ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνικές θωράκισης από την ακτινοβολία στη συνδυαστική λογική των σύγχρονων κυκλωμάτων

Διπλωματική Εργασία

Γεώργιος Παπαγεωργίου

Επιβλέπων: ΣΤΑΜΟΥΛΗΣ ΓΕΩΡΓΙΟΣ

Ιούνιος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

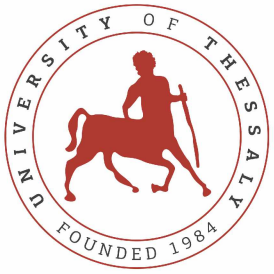
Τεχνικές θωράκισης από την ακτινοβολία στη συνδυαστική λογική των σύγχρονων κυκλωμάτων

Διπλωματική Εργασία

Γεώργιος Παπαγεωργίου

Επιβλέπων: ΣΤΑΜΟΥΛΗΣ ΓΕΩΡΓΙΟΣ

Ιούνιος 2022



UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Radiation hardening techniques in combinational logic of modern circuits

Diploma Thesis

George Papageorgiou

Supervisor: George Stamoulis

June 2022

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων **ΣΤΑΜΟΥΛΗΣ ΓΕΩΡΓΙΟΣ**
Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Μέλος **ΕΥΜΟΡΦΟΠΟΥΛΟΣ ΝΕΣΤΩΡ**
Επίκουρος Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Μέλος **ΔΑΔΑΛΙΑΡΗΣ ΑΝΤΩΝΙΟΣ**
Επίκουρος Καθηγητής, Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τους γονείς μου, των οποίων η αγάπη, η καθοδήγηση και η υποστήριξη μου έδωσαν τη δύναμη να εξερευνήσω τα ακαδημαϊκά μου όρια. Τόσο στις επιτυχίες μου όσο και στις αποτυχίες μου, μπόρεσαν να δώσουν παραδείγματα από το παρελθόν τους που αντικατόπτριζαν (κάτω από τα μάτια) το δικό μου. Αυτή η διατριβή είναι τόσο δικό τους επίτευγμα όσο και δικό μου.

Θα ήθελα επίσης να ευχαριστήσω τον σύμβουλό μου, Δρ. Σταμούλη Γεώργιο. Ήταν χαρά να συνεργάζομαι με κάποιον που μπορούσε να προσφέρει συμβουλές για οποιοδήποτε θέμα χωρίς κρίση ή εγωισμό. Η βελτίωσή μου ως ερευνητή είναι άμεσο αποτέλεσμα των οδηγιών του. Ευχαριστώ θερμά τον Γεώργιο Παλιαρούτη και τον Πελοπίδα Τσουμάνη για τις δύσκολες ερωτήσεις και την απαίτηση εμπεριστατωμένων απαντήσεων καθ' όλη τη διάρκεια της διατριβής μου. Θέλω να ευχαριστήσω τα μέλη της επιτροπής μου, τον Δρ. Σταμούλη Γεώργιο, τον Δρ. Ευμορφόπουλο Νέστωρ, τον Δρ. Δαδαλιάρη Αντώνη για την ερευνητική τους καθοδήγηση.

Επίσης, θα ήθελα να ευχαριστήσω τον Άγγελο Αγγέλη, Ιωάννη Κανελλόπουλο και όλους τους συναδέλφους μου, γιατί με άφησαν να συζητάμε πολλές ώρες για θέματα αναφορικά με την Διπλωματική μου εργασία, όταν μπορούσαμε συνομιλούσε για πιο ανάλαφρα θέματα. Τέλος, θα ήθελα να ευχαριστήσω το Πανεπιστήμιο Θεσσαλίας και τους καθηγητές του για τις γνώσεις που μου προσέφεραν καθ'όλη την ακαδημαική μου θητεία που με βοήθησε στη διατριβή μου.

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

«Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής».

Ο/Η Δηλών/ούσα

Γεώργιος Παπαγεωργίου

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

George Papageorgiou

Διπλωματική Εργασία

Τεχνικές θωράκισης από την ακτινοβολία στη συνδυαστική λογική των σύγχρονων κυκλωμάτων

Γεώργιος Παπαγεωργίου

Περίληψη

Τα ενσωματωμένα κυκλώματα, πλέον χρησιμοποιούνται σε κάθε ηλεκτρονικό εξοπλισμό (αυτοκίνητο, υπολογιστής, δορυφόρος, κινητό τηλέφωνο). Ένα ενσωματωμένο κύκλωμα αποτελείται απο λογικές πύλες, όπου ανάλογα το πλήθος τους, το κύκλωμα γίνεται πιο σύνθετο. Οι πύλες συνδέονται μεταξύ τους, η κάθε είσοδος μιας πύλης μπορεί να είναι η έξοδος μιας άλλης πύλης ή είσοδος του κυκλώματος, για να πραγματοποιήσουν κάποιες λογικές πράξεις. Η ανάγκη σήμερα για περισσότερα κυκλώματα μεγαλώνει. Μαζί με την αύξηση της ζήτησης των ενσωματωμένων, αυξάνεται και η ζήτηση για ενσωματωμένα κυκλώματα, τα οποία χρησιμοποιούνται σε περιβάλλοντα με ακτινοβολία (ιοντίζοντα). Τέτοια περιβάλλοντα μπορεί να είναι τα αεροπλάνα, διαστημόπλοια, πυρηνικοί σταθμοί παραγωγής ενέργειας.

Στα κυκλώματα υπάρχει πιθανότητα να προκληθούν σφάλματα τα οποία είναι προσωρινά και όχι καταστροφικά για το κύκλωμα. Τα σφάλματα τα οποία είναι προσωρινά και όχι καταστροφικά για το κύκλωμα, ονομάζονται μεταβατικά σφάλματα ή Soft Errors. Αυτού του είδους τα σφάλματα δημιουργούνται από την ακτινοβολία:(1) απο σωματίδια Αλφα, (2) κοσμικές ακτίνες που παράγουν πρωτόνια και νετρόνια ή (3) θερμικά νετρόνια. Αυτά τα χτυπήματα μπορούν να αλλάξουν τη λογική κατάσταση (είτε από λογικό 1 σε λογικό 0, είτε από λογικό 0 σε λογικό 1) στην έξοδο της πύλης που συμβαίνουν.

Στόχος της διπλωματικής εργασίας είναι η υλοποίηση ενός προγράμματος προσομοίωσης ενός ενσωματωμένου κυκλώματος, με το οποίο θα μπορούμε να προσομοιώσουμε την εκτέλεση ενός ολοκληρωμένου. Το πρόγραμμα θα μας επιτρέπει (α) να διαβάζουμε ένα κύκλωμα απο ένα αρχείο verilog και να δημιουργούμε ένα πιο ανθεκτικό, (β) να προσομοιώνουμε την εκτέλεση του κυκλώματος και να βλέπουμε τα αποτελέσματα των τιμών των εξόδων κάθε πύλης, (γ) να τροποποιούμε τις πιο ευάλωτες πύλες στην ακτινοβολία προκειμένου να γίνουν πιο ανθεκτικές σε αυτή και (δ) δημιουργώντας κατ' επέκταση ένα ανανεωμένο κύκλωμα λιγότερο ευάλωτο στην ακτινοβολία. Για την τροποποίηση των πιο ευαίσθητων πυλών (sensitive gates) θα δημιουργηθεί ένας υβριδικός αλγόριθμος, που θα δίνει το κατάλληλο masking

στις ευάλωτες πύλες.

Λέξεις Κλειδιά

Ενσωματωμένα κυκλώματα, Ενσωματωμένα κυκλώματα σε ιοντίζον περιβάλλον, Προσομοίωση ενσωματωμένου κυκλώματος, Τεχνικές μείωσης του SER, Logical Masking, Electrical Masking, Soft Error Rate, Mitigation Techniques, sensitive gates

Diploma Thesis

Radiation hardening techniques in combinational logic of modern circuits

George Papageorgiou

Abstract

The integrated circuits are now used in all electronic equipment (car, computer, satellite, mobile phone). An integrated circuit consists of logic gates and becomes more complex due to their number. The gates are interconnected, and each input of one gate can be the output of another gate or the input of the circuit, to perform some logical operations. The need today for more integrated circuits is growing, and there is also an increase in the design of more resistant integrated circuits since many of them are utilized in environments with radiation (ionizing). Such environments can be airplanes, spaceships, nuclear power plants.

Errors can be created in these circuits. Errors that are temporary are called soft errors or SERs. These errors are generated by radiation as a result of particle shocks caused by: (1) Alpha particles, (2) cosmic rays that produce protons and neutrons, or (3) thermal neutrons. These blows can flip bits at the gate output where they fall.

The aim of the dissertation is to implement a simulation program of an integrated circuit, with which we can simulate the execution of an integrated. The program will allow us (a) to read a circuit from a Verilog file and create it, (b) to simulate the execution of the circuit and see the results of its output values, (c) to find the more vulnerable gates to radiation and modify them to make them more resistant to radiation and (d) create the updated circuit in a Verilog file which will be more resistant to radiation. To modify the gates, which are vulnerable to radiation, a hybrid algorithm will be created, which will give the appropriate masking to the vulnerable gates.

Keywords

Integrated circuits, Integrated circuits in ionizing environment, Integrated circuit simulation, SER reduction techniques, Logical Masking, Electrical Masking, Soft Error Rate, Mitigation Techniques

Πίνακας περιεχομένων

Κατάλογος σχημάτων

Κατάλογος πινάκων

Συντομογραφίες

βλπ	βλέπε
κ.λπ.	και λοιπά
κ.ο.κ	και ούτω καθεξής
SER	Soft Error Rate
IC	Integrated Circuit
BPF	Band Pass Filter
SET	single-event transient
SEU	single-event upset
TMR	triple modular redundancy
CMOS	complementary metal oxide semiconductor
Vdd	voltage drain-to-drain
GND	ground
MBUs	Multiple Bit Upsets
METs	Multiple Event Transients.

Κεφάλαιο 1

Εισαγωγή

Ένα μεταβατικό σφάλμα είναι μια προσωρινή διαταραχή αλλάζοντας δεδομένα σε ένα κύκλωμα. Τα σφάλματα που μπορούν να αλλάξουν δεδομένα χωρίζονται σε δύο κατηγορίες, τα μεταβλητά και τα μόνιμα σφάλματα. Τα μεταβλητά σφάλματα χωρίζονται σε επίπεδο chip και συστήματος.

Τα σφάλματα σε επίπεδο chip είναι συμβάντα στα οποία ένα ραδιενεργό στοιχείο διασπάται στο chip διασπάται, απελευθερώνοντας ένα πρωτόνιο. Εάν το πρωτόνιο χτυπήσει ένα κελί μνήμης, μπορεί να αλλάξει την τιμή που περιέχει το κελί. Ακόμη και οι κοσμικές ακτίνες μπορούν να προκαλέσουν αυτό το φαινόμενο.

1.1 Αντικείμενο της διπλωματικής

Το αντικείμενο της διπλωματικής εργασίας είναι η μελέτη και η μείωση των μεταβατικών σφαλμάτων για κυκλώματα τα οποία θα βρίσκονται σε ραδιενεργά περιβάλλοντα. Πιο συγκεκριμένα, δημιουργήθηκε ένα πρόγραμμα που προσομοιώνει ένα κύκλωμα το οποίο γίνεται να δεχτεί ακτινοβολία. Στην συνέχεια βρίσκουμε τις πιθανότητες απώλειας δεδομένων κάθε πύλης του κυκλώματος και εφαρμόζοντας αλγορίθμους γίνονται τροποποιήσεις στο κύκλωμα για να γίνει πιο ανθεκτικό στην ακτινοβολία, δίχως σημαντική αύξηση του μεγέθους του, της κατανάλωσής του καθώς και μείωσης της απόδοσής του, δημιουργώντας ένα νέο πιο ανθεκτικό κύκλωμα.

1.1.1 Συνεισφορά

Κατά την διάρκεια της διπλωματικής εργασίας δημιουργήθηκε το πρόγραμμα προσομοίωσης και τροποποίησης του κυκλώματος.

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήθηκαν κάποια κυκλώματα υλοποιημένα σε Verilog από τη γνωστή σουίτα των ακολουθιακών κυκλωμάτων αναφοράς ISCAS '89 (s27, s298, s344, s400 κλπ)

2. Δημιουργήθηκαν και υλοποιήθηκαν αλγόριθμοι δημιουργίας και προσομοίωσης του κυκλώματος

3. Προσομοιώθηκαν αρχικά κυκλώματα Verilog και αξιολογήθηκαν

4. Δημιουργήθηκαν νέα, με ισοδύναμη λειτουργία, κυκλώματα πιο ανθεκτικά σε ακτινοβολία.

5. Αξιολογήθηκε η επίδοση των αλγορίθμων και βρέθηκε πόσο ανθεκτικότερο γίνεται το κάθε κύκλωμα.

6. Δημιουργήθηκε ένας αλγόριθμος τροποποίησης των πιο ευάλωτων, στην ακτινοβολία, λογικών πυλών.

1.2 Οργάνωση του τόμου

Εργασίες σχετικές με το αντικείμενο της διπλωματικής παρουσιάζονται στο Κεφάλαιο 2. Το Κεφάλαιο 3 συζητά θέματα μοντελοποίησης των κυκλωμάτων και του αντίκτυπου της ακτινοβολίας. Στο Κεφάλαιο 4 αναπτύσσουμε το εργαλείο που δημιουργήθηκε για το ερευνητικό κομμάτι της εργασίας. Παρουσιάζουμε τους αλγορίθμους τροποποίησης των πυλών, την εκτέλεση και τον ορισμό του κυκλώματος σε επίπεδα. Στο Κεφάλαιο 5 θα δούμε βοηθητικές μεθόδους και patterns που χρησιμοποιήθηκαν στον κώδικα. Τέλος το Κεφάλαιο 6 παραθέτει τα πειραματικά αποτελέσματα με στατιστικά για τα κυκλώματα πριν και μετά την βελτίωση των πυλών.

Κεφάλαιο 2

Επιδράσεις της Ακτινοβολίας στα Ηλεκτρονικά

2.1 Εισαγωγή

Η λειτουργία των ενσωματωμένων μπορεί να επηρεαστεί από το περιβάλλον στο οποίο λειτουργεί. Στο διάστημα, η ακτινοβολία από τον ήλιο και οι κοσμικές ακτίνες μπορεί να προκαλέσουν προσωρινή ή μόνιμη βλάβη μιας συσκευής. Σε επίγεια περιβάλλοντα, αυτές οι πηγές ακτινοβολίας εξακολουθούν να επηρεάζουν τη λειτουργία της συσκευής αλλά σε μικρότερο βαθμό. Η έρευνα, για την κατασκευή αξιόπιστων κυκλωμάτων, έχει σημασία και στα δύο περιβάλλοντα. Η επίδραση των πηγών ακτινοβολίας στη λειτουργία μιας συσκευής μπορεί να ταξινομηθεί σε τρεις τύπους βλαβών:

- Μόνιμη. Οι μόνιμες βλάβες είναι μη αναστρέψιμες δυσλειτουργίες της συσκευής.
- Διακοπτόμενη. Τα διακοπτόμενα σφάλματα είναι επαναλαμβανόμενες προσωρινές δυσλειτουργίες σε έναν κόμβο του κυκλώματος που συμβαίνουν ως αποτέλεσμα αστάθειας στο υλικό.
- Παροδική. Τα παροδικά σφάλματα είναι προσωρινές δυσλειτουργίες που προκαλούνται από περιβαλλοντικές συνθήκες.

Σε περιπτώσεις όπου ένα λάθος οδηγεί σε μια παρατηρήσιμη δυσλειτουργία στην έξοδο, το λάθος αναφέρεται ως σφάλμα. Για παράδειγμα, ένα εσφαλμένο σήμα που εμφανίζεται στην έξοδο ενός κυκλώματος αλλά όχι τη στιγμή που μια εφαρμογή χρησιμοποιεί αυτό το

σήμα θα θεωρηθεί σφάλμα σε επίπεδο κυκλώματος αλλά όχι σε επίπεδο εφαρμογής. Η περίπτωση όπου ένα σφάλμα δεν μεταδίδεται στην έξοδο ή εμποδίζεται η διάδοση του εντός κυκλώματος , είναι ένα φαινόμενο γνωστό ως masking. Αυτή η εργασία επικεντρώνεται στα προσωρινά σφάλματα, τις παρατηρήσιμες δυσλειτουργίες στην έξοδο ενός κυκλώματος που προκαλούνται από παλμούς παροδικού ρεύματος που προκαλούνται από την ακτινοβολία.

2.2 Κατηγορίες του masking στα κυκλώματα

Πριν ξεκινήσουμε να αναλύσουμε το είδος του masking που ερευνήθηκε και επιχειρήθηκε να βελτιωθεί στην εργασία, πρέπει να δούμε πόσων ειδών masking μηχανισμοί υπάρχουν και πως χωρίζουμε τις κατηγορίες. Μπορούμε να χωρίσουμε το masking σε 3 βασικές κατηγορίες, οι οποίες να βασίζονται στον τρόπο με τον οποίο θα πραγματοποιηθεί το masking.

- Logical Masking. Ένα SET που δεν βρίσκεται σε ευαισθητοποιημένη διαδρομή από την τοποθεσία από όπου προέρχεται είναι λογικά καλυμμένο. Μόλις ένα SET καλυφθεί λογικά, δεν έχει πλέον καμία επιρροή στη λειτουργία του κυκλώματος. Δηλαδή, τόσο το πλάτος όσο και η διάρκειά του γίνονται μηδέν.

- Electrical Masking. Ένα SET που είναι εξασθενημένο και γίνεται πολύ μικρό σε πλάτος ή διάρκεια για να μανδαλωθεί είναι ηλεκτρικά καλυμμένο. Ενώ ένα SET μπορεί να πιαστεί απο ένα flip flop εάν το εξασθενημένο πλάτος και η διάρκειά του εξακολουθούν να είναι αρκετά μεγάλα, η ηλεκτρική κάλυψη μπορεί να μειώσει τη συνολική επίδραση των SET.

- Latching-Window Masking. Ένα SET που δεν φθάνει "στην ώρα του" καλύπτεται επίσης, ανάλογα με τους χρόνους setup και holdup του κυκλώματος. Η βασική προϋπόθεση για να πιαστεί ένα SET είναι η διάρκειά του να είναι μεγαλύτερη από το άθροισμα των χρόνων setup και holdup και να φτάσει στο στοιχείο μνήμης κατά τη διάρκεια του latching window.

2.3 Μέθοδοι βασισμένες σε πλεονασμό Hardware

Οι μέθοδοι πλεονασμού του hardware βασίζονται στην προσθήκη λογικών πυλών. Η προσθήκη αυτή, μεγιστοποιεί το masking των προσωρινών σφαλμάτων. Η μέθοδος που εκμεταλ-

λεύεται το λογικό masking για την δημιουργία ανθεκτικών κυκλωμάτων στην ακτινοβολία είναι το Triple Modular Redundancy ή TMR [8]. Το TMR αποτελείται από τρία πανομοιότυπα κυκλώματα, με την ίδια λειτουργικότητα, που οδηγούν σε ένα μπλοκ λογικών πυλών το οποίο ονομάζεται voter και είναι υπεύθυνο για την προώθηση του σωστού σήματος σε περίπτωση που ενα από τα τρία κυκλώματα χτυπηθεί από κάποιο σωματίδιο. Το TMR όπως μπορούμε να συμπεράνουμε λοιπόν μειονεκτεί εξαιτίας της αύξησης της απαιτούμενης περιοχής του κυκλώματος, αλλά και της απαιτούμενης ενέργειας για να λειτουργήσει (200%). Έτσι προέκυψε η ιδέα να χρησιμοποιείται αυτή η μέθοδος μόνο στους κόμβους με την μεγαλύτερη πιθανότητα αστοχίας. Έτσι οι συγκεκριμένοι κόμβοι τριπλασιάζονται ενώ οι άλλοι ομαδοποιούνται..

Μια ακόμη πρόταση για την αύξηση του masking, είναι ο πλεονασμός σε επίπεδο τρανζίστορ. Στην δομή των τρανζίστορ έχουμε N blocks, τα οποία είναι συνδεδεμένα μεταξύ τους σε σειρά, έτσι ώστε κάθε block να περιέχει N παράλληλα τρανζίστορ. Έχει αποδειχθεί πως με την χρήση αυτής της τεχνικής, αυξάνουμε την ανοχή των σφαλμάτων του κυκλώματος και μάλιστα και σε μεγαλύτερο βαθμό σε σχέση με την χρήση του TMR ή σε επίπεδο πύλης [8].

2.4 Τεχνικές με βάση τη σύνθεση

Η τεχνική αυτή βασίζεται στην αναδόμηση του κυκλώματος. Κατά την τεχνική αυτή, δημιουργούμε ξανά το κύκλωμα για να μεγιστοποιήσουμε τις ιδιότητες των masking μηχανισμών. Το logical masking είναι ο πρωτεύων παράγοντας που πρέπει να μεγιστοποιηθεί. Μπορούμε να αυξήσουμε την λογική απόκρυψη σφαλμάτων αν εκμεταλλευτούμε τις συνθήκες που υπάρχουν ήδη στο κύκλωμα. Δύο τεχνικές χρησιμοποιούνται για τη βελτίωση της αξιοπιστίας του κυκλώματος: η επανασύνθεση με βάση το don't care-based και η τοπική επανεγγραφή.

Στην πρώτη μέθοδο, εντοπίζονται κόμβοι υψηλής πιθανότητας αστοχίας. Ένας κόμβος έχει μεγάλο αντίκτυπο εάν πολλά παρατηρήσιμα σφάλματα περνούν μέσα από αυτόν. Οι κόμβοι υψηλού κινδύνου χρησιμοποιούνται για την επιλογή περιοχών του κυκλώματος για τροποποίηση. Στις περιοχές αυτές ένας ευάλωτος κόμβος αναπαράγεται με την προσθήκη μιας ενιαίας πύλης.

Η τοπική επανεγγραφή ως μέθοδος, χρησιμοποιείται επίσης για τη βελτιστοποίηση μι-

κρών υποκυκλωμάτων για τη λήψη βελτιώσεων συνολικής περιοχής. Έχουν προταθεί αλγόριθμοι για τη βελτίωση της ανθεκτικότητας των σφαλμάτων εισόδου.Οι αλγόριθμοι αυτοί εστιάζουν στο σφάλμα εισόδου, λόγω διαδομένων αστοχιών από προηγούμενα μπλοκ. Μια ακόμα ενδιαφέρουσα πρόταση πέρα των δύο αλγορίθμων είναι ένα πλαίσιο που βασίζεται στην προσθήκη και αφαίρεση του πλεονασμού για τη μείωση του soft error rate (SER). Εκτελούμε μια σειρά από προσθήκες και αφαιρέσεις καλωδίων αναζητώντας περιττά καλώδια στο κύκλωμα [2], [5].

2.5 Τεχνικές με βάση τα φυσικά χαρακτηριστικά

Οι τεχνικές που βασίζονται στα φυσικά χαρακτηριστικά προσπαθούν να μειώσουν το SER με βάση τα φυσικά χαρακτηριστικά των πυλών για να μεγιστοποιήσουν το ηλεκτρικό masking. Η τεχνική αυτή τροποποιεί το μεγέθους της πύλης. Μειώνει το SER τροποποιώντας την αναλογία πλάτους-μήκους (W/L) των τρανζίστορ των πυλών. Το W/L ορίζεται ως η αναλογία του μεγέθους του πλάτους ως προς το μήκος ενός τρανζίστορ ημιαγωγού μεταλλικού οξειδίου αρνητικού καναλιού (nMOS). Ωστόσο, για να επιτευχθεί σημαντική βελτίωση στο SER, εισάγονται δυνητικά μεγάλα γενικά έξοδα στην περιοχή, την καθυστέρηση και την ισχύ. Τέλος μία ακόμα ιδέα που αξιολογήθηκε ήταν η επιλεκτική σκλήρυνση των ευάλωτων κόμβων στη συνδυαστική λογική, βασιζόμενη στην αντικατάσταση των πυλών δύο εισόδων με πύλες τεσσάρων εισόδων.

Κεφάλαιο 3

Μοντελοποίηση του SER

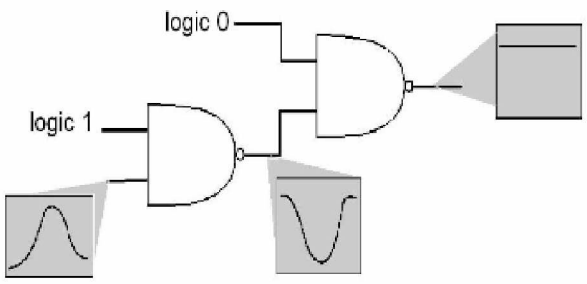
3.1 Εισαγωγή

Η ανάλυση του SER ενός κυκλώματος με σωστό τρόπο είναι ένα σημαντικό βήμα που επηρεάζει σημαντικά την μείωση του. Η ανάλυση πρέπει να γίνει με ακρίβεια και με αποτελεσματικότητα. Μέχρι σήμερα έχει γίνει αρκετή έρευνα επάνω στον τομέα μοντελοποίησης του. Στην ενότητα αυτή θα δούμε το μοντέλο στο οποίο βασίστηκε το εργαλείο.

3.2 Εκτίμηση Soft Error Rate

Τα soft errors εμφανίζονται κυρίως σε δύο τομείς του κυκλώματος. Μέσα στα κελία της μνήμης και στους κόμβους του λογικού κυκλώματος. Από αυτά όσων αφορά το πρώτο κομμάτι η έρευνα που έχει γίνει σε αυτόν τον τομέα είναι εκτενής και υπάρχουν εμπειρικά μοντέλα για τον υπολογισμό του ποσοστού soft error (sort error rate).

Στο κομμάτι που αφορά την αντίστοιχη ανάλυση στα λογικά κυκλώματα, τα πράγματα περιπλέκουν πολύ. Η μελέτη της επίδρασης του παλμού ρεύματος που δημιουργήθηκε από την επίδραση των σωματιδίων και η τοπική ανάλυση της συμπεριφοράς των στοιχείων που επηρεάστηκαν περιγράφει μόνο τη δημιουργία μιας λανθασμένης μεταβολής της κατάστασης στον συγκεκριμένο κόμβο και όχι την επίδραση που αυτό μπορεί να έχει στο ακολουθιακό κομμάτι του κυκλώματος οπότε και θα αποτελούσε ένα soft error. Ο παραγόμενος παλμός τάσης στον κόμβο εξόδου της πύλης δεν είναι απαραίτητο ότι θα οδηγήσει σε αλλοίωση της τιμής της λογικής κατάστασης ενός μανδαλωτή που μπορεί να βρίσκεται αρκετά λογικά επίπεδα μακριά. Υπάρχουν τρεις παράγοντες που μπορεί να αποτρέψουν ένα event upset να



Σχήμα 3.1: Παράδειγμα κυκλώματος με περιπτώσεις logical masking

οδηγήσει σε soft error και οι οποίοι είναι :

- Η ηλεκτρική συμπεριφορά των πυλών.
- Η λογική συμπεριφορά των πυλών.
- Ο χρόνος άφιξης της μεταβολής στα latches.

Ο πρώτος παράγοντας αφορά την συμπεριφορά της πύλης, σε ηλεκτρικό επίπεδο, που θα δεχθεί στην είσοδο της ένα στιγμιαίο σήμα. Ο δεύτερος παράγοντας που μπορεί να αποτρέψει ένα event upset να γίνει soft error είναι η λογική κατάσταση στην οποία βρίσκονται οι πύλες του κυκλώματος που πρέπει να διανύσει το event έως ότου φτάσει σε κάποιο latch. Όπως φαίνεται και στο σχήμα 3.1, ένα σφάλμα μπορεί να καλυφθεί (περίπτωση α) ή να περάσει (περίπτωση β) από μια πύλη ανάλογα με τις τιμές των υπόλοιπων εισόδων της. Τέλος ο τρίτος παράγοντας που καθορίζει αν ένα σφάλμα θα περάσει σε ένα στοιχείο μνήμης είναι η χρονική στιγμή άφιξης του παλμού στην είσοδο της καθώς πρέπει να συμπέσει με την στιγμή της δειγματοληψίας του latch. [4]

3.3 Εύρεση συνθηκών για την δημιουργία ενός soft error

Το μοντέλο αυτό χρησιμοποιήθηκε καθώς μπορεί ταυτόχρονα να ποσοτικοποιήσει το αντίκτυπο του σφάλματος καθώς και να δείξει το αντίκτυπο κάλυψης κάθε πύλης. Για την μοντελοποίηση ενός παροδικού σφαλματος που προέρχεται από μία πύλη G και θα βγεί στην έξοδο της F, μπορούν να οριστούν τα εξής βήματα:

- A: $A > Vs$ αν η σωστή έξοδος είναι 0 ή $A < Vs$ αν η σωστή έξοδος είναι 1, όπου A είναι πλάτος του σφάλματος και Vs είναι το threshold του latch.

- D: D> Tsetup + Thold, όπου D είναι η διάρκεια του σφάλματος και Tsetup και Thold είναι αντίστοιχα ο χρόνος setup και hold του latch.

- T: Η μεταβλητή t που αντιπροσωπεύει χρόνο, πρέπει να βρίσκεται σε ένα περιορισμένο διάστημα. Όπου t είναι ο χρόνος που αρχικά συνέβη το σφάλμα, tp είναι ο χρόνος καθυστέρισης μετάδοσης απο την πύλη G στην έξοδο F και T ειναι ένα event στο latch

$$t \in [T + t_{hold} - t_p - D, T - t_{setup} - t_p]$$

Σε αυτό το μοντέλο [15], η λογική και ηλεκτρική κάλυψη περιλαμβάνονται στο A και D, ενώ κάλυψη latching-window περιλαμβάνεται στο T. Τα 3 αυτά events είναι απαραίτητο να συμβούν για την δημιουργία ενός soft error. Επιπλέον για να συμβεί η συνθήκη D πρέπει να ικανοποιείται ήδη η συνθήκη A. Αν θεωρήσουμε πως η συνθήκη T είναι ομοιόμορφα κατανεμημένη, τότε η πιθανότητα να συμβεί ένα soft error μπορεί να υπολογιστεί ως:

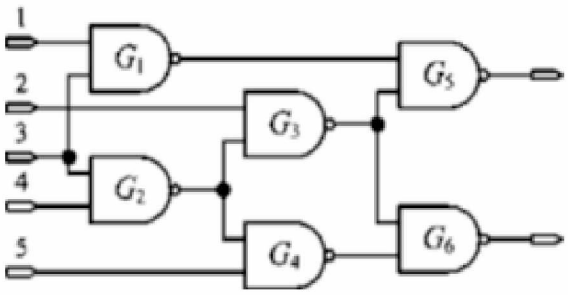
$$\begin{aligned} &P(\mathcal{A} \cap \mathcal{D} \cap \mathcal{T}) \\ &\quad = P(\mathcal{D} \cap \mathcal{T}) = P(\mathcal{T} \mid \mathcal{D}) \cdot P(\mathcal{D}) \\ &\quad = \sum_k (\mathrm{P}\,(t \in [T + t_{\text{hold}} - t_p - D, T - t_{\text{setup}} \\ &\qquad - t_p] \mid D = D_k) \cdot \mathrm{P}\,(D = D_k)) \\ &\quad = \sum_k \left(\frac{D_k - (t_{\text{setup}} + t_{\text{hold}})}{T_{\text{clk}} - d_{\text{init}}} \cdot \mathrm{P}\,(D = D_k) \right) \end{aligned}$$

όπου Dk είναι ένα σύνολο χρόνου μετάδοσης του σφάλματος, Tclk είναι η περίοδος του ρολογιού, dinit η αρχική διάρκεια ενός σφάλματος και t είναι ομοιόμορφα κατανεμημένο στο ολοκλήρωμα

$$[T, T + t_{clk} - d_{init}]$$

Για να βρούμε τις πιθανές τιμές για τη διάρκεια του σφάλματος, χρησιμοποιούμε ένα μοντέλο εξασθένησης, το οποίο εξαρτάται κυρίως από την καθυστέρηση διάδοσης της κάθε πύλης. Για να προσδιορίσουμε την πιθανότητα να έχουμε σφάλμα με διάρκεια, χρησιμοποιούμε δυαδικά διαγράμματα απόφασης (BDDs) και αλγεβρικά διαγράμματα απόφασης (ADDs). Η μεθοδολογία περιγράφεται συνοπτικά στη συνέχεια.

Ο τερματικός κόμβος ”0” του αλγεβρικού διαγράμμτος απόφασης, που σχετίζεται με μια πύλη, αντιπροσωπεύει όλες τις περιπτώσεις όπου ένα σφάλμα καλύπτεται λογικά ή ηλεκτρικά. Διαφορετικοί τερματικοί κόμβοι αντιπροσωπεύουν τις υπόλοιπες τιμές, για τη διάρ-

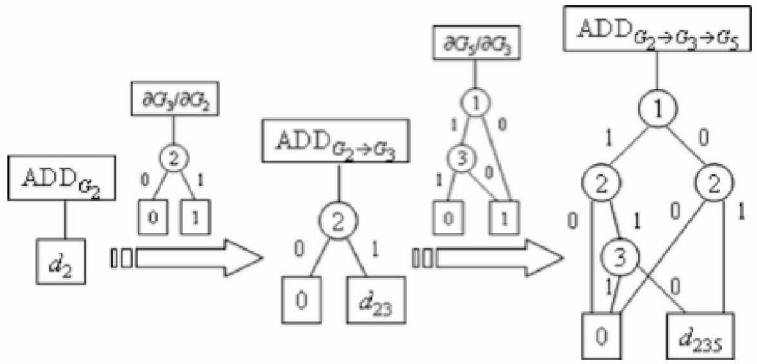


Σχήμα 3.2: Παράδειγμα κυκλώματος που θα αξιοποιήσουμε για την δημιουργία ADD

κεια ή το πλάτος μετά τη διέλευση ενός σφάλματος μέσα από μια πύλη. Η αρχική προσθήκη κάθε πύλης έχει δημιουργηθεί για το σφάλμα που προέρχεται από την πύλη αυτή.

Αποτελείται από έναν μόνο τερματικό κόμβο—αρχική τιμή διάρκειας ή πλάτους. Αυτά τα αρχικά ADD διαδίδονται σε αντίστοιχες fan-out πύλες, οι οποίες τις χρησιμοποιούν για τη δημιουργία νέων ADD με βάση το μοντέλο εξασθένησης και τα σχετικά BDD ευαισθητοποίησης.

Τα BDD ευαισθητοποίησης περιλαμβάνουν πληροφορίες σχετικά με το logical masking. Η ευαισθητοποίηση BDD μιας πύλης G στην πύλη G' είναι απλώς μια Boolean διαφορά της G' σε σχέση με το G(dG'/dG). Τα διανύσματα εισόδου που κάνουν την ευαισθητοποίηση BDD της διαδρομής G -> G' πηγαίνουν στον τερματικό κόμβο "0" και καλύπτουν δυσλειτουργίες από την πύλη G στην πύλη G'. Επομένως, μόνο οι διαδρομές που καταλήγουν στον τερματικό κόμβο "1" του BDD ευαισθητοποίησης και ένας κόμβος διαφορετικός από το "0" του συσχετισμένου ADD, πρέπει να ληφθούν υπόψη για τον υπολογισμό των νέων τιμών που βασίζονται στο μοντέλο εξασθένησης. Όλες οι άλλες περιπτώσεις, που υποδεικνύουν είτε λογική είτε ηλεκτρική κάλυψη, πηγαίνουν στον τερματικό κόμβο "0". Το Σχ. 3.3 δείχνει τη συνολική διαδικασία των προσθηκών για ένα σφάλμα που προέρχεται από την πύλη στο Σχ. 3.2



Σχήμα 3.3: ADDs διάρκειας για ένα σφάλμα που προέρχεται απο την πύλη G2 και περνάει στην πύλη G3 και G5

3.4 Προσεγγιστική εύρεση του SER

Για την εκτίμηση της συχνότητας εμφάνισης λαθών σε κάποιον κόμβο, με διαφορετικά ποσοστά ακρίβειας, (soft error rate SER) έχουν προταθεί αρκετές προσεγγίσεις σε διαφορετικά ακρίβειας όσων αφορά τα συνδυαστικά κυκλώματα. Η κατηγοριοποίηση τους αναφέρεται παρακάτω:

- Απευθείας προσομοίωση ενός SEU εντός του κυκλώματος. Εργαλεία όπως το SEMM χρησιμοποιούν Monte Carlo προσομοίωση για την επίτευξη ακριβέστερων αποτελεσμάτων αλλά με υψηλό τίμημα τον μεγάλο χρόνο εκτέλεσης.

- Ανάμικτες μέθοδοι προσομοίωσης. Η λύση που προτείνει το Dynamo προσπαθεί να μειώσει τις απαιτήσεις της προσομοίωσης, αρχικά με μια στατική προσέγγιση, και στην συνέχεια προσομοιώνοντας το κομμάτι του κυκλώματος που παρουσιάζει την κρίσιμη δραστηριότητα σε επίπεδο κυκλώματος, ενώ το υπόλοιπο προσομοιώνεται σε επίπεδο χρόνου.

- Υβριδικές μέθοδοι όπως το SERA αλλά και το SEUPERFAST. Το SERA χρησιμοποιεί έναν συνδυασμό από μαθηματικά και τεχνικές προσομοίωσης για να φτάσει στην εκτίμηση του SER χωρίς το κόστος της πλήρους προσομοίωσης του κυκλώματος. Αυτό το πετυχαίνει εξομοιώνοντας τις πύλες μεταξύ του κόμβου που συμβαίνει το SEU και των εισόδων των μανδαλωτών με ισοδύναμους inverters και διεξάγει προσομοίωση σε electrical level πάνω στο απλοποιημένο κύκλωμα. Το SEUPERFAST είναι ένας προσομοιωτής που δουλεύει σε υψηλό επίπεδο σχεδίασης όπου δεν μπορεί να γίνει

λεπτομερής ανάλυση μέσω της προσομοίωσης αλλά υπολογίζει το SER με χρήση μαθηματικών μοντέλων.

- Προσομοιωτές σε λογικό επίπεδο με χρήση μοντέλου καθυστέρησης. Αυτή η κατηγορία περιλαμβάνει εργαλεία όπως το FAST , το ASSERTA και το ASSA και προσπαθούν να υπολογίσουν τους τρεις παράγοντες (electrical, timing και logic masking) που περιορίζουν ένα SEU από το να περάσει στο ακολουθιακό κομμάτι. Το FAST συνδυάζει την χρήση ενός κλασσικού event driven προσομοιωτή για εκτίμηση της πορείας διάδοσης και αποθήκευσης ενός SEU με έναν προσομοιωτή μηδενικού μοντέλου χρόνου (zero-delay) ενώ από την άλλη χρησιμοποιείται ένας χρονικός προσομοιωτής για την εκτίμηση της διάρκειας κάθε παλμού. Το ASERTA χρησιμοποιεί πίνακες αληθείας για τον προσδιορισμό του SEU, zero delay προσομοιωτή για την εκτίμηση του logical masking, ένα μοντέλο με χρήση ράμπας για το electrical masking και μελέτη της διάρκειας του παλμού για timing masking. Το ASSA χρησιμοποιεί ένα εκτενές χρονικό παράθυρο για timing masking, διάδοση πιθανότητας για logic masking και μια καμπύλη απόρριψης θορύβου για electrical masking.

3.5 Ανάλυση Soft Error

Θεωρούμε ότι ένα ακολουθιακό κύκλωμα μπορεί να περιγραφεί σαν ένας πίνακας από flip-flops που ελέγχονται από ένα ρολόι και συνδυαστική λογική ανάμεσα στα επίπεδα των flip-flops. Θεωρούμε επίσης ότι το Soft Error Rate ενός κυκλώματος εκφράζεται σαν το ποσοστό των Single Event Upsets (SEUs) που παρουσιάζονται εντός του κυκλώματος και «πιάνονται» από τα στοιχεία μνήμης, δηλαδή τα flip-flops. Επίσης θεωρούμε ότι ένα SEU έχει σταθερή πιθανότητα να εμφανιστεί σε έναν κόμβο κατά την διάρκεια μιας περιόδου.

3.6 Κυκλώματα που δοκιμάστηκαν

Στο πίνακα που ακολουθεί, βλέπουμε όλα τα κυκλώματα που εξετάστηκαν στα πλαίσια της εργασίας αυτής. Τα κυκλώματα αυτά είναι σε Verilog από τη γνωστή σουίτα των ακολουθιακών κυκλωμάτων αναφοράς ISCAS '89 (s27, s298, κλπ). Ακόμα στον Πίνακα βλέπουμε και χαρακτηριστικά για τα κυκλώματα, όπως πόσες πύλες ή flip flops έχει. Δεν καταφέρθηκε να εξεταστούν όλα τα κυκλώματα, λόγω προβλήματος στην ονομασία των καλωδίων των

κυκλωμάτων.

Πίνακας κυκλωμάτων			
Όνομα κυκλώματος	Αριθμός κόμβων	Αριθμός πυλών	D-Flip Flops
S27	17	13	4
S208	149	139	8
S298	169	166	14
S386	284	177	6
S344	240	231	21
S349	224	215	21
S400	203	200	21
S444	211	208	21
S526	280	277	21
S420	252	233	16
S510	293	274	6
S832	457	429	5
S820	443	425	5
S641	517	482	19
S713	539	504	19
S953	496	480	29
S1238	768	754	18
S1196	762	748	18
S9234	7002	6983	228
S13207	9608	9577	669

Κεφάλαιο 4

Παρουσίαση του εργαλείου

4.1 Εισαγωγή

Στην ενότητα αυτή θα δούμε το λογισμικό-εργαλείο που δημιουργήθηκε στα πλαίσια της εργασίας. Θα δούμε τον τρόπο δόμησης και σχεδίασης του εργαλείου, καθώς και τον τρόπο χρήσης του. Κατά την διάρκεια της εργασίας αναπτύχθηκε ένα λογισμικό το οποίο αποσκοπεί στην προσομοίωση και βελτίωση κυκλωμάτων.

4.2 Λίγα πράγματα για τις απαιτήσεις του

Κατά την διάρκεια της διπλωματικής εργασίας κρίθηκε απαραίτητο να δημιουργηθεί ένα εργαλείο προσομοίωσης και αξιολόγησης των κυκλωμάτων. Πιο συγκεκριμένα, ζητήθηκε να δημιουργηθεί ένα εργαλείο, το οποίο θα διαβάζει ένα αρχείο τύπου Verilog και με βάση το αρχείο να δημιουργεί εικονικά το κύκλωμα που το αρχείο περιγράφει. Στην συνέχεια, να παρεχει την δυνατότητα στον χρήστη να εκτελέσει λογική προσομοίωση παρέχοντας τιμές για τις εισόδους του κυκλώματος, το ρολόι και άλλα. Να αποθηκεύει τα δεδομένα αυτά σε ένα αρχείο για να γίνει το αρχείο στοιχείο αναφοράς. Στην συνέχεια να τραβάει δεδομένα από το εργαλείο εργαλείο σχετικά με τις πιο αδύναμες πύλες του κυκλώματος και εφαρμόζοντας αλγορίθμους να τις τροποποιεί και να δίνει ένα νέο κύκλωμα σε αρχείο Verilog, το οποίο θα έχει την ίδια λειτουργία με το αρχικό. Η ίδια λειτουργία του κυκλώματος θα βεβαιώνεται και αυτή μέσω του εργαλείου που δημιουργήθηκε και του αρχείου με στοιχεία από την αρχική εκτέλεση.

```
324         // opening file for reading
325         fp = fopen(fileOpen , "r");
326         if(fp == NULL) {
327             perror("Error opening file");
328 
329           return NULL;
330         }
331 
332       while(fgets(str, inputStringSize, fp)!= NULL){
333 
334             strcpy(readtemp,str);
335 
336             if(readtemp[0] == '/'){
337                 SetZero(str, inputStringSize);
338                 SetZero(readtemp, wireStringSize);
339                 continue;
340              }
341 
342              //Start code here for structure!!!!!
343 
344              //Throw first line saying module
345              //Check first only for m only for speed
346              if(readtemp[0] == 'm'){
347                 SetZero(temp,50);
348                 for(counter=0; counter<6; counter++){
349                     temp[counter] = readtemp[counter];
350                 }
351                 if(!strcmp(temp,"module")){
352                     lineread = 0;
353                     continue;
354                 }
355                 else{
356                     return NULL;
357                 }
358              }
```

Σχήμα 4.1: Διάβασμα του κυκλώματος απο αρχείο

4.2.1 Γλώσσα προγραμματισμού και δυσκολίες

Το εργαλείο που δημιουργήθηκε στα πλαίσια της εργασίας είναι βασισμένο στην γλώσσα προγραμματισμού C. Χρησιμοποιεί γνωστές βιβλιοθήκες, είναι δομημένο με τρόπο να μπορεί κάποιος να το τροποποιήσει και να το βελτιώσει και είναι διαθέσιμο στο github. Η δημιουργία του δεν ήταν απλή. Γνωρίζουμε πως η C είναι αρκετά low level γλώσσα προγραμματισμού και τι δυσκολίες αυτό φέρνει. Χρειάστηκε η πλήρης κατανόηση σε βάθος τόσο της γλώσσας C όσο και Verilog, καθώς επίσης και των αλγορίθμων που εκτελέστηκαν και δημιουργήθηκαν. Κάποια από τα πιο δύσκολα κομμάτια ήταν η δημιουργία του κυκλώματος σε επίπεδα. Η κάθε πύλη ανάλογα την τοποθεσία της έχει ένα επίπεδο. Θα δούμε πιο αναλυτικά τις δυσκολίες σε επόμενη ενότητα. Ενδεικτικά να αναφέρουμε πως για την πλήρη λειτουργία του προγράμματος χρειάστηκαν πάνω από 6.000 σειρές κώδικα.

4.3 Δημιουργία εικονικού κυκλώματος

Όπως αναφέραμε, ο χρήστης ορίζει το αρχείο Verilog που θέλει να προσομοιώσει. Το πρόγραμμα ανοίγει το αρχείο και το διαβάζει. Στην συνέχεια πρέπει με βάση το αρχείο αυτό να δημιουργεί σε μία δομή το κύκλωμα. Αυτή η διαδικασία γίνεται με τον κώδικα στο σχήμα 4.1.

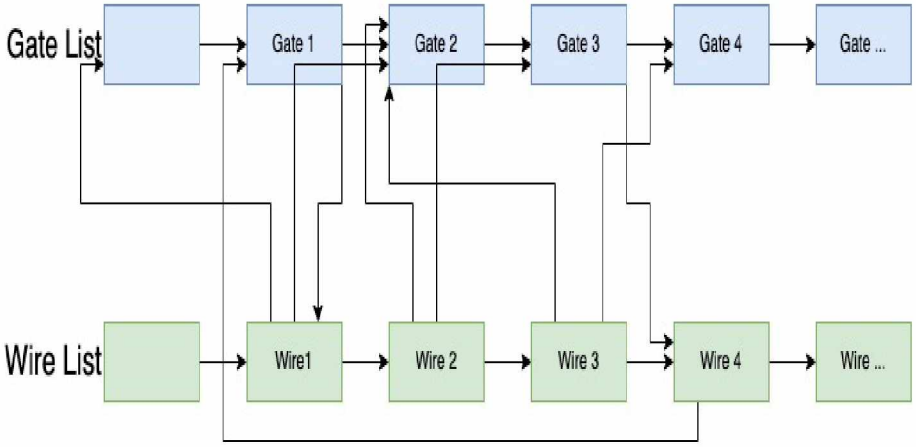
```c
C structs.h
1   struct gate{
2       char gate_name[50];
3       char gate_inputs[200];
4       char gate_output[100];
5       char gate_type[50];
6       int input, output;
7       int layer;
8       int value;
9       struct gate *next;
10      //I will point to node n
11      struct wire *inputs[50]; //Remember to Initialize to NULL
12      struct wire *outputs[2];
13  };
14
15  //For wires
16  struct wire{
17      char node_name[50];
18      int value;
19      int layer;
20      struct wire *next;
21  };
```

Σχήμα 4.2: Η δομή των πυλών και των καλωδίων σε κώδικα

Η ιδέα του κυκλώματος, ως δομή, είναι να παρουσιαστεί σαν μια λίστα από κόμβους. Οι πύλες αποτελούν την δικιά τους συνδεδεμένη λίστα όπως αντίστοιχα και τα καλώδια. Άρα δηλαδή δημιουργήθηκαν 2 λίστες ξεχωριστές, που ταυτόχρονα ενώνονται και μεταξύ τους, όπως ακριβώς και ένα κύκλωμα. Οι λίστες είναι δομές δεδομένων. Είναι πολοί κόμβοι απο structs συνδεδεμμένοι μεταξύ τους. Για την δημιουργία του κυκλώματος χρησιμοποιήθηκαν οι παρακάτω δομές για κάθε node καλωδίου και πύλης.

Το Verilog αρχείο όμως δεν περιέχει τις πύλες στην σωστή σειρά. Η πύλη πριν την έξοδο του κυκλώματος για παράδειγμα μπορεί να βρίσκεται στην αρχή του αρχείου. Αυτό δεν θα μπορούσε να λειτουργήσει καθώς στην συνέχεια κατά την διάρκεια της εκτέλεσης όταν θα περνούσαμε τις τιμές θα βρίσκονταν σε λάθος θέση. Έτσι δημιουργήθηκε μια τρίτη λίστα που δημιουργεί από την πρώτη, η οποία αντιπροσωπεύει τις πύλες, το κύκλωμα σε επίπεδα. Όλες οι λίστες είναι δυναμικά δημιουργημένες ενώ για την μετατροπή του κυκλώματος σε επίπεδα, Τρίτη λίστα, δεν χρησιμοποιήθηκε επιπλέον μνήμη παρά την δημιουργούμε από την 1η λίστα. Θα μπορούσαμε να φανταστούμε τις λίστες, μετά την δημιουργία των επιπέδων, όπως στην εικόνα που ακολουθεί.

Το κύκλωμα επίσης έπρεπε να δημιουργηθεί σε επίπεδα. Στην αρχή της λίστας έπρεπε να αποθηκεύουμε τους κόμβους, οι οποίοι θα έπαιρναν πρώτοι τιμή. Δηλαδή η λίστα δημιουργήθηκε έτσι ώστε στην αρχή να βρίσκονται οι κόμβοι που θα τους ανατεθεί κατευθείαν τιμή από το input του κυκλώματος. Αυτό είναι το επίπεδο 0. Οι πύλες ακριβώς μετά από αυτές,



Σχήμα 4.3: Οπτικοποίηση της δομής του κυκλώματος

που στο επόμενο clock θα πάρουν τιμές βρίσκονται ένα επίπεδο μετά. Βέβαια κάθε πύλη δεν έχει είσοδο ένα καλώδιο το οποίο είναι η έξοδος από μια άλλη πύλη, αλλά έχει πάνω από μία εισόδους. Οπότε γενικά το επίπεδο μίας πύλης N ήταν μέγιστο επίπεδο των καλωδίων που είχε σαν είσοδο συν 1.

$$Level = \max(Inputs + 1) \tag{4.1}$$

Οι δύο λίστες λοιπό αντιπροσωπεύουν το κύκλωμα μας. Όπως μπορούμε να δούμε, η λίστα των καλωδίων είναι συνδεδεμένη με των πυλών. Δηλαδή η κάθε έξοδος μιας πύλης δείχνει προς ένα ή περισσότερα καλώδια τα οποία με την σειρά τους δείχνουν σε μια ή περισσότερες πύλες.

4.3.1 Δημιουργία κάθε κόμβου

Αρχικά καθώς διαβάζουμε το κύκλωμα απο το αρχείο Verilog, δημιουργούμε ταυτόχρονα και κόμβους στην λίστα. Ο κάθε κόμβος της λίστας δημιουργείται δυναμικά. Διαβάζοντας το αρχείο Verilog δημιουργούμε την αρχική λίστα των πυλών που περιέχει όλες τις πύλες που υπάρχουν στο κύκλωμα. Κατά την διάρκεια αυτής της επανάληψης, αρχικοποιούνται οι τιμές κάθε κόμβου στην λίστα. Αρχικοποιούνται οι τιμές των επιπέδων της λίστας σε -1 όπως επίσης και το όνομα κάθε κόμβου, και το είδος της πύλης (AND-NOR-D Flip Flop etc). Κατά την κυρίως μέθοδο δημιουργίας της αρχικής λίστας, χρησιμοποιήθηκαν αρκετές βοηθητικές συναρτήσεις, όπως αρχικοποίησης πινάκων, δεικτών καθώς και συναρτήσεις διαχωρισμού δεδομένων. Ενδεικτικά δείχνουμε τον κώδικα διαχωρισμού των λευκών χαρα-

```
199  void BasicDataSpliter(struct gate *ptr, char tosplit[]){
200      char temp[inputStringSize];
201      int counter=0, counter1=0;
202
203      SetZero(temp,inputStringSize);
204
205      //First get the name
206      while(tosplit[counter] != '('){
207          temp[counter] = tosplit[counter];
208          counter++;
209      }
210      //Delete last space from temp_gate name
211      temp[counter] = '\0';
212      while(counter1 < counter){
213          temp[counter1] = temp[counter1+2];
214          counter1++;
215      }
216
217      //Set it to the temp_gate_name
218      strcpy(ptr->gate_name, temp);
219
220      //Clear temp string
221      SetZero(temp, inputStringSize);
222
223      //Got the Substring
224      if(tosplit[counter] == ' '){
225          counter++;
226      }
227
228      counter1 = 0;
229      while(tosplit[counter] != ';'){
230          temp[counter1] = tosplit[counter];
231          counter++;
232          counter1++;
233      }
234      //Drop space);
235      temp[counter1+2] = ';';
236
237      //Split input-output and type
238      InputOutputSpliter(ptr, temp);
239  }
```

Σχήμα 4.4: Διαχωρισμός δεδομένων απο κενά και ειδικούς χαρακτήρες

κτήρων και κάποιον άλλων special character από ένα string που περιέχει δεδομένα σχετικά με τις εισόδους και τις εξόδους των κυκλωμάτων. Με την δημιουργία της αρχικής λίστας δημιουργήθηκαν και συναρτήσεις εύρεσης μιας πύλης και άλλων βοηθητικών συναρτήσεων. Ακριβώς με παρόμοιο τρόπο δημιουργήθηκε και λίστα των καλωδίων με την διαφορά των τιμών που αρχικοποιούνται, καθώς και στον τρόπο δέσμευσης της μνήμης για την καινούρια λίστα των καλωδίων.

```
290  struct gate *CreateInitialList(char *input, char *output, char *wires){
291
292      //File pointer
293      FILE *fp;
294      char str[inputStringSize]; //String to read from file
295
296      //String for the circuit input - output - wire
297      char inputs[inputStringSize];
298      char outputs[outputStringSize];
299      char wire[wireStringSize];
300      char readtemp[wireStringSize];
301      char read[wireStringSize];
302
303      //String for universal use for small calculations. Does not take date from wires-gates etc
304      char temp[50];
305
306      //Which line from file i read starting from input(0)
307      int counter, lineread = 0;
308      int flag=0;
309
310      //Head of temp linked list
311      struct gate *head = NULL;
312      struct gate *tnode;
313      struct gate *curr;
314
315      //Initialize everything to \0
316      SetZero(str, inputStringSize);
317      SetZero(inputs, inputStringSize);
318      SetZero(outputs, outputStringSize);
319      SetZero(wire, wireStringSize);
320      SetZero(readtemp, wireStringSize);
321      SetZero(read, wireStringSize);
322      SetZero(temp,50);
323
324      // opening file for reading
325      fp = fopen(fileOpen , "r");
326      if(fp == NULL) {
327        perror("Error opening file");
328
329        return NULL;
330      }
331
332     while(fgets(str, inputStringSize, fp)!= NULL){
333
334         strcpy(readtemp,str);
335
336         if(readtemp[0] == '/'){
337             SetZero(str, inputStringSize);
338             SetZero(readtemp, wireStringSize);
```

```
339             continue;
340         }
341
342         //Start code here for structure!!!!!
343
344         //Throw first line saying module
345         //Check first only for m only for speed
346         if(readtemp[0] == 'm'){
347             SetZero(temp,50);
348             for(counter=0; counter<6; counter++){
349                 temp[counter] = readtemp[counter];
350             }
351             if(!strcmp(temp,"module")){
352                 lineread = 0;
353                 continue;
354             }
355             else{
356                 return NULL;
357             }
358         }
359
360         //Throw last line saying endmodule
361         if(readtemp[0] == 'e'){
362             //SetZero(readtemp,10);
363             for(counter=0; counter<9; counter++){
364                 temp[counter] = readtemp[counter];
365             }
366             if(!strcmp(temp,"endmodule")){
367                 break;
368             }
369             else{
370                 printf("Temp is %s\n", temp);
371                 return NULL;
372             }
373         }
374
375         //do it until \n so we can track ,
376         counter=0;
377         while(readtemp[counter] != '\n'){
378             if(readtemp[counter] == ';'){break;}
379             counter++;
380         }
381
```

```
382         if(readtemp[counter-1] == ','){
383             strcat(read, readtemp);
384             flag = 1;
385             continue;
386         }
387         if(flag == 1){
388             strcat(read, readtemp);
389         }
390 
391         if(flag == 0){
392             strcpy(read, readtemp);
393         }
394 
395 
396         //Read inputs
397         if(lineread == 0){
398             strcpy(inputs, read);
399             SetZero(str, inputStringSize);
400             SetZero(readtemp, wireStringSize);
401             SetZero(read, wireStringSize);
402             lineread++;
403             continue;
404         }
405         //Read output
406         if(lineread == 1){
407             strcpy(outputs, read);
408             SetZero(str, inputStringSize);
409             SetZero(readtemp, wireStringSize);
410             SetZero(read, wireStringSize);
411             lineread++;
412             continue;
413         }
414         //Read wire
415         if(lineread == 2){
416             strcpy(wire, read);
417             SetZero(str, inputStringSize);
418             SetZero(readtemp, wireStringSize);
419             SetZero(read, wireStringSize);
420             lineread++;
421             continue;
422         }
423 
```

```
424         //Drop empty lines no line will have length less than 2
425         if(strlen(read) < 3){
426             lineread++;
427             continue;
428         }
429 
430         //lineread is 4 means i just started reading the temp_gates
431         if(lineread == 4){
432             head = (struct gate *)malloc(sizeof(struct gate));
433             if(head == NULL){
434                 printf("Error alocating memmory for the temp_gate!!!\nExiting...\n");
435                 return NULL;
436             }
437             BasicDataSpliter(head, read);
438             head->next = NULL;
439             head->layer = -1;
440             lineread++;
441             SetZero(str, inputStringSize);
442             SetZero(readtemp, wireStringSize);
443             SetZero(read, wireStringSize);
444             tnode = head;
445 
446             continue;
447         }
448         curr = (struct gate *)malloc(sizeof(struct gate));
449         if(curr == NULL){
450             printf("Error alocating memmory for the temp_gate!!!\nExiting...\n");
451             return NULL;
452         }
453 
454         //Fix first list
455         tnode->next = curr;
456         curr->layer = -1;
457         curr->next = NULL;
458         BasicDataSpliter(curr, read);
459         lineread++;
460         tnode = curr;
461 
462         flag = 0;
463 
464         SetZero(str, inputStringSize);
465         SetZero(readtemp, wireStringSize);
466         SetZero(read, wireStringSize);
467 
468     }
469     strcpy(input,inputs);
470     strcpy(output, outputs);
471     strcpy(wires, wire);
```

```
472
473         //Close pointer to file
474         fclose(fp);
475
476         return head;
477     }
478
479     int gateExists(struct gate *head, char stringforsearch[]){
480         struct gate *itr;
481
482         itr = head;
483
484         while(itr!=NULL){
485             if(strcmp(itr->gate_name,stringforsearch) == 0){
486                 return 0;
487             }
488             itr = itr -> next;
489         }
490
491         return -1;
492     }
493
494     struct gate *Find(struct gate *ptr, char stringforsearch[], struct gate *value[]){
495         struct gate *curr;
496         int counter, i=0;
497
498         char *ret;
499         for(counter=0; counter<40; counter++){
500             value[counter] = NULL;
501         }
502
503         curr=ptr;
504
505         while(curr != NULL){
506             ret = strstr(curr->gate_inputs, stringforsearch);
507             if(ret != NULL){
508                 value[i] = curr;
509                 i++;
510             }
511             curr = curr->next;
512         }
513
514         return NULL;
515     }
```

4.4 Ένωση κόμβων στην κάθε λίστα

Η ένωση των κόμβων ήταν σχετικά απλή διαδικασία. Σε μια μεγάλη επανάληψη, όσο διαβάζαμε απο το αρχείο και δεν είχαμε φτάσει στο τέλος του, δημιουργούσαμε κόμβους απο πύλες και τις συνδέαμε μεταξύ τους. Αυτή η σύνδεση βέβαια, δεν ήταν η οριστική. Επειδή απλά καθώς διαβάζαμε δημιουργούσαμε κόμβους και τους ενώναμε δεν σημαίνει πως ήταν η σωστή λίστα, η οποία βρίσκεται σε κατάλληλα επίπεδα. Έτσι έπρεπε να δημιουργηθεί μια νέα λίστα, η οποία θα περιέχει τους κόμβους στα σωστά επίπεδα. Με την λίστα των καλωδίων ήταν κάτι διαφορετικό.

Στο αρχείο verilog όπως θα δούμε, τα καλώδια ορίζονται στην αρχή του κυκλώματος σε συνδιασμό με ποιά απο αυτά είναι και είσοδοι ή έξοδοι του κυκλώματος. Όπως φαίνεται και στο σχήμα 4.5, η γραμμή wire ορίζει ποια καλώδια υπάρχουν στο κύκλωμα, ενώ από επάνω μπορούμε να δούμε και ποιά απο αυτά τα καλώδια λειτουργούν ως είσοδοι ή έξοδοι του κυκλώματος. Πρωτού προχωρήσουμε φυσικά στην ένωση των δύο λιστών έπρεπε να αρχικοποιήσουμε τους πίνακες που θα αποθήκευαν τους δείκτες για τις εισόδους και τις εξόδους των πυλών. Δημιουργήθηκαν αρκετές βοηθητικές συναρτήσεις για διάφορους σκοπούς αρχικοποιήσεων και όχι μόνο, όπως την αρχικοποίηση strings, πινάκων από δείκτες και άλλα.

```
////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : O-2018.06-SP4
// Date      : Sat Nov 21 12:03:26 2020
////////////////////////////////////////////////////////////


module s27 ( GND, VDD, CK, G0, G1, G17, G2, G3 );
  input GND, VDD, CK, G0, G1, G2, G3;
  output G17;
  wire   G10, G13, n1, n2, n3, n4, n5, n6, n7, n8, n10, n11;

  DFF_X1 DFF_0 ( .D(G10), .CK(CK), .QN(n10) );
  DFF_X1 DFF_1 ( .D(n1), .CK(CK), .QN(n11) );
  DFF_X1 DFF_2 ( .D(G13), .CK(CK), .Q(n2) );
  INV_X1 U12 ( .A(G17), .ZN(n1) );
  NAND3_X1 U13 ( .A1(n4), .A2(n5), .A3(n10), .ZN(G17) );
  OR2_X1 U14 ( .A1(n8), .A2(G3), .ZN(n4) );
  NAND2_X1 U15 ( .A1(n6), .A2(n7), .ZN(n5) );
  NOR2_X1 U16 ( .A1(n11), .A2(G0), .ZN(n8) );
  NOR2_X1 U17 ( .A1(G2), .A2(n3), .ZN(G13) );
  NOR2_X1 U18 ( .A1(G1), .A2(n2), .ZN(n3) );
  OR2_X1 U19 ( .A1(G0), .A2(n11), .ZN(n7) );
  OR2_X1 U20 ( .A1(n2), .A2(G1), .ZN(n6) );
  AND2_X1 U21 ( .A1(G17), .A2(G0), .ZN(G10) );
endmodule
```

Σχήμα 4.5: Το κύκλωμα S27 σε verilog

```
void null(struct gate *head){
    struct gate *curr;
    int counter;

    curr = head;

    while(curr!=NULL){
        for(counter=0; counter<50; counter++){
            curr->inputs[counter] = NULL;
        }
        for(counter=0; counter<2; counter++){
            curr->outputs[counter] = NULL;
        }
        curr = curr->next;
    }
}
```

Σχήμα 4.6: Συνάρτηση αρχικοποίησης εισόδων - εξόδων

```
724  void connect(struct gate *headgate, struct wire *headwire){
725      struct gate *currgate;
726      struct wire *currwire;
727      int counter, i=0, j=0;
728      char temp[100] = {'\0'};
729
730      //Loop to go throw gate list and connect it times as inputs to the correct wire list
731      for(currgate = headgate; currgate != NULL; currgate = currgate->next){
732
733          //Adding delimiter for spliting the strings in the string
734          currgate->gate_inputs[strlen(currgate->gate_inputs)-1] = '/';
735          for(counter=0; counter<currgate->input; counter++){
736              while(currgate->gate_inputs[i] != ' '){
737                  if(currgate->gate_inputs[i] == '/'){
738                      break;
739                  }
740                  temp[j] = currgate->gate_inputs[i];
741                  i++;
742                  j++;
743              }
744
745              currwire = FindCheck(headwire, temp);
746              if(currwire == NULL){
747                  printf("Something went wrong!!\n");
748                  exit(1);
749              }
750              SetZero(temp,100);
751              j=0;
752              i++;
753
754              //Build pointers from here on
755              currgate->inputs[counter] = currwire;
756          }
757          i=0;
758
759          //Removing delimiter now
760          currgate->gate_inputs[strlen(currgate->gate_inputs)-1] = '\0';
761      }
762
763      //Set zero to temp string
764      SetZero(temp,100);
765
766      //Loop to go throw gate list and connect it times as outputs to the correct wire list
767      for(currgate = headgate; currgate != NULL; currgate = currgate->next){
768          //Adding delimiter for spliting the strings in the string
769          currgate->gate_output[strlen(currgate->gate_output)-1] = '/';
770          for(counter=0; counter<currgate->output; counter++){
```

Σχήμα 4.7: Συνάρτηση ένωσης των δύο διαφορετικών λιστών, καλωδίων και πυλών

4.5 Ένωση των καλωδίων με τις πύλες

Η ένωση των δύο λιστών ήταν πιο απαιτητική διαδικασία σε σχεση με τις προηγούμενες. Αυτο συνέβη επειδη κατα τη διάρκεια της ανάγνωσης του αρχείου Verilog, σε κάθε γραμμή όπου υπήρχε μια πύλη υπήρχαν και τα καλώδια της πύλης. Έπρεπε κατά την διάρκεια ένωσης των καλωδίων, σε κάθε κόμβο να βλέπουμε τις εισόδους και την έξοδο ή εξόδους του, και να πηγαίνουμε στην λίστα των καλωδίων να κάνουμε την αντιστοίχηση. Αυτή η διαδικασία κρύβει και το πρόβλημα λάθος ονομασίας κάποιας πύλης ή καλωδίου. Δηλαδή δεν γίνεται να ενώσουμε ένα καλώδιο σε μια πύλη το οποίο δεν παίρνει τιμή απο πουθενά και ταυτόχρονα δεν είναι είσοδος του κυκλώματος. Έτσι, κατα την διάρκει ένωσης των καλωδίων και των πυλών, έπρεπε πρώτα να βρούμε το καλώδιο που έχει ως είσοδο, για να επιβεβαιώσουμε ότι υπάρχει, και μετά να αναθέσουμε τον δείκτη του συγκεκριμένου κόμβου στις εισόδους της πύλης.

```
770         for(counter=0; counter<currgate->output; counter++){
771             while(currgate->gate_output[i] != ' '){
772                 if(currgate->gate_output[i] == '/'){
773                     break;
774                 }
775                 temp[j] = currgate->gate_output[i];
776                 i++;
777                 j++;
778             }
779 
780             currwire = FindCheck(headwire, temp);
781             if(currwire == NULL){
782                 printf("Something went wrong1!!\n");
783                 exit(1);
784             }
785             SetZero(temp,100);
786             j=0;
787             i++;
788 
789             //Build pointers from here on
790             currgate->outputs[counter] = currwire;
791         }
792         i=0;
793         //Remove delimiter now
794         currgate->gate_output[strlen(currgate->gate_output)-1] = '\0';
795     }
796 
797     //Setting level to the flip flops after connecting gates with wires
798     levelSetToDFlipFlops(headgate);
799 
800 }
```

Σχήμα 4.8: Συνάρτηση ένωσης των δύο διαφορετικών λιστών, καλωδίων και πυλών

4.6 Δημιουργία της λίστας σε σωστή σειρά - επίπεδα

Για να την εκτέλεση του κυκλώματος, το κύκλωμα έπρεπε να είναι χωρισμένο σε επίπεδα ανάλογα με τις πύλες. Δεν ήταν εφικτό να εκτελεστεί το κύκλωμα αλλιώς, καθώς το εργαλείο δεν θα γνώριζε ποιά πύλη θα έπρεπε πρωτα να πάρει τιμή.

4.6.1 Αρχικοποίηση επιπέδων

Αρχικά, έπρεπε να ορίσουμε τα επίπεδα των πυλών σε μια τιμή προκειμένου να γνωρίζουμε ποιες πύλες έχουν πάρει επίπεδο και ποιες όχι. Έτσι σε μια επανάληψη στην λίστα μας αρχικοποιούμε όλα τα επίπεδα των πυλών σε -1. Έτσι, αν εντοπίσουμε μια πύλη που έχει τιμή, σημαίνει πως την έχουμε ορίσει εμείς. Επίσης στην ίδια επανάλληψη ορίζουμε και τα επίπεδα των D-Flip Flop σε 0 καθώς θεωρούνται ως είσοδοι.

```
944  void restGatesLeveled(struct gate *head, struct wire *headWire, int level){
945
946      struct gate *iterator;
947      int counter, inputCount;
948      int pos;
949      int flag;
950
951      iterator = head;
952
953      //Set levels
954      while(iterator != NULL){
955          if(strcmp(iterator->gate_type,"D_Flif_Flop") == 0 ){
956              iterator = iterator->next;
957              continue;
958          }
959
960          //initialize count
961          counter = 0;
962          inputCount = 0;
963          //Finding number of inputs of a gate so later to check the layer of the wire
964          while(iterator->inputs[counter] != NULL){
965              inputCount++;
966              counter++;
967          }
968          flag = 0;
969          for(pos=0; pos<inputCount; pos++){
970              if(iterator->inputs[pos]->layer <= level && iterator->inputs[pos]->layer >= 0){
971                  //Do nothing
972              }
973              else{
974                  flag=1;
975              }
976          }
977          if(flag == 0){
978              iterator->layer = level;
979          }
980          else{
981              iterator->layer = -1;
982          }
983          iterator = iterator->next;
984      }
985
986  }
```

Σχήμα 4.9: Αρχικοποίηση επιπέδων και D Flip Flops

4.6.2 Αρχικοποίηση επιπέδου 0 και νέας λίστας

Στην συνέχεια έπρεπε να δημιουργήσουμε την νέα λίστα, η οποία θα είχε τις πύλες στα σωστά επίπεδα. Στην αρχή της λίστας θα βρίσκονται οι πύλες με επίπεδο 0 στην συνέχει με επίπδεο 1 κοκ. Έτσι διαβάζουμε απο το αρχικό κύκλωμα τις πύλες και όποιες έχουν επίπεδο 0, τις αφαιρούμε απο την αρχική λίστα και τις προσθέτουμε στην νέα λίστα που θα βρίσκεται δομημένη σε επίπδεα.

```
889  struct gate *rebuildLevelOrderLayer0(struct gate **head){
890      struct gate *newHead;
891      struct gate *newListTemp;
892      struct gate *temp, *ptrReFix;
893
894      temp = *head;
895
896      //Find node with level 0 to make it head to the new list
897      while(temp->layer != 0){
898          ptrReFix = temp;
899          temp = temp->next;
900      }
901
902      //Node must been found
903      newHead = temp;
904
905      //Check if that node was head and build head if so
906      if(temp == *head){
907          *head = temp->next;
908      }
909      else{
910          //Fix pointers for next, and the old list
911          ptrReFix->next = temp->next;
912      }
913
914      newHead->next = NULL;
915
916      temp = *head;
917      newListTemp = newHead;
918
919      //Build evrything in level 0
920      while(temp != NULL){
921
922          if(temp->layer == 0){
923              if(temp == *head){
924                  *head = temp->next;
925                  newListTemp->next = temp;
926                  newListTemp = newListTemp->next;
927                  temp = temp->next;
928                  newListTemp->next = NULL;
929                  continue;
930              }
931              else{
932                  newListTemp->next = temp;
933                  newListTemp = newListTemp->next;
934                  newListTemp->next = NULL;
935                  ptrReFix->next = temp->next;
936              }
937          }
938          ptrReFix = temp;
939          temp = temp->next;
940      }
941      return newHead;
942  }
```

Σχήμα 4.10: Δημιουργία νεας λίστας με επίπεδο 0

```
1003  //This function fixes the level of the wires after a gate in the final list, starting from given level
1004  void levelingWireAfterGate(struct gate *headGate, struct wire *wireHead, int level){
1005      struct gate *iterator;
1006
1007      iterator = headGate;
1008
1009      while(iterator != NULL){
1010          if(iterator->layer == level){
1011              iterator->outputs[0]->layer = iterator->layer + 1;
1012              if(iterator->outputs[1] != NULL){
1013                  iterator->outputs[1]->layer = iterator->layer + 1;
1014              }
1015          }
1016          iterator = iterator->next;
1017      }
1018  }
```

Σχήμα 4.11: Ορισμός επιπέδων σε καλώδια με επίπεδο 0

4.6.3 Επίπεδα στα καλώδια

Για την τοποθέτηση επιπέδων στις πύλες, έπρεπε να δώσουμε επίπεδα στα καλώδια για να μπορούμε να τα υπολογίσουμε. Έτσι ως αρχικό βήμα χρειάστηκε να δώσουμε αρχικοποίηση στα καλώδια και να θέσουμε επίπεδα σε αυτά που θα έχουν επίπεδο 0. Επίπεδο 0 θα έχουν αυτά τα οποία είναι είσοδοι του κυκλώματος (Σχήμα 4.11).

4.6.4 Ορισμός επιπέδων στις υπόλοιπες πύλες

Απο την στιγμή που έχουμε δώσε σε όλες τις πύλες επίπεδα και έχουμε συνδέσει τις πύλες με τα καλώδια, μπορούμε να ξεκινήσουμε να ορίζουμε επίπεδα και στις πύλες με επίπδεο μεγαλύτερο του 0. Ο αλγόριθμος που υλοποιήθηκε για την πράξη αυτή ήταν σχετικά απλός. Ορίζουμε στα καλώδια-εξόδους των πυλών το επίπεδο της πύλης. Σε μία επανάληψη για την κάθε πύλη βρίσκουμε το μέγιστο επίπεδο των εισόδων-καλωδίων της πύλης και θέτουμε στην πύλη το μέγιστο επίπδεο + 1. Με αυτό τον τρόπο βρίσκουμε και θέτουμε επίπεδα σε όλες τις πύλες που δεν έχουν επίπεδο.

Απο την πλευρά του κώδικα, η προηγούμενη συνάρτηση που είδαμε παίρνει και σαν όρι-

```
1115  void buildCircuitLeveled(struct gate **head, struct gate *list3){
1116
1117      int levelToBuild; //Remember level 0 is already built
1118
1119      levelToBuild = 1; //Start from level 1 since level 0 is ready
1120      while(*head != NULL){
1121          fixGateLevel(head, list3, levelToBuild);
1122          levelToBuild++;
1123      }
1124
1125  }
```

Σχήμα 4.12: Ανασχηματισμός του κυκλώματος σε επίπεδα

σμα και το επίπεδο που θέλουμε να φτιάξουμε. Έτσι τώρα στην επανάληψη ξεκινάμε απο επίπεδο 1 και καλούμε την προηγούμενη συνάρτηση για να φτιάξουμε τα επίπεδα. Με αυτό τον τρόπο αυτοματοποιούμε κάποιες διαδικασίες και τις κάνουμε και πιο διαχειρίσημες(Σχήμα 4.12).

4.7 Εκτέλεση του κυκλώματος

Η εκτέλεση του κυκλώματος ήταν απλή διαδικασία. Αρχικά ορίζει ο χρήστης πόσες επανλλήψεις θέλει να τρέξει και στην συνέχεια σε μια μεγάλη επάναλήψη διαβάζουμε τις τιμές των εισόδων (Σχήμα 4.13). Οι τιμές αυτές περνάνε στα καλώδια και ανάλογα την πύλη πραγματοποιούμε την κατάλληλη πράξη. Σε μια επανάλληψη διαβάζουμε τις τιμές των καλωδίων κάθε πύλης και με βάση τον τύπο της κάνουμε την πράξη. Πχ σε μία OR πύλη αν έχουμε έστω και ένα 1 ως είσοδο το αποτέλεσμα θα είναι 1 στην έξοδο.

Την εκτέλεση του κυκλώματος την εκτελεί η μέθοδος run. Στην επανάλληψη αυτή όμως, υπάρχουν και άλλες βοηθητικές συναρτήσεις που κρατάνε logs για τις εισόδους που έκανε ο χρήστης καθώς και τιμές του κυκλώματος. Αυτές οι τιμές γράφονται σε ένα αρχείο για να

```
111
112     //Running circuit
113     for(int counter=0; counter<repeat; counter++){
114         strcat(data, "Run #"); //Fixing string to store input-output into a file
115         sprintf(numberToString, "%d\n", counter);
116         strcat(data, numberToString);
117
118         while(strlen(input) >= counterforinput ){
119             if(input[counterforinput] == ' ' || counterforinput == strlen(input)){
120                 printf("Please enter Value for input wire (%s): ", search);
121                 //Need a scanf here with search wire list and store the value
122                 scanf(" %d", &valuetopass);
123
124                 //All these strcats are building the string to go to output
125                 strcat(data, "\tWire ");
126                 strcat(data, search);
127                 strcat(data, ": ");
128                 sprintf(numberToString, "%d\n", valuetopass);
129                 strcat(data, numberToString);
130
131                 tmp = FindCheck(headwire, search);
132                 if(tmp == NULL ){
133                     printf("Something went wrong!\n");
134                     return -1;
135                 }
136                 tmp->value = valuetopass;
137                 SetZero(search, 20);
138                 counterforinput++;
139                 pos=0;
140             }
141             else{
142                 search[pos] = input[counterforinput];
143                 pos++;
144                 counterforinput++;
145             }
146         }
147
148         //Calculate new values of the circuit
149         //make function void update circuit status
150         run(gateList3Head, headwire);
151         // Update output value on file
152         //Write the data in to the file to have metrics
153         dataToFile(data);
154         printGateToFile(gateList3Head);
155         SetZero(data, logSize);
156         counterforinput=0;
157     }
```

Σχήμα 4.13: Επανάληψη εκτέλεσης του κυκλώματος

κρατάμε τα ένα δείγμα και να βεβαιθούμε ότι το κύκλωμα που δημιουργήσαμε έχει ακριβώς την ίδια συμπεριφορά με το αρχικό.

Ταυτόχρονα, πρίν απο την εκτέλεση του κυκλώματος γίνονται και έλεγχοι ασφαλείας μήπως κάτι έχει πάει στραβά κατά τα προηγούμενα στάδια και δεν έχει αρχικοποιηθεί κάτι ή δεν υπάρχει.

```
529  struct wire *FindCheck(struct wire *ptr, char stringforsearch[]){
530      struct wire *curr;
531
532      curr=ptr;
533
534      while(curr != NULL){
535
536          if(strcmp(curr->node_name, stringforsearch) == 0){
537              return curr;
538          }
539          else{
540              curr = curr->next;
541          }
542      }
543
544      return NULL;
545  }
```

Σχήμα 4.14: Έλεγχος εγκυρότητας πυλών

4.8 Αλγόριθμοι τροποποίησης πυλών

Οι αλγόριθμοι τροποποίησης των πυλών δημιουργήθηκαν σε συνεργασία με τους διδακτορικούς συναδέλφους που συμμετείχαν στην έρευνα. Διαχωρίσαμε τις περιπτώσεις τν πυλών και ανάλογα την πύλη, την τροποποιούσαμε ανάλογα.

Ο χρήστης το μόνο που έχει να κάνει, είναι να δώσει την ονομασία της πύλης που θα ήθελε να τροποποιήσει. Εμείς στην συνέχει ελέγχουμε αν υπάρχει αυτή η πύλη. Σε περίπτωση που δεν υπάρχει τον ειδοποιούμε και συνεχίζουμε.

Στην περίπτωση που η πύλη υπάρχει, τότε την τροποποιούμε με βάση το είδος της. Στην περίπτωση δημιουργίας νέων καλωδίων και πυλών, τότε αυτά έχουν την ονομασία τη προηγούμενης πύλης προσθέτωταν ένα χαρακτηρισμό που δείχνει ότι είναι καινούρια.

Ο τρόπος με τον οποίο αλλάξαμε πύλες βασίστηκε σε διάφορες μεταβλητές. Παρατηρούμε πως οι πύλες οι οποίες έχουν την μεγαλύτερη ευαισθησία είναι οι πύλες οι οποίες βρίσκονται πιο κοντά στα flip flops. Έτσι ξεκινήσαμε να τροποποιούμε τις πύλες κοντά στα Flip Flops επειδή είψαν την μεγαλύτερη ευαισθισία και εμείς στοχεύουμε στην μείωση. Στην συνέχεια βέβαια παρατηρήθηκε πως δεν υπάρχει στην ουσία μεγάλη βελτίωση καθώς επειδή βρίσκεται κοντά σε flip flop δεν υπάρχει ο απαραίτος χρόνος (σε πύλες) για να προλάβει το

μεταβλητό σφάλμα να καλυφθεί. Έτσι στην πορεία καταλήξαμε πως ο καλύτερος τρόπος να το βελτιώσουμε ήταν να αναλύσουμε το critical path του κυκλώματος και να τροποποιούμε τις πύλες που βρίσκονται σε αυτό και δεν έχουν τόση μεγάλη ευαισθησία στην ακτινοβολία. Με αυτό τον τρόπο επικαλύπτουμε το μεταβλητό σφάλμα απο την αρχή του μονοπατιού που μπορεί να συμβεί.

4.8.1 AND

Στην πύλη τύπου AND, η απλή προσσέγκιση που χρησιμοποιήσαμε ήταν να ενώσουμε στην θέση της δύο πύλες τύπου NAND. Αποδυκνύεται εύκολα πως το αποτέλσμα αυτής της πράξης μας δίνει το ίδιο αποτέλεσμα μέσω ενός πίνακα αληθείας. Έτσι αποφασίστηκε να τροποποιήσουμε την πύλη τύπου AND σε μία πύλη τύπου NAND με τον ίδιο αριθμό εισόδων και στην συνέχεια η έξοδος της πύλης αυτής να γίνει η είσοδος στην επόμενη βραχυκυκλωμένη NAND πύλη.

```
72  void changeANDGateWithTwoNANDS(struct gate * gate, struct gate *head){
73  struct gate *temp, *itr;
74  struct wire *newWire, *wireItr;
75  int counter;
76
77  temp = (struct gate *)malloc(sizeof(struct gate));
78  if(temp == NULL){
79      printf("Error allocating memmory for the better circuit!\n");
80      return;
81  }
82
83  //Build the pointers on the first list
84  temp->next = gate->next;
85  gate->next = temp;
86
87  //Copy information for the gate
88  strcpy(gate->gate_type,"NAND");
89  strcpy(temp->gate_type,gate->gate_type);
90  temp->layer = gate->layer;
91  temp->inputs[0] = gate->outputs[0];
92  temp->inputs[1] = gate->outputs[0];
93  strcpy(temp->gate_name, gate->gate_name);
94  strcat(temp->gate_name, "_New");
95
96
97  //now create new wire that will be the output of the new gate
98  newWire = (struct wire *)malloc(sizeof(struct wire));
99  if(newWire == NULL){
100     printf("Error allocating memmory for the better circuit!\n");
101     return;
102 }
103
104 newWire->next = gate->outputs[0]->next;
105 gate->outputs[0]->next = newWire;
106 strcpy(newWire->node_name, gate->outputs[0]->node_name);
107 strcat(newWire->node_name, "_New");
108 newWire->layer = gate->outputs[0]->layer;
109
110 //find all the gates that have the old wire as input and change it
111 itr = head;
```

Σχήμα 4.15: Τροποποίηση AND πύλης σε 2 NAND

```
112 while(itr!=NULL){
113     counter = 0;
114     wireItr = temp->inputs[counter];
115     while(wireItr != NULL){
116         if(wireItr == temp->inputs[0]){
117             itr->inputs[counter] = newWire;
118         }
119         counter++;
120         wireItr = itr->inputs[counter];
121     }
122
123     itr = itr->next;
124 }
125
126 //Attach the wire to the new NAND gate
127 temp->outputs[0] = newWire;
128
129 }
```

Σχήμα 4.16: Τροποποίηση AND πύλης σε 2 NAND

4.8.2 OR

Στην ίδια λογική γίνεται και η τροποποίησης των αδύναμων OR πυλών. Αντικαθιστούμε την OR πύλη με μία NOR πύλη με τον ίδιο αριθμό εισόδων. Στην συνέχεια προσθέτουμε μία νέα NOR πύλη βραχυκυκλωμένη με την έξοδο της παλιάς πύλης. Έτσι πάλι έχουμε την ίδια λειτουργία του υποκυκλώματος, αυξάνοντας όμως το masking.

```
131  void changeORGateWithTwoNORS(struct gate * gate, struct gate *head){
132
133  struct gate *temp, *itr;
134  struct wire *newWire, *wireItr;
135  int counter;
136
137  temp = (struct gate *)malloc(sizeof(struct gate));
138  if(temp == NULL){
139      printf("Error allocating memmory for the better circuit!\n");
140      return;
141  }
142
143  //Build the pointers on the first list
144  temp->next = gate->next;
145  gate->next = temp;
146
147  //Copy information for the gate
148  strcpy(gate->gate_type,"NOR");
149  strcpy(temp->gate_type,gate->gate_type);
150  temp->layer = gate->layer;
151  temp->inputs[0] = gate->outputs[0];
152  temp->inputs[1] = gate->outputs[0];
153  strcpy(temp->gate_name, gate->gate_name);
154  strcat(temp->gate_name, "_New");
155
156
157  //now create new wire that will be the output of the new gate
158  newWire = (struct wire *)malloc(sizeof(struct wire));
159  if(newWire == NULL){
160      printf("Error allocating memmory for the better circuit!\n");
161      return;
162  }
```

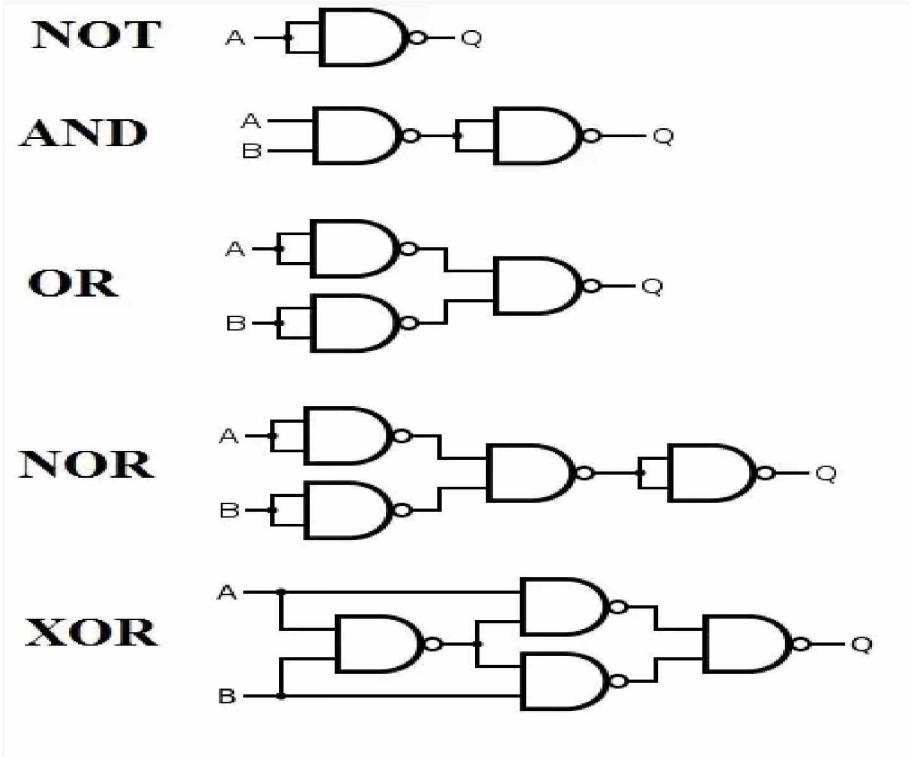
Σχήμα 4.17: Τροποποίηση OR πύλης σε 2 NOR

```
164  newWire->next = gate->outputs[0]->next;
165  gate->outputs[0]->next = newWire;
166  strcpy(newWire->node_name, gate->outputs[0]->node_name);
167  strcat(newWire->node_name, "_New");
168  newWire->layer = gate->outputs[0]->layer;
169
170  //find all the gates that have the old wire as input and change it
171  itr = head;
172  while(itr!=NULL){
173      counter = 0;
174      wireItr = temp->inputs[counter];
175      while(wireItr != NULL){
176          if(wireItr == temp->inputs[0]){
177              itr->inputs[counter] = newWire;
178          }
179          counter++;
180          wireItr = itr->inputs[counter];
181      }
182
183      itr = itr->next;
184  }
185
186  //Attach the wire to the new NOR gate
187  temp->outputs[0] = newWire;
188  }
```

Σχήμα 4.18: Τροποποίηση OR πύλης σε 2 NOR

4.8.3 Τροποποίηση με βάση τις NAND πύλες

Η ιδέα πίσω απο την τροποποίηση των πυλών ήτανε να τροποποιήσουμε τις αδύναμες πύλες. Σε περίπτωση που οι αδύναμες δεν ήταν τύπου NAND, να τις αντικαθιστούμε με NAND πύλες. Στο σχήμα παρακάτω μπορούμε να δούμε τον χάρτη που ακολουθήσαμε για να τροποποιήσουμε τις πύλες με βάση τις NAND πύλες.



Σχήμα 4.19: Τροποποίηση πυλών σε 2 NAND

Κεφάλαιο 5

Περισσότερα στοιχεία για τον κώδικα

5.1 Εισαγωγή

Στην ενότητα αυτή θα δούμε κάποια κομμάτια του κώδικα τα οποία είναι βοηθητικά και χρησιμοποιήθηκαν για αρκετές φορές στο κώδικα. Τα κομμάτια αυτά είναι για την αρχικοποίηση stings, ελευθέρωση μνήμης, καταγραφή αποτελεσμάτων και άλλα.

5.2 Μηδενισμός Strings

Δημιουργήθηκε μια συνάρτηση με την οποία μπορούμε να αρχικοποιούμε οποιουδήποτε String τα δεδομένα. Αυτό ήταν αναγκαίο λόγο της χρήσης αρκετής μνήμης. Η μνήμη θα περιέχει σκουπίδια. (Σχήμα 5.1)

5.3 Καταγραφή αποτελεσμάτων

Η καταγραφή των αποτελεσμάτων πραγματοποιείται κατά την διάρκεια εκτέλεσης του κυκλώματος. Κατά την διάρκεια εκτέλεσης του κυκλώματος καταγράφονται στατιστικά σχετικά με τις πύλες, τον τύπο τους, την τιμή που έχουν και αρκετά άλλα. Αυτό κρίνεται αναγκαίο καθώς μετά την τροποποίηση του κυκλώματος θέλουμε να ξανα εκτελέσουμε το κύκλωμα και να βεβαιώσουμε πως η λειτουργία του δεν έχει τροποποιηθεί.(Σχήμα 5.1 και 5.2)

```
80  //Set string to \0
81  void SetZero(char str[], int size){
82      int counter;
83
84      for(counter=0; counter<size; counter++){
85          str[counter] = '\0';
86      }
87
88  }
```

Σχήμα 5.1: Αρχικοποίηση Strings

```
1127  void dataToFile(char *dataStr){
1128
1129      FILE *fp;
1130
1131      fp = fopen("data/inputs_outputs_logs.txt", "a");
1132
1133
1134      /* fopen() return NULL if last operation was unsuccessful */
1135      if(fp == NULL)
1136      {
1137          /* File not created hence exit */
1138          printf("Unable to create or open file.\n");
1139          exit(EXIT_FAILURE);
1140      }
1141
1142      fprintf(fp, "%s\n", dataStr);
1143
1144      fclose(fp);
1145  }
```

Σχήμα 5.2: Εγγραφή δεδομένων σε αρχείο

```
1314  //Function that prints the gates values at each run to the file
1315  void printGateToFile(struct gate *head){
1316
1317      struct gate *curr = head;
1318      FILE *pFile;
1319
1320      pFile=fopen("data/inputs_outputs_logs.txt", "a");
1321      if(pFile==NULL) {
1322          perror("Error opening file.");
1323          return;
1324      }
1325
1326      while(curr!=NULL){
1327          fprintf(pFile, "Gate with name: %s has value of: %d\n", curr->gate_name, curr->value);
1328          curr = curr->next;
1329      }
1330      fclose(pFile);
1331  }
```

Σχήμα 5.3: Δημιουργία στατιστικών

5.4 Εκκαθάριση μνήμης

Στο τέλος της κάθε διεργασίας και αφού κριθούν κάποια αντικείμενα μη αξιοποιήσιμα πλέον για το πρόγραμμά μας, πρέπει να διαγραφούν. Για αυτό το λόγο δημιουργήθηκε μία μέθοδος να ελευθερώνει την δυναμικά δεσμευμένη μνήμη. Η κάθε συνάρτηση είναι διαφορετική και εξαρτάται από το είδος της δομής-αντικειμένου.

```
529  struct wire *FindCheck(struct wire *ptr, char stringforsearch[]){
530      struct wire *curr;
531
532      curr=ptr;
533
534      while(curr != NULL){
535
536          if(strcmp(curr->node_name, stringforsearch) == 0){
537              return curr;
538          }
539          else{
540              curr = curr->next;
541          }
542      }
543
544      return NULL;
545  }
```

Σχήμα 5.4: Έλεγχος εγκυρότητας πυλών

5.5 Makefile

Για την αυτοματοποίηση της μεταγλώτησης του κώδικα, δημιουργήθηκε ένα αρχείο make-file για να μπορούμε να κάνουμε compile καθε αρχείο χωρίς να χρειάζεται να ξέρει κάποιος τις εξαρτήσεις μεταξύ τους. Ο κώδικας για το makefile φαίνεται στο σχήμα 5.5.

```
CC = gcc
CFLAGS = -Wall
LDFLAGS =
OBJFILES = circuit.o functions.o radiation_reduction_functions.o
TARGET = circuit

all:    $(TARGET)

$(TARGET): $(OBJFILES) circuit.c functions.c radiation_reduction_functions.c
    $(CC) $(CFLAGS) -o $(TARGET) $(OBJFILES) $(LDFLAGS)

run:    circuit
    ./circuit
```

Σχήμα 5.5: Makefile

Κεφάλαιο 6

Πειραματικά αποτελέσματα και βελτιώσεις

6.1 Εισαγωγή

Στο κεφάλαιο αυτό θα δούμε τα αποτελέσματα της έρευνας και των προσομοιώσεων, σε 2 απο τα κυκλώματα που εκτελέστηκαν. Θα δούμε την τοπολογία του κυκλώματος πριν και μετά τις βελτιώσεις σε επίπεδο πυλών, καθώς και στατιστικά σχετικά με τα κυκλώματα.

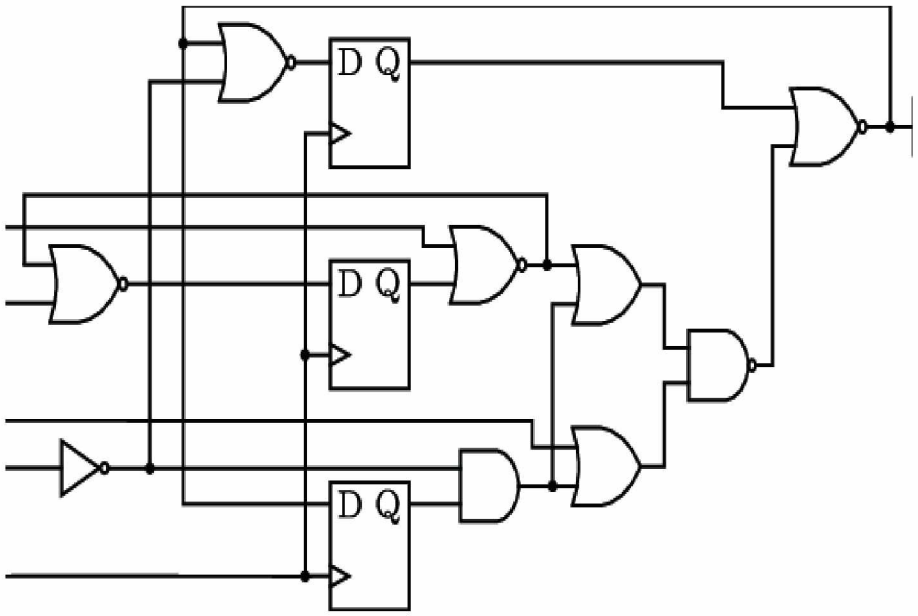
6.1.1 Πειραματικά αποτελέσματα και αρχική κατάσταση του κυκλώματος

Χρησιμοποιηθηκε το πρόγραμμα Innovus για το placement των τροποποιημενων κυκλωμάτων και την εξαγωγή της απόδοσής τους όσον αφορά το power, area, timing. Στη συνέχεια το εργαλειο MAIN_SER για την εκτίμηση του SER των τροποποιημενων κυκλωματων και τέλος το πρόγραμμα που δημιουργήθηκε στο πλαίσιο της εργασίας για να αξιολογούμε τα κυκλώματα, να τροποποιούμε τις πύλες και να βεβαιώνουμε την ίδια λειτουργία του κυκλώματος.

Κύκλωμα S27

Το κύκλωμα αυτό, αποτελείτε απο 13 πύλες και 4 D-Flip-Flops. Το σχέδιο του κυκλώματος θα μπουρούσαμε να πούμε πως είναι όπως φαίνεται στο σχήμα

Στο κύκλωμα αυτό αρχικά, από το innovus παίρνουμε το SER estimation. Χωρίς τροπο-



Σχήμα 6.1: Κύκλωμα s27

ποιήσεις μπορούμε να πούμε πως το κύκλωμα αυτό έχει πιθανότητα αποτυχίας σε περίπτωση έκθεσης του σε ακτινοβολία σε ποσοστό 0.379608%

Κύκλωμα S298

Στο s298 κύκλωμα, τα αποτελέσματα ήταν πιο αντιπροσωπευτικά λόγω του μεγαλύερου αριθμού των πυλών. Το κύκλωμα αυτό έχει 170 πύλες και 8 D-Flip-Flops. Στις προσομοιώσεις που τρέξαμε ο μέσος όρος της ευαισθησίας των πυλών στην ακτινοβολία ήταν στο 0.347329%.

Πρέπει να σημειώσουμε πως και στις δύο περιπτώσεις ο συνολικός αριθμός των πυλών που τροποποιήσαμε ήταν σε ένα συγκεκριμμένο εύρος. Τροποποιήσαμε πύλες σε ποσοστό 2-5% των συνολικών πυλών του κυκλώματος.

Κύκλωμα S344

Στο s344 κύκλωμα αποτελείται από 240 κόμβους και 15 D-Flip-Flops. Στις προσομοιώσεις που τρέξαμε ο μέσος όρος της ευαισθησίας των πυλών (Average SER) ήταν στο 0.350708%. Πρέπει να σημειώσουμε πως ο συνολικός αριθμός των πυλών που τροποποιήσαμε ήταν σε ένα συγκεκριμμένο εύρος. Στο κύκλωμα αυτό πάλι ο συνολικός αριθμός των τροποποιημένων πυλών βρίσκεται στο ίδιο ποσοστό.

Κύκλωμα S400

Το κύκλωμα s400, το μεγαλύτερο κύκλωμα που αναλύουμε σαν αποτελέσματα στην εργασία, που αποτελείται από 203 κόμβους και 21 flip flops είχε αρχική μέση ευαισθησία 0.401997%. Έτσι τροποποιήσαμε πύλες που δεν βρίσκονται στο critical path του κυκλώματος στο ίδιο εύρος τιμών και σε αυτή την περίπτωση για να δούμε τα αποτελέσματα απο τις αλλαγές.

6.1.2 Πειραματικά αποτελέσματα και τελική κατάσταση του κυκλώματος

Για την επαμαξιολόγηση των κυκλωμάτων μετά τις τροποποιήσεις που πραγματοποιήθηκαν σε αυτά χρησιμοποιήθηκε τόσο το innovus όσο και το εργαλείο που δημιουργήθηκε, για να βεβαιωθούμε την ίδια λειτουργία των κυκλωμάτων.

Κύκλωμα S27

Στο καινούριο κύκλωμα s27, τροποποιήσαμε 1 πύλη απο τις 13 που περιέχει το κύκλωμα.

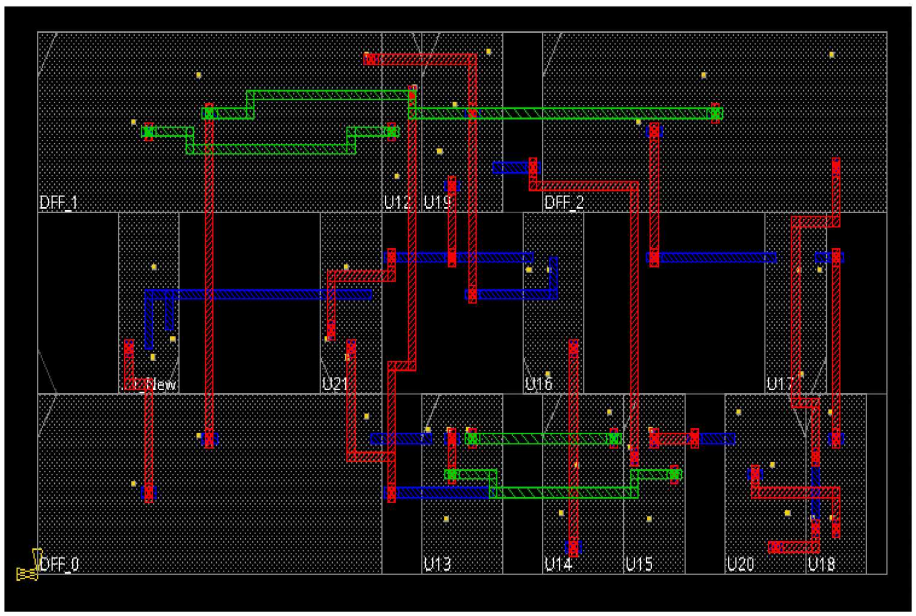
Τα ποσοστά δεν ήταν τόσο ενθαρυντικά καθώς λόγο του μικρού κυκλώματος, η αλλαγή δεν καταφέρνει να κάνει masking το soft error που δημιουργείτε. Το ποσοστό που πετύχαμε ήταν 0.434748% Η αναπαράσταση του κυκλώματος μέσα απο το πρόγραμμα innovus είναι όπως φαίνεται στο σχήμα 6.2

Κύκλωμα S298

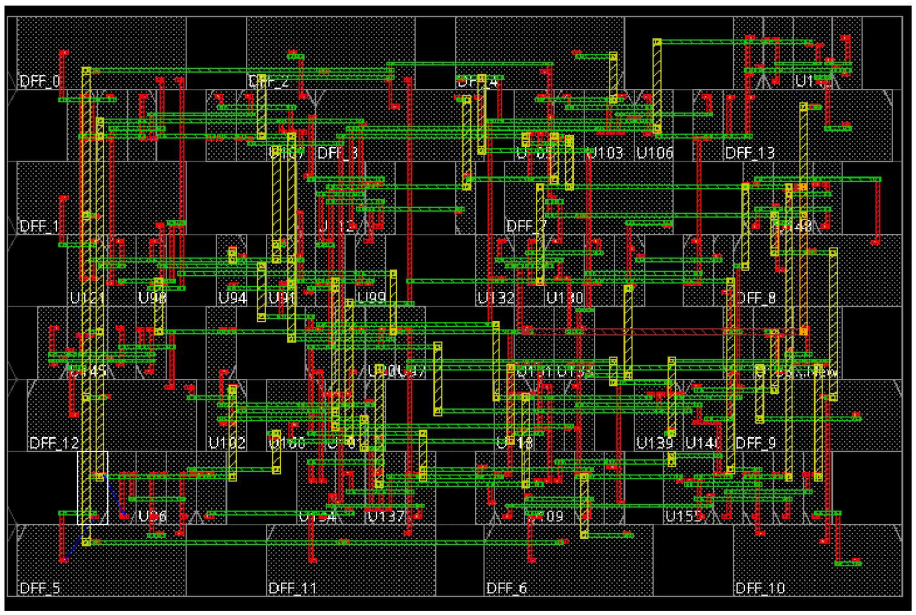
Το καινούριο κύκλωμα s298, τροποποιήσαμε 4 πύλες απο τις 116 που περιέχει το κύκλωμα στο σύνολο του.

Τα ποσοστά στην περίπτωση αυτή ήταν πολύ καλύτερα. Η βελτίωση που παρατηρήθηκε ήταν σε ποσοστό 0.02736%. Πετύχαμε μέσο ποσοστό ευαισθησίας 0.319969. Η αναπαράσταση του κυκλώματος μέσα απο το πρόγραμμα innovus είναι όπως φαίνεται στο σχήμα 6.2

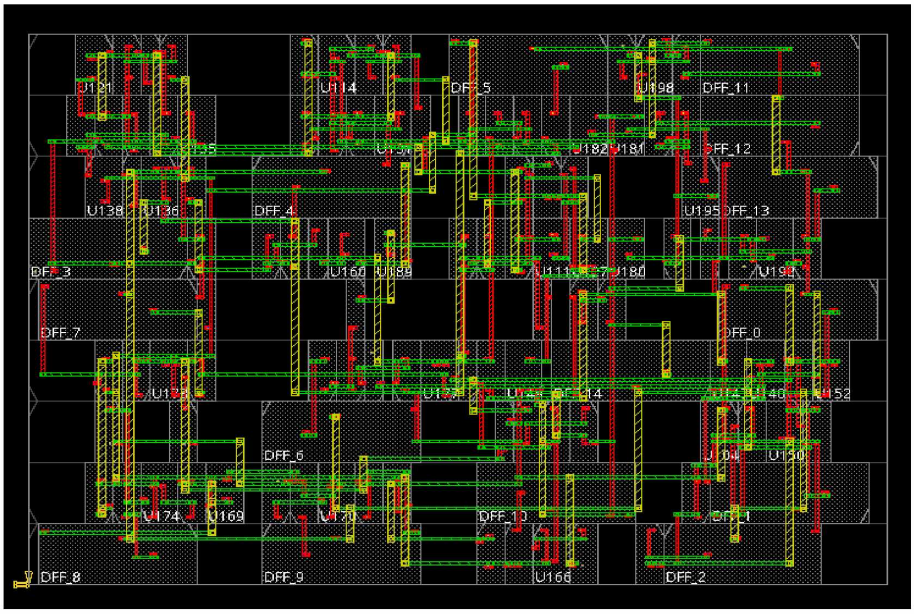
Αυτό που παρατηρήθηκε ήταν πως όσο το κύκλωμα μεγάλωνε σε αριθμό πυλών, τόσο περισσότερο μικραίναμε το ποσοστό επιρεασμού του κυκλώματος απο ένα soft error.



Σχήμα 6.2: Κύκλωμα s27 βελτιωμένο



Σχήμα 6.3: Κύκλωμα s298 βελτιωμένο



Σχήμα 6.4: Κύκλωμα s344

Κύκλωμα S344

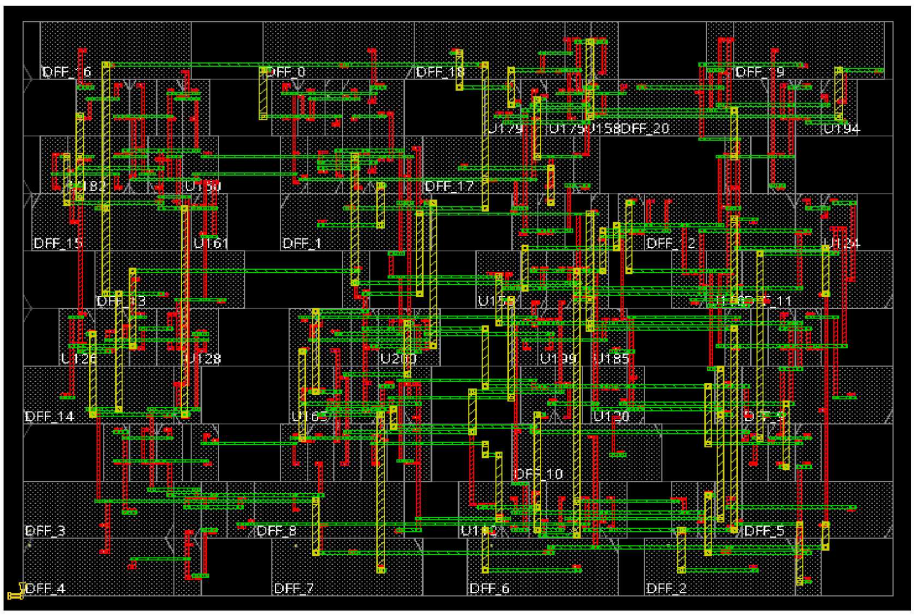
Στο καινούριο κύκλωμα s344, τροποποιήσαμε 6 πύλες απο τις 200 που περιέχει το κύκλωμα. Ποσοστό 3% του συνόλου των πυλών.

Τα ποσοστά στην περίπτωση αυτή δεν ήταν όπως αναμένονταν. Η βελτίωση που παρατηρήθηκε ήταν σε ποσοστό -0.00226%. Πετύχαμε μέσο ποσοστό ευαισθησίας 0.352968. δηλαδή παρατηρήθηκε αύξηση της ευαισθησίας. Αυτό μπορεί να οφείλεται σε αρκετούς παράγοντες, όπως το γεγονός πως οι πύλες που τροποποιήσαμε έτυχε και ήταν στο critical path του κυκλώματος. Η αναπαράσταση του κυκλώματος μέσα απο το πρόγραμμα innovus είναι όπως φαίνεται στο σχήμα 6.3

Κύκλωμα S400

Το καινούριο κύκλωμα s400, τροποποιήσαμε 8 πύλες απο τις 220 που περιέχει το κύκλωμα. Ποσοστό 4% του συνόλου των πυλών.

Στην περίπτωση αυτή, η βελτίωση ήταν μεγαλύτερη από όλες των προηγούμενων περιπτώσεων. Οι πύλες όπου τροποποιήσαμε δεν ήταν πάνω στο critical path του κυκλώματος και απήχαν απο τα Flip Flops. Έτσι η τελική μέση ευαισθησία που επιτευχθηκε ήταν 0.375566%. Υπήρξε δηλαδή μια βελτίωση της τάξης του 2,6431%



Σχήμα 6.5: Κύκλωμα s400

Κεφάλαιο 7

Συμπεράσματα

Στην εννότητα αυτή θα δούμε τα συμπεράσματα της εργασίας αυτής καθώς και μελλοντικές τις επεκτάσεις που μπορούν να πραγματοποιηθούν.

7.1 Σύνοψη και συμπεράσματα

Τα αποτελέσματα της έρευνας ήταν αρκετά ενθαρυντικά. Σε μικρά κυκλώματα δεν είναι εύκολο να βελτιωθεί η επικάλληψη ενός σφάλματος, λόγο του μικρού πλύθους πυλών. Τα κυκλώματα όσο μεγαλώνουν (σε πλύθος πυλών), τόσο περισσότερο μπορεί ένα σφαλμα που θα συμβεί σε μία πύλη να επικαλυφθεί.

Η συγκεκριμμένη έρευνα προσπαθεί να αποτελέσει ένα κομματάκι στη βελτίωση της αξιοπιστίας των κυκλωμάτων, τα οποία βρίσκονται σε ιοντίζοντα περιβάλλοντα. Τα περιβάλλοντα αυτά μπορεί να είναι πυρινικοί αντιδραστήρες, αεροπλάνα, διαστημόπλοια και πολλά άλλα.

7.2 Μελλοντικές επεκτάσεις

Σαν μελλοντική επέκταση της έρευνας, ο πρώτος στόχος θα ήταν η βελτίωση του κώδικα. Να δημιουργηθεί ακόμα πιο πολύ σε ξεχωριστά αρχεία για να είναι πιο εύκολη η επεξεργασία τους και η τροποποίηση τους. Επιπλέον, θα μπορούσαμε να υλοποιήσουμε και άλλους αλγορίθμους και ο χρήστης να επιλλέγει ποιόν απο αυτούς θα αξιοποιήσει. Ισως θα ήταν πιο βολικό η εργασία αυτή να υλοποιούταν και με μια πιο σύγχρονη-ευχρηστη γλώσσα όπως η Java. Με αυτό τον τρόπο θα μπορούσαμε να δημιουργήσουμε και ένα γραφφικό περιβάλλον

και γιατί όχι και να πραγματοποιούνταν και σύνδεση με το innovus για να έπαιρνε κατευθείαν τις πληροφορίες για την ακτινοβολία και τα αρχεία. Σε μεγαλύτερο βάθος θα μπορούσαμε να εισάγαμε και κάποιο νευρονικό δίκτυο, το οποίο να επέλεγε μόνο του ποιον αλγόριθμο θα εφάρμοζε στη κάθε πύλη.

Βιβλιογραφία

[1] Innovus Software: https://www.cadence.com/ko_KR/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html

[2] Georgios Ioannis Paliaroutis, Pelopidas Tsoumanis, Nestor Evmorfopoulos, George Dimitriou, Georgios I. Stamoulis "Placement-based SER estimation in the presence of multiple faults in combinational logic", IEEE 2017

[3] D. P. Siewiorek and R. S. Swarz (1992). Reliable Computer Systems: Design and Evaluation

[4] Daniel Limbrick: "IMPACT OF LOGIC SYNTHESIS ON THE SOFT ERROR RATE OF DIGITAL INTEGRATED CIRCUITS", 2012.

[5] Aiman H. El-Maleh, Member, IEEE, and Khaled A. K. Daud, "Simulation-Based Method for Synthesizing Soft Error Tolerant Combinational Circuits", 2014.

[6] Mohammad Reza Rohanipoor, Behnam Ghavami, Mohsen Raji, "Improving Combinational Circuit Reliability against Multiple Event Transients via a Partition and Restructuring Approach", 2018 IEEE International Conference on Engineering and Technology (ICETECH), IEEE 2018.

[7] Bradley T. Kiddie and William H. Robinson, "Alternative Standard Cell Placement Strategies for Single-Event Multiple-Transient Mitigation", IEEE 2014, Computer Society Annual Symposium on VLSI.

[8] G. I. Paliaroutis, P. Tsoumanis, N. Evmorfopoulos, G. Dimitriou and G. I. Stamoulis, "Multiple Transient Faults in Combinational Logic with Placement Considerations,"

2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST), 2019, pp. 1-4, doi: 10.1109/MOCAST.2019.8741538

[9] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookre- son, A. Vo, S. Mitra, B. Gill, and J. Maiz, "Radiation-induced soft error rates of advanced CMOS bulk devices", 2006 IEEE International Reliability Physics Symposium Proceedings, pp. 217–225.

[10] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," IEEE Transactions on Device and Materials Reliability, 2005.

[11] H. Cai, K. Liu, L. A. de Barros Naviner, Y. Wang, M. Slimani, and J.-F. Naviner, "Efficient reliability evaluation methodologies for combinational circuits," Microelectron. Reliab., vol. 64, pp. 19–25, 2016.

[12] S. Mittal and J. S. Vetter, "A survey of techniques for modeling and improving reliability of computing systems," IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 4, pp. 1226–1238, 2016.

[13] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust system design with built-in soft-error resilience," Computer, vol. 38, no. 2, pp. 43–52, Feb. 2005.

[14] G. E. Moore, "Cramming more components onto integrated circuits," Electronics, vol. 38, no. 8, pp. 114–117, Apr. 1965

[15] Kai-Chiang Wu, Student Member, IEEE, and Diana Marculescu, Senior Member, IEEE, A Low-Cost, Systematic Methodology for Soft Error Robustness of Logic Circuits, 2013

[16] R. Baumann, "Soft errors in advanced computer systems," IEEE De- sign Test Comput., vol. 22, no. 3, pp. 258–266, May 2005

[17] T. Karnik, P. Hazuchu, and J. Patel, "Characterization of soft errors caused by single event upsets in CMOS processes," IEEE Trans. De- pend. Secure Comput., vol. 1, no. 2, pp. 128–143, Apr.–Jun. 2004.

[18] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of com- binational logic," in Proc. Int. Conf. Depend. Syst. Netw. (DSN), 2002, pp. 389–399.

[19] K. Mohanram and N. A. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits," in Proc. Int. Test Conf. (ITC), 2003, pp. 893–901.

[20] Q. Zhou and K. Mohanram, "Gate sizing to radiation harden combina- tional logic," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD), vol. 25, no. 1, pp. 155–166, Jan. 2006.

[21] Y. S. Dhillon, A. U. Diril, A. Chatterjee, and A. D. Singh, "Analysis and optimization of nanometer CMOS circuits for soft-error tolerance," IEEE Trans. Very Large Scale Integr. Syst. (TVLSI), vol. 14, no. 5, pp. 514–524, May 2006.

ΠΑΡΑΡΤΗΜΑΤΑ

Παράρτημα Α

Εγκατάσταση και εκτέλεση του προγράμματος

Για να εκτελέσει κάποιος το λογισμικό αρκεί να το κατεβάσει απο το Github: git clone `https://github.com/gepapageorgiou22/reduction-algorithm-of-the-Soft-Error-Rate-SER-of-integrated-circuits`

Χρειάζεται να έχουμε εγκατεστημένο τον gcc compiler (εμεις είχαμε Apple clang version 13.1.6). Στην συνέχεια πάμε στην τοποθεσία όπου κατεβάσαμε το λογισμικό μας, και στην συνέχεια εκτελούμε το Makefile. Εκτελούμε make all για να κάνουμε compile και make run για να τ εκτελέσουμε. Δίνουμε τα ορίσματα που ζητάει το πρόγραμμα ανάλογα το κύκλωμα καθώς και τις ονομασίες των πυλών των οποίων θέλουμε να τροποποιήσουμε. Ακόμα θα χρειαστεί κανείς και κάποιο πρόγραμμα ανάγνωσης αρχείων για να τροποποιήσει και να ενεργοποιήσει ποιος αλγόριθμος βελτιστοποίησης του κυκλώματος θα τρέξει, καθώς και ποιό αρχείο verilog θα διαβάσει.

Α.1 Παράρτημα

Στο παράρτημα θα δούμε συνοδευτικό, υποστηρικτικό υλικό (πίνακες, φωτογραφίες, στατιστικά στοιχεία, αποδείξεις, περιγραφές λογισμικών προγραμμάτων, παραδείγματα, περιγραφές πολύπλοκων διαδικασιών, λίστα με πρωτογενή στοιχεία, λεπτομερής περιγραφή και προδιαγραφές εξοπλισμού, οδηγίες εγκατάστασης λογισμικού, κ.λπ.).

Α.1.1 Πίνακες

Ο Πίνακας Α.1Περιέχει τις πληροφορίες σχετικά με τα κυκλώματα που μελετήθηκαν κατα την διάρκεια της εργασίας. `tabular`.

Πίνακας Α.1: Παράμετροι πειραμάτων

Πίνακας κυκλωμάτων			
Όνομα κυκλώματος	Αριθμός κόμβων	Αριθμός πυλών	D-Flip Flops
S27	17	13	4
S208	149	139	8
S298	169	166	14
S386	284	177	6
S344	240	231	21
S349	224	215	21
S400	203	200	21
S444	211	208	21
S526	280	277	21
S420	252	233	16
S510	293	274	6
S832	457	429	5
S820	443	425	5
S641	517	482	19
S713	539	504	19
S953	496	480	29
S1238	768	754	18
S1196	762	748	18
S9234	7002	6983	228
S13207	9608	9577	669

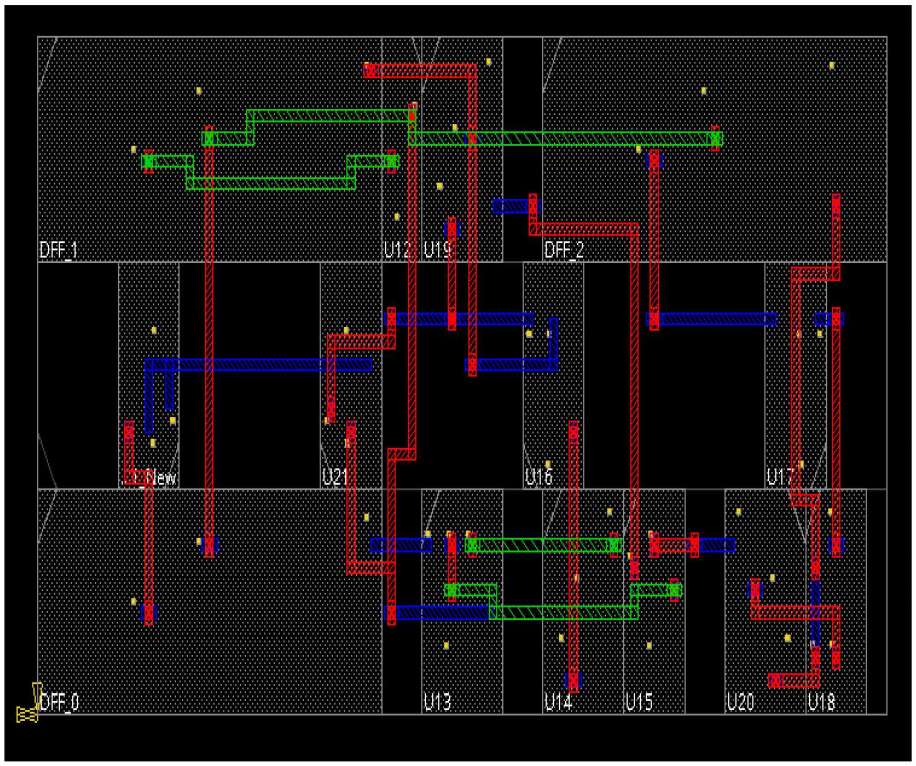
A.1.2 Μαθηματικές εκφράσεις

Ακολουθεί αναφορά σε μαθηματικές εκφράσεις που αξιοποιήθηκαν.

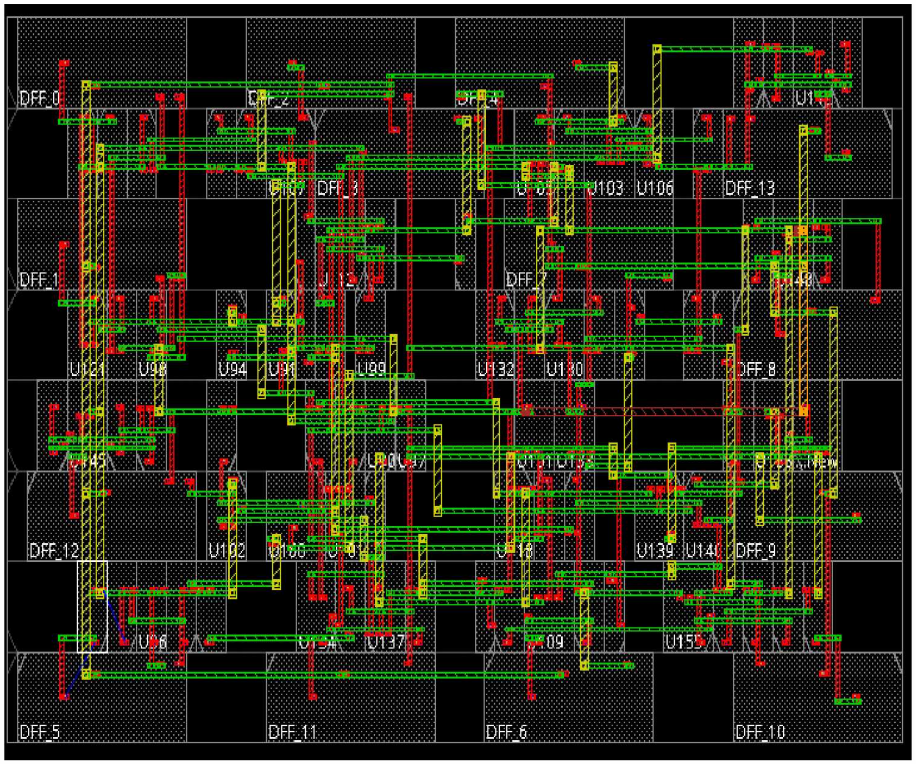
$$
\begin{aligned}
&P(\mathcal{A} \cap \mathcal{D} \cap \mathcal{T}) \\
&\quad = P(\mathcal{D} \cap \mathcal{T}) = P(\mathcal{T} \mid \mathcal{D}) \cdot P(\mathcal{D}) \\
&\quad = \sum_k (\mathrm{P}\left(t \in [T + t_{\text{hold}} - t_p - D, T - t_{\text{setup}} \right. \\
&\qquad \left. - t_p] \mid D = D_k) \cdot \mathrm{P}\left(D = D_k\right)) \\
&\quad = \sum_k \left(\frac{D_k - (t_{\text{setup}} + t_{\text{hold}})}{T_{\text{clk}} - d_{\text{init}}} \cdot \mathrm{P}\left(D = D_k\right) \right)
\end{aligned}
$$

Α.1.3 Σχήματα

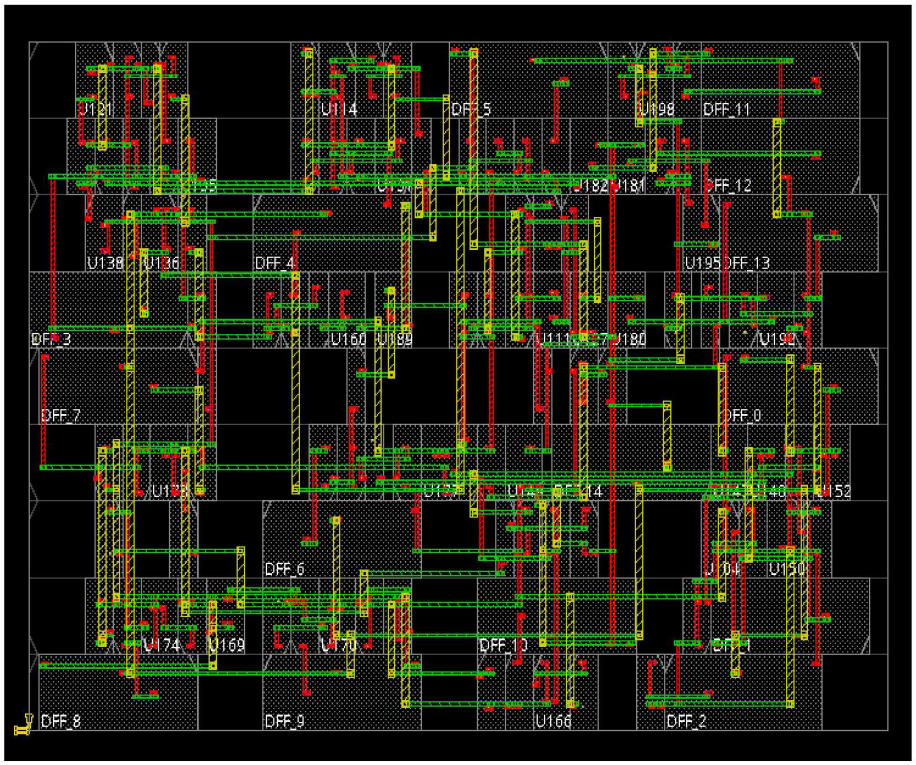
Ακολουθούν τα Σχήματα, τα βελτιωμένα verilog κυκλώματα.



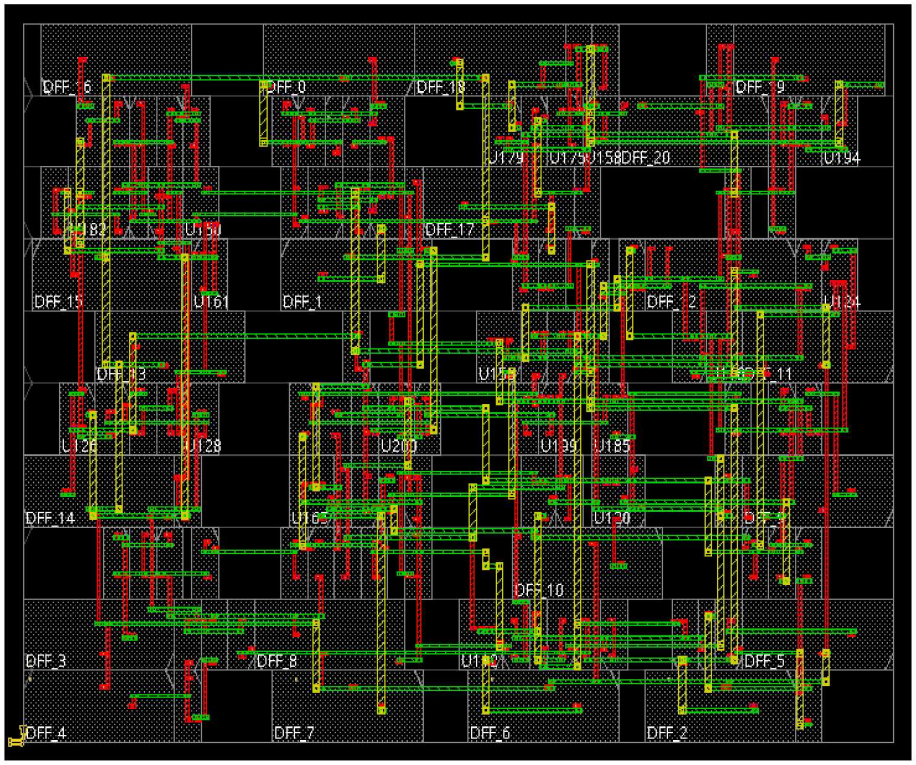
Σχήμα Α.1: S27 circuit in layers



Σχήμα Α.2: S298 circuit in layers



Σχήμα Α.3: S344 circuit in layers



Σχήμα Α.4: S400 circuit in layers

```
72  void changeANDGateWithTwoNANDS(struct gate * gate, struct gate *head){
73  struct gate *temp, *itr;
74  struct wire *newWire, *wireItr;
75  int counter;
76
77  temp = (struct gate *)malloc(sizeof(struct gate));
78  if(temp == NULL){
79      printf("Error allocating memmory for the better circuit!\n");
80      return;
81  }
82
83  //Build the pointers on the first list
84  temp->next = gate->next;
85  gate->next = temp;
86
87  //Copy information for the gate
88  strcpy(gate->gate_type,"NAND");
89  strcpy(temp->gate_type,gate->gate_type);
90  temp->layer = gate->layer;
91  temp->inputs[0] = gate->outputs[0];
92  temp->inputs[1] = gate->outputs[0];
93  strcpy(temp->gate_name, gate->gate_name);
94  strcat(temp->gate_name, "_New");
95
96
97  //now create new wire that will be the output of the new gate
98  newWire = (struct wire *)malloc(sizeof(struct wire));
99  if(newWire == NULL){
100     printf("Error allocating memmory for the better circuit!\n");
101     return;
102 }
103
104 newWire->next = gate->outputs[0]->next;
105 gate->outputs[0]->next = newWire;
106 strcpy(newWire->node_name, gate->outputs[0]->node_name);
107 strcat(newWire->node_name, "_New");
108 newWire->layer = gate->outputs[0]->layer;
109
110 //find all the gates that have the old wire as input and change it
111 itr = head;
```

Σχήμα A.5: AND gate to 2 NAND gates

A.1.4 Αλγόριθμοι τροποποίησης πυλών

Στην υποενότητα αυτή βλέπουμε στιγμιότυπα απο τον κώδικα που υλοποιεί την τροποποίηση των πυλών τύπου AND, OR και NAND.

```
112 while(itr!=NULL){
113     counter = 0;
114     wireItr = temp->inputs[counter];
115     while(wireItr != NULL){
116         if(wireItr == temp->inputs[0]){
117             itr->inputs[counter] = newWire;
118         }
119         counter++;
120         wireItr = itr->inputs[counter];
121     }
122
123     itr = itr->next;
124 }
125
126 //Attach the wire to the new NAND gate
127 temp->outputs[0] = newWire;
128
129 }
```

Σχήμα Α.6: AND gate to 2 NAND gates

```
131 void changeORGateWithTwoNORS(struct gate * gate, struct gate *head){
132
133 struct gate *temp, *itr;
134 struct wire *newWire, *wireItr;
135 int counter;
136
137 temp = (struct gate *)malloc(sizeof(struct gate));
138 if(temp == NULL){
139     printf("Error allocating memmory for the better circuit!\n");
140     return;
141 }
142
143 //Build the pointers on the first list
144 temp->next = gate->next;
145 gate->next = temp;
146
147 //Copy information for the gate
148 strcpy(gate->gate_type,"NOR");
149 strcpy(temp->gate_type,gate->gate_type);
150 temp->layer = gate->layer;
151 temp->inputs[0] = gate->outputs[0];
152 temp->inputs[1] = gate->outputs[0];
153 strcpy(temp->gate_name, gate->gate_name);
154 strcat(temp->gate_name, "_New");
155
156
157 //now create new wire that will be the output of the new gate
158 newWire = (struct wire *)malloc(sizeof(struct wire));
159 if(newWire == NULL){
160     printf("Error allocating memmory for the better circuit!\n");
161     return;
162 }
```

Σχήμα Α.7: OR gate to 2 NOR gates

```
164   newWire->next = gate->outputs[0]->next;
165   gate->outputs[0]->next = newWire;
166   strcpy(newWire->node_name, gate->outputs[0]->node_name);
167   strcat(newWire->node_name, "_New");
168   newWire->layer = gate->outputs[0]->layer;
169
170   //find all the gates that have the old wire as input and change it
171   itr = head;
172   while(itr!=NULL){
173       counter = 0;
174       wireItr = temp->inputs[counter];
175       while(wireItr != NULL){
176           if(wireItr == temp->inputs[0]){
177               itr->inputs[counter] = newWire;
178           }
179           counter++;
180           wireItr = itr->inputs[counter];
181       }
182
183       itr = itr->next;
184   }
185
186   //Attach the wire to the new NOR gate
187   temp->outputs[0] = newWire;
188   }
```

Σχήμα Α.8: OR gate to 2 NOR gates

```
194   void changeNandWithAndInverter(struct gate * gate, struct gate *head){
195       struct gate *temp, *itr;
196       struct wire *newWire, *wireItr;
197       int counter;
198
199       temp = (struct gate *)malloc(sizeof(struct gate));
200       if(temp == NULL){
201           printf("Error allocating memmory for the better circuit!\n");
202           return;
203       }
204
205       //Build the pointers on the first list
206       temp->next = gate->next;
207       gate->next = temp;
208
209       //Copy information for the gate
210       strcpy(gate->gate_type,"AND");
211       strcpy(temp->gate_type, "Inverter");
212       temp->layer = gate->layer;
213       temp->inputs[0] = gate->outputs[0];
214       strcpy(temp->gate_name, gate->gate_name);
215       strcat(temp->gate_name, "_New");
216
217       //now create new wire that will be the output of the new gate
218       newWire = (struct wire *)malloc(sizeof(struct wire));
219       if(newWire == NULL){
220           printf("Error allocating memmory for the better circuit!\n");
221           return;
222       }
223
224       newWire->next = gate->outputs[0]->next;
225       gate->outputs[0]->next = newWire;
226       strcpy(newWire->node_name, gate->outputs[0]->node_name);
227       strcat(newWire->node_name, "_New");
228       newWire->layer = gate->outputs[0]->layer;
229
```

Σχήμα Α.9: NAND gate to AND gate and Inverter

```
230     //find all the gates that have the old wire as input and change it
231     itr = head;
232     while(itr!=NULL){
233         counter = 0;
234         wireItr = temp->inputs[counter];
235         while(wireItr != NULL){
236             if(wireItr == temp->inputs[0]){
237                 itr->inputs[counter] = newWire;
238             }
239             counter++;
240             wireItr = itr->inputs[counter];
241         }
242
243         itr = itr->next;
244     }
245
246 //Attach the wire to the new NOR gate
247 temp->outputs[0] = newWire;
248
249 }
```

Σχήμα Α.10: NAND gate to AND gate and Inverter