Ανίχνευση σφαλμάτων σε πολυπύρηνα συστήματα με την χρήση λογισμικού

Η Μεταπτυχιακή εργασίας Εξειδίκευσης

υποβάλλεται στην ορισθείσα
από τη Συνέλευση
του Τμήματος Μηχανικών Η/Υ και Πληροφορικής
Εξεταστική Επιτροπή

από τον

Παπαϊωάννου Χριστόφορο

ως μέρος των υποχρεώσεων για την απόκτηση του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗ
ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ
ΣΤΙΣ ΤΕΧΝΟΛΟΓΙΕΣ-ΕΦΑΡΜΟΓΕΣ

Πανεπιστήμιο Ιωαννίνων
Ιούλιος 2019

Εξεταστική Επιτροπή:

• **Χρυσοβαλάντης Καβουσιανός**, Αναπληρωτής Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής (Επιβλέπων),

• **Γεώργιος Τσιατούχας**, Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων

• **Αριστείδης Ευθυμίου**, Επίκουρος Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων

ΑΦΙΕΡΩΣΗ

Αφιερώνω αυτή την εργασία στους γονείς μου Αποστόλη και Κωνσταντινιά, στα αδέρφια μου Αλέξη και Ιωάννη και στην συντροφό μου Άρτεμις, για την υποστήριξη και τη βοήθειά τους όλα αυτά τα χρόνια.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω ιδιαιτέρως τον επιβλέποντα καθηγητή μου Κ. Καβουσιανό Χρυσοβαλάντη, για τη βοήθειά, την καθοδήγηση, την ενθάρρυνση και τις πολύτιμες συμβουλές και γνώσεις που μου προσέφερε κατά τη διάρκεια της εκπόνησης της παρούσας μεταπτυχιακής εργασίας, καθώς και την ευκαιρία που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα. Επιπλέον θα ήθελα να ευχαριστήσω την ερευνητική ομάδα του Πολυτεχνείου του Τορίνο, Ιταλίας για την παροχή της αρχικής έκδοσης του προσομοιωτή και τη συνεχή στήριξη σε οποιαδήποτε δυσκολία. Ειδικότερα θα ήθελα να ευχαριστήσω τον Κ. Squillero Giovanni και τον Κ. Matteo Sonza Reorda για την βοήθεια που μου προσφέρανε.

Τελευταίοι και πιο σημαντικοί άνθρωποι, θα ήθελα να ευχαριστήσω, που με στήριξαν και με στηρίζουν όλα αυτά τα χρόνια και χωρίς αυτούς δεν θα είχα πετύχει τίποτα από όσα έχω κάνει, είναι οι γονείς μου και τα αδέρφια μου. Επίσης, ένα μεγάλο ευχαριστώ στην σύντροφό μου την Άρτεμις για την υπομονή και την συνεχή στήριξη στην προσπάθειά μου για την ολοκλήρωση της εργασίας αυτής.

ΠΕΡΙΕΧΟΜΕΝΑ

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Περιληψη

Παπαϊωάννου Χριστόφορος, Μ.Δ.Ε. στην Πληροφορική, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούνιος 2019.

Ανίχνευση σφαλμάτων σε πολυπύρηνα συστήματα με τη χρήση λογισμικού.

Επιβλέπων: Χρυσοβαλάντης Καβουσιανός, Αναπληρωτής Καθηγητής.

Τα πολυπύρηνα Συστήματα-σε-Ολοκληρωμένα (multicore System-On-Chip – SoCs) αποτελούν σήμερα την κυρίαρχη επιλογή σχεδίασης ηλεκτρονικών συστημάτων σε ημιαγωγούς. Την ευρεία διάδοση τους ευνόησε η υψηλή πυκνότητα ολοκλήρωσης που προσέφεραν οι βαθιά υπόμικρονικές τεχνολογίες καθώς και η δραστική μείωση του χρόνου σχεδίασης και του κόστους κατασκευής. Τα πολυπύρηνα SoCs προσφέρουν υψηλή απόδοση και αναπτύχθηκαν ραγδαία λόγω του «νόμου του Moore»[1], ο οποίος όμως δέχεται πλέον υψηλές πιέσεις λόγω των φυσικών ορίων, κάνοντας εξαιρετικά δύσκολη την περαιτέρω αύξηση της πυκνότητας ολοκλήρωσης. Για παράδειγμα, σε τεχνολογίες κάτω των 32 nm τα ρεύματα διαρροής είναι τόσο αυξημένα για υψηλές τάσεις κατωφλίου, που θέτουν πλέον απαγορευτικό το κόστος διαχείρισης της κατανάλωσης ισχύος και της θερμοκρασίας στα SoCs .

Πέρα όμως από τα σημαντικά οφέλη, η υψηλή πολυπλοκότητα των πολυπύρηνων SoCs έχει εισαγάγει νέες προκλήσεις στην σχεδίαση, κατασκευή αλλά και την ανίχνευση ελαττωματικών ολοκληρωμένων συστημάτων. Ειδικότερα στον έλεγχο ορθής λειτουργίας των SoCs σημαντικές προκλήσεις αποτελούν η ενσωμάτωση πολλών πυρήνων στο ίδιο ολοκληρωμένο, η περιορισμένη πρόσβαση μέσω ενός μικρού αριθμού εξωτερικών υποδοχών (pins), η δυσκολία πρόσβασης στους εσωτερικούς κόμβους και ο υψηλός όγκος δεδομένων ελέγχου που πρέπει να μεταφερθούν στους ενσωματωμένους πυρήνες. Στην ιδιαίτερη περίπτωση των πολυπύρηνων επεξεργαστών SoCs (processor-based) ο έλεγχος πρέπει επιπλέον να ακολουθεί αυστηρούς περιορισμούς σχεδίασης, ενώ

οι επεξεργαστικοί πυρήνες (CPUs) πρέπει να ελέγχονται στον κανονικό τρόπο λειτουργίας (mission-mode) με χρήση μη παρεμβατικών μεθόδων ελέγχου. Επομένως, οι λύσεις σχεδιασμού για δοκιμαστικότητα (Design-For-Testability – DFT), όπως οι αλυσίδες ολίσθησης ή οι τεχνικές ενσωματωμένου αυτοελέγχου συμπληρώνονται από λύσεις λειτουργικού ελέγχου (functional testing), όπως ο αυτοέλεγχος βασισμένος σε λογισμικό (Software-Based-Self-Test ή SBST).

Το SBST είναι μία τεχνική κατά την οποία ο έλεγχος του υλικού του επεξεργαστικού πυρήνα διενεργείται από προγράμματα ελέγχου τα οποία ενεργοποιούν πιθανά σφάλματα στον επεξεργαστικό πυρήνα και τα διαδίδουν σε σημεία παρατήρησης, όπως η ενσωματωμένη μνήμη ή/και η εξωτερική μνήμη. Η βασική ιδέα του SBST είναι η χρήση του συνόλου εντολών (Instruction Set Architecture, ISA) ενός επεξεργαστή για την ενεργοποίηση σφαλμάτων και την μεταφορά των προκαλούμενων λαθών σε σημεία παρατήρησης όπως οι καταχωρητές και τελικώς η μνήμη. Κατά το SBST ο μικροεπεξεργαστής εκτελεί ένα σύνολο ρουτινών, οι οποίες εφαρμόζουν στις επιμέρους μονάδες του ακολουθίες διανυσμάτων ελέγχου. Στην συνέχεια, αποθηκεύει τις αποκρίσεις του ελέγχου στη κύρια μνήμη τις οποίες συγκρίνει με τις αναμενόμενες αποκρίσεις για την ανίχνευση των ελαττωμάτων.

Ένα σημαντικό πρόβλημα στην περίπτωση του ελέγχου με την τεχνική SBST είναι η έλλειψη των κατάλληλων εργαλείων εξομοίωσης σφαλμάτων. Ενώ για τον δομικό έλεγχο έχουν αναπτυχθεί αρκετά εργαλεία εξομοίωσης σφαλμάτων και παραγωγής διανυσμάτων ελέγχου, για τον έλεγχο με την βοήθεια λογισμικού δεν υπάρχουν ακόμη τα κατάλληλα εργαλεία. Τα οφέλη της αυτοδοκιμής με λογισμικό είναι πολλαπλά. Καταρχήν, επιτρέπει τη δοκιμή στην πραγματική συχνότητα λειτουργίας με τη χρησιμοποίηση οποιουδήποτε εξωτερικού ελεγκτή χαμηλού κόστους. Επιπλέον, το λογισμικό αυτοδοκιμής “ταιριάζει” περισσότερο για αυτοδοκιμή στο πεδίο λειτουργίας σε σχέση με την κλασική αυτοδοκιμή με ειδικό πρόσθετο υλικό που απαιτεί να τεθεί ο μικροεπεξεργαστής σε ειδική κατάσταση δοκιμής. Τέλος, αν κατά τη διάρκεια της λειτουργίας του συστήματος παραστεί η ανάγκη για αλλαγή, ή βελτίωση του συνόλου των διανυσμάτων δοκιμής για παράδειγμα για ένα άλλο μοντέλο ελαττωμάτων, το λογισμικό αυτοδοκιμής μπορεί εύκολα να προσαρμοστεί, σε αντίθεση με το υλικό.

Ο στόχος της μεταπτυχιακής εργασίας είναι η δημιουργία κατάλληλου εξομοιωτή ο οποίος θα επιτρέψει την ανάπτυξη “έξυπνων” μεθόδων SBST για τον έλεγχο πολυπύρηνων SoCs. Στα πλαίσια της εργασίας αυτής θα αναπτυχθεί και θα

χρησιμοποιηθεί ένας εξομοιωτής σφαλμάτων ειδικά για εφαρμογές SBST που θα υποστηρίζει διάφορα μοντέλα σφαλμάτων χρησιμοποιώντας καινοτόμες τεχνικές απόρριψης σφαλμάτων (fault-dropping). Για την αποτελεσματικότερη ανίχνευση των σφαλμάτων χρησιμοποιήθηκαν αρκετά διαφορετικά προγράμματα εντολών στοχεύοντας έτσι στην ανίχνευση όσο γίνεται δυνατόν περισσότερων ομάδων σφαλμάτων.

Λέξεις κλειδιά:
SoCs, SBST, DFT

Extended Abstract

Papaioannou Christoforos, M.Sc. in Computer Science, Department of Computer Science and Engineering, University of Ioannina, Greece, June 2019.
Fault detection on multi-core systems using software.
Advisor: Xrysovalantis Kavousianos, Associate Professor.

In a competitive semiconductor market, multicore SoCs constitute the dominant design style for high performance and short time-to-market. Multicore SoCs exploit the high integration density offered by the very deep sub-micron technologies in order to increase the performance by integrating on the same die a large variety of homogeneous and heterogeneous cores, and they drastically shorten both the design cycle and the development cost. Multiplex SoCs offer high performance and have developed rapidly due to Moore's law [1], which however now accepts high pressures due to physical limits, making it extremely difficult to further increase the density of integration. For example, in technologies below 32 nm, leakage currents are so elevated for high thresholds, which now prohibit the cost of managing power consumption and temperature in SoCs.

Despite the significant benefits, the high complexity of the SoCs multi-core systems has introduced new challenges in the design, construction and detection of defective integrated systems. Especially in the fault-free operation of SOCs, major challenges are the integration of multiple cores into the same IC, the limited access through a small number of external sockets, the difficulty of accessing internal nodes and the large amount of control data to be transferred to the embedded kernels. In the case of multi-core processors SoCs (processor-based), control must follow strictly restrictive designs, while processor cores (CPUs) must be tested in normal mode (mission mode) using non-invasive control methods. Therefore, DFT, such as sliding chains or built-in

stand-alone techniques, are complemented by functional-testing solutions such as Software-Based Self-Test or SBST.

SBST is a technique in which the testing of the processing core is performed by special test programs that trigger potential errors in the processing kernel and propagate to observation points such as built-in memory and / or external memory. The basic idea of SBST is to use a processor's Instruction Set Architecture (ISA) to trigger errors and transfer the resulting errors to observation points such as registers and ultimately the memory. During SBST, the microprocessor executes a set of routines that apply control vector sequences to all individual units. It then stores the control responses in the master memory which compares with the expected responses to detect the defects.

A major problem with SBST is the lack of proper CAD tools. While for building control several fault-simulation and control vector generation tools have been developed, software-based self-testing is not yet supported by the right tools. The benefits of SBST are many. SBST allows testing the actual operating frequency by using any external low cost controller. In addition, the self-test software is suitable for in-field-testing. Finally, if during the operation of the system there is a need for change, or improvement of the testing process (e.g. for a different defect model), the self-test software can be easily adjusted, while this is not true for the hardware.

The main objective of the master thesis is to develop a software-based fault simulator that will advanced methods for testing multicore SoCs. In this work, an SBST-simulator will be developed and used to support various error models using innovative fault-dropping techniques. Several different command programs were used to more effectively detect errors. In this way, it aims in detecting as many groups of errors as possible.

Keywords:
SoCs, SBST, DFT

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

1.1 Κίνητρα

Η κατασκευή των ολοκληρωμένων κυκλωμάτων εξελίσσεται με ταχείς ρυθμούς, ιδίως ως προς το μέγεθος των τρανζίστορ, που χρησιμοποιούνται. Με τα μεγέθη των τρανζίστορ να έχουν υποχωρήσει ακόμη και κάτω από τα 10 nm, εύκολα μπορεί να γίνει αντιληπτό, ότι και οι μέθοδοι και οι τεχνικές ανίχνευσης σφαλμάτων σε τέτοιες τεχνολογίες απαιτούν ιδιαίτερα περίπλοκες και απαιτητικές διαδικασίες. Παραδοσιακός στόχος των διαδικασιών ελέγχου είναι η αποφυγή της αποδέσμευσης ελαττωματικών προϊόντων στην αγορά.

Η σπουδαιότητα του ελέγχου αναδεικνύεται και από τον προϋπολογισμό που διαθέτουν οι εταιρείες ολοκληρωμένων κυκλωμάτων για αυτόν τον σκοπό. Είναι χαρακτηριστικό το γεγονός ότι το 30% έως 70% του συνολικού προϋπολογισμού της σχεδίασης και κατασκευής ενός σύγχρονου μικροεπεξεργαστή δαπανάται για τον έλεγχο ορθής λειτουργίας [2],[3]. Επειδή το κόστος δοκιμής είναι ιδιαίτερα υψηλό, στη βιομηχανική μικροηλεκτρονική είναι ευρέως αποδεκτό ότι τα ολοκληρωμένα κυκλώματα επιβάλλεται να δοκιμαστούν πριν τοποθετηθούν στις πλακέτες τυπωμένων κυκλωμάτων (Printed Circuit Boards, PCBs). Στη συνέχεια τα PCBs πρέπει να δοκιμαστούν προτού ενσωματωθούν στα τελικά συστήματα. Για το κόστος δοκιμής των προϊόντων μικροηλεκτρονικής ισχύει ο "Κανόνας του Δέκα" (Rule of Ten) [4]. Εάν ένα ελάττωμα σε ένα ολοκληρωμένο κύκλωμα δεν βρεθεί κατά την κατασκευή του τότε το κόστος ανίχνευσης του ελαττώματος είναι 10 φορές μεγαλύτερο στο επίπεδο των καρτών τυπωμένου κυκλώματος από ότι στο επίπεδο του ολοκληρωμένου κυκλώματος. Ομοίως, εάν ένα ελάττωμα δεν βρεθεί κατά τη δοκιμή στο επίπεδο των καρτών τυπωμένου

κυκλώματος, η ανίχνευσή του έχει 10 φορές μεγαλύτερο κόστος στο επίπεδο του συστήματος από ότι στο επίπεδο των τυπωμένων καρτών [5]. Μερικοί υποστηρίζουν ότι ο Κανόνας των Δέκα πρέπει να μετονομαστεί σε Κανόνα των Είκοσι, καθώς σήμερα τα ολοκληρωμένα κυκλώματα, οι πλακέτες τυπωμένων κυκλωμάτων και τα συστήματα είναι πολύ πιο περίπλοκα από ότι ήταν όταν δηλώθηκε αρχικά αυτός ο εμπειρικός κανόνας [4]. Συνεπώς, όσο νωρίτερα ανιχνεύεται ένα ελάττωμα, τόσο μικρότερο είναι το κόστος επιδιόρθωσής του και όσο αποδοτικότερος είναι ο έλεγχος, τόσο ποιοτικότερο είναι το προϊόν στον τελικό χρήστη.

Στο τελικό στάδιο δημιουργίας της συσκευής εκτός από τον παραδοσιακό κατασκευαστικό (structural) τρόπο ελέγχου, η συσκευή δοκιμάζεται και μέσα στο σύστημα για να αξιολογηθεί η συμπεριφορά της σε συγκεκριμένες απαιτήσεις. Το στάδιο αυτό ελέγχου ονομάζεται επίπεδο ελέγχου συστήματος (System Level Testing - SLT), στο οποίο η συσκευή υπόκεινται σε μια σειρά από διάφορες τελικές δοκιμές για τον έλεγχο της ορθή λειτουργίας της. Οι έλεγχοι που πραγματοποιούνται αφορούν ουσιαστικά την εκτέλεση εφαρμογών συστήματος με χρήση και των εξωτερικών περιφερειακών του επεξεργαστή, ελέγχοντας την λειτουργία μεταξύ τους και με το σύστημα σαν σύνολο.

1.2 Στόχος

Στην εργασία αυτή θα υλοποιηθεί ένας εξομοιωτής σφαλμάτων για δοκιμή συστημάτων με χρήση λογισμικού (SBST). Βασικός στόχος του εξομοιωτή είναι η μείωση του χρόνου ελέγχου, και η βελτιστοποίηση της εξομοίωσης για τις συγκεκριμένες συνθήκες που εκτελείται ο έλεγχος SBST. Επιπλέον θα στοχοποιηθούν μοντέλα σφαλμάτων μετάβασης και βραχυκυκλώματος. Απώτερος στόχος είναι η ανάπτυξη αποδοτικών μεθόδων ελέγχου των επεξεργαστών με χρήση λογισμικού.

Ο βασικός στόχος της εργασίας είναι η υλοποίηση ενός εξομοιωτή με δυνατότητα ανίχνευσης διαφόρων σφαλμάτων, συγκεκριμένα, σφαλμάτων μόνιμης τιμής (stuck at faults), σφαλμάτων μετάβασης (transition faults) και σφαλμάτων γεφύρωσης (bridging faults). Ως «Μελέτη Περίπτωσης» στον εξομοιωτή χρησιμοποιήθηκε ο επεξεργαστής i8051. Για την επίτευξή των αλγορίθμων ανίχνευσης σφαλμάτων, χρησιμοποιήθηκαν αρχιτεκτονικές εντολές του επεξεργαστή, που έχουν ως στόχο την μεταφορά δεδομένων εσωτερικά του και την ανάλυση των αποκρίσεών του, για την εξαγωγή συμπερασμάτων. Τα συμπεράσματα για τις υλοποιήσεις των προγραμμάτων ελέγχου, που έγιναν και πιθανές επεκτάσεις και βελτιώσεις τους παραθέτονται στο τέλος της εργασίας.

ΚΕΦΑΛΑΙΟ 2

ΥΠΟΒΑΘΡΟ

2.1 Έλεγχος ορθής λειτουργίας

Κατά τη κατασκευή των ηλεκτρονικών κυκλωμάτων παρουσιάζονται ελαττώματα (manufacturing process defects) που μπορεί να επηρεάσουν την λειτουργία τους κατά τη διάρκεια της χρήσης τους. Για να καθοριστεί αν μια συσκευή έχει κατασκευαστεί σωστά ή ότι συνεχίζει να λειτουργεί σύμφωνα με τον επιθυμητό τρόπο πρέπει να περάσει από μια διαδικασία επαλήθευσης που ονομάζεται έλεγχος ορθής λειτουργίας.

Η βασική μεθοδολογία του ελέγχου ορθής λειτουργίας είναι η εφαρμογή συγκεκριμένων τιμών στις εισόδους του κυκλώματος και η παρακολούθηση των αποκρίσεων στις εξόδους του. Αν οι αποκρίσεις του κυκλώματος είναι οι αναμενόμενες τότε το κύκλωμα λειτουργεί σωστά, αλλιώς είναι ελαττωματικό. Τα διανύσματα εισόδου που δίνονται ως είσοδοι στο κύκλωμα ονομάζονται διανύσματα δοκιμής.

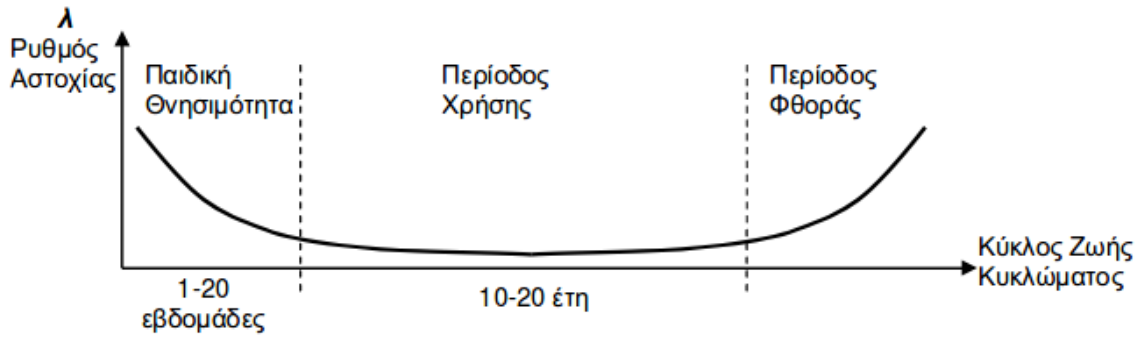
Η ανάπτυξη του ελέγχου ορθής λειτουργίας για κάθε κύκλωμα επιμερίζεται στις ακόλουθες διαδικασίες:

- παραγωγή των διανυσμάτων δοκιμής (test pattern generation),
- εξαγωγή της αναμενόμενης απόκρισης του κυκλώματος μέσω λογικής εξομοίωσης (logic simulation) και
- εκτίμηση της αποδοτικότητας των διανυσμάτων δοκιμής με βάση το χρησιμοποιούμενο μοντέλο σφαλμάτων μέσω της εξομοίωσης σφαλμάτων (fault simulation)

2.2 Είδη σφαλμάτων

Ελαττώματα/Σφάλματα/Λάθη

Τα ελαττώματα (defects) αφορούν δομικές και παραμετρικές αποκλίσεις στην κατασκευή ενός κυκλώματος. Τέτοια ελαττώματα μπορεί να προκύψουν είτε κατά την παραγωγική διαδικασία λόγω κάποιων προβλημάτων στην παραγωγή, είτε κατά την συσκευασία του ολοκληρωμένου κυκλώματος, είτε τυχαία λόγω μεταβολής των παραμέτρων του κυκλώματος (υγρασία, θερμοκρασία). Ως ελάττωμα (defect) ορίζουμε την ανεπιθύμητη διαφορά που προκύπτει μεταξύ του σχεδιασμένου και του κατασκευασμενου κυκλώματος[2]. Τα ελαττώματα είτε μπορεί να οφείλονται στην κατασκευαστική διαδικασία (fabrication defect), όπως τα βραχυκυκλώματα, οι ανοιχτές επαφές και οι μετατοπίσεις μασκών, είτε μπορεί να οφείλονται στην τυχαία διακύμανση των παραμέτρων κατά τη διαδικασία της κατασκευής (process defects) ή και σε ελαττωματικά υλικά κατασκευής (material defects). Επιπλέον ελαττώματα μπορεί να προκληθούν και κατά τη διαδικασία τοποθέτησης του ολοκληρωμένου κυκλώματος στο περίβλημά του (package defects). Τέλος, τα ελαττώματα προκύπτουν σε τυχαίες χρονικές στιγμές καθ'όλη τη διάρκεια της ζωής του κυκλώματος λόγω γήρανσης (aging defects) ή εξαιτίας κάποιας φυσικής αιτίας. Αξίζει να σημειωθεί ότι όταν το ολοκληρωμένο εισέρχεται στην τελευταία περίοδο της ζωής του, ο ρυθμός εμφάνισης των σφαλμάτων αυτών αυξάνεται (Σχήμα 2.1)



Σχήμα 2.1 - Η διάρκεια ζωής των κυκλωμάτων

Το σφάλμα είναι η μαθηματική μοντελοποίηση ενός ελαττώματος στη συμπεριφορά του κυκλώματος. Φυσικά είναι αδύνατον να μοντελοποιήσουμε κάθε ελάττωμα με το δικό του σφάλμα, συνεπώς ομαδοποιούμε τα πιθανά ελαττώματα σε

μοντέλα σφαλμάτων ώστε η διαχείριση τους να είναι πιο εύκολη υπολογιστικά. Ένα μοντέλο σφαλμάτων μπορεί να αναπαριστά έναν αριθμό από ελαττώματα, όμως και κάποιο ελάττωμα μπορεί να αναπαρίσταται από διάφορα μοντέλα σφαλμάτων. Ο πίνακας 2.1 περιέχει τα πιο συχνά χρησιμοποιούμενα μοντέλα σφαλμάτων.

Μοντέλα Σφάλματος	Περιγραφή
Απλά σφάλματα μόνιμης τιμής (single stuck at faults)	Μια γραμμή λαμβάνει συνεχώς τιμή είτε το ‘0’ ή το ‘1’
Πολλαπλά σφάλματα μόνιμης τιμής (multiple stuck at faults)	Δύο ή περισσότερες γραμμές έχουν μια σταθερή τιμή (‘0’ ή ‘1’) όχι όμως απαραίτητα την ίδια
Βραχυκύκλωμα (Bridging faults)	Δύο ή περισσότερες γραμμές που θα πρέπει να είναι ανεξάρτητες, συνδέονται μεταξύ τους.
Σφάλμα ανοιχτού τρανζίστορ (stuck open faults)	Ένα pull-up ή pull-down τρανζιστορ σε λογική CMOS δεν άγει ποτέ με αποτέλεσμα η λογική να συμπεριφέρεται ως στοιχείο μνήμης
Σφάλμα μόνιμα άγοντος τρανζίστορ (stuck-on-faults)	Ένα τρανζίστορ άγει μόνιμα
Σφάλμα καθυστέρησης διάδοσης (delay faults)	Καθυστέρηση διάδοσης του σήματος σε ένα ή περισσότερα μονοπάτια του κυκλώματος
Διαλείποντα σφάλματα (intermittent faults)	Προκαλούνται από εσωτερική φθορά του κυκλώματος. Λάθος αποκρίσεις συμβαίνουν σε κάποιες καταστάσεις του κυκλώματος. Η φθορά είναι προοδευτική μέχρι τη μόνιμη βλάβη του κυκλώματος.
Παροδικά σφάλματα (transient faults)	Λάθος αποκρίσεις από εξωγενείς παράγοντες όπως θόρυβος, διακύμανση στην τροφοδοσία, κοσμική ακτινοβολία

Πίνακας 2.1 - Μοντέλα σφαλμάτων

Τέλος, το άμεσο αποτέλεσμα των ελαττωμάτων στην λειτουργία των ολοκληρωμένων ονομάζεται λάθος (error) και εμφανίζεται με τη μορφή λανθασμένων τιμών στις εξόδους του κυκλώματος. Τα λάθη αφορούν ουσιαστικά την απόκλιση της λειτουργίας του ελαττωματικού κυκλώματος σε σχέση με το ορθό, όπως για παράδειγμα την λανθασμένη απόκριση σε συγκεκριμένη έξοδο του κυκλώματος.

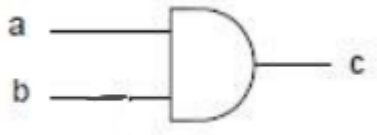
2.2.1 Σφάλματα μόνιμης τιμής

Τα μοντέλα σφαλμάτων αποτελούν την μαθηματική αναπαράσταση των φυσικών ελαττωμάτων. Ένα μοντέλο σφάλματος, όπως και κάθε μοντέλο, δεν είναι απαραίτητο να δίνει μια ακριβή αναπαράσταση των ελαττωμάτων αλλά πρέπει να μπορεί να βοηθάει αποτελεσματικά στην ανίχνευση τους. Ένα από τα πιο συχνά χρησιμοποιούμενα μοντέλα σφαλμάτων είναι το μοντέλο απλού σφάλματος μόνιμης τιμής (single stuck at fault – SSAF), το οποίο συναντάται και με τις ονομασίες κλασσικό (classical) ή καθιερωμένο (standard) μοντέλο [6][7][8].

Ένα απλό σφάλμα μόνιμης τιμής έχει τις εξής ιδιότητες

1. Μόνο μια γραμμή του κυκλώματος παρουσιάζει το σφάλμα
2. Η γραμμή αυτή έχει ως μόνιμη τιμή μία εκ των ‘0’, ‘1’
3. Το σφάλμα μπορεί να παρουσιαστεί μόνο στην είσοδο ή στην έξοδο μιας πύλης και όχι εσωτερικά σε αυτή.

Από εδώ και στο εξής με τον όρο σφάλμα θα αναφερόμαστε αποκλειστικά σε απλό σφάλμα μόνιμης τιμής, εκτός και αν ρητά δηλώνεται κάτι διαφορετικό. Με βάση τα παραπάνω ο αριθμός των πιθανών σφαλμάτων μόνιμης τιμής εξαρτάται από τον αριθμό των γραμμών του κυκλώματος. Αν υπάρχουν n γραμμές στο κύκλωμα, συμπεριλαμβανομένων και των διακλαδώσεων (fan-out branches), τότε ο αριθμός των σφαλμάτων είναι ίσος με 2n. Έστω το κύκλωμα του σχήματος 2.2: μια πύλη AND με δύο εισόδους a, b και μία έξοδο c. Έστω ότι η είσοδος b της πύλης έχει ένα ανεπιθύμητο βραχυκύκλωμα με την γείωση (ελάττωμα). Τότε η είσοδος b φέρεται να είναι μόνιμα στην τιμή 0 (σφάλμα) και η έξοδος του κυκλώματος είναι η c = ‘0’ ακόμη και όταν a = b = ‘1’ (λάθος).



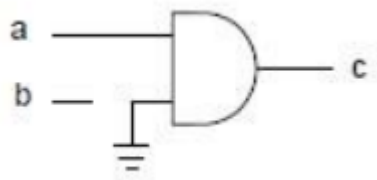
Σχήμα 2.2 - Ορθή λογική πύλη AND

Για αυτό το κύκλωμα παρατηρούμε τα εξής:

Ελάττωμα: Στην είσοδο της πύλης υπάρχει βραχυκύκλωμα με την γείωση

Σφάλμα: Η είσοδος της πύλης έχει μόνιμη τιμή στο ‘0’

Λάθος: Για a = ‘1’ και b = ‘1’ έχουμε c = ‘0’ ενώ θα έπρεπε c = ‘1’. Για οποιονδήποτε άλλο συνδυασμό εισόδων το κύκλωμα λειτουργεί σωστά.

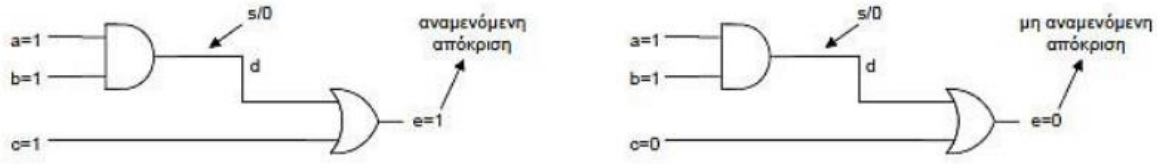


Σχήμα 2.3 - Εσφαλμένη πύλη AND

Όπως φάνηκε από το σχήμα 2.3 η παρουσία ενός σφάλματος δεν είναι πάντα παρατηρήσιμη. Στη συγκεκριμένη περίπτωση αν έστω και μία είσοδος πάρει την τιμή ‘0’ τότε η αναμενόμενη έξοδος του κυκλώματος θα έχει την τιμή ‘0’, η οποία είναι όμοια με την έξοδο του κυκλώματος παρουσία του σφάλματος. Σε αυτή την περίπτωση παρουσιάζεται το φαινόμενο της απόκρυψης σφάλματος (fault masking). Ένα διάνυσμα δοκιμής πρέπει λοιπόν να δημιουργεί τέτοιες συνθήκες στο κύκλωμα ώστε η τιμή μιας υπό εξέταση γραμμής σε συνθήκες ορθής λειτουργίας να είναι διαφορετική από την τιμή που λαμβάνει η γραμμή σε περίπτωση παρουσίας ενός σφάλματος σε αυτή. Πρέπει δηλαδή το διάνυσμα δοκιμής να ενεργοποιεί το σφάλμα (fault activation).

Η ενεργοποίηση όμως ενός σφάλματος από μόνη της δεν είναι αρκετή. Θεωρήστε το κύκλωμα το σχήμα 2.4 όπου η γραμμή d είναι μόνιμα στο ‘0’ (d stuck-at 0, d s/0 ή d/0). Η ενεργοποίηση του σφάλματος απαιτεί a = b = ‘1’, και έστω ότι η είσοδος c έχει την τιμή ‘1’. Με αυτές τις τιμές στις εισόδους η έξοδος e θα λάβει την τιμή ‘1’, που είναι η αναμενόμενη απόκριση του κυκλώματος, ανεξάρτητα της τιμής της γραμμής d, είτε αυτή παρουσιάζει κάποιο σφάλμα είτε όχι, όπως φαίνεται στο αριστερό μέρος του σχήματος 2.4. Αν όμως η είσοδος c έχει την τιμή ‘0’ τότε το σφάλμα στη γραμμή d ανιχνεύεται καθώς η τιμή της εξόδου είναι στο ‘0’ που είναι διαφορετική από την αναμενόμενη απόκριση του κυκλώματος, όπως φαίνεται στο δεξί μέρος του σχήματος

2.4. Θέτοντας λοιπόν την τιμή της γραμμής c στο ‘0’ καταφέρνουμε να διαδώσουμε το αποτέλεσμα της παρουσίας του σφάλματος στην έξοδο του κυκλώματος. Επομένως ένα διάνυσμα εκτός από την ενεργοποίηση ενός σφάλματος πρέπει να παράγει τις κατάλληλες συνθήκες για τη διάδοση του αποτελέσματος της παρουσίας του σφάλματος στις εξόδους του κυκλώματος (fault propagation).



Σχήμα 2.4 - Ενεργοποίηση και διάδοση σφαλμάτος

Ένας όρος που χρησιμοποιείται ευρέως είναι το ποσοστό κάλυψης σφαλμάτων (fault coverage –FC) [9] το οποίο είναι ο αριθμός των σφαλμάτων που ανιχνεύονται από την εφαρμογή ενός συνόλου δοκιμής προς το σύνολο των δυνατών σφαλμάτων που υπάρχουν σε ένα κύκλωμα με βάση το αντίστοιχο μοντέλο σφαλμάτων [10][9]:

$$FC = \frac{\pi\lambda\acute{\eta}\theta o\varsigma\ \alpha\nu\iota\chi\nu\epsilon\acute{\upsilon}\sigma\iota\mu\omega\nu\ \sigma\phi\alpha\lambda\mu\acute{\alpha}\tau\omega\nu}{\pi\lambda\acute{\eta}\theta o\varsigma\ \sigma\phi\alpha\lambda\mu\acute{\alpha}\tau\omega\nu\ \kappa\upsilon\kappa\lambda\acute{\omega}\mu\alpha\tau o\varsigma}$$

Καθώς όμως υπάρχουν σφάλματα που δεν είναι δυνατό να ανιχνευτούν (untestable faults), όπως αυτά που οφείλονται στην ύπαρξη πλεονάζουσας λογικής (redundant logic), το ποσοστό κάλυψης σφαλμάτων υπολογίζεται ως εξής [11]:

$$FC = \frac{\pi\lambda\acute{\eta}\theta o\varsigma\ \alpha\nu\iota\chi\nu\epsilon\upsilon\mu\acute{\epsilon}\nu\omega\nu\ \sigma\phi\alpha\lambda\mu\acute{\alpha}\tau\omega\nu}{\pi\lambda\acute{\eta}\theta o\varsigma\ \alpha\nu\iota\chi\nu\epsilon\acute{\upsilon}\sigma\iota\mu\omega\nu\ \sigma\phi\alpha\lambda\mu\acute{\alpha}\tau\omega\nu}$$

όπου πλήθος ανιχνεύσιμων σφαλμάτων = πλήθος σφαλμάτων – πλήθος μη ανιχνεύσιμων σφαλμάτων.

Το ποσοστό κάλυψης σφαλμάτων εκφράζει την αποδοτικότητα ενός συνόλου δοκιμής και αποτελεί βασικό εργαλείο στις μεθόδους παραγωγής διανυσμάτων που βασίζονται σε εξομοίωση σφαλμάτων. Στο κύκλωμα του σχήματος 2.4 για να δούμε ποια σφάλματα ανιχνεύει το κάθε διάνυσμα πρέπει να κάνουμε εξομοίωση σφαλμάτων για κάθε δυνατό σφάλμα. Στη συγκεκριμένη περίπτωση, το καθένα από τα δύο διανύσματα

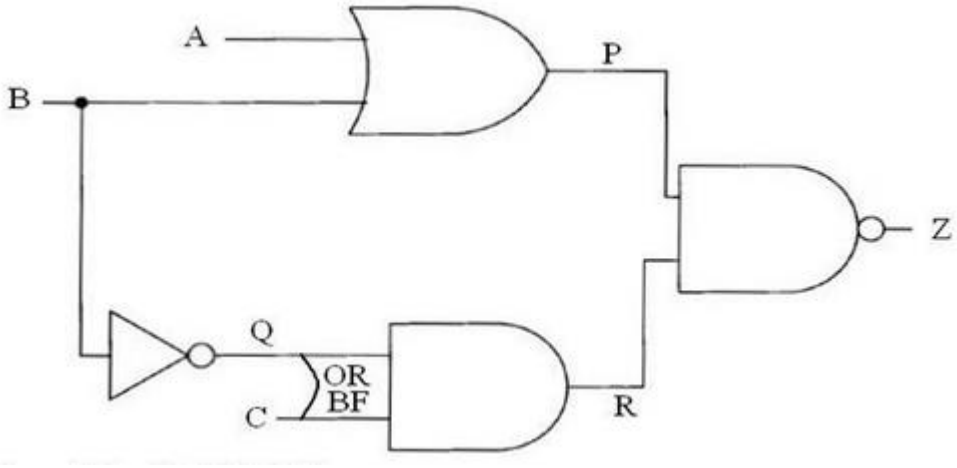
εξομοιώνεται για 10 διαφορετικά σφάλματα και το αποτέλεσμα της κάθε εξομοίωσης συγκρίνεται με την αναμενόμενη απόκριση.

Ο έλεγχος ενός κυκλώματος με τη χρησιμοποίηση ενός συνόλου διανυσμάτων ακόμη και αν αυτό παρέχει ποσοστό κάλυψης σφαλμάτων 100% δεν σημαίνει ότι το υπό δοκιμή κύκλωμα δεν παρουσιάζει κανένα ελάττωμα. Αυτό οφείλεται στο ότι το μοντέλο απλού σφάλματος μόνιμης τιμής, όπως και εν γένει όλα τα μοντέλα, δεν αναπαριστά όλα τα πιθανά ελαττώματα. Αποτελεί όμως ισχυρή ένδειξη ότι το κύκλωμα λειτουργεί σύμφωνα με τις προδιαγραφές που έχει κατασκευαστεί.

2.2.2 Σφάλματα Γεφύρωσης (Bridging faults)

Τα σφάλματα αυτά προκύπτουν όταν δύο ή και περισσότεροι κόμβοι του κυκλώματος είναι βραχυκυκλωμένοι από κατασκευαστικό λάθος. Τα συνηθέστερα σφάλματα γεφύρωσης είναι πολλαπλά, δηλαδή αναφέρονται σε βραχυκύκλωμα περισσότερων των 2 γραμμών ή κόμβων [12] [13] [14] [15].

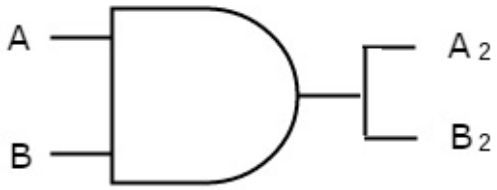
Ο συνδυασμός του μοντέλου μόνιμης τιμής και των μοντέλων σφαλμάτων γεφύρωσης οδηγεί σε ένα μοντέλο που χρησιμοποιείται αρκετά για την ανίχνευση των βραχυκυκλωμάτων. Στην περίπτωση αυτή η σωστή κατάσταση λειτουργίας βρίσκεται είτε με την χρήση της λογικής πράξης OR ή AND, μεταξύ των βραχυκυκλωμένων κόμβων του κυκλώματος. Για παράδειγμα, στο σχήμα 2.5 υπάρχει ένα Or σφάλμα γεφύρωσης μεταξύ των εισόδων C και Q. Η ορθή συνάρτηση του κυκλώματος είναι { (A+B) * (B’ * C) }, ενώ στην περίπτωση που εμφανίζεται το σφάλμα γεφύρωσης η συνάρτηση είναι { (A+ B) * (B’ + C)}’. Για την ανίχνευση του σφάλματος αυτός, το διάνυσμα ελέγχου 100 μπορεί να ανιχνεύσει το σφάλμα, ενώ διανύσματα όπως 101 ή 110 δεν μπορούν να το ανιχνεύσουν.



Σχήμα 2.5 - Σφάλμα γεφύρωσης

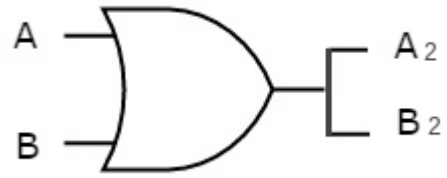
Τα σφάλματα γεφύρωσης χωρίζονται σε τέσσερις κατηγορίες, τα wired-AND και τα wired-OR μοντέλα σφαλμάτων, όπως επίσης και τα A-dominates-B και τα B-dominates-A μοντέλα σφαλμάτων. Στην εργασία αυτή ο εξομοιωτής έχει τη δυνατότητα ανίχνευσης των δύο τελευταίων μοντέλων σφαλμάτων.

Τα wired-AND και τα wired-OR μοντέλα σφαλμάτων είναι βραχυκυκλώματα μεταξύ δύο γραμμών του κυκλώματος, που το αποτέλεσμα από της ένωσής τους είναι ισοδύναμο με μια πύλη AND ή OR αντίστοιχα.



Σχήμα 2.6 - Wired-AND μοντέλα σφαλμάτων

Στο πρώτο σχήμα παρουσιάζεται το wired-AND μοντέλο σφαλμάτων και στην περίπτωση αυτή ανάλογα με τις εισόδους A και B τα αποτελέσματα A_2 και B_2 προκύπτουν από την λογική πράξη AND των εισόδων.



Σχήμα 2.7 - Wired-OR μοντέλα σφαλμάτων

Στην αντίστοιχη περίπτωση το παραπάνω σχήμα παρουσιάζει τα wired-OR μοντέλα σφαλμάτων, με τη μόνη διαφορά που τα A_2 και B_2 προκύπτουν από τη λογική πράξη της OR. Έτσι οι πίνακες αλήθειας που προκύπτουν από την εφαρμογή των δύων μοντέλων είναι η παρακάτω:

A	B	A_2	B_2	Wired-AND		Wired-OR	
0	0	0	0	0	0	0	0
0	1	0	1	0	0	1	1
1	0	1	0	0	0	1	1
1	1	1	1	1	1	1	1

Πίνακας 2.2 - πίνακας αλήθειας για Wired-AND και Wired-OR σφάλματα

Στον πίνακα παρατηρούμε ότι ανάλογα τα δύο μοντέλα προκύπτουν κάποια σφάλματα από την ύπαρξη της AND ή της OR πύλης. Τα σφάλματα εμφανίζονται με την κίτρινη επισήμανση.

Οι επόμενες πιο απλές περιπτώσεις μοντέλων σφαλμάτων βραχυκυκλώματος, που εξομοιώθηκαν στην εργασία αυτή, είναι τα A-dominates-B και B-dominates-A. Στην πρώτη περίπτωση θεωρούμε ότι η τιμή της γραμμής του κυκλώματος A κυριαρχεί στην γραμμή B. Έτσι η τιμής της A επικρατεί και στις δύο γραμμές του κυκλώματος (A και B). Το αντίστοιχο ισχύει και στην δεύτερη περίπτωση, που η τιμή της B γραμμής επικρατεί της γραμμής A. Έτσι οι πίνακες αλήθειας που προκύπτουν είναι οι παρακάτω.

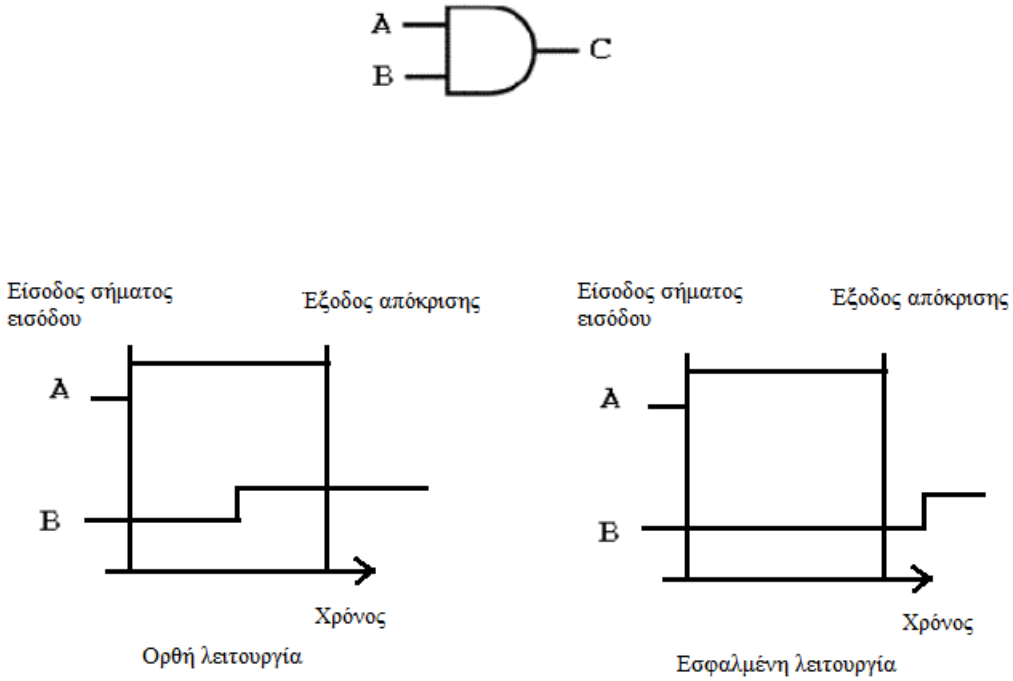
A	B	A_2	B_2	A dom B		B dom A	
0	0	0	0	0	0	0	0
0	1	0	1	0	0	1	1
1	0	1	0	1	1	0	0
1	1	1	1	1	1	1	1

Πίνακας 2.3 - Πίνακας αλήθειας για dominates σφάλματα

Στον πίνακα παρατηρούμε ότι ανάλογα τα δύο μοντέλα προκύπτουν κάποια σφάλματα από την ύπαρξη του A dominates B ή της B dominates A. Τα σφάλματα εμφανίζονται με την κίτρινη επισήμανση.

2.2.3 Σφάλματα μετάβασης (Transition faults)

Στο μοντέλο σφάλματος μετάβασης ένας κόμβος είναι σε θέση να κάνει μια μετάβαση από την τιμή 0 -> 1 αλλά όχι από 1 -> 0 και αντίστροφα. Ένα σφάλμα μετάβασης σε μια γραμμή καθιστά αργή την αλλαγή σήματος στη γραμμή αυτή. Τα δύο πιθανά σφάλματα είναι slow-to-rise και a slow-to-fall. Στην πρώτη περίπτωση, του slow-to-rise, η γραμμή που έχει το σφάλμα αυτό δεν έχει την ικανότητα της μετάβασης από την κατάσταση 0 στην 1. Στην δεύτερη περίπτωση, του slow-to-fall ισχύει το αντίστροφο. Στο παρακάτω παράδειγμα παρουσιάζεται μια περίπτωση slow-to-rise σφάλματος [16][17].



Σχήμα 2.8 - Σφάλματα μετάβασης

Για την ανίχνευση ενός σφάλματος slow-to-rise μπορεί να χρησιμοποιηθεί ένα stuck-at-0 σφάλμα. Συγκεκριμένα τοποθετείται στον κόμβο η λογική τιμή 0, και στον επόμενο κύκλο ρολογιού στοχοποιείται το σφάλμα stuck-at 0. Αν το σφάλμα ανιχνευθεί τότε ανιχνεύεται και το αντίστοιχο slow-to-rise σφάλμα. Με παρόμοιο τρόπο χρησιμοποιείται το σφάλμα stuck-at-1 για να ανιχνευθεί ένα σφάλμα slow-to-fall.

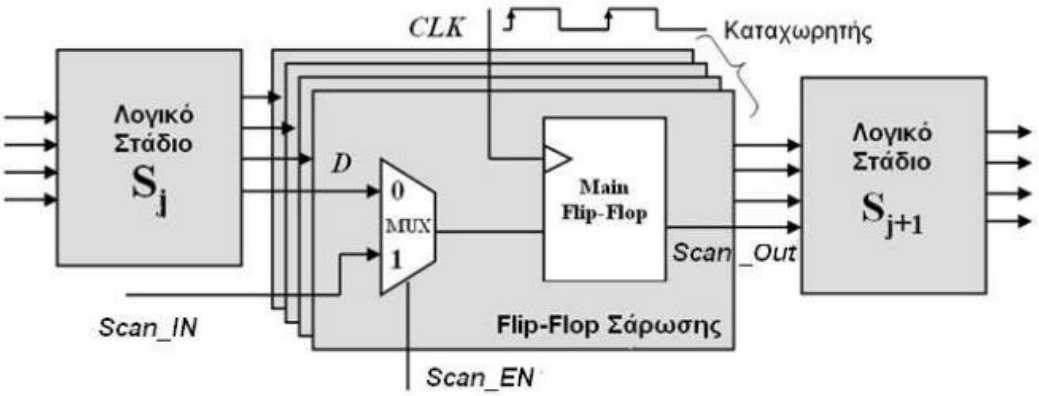
ΚΕΦΑΛΑΙΟ 3

ΑΥΤΟΔΟΚΙΜΗ ΜΕ ΛΟΓΙΣΜΙΚΟ

3.1 Έλεγχος ορθής λειτουργίας επεξεργαστών

Οι επεξεργαστές σε πολλές εφαρμογές χρησιμοποιούνται είτε ως αυτόνομα ολοκληρωμένα κυκλώματα (μικροεπεξεργαστές), είτε ως ενσωματωμένες μονάδες (embedded processors) που και στις δύο περιπτώσεις αποτελούν το κύριο κομμάτι του συστήματος. Ο έλεγχος ορθής λειτουργίας των επεξεργαστών αποτελεί μεγάλη πρόκληση λόγω των ιδιαιτεροτήτων τους [18]. Μια από αυτές είναι οι πολύ υψηλές συχνότητες λειτουργίας του που έχουν ως συνέπεια να απαιτείται εξοπλισμός ελέγχου με αντίστοιχη απόδοση και ακρίβεια του οποίου το κόστος είναι ιδιαίτερα υψηλό [19]. Επίσης λόγω της αυξανόμενης κλίμακας ολοκλήρωσης η πολυπλοκότητα των συστημάτων έχει μεγαλώσει. Αυτό έχει σαν συνέπεια την ανάγκη για πιο αποδοτικές τεχνικές ελέγχου των επεξεργαστών.

Μερικές από τις πιο διαδεδομένες τεχνικές ελέγχου των επεξεργαστών είναι η χρήση αλυσίδων σάρωσης (Scan Testing) [20][21][22][23]. Στην τεχνική αυτή κατά την παραγωγή του ολοκληρωμένου προστίθενται επιπλέον κυκλωματικά στοιχεία που είναι υπεύθυνα για την έλεγχο του ολοκληρώματος [2]. Τα κυκλωματικά στοιχεία αυτά είναι πολυπλέκτες και φλιπ-φλοπ, και επιπλέον είσοδοι και έξοδοι στο κύκλωμα, όπως φαίνεται στο σχήμα 3.1.



Σχήμα 3.1 - Έλεγχος με σάρωση

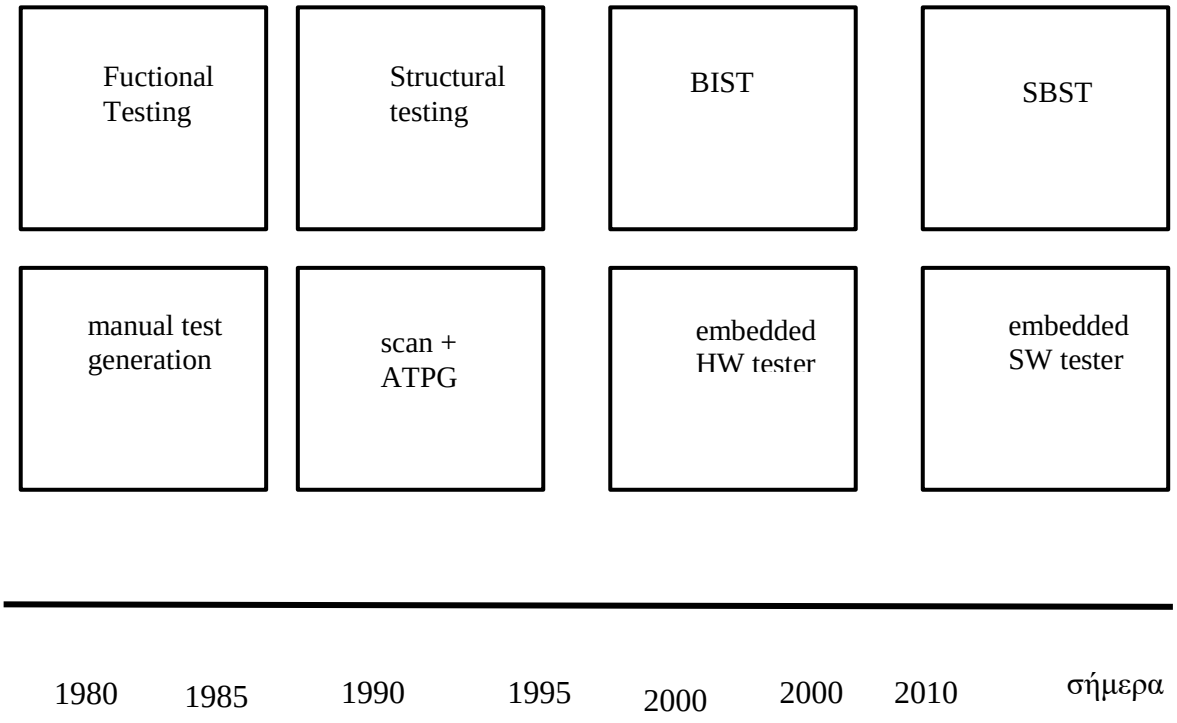
Η χρήση των πολυπλεκτών είναι απαραίτητη για την μετατροπή των φλιπ-φλοπ του κυκλώματος σε φλιπ-φλοπ σάρωσης, δηλαδή στοιχείων μνήμης που διατηρούν τις αποκρίσεις του κυκλώματος. Μέσω των τιμών των φλιπ-φλοπ σάρωσης επιτυγχάνεται η ανίχνευση πιθανών σφαλμάτων που μπορεί να εμφανιστούν στον επεξεργαστή. Το μειονέκτημα των τεχνικών αυτών είναι ότι αυξάνεται η επιφάνεια του ολοκληρωμένου, λόγω της προσθήκης επιπλέον υλικού, ενώ επιπλέον υπάρχει μείωση στην ταχύτητα λειτουργίας έως και 1-2%, αλλά και αύξηση στην κατανάλωση ρεύματος κατά τον έλεγχο [24][25][26]. Επιπλέον, λόγω του μεγάλου όγκου δεδομένων ελέγχου αυξάνεται κατά πολύ και ο χρόνος του ελέγχου.

Για την διενέργεια του ελέγχου με αλυσίδες σάρωσης απαιτείται η χρήση αυτόματου εξοπλισμού ελέγχου δοκιμής (Automatic Test Equipment). Η τεχνική ελέγχου αυτή εφαρμόζεται αφού έχει ολοκληρωθεί η κατασκευή του επεξεργαστή. Επιπλέον, το κόστος του εξοπλισμού ελέγχου αυξάνεται πολύ με την συχνότητα λειτουργίας του και τη μνήμη του. Επειδή οι επεξεργαστές λειτουργούν σε τάξεις GHz είναι αναγκασμένοι και οι εξοπλισμοί ελέγχου να πραγματοποιούν μετρήσεις μικρών μεταβολών τάσης της τάξης των picosecond. Οι λανθασμένες μετρήσεις έχουν σαν συνέπεια ένας ορθά κατασκευασμένος επεξεργαστής να θεωρείται ελαττωματικός και φυσικά προκαλούν την αύξηση του κόστος του επεξεργαστή λόγω της μείωσης της παραγωγής.

Μια άλλη τεχνική ελέγχου είναι ο ενσωματωμένος αυτοέλεγχος του επεξεργαστή (Build-in Self-Test - BIST)[27][28][29]. Στην τεχνική αυτή παράγονται και εφαρμόζονται διανύσματα για τον έλεγχο της ορθής λειτουργίας και συλλέγονται οι αποκρίσεις από τον ίδιο τον επεξεργαστή. Το πλεονέκτημα σε αυτήν την τεχνική είναι η ανίχνευση σφαλμάτων στις ταχύτητες λειτουργίες του επεξεργαστή (at-speed testing),

συμπεριλαμβάνοντας έτσι και τα σφάλματα καθυστέρησης. Επιπλέον, η τεχνική αυτή μπορεί να εφαρμόζεται και για περιοδικό έλεγχο του επεξεργαστή καθ'όλη τη διάρκεια ζωής του. Ωστόσο, το υλικό ελέγχου που προστίθεται στον επεξεργαστή αυξάνει την επιφάνεια του, ενώ απαιτείται μεγάλος αποθηκευτικός χώρος για τα διανύσματα ελέγχου. Ο χώρος αυτός μπορεί να εξοικονομηθεί με την παραγωγή τυχαίων διανυσμάτων ελέγχου, αλλά σε αυτή την περίπτωση μειώνεται σημαντικά η κάλυψη σφαλμάτων και αυξάνεται η κατανάλωση ελέγχου του επεξεργαστή.

3.2 Αυτοέλεγχος με λογισμικό



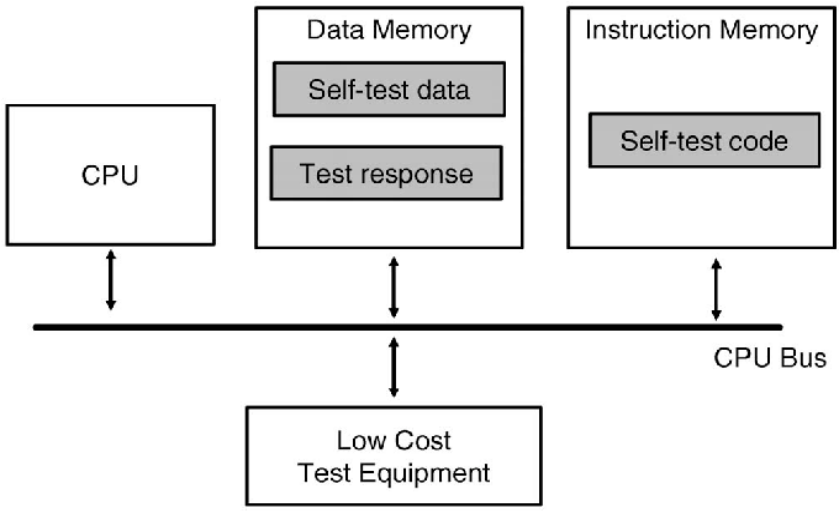
Σχήμα 3.2 - Τεχνικές ελέγχου

Μια άλλη τεχνική ελέγχου είναι ο αυτοέλεγχος με λογισμικό (Software-Based Self-Test, SBST)[30] [31] [32] [33] [34] [35] [36]. Η κεντρική ιδέα του αυτοελέγχου με λογισμικό είναι η βέλτιστη αξιοποίηση της αρχιτεκτονικής του συνόλου εντολών (Instruction Set Architecture - ISA) με απώτερο σκοπό τη δημιουργία διανυσμάτων ελέγχου, τη μεταφορά δεδομένων ελέγχου και την ανάλυση των αποκρίσεων. Κατά τον

αυτοέλεγχο με λογισμικό, ο μικροεπεξεργαστής εκτελεί ένα σύνολο ρουτινών, οι οποίες παίζουν ουσιαστικά τον ρόλο των διανυσμάτων ελέγχου. Στη συνέχεια ο μικροεπεξεργαστής συλλέγει τις αποκρίσεις των ρουτινών σε ενσωματωμένη μνήμη δεδομένων (embedded data memory) και υπολογίζει μία υπογραφή η οποία πρέπει να συμπίπτει με την αναμενόμενη.

Η σημαντική καινοτομία της τεχνικής SBST σε σχέση με την τεχνική BIST είναι η διενέργεια του ελέγχου με λογισμικό. Έτσι ο ρόλος του εξωτερικού ελεγκτή περιορίζεται στο να τοποθετήσει στην ενσωματωμένη μνήμη εντολών και δεδομένων τα προγράμματα αυτοελέγχου (self-test programs) και να συλλέξει τις αποκρίσεις από την ενσωματωμένη μνήμη δεδομένων, μετά την ολοκλήρωση του ελέγχου. Το πρόγραμμα αυτοελέγχου δομείται έτσι ώστε να στοχεύει στην ενεργοποίηση ορισμένων σφαλμάτων, σε συγκεκριμένες μονάδες, τα οποία ανιχνεύονται μέσω των διαφορετικών αποκρίσεών των προγραμμάτων στην ενσωματωμένη μνήμη. Η SBST τεχνική επιτρέπει έλεγχο κατά την πραγματική συχνότητα λειτουργίας με τη χρήση εξωτερικού ελεγκτή χαμηλού κόστους (low-cost external tester), προσφέροντας υψηλή αξιοπιστία και ποιότητα στα αποτελέσματα. Το κύριο πλεονέκτημα της τεχνικής είναι ότι προσφέρει πολύ χαμηλό κόστος ανάπτυξης και εφαρμογής του ελέγχου.

Η τεχνική του αυτοελέγχου με λογισμικό ακολουθεί τα παρακάτω βήματα όπως φαίνονται στο παρακάτω σχήμα 3.3:



Σχήμα 3.3 - Αυτοέλεγχος με λογισμικό

1. Στην αρχή φορτώνεται ο κώδικας αυτοελέγχου στην ενσωματωμένη μνήμη εντολών του επεξεργαστή με τη χρήση εξωτερικού ελεγκτή, μέσω του πρωτοκόλλου πρόσβασης ελέγχου (test access protocol). Εναλλακτικά μπορεί να χρησιμοποιηθεί μνήμη μόνο ανάγνωσης (ROM) με προ-φορτωμένο τον κώδικα ελέγχου.
2. Τα δεδομένα αυτοελέγχου φορτώνονται στην ενσωματωμένη μνήμη δεδομένων με τη χρήση εξωτερικού ελεγκτή. Τα δεδομένα αυτά μπορεί να είναι μεταβλητές, σταθερές ή ακόμη και διανύσματα ελέγχου (για τον έλεγχο κάποιον υπομονάδων του επεξεργαστή)
3. Έπειτα αρχίζει η εκτέλεση του προγράμματος αυτοελέγχου και τα διανύσματα ελέγχου που παράγονται μέσω της εκτέλεσης των εντολών ελέγχου εφαρμόζονται στα επιμέρους τμήματα του επεξεργαστή. Στη συνέχεια οι αποκρίσεις συλλέγονται σε κάποια ενσωματωμένη μνήμη ή χρησιμοποιώντας αλγορίθμους συμπίεσης αποθηκεύεται μόνο μια υπογραφή της απόκρισης.
4. Τέλος τα αποτελέσματα της εσωτερικής μνήμης συλλέγονται από την εξωτερικό ελεγκτή και εξετάζονται ως προς την ορθότητά τους.

Συγκεντρωτικά η διαδικασία αυτοδοκιμής χωρίζεται σε τρία βασικά στάδια:

- Προετοιμασία του ελέγχου: Τοποθετούνται τα δεδομένα ελέγχου σε καταχωρητές ή σε διευθύνσεις της ενσωματωμένης μνήμης
- Εφαρμογή του προγράμματος ελέγχου και συλλογή των αποκρίσεων: Εκτελούνται τα προγράμματα ελέγχου και οι αποκρίσεις τους συλλέγονται σε καταχωρητές ή ενσωματωμένες μνήμες δεδομένων.
- Εξαγωγή αποκρίσεων: Οι αποκρίσεις συλλέγονται εσωτερικά και εξάγονται προς την ενσωματωμένη μνήμη δεδομένων.

Στη συνέχεια παρατίθεται ένα παράδειγμα αυτοελέγχου με τη χρήση λογισμικού. Έστω ότι η μονάδα αριθμητικών και λογικών πράξεων (ALU) περιέχει σφάλμα στο κύκλωμα του αφαιρέτη. Για την ανίχνευση του σφάλματος σχεδιάζεται κώδικας με εντολές που στοχεύουν στην ενεργοποίηση των γραμμών του κυκλώματος του αφαιρέτη και στη συλλογή των αποκρίσεών του. Στο επόμενο σχήμα φαίνεται ένα παράδειγμα κώδικα που θα μπορούσε να χρησιμοποιηθεί για να επιτευχθούν τα παραπάνω.

```
load r0, xx_addr
load r1, yy_addr
sub r2 , r0, r1
store r2, zz_addr
```

Στο παράδειγμα αυτό τα διανυσματα δοκιμής προέρχονται από δύο καταχωρητές (R0, R1) και το αποτέλεσμα της αφαίρεσης αποθηκεύεται στον καταχωρητή R2. Οι μεταβλητές xx_addr και yy_addr χρησιμοποιούνται για την διευθυνσιοδότηση των θέσεων μνήμης που είναι αποθηκευμένα τα διανύσματα xx και yy και η απόκριση της δοκιμής zz = xx - yy. Έτσι με τις δύο πρώτες εντολές load φορτώνονται τα δεδομένα από την ενσωματωμένη μνήμη δεδομένων στην ALU. Στη συνέχεια, υλοποιείται η αφαίρεση και οι αποκρίσεις της πράξης αποθηκεύονται σε μια θέση της ενσωματωμένης μνήμης δεδομένων ώστε να επιβεβαιωθεί η ορθότητά της.

Τα πλεονεκτήματα που εμφανίζει η τεχνική αυτοελέγχου με λογισμικό είναι:

1. Ο αυτοέλεγχος με λογισμικό βασίζεται μόνο στο σύνολο εντολών του επεξεργαστή χωρίς να απαιτείται επιπλέον σχεδιασμός του επεξεργαστή.
2. Η τεχνική αυτοελέγχου με λογισμικό μπορεί να πραγματοποιηθεί σε κανονικές συνθήκες λειτουργίας του επεξεργαστή (at-speed testing), δηλαδή σε υψηλές συχνότητες προσφέροντας έτσι τη δυνατότητα ανίχνευσης σφαλμάτων καθυστέρησης του επεξεργαστή που μπορεί να υπάρχουν.
3. Ο αυτοέλεγχος με λογισμικό δεν αυξάνει την κατανάλωση ρεύματος γιατί η εφαρμογή του γίνεται σε συνθήκες κανονικής λειτουργίας του.
4. Ο αυτοέλεγχος με λογισμικό μπορεί να γίνει τόσο κατά την κατασκευή του επεξεργαστή όσο και κατά τη λειτουργία του (on-line testing).
5. Τέλος η τεχνική αυτή μπορεί να χρησιμοποιηθεί σε συγκεκριμένες μονάδες του επεξεργαστή, ελέγχοντας μόνο σε αυτές την ορθότητα λειτουργίας του.

3.3 Επισκόπηση βιβλιογραφίας αυτοελέγχου με λογισμικό

Οι Shen και Abraham στην εργασία [37] προτείνουν την ανάπτυξη λογισμικού αυτοελέγχου απαριθμώντας όλους τους δυνατούς συνδυασμούς εντολών που ενεργοποιούν τις υπομονάδες του επεξεργαστή, και τα διανύσματα ελέγχου που προέρχονται από γεννήτριες ψευδοτυχαίων ακολουθιών. Η μεθοδολογία εφαρμόστηκε στον επεξεργαστή GL85 και το ποσοστό κάλυψης που επιτεύχθηκε ήταν 90,2% και στην περίπτωση συμπίεσης των αποκρίσεων επιτεύχθηκε 86.7% κάλυψη. Έπειτα, στην εργασία [38] τους οι Batcher και Papachristou, προτείνουν την ανάπτυξη λογισμικού αυτοελέγχου όπου η παραγωγή των διανυσμάτων ελέγχου γίνεται με τη χρήση υλικού, επιβαρύνοντας την επιφάνεια του επεξεργαστή RISC που χρησιμοποιήθηκε περίπου 3%. Ανέπτυξαν δύο πειράματα, στο πρώτο επιτεύχθηκε ποσοστό κάλυψης 92.5% με την εκτέλεση 50.000 κύκλων ρολογιού και στο δεύτερο 94,8% με την εκτέλεση 200.000 κύκλων.

Στη συνέχεια παρουσιάστηκε η εργασία [39] των Parvathala, Lindsay και Maneparambil χρησιμοποιώντας τον επεξεργαστή Intel Pentium 4. Στην εργασία αυτή αναπτύχθηκε το σύστημα FRITS που παρήγαγε λογισμικό αυτοελέγχου βασισμένο σε ψευδοτυχαίες ακολουθίες εντολών συνδυασμένο με ψευδοτυχαία διανύσματα ελέγχου. Το ποσοστό κάλυψης που επιτεύχθηκε ήταν 70%. Έπειτα στην εργασία [40], ο Corno και οι υπόλοιποι ερευνητές ανέπτυξαν το αυτοματοποιημένο σύστημα ανάπτυξης ενσωματωμένου λογισμικού MicroGP το οποίο βασίζεται στους γενετικούς αλγορίθμους. Ακολουθούν στη συνέχεια οι εργασίες [41], [42], [43], που οι ερευνητές βασίζονται στα αποτελέσματα που μπορούν να εξάγουν μελετώντας τη σχεδίαση του μικροεπεξεργαστή στο επίπεδο μεταφοράς καταχωρητών. Έπειτα αναπτύσσουν διανύσματα ελέγχου σε διάφορες μονάδες του μικροεπεξεργαστή, όπως αθροιστές, αφαιρέτες, πολλαπλασιαστές, αυξητές και μειωτές. Στην εργασία [41] χρησιμοποιήθηκε ο μικροεπεξεργαστής Parwan 8-bit και πετύχανε ποσοστό κάλυψης 91%. Στις εργασίες [42], [43] χρησιμοποιήθηκε ο μικροεπεξεργαστής MIPS υλοποιημένος σε αρχιτεκτονική MIPS3000. Στις εργασίες [44], [45] οι ερευνητές εξελίσσουν τη μεθοδολογία των παραπάνω δύο εργασιών εφαρμόζοντας τον αυτοέλεγχο σε μικροεπεξεργαστές που διαθέτουν περίπλοκους μηχανισμούς διοχέτευσης με προώθηση (forwarding). Στις εργασίες αυτές χρησιμοποιήθηκαν δυο μικροεπεξεργαστές με 5 στάδια διοχέτευσης, ο miniMips και ο OpenRick 1200.

Τέλος με την αύξηση της τεχνολογίας στα συστήματα τείνουν να ενσωματώνονται περισσότεροι πυρήνες σε έναν επεξεργαστή. Αυτό έχει σαν συνέπεια ο έλεγχος αυτών των συστημάτων να καθίσταται πιο δύσκολος. Έτσι, θα πρέπει να υλοποιείται πιο σύνθετος κώδικας για τον έλεγχο των πολυπύρηνων επεξεργαστών, ο οποίος καθιστά τον έλεγχο ορθής λειτουργίας του μικροεπεξεργαστή πιο αργό.

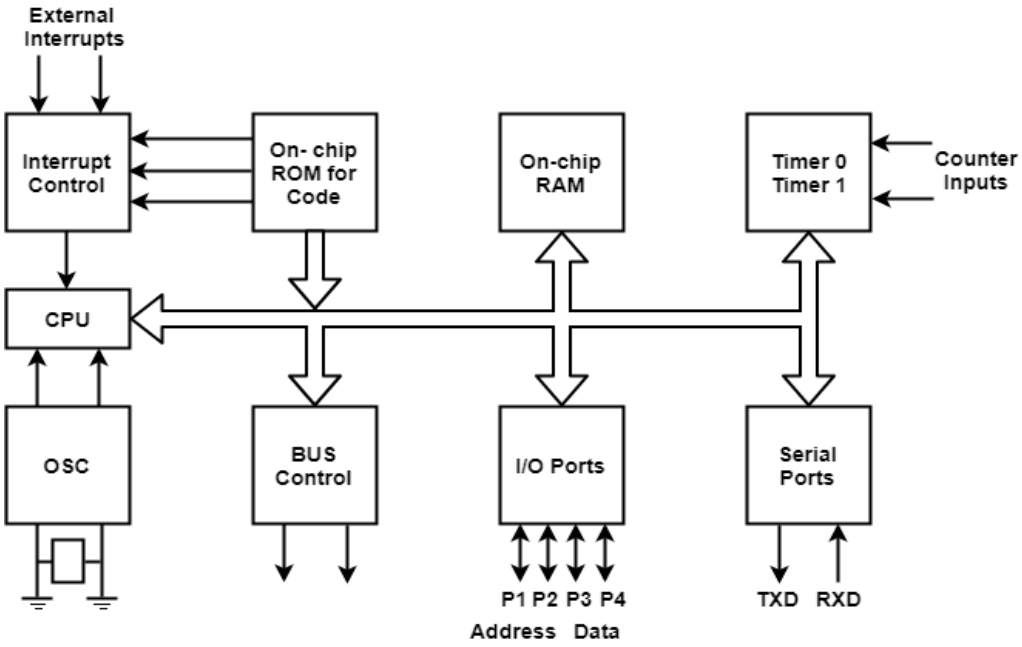
ΚΕΦΑΛΑΙΟ 4

ΕΠΕΞΕΡΓΑΣΤΗΣ I8051

4.1 Περιγραφή του επεξεργαστή i8051

Ο επεξεργαστής της Intel 8051 [46][47] είναι στην ουσία ένας μικροελεγκτής (μC) ο οποίος δημιουργήθηκε από την Intel το 1980 για τη χρήση του σε ενσωματωμένα συστήματα. Ήταν πολύ δημοφιλής από το 1980 μέχρι τις αρχές του 1990 και στην δική μας εργασία αποτελεί ένα απλό μετροκύκλωμα για την αξιολόγηση του εξομοιωτή.

Το αρχιτεκτονικό διάγραμμα ενός τέτοιου μικροελεγκτή φαίνεται στο σχήμα 4.1**.** Σε αυτό φαίνονται οι ενσωματωμένες περιφερειακές μονάδες του μικροελεγκτή, καθώς και οι διασυνδέσεις μεταξύ τους. Επιπλέον, διακρίνεται η ύπαρξη ενός εσωτερικού διαύλου δεδομένων μεταξύ των περιφερειακών, που παρέχει την δυνατότητα άμεσης διευθυνσιοδότησης των περιφερειακών από την CPU.

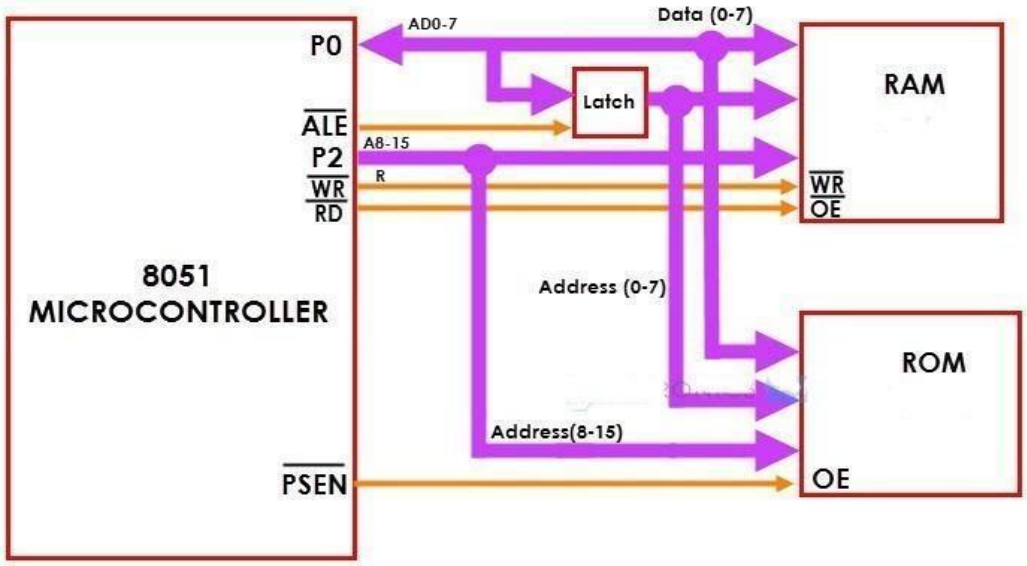


Σχήμα 4.1 - Επεξεργαστής 8051

Ο μικροελεγκτής διαθέτει ξεχωριστό χώρο για μνήμη προγράμματος και μνήμη δεδομένων, όπως φαίνεται και στο σχήμα 4.2. Ο λογικός αυτός διαχωρισμός μεταξύ της μνήμης προγράμματος και της μνήμης δεδομένων επιτρέπει στην μνήμη δεδομένων να προσπελαύνεται από διευθύνσεις των 8 bits, τις οποίες μια CPU των 8 bits μπορεί να αποθηκεύσει και να διαχειριστεί πολύ πιο εύκολα. Επιπλέον υπάρχει η δυνατότητα προσπέλασης και 16μπιτων διευθύνσεων μέσω του καταχωρητή DPTR.

Η πρώτη μνήμη του επεξεργαστή είναι η μνήμη προγράμματος (ROM), η οποία μπορεί να βρίσκεται πάνω ή εκτός του ολοκληρωμένου. Μπορούμε να διαβάσουμε εντολές προγράμματος από αυτόν τον χώρο, αλλά δεν μπορούμε να γράψουμε οτιδήποτε σε αυτόν τον χώρο. Το σύνολο της μνήμης προγράμματος εντός και εκτός του ολοκληρώματος δεν μπορεί να υπερβαίνει τα 64Kb. Η επιλογή για εξωτερική μνήμη γίνεται μέσω του σήματος EA του 8051. Για EA = 0 έχουμε την προσπέλαση μόνο εξωτερικής μνήμης, ενώ για EA = 1 προσπελαύνεται αρχικά η εσωτερική μνήμη, και στη συνέχεια η εξωτερική. Τέλος, το σήμα ανάγνωσης της μνήμης είναι το PSEN.

Η δεύτερη μνήμη είναι η μνήμη δεδομένων (RAM), ανάγνωσης και εγγραφής. Η CPU μπορεί να διαβάσει και να γράψει δεδομένα από και προς τον χώρο αυτόν. Ο 8051 μπορεί να υποστηρίξει εσωτερική μνήμη δεδομένων 256 bytes και έως 64K εξωτερικής μνήμης δεδομένων, τις οποίες τις διαχειρίζεται μέσω των σημάτων RD και WR. Τα σήματα αυτά αφορούν την ανάγνωση και την εγγραφή αντίστοιχα της μνήμης RAM[47].



Σχήμα 4.2 - Μνήμες επεξεργαστή

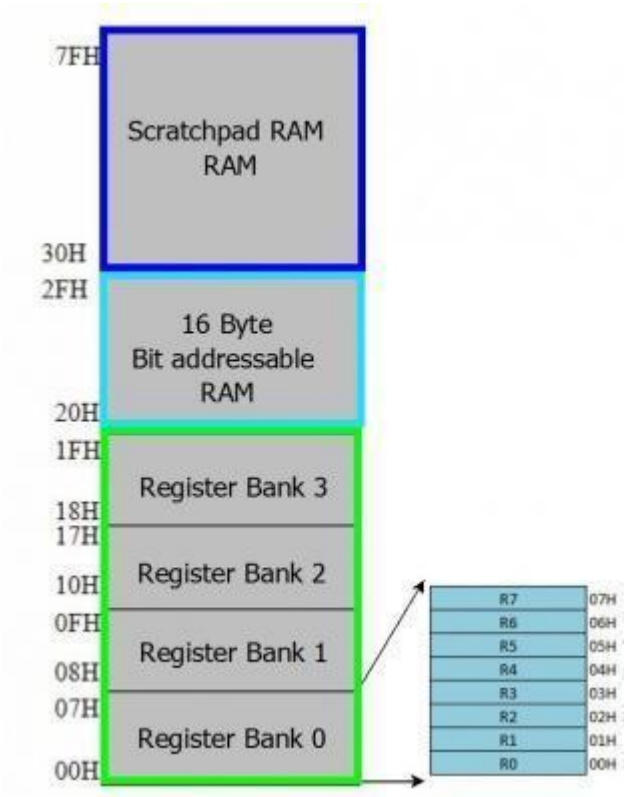
4.2 Οι καταχωρητές και η εσωτερική μνήμη του 8051

Για να παρέχει ευκολία προγραμματισμού, ο 8051 επιτρέπει στους προγραμματιστές να διευθυνσιοδοτήσουν την εσωτερική RAM με πολλούς διαφορετικούς τρόπους. Τα πρώτα 32 bytes της εσωτερικής RAM διευθυνσιοδοτούνται σαν θέσεις μνήμης ή καταχωρητές. Καλώντας κάποιους καταχωρητές σαν θέσεις μνήμης μπορούμε να τους προσπελάσουμε με εντολές του ενός byte. Αυτές οι εντολές εκτελούνται πολύ γρήγορα, και επιτρέπουν στον προγραμματιστή να γράψει πολύ αποδοτικό κώδικα[47].

Τα επόμενα 16 bytes της εσωτερικής μνήμης του 8051, μπορούν να διευθυνσιοδοτηθούν ταυτόχρονα, ανά bit ή ανά byte. Οι συνήθεις εντολές μπορούν να διευθυνσιοδοτήσουν ένα byte δεδομένων σ'αυτές τις θέσεις μνήμης, ενώ υπάρχουν και ειδικές εντολές, οι οποίες μπορούν να διευθυνσιοδοτήσουν ένα συγκεκριμένο bit σε αυτήν την περιοχή της μνήμης. Οι εντολές αυτές, (Bit instructions) είναι πολύ αποδοτικές όταν θέλουμε να επεξεργαστούμε δεδομένα τα οποία προέρχονται από εξωτερικές συσκευές.

Κατόπιν ακολουθούν τα 80 bytes "Μνήμης γενικής χρήσεως" τα οποία χρησιμεύουν σαν χώροι γενικής αποθήκευσης μεταβλητών. Στο τέλος της μνήμης αυτής βρίσκεται ο χώρος των SFRs (Special Function Registers) ή Καταχωρητών Ειδικών

Λειτουργιών, οι οποίοι όμως δεν αναφέρονται σαν μνήμη, αφού έχουν ειδικό σκοπό.[47] Στο σχήμα 4.3 φαίνεται ο χάρτης της εσωτερικής μνήμης των πρώτων 128 bytes της RAM.



Σχήμα 4.3 - Εσωτερικό κύριας μνήμης [48]

Αναλυτικότερα, τα 32 πρώτα bytes μπορούν να χρησιμοποιηθούν σαν 4 ομάδες καταχωρητών (Register Banks). Ως καταχωρητή εννοούμε μια θέση μνήμης ανάγνωσης/εγγραφής 8-bits που μπορεί να διευθυνσιοδοτηθεί με μία απλή εντολή ενός byte. Η ομάδα καταχωρητών που χρησιμοποιείται κάθε φορά, καθορίζεται από τον προγραμματιστή, επιλέγοντας κατάλληλες τιμές για τα bits RS1 και RS0 του καταχωρητή PSW(Program Status Word).

Τέλος, είναι σημαντικό να σημειωθεί ότι μέσα στην μνήμη δεδομένων ορίζεται η στοίβα του επεξεργαστή 8051, δηλαδή η διαδοχική αποθήκευση δεδομένων. Συγκεκριμένα, η αρχή της στοίβας, όπου βρίσκεται το 1ο Byte, είναι η διεύθυνση 07H. Στη συνέχεια με κάθε byte δεδομένων που αποθηκεύεται στη στοίβα, ο stack pointer (SC) αυξάνεται προς τα πάνω, καταλαμβάνοντας τα δεδομένα αυτά επόμενη θέση μνήμης Παρατηρώντας το σχήμα 4.3**,** η στοίβα του μικροεπεξεργαστή συμπίπτει με τον

καταχωρητή R0 της ομάδας 1. Επιπλέον, υπάρχει η δυνατότητα αλλαγής της πρώτης αρχικής διεύθυνσης της στοίβας, πάντα όμως μέσα στην εσωτερική μνήμη.

4.3 Καταχωρητές ειδικών λειτουργιών

Ο καθένας από αυτούς τους καταχωρητές είναι αφοσιωμένος σε μια συγκεκριμένη λειτουργία. Μερικοί από αυτούς είναι μέρος της CPU ενώ οι υπόλοιποι εκτελούν λειτουργίες ελέγχου των διαφόρων εισόδων/εξόδων (I/O)[47]. Σε κάθε τέτοιο καταχωρητή αντιστοιχεί και μια διεύθυνση της εσωτερικής μνήμης. Ακόμη μερικοί από αυτούς μπορούν να διευθυνσιοδοτηθούν και κατά bit. Στο σχήμα 4.4 φαίνεται ένας χάρτης των καταχωρητών ειδικών λειτουργιών με τις διευθύνσεις που τους αντιστοιχούν.

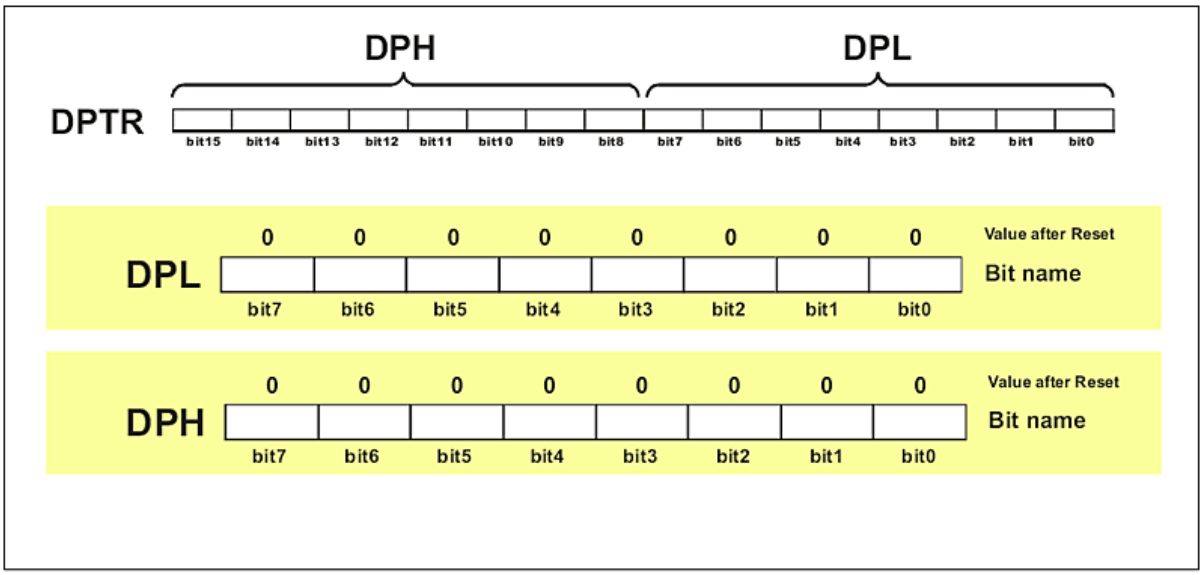
F8								FF
F0	**B**							F7
E8								EF
E0	**ACC**							E7
D8								DF
D0	**PSW**					SPCR		D7
C8	**T2CON**	T2MOD	RCAP2L	RCAP2H	TL2	TH2		CF
C0								C7
B8	**IP**	SADEN						BF
B0	**P3**							IPH B7
A8	**IE**	SADDR	SPSR					AF
A0	**P2**						WDTRST	WDTCON A7
98	**SCON**	SBUF						9F
90	**P1**						EECON	97
88	**TCON**	TMOD	TL0	TL1	TH0	TH1	AUXR	CLKREG 8F
80	**P0**	SP	DP0L	DP0H	DP1L	DP1H	SPDR	PCON 87

Σχήμα 4.4 - Καταχωρητές ειδικών λειτουργιών [48]

Οι SFRs της CPU περιλαμβάνουν τον συσσωρευτή (ACC), τον καταχωρητή B, τη λέξη κατάστασης προγράμματος (PSW), τον δείκτη στοίβας (SP) και τον δείκτη δεδομένων (DPTR), όπως επίσης υπάρχουν και καταχωρητές που ελέγχουν τα περιφερειακά που είναι ενσωματωμένα στον μικροελεγκτή. Παρακάτω θα παρουσιαστούν όλοι οι καταχωρητές περιληπτικά.

4.3.1 Καταχωρητής DPTR - Data Pointer Register

Ο καταχωρητής DPTR είναι ένας δείκτης δεδομένων 16-bits (2 bytes) που είναι απαραίτητος για την διευθυνσιοδότηση των δεδομένων στη μνήμη. Ο καταχωρητής αυτός διαχωρίζεται σε bits υψηλής τάξης (DPH) και bits χαμηλής τάξης (DPL) των 8 bits ο καθένας[47][48]. Ο DPTR χρησιμοποιείται στην ανάγνωση ή την εγγραφή δεδομένων με Indirect ή Indexed Addressing mode όταν διευθυνσιοδοτείται η εξωτερική μνήμη δεδομένων ή προγράμματος. Οι εντολές του 8051 επιτρέπουν τη χρήση του καταχωρητή αυτού, σαν έναν καταχωρητή 16-bit ή σαν δύο των 8-bits.



Σχήμα 4.5 - DPTR καταχωρητής

4.3.2 Καταχωρητής SP - Stack Pointer

SP

Ο καταχωρητής SP είναι ο δείκτης στοίβας του 8051 και αποτελείται από 8 bits.[47] Είναι απαραίτητος για την διευθυνσιοδότηση της εσωτερικής μνήμης του 8051. Αυξάνεται ή μειώνεται λίγο πριν την αποθήκευση ή ανάκτηση δεδομένων που είναι αποτέλεσμα μιας εντολής PUSH ή PULL αντίστοιχα.

4.3.3 Καταχωρητής PSW - Program Status Word

Ο καταχωρητής PSW αποτελείται από 8-bits κατάστασης που αντικατοπτρίζουν την τρέχουσα κατάσταση της CPU[47]. Αναφέρεται επίσης ως καταχωρητής σημαία (flag register). Αν και ο καταχωρητής PSW έχει πλάτος 8 bits, μόνο το 6 bits του χρησιμοποιείται από το 8051. Αναλυτικά στο επόμενη σχήμα φαίνεται η δομή του και η λειτουργία κάθε bit.

(MSB) (LSB)

C	AC	F0	RS1	RS0	OV	-	P

Σχήμα 4.6 - PSW καταχωρητής

C (Carry Flag) : Εμφανίζεται κατά τη διάρκεια ορισμένων αρι-θμητικών και λογικών εντολών.

AC (Auxiliary Carry Flag) : Εμφανίζεται κατά τη διάρκεια προσθέσεων ή αφαιρέσεων, για τη δημιουργία κρατούμενου.

F0 (Flag Zero) : Τίθεται σαν ένα καθοριζόμενο από τον χρήστη flag κατάστασης.

RS1 (Register bank Select control bit) : Με κατάλληλους συνδυασμούς επιλέγεται η κατάλληλη τράπεζα καταχωρητών.

RS0 (Register bank Select control bit) : Με κατάλληλους συνδυασμούς επιλέγεται η κατάλληλη τράπεζα καταχωρητών.

Οι επιτρεπτοί συνδυασμοί είναι οι εξής:

RS0	RS1	BANK	ADDRESS
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Πίνακας 4.1 – PCON

OV (Overflow Flag) : Εμφανίζεται σε περίπτωση υπερχείλισης αρι-θμητικών εντολών

- : Δεσμευμένο

P (Parity Flag) : Τίθεται σε κάθε instruction cycle όταν στον Accumulator υπάρχει ένας άρτιος / περιττός αριθμός από “1” αντίστοιχα.

4.3.4 Καταχωρητής PCON – Power Control Register

Ο καταχωρητής PCON είναι υπεύθυνος για τον έλεγχο της λειτουργίας του επεξεργαστή, για την πτώση ισχύος του επεξεργαστή (powerdown) και για το ρυθμό των σειριακών δεδομένων (serial data bandrate).[47] Συγκεκριμένα το 7ο bit (D7) είναι υπεύθυνο για τη δημιουργία του ρυθμού baud της σειριακής επικοινωνίας. Αναλυτικά στο επόμενο σχήμα φαίνεται η δομή του και η λειτουργία κάθε bit.

D7	D6	D5	D4	D3	D2	D1	D0
SMOD	X	X	X	GF1	GF0	PD	IDL

Σχήμα 4.7 - PCON καταχωρητής

Address: 87H (not bit addressable)

SMOD – Εάν για την δημιουργία baud rate χρησιμοποιείται ο timer 1 και SMOD = 1 τότε το baud rate διπλασιάζεται, εφ’ όσον βέβαια η σειριακή πόρτα δουλεύει στα modes 1, 2 ή 3.

GF1 and GF0: Γενικού σκοπού flag bit

PD : Θέτοντας αυτό το bit σταματούν όλες οι εσωτερικές διεργασίες χωρίς όμως να χαθούν τα δεδομένα που τυχόν υπάρχουν στην RAM. Αυτή η κατάσταση είναι διαθέσιμη μόνο σε ολοκληρωμένα CHMOS της οικογένειας.

IDL : Θέτοντας αυτό το bit βάζουμε το ολοκληρωμένο σε κατάσταση idle.

Σε περίπτωση που τεθούν ταυτόχρονα τα bits PD και IDL τότε επικρατεί το PD.

4.3.5 Καταχωρητής TCON – Timer Control Register

Ο καταχωρητής TCON αποτελείται από 8 bits και είναι υπεύθυνος για την αρχικοποίηση των διακοπτών (interrupts).[47] Τα πιο σημαντικά bits είναι ο χρονομετρητής TR (timer run) και ο χρονόμετρο υπερχείλισης TF (timer overflow) των οποίων οι λειτουργίες τους αναφέρεται παρακάτω στο σχήμα.

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Σχήμα 4.8 - TCON καταχωρητής

Address: 88H (bit addressable)

TF1 – Timer 1, overflow flag : Τίθεται όταν έχουμε υπερχείλιση του Timer/Counter. Εκκαθαρίζεται όταν ο επεξεργαστής δείχνει στη ρουτίνα εξυπηρέτησης διακοπής.

TR1 – Timer 1 run control bit: Τίθεται για να επιτρέψει ή όχι την λειτουργία του Timer/Counter 1.

TF0 – Timer 0 overflow flag: Ίδια λειτουργία με το TF1 όμως αφορά τον Timer/Counter 0

TR0 – Timer 0 run control bit: Ίδια λειτουργία με το TR1 όμως αφορά τον Timer/Counter 0.

IE1 – External interrupt 1 edge flag: Τίθεται όταν ανιχνευτεί η ακμή της εξωτερικής διακοπής 1. Εκκαθαρίζεται μετά το πέρας της διακοπής.

IT1 – Edge control bit for external: Τίθεται για να καθοριστεί εάν η εξωτερική διακοπή θα γίνεται με την ακμή ή με το χαμηλό επίπεδο του σήματος διακοπής.

IE0 – External interrupt 0 edge flag: Έχει την ίδια σημασία με το IE1

IT0 – Edge control bit for external: Έχει την ίδια σημασία με το IT1

4.3.6 Καταχωρητής SCON - Serial Control Register

Ο SCON είναι ένας 8 bit καταχωρητής και χρησιμοποιείται για την σειριακή επικοινωνία[47].

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Σχήμα 4.9 - SCON καταχωρητής

Address: 98H (bit-addressable)

SM0 : Serial port Mode specifier
SM1 : Serial port Mode specifier

Ο συνδυασμός των SM10, SM1 επιλέγει τον τρόπο λειτουργίας της σειριακής θύρας ως εξής:

SM0	**SM1**	**Mode**	**Περιγραφή**	**Baud Rate**
0	0	0	Shift Register	Fosc/12
0	1	1	8-bit UART	Variable
1	0	2	9-bit UART	Fosc/64 OR fosc/32
1	1	3	9-bit UART	Variable

Πίνακας 4.2 - SCON

SM2 – Ενεργοποιεί την δυνατότητα για επικοινωνία μεταξύ επεξεργαστών στα modes 2 & 3. Στα modes 2 ή 3 εάν το SM2 είναι 1 τότε το RI δεν θα ενεργοποιηθεί εάν το λαμβανόμενο 9ο bit (RB8) είναι 0. Στο mode 1 εάν SM2 = 1 τότε το RI δεν θα ενεργοποιηθεί εάν δεν ληφθεί ένα αποδεκτό stop bit. Στο mode 0, το mode 0, το SM2 θα πρέπει να είναι 0.

REN – Receiver enable : Τίθεται ή εκκαθαρίζεται για να ενεργοποιήσει ή απενεργοποιήσει την λήψη δεδομένων.

TB8 – Transmit bit 8: Το 9ο bit το οποίο μεταδίδεται στα modes 2 & 3.

RB8 – Receive bit 8: Το 9ο bit το οποίο λαμβάνεται στα modes 2 & 3. Στο mode 0 το RB8 δεν χρησιμοποιείται.

TI – Transmit interrupt flag: Τίθεται κατά το τέλος του χρόνου του 8ου bit στο mode 0, ενώ κατά τα άλλα modes κατά την αρχή του χρόνου. Θα πρέπει να εκκαθαριστεί από το software.

RI – Receive interrupt flag: Τίθεται κατά το τέλος του χρόνου του 8ου bit στο mode 0, ενώ στα άλλα modes κατά το μισό της διάρκειας του 8ου bit. Θα πρέπει να εκκαθαριστεί από το software.

4.3.7 TMOD – Timer Mode Control Register

Ο TMOD είναι ένα καταχωρητής 8 bit και είναι υπεύθυνος για την επιλογή το ρολογιού[47][48]. Επιλέγει ανάμεσα σε 4 βαθμίδες ανάλογα τις τιμές των M0 και M1, όπως φαίνεται παρακάτω στο σχήμα 4.10.

D7	D6	D5	D4	D3	D2	D1	D0
GATE	C/T	M1	M0	GATE	C/T	M1	M0

Σχήμα 4.10 - TMOD καταχωρητής

TIMER 0 | TIMER 1

Gate - Όταν έχει τεθεί το bit TRx στον TCON και GATE = 1 τότε ο TIMER/Counter θα είναι σε λειτουργία μόνο εάν το Pin INTx είναι σε λογικό ‘1’. Όταν Gate = 0 ο Timer/Counter θα είναι σε λειτουργία μόνο όταν TRx του TCON είναι σε λογικό ‘1’.

T/C - Όταν εκκαθαρίζεται έχουμε λειτουργία Timer (με είσοδο από το εσωτερικό ρολόι). Όταν τίθεται έχουμε λειτουργία Counter (με είσοδο από το pin Tx).

M1 - Mode selector bit
M0 - Mode selector bit

Ο συνδυασμός των M0, M1 καθορίζει τον τρόπο λειτουργίας ως εξής:

M0	**M1**	**Τρόπος λειτουργίας**
0	0	**0** 13-bit Timer
0	1	**1** 16-bit Timer
1	0	**2** 8-bit auto-reload Timer/Counter
1	1	**3** (Timer 0): TL0 είναι ένας 8-bit Timer/Counter ελεγχόμενος από τα bits ελέγχου του Timer 0. Ο TH1 είναι ένας 8-bit Timer/Counter ελεγχόμενος όμως από τα bits ελέγχου του Timer 1
1	1	**4** Ο Timer/COunter 1 εκτός λειτουργίας

Πίνακας 4.3 - TMOD

4.3.8 IE – Interrupt Enable Register

Ο IE καταχωρητής αποτελείται από 8 bit και είναι υπεύθυνος για την ενεργοποίηση και την απενεργοποίηση των διακοπτών (interrupts).[47] Συγκεκριμένα θέτοντας το D7 bit του καταχωρητή σε ‘1’ ενεργοποιούνται οι διακόπτες, ενώ στην περίπτωση του μηδενός όλοι οι διακόπτες απενεργοποιούνται. Περισσότερες λεπτομέρειες φαίνονται στο παρακάτω σχήμα.

D7	D6	D5	D4	D3	D2	D1	D0
EA	-	ET2	ES	ET1	EX1	ET0	EX0

Σχήμα 4.11 - IE καταχωρητής

EA - Αδρανοποιεί όλες τις διακοπές. Εάν EA = 0 τότε καμιά διακοπή δεν θα ληφθεί υπ'όψην. Εάν τώρα EA = 1 τότε κάθε διακοπή θα είναι ανεξάρτητα ενεργή ή μη ενεργή, ανάλογα με το εάν έχει τεθεί ή εκκαθαριστεί το bit ενεργοποίησης της.

\- - Δεσμευμένο για μελλοντική χρήση

ET2 - Ενεργοποιεί ή αδρανοποιεί την διακοπή από υπερχείλιση

ES - Ενεργοποιεί ή αδρανοποιεί την διακοπή από την σειριακή πόρτα

ET1 - Ενεργοποιεί ή αδρανοποιεί την διακοπή από υπερχείλιση του Timer1

EX1 - Ενεργοποιεί ή αδρανοποιεί την διακοπή από την γραμμή εξωτερικής διακοπής 1.

ET0 - Ενεργοποιεί ή αδρανοποιεί την διακοπή από υπερχείλιση του Timer 0.

EX0 - Ενεργοποιεί ή αδρανοποιεί την διακοπή από την γραμμή εξωτερικής διακοπής 0.

4.3.9 IP – Interrupt Priority Register

Ο καταχωρητής IP είναι υπεύθυνος για τον καθορισμό των προτεραιοτήτων των διακοπτών, και λεπτομερών φαίνεται στο παρακάτω σχήμα 4.12 η λειτουργία του [48].

D7	D6	D5	D4	D3	D2	D1	D0
-	-	PT2	PS	PT1	PX1	PT0	PX0

Σχήμα 4.12 - IP καταχωρητής

\- - Δεσμευμένο για μελλοντική χρήση

\- - Δεσμευμένο για μελλοντική χρήση

PT2 - Για τον i8051 δεν χρησιμοποιείται

PS - Καθορίζει το επίπεδο προτεραιότητας της διακοπής από την σειριακή πόρτα.

PT1 - Καθορίζει το επίπεδο προτεραιότητας της διακοπής από τον Timer 1.

PX1 - Καθορίζει το επίπεδο προτεραιότητας της εξωτερικής διακοπής 1.

PT0 - Καθορίζει το επίπεδο προτεραιότητας της διακοπής από τον Timer 0.

PX1 - Καθορίζει το επίπεδο προτεραιότητας της εξωτερικής διακοπής 0

4.3.10 Καταχωρητής A

Ο καταχωρητής A (συσσωρευτής), χρησιμοποιείται για όλες τις αριθμητικές και λογικές πράξεις του επεξεργαστή. Εάν ο συσσωρευτής δεν υπήρχε, τότε το αποτέλεσμα κάθε πράξης (πρόσθεσης, πολλαπλασιασμού, ολίσθησης κλπ.) θα έπρεπε να αποθηκευτεί στην κύρια μνήμη, η πρόσβαση στην οποία όμως είναι πιο αργή από την πρόσβαση σε ένα καταχωρητή όπως ο συσσωρευτής [48].

4.3.11 Καταχωρητής B

Ο καταχωρητής B έχει παρόμοια λειτουργία με τον καταχωρητή A αλλά χρησιμοποιείται μόνο στη διαίρεση και τον πολλαπλασιασμό. Για οποιαδήποτε άλλη πράξη ή λειτουργία, αυτός χρησιμοποιείται σαν καταχωρητής γενικής χρήσης.

4.3.12 Καταχωρητής SBUF - Serial data buffer

Ο καταχωρητής SBUF είναι υπεύθυνος για τη διατήρηση των δεδομένων. Ουσιαστικά ο καταχωρητής αυτός αποτελείται από δύο καταχωρητές, τον καταχωρητή εκπομπής (Transmit buffer register) και τον καταχωρητή λήψης (Receive buffer registers) [48]. Όταν γράφουμε δεδομένα στον SBUF τότε προσπελαύνουμε τον καταχωρητή εκπομπής (transmit buffer) όπου και κρατιούνται για σειριακή μετάδοση. Στην περίπτωση ανάγνωσης δεδομένων από τον SBUF ουσιαστικά προσπελαύνουμε τον καταχωρητή λήψης (Receive buffer).

4.4 Το σετ εντολών του 8051

Οι εντολές του 8051 είναι οι βέλτιστες για εφαρμογές ελέγχου 8-bit. Επίσης παρέχεται ένα πλήθος τρόπων αναφοράς στην μνήμη (addressing modes) για την διαχείριση μεταβλητών ενός byte, ενώ υπάρχει και ένας εκτεταμένος αριθμός εντολών για διαχείριση μεταβλητών του ενός bit, δυνατότητα πολύ χρήσιμη για εφαρμογές ελέγχου [47][48]. Οι τρόποι διευθυνσιοδότησης παραθέτονται παρακάτω.

4.4.1 Register Addressing Mode

Με αυτόν τον τρόπο διευθυνσιοδότησης μπορούμε να αναφερθούμε κατευθείαν σε έναν από τους οκτώ καταχωρητές R0 έως R7. Τους καταχωρητές αυτούς μπορούμε να τους χρησιμοποιήσουμε για διάφορες αριθμητικές και λογικές πράξεις [47][48]. Για παράδειγμα:

```
ADD A, R0
SUBB A, R1
MOV A, R5
```

4.4.2 Direct Byte Addressing

Στην περίπτωση του Direct Byte Addressing η διεύθυνση προορισμού καθορίζεται χρησιμοποιώντας 8-bit δεδομένα στην εντολή. Η επιλογή Direct Byte Addressing μπορεί να χρησιμοποιηθεί μόνο στην εσωτερική μνήμη και συγκεκριμένα στις 128 χαμηλότερες διευθύνσεις της μνήμης. Ακολουθεί μια χρήση του Direct Byte Addressing στο παρακάτω παράδειγμα[48].

```
MOV A, 10H
ADD A, 30H
```

Στο παράδειγμα αυτό προσθέτουμε τις δύο θέσεις της RAM, την θέση 10 και την 30. Ακόμη, υπάρχει η δυνατότητα πρόσθεσης δεδομένων εισόδου P1 σε δεδομένα της εισόδου P2 (port P2).

```
MOV A, P1
MOV A, P2
```

4.4.3 Register Indirect - Addressing

Ο τρόπος αυτός διευθυνσιοδότησης είναι χρήσιμος όταν θέλουμε να διαχειριστούμε μεταβλητές των οποίων οι τιμές καθορίζονται, υπολογίζονται ή διαμορφώνονται καθώς το πρόγραμμα τρέχει. Στην περίπτωση αυτή η διεύθυνση προορισμού δίνεται σε έναν καταχωρητή και όχι άμεσα από τη διεύθυνση της εσωτερικής μνήμης. Χρησιμοποιώντας έμμεση διευθυνσιοδότηση υπάρχει πρόσβαση σε εσωτερικές και εξωτερικές διευθύνσεις, και δηλώνεται με το σύμβολο '@' μπροστά από τον καταχωρητή R0 ή R1[47][48].

MOV A, @R0

Στο παράδειγμα αυτό η τιμή στο R0 θεωρείται ως μια διεύθυνση, η οποία κρατά τα δεδομένα που πρέπει να μεταφερθούν στον συσσωρευτή.

4.4.4 Immediate Addressing

Αυτή η λειτουργία διευθυνσιοδότησης ονομάζεται άμεση γιατί μεταφέρει 8-bit δεδομένα κατευθείαν στον καταχωρητή. Αυτό επιτυγχάνεται χρησιμοποιώντας το σύμβολο '#'στην γλώσσα προγραμματισμού assembly[48]. Για παράδειγμα:

MOV A, #20H

όπου το #20H είναι δεδομένο (data) και αποθηκεύεται στον συσσωρευτή A.

4.4.5 Αριθμητικές εντολές

Οι αριθμητικές εντολές εμφανίζονται στο παρακάτω σχήμα 4.13**:**

Instruction Set Summary

Mnemonic		Description	Byte	Cycle
Arithmetic Operations				
ADD	A,Rn	Add register to accumulator	1	1
ADD	A,direct	Add direct byte to accumulator	2	1
ADD	A, @Ri	Add indirect RAM to accumulator	1	1
ADD	A,#data	Add immediate data to accumulator	2	1
ADDC	A,Rn	Add register to accumulator with carry flag	1	1
ADDC	A,direct	Add direct byte to A with carry flag	2	1
ADDC	A, @Ri	Add indirect RAM to A with carry flag	1	1
ADDC	A, #data	Add immediate data to A with carry flag	2	1
SUBB	A,Rn	Subtract register from A with borrow	1	1
SUBB	A,direct	Subtract direct byte from A with borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with borrow	1	1
SUBB	A,#data	Subtract immediate data from A with borrow	2	1
INC	A	Increment accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
DEC	A	Decrement accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
INC	DPTR	Increment data pointer	1	2
MUL	AB	Multiply A and B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal adjust accumulator	1	1

Σχήμα 4.13 - Αριθμητικές εντολές

Η εντολή ADD A , X υλοποιεί την πράξη της πρόσθεση του συσσωρευτή με X bytes[48]. Ο χρόνος εκτέλεσης της εντολής αυτής είναι 1 μs. Η εντολή SUBB υλοποιεί την πράξη της αφαίρεσης του συσσωρευτή με X bytes και στην περίπτωση αυτήν ο χρόνος εκτέλεσης είναι 1 μs. Στην περίπτωση της εντολής MUL AB, τα δεδομένα του συσσωρευτή πολλαπλασιάζονται με τα δεδομένα του καταχωρητή B και τα αποτελέσματα αποθηκεύονται στους συνεχόμενους καταχωρητές A και B. Το ίδιο ισχύει και στην εντολή DIV AB, που αφορά την διαίρεση. Ο χρόνος που χρειάζονται για την εκτέλεση των δύο αυτών εντολών είναι 4 μs έκαστη.

4.4.6 Λογικές εντολές

Οι λογικές εντολές εμφανίζονται στο παρακάτω σχήμα 4.14**:**

Logic Operations

ANL	A,Rn	AND register to accumulator	1	1
ANL	A,direct	AND direct byte to accumulator	2	1
ANL	A,@Ri	AND indirect RAM to accumulator	1	1
ANL	A,#data	AND immediate data to accumulator	2	1
ANL	direct,A	AND accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to accumulator	1	1
ORL	A,direct	OR direct byte to accumulator	2	1
ORL	A,@Ri	OR indirect RAM to accumulator	1	1
ORL	A,#data	OR immediate data to accumulator	2	1
ORL	direct,A	OR accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive OR register to accumulator	1	1
XRL	A direct	Exclusive OR direct byte to accumulator	2	1
XRL	A,@Ri	Exclusive OR indirect RAM to accumulator	1	1
XRL	A,#data	Exclusive OR immediate data to accumulator	2	1
XRL	direct,A	Exclusive OR accumulator to direct byte	2	1
XRL	direct,#data	Exclusive OR immediate data to direct byte	3	2
CLR	A	Clear accumulator	1	1
CPL	A	Complement accumulator	1	1
RL	A	Rotate accumulator left	1	1
RLC	A	Rotate accumulator left through carry	1	1
RR	A	Rotate accumulator right	1	1
RRC	A	Rotate accumulator right through carry	1	1
SWAP	A	Swap nibbles within the accumulator	1	1

Σχήμα 4.14 - Λογικές εντολές

Στον παραπάνω πίνακα εμφανίζονται οι επιτρεπτές λογικές πράξεις και ο χρόνος εκτέλεσης για την κάθε μία. Οι λογικές πράξεις μεταξύ bytes εκτελούνται bit προς bit[48].

4.4.7 Εντολές μεταφοράς δεδομένων

Οι εντολές μεταφοράς δεδομένων εμφανίζονται στο παρακάτω σχήμα 4.15**:**

Data Transfer

MOV	A,Rn	Move register to accumulator	1	1
MOV	A,direct *)	Move direct byte to accumulator	2	1
MOV	A,@Ri	Move indirect RAM to accumulator	1	1
MOV	A,#data	Move immediate data to accumulator	2	1
MOV	Rn,A	Move accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct byte	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri, #data	Move immediate data to indirect RAM	2	1
MOV	DPTR, #data16	Load data pointer with a 16-bit constant	3	2
MOVC	A,@A + DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC	A,@A + PC	Move code byte relative to PC to accumulator	1	2
MOVX	A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVX	A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX	@Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX	@DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with accumulator	1	1
XCH	A,direct	Exchange direct byte with accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with accumulator	1	1
XCHD	A,@Ri	Exchange low-order nibble indir. RAM with A	1	1

Σχήμα 4.15 - Εντολές μεταφοράς δεδομένων

Στις δύο πρώτες περιπτώσεις του MOV η μεταφορά των δεδομένων γίνονται μέσω του συσσωρευτή ενώ στην επόμενη περίπτωση του MOV <dest>, <src>

επιτρέπεται η μεταφορά δεδομένων μεταξύ δύο θέσεων της RAM ή του SFR. Αξίζει να υπενθυμιστεί ότι τα 128 bytes της μνήμης μπορούν να προσπελαστούν μόνο με indirect addressing ενώ οι SFRs μπορούν να προσπελαστούν μόνο με direct addressing[47][48].

4.4.8 Εντολές Διακλάδωσης

Οι εντολές διακλάδωσης που υπάρχουν στον επεξεργαστή 8051 εμφανίζονται στο παρακάτω σχήμα 4.16 [47]. Χωρίζονται σε δύο κατηγορίες, σε αυτές χωρίς συνθήκη και σε αυτές με συνθήκη.

Mnemonic		Description	Byte	Cycle
Program and Machine Control				
ACALL	addr11	Absolute subroutine call	2	2
LCALL	addr16	Long subroutine call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute jump	2	2
LJMP	addr16	Long iump	3	2
SJMP	rel	Short jump (relative addr.)	2	2
JMP	@A + DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if accumulator is zero	2	2
JNZ	rel	Jump if accumulator is not zero	2	2
JC	rel	Jump if carry flag is set	2	2
JNC	rel	Jump if carry flag is not set	2	2
JB	bit,rel	Jump if direct bit is set	3	2
JNB	bit,rel	Jump if direct bit is not set	3	2
JBC	bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE	A,direct,rel	Compare direct byte to A and jump if not equal	3	2
CJNE	A,#data,rel	Compare immediate to A and jump if not equal	3	2
CJNE	Rn,#data rel	Compare immed. to reg. and jump if not equal	3	2
CJNE	@Ri,#data,rel	Compare immed. to ind. and jump if not equal	3	2
DJNZ	Rn,rel	Decrement register and jump if not zero	2	2
DJNZ	direct,rel	Decrement direct byte and jump if not zero	3	2
NOP		No operation	1	1

Σχήμα 4.16 - Εντολές διακλάδωσης

ΚΕΦΑΛΑΙΟ 5

Η ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ ΕΞΟΜΟΙΩΤΗ

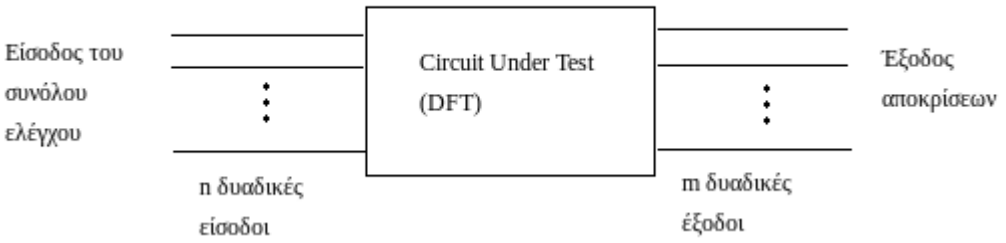
Στόχος της εργασίας είναι η ανάπτυξη ενός εξομοιωτή για Software-Based Self-Testing. Ως βάση χρησιμοποιήθηκε εξομοιωτής για ακολουθιακά κυκλώματα ο οποίος αναπτύχθηκε στο Πανεπιστήμιο του Τορίνο. Ο εξομοιωτής αυτός υποστήριζε την εξομοίωση σφαλμάτων μόνιμης τιμής σε ακολουθιακά κυκλώματα και επεκτάθηκε ώστε να υποστηρίζει τις ακόλουθες λειτουργίες:

1. Ανίχνευση σφαλμάτων μόνιμης τιμής.
2. Ανίχνευση σφαλμάτων γεφύρωσης.
3. Ανίχνευση σφαλμάτων μετάβασης.

Για την ανίχνευση των σφαλμάτων χρησιμοποιήθηκαν προγράμματα ελέγχου τα οποία χρησιμοποίησαν το σύνολο εντολών του μικροεπεξεργαστή. Τα προγράμματα ελέγχου αποθηκεύονται στη μνήμη εντολών (ROM) και εκτελούνται με στόχο να παράξουν και να εφαρμόσουν στις μονάδες του επεξεργαστή λογικές τιμές οι οποίες παίζουν τον ρόλο διανυσμάτων ελέγχου. Ο έλεγχος των αποκρίσεων των διανυσμάτων ελέγχου αποτελεί τμήμα των προγραμμάτων ελέγχου με τις αποκρίσεις να συλλέγονται στη μνήμη δεδομένων όπου και αξιολογούνται για την ορθότητά τους. Ως μνήμη δεδομένων χρησιμοποιήθηκε η RAM και για την αξιολόγηση της ορθότητας των προγραμμάτων ελέγχονται τυχόν διαφοροποιήσεις στις τιμές των κελιών της κύριας μνήμης. Υποστηρίζονται εξομοιώσεις για σφάλματα μόνιμης τιμής, σφάλματα μετάβασης και σφάλματα γεφύρωσης.

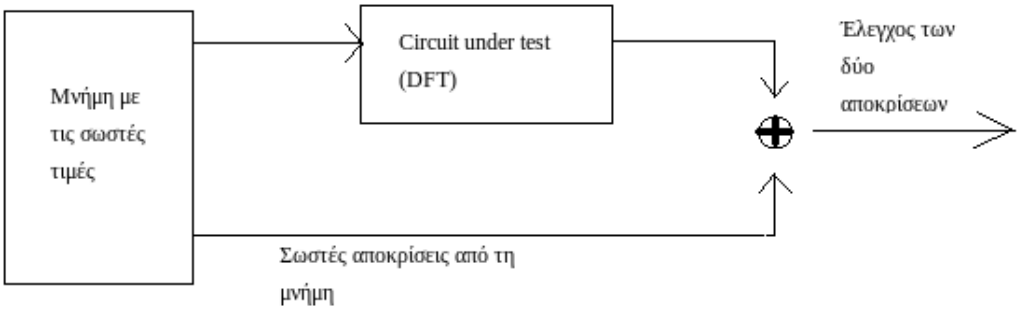
5.1 Προσομοίωση σφαλμάτων

Κάθε έλεγχος περιλαμβάνει μια διαδικασία προσομοίωσης όπως φαίνεται στο παρακάτω σχήμα 5.1: Στα αρχικά στάδια του εξομοιωτή υπήρχε η δυνατότητα να πραγματοποιεί εξομοίωση σφαλμάτων σε ένα ψηφιακό κύκλωμα. Στην συγκεκριμένη περίπτωση εφαρμόζεται ένα διάνυσμα στις εισόδους του κυκλώματος και μελετούνται οι αποκρίσεις των εξόδων του.



Σχήμα 5.1 - Προσομοίωση σφαλμάτων

Σε κάθε βήμα της προσομοίωσης, κάθε απόκριση δοκιμής (test response) ελέγχεται. Ο έλεγχος των αποκρίσεων απαιτεί την προγενέστερη γνώση των σωστών αποκρίσεων και την σύγκριση αυτών με τις αποκρίσεις του κυκλώματος υπό δοκιμή. Αν το κύκλωμα αυτό είναι σχετικά μικρό τότε ο έλεγχος αυτός είναι εύκολος και μπορεί να γίνει εξαντλητικά η χρήση των διανυσμάτων εισόδων. Στην περίπτωση όμως που το κύκλωμα υπό δοκιμή είναι μεγάλο τότε πρέπει να χρησιμοποιείται μνήμη για την αποθήκευση των σωστών αποκρίσεων. Στην συνέχεια οι αποκρίσεις του κυκλώματος υπό δοκιμή ελέγχονται με τις σωστές που βρίσκονται στην μνήμη, όπως φαίνεται στο σχήμα 5.2.



Σχήμα 5.2 - Προσομοίωση σφαλμάτων με τη χρήση μνήμης

Όσο αυξάνεται το μέγεθος των κυκλωμάτων υπό δοκιμή και το πλήθος των εισόδων του τόσο πιο δύσκολη είναι η χρήση της εξαντλητικής χρήσης των διανυσμάτων εισόδων. Έτσι επειδή ο χρόνος ελέγχου είναι πολύ μεγάλος έχει σαν συνέπεια να αυξάνεται το κόστος του ελέγχου. Επομένως ο στόχος είναι να καθοριστεί ένας μη εξαντλητικός έλεγχος για το κύκλωμα, που θα το εξετάζει επαρκώς μέσα σε αποδεκτό χρόνο και με μικρό κόστος. Έτσι θα πρέπει να χρησιμοποιείται ένα σύνολο ελέγχου που θα βασίζεται σε ένα μοντέλο ελαττωμάτων. Ωστόσο, θα πρέπει υπολογιστεί πόσο αποτελεσματικό είναι αυτό το σύνολο ελέγχου για την ανίχνευση σφαλμάτων στο υπό έλεγχο κύκλωμα. Αυτό απαιτεί μια διαδικασία που ονομάζεται προσομοίωση σφαλμάτων (Fault Simulation)[49][50].

Οι πρώτες μέθοδοι προσομοίωσης σφαλμάτων (fault simulation methods) θεωρούσαν για k πιθανά σφάλματα, k πιθανά μοντέλα του κυκλώματος, όπου κάθε μοντέλο περιείχε ένα μόνο σφάλμα. Έτσι το σύνολο ελέγχου εφαρμοζόταν σε κάθε μοντέλο κυκλώματος. Έστω l το πλήθος των σφαλμάτων που δεν ανιχνεύονται, τότε το ποσοστό κάλυψης είναι k/l. Αυτή η διαδικασία προσομοίωσης σφαλμάτων λόγω των μεγάλων κυκλωμάτων και των πολλών εισόδων, ανάγκασε την δημιουργία νέων μεθόδων. Οι νέες αυτές διαδικασίες προσομοίωσης σφαλμάτων μπορούν να εξομοιώσουν παράλληλα πολλά σφάλματα, επιταχύνοντας τη διαδικασία προσομοίωσης. Οι μέθοδοι είναι οι:

- Συμπερασματική προσομοίωση σφαλμάτων (deductive fault simulation)
- Παράλληλη προσομοίωση σφαλμάτων (parallel fault simulation)
- Ταυτόχρονη προσομοίωση σφαλμάτων (concurrent fault simulation)

Στη *συμπερασματική προσομοίωση σφαλμάτων*, γίνεται χρήση μια λίστας σφαλμάτων (fault list), η οποία αποτελεί και τη σχέση μεταξύ της εισόδου και της εξόδου. Η λίστα σφαλμάτων σχετίζεται γραμμές μέσα στο κύκλωμα. Επιπλέον, για κάθε διάνυσμα ελέγχου, οι λίστες σφαλμάτων διαδίδονται σειριακά στο κύκλωμα μέσω των εισόδων του. Ο χρόνος εκτέλεσης είναι αρκετά μεγάλος αλλά έχει το πλεονέκτημα ότι με κάθε πέρασμα μπορεί να ανιχνευθούν περισσότερα σφάλματα.

Στην περίπτωση της *παράλληλης προσομοίωσης*, χρησιμοποιείται ένα πλήθος n δυαδικών ψηφίων, για να καθορίσει τη λογική τιμή σε ένα κόμβο, όταν σε αυτόν δεν υπάρχουν σφάλματα. Στην περίπτωση ύπαρξης σφαλμάτων χρησιμοποιείται πλήθος n-1 δυαδικών ψηφίων για τον καθορισμό της τιμής στον κόμβο. Το n λαμβάνει τιμές όπως n=8, 16 ή 32. Το πλεονέκτημα της μεθόδου αυτής είναι ότι επιτρέπει την ταυτόχρονη προσομοίωση κάθε διανύσματος ελέγχου στα n αντίγραφα, επιταχύνοντας τη διαδικασία περίπου κατά n φορές. Ακόμη ο χρόνος εκτέλεσης της μεθόδου αυτής είναι πιο γρήγορος σε σχέση με την συμπερασματικη προσομοίωση σφαλμάτων.

Τέλος στην περίπτωση της *ταυτόχρονης προσομοίωσης σφαλμάτων*, κάθε γραμμή του κυκλώματος περιέχει μια συνοπτική λίστα σφαλμάτων για κάθε διάνυσμα ελέγχου. Η λίστα αυτή περιέχει ένα συγκεκριμένο σφάλμα συν τα προηγούμενα σφάλματα που διαδίδονται μέσω της γραμμής αυτής. Στην περίπτωση που ένα σφάλμα σε μία γραμμή παράγει μια απόκριση ίδια με την σωστή απόκριση σε αυτήν τη γραμμή, τότε διαγράφεται από τον κατάλογο σφαλμάτων. Η διαδικασία αυτή αποφεύγει την πολυπλοκότητα της συμπερασματικής μεθόδου προσομοίωσης σφαλμάτων και διατηρεί το πλεονέκτημα της παράλληλης προσομοίωσης. Και για τις τρεις περιπτώσεις υπάρχουν λεπτομέρειες στη βιβλιογραφία [32] [44].

Στην αρχική έκδοση του εξομοιωτή χρησιμοποιήθηκε η ταυτόχρονη προσομοίωση σφαλμάτων με λίστα μεγέθους 32 σφαλμάτων. Εφαρμόστηκε ελεύθερη από σφάλματα λειτουργία και στη συνέχεια εσφαλμένη λειτουργία. Έπειτα, οι τιμές της λίστας σφαλμάτων αξιολογούνται σε ένα σημείο. Συγκεκριμένα, θεωρούνται ανιχνεύσιμα τα σφάλματα που οι τιμές τους διαφέρουν από αυτές στην ελεύθερη από σφάλματα λειτουργία του κυκλώματος. Σχετικά με τον καθορισμό των σημείων παρατήρησης, προσφέρει τη δυνατότητα στο χρήστη να επιλέξει ποια θα είναι αυτά. Ειδικότερα, ως σημεία παρατήρησης μπορούν να επιλεγούν τα διάφορα στοιχεία μνήμης του επεξεργαστή. Επιπλέον, μπορούν να επιλεγούν ως σημεία παρατήρησης μια ομάδα των διαφόρων στοιχείων μνήμης.

Στην εργασία αυτή, η επέκταση που πραγματοποιήθηκε ήταν ότι στην περίπτωση ενός επεξεργαστή δεν είναι δυνατή η άντληση αποκρίσεων από ενδιάμεσα σημεία παρατήρησης. Ακόμη λόγω μη ύπαρξης εξόδων σε έναν επεξεργαστή δεν είναι δυνατή η συλλογή αποκρίσεων. Έτσι, χρησιμοποιήθηκαν σαν σημεία παρατήρησης τα κελιά της κύριας μνήμης. Με αυτόν τον τρόπο ο εξομοιωτής έχει πια την δυνατότητα ανίχνευσης σφαλμάτων, συγκρίνοντας τις αποκρίσεις των εσφαλμένων διανυσμάτων στην κύρια μνήμη με αυτές της στην περίπτωση της ορθής λειτουργίας. Βέβαια, για να φτάσουν δεδομένα στην κύρια μνήμη θα πρέπει να δοθούν στον επεξεργαστή κατάλληλες εντολές που μεταφέρουν δεδομένα μέσω του διαύλου δεδομένων (data bus), στον οποιό έχει πρόσβαση η κύρια μνήμη.

Τέλος σημαντική ιδιότητα του εξομοιωτή είναι η υποστήριξη τριαδική λογικής. Συγκεκριμένα, πέρα από τις κλασσικές δύο τιμές 0, 1, ο εξομοιωτής μπορεί να διαχειριστεί και την άγνωστη τιμή X. Η άγνωστη τιμή X δηλώνει κατάσταση που μπορεί να προκύψει όταν δεν είναι σαφής η τιμή σε μία γραμμή του κυκλώματος. Ένα παράδειγμα αποτελεί η ανάγνωση μίας θέσεως μνήμης η οποία δεν έχει αρχικοποιηθεί σε κάποια αρχική τιμή.

Εξομοίωση δύο τιμών (0, 1): Η ύπαρξη δύο τιμών αναπαρίσταται με ένα bit πληροφορίας. Για την αναπαράσταση της αληθής κατάστασης χρησιμοποιείται το -1 και για την αναπαράσταση της ψευδούς το 0.

Εξομοίωση τριών τιμών (0, 1 και X): Για την αναπαράσταση μίας λογικής τιμής απαιτούνται δυο bit. Για την αναπαράσταση της αληθούς κατάστασης ‘1’ χρησιμοποιείται το ζευγάρι bit (0,-1), για την αναπαράσταση της ψευδούς κατάστασης ‘0’ το (-1,0) και για την άγνωστη κατάσταση ‘X’ το (0,0).

Έτσι ανάλογα την κατηγορία που χρησιμοποιείται ένα σφάλμα θεωρείται ανιχνεύσιμο όπως φαίνεται στους παρακάτω πίνακες**:**

Αποτίμηση λογικής τιμής	Εσφαλμένη τιμή (2 - τιμές)	Αποτέλεσμα
0	-1	Ανίχνευση σφάλματος
-1	0	Ανίχνευση σφάλματος

Πίνακας 5.1 - Δυαδική λογική

Αποτίμηση λογικής τιμής	Εσφαλμένη τιμή (3 - τιμές)	Αποτέλεσμα
(0 , 0)	(0, -1) ή (-1, 0)	Ανίχνευση σφάλματος
(0, -1)	(-1, 0) ή (0, 0)	Ανίχνευση σφάλματος
(-1, 0)	(0, -1) ή (0, 0)	Ανίχνευση σφάλματος

Πίνακας 5.2 - Τριαδική λογική

Στη συνέχεια περιγράφεται αναλυτικά η λειτουργία του εξομοιωτή.

5.2 Η λειτουργία του εξομοιωτή

Ο εξομοιωτής που κατασκευάστηκε από το Πολυτεχνείο του Τορίνο, είχε σαν στόχο την ανίχνευση σφαλμάτων σε απλά ακολουθιακά κυκλώματα. Στην περίπτωση αυτή ο εξομοιωτής δέχεται σαν είσοδο ένα ακολουθιακό κύκλωμα, στις εισόδους του οποίου εφαρμόζει ένα συγκεκριμένο διάνυσμα για να καταγράψει τις αποκρίσεις στις εξόδους του (ελεύθερη από σφάλματα λειτουργία). Στη συνέχεια, ο εξομοιωτής δέχεται σαν είσοδο ένα αρχείο που περιέχει τα σφάλματα μόνιμης τιμής τα οποία ομαδοποιούνται σε λίστες των 32 σφαλμάτων η κάθεμία και προστίθενται στο κύκλωμα. Αν και η εξομοίωση των σφαλμάτων γίνεται παράλληλα για όλη την ομάδα, η παρουσία κάθε σφάλματος δεν επηρεάζει την συμπεριφορά του κυκλώματος για τα υπόλοιπα. Αφού εφαρμοστεί ένα διάνυσμα στις εισόδους του ακολουθιακού κυκλώματος, εκτελείται εξομοίωση υπό την παρουσία της λίστας των 32 σφαλμάτων. Στο τέλος της εσφαλμένης λειτουργίας στις εξόδους του ακολουθιακού κυκλώματος, εμφανίζεται το διάνυσμα εξόδου μεγέθους 32 bit. Κάθε bit από το διάνυσμα αυτό αφορά την απόκριση του κυκλώματος υπό την παρουσία του αντίστοιχου σφάλματος. Έτσι, ελέγχοντας το διάνυσμα εξόδου στην εσφαλμένη λειτουργία με το διάνυσμα εξόδου στην ελεύθερη από σφάλματα λειτουργία, τα bit που διαφέρουν υποδεικνύουν τα σφάλματα που έχουν προκαλέσει αλλαγή στην συμπεριφορά του κυκλώματος. Όταν το διάνυσμα αφορά μία έξοδο παρατήρησης τότε τα αντίστοιχα σφάλματα θεωρούνται ανιχνευμένα.

Στη συνέχεια προσθέθηκε η δυνατότητα στον εξομοιωτή να δέχεται μια περιγραφή ενός πιο σύνθετου κυκλώματος όπως είναι ο επεξεργαστής. Για να επιτευχθεί

αυτό ο εξομοιωτής δημιουργεί πίνακες στους οποίους διατηρεί τις τιμές των δομών του επεξεργαστή. Στη συνέχεια, όπως και στην περίπτωση των ακολουθιακών κυκλωμάτων έτσι και εδώ δέχεται ένα αρχείο που περιέχει σφάλματα μόνιμης τιμής που αφορούν διάφορα σφάλματα των διασυνδέσων των δομών του επεξεργαστή. Η διαδικασία στη συνέχεια είναι ακριβώς ίδια με αυτήν στα ακολουθιακά κυκλώματα αλλά στην περίπτωση αυτή ο καθορισμός των σημείων επιτήρησης των αποκρίσεων των διανυσμάτων εισόδου καθορίζονται από το χρήστη. Για παράδειγμα, τέτοια σημεία επιτήρησης μπορεί να είναι κάποιοι καταχωρητές στους οποίους φθάνουν αποκρίσεις του επεξεργαστή. Έτσι, στην περίπτωση που αποκρίσεις του επεξεργαστή φτάνουν στους καταχωρητές αυτούς γίνονται σύγκριση με τις αντίστοιχες τιμές των καταχωρητών στην ελεύθερη από σφάλματα λειτουργία και κρίνεται αν ένα σφάλμα είναι ανιχνεύσιμο ή όχι. Για να θεωρηθεί ανιχνεύσιμο ένα σφάλμα θα πρέπει οι τιμές στα σημεία παρατήρησης κατά την ελεύθερη από σφάλματα λειτουργία και κατά την εσφαλμένη λειτουργία να διαφέρουν.

Μία επιπλέον δυνατότητα που προστέθηκε στον εξομοιωτή είναι η ανίχνευση σφαλμάτων χρησιμοποιώντας ως μέσο παρατήρησης των αποκρίσεων τη μνήμη RAM. Για να επιτευχθεί αυτό ο εξομοιωτής διατηρεί σε πίνακα τα δεδομένα της μνήμης RAM τον οποίο ανανεώνει σε κάθε περίπτωση που υπάρχει διαφοροποίησή τους υπό την παρουσία κάποιου σφάλματος. Ειδικότερα κατά την έναρξη της εξομοίωσης, η πρώτη εντολή της μνήμης ROM αποκωδικοποιείται και στη συνέχεια εκτελείται ελεύθερη από σφάλματα εξομοίωση με τις τιμές της μνήμης RAM να αποθηκεύονται σε πίνακα. Έπειτα πραγματοποιείται εξομοίωση υπό την παρουσία σφάλματος και αποθηκεύονται οι νέες τιμές της μνήμης RAM σε πίνακα. Το ίδιο επαναλαμβάνεται για κάθε εντολή του προγράμματος που αποθηκεύεται στη μνήμη ROM. Στο τέλος, κατά την ολοκλήρωση της εξομοίωσης διατηρούνται όλοι οι πίνακες με τα στοιχεία από την μνήμη RAM που έχουν αποθηκευτεί κατά τη διάρκεια της εξομοίωσης. Δηλαδή, διατηρούνται οι πίνακες από τις εξομοιώσεις ελεύθερες από σφάλματα και τις εξομοιώσεις με την παρουσία σφαλμάτων. Λόγω του μεγάλου όγκου δεδομένων, στην ελεύθερη από σφάλματα λειτουργία διατηρείται ένας πίνακας με τις τιμές της μνήμης RAM και στην εσφαλμένη λειτουργία αποθηκεύονται μόνο οι διαφοροποιήσεις του προηγούμενου πίνακα σε μια λίστα. Έτσι με αυτόν τον τρόπο μειώνεται ο όγκος πληροφορίας που αποθηκεύεται. Τέλος στην περίπτωση, που οι δύο πίνακες διαφέρουν σε ένα τουλάχιστον δεδομένο τους τότε το σφάλμα θεωρείται ανιχνεύσιμο. Ακόμη προσθέθηκε η δυνατότητα ο εξομοιωτής να ανιχνεύει εκτός από σφάλματα μόνιμης τιμής και σφάλματα βραχυκύκλωσης και

μεταβαστικά σφάλματα. Η ανάλυση και οι αλγόριθμοι υλοποίησης αυτών αναλύεται παρακάτω.

5.3 Η δομή του εξομοιωτή

Κατά την αρχικοποίηση του εξομοιωτή έχει υλοποιηθεί μια συνάρτηση create("core.edf"). Η συνάρτηση αυτή είναι υπεύθυνη για την αναπαράσταση της δομής του επεξεργαστή που θα δοθεί ως είσοδος στον εξομοιωτή. Το αρχείο εισόδου είναι τύπου edif, στο οποίο αναγράφονται όλες οι μονάδες του επεξεργαστή αλλά και οι συνδέσεις μεταξύ τους. Ένα edif αρχείο μπορεί να παραχθεί από μια περιγραφή ενός επεξεργαστή στην γλώσσα VHDL ή Verilog χρησιμοποιώντας εργαλεία CAD όπως το Synopsys. Στη συνέχεια η συνάρτηση create αφαιρεί τυχόν σχόλια και διάφορες λεπτομέρειες του edif αρχείου που δεν σχετίζονται με την περιγραφή του επεξεργαστή (parsing) και ξεκινά την συλλογή των διαφόρων μονάδων του επεξεργαστή. Στο πλαίσιο αυτό δημιουργούνται μεταβλητές για τα στοιχεία του επεξεργαστή και αποθηκεύονται επίσης οι συνδέσεις μεταξύ τους. Τέλος κατασκευάζει το netlist του αντίστοιχου επεξεργαστή με τα στοιχεία που έχει συλλέξει από το edif αρχείο.

Αφού πραγματοποιηθεί η αναπαράσταση της δομής του επεξεργαστή, ο εξομοιωτής διαβάζει από την λίστα σφαλμάτων τα σφάλματα που θα στοχοποιηθούν. Η συνάρτηση που είναι υπεύθυνη για τη λειτουργία αυτή είναι η create_fau(). Τα σφάλματα ομαδοποιούνται σε ομάδες των 32 και μέσω της προηγούμενης συνάρτησης προστίθενται στο κύκλωμα του επεξεργαστή για παράλληλη επεξεργασία. Η δομή και η ανάλυση της λίστας σφαλμάτων περιγράφεται αναλυτικά στην ενότητα 4.7.

Αφού έχουν ολοκληρωθεί τα παραπάνω βήματα, στη συνέχεια χρησιμοποιείται ένα αρχείο "input.inp" που είναι υπεύθυνο για την αρχικοποίηση των καταχωρητών του επεξεργαστή. Στη συνέχεια εισάγεται το αρχείο με όνομα "program.rom" που περιέχει το πρόγραμμα με το οποίο θα ελεγχθεί η λειτουργία του επεξεργαστή. Κάθε γραμμή του αρχείου αυτού μπορεί να δηλώνει μια εντολή ή κάποια δεδομένα. Η ανάγνωση και ο καθορισμός κάθε γραμμής του αρχείου αυτού πραγματοποιείται μέσω της συνάρτησης callback(). Η συνάρτηση αυτή είναι υπεύθυνη να αναγνωρίσει ποιά εντολή αφορά κάθε γραμμή του αρχείου "program.rom". Συγκεκριμένα:

1. Αναγνωρίζει την εντολή
2. Αναγνωρίζει τα δεδομένα που αποθηκεύονται σε κάποιον καταχωρητή

3. Ανάλογα με την εντολή αποφασίζει για την μετάβαση του δείκτη του προγράμματος (program counter), για την εκτέλεση της επόμενης εντολής.

Στην συνέχεια, πριν ξεκινήσει η βασική λειτουργία του εξομοιωτή αρχικοποιούνται οι τιμές των καταχωρητών σε μηδέν ή ένα, στην δυαδική λογική και στην περίπτωση της τριαδική λογικής, χρησιμοποιείται και η άγνωστη κατάσταση. Ακόμη αφαιρείται το ρολόι του κυκλώματος και δεν λαμβάνεται υπόψην στην εκτέλεση του εξομοιωτή. Επιπλέον, καθ'όλη τη διάρκεια εκτέλεσης των συναρτήσεων πραγματοποιούνται έλεγχοι για την ορθότητά τους.

Έπειτα, ξεκινάει η κύρια συνάρτηση του εξομοιωτή “Simulation()”. Η συνάρτηση αυτή είναι υπεύθυνη για εκτέλεση της ορθής και της εσφαλμένης λειτουργία του κυκλώματος. Συγκεκριμένα, χωρίζεται σε δύο υποσυναρτήσεις:

- “RefSimulation()”
- “FaultSimulation()”

Η συνάρτηση “RefSimulation()” ξεκινά ενημερώνοντας τους καταχωρητές του επεξεργαστή με τις τιμές που έχουν προκύψει από την αποτίμηση της κάθε εντολής του προγράμματος. Απαραίτητη εσωτερική συναρτήση της “RefSimulation” είναι η “EvalRefGate”. Η συνάρτηση αυτή υλοποιεί λογικές και αριθμητικές πράξεις ανάλογα με την εσωτερική συνδεσμολογία του επεξεργαστή. Συγκεκριμένα, κάθε εντολή από τη μνήμη προγράμματος (ROM) παράγει ένα διάνυσμα σαν είσοδο για τον επεξεργαστή. Με την ολοκλήρωση της κάθε εντολής της μνήμης προγράμματος (ROM) ενημερώνεται ένας πίνακας (free_IRAM) μέσω της συνάρτησης “update_iram()”. Ο πίνακας αυτός είναι απαραίτητος για να υπάρχουν αποθηκευμένες οι ορθές αποκρίσεις από την εκτέλεση των εντολών της ROM.

Στην συνέχεια η συνάρτηση “FaultSimulation()” ξεκινά την εκτέλεση της ίδιας εντολής από την ROM. Ενημερώνει τους καταχωρητές με τις τιμές που θα πρέπει να λάβουν και προσθέτει το σφάλμα που εξομοιώνεται. Στη συνέχεια η συνάρτηση “EvalFGate” εκτελεί την ίδια λειτουργία με την “EvalRefGate”, πραγματοποιώντας τις πράξεις σε κάθε πύλη του επεξεργαστή. Έπειτα, η συνάρτηση “EvalFault()”, τροποποιεί την τιμή της γραμμής που επηρεάζει το σφάλμα. Συγκεκριμένα, χρησιμοποιείται μια μάσκα-OR όταν το σφάλμα είναι το ‘1’ και μια μάσκα-AND, όταν το σφάλμα είναι το ‘0’. Στην πρώτη περίπτωση ανεξάρτητα της τιμής της γραμμής, η γραμμή αποκτά την τιμή ‘1’ ενώ στην δεύτερη περίπτωση το ‘0’. Στο τέλος, κατά την ολοκλήρωση της

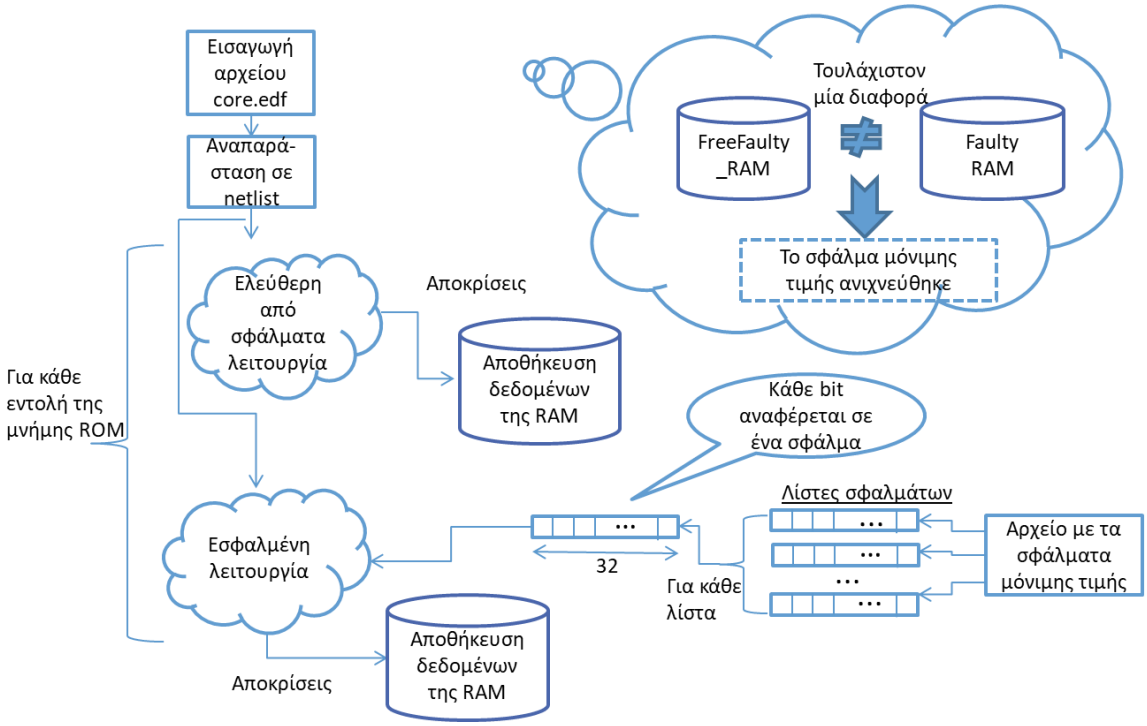
συνάρτησης “FaultSimulation()”, οι τιμές των αποκρίσεων αποθηκεύονται σε έναν πίνακα (Faulty_IRAM).

Οι προηγούμενες συναρτήσεις εκτελούνται για κάθε εντολή της μνήμης δεδομένων και σε κάθε επανάληψη ενημερώνονται οι δύο πίνακες (free_IRAM και Faulty_IRAM). Έτσι, έχοντας τις τιμές από την ορθή και την εσφαλμένη προσομοίωση αποθηκευμένες σε πίνακες, υπάρχει η δυνατότητα σύγκρισης τους. Στην περίπτωση που οι δύο πίνακες διαφέρουν μεταξύ τους τότε ο εξομοιωτής συμπεραίνει ότι το σφάλμα που προκάλεσε έστω μία αλλαγή στην κύρια μνήμη θεωρείται ανιχνεύσιμο.

5.4 Τεχνική ανίχνευσης σφαλμάτων μόνιμης τιμής

Μία από τις τρεις λειτουργίες του εξομοιωτή είναι η ικανότητά του να ανιχνεύει σφάλματα μόνιμης τιμής. Για την περίπτωση ανίχνευσης σφαλμάτων μόνιμης τιμής στον εξομοιωτή φορτώνεται σαν είσοδο η περιγραφή του επεξεργαστή σε μορφή EDIF. Έπειτα σαν είσοδο ο εξομοιωτής δέχεται ένα αρχείο με την λίστα των σφαλμάτων μόνιμης τιμής του επεξεργαστή που θα στοχοποιηθούν. Κάθε σφάλμα της λίστας σχετίζεται με ένα πιθανό σφάλμα σε μία γραμμή του κυκλώματος του επεξεργαστή. Στη συνέχεια, φορτώνεται το πρόγραμμα με τις εντολές στην μνήμη ROM και ξεκινάει η ελεύθερη από σφάλματα λειτουργία του εξομοιωτή. Τέλος συλλέγονται οι αποκρίσεις των διανυσμάτων ελέγχου που δημιουργούνται στην μνήμη δεδομένων (RAM). Στην περίπτωση αυτή τα δεδομένα της κύριας μνήμης αποθηκεύονται στον πίνακα FreeFaulty_RAM.

Μετά την ελεύθερη από σφάλματα εξομοίωση γίνεται προσθήκη μίας ομάδος σφαλμάτων και εκτελείται η προσομοίωση της εσφαλμένης λειτουργίας. Και σε αυτή την περίπτωση συλλέγονται οι αποκρίσεις στην κύρια μνήμη. Στην φάση αυτή τα δεδομένα της κύριας μνήμης που εμφανίστηκαν κατά την εσφαλμένη λειτουργία αποθηκεύονται στον πίνακα Faulty_RAM. Τέλος, ο εξομοιωτής συγκρίνει τις τιμές των δύο πινάκων FreeFaulty_RAM και Faulty_RAM και αν παρατηρηθούν αλλαγές στους δύο πίνακες επιστρέφει ότι το σφάλμα που τις προκάλεσε ανιχνεύτηκε. Ο αλγόριθμος ανίχνευσης σφαλμάτων μόνιμης τιμής φαίνεται στο παρακάτω σχήμα:

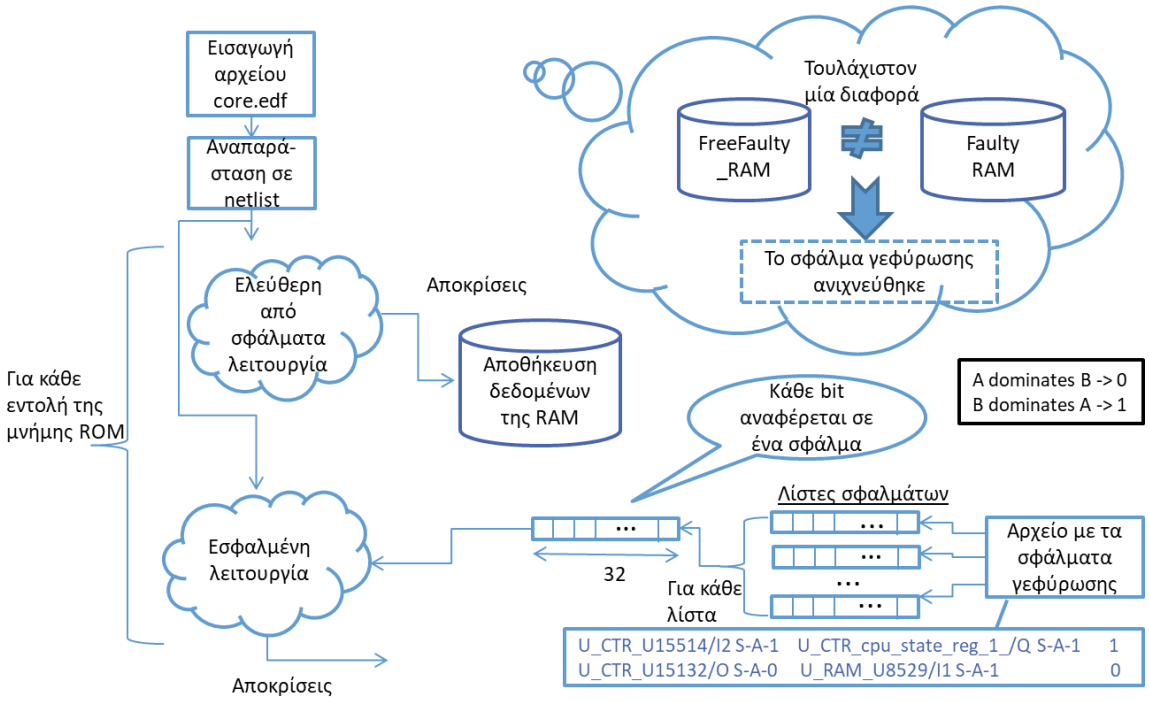


Σχήμα 5.3 - Αλγόριθμος ανίχνευσης σφαλμάτων μόνιμης τιμής

5.5 Τεχνική ανίχνευσης σφαλμάτων γεφύρωσης

Μία δεύτερη δυνατότητα του εξομοιωτή είναι η ανίχνευση σφαλμάτων γεφύρωσης και συγκεκριμένα των μοντέλων σφαλμάτων A dominates B και B dominates A. Ο αλγόριθμος που δημιουργήθηκε έχει πολλά κοινά σημεία με τον αλγόριθμο ανίχνευσης σφαλμάτων μόνιμης τιμής. Όπως και πριν, στον εξομοιωτή φορτώνεται σαν είσοδο η περιγραφή του επεξεργαστή σε μορφή EDIF. Έπειτα σαν είσοδο ο εξομοιωτής δέχεται ένα αρχείο με την λίστα των σφαλμάτων γεφύρωσης του επεξεργαστή που θα στοχοποιηθούν. Το αρχείο αυτό περιέχει ζευγάρια από γραμμές του κυκλώματος. Σε κάθε βραχυκύκλωμα αναφέρεται ένας αριθμός 0 ή 1 που δηλώνει αν το βραχυκύκλωμα αυτό είναι A dominates B ή B dominates A αντίστοιχα. Στην πρώτη περίπτωση η τιμή της γραμμής B ταυτίζεται με την τιμή της γραμμής A ενώ στην δεύτερη περίπτωση η τιμή της γραμμής A ταυτίζεται με την τιμή της γραμμής B. Κάθε σφάλμα της λίστας σχετίζεται με ένα πιθανό σφάλμα σε μία γραμμή του κυκλώματος του επεξεργαστή. Στη συνέχεια, φορτώνεται το πρόγραμμα με τις εντολές στην μνήμη ROM και ξεκινάει η ελεύθερη από σφάλματα λειτουργία του εξομοιωτή. Τέλος συλλέγονται οι αποκρίσεις των διανυσμάτων ελέγχου που δημιουργούνται στην μνήμη δεδομένων (RAM). Στην περίπτωση αυτή τα δεδομένα της κύριας μνήμης αποθηκεύονται στον πίνακα FreeFaulty_RAM.

Μετά την ελεύθερη από σφάλματα εξομοίωση γίνεται προσθήκη μίας ομάδος σφαλμάτων και εκτελείται η προσομοίωση της εσφαλμένης λειτουργίας. Και σε αυτή την περίπτωση συλλέγονται οι αποκρίσεις στην κύρια μνήμη. Στην φάση αυτή τα δεδομένα της κύριας μνήμης που εμφανίστηκαν κατά την εσφαλμένη λειτουργία αποθηκεύονται στον πίνακα Faulty_RAM. Τέλος, ο εξομοιωτής συγκρίνει τις τιμές των δύο πινάκων FreeFaulty_RAM και Faulty_RAM και αν παρατηρηθούν αλλαγές στους δύο πίνακες επιστρέφει ότι το σφάλμα που τις προκάλεσε ανιχνεύτηκε. Ο αλγόριθμος ανίχνευσης σφαλμάτων γεφύρωσης φαίνεται στο παρακάτω σχήμα:



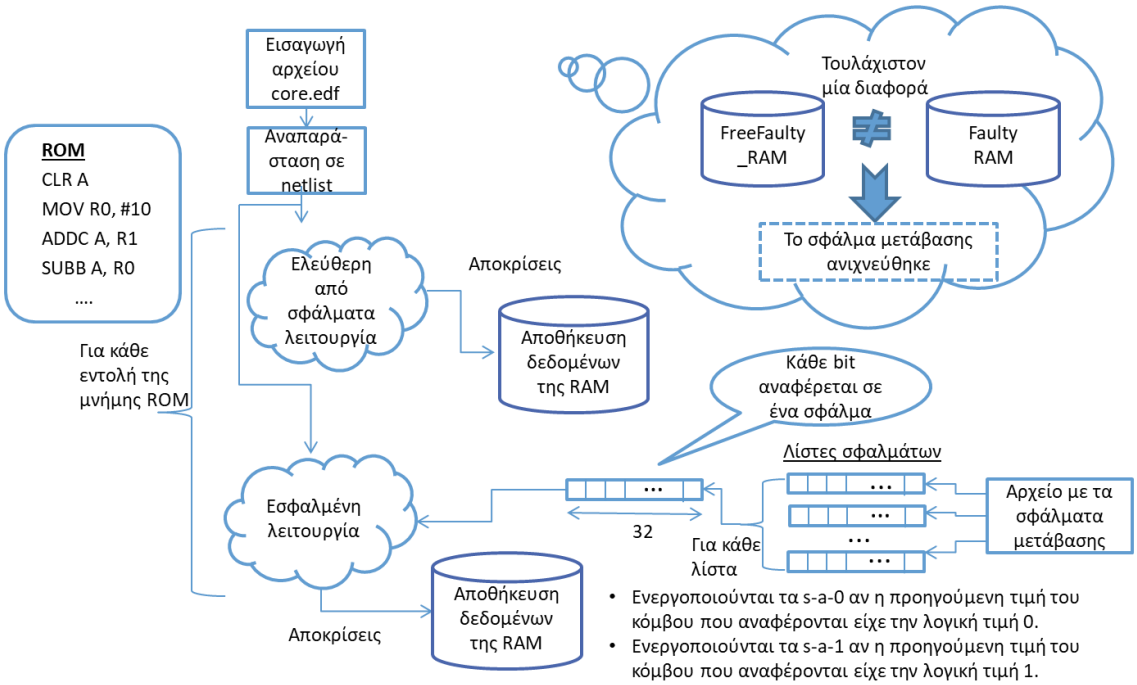
Σχήμα 5.4 - Αλγόριθμος για σφάλματα γεφύρωσης

5.6 Τεχνική ανίχνευσης σφαλμάτων μετάβασης

Η τρίτη δυνατότητα του εξομοιωτή είναι η ικανότητα ανίχνευσης σφαλμάτων μετάβασης. Συγκεκριμένα ο εξομοιωτής μπορεί να ανιχνεύσει σφάλματα μετάβασης slow-to-rise και slow-to-fall. Όπως και πριν, στον εξομοιωτή φορτώνεται σαν είσοδο η περιγραφή του επεξεργαστή σε μορφή EDIF. Για την ανίχνευση ενός σφάλματος slow-to-rise μπορεί να χρησιμοποιηθεί ένα stuck-at-0 σφάλμα. Συγκεκριμένα τοποθετείται στον κόμβο η λογική τιμή 0, και στον επόμενο κύκλο ρολογιού στοχοποιείται το σφάλμα stuck-at-0. Αν το σφάλμα ανιχνευθεί τότε ανιχνεύεται και το αντίστοιχο slow-to-rise σφάλμα. Με παρόμοιο τρόπο χρησιμοποιείται το σφάλμα stuck-at-1 για να ανιχνευθεί

ένα σφάλμα slow-to-fall.Κάθε σφάλμα της λίστας σχετίζεται με ένα πιθανό σφάλμα σε μία γραμμή του κυκλώματος του επεξεργαστή. Στη συνέχεια, φορτώνεται το πρόγραμμα με τις εντολές στην μνήμη ROM και ξεκινάει η ελεύθερη από σφάλματα λειτουργία του εξομοιωτή. Τέλος συλλέγονται οι αποκρίσεις των διανυσμάτων ελέγχου που δημιουργούνται στην μνήμη δεδομένων (RAM). Στην περίπτωση αυτή τα δεδομένα της κύριας μνήμης αποθηκεύονται στον πίνακα FreeFaulty_RAM.

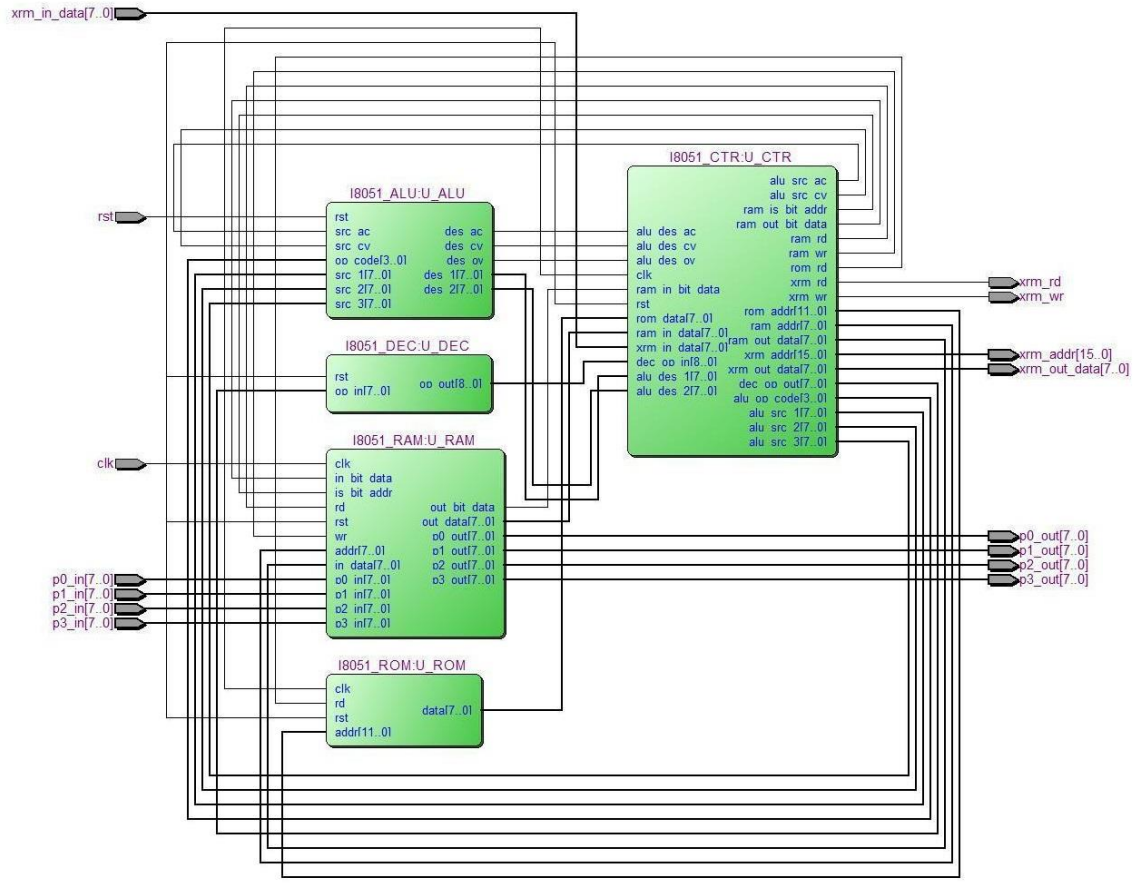
Μετά την ελεύθερη από σφάλματα εξομοίωση γίνεται προσθήκη μίας ομάδος σφαλμάτων και εκτελείται η προσομοίωση της εσφαλμένης λειτουργίας. Και σε αυτή την περίπτωση συλλέγονται οι αποκρίσεις στην κύρια μνήμη. Στην φάση αυτή τα δεδομένα της κύριας μνήμης που εμφανίστηκαν κατά την εσφαλμένη λειτουργία αποθηκεύονται στον πίνακα Faulty_RAM. Τέλος, ο εξομοιωτής συγκρίνει τις τιμές των δύο πινάκων FreeFaulty_RAM και Faulty_RAM και αν παρατηρηθούν αλλαγές στους δύο πίνακες επιστρέφει ότι το σφάλμα που τις προκάλεσε ανιχνεύτηκε. Ο αλγόριθμος ανίχνευσης σφαλμάτων μετάβασης φαίνεται στο παρακάτω σχήμα



Σχήμα 5.5 - Αλγόριθμος για ανίχνευση σφαλμάτων μετάβασης

5.7 Δομή της λίστας σφαλμάτων

Πριν αναλυθεί η δομή της λίστας σφαλμάτων που χρησιμοποιούνται από τον εξομοιωτή να παρουσιαστεί η περιγραφή του edif αρχείου του επεξεργαστή i8051. Στο παρακάτω σχήμα φαίνεται το πως είναι δομημένος ο επεξεργαστής.



Σχήμα 5.6 - Εσωτερική διάταξη του 8051

Μέσα στον επεξεργαστή 8051 υπάρχει η αριθμητική και λογική μονάδα (ALU), η μονάδα προσκόμισης και αποκωδικοποίησης εντολών (instruction fetch and decoding), η μονάδα ελέγχου (control unit), η κύρια μνήμη(RAM), η μνήμη προγράμματος (ROM) και τέλος το σύνολο των διαφόρων καταχωρητών. Επίσης ο 8051 έχει τη δυνατότητα σύνδεσης και με εξωτερικές μνήμες ROM και RAM, οι οποίες όμως δεν συμπεριλήφθηκαν στην συγκεκριμένη υλοποίηση του.

Στη συνέχεια, παρουσιάζεται η μορφή των σφαλμάτων που μοντελοποιούνται για τον επεξεργαστή i8051. Η λίστα σφαλμάτων που δέχεται ο εξομοιωτής είναι αποθηκευμένη στο αρχείο core.fau και έχει την παρακάτω δομή:

U_CTR_U15064/O S-A-1 UNDETECTED (UNTESTED)
= U_CTR_U15064/I1 S-A-0
= U_CTR_U15064/I2 S-A-0
= U_CTR_U15522/O S-A-0
= U_CTR_U15523/O S-A-0
U_CTR_U15514/I2 S-A-1 UNDETECTED (UNTESTED)
U_CTR_U15132/O S-A-0 UNDETECTED (UNTESTED)
U_CTR_U15132/I1 S-A-1 UNDETECTED (UNTESTED)
U_CTR_U15510/I1 S-A-0 UNDETECTED (UNTESTED)
= U_CTR_U15510/O S-A-1
U_CTR_cpu_state_reg_1_/Q S-A-1 UNDETECTED (UNTESTED)
U_RAM_U8526/I1 S-A-1 UNDETECTED (UNTESTED)
U_RAM_U8529/I1 S-A-1 UNDETECTED (UNTESTED)
U_RAM_U8532/I1 S-A-1 UNDETECTED (UNTESTED)

Κάθε γραμμή του αρχείου αφορά ένα σφάλμα, περιγράφοντας λεπτομέρειες σχετικά με την σύνδεση που αφορούν και την τιμή που επιβάλλουν στην σύνδεση αυτήν. Ο τρόπος γραφής κάθε σφάλματος έχει συγκεκριμένη δομή και περιγράφεται αναλυτικά παρακάτω:

Έστω το παρακάτω σφάλμα:
(= | ∅) X / Y (S-A-0 | S-A-1) (UNDETECTED (UNTESTED) | ∅)

(= | ∅) : Στην περίπτωση που υπάρχει το ‘=’ πριν από ένα σφάλμα τότε το σφάλμα αυτό δεν ενσωματώνεται στη λίστα σφαλμάτων που θα πρέπει να ελεγχθούν

X : Είναι το όνομα της μονάδας που αφορά το σφάλμα.

Y : Προσδιορίζει κάποια από τις εισόδους ή την έξοδο του X.

S-A-0 | S-A-1 : Αναφέρεται σε stuck-at-zero ή σε stuck-at-one σφάλμα.

5.8 Παραλληλοποίηση αλγορίθμων

Για την μείωση του χρόνου εξομοίωσης, ο εξομοιωτής υποστηρίζει παράλληλη εξομοίωση σφαλμάτων σε πολλαπλούς πυρήνες του συστήματος. Έτσι για κάθε εσφαλμένη λειτουργία κάθε σφάλματος συλλέγονται οι αποκρίσεις τους στη μνήμη RAM και συγκρίνονται με τις αντίστοιχες στην ελεύθερη από σφάλματα λειτουργία. Οι πρώτες προσπάθειες που έγιναν ήταν η λειτουργία του εξομοιωτή να επιμεριστεί σε 2 πυρήνες ή 4 πυρήνες. Ο τρόπος με τον οποίο επιτεύχθηκε η παραλληλοποίηση του εξομοιωτή είναι η ακόλουθη.

1) Αναγνώριση του πλήθους των πυρήνων του υπολογιστική.
2) Υλοποίηση της ελεύθερης από σφάλματα λειτουργίας.
3) Αποθήκευση των τιμών της μνήμης RAM σε πίνακα (FREE_IRAM).
4) Διάσπαση της λίστας σφαλμάτων ανάλογα τους πυρήνες του υπολογιστή και εκτέλεση της εσφαλμένης λειτουργίας σε κάθε πυρήνα.
5) Αποθήκευση των εσφαλμένων τιμών της μνήμης RAM σε πίνακα (FAULTY_IRAM).
6) Σύγκριση κάθε πίνακα FAULTY_IRAM με τον πίνακα FREE_IRAM και εξαγωγή των αποτελεσμάτων.

Το πλεονεκτήμα της παραλληλοποίησης είναι η μείωση της εκτέλεσης της εξομοίωσης. Συγκεκριμένα ο χρόνος μειώνεται σχεδόν γραμμικά με το πλήθος των επεξεργαστών του συστήματος. Οπότε ανάλογα τον αριθμό των πυρήνων που χρησιμοποιούνται για να εκτελεστεί η εξομοίωση αλλάζει ανάλογα και ο χρόνος εκτέλεσης. Συγκεκριμένα, με την χρήση πολλών πυρήνων μπορεί να επιμεριστεί η εξομοίωση της εσφαλμένης λειτουργίας σε κάθε πυρήνα. Δηλαδή, κάθε πυρήνας να πραγματοποιεί εσφαλμένη λειτουργία για διαφορετικό σφάλμα.

ΚΕΦΑΛΑΙΟ 6

ΠΕΙΡΑΜΑΤΑ

6.1 Πειράματα εξομοίωσης σφαλμάτων μόνιμης τιμής

Πείραμα 1- πρόσθεση τιμών δύο καταχωρητών.

Αρχικά χρησιμοποιούμε ένα απλό πρόγραμμα ελέγχου για να δείξουμε την λειτουργία του εξομοιωτή. Συγκεκριμένα φορτώνουμε τον συσσωρευτή A με την τιμή 06h και τον συσσωρευτή B με την τιμή 01h, εκτελούμε την πρόσθεση και αποθηκεύουμε το αποτέλεσμα στον συσσωρευτή A. Τέλος το αποτέλεσμα μεταφέρεται στην μηδενική θέση της μνήμης δεδομένων (RAM). Το πρόγραμμα σε συμβολική γλώσσα και γλώσσα μηχανής φαίνονται στον ακόλουθο πίνακα

Συμβολική Γλώσσα	Γλώσσα Μηχανής

mov a, #06h
mov b, #01h
add a,b
mov 0h, a
	01110100
	00000110
	01110101
	11110000
	00000001
	00100101
	11110000
	11110101
	00000000

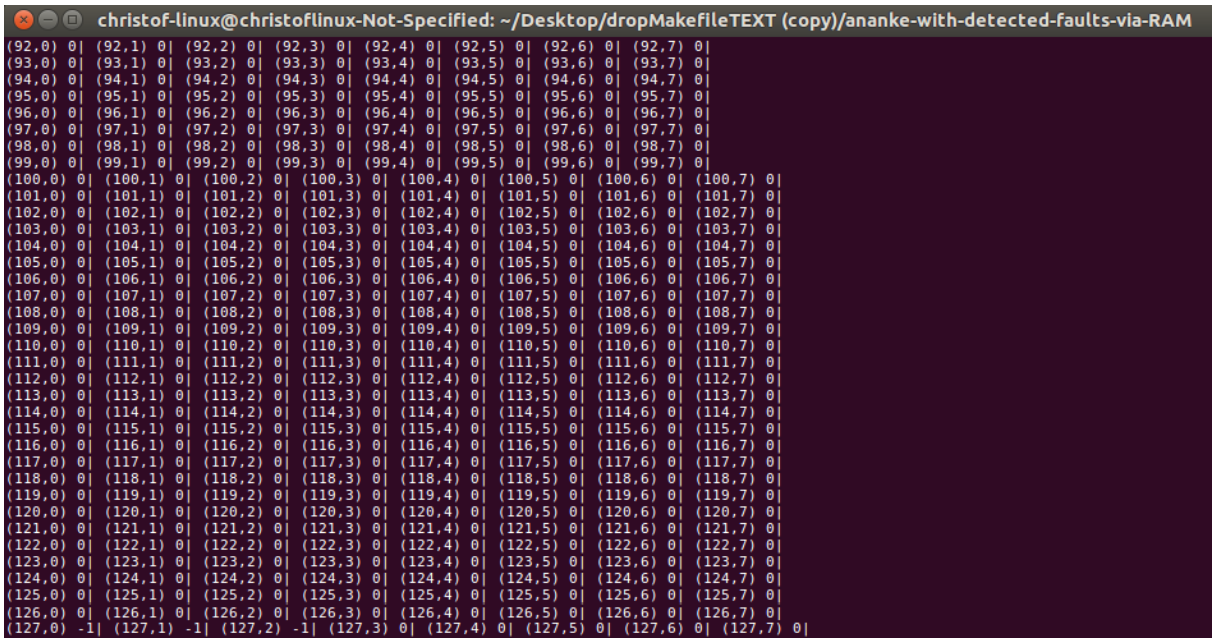
Πίνακας 6.1 - Γλώσσα μηχανής 1ου πειράματος

Το πρόγραμμα αυτό ελέγχει την παρουσία σφαλμάτων μόνιμης τιμής στις ακόλουθες μονάδες του επεξεργαστή:

- Συσσωρευτές A, B

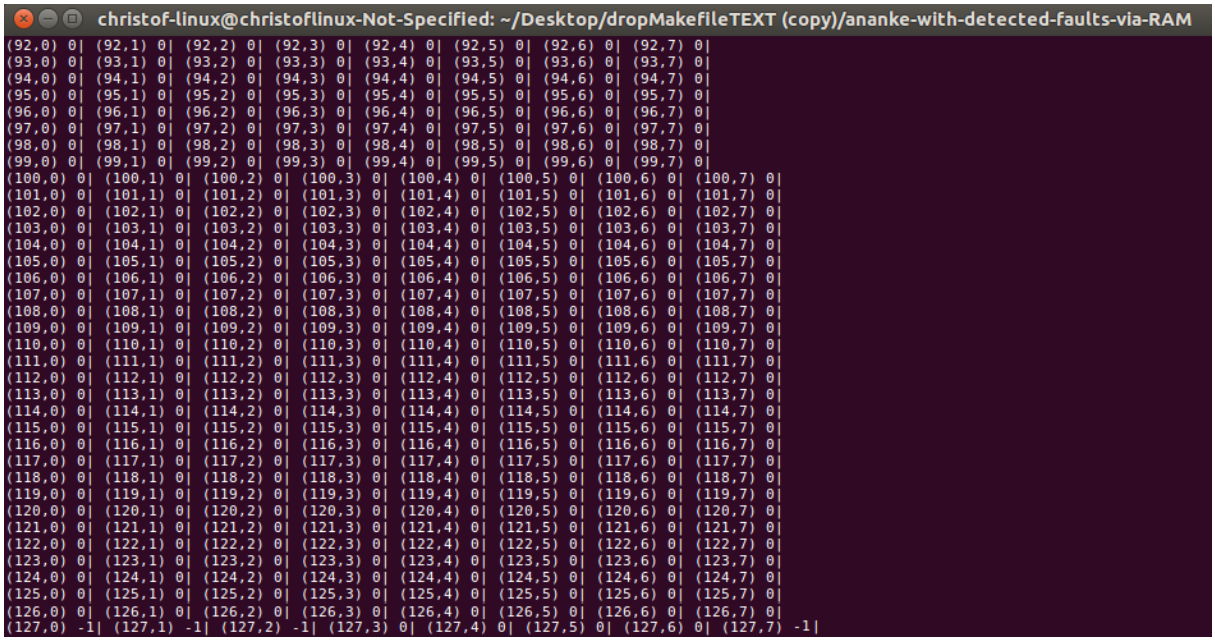
- Αριθμητική λογική μονάδα
- Μνήμη RAM
- Μονάδα ελέγχου
- Αποκωδικοποιητής εντολών

Αφού πρώτα εισάγουμε τον επεξεργαστή 8051 στον εξομοιωτή, εκτελούμε εξομοίωση του προγράμματος ελεύθερη από σφάλματα ώστε να καταγραφούν οι αναμενόμενες τιμές στις γραμμές του κυκλώματος. Ενδεικτικά παραθέτουμε το περιεχόμενο της μνήμης δεδομένων (RAM) παρακάτω:



Σχήμα 6.1 - Απεικόνιση της μνήμης RAM με τις ορθές τιμές

Παρατηρούμε ότι το αποτέλεσμα της πράξης της πρόσθεσης είναι η τιμή 07 και αποθηκεύεται στην πρώτη θέση μνήμης RAM (η γραμμή 127 που φαίνεται στην εικόνα). Κατόπιν προσθέτουμε το πρώτο σφάλμα προς εξομοίωση, το οποίο είναι το σφάλμα μόνιμης τιμής 1 στο 8ο bit της μηδενικής διεύθυνσης της RAM. Μετά ο εξομοιωτής ξεκινά την εξομοίωση για αυτό το σφάλμα. Οι αποκρίσεις των διανυσμάτων ελέγχου εμφανίζονται στην μνήμη δεδομένων, όπως φαίνεται παρακάτω:



Σχήμα 6.2 - Απεικόνιση της μνήμης RAM με τις εσφαλμένες τιμές

Παρατηρούμε ότι το σφάλμα που εξομοιώθηκε προκάλεσε αλλαγή της τιμής στο κελί της μνήμης RAM (127,7), που δηλώνει την πρώτη διεύθυνση και την $8^{η}$ θέση της μνήμης RAM (υπενθυμίζουμε ότι η τιμή -1 δηλώνει την λογική τιμή 1 και η τιμή 0 την λογική τιμή 0). Έτσι, στην εξομοίωση του προγράμματος με το σφάλμα υπάρχει η διαφοροσποίηση σε σχέση με την εξομοίωση ελεύθερη από σφάλματα σε μία θέση μνήμης. Με τα δεδομένα των δύο πινάκων ο εξομοιωτής θα ανιχνεύσει το σφάλμα που κρατάει μόνιμα την τιμή 1 στην θέση της μνήμης RAM (127,7), αφού οι τιμές της μνήμης RAM στην εξομοίωση του προγράμματος ελεύθερη από σφάλματα διαφέρουν από τις τιμές στην εσφαλμένη λειτουργία σε τουλάχιστον ένα κελί.

Πείραμα 2 – ανίχνευση σφαλμάτων στις διευθύνσεις της μνήμης RAM

Στο επόμενο πείραμα αναπτύξαμε πρόγραμμα που αποθηκεύει τιμές σε 6 διαφορετικές διευθύνσεις της μνήμης δεδομένων. Οι τιμές αποθηκεύονται από την 2η θέση της μνήμης δεδομένων μέχρι και την 6η θέση. Επιπλέον τα σφάλματα που ανιχνεύονται αφορούν τον καταχωρητή που διατηρεί τις διεύθυνσεις της μνήμης RAM. Δηλαδή στην περίπτωση που θα πρέπει να αποθηκευτούν τα δεδομένα στην 2η διεύθυνση ο καταχωρητής αυτός έχει την τιμή 0000010 κ.λ.π. Η λίστα σφαλμάτων που

ελέγχθηκε ήταν για τις μόνιμες τιμές σε 1 ή 0 σε κάθε bit του καταχωρητή αυτού. Το πρόγραμμα σε συμβολική γλώσσα φαίνεται στον ακόλουθο πίνακα

Συμβολική Γλώσσα

mov a, #06h
mov 2h, a
mov 3h, a
mov 4h, a
mov 5h, a
mov 6h, a

Πίνακας 6.2 – Συμβολική γλώσσα 2ου πειράματος

Το πρόγραμμα αυτό ελέγχει την παρουσία σφαλμάτων μόνιμης τιμής στις ακόλουθες μονάδες του επεξεργαστή:

- Συσσωρευτής A

Τα αποτελέσματα φαίνονται στο παρακάτω σχήμα. Αναγράφεται με «ΝΑΙ» αν ανιχνεύθηκε το αντίστοιχο σφάλμα ή με «ΟΧΙ» αν δεν ανιχνεύθηκε.

	$2^{η}$ Διεύθυνση	$3^{η}$ Διεύθυνση	$4^{η}$ Διεύθυνση	$5^{η}$ Διεύθυνση	$6^{η}$ Διεύθυνση
U_CTR_ram_addr_reg_0/ Q S-A-0	ΟΧΙ	ΝΑΙ	ΟΧΙ	ΝΑΙ	ΟΧΙ
U_CTR_ram_addr_reg_1/ Q S-A-0	ΝΑΙ	ΝΑΙ	ΟΧΙ	ΟΧΙ	ΝΑΙ
U_CTR_ram_addr_reg_2/ Q S-A-0	ΟΧΙ	ΟΧΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ
U_CTR_ram_addr_reg_3/ Q S-A-0	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ
U_CTR_ram_addr_reg_4/ Q S-A-0	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ
U_CTR_ram_addr_reg_5/ Q S-A-0	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ
U_CTR_ram_addr_reg_6/ Q S-A-0	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ
U_CTR_ram_addr_reg_7/ Q S-A-0	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ

U_CTR_ram_addr_reg_0/ Q S-A-1	ΝΑΙ	ΟΧΙ	ΝΑΙ	ΟΧΙ	ΝΑΙ
U_CTR_ram_addr_reg_1/ Q S-A-1	ΟΧΙ	ΟΧΙ	ΝΑΙ	ΝΑΙ	ΟΧΙ
U_CTR_ram_addr_reg_2/ Q S-A-1	ΝΑΙ	ΝΑΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ
U_CTR_ram_addr_reg_3/ Q S-A-1	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ
U_CTR_ram_addr_reg_4/ Q S-A-1	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ
U_CTR_ram_addr_reg_5/ Q S-A-1	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ
U_CTR_ram_addr_reg_6/ Q S-A-1	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ
U_CTR_ram_addr_reg_7/ Q S-A-1	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ

Πίνακας 6.3 - Αποτελέσματα 2ου πειράματος

Αυτό που παρατηρείται στην παραπάνω εικόνα είναι ότι στην περίπτωση αποθήκευσης δεδομένων στην 2η διεύθυνση της μνήμης δεδομένων, ο καταχωρητής που περιέχει την διεύθυνση αποθήκευσης περιέχει την τιμή 00000010_B. Τα σφάλματα που μπορούμε να ανιχνεύσουμε στην περίπτωση αυτή είναι η μόνιμη τιμή 0 στο 2ο bit του καταχωρητή διεύθυνσης και οι μόνιμες τιμές 1 στα υπόλοιπα bits. Παραπλήσια λογική ακολουθείται και στις επόμενες περιπτώσεις.

Πείραμα 3 – 50 μακροεντολές αριθμητικών πράξεων.

Στο επόμενο πείραμα χρησιμοποιούμε ένα σύνολο από μακροντολές. Ως μακροεντολές ορίζουμε ομάδες εντολών που στόχο έχουν να εκτελέσουν κάποιες βασικές λειτουργίες. Οι μακροεντολές που χρησιμοποιήσαμε σε αυτό το πείραμα περιέχουν αριθμητικές πράξεις: προσθέσεις, αφαιρέσεις, πολλαπλασιασμούς και διαιρέσεις. Σε κάθε μακροεντολή ανατίθει μία τιμή στο συσσωρευτή Α και μία τιμή στο συσσωρευτή Β. Στην συνέχεια γίνονται αριθμητικές πράξεις μεταξύ των Α, Β και το αποτέλεσμα τους αποθηκεύεται σε μία θέση της RAM. Οι θέσεις που αποθηκεύονται τα δεδομένα είναι διαδοχικές χωρίς να υπάρχουν επικαλύψεις. Με την ολοκλήρωση των 50 μακροεντολών συμπληρώνονται με τιμές οι 50 πρώτες θέσεις της μνήμης RAM. Ενδεικτικά, στον ακόλουθο πίνακα φαίνονται κάποιες από τις μακροεντολές σε συμβολική γλώσσα.

Συμβολική Γλώσσα

mov a, #06h
mov b, #05h
add a,b
mov 0h, a
--- 	
mov a, #01h
mov b, #03h
subb a,b
mov 1h, a
--- 	
mov a, #16h
mov b, #05h
mul ab
mov 2h, a
--- 	
mov a, #06h
mov b, #15h
div ab
mov 3h, a

Πίνακας 6.4 – Συμβολική γλώσσα 3ου πειράματος

Το πρόγραμμα αυτό ελέγχει την παρουσία σφαλμάτων μόνιμης τιμής στις ακόλουθες μονάδες του επεξεργαστή:

- Συσσωρευτές A, B
- Αριθμητική λογική μονάδα
- Μνήμη RAM
- Μονάδα ελέγχου
- Αποκωδικοποιητής εντολών

Στη συνέχεια, πραγματοποιήθηκε εξομοίωση του προγράμματος ελεύθερη από σφάλματα και έπειτα εξομοίωση εσφαλμένης λειτουργίας υπό την παρουσία σφαλμάτων μόνιμης τιμής. Το πλήθος των σφαλμάτων είναι 12544 και αφορούν όλα τα πιθανά σφάλματα που μπορούν να εμφανιστούν σε οποιαδήποτε από τις διασυνδέσεις του επεξεργαστή. Τα αποτελέσματα που παράχθηκαν φαίνονται παρακάτω.

Τύπος λογικής	Κάλυψη	Χρόνος εκτέλεσης

	σφαλμάτων	
2-values	7689/12544 = 61,3%	11,2 ώρες

Πίνακας 6.5 – Αποτελέσματα 3ου πειράματος

Από το παραπάνω σχήμα βλέπουμε ότι χωρίς την χρήση αδιάφορων τιμών η κάλυψη σφαλμάτων που επιτυχγάνεται είναι 61,3%. Τα σφάλματα που ανιχνεύονται είναι κυρίως σφάλματα στην αριθμητική λογική μονάδα, στην μνήμη RAM και σφάλματα στην μονάδα ελέγχου. Ένα χαρακτηριστικό του πειράματος αυτού είναι ότι δεν υπάρχει επαν-αποθήκευση στις ίδιες θέσεις της μνήμης RAM, επειδή τα αποτελέσματα από την κάθε μακροεντολή αποθηκεύονται σε διαδοχικές θέσεις της μνήμης RAM. Τα σφάλματα που ανιχνεύονται στο πείραμα αυτό είναι σφάλματα που εμφανίζονται στον διαύλο δεδομένων (data bus). Αφού, η μνήμη RAM έχει πρόσβαση στο data bus, τα σφάλματα αυτά εμφανίζονται και στην μνήμη RAM, όπου και ανιχνεύονται.

Πείραμα 4 - 50 μακροεντολές αριθμητικών πράξεων με τη χρήση της Fault dropping και της Fault no dropping τεχνικής.

Στο πείραμα αυτό χρησιμοποιήθηκε το ίδιο πρόγραμμα εντολών με το προηγούμενο πείραμα (50 μακροεντολές με τέσσερις αριθμητικές πράξεις) και μελετήθηκαν δύο τεχνικές ανίχνευσης σφαλμάτων: η fault dropping και η no fault dropping. Στην fault dropping τεχνική μόλις ένα σφάλμα αλλάξει την τιμή μίας θέσης μνήμης το σφάλμα θεωρείται ότι είναι ανιχνεύσιμο. Στην no fault dropping το σφάλμα θεωρείται ανιχνεύσιμο μόνο αν έχει επηρέσει μία θέση μνήμης μετά την ολοκλήρωση της εξομοίωσης όλων των εντολών του προγράμματος ελέγχου. Η πρώτη τεχνική έχει το μειονέκτημα του μεγάλου χρόνου εξομοίωσης, ενώ η δεύτερη τεχνική έχει το μειονέκτημα της μειωμένης ακρίβειας αφού η απόκρυψη σφαλμάτων (fault masking) στην μνήμη δεν μπορεί να αποφευχθεί.

Στο συγκεκριμένο πείραμα έχουμε 50 μακροεντολές που αποθηκεύουν διαδοχικά τα αποτελέσματα στις πρώτες 50 θέσεις της μνήμης RAM. Στην πρώτη τεχνική fault dropping μόλις μία μακροεντολή εκτελεστεί και αποθηκεύσει εσφαλμένη τιμή τότε το αντίστοιχο σφάλμα αποσύρεται από την λίστα σφαλμάτων και θεωρείται ανιχνευμένο. Στην περίπτωση της τεχνικής no fault dropping οι πίνακες που διατηρούν τα δεδομένα

της μνήμης RAM, που παράχθηκαν κατά την ελεύθερη από σφάλματα λειτουργία και την εξομοίωση υπό σφάλματα θα συγκριθούν αφού εκτελεστούν και οι 50 μακροεντολές της μνήμης ROM.

Στο πείραμα αυτό πραγματοποιήθηκαν οι δύο παραπάνω τεχνικές συμπληρώνοντας την μνήμη ROM με τις 50 προηγούμενες μακροεντολές. Ακόμη, πραγματομοιήθηκαν εξομοιώσεις στην περίπτωση που οι καταχωρητές μπορούν να λάβουν δυαδικές τιμές (0 ή 1) και στην περίπτωση των τριαδικών τιμών (0 ή 1 ή X). Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα.

Τύπος λογικής	Τύπος Fault Simulation	Κάλυψη σφαλμάτων	Χρόνος εκτέλεσης
2-values	Fault dropping	7702/12544 = 61.4%	6,7 ώρες
2-values	No fault dropping	7689/12544 = 61,4%	11,2 ώρες
3-values	Fault dropping	8417/12544 = 67,1%	6,54 ώρες
3-values	No fault dropping	8417/12544 = 67,1%	10,9 ώρες

Πίνακας 6.6 - Αποτελέσματα 4^{ου} πειράματος

Και στις δύο τεχνικές το ποσοστό κάλυψης ήταν περίπου 61.4% στην περίπτωση που οι καταχωρητές ακολουθούν την δυαδική λογική (0,1), ενώ, στην περίπτωση που οι καταχωρητές ακολουθούν την τριαδική λογική (0, 1, X) η κάλυψη ήταν περίπου 67.1%. Ακόμη, παρατηρούμε ότι στην περίπτωση που οι καταχωρητές έχουν τη δυνατότητα χρήσης της τριαδικής λογικής τότε το ποσοστό κάλυψης αυξάνεται. Ο λόγος είναι ότι με τη χρήση της τριαδικής λογικής ένα σφάλμα θεωρείται ανιχνεύσιμο αν στην ελεύθερη από σφάλματα λειτουργία παραχθεί ‘0’ σε μια θέση μνήμης και στην ίδια θέση παραχθεί ‘1’ ή άγνωστη κατάσταση (‘X’) στην εσφαλμένη λειτουργία. Το αντίστοιχο ισχύει αν παραχθεί ‘1’ στην ελεύθερη από σφάλματα λειτουργία και ‘0’ ή ‘X’ στην εσφαλμένη λειτουργία.

Πείραμα 5 – επαναλήψεις, συνθήκες μετάβασης και εκχώρηση τιμών στα κελία της μνήμης RAM.

Στο επόμενο πείραμα χρησιμοποιήθηκε σαν πρόγραμμα στην μνήμη ROM ένας πιο σύνθετος κώδικας με αυξημένη πολυπλοκότητα. Το πρόγραμμα σε συμβολική γλώσσα φαίνονται στον ακόλουθο πίνακα

Συμβολική Γλώσσα

```
CLR SM0
   SETB SM1                    ; | put serial port in 8-bit UART mode

   MOV A, PCON                 ; |
   SETB ACC.7
   MOV PCON, A                 ; | set SMOD in PCON to double baud rate

   MOV TMOD, #20H              ; | put timer 1 in 8-bit auto-reload interval timing
mode
   MOV TH1, #243               ; |put -13 in timer 1 high byte (timer will
overflow every 13 us)
   MOV TL1, #243               ; put same value in low byte so when timer is
first started it will       overflow after 13 us
   SETB TR1                    ; start timer 1

   MOV 30H, #'a'
   MOV 31H, #'b'
   MOV 32H, #'c'                ; | put data to be sent in RAM, start address 30H

   MOV 33H, #0                  ; null-terminate the data (when the accumulator
contains 0, no more data to be sent)
   MOV R0, #30H                ; put data start address in R0
again:
   MOV A, @R0                  ; move from location pointed to by R0 to the
accumulator
   JZ finish                   ; if the accumulator contains 0, no more data to be
sent, jump to finish
   MOV C, P                    ; otherwise, move parity bit to the carry
   MOV ACC.7, C                ; and move the carry to the accumulator MSB
   MOV SBUF, A                 ; move data to be sent to the serial port
   INC R0                      ; increment R0 to point at next byte of data to be
sent
   JNB TI, $                   ; wait for TI to be set, indicating serial port has
finished sending byte
   CLR TI                      ; clear TI
   JMP again                   ; send next byte
finish:
   JMP $
```

Πίνακας 6.7 - Συμβολική γλώσσα 5ου πειράματος

Το πρόγραμμα αυτό ελέγχει την παρουσία σφαλμάτων στις ακόλουθες μονάδες του επεξεργαστή:

- Συσσωρευτής Α
- Μνήμη RAM
- Μονάδα ελέγχου
- Αποκωδικοποιητής εντολών
- Καταχωρητές TMOD, PCON, SMOD και SBUF

Στο πείραμα αυτό χρησιμοποιήθηκαν και οι δύο τεχνικές Fault dropping και No fault dropping τόσο με δυαδική όσο και με τριαδική λογική. Τα αποτελέσματα για τις τέσσερις περιπτώσεις φαίνονται στον παρακάτω πίνακα.

Τύπος λογικής	Τύπος Fault Simulation	Κάλυψη σφαλμάτων	Χρόνος εκτέλεσης
2-values	Fault dropping	7658/12544 = 61%	7,8 ώρες
2-values	No fault dropping	7593/12544 = 60.5	10,21 ώρες
3-values	Fault dropping	7712/12544 = 61.4%	7,2 ώρες
3-values	No fault dropping	7712/12544 = 61.4%	10,02 ώρες

Πίνακα 6.8 - Αποτελέσματα 5ου πειράματος

Στο πείραμα αυτό παρατηρούμε ότι, κατά την εκτέλεση της τεχνικής Fault Dropping φαίνεται να έχουμε μεγαλύτερη κάλυψη σφαλμάτων κατά 0.05%, σε σχέση με την τεχνική No Fault Dropping. Αυτό οφείλεται, στο ότι κάποια σφάλματα κατά την εσφαλμένη λειτουργία έχουν προκαλέσει μια τροποποίηση σε κάποιο κελί της μνήμης RAM. Στη συνέχεια, με την εκτέλεση των επόμενων εντολών της μνήμης ROM, η τιμή των κελιών αυτών άλλαξε με αποτέλεσμα να αντικαθίσταται το λάθος με την σωστή τιμή του. Σφάλματα σαν αυτά δεν μπορούν να ανιχνευθούν με την τεχνική No Fault

Dropping, και αυτό αποτελεί το πρόβλημα της μεθόδου, ωστόσο ο αριθμός τους είναι πολύ μικρός.

Πείραμα 6 – σύνθετο πρόγραμμα με μεγαλύτερη πολυπλοκότητα.

Στο επόμενο πείραμα χρησιμοποιήθηκε σαν πρόγραμμα στην μνήμη ROM ένας ακόμη πιο σύνθετος κώδικας ο οποίος συνδυάζει εντολές επανάληψης και αποθήκευσης δεδομένων στην μνήμη RAM. Το πρόγραμμα σε συμβολική γλώσσα φαίνονται στον ακόλουθο πίνακα

Συμβολική Γλώσσα

```
MOV TMOD, #50H                  ; put timer 1 in event counting mode
   SETB TR1                     ; start timer 1
   MOV DPL, #LOW(LEDcodes)      ; | put the low byte of the start address of
the
                                ; | 7-segment code table into DPL
   MOV DPH, #HIGH(LEDcodes)     ; put the high byte into DPH
   CLR P3.4
   CLR P3.3                     ; | enable Display 0
again:
   CALL set Direction           ; set the motor's direction
   MOV A, TL1                   ; move timer 1 low byte to A
   CJNE A, #10, skip            ; if the number of revolutions is not 10
skip next instruction
   CALL clearTimer              ; if the number of revolutions is 10,
reset timer 1
skip:
   MOVC A, @A+DPTR              ; | get 7-segment code from code table -
the index into the table is
                                ; | decided by the value in A
                                ; | (example: the data pointer points to the
start of the
                                ; | table - if there are two revolutions, then
A will contain two,
                                ; | therefore the second code in the table
will be copied to A)

   MOV C, F0                    ; move motor direction value to the carry
   MOV ACC.7, C                 ; and from there to ACC.7 (this will
ensure Display 0's decimal point
                                ; will indicate the motor's direction)

   MOV P1, A                    ; | move (7-seg code for) number of
revolutions and motor direction
                                ; | indicator to Display 0
```

```
   JMP again                        ; do it all again

set Direction:
   PUSH ACC                         ; save value of A on stack
   PUSH 20H                         ; save value of location 20H (first bit-
addressable
                                    ;    location in RAM) on stack
   CLR A                            ; clear A
   MOV 20H, #0                      ; clear location 20H
   MOV C, P2.0                      ; put SW0 value in carry
   MOV ACC.0, C                     ; then move to ACC.0
   MOV C, F0                        ; move current motor direction in carry
   MOV 0, C                         ; and move to LSB of location 20H
(which has bit address 0)

   CJNE A, 20H, changeDir           ; | compare SW0 (LSB of A) with F0
(LSB of 20H)
                                    ; | - if they are not the same, the motor's
direction needs to be reversed

   JMP finish                       ; if they are the same, motor's direction
does not need to be changed
changeDir:
   CLR P3.0
   CLR P3.1                         ; | stop motor

   CALL clearTimer                  ; reset timer 1 (revolution count restarts
when motor direction changes)
   MOV C, P2.0                      ; move SW0 value to carry
   MOV F0, C                        ; and then to F0 - this is the new motor
direction
   MOV P3.0, C                      ; move SW0 value (in carry) to motor
control bit 1
   CPL C                            ; invert the carry

   MOV P3.1, C                      ; | and move it to motor control bit 0 (it
will therefore have the opposite
                                   ; | value to control bit 1 and the motor will
start
                                   ; | again in the new direction)
finish:
   POP 20H                          ; get original value for location 20H from
the stack
   POP ACC                          ; get original value for A from the stack
   RET                              ; return from subroutine

clearTimer:
   CLR A                            ; reset revolution count in A to zero
   CLR TR1                          ; stop timer 1
   MOV TL1, #0                      ; reset timer 1 low byte to zero
   SETB TR1                         ; start timer 1
   RET                              ; return from subroutine

LEDcodes:                          ; | this label points to the start address of the
```

```
7-segment code table which is
                                        ; | stored in program memory using the DB
command below
  DB  11000000B,  11111001B,  10100100B,  10110000B,  10011001B,
10010010B, 10000010B, 11111000B, 10000000B, 10010000B
```

Πίνακας 6.9 – Συμβολική γλώσσα 6ου πειράματος

Το πρόγραμμα αυτό ελέγχει την παρουσία σφαλμάτων στις ακόλουθες μονάδες του επεξεργαστή:

- Συσσωρευτής A
- Αριθμητική λογική μονάδα
- Μνήμη RAM
- Μονάδα ελέγχου
- Αποκωδικοποιητής εντολών
- Καταχωρητές TMOD, PCON, SMOD και SBUF

Στο πείραμα αυτό όπως και στο προηγούμενο πραγματοποιήθηκε η Fault dropping και η No fault dropping τεχνική. Όπως επίσης το πείραμα αυτό υλοποιήθηκε χρησιμοποιώντας οι καταχωρητές την δυαδική και την τριαδική λογική. Τα αποτελέσματα για τις τέσσερις περιπτώσεις φαίνονται στον παρακάτω πίνακα. Συγκρεκριμένα, παρουσιάζεται μία περίπτωση χρησιμοποιώντας την Fault dropping τεχνική και δυαδική λογική, έπειτα no fault dropping τεχνική και δυαδική λογική. Τέλος, υλοποιήθηκαν άλλες δύο περιπτώσεις χρησιμοποιώντας τις δύο τεχνικές αλλά με τη χρήση της τριαδικής λογικής. Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα.

Τύπος λογικής	Τύπος Fault Simulation	Κάλυψη σφαλμάτων	Χρόνος εκτέλεσης
2-values	Fault dropping	8806/12544 = 70%	8.9 ώρες
2-values	No fault dropping	8806/12544 = 70%	15,25 ώρες
3-values	Fault dropping	9587/12544 = 76.4%	7.9 ώρες
3-values	No fault dropping	9411/12544 = 75.06%	14.74 ώρες

Πίνακας 6.10 - Αποτελέσματα 6ου πειράματος

Στα αποτελέσματα αυτού του πειράματος παρατηρούμε ότι πάλι με η Fault Dropping τεχνική δίνει υψηλότερο ποσοστό κάλυψης σφαλμάτων (οι λόγοι αναλύθηκαν παραπάνω) και μικρότερο χρόνο εκτέλεσης. Επιπλέον, με τη χρήση της δυαδικής λογικής στο παράδειγμα αυτό δεν είχαμε διαφορετικό ποσοστό κάλυψης στην Fault Dropping και στην No Fault Dropping τεχνική. Η κάλυψη σφαλμάτων στο πείραμα αυτό αυξήθηκε επειδή ανιχνεύθηκαν περισσότερα σφάλματα της αριθμητικής λογικής μονάδας, αφού υπάρχουν περισσότερες λογικές και αριθμητικές πράξεις μέσα στο πρόγραμμα, όπως επίσης και συνθήκες επανάληψης.

Πείραμα 7 – εκχώρηση τιμών στα pin εξόδου του επεξεργαστή (P0 – P3).

Στο επόμενο πείραμα το πρόγραμμα που χρησιμοποιήθηκε έχει τη δυνατότητα να εκχωρεί τιμές στα pins εξόδου του επεξεργαστή, (P0 – P3) και στη συνέχεια να πραγματοποιούνται κάποιες λογικές συγκρίσεις μεταξύ των εξόδων αυτών με έναν το περιεχόμενο του καταχωρητή R0. Το πρόγραμμα σε συμβολική γλώσσα φαίνονται στον ακόλουθο πίνακα

Συμβολική Γλώσσα

```
start:
MOV R0, #0              ; clear R0 - the first key is key0

                        ; scan row0
SETB P0.3               ; set row3
CLR P0.0                ; clear row0
CALL colScan            ; call column-scan subroutine
JB F0, finish           ; | if F0 is set, jump to end of program
                        ; | (because the pressed key was found and its
number is in  R0)

; scan row1
SETB P0.0               ; set row0
CLR P0.1                ; clear row1
CALL colScan            ; call column-scan subroutine
JB F0, finish           ; | if F0 is set, jump to end of program
                        ; | (because the pressed key was found and its
number is in  R0)
```

```
   ; scan row2
   SETB P0.1              ; set row1
   CLR P0.2               ; clear row2
   CALL colScan           ; call column-scan subroutine
   JB F0, finish          ; | if F0 is set, jump to end of program
                          ; | (because the pressed key was found and its
number is in  R0)

   ; scan row3
   SETB P0.2              ; set row2
   CLR P0.3               ; clear row3
   CALL colScan           ; call column-scan subroutine
   JB F0, finish          ; | if F0 is set, jump to end of program
                          ; | (because the pressed key was found and its
number is in  R0)


   JMP start              ; | go back to scan row 0
                          ; | (this is why row3 is set at the start of the
program
                          ; | - when the program jumps back to start, row3
has just been scanned)

finish:
   JMP $                  ; program execution arrives here when key is
found - do nothing



                          ; column-scan subroutine
colScan:
   JNB P0.4, gotKey       ; if col0 is cleared - key found
   INC R0                 ; otherwise move to next key
   JNB P0.5, gotKey       ; if col1 is cleared - key found
   INC R0                 ; otherwise move to next key
   JNB P0.6, gotKey       ; if col2 is cleared - key found
   INC R0                 ; otherwise move to next key
   RET                    ; return from subroutine - key not found
gotKey:
   SETB F0                ; key found - set F0
   RET                   ; and return from subroutine
```

Πίνακας 6.11 – Συμβολική γλώσσα 7ου πειράματος

Το πρόγραμμα αυτό ελέγχει την παρουσία σφαλμάτων στις ακόλουθες μονάδες του επεξεργαστή:

- Αριθμητική λογική μονάδα
- Μνήμη RAM
- Μονάδα ελέγχου
- Αποκωδικοποιητής εντολών

Στο πείραμα αυτό όπως και στο προηγούμενο πραγματοποιήθηκε η Fault dropping και η no fault dropping τεχνική. Όπως επίσης το πείραμα αυτό υλοποιήθηκε χρησιμοποιώντας οι καταχωρητές την δυαδική και την τριαδική λογική. Έτσι, τα αποτελέσματα για τις τέσσερις περιπτώσεις φαίνονται στον παρακάτω πίνακα. Συγκρεκριμένα, μία περίπτωση χρησιμοποιώντας την Fault dropping τεχνική και δυαδική λογική, έπειτα no fault dropping τεχνική και δυαδική λογική. Τέλος, υλοποιήθηκαν άλλες δύο περιπτώσεις χρησιμοποιώντας τις δύο τεχνικές αλλά με τη χρήση της τριαδικής λογικής. Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα.

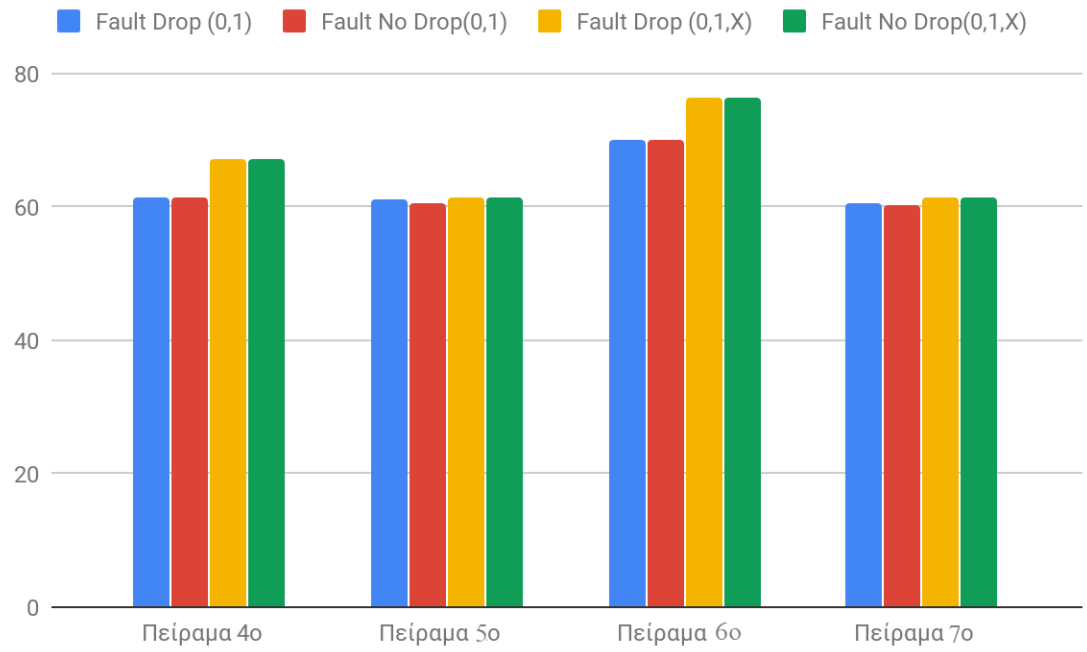
Τύπος λογικής	Τύπος Fault Simulation	Κάλυψη σφαλμάτων	Χρόνος εκτέλεσης
2-values	Fault dropping	7599/12544 = 60,5%	8,5 ώρες
2-values	No fault dropping	7566/12544 = 60,3%	16,21 ώρες
3-values	Fault dropping	7725/12544 = 61,5%	8,1 ώρες
3-values	No fault dropping	7702/12544 = 61,3%	16,02 ώρες

Πίνακας 6.12 - Αποτελέσματα 7ου πειράματος

Επίσης και σε αυτό το πείραμα έχουμε τις ίδιες παρατηρήσεις με τα προηγούμενα. Βλέπουμε ότι με την Fault Drop προσομοίωση έχουμε καλύτερα αποτελέσματα ως προς το χρόνο εκτέλεσης. Ακόμη το ποσοστό κάλυψης σφαλμάτων σε όλες τις περιπτώσεις εκτέλεσης του πειράματος έχει μειωθεί λόγω της μικρής πολυπλοκότητας του προγράμματος. Δηλαδή η λειτουργία του είναι σχετικά απλή χωρίς να εμπλέκονται πολλοί καταχωρητές.

Συγκεντρωτικά, τα τέσσερα τελευταία πειράματα απεικονίζονται στο παρακάτω διάγραμμα. Ο κάθετος άξονας αφορά το ποσοστό κάλυψης των σφαλμάτων και ο οριζόντιος άξονας το κάθε πείραμα. Επίσης για κάθε πείραμα η πρώτη μπάρα αφορά την Fault dropping τεχνική στην δυαδική λογική, η δεύτερη μπάρα αφορά την Fault no dropping τεχνική στην δυαδική λογική, η τρίτη μπάρα αφορά την Fault dropping τεχνική

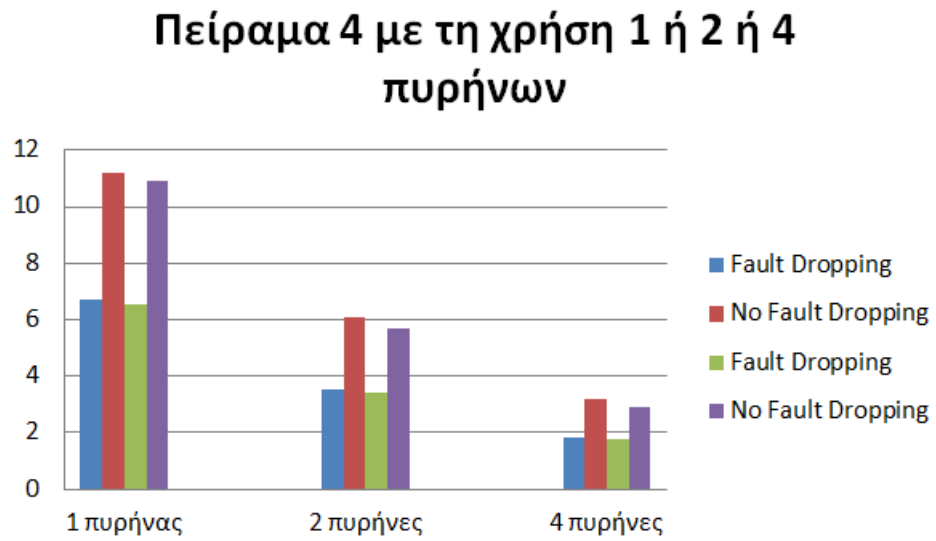
στην τριαδική λογική και η τέταρτη μπάρα αφορά την Fault no dropping τεχνική στην τριαδική λογική.



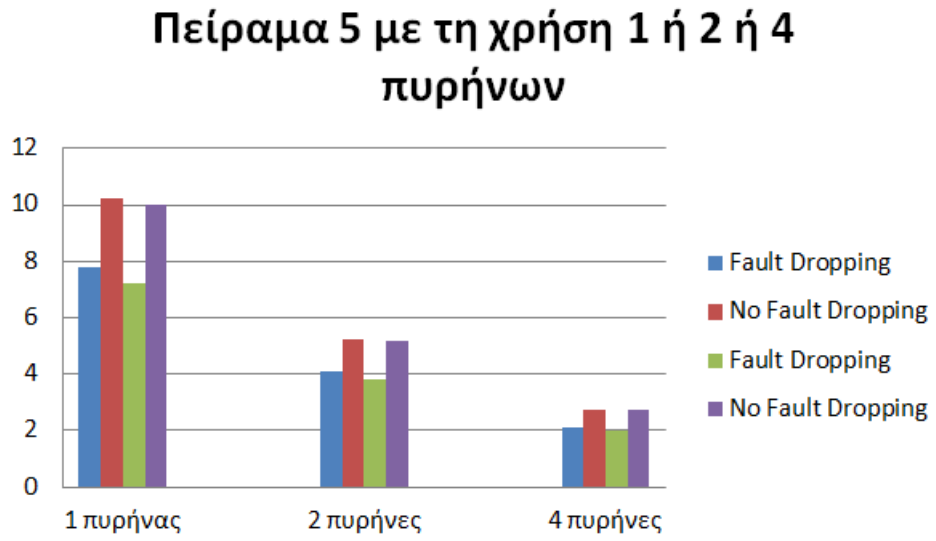
Σχήμα 6.3 - Συγκεντρωτικά αποτελέσματα από τα πειράματα

Πείραμα 8 – Χρήση υπολογιστικών συστημάτων με περισσότερους πυρήνες

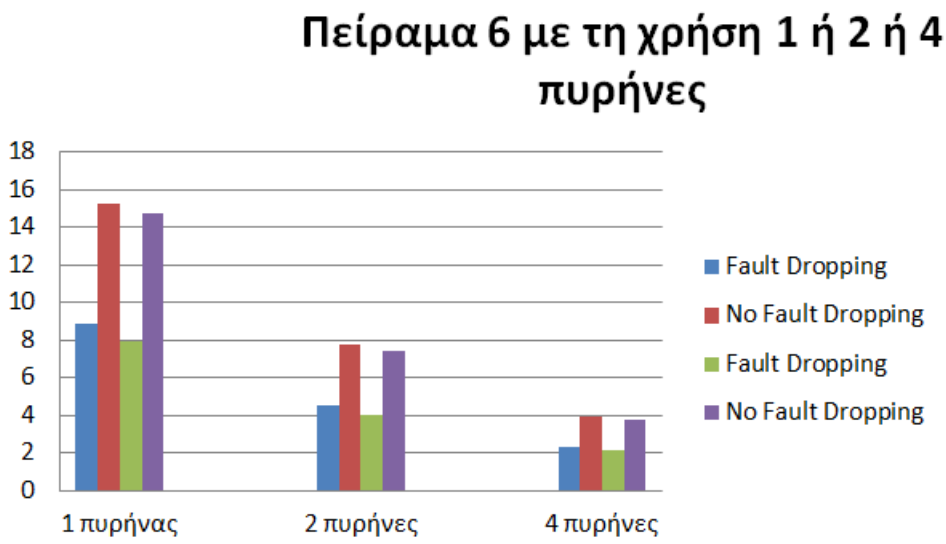
Τα προηγούμενα πειράματα υλοποιήθηκαν σε υπολογιστή με ένα πυρήνα. Στην ενότητα αυτή, εκτελέστηκαν τα προηγούμενα τέσσερα τελευταία πειράματα σε δυο διαφορετικά υπολογιστικά συστήματα. Την πρώτη φορά σε υπολογιστή με δύο πυρήνες και στη συνέχεια σε υπολογιστή με τέσσερις πυρήνες. Επειδή στον εξομοιωτή που κατασκευάστηκε η εξομοίωση ανίχνευσης κάθε σφάλματος είναι ανεξάρτητη από τα υπόλοιπα σφάλματα, η χρήση περισσότερων πυρήνων θα επιφέρει σημαντική επιτάχυνση της διαδικασίας αυτής ανάλογα το πλήθος των πυρήνων. Τα αποτελέσματα φαίνονται στις παρακάτω εικόνες. Η πρώτη εικόνα αφορά το πείραμα 4 με τη χρήση 1 ή 2 ή 4 πυρήνων. Οι επόμενες εικόνες αναφέρονται στα πειράματα 5, 6 και 7 χρησιμοποιώντας πάλι 1 ή 2 ή 4 πυρήνες. Επιπλέον, ο κάθετος άξονας στα παρακάτω διαγράμματα αναφέρεται στον χρόνο εκτέλεσης των πειραμάτων και ο οριζόντιος στον αριθμο των πυρήνων που χρησιμοποιούνται σε κάθε εκτέλεση.



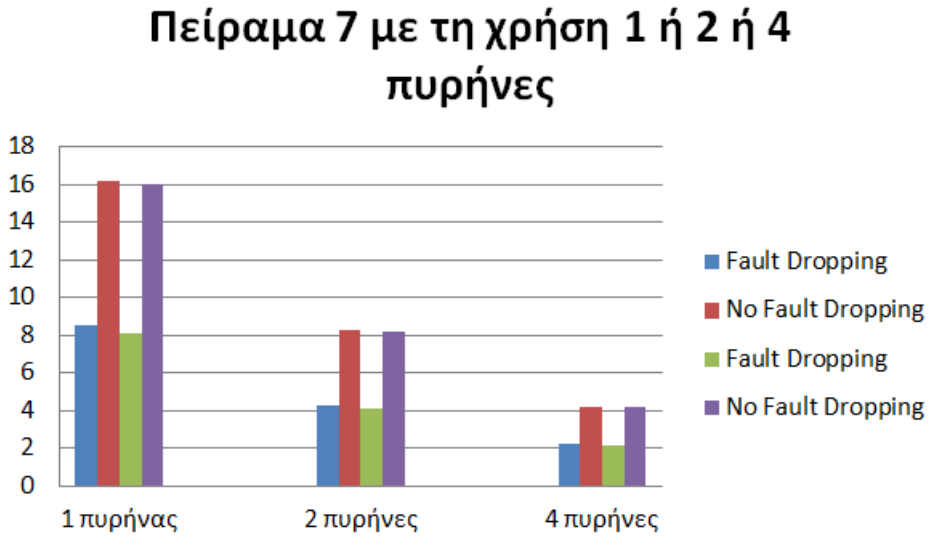
Σχήμα 6.4 - Εκτέλεση του πειράματος 4 με διάφορες επιλογές στον αριθμό των πυρήνων



Σχήμα 6.5 - Εκτέλεση του πειράματος 5 με διάφορες επιλογές στον αριθμό των πυρήνων



Σχήμα 6.6 - Εκτέλεση του πειράματος 6 με διάφορες επιλογές στον αριθμό των πυρήνων



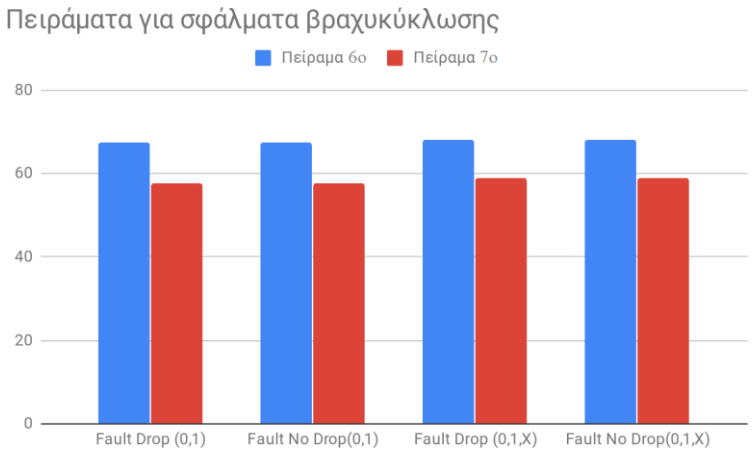
Σχήμα 6.7 - Εκτέλεση του πειράματος 7 με διάφορες επιλογές στον αριθμό των πυρήνων

Η παρατήρηση είναι ότι με τη χρήση ενός συστήματος με δύο πυρήνες οι χρόνοι κάθε πειράματος μειώνονται σχεδόν στο ½ του αρχικού. Τέλος με τη χρήση 4 πυρήνων ο χρόνος των πειραμάτων μειώθηκε σχεδόν στο ¼ του αρχικού.

6.2 Πειράματα εξομοίωσης σφαλμάτων γεφύρωσης

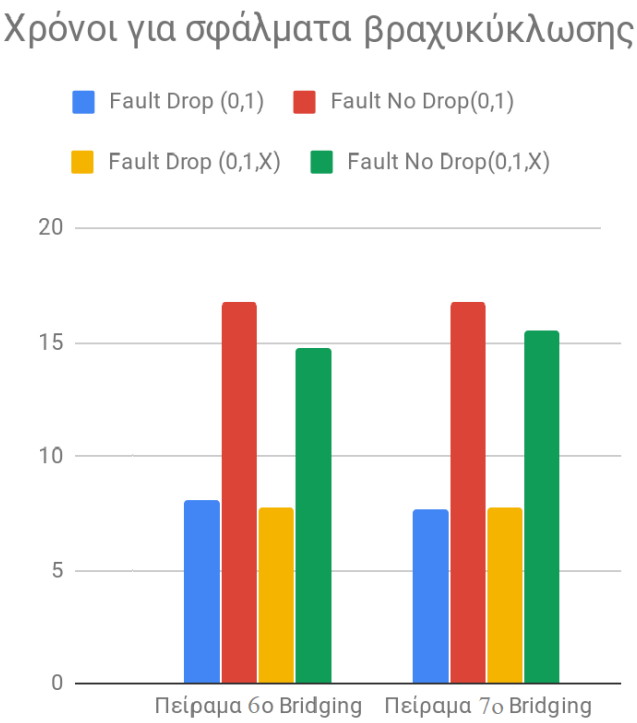
Πείραμα 9 - Εξομοίωση Σφαλμάτων Βραχυκύκλωσης.

Στο επόμενο πείραμα μελετήθηκε η ανίχνευση σφαλμάτων βραχυκύκλωσης. Για τα σφάλματα βραχυκύκλωσης το αρχείο με τα σφάλματα συμπληρώθηκε με 10.000 ζευγάρια βραχυκυκλώματα. Συγκεκριμένα κάθε γραμμή του αρχείου περιείχε τις δύο γραμμές που αφορούν το βραχυκύκλωμα και έναν αριθμό που δηλώνει ποια γραμμή "κυριαρχεί" της άλλης. Τα ζευγάρια επιλέχθηκαν τυχαία, δηλαδή κάθε βραχυκύκλωμα αποτελούνταν από δύο τυχαίες γραμμές του κυκλώματος. Σαν περιεχόμενο της ROM χρησιμοποιήθηκαν οι κώδικες των παραδειγμάτων 6 και 7. Τα αποτελέσματα εμφανίζονται παρακάτω:



Σχήμα 6.8 - Αποτελέσματα πειραμάτων για σφάλματα βραχυκύκλωσης

Ακόμη στο παρακάτω σχήμα φαίνονται οι χρόνοι εξομοίωσης των πειραμάτων αυτών. Να σημειωθεί ότι ο το υπολογιστικό σύστημα αποτελείται από έναν πυρήνα.

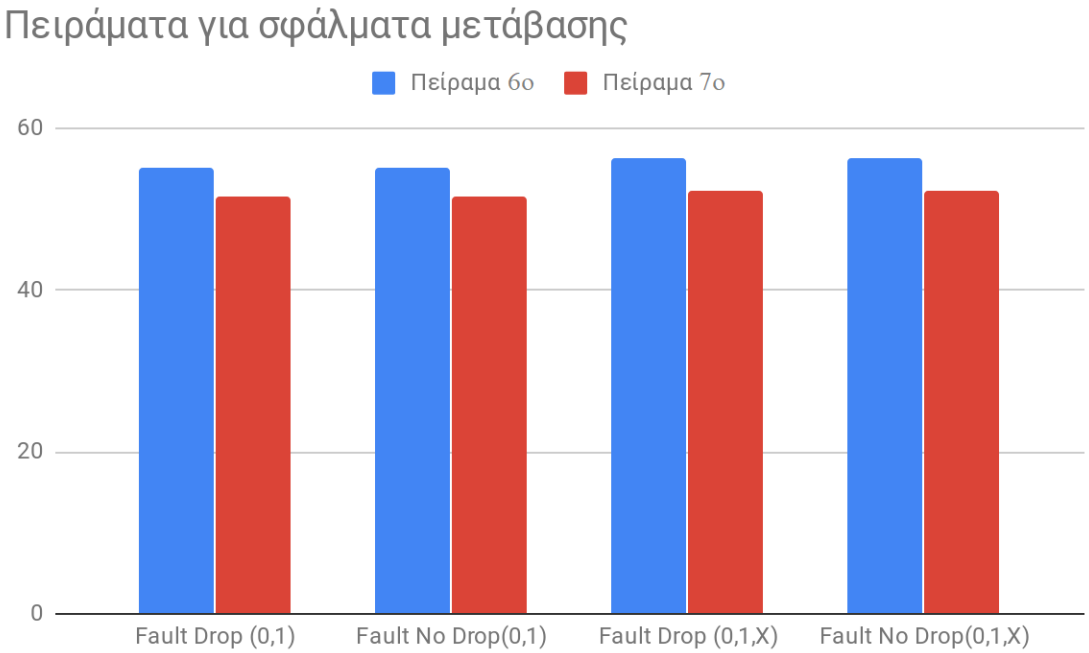


Σχήμα 6.9 - Χρόνοι εκτέλεσης πειραμάτων σφαλμάτων βραχυκυκλώματος

6.3 Πειράματα εξομοίωσης σφαλμάτων γεφύρωσης

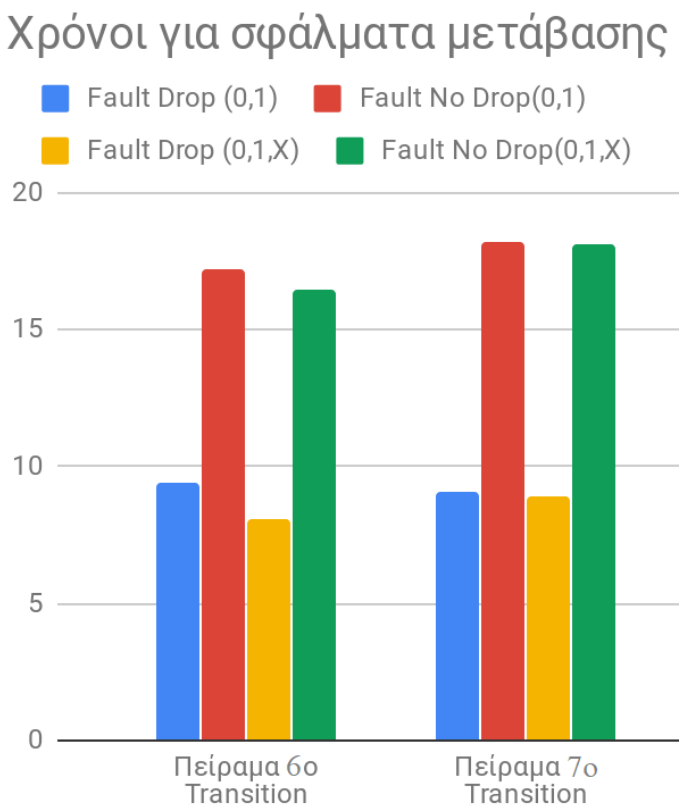
Πείραμα 10 - Εξομοίωση Σφαλμάτων μετάβασης.

Τέλος στην περίπτωση των πειραμάτων για την ανίχνευση των σφαλμάτων μετάβασης, χρησιμοποιήθηκαν τα προγράμματα ελέγχου των πειραμάτων 6 και 7. Τα αποτελέσματα εμφανίζονται στο παρακάτω σχήμα.



Σχήμα 6.10 - Αποτελέσματα ανίχνευσης σφαλμάτων μετάβασης

Οι χρόνοι εκτέλεσης των πειραμάτων για την ανίχνευση σφαλμάτων μετάβασηςαπεικονίζονται παρακάτω. Να σημειωθεί ότι ο το υπολογιστικό σύστημα αποτελείται από έναν πυρήνα.



Σχήμα 6.11 - Χρόνοι εκτέλεσης πειραμάτων σφαλμάτων μετάβασης

ΚΕΦΑΛΑΙΟ 7

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

7.1 Σύνοψη και συμπεράσματα

Στην εργασία αυτή παρουσιάσαμε έναν νέο εξομοιωτή ο οποίος υποστηρίζει την ανίχνευση σφαλμάτων με την χρήση λογισμικού ελέγχου των επεξεργαστικών πυρήνων. Η ανίχνευση των σφαλμάτων γίνεται ολοκληρωτικά στην κύρια μνήμη όπου συγκεντρώνονται τα αποτελέσματα του προγράμματος ελέγχου και συγκρίνονται με τα αναμενόμενα. Σφάλματα τα οποία δεν καταλήγουν στην μνήμη δεν μπορούν να ανιχνευθούν από την τεχνική SBST για αυτό και έχουν αποκλειστεί και από την διαδικασία ανίχνευσης του εξομοιωτή.

Εκτός από τα κλασσικά σφάλματα μόνιμης τιμής, υποστηρίζονται επιπλέον σφάλματα γεφύρωσης και σφάλματα μετάβασης. Επιπλέον η προσομοίωσης μπορεί να γίνει είτε με άμεση διαγραφή σφαλμάτων (fault dropping) μόλις αυτά αλλάξουν έστω και ένα δυαδικό ψηφίο της μνήμης, είτε με διατήρηση τους μέχρι το τέλος της εξομοίωσης. Στην πρώτη τεχνική παρατηρήσαμε απόκρυψη μικρού αριθμού λαθών (fault masking), το οποίο ήταν και αναμενόμενο, αλλά ταυτόχρονα και μειωμένο χρόνο εκτέλεσης. Τέλος, αναπτύχθηκε και παραλληλοποίηση της διαδικασίας ανίχνευσης σφαλμάτων, με τον χρόνο να ελαττώνεται σχεδόν γραμμικά με το πλήθος των πυρήνων που μπορούν να χρησιμοποιηθούν.

7.2 Μελλοντικές επεκτάσεις

Μελλοντικές επεκτάσεις είναι να χρησιμοποιηθεί επεξεργαστής που να διαθέτει εξωτερική μνήμη και σε αυτήν την περίπτωση θα μπορούσε να χρησιμοποιηθεί και αυτή η μνήμη αυτή σαν μέσο προβολής των αποκρίσεων. Στη συνέχεια, μια σημαντική επέκταση θα ήταν να χρησιμοποιηθεί ένας άλλος επεξεργαστής μεταγενέστερος με μεγαλύτερη RAM. Στην περίπτωση αυτή προφανώς αυξάνεται κατά πολύ ο χώρος των αποκρίσεων, συνεπώς και ίσως και το ποσοστό κάλυψης σφαλμάτων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] G.E. Moore, “Cramming more components onto integrated”, in Electronics, vol. 38,no. 8, pp. 114-117, April 19, 1965.

[2] M. Abramovici, M.A Breuer, and A.D Friedman, Digital Systems Testing and Testable Design, IEEE Press, 1990.

[3] G. Singer, "Current Trends and Future Directions in Test and DFT ," in IEEE VLSI Test Symposium, keynote Address , 1997

[4] B. Davis, The Economics of Automatic Testing, 2nd edition, London, United Kingdom: McGraw-Hill,1994.

[5] B. Davis, The Economics of Automatic Testing, 2nd edition, London, United Kingdom: McGraw-Hill,1994.

[6] Kenyon C Y Mei, "Bridging and Stuck-At Faults", in IEEE Transactions on Computers C-23(7):720 - 727 · August 1974

[7] Yi-Cheng Kung, Kuen-Jong Lee , Sudhakar M. Reddy, "Generating Compact Test Patterns for Stuck-at Faults and Transition Faults in One ATPG Run", IEEE, 2018

[8] M. PsarakisD. GizopoulosA. Paschalis, "Test Generation and Fault Simulation for Cell Fault Model using Stuck-at Fault Model based Test Tools", Journal of Electronic Testing, December 1998, Volume 13, Issue 3, pp 315–319.

[9] T. W. Williams and N. C. Brown, Defect level as a function of fault coverage, IEEE Trans. on Computers, 30(12), pp. 987–988, December 1981.

[10] E.J. McCluskey, and F. Buelow, "IC quality and test transparent", in IEEE International Test Conference, pp. 295-301, May 1989

[11] Laung-Terng Wang, Charles E. Stroud, Nur A. Touba, System-On-Chip Test Architectures. Nanometer Design for Testability, Morgan Kaufmann Publishers, 2008.

[12] M. Abramovici and P.R. Menon, “A practical approach to Fault Simulation and Test Gen-eration for Bridging Faults,” IEEE Trans. on Computers, Vol. C-34, No.7, pp. 658-663,July, 1985.

[13] K.C.Y. Mei, “Bridging and Stuck-At Faults,” IEEE Trans. on Computers, Vol. C-23,No.7, pp. 720-727 July, 1974.

[14] Y. Gong, S. Chakravarty, “A Diagnosis Algorithm for Bridging Faults in CombinationalCircuits,” Technical Report, Department of Computer Science, University of Buffalo,State University of New York, 1992.

[15] [http://www.eng.auburn.edu/~strouce/class/elec6970/BISTc2.pdf], 13/3/2019

[16] Irith Pomeranz, Sudhakar M. Reddy, "A Transition Fault Model for At-Speed Fault Simulation and Test Generation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol 27, Jan, 2008.

[17] Shetye K, "Transition Delay Faults", TERM PAPER, ELEC 7250, SPRING 2004.

[18] G Singer, " Current Trends and Future Dırections in Test and DFT", in IEEE VLSI Test Symposioum, Keynote Address, 1997.

[19] P. Nigh, W. Needham, K. Butler, P. Maxwell, and R. Aitken, "An Experimental Fault Study Comparing the Relative Effectiveness of Fuctional, Scan and Delay-Testing", in IEEE VLSI Test Symposium, pp. 459-464, May 1997.

[20] M.L. Bushnell, V.D. Agrawal, "Essentials of electronics testing for digital, memory and mixed-signal VLSI circuits", Kluwer Academic Publishers 2000.

[21] E.B. Eichribger, E Lindbloom, J.A. Waicukauski, T.W. Willias, "Structured logic testing". Englewood Cliffs, New Jersey: Prentice-Hall, 1991.

[22] M.J.Y. Williams, J.B. Angell, "Enchancing testability of Large-scale intergrated circuits via test points and additional logic", in IEEE Transactions on Computers, vol C-22, no. 1, pp.46-60, January 1973.

[23] K.P. Parker, "The boundary-Scan Handbook", Third Edition, Kluwer Academic Publishers,2003.

[24] J. Shen and J. Abraham, ‘ A Novel Functional Test Generation Method for Processors using Commersial ATPG,” in IEEE International Test Conference. pp. 743-752, November 1997.

[25] M. Pedram. Power Minimization in IC Design: Principles and Applications, ACM Transactions on Design Automation of Electronic Systems (TODAES), pp. 3 – 56, 1996.

[26] K. Roy and S. Prasad. Low – Power CMOS VLSI Circuit Design, John Wiley & Sons, 2000.

[27] V.D. Agrawal , C.R. Kime, K.K. Saluja, "A tutorial on Build-In Self_Test, Part 1: Principles" in IEEE Design and Test of Computers, vol.10 no. 2, pp. 69-77, June 1993.

[28] V.D. Agrawal , C.R. Kime, K.K. Saluja, "A tutorial on Build-In Self_Test, Part 2: Applications" in IEEE Design and Test of Computers, vol.10 no. 2, pp. 69-77, June 1993.

[29] P.H. Bardell, W.H. McAnney, J.Savir, "Built-In test for VLSI: Pseudorandom techniwues", John Wiley and Sons, New York, 1987.

[30] IEEE 1500, "Standard for Embedded Core Test", http://grouper.ieee.org/groups/1500.

[31] N. Kranitis, A. Paschalis, D. Gizopoulos, G. Xenoulis, “Software-based Self-Testing of Embedded Processors”, in IEEE Transactions off Computers, vol 54, no 4 pp.461-175., April 2005.

[32] M Psarakis, D. Gizopoulos, M. Hatzimihail, A. Paschalis, A. Ranhunathan, and S. Ravi, “Systematic software-based self-test for pipelined processors”, in IEEE Design Automation Conference. pp. 393-398, 2006

[33] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. S. Reorda, “Micro- processor software-based self-testing,” IEEE Design Test of Computers, vol. 27, no. 3, pp. 4–19, May 2010.

[34] P. Bernardi et al., “Software-based self-test techniques of computational modules in dual issue embedded processors,” in 20th IEEE European Test Symposium (ETS), May 2015, pp. 1–2.

[35] F. Corno, E. Sanchez, M. S. Reorda, and G. Squillero, “Automatic test program generation: a case study,” IEEE Design Test of Computers, vol. 21, no. 2, pp. 102–109, Mar 2004.

[36] P. Georgiou, X. Kavousianos, R. Cantoro, M. S. Reorda, “Fault-Independent Test-Generation for Software-Based Self-Testing”, in 24 th IEEE International Symposium on On-Line Testing and Robust System Design.

[37] J. Shen and J. Abraham, “Native mode functional test generation for microprocessors with applications to self-test and design validation,” in IEEE International Test Conference, pp. 990–998, 1998.

[38] K. Batcher and C. Papachristou, “Instruction randomization self test for processor cores” in IEEE VLSI Test Symposium, pp. 34–40, 1999,.

[39] P. Parvathala, K. Maneparambil, and W. Lindsay, “FRITS – A Microprocessor Functional BIST Method,” in IEEE International Test Conference, pp. 590–598, 2002.

[40] F. Corno, G. Cumani, M. S. Reorda, and G. Squillero, “Fully Automatic Test Program Generation for Microprocessor Cores,” in IEEE Design, Automation and Test in Europe, pp. 1006–1011, 2003.

[41] N. Kranitis, A. Paschalis, D. Gizopoulos, and Y. Zorian, “Instruction-based self-testing of processor cores”, in Journal of Electronic Testing: Theory and Applications, vol. 19, no. 19, pp. 103–112, 2002, special Issue on 20th IEEE VLSI Test Symposium, 2002.

[42] N. Kranitis, G. Xenoulis, A. Paschalis, and D. Gizopoulos, “Application and analysis of rt-level software-based self-testing for embedded processor cores”, in IEEE International Test Conference, pp. 431–440, September-October 2003.

[43] N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, “Software-Based Self-Testing of Embedded Processors”, in IEEE Transactions on Computers, vol. 54, no. 4, pp. 461–475, April 2005.

[44] M. Psarakis, D. Gizopoulos, M. Hatzimihail, A. Paschalis, A. Raghunathan, and S. Ravi, “Systematic software-based self-test for pipelined processors”, in IEEE Design Automation Conference, pp. 393–398, 2006.

[45] D. Gizopoulos, M. Psarakis, M. Hatzimihail, M. Maniatakos, A. Paschalis, A. Raghunathan, and S. Ravi, “Systematic software-based self-test for pipelined processors,” in IEEE Transactions on VLSI Systems, vol. 16, no. 11, pp. 1441–1453, November 2008.

[46] http://eia.udg.es/~gomis/all8051.pdf

[47] http://www.alciro.org/alciro/microcontroladores-8051_24/PSW-Program-Status-Word_363_en.htm

[48] http://www.keil.com/support/man/docs/is51/is51_ov_cpupsw.htm

[49] R.B. Mueller-Thuns, D.G. Saab, R.F. Damiano, J.A. Abraham, “VLSI logic and fault simulation on general-purpose parallel computers”, IEEE, 1993

[50] J.A. Abraham, W.K. Fuchs, “Fault and error models for VLSI”,in IEEE, pp.639-654, 1986.

Σύντομο Βιογραφικό

Ολοκλήρωσα τις προπτυχιακές σπουδές στο Τμήμα Μηχανικών Η/Υ Ιωαννίνων και στη συνέχεια με τη παρούσα εργασία ολοκληρώνω το Μεταπτυχιακό πρόγραμμα σπουδών στο ίδιο τμήμα.