ΠΑΝΕΠΙΣΤΗΜΙΟ ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΒΙΟΙΑΤΡΙΚΗ

Ανάπτυξη λογισμικού βασισμένου σε κυψελιδικά αυτόματα για την παραγωγή διανυσμάτων δοκιμής (test vectors) για ψηφιακά ηλεκτρονικά κυκλώματα/συστήματα

Δημήτρης Φελεκίδης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος

Κακαρούντας Αθανάσιος

Επίκουρος Καθηγητής

Λαμία 2010

ΠΡΩΤΗ ΣΕΛΙΔΑ

ΠΡΟΛΟΓΟΣ

Η επιστήμη των υπολογιστών και της πληροφορικής μας απασχολεί όλους καθημερινά είτε σε μικρό είτε σε μεγάλο βαθμό. Χρησιμοποιούμε τα διάφορα είδη της τεχνολογίας, από το τηλεχειριστήριο της τηλεόρασης μέχρι τον φορητό υπολογιστή μας χωρίς να σκεφτόμαστε τα στάδια που έχουν πραγματοποιηθεί για να φτάσουν αυτά τα προϊόντα στα χέρια μας. Ένα από αυτά τα στάδια είναι και ο έλεγχος σωστής λειτουργίας (testing) το οποίο και είναι από τα πιο σημαντικά.

Η πρόκληση να δημιουργήσω από το μηδέν μια εφαρμογή που μπορεί να χρησιμοποιηθεί σε αυτόν τον τομέα ήταν και ο λόγος για τον οποίο επέλεξα την συγκεκριμένη πτυχιακή. Ύστερα από πολύμηνη και επίπονη ενασχόληση, το αποτέλεσμα είναι το προτεινόμενο λογισμικό το οποίο προσθέτει ένα επιπλέον λιθαράκι στον επιστημονικό κλάδο του testing και ελπίζω ότι θα χρησιμοποιηθεί στο μέλλον για άλλες ερευνητικές εργασίες.

Κλείνοντας θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κύριο Αθανάσιο Κακαρούντα που μου εμπιστεύτηκε την συγκεκριμένη εργασία και με βοήθησε σε όλη την διάρκεια της εκπόνησης της καθώς επίσης και το σύνολο των καθηγητών του ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ που με βοήθησαν να αποκτήσω εξαιρετικά σημαντικές γνώσεις στην διάρκεια των σπουδών μου. Ξεχωριστές ευχαριστίες οφείλω στην συγκάτοικό μου και στον αδερφό μου που μου συμπαραστάθηκαν στις δύσκολες ώρες.

Αυτή η εργασία είναι αφιερωμένη στους γονείς μου , Γεωργία και Κώστα.

«Το να σου δίνουν κάτι έτοιμο και εσύ απλά
να το χρησιμοποιείς σου δίνει μια δόση απόλαυσης.
Το να δημιουργείς κάτι από την αρχή και να το
χρησιμοποιείς, σου δίνει την μεγαλύτερη απόλαυση»
Δ.Φ

Εύχομαι καλή ανάγνωση σε όλους

Δημήτρης Φελεκίδης

ΠΕΡΙΛΗΨΗ

Στην παρούσα πτυχιακή θα πραγματευτούμε την ανάπτυξη λογισμικού το οποίο θα βασίζεται στις ιδιότητες των κυψελιδικών αυτομάτων, για την παραγωγή διανυσμάτων δοκιμής για ηλεκτρονικά κυκλώματα-συστήματα. Στην αρχή γίνεται μία εισαγωγή στον χώρο του testing και παρουσιάζονται εκτενώς τα στάδια, ο σκοπός, τα πλεονεκτήματα και τα μειονεκτήματα του. Στην συνέχεια ο αναγνώστης πληροφορείται για τις μεθόδους με τις οποίες μπορεί να γίνει η διαδικασία του testing καθώς και για τις διαθέσιμες λύσεις που υπάρχουν σήμερα. Έπειτα γίνεται μια εισαγωγική παρουσίαση των κυψελιδικών αυτομάτων και δίνονται οι βασικοί ορισμοί για τις ιδιότητες και τις ιδιαιτερότητες τους.

Μετά το τέλος τις παρουσίασης όλων των εισαγωγικών πληροφοριών που καλείται να κατανοήσει ο αναγνώστης γίνεται αναφορά στην πρώτη από τις τρεις ερευνητικές εργασίες στις οποίες στηρίχθηκε η παρούσα πτυχιακή, η οποία αναλύει τον τρόπο με τον οποίο ένας δυαδικός μετρητής μπορεί να παράγει διανύσματα δοκιμής. Καταγράφονται τα θετικά και τα αρνητικά μιας τέτοιας μεθόδου και γίνεται λεπτομερής ανάλυση των ιδιοτήτων της. Στο επόμενο κεφάλαιο γίνεται αναφορά στην δεύτερη έρευνα και παρουσιάζεται μια προσπάθεια παραγωγής ακολουθιών δοκιμής με χρήση των κυψελιδικών αυτομάτων. Λόγω της παλαιότητας της έρευνας και με δεδομένο ότι εκείνη την χρονική στιγμή οι ιδιότητες συγκεκριμένων κανόνων των κυψελιδικών δεν είχαν ανακαλυφθεί, στο συγκεκριμένο κεφάλαιο γίνεται απλώς μια παρουσίαση των προτεινόμενων διαδικασιών παραγωγής ακολουθιών δοκιμής. Στην συνέχεια γίνεται αναφορά στην τελευταία έρευνα η οποία μας παρουσιάζει με πιο αναλυτικό τρόπο τα 2-Δ κυψελιδικά αυτόματα και των ιδιοτήτων τους. Μετά την παρουσίαση αυτών των τριών κεφαλαίων ο αναγνώστης έχει πλέον το απαραίτητο υπόβαθρο να κατανοήσει και να αξιολογήσει την προτεινόμενη μεθοδολογία η οποία παρουσιάζεται στο επόμενο κεφάλαιο.

Η προτεινόμενη μεθοδολογία αφορά την παραγωγή διανυσμάτων δοκιμής με χρήση απαριθμητών με βάση το 2 κάνοντας χρήση των ιδιοτήτων μερικών κανόνων των 2-Δ κυψελιδικών αυτομάτων. Σε αυτό το κεφάλαιο γίνεται αναφορά των περιπτώσεων χρήσης, του διαγράμματος ροής των εργασιών καθώς επίσης παρουσιάζεται αναλυτικά ο κώδικας που δημιουργήθηκε γι αυτόν τον σκοπό. Τέλος, γίνεται αξιολόγηση της προτεινόμενης προσέγγισης και σχολιάζονται τα συμπεράσματα που προκύπτουν.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Δοκιμαστηκότητα, Διανύσματα δοκιμής, Αυτόματη γεννήτρια προτύπων δοκιμής, Κυψελιδικά αυτόματα

ABSTRACT

In the present thesis, a thorough analysis of the software development is presented, which is based on several properties of 2D cellular automata for generating test vectors for electronic circuits/systems. At the beginning, there is an analysis of the extensive topic of testing and a presentation of its stages, goal, advantages and disadvantages. Then, the reader can read for the methods of conducting the testing process, as well as for the state-of-the-art in the topic. An introductive presentation of cellular automata, their fundamental definitions and properties is given.

After the presentation of the basic scientific background, needed for a comprehensive reading of the thesis, a report on the first of the three research papers, on which the thesis was based on, is offered. That paper analyses the way that a counter may be used to generate the required test vectors. The advantages and disadvantages of such a concretization are recorded and an in-detail analysis of its attributes is performed. In the following chapter, there is a report on the second research paper, in which there is a presentation of an effort to generate test sequences using cellular automata. Due to the precocious stage of the research on cellular automata at that time, and taking into consideration the decade of the research, at this chapter a summary of the proposed processes is offered. The presentation of the third research paper follows, in which a more analytic aspect of cellular automata and their properties is offered. After the presentation of those three chapters, a new methodology for generating a matrix of test vectors is proposed, exploiting the properties of cellular automata.

The proposed methodology has to do with the generation of test vectors with the use of radix-2 base counters using the properties of cellular automata. In this chapter the use cases, the flow chart and the code which was developed for this scope is presented. Finally, an evaluation of the proposed approach is offered.

KEYWORDS

Testability, Test vectors, Automatic Test Pattern Generator, Cellular Automata

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ 1

1.1 Η ΦΑΣΗ ΤΗΣ ΔΟΚΙΜΗΣ (TESTING) ΣΤΗΝ ΑΝΑΠΤΥΞΗ ΜΙΑΣ ΕΦΑΡΜΟΓΗΣ 1
1.2 ΚΡΙΣΙΜΕΣ ΕΦΑΡΜΟΓΕΣ ΚΑΙ Η ΑΝΑΓΚΗ ΓΙΑ ΥΨΗΛΟ ΕΠΙΠΕΔΟ ΑΞΙΟΠΙΣΤΙΑΣ 2
1.3 ΓΕΝΝΗΤΡΙΕΣ ΠΡΟΤΥΠΩΝ ΔΟΚΙΜΗΣ 3
1.4 ΚΥΨΕΛΙΔΙΚΑ ΑΥΤΟΜΑΤΑ 3
1.5 ΣΤΟΧΟΣ ΤΗΣ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ 4

2. ΔΟΚΙΜΗ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ 5

2.1 ΑΥΤΟΜΑΤΗ ΠΑΡΑΓΩΓΗ ΠΡΟΤΥΠΩΝ ΔΟΚΙΜΗΣ (ATPG - AUTOMATIC-TEST-PATTERN-GENERATION) 5
2.2 ΒΑΣΙΚΟΙ ΟΡΙΣΜΟΙ 6
2.3 ΣΚΟΠΟΣ ΤΩΝ ATPG 8
2.4 ΕΝΣΩΜΑΤΩΣΗ ΑΥΤΟ-ΔΟΚΙΜΗΣ (BIST – BUILT-IN SELF TEST) 9
2.5 ΔΙΑΘΕΣΙΜΕΣ ΛΥΣΕΙΣ ΣΤΗΝ ΔΙΕΘΝΗ ΑΓΟΡΑ 10
2.5.1. SOFTWARE SOLUTIONS 10

3. ΚΥΨΕΛΙΔΙΚΑ ΑΥΤΟΜΑΤΑ 13

4. ΠΑΡΑΓΩΓΗ ΔΙΑΝΥΣΜΑΤΩΝ ΔΟΚΙΜΗΣ ΑΠΟ ΜΕΤΡΗΤΗ 17

4.1 ΠΟΙΕΣ ΕΙΝΑΙ ΟΙ ΤΕΧΝΙΚΕΣ ΠΟΥ ΑΠΑΙΤΟΥΝΤΑΙ ΓΙΑ ΤΗΝ ΙΚΑΝΟΠΟΙΗΣΗ ΤΩΝ ΣΤΟΧΩΝ 17
4.2 ΠΙΝΑΚΕΣ ΔΟΚΙΜΗΣ ΧΩΡΙΣ ΑΔΙΑΦΟΡΕΣ ΚΑΤΑΣΤΑΣΕΙΣ 22
4.3 ΠΟΙΑ ΤΑ ΜΕΙΟΝΕΚΤΗΜΑΤΑ ΧΡΗΣΗΣ ΤΟΥ 24

5. ΚΥΨΕΛΙΔΙΚΑ ΑΥΤΟΜΑΤΑ ΓΙΑ ΤΗΝ ΠΑΡΑΓΩΓΗ ΚΑΘΟΡΙΣΜΕΝΩΝ ΑΚΟΛΟΥΘΙΩΝ ΔΟΚΙΜΗΣ 25

5.1 ΔΥΟ ΔΙΑΔΙΚΑΣΙΕΣ ΣΥΝΔΕΣΗΣ 26
5.1.1. Διαδικασία μετατόπισης ακολουθιών 26
5.1.2. Διαδικασία Target Sequence 28

6. ΥΠΟΛΟΓΙΣΤΙΚΕΣ ΑΚΟΛΟΥΘΙΕΣ ΒΑΣΙΣΜΕΝΕΣ ΣΤΑ ΚΥΨΕΛΙΔΙΚΑ ΑΥΤΟΜΑΤΑ 30

7. ΠΡΟΤΕΙΝΟΜΕΝΗ ΜΕΘΟΔΟΛΟΓΙΑ ΓΙΑ ΠΑΡΑΓΩΓΗ ΔΙΑΝΥΣΜΑΤΩΝ ΔΟΚΙΜΗΣ ΜΕ RADIX-2 ΑΠΑΡΙΘΜΗΤΕΣ ΒΑΣΙΣΜΕΝΟΥΣ ΣΕ ΚΥΨΕΛΙΔΙΚΑ ΑΥΤΟΜΑΤΑ 34

7.1 ΠΡΟΔΙΑΓΡΑΦΗ ΑΠΑΙΤΗΣΕΩΝ 34
7.2 ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ ΕΡΓΑΣΙΩΝ 35
7.3 ΑΝΑΛΥΣΗ ΠΕΡΙΠΤΩΣΕΩΝ ΧΡΗΣΗΣ 35
7.4 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ 36
7.4.1. Πρώτο βήμα-Εκτύπωση κανόνα 102 36
7.4.2. Δεύτερο Βήμα-Παραγωγή συμπιεσμένου πίνακα κυψελιδικού 37
7.4.3. Τρίτο Βήμα-Δημιουργία πίνακα δοκιμής 39
7.4.4. Τέταρτο Βήμα-Δημιουργία συμπιεσμένου πίνακα δοκιμής 44
7.4.5. Πέμπτο Βήμα-Παρουσίαση Αποτελεσμάτων 46

8. ΑΞΙΟΛΟΓΗΣΗ ΠΡΟΤΕΙΝΟΜΕΝΗΣ ΠΡΟΣΕΓΓΙΣΗΣ 49

9. ΣΥΜΠΕΡΑΣΜΑΤΑ 59

10. ΠΗΓΕΣ 60

ΠΑΡΑΡΤΗΜΑ Α 61

ΛΙΣΤΑ ΣΧΗΜΑΤΩΝ

ΣΧΗΜΑ 1:ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ TETRAMAX ATPG –ΠΗΓΗ SYNOPSIS 6
ΣΧΗΜΑ 2: ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ LOGIC BIST 10
ΣΧΗΜΑ 3:ΛΙΣΤΑ ΚΑΝΟΝΩΝ CA ΑΠΟ 0 ΕΩΣ 127 14
ΣΧΗΜΑ 4:ΛΙΣΤΑ ΚΑΝΟΝΩΝ CA ΑΠΟ 128 ΕΩΣ 255 15
ΣΧΗΜΑ 5: ΑΡΙΣΤΕΡΑ-ΕΠΑΝΑΛΑΜΒΑΝΟΜΕΝΗ, ΔΕΞΙΑ-ΧΑΟΤΙΚΗ 16
ΣΧΗΜΑ 6:CA ΜΕ ΜΟΝΟΤΟΝΙΚΗ ΣΥΜΠΕΡΙΦΟΡΑ 31
ΣΧΗΜΑ 7: CA 20 ΒΗΜΑΤΩΝ ΜΕ ΚΑΘΟΡΙΣΜΕΝΟ ΣΧΕΔΙΟ 31
ΣΧΗΜΑ 9: ΕΞΑΓΩΓΗ ΤΗΣ ΔΥΑΔΙΚΗΣ ΑΚΟΛΟΥΘΙΑΣ ΜΕ ΤΗΝ ΠΑΡΑΛΕΙΨΗ ΜΕΡΙΚΩΝ ΒΗΜΑΤΩΝ 32
ΣΧΗΜΑ 8:CA 20 ΒΗΜΑΤΩΝ ΜΕ ΕΠΑΝΑΛΑΜΒΑΝΟΜΕΝΗ ΣΥΜΠΕΡΙΦΟΡΑ ΚΑΙ ΜΕΤΑΒΑΛΛΟΜΕΝΗ ΕΞΟΔΟΣ 32

ΛΙΣΤΑ ΠΙΝΑΚΩΝ

ΠΙΝΑΚΑΣ 1:ΈΝΑΣ ΠΙΝΑΚΑΣ ΔΟΚΙΜΗΣ ΜΕ ΚΥΚΛΙΚΗ ΑΠΟΣΤΑΣΗ 128 (Α) 19
ΠΙΝΑΚΑΣ 2:ΜΕ ΤΗΝ ΣΥΓΧΩΝΕΥΣΗ ΙΔΙΩΝ ΣΤΗΛΩΝ, Η ΚΥΚΛΙΚΗ ΑΠΟΣΤΑΣΗ ΤΟΥ ΠΙΝΑΚΑ (Α) ΓΙΝΕΤΑΙ 9 (Β) 19
ΠΙΝΑΚΑΣ 3:ΜΕ ΤΗΝ ΣΥΜΠΛΗΡΩΜΑΤΙΚΗ ΣΥΓΧΩΝΕΥΣΗ ΣΤΗΛΩΝ Η ΚΥΚΛΙΚΗ ΑΠΟΣΤΑΣΗ ΤΟΥ 19
ΠΙΝΑΚΑΣ 4: ΜΕ ΤΗΝ ΜΕΤΑΛΛΑΓΗ ΣΤΗΛΩΝ, Η ΚΥΚΛΙΚΗ ΑΠΟΣΤΑΣΗ ΤΟΥ ΠΙΝΑΚΑ (Α) ΓΙΝΕΤΑΙ 5. (Δ) 20
ΠΙΝΑΚΑΣ 5: ΜΕ ΤΗΝ ΣΥΜΠΛΗΡΩΜΑΤΙΚΗ ΠΑΡΑΓΩΓΗ ΣΤΗΛΩΝ, Η ΚΥΚΛΙΚΗ ΑΠΟΣΤΑΣΗ ΤΟΥ ΠΙΝΑΚΑ (Α) ΓΙΝΕΤΑΙ 128. (Ε) 20
ΠΙΝΑΚΑΣ 6: ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΚΑΙ ΤΙΣ ΠΕΝΤΕ ΔΙΑΔΙΚΑΣΙΕΣ, Η ΚΥΚΛΙΚΗ ΑΠΟΣΤΑΣΗ ΤΟΥ ΠΙΝΑΚΑ (Α) ΓΙΝΕΤΑΙ 4. (Φ) 20
ΠΙΝΑΚΑΣ 7: ΠΑΡΑΔΕΙΓΜΑ ΕΥΡΕΣΗΣ ΕΥΘΕΙΑΣ ΑΠΟΣΤΑΣΗΣ 21
ΠΙΝΑΚΑΣ 8:ΓΚΡΟΥΠ Α, ΓΚΡΟΥΠ Β 22
ΠΙΝΑΚΑΣ 9: ΜΕΤΑΛΛΑΓΗ ΣΤΗΛΩΝ 23
ΠΙΝΑΚΑΣ 10:ΠΙΝΑΚΑΣ ΔΟΚΙΜΗΣ Α 23
ΠΙΝΑΚΑΣ 11: ΠΙΝΑΚΑΣ ΔΟΚΙΜΗΣ Τ ΜΕ ΜΕΤΑΛΛΑΓΗ ΣΤΗΛΩΝ ΠΟΥ ΚΑΝΕΙ ΤΗΝ ΚΥΚΛΙΚΗ ΑΠΟΣΤΑΣΗ ΝΑ ΜΗΝ ΥΠΕΡΒΑΙΝΕΙ ΤΟ $2^{K}=2^{8}$(Κ=8) 24
ΠΙΝΑΚΑΣ 12: ΑΚΟΛΟΥΘΙΑ ΜΕΤΑΤΟΠΙΣΗΣ ΓΙΑ ΤΙΣ ΣΤΗΛΕΣ Α ΚΑΙ Β. Η ΣΤΗΛΗ Β ΜΠΟΡΕΙ ΝΑ ΕΙΣΑΧΘΕΙ ΜΕΤΑ ΤΗΝ ΣΤΗΛΗ ΣΥΝΔΕΣΗΣ $L_2\neg$. ΟΙ ΑΔΙΑΦΟΡΕΣ ΤΙΜΕΣ ΤΙΘΕΝΤΑΙ $X_1=0$, $X_2=1$. 28
ΠΙΝΑΚΑΣ 13:ΚΑΝΟΝΕΣ ΜΕ ΕΠΑΝΑΛΑΜΒΑΝΟΜΕΝΗ ΣΥΜΠΕΡΙΦΟΡΑ 30
ΠΙΝΑΚΑΣ 14: ΣΗΜΑΝΤΙΚΟΤΕΡΟΙ ΚΑΝΟΝΕΣ ΓΙΑ ΤΗΝ ΠΑΡΑΓΩΓΗ ΑΚΟΛΟΥΘΙΩΝ. 33

1. ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο θα πραγματοποιηθεί μια εισαγωγή στο πρόβλημα που πραγματεύεται η παρούσα Πτυχιακή Εργασία. Θα επιχειρηθεί μια εκτενής αναφορά στις παραμέτρους που συνθέτουν το πρόβλημα της δοκιμαστικότητας ηλεκτρονικών κυκλωμάτων και των συνεπειών μιας επιτυχημένης (αλλά και μιας όχι πετυχημένης) δοκιμής ανίχνευσης σφαλμάτων. Θα δοθούν οι πρώτοι ορισμοί που αναφέρονται στο επιστημονικό υπόβαθρο που απαιτείται για την ολοκλήρωση της Πτυχιακής Εργασίας, ενώ θα ακολουθήσει ένας σαφής προσδιορισμός των στόχων της Πτυχιακής Εργασίας.

1.1 Η φάση της δοκιμής (testing) στην ανάπτυξη μιας εφαρμογής

Σε κάθε μοντέλο ανάπτυξης ενός έργου (π.χ. μιας εφαρμογής - προϊόν ή υπηρεσία) απαιτούνται τουλάχιστον πέντε διεργασίες που θα πρέπει να πραγματοποιηθούν σε κάθε φάση ανάπτυξης του έργου: 1) καθορισμός και αξιολόγηση προδιαγραφών, 2) ανάλυση και σχεδίαση, 3) υλοποίηση, 4) έλεγχος, και 5) εγκατάσταση.

Στα πλαίσια του καθορισμού και αξιολόγησης των προδιαγραφών, μελετώνται οι απαιτήσεις του έργου (λειτουργία ενός προϊόντος ή μιας υπηρεσίας, οργάνωση, συμμόρφωση με κανονισμούς, νόμους και περιορισμούς (επιχειρηματικούς, λειτουργικούς, κατασκευαστικούς κ.ο.κ.). Στα πλαίσια της ανάλυσης και σχεδιασμού, αναλύονται τα επί μέρους τμήματα που απαρτίζουν το έργο, καθορίζεται ο σχεδιαστικός στόχος, ορίζεται το αναπτυξιακό περιβάλλον και τα πλεονεκτήματα των τεχνολογιών που θα χρησιμοποιηθούν και σχεδιάζεται σε αφηρημένο επίπεδο. Η τρίτη διεργασία, αυτή της υλοποίησης λαμβάνει υπόψη όλα όσα έχουν καθοριστεί στον καθορισμό των προδιαγραφών, και εφαρμόζοντας όσα έχουν αποφασιστεί κατά την ανάλυση και σχεδιασμό υλοποιείται το προϊόν ή η υπηρεσία. Η τέταρτη διεργασία, είναι αυτή στην οποία επικεντρώνει η παρούσα Πτυχιακή Εργασία και στόχο έχει τον έλεγχο της υλοποίησης και τη λειτουργική συμπεριφορά σύμφωνα με τις προκαθορισμένες προδιαγραφές. Όταν θα έχει γίνει πλήρης έλεγχος, τότε μπορεί να πραγματοποιηθεί η τελευταία (πέμπτη) διεργασία, αυτή της εγκατάστασης, δηλαδή της διάθεσης του προϊόντος ή της υπηρεσίας.

Η ενέργεια του ελέγχου, δυνητικά περιλαμβάνει πολλά σκέλη, τα οποία έχουν άμεση σχέση με τη φύση του παραδοτέου του έργου (π.χ. του προϊόντος ή της υπηρεσίας). Έτσι στα πλαίσια του ελέγχου, συμπεριλαμβάνεται η διαδικασία απασφαλμάτωσης του λογισμικού (debugging) κατά τη λειτουργία, η εφαρμογή αρχικών λειτουργικών δοκιμών κατά την εκκίνηση ενός συστήματος, ο αυτό-έλεγχος κατάστασης, η δοκιμή αντοχής υλικών, ο έλεγχος για τυχόν κενά ασφαλείας σε ένα πρόγραμμα κ.ο.κ. Ιδιαίτερης κρισιμότητας όμως έχει επιπλέον και ο έλεγχος ορθής λειτουργίας ενός συστήματος μετά την διάθεσή του στην αγορά, δηλαδή όταν αυτό έχει αρχικά ελεγχθεί και επιβεβαιωθεί η ορθή του λειτουργία, αλλά απαιτείται η συστηματική διενέργεια ελέγχου ορθής λειτουργίας προκειμένου να

διευκολυνθεί η συντήρησή του ή ακόμα και να ελαχιστοποιηθούν οι πιθανότητες ατυχημάτων.

Ο πιο γνωστός, στο ευρύ κοινό, έλεγχος ορθής λειτουργίας πριν την εκκίνηση ενός συστήματος είναι η δοκιμή της κατάστασης και λειτουργίας ενός υπολογιστή από το BIOS (boot check). Αυτού του είδους ο έλεγχος ονομάζεται off-line, δηλαδή εκτός της κανονικής κατάστασης λειτουργίας. Οι κρίσιμες εφαρμογές όμως απαιτούν πιο συχνό έλεγχο, δηλαδή κατά την κανονική κατάσταση λειτουργία, δηλαδή on-line. Οι δοκιμές που μπορούν να πραγματοποιηθούν κατά την κανονική κατάσταση λειτουργίας, μπορούν να διακριθούν σε αυτές οι οποίες πραγματοποιούνται ταυτόχρονα με την κανονική κατάσταση λειτουργίας (υποστηρίζεται δηλαδή μια επιπλέον ειδική λειτουργία) και αυτές που διακόπτουν το σύστημα, εκτελούν διαγνωστικές δοκιμές και επαναφέρουν το σύστημα στην προ της διακοπής κατάστασής του.

1.2 Κρίσιμες εφαρμογές και η ανάγκη για υψηλό επίπεδο αξιοπιστίας

Το αντικείμενο του Τμήματος Πληροφορικής με Εφαρμογές στη Βιοϊατρική, περιλαμβάνει την ανάπτυξη λογισμικού για εφαρμογές της Βιοϊατρικής, της Βιοπληροφορικής κ.ο.κ., αλλά και την προσαρμογή των υπολογιστικών συστημάτων αυτών των εφαρμογών προκειμένου να ικανοποιούνται οι προδιαγεγραμμένες απαιτήσεις. Ειδικά για το χώρο της Ιατρικής και της Βιολογίας, και των εφαρμογών Πληροφορικής τους, οι κύριες απαιτήσεις είναι η υψηλή απόδοση και η υψηλή αξιοπιστία.

Η αξιοπιστία ενός συστήματος επιτυγχάνεται μέσω της ποιότητας του υλικού που χρησιμοποιείται προκειμένου να μην είναι ιδιαίτερα δεκτικό σε σφάλματα όπως π.χ. η ακτινοβολία. Επιπλέον μέτρα που λαμβάνονται, στην πλειονότητα των υπαρχόντων συσκευών, κυρίως λόγω κόστους των αξιόπιστων υλικών και τεχνολογιών, είναι η χρήση πολλαπλών ομοίων μονάδων προκειμένου να επιτευχθεί αξιοπιστία λόγω πλεονασμού, ή η χρήση ειδικού υλικού το οποίο εφαρμόζει πρότυπα δοκιμής. Η αξιοπιστία σε αυτή την περίπτωση επιτυγχάνεται μέσω του συνδυασμού υλικού και λογισμικού, το οποίο αναπτύσσεται προκειμένου να δημιουργηθούν τα πρότυπα δοκιμής που θα εφαρμοστούν αλλά και για να ελεγχθούν τα αποτελέσματα της απόκρισης.

Όλες οι εφαρμογές οι οποίες μπορούν να θέσουν, σε κάποιο βαθμό, την ακεραιότητα ενός ατόμου, ονομάζονται εφαρμογές Κρίσιμης Ασφαλείας (Safety Critical). Αυτές οι εφαρμογές θα αποτελέσουν το κύριο πεδίο εφαρμογής της παρούσας Πτυχιακής Εργασίας. Η πλειοψηφία των εφαρμογών κρίσιμης ασφαλείας συναντάται στον ιατρικό χώρο, αλλά και στην αεροδιαστημική, τις τηλεπικοινωνίες κ.α.

Η απόδοση ενός υπολογιστικού συστήματος, που υποστηρίζει μια εφαρμογή Κρίσιμης Ασφαλείας, εξαρτάται σημαντικά από το χρόνο που απαιτείται από το λογισμικό να πραγματοποιήσει τον έλεγχο λειτουργίας του συστήματος (δηλαδή, ένα ποσοστό της υπολογιστικής ισχύος δαπανάται για διαφορετικό σκοπό από αυτό της εφαρμογής). Συνεπώς εκτός από την ανάγκη να επιτευχθεί υψηλή αξιοπιστία, δευτερεύουσα αλλά επίσης σημαντική απαίτηση είναι η δυνατότητα του συστήματος να υλοποιεί τη διαδικασία ελέγχου λειτουργίας

το συντομότερο δυνατόν. Αυτό μπορεί να επιτευχθεί είτε με τη δημιουργία ενός παράλληλου συστήματος το οποίο διαρκώς θα ελέγχει το σύστημα υποστήριξης της εφαρμογής κρίσιμης ασφαλείας (πολύ μεγάλο κόστος και υψηλή πολυπλοκότητα σχεδίασης), είτε θα σχεδιαστεί μια αποδοτική λύση για τη δημιουργία των προτύπων δοκιμής. Η πρώτη προσέγγιση βασίζεται κυρίως στην χρήση υλικού ειδικού σκοπού, ενώ η δεύτερη λύση βασίζεται κυρίως στη χρήση μνήμης ή υλικού ειδικού σκοπού (πολύ μικρής επιφάνειας) και στο συνδυασμό ειδικού λογισμικού.

1.3 Γεννήτριες προτύπων δοκιμής

Η αξιοπιστία μπορεί να επιτευχθεί, όπως προαναφέρθηκε, με τη χρήση ειδικού υλικού το οποίο θα παράγει σε σύντομο χρόνο τα πρότυπα δοκιμής. Το υλικό ειδικού σκοπού (special purpose hardware) που είναι επιφορτισμένο με την διεργασία αυτό-ελέγχου του συνολικού συστήματος ονομάζεται Γεννήτρια Προτύπων Δοκιμής (TPG – Test Pattern Generator). Η απαίτηση για υψηλή απόδοση στην παραγωγή των προτύπων δοκιμής, έχει ως αποτέλεσμα την ένταση της έρευνας προς αυτό το πεδίο, με κύριο στόχο την εύρεση αποδοτικών αλγορίθμων που να παράγουν τα απαιτούμενα πρότυπα δοκιμής με τη μικρότερη δυνατή καθυστέρηση αλλά και να πραγματοποιείται η σύγκριση σωστής απόκρισης σχεδόν αμέσως.

Η παρούσα Πτυχιακή Εργασία, έχει ως στόχο τη μελέτη μερικών προσεγγίσεων για τη δημιουργία γεννητριών προτύπων δοκιμής, και τη σύγκρισή τους με μια νέα προσέγγιση η οποία προτείνεται. Η σύγκριση θα πραγματοποιηθεί σε επίπεδο απόδοσης, αξιοπιστίας αλλά και κόστους της λύσης.

1.4 Κυψελιδικά Αυτόματα

Ένα κυψελιδικό αυτόματο (CA-Cellular Automaton) είναι μια συλλογή από "χρωματισμένα" κελιά σε ένα πλέγμα διευκρινισμένης μορφής, που εξελίσσεται σε διάφορα χρονικά βήματα σύμφωνα με ένα σύνολο κανόνων βασισμένων στις καταστάσεις των γειτονικών κελιών. Οι κανόνες εφαρμόζονται συνεχώς για τόσα χρονικά βήματα όσα επιθυμείται. Ο von Neumann ήταν ένας από τους πρώτους ανθρώπους που εξέτασε ένα τέτοιο πρότυπο. Τα CA μελετήθηκαν στις αρχές της δεκαετίας του 1960 ως πιθανό πρότυπο για τα βιολογικά συστήματα (Wolfram 2002). Περιεκτικές μελέτες των CA έχουν διεξαχθεί από το S. Wolfram που αρχίζουν στη δεκαετία του '80, και η βασική έρευνα του Wolfram στον τομέα κατέληξε στη δημοσίευση του βιβλίου του "Ένα νέο είδος επιστήμης" (A NEW KIND OF SCIENCE) (Wolfram 2002) στο οποίο παρουσιάζεται μια εκτενής συλλογή αποτελεσμάτων σχετικά με τα CA, μεταξύ των οποίων υπάρχουν διάφορες απίστευτες νέες ανακαλύψεις.

Η υλοποίηση CA βασίζεται σε απλούς λογικούς κανόνες, επιτρέποντας την υλοποίηση τους με πολύ μικρή σχεδιαστική πολυπλοκότητα. Χαρακτηρίζονται από την παραλληλία, κυρίως λόγω της ελάχιστης τοπικότητας (spatiality) των αλγορίθμων. Ένας αλγόριθμος ο οποίος θα βασίζεται σε κυψελιδικά αυτόματα, θα παρουσιάζει πολύ μικρή καθυστέρηση στην παραγωγή αριθμητικών ακολουθιών. Συνεπώς, τα κυψελιδικά αυτόματα δυνητικά αποτελούν ένα πολύ καλό υποψήφιο (αλγόριθμο) για τη δημιουργία μιας γεννήτριας προτύπων δοκιμής.

1.5 Στόχος της Πτυχιακής Εργασίας

Η πτυχιακή εργασία αφορά στην ανάπτυξη λογισμικού (software) το οποίο θα παράγει ακολουθίες αριθμών κάνοντας χρήση των ιδιοτήτων των CA. Οι αυτόματα παραγόμενες ακολουθίες, θα χρησιμοποιηθούν ως ακολουθίες διανυσμάτων δοκιμής για ψηφιακά ηλεκτρονικά κυκλώματα. Οι Αυτόματες Γεννήτριες Προτύπων Δοκιμής αποτελούν, όπως αναφέρθηκε, βασικό δομικό κομμάτι ενός ψηφιακού συστήματος και η σωστή ανάπτυξή τους (είτε σε υλικό είτε σε λογισμικό) θεωρείται κρίσιμη για την επιτυχή και έγκαιρη διάγνωση τυχόν δυσλειτουργιών του συστήματος.

2. ΔΟΚΙΜΗ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

2.1 Αυτόματη Παραγωγή Προτύπων Δοκιμής (ATPG - Automatic-Test-Pattern-Generation)

Η Αυτόματη Παραγωγή Προτύπων Δοκιμής, είναι μια διαδικασία κατά την οποία με αυτοματοποιημένο τρόπο παράγονται διανύσματα δοκιμής από ένα πρόγραμμα (ή ηλεκτρονικό κύκλωμα), ως είσοδοι σε μια μονάδα υπό έλεγχο, προκειμένου να ανιχνευθούν λάθη λειτουργίας της. Ο κύκλος λειτουργίας ενός ATPG μπορεί γενικά να χωριστεί σε 2 φάσεις :

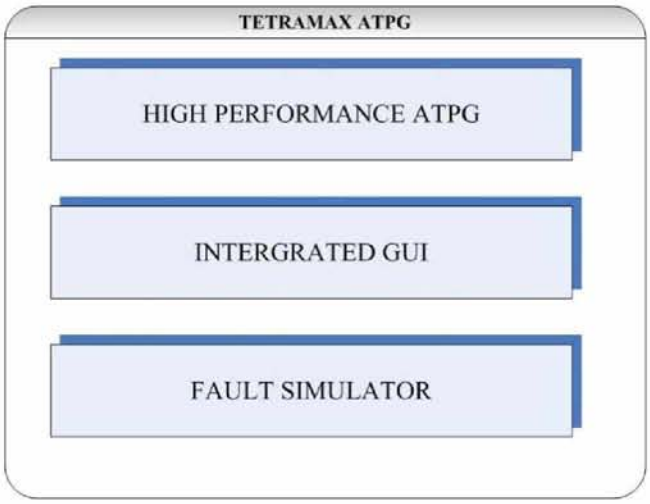
- Την δημιουργία της δοκιμής, και
- Την εφαρμογή της δοκιμής

Κατά την διάρκεια της δημιουργίας της δοκιμής, παράγονται τα κατάλληλα διανύσματα σε επίπεδο λογισμικού ή υλικού (επίπεδο πύλης ή transistor) με τέτοιον τρόπο έτσι ώστε οι αποκρίσεις μιας ελαττωματικής συσκευής για ένα σύνολο τέτοιων διανυσμάτων θα διαφέρουν από τις αναμενόμενες αποκρίσεις. Η δημιουργία των διανυσμάτων δοκιμής είναι μια ρητά ορισμένη από μαθηματικές συναρτήσεις διαδικασία, και η οποία μπορεί να ολοκληρωθεί με τρεις τρόπους :

1. Με μη-αυτόματες μεθόδους
2. με αλγοριθμικές μεθόδους, και
3. με ψευδοτυχαίες μεθόδους

Το λογισμικό που απαιτείται για πολύπλοκες ATPG εφαρμογές είναι αρκετά ακριβό, αλλά το θετικό είναι ότι η διαδικασία πρέπει να εφαρμοστεί μια μόνο φορά (off-line) στο τέλος της διαδικασίας σχεδιασμού του κυκλώματος.

Στο Σχήμα 1 παρουσιάζεται η αρχιτεκτονική του TetraMAX® ATPG της εταιρίας Synopsis [3]. Το συγκεκριμένο ATPG εγγυάται υψηλής ποιότητας test δοκιμής. Όπως βλέπουμε ενσωματώνει ένα υψηλής απόδοσης ATPG το οποίο παράγει ένα πολύ μεγάλο αριθμό διανυσμάτων δοκιμής σε πολύ μικρό χρόνο, ένα ενσωματωμένο γραφικό περιβάλλον για τον χρήστη, καθώς και έναν προσομοιωτή σφαλμάτων προκειμένου να προσομοιωθούν τα λάθη λειτουργίας και να ανιχνευθούν.



Σχήμα 1:Αρχιτεκτονική του TetraMax ATPG –Πηγή Synopsis

2.2 Βασικοί ορισμοί

Προκειμένου να είναι ομαλή η ροή του κειμένου και της παρουσίασης της Πτυχιακής Εργασίας θα πρέπει να οριστούν κάποιες έννοιες:

Ως **ελάττωμα** ενός ηλεκτρονικού συστήματος ορίζουμε την άνευ πρόθεσης διαφοροποίηση του υπάρχοντος ολοκληρωμένου συστήματος από το υποθετικά διαθέσιμο. Μερικές τυπικές μορφές ελαττωμάτων στα VLSI κυκλώματα, είναι:

1. Ελαττώματα διεργασιών – έλλειψη επαφών, παρασιτικά τρανζίστορ κ.ο.κ.
2. Ελαττώματα υλικών – ρωγμές υποστρώματος, νόθευση υλικού κ.ο.κ.
3. Ελαττώματα γήρανσης – ηλεκτρομετανάστευση κ.ο.κ.
4. Ελαττώματα ενθυλάκωσης – οξειδωμένοι ακροδέκτες, ανοιχτοκυκλώματα κ.ο.κ.

Ως **λάθος** ορίζεται κάθε λανθασμένη έξοδο που παράγεται από ένα ελαττωματικό κύκλωμα. Συνεπώς αν το «ελάττωμα» είναι το αίτιο, το «λάθος» είναι το αιτιατό.

Ως **σφάλμα** ορίζεται η λανθασμένη λογική συμπεριφορά του κυκλώματος εξαιτίας της ύπαρξης κάποιου ελαττώματος. Μπορεί να ειπωθεί ότι το «σφάλμα» είναι η αφηρημένη αναπαράσταση του «ελαττώματος» στο λογικό επίπεδο.

Παρακάτω θα αναφερθούν διάφορα μοντέλα σφάλματος τα οποία αποτελούν και τα πλέον χρησιμοποιούμενα, σε όλα σχεδόν τα επίπεδα σχεδιασμού, όσο αφορά το σχεδιασμό για ελεγξιμότητα.

Σφάλματα Συμπεριφοράς: Αυτά τα σφάλματα, όπως έχει αναφερθεί και νωρίτερα, δεν είναι σαφώς σχετιζόμενα με το ελάττωμα που τα προκαλεί. Περισσότερο δε, επειδή έχουν άμεση σχέση με ποιο υψηλού επιπέδου αναπαραστάσεις που κατανοεί το σύστημα, π.χ. κώδικας εφαρμογής, τα σφάλματα δεν μπορούν να χαρακτηριστούν υποχρεωτικά ηλεκτρικά. Στην κατηγορία αυτή ανήκουν τα σφάλματα διακλάδωσης και τα σφάλματα εντολών.

Σφάλματα Διακλάδωσης: Αυτά τα σφάλματα αναφέρονται σε εσφαλμένη αλλαγή της ροής εκτέλεσης του κώδικα.

Σφάλματα Εντολών: Περιλαμβάνουν σφάλματα που συμβαίνουν κατά την ανεπιτυχή εκτέλεση κάποιας εντολής, είτε προκαλώντας την έξοδο λάθος αποτελέσματος, είτε εκτελώντας μια άλλη εντολή χωρίς πρόθεση.

Σφάλματα Πλεονάζοντα: Πρόκειται για σφάλματα τα οποία δεν αλλάζουν την τιμή της εξόδου και κατά συνέπεια δεν είναι ανιχνεύσιμα. Η αποσφαλμάτωση κυκλωμάτων που περιέχουν τέτοια σφάλματα γίνεται συνήθως για λόγους βελτιστοποίησης.

Σφάλματα Λογικής: Πρόκειται για σφάλματα τα οποία μεταβάλουν την κατάσταση μιας γραμμής του κυκλώματος (εσωτερική, έξοδος κ.ο.κ.) σε σταθερή τιμή (0, 1, X ή Z). Συνήθως αυτά τα σφάλματα αναφέρονται ως σφάλματα κολλημένο-σε λόγω του παρόμοιου χαρακτήρα τους.

Σφάλματα Κολλημένο-σε: Αυτά τα σφάλματα αναγκάζουν τη γραμμή στην οποία εφαρμόζονται να διατηρεί εσαεί την τιμή της σταθερή είτε ίση με 1, οπότε και αποκαλείται κολλημένο-σε-1 (κ-σ-1), είτε ίση με 0, οπότε αποκαλείται κολλημένο-σε-0 (κ-σ-0). Ως γραμμή χαρακτηρίζουμε οποιαδήποτε είσοδο ή έξοδο λογικής πύλης ή στοιχείου μνήμης (flip-flop). Οι πιο συνηθισμένες μορφές του είναι το μοντέλο μοναδικού σφάλματος κολλημένο-σε, δύο σφάλματα στη γραμμή κ.ο.κ.

Σφάλματα Διαύλου (Bus): Αυτά τα σφάλματα αναφέρονται στην αλλαγή των περιεχομένων των γραμμών ενός διαύλου δεδομένων όπου μπορούν να είναι είτε κ-σ-0, είτε κ-σ-1 ή ελεύθερες από σφάλματα. Συνεπώς ένας δίαυλος N γραμμών, έχει 3^N-1 πιθανούς συνδυασμούς σφαλμάτων.

Σφάλματα Αρχικοποίησης: Η συντριπτική πλειοψηφία των ολοκληρωμένων συστημάτων περιλαμβάνουν ακολουθιακά κυκλώματα τα οποία απαιτείται κατά την έναρξη της λειτουργίας του συστήματος να λάβουν τις αρχικές τους τιμές προκειμένου να σηματοδοτηθεί η αρχή λειτουργίας σε όλες τις μονάδες μνήμης του συστήματος. Πολλά σφάλματα παρατηρούνται σε αυτή τη διαδικασία λόγω ελαττωμάτων ή σφαλμάτων κολλημένο-σε είτε στη γραμμή διάδοσης του ρολογιού (έλλειψη συγχρονισμού), είτε στην είσοδο του στοιχείου μνήμης. Η είσοδος του στοιχείου μνήμης συνήθως οδηγείται από την έξοδο ενός συνδυαστικού κυκλώματος το οποίο είναι εκτεθειμένο σε κάθε είδος σφάλματος που αναφέρεται σε αυτή την υποενότητα.

Σφάλματα Μνήμης: Σε αυτή την κατηγορία σφαλμάτων περιλαμβάνονται σφάλματα που επηρεάζουν την τιμή είτε ενός κυττάρου της μνήμης, είτε μιας συγκεκριμένης περιοχής της μνήμης με διακριτή συμπεριφορά, είτε ζεύγους κυττάρων, καθώς επίσης αναφέρονται και σε σφάλματα κολλημένο-σε στις γραμμές του αποκωδικοποιητή της διεύθυνσης.

Σφάλματα Καθυστέρησης: Πρόκειται για λάθη τα οποία προκαλούν καθυστέρηση στη μετάδοση της τιμής μιας γραμμής, είτε στην είσοδο μιας λογικής πύλης, είτε στην είσοδο ενός στοιχείου μνήμης κ.ο.κ. με αποτέλεσμα η συνδυασμένη καθυστέρηση του κυκλώματος να υπερβαίνει την περίοδο του ρολογιού.

Σφάλματα Ακίδας: Όταν ένα σύστημα αποτελείται από υπομονάδες, τότε οι επαφές διασύνδεσής τους ονομάζονται ακίδες. Τα σφάλματα (συνήθως σφάλματα κολλημένο-σε) τα οποία συμβαίνουν σε αυτά τα σημεία του συστήματος ονομάζονται σφάλματα ακίδας.

Σφάλματα Υπερδιέγερσης: Ένα σφάλμα τέτοιου είδους ευθύνεται για την αλλαγή στην τιμή πολλών γραμμών του συστήματος. Συνήθως αυτά τα σφάλματα εξαλείφονται δύσκολα λόγω κυρίως της πολυπλοκότητάς τους.

Σφάλματα Γεφύρωσης: Πρόκειται για την αναπαράσταση ενός βραχυκυκλώματος σε επίπεδο σφάλματος, το οποίο συμβαίνει σε μια ομάδα σημάτων.

Σφάλματα Μόνιμα: Τα σφάλματα των οποίων η συμπεριφορά δεν αλλάζει με την πάροδο του χρόνου ονομάζονται μόνιμα.

Σφάλματα Σποραδικά: Τα σφάλματα τα οποία εμφανίζονται και εξαφανίζονται με συγκεκριμένη συχνότητα, ή εξαρτώνται από την πάροδο του χρόνου ονομάζονται σποραδικά.

Σφάλματα Πρόσκαιρα: Αυτά τα σφάλματα είναι κυρίως τυχαία και οφείλονται στην πλειοψηφία τους στις περιβαλλοντολογικές συνθήκες.

Σφάλματα Προσωρινά: Πρόκειται για κατηγορία σφαλμάτων που περιλαμβάνει τα πρόσκαιρα και τα σποραδικά.

Σφάλματα Διαταραχής Απλού-Γεγονότος: Πρόκειται για σφάλματα τα οποία προκαλούνται κυρίως στο διάστημα από βαριά ιόντα (πρόσκαιρα σφάλματα) ή επιδράσεις ιονισμένης ακτινοβολίας (μόνιμα σφάλματα).

2.3 Σκοπός των ATPG

Κατά την δημιουργία μιας δοκιμής, ο στόχος πρέπει να είναι η απόδοση, σε όρους μεγέθους μνήμης και απαιτήσεων χρόνου. Για αυτόν τον λόγο η διαδικασία πρέπει να παράγει τον ελάχιστο ή κοντά στον ελάχιστο αριθμό διανυσμάτων που απαιτούνται για την ανίχνευση όλων των σημαντικών λαθών μιας συσκευής, και στο συντομότερο δυνατό χρόνο. Οι κύριοι στόχοι για την δημιουργία μιας δοκιμής είναι:

- Ο χρόνος που απαιτείται για την κατασκευή του ελάχιστου συνόλου διανυσμάτων
- Οι απαιτήσεις ATPG, δηλαδή το υλικό και το λογισμικό που χρειάζεται για να υποκινήσει κατάλληλα τις συσκευές για τη δοκιμή
- Το μέγεθος της ίδιας της διαδικασίας της δοκιμής
- Τον χρόνο που απαιτείται για την «φόρτωση» των διανυσμάτων δοκιμής, και
- Τον εξωτερικό εξοπλισμό (αν αυτός υπάρχει)

Κατά την εφαρμογή μιας δοκιμής ένα ATPG χρησιμοποιεί αλγοριθμικές μεθόδους για την παραγωγή των προτύπων δοκιμής. Κάθε μια από αυτές τις αλγοριθμικές μεθόδους απαιτούν αυτό που είναι γνωστό σαν «ευαισθητοποίηση μονοπατιού» (path sensitization).Το path sensitization αναφέρεται στην εύρεση ενός μονοπατιού στο κύκλωμα, το οποίο θα επιτρέψει σε ένα σφάλμα να εμφανιστεί (ανιχνευθεί).

Η διαδικασία της παραγωγής των προτύπων δοκιμής αποτελείται από τα εξής βήματα :

1. την επιλογή ελαττωμάτων, δηλαδή την επιλογή ενός ελαττώματος που πρέπει να ανιχνευθεί,
2. την εύρεση ενός προτύπου εισαγωγής που μετά την εφαρμογή του σε μια συσκευή κάνει την έξοδό της να δείξει ένα σφάλμα εάν υπάρξει μέσα στο κύκλωμα,
3. την διάδοση του σφάλματος, εάν αυτό εμφανιστεί, σε μια έξοδο χρησιμοποιώντας την μικρότερη δυνατή διαδρομή,
4. την επικύρωση του σφάλματος, δηλαδή την ταυτοποίηση της εξόδου του προηγούμενου βήματος συγκρίνοντας την με εισόδους που έχουν νέες τιμές.

Εάν παρατηρηθεί κάποια ασυνέπεια στα προηγούμενα βήματα, τότε η διαδικασία επαναλαμβάνεται χρησιμοποιώντας ένα διαφορετικό μονοπάτι. Αυτός ο κύκλος γίνεται έως ότου βρεθούν τα σωστά πρότυπα εισαγωγής που θα βρουν το κατάλληλο μονοπάτι για την ανάδειξη του σφάλματος.

2.4 Ενσωμάτωση Αυτό-Δοκιμής (BIST – Built-In Self Test)

Η ενσωμάτωση αυτό-δοκιμής (BIST), αποτελεί μια τεχνική σχεδίασης πρόσθετου υλικού και λογισμικού σε ολοκληρωμένα κυκλώματα και συστήματα αντίστοιχα. Ένα BIST επιτρέπει στα κυκλώματα να πραγματοποιήσουν μια δοκιμή του «εαυτού» τους, δηλαδή μια δοκιμή της ορθής λειτουργίας τους. Με αυτόν τον τρόπο επιδιώκεται να μειωθεί η εξάρτηση από έναν εξωτερικό Αυτοματοποιημένο Εξοπλισμό Δοκιμής (ATE - External Automated Test Equipment). Ένα BIST είναι μια τεχνική η οποία κάνει την ηλεκτρονική δοκιμή ευκολότερη, πιο γρήγορη, πιο αποδοτική και λιγότερο δαπανηρή για μικρής κλίμακας δοκιμές. Η έννοια του BIST είναι εφαρμόσιμη σε σχεδόν όλα τα είδη κυκλωμάτων, οπότε η εφαρμογή του μπορεί να ποικίλει. Η πολύ μεγάλη ανάπτυξη των BIST οφείλεται στην αυξανόμενη πολυπλοκότητα των ολοκληρωμένων κυκλωμάτων.

Τα πλεονεκτήματα της εφαρμογής ενός BIST είναι αρκετά όπως :

- Χαμηλό κόστος δοκιμής, αφού δεν χρειάζεται επιπλέον εξοπλισμός.
- Καλύτερη κάλυψη λαθών αφού ειδικές δομές δοκιμής μπορούν να ενσωματωθούν στα ολοκληρωμένα συστήματα.
- Χαμηλοί χρόνοι δοκιμής καθώς το BIST μπορεί να σχεδιαστεί να πραγματοποιήσει παράλληλα δοκιμές σε διαφορετικές δομές.

Παράλληλα όμως υπάρχουν και μειονεκτήματα όπως:

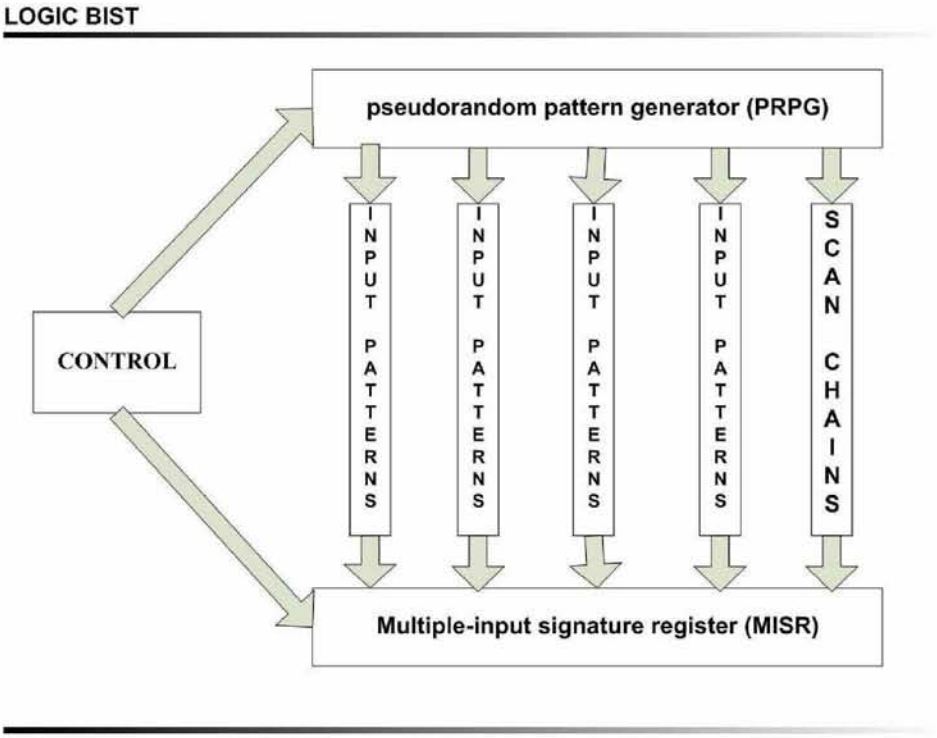
- Αύξηση της επιφάνειας πυριτίου (κόστος).
- Μείωση χρόνου εκκίνησης της συσκευής.
- Απαιτούνται επιπλέον ακροδέκτες (pins) στο κύκλωμα καθώς το κύκλωμα του BIST πρέπει να επικοινωνήσει με το υπόλοιπο υλικό.

Αυτά που πρέπει να αναλογιστούμε πριν την υιοθέτηση μιας λύσης βασισμένης σε BIST είναι τα εξής:

- Τα σφάλματα τα οποία πρέπει να καλυφθούν και πως πρέπει αυτά να ελεγχθούν.
- Πόση περιοχή του ολοκληρωμένου chip απαιτείται να καταληφθεί.
- Οι πιθανές αλλαγές στην εξωτερική τροφοδοσία μετά την εφαρμογή του BIST.
- Ο χρόνος δοκιμής και η αποτελεσματικότητα του.
- Η ευελιξία και η δυνατότητα αλλαγής του BIST, αν θα μπορεί δηλαδή να επαναπρογραμματιστεί ή επαναχρησιμοποιηθεί.

Στο Σχήμα 2 παρουσιάζεται η αρχιτεκτονική ενός LOGIC BIST της εταιρίας Mentor Graphics. Η συγκεκριμένη αρχιτεκτονική εφαρμόζεται σε κάθε διαθέσιμο πυρήνα υλικού και περιλαμβάνει: μια Ψευδοτυχαία Γεννήτρια Προτύπων (PRPG - Pseudorandom Pattern Generator) για να παράγει τα πρότυπα εισόδου για την ανίχνευση εσωτερικά στην συσκευή για λάθη, έναν Πολαπλών Εισόδων Καταχωρητή Υπογραφών (MIST - Multiple Input Signature Register) για την λήψη της απόκρισης της συσκευής σε αυτά τα πρότυπα εισόδου

και έναν ελεγκτή, ο οποίος ελέγχει εάν η είσοδος παράγει την αναμενόμενη έξοδο, δηλαδή εάν τα πρότυπα εισόδου ταιριάζουν με την έξοδο του MIST. Μια λανθασμένη έξοδος του MIST σημαίνει και ελάττωμα στην συσκευή.



Σχήμα 2: Αρχιτεκτονική του LOGIC BIST

Τα BIST αποτελούν μια αξιόπιστη λύση λόγω του αυξανόμενου κόστους της «εξωτερικής» δοκιμής αλλά και της αυξανόμενης πολυπλοκότητας των κυκλωμάτων.

2.5 Διαθέσιμες λύσεις στην διεθνή αγορά

Στη συνέχεια θα παρουσιαστούν διάφορες λύσεις αυτόματης παραγωγής .

2.5.1. SOFTWARE SOLUTIONS

Το TetraMAX® ATPG παράγει αυτόματα υψηλής ποιότητας πρότυπα δοκιμής. Είναι η μόνη λύση ATPG για ένα ευρύ φάσμα μεθοδολογιών δοκιμής με την κατοχυρωμένη με δίπλωμα ευρεσιτεχνίας συμπίεση Synopsys DFTMAX™. Η υψηλή του αποδοτικότητα σε συνδυασμό με την ευκολία στην χρήση του επιτρέπει στους σχεδιαστές ηλεκτρονικών κυκλωμάτων να δημιουργήσουν σε λίγο χρόνο, αποδοτικά και σταθερά δοκιμές ακόμα και για τα πιο πολύπλοκα κυκλώματα.

Ελεγκτής κανόνα σχεδιασμού ATPG

Ο ελεγκτής κανόνα σχεδιασμού (DRC-design rule checker) προσδιορίζει τα ζητήματα test σε επίπεδο chip. Ο DRC ελέγχει για τα εξής προβλήματα:

- Flip-flops που παραβιάζουν τους κανόνες σχεδιασμού αλυσίδων ανίχνευσης.
- Ασύγχρονη λογική που μπορεί να αυξήσει τον χρόνο εκτέλεσης ή να μειώσει την κάλυψη ελαττωμάτων.
- Πρωτόκολλα δοκιμής που μπορούν να προκαλέσουν λανθασμένη συμπεριφορά στον ελεγκτή.

Συμπίεση προτύπων δοκιμής

Τα ATPGs χρησιμοποιούν προηγμένες τεχνικές συμπίεσης για την ελαχιστοποίηση του αριθμού των προτύπων δοκιμής κατά τη διάρκεια της διαδικασίας τους. Με αυτές τις τεχνικές, τα ATPGs μειώνουν τον αριθμό κύκλων δοκιμής που απαιτούνται για να εξετάσουν κάθε συσκευή, με συνέπεια χαμηλότερο κόστος για τον εξοπλισμό των ελεγκτών.

Δοκιμή βάθους-Submicron

Πολλές κατασκευαστικές ατέλειες δεν αναγνωρίζονται χωρίς πρόσθετη δοκιμή που εξετάζει συγκεκριμένες ατέλειες νανομέτρων. Με την επιλογή του DSM Test (Deep Submicron Testing), οι σχεδιαστές και οι μηχανικοί δοκιμής μπορούν εύκολα να παράγουν καθυστέρηση μετάβασης, καθυστέρηση πορειών, το γεφύρωμα ή δυναμικά πρότυπα δοκιμής γεφυρώματος.

Κάποια προηγμένα χαρακτηριστικά γνωρίσματα είναι :

- Πλήρης υποστήριξη για χρονομετρητές που βρίσκονται πάνω στο chip.
- Εύχρηστη ροή με γραφική υποστήριξη για ευκολότερη ανάλυση.
- Βελτιστοποιημένοι αλγόριθμοι για κάθε συγκεκριμένο test καθυστέρησης.
- Συγχώνευση προτύπων για την μεγιστοποίηση της αποδοτικότητας της καθυστέρησης δοκιμών.
- Έτοιμα πρότυπα προς δοκιμή με πλήρη συγχρονισμό.

DSM Test για testing σφάλματος μικρής καθυστέρησης

Το DSM Test δίνει την δυνατότητα στο ATPG να στοχεύει μικρές ατέλειες καθυστέρησης μέσα στα ολοκληρωμένα κυκλώματα που θα μπορούσαν να οδηγήσουν σε σφάλματα όταν λειτουργούν οι συσκευές με την πλήρη ταχύτητα τους. Με την χρήση του συγκεκριμένου test η ανίχνευση σφαλμάτων είναι αποδοτικότερη σε σχέση με την χρήση τυποποιημένων προτύπων καθυστέρησης και παράλληλα μειώνει και το κόστος παραγωγής του test.

Με το συγκεκριμένο test δεν υπάρχει καμία περιττή απώλεια παραγωγής επειδή δεν υπάρχει καμία ανάγκη να εξεταστούν τα μέρη με ποιο γρήγορες συχνότητες. Μαζί με την επιλογή DSM Test, τα ATPG μικρής ατέλειας καθυστέρησης παρέχουν τα ακόλουθα οφέλη:

- Εξαιρετικά υψηλή ποιότητα δοκιμής
- Ακριβείς πληροφορίες συγχρονισμού

- Ροή ενός-περάσματος :
    1) Βραδύτερα-βασισμένο ATPG για μικρές ατέλειες καθυστέρησης
    2) ATPG με σταθερή καθυστέρηση μετάβασης για σφάλματα μεγάλης καθυστέρησης
- Εκθέσεις και ιστογράμματα, που περιλαμβάνουν:

    1)μετρική αποτελεσματικότητα καθυστέρησης
    2)Μετρικό στατιστικό επίπεδο καθυστέρησης (SDQL)

- Δεν απαιτείται κανένας νέος σχεδιασμός

Δοκιμή IDDQ

Η δοκιμή IDDQ είναι μια μέθοδος για την ενίσχυση της ποιότητας των δοκιμών ενός ολοκληρωμένου κυκλώματος με τη μέτρηση της παροχής ρεύματος ενός κυκλώματος CMOS (Complementary metal – oxide – semiconductor). Τα ελεύθερα από ατέλειες κυκλώματα CMOS απαιτούν πολύ χαμηλά επίπεδα ρεύματος κατά τη διάρκεια της ήρεμης κατάστασης. Η δοκιμή IDDQ στοχεύει τις φυσικές ατέλειες που δημιουργούν μια πορεία διεξαγωγής από την παροχή του ηλεκτρικού ρεύματος στην γείωση με αποτέλεσμα να οδηγούν σε κατανάλωση υπερβολικού ρεύματος.

Διαγνωστικά παραγωγής

Εκτός από τον προσδιορισμό των ελαττωματικών μερών από την κατασκευή, ένα ATPG μπορεί επίσης να απομονώσει τη θέση των ατελειών στις συσκευές που αποτυγχάνουν στα πρότυπα δοκιμής του ATPG. Η αυτόματη και ακριβής απομόνωση ατελειών είναι ένα σημαντικό βήμα για να εντοπιστούν τα κρίσιμα ζητήματα παραγωγής και κατά τη διάρκεια της παραγωγής καθώς επίσης και κατά την διάρκεια της κατασκευής. Τα διαγνωστικά ATPG διαβάζουν τα πρότυπα δοκιμής και τα στοιχεία αποτυχίας των δοκιμών, δηλαδή τις διαφορές μεταξύ των μετρημένων απαντήσεων και των αναμενόμενων απαντήσεων στα συγκεκριμένα πρότυπα δοκιμής. Υποδεικνύουν επίσης τις θέσεις των ελαττωμάτων που πιθανότατα εξηγούν την ελαττωματική συμπεριφορά των συσκευών που παρατηρήθηκαν στον ελεγκτή. Τα διαγνωστικά ATPG χρησιμοποιούν έναν υψηλής απόδοσης προσομοιωτή ελαττωμάτων για γρήγορα και αξιόπιστα αποτελέσματα σε ένα κατασκευαστικό περιβάλλον.

3. ΚΥΨΕΛΙΔΙΚΑ ΑΥΤΟΜΑΤΑ

Η έρευνα για τα Κυψελιδικά Αυτόματα (CA - Cellular Automata) άρχισε από τους McCulloch, Pitts, von Neumann και Stanislaw Ulam στη δεκαετία του '40.Ο John von Neumann συμμετείχε αρχικά σε αυτήν την έρευνα κατά τη διάρκεια της προσπάθειάς του να διαμορφώσει αναπαραγόμενα ρομπότ, κάτι που αποδείχθηκε όμως αδύνατο. Εντούτοις, μετά από μια λεπτομερή συζήτηση με τον Ulam εμφανίστηκε η έννοια των CA. Στην πραγματικότητα τα CA είναι «το απλούστερο μοντέλο μιας διανεμημένης στο χώρο διαδικασίας που μπορεί να χρησιμοποιηθεί για να μιμηθεί διάφορες πραγματικές διαδικασίες».

Ένα CA είναι μια ομάδα κελιών που εξελίσσεται μόνο από την κοντινότερη αλληλεπίδραση γειτόνων. Σε μια διάσταση, τα κελιά είναι μια γραμμή σημείων. Κάθε σημείο έχει μια αξία, αντιπροσωπεύοντας ένα χρώμα. Η εξέλιξη κάθε σημείου καθορίζεται από την αξία του και από την αξία των γειτονικών σημείων. Με τον καθορισμό μερικών απλών κανόνων για την εξέλιξη και την απεικόνιση της εξέλιξης της γραμμής των κελιών, είναι δυνατό να ληφθούν πολύ σύνθετα και όμορφα πρότυπα.

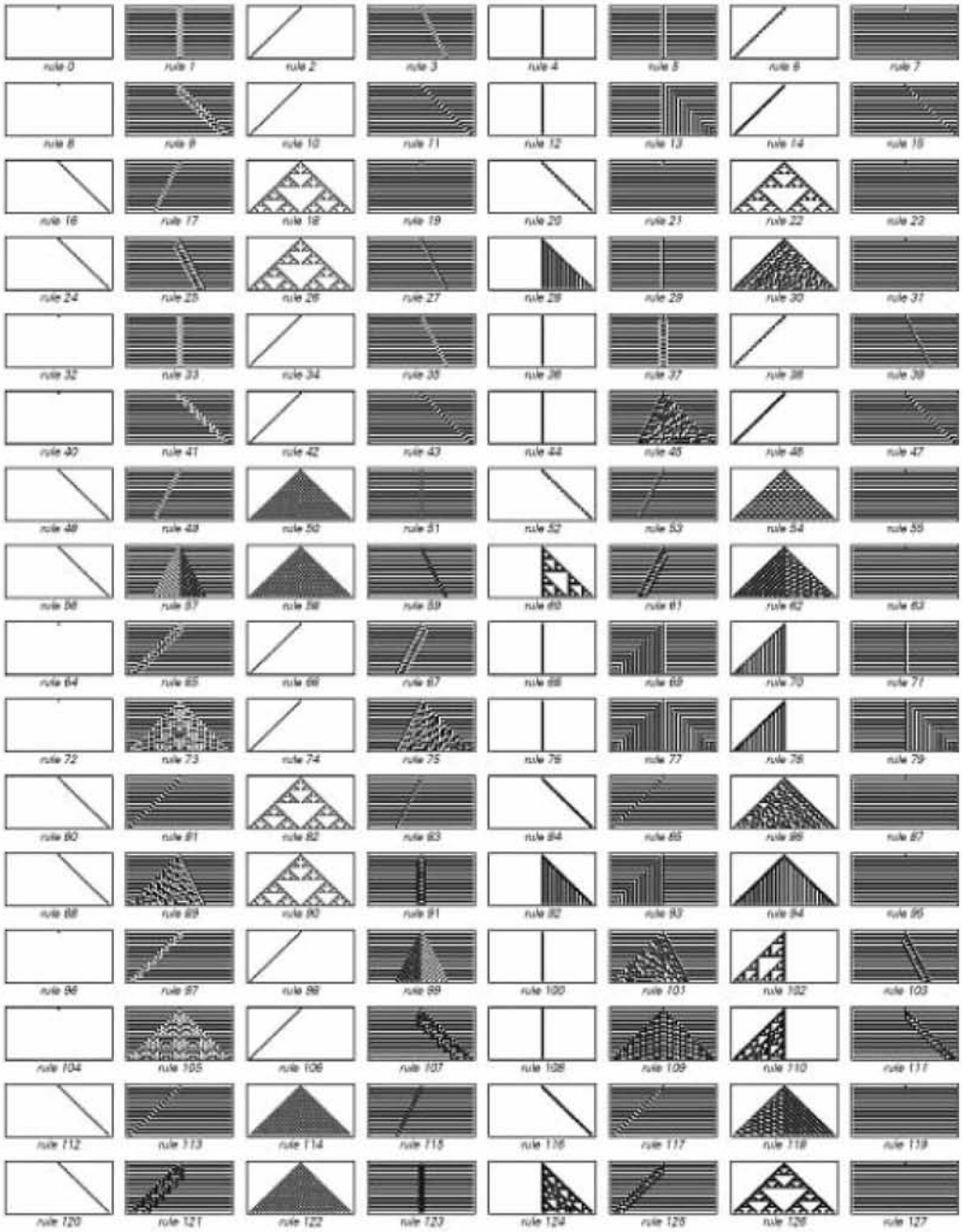
Τα CA έρχονται σε ποικίλες μορφές. Μια από τις πιο θεμελιώδεις ιδιότητες ενός CA είναι ο τύπος πλέγματος στον οποίο υπολογίζεται. Το απλούστερο τέτοιο πλέγμα είναι μια μονοδιάστατη γραμμή. Σε δύο διαστάσεις, τα τετραγωνικά, τριγωνικά, και εξαγωνικά πλέγματα μπορούν να εξεταστούν. Τα CA μπορούν επίσης να κατασκευαστούν στα καρτεσιανά πλέγματα σε αυθαίρετους αριθμούς διαστάσεων, με το δ-διαστατικό δικτυωτό πλέγμα Z^d ακέραιων αριθμών που είναι η πιο κοινή επιλογή και άλλα.

Ο αριθμός χρωμάτων (ή ευδιάκριτων καταστάσεων) K που ένα CA μπορεί να υποθέσει πρέπει επίσης να διευκρινιστεί. Αυτός ο αριθμός είναι συνήθως ένας ακέραιος αριθμός, με k=2 (δυαδικό) όντας η απλούστερη επιλογή. Για ένα δυαδικό αυτόματο, το χρώμα 0 είναι συνήθως το λευκό και το χρώμα 1 είναι συνήθως το μαύρο.

Τα CA είναι - εξ ορισμού - δυναμικά συστήματα που είναι ιδιαίτερα στο διάστημα και το χρόνο, λειτουργούν σε ένα ομοιόμορφο, κανονικό δικτυωτό πλέγμα - και χαρακτηρίζονται από " τοπικές" αλληλεπιδράσεις.

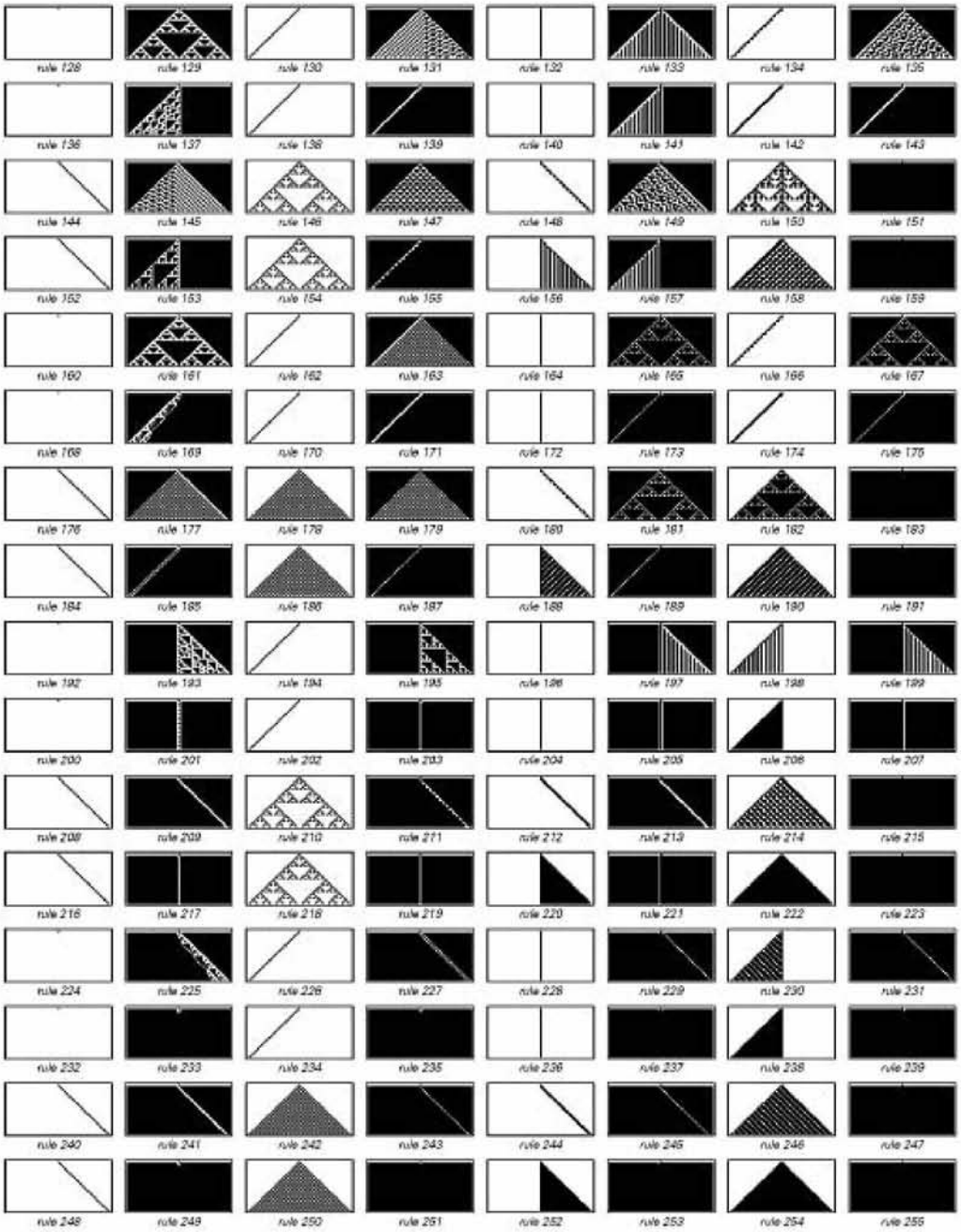
Αν και η πολυπλοκότητα της παραγωγής των CA μέσω των απλών λογικών κανόνων είναι χαμηλή, δεν υπάρχει μέχρι τώρα καμία πραγματική εφαρμογή τους στην τεχνολογία. Ο κύριος λόγος είναι η έλλειψη ενός ιδιαίτερου μαθηματικού προτύπου που μπορεί να γίνει κατανοητό και να χρησιμοποιηθεί από τους επαγγελματίες και τους μηχανικούς.
Υπάρχουν 256 δυνατά σύνολα επιλογών που μπορούν να δημιουργηθούν, για τη χρήση 2 χρωμάτων. Αυτές οι επιλογές μετρούνται από το 0 έως το 255. Στις πιο απλές περιπτώσεις

όλα τα κελιά καταλήγουν στο να έχουν το ίδιο χρώμα μετά από ένα βήμα. Για παράδειγμα,



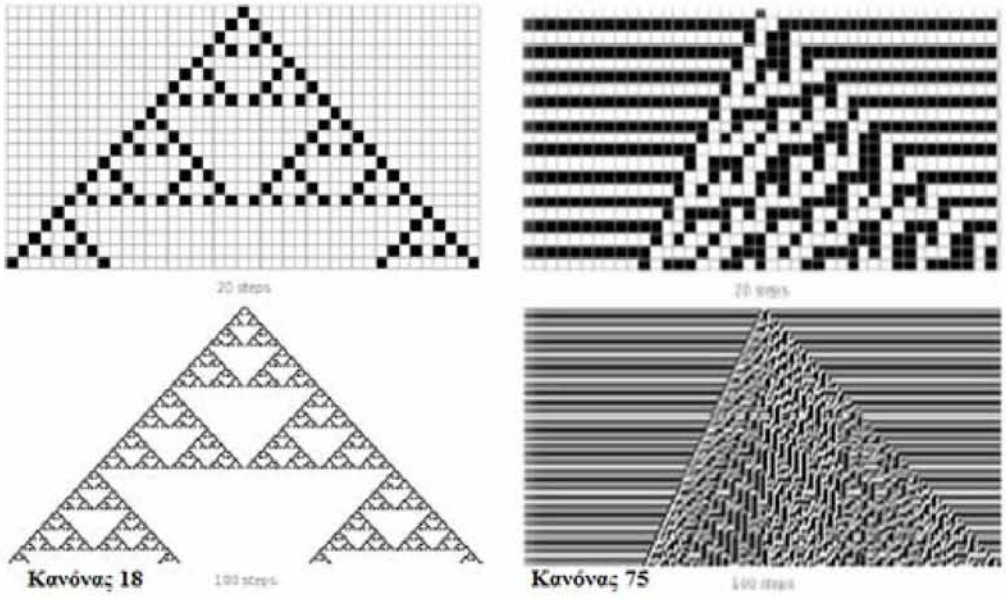
Σχήμα 3:Λίστα κανόνων CA από 0 έως 127

Στους κανόνες 0 και 96 όλα τα κελιά γίνονται άσπρα, ενώ στον κανόνα 255 όλα είναι γίνονται μαύρα. Υπάρχουν όμως και κανόνες όπως ο 7 και ο 127 στους οποίους όλα τα κελιά εναλλάσσονται μεταξύ άσπρου και μαύρου σε διαδοχικά βήματα.



Σχήμα 4:Λίστα κανόνων CA από 128 έως 255

Κάποιοι από τους κανόνες παρουσιάζουν χαοτική συμπεριφορά (μπορεί να χρησιμοποιηθεί για την παραγωγή τυχαίων αριθμών **(Σχήμα 5 δεξιά)**)και κάποιοι άλλοι παρουσιάζουν επαναλαμβανόμενη συμπεριφορά **(Σχήμα 5 αριστερά)**.



Σχήμα 5: Αριστερά-επαναλαμβανόμενη, Δεξιά-χαοτική

Οι κανόνες που παρουσιάζουν χαοτική συμπεριφορά δεν παρουσιάζουν ιδιαίτερο ενδιαφέρον καθώς η έξοδος είναι ανεξέλεγκτη και δεν μπορεί να παραχθεί κάποια γνωστή ακολουθία.

Περισσότερα για την σημαντικότητα των κυψελιδικών αυτομάτων για την παρούσα πτυχιακή θα αναλυθούν σε επόμενο κεφάλαιο.

4. Παραγωγή Διανυσμάτων Δοκιμής από Μετρητή

4.1 Ποιες είναι οι τεχνικές που απαιτούνται για την ικανοποίηση των στόχων

Η παραγωγή προτύπων δοκιμής σε ένα σε ένα Build in self test απαιτεί όσο το δυνατόν μικρότερη αύξηση του επιπλέον υλικού. Έχουν προταθεί αρκετές μέθοδοι (βασισμένες σε ROM, μετρητές με προσθετική λογική, FPGAs) αλλά η πολυπλοκότητα των κυκλωμάτων δοκιμής είναι ένας περιοριστικός παράγοντας για την εφαρμογή αυτών των μεθόδων.

Σε αυτό το κεφάλαιο θα εξετάσουμε την δυνατότητα ενός μετρητή, ως μηχανισμό παραγωγής σχεδίων δοκιμής, να αναπαράγει ένα σύνολο δοκιμής. Ένας βασικός μετρητής προσφέρει το ελάχιστο κόστος σε υλικό (ένας μετρητής με μήκος w αποτελείται από ακριβώς w flip-flops).

Ας υποθέσουμε ότι **τ** είναι το μέγεθος του δοσμένου συνόλου δοκιμής T και **w** το μήκος κάθε σχεδίου δοκιμής στον T. Προσδιορίζουμε τον T ως έναν πίνακα διάστασης **τ x w**. Σε αυτό το σημείο ορίζουμε τα εξής:

- **η ευθεία απόσταση** του T είναι $\max\{r_i\}$-$\min\{r_i\}$ (η διαφορά της μέγιστης από την ελάχιστη ανά γραμμή) όπου r_i, $1 \leq i \leq \tau$ είναι οι τιμές των δυαδικών σχεδίων των σειρών του T.

- **η wraparound απόσταση**(απόσταση περιτύλιξης) του T η οποία ορίζεται ως η τιμή 2^w - $\max\{r_i-r_j\}$, όπου r_i-r_j είναι διαδοχικά διανύσματα δοκιμής σε ταξινομημένη σειρά $r_i>r_j$.

- **η κυκλική απόσταση** του T είναι το ελάχιστο της ευθείας και της wraparound απόστασης.

Εάν η κυκλική απόσταση ενός τ x w πίνακα δοκιμής T είναι d, τότε ένας μετρητής μήκους w αρχίζοντας με το αντίστοιχο αρχικό διάνυσμα, θα παράγει όλα τα τ διανύσματα σε d κύκλους ρολογιού. Μπορούμε να μετατρέψουμε τον πίνακα T σε έναν άλλον πίνακα T' ο οποίος έχει μικρότερη κυκλική απόσταση χρησιμοποιώντας τις ακόλουθες διαδικασίες:

Διαδικασίες Προ-επεξεργασίας :

- αποβολή σταθερών στηλών
- συγχώνευση ίδιων στηλών
- συμπληρωματική συγχώνευση στηλών και ,

Βασικές Διαδικασίες :

- μεταλλαγή στηλών
- συμπληρωματική παραγωγή στηλών.

Από όλες αυτές τις διαδικασίες η μεταλλαγή στηλών είναι η πιο σημαντική καθώς όλες οι άλλες διαδικασίες στηρίζονται σε κατάλληλες διατάξεις στηλών.

Πριν όμως αναλύσουμε αυτές τις διαδικασίες πρέπει να ορίσουμε τα εξής:

- μια στήλη είναι **σταθερή** εάν αποτελείται από 0 ή 1
- δύο στήλες είναι **συμπληρωματικές** εάν σε κάθε μια θέση που η μία έχει 0 (ή 1) η άλλη έχει 1 (ή 0 αντίστοιχα).
- δύο στήλες είναι **ίδιες** εάν έχουν 1 και 0 στις ίδιες θέσεις.

Όλες οι διαδικασίες προ-επεξεργασίας στοχεύουν στο να μειώσουν τον αριθμό των κελιών (επιβάρυνση υλικού) του μετρητή και ως συνέπεια παράγουν έναν πίνακα T' με όχι μεγαλύτερη κυκλική απόσταση από τον T.Η διαδικασία της αποβολής σταθερών στηλών αφαιρεί όλες τις σταθερές στήλες του T. Μια σταθερή στήλη δεν χρειάζεται μετρητή γιατί το αντίστοιχο διάνυσμα δοκιμής μπορεί να συνδεθεί στην τροφοδοσία ή στο καλώδιο γείωσης. Οι άλλες δύο διαδικασίες προ-επεξεργασίας έχουν να κάνουν με το γεγονός ότι κάθε κελί μετρητή (flip flop) μπορεί να εφοδιάσει την τιμή του σε περισσότερα από ένα διανύσματα δοκιμής και επίσης σε κοινές εφαρμογές κάθε κελί μετρητή παρέχει δύο συμπληρωματικές εξόδους κατάστασης. Πράγματι εάν ο αριθμός των εισαγωγών που μπορεί να οδηγηθεί από κάθε κατάσταση εξόδου είναι f, ένα κελί μετρητή αρκείται να παράγει μέχρι f ίδιες στήλες του T καθώς επίσης και f περισσότερες ίδιες στήλες που συμπληρώνουν τον πρώτο. Η συγχώνευση ίδιων στηλών βρίσκει ομάδες μέχρι f ίδιες στήλες, όπου f είναι το μέγιστο επιτρεπτό fan-out(Βαθμός οδήγησης μιας λογικής πύλης, δηλαδή πόσες άλλες μπορεί να τροφοδοτήσει με ηλεκτρικό ρεύμα-σήμα), και αντικαθιστά κάθε ομάδα από μια αντιπροσωπευτική στήλη, ενώ η συμπληρωματική συγχώνευση στηλών βρίσκει ζευγάρια από συμπληρωματικές αντιπροσωπευτικές στήλες και τις ορίζει στο ίδιο flip-flop.

Η μέγιστη επιτρεπτή τιμή του f περιορίζεται από την επίδραση που έχουν τα μεγάλα fan-out στην καθυστέρηση του κυκλώματος. Λαμβάνοντας το κύκλωμα προς τεστ καθώς και την περίοδο του ρολογιού του μηχανισμού του τεστ , κάποιος μπορεί εύκολα να παράγει ένα άνω όριο της τιμής f, των μέγιστο αριθμό των ίδιων στηλών που μπορούν να συγχωνευθούν σε μια στήλη, έτσι ώστε οι καθυστερήσεις στα κρίσιμα μονοπάτια κατά την διάρκεια του τεστ να μην υπερβαίνει την περίοδο ρολογιού.

Ο πίνακας 1(β) παρουσιάζει την επίδραση της συγχώνευσης ίδιων στηλών στο παράδειγμα του πίνακα 1(α). Για f=5, οι 5 ίδιες στήλες του T συγχωνεύονται σε μια κοινή στήλη σώζοντας με αυτόν τον τρόπο 4 κελιά μετρητή. Ο πίνακας δοκιμής που προκύπτει έχει 4 στήλες και η κυκλική απόσταση είναι 9. Αυτό πραγματοποιείται αρχίζοντας με το διάνυσμα v_2 και τερματίζοντας με το διάνυσμα v_3. Ο πίνακας 1(γ) δείχνει την επίδραση της συγχώνευσης συμπληρωματικών στηλών του T στον πίνακα δοκιμής του πίνακα 1(α). Ένα flip-flop μπορεί να χρησιμοποιηθεί και για τα δύο c_3 και c_4 για συνολική απαίτηση 7 flip-flops. Με αυτόν τον τρόπο η κυκλική απόσταση γίνεται 64 αρχίζοντας με το διάνυσμα v_2 και τερματίζοντας με το v_3.

	C1	C2	C3	C4	C5	C6	C7	C8
V1	1	1	0	1	1	1	0	1
V2	0	1	0	1	1	1	0	1
V3	1	1	0	1	1	1	1	1
V4	1	0	1	0	0	0	0	0

Πίνακας 1:Ένας πίνακας δοκιμής με κυκλική απόσταση 128 (α)

	C1	C2	C3	C7
V1	1	1	0	0
V2	0	1	0	0
V3	1	1	0	1
V4	1	0	1	0

Πίνακας 2:Με την συγχώνευση ίδιων στηλών, η κυκλική απόσταση του πίνακα (α) γίνεται 9 (β)

	C1	C2	C4	C5	C6	C7	C8
V1	1	1	1	1	1	0	1
V2	0	1	1	1	1	0	1
V3	1	1	1	1	1	1	1
V4	1	0	0	0	0	0	0

Πίνακας 3:Με την συμπληρωματική συγχώνευση στηλών η κυκλική απόσταση του πίνακα (α) γίνεται 64. (γ)

	C3	C2	C4	C5	C6	C8	C7	C1
V1	0	1	1	1	1	1	0	1
V2	0	1	1	1	1	1	0	0
V3	0	1	1	1	1	1	1	1
V4	1	0	0	0	0	0	0	1

Πίνακας 4: Με την μεταλλαγή στηλών, η κυκλική απόσταση του πίνακα (α) γίνεται 5. (δ)

	C1	C2	C3	C4	C5	C6	C'7	C8
V1	1	1	0	1	1	1	1	1
V2	0	1	0	1	1	1	1	1
V3	1	1	0	1	1	1	0	1
V4	1	0	1	0	0	0	1	0

Πίνακας 5: Με την συμπληρωματική παραγωγή στηλών, η κυκλική απόσταση του πίνακα (α) γίνεται 128. (ε)

	C3	C'1	C7
V1	0	0	0
V2	0	1	0
V3	0	0	1
V4	1	0	0

Πίνακας 6: Χρησιμοποιώντας και τις πέντε διαδικασίες, η κυκλική απόσταση του πίνακα (α) γίνεται 4. (φ)

Σε αυτό το σημείο θα δώσουμε ένα παράδειγμα για το πώς βρίσκουμε την ευθεία απόσταση, την wraparound απόσταση και την κυκλική απόσταση σε έναν πίνακα δοκιμής. Έστω ότι έχουμε τον παρακάτω πίνακα :

	128	64	32	16	8	4	2	1	
	C1	C2	C3	C4	C5	C6	C7	C8	
V1	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**=221**
V2	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**= 93**
V3	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**=223**
V4	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**=160**

Πίνακας 7: Παράδειγμα εύρεσης ευθείας απόστασης

Αφού έχουμε βρει τα αθροίσματα των δυαδικών τιμών ανά γραμμή η ευθεία απόσταση ορίζεται ως η διαφορά της μέγιστης από την ελάχιστη τιμή, δηλαδή σε αυτήν την περίπτωση 223-93=**130** . Η wraparound απόσταση ορίζεται ως $2^w - \max(r_i-r_j)$ με $r_i>r_j$, δηλαδή σε αυτήν την περίπτωση έχουμε w=8, όσα είναι και τα κελιά του μετρητή άρα $2^w=2^8=256$.Τώρα οι διαφορές είναι οι εξής : $r_0-r_1=221-93=128$, $r_1-r_2=93-223$ μη επιτρεπτό, $r_2-r_3=223-160=63$.Τώρα η wraparound απόσταση ορίζεται ως η διαφορά του 256 με το μέγιστο των διαφορών δηλαδή 256-128=**128**. Οπότε η κυκλική απόσταση είναι ο μικρότερος αριθμός ανάμεσα στους δυο που βρήκαμε και στην προκειμένη περίπτωση **128**.

Η πρώτη βασική **διαδικασία**, η μεταλλαγή στηλών, στοχεύει στην εύρεση μιας κατάλληλης χαρτογράφησης των flip-flops του μετρητή και των εισαγωγών του κυκλώματος υπό δοκιμή. Η μεταλλαγή στηλών δεν προκαλεί καμία σημαντική επιβάρυνση υλικού καθώς το μόνο που χρειάζεται είναι να συνδέσουμε το κατάλληλο flip-flop του μετρητή στην αντίστοιχη – υπό μεταλλαγή – κύρια είσοδο του κυκλώματος.

Η δεύτερη βασική διαδικασία, η συμπληρωματική παραγωγή στηλών, έχει να κάνει με το γεγονός ότι κάθε κελί μετρητή παράγει σε κάθε περίπτωση δύο συμπληρωματικές καταστάσεις και επομένως στον υπολογισμό της κυκλικής απόστασης έχουμε την επιλογή να χρησιμοποιήσουμε μια στήλη στην κανονική της ή στην συμπληρωμένη της μορφή. Το όφελος στην μείωση της κυκλικής απόστασης με αυτήν την διαδικασία παρουσιάζεται στον πίνακα 1(γ) για τον πίνακα δοκιμής 1(α). Η στήλη c_7 έχει συμπληρωθεί και η κυκλική απόσταση έπεσε σε 128, αρχίζοντας με το διάνυσμα v_2 και τερματίζοντας με το διάνυσμα v_1.

Το συνδυαστικό αποτέλεσμα των βασικών διαδικασιών και των διαδικασιών προ-επεξεργασίας παρουσιάζεται στην εικόνα 1(φ). Χρειάζεται ένας μετρητής με μόνο 3 flip-flop. Η Q κατάσταση του πρώτου flip-flop παράγει το c_3 και η Q' κατάσταση παράγει τα c_2, c_4, c_5, c_6 και c_8. Η Q' κατάσταση του δεύτερου flip-flop παράγει το c_1. Η Q κατάσταση του τελευταίου flip-flop παράγει το c_7. Η κυκλική απόσταση γίνεται 4, αρχίζοντας με το διάνυσμα v_1 και τερματίζοντας με το διάνυσμα v_4.

4.2 Πίνακες δοκιμής χωρίς αδιάφορες καταστάσεις

Σε αυτό το κομμάτι θα επικεντρωθούμε στην ειδική περίπτωση όπου ο πίνακας δοκιμής δεν περιέχει αδιάφορες καταστάσεις (no don't care values).

Υποθέτουμε ότι οι διαδικασίες προ-επεξεργασίας έχουν εφαρμοστεί στον πίνακα δοκιμής T. Υποθέτουμε ότι ο T περιέχει μη σταθερές στήλες, αλλά μπορεί να περιέχει ίδιες ή συμπληρωματικές στήλες που μπορούν να συγχωνευθούν από τις διαδικασίες συγχώνευσης ίδιων και συμπληρωματικών στηλών λόγω ενός ορίου στις τιμές του fan-out f.

Παίρνοντας έναν πίνακα T και ένα σετ S από στήλες του T δείχνουμε με rem(T,S) τον υπό-πίνακα του T ο οποίος περιλαμβάνει όλες τις στήλες του T όχι στο S. Παίρνοντας έναν υπό-πίνακα M του T και μια στήλη C του T που δεν ανήκει όμως στο M, δείχνουμε με indr(M,T,x,C) το σύνολο των δεικτών των γραμμών του T που περιέχουν τιμές x στην στήλη C και συμμετέχουν στον M. Υποθέτουμε επίσης την ύπαρξη μιας διαδικασίας ORDER(M,A,B) η οποία, όταν της δοθεί ένας πίνακας δοκιμής M και ένα χώρισμα των διανυσμάτων δοκιμής του M σε δυο γκρουπ A και B, θα βρει μια μεταλλαγή των στηλών του M (εάν μπορεί να υπάρξει μια μεταλλαγή) έτσι ώστε με την μεταλλαγή κάθε διάνυσμα δοκιμής του γκρουπ A θα γίνει ίσο ή μεγαλύτερο από κάθε διάνυσμα δοκιμής του γκρουπ B. Σαν ένα παράδειγμα, ας σκεφτούμε τον πίνακα δοκιμής M στον πίνακα 8, με το γκρουπ A να αποτελείται από τα διανύσματα v_1, v_2, v_3 και το γκρουπ B να αποτελείται από τα διανύσματα v_4, v_5, v_6, v_7. Η διαδικασία ORDER(M,A,B) σε αυτήν την περίπτωση θα κάνει την μεταλλαγή όπως φαίνεται στον πίνακα 9, η οποία εγγυάται ότι κάθε διάνυσμα δοκιμής στο γκρουπ A δεν είναι μικρότερο από οποιοδήποτε διάνυσμα δοκιμής του γκρουπ B.

	C1	C2	C3	C4	C5	C6	C7	C8
V1	1	0	0	0	0	1	1	1
V2	1	1	0	1	1	0	1	1
V3	1	1	0	1	1	0	1	1
V4	0	1	1	0	0	0	0	0
V5	1	1	0	1	1	0	1	1
V6	1	1	0	0	0	0	1	1
V7	0	0	1	0	1	1	0	1

Πίνακας 8:γκρουπ A, γκρουπ B

(α)

	C8	C1	C7	C3	C6	C2	C4	C5
V1	1	1	1	0	1	0	0	0
V2	1	1	1	0	0	1	1	1
V3	1	1	1	0	0	1	1	1
V4	0	0	0	1	0	1	0	0
V5	1	1	1	0	0	1	1	1
V6	1	1	1	0	0	1	0	0
V7	1	0	0	1	1	0	0	1

Πίνακας 9: Μεταλλαγή στηλών

Σε αυτό το σημείο θα δώσουμε ένα παράδειγμα για τον παρακάτω πίνακα δοκιμής(πίνακας 8). Για να κάνουμε την κυκλική απόσταση να είναι μικρότερη ή ίση από 2^k για k=8 μπορούμε να μεταλλάξουμε τις στήλες όπως δείχνεται στον πίνακα 9. Σε αυτό το παράδειγμα, η στήλη $C_c = c_8$, ο υπό-πίνακας M αποτελείται από τις στήλες c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_{12} του T, το γκρουπ A από τις γραμμές 1,2,3 του M και το γκρουπ B από τις γραμμές 4,5,6,7 του M. Ο μετρητής θα καλύψει όλα τα διανύσματα εάν αρχίσει από το διάνυσμα v_2 και τερματίσει στο διάνυσμα v_5.

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}
V_1	1	0	0	0	0	1	1	0	1	1	1	1
V_2	1	1	0	1	1	0	1	0	1	1	1	1
V_3	1	1	0	1	1	0	1	0	1	1	1	1
V_4	0	1	1	0	0	0	0	1	0	0	0	0
V_5	1	1	0	1	1	0	1	1	0	0	0	1
V_6	1	1	0	0	0	0	1	1	0	0	0	1
V_7	0	0	1	0	1	1	0	1	0	0	0	1

Πίνακας 10:Πίνακας δοκιμής A

	C_8	C_9	C_{10}	C_{11}	C_{12}	C_1	C_7	C_3	C_6	C_2	C_4	C_5
V_1	0	1	1	1	1	1	1	0	1	0	0	0
V_2	0	1	1	1	1	1	1	0	0	1	1	1
V_3	0	1	1	1	1	1	1	0	0	1	1	1
V_4	1	0	0	0	0	0	0	1	0	1	0	0
V_5	1	0	0	0	1	1	1	0	0	1	1	1
V_6	1	0	0	0	1	1	1	0	0	1	0	0
V_7	1	0	0	0	1	0	0	1	1	0	0	1

Πίνακας 11: πίνακας δοκιμής Τ με μεταλλαγή στηλών που κάνει την κυκλική απόσταση να μην υπερβαίνει το $2^k=2^8(k=8)$

Ας δούμε λίγο πιο προσεκτικά τι κάνει η διαδικασία ORDER.

Ο αλγόριθμος ORDER βασίζεται σε ένα greedy scheme. Θέλουμε να κάνουμε κάθε γραμμή του γκρουπ Α να είναι όχι μικρότερη από κάθε γραμμή του γκρουπ Β, ή αλλιώς, θέλουμε να κάνουμε την μικρότερη γραμμή του Α να είναι μεγαλύτερη ή ίση από τη μεγαλύτερη γραμμή του Β. Για κάθε στήλη του Μ, σκεφτόμαστε το ''Α'' και ''Β'' κομμάτι της. Παίρνουμε μια στήλη C_x του Μ που περιέχει όλο 1 στο Α-κομμάτι. Βάζοντας την στήλη C_x στην πιο σημαντική θέση, εγγυόμαστε ότι όλες οι γραμμές του Β που περιέχουν 0 στο C_x θα είναι μικρότερες από οποιαδήποτε γραμμή του Α. Παρόλα αυτά, οι γραμμές του Β που περιέχουν 1 στην C_x πρέπει τελικά να γίνουν μικρότερες από οποιαδήποτε γραμμή του Α. Για να το εγγυηθούμε αυτό, εξετάζουμε κατ' επανάληψη την ίδια κατάσταση στον induced υπό-πίνακα που αποτελείται από τις γραμμές του Α συν τις τελευταίες γραμμές του Β καθώς αποκλείουμε την στήλη C_x. Μια παρόμοια στρατηγική ακολουθείται σε περίπτωση που υπάρχει μια στήλη που περιέχει 0 στο Β μέρος της.

4.3 Ποια τα μειονεκτήματα χρήσης του

Η χρήση ενός μετρητή για την παραγωγή διανυσμάτων δοκιμής έχει αρκετά μικρή απαίτηση σε υλικό και ιδιαίτερα μετά την εφαρμογή των διαδικασιών για την απλοποίηση του πίνακα δοκιμής. Παρόλα αυτά παρουσιάζονται και κάποια μειονεκτήματα. Αρχικά, η χρήση ενός μετρητή δεν είναι αποδοτική σε περιπτώσεις που ο πίνακας δοκιμής είναι πολύ μεγάλος. Σε τέτοιες περιπτώσεις άλλες τεχνικές έχουν αποδειχτεί αποδοτικότερες. Τέλος, η συγκεκριμένη τεχνική παραγωγής διανυσμάτων δεν είναι ιδιαίτερα γρήγορη.

5. Κυψελιδικά αυτόματα για την παραγωγή καθορισμένων ακολουθιών δοκιμής

Τα κυψελιδικά αυτόματα(CA) σαν πιθανοί μηχανισμοί για την παραγωγή σχεδίων δοκιμής έχουν μελετηθεί από πολλούς ερευνητές. Σε αυτό το κεφάλαιο επικεντρωνόμαστε στα CA για την on-chip παραγωγή σχεδίων δοκιμής μιας διατεταγμένης ακολουθίας σχεδίων δοκιμής, παραγόμενη από ένα ATPG εργαλείο. Αυτό είναι ένα πολύ δύσκολο εγχείρημα. Άλλοι τύποι ενσωματωμένων μηχανισμών, περιλαμβάνοντας weighted-random LFSR's και δυαδικούς μετρητές, οι οποίοι έχουν προταθεί ως αποδοτικοί μηχανισμοί για μη-διατεταγμένες ακολουθίες σχεδίων, απαιτούν πολύ εντατική χαρτογράφηση υλικού για να εγγυηθούν την παραγωγή της εισαγωγικής ακολουθίας δοκιμής T, και είναι μη πρακτικοί.

Εξετάζουμε ένα δυαδικό μονοδιάστατο CA με την επικοινωνία μεταξύ των κελιών να ορίζεται ως σχέση μιας-γειτονιάς. Εάν η κατάσταση του κελιού i, $1<=i<=n$, όπου n είναι ο συνολικός αριθμός των κελιών, στο χρόνο t είναι x_i^t, τότε η επόμενη κατάσταση x_i^{t+1} υπολογίζεται από την συνάρτηση $x_i^{t+1} = g_i(x_{i-1}^t, x_i^t, x_{i+1}^t)$. Οι τιμές x_0^t και x_1^t είναι σταθερές (θέτονται 1 ή 0) για κάθε t. Παίρνοντας έναν p x w πίνακα δοκιμής T , ο στόχος είναι να εδραιώσουμε έγκυρες συναρτήσεις $g_i(x_{i-1}^t, x_i^t, x_{i+1}^t)$ έτσι ώστε για κάθε στήλη (p x 1 διάνυσμα) C_j του T , υπάρχουν κάποια κελιά i έτσι ώστε $C_j^t = x_i^t$, $1\leq t\leq p$. Αυτά τα κελιά παράγουν στήλες, οι οποίες αναφέρονται ως στήλες σύνδεσης, οι οποίες δεν αντιστοιχούν σε στήλες του πίνακα δοκιμής. Η δομή των στηλών σύνδεσης επιλέγεται έτσι ώστε να σχηματίζονται έγκυρες συναρτήσεις . Οι συναρτήσεις μπορούν να εφαρμοστούν από πολυπλέκτες 8:1. Βέβαια, ο στόχος είναι να εισάγουμε όσο το δυνατόν λιγότερες στήλες σύνδεσης , καθώς κάθε στήλη σύνδεσης χρειάζεται ένα κελί και έναν πολυπλέκτη.

Σε αυτό το κεφάλαιο προτείνετε ένα σχέδιο βασισμένο σε συστηματικές (προγραμματίσιμες) διαδικασίες P_j για τη λήψη έγκυρων συναρτήσεων $g_i()$. Αυτές οι διαδικασίες αναφέρονται ως διαδικασίες σύνδεσης. Παίρνοντας δύο στήλες C_u και C_v του T, η διαδικασία σύνδεσης P_j υπολογίζει ένα σετ από επιπλέον στήλες σύνδεσης για να είναι δυνατή η παραγωγή των C_u και C_v . Αυτός ο αριθμός χρησιμοποιείτε σαν βάρος $w_j(u,v)$ στην ακμή (u,v) του γράφου G.Το πρόβλημα διατυπώνεται χρησιμοποιώντας έναν γράφο G, όπου κάθε κόμβος αντιστοιχεί σε μια στήλη και το βάρος της κάθε ακμής (u,v) είναι $\min_j\{w_j(u,v)\}$, όπου το ελάχιστο λαμβάνεται από όλες τις εξεταζόμενες διαδικασίες P_j. Οι προτεινόμενες διαδικασίες όχι μόνο εγγυώνται έγκυρα σετ από στήλες σύνδεσης μεταξύ οποιονδήποτε δυο στηλών C_u και C_v αλλά επίσης ότι το CA παράγει την συνδεδεμένη ακολουθία από οποιεσδήποτε δύο ήδη παραγμένες ακολουθίες (C_u, C_v) και (C_v, C_w) χωρίς κάποιες πρόσθετες στήλες σύνδεσης. Κατά αυτόν τον τρόπο , ο αλγόριθμος εγγυάται το μικρότερο δυνατό κόστος υλικού στο σετ των εξεταζόμενων διαδικασιών σύνδεσης $\Pi = U_j P_j$. Κατά συνέπεια όσο πιο αποδοτικές είναι οι διαδικασίες σύνδεσης, τόσο μικρότερο θα είναι το κόστος υλικού. Το βασικό σχήμα

Εξετάζουμε έναν πίνακα δοκιμής T **p x w** συμπεριλαμβάνοντας p διατεταγμένα διανύσματα δοκιμής. Έστω ότι X,X_L και X_R να δείχνουν οποιαδήποτε στήλη (1 x p). Καθορίζουμε τα εξής:

Λαμβάνοντας μια ακολουθία από τρεις στήλες (X_L ,X, X_R) , συνδέουμε σε κάθε γραμμή i , 1≤ i≤ p-1 , την τ-template $tau_i=[x_L^i\ x^i\ X^{t+1}\ \ x_R^i\ ]$.(καμία τ-template δεν συνδέετε με την τελευταία γραμμή p). Θα χρησιμοποιήσουμε $H(\tau_i)$ για να συνδέσουμε το επάνω μέρος $=[x_L^i\ x^i\ \ x_R^i\ ]$ του τ_i και το $L(\tau_i)$ για να συνδέσουμε το κατώτερο μέρος $[x^{t+1}]$.

Λαμβάνοντας μια ακολουθία από στήλες (X_L ,X, X_R) λέμε ότι δυο τ-templates τ_i και τ_j ,1<=i , j<=p-1 , είναι συγκρουόμενα εάν και μόνο αν συμβεί

- $H(\tau_i)=H(\tau_j)$ και $L(\tau_i)<>L(\tau_j)$.

Ορισμός 1 : Μια ακολουθία από τρεις στήλες (X_L ,X, X_R) είναι έγκυρο τρίδυμο(τριπλέτα) εάν και μόνο αν δεν υπάρχουν συγκρουόμενα τ-templates.

Ορισμός 2 :Μια ακολουθία από στήλες (X_1,X_2, …X_i) είναι έγκυρη ακολουθία εάν και μόνο αν όλες οι υπό-ακολουθίες ((X_1,X_2, X_3) , (X_2, X_3,X_4), ….. (X_{i-3}, X_{i-2},X_{i-1}), (X_{i-2}, X_{i-1},X_i) είναι έγκυρα τρίδυμα.

Ορισμός 3: Λαμβάνοντας οποιεσδήποτε δυο στήλες A και B , μια διαδικασία σύνδεσης P είναι ένα πρόγραμμα που επιστρέφει μια έγκυρη ακολουθία στηλών (A,L_0,L_1,L_2,…..L_j,B). Οι στήλες L_i , 0 <=i<=j ονομάζονται στήλες σύνδεσης της έγκυρης διαδικασίας.

Θεώρημα 2.1:Το προτεινόμενο πλαίσιο εγγυάται το μικρότερο πιθανό κόστος που απαιτείται από ένα CA μιας γειτονιάς για να παράγει οποιαδήποτε ακολουθία δοκιμής T από ένα δοσμένο σετ από διαδικασίες σύνδεσης.

Από τα παραπάνω γίνεται σαφές ότι το προτεινόμενο σχέδιο ανοίγει μια πολύ ενδιαφέρουσα κατεύθυνση έρευνας. Αυτήν της εύρεσης ενός κατάλληλου εργαλείου Π(T) από διαδικασίες σύνδεσης P_j για έναν δοσμένο πίνακα T έτσι ώστε το κόστος παραγωγής του T χρησιμοποιώντας CA μιας γειτονιάς είναι το μικρότερο δυνατό. Στην συνέχεια, δίνονται δυο τέτοιες διαδικασίες σύνδεσης οι οποίες συνεισφέρουν πολλά στην ελαχιστοποίηση του αριθμού τον στηλών σύνδεσης. Οι προτεινόμενες διαδικασίες είναι γενικές, δηλαδή μπορούν να εφαρμοστούν σε οποιονδήποτε πίνακα δοκιμής T. Επιπλέον, εγγυώνται ότι το CA παράγει την συνδεδεμένη ακολουθία οποιονδήποτε δυο ήδη παραγμένων υπό-ακολουθιών (C_u,C_v) και (C_v,C_w) χωρίς επιπλέον στήλες σύνδεσης. Το προτεινόμενο εργαλείο σύνθεσης του CA χρησιμοποιεί αυτές τις διαδικασίες, αλλά επίσης μπορεί να δεχτεί και άλλες διαδικασίες σύνδεσης χωρίς να γίνουν αλλαγές στο εργαλείο.

5.1 Δυο Διαδικασίες Σύνδεσης

5.1.1. Διαδικασία μετατόπισης ακολουθιών

Ορισμός: Παίρνοντας μια στήλη $X=(x^1, x^2 \dots x^p)^{t\ r}$, η μετατοπισμένη στήλη του X είναι η στήλη $X=(x^2, x^3, \dots x^p,d)^{t\ r}$,όπου d είναι μια αδιάφορη κατάσταση.

Lemma1: Παίρνοντας μια στήλη X, η ακολουθία στηλών (X_L ,X, X^) είναι ένα έγκυρο τρίδυμο(τριπλέτα) οποιασδήποτε στήλης X_L.

Lemma2.2: Έστω ότι X είναι μια στήλη η οποία περιέχει αδιάφορες καταστάσεις μόνο σε κάθε είσοδο με περιεχόμενο i ή υψηλότερο για κάποια δοσμένα i ,$1 \leq i \leq p$. Έπειτα, για κάθε στήλη X_R, η ακολουθία $(X,X^,X_R)$ είναι μια έγκυρη τριπλέτα εάν και μόνο αν δεν υπάρχουν συγκρουόμενα τ-templates που δεν περιέχουν αδιάφορες καταστάσεις στα L μέρη τους.

Lemma2.3:Έστω $(A,L^{AB}_0, L_1^{AB}, L_2^{AB} \dots L_{jAB}^{AB},B)$ είναι μια ακολουθία μετατόπισης από το A στο B και $(B,L^{BC}_0, L_1^{BC}, L_2^{BC} \dots L_{jBC}^{BC},C)$ είναι μια ακολουθία μετατόπισης από το B στο C. Τότε $(A,L^{AB}_0, L_1^{AB}, L_2^{AB} \dots L_{jAB}^{AB},B, L^{BC}_0, L_1^{BC}, L_2^{BC} \dots L_{jBC}^{BC},C)$ είναι μια έγκυρη ακολουθία.

Παίρνοντας δυο οποιεσδήποτε στήλες A και B από έναν πίνακα δοκιμής, ενδιαφερόμαστε στο να βρούμε μια ακολουθία μετατόπισης $(A,L_0, L_1, L_2 \dots L_{jAB},B)$ με το ελάχιστο μήκος. Αυτό σημαίνει ελαχιστοποίηση του αριθμού j_{AB} των στηλών σύνδεσης $L_1, L_2 \dots L_{jAB}$ (το L_0 πρέπει πάντα να είναι ίσο με $\hat{A}$). Αυτός ο ελάχιστος αριθμός (δειγμένος από m_{AB}) μπορεί να βρεθεί από επιτυχημένα shift-ups από $L_0=\hat{A}$ μέχρι να διαμορφωθεί μια έγκυρη τριπλέτα να τερματίζει με την στήλη B. Το βάρος $w_1(A,B)$ που αποκτήθηκε από αυτήν την διαδικασία σύνδεσης (που αναφέρεται σε μας ως P_j) για δυο οποιεσδήποτε στήλες A,B τίθεται $w_1(A,B)=m_{AB}+1$.

Ο έλεγχος για μια έγκυρη τριπλέτα κάθε φορά γίνεται σύμφωνα με το Lemma2.2 αντικαθιστώντας την στήλη X_R στο Lemma με την στήλη B. Η εικόνα 1 δίνει ένα παράδειγμα. Μια διαδικασία που βρίσκει τις στήλες σύνδεσης δίνεται παρακάτω.

```
Procedure MINSL(A,B)

m_AB:=0;

X_L:=A; X:=Â;

While (X_L,X,B) is not valid according to Lemma 2.2

        m_AB:= m_AB+1;

       X_L:=X, X:=X̂_L;

 End While

Return m_AB;

END MINSL
```

A	L_1	L_2
1	0	1
0	1	1
1	1	0
1	0	0
0	0	1
0	1	0
1	0	x_1
0	x_1	x_2

B
0
0
1
0
1
1
0
0

Πίνακας 12: Ακολουθία μετατόπισης για τις στήλες A και B. Η στήλη B μπορεί να εισαχθεί μετά την στήλη σύνδεσης $L_2\neg$. Οι αδιάφορες τιμές τίθενται $x_1=0$, $x_2=1$.

Ας δούμε όμως τι κάνει ο αλγόριθμος MINSL(A,B) καλύτερα:

Ορίζουμε την στήλη A σαν X_L, την μετατοπισμένη στήλη A σαν $\hat{X}$ και το $m_{AB}=0$(αριθμός στηλών σύνδεσης). Τώρα ο αλγόριθμος 'τρέχει' όσο η τριπλέτα X_L, $\hat{X}$, B δεν είναι έγκυρη σύμφωνα με το Lemma2.2, δηλαδή ελέγχουμε τις τριάδες οι οποίες δεν έχουν αδιάφορες τιμές. Βλέπουμε ότι η $1^η$ με την $4^η$ γραμμή είναι ίδιες δηλαδή τα επάνω μέρη τους είναι ίδια. Για να είναι αυτές οι τριπλέτες συγκρουόμενες πρέπει τα κάτω μέρη τους να είναι διαφορετικά, και στην συγκεκριμένη περίπτωση είναι καθώς στην $1^η$ σειρά το κάτω μέρος είναι 1 και στην $4^η$ το κάτω μέρος είναι 0. Οπότε μπαίνουμε μέσα στο WHILE και κάνουμε το $m_{AB}=1$ και όπου X_L βάζουμε το X που ήταν η στήλη $\hat{A}$ και όπου X βάζουμε την μετατοπισμένη στήλη της ήδη μετατοπισμένης στήλης A δηλαδή την στήλη $\hat{\hat{A}}$. Συνεχίζουμε και ελέγχουμε αν υπάρχουν συγκρουόμενες τριπλέτες για την νέα τριάδα στηλών. Βλέπουμε ότι δεν υπάρχουν καθώς στις περιπτώσεις όπου τα επάνω μέρη δύο τριάδων είναι ίδια, το κάτω μέρος της $2^{ης}$ τριάδας είναι μια αδιάφορη κατάσταση, στην οποία μπορούμε να βάλουμε όποια τιμή θέλουμε. Στην προκειμένη περίπτωση για να μην υπάρξουν συγκρουόμενα t-templates βάζουμε όπου x_1 το 0 και όπου x_2 το 1. Κάνουμε το $m_{AB}=1+m_{AB}=1+1=2$, που ισχύει καθώς έχουμε δυο στήλες σύνδεσης και τερματίζουμε το WHILE.

5.1.2. Διαδικασία Target Sequence

Δοσμένης οποιασδήποτε στήλης A ενός πίνακα δοκιμής , ορίζουμε μια στοχευόμενη διαδικασία για την A ως μια διαδικασία (C_1,A,L_1, L_2, . . . L_j,C_2) η οποία αρχίζει με μια σταθερή στήλη C_1 ακολουθούμενη από την A, περιέχει μερικές (πιθανόν καμία) στήλες σύνδεσης, τερματίζει με μια σταθερή στήλη C_2 , και είναι έγκυρη. Οι C_1 και C_2 μπορούν να

είναι ανεξάρτητα οποιεσδήποτε σταθερές στήλες, αλλά στην συνέχεια υποθέτουμε ότι και οι δύο είναι σταθερά 0. Η εισαγωγή των στηλών σύνδεσης 'στοχεύετε' για να επιτρέψει την σταθερή στήλη C_2 να τοποθετηθεί σε κάποια θέση μετά την στήλη Α χωρίς να δημιουργεί κάποια μη έγκυρη τριπλέτα. Οι στήλες σύνδεσης δεν παράγονται απαραίτητα από το σενάριο της μετατόπισης, αλλά μάλλον παράγονται έτσι ώστε να αυξήσουν τον αριθμό των 0 σε κάθε επιτυχημένη στήλη σύνδεσης. Ιδιαίτερα, η παραγωγή κάθε καινούργιας στήλης σύνδεσης γίνεται έτσι ώστε να 'πιέσει' την τελευταία είσοδο 1 να εμφανιστεί στην στήλη όσο πιο ψηλά γίνεται (προς ή μετά την πρώτη είσοδο της στήλης).

6. Υπολογιστικές Ακολουθίες Βασισμένες στα Κυψελιδικά αυτόματα

Στο κεφάλαιο 3 δόθηκαν οι ορισμοί και έγινε μια πρώτη έρευνα για τα κυψελιδικά αυτόματα. Σε αυτό το κεφάλαιο θα παρουσιαστούν πιο συγκεκριμένα συμπεράσματα καθώς και ο κανόνας στον οποίο βασίζεται η εφαρμογή που έχει αναπτυχθεί για την παρούσα πτυχιακή.

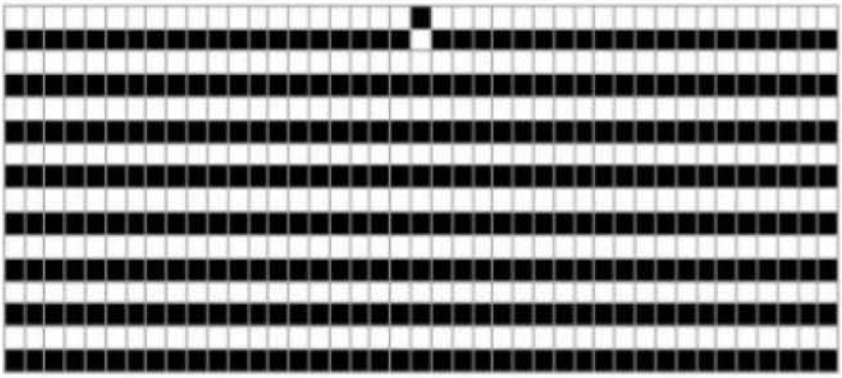
Όπως έχει ήδη αναφερθεί κάποιοι από τους κανόνες των κυψελιδικών αυτομάτων παρουσιάζουν χαοτική συμπεριφορά και κάποιοι επαναλαμβανόμενη συμπεριφορά. Τα κυψελιδικά με χαοτική έξοδο δεν μας ενδιαφέρουν καθώς από αυτήν δεν μπορεί να παραχθεί μια γνωστή ακολουθία.

Στον επόμενο πίνακα παρουσιάζονται οι κανόνες που έχουν επαναλαμβανόμενη συμπεριφορά.

RULE 18	RULE 22	RULE 26	RULE 60	RULE 86
RULE 90	RULE 102	RULE 122	RULE 126	RULE 129
RULE 146	RULE 150	RULE 153	RULE 154	RULE 161
RULE 167	RULE 181	RULE 195	RULE 210	

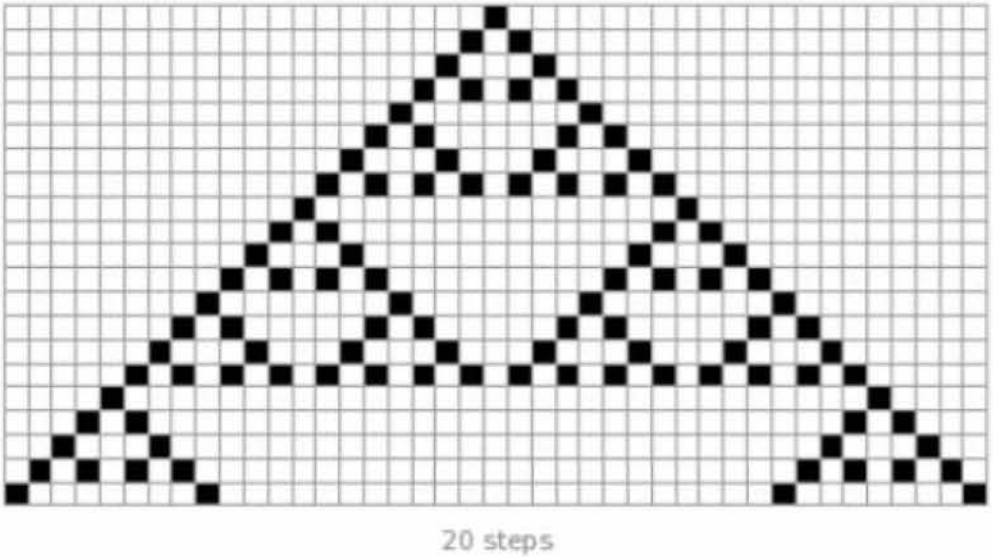
Πίνακας 13:Κανόνες με επαναλαμβανόμενη συμπεριφορά

Από τους κανόνες με επαναλαμβανόμενη συμπεριφορά 2 είδη προσδιορίστηκαν. Αυτά με μονοτονική συμπεριφορά όπως στο **Σχήμα 6**, που παρόλο που ένα πολύ καλό σχέδιο αναπτύσσεται, η έξοδος είναι συνεχώς η ίδια χωρίς καμία αλλαγή στην ακολουθία τον αριθμών.



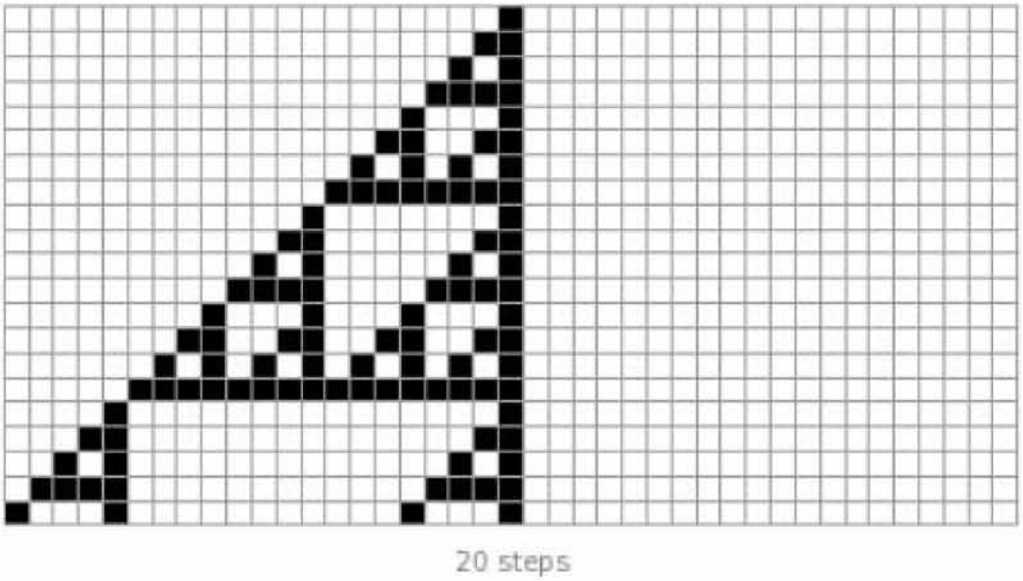
Σχήμα 6:CA με μονοτονική συμπεριφορά

Οι κανόνες που παρουσιάζουν ενδιαφέρον είναι εκείνοι που εξελίσσονται και μεταβάλλουν την έξοδό τους στον χρόνο παράγοντας καλά καθορισμένα πρότυπα όπως στο **Σχήμα 7**.



Σχήμα 7: CA 20 βημάτων με καθορισμένο σχέδιο

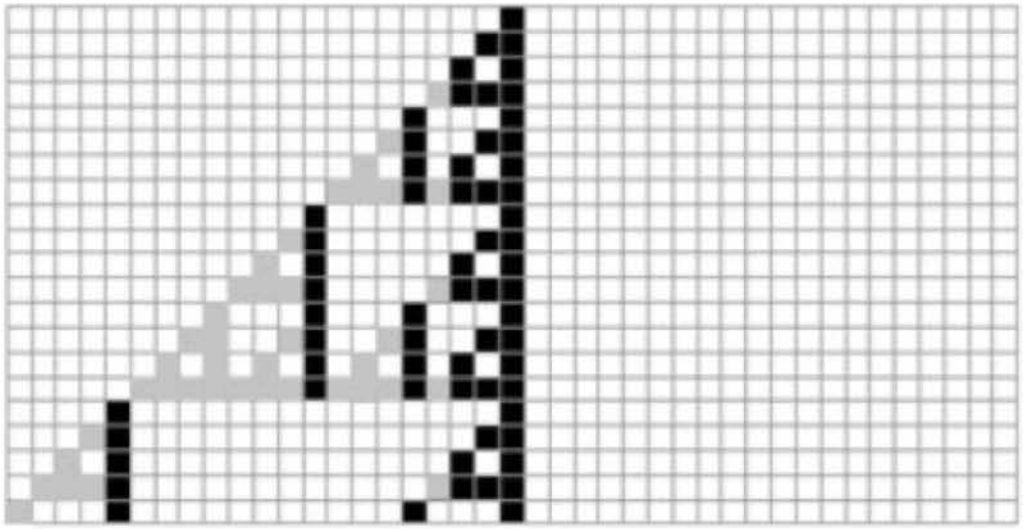
Ο κανόνας 102 **(Σχήμα 8)** παρουσιάζει ενδιαφέρον καθώς είναι επαναλαμβανόμενος και παράλληλα έχει μεταβαλλόμενη έξοδο.



Σχήμα 8:CA 20 βημάτων με επαναλαμβανόμενη συμπεριφορά και μεταβαλλόμενη έξοδος

Η πρώτη στήλη από τα αριστερά αποτελεί ένα bit για επαναλαμβανόμενα πρότυπα. Στην επόμενη στήλη διακρίνεται μια τραβέρσα 2 τιμών για κάθε κύτταρο (0 ή 1) και στη στήλη μετά από αυτήν μια τραβέρσα 2 τιμών για κάθε δύο κύτταρα. Η τέταρτη στήλη είναι περιττό κομμάτι και η χρήση της είναι να παραγάγει το επόμενο σχέδιο. Η πέμπτη στήλη είναι μια τραβέρσα 2 τιμών για κάθε 4 κύτταρα. Οι επόμενες τρεις είναι μια επανάληψη του προηγούμενου προτύπου έτσι δεν λαμβάνονται υπόψη. Αυτά τα πρότυπα παράγονται από την μορφή Boolean : q XOR r ή από την αλγεβρική μορφή : $(q+r)\text{mod}2$.

Από τα παραπάνω μπορεί να παραχθεί η επόμενη εικόνα :



Σχήμα 9: Εξαγωγή της δυαδικής ακολουθίας με την παράλειψη μερικών βημάτων

Το βασικό σχέδιο χρησιμοποιείται αρχικά για να παραγάγει ένα τμήμα. Στο υπόλοιπο της ακολουθίας, το σχέδιο παραλείπεται. Μόνο τα πρόσθετα κομμάτια επιλέγονται, τα οποία παρουσιάζουν μια καλά καθορισμένη ιδιοκτησία. Διαφέρουν μεταξύ τους 2^n bits από το προηγούμενο αποτελεσματικό bit, όπου το n είναι $[0,\infty]$ αντιπροσωπεύει το αποτελεσματικό bit αρχίζοντας από την πρώτη στήλη από τα αριστερά μετά από το επιτρεπόμενο σήμα.

Ο συγκεκριμένος κανόνας είναι και αυτός του οποίου οι ιδιότητα θα χρησιμοποιηθεί για την παραγωγή των διανυσμάτων δοκιμής κατά την εκτέλεση της εφαρμογής. Οι υπό-κανόνες από τους οποίους αποτελείται ο κανόνας 102, είναι οι εξής:

Τρέχον Σχέδιο	111	110	101	100	011	010	001	000
Καινούργια κατάσταση	0	1	1	0	0	1	1	0

Οι κανόνες μπορούν να ερευνηθούν προκειμένου να προσδιοριστούν ποιες ακολουθίες παράγουν (εάν είναι γνωστές). Έχει παρατηρηθεί ότι στο σύνολο μόνο 7 κανόνες είναι χρήσιμοι σε έναν σχεδιαστή κυκλωμάτων(**Πίνακας 2**) που θέλει να αναπτύξει έναν μεγάλο μετρητή (ή ακριβέστερα μια γεννήτρια ακολουθίας).

Κανόνας	Χαρακτηριστικά ακολουθίας		
	Αριθμητική Ακολουθία	Κωδικός	Εφαρμογή
26	Χρονική καθυστέρηση της παραγωγής του 1	-	έλεγχος
60	Δυαδική ακολουθία	Βάση του 2	Επεξεργασία Στοιχείων
102	Δυαδική ακολουθία	Βάση του 2	Επεξεργασία Στοιχείων
122	Toggling bits	βήμα	testing
150	Αύξηση απόστασης Hamming	-	testing
153	Συμπληρωματική δυαδική ακολουθία	Βάση του 2	Επεξεργασία Στοιχείων
195	Συμπληρωματική δυαδική ακολουθία	Βάση του 2	Επεξεργασία Στοιχείων

Πίνακας 14: Σημαντικότεροι κανόνες για την παραγωγή ακολουθιών.

Αν και οι εκμεταλλεύσιμοι κανόνες είναι λίγοι, ο στόχος να εισαχθούν γρήγορες γεννήτριες ακολουθιών βασισμένες σε απλές δομές (χαμηλή-πολυπλοκότητα) ικανοποιείται. Τέλος, αποδεικνύεται σε πολλές περιπτώσεις, ότι ένας μετρητής ο οποίος βασίζεται στα κυψελιδικά αυτόματα παρουσιάζει έως και 50% μεγαλύτερη ταχύτητα από άλλες προτεινόμενες υλοποιήσεις.

7. Προτεινόμενη μεθοδολογία για παραγωγή διανυσμάτων δοκιμής με Radix-2 απαριθμητές βασισμένους σε κυψελιδικά αυτόματα

7.1 Προδιαγραφή απαιτήσεων

Στο σημείο αυτό θα αναλυθούν και θα καθοριστούν οι λειτουργικές και μη λειτουργικές απαιτήσεις του προς ανάπτυξη λογισμικού.

Λειτουργικές απαιτήσεις:

Οι λειτουργικές απαιτήσεις περιγράφουν τις εργασίες(λειτουργίες) που ια πρέπει να εκτελεί το λογισμικό:

1. Παραγωγή διανυσμάτων δοκιμής: Η εφαρμογή θα πρέπει να εμφανίζει τα διανύσματα που παράγονται με την εισαγωγή κάποιον τιμών και κανόνων από τον χρήστη.
2. Δυνατότητα άμεσης παραγωγής νέων διανυσμάτων δοκιμής: Ο χρήστης θα μπορεί να παράγει σε πολύ μικρό χρόνο τα νέα διανύσματα δοκιμής απλά και μόνο αντικαθιστώντας ένα αρχείο τύπου .txt στον φάκελο που περιέχεται η εφαρμογή.

Μη λειτουργικές απαιτήσεις:

Οι μη λειτουργικές απαιτήσεις περιγράφουν χαρακτηριστικά που πρέπει να έχει το λογισμικό, τα οποία δεν αφορούν την εκτέλεση κάποιας λειτουργίας από αυτό.

Απαιτήσεις χρήσης: Η εφαρμογή προορίζεται να χρησιμοποιηθεί σε περιβάλλον Windows και Linux καθώς σε αυτά τα δυο λειτουργικά συστήματα έχει δοκιμαστεί στην πράξη.

Φυσική απαίτηση: Η εφαρμογή προορίζεται για χρήση σε ηλεκτρονικό υπολογιστή.

7.2 ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ ΕΡΓΑΣΙΩΝ

Για να επιτευχθεί η σωστή ανάπτυξη της εφαρμογής αυτής της πτυχιακής εργασίας, θα πρέπει ακολουθηθεί ένα διάγραμμα ροής εργασιών, που μέσω αυτού θα γίνει εφικτή η ανάπτυξή της.

1. Κατανόηση του τομέα testing και του τρόπου λειτουργίας ενός ATPG εργαλείου.
2. Κατανόηση των στόχων ενός ATPG εργαλείου.
3. Έρευνα των διαθέσιμων λύσεων ATPG και διαφόρων τεχνικών στην διεθνή αγορά.
4. Κατανόηση της επιστήμης των κυψελιδικών αυτομάτων.
5. Μελέτη των διαφορετικών προσεγγίσεων για την παραγωγή προτύπων δοκιμής(μετρητής, κυψελιδικά αυτόματα).
6. Εύρεση του απαραίτητου προγράμματος για την ανάπτυξη της εφαρμογής.
7. Υλοποίηση εφαρμογής συνδυάζοντας τα βήματα 4,5,6.

7.3 ΑΝΑΛΥΣΗ ΠΕΡΙΠΤΩΣΕΩΝ ΧΡΗΣΗΣ

Σε αυτό το σημείο θα αναγνωρισθούν οι περιπτώσεις χρήσης της εφαρμογής που έχει αναπτυχθεί. Για την συγκεκριμένη εφαρμογή η περίπτωση χρήσης είναι μια, αυτή της παραγωγής των διανυσμάτων δοκιμής.

Τίτλος περίπτωσης χρήσης

Παραγωγή διανυσμάτων δοκιμής

Σύντομη Περιγραφή

Με την εκκίνηση της εφαρμογής εμφανίζονται ο αρχικός πίνακας δοκιμής και ο αντίστοιχος συμπιεσμένος. Επιπλέον εμφανίζεται μια αναφορά ως προς την απόδοση των επιλογών που κάνει ο χρήστης όσον αφορά τους κανόνες που χρησιμοποιεί.

Χειριστές

Οποιοδήποτε

Ροή Γεγονότων

Εκκίνηση εφαρμογής.
Ο χρήστης μελετά τον αρχικό πίνακα δοκιμής και αναζητά τους κατάλληλους κανόνες.
Κλείσιμο της εφαρμογής.
Εγγραφή κανόνων από τον χρήστη σε ένα αρχείο txt.
Επανεκκίνηση της εφαρμογής.
Προβολή συμπιεσμένου πίνακα δοκιμής και αποτελεσμάτων.
Η περίπτωση χρήσης τελειώνει με την έξοδο από την εφαρμογή.

Κατάσταση Εξόδου

7.4 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

7.4.1. Πρώτο βήμα-Εκτύπωση κανόνα 102

Αρχικός στόχος ήταν η υλοποίηση μιας συνάρτησης η οποία θα εκτύπωνε στην οθόνη τον κανόνα 102 των κυψελιδικών αυτομάτων με την βοήθεια ενός δισδιάστατου πίνακα. Η συνάρτηση πρέπει να εκτυπώνει 256 βήματα του κανόνα 102 καθώς μετά από 256 βήματα μπορούμε να μετρήσουμε δυαδικά μέχρι την δεκαδική τιμή 255 όπου είναι και το επιθυμητό αποτέλεσμα. Οπότε το σύνολο των γραμμών του πίνακα είναι 256. Το σύνολο των στηλών από την στιγμή που ξέρουμε μέχρι ποια τιμή θέλουμε να μετρήσουμε είναι 128+1+1=130. Η τελευταία στήλη έχει παντού 1 λόγω του κανόνα 102 και η πρώτη στήλη είναι πρόσθετη γιατί αλλιώς θα υπήρχε πρόβλημα στις συγκρίσεις των τριάδων των κελιών κατά την διάρκεια της εφαρμογής των υπό-κανόνων του κανόνα 102. Οι 128 στήλες υπολογίζονται απλά και μόνο κατανοώντας τον τρόπο ανάπτυξης του κανόνα 102 ο οποίος στις στήλες1,2,4,8,16,32,64,128 από το τέλος προς την αρχή εναλλάσσει αντίστοιχα τα 0 και 1 έτσι ώστε να μπορούμε συμπτύσσοντας αυτές τις στήλες να μετρήσουμε δυαδικά έως το 255. Με αυτόν τον τρόπο ο πίνακας μπορεί να γίνει στατικός. Σε αυτό το σημείο πρέπει να αναφέρουμε ότι έχει παραληφθεί το δεξί μέρος του πίνακα του κυψελιδικού το οποίο λόγο των υπό-κανόνων έχει τιμές μόνο μηδενικές. Το κομμάτι αυτό αφαιρέθηκε πρώτον γιατί δεν επηρεάζει σε καμία περίπτωση τον στόχο μας και δεύτερον διότι θα έπρεπε ο πίνακας να έχει διπλάσιες στήλες που σημαίνει και μεγαλύτερη κατανάλωση μνήμης. Παρακάτω δίνεται ο κώδικας ο οποίος μηδενίζει τον πίνακα και αρχικοποιεί το δεξιότερο στοιχείο της πρώτης γραμμή με '1' έτσι ώστε να μπορέσει να 'αναπτυχθεί' ο κανόνας 102.

```
for (i=0; i<256; i++)       {

    for(j=0; j<130; j++)    {

            K[i][j]=0;

           if (i==0 && j==129)

                K [i][j]=1;     } }
```

Στην συνέχεια λαμβάνονται υπόψη οι υπό-κανόνες του κανόνα 102 και έτσι αναπτύσσεται το κυψελιδικό.

Το σύνολο των υπό-κανόνων είναι 8 και δηλώνουν τα εξής (με την σειρά που βρίσκονται στον κώδικα):

Έστω ότι τα κελιά που ελέγχονται είναι τα j-1,j,j+1.

1. Όταν όλα τα κελιά είναι 1 τότε κάνε το κελί [i+1,j] '0'.
2. Όταν το j-1 και j είναι 1 και το j+1 είναι 0 τότε κάνε το [i+1,j] '1'.
3. Όταν το j-1 και j+1 είναι 1 και το j είναι 0 τότε κάνε το [i+1,j] '1'.
4. Όταν το j και j+1 είναι 0 και το j-1 είναι 1 τότε κάνε το [i+1,j] '0'.
5. Όταν το j και j+1 είναι 1 και το j-1 είναι 0 τότε κάνε το [i+1,j] '0'.
6. Όταν το j-1 και j+1 είναι 0 και το j είναι 1 τότε κάνε το [i+1,j] '1'.
7. Όταν το j-1 και j είναι 0 και το j+1 είναι 1 τότε κάνε το [i+1,j] '1'.
8. Όταν το j-1,j,j+1 είναι 0 τότε κάνε το [i+1,j] '1'.

```
for (i=0; i<256; i++)       {
        for(j=1; j<130; j++){
            if(K[i][j-1]==1 && K[i][j]==1 && K[i][j+1]==1)
                K[i+1][j]=0;
            else if(K[i][j-1]==1 && K[i][j]==1 && K[i][j+1]==0)
                K[i+1][j]=1;
            else if (K[i][j-1]==1 && K[i][j+1]==1 && K[i][j]==0)
                K[i+1][j]=1;
            else if (K[i][j-1]==1 && K[i][j]==0 && K[i][j+1]==0)
                K[i+1][j]=0;
            else if (K[i][j-1]==0 && K[i][j]==1 && K[i][j+1]==1)
                K[i+1][j]=0;
            else if (K[i][j-1]==0 && K[i][j+1]==0 && K[i][j]==1)
                K[i+1][j]=1;
            else if (K[i][j-1]==0 && K[i][j]==0 && K[i][j+1]==1
                K[i+1][j]=1;
            else
                K[i+1][j]=0; }  }
```

Τέλος, απλά εκτυπώνεται ο γεμάτος πλέον πίνακας.

```
for (i=0; i<256; i++)       {
        for(j=0; j<130; j++){
            printf("%d",K[i][j]); }
            printf("\n");         }
```

Όλα τα παραπάνω υλοποιούνται μέσα στην συνάρτηση Cellular256( ).

7.4.2. Δεύτερο Βήμα-Παραγωγή συμπιεσμένου πίνακα κυψελιδικού

Στην συνέχεια του προγράμματος έπρεπε να συμπτύξουμε τις στήλες οι οποίες με τον τρόπο που εναλλάσσουν τα 0 και 1 μπορούν να μετρήσουν έως το 255.Στην αρχή μηδενίζουμε τον πίνακα CP ο οποίος έχει 256 γραμμές και 9 στήλες. Ο αριθμός των γραμμών επεξηγήθηκε στο πιο πάνω κομμάτι. Ο αριθμός των στηλών είναι 9 καθώς οι 8 πρώτες στήλες χρειάζονται για την αποθήκευση των bits που θα μετρήσουν μέχρι την τιμή 255 και η $9^{η}$ στήλη για την εισαγωγή της αντίστοιχης δεκαδικής τιμής, έτσι ώστε να γνωρίζει άμεσα ο χρήστης την

αναπαράσταση των αριθμών σε δυαδική μορφή. Ο παρακάτω κώδικας μηδενίζει τον πίνακα CP.

```
for(k=0; k<256; k++)      {
        for(l=0; l<9; l++){
            CP[k][l]=0;     }}
```

Μετά τον μηδενισμό του πίνακα πρέπει να γίνει η μεταφορά των στηλών του πίνακα K στον πίνακα CP. Βλέπουμε ότι οι στήλες που μας ενδιαφέρουν από τον πίνακα K βρίσκονται στην ίδια θέση με τον αριθμό που μετράνε. Πιο συγκεκριμένα η στήλη που θα αντιπροσωπεύει τον αριθμό 1 βρίσκεται στην $1^{η}$ στήλη του πίνακα K μετρώντας από το τέλος προς την αρχή. Η στήλη που αντιπροσωπεύει τον αριθμό 2 βρίσκεται στην $2^{η}$ στήλη του πίνακα K, η στήλη που αντιπροσωπεύει τον αριθμό 4 βρίσκεται στην $4^{η}$ στήλη του πίνακα K … μέχρι την στήλη 128 της οποίας το bit αντιπροσωπεύει τον αριθμό 128. Δηλαδή η αφαίρεση 129-128=1 μας δίνει την στήλη του πρώτου bit. Η αφαίρεση 129-1=128 μας δίνει την στήλη του τελευταίου bit. Με αυτό το σκεπτικό αναπτύχθηκε ο παρακάτω κώδικας.

```
for(p=129; p>-1; p--){
    if(p==129-128)                {
            for (i=0; i<256; i++) {
               CP[i][0]=K[i][p];   }}
          if(p==129-64)           {
          for (i=0; i<256; i++)   {
                CP[i][1]=K[i][p]; }}
           if(p==129-32)           {
            for (i=0; i<256; i++)  {
                CP[i][2]=K[i][p];  }}
          if(p==129-16)           {
            for (i=0; i<256; i++)  {
                CP[i][3]=K[i][p];   }}
         else if(p==129-8)        {
           for (i=0; i<256; i++)   {
                CP[i][4]=K[i][p];   }}
          else if(p==129-4)       {
          for (i=0; i<256; i++)    {
                CP[i][5]=K[i][p];   }}
            else if(p==129-2)     {
```

```
          for (i=0; i<256; i++)     {
                CP[i][6]=K[i][p];}}
         else if(p==129-1)          {
        for (i=0; i<256; i++)       {
           CP[i][7]=K[i][p];      }}
                                    }
```

Αφού έχουν εισαχθεί οι στήλες στον πίνακα CP μένει να βάλουμε στην τελευταία του στήλη τους αντίστοιχους δεκαδικούς αριθμούς. Αυτό πραγματοποιείται με τον παρακάτω κώδικα. Σημείωση: βάζουμε στην $9^{η}$ στήλη το –i για λόγους ευκρίνειας καθώς υπήρχε πρόβλημα στον διαχωρισμό της δυαδικής τιμής από την δεκαδική.

```
for (i= 0; i<256; i++){
        CP[i][8]= -i; }
```

Τέλος, απλά εκτυπώνεται ο γεμάτος πλέον πίνακας CP.

```
for(k=0; k<256; k++)   {
        for(l=0; l<9; l++) {
        printf("%d",CP[k][l]);}
        printf("\n")          }
```

Όλα τα παραπάνω υλοποιούνται μέσα στην συνάρτηση CollapsedCellular( ).

7.4.3. Τρίτο Βήμα-Δημιουργία πίνακα δοκιμής

Στην συνέχεια του προγράμματος πρέπει να δημιουργηθεί ο πίνακας δοκιμής ο οποίος θα έχει μοναδικές τιμές σε ταξινομημένη σειρά και σε δυαδική μορφή.

Σε αυτό το σημείο το πρόγραμμα δέχεται ένα αρχείο με όνομα MatrixA.txt το οποίο περιέχει τις τιμές του πίνακα δοκιμής σε δεκαδική μορφή. Το πρόγραμμα διαβάζει μια προς μια τις τιμές του αρχείου και τις μετρά με την χρήση της μεταβλητής arithmos. Η διαδικασία ολοκληρώνεται όταν βρεθεί η τιμή 500 που δηλώνει και το τέλος του αρχείου. Παρακάτω δίνεται ένα παράδειγμα ενός αρχείου MatrixA.txt.

```
48 48 48 48 48 48 48 48 60 60 60 60 60 60 60 16 16 16 16 16 16 20 20 20 20 20 20 20 20  40
40 40 40 40 40 40 40 40 40 40 40 40 16 16 65 65 65 65 65 16 16 16 16 16 16 16 16 16 20 20
20 20 20 20 20 16 16 16 16 16 16 16 16 16 16 65 65 65 65 65 65 65 65 65 65 65 65 60 60 60 60
60 60 60 60 48 48 48 48 48 48 48 48 48 48 48 48 48  500
```

Η διαδικασία πραγματοποιείται με τον παρακάτω κώδικα.

```
FILE *fp = fopen("MatrixA.txt", "r");

if (fp == NULL) printf("****File not found\n****");

while(ch!=500) {

    fscanf(fp,"%d",&ch);

    if(ch==500)

    break;

    else

    arithmos=arithmos+1;}

fclose(fp);
```

Τελειώνοντας με την παραπάνω διαδικασία γνωρίζουμε τον αριθμό των τιμών στο αρχείο Matrix.txt. Δημιουργούμε έτσι έναν πίνακα Α με μέγεθος τον συνολικό αριθμό τιμών του αρχείου και περνάμε τις τιμές του αρχείου μια προς μια στον πίνακα Α. Αυτό πραγματοποιείται με τον παρακάτω κώδικα.

```
FILE *fp2 = fopen("MatrixA.txt", "r");

if (fp == NULL) printf("****File not found\n****");

i=0;

while(ck!=500){

    fscanf(fp2,"%d",&ck);

    if(ck==500)

    break;

    A[i]=ck;

    i=i+1; }

printf("%d\n\n\n\n");

fclose(fp2);
```

Μετά το τέλος της παραπάνω διαδικασίας ο πίνακας Α έχει όλες τις τιμές του .txt αρχείου. Αυτό που μένει τώρα είναι να τις ταξινομήσουμε. Για να το κάνουμε αυτό χρησιμοποιούμε

την ταξινόμηση της φυσαλίδας, η οποία ταξινομεί τις τιμές από την μικρότερη στην μεγαλύτερη τιμή. Όσο το doMore είναι 1 δηλαδή όσο πραγματοποιούνται αντιμεταθέσεις η διαδικασία συνεχίζεται. Η διαδικασία αυτή πραγματοποιείται με τον παρακάτω κώδικα.

```
int doMore=1;

while(doMore==1)                {

       doMore=0;

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

          if(A[i]>A[i+1])       {

          temp=A[i+1];

          A[i+1]=A[i];

          A[i]=temp;

          doMore=1;             } } }
```

Μετά το τέλος της διαδικασίας ο πίνακας Α έχει ταξινομημένες όλες του τις τιμές. Τώρα επειδή οι τιμές αυτές δεν είναι μοναδικές αλλά μπορεί μια τιμή να υπάρχει παραπάνω από μια φορές πρέπει να πάρουμε από τον πίνακα Α μια φορά κάθε τιμή. Στην αρχή δημιουργούμε έναν πίνακα Β μεγέθους ίδιου με τον Α και τον γεμίζουμε με -1 τιμές και όχι με 0 όπως συνηθίζεται τις πιο πολλές φορές. Αυτό το κάνουμε γιατί ο Α δεν μπορεί να περιέχει αρνητικές τιμές αλλά μπορεί να περιέχει τιμές 0 και θα υπάρξει πρόβλημα στην συνέχεια. Αυτό πραγματοποιείται με μια επαναληπτική δομή η οποία εξετάζει εάν η τιμή του A[i] είναι διαφορετική από την τιμή A[i+1].Αν ισχύει αυτό σημαίνει ότι υπάρχει αλλαγή τιμής οπότε αντιγράφουμε την τιμή A[i] στον πίνακα Β. Για παράδειγμα, έστω ότι ο Α έχει τις τιμές 0 0 0 0 3 3 3 3 3 5 5 6 8 8 8. Αν ο δείκτης φτάσει στον 4^{ο} αριθμό που είναι το 0 και ο αμέσως επόμενος είναι το 3 τότε θα αντιγράψει την τιμή 0 στον Β. Μετά θα ελέγξει τα 3 μεταξύ τους και δεν θα κάνει απολύτως τίποτα. Όταν ο δείκτης φτάσει στο 3 πριν από το 5,αφού υπάρχει αλλαγή τιμής αντιγράφουμε το 3 στον πίνακα Β. Η διαδικασία αυτή πραγματοποιείται από τον παρακάτω αλγόριθμο.

```
int B[arithmos];

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

B[i]=-1;              }

j=0;

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

    if(A[i]!=A[i+1])    {

    B[j]=A[i];

    j=j+1;              }}
```

Όταν τελειώσει η παραπάνω διαδικασία, ο πίνακας Β έχει ταξινομημένες τις μοναδικές τιμές του πίνακα Α και συνεπώς του αρχείου. Όμως από ένα σημείο και μετά έχει τιμές -1. Για να

απομονώσουμε αυτές τις τιμές μετράμε τις μοναδικές τιμές του πίνακα B και δημιουργούμε έναν πίνακα C με μέγεθος των αριθμό των μοναδικών τιμών του B. Ο αριθμός των μοναδικών τιμών αποθηκεύεται στην μεταβλητή numbers. Στην συνέχεια απλά αντιγράφουμε από τον πίνακα B στον πίνακα C τις τιμές που είναι διάφορες του -1 και εκτυπώνουμε τον πίνακα C. Αυτή η διαδικασία πραγματοποιείται με τον παρακάτω αλγόριθμο.

```
int numbers=0;

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

   if(B[i]!=-1)

   numbers=numbers+1; }

int C[numbers];

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

       C[i]=B[i];}

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

printf("%d\n\n",C[i]);}
```

Το μόνο που μένει να κάνουμε για να δημιουργήσουμε τον πίνακα δοκιμής είναι να μετατρέψουμε τις δεκαδικές τιμές του πίνακα C σε δυαδικές και να τις αποθηκεύσουμε σε έναν πίνακα. Πρώτα πρέπει να βρούμε τον αριθμό των στηλών που θα χρειαστεί ο καινούργιος πίνακας καθώς ο αριθμός των γραμμών είναι γνωστός. Ο αριθμός των στηλών δηλώνει τον αριθμό των bits που χρειάζεται η μέγιστη τιμή του πίνακα C για να αναπαρασταθεί. Αφού ο πίνακας είναι ταξινομημένος άρα η μέγιστη τιμή βρίσκεται στην τελευταία θέση. Οπότε εκχωρούμε στην μεταβλητή MaxBigMatrix την τιμή του τελευταίου κελιού του πίνακα καθώς και στην μεταβλητή MinBigMatrix την ελάχιστη τιμή του πίνακα που θα μας χρησιμεύσει στην συνέχεια. Τώρα για να βρούμε πόσα bits θέλει η μέγιστη τιμή για να αναπαρασταθεί δυαδικά πρέπει να ελέγξουμε ανάμεσα σε ποιες τιμές βρίσκεται. Για παράδειγμα εάν βρίσκεται ανάμεσα στις τιμές >63 και <128 τότε χρειάζεται 7 bits για να αναπαρασταθεί. Ο αριθμός των bits αποθηκεύεται στην μεταβλητή counter. Όταν αποθηκεύσουμε τον αριθμό των bits στην μεταβλητή counter δημιουργούμε έναν πίνακα BigMatrix [numbers][counter] και τον μηδενίζουμε. Τα παραπάνω πραγματοποιούνται με τον παρακάτω κώδικα.

```
MaxBigMatrix=C[numbers-1];

MinBigMatrix=C[0];

if(MaxBigMatrix>127 && MaxBigMatrix<256)

        counter=8;

    else if(MaxBigMatrix>63 && MaxBigMatrix<128)

        counter=7;

    else if(MaxBigMatrix>31 && MaxBigMatrix<64)

        counter=6;
```

```
    else if(MaxBigMatrix>15 && MaxBigMatrix<32)
        counter=5;
    else if(MaxBigMatrix>7 && MaxBigMatrix<16)
        counter=4;
    else if(MaxBigMatrix>3 && MaxBigMatrix<8)
        counter=3;
    else if(MaxBigMatrix>1 && MaxBigMatrix<3)
        counter=2;
    else if(MaxBigMatrix==1 || MaxBigMatrix==0)
        counter=1;
int BigMatrix[numbers][counter];
for(a=0; a<numbers; a++)     {
    for(b=0; b<counter; b++) {
            BigMatrix[a][b]=0;   }
            printf("\n"); }
```

Μετά το τέλος της παραπάνω διαδικασίας έχουμε δημιουργήσει τον πίνακα BigMatrix με τις διαστάσεις που χρειαζόμαστε. Τώρα πρέπει να τον γεμίσουμε με τις αντίστοιχες δυαδικές τιμές. Για να το πετύχουμε αυτό χρησιμοποιούμε τον πίνακα CP ο οποίος όπως έχει ήδη αναφερθεί μετράει δυαδικά από το 0 έως το 255. Για αυτήν την διαδικασία θα πρέπει να χρησιμοποιήσουμε ένα for το οποίο θα τρέχει για κάθε γραμμή του πίνακα BigMatrix και ένα δεύτερο for το οποίο θα τρέχει τον πίνακα CP. Μετά τα δυο for θα πρέπει να γίνεται ο έλεγχος εάν η τιμή του πίνακα C είναι όμοια με την τιμή που έχει ο πίνακας CP στην τελευταία στήλη του (δηλαδή εκεί που υπάρχουν οι δεκαδικές τιμές). Η συνθήκη αυτή θα ισχύσει οπωσδήποτε οπότε μπαίνουμε σε ένα ακόμα for το οποίο τρέχει ανάλογα με την τιμή του counter. Πιο συγκεκριμένα, εάν π.χ ο counter έχει τιμή 5 τότε το for το οποίο ξεκινάει πάντα από την στήλη 7 του πίνακα CP θα 'τρέξει' μέχρι την τιμή 7-counter>2. Δηλαδή θα αντιγραφούν οι τιμές C[][7] C[][6] C[][5] C[][4] C[][3]. Η πρώτη διάσταση είναι ανάλογη στο που βρέθηκε ο αριθμός. Οπότε αυτές οι τιμές θα μπουν στην αντίστοιχη γραμμή του πίνακα BigMatrix. Όταν γίνουν όλες οι αντιστοιχίσεις εκτυπώνεται ο πίνακας BigMatrix περιέχοντας τις δυαδικές τιμές. Η παραπάνω διαδικασία γίνεται με τον παρακάτω κώδικα.

```
for(a=0; a<numbers; a++)                    {
        for(b=0; b<256; b++)                {
            if(-CP[b][8]==C[a])
                for(i=7; i>7-counter; i--)   {
                    if(counter==8)
```

```
BigMatrix[a][i]=CP[b][i];
else if(counter==7)
BigMatrix[a][i-1]=CP[b][i];
else if(counter==6)
BigMatrix[a][i-2]=CP[b][i];
else if(counter==5)
BigMatrix[a][i-3]=CP[b][i];
else if(counter==4)
BigMatrix[a][i-4]=CP[b][i];
else if(counter==3)
BigMatrix[a][i-5]=CP[b][i];
else if(counter==2)
BigMatrix[a][i-6]=CP[b][i];
else if(counter==1)
BigMatrix[a][i-7]=CP[b][i];             }}}
for(a=0; a<numbers; a++)                 {
    for(b=0; b<counter; b++)             {
        printf("%d ",BigMatrix[a][b]);   }
        printf("\n");                                }
```

Με την εκτύπωση του BigMatrix έχει ολοκληρωθεί πλέον η διαδικασία της δημιουργίας του πίνακα δοκιμής.

7.4.4. Τέταρτο Βήμα-Δημιουργία συμπιεσμένου πίνακα δοκιμής

Στην συνέχεια του προγράμματος πρέπει να δημιουργηθεί ο συμπιεσμένος πίνακας δοκιμής που είναι και ο τελικός στόχος.

Σε αυτό το σημείο το πρόγραμμα διαβάζει ένα αρχείο με όνομα KANONES.txt το οποίο περιέχει τους κανόνες με τους οποίους θα συμπτυχθεί ο πίνακας δοκιμής. Παρακάτω δίνεται ένα παράδειγμα ενός τέτοιου αρχείου.

```
1 -1000
2 -1000
-1000
3 -1000
4 -1000
-1000
```

```
5 -1000
-1000
```

Κάθε γραμμή του αρχείου αντιστοιχεί σε κάθε στήλη του πίνακα BigMatrix. Ο αριθμός που βρίσκεται σε κάθε γραμμή δηλώνει σε ποια στήλη του νέου πίνακα θα μπει η στήλη του BigMatrix. Εάν ο αριθμός είναι αρνητικός, τότε σημαίνει ότι στην στήλη του νέου πίνακα θα μπει η αντίστοιχη συμπληρωματική στήλη του BigMatrix. Εάν δεν υπάρχει τιμή σε κάποια γραμμή του αρχείου τότε αυτό σημαίνει ότι η συγκεκριμένη στήλη του BigMatrix δεν μπαίνει πουθενά καθώς δεν χρειάζεται. Το -1000 δηλώνει το τέλος της κάθε γραμμής.

Στο σημείο αυτό ο κώδικας ανοίγει το αρχείο KANONES.txt, διαβάζει τις απόλυτες τιμές και αποθηκεύει την μεγαλύτερη στην μεταβλητή max2. Αυτή η τιμή μας δείχνει πόσες στήλες θα χρειαστούμε για τον συμπιεσμένο πίνακα δοκιμής. Στην συνέχεια δημιουργούμε τον συγκεκριμένο πίνακα, τον οποίο τον ονομάζουμε SmallMatrix και έχει διαστάσεις [numbers]x[max2] και τον μηδενίζουμε. Παρακάτω δίνεται ο κώδικας που κάνει την παραπάνω διαδικασία.

```
FILE *fp3=fopen("KANONES.txt", "r");
if (fp3 == NULL) printf("****File not found\n****");
for(i=0;i<counter;i++)          {
    for(j=1;j=10000;j++)        {
    fscanf(fp,"%d",&ARITHMOS);
    if(ARITHMOS==-1000)
    break;
    if(max2<abs(ARITHMOS))
    max2=abs(ARITHMOS);    }}
fclose(fp3);
int SmallMatrix[numbers][max2];
for(i=0;i<numbers;i++)     {
    for(j=1;j<max2+1;j++) {
    SmallMatrix[i][j]=0;      }
    printf("\n");             }
```

Μετά το τέλος της παραπάνω διαδικασίας έχουμε δημιουργήσει τον συμπιεσμένο πίνακα δοκιμής και το μόνο που μένει είναι να διαβάσουμε τους κανόνες και να τον γεμίσουμε με δυαδικές τιμές.

Για να το κάνουμε αυτό ανοίγουμε εκ νέου το αρχείο KANONES.txt και διαβάζουμε του κανόνες. Όταν συναντήσουμε το -1000 τερματίζεται η επαναληπτική δομή. Εάν η τιμή που διαβάζουμε από το αρχείο είναι αρνητική τότε εάν η τιμή στον πίνακα BigMatrix είναι 0 την

κάνουμε 1 και το αντίστροφο, δηλαδή εισάγουμε στον καινούργιο πίνακα την συμπληρωματική στήλη του BigMatrix. Εάν η τιμή είναι μεγαλύτερη του μηδενός βάζουμε κανονικά τις τιμές από τον έναν πίνακα στον άλλο. Στο τέλος εκτυπώνεται ο πίνακας SmallMatrix και έτσι έχουμε πλέον τον συμπιεσμένο πίνακα δοκιμής. Η παραπάνω διαδικασία γίνεται με τον παρακάτω κώδικα.

```
FILE *fp4 = fopen("KANONES.txt", "r");
if (fp4 == NULL) printf("****File not found\n****");


     for(i=0;i<counter;i++)            {
          for(j=0;j=100;j++)           {
                fscanf(fp,"%d",&cp);
                if(cp==-1000)
                break;
                for(a=0;a<numbers;a++) {
                if(cp<0)
                   if(BigMatrix[a][i]==0)
                   SmallMatrix[a][-cp]=1;
                   else
                   SmallMatrix[a][-cp]=0;
                 else
                 SmallMatrix[a][cp]=BigMatrix[a][i]; }}}
    fclose(fp4);
     for(i=0;i<numbers;i++)     {
        for(j=1;j<max2+1;j++)   {
          printf("%d ",SmallMatrix[i][j]);}
          printf("\n");}
```

7.4.5. Πέμπτο Βήμα-Παρουσίαση Αποτελεσμάτων

Στο τέλος του προγράμματος ο χρήστης καλείται να εισάγει την μέγιστη και την ελάχιστη δεκαδική τιμή του συμπιεσμένου πίνακα SmallMatrix. Αυτό γίνεται για να δείξουμε ποιο μέρος του κυψελιδικού του πίνακα K πρέπει να παραχθεί για να παράγουμε τα διανύσματα για τον πίνακα μας. Στην συνέχεια εκτυπώνουμε το κυψελιδικό από min έως max του SmallMatrix. Αυτό γίνεται με τον παρακάτω κώδικα.

```
printf("doste elaxisti kai megisti timi tou SmallMatrix\n\n");
```

```
scanf("%d",&MinSmallMatrix);

scanf("%d",&MaxSmallMatrix);

for(i=MinSmallMatrix;i<MaxSmallMatrix+1;i++){

    for(j=0;j<130;j++)    {

 printf("%d",K[i][j]);}
```

Στην συνέχεια γίνεται γίνεται ένας υπολογισμός του αριθμού των διανυσμάτων που πρέπει να παραχθούν για τον αρχικό πίνακα δοκιμής και για τον συμπιεσμένο. Τα αποτελέσματα αποθηκεύονται στις μεταβλητές TestVectorNumberBigMatric και TestVectorNumberSmallMatrix αντίστοιχα. Ο τύπος που χρησιμοποιείται για κάθε πίνακα είναι max-min+1 δηλαδή η μέγιστη δεκαδική τιμή μείον την ελάχιστη συν 1. Κατόπιν υπολογίζεται το ποσοστό εγκυρότητας κάθε πίνακα από τον τύπο: efficiency=(numbers*100)/TestVectorNumberSmallMatrix-BigMatrix, δηλαδή από την διαίρεση του αριθμού των διανυσμάτων που έχουμε με τον αριθμό των διανυσμάτων που πρέπει να παραχθούν επί 100. Το αποτέλεσμα μας δίνει το ποσοστό εγκυρότητας των παραχθέντων διανυσμάτων. Όλα τα αποτελέσματα εκτυπώνονται στο τέλος στο πεδίο ΑΝΑΦΟΡΑ. Ο παρακάτω κώδικας υλοποιεί την παραπάνω διαδικασία.

```
TestVectorNumberBigMatrix=MaxBigMatrix-MinBigMatrix+1;

printf("O arithmos ton Test Vectors pou prepei na paraxthoun gia ton pinaka
dokimis einai %d\n\n\n",TestVectorNumberBigMatrix);

TestVectorNumberSmallMatrix=MaxSmallMatrix-MinSmallMatrix+1;

printf("O arithmos ton Test Vectors pou prepei na paraxthoun gia ton
sumpiesmeno pinaka einai %d\n\n\n",TestVectorNumberSmallMatrix);

printf("***********ANAFORA************\n\n\n");

efficiencyBigMatrix=(numbers*100)/TestVectorNumberBigMatrix;

efficiencySmallMatrix=(numbers*100)/TestVectorNumberSmallMatrix;

printf("Gia ton pinaka dokimis prepei na paraxthoun %d dianismata ek ton
opoion to %d tis ekato einai
eguro\n\n\n",TestVectorNumberBigMatrix,efficiencyBigMatrix);

printf("Gia ton sympiesmeno pinaka prepei na paraxthoun %d dianismata ek ton
opoion to %d tis ekato einai
eguro\n\n\n",TestVectorNumberSmallMatrix,efficiencySmallMatrix);
```

Όλα τα παραπάνω υλοποιούνται μέσα στην συνάρτηση MatrixA( ).

Οι παραπάνω συναρτήσεις καλούνται από την main μέσα στην οποία γίνονται και οι δηλώσεις των δυο αρχικών πινάκων K και CP. Παρακάτω δίνεται ο κώδικας.

```
main ()         {

int K[256][130];

int CP[256][9];
```

```
Cellular255(K);

CollapsedCellular(K,CP);

MatrixA(K,CP);  }
```

8. Αξιολόγηση προτεινόμενης προσέγγισης

Μετά την ολοκλήρωση της ανάπτυξης του λογισμικού, πρέπει να πραγματοποιηθούν μετρήσεις, πρώτον για τον έλεγχο της σωστής λειτουργίας του και δεύτερον για την αξιολόγηση της απόδοσης του. Για την επίτευξη αυτών των στόχων χρησιμοποιήθηκαν 3 διαφορετικοί πίνακες δοκιμής, έτσι ώστε να έχουμε μια όσο το δυνατόν περισσότερο εμπεριστατωμένη άποψη για την λειτουργία του λογισμικού.

Σε αυτό το σημείο να υπενθυμίσουμε τους κανόνες σύμφωνα με τους οποίους μπορούμε να συμπιέσουμε έναν πίνακα δοκιμής.

Διαδικασίες Προ-επεξεργασίας :

- αποβολή σταθερών στηλών
- συγχώνευση ίδιων στηλών
- συμπληρωματική συγχώνευση στηλών και ,

Βασικές Διαδικασίες :

- μεταλλαγή στηλών
- συμπληρωματική παραγωγή στηλών.

Παράδειγμα 1

Για την πρώτη μέτρηση χρησιμοποιήθηκε ένας πίνακας δοκιμής με τις παρακάτω τιμές(αρχείο MatrixA.txt):

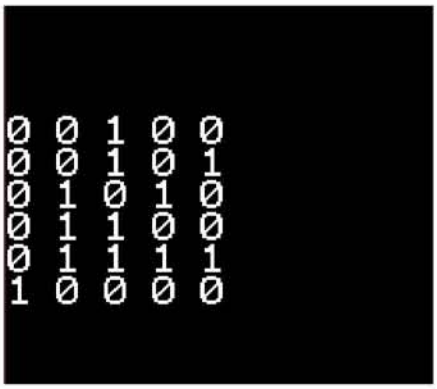
```
48 48 48 48 48 48 48 48 60 60 60 60 60 60 60 16 16 16 16 16 16 20 20 20 20
20 20 20 20  40 40 40 40 40 40 40 40 40 40 40 40 16 16 65 65 65 65 65 16 16
16 16 16 16 16 16 16 16 20 20 20 20 20 20 16 16 16 16 16 16 16 16 16 16 65
65 65 65 65 65 65 65 65 65 65 65 60 60 60 60 60 60 60 60 48 48 48 48 48 48
48 48 48 48 48 48  500
```

Με την εισαγωγή του παραπάνω αρχείου στο πρόγραμμα προκύπτει ο παρακάτω πίνακας δοκιμής.

Βλέπουμε ότι ο πίνακας δοκιμής περιέχει τις τιμές 16,20,40,48,60,65 και πιο κάτω βλέπουμε την δυαδική αναπαράσταση του. Τώρα πρέπει να βρούμε τους κατάλληλους κανόνες σύμφωνα με τους οποίους θα συμπιεστεί ο συγκεκριμένος πίνακας. Βλέπουμε ότι η προτελευταία στήλη του είναι σταθερή οπότε μπορούμε να την αφαιρέσουμε. Επίσης βλέπουμε ότι η πρώτη και η τελευταία στήλη είναι όμοιες οπότε μπορούμε να τις συγχωνεύσουμε. Με αυτόν τον τρόπο δημιουργούμε ένα αρχείο με τους εξής κανόνες:

```
1 -1000

2 -1000

3 -1000

4 -1000

5 -1000
```

Μετά την εφαρμογή των κανόνων προκύπτει ο παρακάτω πίνακας:



Για να παράγουμε τον παραπάνω πίνακα αρκεί να παράγουμε το παρακάτω κομμάτι του κυψελιδικού το οποίο αναπτύσσεται από την ελάχιστη έως την μέγιστη τιμή του πίνακα (4-16).

Τέλος εκτυπώνεται η αναφορά η οποία μας ενημερώνει για το τελικό αποτέλεσμα.

```
O arithmos ton Test Vectors pou prepei na paraxthoun gia ton pinaka dokimis einai 50

O arithmos ton Test Vectors pou prepei na paraxthoun gia ton sumpiesmeno pinaka einai 13

**********ANAFORA**********

Gia ton pinaka dokimis prepei na paraxthoun 50 dianismata ek ton opoion to 12 tis ekato einai eguro

Gia ton sympiesmeno pinaka prepei na paraxthoun 13 dianismata ek ton opoion to 46 tis ekato einai eguro
```

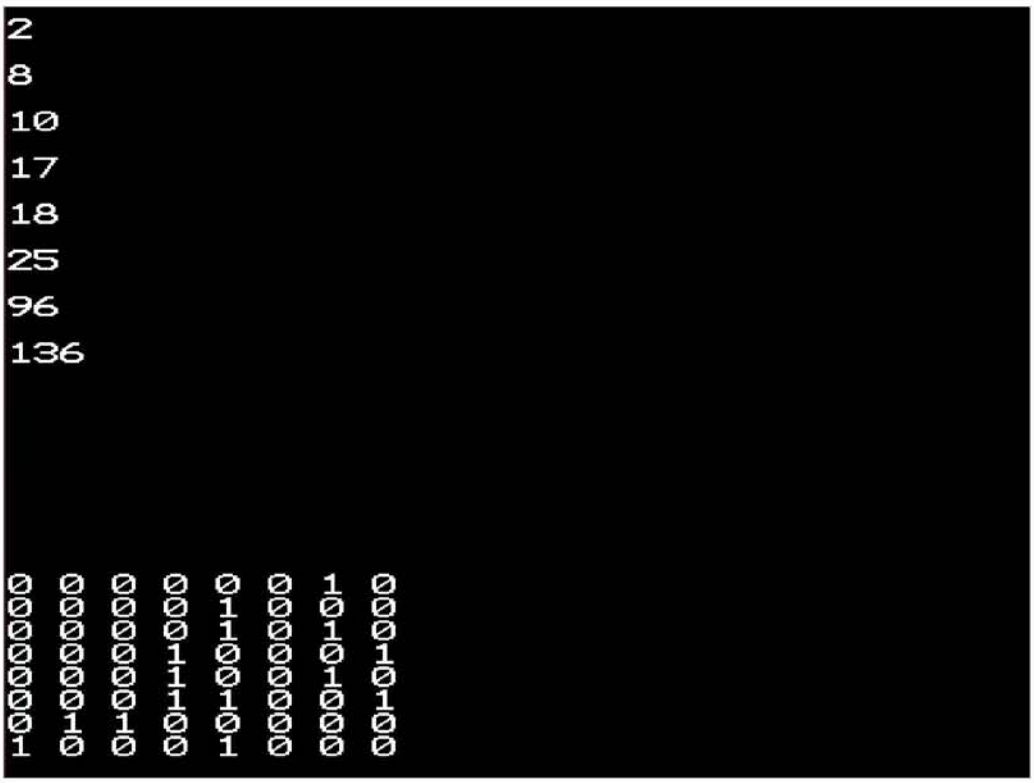
Βλέπουμε ότι για τον πίνακα δοκιμής πρέπει να παραχθούν συνολικά 50 διανύσματα ενώ για τον συμπιεσμένο πίνακα πρέπει να παραχθούν μόνο 13. Αυτό έχει ως αποτέλεσμα να ανεβαίνει αξιοσημείωτα το ποσοστό εγκυρότητας των διανυσμάτων και από εκεί που ήταν αρχικά στο 12% να ανεβαίνει στο 46%.

Παράδειγμα 2

Για την δεύτερη μέτρηση χρησιμοποιήθηκε ένας πίνακας δοκιμής με τις παρακάτω τιμές(αρχείο MatrixA.txt):

```
2 2 2 2 2 2 2 2 8 8 8 8 8 8 8 8 10 10 10 10 10 10 10 10 10 10 10 10 17 17 17 17
17 18 18 18 18 25 25 25 96 96 96 96 96 96 96 136 136 136 136 136  10 10 10
10 10 10 10 10 10 10 10 10 17 17 17 17 18 18 17 17 17 17 18 18 17 17 17 17
18 18 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 136 136 136 136 136 136 136
136 18 18 18 18 25 25 25 136 136 136 136 136  10 10 10 10 10 10 10 10 10 10
10 10 17 17 17 17 18 18 17 17 17 17 18 18 18 18 25 25 25 96 96 96 96 96 96
96 136 136 136 136 136  10 10 10 10 10 10 10 10 10 10 10 10 17 17 17 17 18
18 17 17 17 17 10 10 10 10 10 10 10 10 10 10 10 10 17 17 17 17 18 18 17 17
17 17 18 18 17 17 17 17 18 18 8 8 8 8 8 8 10 10 10 10 10 10 10 10 10 10 10
10 17 17 17 17 18 18 17 17 17 17 18 18 17 17 17 17 18 18 8 8 8 8 8 8 10 10
10 10 10 10 10 10 10 10 10 10 17 17 17 17 18 18 17 17 17 17 18 18 17 17 17
17 18 18 8 8 8 8 8 8 2 2 2 2 2 2 2 8 8 8 8 8 8 8 8 10 10 10 10 10 10 10 10
10 10 10 17 17 17 17 17 18 18 18 18 25 25 25 96 96 96 96 96 96 96 136 136
136 136 136  10 10 10 10 10 10 10 10 10 10 10 10 17 17 17 17 18 18 17 2 2 2
2 2 2 2 2 8 8 8 8 8 8 8 8 10 10 10 10 10 10 10 10 10 10 10 17 17 17 17 17 18
18 18 18 25 25 25 96 96 96 96 96 96 96 136 136 136 136 136  10 10 10 10 10
10 10 10 10 10 10 10 17 17 17 17 18 18 17 500
```

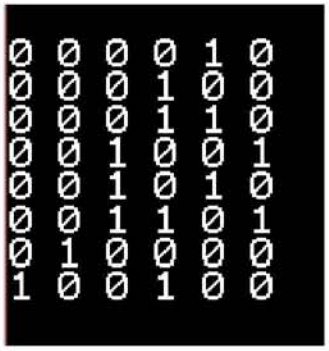
Με την εισαγωγή του παραπάνω αρχείου στο πρόγραμμα προκύπτει ο παρακάτω πίνακας δοκιμής:



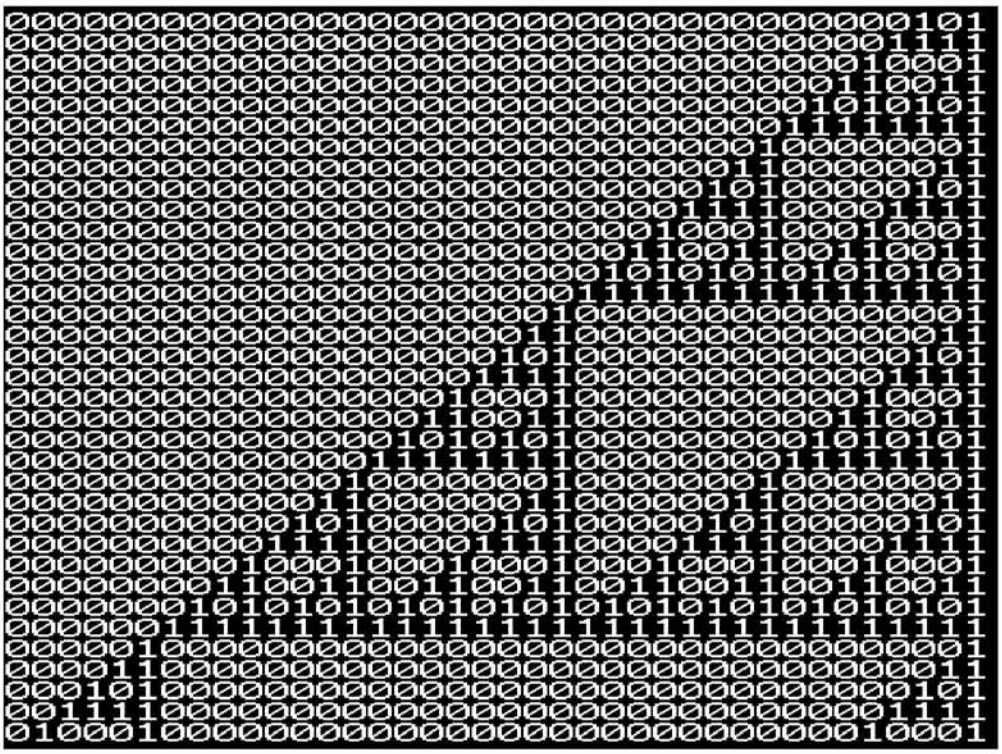
Παρατηρούμε ότι ο πίνακας δοκιμής περιέχει τις τιμές 2,8,10,17,18,25,96,136 και πιο κάτω βλέπουμε την δυαδική αναπαράστασή του. Τώρα πρέπει να εφαρμόσουμε τους κατάλληλους κανόνες σύμφωνα με τους οποίους θα παραχθεί ο συμπτυγμένος πίνακας. Βλέπουμε ότι η $2^{η}$ και η $3^{η}$ στήλη είναι όμοιες οπότε μπορούν να συγχωνευθούν. Επίσης βλέπουμε ότι η $6^{η}$ στήλη είναι σταθερή οπότε μπορεί να παραληφθεί. Με αυτό το σκεπτικό δημιουργούμε τους παρακάτω κανόνες:

```
1 -1000

2 -1000

-1000

3 -1000

4 -1000

-1000

5 -1000

-1000
```

Μετά την εφαρμογή των κανόνων προκύπτει ο παρακάτω πίνακας:



Για να παράγουμε τον παραπάνω πίνακα αρκεί να παράγουμε το παρακάτω κομμάτι του κυψελιδικού το οποίο αναπτύσσεται από την ελάχιστη έως την μέγιστη τιμή του πίνακα (2-36).



Τέλος εκτυπώνεται η αναφορά η οποία μας ενημερώνει για το τελικό αποτέλεσμα:

```
0 arithmos ton Test Vectors pou prepei na paraxthoun gia ton pinaka dokimis einai 135

0 arithmos ton Test Vectors pou prepei na paraxthoun gia ton sumpiesmeno pinaka einai 35

xxxxxxxxxxANAFORAxxxxxxxxxxxxx

Gia ton pinaka dokimis prepei na paraxthoun 135 dianismata ek ton opoion to 5 tis ekato einai eguro

Gia ton sympiesmeno pinaka prepei na paraxthoun 35 dianismata ek ton opoion to 22 tis ekato einai eguro
```

Βλέπουμε ότι για τον πίνακα δοκιμής πρέπει να παραχθούν συνολικά 135 διανύσματα ενώ για τον συμπιεσμένο πίνακα πρέπει να παραχθούν μόνο 35. Αυτό έχει ως αποτέλεσμα να ανεβαίνει αξιοσημείωτα το ποσοστό εγκυρότητας των διανυσμάτων και από εκεί που ήταν αρχικά στο 5% να ανεβαίνει στο 22%.

Παράδειγμα 3

Για την τρίτη μέτρηση χρησιμοποιήθηκε ένας πίνακας δοκιμής με τις παρακάτω τιμές(αρχείο MatrixA.txt):

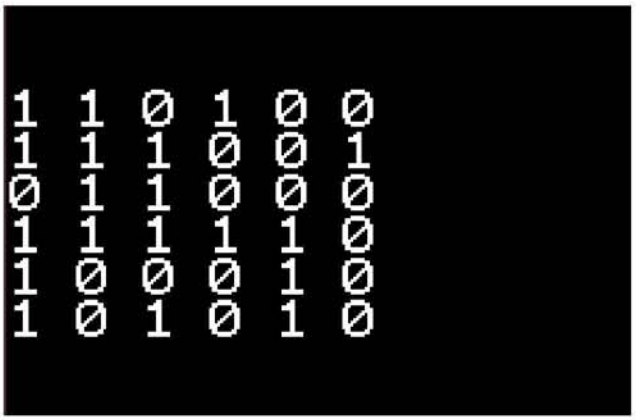
```
225 225 225 225 225 225 225 225 225 225 225 225 225 225 225 193 193 193 193
193 193 193 193 193 193 193 193 193 193 193 193 193 193 193 193 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 34 34 34 34
34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 177 177 177
177 177 177 177 177 177 177 225 225 225 225 41 41 41 41 41 34 34 34 34 34
193 193 193 193 193 193 500
```

Με την εισαγωγή του παραπάνω αρχείου στο πρόγραμμα προκύπτει ο παρακάτω πίνακας δοκιμής:

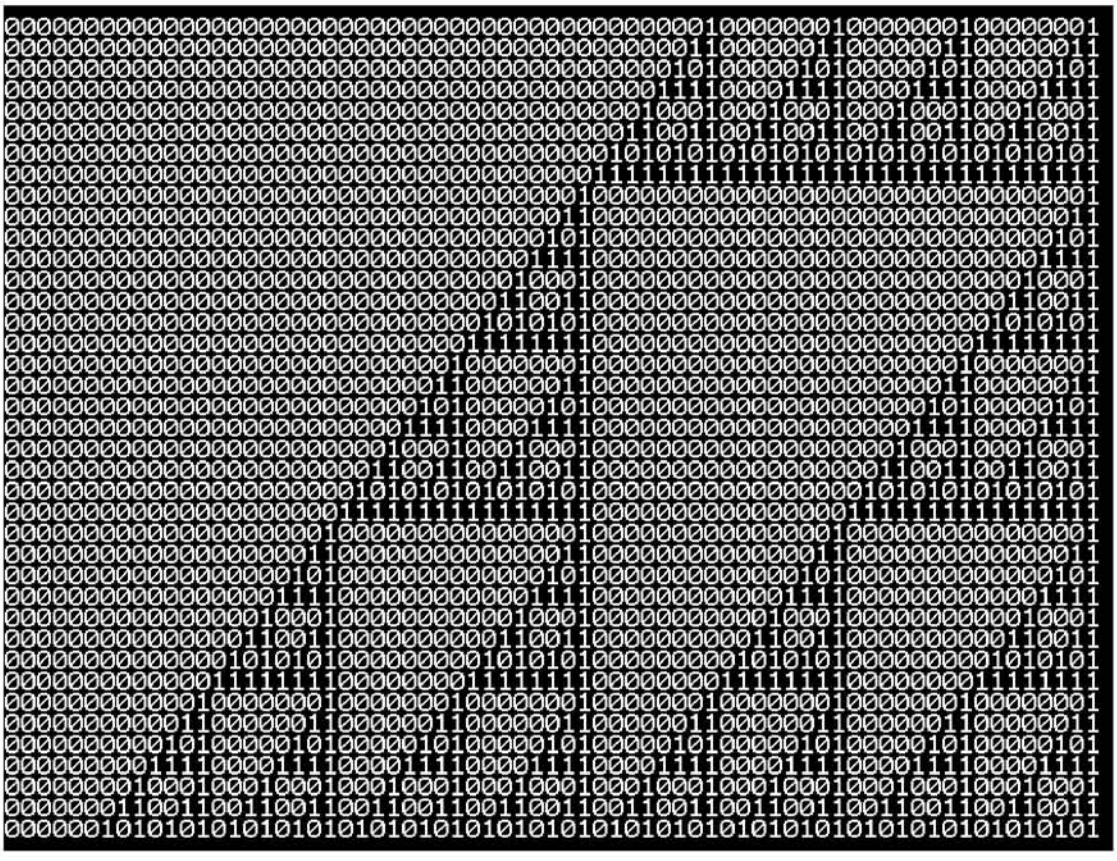
Παρατηρούμε ότι ο πίνακας δοκιμής περιέχει τις τιμές 17,34,41,177,193,225 και πιο κάτω βλέπουμε την δυαδική αναπαράστασή του. Τώρα πρέπει να εφαρμόσουμε τους κατάλληλους κανόνες σύμφωνα με τους οποίους θα παραχθεί ο συμπτυγμένος πίνακας. Βλέπουμε ότι η $6^{η}$ στήλη είναι σταθερή οπότε μπορεί να διαγραφεί ως σταθερή. Επίσης βλέπουμε ότι οι στήλες 7 και 8 είναι συμπληρωματικές οπότε μπορούν να συγχωνευθούν. Παράλληλα, ύστερα από δοκιμές μπορούμε να εφαρμόσουμε κάποιες μεταλλαγές στηλών και κάποιες παραγωγές συμπληρωματικών στηλών έτσι ώστε να μειώσουμε ακόμα περισσότερο το εύρος των διανυσμάτων. Πιο συγκεκριμένα μπορούμε να αλλάξουμε τις θέσεις των στηλών 1 και 5, και να βάλουμε στην πρώτη στήλη την συμπληρωματική της $5^{ης}$. Επίσης μπορούμε να βάλουμε στην θέση της δεύτερης στήλης την συμπληρωματική της. Με το σκεπτικό αυτό δημιουργούμε τους παρακάτω κανόνες:

```
5 -1000

-2 -1000

3 -1000

4 -1000

-1 -1000

-1000

6 -1000

-1000
```

Μετά την εφαρμογή των κανόνων προκύπτει ο παρακάτω πίνακας:



Για να παράγουμε τον παραπάνω πίνακα αρκεί να παράγουμε το παρακάτω κομμάτι του κυψελιδικού το οποίο αναπτύσσεται από την ελάχιστη έως την μέγιστη τιμή του πίνακα (24-62).



Τέλος εκτυπώνεται η αναφορά η οποία μας ενημερώνει για το τελικό αποτέλεσμα:

```
0 arithmos ton Test Vectors pou prepei na paraxthoun gia ton pinaka dokimis einai 209

0 arithmos ton Test Vectors pou prepei na paraxthoun gia ton sumpiesmeno pinaka einai 39

**********ANAFORA**********

Gia ton pinaka dokimis prepei na paraxthoun 209 dianismata ek ton opoion to 2 tis ekato einai eguro

Gia ton sympiesmeno pinaka prepei na paraxthoun 39 dianismata ek ton opoion to 15 tis ekato einai eguro
```

Βλέπουμε ότι για τον πίνακα δοκιμής πρέπει να παραχθούν 209 διανύσματα ενώ για τον συμπτυγμένο πίνακα μόνο 39. Αυτό έχει ως αποτέλεσμα να ανεβαίνει το ποσοστό εγκυρότητας των διανυσμάτων και από εκεί που ήταν αρχικά στο 2% να ανεβαίνει στο 15%.

9. Συμπεράσματα

Μετά την ολοκλήρωση της αξιολόγησης του προτεινόμενου λογισμικού ολοκληρώθηκε η παρούσα πτυχιακή εργασία.

Για την ανάπτυξη του λογισμικού μελετήθηκαν δυο διαφορετικοί επιστημονικοί κλάδοι, αυτός του testing ηλεκτρονικών κυκλωμάτων/συστημάτων και αυτός των κυψελιδικών αυτομάτων. Λαμβάνοντας υπόψη διαφορετικές προσεγγίσεις των συγκεκριμένων επιστημών για την παραγωγή διανυσμάτων δοκιμής και για τα κυψελιδικά αυτόματα υπήρξε δυνατό να φθάσουμε σε αυτό το αποτέλεσμα και να μπορέσουμε να συνδυάσουμε αποδοτικά τα πλεονεκτήματα της καθεμίας.

Η συγκεκριμένη υλοποίηση στόχευσε κυρίως στην αδυναμία προηγούμενων τεχνικών να παράγουν διανύσματα δοκιμής **γρήγορα**, **απλά** και όσο το δυνατόν πιο **αποδοτικά** από άποψη εγκυρότητας των παραγόμενων διανυσμάτων.

Μπορούμε εύκολα να διαπιστώσουμε ότι η προτεινόμενη προσέγγιση είναι αποδοτική καθώς με την χρήση ενός απαριθμητή ο οποίος κάνει χρήση των κανόνων των κυψελιδικών αυτομάτων, μπορούμε να έχουμε μεγαλύτερο εύρος εγκυρότητας στα παραγόμενα διανύσματα δοκιμής, μεγαλύτερη ταχύτητα στην παραγωγή τους καθώς επίσης και μεγαλύτερη απλότητα στην παραγωγή τους καθώς το μόνο που έχουμε να κάνουμε είναι να παράγουμε κάθε φορά συγκεκριμένο κομμάτι από τον κανόνα 102 στον οποίο βασίστηκε η συγκεκριμένη υλοποίηση.

Παρόλα κάθε προσέγγιση παραγωγής διανυσμάτων παρουσιάζει μια αδυναμία και αυτή είναι η απαιτούμενη μνήμη η οποία είναι απαραίτητη για την αποθήκευση των διανυσμάτων. Γνωρίζουμε ότι η μνήμη αποτελεί ακριβό υλικό στην σημερινή εποχή και πρέπει να διαχειρίζεται κατάλληλα. Με την υλοποίηση που πραγματοποιήθηκε μειώσαμε όσο ήταν δυνατό την απαιτούμενη μνήμη καθώς τα παραγόμενα διανύσματα είναι κατά πολύ λιγότερα από τι αρχικά.

Η συγγραφή του κώδικα έγινε με γνώμονα την απλότητα και την πληρότητα της εφαρμογής. Σίγουρα το προτεινόμενο λογισμικό επιδέχεται διορθώσεις αλλά αποτελεί μια πρώτη προσπάθεια προσέγγισης του τομέα του testing εισάγοντας τις ιδιότητες των κυψελιδικών αυτομάτων στον συγκεκριμένο χώρο.

10. ΠΗΓΕΣ

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Stephen Wolfram – *'A new kind of science'*,(2002)

2. Efthymia Arvaniti, Ilias Mavridis, Athanasios Kakarountas – *'Exploration of 2D Cellular Automata as Binary Sequence Generators'*, to appear in Proc. of IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2010, Lixouri Kefalonia, Greece, pp 41–45 July 2010.

3. Dimitrios Kagaris, Spyros Tragoudas – *'Cellular Automata for Generating Deterministic Test Sequences'* ,IEEE European Design and Test Conference, Mar. 1997, pp 77-81.

4. Dimitrios Kagaris ,Spyros Tragoudas, Amitava Majumdar – *'Deterministic Test Pattern Reproduction by a Counter'* ,Mar.1996 pp 37-41.

ΙΣΤΟΣΕΛΙΔΕΣ

5. **Wolfram Mathworld- The web's most extensive mathematic resource** *[14/07/2010]*
http://mathworld.wolfram.com/CellularAutomaton.html

6. **Synopsis** *[20/07/10]*
http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Documents/Tetramax_ds.pdf

7. **Mentor Graphics- The EDA technology Leader** *[18/06/10]*

   http://www.mentor.com/products/silicon-yield/products/upload/logicbist-ds.pdf

8. **Siliconfareast – One of the mostcomprehensive online references of semiconductor manufacturing** *[22/02/10]*

   http://www.siliconfareast.com/atpg.htm

9. **Siliconfareast – One of the mostcomprehensive online references of semiconductor manufacturing** *[22/02/10]*

   http://www.siliconfareast.com/bist.htm

10. **Portal** *[12/04/10]*
http://portal.acm.org/results.cfm?coll=GUIDE&dl=GUIDE&CFID=103343645&CFTOKEN=58684778

ΠΑΡΑΡΤΗΜΑ Α

ΚΩΔΙΚΑΣ

```
#include <stdio.h>

#include <time.h>

#include <stdlib.h>


int Cellular255 (int K[][130])

{

int i=0,j=0;


    for (i=0; i<256; i++){

    for(j=0; j<130; j++) {

            K[i][j]=0;

            if (i==0 && j==129)

                K[i][j]=1;

                        }//for

                        }//for


    for (i=0; i<256; i++)   {

        for(j=1; j<130; j++){

            if(K[i][j-1]==1 && K[i][j]==1 && K[i][j+1]==1)

K[i+1][j]=0;)

            else if(K[i][j-1]==1 && K[i][j]==1 && K[i][j+1]==0)

K[i+1][j]=1;

            else if (K[i][j-1]==1 && K[i][j+1]==1 && K[i][j]==0)

K[i+1][j]=1;

            else if (K[i][j-1]==1 && K[i][j]==0 && K[i][j+1]==0)

K[i+1][j]=0;
```

```
            else if (K[i][j-1]==0 && K[i][j]==1 && K[i][j+1]==1)
K[i+1][j]=0;
            else if (K[i][j-1]==0 && K[i][j+1]==0 && K[i][j]==1)
K[i+1][j]=1;
            else if (K[i][j-1]==0 && K[i][j]==0 && K[i][j+1]==1)
K[i+1][j]=1;
            else
                K[i+1][j]=0; }
}



    for (i=0; i<256; i++)   {
        for(j=0; j<130; j++){

            printf("%d",K[i][j]);
                            } //for
            printf("\n");
                            } //for
            printf("\n");


            system("PAUSE");
}
int CollapsedCellular(int K[][130],int CP[][9]){

int k=0,l=0,p=0,i;


     for(k=0; k<256; k++) {
        for(l=0; l<9; l++){
            CP[k][l]=0;
                        }
                        }
    for(p=129; p>-1; p--){
```

```
if(p==129-128)   {
    for (i=0; i<256; i++)
    {
       CP[i][0]=K[i][p];
    }
                }
if(p==129-64)
{
    for (i=0; i<256; i++)
    {
        CP[i][1]=K[i][p];
    }                    }

if(p==129-32)
{
    for (i=0; i<256; i++)
    {
        CP[i][2]=K[i][p];
    }
                         }

if(p==129-16)
{
    for (i=0; i<256; i++)
    {
        CP[i][3]=K[i][p];
    }
                          }

else if(p==129-8)
{
```

```
            for (i=0; i<256; i++)
            {
                CP[i][4]=K[i][p];
            }
                                    }

        else if(p==129-4)
        {
            for (i=0; i<256; i++)
            {
                CP[i][5]=K[i][p];
            }
                                    }

        else if(p==129-2)
        {
            for (i=0; i<256; i++)
            {
               CP[i][6]=K[i][p];
            }
                                    }

        else if(p==129-1)
        {
            for (i=0; i<256; i++)
            {
               CP[i][7]=K[i][p];
            }
                                    }
                        }

    for(i=0; i<256; i++){
```

```
        CP[i][8]=-i;}


    for(k=0; k<256; k++)   {
        for(l=0; l<9; l++)
                            {
        printf("%d",CP[k][l]);
                            }
        printf("\n");
                            }


        printf("\n");
        printf("\n");


        system("PAUSE");
}
int MatrixA(int K[][130],int CP[][9]){


int ch,ck,ARITHMOS,cp;
int arithmos=0,i,temp,j,a,b,max2=-1,r;


FILE *fp = fopen("MatrixA.txt", "r");
if (fp == NULL) printf("****File not found\n****");


while(ch!=500) {
    fscanf(fp,"%d",&ch);
    if(ch==500)
    break;
    else
    arithmos=arithmos+1;
                }
fclose(fp);
int A[arithmos];
```

```
FILE *fp2 = fopen("MatrixA.txt", "r");
if (fp == NULL) printf("****File not found\n****");


i=0;
while(ck!=500){
    fscanf(fp2,"%d",&ck);
    if(ck==500)
    break;
    A[i]=ck;
    i=i+1;
              }
printf("%d\n\n\n\n");
fclose(fp2);


int doMore=1;
while(doMore==1)                {
       doMore=0;
       for(i=0;i<arithmos;i++)  {
          if(A[i]>A[i+1])       {
          temp=A[i+1];
          A[i+1]=A[i];
          A[i]=temp;
          doMore=1;             }
                                }
                                }
int B[arithmos];
for(i=0;i<arithmos;i++){
         B[i]=-1;
                        }


j=0;
for(i=0;i<arithmos;i++){
```

```
    if(A[i]!=A[i+1])   {
    B[j]=A[i];
    j=j+1;             }
                       }


int numbers=0;
for(i=0;i<arithmos;i++){
   if(B[i]!=-1)
   numbers=numbers+1;
                       }
j=0;
for(i=0;i<arithmos;i++){
    if(A[i]!=A[i+1])   {
    B[j]=A[i];
    j=j+1;             }
                       }


int numbers=0;
for(i=0;i<arithmos;i++){
   if(B[i]!=-1)
   numbers=numbers+1;
                       }
int C[numbers];
for(i=0;i<numbers;i++){
       C[i]=B[i];
                      }
for(i=0;i<numbers;i++){
printf("%d\n\n",C[i]);
                      }
int MaxBigMatrix,MinBigMatrix,TestVectorNumberBigMatrix,max;


MaxBigMatrix=C[numbers-1];
```

```
MinBigMatrix=C[0];

int counter;


 if(MaxBigMatrix>127 && MaxBigMatrix<256)

        counter=8;

    else if(MaxBigMatrix>63 && MaxBigMatrix<128)

        counter=7;

    else if(MaxBigMatrix>31 && MaxBigMatrix<64)

        counter=6;

    else if(MaxBigMatrix>15 && MaxBigMatrix<32)

        counter=5;

    else if(MaxBigMatrix>7 && MaxBigMatrix<16)

        counter=4;

    else if(MaxBigMatrix>3 && MaxBigMatrix<8)

        counter=3;

    else if(MaxBigMatrix>1 && MaxBigMatrix<3)

        counter=2;

    else if(MaxBigMatrix==1 || MaxBigMatrix==0)

        counter=1;

int BigMatrix[numbers][counter];

for(a=0; a<numbers; a++)     {

    for(b=0; b<counter; b++) {

            BigMatrix[a][b]=0;

                            }

            printf("\n");

                            }

for(a=0; a<numbers; a++)                            {

        for(b=0; b<256; b++)                        {

            if(-CP[b][8]==C[a])

                for(i=7; i>7-counter; i--)

                                                    {

                    if(counter==8)
```

```
                        BigMatrix[a][i]=CP[b][i];
                    else if(counter==7)
                        BigMatrix[a][i-1]=CP[b][i];
                    else if(counter==6)
                        BigMatrix[a][i-2]=CP[b][i];
                    else if(counter==5)
                      BigMatrix[a][i-3]=CP[b][i];
                    else if(counter==4)
                        BigMatrix[a][i-4]=CP[b][i];
                    else if(counter==3)
                        BigMatrix[a][i-5]=CP[b][i];
                    else if(counter==2)
                       BigMatrix[a][i-6]=CP[b][i];
                    else if(counter==1)
                        BigMatrix[a][i-7]=CP[b][i];
                                                }
                                                }
                                                }
for(a=0; a<numbers; a++)      {
    for(b=0; b<counter; b++)  {
        printf("%d ",BigMatrix[a][b]);
                              }
        printf("\n");
                              }
FILE *fp3=fopen("KANONES.txt", "r");
if (fp3 == NULL) printf("****File not found\n****");
for(i=0;i<counter;i++)          {
    for(j=1;j=10000;j++)        {
    fscanf(fp,"%d",&ARITHMOS);
    if(ARITHMOS==-1000)
    break;
    if(max2<abs(ARITHMOS))
```

```
    max2=abs(ARITHMOS);
                                    }
                                    }
fclose(fp3);
int SmallMatrix[numbers][max2];
for(i=0;i<numbers;i++)    {
    for(j=1;j<max2+1;j++){
    SmallMatrix[i][j]=0;
                            }
    printf("\n");
                            }
    printf("\n\n\n");
FILE *fp4 = fopen("KANONES.txt", "r");
if (fp4 == NULL) printf("****File not found\n****");


     for(i=0;i<counter;i++)             {
          for(j=0;j=100;j++)            {
                fscanf(fp,"%d",&cp);
                if(cp==-1000)
                break;
                for(a=0;a<numbers;a++) {
                if(cp<0)
                   if(BigMatrix[a][i]==0)
                   SmallMatrix[a][-cp]=1;
                   else
                   SmallMatrix[a][-cp]=0;
                 else
                 SmallMatrix[a][cp]=BigMatrix[a][i];
                                    }
                                    }
                                    }
    fclose(fp4);
```

```
for(i=0;i<numbers;i++)      {

    for(j=1;j<max2+1;j++)    {

       printf("%d ",SmallMatrix[i][j]);

                              }

       printf("\n");

                              }

       printf("\n\n\n");


int MinSmallMatrix,MaxSmallMatrix;

printf("doste elaxisti kai megisti timi tou SmallMatrix\n\n");

scanf("%d",&MinSmallMatrix);

scanf("%d",&MaxSmallMatrix);

for(i=MinSmallMatrix;i<MaxSmallMatrix+1;i++){

    for(j=0;j<130;j++)     {

     printf("%d",K[i][j]);}

               printf("\n");

                                          }

            system("PAUSE");


int TestVectorNumberSmallMatrix;

TestVectorNumberBigMatrix=MaxBigMatrix-MinBigMatrix+1;


printf("O arithmos ton Test Vectors pou prepei na paraxthoun gia ton pinaka
dokimis einai %d\n\n\n",TestVectorNumberBigMatrix);

TestVectorNumberSmallMatrix=MaxSmallMatrix-MinSmallMatrix+1;


printf("O arithmos ton Test Vectors pou prepei na paraxthoun gia ton
sumpiesmeno pinaka einai %d\n\n\n",TestVectorNumberSmallMatrix);


printf("***********ANAFORA************\n\n\n");


int efficiencyBigMatrix,efficiencySmallMatrix;
```

```
efficiencyBigMatrix=(numbers*100)/TestVectorNumberBigMatrix;

efficiencySmallMatrix=(numbers*100)/TestVectorNumberSmallMatrix;



printf("Gia ton pinaka dokimis prepei na paraxthoun %d dianismata ek ton
opoion          to          %d          tis          ekato          einai
eguro\n\n\n",TestVectorNumberBigMatrix,efficiencyBigMatrix);

printf("Gia ton sympiesmeno pinaka prepei na paraxthoun %d dianismata ek ton
opoion          to          %d          tis          ekato          einai
eguro\n\n\n",TestVectorNumberSmallMatrix,efficiencySmallMatrix);

system("PAUSE");

}

main ()

{

int K[256][130];

int CP[256][9];

Cellular255(K);

CollapsedCellular(K,CP);

MatrixA(K,CP);



system("PAUSE");

}
```