατά τέτοιο τρόπο ώστε να μειωθεί η κατανάλωση ισχύος και να εξομαλυνθούν οι μεγάλες θερμοκρασιακές διαφορές στην επιφάνεια του **chip**.

Επιπλέον αξία θα είχε η επέκταση του παρόντος προγράμματος ώστε να είναι σε θέση να επεξεργάζεται δεδομένα αρχεία εισόδου από περισσότερες γλώσσες περιγραφής υλικού (π.χ. **verilog**). Μια δουλειά που δεν φαντάζει ιδιαίτερα επίπονη, καθώς απαιτεί απλώς την τροποποίηση-προσαρμογη της ήδη υπάρχουσας, στις ιδιαιτερότητες και το συντακτικό μιας άλλης γλώσσας.

Τέτοιες αλλαγές θα καθιστούσαν την εργασίας αυτή πολυεργαλιο επεξεργασίας οποιουδήποτε ολοκληρωμένου κυκλώματος σε οποιαδήποτε γλώσσα και αν περιγράφεται.

5.4 Επιστημονική προοπτική

Με τις κατάλληλες βελτιστοποιήσεις και επεκτάσεις λοιπόν το πρόγραμμα της παρούσας εργασίας μπορεί να μετατραπεί σε ένα καθαρά εμπορικό προϊόν με πολλες εφαρμογές στη βιομηχανία του hardware για την ανάλυση χρονισμού ολοκληρωμένων κυκλωμάτων. Εξάλλου η συνεχής πρόοδος της τεχνολογίας στους συγκεκριμένους τομείς, επιβάλει τη δημιουργία όλο και γρηγορότερων κυκλωμάτων.

Αν αναλογιστεί κανείς το πλήθος των **transistor** σε ένα σύγχρονο επεξεργαστή που ξεπερνάει τα 2 δις η μέγιστη πρόκληση είναι η τοποθέτηση αυτών στην αντίστοιχη πλακέτα κατά βέλτιστο τρόπο τηρουμένων ορισμένων προδιαγραφών.

Εικόνα 5.1 Intel Core I7

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

- Static Timing Analysis for Nanometer Designs. A Practical Approach (J. Bhasker, Rakesh Chadha)
- CMOS VLSI Design (Neil Waste, David Harris)

Ιστότοποι

- http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html#SEC1
- http://linux.about.com/library/cmd/blcmdl1_time.htm
- http://en.wikipedia.org/wiki/Static_timing_analysis
- http://wiki.answers.com/Q/Difference_between_the_dynamic_timing_analysis_and_static_timing_analysis
- http://asic-soc.blogspot.com/2008/08/dynamic-vs-static-timing-analysis.html
- http://asic-soc.blogspot.com/2009/06/timing-paths.html
- http://www.nangate.com/
- http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html
- http://en.wikipedia.org/wiki/Time_%28Unix%29
- http://wiki.answers.com/Q/What_is_the_advantages_of_c_language
- http://en.kioskea.net/forum/affich-21243-advantages-and-disadvantages-of-c-language