ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

Μεθοδολογίες Δοκιμής για Πολυεπεξεργαστές και Πολυπύρηνα Ολοκληρωμένα Κυκλώματα

Διδακτορική Διατριβή

Ανδρέας Αποστολάκης

Εργαστήριο Υπολογιστικών Συστημάτων

Πειραιάς, Ιούνιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

Μεθοδολογίες Δοκιμής για Πολυεπεξεργαστές και Πολυπύρηνα Ολοκληρωμένα Κυκλώματα

Διδακτορική Διατριβή του
Ανδρέα Αποστολάκη

Τριμελής Συμβουλευτική Επιτροπή

Δημήτριος Γκιζόπουλος, *Αναπληρωτής Καθηγητής Πανεπιστημίου Πειραιώς* (επιβλέπων)
Αντώνης Πασχάλης, *Καθηγητής Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών*
Μαρία Βίρβου, *Αναπληρώτρια Καθηγήτρια Πανεπιστημίου Πειραιώς*

Επταμελής Εξεταστική Επιτροπή

. . . Αντώνιος Πασχάλης *Καθηγητής* *Ε.Κ.Π.Α.*	. . . Μαρία Βίρβου *Αναπληρώτρια Καθηγήτρια* *ΠΑ.ΠΕΙ.*	. . . Δημήτριος Γκιζόπουλος *Αναπληρωτής Καθηγητής* *ΠΑ.ΠΕΙ.*
. . . Νεκτάριος Κοζύρης *Αναπληρωτής Καθηγητής* *Ε.Μ.Π.*	. . . Δημήτριος Σούντρης *Επίκουρος Καθηγητής* *Ε.Μ.Π.*	. . . Χαράλαμπος Κωνσταντόπουλος *Λέκτορας* *ΠΑ.ΠΕΙ.*

. . .
Μιχαήλ Ψαράκης
Λέκτορας
ΠΑ.ΠΕΙ.

Εργαστήριο Υπολογιστικών Συστημάτων

Πειραιάς, Ιούνιος 2009

Περίληψη

Οποιοδήποτε ολοκληρωμένο κύκλωμα από το πιο απλό ως το πιο σύνθετο είναι δυνατόν να παρουσιάσει εσφαλμένη λειτουργία η οποία μπορεί να οφείλεται σε ποικίλους παράγοντες. Κάποια κυκλώματα παρουσιάζουν ελαττώματα (faults) τα οποία εισήχθησαν κατά τη διάρκεια της διαδικασίας κατασκευής (manufacturing process), ενώ βλάβες (failures) μπορεί να παρουσιαστούν και κατά τη διάρκεια της χρήσης τους. Για να καθοριστεί αν ένα ολοκληρωμένο κύκλωμα έχει κατασκευαστεί σωστά, ή ότι συνεχίζει να λειτουργεί σύμφωνα με τον επιθυμητό τρόπο, πρέπει να περάσει από μια διαδικασία ελέγχου ορθής λειτουργίας που ονομάζεται δοκιμή (testing). Παλαιότερα, η ανάγκη υψηλής αξιοπιστίας των ηλεκτρονικών κυκλωμάτων περιοριζόταν μόνο στους τομείς των στρατιωτικών και τραπεζικών εφαρμογών, όπου μια δυσλειτουργία μπορεί να είχε καταστροφικά αποτελέσματα. Καθώς όμως τα ηλεκτρονικά κυκλώματα παρουσιάζουν ευρεία εξάπλωση σε πολλούς τομείς της καθημερινής μας ζωής, ενώ παράλληλα το μέγεθος της τεχνολογίας κατασκευής τους συρρικνώνεται γρήγορα, η δοκιμή κατασκευής (manufacturing testing) λαμβάνει σημαντικό ρόλο στην εξασφάλιση της ποιότητας και της αξιοπιστίας των τελικών προϊόντων.

Στη διδακτορική αυτή διατριβή προτείνονται μεθοδολογίες για την επίλυση του προβλήματος της αυτοδοκιμής (self-test) πολυπύρηνων συστημάτων σε ολοκληρωμένο κύκλωμα (System-on-Chip, SoC) με έμφαση στους πυρήνες ελεγκτών περιφερειακών συσκευών καθώς και τους πολυεπεξεργαστές (multiprocessor) με σύνδεση κοινού διαύλου (shared bus) και σταυρωτού μεταγωγέα (crossbar switch). Στο πρώτο μέρος της διατριβής προτάθηκε μια συστηματική μεθοδολογία για την αυτοδοκιμή SoC με λογισμικό με έμφαση σε πυρήνες περιφερειακών ελεγκτών επικοινωνίας (communication peripheral controllers). Η προτεινόμενη μεθοδολογία βασίζεται στη ντετερμινιστική δοκιμή και είναι γενική και πλήρως εφαρμόσιμη σε μια μεγάλη ομάδα περιφερειακών συσκευών. Επιπλέον, προτάθηκε μια υβριδική μεθοδολογία αυτοδοκιμής SoC με λογισμικό που συνδυάζει την παραπάνω ντετερμινιστική μεθοδολογία με μια ημι-αυτοματοποιημένη, ψευδοτυχαία μεθοδολογία. Στο δεύτερο μέρος της διατριβής παρουσιάζεται μια μεθοδολογία για τον χρονοπρογραμματισμό δοκιμής (test scheduling) πολυεπεξεργαστών με συνδεσμολογία κοινού διαύλου και σταυρωτού μεταγωγέα. Η λύση που προτάθηκε έχει εξαιρετική σημασία τώρα που οι πολυεπεξεργαστές σε ολοκληρωμένο κύκλωμα (Chip Multiprocessors), είτε με πολλαπλούς πυρήνες (multicores), είτε με πολλαπλά νήματα (multithread) χρησιμοποιούνται ευρύτερα από ποτέ. Η παραπάνω μεθοδολογία χρονοπρογραμματισμού δοκιμής χρησιμοποιεί τον έμφυτο παραλληλισμό των αρχιτεκτονικών των πολυεπεξεργαστών για να μειώσει δραματικά το συνολικό χρόνο εκτέλεσης προγραμμάτων αυτοδοκιμής (self-test programs).

Abstract

Any integrated circuit from simplest to complex it is possible to present faulty operation which can happen from various factors. Some circuits present faults which were imported during the manufacturing process, while failures can also presented during their operation. In order to determine if an integrated circuit has been manufactured correctly, or that it continues its functionality according to the specifications, it should successfully complete a process called testing. Few years ago, the need of high reliability in electronic circuits was limited in military and banking applications, where a dysfunction can have devastating results. However the electronic circuits present wide spread in a lot of sectors of our life, while at the same time the size of manufacture technology shrinks fast, the manufacturing testing receives important role in guarantee the quality and the reliability of final products.

In this PhD thesis are proposed methodologies for self-testing problem of System-on-Chip, SoC and especially for communication controllers of these integrated circuits, as well as for symmetric multiprocessors with different interconnection schemes such as shared bus and crossbar switch. In the first chapters of this thesis we proposed a systematic methodology for self-testing of communication peripheral controllers that are integrated in a SoC. The proposed methodology is based in deterministic testing and is general and completely applicable in any communication peripheral. Moreover, we proposed a hybrid self-test methodology for SoC that combines the deterministic methodology with a semi-automated, pseudorandom methodology. In the second part of this thesis we presente a methodology for test routines scheduling of symmetric multiprocessors with different interconnection schemes such as shared bus and crossbar switch. The solution that we proposed has exceptional importance now that chip multiprocessors, or with multiple cores (multicores), or with multiple threads (multithread) are used more widely than never. The test scheduling methodology uses the parallelism of multiprocessors architecture in order to decrease dramatically the total test application time of the self-test programs.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον επιβλέποντά μου, κ. Δημήτρη Γκιζόπουλο Αναπληρωτή Καθηγητή του Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς για την υποστήριξη και την καθοδήγησή του όλα αυτά τα χρόνια από το μεταπτυχιακό "Προηγμένα Συστήματα Πληροφορικής" και την κατεύθυνση της "Τεχνολογίας Ενσωματωμένων Υπολογιστικών Συστημάτων". Οι γνώσεις του με βοήθησαν όλες εκείνες τις στιγμές που τα προβλήματα φαινόντουσαν ανυπέρβλητα και η στάση του με ενέπνευσε τόσο σε ερευνητικό όσο και σε ανθρώπινο επίπεδο. Επίσης θα ήθελα να τον ευχαριστήσω για τις πολύτιμες εμπειρίες από την συμμετοχή και την παρουσίαση των εργασιών της διατριβής σε διεθνή συνέδρια.

Μεγάλο μέρος αυτής της δουλειάς είναι αποτέλεσμα κοινής προσπάθειας με τον κ. Μιχάλη Ψαράκη, Λέκτορα του Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς τον οποίο και ευχαριστώ ιδιαίτερα για τις συζητήσεις και τη βοήθειά του, καθώς και για την εμπιστοσύνη που μου έδειξε όλα αυτά τα χρόνια.

Θα ήθελα ακόμα να ευχαριστήσω και τα δύο άλλα μέλη της τριμελούς συμβουλευτικής επιτροπής, τον κ. Αντώνη Πασχάλη, Καθηγητή του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Αθηνών και την κ. Μαρία Βίρβου, Αναπληρώτρια Καθηγήτρια του Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς, για την υποστήριξή τους κατά την εκπόνηση της διατριβής.

Επίσης θα ήθελα να ευχαριστήσω το ΠΕΝΕΔ για την φυσική και οικονομική ενίσχυση που μου παρείχε κατά την διάρκεια εκπόνησης της διδακτορικής μου διατριβής.

Θα ήθελα να ευχαριστήσω όλα τα μέλη του εργαστηρίου για τη συνεργασία και τη φιλία τους όλα αυτά τα χρόνια. Επιπλέον, θα ήθελα να ευχαριστήσω τους φίλους μου Βασίλη, Νίκο, Αλέκο, Μάτα, Γιώργο που ήταν πάντα δίπλα μου. Πρόσθετα ευχαριστώ τον Βασίλη για τη βοήθεια και συνεισφορά του στην θεωρία των ουρών αναμονής. Επιπλέον ευχαριστώ την Δέσποινα για τη πολύτιμη βοήθεια της με το κείμενο του διδακτορικού.

Θα ήθελα να ευχαριστήσω και τα αδέρφια μου Θεολόγο και Δημήτρη που είναι πάντα δίπλα μου με την αγάπη και τη φροντίδα τους. Τέλος ευχαριστώ τους γονείς μου, που μου προσέφεραν την υποστήριξή τους, υλική και ηθική και με στήριξαν τόσο πολύ σε κάθε βήμα της ζωής μου, ώστε η κάθε μου προσπάθεια να αποβεί καθοριστική για την πορεία της μετέπειτα ζωής μου. Τους ευχαριστώ όμως κυρίως, γιατί μου έδειξαν τον δρόμο της ζωής και με δίδαξαν πως πρέπει κάποιος να αντιμετωπίζει με σεμνότητα τις επιτυχίες και με θάρρος τις αποτυχίες.

Ανδρέας Αποστολάκης

... αφιερώνεται στους γονείς μου

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή στην Ερευνητική Περιοχή και Συνεισφορά της Διδακτορικής Διατριβής

The really valuable thing in the pageant of human life seems to me not the State but the creative, sentient individual, the personality...

Albert Einstein

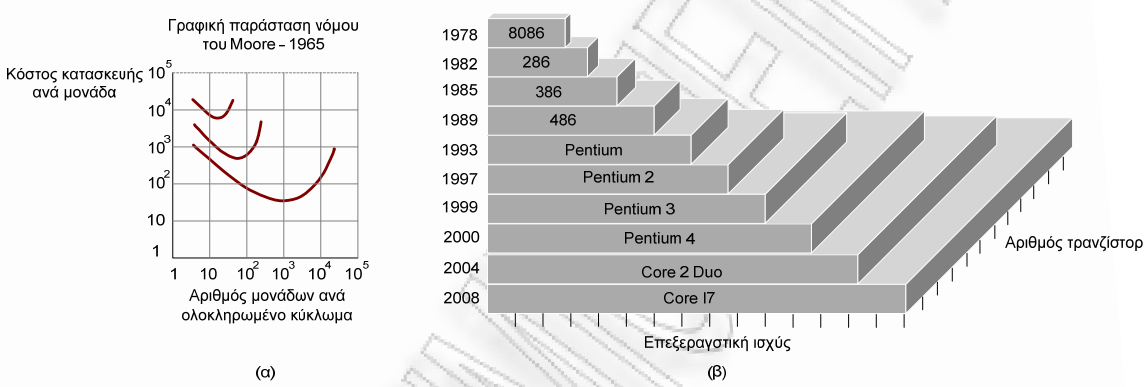
Η τεχνολογία και η πρόοδος της ανθρωπότητας ήταν ανέκαθεν έννοιες τόσο στενά δεμένες μεταξύ τους που κάθε ιστορική περίοδος δανειζόταν το όνομα της από την αντίστοιχη τεχνολογία, ενώ η μετάβαση από την μια ιστορική περίοδο στην επόμενη αποδίδεται από τον όρο "επανάσταση". Πάντα αυτές οι μεταβάσεις-επαναστάσεις σημάδευαν ένα σημαντικό σημείο καμπής στης ζωές των ανθρώπων που βίωναν πριν και μετά την εμφάνιση τους. Η εξέλιξη της τεχνολογίας των *ολοκληρωμένων κυκλωμάτων* (*integrated circuits, ICs*) [1], [2], [3] αποτέλεσε αναμφίβολα το εφαλτήριο της σημαντικότερης τεχνολογικής επανάστασης που συντελέστηκε τον 20ο αιώνα και δεν είναι άλλη από την επανάσταση των υπολογιστών και της επιστήμης της πληροφορικής. Τα κύρια χαρακτηριστικά των ολοκληρωμένων κυκλωμάτων που διαδραμάτισαν τον σημαντικότερο ρόλο στην ραγδαία εξέλιξή τους είναι η υψηλή υπολογιστική ισχύς και η ευκολία κατασκευής τους. Τα χαρακτηριστικά αυτά οφείλονται στην *τεχνολογία των ημιαγωγών* (*semiconductor technology*) που χρησιμοποιείται για την κατασκευή των ολοκληρωμένων

κυκλωμάτων. Τα τελευταία χρόνια είμαστε μάρτυρες μιας συνεχούς και εκτεταμένης ανάπτυξης της μικροηλεκτρονικής. Βασικό στοιχείο αυτής της ανάπτυξης είναι η συνεχής συρρίκνωση των ολοκληρωμένων κυκλωμάτων ως αποτέλεσμα των απαιτήσεων για ηλεκτρονικές διατάξεις υψηλότερων επιδόσεων, μειωμένου κόστους και χαμηλότερης κατανάλωσης ισχύος.

Η *κλίμακα ολοκλήρωσης* (*scale of integration*), που εκφράζεται σε αριθμό τρανζίστορ ανά ολοκληρωμένο κύκλωμα, καθώς και το ελάχιστο *χαρακτηριστικό μέγεθος* (*feature size*) ενός ολοκληρωμένου κυκλώματος, είναι δύο χαρακτηριστικά μεγέθη που ποσοτικοποιούν την εξέλιξη της τεχνολογίας των ολοκληρωμένων κυκλωμάτων τις τελευταίες δεκαετίες. Στις αρχές της δεκαετίας του 1960 τα ολοκληρωμένα κυκλώματα *μικρής κλίμακας ολοκλήρωσης* (*Small Scale of Integration, SSI*) περιείχαν από 1 έως 100 τρανζίστορ ανά ολοκληρωμένο κύκλωμα, με ελάχιστο *χαρακτηριστικό μέγεθος* (*feature size*) 30 μm. Στο τέλος της δεκαετίας του 1970, η πυκνότητα αυξήθηκε στα επίπεδα του 100 έως 1000 τρανζίστορ ανά ολοκληρωμένο κύκλωμα (*Medium Scale Integration, MSI*), με χαρακτηριστικό μέγεθος 3 μm, ενώ αργότερα στα 10^3 έως 10^5 τρανζίστορ (*Large Scale Integration, LSI*) με χαρακτηριστικό μέγεθος 1.5 μm. Προς το τέλος της δεκαετίας του 1980, η πυκνότητα διαμορφώθηκε στα 10^5 έως 10^6 τρανζίστορ ανά ολοκληρωμένο κύκλωμα (*Very Large Scale Integration, VLSI*) με χαρακτηριστικό μέγεθος μικρότερο 0.8 μm, ενώ στο τέλος της δεκαετίας του 1990 η πυκνότητα κυμαίνεται στα επίπεδα των 10^7 έως 10^9 τρανζίστορ ανά ολοκληρωμένο κύκλωμα (*Ultra Large Scale Integration, ULSI*), με χαρακτηριστικό μέγεθος 0.25 μm. Στις αρχές της νέας χιλιετίας οι βιομηχανίες των ημιαγωγών ξεκίνησαν την παραγωγή των λεγόμενων "nanochips", κυκλωμάτων δηλαδή, με χαρακτηριστικό μέγεθος μικρότερο από 0,09 μm. Οι απαιτήσεις για αύξηση της πυκνότητας ολοκλήρωσης και μείωση του κόστους έδωσαν εντυπωσιακή ώθηση, τα τελευταία χρόνια, στο ρυθμό μείωσης των διαστάσεων.

Η πρόοδος αυτή επαληθεύει την πρόβλεψη που έκανε το 1965, ακριβώς 4 χρόνια μετά την εφεύρεση του πρώτου ολοκληρωμένου κυκλώματος, ο Gordon Moore, συνιδρυτής της Intel και νυν επίτιμος πρόεδρός της, βασιζόμενος σε ένα ημι-λογαριθμικό διάγραμμα Εικόνα 1.1(α), παρατήρησε πως ο αριθμός των τρανζίστορ ανά ολοκληρωμένο κύκλωμα θα διπλασιάζεται κάθε περίπου 18 μήνες. Η πρόβλεψη αυτή έμεινε γνωστή ως "νόμος του Moore" [4]. Ο πρώην διευθυντής της Intel, David House στα τέλη της δεκαετίας του 1980 ήταν αυτός που ανέφερε το χρονικό διάστημα των 18 μηνών το οποίο αφορούσε τον διπλασιασμό της υπολογιστικής ισχύος, όπως δείχνει η Εικόνα 1.1(β). Χαρακτηριστικό παράδειγμα επιβεβαίωσης του νόμου του Moore αποτελεί η εξέλιξη των επεξεργαστών της ίδιας της Intel. Στις αρχές της δεκαετίας του 1970, οι επεξεργαστές της περιείχαν μόνο μερικές εκατοντάδες τρανζίστορ ανά ολοκληρωμένο κύκλωμα, ενώ στην δεκαετία του 1990 ο επεξεργαστής Pentium περιείχε ήδη τρία εκατομμύρια τρανζίστορ [5]. Ο επεξεργαστής Pentium 4 που εμφανίστηκε στην αγορά το 2000 περιείχε 42 εκατομμύρια τρανζίστορ, ενώ ο πρόσφατος Core 2 Quad περιέχει 582 εκατομμύρια τρανζίστορ. Στα τέλη του 2000, η Intel ανακοίνωσε την κατασκευή, σε ερευνητικό επίπεδο, τρανζίστορ με ελάχιστο

χαρακτηριστικό μέγεθος 30nm και πάχος που αντιστοιχεί σε 3 στρώματα ατόμων, ενώ σήμερα οι εμπορικοί επεξεργαστές με τέσσερις πυρήνες κατασκευάζονται ήδη στα 45nm. Μελλοντικές προβλέψεις μιλούν για σχεδίαση στα 10nm το 2022 [6]. Τον ίδιο αλματώδη ρυθμό εξέλιξης έχουν και οι επεξεργαστές των άλλων εταιριών (IBM, AMD, Sun, Philips κλπ), αφού με την ολοκλήρωση πολύ περισσότερων τρανζίστορ σε ένα και μόνο ολοκληρωμένο κύκλωμα ικανοποιείται το αίτημα για μείωση του κόστους.



Εικόνα 1.1 (α) Αρχικό διάγραμμα του Gordon Moore [4], (β) Αναλογία επεξεργαστικής ισχύς και πλήθους τρανζίστορ ανά επεξεργαστή

Η *ολοκλήρωση πολύ μεγάλης κλίμακας* (*Very Large Scale Integration, VLSI*) έχει αποτελέσει έναν από τους κυριότερους λόγους για την ανάπτυξη της έρευνας στο χώρο της *δοκιμής* (*testing*)[1], αλλά και για τις δυσκολίες εφαρμογής αυτής, λόγω του περιορισμένου αριθμού ακροδεκτών σε ένα κύκλωμα VLSI. Εάν τα διάφορα στάδια κατασκευής του ολοκληρωμένου (κατασκευή *πλακιδίων* (*wafer*) χάραξη του τσιπ, συνδέσεις, συσκευασία, κλπ) ήταν τέλεια, τότε δεν θα υπήρχε καμία ανάγκη για τη δοκιμή του κυκλώματος. Κάθε κύκλωμα θα ήταν πλήρως λειτουργικό, υποθέτοντας ότι η σχεδίαση του είχε επαληθευτεί πλήρως. Τα διάφορα στάδια κατασκευής όμως δεν είναι τέλεια και η παραμικρή φυσική ατέλεια μπορεί να οδηγήσει σε λανθασμένη λειτουργία του προϊόντος. Συνεπώς, η δοκιμή όλων των ολοκληρωμένων κυκλωμάτων που κατασκευάζονται είναι απολύτως απαραίτητη.

Τι σημαίνει όμως δοκιμή των ολοκληρωμένων κυκλωμάτων και που μπορεί να οφείλεται η λανθασμένη λειτουργία ενός κυκλώματος; Δοκιμή ενός ολοκληρωμένου κυκλώματος είναι μία διαδικασία κατά την οποία ένα σύστημα εξετάζεται εάν λειτουργεί σωστά, ή λανθασμένα. Η λανθασμένη λειτουργία ενός κυκλώματος μπορεί να οφείλεται σε λάθη σχεδίασης, λάθη στην λειτουργία των *αυτόματων εργαλείων λογισμικού* (*Computer-Aided Design tools*) που χρησιμοποιούνται στην διαδικασία σχεδίασης, λάθη στην διαδικασία κατασκευής που μπορεί να

[1] Ο όρος testing αναφέρεται πολλές φορές στην βιβλιογραφία και ως "έλεγχος ορθής λειτουργίας" για να διαχωριστεί από τον έλεγχο = control.

οφείλονται σε ατέλειες των υλικών, ή σε ατέλειες των συσκευών – μηχανημάτων που χρησιμοποιούνται για την κατασκευή των ολοκληρωμένων κυκλωμάτων, στην *γήρανση* (*aging*), ή *φθορά* (*wearout*) των υλικών του κυκλώματος με την πάροδο του χρόνου και τέλος λάθη που προέρχονται από περιβαλλοντικούς παράγοντες (ακτινοβολίες, θερμοκρασία, δονήσεις, σωματίδια σκόνης) ανάλογα με το περιβάλλον κατασκευής και την εφαρμογή στην οποία χρησιμοποιείται το ολοκληρωμένο κύκλωμα.

Η εμφάνιση οποιουδήποτε από τα παραπάνω λάθη, μπορεί να οδηγήσει από την αχρήστευση ενός ολοκληρωμένου κυκλώματος μέχρι και την πλήρη καταστροφή του συστήματος στο οποίο μετέχει. Οι παραπάνω συνέπειες έχουν σαν αποτέλεσμα μεγάλες οικονομικές ζημιές, παρόλο που το πραγματικό κόστος ενός ολοκληρωμένου διαφέρει από σχεδίαση σε σχεδίαση και καθορίζεται κυρίως από το πλήθος όμοιων ολοκληρωμένων κυκλωμάτων που θα παραχθούν στην ίδια γραμμή παραγωγής. Χαρακτηριστικό παράδειγμα αποτελεί το σχεδιαστικό λάθος και όχι κατασκευαστικό, στη μονάδα κινητής υποδιαστολής του πρώτου επεξεργαστή Pentium το 1994 [7] το οποίο οδήγησε σε μεγάλες οικονομικές ζημιές για την Intel.

Η σπουδαιότητα της δοκιμής αναδεικνύεται 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 της ορθής του λειτουργίας. Η Intel αναφέρει ότι ο συνδυασμός της *επαλήθευσης* (*verification*) και της *δοκιμής κατασκευής* (*manufacturing testing*) αποτελεί το κύριο κόστος κατά την κατασκευή ενός ολοκληρωμένου κυκλώματος και όχι η αξίας δύο δισεκατομμυρίων δολαρίων γραμμή παραγωγής της [8].

Αν και το κόστος της δοκιμής είναι ιδιαίτερα υψηλό, στη βιομηχανία μικροηλεκτρονικής είναι ευρέως αποδεκτό ότι τα ολοκληρωμένα κυκλώματα επιβάλλεται να δοκιμαστούν πριν τοποθετηθούν στις *πλακέτες τυπωμένων κυκλωμάτων* (*Printed Circuit Boards, PCBs*), οι οποίες στη συνέχεια πρέπει να δοκιμαστούν και αυτές προτού ενσωματωθούν στα τελικά συστήματα. Για το κόστος δοκιμής των προϊόντων μικροηλεκτρονικής ισχύει ο "Κανόνας του Δέκα" (Rule of Ten) [9]. Εάν ένα ελάττωμα σε ένα ολοκληρωμένο κύκλωμα δεν βρεθεί κατά τη δοκιμή, τότε το κόστος ανίχνευσης του ελαττώματος είναι 10 φορές μεγαλύτερο στο επίπεδο των καρτών τυπωμένου κυκλώματος απ' ότι στο επίπεδο του ολοκληρωμένου κυκλώματος. Ομοίως, εάν ένα ελάττωμα δεν βρεθεί κατά τη δοκιμή στο επίπεδο των καρτών τυπωμένου κυκλώματος, η ανίχνευση του έχει 10 φορές μεγαλύτερο κόστος στο επίπεδο τους συστήματος απ' ότι στο επίπεδο των καρτών τυπωμένου κυκλώματος. Μερικοί υποστηρίζουν ότι ο Κανόνας του Δέκα πρέπει να μετονομαστεί σε Κανόνα του Είκοσι, καθώς σήμερα τα ολοκληρωμένα κυκλώματα, οι πλακέτες τυπωμένων κυκλωμάτων και τα συστήματα είναι πολύ πιο πολύπλοκα από ότι ήταν όταν δηλώθηκε αρχικά αυτός ο εμπειρικός κανόνας. Συνεπώς, όσο νωρίτερα ανιχνεύεται ένα ελάττωμα, τόσο μικρότερο είναι το κόστος επιδιόρθωσης του και όσο αποδοτικότερη είναι η δοκιμή, τόσο ποιοτικότερο είναι το προϊόν ως προς τον τελικό χρήστη.

Η ποιότητα μιας *διαδικασίας παραγωγής (manufacturing process)* υπολογίζεται με βάση την *εσοδεία* (*yield*). Η εσοδεία υπολογίζεται ως ο λόγος των ολοκληρωμένων κυκλωμάτων που είναι *δεν περιέχουν ατέλειες* (*defect free*) προς το συνολικό αριθμό των ολοκληρωμένων κυκλωμάτων που παρήχθησαν από μια διαδικασία παραγωγής. Το *επίπεδο ατέλειας* (*Defect Level, DL*) είναι το μέρος των ελαττωματικών κυκλωμάτων τα οποία ολοκλήρωσαν επιτυχώς όλες τις δοκιμές, δηλαδή διέφυγαν του ελέγχου. Οι τιμές επιπέδου ατέλειας μετρώνται συνήθως από το *πλήθος των ατελειών ανά εκατομμύριο* (*Defective Parts Per Million. DPPM*). Η επιθυμητή τιμή είναι λιγότερο από 200 DPPM [10], ενώ ο αριθμός DPPM αποτελεί συνάρτηση της εσοδείας και της ποιότητας της δοκιμής. Το επίπεδο ατέλειας γίνεται γνωστό αφού τα κυκλώματα φθάσουν στην αγορά, τοποθετηθούν σε συστήματα και παρουσιάσουν αστοχίες στις υπηρεσίες που πρέπει να παρέχουν.

Η ποιότητα της δοκιμής καθορίζεται από το *ποσοστό κάλυψης ελαττωμάτων* (*fault coverage*), το οποίο ορίζεται ως το ποσοστό των συνολικών ελαττωμάτων του κυκλώματος που ανιχνεύονται από ένα σύνολο *διανυσμάτων δοκιμής* (*test patterns*)[2]. Τα *ελαττώματα* (*faults*) αντιπροσωπεύουν το αποτέλεσμα των *φυσικών ατελειών* (*physical faults*) στην συμπεριφορά του μοντέλου ενός συστήματος. Η υψηλή ποιότητα δοκιμής απαιτεί την χρήση αποτελεσματικών μοντέλων ελαττωμάτων. Το τεράστιο πλήθος των φυσικών ατελειών δεν επιτρέπει όμως την αποτελεσματική και ενιαία αντιμετώπιση τους από ένα μαθηματικό μοντέλο στα πλαίσια της δοκιμής των ολοκληρωμένων κυκλωμάτων. Αυτόν ακριβώς τον σκοπό εξυπηρετεί η ύπαρξη των *μοντέλων ελαττωμάτων* (*fault models*). Τα *λογικά μοντέλα ελαττωμάτων* (*logical fault models*) προσεγγίζουν την συμπεριφορά των *φυσικών ατελειών* (*physical defects*), καθιστώντας υπολογιστικά εφικτή την εξαγωγή διανυσμάτων δοκιμής. Η εκτίμηση των πιθανών ελαττωμάτων σε ένα ψηφιακό κύκλωμα πραγματοποιείται προκειμένου να καθοριστεί ένα ελάχιστο σύνολο διανυσμάτων δοκιμής το οποίο θα εξετάσει εάν τα ελαττώματα είναι ή δεν είναι παρόντα σε ένα συγκεκριμένο κατασκευασμένο ολοκληρωμένο κύκλωμα. Εάν κανένα από τα προκαθορισμένα ελαττώματα δεν ανιχνευθεί, τότε το κύκλωμα θεωρείται *χωρίς ελαττώματα* (*fault free*).

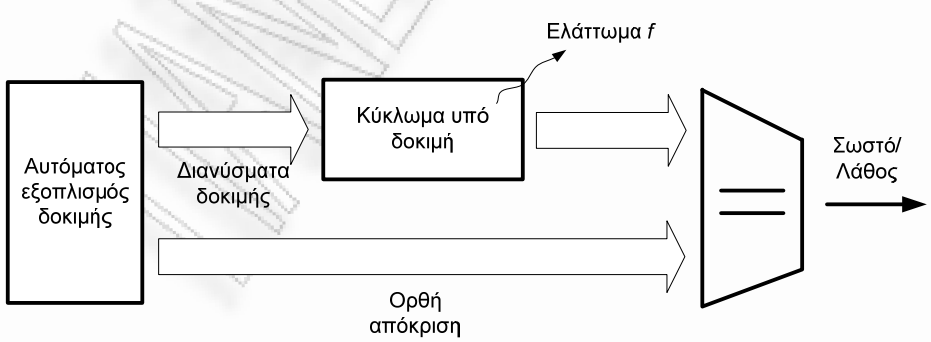
Πολλές φυσικές ατέλειες συνήθως αναπαρίστανται από το ίδιο λογικό ελάττωμα [11], όπως επίσης μερικές ατέλειες μπορεί να αναπαρασταθούν από περισσότερα του ενός ελαττώματα. Στη μοντελοποίηση κάνουμε ένα διαχωρισμό μεταξύ της λογικής συνάρτησης και του *χρονισμού* (*timing*). Έτσι, διακρίνουμε ελαττώματα που επηρεάζουν τη λογική συνάρτηση, όπως τα *ελαττώματα προσκόλλησης* (*stuck-at faults*) και ελαττώματα που επηρεάζουν την ταχύτητα λειτουργίας του κυκλώματος, όπως τα *ελαττώματα καθυστέρησης μετάβασης* (*transition fault*), και τα *ελαττώματα καθυστέρησης μονοπατιού* (*path delay faults*). Το πιο διαδεδομένο μοντέλο ελαττώματος, που πρωτοπαρουσιάστηκε εδώ και περισσότερα από 40 χρόνια και χρησιμοποιείται για τα ελαττώματα στα ψηφιακά κυκλώματα, είναι το *μοντέλο ελαττώματος*

[2] Οι τιμές που εφαρμόζονται στις εισόδους του κυκλώματος υπό δοκιμή καλούνται διανύσματα δοκιμής.

προσκόλλησης (*stuck-at fault model*) [12]. Το μοντέλο αυτό υποθέτει ότι οποιαδήποτε φυσική ατέλεια σε ένα ψηφιακό κύκλωμα οδηγεί έναν κόμβο στο κύκλωμα (την είσοδο ή την έξοδο μιας πύλης) να παραμένει *προσκολλημένο σε λογική κατάσταση "0"* (*stuck-at-0, SA0*), ή να παραμένει *προσκολλημένο σε λογική κατάσταση "1"*, (*stuck-at-1, SA1*).

Η *παραγωγή διανυσμάτων δοκιμής* (*test pattern generation*) είναι μια διαδικασία κατά την οποία παράγονται τα κατάλληλα διανύσματα εισόδου για να δοκιμαστεί ένα δεδομένο ψηφιακό κύκλωμα. Η *εξαντλητική δοκιμή* (*exhaustive testing*), δηλαδή η εφαρμογή όλων των πιθανών διανυσμάτων εισόδου, είναι σήμερα απαγορευτικά χρονοβόρα. Επομένως, κρίνεται απαραίτητη η ύπαρξη μιας αποτελεσματικής μεθόδου, σύμφωνα με την οποία ένα περιορισμένο – *μη εξαντλητικό* (*nonexaustive*) σύνολο δοκιμής πρέπει να εφαρμοστεί στο εκάστοτε ολοκληρωμένο κύκλωμα. Η *αυτόματη παραγωγή διανυσμάτων δοκιμής* (*Automatic Test Pattern Generation, ATPG*), είναι πλέον επιβεβλημένη καθώς ο αριθμός των πυλών σε ένα ολοκληρωμένο κύκλωμα αυξάνεται ολοένα και περισσότερο στις μέρες μας.

Η θέση των ελαττωμάτων εξαρτάται από το εκάστοτε ολοκληρωμένο κύκλωμα. Η δοκιμή κατασκευής των ολοκληρωμένων κυκλωμάτων πραγματοποιείται με την βοήθεια του *αυτόματου εξοπλισμού δοκιμής* (*Automatic Test Equipment, ATE*), ή *ελεγκτή* (*tester*), όπως παρουσιάζεται στην Εικόνα 1.2. Ο εξωτερικός ελεγκτής εφαρμόζει ένα σύνολο *διανυσμάτων δοκιμής* (*test vectors*) στο ολοκληρωμένο κύκλωμα με στόχο την *ενεργοποίηση του ελαττώματος* (*fault activation*) και την διάδοση της *λανθασμένης τιμής*, ή των *λανθασμένων τιμών* (*fault propagation*) στις κύριες εξόδους του κυκλώματος. Σε κάθε βήμα της δοκιμής κάθε *απόκριση δοκιμής* (*test response*) πρέπει να συγκριθεί με την αναμενόμενη. Ο έλεγχος των αποκρίσεων απαιτεί την προγενέστερη γνώση των *ορθών ή αναμενόμενων αποκρίσεων δοκιμής* (*correct responses*) που έχουν προκύψει από την προσομοίωση της σχεδίασης του κυκλώματος και την σύγκριση αυτών με τις πραγματικές αποκρίσεις του κυκλώματος υπό δοκιμή. Οι ορθές αυτές αποκρίσεις της δοκιμής αποθηκεύονται στη μνήμη του εξωτερικού ελεγκτή για σύγκριση με τις πραγματικές αποκρίσεις του κυκλώματος υπό δοκιμή.



Εικόνα 1.2 Διαδικασία δοκιμής

Ανάλογα με το εάν ένα σύνολο δοκιμής εφαρμόζεται στην πραγματική συχνότητα λειτουργίας του ολοκληρωμένου κυκλώματος υπό δοκιμή ή όχι, η δοκιμή διακρίνεται σε *στατική δοκιμή* (*static testing*) και δοκιμή στην *ταχύτητα λειτουργίας* (*at-speed testing*). Η στατική δοκιμή εφαρμόζει τα διανύσματα δοκιμής σε συχνότητες μικρότερες από την συχνότητα λειτουργίας του κυκλώματος. Η εφαρμογή της στατικής δοκιμής, οφείλεται στους περιορισμούς που θέτουν οι εξωτερικοί ελεγκτές, δεδομένου ότι ακόμα και οι πιο προηγμένοι εξωτερικοί ελεγκτές δεν μπορούν να λειτουργήσουν σε τόσο υψηλές συχνότητες όσο αυτές των ολοκληρωμένων κυκλωμάτων. Επίσης, το δραματικά αυξανόμενο κόστος των εξωτερικών ελεγκτών υψηλών συχνοτήτων και μεγάλου μεγέθους μνήμης μπορεί να φτάσει και τα 20 εκατομμύρια δολάρια μέχρι το 2014 [13]. Στον αντίποδα, η δοκιμή στην *ταχύτητα λειτουργίας* (*at-speed*) είναι σε θέση να ανιχνεύσει περισσότερα ελαττώματα που παρουσιάζονται στα σύγχρονα κυκλώματα υψηλών ταχυτήτων, επιτυγχάνοντας χαμηλά επίπεδα DPPM [14], [15]. Επιπλέον, είναι ιδιαίτερα χρήσιμη στα πλαίσια του *χαρακτηρισμού των ολοκληρωμένων κυκλωμάτων σύμφωνα με την ταχύτητα λειτουργίας τους* (*speed spinning*), ώστε να διαχωριστούν τα ολοκληρωμένα κυκλώματα υψηλής απόδοσης από τα υπόλοιπα που λειτουργούν ορθά μεν, αλλά σε μικρότερες συχνότητες ρολογιού.

Διάφορες μελέτες πρότειναν να δημιουργηθεί μια σχέση κόστους, η οποία να σχετίζεται με την διαδικασία της δοκιμής. Ο όρος *δοκιμαστικότητα* (*testability*) είναι μία χαρακτηριστική ιδιότητα ενός κυκλώματος που επηρεάζει το κόστος της δοκιμής. Οι δύο παράγοντες που επηρεάζουν την δοκιμαστικότητα είναι η *ελεγξιμότητα* (*controllability*) και η *παρατηρησιμότητα* (*observability*). Η ελεγξιμότητα είναι ένα μέτρο που καθορίζει πόσο εύκολα ένας δεδομένος κόμβος σε ένα ψηφιακό κύκλωμα μπορεί να τεθεί σε λογική τιμή "0", ή σε λογική τιμή "1" εφαρμόζοντας τιμές στις *κύριες εισόδους* (*primary inputs*) του κυκλώματος. Ομοίως, η έννοια της παρατηρησιμότητας χρησιμοποιείται για να δηλώσει πόσο εύκολα η τιμή ενός δεδομένου κόμβου (λογική τιμή "0", ή λογική τιμή "1") μπορεί να διαδοθεί στις κύριες εξόδους (*primary outputs*). Για να κρατηθεί το κόστος σε λογικά επίπεδα, εφαρμόζονται τεχνικές σχεδίασης που διευκολύνουν την δοκιμή του κυκλώματος, και οι οποίες ονομάζονται *τεχνικές σχεδίασης για δοκιμαστικότητα* (*Design For Testability, DFT*). Η συστηματική σχεδίαση για αυξημένη δοκιμαστικότητα, η οποία συνήθως ενσωματώνεται στα αυτόματα *εργαλεία σχεδίασης* (*CAD tools*), ονομάζεται *σύνθεση δοκιμής* (*test synthesis*).

Η κυρίαρχη τεχνική των περισσότερων εργαλείων σύνθεσης δοκιμής, είναι η *σχεδίαση σάρωσης* (*scan design*) [16]. Σχεδίαση σάρωσης είναι ο διαχωρισμός των ακολουθιακών μονάδων του κυκλώματος από τις συνδυαστικές μονάδες κατά την διάρκεια της δοκιμής. Ακολουθιακά στοιχεία, όπως flip-flops, ενώνονται μεταξύ τους για να σχηματίσουν μία αλυσίδα υπό την μορφή *καταχωρητή ολίσθησης* (*shift registers*) όταν ενεργοποιηθεί η διαδικασία της δοκιμής. Η αλυσίδα αυτή ονομάζεται *αλυσίδα σάρωσης* (*scan chain*). Η τεχνική διαχωρίζει το κύκλωμα σε ένα σύνολο υπομονάδων οι είσοδοι και οι έξοδοι των οποίων ενώνονται απευθείας

στην αλυσίδα σάρωσης. Η σχεδίαση σάρωσης παρουσιάζει όμως και κάποια σημαντικά μειονεκτήματα. Οι αλυσίδες σάρωσης αυξάνουν την επιφάνεια που καταλαμβάνει το ολοκληρωμένο κύκλωμα και είναι πιθανό να επηρεάσουν αρνητικά την ταχύτητα του κυκλώματος. Επίσης, σε κυκλώματα με μεγάλο αριθμό ακολουθιακών στοιχείων, οι αλυσίδες σάρωσης είναι ιδιαίτερα μεγάλες, το οποίο οδηγεί σε μεγάλο χρόνο ολίσθησης των δεδομένων δοκιμής από την είσοδο της αλυσίδας (δεδομένα εισόδου), προς την έξοδο της αλυσίδας (δεδομένα εξόδου) και απαιτεί εξωτερικούς ελεγκτές με πολύ μεγάλο μέγεθος μνήμης.

Μια επίσης σημαντική μέθοδος για την αύξηση της δοκιμαστικότητας είναι η *ενσωματωμένη αυτοδοκιμή* (*Built-In Self-Test, BIST*), η οποία αναφέρεται σε τεχνικές που επιτρέπουν σε ένα ολοκληρωμένο κύκλωμα να πραγματοποιήσει δοκιμή στον εαυτό του [16], [17], [18]. Σύμφωνα με την μεθοδολογία της ενσωματωμένης αυτοδοκιμής, τα διανύσματα δοκιμής καθώς και οι αποκρίσεις του κυκλώματος, παράγονται και συλλέγονται αντίστοιχα με την χρήση επιπλέον υλικού μέσα στο ίδιο το ολοκληρωμένο κύκλωμα. Στην πραγματικότητα το επιπλέον υλικό αναλαμβάνει τον ρόλο του συστήματος δοκιμής, με αποτέλεσμα ο εξοπλισμός δοκιμής να ενσωματώνεται στο κύκλωμα. Η ενσωματωμένη αυτοδοκιμή προσφέρει πολλά πλεονεκτήματα έναντι της εξωτερικής δοκιμής με *αυτόματο εξοπλισμό δοκιμής* (*Automatic Test Equipment, ATE*). Στην ενσωματωμένη αυτοδοκιμή, το κύκλωμα δοκιμής ενσωματώνεται μέσα στο ολοκληρωμένο κύκλωμα, αποφεύγοντας την χρήση εξωτερικού ελεγκτή, παράγοντας μόνο του τα διανύσματα της δοκιμής. Αυτό είναι ιδιαίτερα χρήσιμο για κυκλώματα υψηλών ταχυτήτων τα οποία σε αντίθετη περίπτωση θα απαιτούσαν εξωτερικούς ελεγκτές πολύ υψηλού κόστους. Επιπλέον, μπορεί να εφαρμοστεί στην πραγματική συχνότητα λειτουργίας του κυκλώματος υπό δοκιμή, παρέχοντας ιδιαίτερα πλεονεκτήματα ως προς το ποσοστό κάλυψης ελαττωμάτων, τα οποία σε αντίθετη περίπτωση ανιχνεύονταν μόνο όταν το κύκλωμα υπό δοκιμή συνδεόταν στο τελικό σύστημα. Πειραματικές μετρήσεις [19], [20], [21] έχουν αποδείξει πως όταν η δοκιμή πραγματοποιείται στην πραγματική συχνότητα λειτουργίας του κυκλώματος, ανιχνεύονται ελαττώματα που δεν είναι δυνατό να ανιχνευτούν από μια τυπική δοκιμή. Τέλος, ένα κύκλωμα με δυνατότητα αυτοδοκιμής έχει την ικανότητα να ελέγχει τον εαυτό του ακόμα και όταν έχει τοποθετηθεί στο τελικό σύστημα. Αυτό είναι ιδιαίτερα χρήσιμο τόσο για *περιοδική δοκιμή* (*periodic testing*), όσο και σποραδικούς *διαγνωστικούς ελέγχους* (*diagnostic tests*) στο πεδίο εφαρμογής.

Όμως, η ενσωματωμένη αυτοδοκιμή έχει και την αρνητική πλευρά της. Για την υλοποίηση μίας αρχιτεκτονικής ενσωματωμένης αυτοδοκιμής είναι δυνατόν είτε να χρησιμοποιηθούν υπάρχουσες μονάδες του κυκλώματος, η λειτουργία των οποίων θα επαναπροσδιορίζεται κατά την διάρκεια της δοκιμής, είτε να προστεθούν νέες μονάδες. Και στις δύο περιπτώσεις, η επιφάνεια που καταλαμβάνει το ολοκληρωμένο κύκλωμα αυξάνεται και η απόδοσή του μπορεί να μειωθεί λόγω τροποποιήσεων που επιβαρύνουν τα κρίσιμα μονοπάτια του κυκλώματος. Με

άλλα λόγια, η τεχνική ενσωματωμένης αυτοδοκιμής ως μία υποπερίπτωση των τεχνικών σχεδίασης για αυξημένη δοκιμαστικότητα κληρονομεί τα μειονεκτήματα της τελευταίας.

Πρόσφατα, η *αυτοδοκιμή με λογισμικό* (*Software-Based Self-Test, SBST*) [22] προτάθηκε ως μία εναλλακτική προσέγγιση με απώτερο στόχο να επιλύσει τα προβλήματα που προκαλούνται τόσο από τις μεθοδολογίες σάρωσης, όσο και από την αυτοδοκιμή με υλικό, μειώνοντας το συνολικό κόστος της δοκιμής. Η βασική ιδέα της αυτοδοκιμής με λογισμικό είναι η βέλτιστη αξιοποίηση της *αρχιτεκτονικής του συνόλου εντολών* (*Instruction Set Architecture, ISA*) ενός επεξεργαστή με απώτερο στόχο την παραγωγή διανυσμάτων δοκιμής, τη μεταφορά δεδομένων δοκιμής, την ανάλυση των αποκρίσεων, ή και ακόμη τη διάγνωση ενός επεξεργαστή. Κατά την αυτοδοκιμή με λογισμικό, ο μικροεπεξεργαστής εκτελεί ένα σύνολο ρουτινών, οι οποίες εφαρμόζουν στις επιμέρους μονάδες του ακολουθίες διανυσμάτων δοκιμής. Στη συνέχεια, αποθηκεύει τις αποκρίσεις της δοκιμής στη *μνήμη δεδομένων* (*data memory*), από όπου και συλλέγονται με την βοήθεια ενός εξωτερικού ελεγκτή, χαμηλής συχνότητας λειτουργίας και μικρού μεγέθους μνήμης και συγκρίνονται με τις αναμενόμενες αποκρίσεις για την ανίχνευση των ελαττωμάτων.

Τα οφέλη της αυτοδοκιμής με λογισμικό είναι πολλαπλά. Καταρχήν, επιτρέπει τη δοκιμή στην πραγματική συχνότητα λειτουργίας με τη χρησιμοποίηση οποιουδήποτε εξωτερικού ελεγκτή χαμηλού κόστους. Η αυτοδοκιμή με λογισμικό βασίζεται αποκλειστικά και μόνο στο σύνολο εντολών του επεξεργαστή και δεν απαιτεί καμία μετατροπή στη σχεδίαση του επεξεργαστή για αύξηση της δοκιμαστικότητας. Δυσάρεστες συνέπειες όπως επιβάρυνση της επιφάνειας, αύξηση της κατανάλωση ενέργειας, ή μείωση της απόδοσης του επεξεργαστή δεν υφίστανται. Το λογισμικό αυτοδοκιμής μπορεί να χρησιμοποιηθεί τόσο για τη *δοκιμή κατασκευής* (*manufacturing testing*) του μικροεπεξεργαστή, όσο και στο *πεδίο λειτουργίας του* (*in-field testing*). Επιπλέον, το λογισμικό αυτοδοκιμής εκτελείται όπως και το υπόλοιπο λογισμικό και έτσι "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. Η δοκιμή με λογισμικό επιτρέπει την επαναχρησιμοποίηση του ίδιου του μικροεπεξεργαστή για τη δημιουργία και εφαρμογή διανυσμάτων και τη συλλογή αποκρίσεων για τη δοκιμή άλλων υποσυστημάτων που βρίσκονται σε ένα *σύστημα σε ολοκληρωμένο κύκλωμα* (*System on Chip, SoC*). 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.

Η επιστημονική περιοχή στην οποία εντάσσεται η διδακτορική διατριβή κατά το πρώτο σκέλος της, αναφέρεται στην δοκιμή των *ελεγκτών περιφερειακών επικοινωνίας* (*communication peripherals controllers*) σε *συστήματα σε ολοκληρωμένο* (*System on Chip, SoC*). Ο όρος SoC

αναφέρεται στην ενσωμάτωση όλων των συστατικών ενός υπολογιστή, ή ενός ηλεκτρονικού συστήματος σε ένα ενιαίο ολοκληρωμένο κύκλωμα.

Η εκτεταμένη χρήση των SoC τα τελευταία χρόνια, οφείλεται στις απαιτήσεις για χαμηλότερο συνολικό κόστος παραγωγής, μικρό μέγεθος ολοκληρωμένου κυκλώματος και χαμηλότερης κατανάλωσης ενέργειας. Αυτές οι απαιτήσεις, εν μέρει, είναι οδηγοί του υψηλότερου βαθμού ολοκλήρωσης, αλλά ωθούν επίσης τους σχεδιαστές όλο και περισσότερο στην σχεδίαση *αρχιτεκτονικών ενσωματωμένων πυρήνων οριζόμενων από την εκάστοτε εφαρμογή* (*application-specific embedded cores*), ή στην προσαρμογή των υπαρχόντων ενσωματωμένων πυρήνων στην εκάστοτε εφαρμογή. Δεδομένης μιας οριζόμενης από την εφαρμογή απόδοσης, οι σχεδιαστές είναι σε θέση να κρατήσουν το κόστος των πυρήνων στο ελάχιστο, διατηρώντας παράλληλα την ευελιξία που αυτοί παρέχουν. Το κόστος των πυρήνων καθορίζεται από την απόδοση, την περιοχή υλικού που καταλαμβάνουν και στην κατανάλωση ενέργειας αυτών. Ο Πίνακας 1.1 περιέχει μερικά παραδείγματα εφαρμογών που ενσωματώνουν SoC. Πολλές μάλιστα από αυτές τις εφαρμογές, όπως για παράδειγμα οι παιχνιδομηχανές, ή τα συστήματα πλοήγησης στην αεροπλοΐα έχουν επεξεργαστική ισχύ όσο και ένας υψηλής απόδοσης *προσωπικός υπολογιστής* (*personal computer*).

Κατηγορία	Προϊόντα
Οικιακές Συσκευές	Πλυντήρια, Συσκευές παραγωγής πολυμέσων, Παιχνιδομηχανές
Συσκευές γραφείου	Εκτυπωτής, Ασύρματο δίκτυο, Αυτοματισμός κτηρίου
Αυτοκινητοβιομηχανία	Εγκέφαλος, Φρένα ABS, Συστήματα ασφαλείας
Αεροπλοΐα	Αυτόματος πιλότος, Συστήματα πλοήγησης
Βιομηχανία	Έλεγχος διαδικασίας παραγωγής, Απεικόνιση διαδικασίας παραγωγής, Ρομποτικές εφαρμογές
Καθημερινότητα	Κινητά τηλέφωνα, Πιστωτικές κάρτες, Ηλεκτρονικά εισιτήρια λεωφορείου

Πίνακας 1.1 Εφαρμογές συστημάτων σε ολοκληρωμένο κύκλωμα

Ένα από τα σημαντικότερα ζητήματα των SoC είναι η δυσκολία δοκιμής τους, λόγω της μειωμένης ελεγξιμότητας και παρατηρησιμότητας των πυρήνων. Τα τελευταία χρόνια έγινε μια προσπάθεια ώστε να ορισθεί μια ενιαία προσέγγιση δοκιμής για τους ενσωματωμένους πυρήνες σε SoC. Η προσπάθεια αυτή ολοκληρώθηκε με την σύνταξη ενός *προτύπου* (*standard*) που ονομάζεται *πρότυπο δοκιμής για ενσωματωμένους πυρήνες IEEE 1500* (*Standard for Embedded Core Test*) [23]. Η υιοθέτηση του προτύπου εξασφαλίζει αυτοματοποίηση και μεταφερσιμότητα της διαδικασίας της δοκιμής, αλλά οι σχετικές τροποποιήσεις του SoC μπορούν να επιβαρύνουν τόσο την επιφάνεια του κυκλώματος, όσο και την απόδοση του τελικού συστήματος. Όταν η δοκιμή του SoC, χρησιμοποιώντας το πρότυπο IEEE 1500, εκτελείται από τον εξωτερικό εξοπλισμό δοκιμής, η πραγματική συχνότητα λειτουργίας είναι δύσκολο να επιτευχθεί,

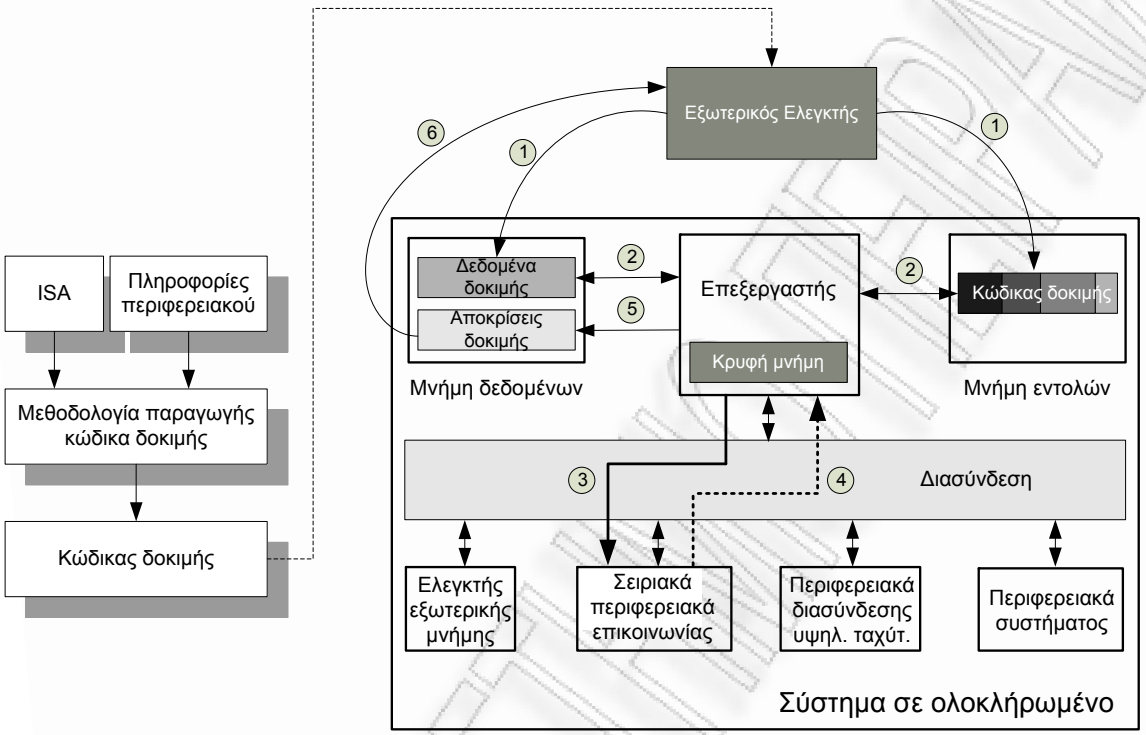
δεδομένου ότι οι εξωτερικοί ελεγκτές λειτουργούν σε μικρότερες συχνότητες σε σχέση με ένα SoC. Επιπλέον ο αυξημένος όγκος δεδομένων είναι δυνατόν να οδηγήσει σε ιδιαίτερα μεγάλο χρόνο εφαρμογής της δοκιμής. Είναι όμως δυνατόν τα διανύσματα δοκιμής να παράγονται από έναν εσωτερικό ελεγκτή (self-test) εξαλείφοντας την ανάγκη ύπαρξης του εξωτερικού ελεγκτή δοκιμής. Αν και οι παραδοσιακές αρχιτεκτονικές αυτοδοκιμής με υλικό προσφέρουν μια αυτοματοποιημένη, δομημένη λύση για την δοκιμή ενός SoC, φέρουν επιβάρυνση στο υλικό του συστήματος, υποβάθμιση της απόδοσης και αύξηση της κατανάλωσης ενέργειας.

Η πρώτη συνεισφορά της διατριβής, είναι μία ενιαία μεθοδολογία για την ανάπτυξη λογισμικού αυτοδοκιμής για περιφερειακά επικοινωνίας σε SoC. Οι σύγχρονες αρχιτεκτονικές SoC ενσωματώνουν μια μεγάλη ποικιλία περιφερειακών συσκευών που ικανοποιούν τις διαφορετικές απαιτήσεις επικοινωνίας της εκάστοτε εφαρμογής. Μερικά χαρακτηριστικά παραδείγματα των περιφερειακών πυρήνων επικοινωνίας σε ένα SoC είναι τα ακόλουθα: Universal Asynchronous Receiver Transmitter, (UART), I^2C, Microwire, Bluetooth, για τη σύνδεση χαμηλής ταχύτητας περιφερειακών συσκευών επικοινωνίας και για λόγους αποσφαλμάτωσης (debug) και Ethernet, Universal Serial Bus, (USB), Firewire, Wi-Fi, για τη σύνδεση υψηλής ταχύτητας περιφερειακών συσκευών, ή για σκοπούς δικτύωσης. Όλοι αυτοί οι περιφερειακοί πυρήνες επικοινωνούν συνήθως με τον επεξεργαστή του συστήματος μέσω ενός κοινόχρηστου διαύλου επικοινωνίας. Εστιάζουμε σε αυτήν την κατηγορία πυρήνων επικοινωνίας διότι χρησιμοποιούνται ευρύτατα στα SoC, ενώ παράλληλα καταλαμβάνουν ένα σημαντικό ποσοστό επιφάνειας υλικού και συνεπώς ένα σημαντικό ποσοστό των ελαττωμάτων του συστήματος. Επιπλέον, δεν είχε παρουσιαστεί προηγουμένως καμία εργασία στην βιβλιογραφία για την εφαρμογή της αυτοδοκιμής με λογισμικό με αποδεκτά ποσοστά κάλυψης ελαττωμάτων για πολύπλοκους πυρήνες αυτής της κατηγορίας.

Η προτεινόμενη μεθοδολογία αυτοδοκιμής με λογισμικό βασίζεται στην αρχιτεκτονική του συνόλου εντολών του επεξεργαστή, επιτυγχάνοντας τη δοκιμή κατά την πραγματική συχνότητα λειτουργίας, χρησιμοποιώντας εξωτερικούς ελεγκτές χαμηλού κόστους. Δεδομένου ότι βασίζεται στην δοκιμή λειτουργίας, η μεθοδολογία μοιάζει με την επαλήθευση σχεδίασης για τους περιφερειακούς πυρήνες, αλλά εφαρμόζει ένα εμπλουτισμένο *ντετερμινιστικό* (*deterministic*)[3] σύνολο διανυσμάτων δοκιμής για να ανιχνεύσει τα ελαττώματα των περιφερειακών επικοινωνίας [26], [27]. Χωρίζεται σε τέσσερα βασικά βήματα καθένα από τα οποία ανιχνεύει τα ελαττώματα σε τέσσερις διαφορετικές κύριες μονάδες του περιφερειακού επικοινωνίας. Ο διαχωρισμός της δοκιμής σε τέσσερα επιμέρους βήματα πραγματοποιείται για την διευκόλυνση της διαδικασίας παραγωγής του κώδικα δοκιμής. Η μεθοδολογία χρησιμοποιεί την αρχιτεκτονική του συνόλου εντολών του επεξεργαστή και τις πληροφορίες του περιφερειακού ώστε να παράγει το τελικό κώδικα δοκιμής. Επαληθεύτηκε στους πυρήνες

[3] Ντετερμινιστικό ονομάζεται ένα σύνολο δοκιμής που επιτυγχάνει υψηλό ποσοστό κάλυψης ελαττωμάτων με προϋπολογισμένα διανύσματα σε μια μονάδα ενός ολοκληρωμένου κυκλώματος.

UART και Ethernet [24] επιτυγχάνοντας 93.92% και 93.39% ποσοστό κάλυψης ελαττωμάτων αντίστοιχα. Τα βήματα της μεθοδολογίας απεικονίζεται στην Εικόνα 1.3 όπου:



Εικόνα 1.3 Συνεισφορά της διδακτορικής διατριβής στην εξαγωγή προγράμματος δοκιμής για τα περιφερειακά επικοινωνίας

- Τα προγράμματα και τα δεδομένα αυτοδοκιμής φορτώνονται στην ενσωματωμένη μνήμη εντολών και δεδομένων αντίστοιχα του SoC (1).
- Ο επεξεργαστής αναλαμβάνει την εκτέλεση τους (2), εφαρμόζοντας τα διανύσματα δοκιμής στους πυρήνες περιφερειακών επικοινωνίας (3).
- Ο επεξεργαστής λαμβάνει τις αποκρίσεις δοκιμής από τα περιφερειακά (4) και τις αποθηκεύει στην ενσωματωμένη μνήμη δεδομένων (5).
- Μετά την ολοκλήρωση της δοκιμής, οι αποκρίσεις δοκιμής των περιφερειακών επικοινωνίας συλλέγονται από την ενσωματωμένη μνήμη δεδομένων του SoC στον εξωτερικό ελεγκτή (6).

Συνεπώς, ο ενσωματωμένος επεξεργαστής έχει στην ουσία τον ρόλο του εσωτερικού ελεγκτή δοκιμής (internal tester).

Η προτεινόμενη μεθοδολογία έχει τα εξής πλεονεκτήματα:

- Δεν επιφέρει επιπλέον κατανάλωση ενέργειας πέραν της προδιαγραμμένης από τους σχεδιαστές του συστήματος.
- Δεν επιβαρύνει το σύστημα με επιπλέον υλικό για τη δοκιμή και επομένως δεν επηρεάζει την απόδοση του συστήματος.
- Εφαρμόζεται σε όλες τις κατηγορίες των περιφερειακών επικοινωνίας επιτυγχάνοντας υψηλό ποσοστό κάλυψης ελαττωμάτων.

Η προέκταση της ντετερμινιστικής μεθοδολογίας αυτοδοκιμής με λογισμικό, περιλαμβάνει την αυτοματοποίηση αυτής, δημιουργώντας την υβριδική προσέγγιση [28]. Η υβριδική προσέγγιση αποτελεί συνδυασμό δύο διαφορετικών μεθόδων παραγωγής κώδικα δοκιμής (ντετερμινιστικό και αυτόματο), συνδυάζοντας τα πλεονεκτήματα και των δύο προσεγγίσεων και εξαλείφοντας κατά το μέγιστο δυνατό τα μειονεκτήματα τους. Η υβριδική μεθοδολογία εκμεταλλεύεται τις ρουτίνες δοκιμής της ντετερμινιστικής προσέγγισης για να καθοδηγήσει αποδοτικότερα την αυτόματη παραγωγή προγράμματος δοκιμής της αυτόματης προσέγγισης με απώτερο στόχο την επίτευξη υψηλότερου ποσοστού κάλυψης ελαττωμάτων με την μικρότερη δυνατή ανθρώπινη παρέμβαση. Τα πλεονεκτήματα της υβριδικής μεθοδολογίας έναντι των προηγούμενων προσεγγίσεων επαληθεύονται στους πυρήνες UART, HDLC και Ethernet [24], ως προς το ποσοστό κάλυψης ελαττωμάτων, το μέγεθος του κώδικα δοκιμής και το συνολικό χρόνο της δοκιμής. Το ποσοστό κάλυψης ελαττωμάτων της υβριδικής μεθοδολογίας για τους πυρήνες UART, HDLC και Ethernet είναι 93.13%, 97.43% και 91.79% αντίστοιχα. Ιδιαίτερα σημαντική είναι η μείωση του χρόνου παραγωγής του κώδικα δοκιμής (έως και 20 φορές), σε σύγκριση με την ντετερμινιστική προσέγγιση.

Το δεύτερο σκέλος της διατριβής πραγματεύεται τη συστηματική ανάπτυξη λογισμικού για την αποδοτική αυτοδοκιμή *συμμετρικών πολυεπεξεργαστών* (*symmetric multiprocessor*) με στόχο τη μείωση του συνολικού χρόνου εφαρμογής της δοκιμής. Παρά τις συνεχώς αυξανόμενες επιδόσεις των μονοπύρηνων επεξεργαστών, η προσπάθεια για περαιτέρω βελτίωση της απόδοσης των μικροεπεξεργαστών, φαίνεται να έχει φτάσει σε αδιέξοδο για την τεχνολογία των τρανζίστορ. Ένας λόγος είναι η κατανάλωση ενέργειας και η ανάλογη απαίτηση για πιο προηγμένες μορφές ψύξης. Η κατανάλωση ενέργειας ενός μικροεπεξεργαστή είναι ευθέως ανάλογη της συχνότητας ρολογιού. Συνεπώς, ακόμα και αν όλες οι άλλες παράμετροι είναι ίδιες ο διπλασιασμός της συχνότητας θα προκαλέσει διπλασιασμό της κατανάλωσης ισχύος. Επιπλέον η συρρίκνωση του μεγέθους των τρανζίστορ προκαλεί εκθετική αύξηση του *ρεύματος διαρροής* (*leakage power*).

Η είσοδος των πολυπύρηνων επεξεργαστών έχει ως στόχο να λύσει τα παραπάνω προβλήματα, αλλά αντίθετα με προηγούμενες βελτιώσεις στην ψηφιακή σχεδίαση, τα οφέλη που

προσφέρει η παράλληλη επεξεργασία δεν είναι εγγυημένα. Επιπλέον, οι σημερινές εφαρμογές είναι ιδιαίτερα απαιτητικές σε υπολογιστική ισχύ. Για αυτούς τους λόγους, πολλοί κατασκευαστές επεξεργαστών παρουσιάζουν νέες γενιές οι οποίοι περιλαμβάνουν πολλούς πυρήνες σε έναν ενιαίο ολοκληρωμένο κύκλωμα, όπου αν και λειτουργούν σε χαμηλότερη συχνότητα λειτουργίας από τους προηγούμενης γενιάς επεξεργαστές, είναι ικανοί να προσφέρουν υψηλότερη επεξεργαστική ισχύ και παραλληλισμό μεταξύ των εφαρμογών.

Στην διδακτορική διατριβή μελετάμε την αυτοδοκιμή των *συμμετρικών πολυεπεξεργαστών* (*Symmetric Multiprocessors*, *SMP*). Ο όρος συμμετρικοί πολυεπεξεργαστές αναφέρεται τόσο στην αρχιτεκτονική του υλικού του υπολογιστή, όσο και στην συμπεριφορά του λειτουργικού συστήματος. Ένας συμμετρικός πολυεπεξεργαστής μπορεί να θεωρηθεί ως ένα μεμονωμένο σύστημα υπολογιστή με τα ακόλουθα χαρακτηριστικά. Υπάρχουν δύο ή περισσότεροι όμοιοι επεξεργαστές. Οι επεξεργαστές μοιράζονται την ίδια μνήμη και τις συσκευές εισόδου-εξόδου, καθώς επίσης και το κοινό μέσο επικοινωνίας, ούτως ώστε οι προσπελάσεις στην μνήμη να απαιτούν τον *ίδιο χρόνο προσπέλασης* (*Uniform Memory Access, UMA*) από όλους τους επεξεργαστές. Όλοι οι επεξεργαστές διαμοιράζονται την πρόσβαση στις συσκευές εισόδου-εξόδου, είτε μέσω των ίδιων καναλιών επικοινωνίας, είτε μέσω διαφορετικών καναλιών που παρέχουν πρόσβαση στις αναφερόμενες συσκευές. Όλοι οι επεξεργαστές είναι ικανοί να χειριστούν όλες τις πιθανές διεργασίες.

Στην αρχιτεκτονική των συμμετρικών πολυεπεξεργαστών οι δύο πιο κοινές αρχιτεκτονικές επιλογές για το δίκτυο διασύνδεσης είναι ο *κοινός δίαυλος* (*shared bus*) και ο *σταυρωτός μεταγωγέας* (*crossbar switch*). Σύμφωνα με την αρχιτεκτονική του κοινού διαύλου το σύστημα του πολυεπεξεργαστή χρησιμοποιεί ένα κοινό δίαυλο επικοινωνίας, όπου ένα *κύκλωμα διαιτησίας* (*bus arbiter*) καθορίζει ποιος από τους *n* επεξεργαστές αποκτά πρόσβαση κάθε φορά προς την κοινόχρηστη κρυφή μνήμη. Το κύκλωμα διαιτησίας διανέμει συνήθως το δίαυλο επικοινωνίας στους επεξεργαστές, ακολουθώντας μία *εξυπηρέτηση εκ' περιτροπής* (*round robin*). Στον αντίποδα, ο κοινός δίαυλος επικοινωνίας αντικαθίσταται από ένα *σταυρωτό μεταγωγέα* (*crossbar switch*) προκειμένου να πραγματοποιηθεί η σύνδεση μεταξύ των *n* επεξεργαστών με τις *m σειρές της κοινόχρηστης κρυφής μνήμης* (*shared cache banks*). Η κοινόχρηστη κρυφή μνήμη έχει διαιρεθεί σε πολλές διαφορετικές *σειρές* (*banks*) καθεμία από τις οποίες συνδέεται σε διαφορετικό διασυνδετικό δίαυλο του σταυρωτού μεταγωγέα. Κάθε σειρά της κοινόχρηστης κρυφής μνήμης ενσωματώνει ένα διαιτητή κρυφής μνήμης, ο οποίος διαχειρίζεται τις συναλλαγές από τους πυρήνες επεξεργαστών προς την σειρά αυτή. Ο σταυρωτός μεταγωγέας συνδέει κατ' εντολή τους επεξεργαστές με τις σειρές της κοινόχρηστης κρυφής μνήμης.

Στην παρούσα διδακτορική διατριβή, εξετάζουμε την ικανότητα παράλληλης εκτέλεσης του προγράμματος δοκιμής σε συμμετρικούς πολυεπεξεργαστές, όπου η προτεινόμενη προσέγγιση στοχεύει στη ελαχιστοποίηση του συνολικού χρόνου εφαρμογής της δοκιμής διατηρώντας την ικανότητα προσδιορισμού των ελαττωματικών πυρήνων. Ένα πρόγραμμα δοκιμής για ένα

συμμετρικό πολυεπεξεργαστή πρέπει να εκτελέσει μια σειρά από λειτουργίες σε κάθε επεξεργαστή, περιλαμβάνοντας λειτουργίες που συντονίζουν τους χωριστούς επεξεργαστές στην εκτέλεση ενός ενιαίου προγράμματος. Επίσης, κάθε επεξεργαστής πρέπει να εκτελέσει όλες τις ρουτίνες δοκιμής ώστε να επιτευχθεί το επιθυμητό ποσοστό κάλυψης ελαττωμάτων. Η αναγκαιότητα αυτή της δημιουργίας και του συντονισμού πολλών ρουτινών δοκιμής προσθέτει μια καινούργια διάσταση στη διαδικασία της αυτοδοκιμής με λογισμικό. Οι ρουτίνες δοκιμής πρέπει να σχεδιαστούν κατάλληλα ώστε να μειωθεί ο αντίκτυπος της κοινής χρήσης των πόρων (δίκτυο επικοινωνίας και κοινόχρηστη κρυφή μνήμη) και να ελαχιστοποιηθούν τα αδρανή διαστήματα των επεξεργαστών. Ακολουθεί μια λίστα που περιγράφει τα προβλήματα που περιορίζουν την απόδοση των παράλληλων προγραμμάτων αυτοδοκιμής:

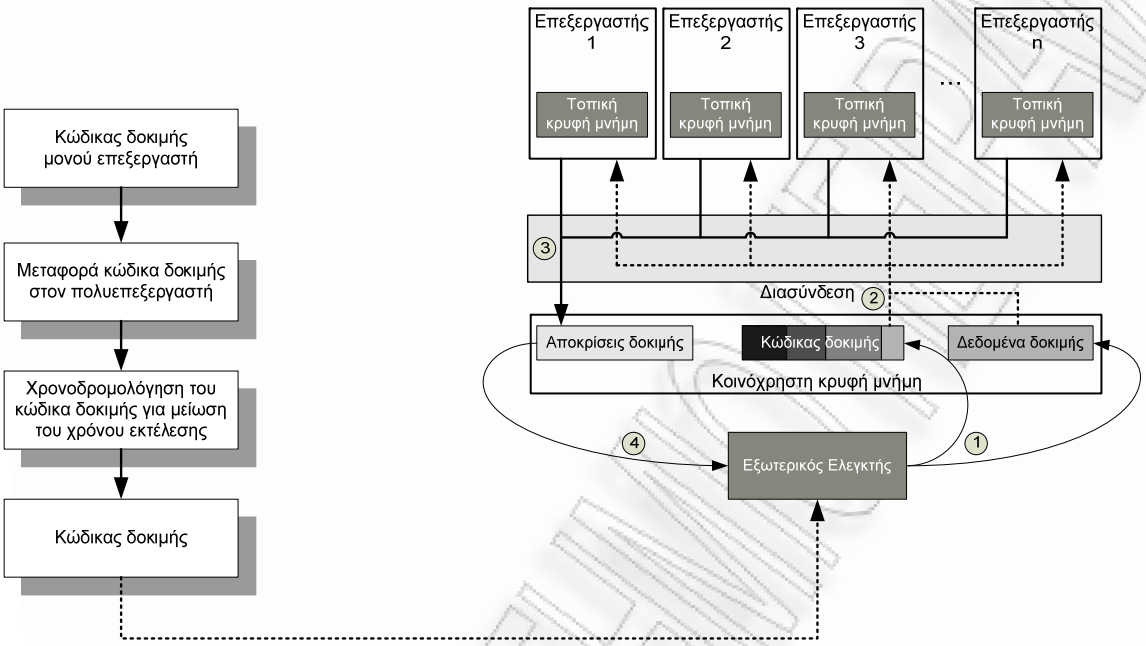
1. Ο ανταγωνισμός του διαύλου επικοινωνίας (bus Contention). Η εκτέλεση του προγράμματος δοκιμής σε έναν επεξεργαστή καθυστερεί όταν αυτός περιμένει για να αποκτήσει πρόσβαση στην κοινόχρηστη κρυφή μνήμη η οποία χρησιμοποιείται εκείνη τη χρονική στιγμή από κάποιον άλλο επεξεργαστή.

2. Η ανεπαρκής αξιοποίηση του διαθέσιμου εύρους ζώνης επικοινωνίας μεταξύ των πυρήνων με τις σειρές της κοινόχρηστης κρυφής μνήμης του συστήματος στην αρχιτεκτονική του σταυρωτού μεταγωγέα. Το πρόβλημα αυτό παρουσιάζεται όταν δεν εκμεταλλεύονται οι πιθανές ελεύθερες συνδέσεις του σταυρωτού μεταγωγέα ώστε να εξυπηρετηθούν παράλληλα διαφορετικά τμήματα του προγράμματος δοκιμής.

3. Οι ακυρώσεις συνοχής της κρυφής μνήμης (cache coherency invalidation). Κάθε φορά που μια λέξη γράφεται από έναν επεξεργαστή στην τοπική κρυφή μνήμη, ή στην κοινόχρηστη κρυφή μνήμη, όλοι οι άλλοι επεξεργαστές ακυρώνουν τα τοπικά αντίγραφα αυτής της λέξης. Επομένως, η εκτέλεση των προγραμμάτων αυτοδοκιμής μπορεί να επιβραδυνθεί σημαντικά, λόγω του μεγάλου αριθμού ακυρώσεων της κρυφής μνήμης που μπορεί να οδηγήσει στην επαναπροσκόμιση δεδομένων στην τοπική κρυφή μνήμη από την κοινόχρηστη, προσθέτοντας κύκλους κατά την εφαρμογή της αυτοδοκιμής.

Η προτεινόμενη προσέγγιση περιγράφει μια γενική αποδοτική μεθοδολογία αυτοδοκιμής με λογισμικό για την *κατανομή του κώδικα και των δεδομένων της δοκιμής* (*test code and data allocation*) στην κοινόχρηστη κρυφή μνήμη, τη *μείωση των ακυρώσεων συνοχής κρυφής μνήμης* (*test responses invalidation reduction*) και τη *δρομολόγηση των ρουτινών δοκιμής* (*test routines scheduling*) στους πυρήνες επεξεργαστών. Η μεθοδολογία βασίζεται σε στατιστικά δεδομένα για το χρόνο εκτέλεσης και τις αιτήσεις προς την κοινόχρηστη κρυφή μνήμη των ρουτινών

αυτοδοκιμής. Η συνεισφορά της διδακτορικής διατριβής για συμμετρικούς πολυεπεξεργαστές απεικονίζεται στην Εικόνα 1.4.



Εικόνα 1.4 Συνεισφορά της διδακτορικής διατριβής με στόχο την μείωση του συνολικού χρόνου εφαρμογής της δοκιμής σε συμμετρικούς πολυεπεξεργαστές

1. Τα προγράμματα και τα διανύσματα δοκιμής φορτώνονται στην κοινόχρηστη κρυφή μνήμη (1), χρησιμοποιώντας εξωτερικό ελεγκτή χαμηλού κόστους.

2. Εκτελούνται από όλους του επεξεργαστές παράλληλα (2), στην πραγματική συχνότητα λειτουργίας και οι αποκρίσεις δοκιμής αποθηκεύονται στην κοινόχρηστη κρυφή μνήμη (3).

3. Οι αποκρίσεις συλλέγονται στη μνήμη του εξωτερικού ελεγκτή (4) για την επαλήθευση της ορθής λειτουργίας των επεξεργαστών.

Η προτεινόμενη μεθοδολογία [29], [30], [31] αντιμετωπίζει τα προβλήματα που αναφέρθηκαν παραπάνω τα οποία μειώνουν τον παραλληλισμό κατά την διάρκεια εφαρμογής της δοκιμής, ενώ παράλληλα διατηρεί την ανίχνευση ελαττωμάτων σε υψηλά επίπεδα για κάθε πυρήνα επεξεργαστή του συστήματος. Συγκεκριμένα επιτυγχάνει:

- Τη μείωση των απαιτήσεων αποθήκευσης στην κοινόχρηστη κρυφή μνήμη με τη χρήση ενός ενιαίου αντιγράφου προγράμματος.

- Τη μείωση του χρόνου εκτέλεσης δοκιμής αυξάνοντας τον παραλληλισμό κατά τη διάρκεια της εκτέλεσης των προγραμμάτων δοκιμής.

- Την ελαχιστοποίηση του ανταγωνισμού του μέσου επικοινωνίας και των ακυρώσεων δεδομένων λόγω συνοχής της κρυφής μνήμης, τα οποία και επιβραδύνουν την εκτέλεση του κώδικα δοκιμής.

- Την παραγωγή διαφορετικών αποκρίσεων δοκιμής για κάθε επεξεργαστή, ώστε να επιτραπεί η διάγνωση στο επίπεδο των πυρήνων.

Η μεθοδολογία [29], [31] εφαρμόστηκε σε συμμετρικούς πολυεπεξεργαστές δύο, τεσσάρων και οκτώ πυρήνων οι οποίοι βασίζονται στον επεξεργαστή RISC, OpenRISC 1200 [24], ο οποίος έχει χρησιμοποιηθεί εκτενώς στην έρευνα της αυτοδοκιμής με λογισμικό στους μονοπύρηνους επεξεργαστές. Δημιουργήθηκαν συμμετρικοί πολυεπεξεργαστές με βάση τον OpenRISC 1200 ώστε να μελετηθεί η δυνατότητα εφαρμογής της προτεινόμενης μεθοδολογίας σε αρχιτεκτονικές συμμετρικών επεξεργαστών με διαφορετικό αριθμό επεξεργαστών και τύπο διασύνδεσης. Μελετήθηκαν οι δύο πιο κοινές αρχιτεκτονικές διασύνδεσης: ο κοινός δίαυλος ο οποίος αξιολογείται σε συμμετρικούς πολυεπεξεργαστές δύο και τεσσάρων πυρήνων, και ο σταυρωτός μεταγωγέας ο οποίος αξιολογείται σε συμμετρικούς πολυεπεξεργαστές τεσσάρων και οκτώ πυρήνων.

Παρουσιάζονται πειραματικά αποτελέσματα ως προς τον χρόνο εφαρμογής και το μέγεθος κώδικα του προγράμματος δοκιμής. Επίσης, παρέχονται αποτελέσματα προσομοίωσης ελαττωμάτων για ολόκληρη τη λογική των συμμετρικών πολυεπεξεργαστών (όχι μόνο της εσωτερικής λογικής των επεξεργαστών, αλλά και της λογικής διαλειτουργικότητας μεταξύ των πυρήνων συμπεριλαμβανομένων των ελεγκτών της κρυφής μνήμης και του δικτύου διασύνδεσης). Στη συνέχεια, η μεθοδολογία αξιολογείται με βάση ένα σύνολο *μετροπρογραμμάτων* (*benchmarks*) για επεξεργαστές του Πανεπιστημίου Stanford [25], με διαφορετικά χαρακτηριστικά εκτέλεσης. Η μεθοδολογία επιταχύνει τον χρόνο εκτέλεσης του προγράμματος δοκιμής κατά 2.0, 3.9 και 7.4 φορές για τους διπύρηνους, τετραπύρηνους και οκταπύρηνους συμμετρικούς πολυεπεξεργαστές OpenRISC 1200 αντίστοιχα, σε σύγκριση με την ακολουθιακή εκτέλεση του ίδιου προγράμματος δοκιμής.

Στο δεύτερο μέρος των πειραματικών αποτελεσμάτων για συμμετρικούς πολυεπεξεργαστές, παρουσιάζεται η εφαρμογή της μεθοδολογίας [30] στον επεξεργαστή OpenSPARC T1 της Sun Microsystems. Τα αποτελέσματα προστέθηκαν με σκοπό την επαλήθευση της μεθοδολογία σε μια πραγματική υλοποίηση σύγχρονου πολυεπεξεργαστή υψηλών επιδόσεων. Η επιτάχυνση του χρόνου εκτέλεσης δοκιμής στον επεξεργαστή OpenSPARC T1 είναι 7.9 φορές σε σύγκριση με την ακολουθιακή εκτέλεση του ίδιου προγράμματος .

Η διάρθρωση της διατριβής σε κεφάλαια είναι η ακόλουθη:

- Το Κεφάλαιο 1 είναι η παρούσα Εισαγωγή.

- Στο Κεφάλαιο 2 παρουσιάζονται οι θεμελιώδεις έννοιες και οι μεθοδολογίες για τη δοκιμή των ολοκληρωμένων κυκλωμάτων. Στη συνέχεια, αναπτύσσεται η έννοια της αυτοδοκιμής με λογισμικό.

- Στο Κεφάλαιο 3 παρουσιάζονται η αρχιτεκτονική των SoC καθώς επίσης και η αρχιτεκτονική των περιφερειακών επικοινωνίας. Στη συνέχεια, παρουσιάζονται οι τεχνικές δοκιμής των συστημάτων αυτών, όπως και η σχετική ως προς τη δοκιμή βιβλιογραφία για περιφερειακές μονάδες και οι αδυναμίες που παρουσιάζουν οι υπάρχουσες προσεγγίσεις. Έπειτα, παρουσιάζεται η ντετερμινιστική και η υβριδική μεθοδολογία αυτοδοκιμής με λογισμικό για περιφερειακά επικοινωνίας. Τα αποτελέσματα του κεφαλαίου αυτού που αποτελούν και το πρώτο σκέλος της διατριβής έχουν δημοσιευτεί στις εργασίες [26], [27] και [28].

- Στο Κεφάλαιο 4 γίνεται αρχικά μια σύντομη αναφορά των τεχνικών παραλληλίας των επεξεργαστών. Στη συνέχεια, αναλύονται διεξοδικά οι αρχιτεκτονικές των συμμετρικών πολυεπεξεργαστών, που βασίζονται στον κοινό δίαυλο και τον ραβδωτό μεταγωγέα. Επίσης, παρουσιάζονται οι ευρέως διαδεδομένοι πυρήνες συμμετρικών πολυεπεξεργαστών (Intel Core 2 Duo, Sun UltraSPARC T1 και Sun UltraSPARC T2), καθώς και η σχετική ως προς τη δοκιμή βιβλιογραφία για τους επεξεργαστές αυτούς καθώς και οι αδυναμίες που παρουσιάζουν οι υπάρχουσες τεχνικές. Έπειτα, παρουσιάζεται η μεθοδολογία αυτοδοκιμής με λογισμικό με στόχο τη μείωση του συνολικού χρόνου εφαρμογής της δοκιμής για συμμετρικούς πολυεπεξεργαστές. Στη συνέχεια, περιγράφονται τα πειραματικά αποτελέσματα τα οποία επαληθεύτηκαν σε πολλούς διαφορετικούς συμμετρικούς πολυεπεξεργαστές και συγκρίθηκαν με όλες τις πιθανές εναλλακτικές προσεγγίσεις αυτοδοκιμής με λογισμικό. Τα αποτελέσματα του κεφαλαίου αυτού που αποτελεί το δεύτερο σκέλος της διδακτορικής διατριβής έχουν δημοσιευτεί στις εργασίες [29], [30] και [31].

- Στο Κεφάλαιο 5 παρουσιάζονται τα συμπεράσματα της διατριβής καθώς και κατευθύνσεις για μελλοντική έρευνα.

Λίστα Δημοσιεύσεων

- A. Apostolakis, M. Psarakis, D. Gizopoulos, A. Paschalis, "A Functional Self-Test Approach for Peripheral Cores in Processor-Based SoCs", in 13th IEEE International On-Line Testing Symposium, pp. 271 - 276, July, 2007.

- A. Apostolakis, M. Psarakis, D. Gizopoulos, A. Paschalis, "Functional Processor-Based Testing of Communication Peripherals in Systems-on-Chip", in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 971 - 975, August, 2007.

- A. Apostolakis, D. Gizopoulos, M. Psarakis, D. Ravotto, M. Sonza Reorda, "Test Program Generation for Communication Peripherals in Processor-Based System-on-Chips", in IEEE Design and Test of Computers, pp. 52 - 63, March-April, 2009.

- A. Apostolakis, M. Psarakis, D. Gizopoulos, A. Paschalis, "Functional Self-Testing for Bus-Based Symmetric Multiprocessors", in Design, Automation and Test in Europe, (DATE) pp. 1304 - 1309, March, 2008.

- A. Apostolakis, D. Gizopoulos, M. Psarakis, I. Parulkar, "Exploiting Execution Parallelism in Functional Self-Testing of Chip Multiprocessors", in 1st Design for Reliability Workshop, (DFR), January 2009.

- A. Apostolakis, M. Psarakis, D. Gizopoulos, A. Paschalis, "Software-Based Self-Testing of Symmetric Shared-Memory Multiprocessors", in IEEE Transactions on Computers, (accepted).

Κεφάλαιο 2

Βασικές Αρχές Δοκιμής Ολοκληρωμένων Κυκλωμάτων

The state of mind which enables a man to do work of this kind is akin to that of the religious worshiper or the lover; the daily effort comes from no deliberate intention or program, but straight from the heart.

Albert Einstein

2.1 Εισαγωγή

Η κατασκευή της πλειονότητας των ολοκληρωμένων κυκλωμάτων περιλαμβάνει μια αυστηρή διαδικασία σχεδίασης η οποία ακολουθείται από μια λεπτομερή δοκιμή και αξιολόγηση των πρωτοτύπων πριν την έναρξη της παραγωγής του τελικού προϊόντος. Κατά την παραγωγή, ο συνεχής ποιοτικός έλεγχος των προϊόντων είναι επιθυμητός και επιβεβλημένος, αλλά η εξαντλητική δοκιμή κάθε ολοκληρωμένου κυκλώματος δεν είναι εφικτή λόγω υψηλού κόστους. Ωστόσο, όσο πιο περίπλοκο είναι το προϊόν, ή οι *μονάδες* (*components*) που χρησιμοποιούνται μέσα σε αυτό, τόσο μεγαλύτερη είναι η ανάγκη να διασφαλιστεί ότι το τελικό προϊόν είναι πλήρως λειτουργικό. Η δοκιμή συνεπώς των τελικών προϊόντων με όσο το δυνατόν

πιο αποδοτικό τρόπο με περιορισμούς στον προϋπολογισμό και στον χρόνο της δοκιμής αποτελεί ένα καθολικό πρόβλημα.

Στην επιστήμη της ηλεκτρονικής, και πιο συγκεκριμένα της μικροηλεκτρονικής, σε αντίθεση με το πεδίο της μηχανικής και της ηλεκτρομηχανικής, η *οπτική επιθεώρηση* (*visual inspection*) έχει ιδιαίτερα μικρή χρησιμότητα. Σε αυτή την επιστήμη, πρέπει να χρησιμοποιηθεί η παραμετρική ή λειτουργική δοκιμή ώστε να επαληθευτεί η ορθή λειτουργία του τελικού προϊόντος. Η διαδικασία της δοκιμής περιλαμβάνει τις παρακάτω κύριες κατηγορίες ελέγχων:

1. *Έλεγχοι σχεδίασης ολοκληρωμένου κυκλώματος,* (*IC design checks*)*:* Ελέγχουν αν τα πρωτότυπα κυκλώματα λειτουργούν με βάση τις προδιαγραφές.

2. *Έλεγχοι κατασκευής ολοκληρωμένου κυκλώματος,* (*IC fabrication checks*)*:* Πραγματοποιούνται από τον κατασκευαστή ολοκληρωμένου κυκλώματος ώστε να ελεγχθεί αν τα στάδια επεξεργασίας του κυκλώματος κατά την διάρκεια της κατασκευής του έχουν εκτελεστεί σωστά.

3. *Έλεγχοι παραγωγής ολοκληρωμένου κυκλώματος,* (*IC production checks*)*:* Ελέγχουν αν τα παραγόμενα ολοκληρωμένα κυκλώματα περιέχουν *ατέλειες* (*defects*).

4. *Δοκιμή προϊόντος,* (*Product test*)*:* Δοκιμές για την επιβεβαίωση της ορθής λειτουργία τους του τελικού προϊόντος.

Η διαδικασία της σχεδίασης των κυκλωμάτων δεν μπορεί να διαχωριστεί από την διαδικασία δοκιμής. Στην περίπτωση των πολύ μικρών κυκλωμάτων, που περιέχουν για παράδειγμα μόνο μερικές εκατοντάδες πύλες, το πρόβλημα δεν παρουσιάζει ιδιαίτερη δυσκολία. Στα κυκλώματα αυτού του μεγέθους μπορεί συχνά να εκτελεστεί μια πλήρως λειτουργική δοκιμή, ιδιαίτερα εάν η ποσότητα παραγωγής δεν είναι μεγάλη. Καθώς όμως αυξάνεται η πολυπλοκότητα των ολοκληρωμένων κυκλωμάτων, αλλά και η ποσότητα της παραγωγής, η πίεση για την πραγματοποίηση ενός περιορισμένου και αποδοτικού αριθμού δοκιμών μεγαλώνει διαρκώς. Για παράδειγμα, δεν θα ήταν πρακτικό να δοκιμάσουμε ένα μικροεπεξεργαστή εφαρμόζοντας όλες τις διαφορετικές καταστάσεις στις οποίες μπορεί να βρεθεί κατά την κανονική λειτουργία. Στην πραγματικότητα ένα υποσύνολο δοκιμών θα πρέπει να καθοριστεί και να εγγυηθεί σύμφωνα με μια αποδεκτή πιθανότητα ότι το τελικό προϊόν δεν περιέχει ελαττώματα.

Η *ολοκλήρωση πολύ μεγάλης κλίμακας* (*Very Large Scale Integration, VLSI*) έχει αποτελέσει έναν από τους κυριότερους λόγους ύπαρξης της δοκιμής, αλλά και για τις δυσκολίες εφαρμογής αυτής, λόγω του περιορισμένου αριθμού ακροδεκτών σε ένα κύκλωμα VLSI. Εάν τα διάφορα στάδια κατασκευής του ολοκληρωμένου (κατασκευή *πλακιδίων* (*wafer*), χάραξη του τσιπ, συνδέσεις, συσκευασία, κλπ) εκτελούνταν τέλεια, τότε δεν θα υπήρχε καμία ανάγκη για τη

δοκιμή του κυκλώματος. Κάθε κύκλωμα θα ήταν πλήρως λειτουργικό, υποθέτοντας η σχεδίαση του είχε επαληθευτεί πλήρως. Τα διάφορα στάδια όμως δεν είναι τέλεια και η παραμικρή ατέλεια μπορεί να προκαλέσει τη μη λειτουργική συμπεριφορά του προϊόντος. Συνεπώς, η δοκιμή όλων των παραγόμενων ολοκληρωμένων κυκλωμάτων κρίνεται απολύτως απαραίτητη.

Υπάρχει όμως η πιθανότητα ένα κύκλωμα να περάσει τις επιλεγμένες δοκιμές, αλλά ωστόσο να περιέχει ελαττώματα, εκτός αν πραγματοποιήσουμε εξαντλητική δοκιμή κάθε ολοκληρωμένου κυκλώματος. Αυτή η πιθανότητα μπορεί να αναλυθεί από μαθηματική άποψη. Σε αυτό το σημείο παρουσιάζουμε μια εξίσωση που αφορά το *επίπεδο ατέλειας* (*defect level*) ενός τσιπ συναρτήσει της εσοδείας και του ποσοστού κάλυψης των ελαττωμάτων [32], [33], [34], [35]. Αυτή η εξίσωση μπορεί να χρησιμοποιηθεί ώστε να υπολογίσουμε την ποιότητα δοκιμής, λαμβάνοντας υπόψη την *εσοδεία παραγωγής* (*yield*) και το επιθυμητό *επίπεδο ατέλειας* (*defect level*). Πρώτα δίνουμε τους ορισμούς των παραπάνω εννοιών.

Επίπεδο ατέλειας, (*Defect Level, DL*): Αποτελεί το κλάσμα των ελαττωματικών κυκλωμάτων τα οποία ολοκλήρωσαν επιτυχώς όλες τις δοκιμές, δηλαδή διέφυγαν του ελέγχου. Οι τιμές για το επίπεδο ατέλειας δίνονται συνήθως με χρήση του όρου των *ελαττωματικών μονάδων ανά εκατομμύριο,* (*Defective Parts Per Million. DPPM*). Η επιθυμητή τιμή είναι λιγότερο από 200 DPPM [10].

Εσοδεία παραγωγής (*Yield, Y*): Ορίζεται ως το κλάσμα των κατασκευασμένων ολοκληρωμένων κυκλωμάτων τα οποία δεν περιέχουν ελαττώματα. Η ακριβής τιμή του *Y* είναι σπάνια γνωστή, καθώς δεν είναι δυνατό να πραγματοποιηθεί δοκιμή για όλα τα πιθανά ελαττώματα. Συνεπώς, η τιμή του *Y* συνήθως προσεγγίζεται από το λόγο:

$$\text{Εσοδεία} = \frac{\text{Αριθμός μη ελαττωματικών κυκλωμάτων}}{\text{Σύνολο των κυκλωμάτων}}$$

όπου ο αριθμός των μη ελαττωματικών κυκλωμάτων ισούται με τον αριθμό των κυκλωμάτων που ολοκλήρωσαν επιτυχώς τη διαδικασία δοκιμής.

Ποσοστό κάλυψης ελαττωμάτων, (*Fault Coverage, FC*): Είναι ένα ποσοστό το οποίο προσδιορίζει την ποιότητα της δοκιμής και ορίζεται ως το πηλίκο:

$$\text{Ποσοστό κάλυψης ελαττωμάτων} = \frac{\text{Αριθμός ελαττωμάτων ππο ανιχνεύτηκαν}}{\text{Συνολικός αριθμός ελαττωμάτων}}$$

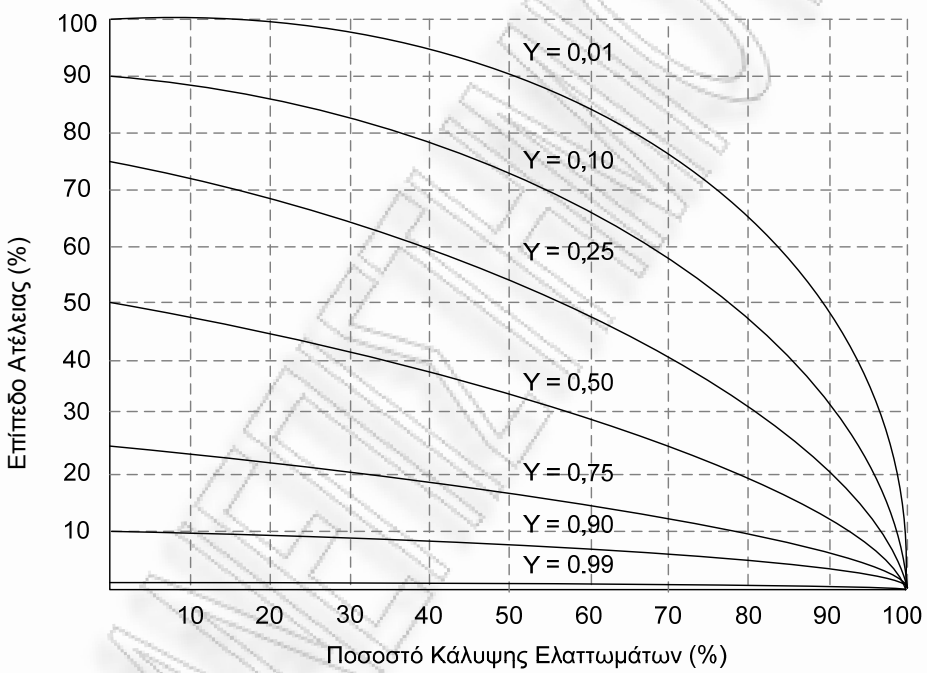
Το εφικτό μέγιστο ποσοστό κάλυψης ελαττωμάτων μπορεί να είναι μικρότερο του 100% εξαιτίας της πιθανής ύπαρξης *μη-ανιχνεύσιμων* (*undetectable*) ελαττωμάτων. Το *πραγματικό ποσοστό κάλυψης ελαττωμάτων* (*effective fault coverage ή test coverage*) ορίζεται ως η αναλογία των ελαττωμάτων που ανιχνεύτηκαν από τα διανύσματα δοκιμής έναντι των *ανιχνεύσιμων* (*detectable*) ελαττωμάτων.

$$\text{Πραγματικό ποσ. κάλ. ελατ.} = \frac{\text{Αριθμός ελαττωμάτων που ανιχνεύθηκαν}}{\text{Συνολικός αριθμός ελαττωμ.} - \text{Αριθμος μη ανιχνεύσιμων ελαττωμ.}}$$

Υποθέστε ότι η εσοδεία παραγωγής των ολοκληρωμένων κυκλωμάτων που δεν περιέχουν ελαττώματα σε ένα πλακίδιο είναι *Y*, όπου το *Y* έχει μία τιμή μεταξύ του "0" (όλα τα κυκλώματα ελαττωματικά) και του "1" (όλα τα κυκλώματα είναι χωρίς ελαττώματα). Υποθέστε, επίσης, ότι οι δοκιμές που εφαρμόζονται σε κάθε κύκλωμα έχουν *κάλυψη ελαττωμάτων* (*αποδοτικότητα ανίχνευσης ελαττωμάτων*) *FC,* όπου το *FC* έχει επίσης μια τιμή μεταξύ του "0" (οι δοκιμές δεν ανιχνεύουν κανένα από τα πιθανά ελαττώματα) και του "1" (οι δοκιμές ανιχνεύουν όλα τα πιθανά ελαττώματα). Συνεπώς, το ποσοστό των ολοκληρωμένων κυκλωμάτων που θα ολοκληρώσουν επιτυχώς τις δοκιμές, αλλά εξακολουθούν να περιέχουν ένα ή περισσότερα ελαττώματα (*το επίπεδο ατέλειας μετά από τη δοκιμή, Detect Level, DL*), δίνεται από τον τύπο:

$$DL = \{1 - Y^{(1-FC)}\} * 100\%$$

ενώ εξίσωση αυτή παριστάνεται στην Εικόνα 2.1 [32]**.**



Εικόνα 2.1 Επίπεδο ατέλειας μετά από τη δοκιμή (Detect Level, DL), συναρτήσει διαφορετικών εσοδειών παραγωγής (Yield, Y) και κάλυψης ελαττωμάτων (Fault Coverage, FC).

Η σημασία αυτής της σημαντικής εξίσωσης είναι η ακόλουθη. Υποθέστε ότι η εσοδεία παραγωγής είναι 50% (*Y*=0.5). Εάν δεν πραγματοποιηθεί δοκιμή (*FC*=0), το 50% των ολοκληρωμένων κυκλωμάτων που θα διανεμηθούν στην αγορά θα είναι ελαττωματικά. Εάν όμως πραγματοποιήσουμε δοκιμή και η αποδοτικότητα της είναι μόλις 80% (*FC*=0.8), τότε το ποσοστό των ελαττωματικών κυκλωμάτων θα είναι περίπου 15% (85% ελεύθερα ελαττωμάτων -

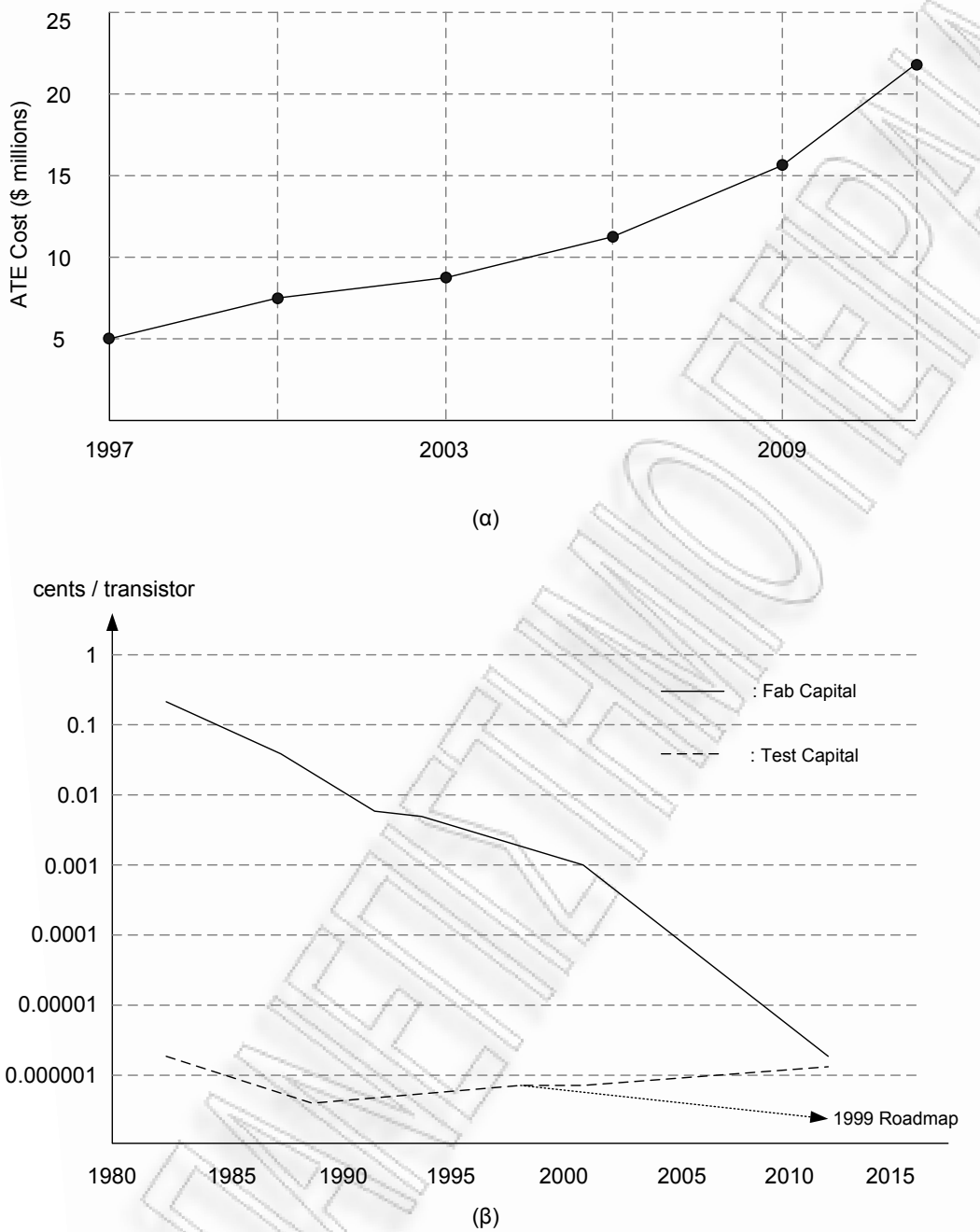
15% ελαττωματικά). Το ποσοστό αυτό σημαίνει ότι ένα στα επτά ολοκληρωμένα κυκλώματα είναι ελαττωματικά, ποσοστό ιδιαίτερα υψηλό για οποιαδήποτε αγορά. Κατά συνέπεια, για να διασφαλιστεί ότι ένα υψηλό ποσοστό κυκλωμάτων δεν περιέχουν ελαττώματα μετά από τη δοκιμή, είτε η εσοδεία παραγωγής, είτε η κάλυψη ελαττωμάτων, είτε και τα δύο, πρέπει να μεγιστοποιηθούν.

Αυτό αποτελεί και το κύριο δίλημμα της δοκιμής των πολύπλοκων ολοκληρωμένων κυκλωμάτων, ή γενικότερα οποιουδήποτε πολύπλοκου συστήματος. Εάν η αποδοτικότητα της δοκιμής είναι χαμηλή, τα ελαττωματικά κυκλώματα μπορεί να μην εντοπιστούν κατά την διαδικασία της δοκιμής. Στον αντίποδα, το να επιτύχουμε ανίχνευση ελαττωμάτων κοντά στο 100% (FC=1.0) μπορεί να απαιτεί πολύ εκτενείς δοκιμές και κατά συνέπεια απαγορευτικά δαπανηρές όπως εξηγούμε στην επόμενη ενότητα.

2.2 Κόστος δοκιμής και ποιότητα προϊόντος

Στις βιομηχανίες υπολογιστών και επικοινωνιών, η δοκιμή των ηλεκτρονικών ολοκληρωμένων κυκλωμάτων είναι ένας σημαντικός παράγοντας της επιτυχίας των προϊόντων. Οι πελάτες αναζητούν ένα αξιόπιστο προϊόν με λογικό πάντα κόστος. Ένας κατασκευαστής ολοκληρωμένου κυκλώματος, είναι δυνατόν να βελτιώσει την αξιοπιστία του προϊόντος του, εφαρμόζοντας επιπρόσθετη δοκιμή με το προστιθέμενο κόστος να μεταφέρεται προς τον πελάτη. Σε μια οικονομία ανταγωνιστικών αγορών, ο καταναλωτής ωφελείται με την επιλογή του καλύτερου προϊόντος. Για να μείνει στην αγορά, ο κατασκευαστής οφείλει να οριοθετήσει τους στόχους σχεδίασης και δοκιμής του κυκλώματος ώστε να παρέχει το καλύτερο δυνατόν προϊόν, με το χαμηλότερο δυνατό κόστος. Αυτό, όπως είναι προφανές, απαιτεί την παραγωγή ολοκληρωμένων κυκλωμάτων με τη μέγιστη δυνατή οικονομική αποδοτικότητα.

Οι εξελισσόμενες τάσεις της τεχνολογίας ολοκλήρωσης πολύ μεγάλης κλίμακας έχουν δώσει μια νέα προοπτική στο κόστος παραγωγής στην επιστήμη της μικροηλεκτρονικής. Η Intel [8] αναφέρει ότι ο συνδυασμός της *επαλήθευσης* (*verification*) και της *δοκιμής κατασκευής* (*manufacturing testing*) αποτελεί τον κυριότερο παράγοντα κόστους κατά την κατασκευή ενός ολοκληρωμένου κυκλώματος και όχι η αξίας 2 δισεκατομμυρίων δολαρίων γραμμή παραγωγής της. Σε πολλές επιχειρήσεις το κόστος του εξοπλισμού δοκιμής φτάνει το 50 - 60% του συνολικού κόστους του εξοπλισμού παραγωγής. Πολλές φορές μπορεί να είναι πιο οικονομική η ενσωμάτωση του υλικού δοκιμής εντός του ολοκληρωμένου κυκλώματος, σε σχέση με την χρησιμοποίηση *αυτόματου εξοπλισμού δοκιμής* (*Automatic Test Equipment, ATE*) και αυτό γιατί το κόστος εξοπλισμού δοκιμής είναι ιδιαίτερα υψηλό. Όπως έχει αποδειχθεί [36], η δοκιμή κατά την πραγματική συχνότητα λειτουργίας είναι αποτελεσματικότερη από τη δοκιμή με μειωμένη ταχύτητα. Ωστόσο, ο γρηγορότερος εξοπλισμός δοκιμής είναι πάντα πιο αργός από τα σύγχρονα ολοκληρωμένα κυκλώματα υψηλής συχνότητας λειτουργίας όπως οι επεξεργαστές..



Εικόνα 2.2 Κόστη δοκιμής

Τα παραπάνω επιβεβαιώνονται και από τα διαγράμματα που απεικονίζονται στην Εικόνα 2.2, [37]. Στην Εικόνα 2.2(α) φαίνεται η αύξηση του κόστους του *εξοπλισμού αυτόματης δοκιμής* (*Automatic Test Equipment, ATE*), το οποίο προβλέπεται ότι το 2012 θα ξεπερνά τα 20M$. Η Εικόνα 2.2(β) απεικονίζει το κόστος δοκιμής ανά τρανζίστορ σε σύγκριση με το κόστος

κατασκευής ανά τρανζίστορ. Σύμφωνα με το διάγραμμα τα δύο αυτά κόστη τείνουν να συγκλίνουν στο προσεχές μέλλον.

Στη βιομηχανία μικροηλεκτρονικής είναι ευρέως αποδεκτό ότι τα ολοκληρωμένα κυκλώματα επιβάλλεται να δοκιμαστούν προτού τοποθετηθούν στις *πλακέτες τυπωμένου κυκλώματος* (*Printed Circuit Boards, PCBs*), τα οποία στη συνέχεια πρέπει να δοκιμαστούν προτού ενσωματωθούν στα τελικά συστήματα. Για το κόστος δοκιμής των προϊόντων μικροηλεκτρονικής ισχύει ο *κανόνας του δέκα* (*Rule of ten*) [9]. Εάν ένα ελάττωμα σε ένα ολοκληρωμένο κύκλωμα δεν βρεθεί κατά την δοκιμή, τότε το κόστος ανίχνευσης του ελαττώματος είναι 10 φορές μεγαλύτερο στο επίπεδο της πλακέτας τυπωμένου κυκλώματος απ' ότι στο επίπεδο του ολοκληρωμένου κυκλώματος. Ομοίως, εάν ένα ελάττωμα δεν βρεθεί κατά την δοκιμή στο επίπεδο της πλακέτας τυπωμένου κυκλώματος, η ανίχνευση του έχει 10 φορές μεγαλύτερο κόστος στο επίπεδο τους συστήματος απ' ότι στο επίπεδο της πλακέτας τυπωμένου κυκλώματος. Μερικοί υποστηρίζουν ότι ο κανόνας του δέκα πρέπει να μετονομαστεί σε κανόνα τον είκοσι, καθώς σήμερα τα ολοκληρωμένα κυκλώματα, οι πλακέτες τυπωμένου κυκλώματος και τα συστήματα είναι πολύ πιο πολύπλοκα από ότι ήταν όταν δηλώθηκε αρχικά αυτός ο εμπειρικός κανόνας. Συνεπώς, όσο νωρίτερα ανιχνεύεται ένα ελάττωμα, τόσο μικρότερο είναι το κόστος επιδιόρθωσης του και όσο αποδοτικότερη είναι η δοκιμή, τόσο ποιοτικότερο είναι το προϊόν ως προς τον τελικό χρήστη.

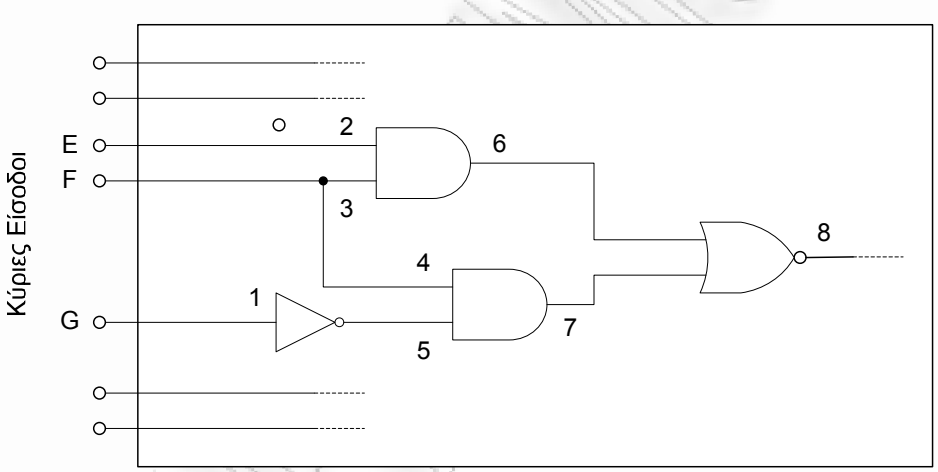
Η ποιότητα ενός προϊόντος εξαρτάται από το βαθμό ικανοποίησης του χρήστη. Ένα προϊόν έχει υψηλή ποιότητα όταν καλύπτει τις απαιτήσεις του τελικού χρήστη με το χαμηλότερο δυνατό κόστος. Σε μια διαδικασία παραγωγής, τα δύο κριτήρια, κάλυψη αναγκών του χρήστη και μείωση του κόστους, καθορίζουν την τελική ποιότητα του προϊόντος. Μια περιεκτική δοκιμή εξετάζει λεπτομερώς ένα ολοκληρωμένο κύκλωμα. Για τα πολύπλοκα ολοκληρωμένα κυκλώματα, τέτοιες δοκιμές μπορεί να είναι πολύ ακριβές. Μια δοκιμή η οποία μπορεί να μειώσει τον αριθμό των ελαττωματικών κυκλωμάτων που θα διαφύγουν της δοκιμής σε έναν αποδεκτό αριθμό μπορεί να θεωρηθεί καλή δοκιμή, ειδικά εάν το κόστος της δοκιμής αυτής είναι επίσης αποδεκτό. Για τα ολοκληρωμένα κυκλώματα, η ποιότητα δοκιμής όπως προαναφέραμε, καθορίζεται από το *επίπεδο ατέλειας* (*defect level*) το οποίο εκφράζεται σε *PPM* (*Parts Per Million*), ή σε *DPPM* (*Defects Parts Per Million*).

2.3 Ελεγξιμότητα και παρατηρησιμότητα

Προτού αναλύσουμε περαιτέρω τις πτυχές της δοκιμής ψηφιακών κυκλωμάτων είναι αναγκαίο να ορίσουμε δύο βασικούς όρους. Αυτοί είναι η *ελεγξιμότητα* (*controllability*) και η *παρατηρησιμότητα* (*obesrvability*). Οι όροι αυτοί εισήχθησαν αρχικά κατά την δεκαετία του '70 σε μία προσπάθεια να καθοριστούν η ευκολία, ή η δυσκολία της δοκιμής ενός ψηφιακού κυκλώματος.

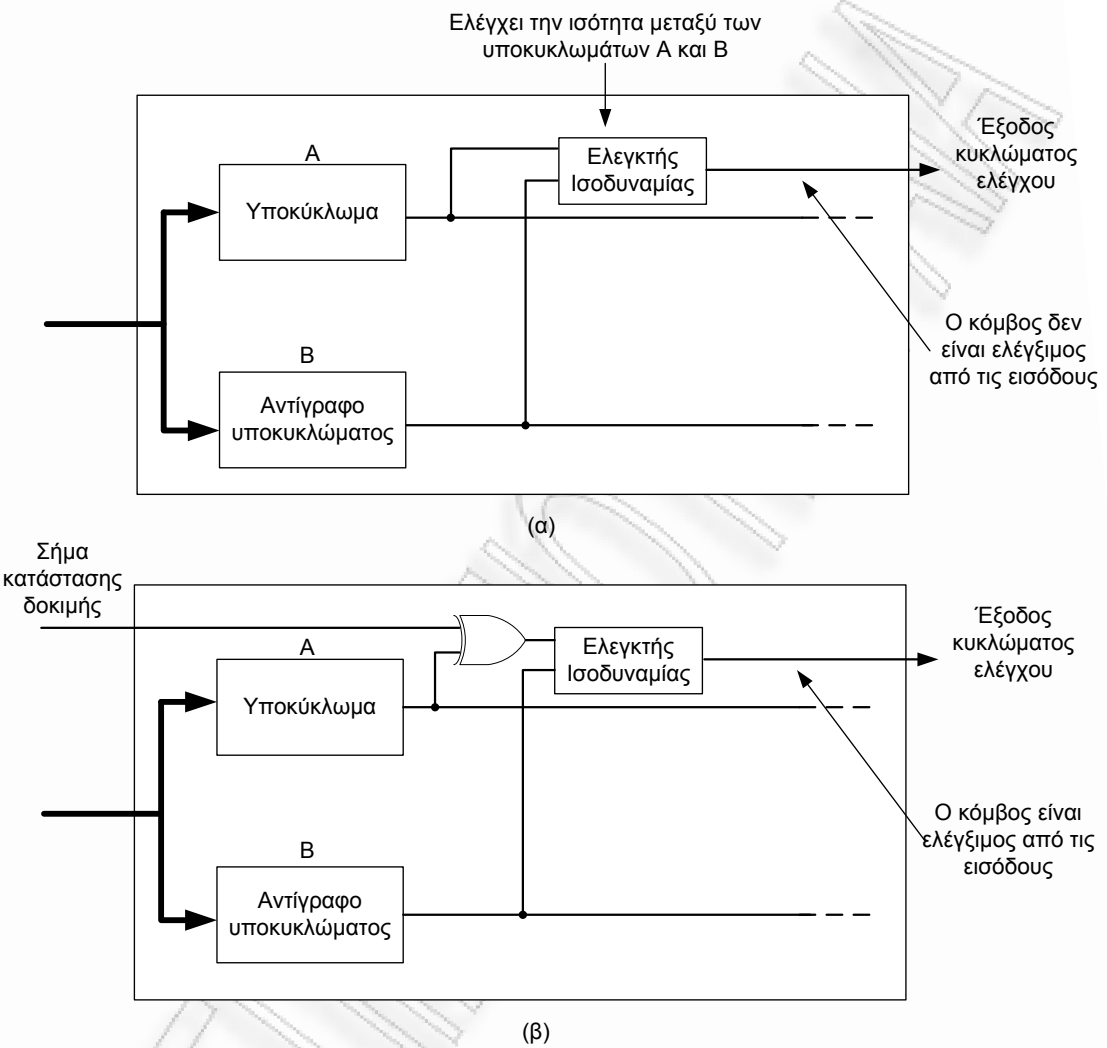
Η *ελεγξιμότητα* (*controllability*) είναι ένα μέτρο που καθορίζει πόσο εύκολα ένας δεδομένος κόμβος σε ένα ψηφιακό κύκλωμα μπορεί να τεθεί σε λογική τιμή "0" ή σε λογική τιμή "1", εφαρμόζοντας τιμές στις *κύριες εισόδους* (*primary inputs*) του κυκλώματος. Ομοίως, η έννοια της *παρατηρησιμότητας* (*observability*) χρησιμοποιείται για να δηλώσει πόσο εύκολα η τιμή ενός δεδομένου κόμβου (λογική τιμή "0" ή λογική τιμή "1"), μπορεί να προσδιοριστεί από τις τιμές των *κύριων εξόδων* (*primary inputs*).

Για παράδειγμα, θεωρήστε το κύκλωμα που απεικονίζεται στην Εικόνα 2.3. Οι κόμβοι 2, 3, και 4 είναι άμεσα ελέγξιμοι, δεδομένου ότι συνδέονται άμεσα με τις κύριες εισόδους. Ο κόμβος 8 από την άλλη δεν άμεσα ελέγξιμος. Για να θέσουμε τον κόμβο 8 στην τιμή 0(1) πρέπει είτε να θέσουμε τον κόμβο 7 στην τιμή "0" και τον κόμβο 6 στην τιμή 1(0), είτε αντίστροφα να θέσουμε τον κόμβο 6 στην τιμή "0" και τον κόμβο 7 στην τιμή 0(1). Ένα σύνολο διανυσμάτων εισόδου που δίνει τις τιμές "0" και "1" στον κόμβο 8, προκύπτει αν θέσουμε τις εισόδους F και G στη λογική τιμή "1" και την είσοδο E είτε στην τιμή "0" είτε στην τιμή "1". Συνεπώς, όσο πιο "απομακρυσμένος" είναι ένας κόμβος από τις κύριες εισόδους τόσο πιο δύσκολο γενικά είναι να ελέγξουμε την τιμή του κόμβου αυτού.



Εικόνα 2.3 Παράδειγμα ελεγξιμότητας ενός κυκλώματος

Μερικοί κόμβοι σε ένα κύκλωμα μπορεί να μην είναι ελέγξιμοι από τις κύριες εισόδους. Θεωρήστε τον *ελεγκτή ισοδυναμίας* (*equality checker*) που παρουσιάζεται στην Εικόνα 2.4(α). Ο ελεγκτής ισοδυναμίας εξετάζει την ισότητα μεταξύ των δύο υποκυκλωμάτων Α και Β. Σε ένα κύκλωμα χωρίς ελαττώματα τα αποτελέσματα από τα δύο υποκυκλώματα θα είναι πάντα ίδια. Συνεπώς, δεν μπορούμε να θέσουμε τις εξόδους των υποκυκλωμάτων Α και Β σε διαφορετική τιμή, ώστε να επηρεάσουμε με αυτόν τον τρόπο την τιμή της εξόδου του ελεγκτή ισοδυναμίας. Μπορούμε όμως να προσθέσουμε μια πύλη όπως φαίνεται στην Εικόνα 2.4(β) προκειμένου να αυξήσουμε την ελεγξιμότητα της εξόδου του κυκλώματος, δεδομένου ότι το *σήμα κατάστασης δοκιμής* (*test mode signal*), μπορεί να αντιστρέψει την τιμή της εξόδου του υποκυκλώματος Α.

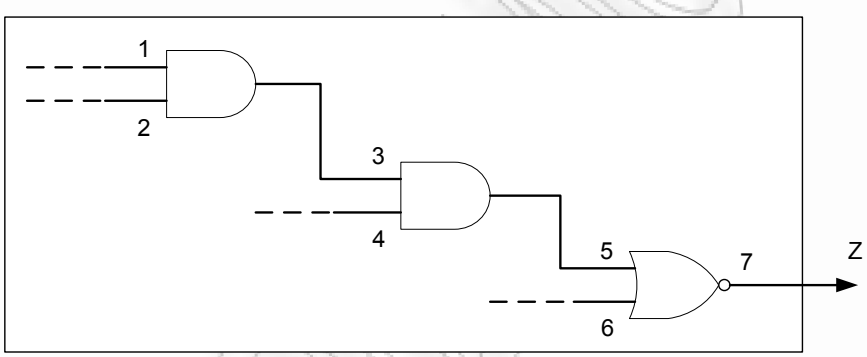


Εικόνα 2.4 Παράδειγμα κόμβου κυκλώματος που δεν είναι άμεσα ελέγξιμος από τις κύριες εισόδους: (α) Αρχικό κύκλωμα, (β) Προσθήκη σήματος δοκιμής για αύξηση ελεγξιμότητας.

Κυκλώματα που περιέχουν *πλεονασμό* (*redundancy*), όπως αυτό που παρουσιάζεται στην Εικόνα 2.4, εμφανίζουν έμφυτες δυσκολίες ως προς την ελεγξιμότητα. Σε αυτά τα κυκλώματα επιπρόσθετες *κύριες είσοδοι* (*primary inputs*) κρίνονται απαραίτητες για λόγους δοκιμής. Προβλήματα ελεγξιμότητας παρουσιάζονται επίσης στις περιπτώσεις όπου το ολοκληρωμένο κύκλωμα περιέχει μια δομή μνήμης μόνο για ανάγνωση (Read Only Memory, ROM), τα περιεχόμενα της οποίας χρησιμοποιούνται για να οδηγήσουν την υπόλοιπη λογική του κυκλώματος. Εδώ, η προγραμματισμένη ROM μπορεί να αποκλείσει την εφαρμογή ορισμένων συνδυασμών στα σήματα της λογικής, με αποτέλεσμα να περιορίζεται η ελεγξιμότητα αυτών των σημάτων.

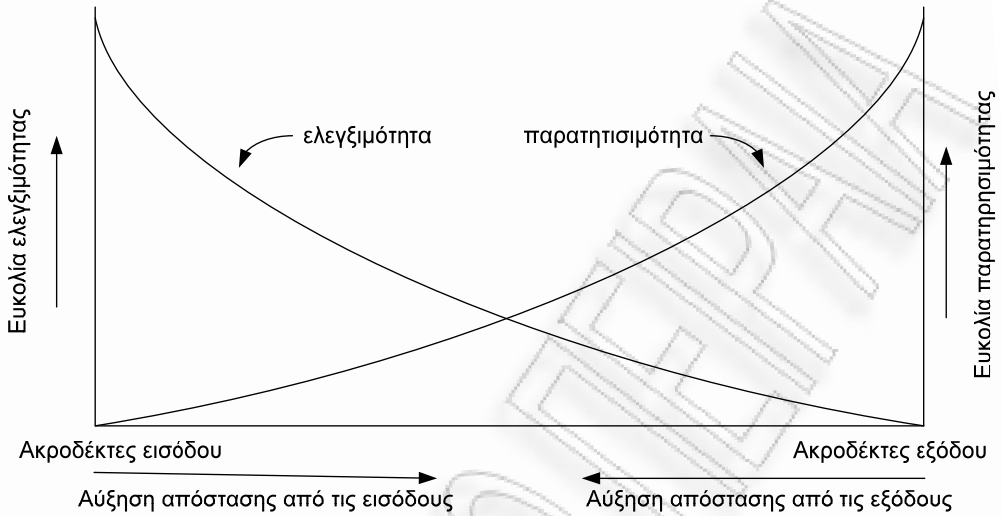
Σε ακολουθιακά κυκλώματα που περιέχουν κυκλώματα *μανδάλωσης* (*latches*) και *φλιπ-φλοπ* (*flip-flop*), η δοκιμή για ορισμένους κόμβους είναι συχνά πολύ δύσκολη ή ακόμα και αδύνατη, δεδομένου ότι ένας πολύ μεγάλος αριθμός διανυσμάτων δοκιμής θα πρέπει να εφαρμοστεί ώστε να αλλάξει η τιμή του κόμβου. Επιπλέον, συχνά υπάρχουν ορισμένες καταστάσεις του κυκλώματος που δεν χρησιμοποιούνται ποτέ. Παράδειγμα αποτελεί ένας *δεκαδικός μετρητής με δυαδική κωδικοποίηση* (*Binary-Coded Decimal Counter, BCD Counter*) των 4 bit ο οποίος μετράει μόλις δέκα από τους 16 πιθανούς δυαδικούς συνδυασμούς.

Επιστρέφοντας στην παρατηρησιμότητα, θεωρήστε το απλό κύκλωμα που παρουσιάζεται στην Εικόνα 2.5. Υποθέστε ότι πρέπει να παρατηρήσουμε την τιμή του κόμβου 2. Για να μεταδώσουμε την λογική τιμή του κόμβου 2 στην έξοδο Z, είναι σαφές ότι οι κόμβοι 1 και 4 πρέπει να τεθούν σε λογική τιμή "1" και ο κόμβος 6 σε λογική τιμή "0". Ως εκ τούτου, τα αρχικά σήματα πρέπει να επιλεχθούν έτσι ώστε οι κόμβοι 1, 4 και 6, να πάρουν αυτές τις λογικές τιμές, ώστε στη προκειμένη περίπτωση η λογική τιμή της εξόδου Z να εξαρτάται αποκλειστικά και μόνο από την τιμή του κόμβου 2.



Εικόνα 2.5 Παράδειγμα παρατηρησιμότητας ενός κυκλώματος

Τα γενικά χαρακτηριστικά της ελεγξιμότητας και της παρατηρησιμότητας για οποιοδήποτε κύκλωμα απεικονίζονται στην Εικόνα 2.6. Υπό τον όρο ότι δεν υπάρχει κανένας πλεονασμός στο ολοκληρωμένο κύκλωμα, τότε είναι πάντα δυνατόν να καθοριστούν δύο (ή περισσότερα) διανύσματα εισόδου που θα ελέγχουν κάθε εσωτερικό κόμβο του κυκλώματος εάν περιέχει ελαττώματα ή όχι. Ωστόσο, η πολυπλοκότητα της εύρεσης του μικρότερου δυνατού συνόλου διανυσμάτων δοκιμής που είναι ικανά εξετάσουν όλους τους εσωτερικούς κόμβους είναι υψηλή και ο υπολογισμός τους με το χέρι, εκτός από την περίπτωση των εξαιρετικά απλών κυκλωμάτων, σχεδόν αδύνατος.



Εικόνα 2.6 Γενικά χαρακτηριστικά ελεγξιμότητας και παρατηρησιμότητας

Πολλές ερευνητικές προσπάθειες [38], [39], [40], [41], [42], [43] επιχειρούν να καθορίσουν την ελεγξιμότητα και την παρατηρησιμότητα ενός δεδομένου κυκλώματος, ώστε να είναι εφικτό να προσδιοριστούν τα κυκλώματα που παρουσιάζουν έμφυτες δυσκολίες δοκιμής και να τροποποιηθούν κατά τη διάρκεια της σχεδίασης. Οι βασικές έννοιες που χρησιμοποιούνται στην πλειοψηφία αυτών των προσεγγίσεων περιλαμβάνουν:

1. Τον υπολογισμό ενός αριθμού που αντιπροσωπεύει το βαθμό δυσκολίας να θέσουμε κάθε εσωτερικό κόμβου του κυκλώματος υπό δοκιμή, στη λογική τιμή "0" και "1" (0-ελεγξιμότητα και 1-ελεγξιμότητα).

2. Τον υπολογισμό ενός άλλου αριθμού που αντιπροσωπεύει τη δυσκολία να διαδώσουμε τη λογική τιμή ενός κόμβου στις κύριες εξόδους του κυκλώματος.

Η δυσκολία δοκιμής ενός κυκλώματος συσχετίζεται με κάποια γενική εκτίμηση αυτών των μεμονωμένων αριθμητικών τιμών. Στην πλειοψηφία αυτών των προσεγγίσεων, η ελεγξιμότητα κανονικοποιήθηκε μεταξύ των τιμών "0" και "1", με την τιμή "0" να αντιπροσωπεύει έναν κόμβο που δεν μπορεί να ελεγχτεί από τις κύριες εισόδους και την τιμή "1" να αντιπροσωπεύει έναν κόμβο με άμεση ελεγξιμότητα. Χαρακτηριστικά, ένας *παράγοντας μεταφοράς ελεγξιμότητας,* (*Controllability Transfer Factor, CTF*), για κάθε τύπο συνδυαστικής πύλης ορίζετε από την εξίσωση:

$$CFT = \left| \frac{N(0) - N(1)}{N(0) + N(1)} \right|$$

όπου N(0) και N(1) είναι ο αριθμός των διανυσμάτων δοκιμής για τα οποία η έξοδος του κυκλώματος έχει λογική τιμή "0" και λογική τιμή "1", αντίστοιχα.

2.4 Μοντέλα Ελαττωμάτων

Τα *μοντέλα λογικών ελαττωμάτων* (*logical fault models*) προσεγγίζουν την συμπεριφορά των *φυσικών ατελειών* (*physical defects*) καθιστώντας υπολογιστικά εφικτή την εξαγωγή διανυσμάτων δοκιμής. Η εκτίμηση των πιθανών ελαττωμάτων σε ένα ψηφιακό κύκλωμα πραγματοποιείται προκειμένου να καθοριστεί ένα ελάχιστο σύνολο διανυσμάτων δοκιμής, το οποίο θα εξετάσει εάν τα ελαττώματα είναι ή δεν είναι παρόντα σε ένα ολοκληρωμένο κύκλωμα. Εάν κανένα από τα πιθανά ελαττώματα δεν ανιχνεύεται, τότε το κύκλωμα θεωρείται ότι δεν περιέχει ελαττώματα.

Τα *λογικά ελαττώματα* (*logical faults*) αντιπροσωπεύουν το αποτέλεσμα των *φυσικών ελαττωμάτων* (*physical faults*) στην συμπεριφορά του μοντέλου ενός συστήματος. Στην μοντελοποίηση των διαφόρων ελαττωμάτων κάνουμε έναν διαχωρισμό μεταξύ της λογικής συνάρτησης και του *χρονισμού* (*timing*). Έτσι, διακρίνουμε ελαττώματα που επηρεάζουν την λογική συνάρτηση και ελαττώματα καθυστέρησης που επηρεάζουν την ταχύτητα λειτουργίας του κυκλώματος.

Ένα μοντέλο για λογικά ελαττώματα μπορεί να είναι είτε ρητά ορισμένο είτε υπονοούμενο. Στην πρώτη περίπτωση έχουμε το *ρητό μοντέλο ελαττωμάτων* (*explicit fault model*), ενώ στην δεύτερη το *υπονοούμενο μοντέλο ελαττωμάτων* (*implicit fault model*). Στο ρητό μοντέλο ελαττωμάτων, κάθε ελάττωμα είναι αναγνωρίσιμο με αποτέλεσμα να μπορούμε να απαριθμήσουμε τα ελαττώματα που περιέχονται στο κύκλωμα. Στο υπονοούμενο μοντέλο ελαττωμάτων γίνεται μια περιληπτική περιγραφή των ελαττωμάτων, όπου αναγνωρίζονται και καθορίζονται οι χαρακτηριστικές τους ιδιότητες.

Τα μοντέλα ελαττωμάτων αποτελούν μια προσέγγιση και δεν μπορούν να περιγράψουν με απόλυτη ακρίβεια τη συμπεριφορά των ψεγαδιών/ατελειών σε ένα κύκλωμα. Αυτό έχει ως αποτέλεσμα, κυκλώματα που περνούν επιτυχώς τη δοκιμή και αποστέλλονται στην αγορά, να περιέχουν ατέλειες. Στην προσπάθεια να προσεγγιστεί όσο το δυνατό καλύτερα η συμπεριφορά των ατελειών, έχουν αναπτυχθεί διάφορα μοντέλα ελαττωμάτων. Μια κατηγοριοποίηση μπορεί να γίνει με βάση το *επίπεδο αφαίρεσης* (*abstraction level*) περιγραφής του κυκλώματος.

- *Δομικά ελαττώματα* (*structural faults*): Είναι ελαττώματα που σχετίζονται με το δομικό μοντέλο του συστήματος, με αποτέλεσμα να επηρεάζουν τις συνδέσεις μεταξύ των διαφόρων τμημάτων. Στα δομικά μοντέλα ελαττωμάτων, υποθέτουμε ότι τα διάφορα τμήματα δεν έχουν λάθη και ότι οι μεταξύ τους συνδέσεις μπορεί είτε να έχουν *διακοπεί* (*opens*) είτε να έχουν *βραχυκυκλώσει* (*shorts*).

- *Λειτουργικά ελαττώματα* (*functional faults*)*:* Είναι ελαττώματα που σχετίζονται με το λειτουργικό μοντέλο του συστήματος. Έτσι για παράδειγμα, ένα λειτουργικό ελάττωμα μπορεί να επιφέρει αλλαγές στον πίνακα αλήθειας μιας πύλης.

- *Περιοδικά και παροδικά ελαττώματα* (*intermittent and transient faults*)*:* Είναι ελαττώματα που δεν εμφανίζονται συνεχώς και χρειάζονται στατιστικά δεδομένα για να καθορίσουμε την πιθανότητα να εμφανισθούν.

Τα μοντέλα ελαττωμάτων που συνήθως χρησιμοποιούνται στην βιομηχανία ημιαγωγών είναι τα ακόλουθα:

- *Μοντέλο ελαττώματος προσκόλλησης* (*stuck-at fault model*)

- *Μοντέλο ελαττώματος καθυστέρησης μετάβασης* (*transition delay fault model*)

- *Μοντέλο ελαττώματος καθυστέρησης μονοπατιού* (*path delay fault model*)

2.4.1 Μοντέλο ελαττώματος προσκόλλησης

Το πιο διαδεδομένο μοντέλο ελαττώματος στα ψηφιακά κυκλώματα είναι το *μοντέλο ελαττώματος προσκόλλησης μίας μόνο γραμμής* (*single stuck-at fault model*) [12]. Το μοντέλο αυτό υποθέτει ότι οποιαδήποτε φυσική ατέλεια σε ένα ψηφιακό κύκλωμα οδηγεί έναν κόμβο στο κύκλωμα (την είσοδο ή την έξοδο μιας πύλης), να παραμένει *προσκολλημένο σε λογική κατάσταση "0"* (*stuck-at-0, SA0*), ή να παραμένει *προσκολλημένο σε λογική κατάσταση "1",* (*stuck-at-1, SA1*). Αυτό το ελάττωμα μπορεί να βρίσκεται είτε στην ίδια την πύλη, είτε στις διασυνδέσεις της πύλης, έτσι ώστε ο κόμβος να μην μπορεί πλέον να αλλάξει τιμή μεταξύ των καταστάσεων "0" και "1". Αν ένα κύκλωμα αποτελείται από k γραμμές, το πλήθος των ελαττωμάτων προσκόλλησης είναι $2k$. Ακόμα κι αν ένας κόμβος είναι προσκολλημένος σε μια τιμή διαφορετική από τις λογικές τιμές "0" ή "1", οι επόμενες λογικές πύλες θα ερμηνεύσουν την τιμή αυτήν είτε σαν λογική τιμή "0" είτε σαν λογική τιμή "1".

Στην πραγματικότητα, περισσότερες από μία γραμμές μπορεί να βρίσκονται προσκολλημένες σε μία τιμή. Το *μοντέλο ελαττώματος προσκόλλησης πολλαπλών γραμμών* (*multiple stuck-at fault model*) εκφράζει την παραπάνω περίπτωση. Αν ένα κύκλωμα έχει k γραμμές, το συνολικό πλήθος ελαττωμάτων σε πολλαπλές γραμμές είναι 3^k-1. Είναι ευνόητο πως ακόμα και για μικρά κυκλώματα το πλήθος των ελαττωμάτων προσκόλλησης σε πολλαπλές γραμμές είναι απαγορευτικά μεγάλο για εξαγωγή διανυσμάτων δοκιμής. Ωστόσο, έχει αποδειχτεί πως σε μεγάλα κυκλώματα τα διανύσματα δοκιμής για ελαττώματα προσκόλλησης μιας μόνο γραμμής, ανιχνεύουν τουλάχιστον το 99,6% των ελαττωμάτων προσκόλλησης πολλαπλών γραμμών [44], [45].

Είσοδοι ABC	Έξοδος Z								
	Χωρίς Ελαττώματα	A SA0	A SA1	B SA0	B SA1	C SA0	C SA1	Z SA0	Z SA1
000	1	1	1	1	1	1	1	**(0)**	1
001	1	1	1	1	1	1	1	**(0)**	1
010	1	1	1	1	1	1	1	**(0)**	1
011	1	1	**(0)**	1	1	1	1	**(0)**	1
100	1	1	1	1	1	1	1	**(0)**	1
101	1	1	1	1	**(0)**	1	1	**(0)**	1
110	1	1	1	1	1	1	**(0)**	**(0)**	1
111	0	**(1)**	0	**(1)**	0	**(1)**	0	0	**(1)**

Πίνακας 2.1 Τα πιθανά ελαττώματα προσκόλλησης σε μία πύλη NAND τριών εισόδων A,B,C. Οι λανθασμένες τιμές εξόδου είναι σε παρένθεση.

Διανύσματα δοκιμής ABC	Έξοδος		Ελαττώματα που ανιχνευθήκαν από τα διανύσματα δοκιμής
	Χωρίς ελαττώματα	Με παρουσία ελαττωμάτων	
011	1	0	A SA1 or Z SA0
101	1	0	B SA1 or Z SA0
110	1	0	C SA1 or Z SA0
111	0	1	A or B or C SA0 or Z SA1

Πίνακας 2.2 Τα ελάχιστα διανύσματα δοκιμής για την ανίχνευση όλων των πιθανών ελαττωμάτων προσκόλλησης σε μια πύλη NAND τριών εισόδων

Για παράδειγμα, σε μια πύλη NAND τριών εισόδων υπάρχουν οκτώ πιθανά ελαττώματα προσκόλλησης, όπως απεικονίζει ο Πίνακας 2.1. Ωστόσο, τέσσερα από αυτά (SA0 στις γραμμές A,B,C και Z) έχουν το ίδιο αποτέλεσμα στην έξοδο της πύλης. Συνολικά, τέσσερα διανύσματα δοκιμής (Πίνακας 2.2) θα ανιχνεύσουν (αλλά δεν θα προσδιορίσουν) την παρουσία και των οκτώ πιθανών ελαττωμάτων προσκόλλησης. Παρόμοιες αναλύσεις μπορούν να παρουσιαστούν και για όλες τις άλλες λογικές πύλες, όπου σε όλες τις περιπτώσεις απαιτούνται λιγότερα από 2^n διανύσματα δοκιμής για να ανιχνεύσουν όλα τα πιθανά ελαττώματα προσκόλλησης. Το μοντέλο ελαττώματος προσκόλλησης αποτελεί το δημοφιλέστερο μοντέλο ελαττωμάτων και έτσι έχουν αναπτυχθεί αρκετοί αλγόριθμοι εξαγωγής διανυσμάτων για αυτό. Οι αλγόριθμοι D [46], PODEM [47], FAN [48] εξάγουν διανύσματα δοκιμής για ελαττώματα προσκόλλησης, σε συνδυαστικά κυκλώματα.

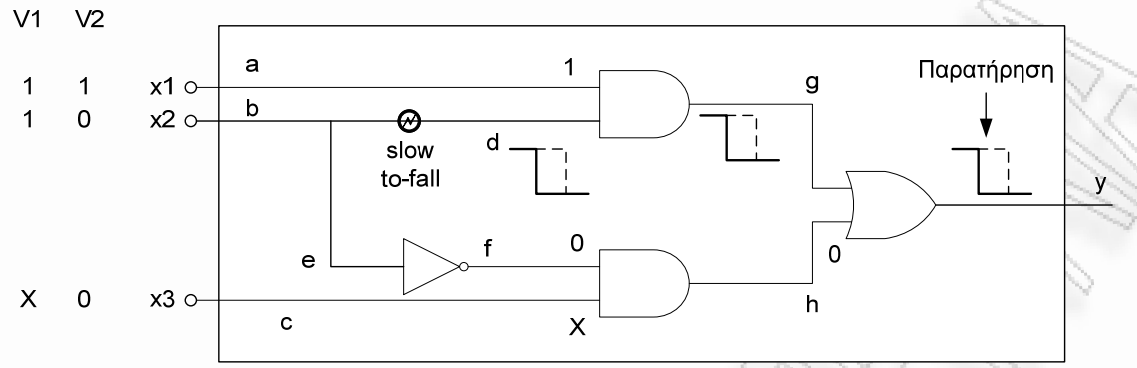
2.4.2 Μοντέλο ελαττώματος καθυστέρησης μετάβασης

Το *μοντέλο ελαττώματος καθυστέρησης μετάβασης* (*transition delay fault model*) [49], [50], [51], [52] υποθέτει ότι ένα ελάττωμα καθυστέρησης έχει επιπτώσεις μόνο σε μια πύλη του κυκλώματος. Υπάρχουν δύο ελαττώματα μετάβασης που συνδέονται με κάθε πύλη: την *μετάβαση από την τιμή 0 στην τιμή 1* (*slow-to-rise*) και την *μετάβαση από την τιμή 1 στην τιμή 0* (*slow-to-fall*).

Σύμφωνα με το μοντέλο ελαττώματος καθυστέρησης μετάβασης, η πρόσθετη καθυστέρηση που προκαλείται από ένα ελάττωμα υποτίθεται ότι είναι αρκετά μεγάλη ώστε να μην επιτρέπει στην μετάβαση να φτάσει σε οποιαδήποτε κύρια έξοδο μέσα στο χρονικό παράθυρο της παρατήρησης. Με άλλα λόγια, το ελάττωμα καθυστέρησης μετάβασης μπορεί να παρατηρηθεί είτε η μετάβαση διαδίδεται μέσω μιας μακριάς, είτε μέσω μιας κοντινής διαδρομής, σε οποιαδήποτε *κύρια έξοδο* (*primary output*). Το μοντέλο ελαττωμάτων καθυστέρησης μετάβασης χρησιμοποιείται και ως μοντέλο για *ελαττώματα μόνιμα ανοικτού* τρανζίστορ (*stuck-open faults*) [53]. Τα ελαττώματα μόνιμα ανοικτού τρανζίστορ μπορούν να αντιμετωπιστούν ως ελαττώματα, που είτε καταστέλλουν, είτε καθυστερούν ορισμένες μεταβάσεις. Στην πράξη, η πρόσθετη καθυστέρηση που προκαλείται από ένα ελάττωμα μόνιμα ανοιχτού τρανζίστορ, εξαρτάται από τα ηλεκτρικά χαρακτηριστικά του ελαττωματικού τρανζίστορ.

Για να ανιχνευθεί ένα ελάττωμα καθυστέρησης μετάβασης σε ένα συνδυαστικό κύκλωμα, είναι απαραίτητο να εφαρμοστούν δύο διανύσματα, $V = (V1, V2)$. Το πρώτο διάνυσμα, *V1*, αρχικοποιεί το κύκλωμα, ενώ το δεύτερο διάνυσμα, *V2*, ενεργοποιεί το ελάττωμα και διαδίδει την επίδρασή του σε κάποια κύρια έξοδο. Για παράδειγμα, για τη δοκιμή της καθυστέρησης μετάβασης ενός κόμβου από την τιμή “0” στην τιμή “1”, το πρώτο διάνυσμα αρχικοποιεί τον κόμβο στην τιμή “0” και το δεύτερο διάνυσμα είναι ένα διάνυσμα δοκιμής για το ελάττωμα προσκόλλησης *SA0* στο συγκεκριμένο κόμβο.

Η Εικόνα 2.7 παρουσιάζει την ανίχνευση του ελαττώματος καθυστέρησης μετάβασης από την τιμή “1” στη τιμή “0” για την γραμμή *d*. Αρχικά το διάνυσμα *V1* θέτει την τιμή “1” στην είσοδο *x2* οπότε και η γραμμή *d* τίθεται στην τιμή “1”. Κατόπιν το διάνυσμα δοκιμής *V2* θέτει την τιμή “0” στην είσοδο *x2* δημιουργώντας μια μετάβαση από την τιμή “1” στην τιμή “0” για την γραμμή *d*. Οι τιμές των εισόδων *x1, x3* λαμβάνουν κατάλληλες τιμές, ώστε η μετάβαση να διαδοθεί στην κύρια έξοδο *y*. Εξαιτίας του ελαττώματος καθυστέρησης μετάβασης, η μετάβαση από την τιμή “1” στην τιμή “0” αργεί να ολοκληρωθεί (η καθυστέρηση παρουσιάζεται με διακεκομμένη γραμμή στην Εικόνα 2.7). Τη χρονική στιγμή της παρατήρησης, η μετάβαση δεν έχει φτάσει στην κύρια έξοδο *y,* οπότε η τιμή της εξόδου είναι “1” ενώ θα έπρεπε να είναι “0”. Η παρατήρηση αυτή οδηγεί στο συμπέρασμα πως υπάρχει κάποιο ελάττωμα καθυστέρησης μετάβασης στο κύκλωμα, επομένως το ελάττωμα καθυστέρησης μετάβασης από την τιμή “1” στην τιμή “0” για την γραμμή *d* έχει ανιχνευθεί.



Εικόνα 2.7 Παράδειγμα ελαττώματος καθυστέρησης μετάβασης 1→ 0 (slow-to-fall,STF) της γραμμής d

Το *μοντέλο ελαττώματος καθυστέρησης πύλης* (*gate delay fault model*) [54] επίσης υποθέτει πως υπάρχει ελάττωμα καθυστέρησης σε μία μόνο πύλη, το οποίο καθυστερεί την μετάβαση της εξόδου της πύλης είτε από την τιμή "0" στην τιμή "1", είτε από την τιμή "1" στην τιμή "0". Αντίθετα από το μοντέλο καθυστέρησης μετάβασης, η χρονική καθυστέρηση δεν είναι βέβαιο ότι θα παρατηρηθεί στις κύριες εξόδους. Η παρατήρηση εξαρτάται από την καθυστέρηση του μονοπατιού, μέσα από το οποίο θα διαδοθεί η μετάβαση. Το μοντέλο ελαττώματος καθυστέρησης πύλης βασίζεται στα φυσικά χαρακτηριστικά καθυστέρησης των πυλών. Το πλήθος των ελαττωμάτων καθυστέρησης σύμφωνα με τα παραπάνω μοντέλα είναι ανάλογο του αριθμού των γραμμών του κυκλώματος. Το μοντέλο αυτό γενικά δεν χρησιμοποιείται ιδιαίτερα στην βιομηχανία.

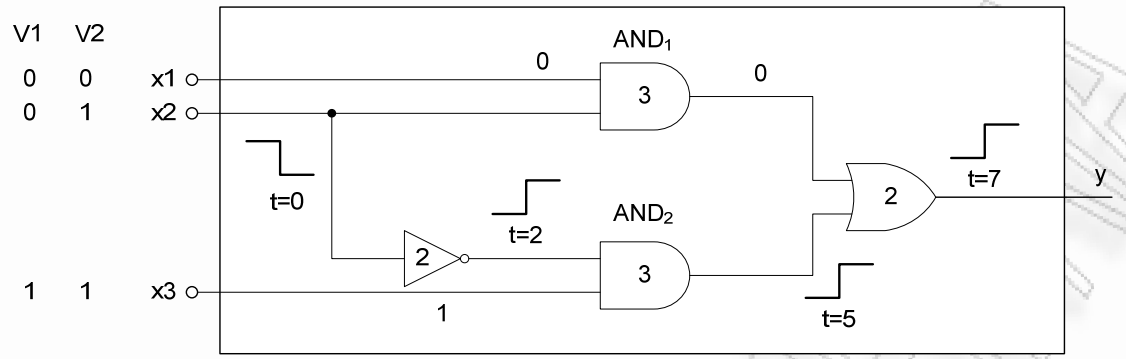
2.4.3 Μοντέλο ελαττώματος καθυστέρησης μονοπατιού

Κατά το *μοντέλο ελαττωμάτων καθυστέρησης μονοπατιού* (*path delay fault model*) [55], ένα συνδυαστικό κύκλωμα θεωρείται ελαττωματικό εάν η καθυστέρηση οποιουδήποτε μονοπατιού του υπερβαίνει ένα συγκεκριμένο όριο. Ένα μονοπάτι ορίζεται ως ένα διατεταγμένο σύνολο πυλών {*g0,g1,⋯,gn*), όπου τα *g0* και *gn* αντιστοιχούν στην κύρια είσοδο και έξοδο αντίστοιχα, ενώ η έξοδος της πύλης *gi* αποτελεί την είσοδο της πύλης *gi+1* ($0 \le i \le n-1$). Ένα ελάττωμα καθυστέρησης μονοπατιού σε ένα μονοπάτι, είναι δυνατόν να παρατηρηθεί με τη διάδοση μιας μετάβασης μέσω του μονοπατιού αυτού. Συνεπώς, η προδιαγραφή ενός ελαττώματος καθυστέρησης μονοπατιού αποτελείται από ένα φυσικό μονοπάτι και μια μετάβαση που θα εφαρμοστεί στην αρχή αυτού του μονοπατιού. Η καθυστέρηση, ή το μήκος του μονοπατιού, αντιπροσωπεύει το άθροισμα των καθυστερήσεων των πυλών και των διασυνδέσεων σε αυτό το μονοπάτι.

Οι δοκιμές για το μοντέλο ελαττωμάτων καθυστέρησης μονοπατιού, μπορούν να ανιχνεύσουν τα μικρά κατανεμημένα λάθη καθυστέρησης. Ένας σημαντικός περιορισμός αυτού

του μοντέλου ελαττωμάτων, είναι ότι ο αριθμός των μονοπατιών σε ένα κύκλωμα μπορεί να είναι ιδιαίτερα μεγάλος (εκθετικός ως προς τον αριθμό των πυλών). Για αυτόν τον λόγο, η εξέταση όλων των πιθανών ελαττωμάτων καθυστέρησης μονοπατιού για ένα κύκλωμα δεν είναι ιδιαίτερα πρακτική. Δύο κύριες μεθοδολογίες χρησιμοποιούνται για την επιλογή του συνόλου ελαττωμάτων καθυστέρησης μονοπατιού. Σύμφωνα με την πρώτη προσέγγιση εξάγεται ένα ελάχιστο σύνολο μονοπατιών ως εξής: για κάθε γραμμή στο κύκλωμα επιλέγεται το μακρύτερο μονοπάτι [56], [57], [58]. Κατά την δεύτερη προσέγγιση επιλέγονται όλα τα μακρύτερα μονοπάτια, δηλαδή όλα τα μονοπάτια που η καθυστέρηση τους υπερβαίνει ένα προσδιορισμένο κατώτατο όριο. Ο λόγος της επιλογής των μακρύτερων μονοπατιών υφίστανται, καθώς τα λάθη καθυστέρησης στα κοντά μονοπάτια είναι πιθανόν να μην είναι αρκετά μεγάλα ώστε να έχουν επιπτώσεις στην απόδοση του κυκλώματος. Ωστόσο, σε βελτιστοποιημένα ως προς την απόδοση κυκλώματα, όπως οι ενσωματωμένοι επεξεργαστές, σχεδόν όλα τα μονοπάτια έχουν μεγάλες καθυστερήσεις, γεγονός που καθιστά αδύνατη την δοκιμή όλων των μακριών μονοπατιών [59]. Επομένως, ακόμα και μετά από τη δοκιμή ελαττωμάτων καθυστέρησης μονοπατιού, η χρονική ακρίβεια του κυκλώματος υπό δοκιμή συχνά δεν είναι εγγυημένη. Το πρόβλημα μπορεί να μειωθεί από τεχνικές οι οποίες εναπασυνθέτουν τα κυκλώματα με απώτερο σκοπό την μείωση του αριθμού των μονοπατιών [60], [61].

Όπως και στα ελαττώματα καθυστέρησης μετάβασης, για να ανιχνεύσουμε τα ελαττώματα καθυστέρησης μονοπατιού απαιτείται να εφαρμοστούν δύο διανύσματα δοκιμής, όπου το πρώτο θα αρχικοποιήσει το κύκλωμα, ενώ το δεύτερο θα ενεργοποιήσει το ελάττωμα και θα διαδώσει την επίδρασή του σε κάποια κύρια έξοδο. Για παράδειγμα, θεωρήστε το κύκλωμα που απεικονίζεται στην Εικόνα 2.8, στο οποίο η καθυστέρηση διάδοσης χωρίς ελαττώματα αναγράφεται σαν ετικέτα σε κάθε πύλη με ένα ακέραιο αριθμό. Τα δύο διανύσματα δοκιμής (*V1* και *V2*), χρησιμοποιούνται για να ανιχνεύσουν το ελάττωμα καθυστέρησης μονοπατιού από την είσοδο *x2*, η οποία οδηγείται από τον αντιστροφέα (NOT), την πύλη AND_2 και την πύλη OR, στην έξοδο y του κυκλώματος. Υποθέτοντας ότι η μετάβαση μεταξύ των δύο διανυσμάτων δοκιμής συμβαίνει την χρονική στιγμή *t=0*, η μετάβαση προωθείται στο κύκλωμα την χρονική στιγμή *t*=2 στην έξοδο του αντιστροφέα, *t*=5 στην έξοδο της πύλης AND_2 και τέλος η αναμενόμενη μετάβαση εμφανίζεται στην έξοδο την χρονική στιγμή *t*=7. Ένα ελάττωμα καθυστέρησης μονοπατιού για το συγκεκριμένο μονοπάτι θα δημιουργήσει καθυστέρηση στη μετάβαση η οποία θα εμφανιστεί στην έξοδο σε χρονική στιγμή $t > 7$.

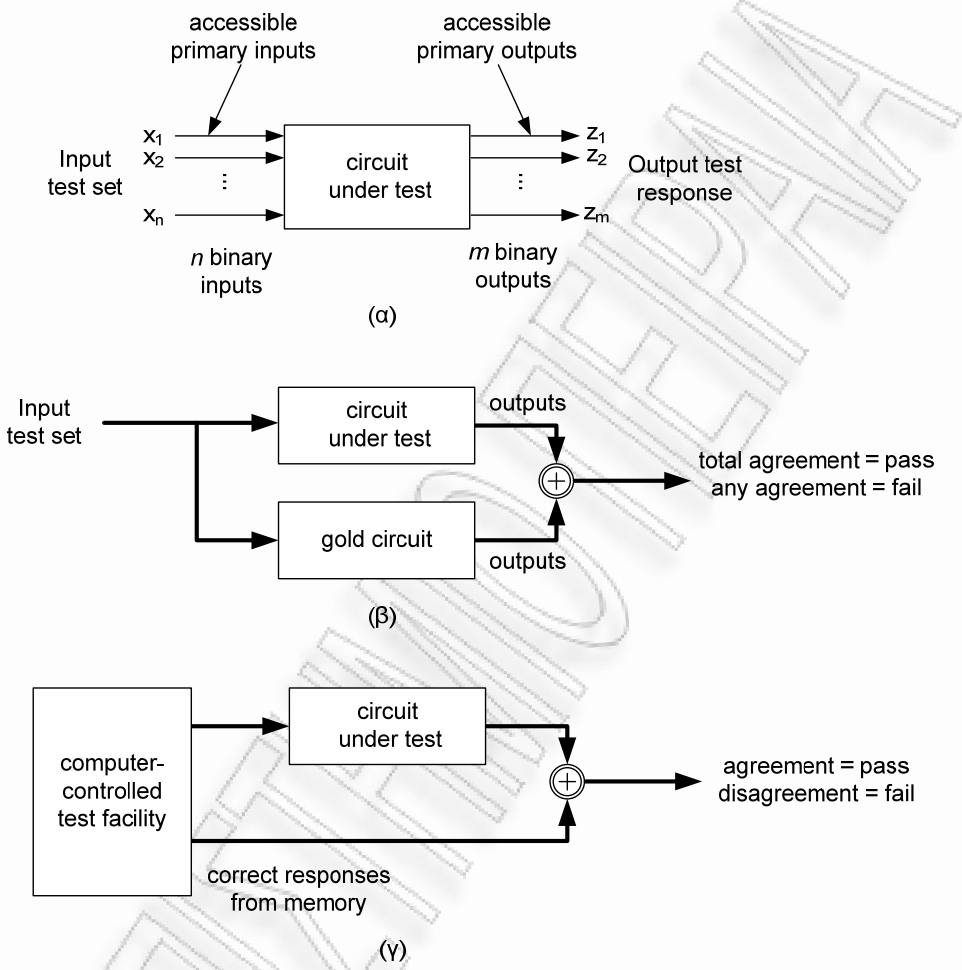


Εικόνα 2.8 Παράδειγμα ελαττώματος καθυστέρησης μονοπατιού

2.5 Προσομοίωση ελαττωμάτων

Κάθε δοκιμή περιλαμβάνει μια διαδικασία προσομοίωσης, όπως αυτή που παρουσιάζεται στην Εικόνα 2.9(α). Σε κάθε βήμα της προσομοίωσης κάθε *απόκριση δοκιμής* (*test response*) πρέπει να ελεγχθεί. Ο έλεγχος των αποκρίσεων απαιτεί την προγενέστερη γνώση των σωστών αποκρίσεων και την σύγκριση αυτών με τις αποκρίσεις του κυκλώματος υπό δοκιμή. Για τα πολύ απλά κυκλώματα, είναι δυνατόν να χρησιμοποιηθεί η διαδικασία προσομοίωσης που παρουσιάζεται στην Εικόνα 2.9(β). Για πιο σύνθετα κυκλώματα οι σωστές αποκρίσεις παραγωγής μπορούν να αποθηκευτούν στη μνήμη, όπως φαίνεται στην Εικόνα 2.9(γ). Στην Εικόνα 2.9(β) το σύνολο των διανυσμάτων δοκιμής είναι συνήθως εξαντλητικό, καθώς εξετάζει το κύκλωμα εφαρμόζοντας όλες τις πιθανές εισόδους. Ωστόσο, καθώς το μέγεθος των κυκλωμάτων και κατά συνέπεια ο αριθμός των ακροδεκτών εισόδου αυξάνεται, αυτό γίνεται ολοένα και λιγότερο πρακτικό.

Επομένως, ο στόχος είναι να καθοριστεί μια μη *εξαντλητική* (*nonexhaustive*) δοκιμή για το κύκλωμα, όπου θα το εξετάσει επαρκώς μέσα σε ένα αποδεκτό χρόνο και κόστος. Αυτό το μειωμένο σύνολο δοκιμής μπορεί να βασιστεί σε ένα μοντέλο ελαττώματος. Ωστόσο, πρέπει να υπολογίσουμε πόσο αποτελεσματικό είναι αυτό το μη εξαντλητικό σύνολο δοκιμής κατά την ανίχνευση των πιθανών ελαττωμάτων στο κύκλωμα υπό δοκιμή. Αυτό απαιτεί μια διαδικασία που ονομάζεται *προσομοίωση ελαττωμάτων* (*fault simulation*) και η οποία μπορεί να εκτελεστεί παράλληλα με την διαδικασία παραγωγής διανυσμάτων δοκιμής [62], [63], [64], [65], [66], [67].



Εικόνα 2.9 Προσομοίωση δοκιμής: (α) η βασική διαδικασία προσομοίωσης, (β) η απόκριση συγκρίνεται για κάθε διάνυσμα δοκιμής με την απόκριση ενός κυκλώματος, γνωστό ως χρυσό κύκλωμα (golden circuit), (γ) η απόκριση για κάθε διάνυσμα δοκιμής συγκρίνεται με τις σωστές αποκρίσεις οι οποίες έχουν αποθηκευτεί χωρίς ελαττώματα στη μνήμη ενός υπολογιστή.

Οι πρώτες *μέθοδοι προσομοίωσης ελαττωμάτων* (*fault simulation methods*) θεωρούσαν σαν μοντέλο ελαττωμάτων το μοντέλο ελαττώματος προσκόλλησης. Εάν υπάρχουν f πιθανά ελαττώματα προσκόλλησης που πρέπει να εξεταστούν, τότε f μοντέλα του κυκλώματος υπό δοκιμή έπρεπε να παραχθούν, καθένα από τα οποία περιείχε ένα και μόνο ελάττωμα. Το προτεινόμενο σύνολο δοκιμής εφαρμόζεται στη συνέχεια σε κάθε μοντέλο κυκλώματος και ένας αριθμός t καθορίζει των αριθμό των ελαττωματικών κυκλωμάτων που δεν ανιχνεύθηκαν από το προτεινόμενο σύνολο δοκιμής. Από αυτήν την διαδικασία προκύπτει το ποσοστό κάλυψης ελαττωμάτων του προτεινόμενου συνόλου δοκιμής, το οποίο ισούται με t/f , όπως είδαμε στην ενότητα 2.1. Εάν η τιμή του ποσοστού κάλυψης ελαττωμάτων είναι αρκετά χαμηλή, πρόσθετα

διανύσματα δοκιμής παράγονται για να καλύψουν τα ελαττώματα που δεν ανιχνεύτηκαν από το σύνολο των διανυσμάτων δοκιμής.

Αυτή η διαδικασία προσομοίωσης ελαττωμάτων έγινε γρήγορα ανεπαρκής καθώς το μέγεθος των κυκλωμάτων αυξήθηκε και εκτοπίστηκε από τις τρείς ακόλουθες εναλλακτικές τεχνικές:

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

Όλες αυτές οι τεχνικές διαφέρουν από τις πρώτες μεθόδους προσομοίωσης ελαττωμάτων, καθώς μπορούν να εξομοιώσουν παράλληλα πολλαπλά ελαττώματα, σε αντίθεση με την προσομοίωση ενός ελαττώματος τη φορά, επιταχύνοντας κατά συνέπεια την διαδικασία της προσομοίωσης.

Στην *παράλληλη προσομοίωση ελαττωμάτων* (*parallel fault simulation*), μια λέξη *n*-δυαδικών ψηφίων στο πρόγραμμα προσομοίωσης (όπου n = 8, 16 ή 32 δυαδικά ψηφία) χρησιμοποιείται για να καθορίσει τη λογική τιμή σε έναν κόμβο, όταν ο κόμβος είναι χωρίς ελαττώματα και όταν n-1 επιλεγμένα ελαττώματα υπάρχουν στο κύκλωμα. Σύμφωνα με αυτή την προσέγγιση, επιτρέπεται η ταυτόχρονη προσομοίωση κάθε διανύσματος δοκιμής στα n αντίγραφα του κυκλώματος, επιταχύνοντας κατά συνέπεια την διαδικασία κατά έναν παράγοντα περίπου n.

Η *συμπερασματική προσομοίωση ελαττωμάτων* (*deductive fault simulation*), βασίζεται σε ένα *κατάλογο ελαττωμάτων* (*fault list*), ο οποίος και αποτελεί την σχέση μεταξύ της εισόδου και της εξόδου των λογικών πυλών. Ένας κατάλογος ελαττωμάτων συσχετίζεται με κάθε γραμμή σήματος μέσα στο κύκλωμα. Για κάθε διάνυσμα δοκιμής, οι κατάλογοι ελαττωμάτων διαδίδονται σειριακά μέσω του κυκλώματος στις κύριες εξόδους. Ο χρόνος που απαιτείται για ένα πέρασμα μέσω του προσομοιωτή είναι πολύ μεγαλύτερος από το χρόνο που απαιτείται για ένα πέρασμα μέσω ενός παράλληλου προσομοιωτή. Η διαφορά τους, είναι ότι κατά την συμπερασματική προσομοίωση ελαττωμάτων σε κάθε πέρασμα θα καλυφτεί μεγαλύτερος αριθμός ελαττωμάτων.

Η *ταυτόχρονη προσομοίωση ελαττωμάτων* (*concurrent fault simulation*) είναι η συνιστώμενη παρούσα μέθοδος προσομοίωσης ελαττωμάτων. Αποφεύγει την πολυπλοκότητα της εφαρμογής της συμπερασματικής προσομοίωσης ελαττωμάτων, όμως διατηρεί το πλεονέκτημα ταχύτητας έναντι της σειριακής και της παράλληλης προσομοίωσης ελαττωμάτων. Για κάθε γραμμή του κυκλώματος συντάσσεται ένας περιεκτικός κατάλογος ελαττωμάτων για κάθε διάνυσμα δοκιμής, ο οποίος περιλαμβάνει ένα επιλεγμένο ελάττωμα συν όλα τα προηγούμενα ελαττώματα που διαδίδονται μέσω αυτής της γραμμής. Εάν ένα ελάττωμα παράγει

μια απόκριση που είναι ίδια με την σωστή απόκριση σε αυτήν την γραμμή κατά την εφαρμογή του διανύσματος δοκιμής, τότε το πρώτο διαγράφεται από τον ταυτόχρονο κατάλογο ελαττωμάτων. Περαιτέρω λεπτομέρειες για τις διαδικασίες προσομοίωσης ελαττωμάτων μπορούν να βρεθούν στην βιβλιογραφία [16], [64], [65], [68], [69], [70], [71].

2.6 Παραγωγή διανυσμάτων δοκιμής

Η *παραγωγή διανυσμάτων δοκιμής* (*test pattern generation*) είναι μια διαδικασία κατά την οποία παράγονται τα κατάλληλα διανύσματα εισόδου για να δοκιμαστεί ένα δεδομένο ψηφιακό κύκλωμα. Η παραγωγή ενός αποδεκτού μη εξαντλητικού συνόλου διανυσμάτων δοκιμής μπορεί να γίνει με τους ακόλουθους τρόπους:

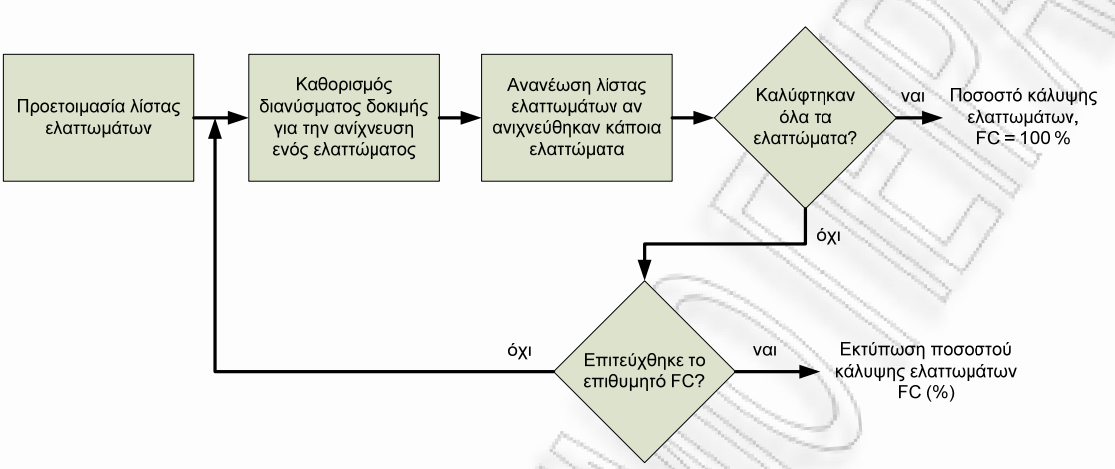
- *Χειροκίνητη παραγωγή* (*manual generation*)
- *Αλγοριθμική παραγωγή* (*algorithmic generation*)

Η *χειροκίνητη παραγωγή διανυσμάτων δοκιμής* (*manual generation*) είναι μια μέθοδος που μπορεί να υιοθετηθεί από τον αρχικό σχεδιαστή του κυκλώματος, γνωρίζοντας τις λεπτομέρειες σχεδίασης και λειτουργίας του κυκλώματος. Τα διανύσματα δοκιμής μπορούν να καθοριστούν βάση των λειτουργικών καταστάσεων του κυκλώματος, την απαρίθμηση των διανυσμάτων δοκιμής και των σωστών αποκρίσεων *responses*) που λαμβάνονται από τις καταστάσεις αυτές. Εναλλακτικά, πρέπει να εφαρμοστούν όλα τα διανύσματα δοκιμής που θα αναγκάσουν όλες τις πύλες του κυκλώματος να αλλάξουν τιμή τουλάχιστον μία φορά. Αυτή η διαδικασία μπορεί να πραγματοποιηθεί μόνο για κυκλώματα μικρού μεγέθους.

Η *παραγωγή διανυσμάτων αυτόματης* (*ή αλγοριθμικής*) *δοκιμής* (*Automatic Test Pattern Generation, ATPG*), κρίνεται επιβεβλημένη καθώς ο αριθμός των πυλών σε ένα ολοκληρωμένο κύκλωμα αυξάνεται ολοένα και περισσότερο στις μέρες μας. Τα προγράμματα αυτόματης παραγωγής διανυσμάτων δοκιμής χρησιμοποιούν την αναπαράσταση του κυκλώματος σε επίπεδο πύλης. Κατά την διαδικασία της παραγωγής διανυσμάτων δοκιμής, εξάγεται ένα νέο διάνυσμα για ένα συγκεκριμένο ελάττωμα στο κύκλωμα. Αυτή η διαδικασία επαναλαμβάνεται για όλο το επιλεγμένο σύνολο ελαττωμάτων στο κύκλωμα όπως αυτό απεικονίζεται στην Εικόνα 2.10.

Τα πρώτα διανύσματα δοκιμής καλύπτουν ένα μεγάλο αριθμό ελαττωμάτων στο κύκλωμα. Κατά συνέπεια, η αρχική κάλυψη ελαττωμάτων μπορεί να είναι ιδιαίτερα γρήγορη. Ωστόσο, για τα πιο σύνθετα κυκλώματα, ο χρόνος επεξεργασίας με απώτερο σκοπό να βρεθούν τα διανύσματα δοκιμής και για τα υπόλοιπα ελαττώματα, μπορεί να είναι ιδιαίτερα μεγάλος. Επίσης, εάν υπάρχει πλεονασμός σε ένα κύκλωμα, ο αλγόριθμος παραγωγής διανυσμάτων δοκιμής δεν θα μπορέσει να εξάγει κατάλληλα διανύσματα δοκιμής για ορισμένους κόμβους του

κυκλώματος. Επομένως, οι οικονομικοί περιορισμοί χρόνου/απόδοσης μπορούν να υπαγορεύσουν τη λήξη ενός προγράμματος αυτόματης παραγωγής διανυσμάτων είτε όταν το ποσοστό κάλυψης ελαττωμάτων φθάσει σε ένα αποδεκτό επίπεδο, για παράδειγμα 99.5%, είτε εάν ο αλγόριθμος έχει εκτελεστεί για έναν προκαθορισμένο χρόνο.



Εικόνα 2.10 Διαδικασία παραγωγής διανυσμάτων δοκιμής

Όλα τα προγράμματα αυτόματης παραγωγής διανυσμάτων δοκιμής που βασίζονται σε μοντέλα ελαττωμάτων υποθέτουν ότι ένα μόνο ελάττωμα υπάρχει στο κύκλωμα κάθε φορά. Κατά την παραγωγή διανυσμάτων δοκιμής, απαιτείται η διάδοση του ελαττώματος ενός δεδομένου κόμβου στις κύριες εξόδους του κυκλώματος, έτσι ώστε η τιμή της εξόδου όταν υπάρχει το ελάττωμα να είναι διαφορετική από την τιμή της εξόδου χωρίς το ελάττωμα όταν εφαρμοστεί το ίδιο διάνυσμα δοκιμής. Αυτή η διαδικασία ονομάζεται *ευαισθητοποίηση διαδρομής* (*sensitizing path*). Μια δεύτερη απαίτηση είναι ότι το διάνυσμα δοκιμής πρέπει να *διεγείρει το ελάττωμα* (*fault excitation*). Για παράδειγμα, στην περίπτωση του ελαττώματος προσκόλλησης, το διάνυσμα δοκιμής πρέπει να θέσει τον κόμβο υπό δοκιμή σε μια τιμή η οποία να είναι αντίθετη από την τιμή προσκόλλησης. Δηλαδή για να δοκιμαστεί ένα ελάττωμα *SA0* στον κόμβο *x*, το διάνυσμα δοκιμής θα πρέπει να θέσει τον κόμβο *x* στην τιμή "1".

Οι περισσότεροι αλγόριθμοι παραγωγής διανυσμάτων δοκιμής ακολουθούν τα παρακάτω βήματα:

1. Επιλογή ενός *ελαττωματικού κόμβου* (*faulty node*) στο κύκλωμα.
2. *Διέγερση του ελαττώματος* (*fault excitation*).
3. *Διάδοση* (*propagate*) της τιμής του κόμβου στις κύριες εξόδους.

4. Επιστροφή στις κύριες εισόδους προκειμένου να καθοριστεί η λογική τιμή των εισόδων που *διαδίδουν* (*propagate*) την τιμή του ελαττώματος στις κύριες εξόδους.

2.7 Λειτουργική δοκιμή

Τα μοντέλα ελαττωμάτων που μελετήσαμε στα προηγούμενα κεφάλαια βασίζονται στην *δομική αναπαράσταση* (*structural model*) του κυκλώματος υπό δοκιμή. Τι συμβαίνει όμως όταν η δομική αναπαράσταση των κυκλωμάτων δεν είναι γνωστή; Σε αυτό το σημείο, εξετάζουμε την *λειτουργική δοκιμή* (*functional testing*), η οποία βασίζεται στο λειτουργικό μοντέλο ενός συστήματος.

Ένα λειτουργικό μοντέλο αναπαριστά τις λειτουργικές προδιαγραφές ενός συστήματος και είναι εντελώς ανεξάρτητο από την υλοποίηση τού. Συνεπώς, η λειτουργική δοκιμή μπορεί να χρησιμοποιηθεί όχι μόνο για την ανίχνευση των φυσικών ελαττωμάτων τα οποία μπορεί να συμβούν σε ένα σύστημα, αλλά και για τις *δοκιμές επαλήθευσης της σχεδίασης* (*design verification tests*) για την ανίχνευση λαθών σχεδίασης.

Ας εξετάσουμε την δοκιμή μιας πύλης AND δέκα εισόδων. Υποθέστε ότι εφαρμόζουμε το διάνυσμα εισόδου 0101010101 και παρατηρούμε ότι η έξοδος έχει τιμή "0". Αν αυτή είναι η σωστή τιμή εξόδου, τι συμπεράσματα μπορούμε να βγάλουμε από αυτή και μόνο την τιμή; Η πύλη υπό δοκιμή: (α) είναι AND, (β) δεν είναι NAND, (γ) είναι NOR, (δ) δεν είναι OR. Δεδομένου ότι η συγκεκριμένη έξοδος παραβιάζει τους πίνακες αλήθειας των πυλών NAND και OR, μόνο οι απαντήσεις (β) και (δ) είναι σωστές. Θα μπορούσαμε να χρησιμοποιήσουμε άλλο ένα διάνυσμα δοκιμής, 1111111111, για να σιγουρευτούμε ότι η πύλη δεν είναι NOR. Αυτό όμως δεν εγγυάται ότι το δεδομένο κύκλωμα θα λειτουργήσει σωστά ως πύλη AND για όλα τα πιθανά διανύσματα δοκιμής. Μια πλήρης λειτουργική δοκιμή θα ελέγξει κάθε είσοδο του πίνακα αλήθειας. Ωστόσο, μια τέτοια δοκιμή θα ήταν ιδιαίτερα χρονοβόρα εάν εφαρμοζόταν σε ένα πραγματικό κύκλωμα με αρκετές γραμμές εισόδου.

Η λειτουργική δοκιμή μπορεί να πραγματοποιηθεί σύμφωνα με τρεις διαφορετικές προσεγγίσεις. Η πρώτη προσέγγιση υιοθετεί ένα συγκεκριμένο *λειτουργικό μοντέλο ελαττωμάτων* (*functional fault model*), ενώ η δεύτερη δεν λαμβάνει υπόψη τις πιθανές περιπτώσεις λανθασμένης συμπεριφοράς του κυκλώματος και εφαρμόζει διανύσματα δοκιμής βασισμένη μόνο στην συμπεριφορά του κυκλώματος χωρίς ελαττώματα. Για παράδειγμα, μια λειτουργική δοκιμή για ένα φλιπ-φλοπ περιλαμβάνει τους εξής ελέγχους: αν μπορεί να γίνει αρχικοποίηση των φλιπ-φλοπ (*set* ή *reset*) και αν το φλιπ-φλοπ μπορεί να διατηρήσει την κατάστασή του. Μεταξύ αυτών των δύο προσεγγίσεων, υπάρχει και μία τρίτη η οποία δεν καθορίζει αλλά *υπονοεί ένα μοντέλο ελαττωμάτων* (*implicit fault model*), το οποίο υποθέτει ότι μπορεί να υπάρξει οποιοδήποτε ελάττωμα στο κύκλωμα. Χρησιμοποιώντας την γνώση για την

δομή ενός κυκλώματος και περιορίζοντας το σύνολο των ελαττωμάτων σε ένα υποσύνολο τα οποία εγγυημένα μπορούν να ανιχνευτούν, οδηγούμαστε στην *ψευδό-εξαντλητική δοκιμή* (*pseudo-exhaustive test*) η οποία είναι σημαντικά μικρότερη από την εξαντλητική δοκιμή. Συνοψίζοντας, η λειτουργική δοκιμή μπορεί να πραγματοποιηθεί με τρεις διαφορετικές προσεγγίσεις [16]:

1. *Λειτουργική δοκιμή χωρίς μοντέλα ελαττωμάτων* (*Functional testing without fault model*).

2. *Λειτουργική δοκιμή με συγκεκριμένα μοντέλα ελαττωμάτων* (*Functional testing using specific fault models*).

3. *Εξαντλητική και ψευδο-εξαντλητική λειτουργική δοκιμή* (*Exhaustive and pseudoexhaustive functional testing*).

2.8 Τεχνικές σχεδίασης για αυξημένη δοκιμαστικότητα

Διάφορες μελέτες έδειξαν ότι είναι δυνατόν να δημιουργηθεί μια σχέση κόστους, η οποία να σχετίζεται με την διαδικασία της δοκιμής. Υπάρχουν πολλές παράμετροι που επηρεάζουν το κόστος, της δοκιμής:

- Το κόστος παραγωγής των διανυσμάτων δοκιμής,
- Το κόστος προσομοίωσης των ελαττωμάτων,
- Το κόστος του εξοπλισμού δοκιμής,
- Το κόστος για την χρήση του εξωτερικού ελεγκτή, που αυξάνει με τον χρόνο της δοκιμής.

Για να κρατηθεί το κόστος σε λογικά επίπεδα εφαρμόζονται τεχνικές σχεδίασης που διευκολύνουν την δοκιμή του κυκλώματος και ονομάζονται, *τεχνικές σχεδίασης για αυξημένη δοκιμαστικότητα* (*design for testability techniques*), οι οποίες μειώνουν το κόστος από τις παραπάνω παραμέτρους.

Ο όρος *δοκιμαστικότητα* (*testability*) είναι ένα χαρακτηριστικό γνώρισμα της σχεδίασης που επηρεάζει το κόστος της δοκιμής. Συνήθως, σε ένα κύκλωμα με υψηλή δοκιμαστικότητα μπορούμε να καθορίσουμε με σχετική ευκολία την κατάσταση του κυκλώματος και να ανιχνεύσουμε τα πιθανά ελαττώματα. Οι δύο παράγοντες που επηρεάζουν την δοκιμαστικότητα,

είναι η *ελεγξιμότητα* (*controllability*) και η *παρατηρησιμότητα* (*observability*), όπως τους περιγράψαμε στην ενότητα 2.3.

Οι περισσότερες τεχνικές σχεδίασης για αυξημένη δοκιμαστικότητα είτε επανασυνθέτουν την αρχική σχεδίαση, είτε προσθέτουν επιπλέον υλικό. Οι περισσότερες τεχνικές απαιτούν αλλαγές στο κύκλωμα και επηρεάζουν την επιφάνεια του κυκλώματος, τον αριθμό των ακροδεκτών εισόδου/εξόδου και την καθυστέρηση του κυκλώματος. Γενικά, πρέπει να βρεθεί η χρυσή τομή μεταξύ του επιπλέον υλικού που θα προσθέσουμε για να αυξήσουμε την δοκιμαστικότητα και του κέρδους που θα έχουμε εφαρμόζοντας αυτή την τεχνική, ώστε τελικά να πετύχουμε μείωση του κόστους, αύξηση της ποιότητας της δοκιμής και μείωση του επιπέδου ατέλειας. Επίσης, οι τεχνικές αυτές επηρεάζουν τον αριθμό των διανυσμάτων δοκιμής, το απαιτούμενο ποσό μνήμης στον εξωτερικό ελεγκτή και τον χρόνο εφαρμογής της δοκιμής.

Για παράδειγμα, αύξηση στην επιφάνεια του κυκλώματος και στην πολυπλοκότητα της λογικής του, έχει σαν αποτέλεσμα την αύξηση της κατανάλωσης ενέργειας και την μείωση της εσοδείας. Όμως, μείωση της εσοδείας σημαίνει αύξηση στον αριθμό των παραγόμενων ελαττωματικών ολοκληρωμένων κυκλωμάτων. Στην συνέχεια παρουσιάζονται εν' συντομία οι κυριότερες τεχνικές σχεδίασης για αυξημένη δοκιμαστικότητα [16]:

- *Εισαγωγή σημείων δοκιμής* (*test points insertion*): Εισάγουμε σημεία δοκιμής για να αυξήσουμε την ελεγξιμότητα και την παρατηρησιμότητα του κυκλώματος. Υπάρχουν δύο τύποι σημείων δοκιμής: τα *σημεία ελέγχου* (*control points, CP*) και τα *σημεία παρατήρησης* (*observation points, OP*). Τα πρώτα είναι κύριες είσοδοι, ενώ τα δεύτερα είναι κύριες έξοδοι του κυκλώματος.

- *Αρχικοποίηση* (*initialization*): Σχεδιάζουμε το κύκλωμα ώστε να μπορεί εύκολα να αρχικοποιηθεί. Η αρχικοποίηση είναι μια διαδικασία που εφαρμόζεται στα ακολουθιακά κυκλώματα, συνήθως κατά την εκκίνηση τους, ώστε να τεθούν σε μια γνωστή κατάσταση σε συγκεκριμένο αριθμό κύκλων.

- *Μονοσταθείς πολυταλαντωτές* (*monostable multivibrators ή one-shots*)*:* Απενεργοποιούμε τους εσωτερικούς μονοσταθείς πολυταλαντωτές κατά την διάρκεια της δοκιμής. Οι μονοσταθείς πολυταλαντωτές δημιουργούν παλμούς εσωτερικά στο κύκλωμα, γεγονός που οδηγεί τους εξωτερικούς ελεγκτές να χάσουν τον συγχρονισμό τους με το κύκλωμα, κάνοντας την εξωτερική δοκιμή πολύ δύσκολη.

- *Ταλαντωτές και ρολόγια* (*oscillators, clocks*): Απενεργοποιούμε τους εσωτερικούς ταλαντωντές και τα ρολόγια, για τους ίδιους λόγους που απενεργοποιούμε και τους ταλαντωτές.

- *Μετρητές και καταχωρητές ολίσθησης* (*counters, shift registers*): Χωρίζουμε τους μεγάλους μετρητές και τους καταχωρητές ολίσθησης σε μικρότερες υπομονάδες. Οι μετρητές και σε μικρότερο βαθμό οι καταχωρητές ολίσθησης είναι μονάδες δύσκολες στην δοκιμή καθώς οι ακολουθίες δοκιμής απαιτούν πολλούς κύκλους ρολογιού.

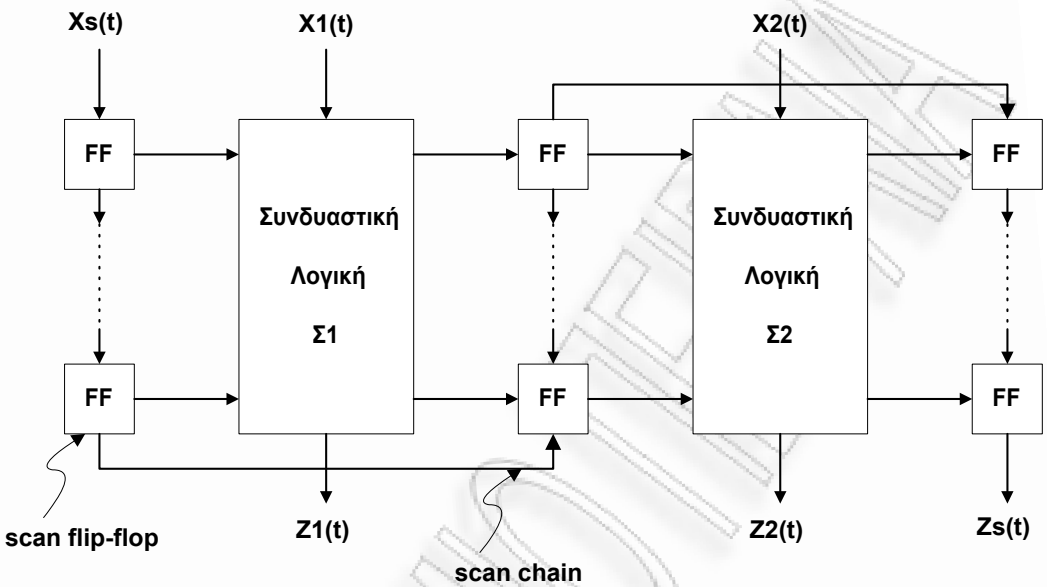
- *Διαμέριση μεγάλων συνδυαστικών μονάδων* (*partitioning of large combinational circuits*): Χωρίζουμε τα μεγάλα συνδυαστικά κυκλώματα σε μικρότερα υποκυκλώματα ώστε να μειωθεί το κόστος παραγωγής του συνόλου δοκιμής.

- *Λογικός πλεονασμός* (*logical redundancy*): Αποφεύγουμε την χρήση *πλεονάζουσας λογικής* (*redundant logic*). Αν και μερικές φορές, προσθέτουμε πλεονάζουσα λογική σε ένα κύκλωμα, αυτό πρέπει γενικότερα να αποφεύγεται διότι: (α) ένα ελάττωμα στην πλεονάζουσα λογική μπορεί να ακυρώσει την δοκιμή για άλλα ελαττώματα, (β) δημιουργείται πρόβλημα στον υπολογισμό του ποσοστού κάλυψης των ελαττωμάτων και (γ) αυξάνεται ο χρόνος για την παραγωγή του συνόλου της δοκιμής.

- *Μονοπάτια ανάδρασης* (*feedback paths*): Εισάγουμε λογική ώστε να "διακόψουμε" τα μονοπάτια ανάδρασης.

Με την χρήση σημείων δοκιμής αυξάνεται η ελεγξιμότητα και η παρατηρησιμότητα ενός κυκλώματος. Αυτό όμως έχει επιπτώσεις στον αριθμό των ακροδεκτών εισόδου - εξόδου του κυκλώματος. Ένας άλλος τρόπος για να αυξήσουμε την ελεγξιμότητα και την παρατηρησιμότητα είναι η συστηματική σχεδίαση για αυξημένη δοκιμαστικότητα, η οποία συνήθως ενσωματώνεται στα *αυτόματα εργαλεία σχεδίασης* (*Computer-Aided Design, CAD tools*) και ονομάζεται *σύνθεση δοκιμής* (*test synthesis*). Η βασική ιδέα των περισσότερων τεχνικών σύνθεσης δοκιμής είναι η *σχεδίαση σάρωσης* (*scan design*) [16] η οποία βασίζεται στην χρήση ενός ειδικού καταχωρητή που ονομάζεται *καταχωρητής σάρωσης* (*Scan Register, SR*). Ο καταχωρητής αυτός έχει δυνατότητα τόσο παράλληλης όσο και σειριακής φόρτωσης και τα κελιά του χρησιμοποιούνται σαν σημεία παρατήρησης, έλεγχου ή και τα δύο.

Σχεδίαση σάρωσης είναι ο διαχωρισμός των ακολουθιακών μονάδων του κυκλώματος από τις συνδυαστικές μονάδες κατά την διάρκεια της δοκιμής. Ακολουθιακά στοιχεία, όπως φλιπ-φλοπ, ενώνονται μεταξύ τους για να σχηματίσουν μία αλυσίδα υπό την μορφή *καταχωρητή ολίσθησης* (*shift register*), όταν ενεργοποιηθεί η διαδικασία του ελέγχου. Η αλυσίδα αυτή ονομάζεται *αλυσίδα σάρωσης* (*scan chain*). Όπως φαίνεται και στην Εικόνα 2.11, αυτή η τεχνική διαχωρίζει το κύκλωμα σε ένα σύνολο υπομονάδων, οι είσοδοι και οι έξοδοι των οποίων ενώνονται απευθείας στην αλυσίδα σάρωσης.



Εικόνα 2.11 Τεχνική σχεδίασης σάρωσης

Κάθε συνδυαστικό τμήμα μπορεί να ελεγχθεί ολισθαίνοντας σειριακά τα δεδομένα δοκιμής από την είσοδο της αλυσίδας *Xs*. Τα δεδομένα εφαρμόζονται στην συνδυαστική λογική διαδοχικά και οι έξοδοι φορτώνονται παράλληλα σε άλλο τμήμα της αλυσίδας. Τελικά, τα δεδομένα εξόδου ολισθαίνουν σειριακά προς την έξοδο της αλυσίδας *Zs*. Τα διανύσματα δοκιμής για κάθε συνδυαστικό τμήμα, είναι δυνατόν να παραχθούν από οποιοδήποτε εργαλείο παραγωγής διανυσμάτων. Συνδέοντας όλα τα ακολουθιακά στοιχεία ενός κυκλώματος στην αλυσίδα σάρωσης, το οποίο ονομάζεται *πλήρης σάρωση* (*full scan*), αποφεύγεται η ανάγκη για παραγωγή διανυσμάτων για την ακολουθιακή λογική.

Υπάρχουν διαφορετικές παραλλαγές της τεχνικής σάρωσης, όλες όμως απαιτούν την τήρηση ορισμένων κανόνων σχεδίασης. Σήμερα, τα ολοκληρωμένα κυκλώματα αναλύονται για την ανίχνευση παραβίασης των κανόνων σχεδίασης, οι αλυσίδες σάρωσης εισάγονται και τα σύνολα δοκιμής παράγονται χωρίς την παρέμβαση των μηχανικών, όλα από αυτόματα εργαλεία λογισμικού [72].

Οι τεχνικές σάρωσης μπορούν επίσης να χρησιμοποιηθούν και για άλλους σκοπούς εκτός από την αποφυγή της παραγωγής διανυσμάτων δοκιμής για τις ακολουθιακές μονάδες. Για παράδειγμα, φλιπ-φλοπ σάρωσης είναι δυνατόν να προστεθούν στους ακροδέκτες του ολοκληρωμένου ώστε να παρέχουν πρόσβαση στις διασυνδέσεις μεταξύ των ολοκληρωμένων, ή να παρέχουν πρόσβαση σε εσωτερικούς κόμβους του κυκλώματος κατά την διάρκεια της δοκιμής.

Η σχεδίαση σάρωσης λύνει αρκετά προβλήματα, αλλά παρουσιάζει δύο πολύ σημαντικά μειονεκτήματα. Πρώτον, οι κανόνες σχεδίασης σάρωσης συχνά αποκλείουν τεχνικές σχεδίασης, οι οποίες γενικώς είναι πολύ χρήσιμες. Για παράδειγμα, ενώ οι "υποστηρικτές" της σχεδίασης σάρωσης επικρίνουν το φιλτράρισμα των ρολογιών ως κακή πρακτική σχεδίασης, αυτή η τεχνική συχνά προσφέρεται για κυκλώματα υψηλών ταχυτήτων. Από την άλλη, οι αλυσίδες σάρωσης δημιουργούν μακριά μονοπάτια τα οποία είναι δυνατόν να επηρεάσουν αρνητικά την ταχύτητα του κυκλώματος [73]. Επίσης, σε κυκλώματα με μεγάλο αριθμό ακολουθιακών στοιχείων, ο χρόνος ολίσθησης των δεδομένων δοκιμής από την είσοδο της αλυσίδας (δεδομένα εισόδου) προς την έξοδο της αλυσίδας (δεδομένα εξόδου) είναι απαγορευτικά μεγάλος και απαιτεί εξωτερικούς ελεγκτές με πολύ μεγάλο μέγεθος μνήμης. Τέλος, οι σχεδιάσεις σάρωσης δεν είναι δυνατόν να εφαρμοστούν σε όλες τις κατηγορίες κυκλωμάτων, ειδικά στις μονάδες υψηλής πυκνότητας, όπως οι *μνήμες τυχαίας προσπέλασης* (*Random Access Memories, RAMs*).

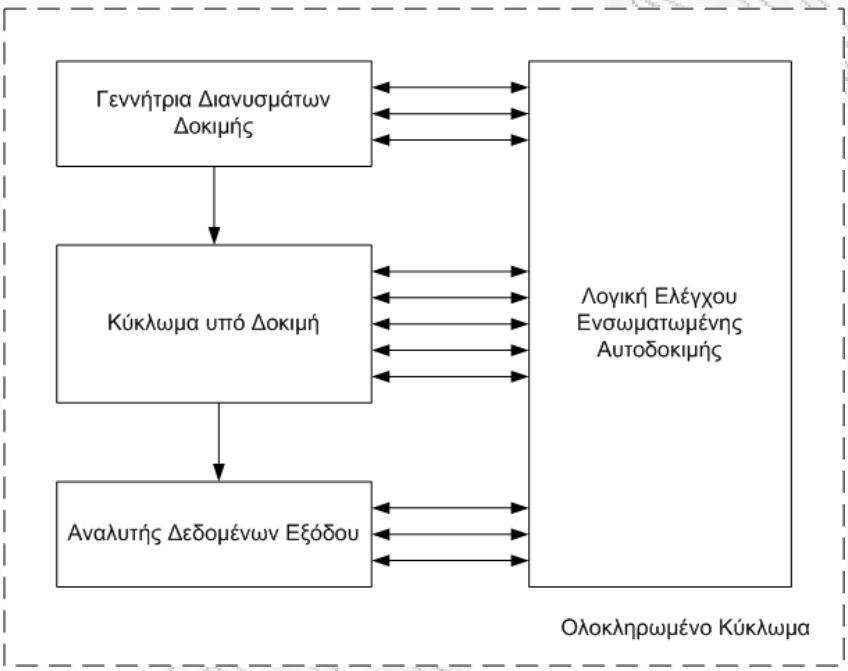
2.9 Τεχνικές ενσωματωμένης αυτοδοκιμής

Η *ενσωματωμένη αυτοδοκιμή* (*Built-In Self-Test, BIST*) αναφέρεται σε τεχνικές οι οποίες επιτρέπουν σε ένα ολοκληρωμένο κύκλωμα να πραγματοποιήσει δοκιμή στον εαυτό του [16], [17], [18]. Σύμφωνα με την μεθοδολογία της ενσωματωμένης αυτοδοκιμής, τα διανύσματα δοκιμής παράγονται και οι αποκρίσεις του κυκλώματος λαμβάνονται και αξιολογούνται μέσα στο ίδιο το ολοκληρωμένο κύκλωμα. Μία τυπική δομή ενσωματωμένης αυτοδοκιμής περιλαμβάνει την εισαγωγή τριών βασικών μονάδων στο κύκλωμα, όπως παρουσιάζονται στην Εικόνα 2.12: την *γεννήτρια διανυσμάτων δοκιμής* (*Test Pattern Generator, TPG*), τον *αναλυτή δεδομένων εξόδου* (*Output Data Analyzer, ODA*) και την *λογική ελέγχου της ενσωματωμένης αυτοδοκιμής* (*BIST controller*).

Οι τεχνικές ενσωματωμένης αυτοδοκιμής χωρίζονται σε δύο βασικές κατηγορίες, την *ενσωματωμένη αυτοδοκιμή κατά τη λειτουργία* (*on-line BIST*) η οποία περιλαμβάνει *ταυτόχρονες* (*concurrent*) και *ετερόχρονες* (*non- concurrent*) τεχνικές και την *ενσωματωμένη αυτοδοκιμή εκτός λειτουργίας* (*off-line BIST*) η οποία περιλαμβάνει *λειτουργικές* (*functional*) και *δομικές* (*structural*) προσεγγίσεις.

Σύμφωνα με την ενσωματωμένη δοκιμή κατά τη λειτουργία, το κύκλωμα υπό δοκιμή δεν χρειάζεται να εισέλθει σε κάποια κατάσταση δοκιμής, στην οποία η πραγματική του λειτουργία παύει να υφίσταται. Η ταυτόχρονη ενσωματωμένη αυτοδοκιμή κατά τη λειτουργία πραγματοποιείται παράλληλα με την κανονική λειτουργία του κυκλώματος. Η δοκιμή συχνά πραγματοποιείται χρησιμοποιώντας *τεχνικές κωδικοποίησης* (*coding techniques*), ή *διπλασιασμού* (*duplication*) και *σύγκρισης* (*comparison*). Αντίθετα, η ετερόχρονη ενσωματωμένη αυτοδοκιμή κατά τη λειτουργία, εκτελείτε μόνο όταν το κύκλωμα υπό δοκιμή βρίσκεται σε

κατάσταση *αδράνειας* (*idle*). Η δοκιμή αυτή, συχνά πραγματοποιείται με την χρήση *διαγνωστικών ρουτινών ενσωματωμένου υλικολογισμικού* (*firmware*).



Εικόνα 2.12 : Τυπική δομή ενσωματωμένης αυτοδοκιμής

Η ενσωματωμένη αυτοδοκιμή εκτός λειτουργίας εκτελείτε μόνο όταν το κύκλωμα δεν βρίσκεται σε κατάσταση λειτουργίας. Συστήματα, πλακέτες και ολοκληρωμένα κυκλώματα είναι δυνατόν να δοκιμαστούν με αυτή την μέθοδο. Συχνά, η ενσωματωμένη αυτοδοκιμή εκτός λειτουργίας πραγματοποιείται χρησιμοποιώντας γεννήτριες διανυσμάτων δοκιμής οι οποίες είναι ενσωματωμένες μέσα στο ολοκληρωμένο κύκλωμα ή στην πλακέτα του συστήματος καθώς επίσης και αναλυτές δεδομένων εξόδου ή διαγνωστικές ρουτίνες λογισμικού. Η ενσωματωμένη αυτοδοκιμή εκτός λειτουργίας δεν εντοπίζει ελαττώματα σε πραγματικό χρόνο, όπως για παράδειγμα πολλές ταυτόχρονες τεχνικές ενσωματωμένης αυτοδοκιμής που ανιχνεύουν τα ελαττώματα όταν αυτά συμβούν.

Η λειτουργική ενσωματωμένη αυτοδοκιμή εκτός λειτουργίας συχνά χρησιμοποιεί ένα λειτουργικό ή υψηλού επιπέδου μοντέλο ελαττωμάτων και αναφέρεται σε δοκιμές οι οποίες βασίζονται στην λειτουργική περιγραφή του κυκλώματος υπό δοκιμή. Οι δοκιμές εφαρμόζονται με την χρησιμοποίηση διαγνωστικού λογισμικού.

Η δομική ενσωματωμένη αυτοδοκιμή εκτός λειτουργίας χρησιμοποιεί ένα μοντέλο ελαττωμάτων και αναφέρεται σε δοκιμές οι οποίες βασίζονται στο δομικό μοντέλο του

κυκλώματος υπό δοκιμή. Το ποσοστό κάλυψης των ελαττωμάτων βασίζεται στην ανίχνευση δομικών ελαττωμάτων του κυκλώματος. Συχνά, οι δοκιμές εφαρμόζονται και οι αποκρίσεις συμπιέζονται με την χρήση *καταχωρητών ολίσθησης γραμμικής ανάδρασης* (*Linear Feedback Shift Registers, LFSRs*).

Η ενσωματωμένη αυτοδοκιμή προσφέρει πολλά πλεονεκτήματα έναντι της δοκιμής με αυτόματο ελεγκτή δοκιμής. Στην ενσωματωμένη αυτοδοκιμή, το κύκλωμα δοκιμής ενσωματώνεται μέσα στο ολοκληρωμένο κύκλωμα αποφεύγοντας την χρήση *εξωτερικού ελεγκτή* (*external tester*), παράγοντας τόσο τα διανύσματα όσο και το ρολόι της δοκιμής. Αυτό είναι ιδιαίτερα χρήσιμο για κυκλώματα υψηλών ταχυτήτων τα οποία σε αντίθετη περίπτωση θα απαιτούσαν εξωτερικούς ελεγκτές πολύ υψηλού κόστους. Επιπλέον μπορεί να εφαρμοστεί στην πραγματική συχνότητα ρολογιού του κυκλώματος υπό δοκιμή, παρέχοντας ιδιαίτερα πλεονεκτήματα ως προς το ποσοστό κάλυψης ελαττωμάτων, τα οποία σε αντίθετη περίπτωση θα ανιχνεύονταν μόνο όταν το κύκλωμα υπό δοκιμή ενσωματωνόταν στο τελικό σύστημα. Τέλος, ένα ολοκληρωμένο κύκλωμα με δυνατότητα αυτοδοκιμής έχει την ικανότητα να ελέγχει τον εαυτό του ακόμα και όταν αυτό έχει ενσωματωθεί σε ένα σύστημα. Αυτό είναι ιδιαίτερα χρήσιμο τόσο για *περιοδική δοκιμή* (*periodic testing*), όσο και *διαγνωστικούς ελέγχους* (*diagnostic tests*) στο πεδίο εφαρμογής.

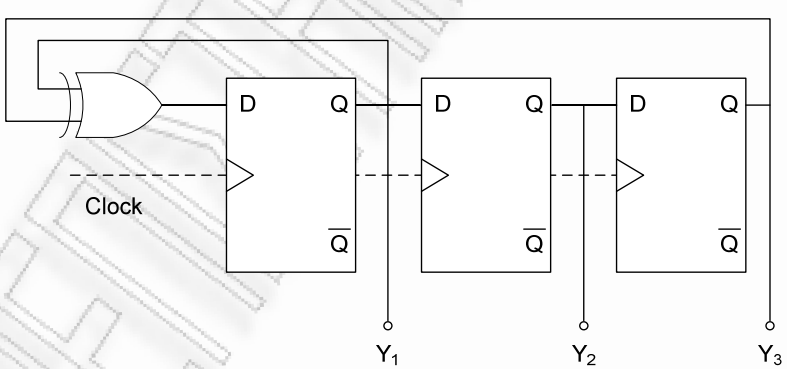
Για την υλοποίηση μίας αρχιτεκτονικής ενσωματωμένης αυτοδοκιμής, είναι δυνατόν είτε να χρησιμοποιηθούν υπάρχουσες μονάδες του κυκλώματος η λειτουργία των οποίων θα επαναπροσδιορίζεται κατά την διάρκεια της δοκιμής, είτε να προστεθούν νέες μονάδες. Και στις δύο περιπτώσεις, η περιοχή που καταλαμβάνει το κύκλωμα αυξάνεται και η απόδοσή του μπορεί να μειωθεί. Επίσης, μία αρχιτεκτονική ενσωματωμένης αυτοδοκιμής μπορεί να εισάγει ανεπιθύμητες καθυστερήσεις στα κρίσιμα μονοπάτια του κυκλώματος και επομένως να επιφέρει μείωση της απόδοσης του. Με άλλα λόγια, η τεχνική ενσωματωμένης αυτοδοκιμής ως μία υποπερίπτωση των τεχνικών σχεδίασης για αυξημένη δοκιμαστικότητα, κληρονομεί τα μειονεκτήματα της τελευταίας.

Υπάρχουν διάφορες τεχνικές για την παραγωγή των διανυσμάτων και την επαλήθευση των αποκρίσεων, με σημαντικότερες τις *γεννήτριες ψευδοτυχαίων διανυσμάτων* (*pseudorandom sequence generators*) και την *ανάλυση υπογραφής* (*signature analysis*), αντίστοιχα.

Οι γεννήτριες ψευδοτυχαίων διανυσμάτων χρησιμοποιούν *καταχωρητές ολίσθησης γραμμικής ανάδρασης* (*Linear Feedback Shift Registers, LFSRs*). Οι καταχωρητές αυτοί συνήθως απαιτούν περισσότερα διανύσματα σε σχέση με ένα σύνολο δοκιμής που έχει προκύψει από μια αυτόματη παραγωγή διανυσμάτων δοκιμής για την επίτευξη του ίδιου ποσοστού κάλυψης ελαττωμάτων, αλλά το κόστος παραγωγής τους σε υλικό είναι πολύ μικρότερο. Κατά την διαδικασία της τυχαίας παραγωγής διανυσμάτων, μπορεί να συμβεί ένα διάνυσμα να επαναληφθεί περισσότερες από μία φορές. Κατά την χρήση μιας ψευδοτυχαίας διαδικασίας παραγωγής, απαιτείται η εξαγωγή διανυσμάτων χωρίς επαναλήψεις. Αυτό μπορεί να υλοποιηθεί

με χρήση κατάλληλου λογισμικού και αποθήκευση των διανυσμάτων δοκιμής σε μια μονάδα μνήμης που βρίσκεται εντός του ολοκληρωμένου. Ο τρόπος αυτός αυξάνει υπερβολικά τις απαιτήσεις σε υλικό καθώς και το κόστος υλοποίησης του ολοκληρωμένου. Για τον λόγο αυτό έχουν αναπτυχθεί δύο κύριες τεχνικές που στοχεύουν στην παραγωγή ψευδοτυχαίων διανυσμάτων με χρήση υλικού χαμηλού κόστους με χρήση υλικού: τα *κυψελωτά αυτόνομα* (*cellular autonoma*) και τα LFSR.

Η σημαντικότερη από αυτές τις τεχνικές, το LFSR, είναι ένας καταχωρητής ολίσθησης ο οποίος υλοποιεί ένα μονοπάτι ανάδρασης από το τελευταίο φλιπ-φλοπ και ενδεχομένως από ενδιάμεσα φλιπ-φλοπ στην είσοδο του πρώτου φλιπ-φλοπ του καταχωρητή. Παράλληλα, εκτός από το σήμα χρονισμού δεν δέχεται καμία άλλη είσοδο, γεγονός που μειώνει σημαντικά το επιπλέον υλικό και τους ακροδέκτες που εισάγονται στην σχεδίαση. Για τον λόγο αυτό είναι γνωστό και σαν *αυτόνομο LFSR*. Το παράδειγμα ενός LFSR των 3-bit απεικονίζεται στην Εικόνα 2.13. Οι έξοδοι από όλα τα φλιπ-φλοπ σχηματίζουν το διάνυσμα δοκιμής που εξάγεται σε κάθε βήμα, ενώ ο συνολικός αριθμός των διανυσμάτων δοκιμής που εξάγονται ισούται με το αριθμό των δυνατών καταστάσεων του, ο οποίος καθορίζεται από τον αριθμό και την τοποθεσία των ξεχωριστών αναδράσεων που περιέχει. Ο καταχωρητής αρχικοποιείται σε μια αρχική τιμή (seed).



Εικόνα 2.13 Αυτόνομος καταχωρητής ολίσθησης γραμμικής ανάδρασης

Το LFSR μπορεί να περιγραφτεί μαθηματικά με την χρήση ενός χαρακτηριστικού *πολυωνύμου* (*polynomial*), ένα πολυώνυμο βαθμού *N, $P_N(x)$* ορίζεται ως:

$$P_N(X) = 1 + \sum_{i=1}^{j=N} C_j X^j$$

όπου το *Cj* ορίζεται ως το βάρος κάθε φλιπ-φλοπ και παίρνει την τιμή "1" αν η έξοδος συμμετέχει στο μονοπάτι ανάδρασης προς την πρώτη είσοδο του LFSR και την τιμή "0" αν δεν συμμετέχει. Θεωρήστε για παράδειγμα το πολυώνυμο $P_3=X^3+X+1$ που περιγράφεται στην Εικόνα 2.13. Τα βάρη του πολυωνύμου είναι: $C_3=1$, $C_2=0$ και $C_1=1$. Επειδή τα πολυώνυμα αυτά

περιγράφουν κυκλώματα που περιέχουν αποκλειστικά πύλες XOR όλες οι πράξεις που γίνονται σε αυτά εκτελούνται με χρήση Modulo 2 αριθμητικής.

Ένας καταχωρητής LFSR είναι δυνατόν να δημιουργηθεί με την χρήση λογισμικού δοκιμής. Το λογισμικό εξομοιώνει την παραγωγή των διανυσμάτων δοκιμής για έναν καταχωρητή LFSR μέσω λογισμικού έναντι υλικού. Υπάρχουν δύο κύριες τεχνικές για την παραγωγή των διανυσμάτων δοκιμής:

- Δημιουργούνται διαφορετικές ρουτίνες ψευδοτυχαίας παραγωγής διανυσμάτων δοκιμής για κάθε μονάδα υπό δοκιμή.

- Δημιουργείται μία κύρια ρουτίνα για την παραγωγή των διανυσμάτων δοκιμής, η οποία καλείται από τις επιμέρους ρουτίνες δοκιμής κάθε μονάδας υπό δοκιμή.

Το πλεονέκτημα την πρώτης προσέγγισης είναι ότι η αλληλεπίδραση της ρουτίνας δοκιμής με την μνήμη είναι ιδιαίτερα μικρή ενώ τα διανύσματα δοκιμής εφαρμόζονται απευθείας στην μονάδα υπό δοκιμή. Το μειονέκτημα αυτής της προσέγγισης είναι ότι αυξάνονται οι απαιτήσεις ως προς το απαιτούμενο μέγεθος μνήμης για την αποθήκευση του κώδικα δοκιμής, λόγω των διαφορετικών ρουτινών παραγωγής των διανυσμάτων για κάθε μονάδα υπό δοκιμή.

Ένα παράδειγμα για την παραγωγή ψευδοτυχαίων διανυσμάτων δοκιμής για την μονάδα αφαίρεσης ενός επεξεργαστή παρουσιάζεται στην Εικόνα 2.14. Υποθέτουμε ότι ο καταχωρητής LFSR εξομοιώνεται μέσω των εντολών του επεξεργαστή. Στο παράδειγμα αυτό οι καταχωρητές `R2` και `R3` είναι τα διανύσματα που θα εφαρμοστούν στον αφαιρέτη του επεξεργαστή σε κάθε επανάληψη. Το αποτέλεσμα της πράξης συλλέγεται στον καταχωρητή `R1` και αποθηκεύεται στην μνήμη μετά την πραγματοποίηση της αφαίρεσης. Ο καταχωρητής `R8` αποτελεί τον μετρητή των ψευδοτυχαίων διανυσμάτων, ενώ η αρχική τιμή του δηλώνει των αριθμό επαναλήψεων του βρόχου. Οι καταχωρητές `R9, R10` περιέχουν την μάσκα η οποία δημιουργεί το χαρακτηριστικό πολυώνυμο του καταχωρητή LFSR. Οι καταχωρητές `R2, R3` αρχικοποιούνται σε μία τιμή (seed). Στο παράδειγμα αυτό υποθέτουμε ότι έχουμε διαφορετικά LFSR για κάθε τελεστή `X, Y` της αφαίρεσης (καταχωρητές `R2` και `R3`). Οι καταχωρητές `R4` και `R5` χρησιμοποιούνται για την δημιουργία του αλγορίθμου του LFSR, δηλαδή του υπολογισμού της επόμενης τιμής των τελεστών. Τέλος ο καταχωρητής `R11` χρησιμοποιείται για την διευθυνσιοδότηση των αποκρίσεων της δοκιμής στην μνήμη του συστήματος.

```
1.   test-substracter:   sub R11, R11, R11
2.                       ori R8, R0, max-patterns
3.                       lw R9, polynomial-mask-x
4.                       lw R10, polynomial-mask-y
5.                       lw R2, seed-x
6.                       lw R3, seed-y
7.   next-sub:           sub R1, R2, R3
8.                       sw R1, restart(R11)
9.                       addi R11, R11, 4
10.                      subi R8, R8, 1
11.                      beq R8, R0, exit
12.                      ori R4, R2, 0
12.                      andi R4, R4, 1
13.                      srl R5, R2, 1
14.                      beq R4, R0, complete-x
15.                      xor R5, R5, R9
16.  complete-x:         andi R2, R5, FFFF
17.                      ori R4, R3, 0
18.                      andi R4, R4, 1
19.                      srl R5, R3, 1
20.                      beq R4, R0, complete-y
21.                      xor R5, R5, R10
22.  complete-y:         andi R3, R5, FFFF
23.                      j next-sub
24.  exit:
```

Εικόνα 2.14 Παράδειγμα παραγωγής ψευδοτυχαίων διανυσμάτων (μέσω διαφορετικών ρουτινών δοκιμής για κάθε μονάδα του επεξεργαστή)

Όπως προαναφέραμε, η δεύτερη προσέγγιση χρησιμοποιεί μια κύρια ρουτίνα για την παραγωγή των διανυσμάτων δοκιμής, η οποία καλείται από τις επιμέρους ρουτίνες δοκιμής κάθε μονάδας της σχεδίασης. Το κύριο πλεονέκτημα αυτής της προσέγγισης είναι η χαμηλή απαίτηση ως προς το απαιτούμενο μέγεθος μνήμης για την αποθήκευση της ρουτίνας παραγωγής των ψευδοτυχαίων διανυσμάτων δοκιμής. Το μειονέκτημα αυτής της προσέγγισης είναι ότι απαιτεί πολλές κλήσεις από τις άλλες ρουτίνες του κώδικα, με αποτέλεσμα να αυξάνεται ο συνολικός χρόνος εφαρμογής της δοκιμής. Συνεπώς, η επιλογή της κατάλληλης τεχνικής για την εφαρμογή των ψευδοτυχαίων διανυσμάτων δοκιμής, εξαρτάτε από τις απαιτήσεις τόσο ως προς το μέγεθος του κώδικα, όσο και ως προς τον χρόνο εφαρμογής της δοκιμής. Ένα παράδειγμα εφαρμογής των διανυσμάτων δοκιμής σύμφωνα με την δεύτερη προσέγγιση απεικονίζεται στην Εικόνα 2.15.

```
1.   test-component:     ori R8, R8, max-patterns
2.                       # initialize LFSR routine (seed, mask)
3.   next-pattern:       # call LFSR routine for first operand
4.                       # call LFSR routine for second operand
5.                       # apply the target instruction
6.                       sw R1, restart(R11)
7.   next-sub:           addi R11, R11, 4
8.                       subi R8, R8, 1
9.                       beq R8, R0, exit
10.                      j next-pattern
11.  exit:
```

Εικόνα 2.15 Παράδειγμα παραγωγής ψευδοτυχαίων διανυσμάτων (μέσω μίας κύριας ρουτίνας παραγωγής διανυσμάτων και κλήσεων από τις άλλες ρουτίνες του κώδικα)

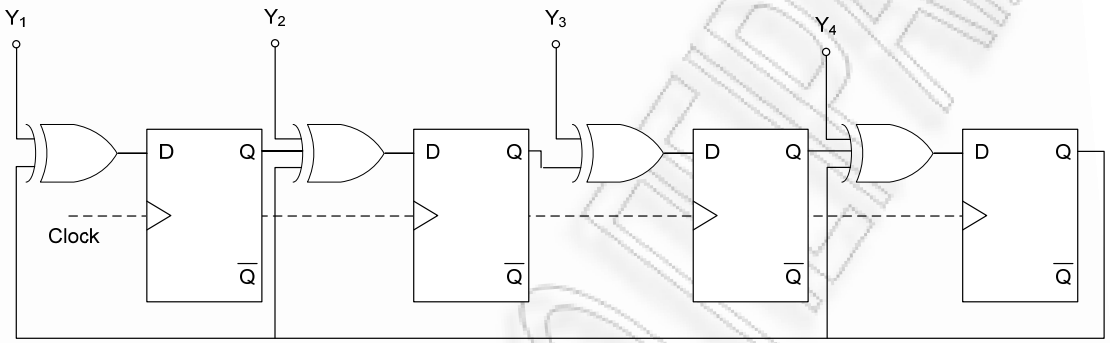
Στον παραπάνω κώδικα υποθέτουμε ότι η απόκριση της δοκιμής συλλέγεται στον καταχωρητή `R1` και έπειτα αποθηκεύεται στην μνήμη. Η ρουτίνα LFSR μπορεί να κληθεί όσες φορές κριθεί απαραίτητο για κάθε πράξη (συνήθως δύο φορές για τους δύο τελεστές της πράξης).

Η ανάλυση υπογραφής ανήκει στην γενική κατηγορία της *σύμπτυξης αποκρίσεων* (*responses compaction*), η οποία αποσκοπεί στην μείωση του όγκου των δεδομένων εξόδου των κυκλωμάτων που προκύπτουν ως αποκρίσεις κατά την διάρκεια της δοκιμής. Οι πιο διαδεδομένες τεχνικές σύμπτυξης των αποκρίσεων είναι ο *καταχωρητής υπογραφής πολλαπλών εισόδων* (*Multiple Input Signature Register, MISR*), η οποία υλοποιείται με την χρήση ενός τροποποιημένου LFSR και η *σύμπτυξη βασισμένη στην συσσώρευση* (*Accumulation-Based Compaction, ABC*).

Στην πρώτη περίπτωση η σύμπτυξη πραγματοποιείται με την εφαρμογή των αποκρίσεων του κυκλώματος στον καταχωρητή MISR. Το περιεχόμενο του καταχωρητή αποτελεί την υπογραφή του κυκλώματος. Η διαδικασία στην ουσία συμπιέζει τις αποκρίσεις όλων των διανυσμάτων δοκιμής σε N ψηφία, όσα είναι και τα φλιπ-φλοπ που περιέχει ο καταχωρητής. Στην Εικόνα 2.16 παρουσιάζεται ένας καταχωρητής MISR των 4-bit ο οποίος περιγράφεται από το χαρακτηριστικό πολυώνυμο $P_4(x) = x^4 + x^3 + x + 1$.

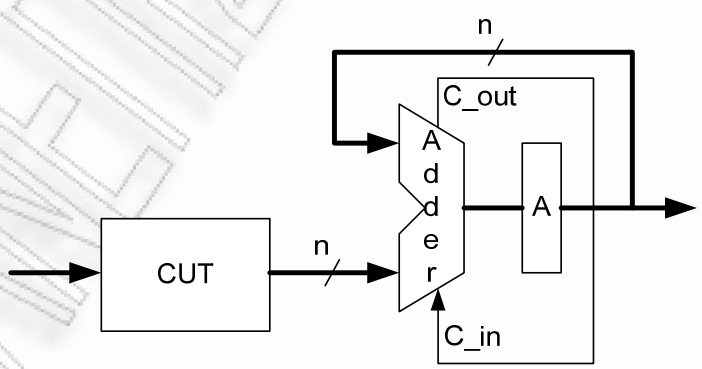
Όπως φαίνεται και από την Εικόνα 2.16, ένας καταχωρητής MISR είναι ουσιαστικά ένας καταχωρητής LFSR με τη διαφορά ότι υπάρχουν πύλες XOR πριν από την είσοδο κάθε φλιπ-φλοπ. Το χαρακτηριστικό πολυώνυμο καθορίζει τη θέση των τυχόν διακλαδώσεων του μονοπατιού ανάδρασης που καταλήγουν στην αντίστοιχη πύλη XOR. Μόλις δοθεί σήμα ρολογιού στον καταχωρητή MISR, τότε οι έξοδοι των πυλών XOR θα αποθηκευτούν στα φλιπ-φλοπ. Δεδομένου ότι μερικές από τις εισόδους των φλιπ-φλοπ είναι ουσιαστικά έξοδοι άλλων φλιπ-φλοπ, προκύπτει ότι η εσωτερική κατάσταση του καταχωρητή MISR εξαρτάται τόσο από

τις εισόδους Y_i όσο και από την προηγούμενη κατάστασή. Συνεπώς, ο καταχωρητής MISR παράγει σε κάθε κύκλο ρολογιού μια υπογραφή των εισόδων του.



Εικόνα 2.16 Καταχωρητής υπογραφής πολλαπλών εισόδων

Στην δεύτερη προσέγγιση, ένας αθροιστής με *n* δυαδικά ψηφία αναλαμβάνει την *συσσώρευση* (*accumulation*) των αποκρίσεων. Η διαδικασία πραγματοποιείται προσθέτοντας ακολουθίες δυαδικών ψηφίων και συνεπώς συσσωρεύοντας τις αποκρίσεις του κυκλώματος σε μία τελική υπογραφή. Η δομή της συσσώρευσης απεικονίζεται στην Εικόνα 2.17. Η έξοδος του κυκλώματος υπό δοκιμή προστίθεται με την προηγούμενη τιμή η οποία είναι αποθηκευμένη στον καταχωρητή Α, μαζί με το κρατούμενο της πρόσθεσης. Η διαδικασία αυτή επαναλαμβάνεται για κάθε απόκριση του κυκλώματος.



Εικόνα 2.17 Συσσώρευση με πρόσθεση κρατούμενου

Η συσσώρευση των αποκρίσεων μπορεί να πραγματοποιηθεί και με την χρήση λογισμικού σε έναν επεξεργαστή, ενώ τον ρόλο του αθροιστή αναλαμβάνει η *αριθμητική και λογική μονάδα* (*Arithmetic and Logic Unit, ALU*). Η Εικόνα 2.18 παρουσιάζει την συσσώρευση των

αποκρίσεων κατά την εφαρμογή διανυσμάτων δοκιμής σε ένα πολλαπλασιαστή. Οι καταχωρητές `R5` και `R6` χρησιμοποιούνται για να φορτωθούν τα διανύσματα δοκιμής από την μνήμη, στους καταχωρητές `R1` και `R2` αντίστοιχα. Στον καταχωρητή `R3` αποθηκεύεται το αποτέλεσμα του πολλαπλασιασμού. Ο καταχωρητής `R4` χρησιμοποιείται για την προσωρινή αποθήκευση του αποτελέσματος της συσσώρευσης, ενώ ο καταχωρητής `R7` για την διευθυνσιοδότηση της θέσης μνήμης αποθήκευσης της τελική υπογραφής. Τέλος ο καταχωρητής `R8` αποτελεί το μετρητή επαναλήψεων του βρόχου και κατά συνέπεια τον αριθμό των διανυσμάτων δοκιμής,

```
1.   test-mult:          ori R8, R0, max-patterns
2.                       add R0, R0, R4
3.                       ori R5, R5, X_addr
4.                       ori R6, R6, Y_addr
5.                       ori R7, R7, Z_addr
6.   next-pattern:       lw R1, [R5]
7.                       lw R2, [R6]
8.                       mult R1, R2 ,R3
9.                       addc R4, R3, R4
10.                      addi R5, R5, 0x4
11.                      addi R6, R6, 0x4
12.  next-sub:           subi R8, R8, 1
13.                      beq R8, R0, exit
14.                      j next-pattern
15.  exit:               store  R4, [R7]
```

Εικόνα 2.18 Συσσώρευση αποκρίσεων κατά την εφαρμογής διανυσμάτων δοκιμής σε έναν επεξεργαστή

2.10 Δοκιμή επεξεργαστών

Η ευρύτατη χρήστη επεξεργαστών σε μυριάδες εφαρμογές είναι δεδομένη. Είτε ως αυτόνομο διακριτό ολοκληρωμένο κύκλωμα (μικροεπεξεργαστές), είτε ως ενσωματωμένη μονάδα (embedded processor), ο επεξεργαστής αποτελεί την καρδιά του συστήματος και πρέπει να λειτουργεί ορθά. Η δοκιμή των επεξεργαστών αποτελεί υπερσύνολο των προκλήσεων που αντιμετωπίζει η βιομηχανία καθώς παρουσιάζει αρκετές ιδιαιτερότητες. Οι επεξεργαστές λειτουργούν σε πολύ υψηλές συχνότητες λειτουργίας με αποτέλεσμα να απαιτείται εξοπλισμός δοκιμής αντίστοιχης απόδοσης και ακρίβειας, με κόστος ιδιαίτερα υψηλό. Επιπλέον λόγω της υψηλής κλίμακας ολοκλήρωσης τους, οι επεξεργαστές ενσωματώνουν εκατομμύρια τρανζίστορ εντός του ολοκληρωμένου, αποτελώντας τα πιο πολύπλοκα συστήματα που έχουν σχεδιαστεί ποτέ. Τέλος, οι επεξεργαστές αποτελούν προϊόντα υψηλού όγκου παραγωγής, εξαναγκάζοντας

τους μηχανικούς να δημιουργήσουν αποτελεσματικές διαδικασίες δοκιμής, διατηρώντας το κόστος στα δυνατότερο χαμηλά επίπεδα.

Όταν οι τεχνικές *για αυξημένη δοκιμαστικότητα που βασίζονται στην σάρωση* (*structured scan-based DFT*) [16], [74], [75], [76], [77] εφαρμοστούν σε επεξεργαστές, επιτυγχάνουν μείωση της πολυπλοκότητα παραγωγής του συνόλου δοκιμής και διευκολύνουν την πρόσβαση για την δοκιμή των ενσωματωμένων κόμβων. Τα κύρια μειονεκτήματα αυτών των προσεγγίσεων είναι η *επιβάρυνση μεγέθους* (*area overhead*) και η *υποβάθμιση απόδοσης* (*performance degradation*). Κατά την εφαρμογή αυτών τον τεχνικών, εισάγονται πολυπλέκτες, τόσο λόγω των σημείων δοκιμής, όσο και για την μετατροπή των φλιπ-φλοπ σε φλιπ-φλοπ σάρωσης, ενώ ταυτόχρονα προστίθενται επιπλέον κύριες είσοδοι – έξοδοι στο κύκλωμα. Επιπλέον, δημιουργούν καθυστερήσεις στα *κρίσιμα μονοπάτια* (*critical paths*) του κυκλώματος. Αυτά τα μειονεκτήματα αποτελούν σημαντικό πρόβλημα, ιδιαίτερα στις περιπτώσεις των επεξεργαστών, οι οποίοι σχεδιάζονται πάντα με γνώμονα την ταχύτητα. Σε επεξεργαστές υψηλής απόδοσης, μια υποβάθμιση της τάξεως ακόμα και του 1% η 2% δεν είναι επιθυμητή. Ακόμα, η αύξηση της επιφάνειας πυριτίου, οδηγεί σε *αύξηση της κατανάλωσης ενέργειας* (*power consumption increase*) κατά την εφαρμογή της δοκιμής. Τέλος, οι τεχνικές αυτές απαιτούν ένα μεγάλο πλήθος δεδομένων δοκιμής και κατά συνέπεια ο χρόνος που απαιτείται για την εφαρμογή αυτών των δεδομένων είναι ιδιαίτερα μεγάλος.

Με την χρήση αυτόματου εξοπλισμού δοκιμής, όπως προαναφέραμε, μπορούμε να δοκιμάσουμε ένα ολοκληρωμένο κύκλωμα αφού πρώτα κατασκευαστεί. Στα κύρια μειονεκτήματα αυτών των προσεγγίσεων είναι το τεράστιο κόστος του εξοπλισμού δοκιμής, το οποίο καθορίζεται από την συχνότητα λειτουργίας και την διαθέσιμη μνήμη. Για την δοκιμή των σύγχρονων μικροεπεξεργαστών των εκατομμυρίων τρανζίστορ και της υψηλής συχνότητας λειτουργίας, το κόστος εξοπλισμού δοκιμής είναι ιδιαίτερα μεγάλο και απαγορευτικό για κάποιες εταιρίες. Καθώς όμως η ταχύτητα των μικροεπεξεργαστών αυξάνεται ο εξοπλισμός δοκιμής έρχεται αντιμέτωπος και με το πρόβλημα της *ανακρίβειας των μετρήσεων* (*ATE/tester measurement inaccuracies*). Δεδομένου ότι η συχνότητα λειτουργίας των μικροεπεξεργαστών είναι της τάξης των GHz και σε συνδυασμό με τη χαμηλότερη τάση λειτουργίας τους, ο εξοπλισμός δοκιμής καλείται να πραγματοποιήσει μετρήσεις μικρών μεταβολών τάσης σε διάστημα μερικών picosecond. Εξαιτίας της δυσκολίας στις μετρήσεις της τάσης στους ακροδέκτες του μικροεπεξεργαστή από τον εξοπλισμό δοκιμής, μια πιθανή απόκλιση ή ανακρίβεια, σε κάποιες μετρήσεις μπορεί να οδηγήσει τον εξοπλισμό δοκιμής να χαρακτηρίσει ένα ορθά κατασκευασμένο επεξεργαστή ως ελαττωματικό. Σε μια τέτοια περίπτωση ο επεξεργαστής δε θα προωθηθεί στην αγορά. Επομένως, οι ανακριβείς μετρήσεις του εξοπλισμού δοκιμής μπορεί να οδηγήσουν στην μείωση της εσοδείας και στην αύξηση του κόστους του μικροεπεξεργαστή. Τέλος, η *υπερδοκιμή* (*overtesting*) αποτελεί ένα ακόμα μειονέκτημα το οποίο οδηγεί επίσης στην μείωση της εσοδείας του επεξεργαστή. Οι προσεγγίσεις που βασίζονται στη

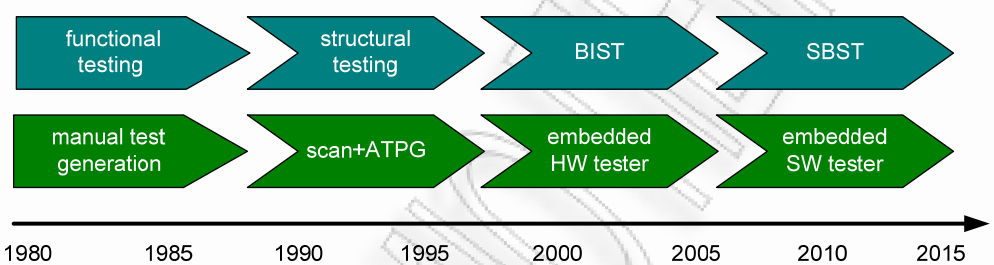
σάρωση ενεργοποιούν τμήματα του επεξεργαστή τα οποία θα παρέμεναν ανενεργά κατά την κανονική λειτουργία. Αν η διαδικασία δοκιμής εντοπίσει ένα τέτοιο ελάττωμα, το οποίο στην ουσία δεν θα αλλοίωνε ποτέ την ορθή λειτουργία του επεξεργαστή, τότε αυτός απορρίπτεται και χαρακτηρίζεται ως ελαττωματικός. Είναι προφανές ότι ο χαρακτηρισμός ενός μη ελαττωματικού επεξεργαστή ως ελαττωματικό αυξάνει για ακόμα μια φορά το κόστος της δοκιμής μειώνοντας την εσοδεία παραγωγής.

Ένα γενικό πρόβλημα των μεθόδων που αναφέραμε είναι ότι στερούνται τη δυνατότητα της αυτοδοκιμής. Οι δομικές τεχνικές αυτοδοκιμής που βασίζονται στο υλικό, όπως η *ενσωματωμένη αυτοδοκιμή* (*Built-In Self-Test, BIST*) αποτελούν μια λύση στο παραπάνω πρόβλημα. Πολλές προσεγγίσεις έχουν προταθεί [78], [79], [80], [81], για την αύξηση της κάλυψης των ελαττωμάτων ενός κυκλώματος, μειώνοντας ταυτόχρονα την επιβάρυνση του μεγέθους και τον χρόνο ανάπτυξης του επεξεργαστή. Σύμφωνα με την ενσωματωμένη αυτοδοκιμή, η παραγωγή των διανυσμάτων δοκιμής καθώς και η συλλογή των αποκρίσεων πραγματοποιείται εντός του επεξεργαστή. Ενσωματώνοντας τη δοκιμή εντός του επεξεργαστή μειώνεται σημαντικά το κόστος δοκιμής, αφού η ανάγκη ακριβών εξωτερικών ελεγκτών για την παραγωγή και την συλλογή των διανυσμάτων και των αποκρίσεων της δοκιμής αντίστοιχα παύει να υφίσταται. Επιπλέον, η δοκιμή πραγματοποιείται στην πραγματική συχνότητα λειτουργίας του επεξεργαστή (at-speed testing), παρέχοντας την δυνατότητα ανίχνευσης ελαττωμάτων καθυστέρησης. Κατά την ενσωματωμένη αυτοδοκιμή δεν έχουμε μείωση εσοδείας λόγω ανακρίβειας του εξωτερικού ελεγκτή, όπως παρατηρήθηκε στις προηγούμενες προσεγγίσεις. Τέλος, το υλικό δοκιμής αποτελεί μια προστιθέμενη αξία που μπορεί να επαναχρησιμοποιηθεί και στα επόμενα στάδια της ζωής του επεξεργαστή, για την εφαρμογή *περιοδικής δοκιμής κατά την λειτουργία* (*periodic on-line testing*).

Στα μειονεκτήματα αυτών των προσεγγίσεων, συγκαταλέγεται η αύξηση της επιφάνειας του επεξεργαστή, καθώς ο ελεγκτής δοκιμής ενσωματώνεται μέσα στον επεξεργαστή. Σε αυτό το σημείο, τίθενται κάποια σημαντικά ζητήματα ως προς τον ελεγκτή δοκιμής. Εάν το πλήθος των *διανυσμάτων δοκιμής* (*test patterns*) είναι μικρό, τότε χρησιμοποιείται ένας μικρός χώρος στη μνήμη για να αποθηκευτούν και ο ελεγκτής εφαρμόζει τα διανύσματα αυτά στον επεξεργαστή. Εάν σε αντίθετη περίπτωση, το πλήθος των διανυσμάτων δοκιμής είναι μεγάλο (αποτελεί την πιο συνηθισμένη περίπτωση) τότε συνήθως μια γεννήτρια ψευδότυχαίων ακολουθιών αναλαμβάνει τον ρόλο της παραγωγής των διανυσμάτων δοκιμής. Η υποβάθμιση της απόδοσης και η αύξηση της κατανάλωσης ενέργειας αποτελούν επίσης μειονεκτήματα αυτών των προσεγγίσεων. Όπως και στις δομικές προσεγγίσεις που βασίζονται στην σάρωση, οι τεχνικές σχεδίασης για αυξημένη δοκιμαστικότητα δημιουργούν καθυστερήσεις στα *κρίσιμα μονοπάτια* (*critical paths*) ενός κυκλώματος.

2.11 Αυτοδοκιμή με λογισμικό

Στην Εικόνα 2.19 απεικονίζεται η τάση των προσεγγίσεων δοκιμής μέσα στην πάροδο του χρόνου. Όλες οι προσεγγίσεις που έχουμε αναφέρει μέχρι τώρα απαιτούν υψηλό κόστος δοκιμής, είτε λόγω του αυτόματου εξοπλισμού δόκιμής υψηλής συχνότητας λειτουργίας και του μεγάλου μεγέθους μνήμης, είτε λόγω της μείωσης της εσοδείας παραγωγής του τελικού προϊόντος.



Εικόνα 2.19 Χρονοδιάγραμμα προσεγγίσεων δοκιμής

Πρόσφατα, η *αυτοδοκιμή με λογισμικό* (*Software-Based Self-Test, SBST*) [22] προτάθηκε ως μία εναλλακτική προσέγγιση με απώτερο στόχο να επιλύσει τα προβλήματα που προκαλούνται τόσο από τις μεθοδολογίες που βασίζονται στην σάρωση όσο και από την ενσωματωμένη αυτοδοκιμή στο υλικό, μειώνοντας δραματικά το συνολικό κόστος της δοκιμής. Η βασική ιδέα της αυτοδοκιμής με λογισμικό είναι η βέλτιστη αξιοποίηση της *αρχιτεκτονικής του σύνολο εντολών* (*Instruction Set Architecture, ISA*) ενός ενσωματωμένου επεξεργαστή, με απώτερο στόχο τη παραγωγή διανυσμάτων δοκιμής, τη μεταφορά δεδομένων δοκιμής, την ανάλυση των αποκρίσεων ή και ακόμη τη διάγνωση ενός επεξεργαστή. Κατά την αυτοδοκιμή με λογισμικό, ο μικροεπεξεργαστής εκτελεί ένα σύνολο ρουτινών, οι οποίες εφαρμόζουν στις επιμέρους μονάδες του ακολουθίες διανυσμάτων δοκιμής. Στη συνέχεια, ο μικροεπεξεργαστής συλλέγει τις αποκρίσεις των διανυσμάτων και τις αποθηκεύει στη *ενσωματωμένη μνήμη δεδομένων* (*embedded data memory*). Εναλλακτικά, το ενσωματωμένο λογισμικό μπορεί να *συμπτύσσει* (*compact*) τις αποκρίσεις ώστε στη ενσωματωμένη μνήμη δεδομένων να αποθηκεύεται μόνο μια *υπογραφή* (*signature*). Η αυτοδοκιμή με λογισμικό, μπορεί να αποτελέσει την συνιστώσα-κλειδί στα "χέρια" της βιομηχανίας των επεξεργαστών στα πλαίσια της ευρύτερης κατανεμημένης στρατηγικής δοκιμής που πλέον ακολουθείται, συμπληρώνοντας τεχνικές σάρωσης και ενσωματωμένης αυτοδοκιμής για την δοκιμή των επεξεργαστών αντιμετωπίζοντας με καινοτομία τους περιορισμούς που θέτει ο παραδοσιακός λειτουργικός έλεγχος στην ταχύτητα λειτουργίας.

Η πιο σημαντική καινοτομία όμως που διαφοροποιεί την αυτοδοκιμή με λογισμικό από τις γνωστές τεχνικές *ενσωματωμένης αυτοδοκιμής που υλοποιούνται στο υλικό* (*Hardware BIST*), οι οποίες επίσης αφορούν δοκιμή με εσωτερικούς πόρους, είναι πως *ο ενσωματωμένος ελεγκτής υλοποιείται με λογισμικό* (*software tester*). Δηλαδή, ο ρόλος του εξωτερικού ελεγκτή υποβιβάζεται στο να τοποθετήσει στην ενσωματωμένη μνήμη εντολών και δεδομένων τα *προγράμματα αυτοδοκιμής* (*self-test programs*) και να ανακτήσει μετά το πέρας της εφαρμογής της δοκιμής, τις αποκρίσεις από την ενσωματωμένη μνήμη δεδομένων. Τα προγράμματα αυτοδοκιμής που αναπτύσσονται με βάση το σύνολο εντολών του επεξεργαστή υπό δοκιμή και αποτελούν τον ενσωματωμένο ελεγκτή που υλοποιείται με λογισμικό, κατά την εκτέλεση τους από τον ίδιο τον επεξεργαστή εφαρμόζουν τα απαραίτητα διανύσματα δοκιμής στις υπομονάδες που τον αποτελούν, στοχεύοντας δομικά μοντέλα ελαττωμάτων για υψηλή αξιοπιστία. Η αυτοδοκιμή με λογισμικό, επιτρέπει τον έλεγχο κατά την *πραγματική συχνότητα λειτουργίας* (*at-speed testing*), με την χρησιμοποίηση εξοπλισμού *εξωτερικού ελεγκτή χαμηλού κόστους* (*low-cost external tester*), προσφέροντας υψηλή αξιοπιστία και ποιότητα του τελικού προϊόντος, η οποία βασίζεται σε δομικά μοντέλα ελαττωμάτων. Το κύριο πλεονέκτημα της αυτοδοκιμής με λογισμικό, είναι ότι προσφέρει πολύ χαμηλό κόστος ανάπτυξης και εφαρμογής της δοκιμής, εξαλείφοντας την ανάγκη ακριβών εξωτερικών ελεγκτών.

Η εφαρμογή της αυτοδοκιμής με λογισμικό για έναν επεξεργαστή βασίζεται στα παρακάτω βήματα και απεικονίζεται στην Εικόνα 2.20.

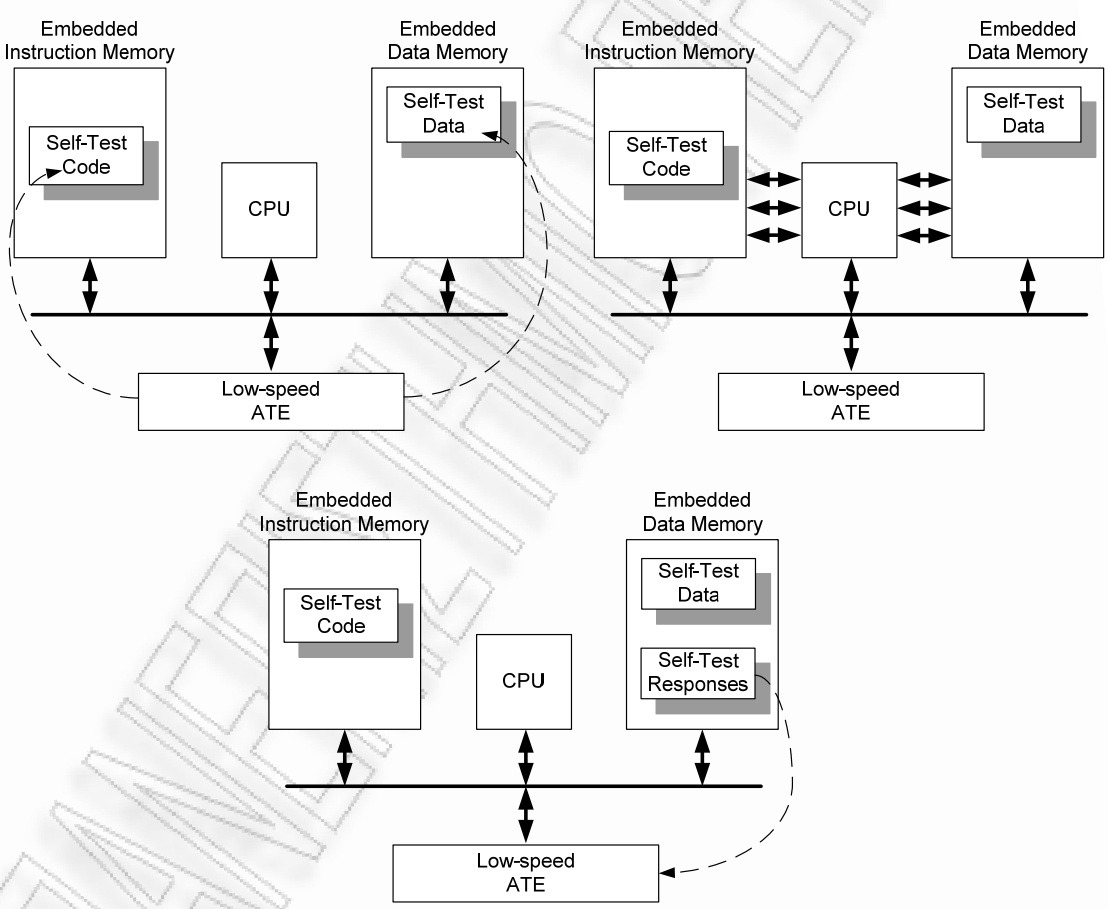
- Ο *κώδικας αυτοδοκιμής* (*self-test code*) *φορτώνεται* (*download*) στην *ενσωματωμένη μνήμη εντολών* (*embedded instruction memory*) του επεξεργαστή, με την χρήση εξωτερικού ελεγκτή, μέσω του *πρωτόκολλου πρόσβασης δοκιμής* (*test access protocol*). Εναλλακτικά, ο κώδικας αυτοδοκιμής μπορεί να είναι μία μνήμη μόνο ανάγνωσης, εξαλείφοντας την ανάγκη του πρώτου βήματος.

- Τα *δεδομένα αυτοδοκιμής* (*self-test data*) φορτώνονται στην *ενσωματωμένη μνήμη δεδομένων* (*embedded data memory*) του επεξεργαστή, με την χρήση εξωτερικού ελεγκτή, μέσω του πρωτόκολλου πρόσβασης δοκιμής. Τα δεδομένα αυτά μπορεί να είναι (α) παράμετροι, μεταβλητές ή σταθερές του ενσωματωμένου επεξεργαστή, (β) *διανύσματα δοκιμής* (*test patterns*) για κάποια υποσυστήματα του επεξεργαστή, (γ) οι αναμενόμενες ελεύθερες ελαττωμάτων αποκρίσεις ώστε να συγκριθούν με τις αντίστοιχες αποκρίσεις του κυκλώματος στην περίπτωση που ο έλεγχος γίνει μέσα στον επεξεργαστή.

- Ο έλεγχος περνάει στο πρόγραμμα αυτοδοκιμής το οποίο και ξεκινά την εκτέλεση του. Τα διανύσματα δοκιμής εφαρμόζονται στα επιμέρους τμήματα του επεξεργαστή μέσω της εκτέλεσης των εντολών. Οι αποκρίσεις των τμημάτων συλλέγονται στην

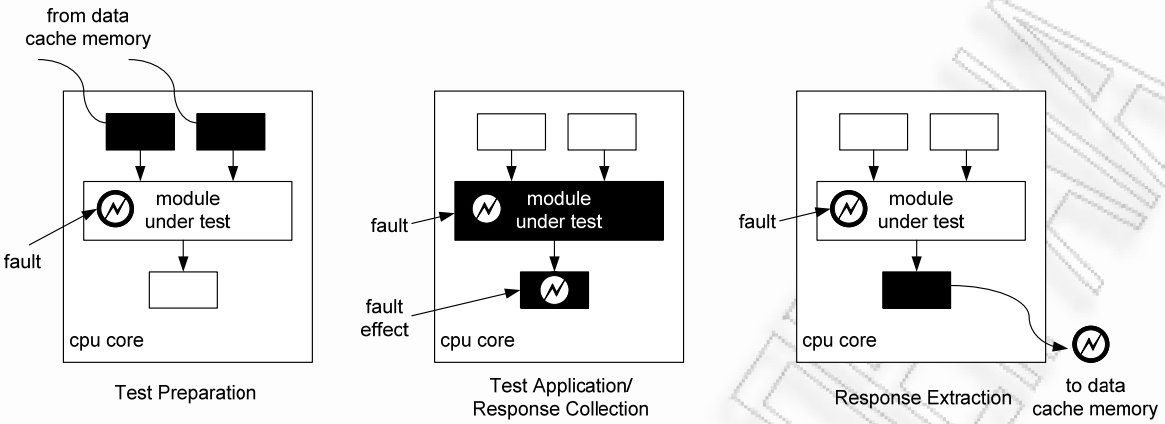
ενσωματωμένη μνήμη δεδομένων. Εναλλακτικά, πραγματοποιείται σύμπτυξη των αποκρίσεων χρησιμοποιώντας έναν γνωστό αλγόριθμο σύμπτυξης, όπου η τελική υπογραφή αποθηκεύεται στην ενσωματωμένη μνήμη δεδομένων.

- Μετά την ολοκλήρωση της εκτέλεσης του κώδικα αυτοδοκιμής, οι αποκρίσεις είτε υπό μορφή υπογραφής, είτε υπό μορφή πολλών διαφορετικών αποκρίσεων, διαβάζονται από τον εξωτερικό ελεγκτή δοκιμής και εξετάζονται ως προς την ορθότητα τους.



Εικόνα 2.20 Αυτοδοκιμή μικροεπεξεργαστών με ενσωματωμένο λογισμικό

Η διαδικασία εφαρμογής της αυτοδοκιμής χωρίζεται σε τρία βασικά βήματα και απεικονίζεται στην Εικόνα 2.21.



Εικόνα 2.21 Βήματα εφαρμογής αυτοδοκιμής με λογισμικό

- *Προετοιμασία της δοκιμής* (*Test Preparation*): Τα διανύσματα δοκιμής τοποθετούνται σε καταχωρητές ή σε διεύθυνση της ενσωματωμένης μνήμης.

- *Εφαρμογή της δοκιμής και συλλογή απόκρισης* (*Test Application and response collection*): Τα διανύσματα δοκιμής εφαρμόζονται στη μονάδα υπό δοκιμή και η απόκριση αυτής συλλέγεται σε ένα καταχωρητή ή σε μία διεύθυνση της ενσωματωμένης μνήμης δεδομένων.

- *Εξαγωγή απόκρισης* (*Response Extraction*): Οι αποκρίσεις συλλέγονται εσωτερικά και εξάγονται προς την ενσωματωμένη μνήμη δεδομένων (εάν δεν είναι ήδη μέσω του βήματος εφαρμογής της δοκιμής και συλλογής απόκρισης).

Για να περιγράψουμε την διαδικασία εφαρμογής ενός διανύσματος αυτοδοκιμής με λογισμικό, θεωρήστε ότι μία *αριθμητική και λογική μονάδα* (*Arithmetic Logic Unit, ALU*) σε έναν επεξεργαστή περιέχει ένα ελάττωμα στο κύκλωμα του αθροιστή. Ο ψευδοκώδικας που περιγράφεται στην Εικόνα 2.22 παρουσιάζει τα βήματα εφαρμογής της δοκιμής για την ανίχνευση αυτού του ελαττώματος. Τα διανύσματα δοκιμής προέρχονται από δύο καταχωρητές του επεξεργαστή, ενώ το αποτέλεσμα της πρόσθεσης αποθηκεύεται σε έναν άλλο καταχωρητή. Οι καταχωρητές που χρησιμοποιούμε στο παρόν παράδειγμα είναι οι `r1`, `r2`, `r3`, όπου οι `r1`, `r2` χρησιμοποιούνται για την εφαρμογή των διανυσμάτων δοκιμής, ενώ ο καταχωρητής `r3` για την συλλογή της απόκρισης. Οι μεταβλητές `X_addr, Y_addr` και `Z_addr`, χρησιμοποιούνται για την διευθυνσιοδότηση των θέσεων μνήμης που είναι αποθηκευμένα τα διανύσματα δοκιμής `X,Y` και η απόκριση της δοκιμής `Z=X+Y` αντίστοιχα. Κατά τις δύο πρώτες εντολές, οι καταχωρητές `r1, r2` φορτώνουν από την ενσωματωμένη μνήμη δεδομένων τα διανύσματα δοκιμής για την αριθμητική λογική μονάδα. Η εντολή πρόσθεσης `(add)` μεταφέρει τα διανύσματα δοκιμής στην

αριθμητική λογική μονάδα, όπου πραγματοποιείται η πράξη της πρόσθεσης. Τέλος, με την εντολή αποθήκευσης (store) η απόκριση δοκιμής αποθηκεύεται σε μία θέσης της ενσωματωμένης μνήμης δεδομένων του επεξεργαστή, ώστε να διαβαστεί από τον εξωτερικό ελεγκτή και να επιβεβαιωθεί η ορθότητα της.

```
1.  load   r1,X_addr     -- Προετοιμασία δοκιμής (βήμα 1)
2.  load   r2,Y_addr     -- Προετοιμασία δοκιμής (βήμα 2)
3.  add    r3,r1,r2      -- Εφαρμογή δοκιμής και συλλογή απόκρισης
4.  store  r3,Z_addr     -- Εξαγωγή απόκρισης
```

Εικόνα 2.22 Ψευδοκώδικας εφαρμογής δοκιμής για την αριθμητική λογική μονάδα

Η αυτοδοκιμή με λογισμικό αποτελεί τον πιο ”φυσικό” τρόπο αυτοδοκιμής των επεξεργαστών καθώς είναι τελείως *διαφανής* (*transparent*) ως προς τον μικροεπεξεργαστή και το μόνο που απαιτεί είναι η ύπαρξη ενσωματωμένης μνήμης στο ίδιο ολοκληρωμένο κύκλωμα με τον μικροεπεξεργαστή. Τα πλεονεκτήματα που παρουσιάζει η αυτοδοκιμή με λογισμικό είναι ποικίλα:

- Η αυτοδοκιμή με λογισμικό βασίζεται αποκλειστικά και μόνο στο σύνολο εντολών του επεξεργαστή και δεν απαιτεί καμία μετατροπή στη σχεδίαση του για αύξηση της δοκιμαστικότητας. Δυσάρεστες συνέπειες, όπως επιβάρυνση του μεγέθους και μείωση της απόδοσής του παύουν να υφίστανται.

- Η δοκιμή είναι ανεξάρτητη του εξοπλισμού δοκιμής και για αυτόν το λόγο η αυτοδοκιμή με λογισμικό αποτελεί μια λύση χαμηλού κόστους για την δοκιμή των μικροεπεξεργαστών, δεδομένου ότι μπορεί να χρησιμοποιηθεί εξοπλισμός δοκιμής χαμηλής ταχύτητας και κατά συνέπεια χαμηλού κόστους. Επιπλέον, η δοκιμή δεν εξαρτάται από τυχόν ανακρίβειες στις μετρήσεις του εξοπλισμού δοκιμής οι οποίες μειώνουν την εσοδεία.

- Όπως και στην αυτοδοκιμή με υλικό, η αυτοδοκιμή με λογισμικό επιτυγχάνει την πραγματοποίηση της δοκιμής στην *ταχύτητα λειτουργίας του επεξεργαστή* (*at-speed*) και όχι σε κάποια χαμηλότερη, επιτρέποντας την ανίχνευση *ελαττωμάτων καθυστέρησης* (*timing faults*) που εμφανίζονται μόνο σε υψηλές συχνότητες λειτουργίας.

- Η αυτοδοκιμή με λογισμικό δεν αυξάνει την κατανάλωση ενέργειας, όπως η ενσωματωμένη αυτοδοκιμή με υλικό, καθώς η δοκιμή πραγματοποιείται στην κανονική λειτουργία και όχι σε κάποια ειδική κατάσταση δοκιμής.

- Η δοκιμή πραγματοποιείται μόνο για εκείνα τα ελαττώματα που επηρεάζουν την ορθή λειτουργία του επεξεργαστή. Το φαινόμενο της *υπερδοκιμής* (*overtesting*) αποφεύγεται και εξασφαλίζεται μεγαλύτερη εσοδεία η οποία μειώνει το συνολικό κόστος δοκιμής.

- Το ενσωματωμένο λογισμικό αυτοδοκιμής μπορεί να χρησιμοποιηθεί τόσο για τη δοκιμή αμέσως μετά την *κατασκευή του μικροεπεξεργαστή* (*manufacturing testing*), όσο και στο *πεδίο λειτουργίας του* (*on-line, in-field testing*). Το λογισμικό αυτοδοκιμής εκτελείται, όπως και το υπόλοιπο ενσωματωμένο λογισμικό και έτσι "ταιριάζει" περισσότερο για αυτοδοκιμή στο πεδίο λειτουργίας σε σχέση με την αυτοδοκιμή βασισμένη σε υλικό που απαιτεί να τεθεί ο μικροεπεξεργαστής σε ειδική κατάσταση δοκιμής. Καθώς επίσης, δεν αυξάνει την κατανάλωση ενέργειας του μικροεπεξεργαστή, αποτελεί ιδανική επιλογή για τη δοκιμή ενσωματωμένων συστημάτων που τροφοδοτούνται από μπαταρίες.

- Η δοκιμή με ενσωματωμένο λογισμικό επιτρέπει την επαναχρησιμοποίηση του ίδιου του μικροεπεξεργαστή για την εξαγωγή, εφαρμογή και συλλογή αποκρίσεων για τη δοκιμή άλλων υποσυστημάτων που βρίσκονται σε ένα *σύστημα σε ένα ολοκληρωμένο κύκλωμα* (*System on Chip, SoC*). Επιπλέον, εάν κατά το πείραμα της δοκιμής παραστεί η ανάγκη για βελτίωση του συνόλου των διανυσμάτων δοκιμής, ίσως για παράδειγμα για ένα άλλο μοντέλο ελαττωμάτων, το λογισμικό αυτοδοκιμής μπορεί πολύ πιο εύκολα να προσαρμοστεί σε σχέση με το υλικό.

Το κόστος ανάπτυξης της δοκιμής ακολουθώντας τεχνικές αυτοδοκιμής που βασίζονται στο λογισμικό αφορά στην ανάπτυξη του κατάλληλου κώδικα αυτοδοκιμής ο οποίος θα φορτωθεί στην ενσωματωμένη μνήμη από εξωτερικό εξοπλισμό χαμηλού κόστους και χαμηλής ταχύτητας, αποτελώντας τον ενσωματωμένο software-tester του επεξεργαστή. Κατά την ανάπτυξη του κώδικα αυτοδοκιμής πρέπει να λαμβάνεται υπόψη η ισχύς του συνόλου εντολών του κάθε επεξεργαστή, ώστε τα διανύσματα δοκιμής να εφαρμοστούν με τον βέλτιστο δυνατό τρόπο, δηλαδή με τον βέλτιστο δυνατό κόστος εφαρμογής.

Οι δύο βασικοί τρόποι παραγωγής των διανυσμάτων δοκιμής είναι:

1. *Ψευδοτυχαίος* (*pseudorandom*).
2. *Ντετερμινιστικός* (*deterministic*).

Ο ψευδοτυχαίος τρόπος παραγωγής είναι ανεξάρτητος του μοντέλου ελαττωμάτων, ενώ ο ντετερμινιστικός εξαρτάται από το μοντέλο ελαττωμάτων (stuck-at, path delay, κλπ). Η ανάπτυξη κώδικα αυτοδοκιμής για την εφαρμογή τεχνικών SBST βασίζεται σε αυτές τις δύο προσεγγίσεις για την παραγωγή των διανυσμάτων δοκιμής. Για παράδειγμα, εάν ένας κώδικας

αυτοδοκιμής ακολουθεί την ψευδοτυχαία προσέγγιση θα βασίζεται στην υλοποίηση στο λογισμικό αλγορίθμων που θα παράγουν ψευδοτυχαία διανύσματα δοκιμής. Εάν ένας κώδικας αυτοδοκιμής ακολουθεί την ντετερμινιστική προσέγγιση, εφαρμόζει ντετερμινιστικά διανύσματα δοκιμής που στοχεύουν δομικά μοντέλα ελαττωμάτων, τα οποία έχουν προκύψει για παράδειγμα από ένα εργαλείο αυτόματης παραγωγής διανυσμάτων δοκιμής, ή άλλες αλγοριθμικές μεθόδους.

Το κόστος εφαρμογής της δοκιμής, σύμφωνα με την μεθοδολογία αυτοδοκιμής με λογισμικό, διαφοροποιείται σημαντικά σε σχέση με τις παραδοσιακές μεθοδολογίες που βασίζονται σε τεχνικές σχεδίασης για αυξημένη δοκιμαστικότητα. Η αυτοδοκιμή με λογισμικό πραγματοποιείται την χρήση εξωτερικού ελεγκτή ο οποίος χρησιμοποιείται μόνο για την διεπαφή με την ενσωματωμένη μνήμη μόνο στο αρχικό και στο τελικό στάδιο της διαδικασίας της δοκιμής και όχι στο κυρίως στάδιο της εφαρμογής της. Το κόστος εφαρμογής της δοκιμής στην αυτοδοκιμή με λογισμικό αναλύεται στις ακόλουθες τρεις επιμέρους συνιστώσες:

1. Τον χρόνο που χρειάζεται για να φορτωθούν ο κώδικας και τα δεδομένα αυτοδοκιμής στην ενσωματωμένη μνήμη.

2. Τον χρόνο που απαιτείται για την εκτέλεση κατά τον κανονικό τρόπο λειτουργίας του προγράμματος αυτοδοκιμής.

3. Τον χρόνο που απαιτείται για την εκφόρτωση των αποκρίσεων από την ενσωματωμένη μνήμη.

Συνεπώς, το συνολικό κόστος εφαρμογής της αυτοδοκιμής με λογισμικό, μειώνει δραματικά το συνολικό κόστος δοκιμής των ολοκληρωμένων κυκλωμάτων. Από τα παραπάνω είναι σαφές, ότι οι συνεχώς αυξανόμενες τάσεις του κόστους της δοκιμής στην βιομηχανία ημιαγωγών, υπαγορεύουν την υιοθέτηση τεχνικών αυτοδοκιμής με λογισμικό, με απώτερο στόχο την μείωση τόσο του συνολικού *κόστους δοκιμής* (*test cost*) όσο και του *χρόνου προς διάθεση* (*time-to-market*), διατηρώντας την αξιοπιστία – ποιότητα του τελικού προϊόντος σε υψηλά επίπεδα. Η αποδοτική οριοθέτηση της απαιτούμενης προσπάθειας που πρέπει να καταβληθεί τόσο από τον μηχανικό δοκιμής όσο και από τον υπολογιστή, συναρτήσει του υψηλού επιπέδου κάλυψης ελαττωμάτων βασιζόμενο σε ένα γνωστό μοντέλο ελαττωμάτων, έχει αποτελέσει αντικείμενο μελέτης πολλών ερευνητικών ομάδων και θα αναλυθεί εκτενέστερα στο επόμενο κεφάλαιο.

2.11.1 Επισκόπηση βιβλιογραφίας αυτοδοκιμής με λογισμικό

Αν και τα πρώτα βήματα της αυτοδοκιμής των μικροεπεξεργαστών με ενσωματωμένο λογισμικό έγιναν από τις αρχές της δεκαετίας του 1970, η εργασία [82] αποτελεί την πρώτη

σημαντική μελέτη. Στην εργασία [82], οι Thatte και Abraham ανέπτυξαν ένα θεωρητικό μοντέλο, *S*-graph για την αναπαράσταση του μικροεπεξεργαστή, η κατασκευή του οποίου βασίστηκε στο σύνολο εντολών και την περιγραφή του μικροεπεξεργαστή σε μοντέλο RTL. Οι ερευνητές ανέπτυξαν λογισμικό αυτοδοκιμής για ένα 8-bit μικροεπεξεργαστή της εταιρίας Helwett-Packard. Το λογισμικό αυτοδοκιμής επιτύγχανε ποσοστό κάλυψης ελαττωμάτων προσκόλλησης 96%. Στην εργασία [83], οι Tupuri και Abraham χρησιμοποίησαν εμπορικά εργαλεία εξαγωγής διανυσμάτων για τις υπομονάδες του επεξεργαστή. Κατά την εξαγωγή των διανυσμάτων, εφάρμοσαν *περιορισμούς* (*constraints*) ώστε τα διανύσματα δοκιμής να είναι εφαρμόσιμα από το σύνολο εντολών του μικροεπεξεργαστή. Μια αναλυτική ιστορική ανασκόπηση των διαφόρων μεθοδολογιών ανάπτυξης λογισμικού αυτοδοκιμής για μικροεπεξεργαστές παρουσιάζεται στην αναφορά [22].

Οι Shen και Abraham στην εργασία [84] προτείνουν την ανάπτυξη λογισμικού αυτοδοκιμής απαριθμώντας όλους τους δυνατούς συνδυασμούς εντολών που ενεργοποιούν τις υπομονάδες του επεξεργαστή, ενώ τα διανύσματα δοκιμής προέρχονται από γεννήτριες ψευδοτυχαίων ακολουθιών. Οι ερευνητές εφάρμοσαν τη μεθοδολογία τους στο μικροεπεξεργαστή GL85. Συνολικά, το λογισμικό αυτοδοκιμής επέτυχε ποσοστό κάλυψης ελαττωμάτων 90.2%, ενώ όταν εφάρμοσαν σύμπτυξη των αποκρίσεων δοκιμής, το ποσοστό κάλυψης μειώθηκε στο 86.7%. Οι Batcher και Papachristou στην εργασία [85], προτείνουν την ανάπτυξη λογισμικού αυτοδοκιμής, όπου η παραγωγή των διανυσμάτων γίνεται με τη βοήθεια υλικού ώστε να είναι ταχύτερη και να μειωθεί ο συνολικός χρόνος εκτέλεσης του λογισμικού. Σύμφωνα με τις πειραματικές μετρήσεις τους, η επιβάρυνση σε υλικό είναι περίπου 3% για ένα επεξεργαστή τύπου RISC. Στα πειράματά τους, ανέπτυξαν δύο διαφορετικά λογισμικά αυτοδοκιμής, με το πρώτο να επιτυγχάνει ποσοστό κάλυψης ελαττωμάτων προσκόλλησης 92.5% έπειτα από εκτέλεση 50.000 κύκλων ρολογιού και το δεύτερο να επιτυγχάνει ποσοστό κάλυψης 94.8% έπειτα από εκτέλεση 200.000 κύκλων.

Η εργασία [87] των Parvathala, Maneparambil και Lindsay έχει ξεχωριστή αξία διότι πραγματεύεται την ανάπτυξη λογισμικού αυτοδοκιμής για τον επεξεργαστή Intel Pentium4. Το σύστημα FRITS που αναπτύχθηκε παράγει λογισμικό αυτοδοκιμής βασισμένο σε ψευδοτυχαίες ακολουθίες εντολών συνδυασμένες με ψευδοτυχαία διανύσματα δοκιμής. Το λογισμικό αυτοδοκιμής φορτώνεται στην *κρυφή μνήμη* (*cache memory*) από τον εξωτερικό ελεγκτή και στη συνέχεια εκτελείται από τον επεξεργαστή. Η μεθοδολογία FRITS εφαρμόστηκε στον επεξεργαστή Pentium4 και πέτυχε ποσοστό κάλυψης ελαττωμάτων 70%. Αντίστοιχη εφαρμογή της μεθοδολογίας στις ακέραιες μονάδες και στις μονάδες κινητής υποδιαστολής του επεξεργαστή Intel Itanium πέτυχε ποσοστό κάλυψης ελαττωμάτων 85%. Στην εργασία [86], ο Corno και οι υπόλοιποι ερευνητές ανέπτυξαν το αυτοματοποιημένο σύστημα ανάπτυξης ενσωματωμένου λογισμικού MicroGP το οποίο βασίζεται στην θεωρία των γενετικών αλγορίθμων. Σε πρώτη φάση, αναπτύσσεται λογισμικό αυτοδοκιμής και έπειτα αξιολογείται το ποσοστό κάλυψης ελαττωμάτων που επιτυγχάνει. Στην συνέχεια, το σύστημα MicroGP

λαμβάνει πληροφορίες από την προσομοίωση των ελαττωμάτων την οποία αξιοποιεί με τους γενετικούς αλγορίθμους και εξελίσσει το αρχικό λογισμικό αυτοδοκιμής σε ένα νέο. Η διαδικασία επαναλαμβάνεται μέχρι να επιτευχθεί το επιθυμητό ποσοστό κάλυψης. Στις πειραματικές μετρήσεις, το σύστημα MicroGP παρήγαγε λογισμικό αυτοδοκιμής για τον μικροελεγκτή 8051 επιτυγχάνοντας ποσοστό κάλυψης ελαττωμάτων 90%.

Στην εργασία [88] οι Chen και Dey δημιουργούν λογισμικό αυτοδοκιμής το οποίο εφαρμόζει διανύσματα δοκιμής τα οποία παράγονται είτε από γεννήτριες ψευδοτυχαίων διανυσμάτων είτε από αυτόματα εργαλεία εξαγωγής διανυσμάτων. Κατά τη διαδικασία της εξαγωγής διανυσμάτων λαμβάνονται υπόψη τυχόν *περιορισμοί* (*constraints*) όπως αυτοί ορίζονται από το σύνολο εντολών του μικροεπεξεργαστή και από τις μικρολειτουργίες που εκτελούν οι υπομονάδες του επεξεργαστή. Η εφαρμογή της μεθοδολογίας της εργασίας [88] στον επεξεργαστή Parwan 8-bit *αρχιτεκτονικής συσσωρευτή* (*accumulator based*) παρήγαγε λογισμικό αυτοδοκιμής με ποσοστό κάλυψης ελαττωμάτων 91.4%. Στην εργασία [89], οι ερευνητές εξελίσσουν την μεθοδολογία ανάπτυξης λογισμικού αυτοδοκιμής και εφαρμόζουν πολλαπλές προσομοιώσεις *προτύπων λογισμικού* (*program templates*) συνδυάζοντας διάφορες εντολές. Έπειτα, χρησιμοποιούν στατιστική ανάλυση ώστε να αναγνωρίσουν τους *περιορισμούς* (*constraints*) που εισάγονται από το σύνολο εντολών ώστε να επιτύχουν την αποτελεσματική εξαγωγή διανυσμάτων δοκιμής. Η εφαρμογή της μεθοδολογίας στο *στάδιο εκτέλεσης εντολών* (*execution stage*) του επεξεργαστή Tensilica (Xtensa™) παρήγαγε λογισμικό αυτοδοκιμής με ποσοστό κάλυψης ελαττωμάτων 95.2%. Η εργασία [90] προσεγγίζει την ανάπτυξη λογισμικού με παρόμοιο τρόπο όπως η εργασία [89]. Η διαφορά έγκειται στον τρόπο εξαγωγής και χρήσης των περιορισμών κατά την εξαγωγή των διανυσμάτων δοκιμής. Αναλυτικότερα, οι ερευνητές εξάγουν γνώση χρησιμοποιώντας τα *δυαδικά διαγράμματα αποφάσεων* (*binary decision diagrams*) για την *μονάδα ελέγχου* (*control logic*) και την *παρεμβολή με πολυώνυμα* (*polynomial interpolation*) για το *μονοπάτι δεδομένων* (*datapath*). Εφάρμοσαν την μεθοδολογία τους σε ένα τμήμα ελέγχου και στην *αριθμητική και λογική μονάδα* (*Arithmetic Logic Unit, ALU*) του επεξεργαστή OpenRisc 1200 επιτυγχάνοντας ποσοστά κάλυψης ελαττωμάτων 69.39% και 99.94%, αντίστοιχα.

Στις εργασίες [91], [92] οι ερευνητές αντίστοιχα με τις προηγούμενες ερευνητικές ομάδες κατασκευάζουν λογισμικό αυτοδοκιμής βασισμένο σε διανύσματα που εξάγονται από αυτόματα εργαλεία. Αντί όμως να προσπαθούν να περιγράψουν τους περιορισμούς που ορίζει το σύνολο εντολών του μικροεπεξεργαστή, εφαρμόζουν τυπικές *μεθόδους επαλήθευσης σχεδίασης* (*formal verification methods*) χρησιμοποιώντας την έννοια του *αντιπαραδείγματος* (*counter example*). Αρχικά, περιγράφεται με τυπική γλώσσα η ιδιότητα πως ένα διάνυσμα δοκιμής δεν είναι δυνατό να εφαρμοστεί σε μια υπομονάδα του επεξεργαστή. Την παραπάνω ιδιότητα προσπαθεί να την αποδείξει ένα εργαλείο επαλήθευσης. Αν το εργαλείο αποδείξει πως η ιδιότητα δεν ισχύει, θα

γεννήσει ένα αντιπαράδειγμα, το οποίο περιγράφει πως θα εφαρμοστεί το διάνυσμα δοκιμής στην υπομονάδα.

Στις εργασίες [93], [94], [95], οι ερευνητές βασίζονται στη γνώση που εξάγεται μελετώντας τη σχεδίαση του μικροεπεξεργαστή σε *επίπεδο μεταφοράς καταχωρητή* (*Register Transfer Level - RTL*) ώστε να αναγνωρίσουν τους περιορισμούς που ορίζει το σύνολο εντολών του μικροεπεξεργαστή. Έπειτα αναπτύσσουν διανύσματα δοκιμής τα οποία εξάγονται βάσει της συμμετρικής δομής των υπομονάδων. Τέτοιες μονάδες είναι οι αθροιστές, αφαιρέτες, συγκριτές, αυξητές, μειωτές, πολλαπλασιαστές αλλά και το *αρχείο καταχωρητών* (*register file*) του μικροεπεξεργαστή. Οι εν λόγω μονάδες αποτελούν ένα μεγάλο ποσοστό του μικροεπεξεργαστή. Καθώς οι επεξεργαστές μεταβαίνουν από αρχιτεκτονικές 32-bit σε αρχιτεκτονικές 64-bit το ποσοστό μεγαλώνει. Στην εργασία [93], οι ερευνητές εφάρμοσαν τη μεθοδολογία τους στον μικροεπεξεργαστή Parwan 8-bit και παρήγαγαν λογισμικό αυτοδοκιμής με ποσοστό κάλυψης ελαττωμάτων 91%, χρόνο εκτέλεσης 16667 κύκλους και μέγεθος μνήμης 923 bytes. Σε σχέση με τα πειραματικά αποτελέσματα της εργασίας [88], διαπιστώνεται η αποτελεσματικότητα της μεθοδολογίας αφού με πολύ μικρότερο και ταχύτερο στην εκτέλεση λογισμικό αυτοδοκιμής επιτυγχάνεται εξίσου υψηλό ποσοστό κάλυψης ελαττωμάτων. Στις εργασίες [94], [95], οι ερευνητές εξελίσσουν τη μεθοδολογία τους και την εφαρμόζουν για πρώτη φορά σε ένα μικροεπεξεργαστή MIPS με τρία στάδια διοχέτευσης, που ονομάζεται Plasma και σε ένα μικροεπεξεργαστή με *σύνολο εντολών ειδικού σκοπού* (*Application Specific Instruction Set-ASIP*) υλοποιημένο σε αρχιτεκτονική MIPS R3000.

Στην εργασίες [96] και [97], οι ερευνητές εξελίσσουν τη μεθοδολογία των εργασιών [94] και [95] και την προσαρμόζουν για την αυτοδοκιμή των μικροεπεξεργαστών που διαθέτουν πολύπλοκους μηχανισμούς διοχέτευσης με *προώθηση* (*forwarding*). Η εφαρμογή της μεθοδολογίας στους μηχανισμούς διοχέτευσης δύο σύγχρονων μικροεπεξεργαστών με 5 στάδια διοχέτευσης, miniMips και OpenRisc 1200 αυξάνει το ποσοστό κάλυψης ελαττωμάτων κατά 22% στο μηχανισμό διοχέτευσης και συνολικά 12% στον μικροεπεξεργαστή.

Η αυτοδοκιμή με λογισμικό για *περιφερειακές μονάδες* (*peripheral devices*) σε ένα *σύστημα σε ολοκληρωμένο* (*System on Chip*) είναι ένα αντικείμενο έρευνας για το οποίο δεν υπήρχαν επαρκείς αναφορές στην διεθνή βιβλιογραφία. Η αυτοδοκιμή με λογισμικό για *περιφερειακές μονάδες επικοινωνίας* (*communication peripherals*) αποτελεί το πρώτο αντικείμενο έρευνας αυτής της διδακτορικής διατριβής.

Επιπλέον, καθώς ο αριθμός των επεξεργαστών στα συστήματα αυξάνει συνεχώς τις μέρες μας, αυξάνεται τόσο το μέγεθος του κώδικα όσο και χρόνος ο εφαρμογής της δοκιμής. Αν και στην βιβλιογραφία υπάρχουν πολλές αναφορές για την αυτοδοκιμή με λογισμικό για έναν ενσωματωμένο επεξεργαστή, η αποδοτική μεταφορά τους στην περίπτωση των *πολυεπεξεργαστών* (*multiprocessor*) δεν είχε αποτελέσει αντικείμενο έρευνας τα τελευταία χρόνια. Το κενό αυτό στη διεθνή βιβλιογραφία, δηλαδή την αυτοδοκιμή των πολυεπεξεργαστών

με ενσωματωμένο λογισμικό, έρχεται να καλύψει το δεύτερο μέρος της παρούσας διδακτορικής διατριβής.

2.12 Ανακεφαλαίωση

Στο Κεφάλαιο 2,

- έγινε η εισαγωγή στην έννοια της δοκιμής, στους τρόπους πραγματοποίησής της και στις δυσκολίες της
- παρουσιάστηκαν τα μοντέλα ελαττωμάτων
- παρουσιάστηκαν οι τεχνικές σχεδίασης για αυξημένη δοκιμαστικότητα
- έγινε η εισαγωγή στην αυτοδοκιμή και στην αυτοδοκιμή μικροεπεξεργαστών με λογισμικό

Κεφάλαιο 3

Δοκιμή Συστημάτων σε Ολοκληρωμένο

The most beautiful experience we can have is the mysterious. It is the fundamental emotion that stands at the cradle of true art and true science...

Albert Einstein

3.1 Εισαγωγή

Η πολυπλοκότητα της σχεδίασης των ολοκληρωμένων κυκλωμάτων με το πέρασμα του χρόνου ολοένα και αυξάνεται καθιστώντας αδύνατο να σχεδιαστεί ένα ολοκληρωμένο κύκλωμα από μηδενική βάση. Ενσωματώνοντας πολλούς διαφορετικούς πυρήνες, μπορούμε να δημιουργήσουμε ένα ενιαίο σύστημα το οποίο ονομάζεται *Σύστημα σε Ολοκληρωμένο* (*System on Chip, SoC*). Τα SoC αποτελούν την πιο συνηθισμένη πρακτική σχεδίασης πολύπλοκων ολοκληρωμένων κυκλωμάτων. Αυτό οφείλεται στο γεγονός της ύπαρξης μιας ιδιαίτερα μεγάλης ποικιλίας *προ-σχεδιασμένων* (*pre-designed*) και *προ-επαληθευμένων* (*pre-verified*) πυρήνων, οι οποίοι είναι διαθέσιμοι στους σχεδιαστές των SoC.

Είναι όμως αναγκαίο να "κτιστεί" μια βιβλιοθήκη που να επιτρέπει την επιλογή των κατάλληλων πυρήνων για την δημιουργία της τελικής εφαρμογής. Τέτοιοι πυρήνες πρέπει να σχεδιαστούν με αυστηρές προδιαγραφές που επιτρέπουν παράλληλα την ενσωμάτωση τους σε διαφορετικά συστήματα. Χαρακτηριστικά παραδείγματα τέτοιων πυρήνων αποτελούν οι

ενσωματωμένοι επεξεργαστές (*embedded processors*), οι *ενσωματωμένες μνήμες* (*embedded memories*) διαφορετικού τύπου, μεγέθους και διαύλου επικοινωνίας, καθώς επίσης οι *περιφερειακοί ελεγκτές επικοινωνίας* (*communication peripheral controllers*) για οποιοδήποτε πρωτόκολλο και μέσο επικοινωνίας. Οι ενσωματωμένοι πυρήνες μπορούν να ανήκουν σε τρεις κύριες κατηγορίες:

1. *Σκληροί πυρήνες* (*Hard cores*): Στην πρώτη κατηγορία, ο σχεδιαστής γνωρίζει μόνο την *φυσική περιγραφή* (*physical description*) των πυρήνων με την μορφή *γεωμετρικής διάταξης* (*geometric layout*). Σε αυτούς τους πυρήνες δεν μπορεί να γίνει καμία απολύτως μετατροπή στην δομή τους.

2. *Ενδιάμεσοι πυρήνες* (*Firm cores*): Στην δεύτερη κατηγορία, ο σχεδιαστής γνωρίζει την *δομική περιγραφή* (*structural description*), όπου περιγράφονται οι συνδέσεις του πυρήνα. Ο πυρήνας έχει ήδη προδιαγραφεί και η λειτουργία του δεν μπορεί να αλλάξει. Ωστόσο, άλλες παράμετροι του πυρήνα όπως η συχνότητα λειτουργίας, η κατανάλωση ισχύος, η φυσική δρομολόγηση (physical routing), οι θέσεις των εισόδων – εξόδων κλπ μπορούν να καθοριστούν για έναν πυρήνα από τον σχεδιαστή.

3. *Μαλακοί πυρήνες* (*Soft cores*): Στην τρίτη κατηγορία ο σχεδιαστής γνωρίζει την *περιγραφή συμπεριφοράς* (*behavioral description*) σε *επίπεδο μεταφοράς καταχωρητή* (*Register Transfer Level, RTL*) όπου η λειτουργία του έχει πλήρως προδιαγραφεί. Οι υλοποιήσεις του πυρήνα μπορούν να είναι ποικίλες και ο σχεδιαστής επιλέγει την υλοποίηση που ταιριάζει στην εκάστοτε εφαρμογή. Σε αυτούς τους πυρήνες είναι εφικτό να αλλάξει ακόμα και η λειτουργία τους.

Τα SoC με τον καιρό γίνονται ολοένα και μικρότερα με την βοήθεια προηγμένων εργαλείων σχεδίασης, κάτι το οποίο όμως με την σειρά του οδηγεί σε νέες προκλήσεις στις τεχνολογίες κατασκευής τους. Ένα από τα σημαντικότερα ζητήματα των SoC είναι η δυσκολία δοκιμής αυτών, λόγω της μειωμένης ελεγξιμότητας και παρατηρησιμότητας των πυρήνων. Οι υπάρχουσες προσεγγίσεις δοκιμής χρησιμοποιούν μία από παραπάνω τις όψεις πυρήνων για τα διαφορετικά στάδια της σχεδίασης. Για παράδειγμα η *αυτόματη παραγωγή διανυσμάτων δοκιμής* (*Automatic Test Pattern Generation, ATPG*) χρησιμοποιεί την δομική περιγραφή, ενώ η *υψηλού επιπέδου προσομοίωση* (*High Level Simulation*) χρησιμοποιεί την περιγραφή συμπεριφοράς. Οι ενδιάμεσοι πυρήνες είναι αυτοί που χρησιμοποιούνται συχνότερα, ενώ οι μαλακοί πυρήνες μπορούν να γίνουν σύνθεση ώστε να προκύψει η περιγραφή δομής.

Οι ενσωματωμένοι πυρήνες σχεδιάζονται λαμβάνοντας υπόψη τις απαιτήσεις διαφορετικών ολοκληρωμένων κυκλωμάτων και συστημάτων. Για παράδειγμα, ας υποθέσουμε τέσσερις διαφορετικές απαιτήσεις:

1. Υψηλά αξιόπιστο σύστημα που προϋποθέτει υψηλό ποσοστό κάλυψης ελαττωμάτων.

2. Υψηλή απόδοση συστήματος που προϋποθέτει υψηλή συχνότητα λειτουργίας.

3. Φορητότητα συστήματος που προϋποθέτει πολύ χαμηλή κατανάλωση ενέργειας.

4. Χαμηλό κόστος ως προς τον τελικό χρήστη.

Ένας σκληρός πυρήνας ο οποίος σχεδιάζεται με γνώμονα την απαίτηση (2), μπορεί να μην είναι οικονομικά αποδοτικός για ένα σύστημα που σχεδιάζεται με γνώμονα την απαίτηση (4). Ωστόσο, εάν ένας πυρήνας σχεδιαστεί κατάλληλα, μπορεί να επαναχρησιμοποιηθεί σε διάφορες εφαρμογές καθώς το κόστος να προσαρμόσουμε τον πυρήνα σε κάποιες άλλες απαιτήσεις (πέραν από το κόστος σχεδίασης), π.χ. για υψηλότερη κάλυψη ελαττωμάτων, ή για υψηλότερη απόδοση, ή για χαμηλότερη κατανάλωση ενέργειας, μπορεί να είναι μικρότερο από το κόστος επανασχεδίασης από την αρχή. Κατά συνέπεια, οι ενσωματωμένοι πυρήνες πρέπει να σχεδιάζονται με γνώμονα την επαναχρησιμοποίηση τους σε διαφορετικές εφαρμογές.

Επιπλέον κατά την σχεδίαση ενός σκληρού πυρήνα πρέπει να γίνουν προσεκτικά διάφορες σχεδιαστικές επιλογές, όπως η επιλογή της βιβλιοθήκης, τα χαρακτηριστικά εισόδων – εξόδων, η επικοινωνία με τους άλλους πυρήνες, η εύκολη φυσική ενσωμάτωση, οι μηχανισμοί των ρολογιών, κλπ.

Οι σημαντικότερες απαιτήσεις δοκιμής για τους ενσωματωμένους πυρήνες περιλαμβάνουν:

1. Υψηλό ποσοστό κάλυψης ελαττωμάτων προσδίδοντας ποιότητα στο τελικό SoC.

2. Περιορισμένο αριθμό ακροδεκτών δοκιμής.

3. Επαναχρησιμοποίηση των διανυσμάτων δοκιμής για εύκολη δοκιμή του συστήματος.

4. Απομόνωση δοκιμής για κάθε πυρήνα ανεξάρτητα της λογικής που βρίσκεται γύρω του.

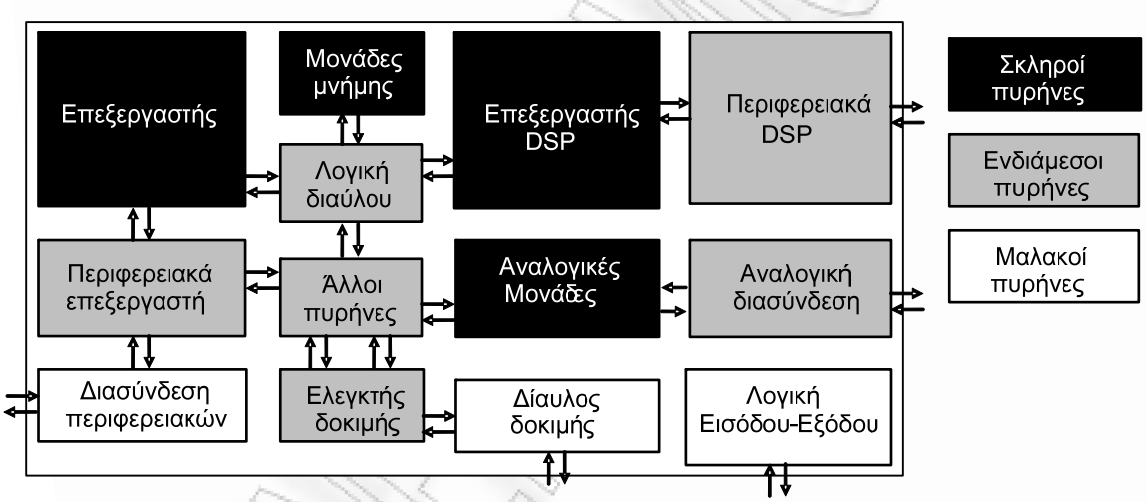
5. *Μηχανισμούς πρόσβασης δοκιμής* (*Test Access Mechanism, TAM*) για εύκολη ενσωμάτωση και δοκιμή.

6. Βελτιστοποίηση δοκιμής όσων αφορά το πλήθος των διανυσμάτων, του χρόνου της δοκιμής, την κατανάλωση της ενέργειας δοκιμής, κλπ.

Οι απαιτήσεις 1 έως 3 είναι συχνά υποχρεωτικές, ενώ οι απαιτήσεις 4 έως 6 μπορούν να είναι προαιρετικές, ανάλογα με τον τύπο του εκάστοτε πυρήνα.

3.2 Αρχιτεκτονική συστημάτων σε ολοκληρωμένο (SoC)

Ένα τυπικό διάγραμμα της αρχιτεκτονικής ενός SoC [98] απεικονίζεται στην Εικόνα 3.1. Περιλαμβάνει δύο πυρήνες επεξεργαστών, συνήθως έναν επεξεργαστή *περιορισμένου συνόλου εντολών* (*Reduced Instruction Set Computer, RISC*) και ένα *ψηφιακό επεξεργαστή σήματος* (*Digital Signal Processor, DSP*), τις *ενσωματωμένες μνήμες* (*embedded memories*), τις *περιφερειακές μονάδες* (*peripheral devices*), τις *αναλογικές μονάδες* (*analog devices*) και τον *δίαυλο επικοινωνίας* (*bus interface*). Η κύρια μονάδα σε ένα ενσωματωμένο σύστημα είναι ο επεξεργαστής ο οποίος και αναλαμβάνει όλες τις κύριες λειτουργίες εκτέλεσης ενός κώδικα, καθώς και τον συγχρονισμό και την επικοινωνία των υπόλοιπων μονάδων εντός του SoC. Ο επεξεργαστής DSP αποτελεί μια πιο ειδικευμένη μονάδα. Σκοπός της είναι η πραγματοποίηση πολύπλοκων μαθηματικών πράξεων αλλά και η μετατροπή σημάτων από αναλογικά σε ψηφιακά. Τόσο ο κύριος επεξεργαστής όσο και ο επεξεργαστής ψηφιακής επεξεργασίας σήματος συνοδεύονται από έναν αριθμό περιφερειακών.



Εικόνα 3.1 Τυπικό διάγραμμα ενός συστήματος σε ολοκληρωμένο (SoC)

Επιπλέον, ένα SoC περιλαμβάνει αναλογικές μονάδες για των επεξεργαστών αναλογικών σημάτων που μπορεί να εισέρχονται σε μία εφαρμογή. Τέτοια αναλογικά σήματα μπορεί να είναι σήματα ήχου, θερμοκρασίας, και άλλα. Επιπλέον, οι σύγχρονες αρχιτεκτονικές SoC ενσωματώνουν μια μεγάλη ποικιλία περιφερειακών συσκευών που ικανοποιούν τις διαφορετικές απαιτήσεις επικοινωνίας της εκάστοτε εφαρμογής. Μερικά χαρακτηριστικά παραδείγματα των περιφερειακών πυρήνων επικοινωνίας σε ένα SoC είναι τα ακόλουθα:

- *Universal Asynchronous Receiver Transmitter,* (*UART*), *I2C*, *Microwire, High-Level Data Link* (*HDLC*) για τη σύνδεση χαμηλής ταχύτητας περιφερειακών συσκευών επικοινωνίας και για λόγους αποσφαλμάτωσης και
- *Ethernet*, *Universal Serial Bus,* (*USB*), *Firewire*, για τη σύνδεση περιφερειακών συσκευών υψηλής ταχύτητας, ή για λόγους σύνδεσης σε δίκτυα επικοινωνίας. Όλοι αυτοί οι περιφερειακοί πυρήνες επικοινωνούν συνήθως με τον επεξεργαστή του συστήματος (είτε τον ενσωματωμένο επεξεργαστή είτε τον επεξεργαστή ψηφιακής επεξεργασίας σήματος είτε και με τους δύο) μέσω μιας διασύνδεσης επικοινωνίας.

Τέλος, ένα SoC περιλαμβάνει έναν *ελεγκτή δοκιμής* (*test controller*) εάν η δοκιμή υλοποιείται μέσω του υλικού. Ο ελεγκτής δοκιμής αναλαμβάνει να δοκιμάσει την ορθή λειτουργία όλων των μονάδων σε ένα SoC μέσω διάφορων τεχνικών δοκιμής. Αυτές οι τεχνικές μπορεί να εκτελεστούν είτε κατά την *κατασκευή του συστήματος* (*manufacturing testing*), είτε *κατά τη λειτουργία του* (*on-line testing*).

Η διαδικασία σχεδίασης ενός SoC περιλαμβάνει [99]:

1. Την αναγνώριση των *λειτουργικών τμημάτων* (*functional blocks*), που μπορούν να υλοποιηθούν χρησιμοποιώντας υπάρχοντες πυρήνες και αυτών που θα πρέπει να σχεδιαστούν από την αρχή.
2. Την σχεδίαση και *ολοκλήρωση* (*integration*) τους σε ένα *σχέδιο κάτοψης* (*floorplan*).
3. Τις *απαιτήσεις διασύνδεσης* (*interface requirements*).
4. Τον καθορισμό των ακροδεκτών του συστήματος.
5. Την *επαλήθευση λειτουργίας* (*verification of functionality*) των διαφορετικών πυρήνων και μονάδων, καθώς και την αλληλεπίδραση τους στο επίπεδο του συστήματος.
6. Την *σχεδίαση για δοκιμαστικότητα* (*design for testability*) όσον αφορά τις διαφορετικές μονάδες ξεχωριστά, αλλά και την ολοκλήρωση τους.
7. Την *φυσική σχεδίαση* (*physical design*) του συστήματος.
8. Τις *ηλεκτρικές και χρονικές αναλύσεις* (*electrical, timing analysis*).
9. Την *παραγωγή των διανυσμάτων δοκιμής* (*test pattern generation*).
10. Την *κατασκευή του τελικού προϊόντος*.

Η *αρχιτεκτονική δοκιμής* (*test architecture*) ενός SoC επηρεάζεται άμεσα από τις αντίστοιχες αρχιτεκτονικές δοκιμής των ενσωματωμένων πυρήνων. Ένα SoC μπορεί να είναι εύκολο η δύσκολο να δοκιμαστεί ανάλογα με τις τεχνικές *σχεδίασης για δοκιμαστικότητα* (*Design For Testability, DFT*) των ενσωματωμένων πυρήνων και την υλοποίηση της *διασύνδεσης δοκιμής* (*test interface*). Σημαντικά ζητήματα δοκιμής ενός SoC είναι:

- Η πρόσβαση της *διασύνδεσης δοκιμής* (*test interface*) κάθε πυρήνα, από τον *δίαυλο διασύνδεσης* (*bus interface*) του SoC, υποθέτοντας ότι δεν είναι όλοι οι ακροδέκτες των ενσωματωμένων πυρήνων προσπελάσιμοι από τις εισόδους εξόδους του SoC.

- Η υψηλή κάλυψη ελαττωμάτων των διαφορετικών πυρήνων και της λογικής των διασυνδέσεων που τα περιβάλλει.

- Οι μηχανισμοί δοκιμής για σειριακή και παράλληλη εφαρμογή των διανυσμάτων δοκιμής στους διαφορετικούς πυρήνες, με στόχο την μείωση του συνολικού χρόνου εφαρμογής της δοκιμής.

- Οι ποικίλες επιλογές για την παραγωγή της δοκιμής και την εφαρμογή της τόσο για τους ενσωματωμένους πυρήνες όσο και για τις συσκευές εισόδου εξόδου.

3.3 Μέθοδοι δοκιμής συστημάτων σε ολοκληρωμένο

Μια από τις μεγαλύτερες προκλήσεις κατά την ανάπτυξη ενός SoC είναι η δοκιμή αυτού λόγω της περιορισμένης δυνατότητας πρόσβασης των ενσωματωμένων πυρήνων από τις εξωτερικές εισόδους – εξόδους του ολοκληρωμένου [98], [100]. Διάφορες προσεγγίσεις *σχεδίασης για δοκιμαστικότητα* (*Design For Testability, DFT*) έχουν ενσωματωθεί στη διαδικασία της σχεδίασης κατά τη διάρκεια των τελευταίων δύο έως τριών δεκαετιών, οδηγούμενες από τις αυξανόμενες απαιτήσεις για ποιότητα μέσω των οικονομικώς αποδοτικών *μηχανισμών δοκιμής* (*test mechanisms*) και μέσω της βελτίωσης της υποστήριξης στα εργαλεία αυτοματοποίησης σχεδίασης για την εισαγωγή σχεδίασης για δοκιμάστικότητα, καθώς και την ανάλυση αυτής, την *αυτόματη παραγωγή διανυσμάτων δοκιμής* (*Automatic Test Pattern Generation, ATPG*) και της *ενσωματωμένης αυτοδοκιμής* (*Built In Self Test, BIST*). Οι γνωστές τεχνικές σχεδίασης για δοκιμαστικότητα περιλαμβάνουν τεχνικές για τη βελτίωση της *ελεγξιμότητας* (*controllability*) και της *παρατηρησιμότητας* (*observability*) για όλες τις δομές λογικής, όπως η *σχεδίαση σάρωσης* (*scan design*) για ακολουθιακά κυκλώματα, η *λογική ελέγχου και παρατήρησης* (*control and observe logic*) για πυρήνες εισόδου - εξόδου και ο *χρονικός*

συγχρονισμός (*timing synchronization*) για ασύγχρονα τμήματα των ολοκληρωμένων κυκλωμάτων.

3.3.1 Πρότυπο δοκιμής IEEE 1500

Τα τελευταία χρόνια καταβλήθηκε μια προσπάθεια ώστε να περιγραφεί μια ενιαία προσέγγιση δοκιμής για τα SoC. Αποτέλεσμα αυτής της προσπάθεια είναι το *πρότυπο δοκιμής ενσωματωμένων πυρήνων IEEE 1500* (*Standard for Embedded Core Test*) [23]. Το IEEE 1500 περιγράφει μια αρχιτεκτονική για την ανάπτυξη και την εφαρμογή της δοκιμής για ενσωματωμένους πυρήνες σε ένα SoC.

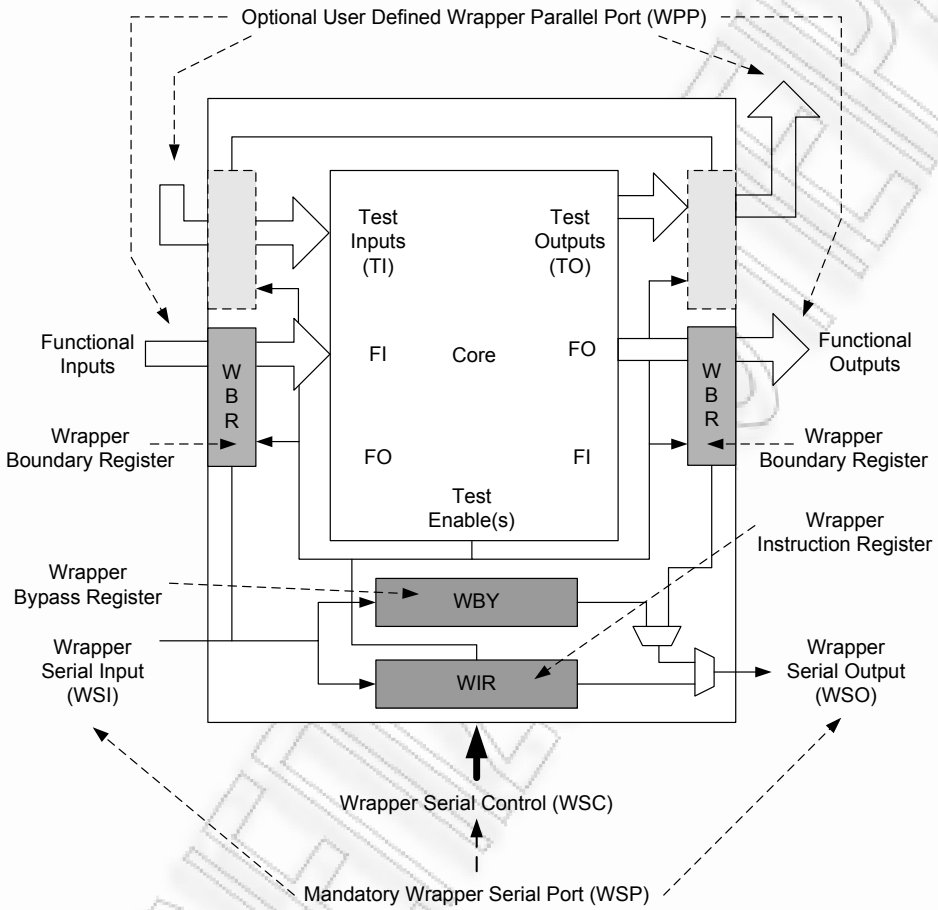
Σύμφωνα με την αρχιτεκτονική του προτύπου IEEE 1500, για να επιτραπεί η πρόσβαση στις εισόδους - εξόδους τους, οι ενσωματωμένοι πυρήνες συνδέονται με τους *μηχανισμούς πρόσβασης δοκιμής* (*Test Access Mechanism, TAM*) μέσω μιας τυποποιημένης εξειδικευμένη; διασύνδεσης, που ονομάζεται *πλαίσιο δοκιμής πυρήνα* (*core test wrapper*). Τα πλαίσια δοκιμής αποτελούνται από ένα κύκλωμα το οποίο προστίθεται περιμετρικά σε έναν ενσωματωμένο πυρήνα, με απώτερο σκοπό την διευκόλυνση εφαρμογής υπαρχόντων διανυσμάτων του πυρήνα και την επίτευξη της επικοινωνίας μεταξύ του μηχανισμού πρόσβασης δοκιμής και του πυρήνα. Τα πλαίσια δοκιμής περιέχουν μια θύρα διασύνδεσης με έξι ακροδέκτες και *δύο σειριακές θύρες εισόδου εξόδου* (*Wrapper Serial Input, WSI και Wrapper Serial Output, WSO*). Ο μηχανισμός πρόσβασης δοκιμής απαιτείται σύμφωνα με το πρότυπο IEEE 1500 ώστε να καταστεί δυνατή η δοκιμή των SoC και επιτρέπει την μετάδοση των δεδομένων δοκιμής από και προς τους πυρήνες ή τα πλαίσια πυρήνων, βελτιώνοντας την ελεγξιμότητα και παρατηρησιμότητα τους. Ένα πλαίσιο δοκιμής σύμφωνα με το IEEE 1500, όπως απεικονίζεται στην Εικόνα 3.2, περιλαμβάνει:

- *Τον καταχωρητή εντολών πλαισίου* (*Wrapper Instruction Register, WIR*): Είναι ένας καταχωρητής σειριακής παράλληλης εισόδου και εξόδου. Αποτελείται από δύο στάδια. Κάθε flip-flop του καταχωρητή οδηγεί ένα κύκλωμα *μανδάλωσης* (*latch*), το οποίο κρατάει αποθηκευμένη την τρέχουσα εντολή όταν ο καταχωρητής ενημερώνεται με νέα δεδομένα (εντολές).

- *Τον καταχωρητή παράκαμψης πλαισίου* (*Wrapper Bypass Register, WBY*): Είναι ένας καταχωρητής ενός σταδίου, ο οποίος επιτρέπει την *σειριακή είσοδο του πλαισίου δοκιμής* (*Wrapper Serial Input, WSI*) να περάσει απευθείας στην *σειριακή έξοδο του πλαισίου* (*Wrapper Serial Output, WSO*), παρακάμπτοντας οποιοδήποτε άλλο καταχωρητή.

- *Τον καταχωρητή περιφερειακής σάρωσης πλαισίου* (*Wrapper Boundary Register, WBR*): Ο καταχωρητής αυτός βρίσκεται στην περιφέρεια του πυρήνα, μεταξύ των

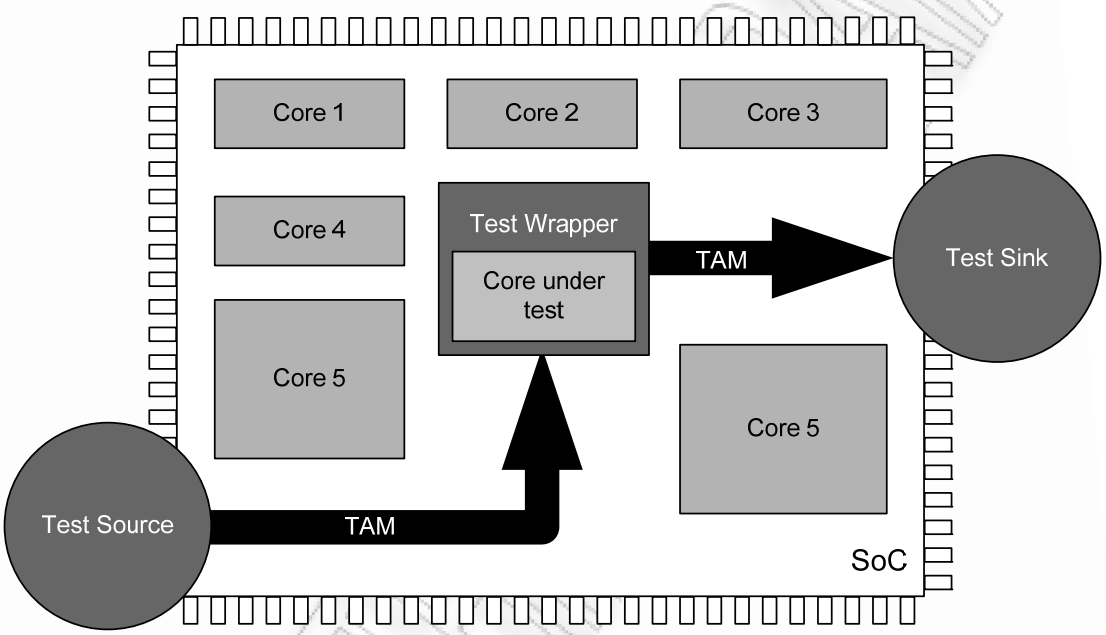
ακροδεκτών εισόδου εξόδου και της εσωτερικής λογικής. Δομείται από τις περιφερειακές *κυψέλες του πλαισίου* (*Wrapper Boundary Cells, WBC*) και συμβάλλει στον έλεγχο της λογικής του πυρήνα καθώς και των διασυνδέσεων του με τους άλλους πυρήνες στο ολοκληρωμένο κύκλωμα.



Εικόνα 3.2 Πλαίσιο δοκιμής (test wrapper) σύμφωνα με το πρότυπο IEEE 1500

Η εφαρμογή της δοκιμής ενός ενσωματωμένου πυρήνα σύμφωνα με το πρότυπο IEEE 1500 απεικονίζεται στην Εικόνα 3.3 και αποτελείται από την *πηγή δοκιμής* (*test source*), την *συλλογή της δοκιμής* (*test sink*), το *πλαίσιο δοκιμής* (*test wrapper*) και τους *μηχανισμούς πρόσβασης δοκιμής* (*test access mechanisms*). Η πηγή δοκιμής είναι δυνατό να εφαρμόσει τα διανύσματα δοκιμής με τρεις διαφορετικούς τρόπους. *Εκτός του ολοκληρωμένου κυκλώματος* (*off-chip*) μέσω ενός αυτόματου εξωτερικού ελεγκτή, *εντός του ολοκληρωμένου κυκλώματος* (*on-chip*) μέσω της ενσωματωμένης αυτοδοκιμής, ή μέσω ενός συνδυασμού και των δύο. Οι μηχανισμοί πρόσβασης δοκιμής μεταφέρουν τα *δεδομένα δοκιμής* (*test data*) στο πλαίσιο δοκιμής που εφαρμόζει με την

σειρά του τα *διανύσματα δοκιμής* (*test patterns*) στον ενσωματωμένο πυρήνα. Στο τέλος, οι αποκρίσεις του πυρήνα μεταφέρονται από το πλαίσιο δοκιμής μέσω του μηχανισμού πρόσβασης δοκιμής, στην συλλογή των αποκρίσεων δοκιμής, για να επαληθευτεί η ορθότητα τους.



Εικόνα 3.3 Αρχιτεκτονική δοκιμής σύμφωνα με το πρότυπο IEEE 1500 αποτελούμενη από 3 στοιχεία: (1) πηγή-συλλογή δοκιμής, (2) μηχανισμός πρόσβασης δοκιμής και (γ) πλαίσιο δοκιμής

Κατά την κανονική λειτουργία του πυρήνα το πρόσθετο υλικό του προτύπου IEEE 1500 παραμένει διαφανές και για τον πυρήνα αλλά και για το σύστημα. Υπάρχουν τρεις βασικοί τρόποι λειτουργίας:

1. *Κανονική λειτουργία* (*normal operation*), όπου το πλαίσιο του πυρήνα παραμένει διαφανές.

2. *Εσωτερική δοκιμή του πυρήνα* (*core internal testing*), όπου δοκιμάζεται η λογική του ενσωματωμένου πυρήνα.

3. *Εξωτερική δοκιμή του πυρήνα* (*core external testing*), όπου δοκιμάζονται οι διασυνδέσεις μεταξύ των πυρήνων και η αντίστοιχη *λογική διασύνδεσης* (*glue logic*).

Η αρχιτεκτονική του IEEE 1500 χρησιμοποιεί την *IEEE 1450 πρότυπη γλώσσα διασύνδεσης δοκιμής* (*Standard Test Interface Language, STIL*) [101] για να μεταβιβάσει τις πληροφορίες μεταξύ των κατασκευαστών των πυρήνων και των σχεδιαστών που τις χρησιμοποιούν. Η

γλώσσα αυτή χρησιμοποιείται για την περιγραφή των διανυσμάτων δοκιμής. Η *IEEE P1450.6 γλώσσα δοκιμής πυρήνα* (*Core Test Language, CTL*) [102] αποτελεί μια προέκταση της γλώσσας διασύνδεσης δοκιμής. Η CTL παρέχει είναι έναν πρότυπο μηχανισμό για την περιγραφή των πλαισίων και των δεδομένων δοκιμής, σύμφωνα με το πρότυπο IEEE 1500.

Η χρησιμοποίηση των πλαισίων δοκιμής εξασφαλίζει υψηλή ποιότητα δοκιμής, αλλά οι σχετικές τροποποιήσεις των κυκλωμάτων είναι πιθανόν να έχουν αρνητική επίδραση στην περιοχή του υλικού και ιδιαίτερα στην απόδοση του τελικού συστήματος. Όταν η δοκιμή του συστήματος σε ολοκληρωμένο, χρησιμοποιώντας μηχανισμούς πρόσβασης δοκιμής, εκτελείται από εξωτερικό εξοπλισμό δοκιμής, η πραγματική συχνότητα λειτουργίας είναι δύσκολο να επιτευχθεί και ο αυξημένος όγκος δεδομένων είναι δυνατόν να οδηγήσει σε ιδιαίτερα μεγάλο χρόνο εφαρμογής της δοκιμής. Επιπλέον, με την χρησιμοποίηση εξωτερικού ελεγκτή δοκιμής τα φαινόμενα υπερδοκιμής υποβαθμίζουν την ποιότητα της δοκιμής και μειώνουν την εσοδεία του τελικού προϊόντος.

Εναλλακτικά, όταν τα διανύσματα δοκιμής παράγονται από έναν εσωτερικό ελεγκτή εξαλείφεται η ανάγκη ύπαρξης εξωτερικού εξοπλισμού δοκιμής. Η *ενσωματωμένη αυτοδοκιμή* (*Built-In Self-Test, BIST*) [103] έχει χρησιμοποιηθεί εκτενώς ως μια αποδοτική λύση για μια μεγάλη ποικιλία πυρήνων, συμπεριλαμβανομένων των επεξεργαστών και των μνημών, καθώς επίσης και των αναλογικών κυκλωμάτων και παρουσιάζει μια συνεχώς αυξανόμενη σημασία και εφαρμοσιμότητα στην εποχή του SoC. Επιτρέπει την δοκιμή στην πραγματική συχνότητα λειτουργίας με σχετικά μικρό κόστος και με καλύτερη ακρίβεια από αυτήν του εξωτερικού ελεγκτή. Επιπλέον, η λογική της ενσωματωμένης αυτοδοκιμής αποτελεί προστιθέμενη αξία για έναν πυρήνα καθώς μπορεί να επαναχρησιμοποιηθεί για διαγνωστικούς ελέγχους στο επίπεδο του SoC, κάτι που είναι ιδιαίτερα χρήσιμο και για τους σχεδιαστές των SoC και για τους χρήστες κρίσιμων εφαρμογών. Αν και οι παραδοσιακές αρχιτεκτονικές ενσωματωμένης αυτοδοκιμής που βασίζονται στο υλικό, όπως τα *ψευδοτυχαία διανύσματα* (*pseudorandom patterns*) και η *βασισμένη σε αλυσίδες σάρωσης ενσωματωμένη αυτοδοκιμή* (*scan-based BIST*), προσφέρουν μια αυτοματοποιημένη, δομημένη λύση για την δοκιμή ενός SoC, ωστόσο επιβάλλουν μια όχι αμελητέα επιβάρυνση στο υλικό του συστήματος, υποβαθμίζοντας την απόδοσης και αυξάνοντας την κατανάλωση ισχύος. Επιπλέον, διάφορες τροποποιήσεις πρέπει να εφαρμοστούν στους πυρήνες προκειμένου αυτοί να καταστούν "έτοιμοι" για αυτού του είδους τις προσεγγίσεις ώστε να ξεπεραστούν προβλήματα, όπως ελαττώματα που δεν ανιχνεύονται εύκολα με τυχαία διανύσματα (random-pattern resistance) και διενέξεις στον δίαυλο επικοινωνίας (bus conflicts) κατά την δοκιμή [103].

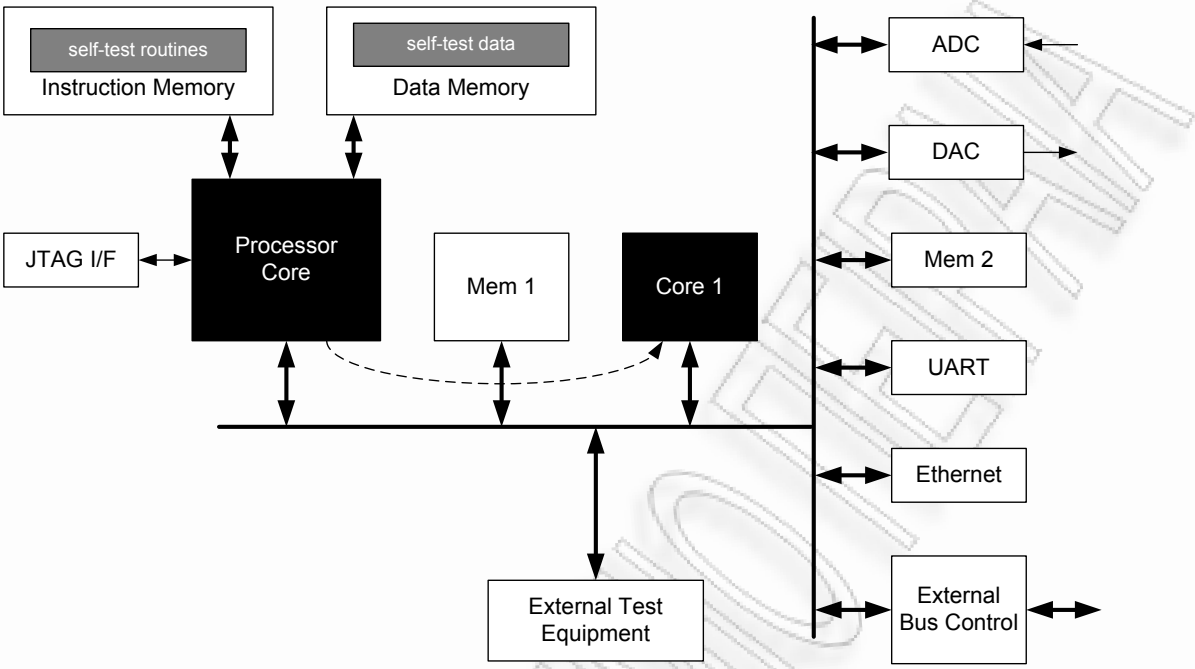
3.3.2 Αυτοδοκιμή βασισμένη στον επεξεργαστή σε συστήματα σε ολοκληρωμένο

Η *αυτοδοκιμή με λογισμικό* (*Software-Based Self-Test, SBST*), που αναλύθηκε στο Κεφάλαιο 2, πέραν της δοκιμής του μικροεπεξεργαστή είναι δυνατόν να χρησιμοποιηθεί και για την δοκιμή των υπόλοιπων μονάδων σε ένα SoC. Μια γενική δομή ενός SoC κατά την εφαρμογή της αυτοδοκιμής με λογισμικό απεικονίζεται στην Εικόνα 3.4 όπου ο "καρδιά" του συστήματος αποτελείται από έναν ενσωματωμένο επεξεργαστή. Τόσο ο κώδικας όσο και τα διανύσματα δοκιμής είναι αποθηκευμένα (όπως και στην περίπτωση της αυτοδοκιμής με λογισμικό για έναν επεξεργαστή), στην ενσωματωμένη μνήμη εντολών και δεδομένων του επεξεργαστή, ή στην εσωτερική μνήμη του SoC.

Αφού ο ενσωματωμένος επεξεργαστής δοκιμαστεί με τεχνικές αυτοδοκιμής με λογισμικό στην ταχύτητα λειτουργίας του, ο ίδιος μπορεί να αποτελέσει μια άριστη πλατφόρμα όπου επάνω της, εκμεταλλευόμενος κανείς την ιδιότητα του εύκολου προγραμματισμού και την επεξεργαστική ισχύ που προσφέρει, μπορεί να τον αξιοποιήσει για την δοκιμή ολόκληρου του ολοκληρωμένου συστήματος. Σύμφωνα με αυτή την προσέγγιση, ο ενσωματωμένος επεξεργαστής εφαρμόζει τα διανύσματα δοκιμής στους ενσωματωμένους πυρήνες έχοντας στην ουσία τον ρόλο του *εσωτερικού ελεγκτή δοκιμής* (*internal test controller*).

Όσο αφορά τους ενσωματωμένους πυρήνες σε ένα SoC είναι δυνατόν να ενσωματώνουν χαρακτηριστικά δοκιμής (αλυσίδες σάρωσης, τεχνικές για αυξημένη δοκιμαστικότητα, ή πλαίσια δοκιμής) από τον ίδιο τον προμηθευτή καθώς επίσης να παρέχονται και τα διανύσματα δοκιμής για τους πυρήνες αυτούς. Σε αυτή την περίπτωση, ο ενσωματωμένος επεξεργαστής θα πρέπει να εφοδιαστεί από τον εξωτερικό ελεγκτή με όλες τις απαραίτητες πληροφορίες (διανύσματα και πρωτόκολλο εφαρμογής της δοκιμής) ώστε να μπορέσει να εφαρμόσει αποδοτικά την δοκιμή και να συλλέξει τις αποκρίσεις.

Επιπλέον, υπάρχουν πυρήνες που δεν ενσωματώνουν κανένα χαρακτηριστικό δοκιμής. Τα διανύσματα δοκιμής σε αυτή την περίπτωση μπορούν να αναπαραχθούν είτε με την βοήθεια ενός εμπορικού αυτόματου εργαλείου παραγωγής διανυσμάτων, είτε μέσω μιας γεννήτριας ψευδοτυχαίων διανυσμάτων. Η εφαρμογή των διανυσμάτων δοκιμής σε αυτές τις περιπτώσεις θα πραγματοποιηθεί από τον ενσωματωμένο επεξεργαστή κατά την *πραγματική συχνότητα λειτουργίας* (*at-speed testing*) του SoC.



Εικόνα 3.4 Παράδειγμα αυτοδοκιμής με λογισμικό σε ένα σύστημα σε ένα ολοκληρωμένο.

Κάποιοι πυρήνες είναι δυνατόν να περιλαμβάνουν μια ενσωματωμένη προσέγγιση δοκιμής. Σε αυτή την κατηγορία πυρήνων περιλαμβάνονται οι μνήμες, οι οποίες συνήθως περιλαμβάνουν μηχανισμούς *ενσωματωμένης αυτοδοκιμής* (*BIST*),. Σε αυτές τις περιπτώσεις, ο ρόλος του ενσωματωμένου επεξεργαστή είναι η έναρξη ή η λήξη της δοκιμής (εάν αυτό κριθεί απαραίτητο) καθώς επίσης και η συλλογή των τελικών υπογραφών της ενσωματωμένης αυτοδοκιμής για περαιτέρω αξιολόγηση. Οι εν λόγω τεχνικές δοκιμής δεν περιορίζονται μόνο στις μνήμες αλλά χρησιμοποιούνται και σε άλλους πυρήνες του συστήματος.

Εκτός από την αξιοποίηση του ενσωματωμένου επεξεργαστή για την δοκιμή των πυρήνων χωρίς καμία μετατροπή στην σχεδίαση του, είναι δυνατόν να χρησιμοποιηθούν επιπλέον δύο προσεγγίσεις. Σύμφωνα με την πρώτη προσέγγιση, ο επεξεργαστής ενσωματώνει επιπλέον *εντολές για αυξημένη δοκιμαστικότητα* (*instruction-level DFT*) ώστε να διευκολύνει τόσο την δοκιμή αυτού, όσο και την δοκιμή των υπόλοιπων πυρήνων. Κατά την δεύτερη προσέγγιση ένας επεξεργαστής προστίθεται, ο οποίο χρησιμοποιείται αποκλείστηκα για την δοκιμή των ενσωματωμένων πυρήνων. Τόσο στη πρώτη όσο και στη δεύτερη περίπτωση, ο κύριος στόχος είναι η μείωση του χρόνου εφαρμογής της δοκιμής του SoC.

Σε όλες τις προηγούμενες περιπτώσεις, είτε οι πυρήνες, είτε ο ίδιος ο επεξεργαστής έπρεπε να ενσωματώσουν χαρακτηριστικά δοκιμής (scan chains, DFT, IEEE 1500, instruction-level DFT) τα οποία διευκόλυναν την διαδικασία δοκιμής. Ενσωματώνοντας τα εν λόγω χαρακτηριστικά κληρονομούμε αναπόφευκτα και όλα τα μειονεκτήματα αυτών των

προσεγγίσεων, όπως η αύξηση της κατανάλωσης ισχύος, η επιβάρυνση της επιφάνειας του ολοκληρωμένου, η μείωση της εσοδείας και η υποβάθμιση της απόδοσης. Από τους πιο σημαντικούς ενσωματωμένους πυρήνες σε ένα SoC είναι τα *περιφερειακά επικοινωνίας* (*communication peripherals*). Τα περιφερειακά επικοινωνίας καταλαμβάνουν ιδιαίτερα μεγάλη επιφάνεια σε ένα SoC ενώ σε κάποιες περιπτώσεις είναι δυνατό να καταλαμβάνουν επιφάνεια μεγαλύτερη ακόμα και από έναν ενσωματωμένο επεξεργαστή [27]. Η αυτοδοκιμή με λογισμικό των ενσωματωμένων περιφερειακών επικοινωνίας, χωρίς να ενσωματώνει κανένα χαρακτηριστικό δοκιμής, καλύπτεται από το πρώτο σκέλος της παρούσας διδακτορικής διατριβής.

3.4 Ανασκόπηση βιβλιογραφίας δοκιμής συστημάτων σε ολοκληρωμένο

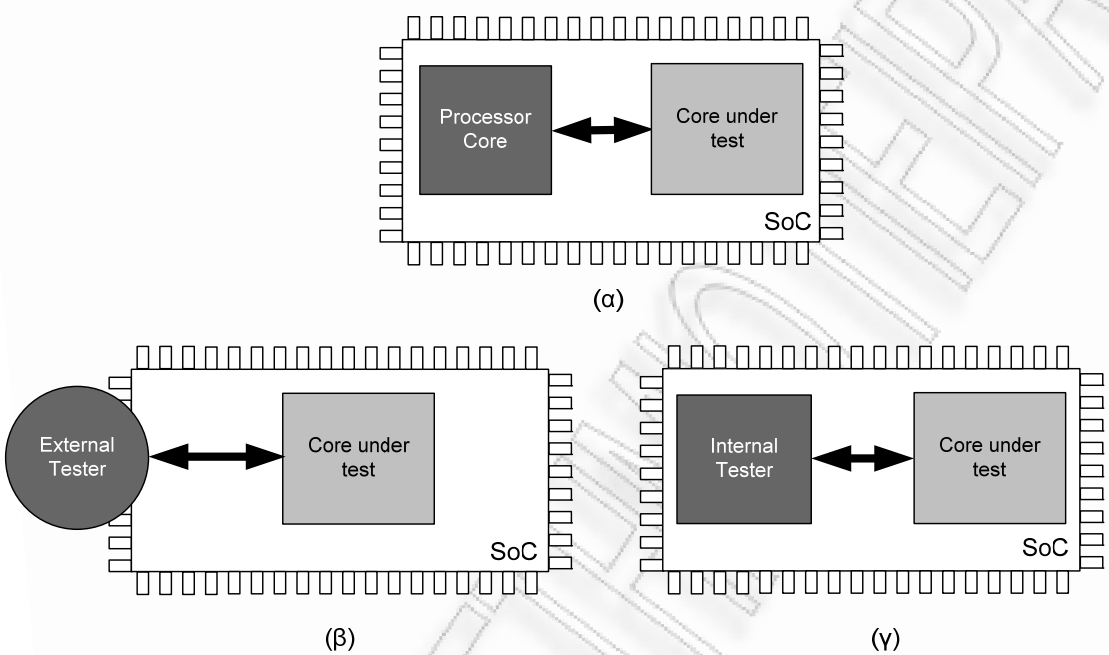
Προηγούμενες προσεγγίσεις [104]-[108] προτείνουν την δοκιμή ενός SoC βασισμένες στο πρότυπο IEEE 1500. Οι προσεγγίσεις αυτές διερευνούν τις τάσεις στην σχεδίαση των SoC και εξετάζουν τα προβλήματα δοκιμής τους. Κάποιες άλλες εργασίες βασισμένες το πρότυπο IEEE 1500 εξετάζουν τις πτυχές του λογισμικό υποστήριξης του προτύπου, τόσο για την υλοποίηση του SoC με στόχο την αποδοτική υποστήριξη της δοκιμής, όσο και του λογισμικού για την εφαρμογή της δοκιμής. Οι προσεγγίσεις αυτές χωρίζονται σε τρεις βασικές κατηγορίες:

- *Σχεδίαση πλαισίων πυρήνων* (*Wrapper Design*) [109] - [113]: Περιλαμβάνουν την αυτοματοποίηση της παραγωγή πλαισίων δοκιμής καθώς και την επαλήθευση τους, εάν το πλαίσιο πυρήνα του συστήματος προσαρμόζεται στο πρότυπο IEEE 1500. Οι προτεινόμενες υλοποιήσεις πλαισίων πυρήνων σύμφωνα με το πρότυπο IEEE 1500 αντιμετωπίζουν πολλούς τύπους πυρήνων και ποικίλα μοντέλων ελαττωμάτων.

- *Τεχνικές σχεδίασης για την ολοκλήρωση σε ένα SoC* (*SoC Design Integration*) [114] - [119]: Η ενσωμάτωση ενός πλαισίου πυρήνα σύμφωνα με το πρότυπο IEEE 1500 σε ένα SoC περιλαμβάνει τη βελτιστοποίηση της δοκιμής ως προς διάφορους παράγοντες κόστους, όπως η επιφάνεια, η απόδοση, ο χρόνος δοκιμής, η ποιότητα δοκιμής, κλπ. Το στάδιο σχεδίασης της δοκιμής περιλαμβάνει την αποδοτική χρονοδρομολόγηση των διαφορετικών δοκιμών του SoC, προκειμένου να βελτιστοποιηθούν η *έγκαιρη ακύρωση της δοκιμής* (*early test abortion*), ο χρόνος εφαρμογής δοκιμής και η κατανάλωση ισχύος.

- *Δημιουργία προγράμματος δοκιμής* (*Test Program Creation*) [120] - [122]: Η δημιουργία δοκιμής για ολόκληρο το SoC περιλαμβάνει τις ακόλουθες δύο πτυχές: (1) τη μετάφραση της δοκιμής από τους ακροδέκτες των πυρήνων στους ακροδέκτες

του SoC για όλους τους πυρήνες που υποστηρίζουν το πρότυπο και (2) την παραγωγή δοκιμής για όλες τις μονάδες του κυκλώματος. Η διαδικασία αυτή βασίζεται στο διαχωρισμό των πυρήνων της δοκιμής, ως προς το πρωτόκολλο και ως προς τα δεδομένα.



Εικόνα 3.5 Εναλλακτικές προσεγγίσεις δοκιμής πυρήνων: (α) ο πυρήνας δοκιμάζεται μέσω του επεξεργαστή του συστήματος, (β) ο πυρήνας δοκιμάζεται μέσω ενός εξωτερικού εξοπλισμού δοκιμής (external tester) και (γ) ο πυρήνας δοκιμάζεται μέσω ενός εσωτερικού ελεγκτή δοκιμής (internal tester).

Προσεγγίσεις στη βιβλιογραφία [123] - [128] έχουν προτείνει την επέκταση της αυτοδοκιμής με λογισμικό στο επίπεδο του SoC για την δοκιμή άλλων μη προγραμματιζόμενων πυρήνων. Αυτές οι προσεγγίσεις στοχεύουν στο να χρησιμοποιήσουν τον επεξεργαστή ως πηγή δοκιμής για τους πυρήνες (Εικόνα 3.5 (α)) σε αντίθεση με τις προηγούμενες προσεγγίσεις που είτε καθοδηγούσαν τους μηχανισμούς πρόσβασης δοκιμής στις εισόδους εξόδους του ολοκληρωμένου προκειμένου να εφαρμοστούν τα διανύσματα δοκιμής από έναν *εξωτερικό εξοπλισμό δοκιμής* (*external tester*) (Εικόνα 3.5 (β)), είτε χρησιμοποιούσαν έναν ή περισσότερους πρόσθετους πυρήνες στο ρόλο της πηγής δοκιμής (Εικόνα 3.5 (γ).

Οι Hwang και Abraham [123] ανέπτυξαν μια αρχιτεκτονική βασισμένη σε υπάρχοντες διαύλους, η οποία χρησιμοποιεί τον ενσωματωμένο επεξεργαστή ως πηγή δοκιμής, τον υπάρχοντα δίαυλο του συστήματος ως μηχανισμό πρόσβασης δοκιμής και την υπάρχουσα λογική διευθύνσεων για την προσπέλαση των καταχωρητών των πυρήνων υπό δοκιμή. Ο μηχανισμός πρόσβασης δοκιμής υλοποιείται μετατρέποντας όλες τις εισόδους εξόδους ενός

περιφερειακού σε καταχωρητές άμεσα προσπελάσιμους από τον ενσωματωμένο επεξεργαστή. Σύμφωνα με αυτήν την προσέγγιση, δεν πραγματοποιείται καμία αλλαγή στον υπάρχοντα δίαυλο ενώ παράλληλα επιτυγχάνεται ο έλεγχος και η παρατήρηση των πυρήνων. Η *διασύνδεση πρόσβασης δοκιμής* (*Test Access Interface, TAI*) παρέχει δύο εναλλακτικούς τρόπους λειτουργίας. Ο πρώτος είναι η *κανονική λειτουργία* (*normal mode*) και ο δεύτερος η *λειτουργία δοκιμής* (*test mode*). Η διασύνδεση πρόσβασης δοκιμής συνδέεται απευθείας στους καταχωρητές εισόδου εξόδου, οι οποίοι στην ουσία έχουν τον ρόλο του πλαισίου δοκιμής. Η προτεινόμενη αρχιτεκτονική παρουσιάστηκε σε ένα SoC βασισμένο σε έναν ενσωματωμένο επεξεργαστή ARM με δύο πυρήνες ISCAS89, έναν ελεγκτή USB και έναν κωδικοποιητή Viterbi. Η μεθοδολογία επιτυγχάνει καλύτερα αποτελέσματα σε σύγκριση με την μέθοδο boundary scan ως προς την επιβάρυνση του υλικού και το συνολικό χρόνο εφαρμογής της δοκιμής.

Ο Huang κ.α. [124] πρότειναν μια προσέγγιση αυτοδοκιμής με λογισμικό για SoC βασισμένα σε δίαυλο, όπου ο ενσωματωμένος επεξεργαστής MIPS εφαρμόζει διανύσματα δοκιμής τα οποία εμπεριέχονται σε *πακέτα δοκιμής* (*test packets*). Τα διανύσματα δοκιμής μπορεί να είναι είτε ντετερμινιστικά, είτε ψευδοτυχαία. Η αποστολή των διανυσμάτων γίνεται μέσω του δίαυλου PCI με απλές εντολές. Τα πακέτα δοκιμής χωρίζονται σε τρία βασικά μέρη: (α) την *εντολή δοκιμής* (*test command*), (β) τον *αριθμό της ακολουθίας* (*sequence number*) και (γ) τα *δεδομένα δοκιμής προς παράδοση* (*test data delivery*). Τα πλαίσια των πυρήνων αποκωδικοποιούν τα πακέτα δοκιμής, αποθηκεύουν τα δεδομένα στους καταχωρητές και εκτελούν τις ακολουθίες δοκιμής. Τέλος, ο ενσωματωμένος επεξεργαστής λαμβάνει τις αποκρίσεις και αξιολογεί εάν ο πυρήνας περιέχει, ή όχι ελαττώματα. Τα πειραματικά αποτελέσματα για ελαττώματα προσκόλλησης και καθυστέρησης μονοπατιού εφαρμόστηκαν σε πυρήνες ISCAS89. Οι Lai και Cheng [125] χρησιμοποίησαν το μηχανισμό που προτάθηκε στην ερευνητική εργασία [124], σε ένα SoC με έναν ενσωματωμένο επεξεργαστή DLX, ώστε να εφαρμόσουν τα διανύσματα δοκιμής σε τέσσερις πυρήνες κυκλωμάτων ISCAS89.

Ο Jayaraman κ.α. [126] πρότειναν μια μεθοδολογία αυτοδοκιμής για ένα SoC, όπου ο ενσωματωμένος επεξεργαστής χρησιμοποιεί τις υπάρχουσες εντολές ώστε να ελέγξει τον εαυτό του καθώς επίσης και τα περιφερειακά του SoC. Οι ερευνητές εφάρμοσαν την προτεινόμενη μεθοδολογία σε ένα SoC το οποίο περιλαμβάνει έναν ενσωματωμένο επεξεργαστή GL85 (μοντέλο του Intel 8085) και ένα συμβατό με το 8251 UART. Τα πειραματικά αποτελέσματα έδειξαν ποσοστό 67.89% κάλυψης ελαττωμάτων για τον περιφερειακό πυρήνα UART κατά την εφαρμογή των *λειτουργικών δοκιμών* (*functional tests*), σε σύγκριση με την *μηχανή παραγωγής δοκιμής για ακολουθιακά κυκλώματα* (*sequential test generation engine*) ενός εμπορικού προγράμματος *αυτόματης παραγωγής διανυσμάτων δοκιμής* (*Automatic Test Pattern Generation, ATPG*) το οποίο πέτυχε ποσοστό κάλυψης ελαττωμάτων μόλις 39.80%. Το σχετικά χαμηλό ποσοστό κάλυψης ελαττωμάτων αυτής της προσέγγισης οφείλεται στον μικρό βαθμό παρατηρησιμότητας του περιφερειακού επικοινωνίας UART.

Ο Bolzani κ.α. [127], προτείνουν μια αυτοματοποιημένη προσέγγιση ικανή να παράγει κώδικα αυτοδοκιμής με λογισμικό για περιφερειακούς πυρήνες που ενσωματώνονται σε ένα SoC. Η προσέγγιση βασίζεται σε έναν *εξελικτικό αλγόριθμο* (*evolutionary algorithm*) που εκμεταλλεύεται την *προσομοίωση υψηλού επιπέδου* (*high-level simulation*) και συγκεντρώνει πληροφορίες τις οποίες και αξιοποιεί με *γενετικούς αλγορίθμους* (*genetic algorithms*) ώστε να εξελίσσει το λογισμικό αυτοδοκιμής. Η παραγωγή του κώδικα δοκιμής εκτελείται με το στόχο την μεγιστοποίηση των *μετρικών κάλυψης κώδικα* (*code coverage metrics*). Μερικές μετρικές κάλυψης κώδικα κατάλληλες για την ανάπτυξη των συνόλων δοκιμής για τους περιφερειακούς πυρήνες που ενσωματώνονται σε ένα SoC είναι οι εξής: *κάλυψη εντολών* (*statement coverage*), *κάλυψη διακλαδώσεων* (*branch coverage*), *κάλυψη συνθηκών* (*condition coverage*), *κάλυψη εκφράσεων* (*expression coverage*) και *κάλυψη εναλλαγών* (*toggle coverage*).

Η προτεινόμενη μέθοδος εφαρμόστηκε σε ένα SoC που βασίζεται στον ενσωματωμένο επεξεργαστή Motorola 6809 και το οποίο περιλαμβάνει δύο περιφερειακά επικοινωνίας, ένα PIA και ένα UART. Για τα δύο αυτά περιφερειακά επιτύχανε ποσοστό κάλυψης ελαττωμάτων 90.20% και 91.43%, αντίστοιχα. Οι ερευνητές σύγκριναν τα αποτελέσματα που επιτυγχάνονται από την μεθοδολογία, με αυτά που παράγονται χειροκίνητα από ένα μηχανικό δοκιμής (89.78% και 80.96% για το PIA και το UART, αντίστοιχα). Αν και η μεθοδολογία κατάφερε να επιτύχει υψηλό ποσοστό κάλυψης ελαττωμάτων και για τα δύο περιφερειακά επικοινωνίας, η μικρή σχετικά πολυπλοκότητα των πυρήνων δεν επιτρέπει την ανάδειξη της αποτελεσματικότητας και τους περιορισμούς της μεθοδολογίας για πιο σύνθετα προβλήματα.

Τέλος, η Kristic κ.α. [128], παρείχαν μια επισκόπηση της αυτοδοκιμής με λογισμικό και των μεθόδων αυτοδοκιμής για τα SoC. Στην εργασία περιγράφονται μεθοδολογίες δοκιμής για τον επεξεργαστή (τόσο για ελαττώματα προσκόλλησης, όσο και για ελαττώματα καθυστέρησης μετάβασης), τον δίαυλο διασύνδεσης και τους ενσωματωμένους πυρήνες. Τα διανύσματα δοκιμής για τους ενσωματωμένους πυρήνες εφαρμόζονται μέσω του διαύλου επικοινωνίας του SoC. Κάθε πυρήνας περιβάλλεται από ένα πλαίσιο δοκιμής που αποκωδικοποιεί τα διανύσματα δοκιμής, τα οποία εφαρμόζονται μέσω αλυσίδων σάρωσης.

Τα βασικά πλεονεκτήματα της αυτοδοκιμής με λογισμικό, είναι ότι δεν επιφέρει επιπλέον κατανάλωση ισχύος στο SoC πέραν της προδιαγραμμένης από τους σχεδιαστές του συστήματος, δεν έχει επιπλέον επιβάρυνση στη λογική και στην απόδοση του SoC. Ωστόσο, οι προηγούμενες προσεγγίσεις [123] - [125] εστιάζουν στην εφαρμογή του προτύπου δοκιμής IEEE 1500 για ένα SoC όπου οι πυρήνες ελέγχονται από τον ενσωματωμένο επεξεργαστή. Αντίθετα, η παρούσα διδακτορική διατριβή υπακούει στις βασικές αρχές εφαρμογής της αυτοδοκιμής με λογισμικό. Οι λειτουργικές προσεγγίσεις της αυτοδοκιμής με λογισμικό των ερευνητικών εργασιών [126] και [127] κατέδειξαν τα προβλήματα αυτοδοκιμής των περιφερειακών πυρήνων, δεδομένου του χαμηλού ποσοστού κάλυψης ελαττωμάτων που η [126] κατάφερε να επιτύχει και δεδομένου της

μικρής πολυπλοκότητας των περιφερειακών επικοινωνίας που χρησιμοποιήθηκαν στην εργασία [127].

3.5 Ζητήματα δοκιμής και περιορισμοί των περιφερειακών επικοινωνίας

Στο πρώτο μέρος της διδακτορικής διατριβής διερευνάται η δυνατότητα εφαρμογής και η αποτελεσματικότητα της αυτοδοκιμής με λογισμικό σε μια δημοφιλή κατηγορία περιφερειακών, τα *περιφερειακά επικοινωνίας* (*communication peripherals*) και παρέχεται μια λεπτομερή πειραματική ανάλυση σε τρία χαρακτηριστικά κυκλώματα αυτής της κατηγορίας: το UART, το HDLC και το Ethernet. Η συμβολή της διατριβής είναι διπλή. Καταρχήν, περιγράφει μια γενική και συστηματική ροή της εφαρμογής της αυτοδοκιμής με λογισμικό στους περιφερειακούς πυρήνες επικοινωνίας σε SoC και δεύτερον, αξιολογεί τη δυνατότητα μιας καθαρά λειτουργικής προσέγγισης για αυτήν την μεγάλη και σημαντική κατηγορία πυρήνων χωρίς την χρησιμοποίηση πρόσθετου υλικού δοκιμής, μηχανισμών πρόσβασης δοκιμής ή πλαισίων πυρήνων και χωρίς να θέτουμε τους πυρήνες σε ειδικές καταστάσης δοκιμής. Οι πυρήνες επικοινωνίας είναι ιδιαίτερα σημαντικοί διότι:

- Χρησιμοποιούνται συνήθως σε ένα ευρύ φάσμα των SoC για διάφορες εφαρμογές όπως, ψηφιακά τηλεοπτικά συστήματα, ασύρματες επικοινωνίες, φορητές εφαρμογές πολυμέσων, εφαρμογές δικτύων, κλπ.

- Καταλαμβάνουν ένα σημαντικό ποσοστό επιφάνειας του κυκλώματος στα SoC. Σε ορισμένες περιπτώσεις είναι δυνατόν να καταλαμβάνουν επιφάνεια υλικού μεγαλύτερη και από αυτή του ενσωματωμένου επεξεργαστή [27]. Συνεπώς, και ο συνολικός αριθμός ελαττωμάτων αυτών των περιφερειακών δεν είναι αμελητέος.

- Παρέχονται συνήθως ως *ενδιάμεσοι ή μαλακοί πυρήνες* (*firm or soft cores*) επιτρέποντας στους σχεδιαστές των συστημάτων να αναπτύξουν προγράμματα αυτοδοκιμής με λογισμικό για αυτούς τους σημαντικούς πυρήνες.

- Δεν έχει παρουσιαστεί καμιά μέθοδος στην βιβλιογραφία για την εφαρμογή της αυτοδοκιμής με λογισμικό με αποδεκτά ποσοστά κάλυψης ελαττωμάτων για πολύπλοκους πυρήνες αυτής της κατηγορίας.

Οι κύριες δυσκολίες στην εφαρμογή της δοκιμής βασισμένης στο λογισμικό για τις περιφερειακές μονάδες επικοινωνίας σε ένα SoC είναι:

- Περιορισμένη ελεγξιμότητα και παρατηρησιμότητα των εσωτερικών καταχωρητών από τους ακροδέκτες του SoC.

- Χαμηλή ταχύτητα μετάδοσης λήψης έναντι της συχνότητας του ενσωματωμένου επεξεργαστή. Αυτό δημιουργεί μια πρόσθετη πρόκληση στην εφαρμογή της αυτοδοκιμής με λογισμικό, δεδομένου ότι ο επεξεργαστής πρέπει να περιμένει κάποια χρονικά διαστήματα πριν αποστείλει την επόμενη ακολουθία δοκιμής. Επομένως, ο χρόνος προσομοίωσης ελαττωμάτων και οι πραγματικός χρόνος εφαρμογής της δοκιμής αυξάνονται ανάλογα. Η ανάπτυξη των ρουτινών αυτοδοκιμής για τις περιφερειακές μονάδες επικοινωνίας επιβάλλει την προσεκτική εκτέλεση τους, ώστε να αποφευχθεί ο υπερβολικός χρόνος εφαρμογής της δοκιμής.

- Για την δοκιμή της λογικής της μηχανής λήψης πρέπει να υπάρχει η δυνατότητα να οδηγηθεί η γραμμή λήψης του περιφερειακού. Ομοίως, για την δοκιμή της λογικής της μηχανής μετάδοσης πρέπει να υπάρχει η δυνατότητα να παρατηρηθεί η γραμμή μετάδοσης του περιφερειακού.

- Τα περιφερειακά τα οποία απαιτούν μεταφορά μεγάλου πλήθους δεδομένων μεταξύ της μνήμης του συστήματος και της εσωτερικής μνήμης του πυρήνα, είναι πιθανόν να ενσωματώνουν ένα *ελεγκτήν άμεσης προσπέλασης μνήμης* (*Direct Memory Access, DMA*). Η ύπαρξη του ελεγκτή DMA εισάγει νέες προκλήσεις κατά την φάση της δοκιμής, δεδομένου ότι προσθέτει επιπλέον λογική στον πυρήνα και κατά συνέπεια περισσότερα πιθανά ελαττώματα.

- Οι περιφερειακοί πυρήνες επικοινωνίας μπορούν να λειτουργήσουν, σύμφωνα με το πρωτόκολλο, σε πολλές διαφορετικές καταστάσεις. Ένα αποτελεσματικό πρόγραμμα δοκιμής πρέπει να τις διερευνήσει όλες αυτές.

Στις ενότητες που ακολουθούν, περιγράφεται μια ντετερμινιστική προσέγγιση αυτοδοκιμής με λογισμικό για περιφερειακούς πυρήνες επικοινωνίας [26], [27]. Η προτεινόμενη προσέγγιση εκμεταλλεύεται τις έμφυτες ιδιότητες των περιφερειακών επικοινωνίας προκειμένου να εφαρμοστούν τόσο οι εξερχόμενες όσο και οι εισερχόμενες λειτουργίες του περιφερειακού με απώτερο σκοπό να ανιχνευτούν τα ελαττώματα και στις δύο πλευρές του πρωτοκόλλου επικοινωνίας. Η προτεινόμενη μεθοδολογία εφαρμόζει ένα εμπλουτισμένο σύνολο *ντετερμινιστικών διανυσμάτων δοκιμής* (*deterministic test patterns*) με στόχο το υψηλό ποσοστό κάλυψης ελαττωμάτων.

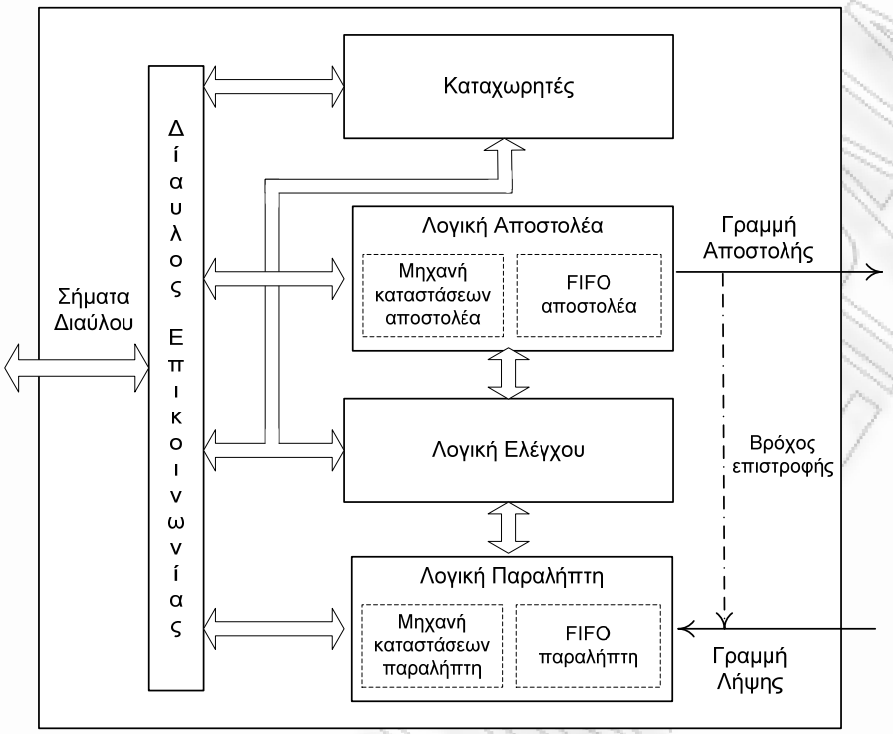
Στη συνέχεια, περιγράφεται μία *υβριδική* προσέγγιση η οποία αποτελεί προέκταση της πρώτης, όπου αυξάνεται ο βαθμός αυτοματοποίησης της μεθοδολογίας. Η υβριδική προσέγγιση [28] αποτελεί συνδυασμό δύο διαφορετικών μεθόδων παραγωγής κώδικα δοκιμής

(ντετερμινιστική προσέγγιση [26], [27] και αυτοματοποιημένη προσέγγιση [127]) συνδυάζοντας τα πλεονεκτήματα και των δύο προσεγγίσεων και εξαλείφοντας κατά το μέγιστο δυνατό τα μειονεκτήματα τους. Η προτεινόμενη μεθοδολογία εκμεταλλεύεται τις ρουτίνες δοκιμής της ντετερμινιστικής προσέγγισης για να καθοδηγήσει αποδοτικότερα την παραγωγή προγράμματος δοκιμής της αυτοματοποιημένης προσέγγισης με απώτερο στόχο την επίτευξη υψηλότερου ποσοστού κάλυψης ελαττωμάτων με την μικρότερη δυνατή ανθρώπινη παρέμβαση. Τα πλεονεκτήματα της υβριδικής μεθοδολογίας έναντι των προηγούμενων προσεγγίσεων επαληθεύονται στους ίδιους πυρήνες επικοινωνίας, ως προς το ποσοστό κάλυψης ελαττωμάτων, το μεγέθους του κώδικα δοκιμής και του συνολικού χρόνου της δοκιμής, αλλά και ως προς το χρόνο που απαιτείται για την παραγωγή του προγράμματος και ως προς τον συνολικό χρόνο εκτέλεσης του κώδικα αυτοδοκιμής.

3.6 Αρχιτεκτονική περιφερειακών επικοινωνίας

Μια τυπική δομή των περιφερειακών επικοινωνίας όπως το UART, το HDLC, το USB και το Ethernet απεικονίζεται στην Εικόνα 3.6. Αποτελείται από πέντε κύριες δομικές μονάδες οι οποίες υλοποιούν την λειτουργία του περιφερειακού επικοινωνίας: Την μονάδα *διασύνδεσης* (*bus interface*), την *μονάδα καταχωρητών* (*registers*), την *μονάδα αποστολέα* (*transmit module*), την *μονάδα ελέγχου* (*control module*) και την *μονάδα του παραλήπτη* (*receive module*).

Οι μονάδες αποστολής και λήψης ενσωματώνουν τις *μηχανές καταστάσεων* (*state machines*) αποστολής και λήψης αντίστοιχα, οι οποίες είναι υπεύθυνες για την μετάδοση των μηνυμάτων από και προς το περιφερειακό επικοινωνίας. Η λογική του διαύλου συνδέει το περιφερειακό με τον δίαυλο επικοινωνίας και χειρίζεται τις λειτουργίες ανάγνωσης - εγγραφής στο αρχείο καταχωρητών του περιφερειακού επικοινωνίας μέσω του κεντρικού ενσωματωμένου επεξεργαστή του συστήματος. Οι καταχωρητές χωρίζονται σε τρεις διαφορετικές κατηγορίες: τους *καταχωρητές έλεγχου* (*control registers*), τους *καταχωρητές κατάστασης* (*status registers*) και τους *καταχωρητές δεδομένων* (*data registers*). Στο κεντρικό αρχείο καταχωρητών που απεικονίζεται στην Εικόνα 3.6 υπάρχουν οι καταχωρητές ελέγχου και κατάστασης, ενώ οι καταχωρητές δεδομένων υλοποιούνται μέσω προσωρινών μνημών FIFO του αποστολέα και του παραλήπτη. Οι τρεις αυτές κατηγορίες καταχωρητών υλοποιούν τις διαφορετικές λειτουργίες για κάθε περιφερειακό επικοινωνίας.



Εικόνα 3.6 Τυπική δομή περιφερειακού επικοινωνίας.

Οι καταχωρητές ελέγχου θέτουν το περιφερειακό σε μια ορισμένη κατάσταση, βάση των απαιτήσεων επικοινωνίας και του πρωτόκολλου μετάδοσης του μηνύματος. Για παράδειγμα, βάση των τιμών που εμπεριέχονται σε αυτούς τους καταχωρητές, ένα UART είναι σε θέση να γνωρίζει *πόσα αρχικά και τελικά ψηφία* (*start and stop bits*) θα πρέπει να μεταδώσει, ή αν θα πρέπει να μεταδώσει το *ψηφίο ισοτιμίας* (*parity bit*). Οι καταχωρητές κατάστασης είναι αρμόδιοι για την απεικόνιση της υπάρχουσας κατάστασης του περιφερειακού επικοινωνίας. Τέτοιες καταστάσεις είναι για παράδειγμα, η μη ορθή λήψη ή αποστολή ενός μηνύματος, η ολοκλήρωση λήψης μηνύματος, το σφάλμα ισοτιμίας του μηνύματος, κλπ. Τέλος οι καταχωρητές δεδομένων περιέχουν τα δεδομένα που η εκάστοτε εφαρμογή επιχειρεί να αποστείλει στον τελικό παραλήπτη, ή έχει λάβει από το εξωτερικό περιβάλλον. Οι καταχωρητές δεδομένων χωρίζονται στους καταχωρητές δεδομένων του αποστολέα και στους καταχωρητές δεδομένων του παραλήπτη. Κατά την μετάδοση ενός μηνύματος, ο ενσωματωμένος επεξεργαστής γράφει τα δεδομένα προς αποστολή στους καταχωρητές δεδομένων του αποστολέα και ξεκινάει την αποστολή. Κατά την λήψη, τα δεδομένα λαμβάνονται από την λογική του παραλήπτη και αποθηκεύονται στους καταχωρητές δεδομένων του παραλήπτη. Ο επεξεργαστής στην τελευταία περίπτωση είναι υπεύθυνος ώστε να διαβάσει τα δεδομένα από τους καταχωρητές δεδομένων του παραλήπτη.

Σε ορισμένες περιπτώσεις, τα περιφερειακά επικοινωνίας ενσωματώνουν έναν *ελεγκτή άμεσης προσπέλασης μνήμης* (*Direct Memory Access Controller, DMA Controller*), υπεύθυνο για την μεταφορά των δεδομένων μεταξύ της κεντρικής μνήμης και των καταχωρητών δεδομένων. Κατά την μετάδοση ενός μηνύματος ο ελεγκτής DMA μεταφέρει τα δεδομένα από την μνήμη του συστήματος στους καταχωρητές του αποστολέα, ενώ κατά την λήψη αναλαμβάνει την μεταφορά των δεδομένων από τους καταχωρητές δεδομένων του παραλήπτη προς την μνήμη του συστήματος. Ο επεξεργαστής σε αυτήν την περίπτωση πραγματοποιεί τις εγγραφές και αναγνώσεις των δεδομένων του μηνύματος αποστολής και λήψης αντίστοιχα στην κεντρική μνήμη του συστήματος. Ο σκοπός ύπαρξης του ελεγκτή DMA είναι αφενός η απελευθέρωση του κεντρικού επεξεργαστή από τις διαδικασίες εγγραφής και ανάγνωσης προς το περιφερειακό επικοινωνίας, ώστε να έχει περισσότερο χρόνο για την εκτέλεση του κώδικα της εφαρμογής. Αφετέρου, επιτυγχάνεται ταχύτερη εγγραφή και ανάγνωση των καταχωρητών δεδομένων του αποστολέα και του παραλήπτη αντίστοιχα. Αυτό συμβαίνει διότι σε ένα SoC ο ρυθμός μεταφοράς δεδομένων μεταξύ ενός επεξεργαστή και ενός περιφερειακού μέσω του διαύλου επικοινωνίας, είναι μικρότερος από τον ρυθμό μεταφοράς των δεδομένων απευθείας από την μνήμη προς το περιφερειακό μέσω του *ελεγκτή άμεσης προσπέλασης μνήμης* (*DMA controller*) λόγω ανταγωνισμού του διαύλου.

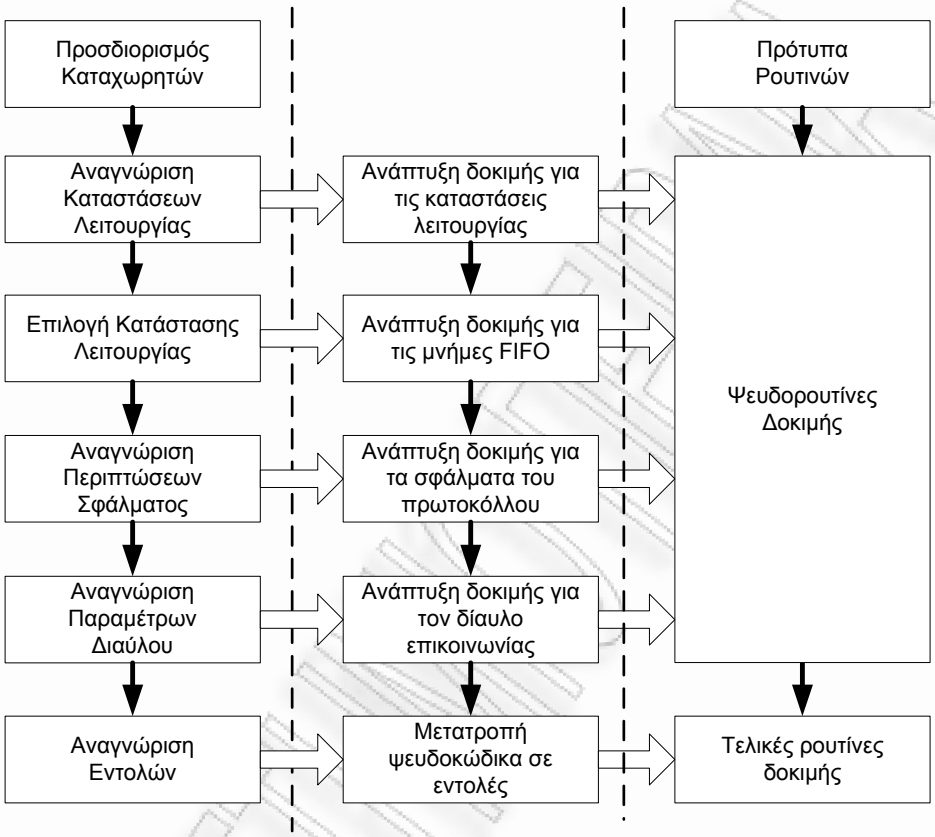
Η λογική του αποστολέα αποτελείται από τη μηχανή μετάδοσης που εκτελεί τις σχετικές διαδικασίες αποστολής σύμφωνα με το πρωτόκολλο επικοινωνίας καθώς και μια μνήμη FIFO. Η μηχανή μετάδοσης λειτουργεί σύμφωνα με τις τιμές των καταχωρητών ελέγχου από το αρχείο καταχωρητών του περιφερειακού. Ομοίως, η λογική του παραλήπτη αποτελείται από τη μηχανή λήψης που εκτελεί όλες τις σχετικές με την λήψη διαδικασίες και μια μνήμη FIFO. Η μηχανή του παραλήπτη λαμβάνει τα εξωτερικά μηνύματα σύμφωνα με τις τιμές των καταχωρητών ελέγχου από το αρχείο καταχωρητών. Οι μνήμες FIFO χρησιμοποιούνται για τον συγχρονισμό μεταξύ του εξωτερικού αποστολέα και του ενσωματωμένου επεξεργαστή του συστήματος. Η λογική του αποστολέα συνδέεται με την λογική του παραλήπτη μέσω του *βρόχου επιστροφής* (*loopback*). Ενεργοποιώντας τον βρόχο επιστροφής από τους καταχωρητές ελέγχου είναι δυνατόν να στείλουμε ένα μήνυμα με την μηχανή του αποστολέα και να το λάβουμε ταυτόχρονα από την μηχανή του παραλήπτη. Ο βρόχος επιστροφής στα περιφερειακά επικοινωνίας χρησιμοποιείται κυρίως για σκοπούς *αποσφαλμάτωσης* (*debugging*) του περιφερειακού. Τέλος, η λογική ελέγχου χρησιμοποιείται τόσο για τις εγγραφές και αναγνώσεις του ενσωματωμένου επεξεργαστή προς και από το αρχείο καταχωρητών αντίστοιχα, όσο και για την μεταφορά των σημάτων και τις τιμές αυτών εντός του περιφερειακού επικοινωνίας.

3.7 Ντετερμινιστική μεθοδολογία αυτοδοκιμής με λογισμικό για περιφερειακά επικοινωνίας

Στην διδακτορική διατριβή παρουσιάζεται μια μεθοδολογία αυτοδοκιμής με λογισμικό για τις περιφερειακές μονάδες επικοινωνίας, όπου ο ενσωματωμένος επεξεργαστής εφαρμόζει ρουτίνες για τη δοκιμή περιφερειακών πυρήνων, χωρίς να χρειάζεται οποιονδήποτε άλλο μηχανισμό πρόσβασης δοκιμής και κατά συνέπεια χωρίς πρόσθετο υλικό, αύξηση της κατανάλωσης ενέργειας, ή επιβάρυνση της απόδοσης του συστήματος.

Στην συγκεκριμένη προσέγγιση, τα δεδομένα δοκιμής είτε είναι αποθηκευμένα στην ενσωματωμένη μνήμη του συστήματος, είτε παράγονται από προγράμματα αυτοδοκιμής που εκτελούνται από τον ενσωματωμένο επεξεργαστή. Τα διανύσματα δοκιμής που εφαρμόζονται στον περιφερειακό πυρήνα επικοινωνίας είναι ουσιαστικά τιμές των καταχωρητών ελέγχου και δεδομένων του περιφερειακού. Ο ενσωματωμένος επεξεργαστής εφαρμόζει τα διανύσματα δοκιμής, εκτελώντας ρουτίνες εγγραφής στους καταχωρητές ελέγχου και θέτοντας το περιφερειακό επικοινωνίας στην κανονική κατάσταση λειτουργίας του. Οι καταχωρητές στους οποίους αποθηκεύεται η απόκριση του περιφερειακού, είναι οι *καταχωρητές κατάστασης και δεδομένων* (*status and data registers*). Ο ενσωματωμένος επεξεργαστής συλλέγει τις αποκρίσεις της δοκιμής εκτελώντας ρουτίνες ανάγνωσης και τις αποθηκεύει στην ενσωματωμένη μνήμη δεδομένων του συστήματος. Κατά συνέπεια, ο ενσωματωμένος επεξεργαστής χρησιμοποιείται ως πηγή δοκιμής, ο δίαυλος επικοινωνίας ως μηχανισμός πρόσβασης δοκιμής και η υπάρχουσα λογική διευθυσιοδότησης χρησιμοποιείται για την πρόσβαση των εσωτερικών καταχωρητών του περιφερειακού.

Όπως σε κάθε περίπτωση αυτοδοκιμής με λογισμικό σε ένα SoC, πριν την έναρξη εφαρμογής της δοκιμής προς τα περιφερειακά, ο ενσωματωμένος επεξεργαστής εκτελεί ένα κώδικα αυτοδοκιμής ώστε να επαληθεύσει την ορθότητα λειτουργίας του. Εάν διαπιστωθεί ότι ο ενσωματωμένος επεξεργαστής είναι χωρίς ελαττώματα, τότε λαμβάνει τον ρόλο της πηγής της δοκιμής και εφαρμόζει τα διανύσματα αυτής προς τα περιφερειακά του SoC. Σε αντίθετη περίπτωση, όπου διαπιστωθεί ότι ο ενσωματωμένος επεξεργαστής εμπεριέχει ελαττώματα, τότε το SoC χαρακτηρίζεται ως ελαττωματικό και η διαδικασία δοκιμής τερματίζεται.



Εικόνα 3.7 Βήματα της μεθοδολογίας.

Στην Εικόνα 3.7 παρουσιάζονται τα βασικά βήματα της ανάπτυξης του κώδικα δοκιμής της ντετερμινιστικής μεθοδολογίας για τα περιφερειακά επικοινωνίας. Όντας *λειτουργική* (*functional*), η μεθοδολογία μοιάζει με τη *λειτουργική δοκιμή επαλήθευσης* (*verification*) για τους περιφερειακούς πυρήνες, αλλά εφαρμόζει ένα εμπλουτισμένο σύνολο διανυσμάτων δοκιμής για να ανιχνεύσει τα ελαττώματα των περιφερειακών επικοινωνίας. Τα πρώτα βήματα της μεθοδολογίας αφορούν την παραγωγή του ψευδοκώδικα αυτοδοκιμής για τις διαφορετικές καταστάσεις λειτουργίας του πυρήνα. Η παραγωγή των ρουτινών δοκιμής βασίζεται κυρίως στην ύπαρξη *πρότυπης ρουτίνας* (*routine template*) δοκιμής για κάθε βήμα της μεθοδολογίας. Συνεπώς, σε κάθε βήμα, ο *μηχανικός δοκιμής* (*test engineer*) πρέπει να προσδιορίσει τις λειτουργικές παραμέτρους του περιφερειακού πυρήνα. Αυτές οι παράμετροι περιγράφονται στις ακόλουθες ενότητες όπου αναλύουμε τα βήματα της μεθοδολογίας. Στο τελικό βήμα, όπου οι λεπτομέρειες του *συνόλου εντολών του επεξεργαστή* (*Instruction Set Architecture, ISA*) είναι διαθέσιμες, οι ψευδορουτίνες δοκιμής μετασχηματίζονται στα τελικά προγράμματα δοκιμής. Τα βήματα παραγωγής της δοκιμής, μαζί με το βήμα μετασχηματισμού των ρουτινών, θα μπορούσαν κατά ένα μεγάλο ποσοστό να αυτοματοποιηθούν και να ενσωματωθούν σε ένα

αυτόματο εργαλείο λογισμικού που απαιτεί την ελάχιστη παρέμβαση από τους χρήστες για τον καθορισμό των παραμέτρων του πυρήνα.

Η ντετερμινιστική μεθοδολογία χωρίζεται σε τέσσερα βασικά βήματα καθένα από τα οποία εξετάζει την ανίχνευση των ελαττωμάτων στις τέσσερεις κύριες μονάδες του περιφερειακού επικοινωνίας. Ο διαχωρισμός της δοκιμής σε τέσσερα επιμέρους τμήματα πραγματοποιείται για την διευκόλυνση της διαδικασίας παραγωγής του κώδικα δοκιμής, σύμφωνα με τα πρότυπα ρουτινών της κάθε μονάδας. Επιπλέον, η ντετερμινιστική φύση της προτεινόμενης μεθοδολογίας, δεν επιτρέπει την αντιμετώπιση όλων των μονάδων του περιφερειακού επικοινωνίας με μία ενιαία προσέγγιση με τον στόχο το υψηλό ποσοστό κάλυψης ελαττωμάτων για καθεμία από αυτές τις μονάδες. Τα επιμέρους βήματα ροής της μεθοδολογίας είναι:

1. *Ανάπτυξη δοκιμής για τις καταστάσεις λειτουργίας* (*test development for the operating modes*).

2. *Ανάπτυξη δοκιμής για τις μνήμες FIFO* (*test development for FIFOs*).

3. *Ανάπτυξη δοκιμής για τα σφάλματα του πρωτοκόλλου επικοινωνίας* (*test development for protocol error handling*).

4. *Ανάπτυξη δοκιμής για τα σφάλματα της λογικής διασύνδεσης διαύλου* (*test development for the bus interface logic*).

3.7.1 Ανάπτυξη δοκιμής για τις καταστάσεις λειτουργίας

Το πρώτο βήμα της μεθοδολογίας είναι η εφαρμογή όλων των πιθανών σεναρίων λειτουργίας του περιφερειακού. Το βήμα αυτό ακολουθεί τις αρχές της *λειτουργικής επαλήθευσης* (*functional verification*) της σχεδίασης ενός περιφερειακού πυρήνα. Το πλήθος και το είδος των καταστάσεων λειτουργίας του περιφερειακού εξαρτώνται από το πρωτόκολλο επικοινωνίας και μεταξύ των περιφερειακών της κατηγορίας που μελετάμε υπάρχουν πολλές ομοιότητες. Αυτό το βήμα στοχεύει στο να εξετάσει τη λογική μετάδοσης και λήψης, οι οποίες επιτελούν τις σημαντικότερες λειτουργίες του πρωτοκόλλου επικοινωνίας και μαζί με τις μνήμες FIFO καταλαμβάνουν σχεδόν πάντα την μεγαλύτερη επιφάνεια υλικού ενός ενσωματωμένου περιφερειακού επικοινωνίας. Για παράδειγμα, στα δύο από τα τρία περιφερειακά επικοινωνίας που μελετούνται στην παρούσα διδακτορική διατριβή, στο UART και στο Ethernet, οι μηχανές αποστολής και λήψης (χωρίς τις μνήμες FIFO) καταλαμβάνουν το 16.80% και 13.56% του συνολικού αριθμού πυλών του πυρήνα, αντίστοιχα [27].

Μια κατάσταση λειτουργίας καθορίζει τις παραμέτρους του αντίστοιχου πρωτοκόλλου επικοινωνίας. Για παράδειγμα, το UART περιλαμβάνει έναν καταχωρητή ελέγχου των 8-bit που

καθορίζει τις παραμέτρους της ασύγχρονης μετάδοσης των δεδομένων: το *μήκος του χαρακτήρα* (*number of bits character*), τον *αριθμό των ψηφίων τέλους του μηνύματος* (*number of stop bits*), την *ενεργοποίηση των ψηφίων ισοτιμίας* (*parity enable*), την *ενεργοποίηση της άρτιας ισοτιμίας* (*even parity enable*), την δυνατότητα *μετάδοσης παύσης* (*break control*) κλπ. Ομοίως, το HDLC ενσωματώνει έναν καταχωρητή των 8-bit για τον καθορισμό της έναρξης αποστολής και λήψης ενός μηνύματος, την ενεργοποίηση ή απενεργοποίηση της αποστολής *ακολουθίας ελέγχου μηνύματος* (*Frame Check Sequence, FCS*), κλπ. Τέλος, το Ethernet ενσωματώνει έναν καταχωρητή των 16-bit ο οποίος καθορίζει τις παραμέτρους της επικοινωνίας. Για παράδειγμα, την παραγωγή *κυκλικού ελέγχου πλεονασμού* (*Cyclic Redundancy Check, CRC*), την *ημιαμφίδρομη η πλήρως αμφίδρομη επικοινωνία* (*half/full duplex*), κλπ. Η ανάπτυξης δοκιμής παράγει ένα σενάριο δοκιμής για κάθε πιθανό συνδυασμό των παραμέτρων μετάδοσης των περιφερειακών επικοινωνίας, εκτός από αυτές που δεν επιτρέπει το πρωτόκολλο.

Ένα σενάριο δοκιμής για μια κατάσταση λειτουργίας περιλαμβάνει τη μετάδοση και τη λήψη ενός πακέτου δεδομένων, έτσι ώστε οι αντίστοιχες λειτουργίες του πρωτοκόλλου της μετάδοσης και λήψης να ενεργοποιηθούν και τα ελαττώματα σε αυτές να ανιχνευτούν. Το σενάριο αυτό αποτελεί ένα σενάριο δοκιμής στο επίπεδο του συστήματος και δεν μπορεί να εφαρμοστεί άμεσα από οποιαδήποτε προσέγγιση αυτοδοκιμής με λογισμικό καθώς ο ενσωματωμένος επεξεργαστής δεν έχει πρόσβαση στις γραμμές μετάδοσης και λήψης της διασύνδεσης του περιφερειακού επικοινωνίας. Προκειμένου να είναι εφικτή η μετάδοση και λήψη ενός πακέτου δεδομένων, ενεργοποιούμε τον βρόχο επιστροφής του περιφερειακού μέσω του αντίστοιχου καταχωρητή ελέγχου.

Για παράδειγμα, ο Πίνακας 3.1 παρουσιάζει όλες τις πιθανές καταστάσεις λειτουργίας χωρισμένες σε ομάδες για τον ελεγκτή UART. Όταν τα bit 3, 6 και 7 έχουν τιμή “0” τότε η αποστολή καθορίζεται από τα bit 0-2 (2^3 = 8 καταστάσεις λειτουργίας), δηλαδή από το μήκος του χαρακτήρα και τον αριθμό των ψηφίων τέλους μηνύματος. Όταν τα bit 6 και 7 έχουν τιμή “0” και το bit 3 την τιμή 1 (ενεργοποίηση ψηφίων ισοτιμίας), τότε η αποστολή καθορίζεται από τα bit 0-5 (2^5 = 32 καταστάσεις λειτουργίας). Όταν το bit 6 έχει τιμή “1” (παύση λειτουργίας) τότε όλα τα υπόλοιπα bit είναι αδιάφοροι όροι και δεν πραγματοποιείται μετάδοση μηνύματος. Τέλος, όταν το bit 7 έχει τιμή “1” τότε αποκτάται πρόσβαση στον καταχωρητή διαίρεσης και δεν πραγματοποιείται αποστολή μηνύματος..

Number of bits [0-1]	Number of stop bits [2]	Parity Enable [3]	Even Parity Enable [4]	Stick Parity [5]	Break Control [6]	Divisor Latch Access [7]
00-11	0-1	0	X	X	0	0
00-11	0-1	1	0-1	0-1	0	0
X	X	X	X	X	1	X
X	X	X	X	X	X	1

Πίνακας 3.1 Καταχωρητής ελέγχου γραμμής (line control register) για τον ελεγκτή UART

Η διαδικασία παραγωγής της ρουτίνας αυτοδοκιμής για τις καταστάσεις λειτουργίας απεικονίζεται στην Εικόνα 3.8. Ο μηχανικός δοκιμής καθορίζει τα διαφορετικά σενάρια λειτουργίας του περιφερειακού. Στην συνέχεια, παράγεται ο τελικός κώδικα δοκιμής βάση ενός πρότυπου ψευδοκώδικα για τις καταστάσεις λειτουργίας.

Εικόνα 3.8 Δημιουργία κώδικα δοκιμής για τις καταστάσεις λειτουργίας.

Μια ρουτίνα αυτοδοκιμής για κάθε μια από τις καταστάσεις λειτουργίας εκτελεί τις ακόλουθες ενέργειες με στόχο την ενεργοποίηση και την ανίχνευση των ελαττωμάτων στις μηχανές αποστολής και λήψης του περιφερειακού επικοινωνίας:

1. Ενεργοποιεί την εκάστοτε κατάσταση λειτουργίας θέτοντας τους κατάλληλους καταχωρητές ελέγχου.

2. Ενεργοποιεί τον βρόχο επιστροφής μέσω του αντίστοιχου καταχωρητή ελέγχου.

3. Αποθηκεύει τα δεδομένα που πρόκειται να αποστείλει στην μνήμη FIFO του αποστολέα.

4. Ενεργοποιεί τη μετάδοση. Σε αυτό το σημείο, οι ρουτίνες αυτοδοκιμής περιμένουν έως ότου ολοκληρωθεί η μετάδοση και η λήψη των δεδομένων.

5. Τέλος, η ρουτίνα διαβάζει την μνήμη FIFO του παραλήπτη και τους καταχωρητές κατάστασης, οι οποίοι περιέχουν τις αποκρίσεις των μηχανών αποστολής και λήψης.

Το πρότυπο της ρουτίνας για την δοκιμή των καταστάσεων λειτουργίας ενός περιφερειακού επικοινωνίας παρουσιάζεται στην Εικόνα 3.9 με μορφή ψευδοκώδικα.

```
1.  Initialize General Registers
 # Application of Operating Mode
2.  Set Control Registers
3.  Activate Loopback Mode
 # Apply Test Data
4.  Write Transmit FIFO
5.  Transmit Frame
6.  Wait for Transmit Time Period
 # Capture Test Responses
7.  Read Receive FIFO
8.  Read Status Registers
```

Εικόνα 3.9 Πρότυπο ρουτίνας δοκιμής για τις καταστάσεις λειτουργίας

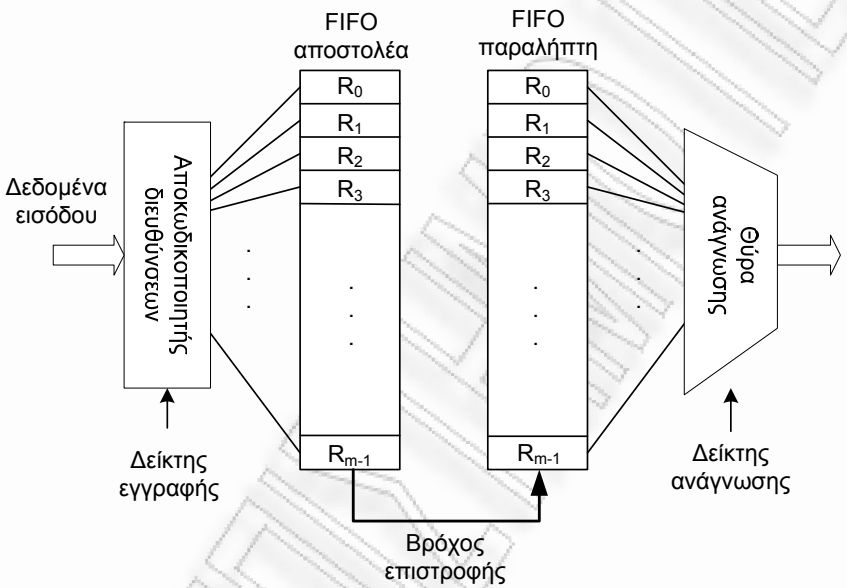
Η μετάδοση των πακέτων δεδομένων εκτελείται με τη μέγιστη δυνατή ταχύτητα και το μικρότερο δυνατό μήκος δεδομένων, προκειμένου να μειώσουμε τις *αδρανείς περιόδους* (*idle periods*) της εκάστοτε ρουτίνας, δηλαδή τα χρονικά διαστήματα που ο επεξεργαστής περιμένει να ολοκληρωθεί η αποστολή και λήψη των πακέτων δεδομένων και συνεπώς να μειωθεί ο συνολικός χρόνος εφαρμογής της δοκιμής. Εδώ πρέπει να σημειωθεί ότι οι αδρανείς χρονικές περίοδοι του επεξεργαστή καθορίζουν σε μεγάλο βαθμό το συνολικό χρόνο εφαρμογής της δοκιμής Συνεπώς, κρίνεται απαραίτητο να βρεθεί ένας αποτελεσματικός τρόπος εφαρμογής της δοκιμής μειώνοντας στο ελάχιστο τις αδρανείς περιόδους του επεξεργαστή. Θα επιστρέψουμε σε αυτό το ζήτημα σε επόμενη ενότητα.

3.7.2 Ανάπτυξη δοκιμής για τις μνήμες FIFO

Η πλειονότητα των περιφερειακών επικοινωνίας ενσωματώνει τουλάχιστον δύο μνήμες FIFO, μία σε κάθε κατεύθυνση επικοινωνίας, Οι προμηθευτές των περιφερειακών πυρήνων επικοινωνίας ενσωματώνουν συνήθως δομές μνήμης FIFO με προγραμματιζόμενο μέγεθος, έτσι ώστε οι σχεδιαστές των SoC να μπορούν να προσαρμόσουν το μέγεθος της μνήμης FIFO σύμφωνα με τις απαιτήσεις της εκάστοτε εφαρμογής. Οι μνήμες FIFO αποτελούν μια βασική παράμετρο σχεδίασης των πυρήνων επικοινωνίας καθώς καταλαμβάνουν σημαντική επιφάνεια της λογικής του πυρήνα. Για παράδειγμα οι μνήμες FIFO στους περιφερειακούς πυρήνες επικοινωνίας που μελετήσαμε, UART, HDLC και Ethernet, καταλαμβάνουν 34.92%, 87.13% και 40.46% του συνολικού αριθμού πυλών, αντίστοιχα.

Ο επεξεργαστής αποκτά πρόσβαση στις μνήμες FIFO του αποστολέα και του παραλήπτη εκτελώντας εντολές εγγραφής και ανάγνωσης, αντίστοιχα. Σε μερικές περιπτώσεις, το περιφερειακό μπορεί να ενσωματώνει έναν ελεγκτή DMA ο οποίος επιτρέπει την μετάδοση μεγάλου όγκου δεδομένων μεταξύ της μνήμης του συστήματος και της εσωτερικής μνήμης του περιφερειακού επικοινωνίας.

Για την δοκιμή των μνημών FIFO χρησιμοποιούμε ένα ντετερμινιστικό σύνολο δοκιμής το οποίο έχει προταθεί στην εργασία [95] για την δοκιμή του *αρχείου καταχωρητών* (*register file*) ενός επεξεργαστή.

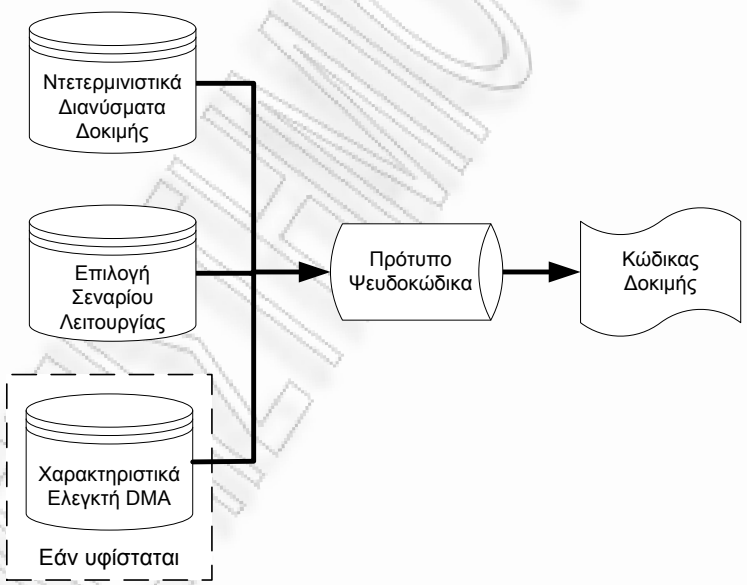


Εικόνα 3.10 Μια FIFO δομημένη σαν αρχείο καταχωρητών.

Ένα αρχείο καταχωρητών αποτελείται από τις ακόλουθες μονάδες: τον *αποκωδικοποιητή διευθύνσεων εγγραφής* (*write address decoder*), τη *δομή καταχωρητών* (*register array*) και μια θύρα ανάγνωσης που χρησιμοποιεί πολυπλέκτες (*multiplexer-based read port*). Το προτεινόμενο σύνολο δοκιμής για αρχείο καταχωρητών επιτυγχάνει ποσοστό κάλυψης ελαττωμάτων για ελαττώματα προσκόλλησης περισσότερο από 99.5%. Οι δύο μνήμες FIFO του αποστολέα και του παραλήπτη μπορούν να θεωρηθούν ως ένα ενιαίο αρχείο καταχωρητών με ανεξάρτητη πρόσβαση εγγραφής και ανάγνωσης από τον επεξεργαστή, όπως απεικονίζεται στην Εικόνα 3.10. Η βασική διαφορά είναι ότι η σειρά ανάγνωσης ή εγγραφής δεν μπορεί να διευθυνσιοδοτηθεί τυχαία από το σύνολο εντολών ενός επεξεργαστή (όπως στο αρχείο καταχωρητών του επεξεργαστή), αλλά από τους *δείκτες εγγραφής και ανάγνωσης* (*write and read FIFO pointers*). Όπως σε ένα κανονικό αρχείο καταχωρητών ενός επεξεργαστή, ένας αποκωδικοποιητής διευθύνσεων διαχειρίζεται τις εγγραφές στην μνήμη FIFO του αποστολέα

και μια πολυπλεγμένη θύρα ανάγνωσης χειρίζεται τις αναγνώσεις από την μνήμη FIFO του παραλήπτη. Ενεργοποιώντας το βρόχο επιστροφής του περιφερειακού, είναι δυνατόν να μεταδοθούν σειριακά τα περιεχόμενα της μνήμης FIFO του αποστολέα και να *ανακατευθυνθούν* (*redirect*) τα ίδια δεδομένα στην μνήμη FIFO του παραλήπτη, εφαρμόζοντας αποτελεσματικά το σύνολο δοκιμής και στις δύο μνήμες FIFO.

Η διαδικασίας παραγωγής του κώδικα δοκιμής για τις μνήμες FIFO ενός περιφερειακού επικοινωνίας απαιτεί την γνώση των ντετερμινιστικών διανυσμάτων δοκιμής, την επιλογή της κατάλληλης κατάστασης λειτουργίας, καθώς και τα χαρακτηριστικά του ελεγκτή DMA εάν αυτός ενσωματώνεται σε ένα περιφερειακό επικοινωνίας. Η διαδικασία παραγωγής του κώδικα δοκιμής παρουσιάζεται στην Εικόνα 3.11, όπου η επιλογή του κατάλληλου σεναρίου καθορίζεται με γνώμονα την υψηλότερη ταχύτητα μετάδοσης του μηνύματος και το συνολικό μήκος αποστολής μηνύματος, ίσο με το μήκος των ντετερμινιστικών δεδομένων.



Εικόνα 3.11 Δημιουργία κώδικα δοκιμής για τις μνήμες FIFO

Μια ρουτίνα δοκιμής για ένα ζεύγος μνημών FIFO εκτελεί τις ακόλουθες ενέργειες:

1. Ενεργοποιεί τον βρόχο επιστροφής μέσω του αντίστοιχου καταχωρητή ελέγχου στο εκάστοτε περιφερειακό επικοινωνίας.

2. Γράφει τα ντετερμινιστικά διανύσματα δοκιμής στη μνήμη FIFO του αποστολέα.

3. Ενεργοποιεί τη μετάδοση. Έπειτα οι ρουτίνες δοκιμής περιμένουν μέχρι να ολοκληρωθεί η μετάδοση. Τα ντετερμινιστικά διανύσματα δοκιμής εφαρμόζονται διαδοχικά στην μνήμη FIFO του παραλήπτη μέσω της πορείας του βρόχου επιστροφής.

4. Τέλος, ο επεξεργαστής συλλέγει τις αποκρίσεις της δοκιμής διαβάζοντας την μνήμη FIFO του παραλήπτη.

Όταν ένα περιφερειακό περιέχει ελεγκτή DMA, ο επεξεργαστής αποθηκεύει τα διανύσματα δοκιμής και διαβάζει τις αποκρίσεις της δοκιμής από την μνήμη του συστήματος. Αυτό συμβαίνει διότι ο ελεγκτής DMA είναι υπεύθυνος για την μεταφορά των δεδομένων δοκιμής μεταξύ της μνήμης του συστήματος και της μνήμης FIFO του περιφερειακού επικοινωνίας.

Η δοκιμή του αρχείου m καταχωρητών ενός επεξεργαστή απαιτεί την εφαρμογή $2m$ διανυσμάτων δοκιμής. Στην περίπτωση των μνημών FIFO, αυτό αντιστοιχεί στη μετάδοση δύο πακέτων δεδομένων μεγέθους m. Η ντετερμινιστική προσέγγιση δοκιμής επιτυγχάνει πολύ υψηλό ποσοστό κάλυψης ελαττωμάτων (περισσότερο από 99%) για τις δομές μνημών FIFO των πυρήνων επικοινωνίας. Ένα πρότυπο ρουτίνας με τη μορφή ψευδοκώδικα για την δοκιμή των μνημών FIFO παρουσιάζεται στην Εικόνα 3.12.

```
1.  Initialize General Registers
2.  Activate Loopback Mode
 # Apply Test Set #1
 # Patterns={0x0000, 0xFFFF, 0xFFFF, 0x0000,…}
3.  Write Transmit FIFO
4.  Transmit Frame
5.  Wait for Transmit Time Period
 # Capture Test Responses
6.  Read Receive FIFO
 # Apply Test Set #2
 # Patterns={0xFFFF, 0x0000, 0x0000, 0xFFFF,…}
7.  Transmit Frame
8.  Wait for Transmit Time Period
 # Capture Test Responses
9.  Read Receive FIFO
```

Εικόνα 3.12 Πρότυπο ρουτίνας για την δοκιμή των μνημών FIFO

Σε αυτό το σημείο πρέπει να σημειωθεί ότι στην παρούσα διατριβή εξετάζεται η περίπτωση για την υλοποίηση της μνήμης FIFO, η οποία χρησιμοποιεί flip-flop και τόσο η μνήμη όσο και η λογική ελέγχου της μνήμης FIFO γίνονται σύνθεση μαζί με το υπόλοιπο κύκλωμα χρησιμοποιώντας μια τυποποιημένη βιβλιοθήκη πυλών. Στην περίπτωση που η μνήμη FIFO

γίνει σύνθεση χρησιμοποιώντας *κελιά μνήμης* (*memory cells*) μιας βιβλιοθήκης, τότε η ρουτίνα αυτοδοκιμής πρέπει να παραχθεί, σύμφωνα με έναν από τους γνωστούς αλγόριθμους δοκιμής μνήμης FIFO [129], [130], [131], ώστε να καλύπτει και τα αντίστοιχα μοντέλα ελαττώματος μνήμης.

3.7.3 Ανάπτυξη δοκιμής για τη λογική χειρισμού των σφαλμάτων επικοινωνίας

Το επόμενο βήμα της μεθοδολογίας στοχεύει στην ανίχνευση ελαττωμάτων στη *λογική χειρισμού των σφαλμάτων* (*error handling logic*) μετάδοσης και λήψης του περιφερειακού επικοινωνίας. Τα σφάλματα αυτά εξαρτώνται από το πρωτόκολλο επικοινωνίας και μπορούν να προκαλέσουν διαφορετικούς τύπους δυσλειτουργιών στη μετάδοση ή την λήψη ενός πακέτου δεδομένων. Για παράδειγμα, ένα ελάττωμα μπορεί να οδηγήσει στην *ακύρωση της μετάδοσης ενός πακέτου* (*abortion of the packet transmission*), την *απόρριψη του πακέτου από τον παραλήπτη* (*discard of the packet at the receiver*), ή την *ανίχνευση και τη διόρθωση ενός λάθους του μηνύματος* (*detection and correction of error*).

Μια προσέγγιση βασισμένη στο λογισμικό δεν μπορεί να ενεργοποιήσει τέτοια σφάλματα που στην ουσία προκαλούνται από *φυσικά ελαττώματα* (*physical faults*) στη γραμμή μεταδοσης. Η μεθοδολογία μας προσπαθεί να "μιμηθεί" αυτά τα φυσικά ελαττώματα, όμοια με μια μεθοδολογία επαλήθευσης της σχεδίασης, απενεργοποιώντας τον βρόχο επιστροφής κατά τη διάρκεια της μετάδοσης των δεδομένων, προκαλώντας έτσι την διακοπή της συνεχούς εισερχόμενης ροής δεδομένων και κατά συνέπεια την μη ορθή λήψη τους. Ανάλογα με το χρονικό σημείο και τη διάρκεια της απενεργοποίησης του βρόχου επιστροφής, διαφορετικοί τύποι σφαλμάτων μπορούν να εμφανιστούν.

Για παράδειγμα, στην περίπτωση του ελεγκτή UART μπορούμε να προκαλέσουμε ένα *σφάλμα πλαισίου* (*framing error*) το οποίο θα απορρίψει το λαμβανόμενο χαρακτήρα εισάγοντας ένα σφάλμα κατά τη διάρκεια της μετάδοσης του *τελικού ψηφίου* (*stop bit*). Ένα άλλο παράδειγμα αποτελεί η λήψη ενός λανθασμένου μηνύματος, όπου στην περίπτωση του ελεγκτή Ethernet ή του ελεγκτή HDLC υποδεικνύεται από τον *έλεγχο πλεονασμού* (*CRC*), ή από την *ακολουθία ελέγχου μηνύματος* (*FCS*), αντίστοιχα. Τέτοια σφάλματα είναι δυνατόν να προκληθούν με την απενεργοποίηση του βρόχου επιστροφής για μια ή περισσότερες περιόδους μετάδοσης των ψηφίων κατά τη διάρκεια της μετάδοσης του μηνύματος. Ο Πίνακας 3.2 παρουσιάζει όλα τα πιθανά σφάλματα που μπορούν να εμφανιστούν κατά την λήψη ενός μηνύματος για τον ελεγκτή UART.

Πιθανά Σφάλματα	Περιγραφή
Σφάλμα υπερχείλισης (Overrun Error)	Το σφάλμα αυτό εμφανίζεται όταν η μνήμη FIFO είναι πλήρης και ένας χαρακτήρας έχει ληφθεί στον προσωρινό καταχωρητή του παραλήπτη. Εάν ένας ακόμα χαρακτήρας αρχίσει να λαμβάνεται τότε το περιεχόμενο του καταχωρητή θα αντικατασταθεί, ενώ η μνήμη FIFO θα μείνει ανέπαφη.
Σφάλμα Ισοτιμίας (Parity Error)	Ο χαρακτήρας που βρίσκεται στην κορυφή της μνήμης FIFO παραλήφθηκε με λανθασμένη τιμή στο bit ισοτιμίας. Το σφάλμα αυτό εμφανίζεται μόνο εάν η κατάσταση λειτουργίας του περιφερειακού ενεργοποιεί την μετάδοση του ψηφίου ισοτιμίας.
Σφάλμα Πλαισίου (Framing Error)	Ο χαρακτήρας στην κορυφή της μνήμη FIFO του παραλήπτη δεν έχει σωστό ψηφίο τέλους μηνύματος.

Πίνακας 3.2 Πιθανά σφάλματα λειτουργίας του ελεγκτή UART

Μια ρουτίνα αυτοδοκιμής για τη λογική του πρωτοκόλλου επικοινωνίας εκτελεί τις ακόλουθες ενέργειες:

1. Ενεργοποιεί τον βρόχο επιστροφής μέσω του καταχωρητή λειτουργίας ή μέσω του αντίστοιχου καταχωρητή ελέγχου στο εκάστοτε περιφερειακό επικοινωνίας.

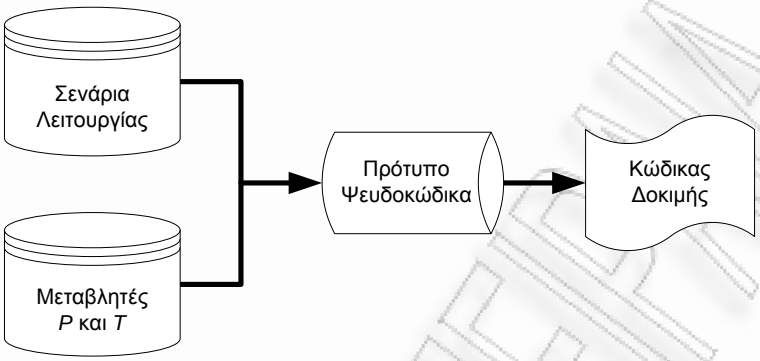
2. Γράφει τη μνήμη FIFO του αποστολέα και ενεργοποιεί τη μετάδοση.

3. Σε χρόνο T μετά από την έναρξη της μετάδοσης η ρουτίνα αυτοδοκιμής απενεργοποιεί τον βρόχο επιστροφής για χρονική περίοδο P.

4. Ενεργοποιεί τον βρόχο επιστροφής και περιμένει μέχρι την ολοκλήρωση της μετάδοσης. Οι χρόνοι T και P που μπορούν να ποικίλουν μεταξύ της χρονικής διάρκειας που απαιτείται για την μετάδοση ενός ψηφίου και την μετάδοση ενός πακέτου.

5. Τέλος, ο επεξεργαστής διαβάζει τις *σημαίες σφάλματος* (*error flags*) από τους καταχωρητές κατάστασης του περιφερειακού.

Για την παραγωγή του κώδικα δοκιμής για τη λογική του πρωτοκόλλου επικοινωνίας, απαιτούνται τόσο τα εναλλακτικά σενάρια λειτουργίας του περιφερειακού, όσο και οι χρονικές παράμετροι P και T όπως παρουσιάζεται στην Εικόνα 3.13. Οι χρονικές παράμετροι P και T, διαμορφώνουν τα εναλλακτικά σφάλματα κατά την μετάδοση η λήψη ενός μηνύματος, χρησιμοποιώντας το πρότυπο ψευδοκώδικα ώστε να δημιουργηθούν τα διαφορετικά σενάρια ρουτινών για την παραγωγή των σφαλμάτων επικοινωνίας.



Εικόνα 3.13 Δημιουργία κώδικα δοκιμής για τη λογική του πρωτοκόλλου επικοινωνίας

Ένα πρότυπο ρουτίνας με τη μορφή ψευδοκώδικα για την ανίχνευση των σφαλμάτων του πρωτόκολλου επικοινωνίας απεικονίζεται στην Εικόνα 3.14.

```
1.   Initialize General Registers
2.   Activate Loopback Mode
  # Apply Test Data
3.   Write Transmit FIFO
4.   Transmit Frame
  # Activation of a Specific Error Type
5.   Wait for T
6.   De-activate Loopback Mode
7.   Wait for P
8.   Reactivate Loopback Mode
9.   Wait for remaining Transmit Time Period
  # Capture Test Responses
10.  Read Status Registers
```

Εικόνα 3.14 Πρότυπο ρουτίνας για την της λογικής του πρωτοκόλλου επικοινωνίας

3.7.4 Ανάπτυξη δοκιμής για τον δίαυλο επικοινωνίας

Τα σύγχρονα SoC ενσωματώνουν έναν ή περισσότερους διαύλους επικοινωνίας ώστε να μεγιστοποιήσουν το *εύρος μετάδοσης δεδομένων* (*data bandwidth*) μεταξύ του επεξεργαστή, των μνημών και των περιφερειακών συσκευών. Μεταξύ των πιο δημοφιλών διαύλων είναι ο δίαυλος CoreConnect της IBM [132], ο δίαυλος AMBA της ARM [133], και ο δίαυλος ανοικτού κώδικα Wishbone [24].

Η προτεινόμενη μεθοδολογία χειρίζεται χωριστά τη λογική διασύνδεσης του διαύλου καθώς δεν μπορεί να δοκιμαστεί επαρκώς με την εκτέλεση της δοκιμής των υπολοίπων μονάδων

του περιφερειακού επικοινωνίας. Τα προβλήματα δοκιμής της λογικής των διαύλων επικοινωνίας οφείλονται κυρίως στη *λογική αποκωδικοποίησης διευθύνσεων* (*address decoding logic*). Κάθε ενσωματωμένο σύστημα έχει μια προκαθορισμένη περιοχή μνήμης στην οποία αντιστοιχίζονται (*mapping*) οι διευθύνσεις των εσωτερικών καταχωρητών του περιφερειακού. Επομένως, όλες οι προσπελάσεις (εγγραφές – αναγνώσεις) των καταχωρητών ενός πυρήνα περιλαμβάνουν διευθύνσεις μνήμης που προκύπτουν από το άθροισμα της διεύθυνσης βάσης του πυρήνα (base offset) και της μετατόπισης του αντίστοιχου καταχωρητή (register offset). Ωστόσο, η διέγερση των ελαττωμάτων της λογικής που χειρίζεται τις διευθύνσεις των καταχωρητών και της λογικής του διαύλου απαιτεί την εφαρμογή διανυσμάτων δοκιμής που θέτουν όλα τα ψηφία των διευθύνσεων στις τιμές “0” και “1”. Το πρόβλημα γίνεται εντονότερο στην περίπτωση που ο περιφερειακός πυρήνας ενσωματώνει έναν ελεγκτή DMA. Για τις μεταφορές δεδομένων μέσω του ελεγκτή DMA, προκαθορίζονται συγκεκριμένες περιοχές στη μνήμη δεδομένων από όπου γίνεται η φόρτωση και η αποθήκευση των πακέτων της μετάδοσης και λήψης, αντίστοιχα.

Η μεθοδολογία υιοθετεί τη λύση που παρουσιάστηκε στην εργασία [97] για τη δοκιμή της λογικής που χειρίζεται τις διευθύνσεις ενός επεξεργαστή με *διοχέτευση* (*pipeline*). Στην εργασία [97] προτάθηκε η *διαμέριση* (*partitioning*) ενός δεδομένου προγράμματος αυτοδοκιμής με λογισμικό σε πολλά τμήματα κώδικα. Τα τμήματα του κώδικα αυτοδοκιμής αποθηκεύονται και εκτελούνται από διαφορετικές περιοχές μνήμης. Η λύση προσαρμόστηκε στις ανάγκες δοκιμής της προτεινόμενης μεθοδολογίας. Οι διαφορετικές περιοχές της ενσωματωμένης μνήμης δεδομένων ορίζονται ως περιοχή του ελεγκτή DMA του περιφερειακού και για κάθε μία εκτελείται μια μετάδοση μηνύματος. Κατά συνέπεια, οι μεταφορές δεδομένων μέσω του ελεγκτή DMA που εκτελούνται για κάθε περιοχή μνήμης, περιλαμβάνουν διευθύνσεις με συμπληρωματικές τιμές στα περισσότερο σημαντικά bit.

Στην περίπτωση που το SoC παρέχει *εξωτερική διασύνδεση μνήμης* (*external memory interface*) ώστε να υποστηρίξει επέκταση της μνήμης (το οποίο αποτελεί τη συνήθη περίπτωση), η προτεινόμενη λύση μπορεί να επεκταθεί με τον καθορισμό *εικονικών περιοχών μνήμης* (*virtual memory regions*) του ελεγκτή DMA. Κατά συνέπεια, είναι δυνατόν να οριστούν περιοχές εικονικής μνήμης, που εκτείνονται σε ολόκληρη την περιοχή μνήμης του συστήματος, σαν περιοχές του ελεγκτή DMA. Στην συνέχεια, χρησιμοποιώντας μια *μονάδα αντιστοίχησης της μνήμης* (*memory mapping module*), αντιστοιχούμε τις εικονικές διευθύνσεις σε πραγματικές διευθύνσεις της εξωτερικής μνήμης του συστήματος. Η μονάδα αντιστοίχησης, που δεν αποτελεί μέρος του SoC, αλλά χρησιμοποιείται μόνο για λόγους προσομοίωσης των ελαττωμάτων, τοποθετείται μεταξύ του εξωτερικού δίαυλου μνήμης του συστήματος και της εξωτερικής μνήμης.

Η εφαρμογή της μεθοδολογίας με την χρήση μονάδας αντιστοίχησης απεικονίζεται στην Εικόνα 3.15, όπου οι εικονικές διευθύνσεις 0x00000000–0xFFFFFFFF αντιστοιχούν στις

διευθύνσεις 0x1000–0x2000 του ελεγκτή DMA. Σύμφωνα με τα παραπάνω, οι εικονικές διευθύνσεις του περιφερειακού επικοινωνίας αντιστοιχίζονται σε πραγματικές διευθύνσεις της εξωτερικής μνήμης του συστήματος και για κάθε μία από αυτές δημιουργείται ένα σενάριο αποστολής και λήψης μηνύματος.

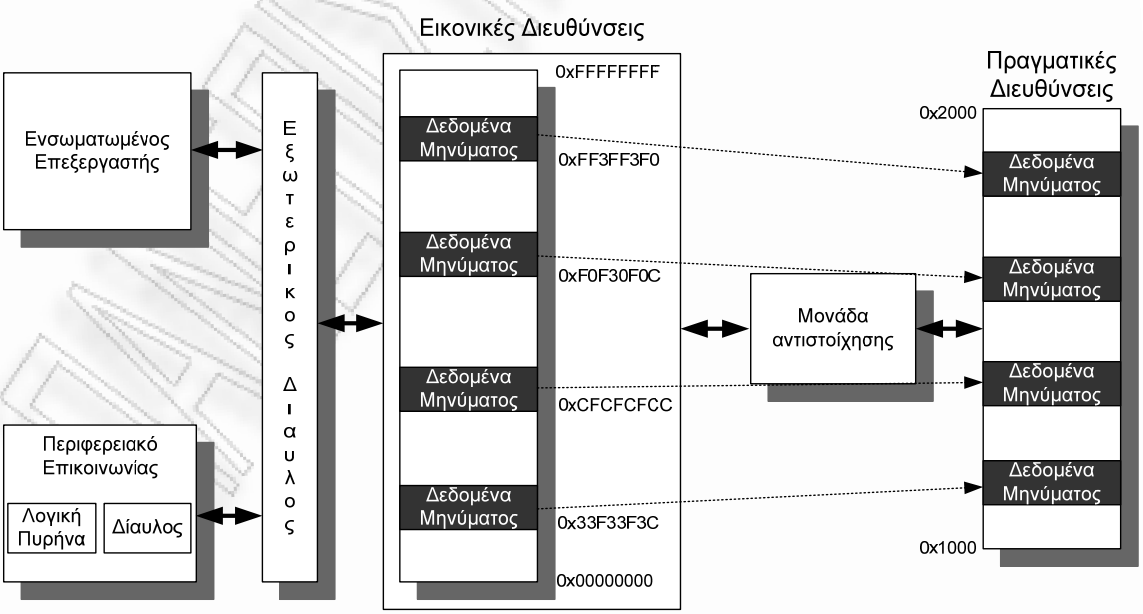
Για την δημιουργία του κώδικα δοκιμής του διαύλου επικοινωνίας, απαιτούνται τα χαρακτηριστικά του ελεγκτή DMA και οι εικονικές διευθύνσεις της μονάδας αντιστοίχησης όπως παρουσιάζεται στην Εικόνα 3.16. Μια ρουτίνα δοκιμής για τη λογική των διαύλων εκτελεί τις ακόλουθες ενέργειες:

1. Ρυθμίζει το περιφερειακό επικοινωνίας σύμφωνα με μία εικονική περιοχή μνήμης του ελεγκτή DMA.

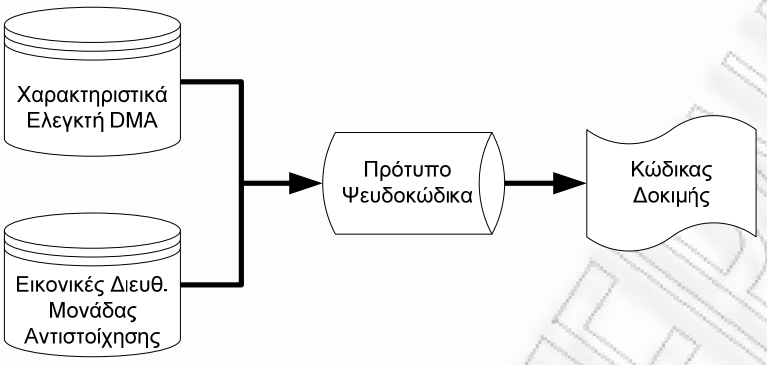
2. Αποθηκεύει το πακέτο μετάδοσης στη μνήμη του συστήματος.

3. Ενεργοποιεί τον βρόχο επιστροφής μέσω των καταχωρητών ελέγχου και εκκινεί την μετάδοση του μηνύματος. Σε αυτό το σημείο, οι ρουτίνες δοκιμής περιμένουν μέχρι την ολοκλήρωση της αποστολής του μηνύματος.

4. Τέλος, ο επεξεργαστής λαμβάνει τις αποκρίσεις δοκιμής από τη μνήμη δεδομένων. Ο ελεγκτής DMA εκτελεί τη μεταφορά των δεδομένων μεταξύ της μνήμης και των μνημών FIFO του περιφερειακού.



Εικόνα 3.15 Χρήση μονάδας αντιστοίχησης για την ανίχνευση των ελαττωμάτων του διαύλου επικοινωνίας.



Εικόνα 3.16 Δημιουργία κώδικα δοκιμής για ελαττώματα του διαύλου επικοινωνίας.

Ένα πρότυπο ρουτίνας με την μορφή ψευδοκώδικα για την ανίχνευση των ελαττωμάτων του πρωτόκολλου επικοινωνίας απεικονίζεται στην Εικόνα 3.17

```
1.  Initialize General Registers
    Activate Loopback Mode
 # Define virtual DMA regions
 # Define virtual DMA regions
2.  Configure the DMA region of Tx FIFO
3.  Configure the DMA region of Rx FIFO
 # Apply Test Data
4.  Store the Transmit Frame to Memory
5.  Transmit Frame
6.  Wait for Transmit Time Period
 # Capture Test Responses
8.  Load the Receive Frame from Memory
```

Εικόνα 3.17 Πρότυπο ρουτίνας για την δοκιμή του δίαυλου επικοινωνίας

3.8 Σύγκριση μεθοδολογιών

Η υβριδική μεθοδολογία που παρουσιάζεται στο δεύτερο μέρος αυτού του κεφαλαίου (Ενότητα 3.9) της διατριβής βασίζεται σε δύο προηγούμενες προσεγγίσεις αυτοδοκιμής με λογισμικό για περιφερειακά επικοινωνίας σε ένα SoC, την αυτόματη [127] και την ντετερμινιστική [26], [27].

Στην ερευνητική εργασία [127] όπως είδαμε και στην ανασκόπηση της βιβλιογραφίας, έχει παρουσιαστεί μια αυτοματοποιημένη μεθοδολογία για την παραγωγή των προγραμμάτων δοκιμής για τον ελεγκτή UART και για τον ελεγκτή PIA που ενσωματώνονται σε SoC. Η

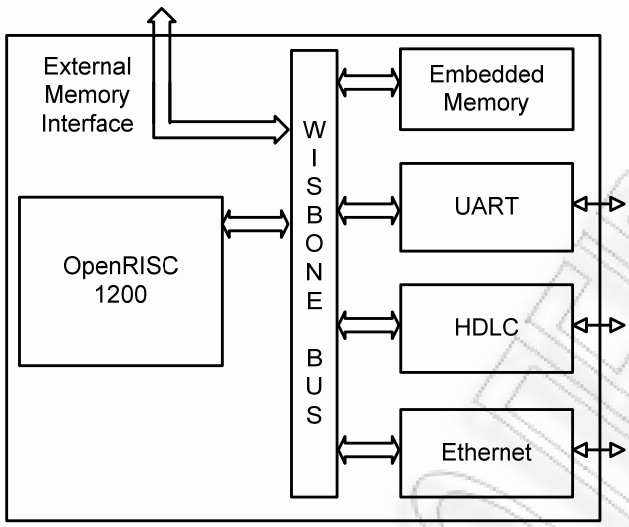
μεθοδολογία είναι βασισμένη στην εκμετάλλευση του συσχετισμού μεταξύ των υψηλού επιπέδου μετρικών και της κάλυψης ελαττωμάτων σε κάθε ολοκληρωμένο κύκλωμα. Χρησιμοποιεί ένα εξελικτικό αλγόριθμο ο οποίος παράγει τα προγράμματα δοκιμής με στόχο την μεγιστοποίηση των επιλεγμένων μετρικών κάλυψης σε μια υψηλού επιπέδου περιγραφή ενός SoC. Ο εξελικτικός αλγόριθμος παράγει τα προγράμματα δοκιμής σύμφωνα με τις πληροφορίες για το περιφερειακό πυρήνα και τη σύνταξη εντολών του επεξεργαστή (βιβλιοθήκη περιορισμών). Η διαδικασία παραγωγής βασίζεται σε ανατροφοδότηση, η οποία προέρχεται από έναν προσομοιωτή υψηλού-επιπέδου για κάθε πρόγραμμα δοκιμής και ολοκληρώνεται είτε όταν οι τιμές των στοχοθετημένων υψηλού επιπέδου μετρικών είναι ικανοποιητικές, είτε μετά από μια προκαθορισμένη χρονική προθεσμία. Όταν υπάρχει συσχετισμός μεταξύ των υψηλού επιπέδου μετρικών και της κάλυψης ελαττωμάτων, η μεθοδολογία είναι σε θέση να επιτύχει αποδεκτή κάλυψη ελαττωμάτων με ιδιαίτερα χαμηλή ανθρώπινη παρέμβαση. Το μέγεθος κώδικα των προγραμμάτων δοκιμής δεν είναι το καλύτερο δυνατό, δεδομένου ότι η προσέγγιση σπάνια είναι ικανή να παράγει προγράμματα αυτοδοκιμής *βασισμένα σε βρόχο* (*loop-based*).

Στις ερευνητικές εργασίες [26], [27] όπως είδαμε και στην Ενότητα 3.7, παρουσιάστηκε μια ντετερμινιστική μεθοδολογία για την εφαρμογή της δοκιμής βασισμένης στο λογισμικό για τους περιφερειακούς πυρήνες επικοινωνίας. Σύμφωνα με αυτή την προσέγγιση, τα διανύσματα δοκιμής για τις μονάδες του περιφερειακού πυρήνα επικοινωνίας προυπολογίζονται. Τα διανύσματα δοκιμής φορτώνονται στην ενσωματωμένη μνήμη δεδομένων του ολοκληρωμένου, ή αναπαράγονται κατά τη διάρκεια της εκτέλεσης του προγράμματος δοκιμής. Συνεπώς, τα διανύσματα δοκιμής μπορούν να εφαρμοστούν με δύο διαφορετικούς τρόπους:

1. Ως συμπαγές *βασισμένο σε βρόχο* (*loop-based*) πρόγραμμα δοκιμής που προσκομίζει τα διανύσματα δοκιμής από τη μνήμη δεδομένων.

2. Ως *ξετυλιγμένο* (*unroll-based*) πρόγραμμα δοκιμής όπου τα διανύσματα αναπαράγονται απευθείας με εντολές του επεξεργαστή.

Η μεθοδολογία αυτή επιτυγχάνει υψηλό ποσοστό κάλυψης ελαττωμάτων, με την χρήση ενός μικρού προγράμματος δοκιμής, τόσο ως προς τα δεδομένα δοκιμής (ντετερμινιστικά διανύσματα), όσο και ως προς τον κώδικα του προγράμματος. Δυστυχώς, η ντετερμινιστική φύση αυτής της προσέγγισης απαιτεί υψηλή ανθρώπινη συμμετοχή για τη συγγραφή του τελικού προγράμματος δοκιμής.



Εικόνα 3.18 Διάγραμμα SoC που χρησιμοποιήθηκε στα πειραματικά αποτελέσματα

Για την ποσοτική σύγκριση των δύο μεθοδολογιών, εκτελέσαμε πειράματα σε ένα SoC (Εικόνα 3.18) το οποίο βασίζεται σε έναν ενσωματωμένο επεξεργαστή με *διοχέτευση* (*pipeline*), τον OpenRISC 1200 [24]. Το σύστημα περιλαμβάνει 64KB ενσωματωμένης μνήμης RAM και τρία τυπικά παραδείγματα περιφερειακών επικοινωνίας: UART, HDLC και Ethernet. Ο δίαυλος επικοινωνίας του SoC βασίζεται στον δίαυλο ανοικτού κώδικα Wishbone [24].

Η πολυπλοκότητα των τριών πυρήνων επικοινωνίας ποικίλλει, από τον απλό ελεγκτή UART, έως τον πολύπλοκο ελεγκτή Ethernet και η μελέτη τους επιτρέπει να επισημάνουμε τα μειονεκτήματα και τα πλεονεκτήματα των δύο μεθοδολογιών. Η πολυπλοκότητα των πυρήνων εξαρτάται κυρίως από το πρωτόκολλο επικοινωνίας, ενώ το μέγεθός τους επηρεάζεται κατά μεγάλο ποσοστό από τις παραμέτρους των ενσωματωμένων δομών μνήμης FIFO. Οι παράμετροι των μνημών FIFO για τους τρεις πυρήνες επικοινωνίας επιλέχθηκαν ως εξής: για τον ελεγκτή UART, 8 bit μήκος λέξης με βάθος μνήμης FIFO ίσο με 16, για τον ελεγκτή HDLC και τον ελεγκτή Ethernet, 32 bit μήκος λέξης με βάθος FIFO ίσο με 32. Ο Πίνακας 3.3 παρουσιάζει τα μεγέθη (ως προς τον αριθμό των πυλών και τον αριθμό των ελαττωμάτων προσκόλλησης) τόσο του ενσωματωμένου επεξεργαστή όσο και των περιφερειακών επικοινωνίας του SoC. Το σύστημα έχει γίνει σύνθεση με μια τεχνολογική βιβλιοθήκη των 0.35μm, ενώ η συχνότητα λειτουργίας του SoC είναι 102MHz.

Πυρήνας	Αριθμός πυλών	Αριθμός Ελαττωμάτων Προσκόλλησης (stuck-at)
UART	7,771	20,558
HDLC	22,010	57,868
Ethernet	35,617	77,574
OpenRISC	35,657	85,224
Σύνολο	**101,055**	**241,224**

Πίνακας 3.3 Στατιστικά σύνθεσης του συστήματος σε ολοκληρωμένο

Για να μελετηθούν τα πλεονεκτήματα και τα μειονεκτήματα των δύο προσεγγίσεων αξιολογήθηκαν προσεκτικά τα αποτελέσματα και στους τρεις πυρήνες επικοινωνίας του SoC που χρησιμοποιήθηκαν. Οι ακόλουθοι πίνακες συνοψίζουν τα συγκριτικά αποτελέσματα της μελέτης μας και για τις δύο προσεγγίσεις.

Ο Πίνακας 3.4 συγκρίνει τα αποτελέσματα προσομοίωσης ελαττωμάτων προσκόλλησης για την αυτόματη και την ντετερμινιστική προσέγγιση. Η ντετερμινιστική προσέγγιση επιτυγχάνει ποσοστό κάλυψης ελαττωμάτων μεγαλύτερο από 93% για ελαττώματα προσκόλλησης σε όλους τους πυρήνες επικοινωνίας, ενώ τα αποτελέσματα ποσοστού κάλυψης ελαττωμάτων της αυτόματης προσέγγισης είναι αρκετά χαμηλότερα, ειδικά για τον πυρήνα HDLC όπου ο συσχετισμός μεταξύ των υψηλού επιπέδου μετρικών και της κάλυψης ελαττωμάτων είναι μικρός. Το παραπάνω συμπέρασμα για τον πυρήνα HDLC επιβεβαιώνεται από τις υψηλές των μετρικών κάλυψης υψηλού επιπέδου (Πίνακας 3.5) που επιτυγχάνονται από την αυτόματη προσέγγιση, την στιγμή που η αντίστοιχη κάλυψη ελαττωμάτων είναι μόλις 68.65%.

Πυρήνας	Ποσοστό Κάλυψης Ελαττωμάτων (%)	
	Αυτόματη	**Ντετερμινιστική**
UART	86.35	**93.92**
HDLC	68.65	**97.34**
Ethernet	86.57	**93.39**

Πίνακας 3.4 Σύγκριση ποσοστού κάλυψης ελαττωμάτων

Ο Πίνακας 3.5 παρουσιάζει τις τιμές των υψηλού επιπέδου μετρικών που επιτυγχάνει η αυτόματη προσέγγιση [127] για κάθε εξεταζόμενο πυρήνα. Οι υψηλού επιπέδου μετρικές μπορούν να καθοδηγήσουν αποτελεσματικά τη διαδικασία παραγωγής του κώδικα δοκιμής, με κύριο μειονέκτημα το σχετικά χαμηλό ποσοστό κάλυψη ελαττωμάτων (Πίνακας 3.4, δεύτερη στήλη) για όλους τους περιφερειακούς πυρήνες επικοινωνίας.

Πυρήνας	Κάλυψη Εναλλαγών	Κάλυψη Εκφράσεων	Κάλυψη Συνθηκών	Κάλυψη Διακλαδώσεων	Κάλυψη Δηλώσεων
UART	90.0	92.0	95.5	97.2	97.3
HDLC	96.2	96.4	94.4	97.5	95.8
Ethernet	87.9	99.4	94.5	97.8	97.7

Πίνακας 3.5 Μετρικά κάλυψης της αυτόματης προσέγγισης για τα τρία περιφερειακά.

Ο Πίνακας 3.6 συγκρίνει το χρόνο ανάπτυξης του προγράμματος δοκιμής των δύο προσεγγίσεων για τους τρεις περιφερειακούς πυρήνες του SoC. Ο χρόνος ανάπτυξης δοκιμής για την ντετερμινιστική προσέγγιση είναι περίπου 30 φορές μεγαλύτερος από αυτόν της αυτόματης προσέγγισης. Η μεγάλη αυτή διαφορά οφείλεται στο χρόνο που απαιτείται για τη χειροκίνητη παραγωγή του προγράμματος δοκιμής. Προφανώς, οι αριθμοί που αναφέρει ο Πίνακας 3.6 ποικίλλουν ανάλογα με την πείρα του μηχανικού δοκιμής, αλλά η διαφορά στο χρόνο ανάπτυξης δοκιμής μεταξύ των δύο προσεγγίσεων είναι σαφής.

Πυρήνας	Χρόνος Ανάπτυξης Δοκιμής (~Μέρες)	
	Αυτόματη	Ντετερμινιστική
UART	**1**	30
HDLC	**0.5**	15
Ethernet	**2**	45

Πίνακας 3.6 Σύγκριση χρόνου ανάπτυξης του κώδικα δοκιμής

Ο Πίνακας 3.7 συνοψίζει τα χαρακτηριστικά των δύο προσεγγίσεων. Στα πλεονεκτήματα της αυτόματης προσέγγισης συγκαταλέγονται ο μικρός χρόνος ανάπτυξης του προγράμματος δοκιμής και η μικρή ανθρώπινη συμμετοχή στην συγγραφή του κώδικα δοκιμής. Τα μειονεκτήματα της αυτόματης προσέγγισης είναι το χαμηλό ή πολύ χαμηλό ποσοστό κάλυψης ελαττωμάτων και ο μεγάλος σε μέγεθος κώδικας δοκιμής. Στον αντίποδα η ντετερμινιστική προσέγγιση εγγυάται υψηλό ποσοστό κάλυψης ελαττωμάτων, με μικρότερα σε μέγεθος προγράμματα δοκιμής, αλλά απαιτεί ιδιαίτερα μεγάλο χρόνο ανάπτυξης του προγράμματος δοκιμής και υψηλή ανθρώπινη συμμετοχή για την συγγραφή του κώδικα.

Χαρακτηριστικά	Αυτόματη	Ντετερμινιστική
Κάλυψη Ελαττωμάτων	Χαμηλή	**Υψηλή**
Μέγεθος Κώδικα Δοκιμής	Μεγάλο	**Μικρό**
Χρόνος Ανάπτυξης	**Μικρός (Μέρες)**	Μεγάλος (Μήνες)
Ανθρώπινη Συμμετοχή	**Χαμηλή**	Υψηλή

Πίνακας 3.7 Σύγκριση των δύο μεθοδολογιών

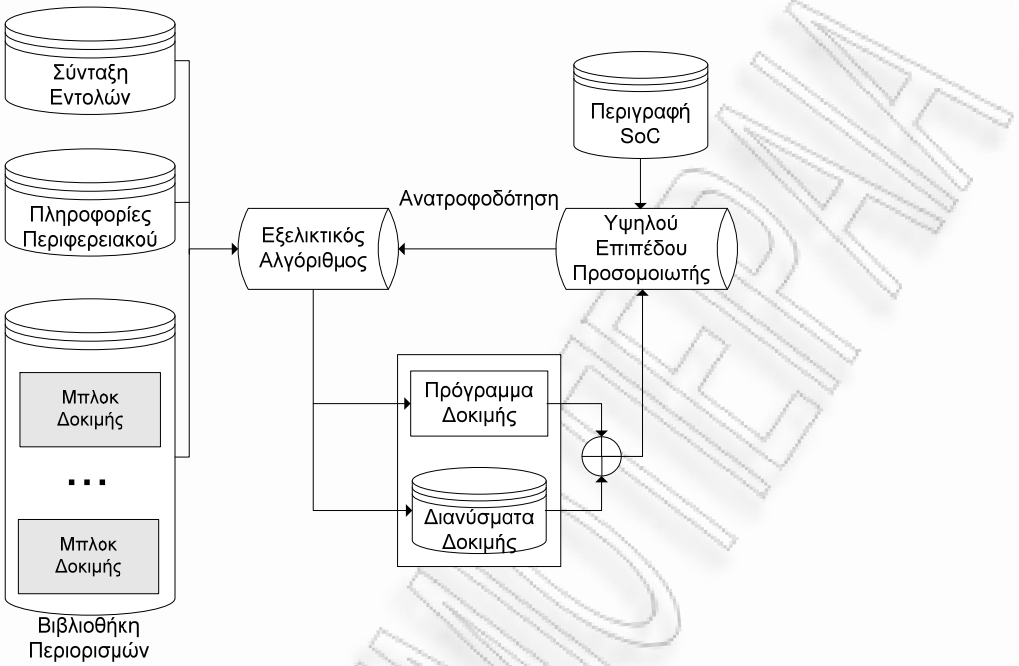
Η συγχώνευση των πλεονεκτημάτων των δύο προσεγγίσεων αποφεύγοντας τα μειονεκτήματά τους είναι η βασική ιδέα πίσω από τη δεύτερη μεθοδολογία που προτείνεται σε αυτή την διατριβή και αναλύεται στην επόμενη ενότητα.

3.9 Υβριδική μεθοδολογία αυτοδοκιμής με λογισμικό για περιφερειακά επικοινωνίας

Σε αυτή την ενότητα προτείνεται μια υβριδική μεθοδολογία παραγωγής προγράμματος δοκιμής που συνδυάζει αποτελεσματικά τα πλεονεκτήματα των δύο προηγούμενων προσεγγίσεων. Στην Εικόνα 3.19 παρουσιάζεται η ροή παραγωγής του προγράμματος δοκιμής της υβριδικής μεθοδολογίας.

Το διάγραμμα ροής είναι παρόμοιο με αυτό της αυτόματης μεθοδολογίας [127], όπου ο *εξελικτικός αλγόριθμος* (*evolutionary algorithm*) παράγει τα προγράμματα δοκιμής έχοντας μόνο γνώση της σύνταξης των εντολών του επεξεργαστή και πληροφοριών που αφορούν την λειτουργία του περιφερειακού πυρήνα. Αυτό το είδος πληροφοριών παρέχεται μέσω μιας *βιβλιοθήκης περιορισμών* (*constraints library*) που έχει καθοριστεί από τον χρήστη και περιγράφει το σκελετό των προγραμμάτων δοκιμής και των παραμέτρων που το εργαλείο πρέπει να βελτιστοποιήσει. Στην υβριδική μεθοδολογία διατηρούμε αυτή την αυτόματη διαδικασία, αλλά οι πληροφορίες που παρέχονται στη βιβλιοθήκη περιορισμών (καθώς και τα προγράμματα δοκιμής) είναι διαφορετικές.

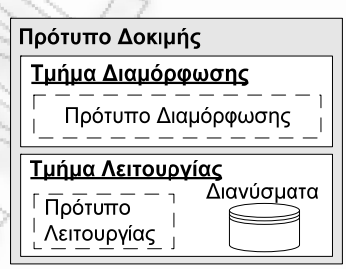
Προκειμένου να καθοδηγήσουμε αποτελεσματικότερα το εξελικτικό εργαλείο, η διαδικασία παραγωγής της υβριδικής μεθοδολογίας εστιάζει στις κύριες μονάδες ενός περιφερειακού πυρήνα επικοινωνίας. Για κάθε μονάδα καθοδηγούμε τον εξελικτικό αλγόριθμο χρησιμοποιώντας τις πληροφορίες από τον αντίστοιχο προυπολογισμένο κώδικα δοκιμής της ντετερμινιστικής μεθοδολογίας. Συνεπώς, η υβριδική βιβλιοθήκη περιορισμών μειώνει σημαντικά την *περιοχή αναζήτησης* (*search space*) του εξελικτικού αλγορίθμου, που πλέον είναι ικανός να παράγει αποτελεσματικότερα προγράμματα δοκιμής σε λιγότερες επαναλήψεις.



Εικόνα 3.19 Διάγραμμα ροής της υβριδικής προσέγγισης

Η βιβλιοθήκη περιορισμών αποτελείται από διάφορα *μπλοκ δοκιμής* (*test blocks*). Κάθε μπλοκ δοκιμής αποτελείται από δύο τμήματα, όπως φαίνεται στην Εικόνα 3.20:

- *Τμήμα Διαμόρφωσης* (*Configuration Part*): περιλαμβάνει ένα πρότυπο προγράμματος που περιγράφει τις λειτουργίες που χρησιμοποιούνται από το περιφερειακό.
- *Τμήμα Λειτουργίας* (*Functional Part*): περιέχει ένα ή περισσότερα πρότυπα προγράμματος που ενεργοποιούν τις λειτουργίες του περιφερειακού.



Εικόνα 3.20 Εννοιολογικό σχέδιο του μπλοκ δοκιμής

Η δημιουργία της βιβλιοθήκης περιορισμών απαιτεί τη γνώση των καταστάσεων λειτουργίας των περιφερειακών επικοινωνίας, αλλά σε αντίθεση με την ντετερμινιστική μεθοδολογία δεν απαιτεί οποιαδήποτε γνώση της εσωτερικής δομής του περιφερειακού.

Ο παραγόμενος κώδικας δοκιμής περιέχει τα *μπλοκ δοκιμής* (*test blocks*) με την ίδια σειρά που εμφανίζονται στη βιβλιοθήκη περιορισμών. Κάθε κώδικας δοκιμής που παράγεται από τον *εξελικτικό αλγόριθμο* (*evolutionary algorithm*) αξιολογείται χρησιμοποιώντας έναν *υψηλού επιπέδου εξομοιωτή* (*high-level simulator*). Το στάδιο αξιολόγησης ορίζει την *ευρωστία* (*fitness*) σε κάθε πρόγραμμα δοκιμής, δηλαδή ένα μέτρο για το πόσο καλά επιλύει το δεδομένο πρόβλημα. Η ευρωστία είναι βασισμένη στις ακόλουθες υψηλού επιπέδου μετρικές: τ*ην Κάλυψη Εναλλαγών* (*Toggle Coverage, TC*)*, την Κάλυψη Εκφράσεων* (*Expression Coverage, EC*)*, την Κάλυψη Συνθηκών* (*Condition Coverage, CC*)*, την Κάλυψη Διακλαδώσεων* (*Branch Coverage, BC*) *και την Κάλυψη Εντολών* (*Statement Coverage, SC*)*.*

Οι μετρικές κάλυψης ταξινομούνται βάση αποδοτικότητας: συγκρίνουμε αρχικά την ευρωστία δύο προγραμμάτων δοκιμής βασιζόμενοι στη πρώτη κατά σειρά μετρική και επιλέγουμε αυτό με την υψηλότερη τιμή. Εάν έχουν την ίδια τιμή τότε η ευρωστία συγκρίνεται βάση της δεύτερης μετρικής και ούτω καθεξής. Με αυτόν τον τρόπο μπορούμε να μεγιστοποιήσουμε την αποδοτικότητα των προγραμμάτων δοκιμής. Ωστόσο, η διάταξη στην οποία οι μετρικές μεγιστοποιούνται, δεν προσκρούει σημαντικά στο χρόνο παραγωγής του προγράμματος δοκιμής. Επιπλέον, μετά την ολοκλήρωση της παραγωγής του κώδικα δοκιμής όλες οι μετρικές έχουν σχεδόν ικανοποιηθεί, επιβεβαιώνοντας την αποτελεσματικότητα της προτεινόμενης μεθοδολογίας ανεξάρτητα από το περιφερειακό πυρήνα επικοινωνίας.

Η προτεινόμενη μεθοδολογία ξεκινάει από προγράμματα δοκιμής στα οποία οι παράμετροι επιλέγονται τυχαία και στην συνέχεια εξελίσσονται σύμφωνα με τον κλασσικό εξελικτικό αλγόριθμο μέσω των γενεών. Σε κάθε γενιά, τα τμήματα του κώδικα δοκιμής επιλέγονται σύμφωνα με το κριτήριο της ευρωστίας, οδηγώντας στην παραγωγή νέων (και βελτιωμένων ενδεχομένως) προγραμμάτων δοκιμής. Η διαδικασία ολοκληρώνεται όταν εκπνεύσει μια δεδομένη χρονική προθεσμία, ή όταν παραχθεί ένας προκαθορισμένος αριθμός προγραμμάτων δοκιμής χωρίς να υπάρξει βελτίωση των μετρικών υψηλού επιπέδου.

Τα πρότυπα δοκιμής της ντετερμινιστικής μεθοδολογίας μετασχηματίζονται σε μπλοκ δοκιμής της βιβλιοθήκης περιορισμών. Συνεπώς, η βιβλιοθήκη περιορισμών για έναν περιφερειακό πυρήνα επικοινωνίας περιλαμβάνει τέσσερα μπλοκ δοκιμής, για κάθε μεμονωμένο βήμα της ντετερμινιστικής μεθοδολογίας όπως περιγράφηκε στην Ενότητα 3.7:

- *Μπλοκ δοκιμής για τις καταστάσεις λειτουργίας* (*Test block for ooperation modes*)*.*
- *Μπλοκ δοκιμής για τις μνήμες FIFO* (*Test block for FIFOs testing*)*.*

- *Μπλοκ δοκιμής για την λογική χειρισμού των λαθών του πρωτόκολλου επικοινωνίας* (*Test block for protocol error handling.*)

- *Μπλοκ δοκιμής για την λογική διασύνδεσης του διαύλου* (*Test block for bus interface logic*).

3.9.1 Μπλοκ δοκιμής για τις καταστάσεις λειτουργίας

Το πρώτο μπλοκ δοκιμής στοχεύει στη δοκιμή του περιφερειακού πυρήνα σε πολλές διαφορετικές καταστάσεις λειτουργίας, απευθυνόμενο στη λογική που εφαρμόζει το πρωτόκολλο επικοινωνίας. Το μπλοκ δοκιμής περιγράφει τους καταχωρητές των περιφερειακών πυρήνων, την διεύθυνση μνήμης του και τις διεπαφές εγγραφής/ανάγνωσης του πρωτοκόλλου πρόσβασης. Περιγράφει επίσης τις θέσεις μνήμης ανάγνωσης και εγγραφής κατά την αποστολή ή λήψη ενός μηνύματος στην περίπτωση που το περιφερειακό ενσωματώνει έναν ελεγκτή DMA. Ο *εξελικτικός αλγόριθμος* (*evolutionary algorithm*) βελτιστοποιεί τον αριθμό των προσβάσεων εγγραφής και ανάγνωσης στους καταχωρητές του περιφερειακού, καθώς και τις τιμές που ορίζονται σε αυτούς. Με βάση τις υψηλού επιπέδου μετρικές ο εξελικτικός αλγόριθμος εξελίσσει το μπλοκ δοκιμής που προσπαθεί να θέσει το περιφερειακό πυρήνα σε πολλές καταστάσεις λειτουργίας. Σαφώς, αυτό το βήμα απαιτεί ελάχιστη ανθρώπινη παρέμβαση δεδομένου ότι αφήνουμε στο εργαλείο την εκμάθηση των καταστάσεων λειτουργίας του εκάστοτε περιφερειακού επικοινωνίας.

Σύμφωνα με την ντετερμινιστική προσέγγιση, οι μηχανικοί δοκιμής πρέπει να προσδιορίσουν όλες τις πιθανές καταστάσεις λειτουργίας του πυρήνα υπό δοκιμή και να παραγάγουν ένα πρόγραμμα δοκιμής για την καθεμία από αυτές. Στην υβριδική προσέγγιση, το εργαλείο παράγει αυτόματα τον κώδικα δοκιμής για τις διαφορετικές καταστάσεις λειτουργίας και επιλέγει τον καλύτερο βασιζόμενο στην ευρωστία, μειώνοντας έτσι το συνολικό χρόνο εφαρμογής της δοκιμής. Κατά συνέπεια, η προτεινόμενη προσέγγιση συμβάλλει στη μείωση του αριθμού των καταστάσεων λειτουργίας που ενεργοποιούνται από το πρόγραμμα δοκιμής, δίνοντας μια αποδοτική λύση σε ένα από τα σημαντικότερα προβλήματα κατά την εφαρμογή μιας προσέγγισης αυτοδοκιμής με λογισμικό για έναν περιφερειακό πυρήνα επικοινωνίας.

Όταν η μετάδοση ενός πακέτου δεδομένων ξεκινήσει από ένα πρόγραμμα δοκιμής για τον έλεγχο μιας συγκεκριμένης κατάστασης λειτουργίας, ο επεξεργαστής δεν χρονοτριβεί στην αναμονή των πακέτων λήψης αλλά συνεχίζει να εκτελεί περισσότερες εντολές εφαρμόζοντας διανύσματα και λαμβάνοντας αποκρίσεις δοκιμής προς και από τον πυρήνα επικοινωνίας, αντίστοιχα. Έτσι, μειώνονται τα *αδρανή* (*idle*) διαστήματα του επεξεργαστή κατά τη διάρκεια της μετάδοσης των δεδομένων δοκιμής, επιλύοντας το πρόβλημα που προκύπτει από την

χαμηλή συχνότητα μετάδοσης και λήψης του πυρήνα, έναντι της συχνότητας λειτουργίας του ενσωματωμένου επεξεργαστή.

3.9.2 Μπλοκ δοκιμής για τις μνήμες FIFO

Το δεύτερο μπλοκ δοκιμής στοχεύει στη δοκιμή των δομών μνήμης FIFO που ενσωματώνονται στο περιφερειακό επικοινωνίας. Η αυτόματη προσέγγιση στηρίζεται μόνο στις υψηλού επιπέδου μετρικές και για τις μνήμες FIFO ο συσχετισμός των υψηλού επιπέδου μετρικών με το ποσοστό κάλυψης ελαττωμάτων είναι μικρός (ιδιαίτερα όταν αυξάνεται το βάθος της μνήμης FIFO), με σημαντικό αντίκτυπο στο τελικό ποσοστό κάλυψης ελαττωμάτων. Αντίθετα, η υβριδική προσέγγιση υιοθετεί τα διανύσματα δοκιμής της ντετερμινιστικής προσέγγισης, η οποία χρησιμοποιεί έναν μικρό κώδικα δοκιμής και είναι σε θέση να επιτύχει ποσοστό περισσότερο από 99% κάλυψης ελαττωμάτων για ελαττώματα προσκόλλησης και για οποιοδήποτε μέγεθος και βάθος μνήμης FIFO.

Το αντίστοιχο μπλοκ δοκιμής της βιβλιοθήκης περιορισμών περιγράφει την εφαρμογή του ντετερμινιστικού συνόλου δοκιμής. Το τμήμα διαμόρφωσης του μπλοκ περιγράφει τα διανύσματα δοκιμής για τις δομές μνημών FIFO τα οποία αποθηκεύονται σε δύο μεταβλητές, ενώ το τμήμα λειτουργίας περιγράφει τον κώδικα δοκιμής που εφαρμόζει τα διανύσματα και διαβάζει τις αποκρίσεις. Το πρότυπο ψευδοκώδικα του μπλοκ δοκιμής παρουσιάζεται στην Εικόνα 3.21, όπου τα `tx_fifo_reg` και `rx_fifo_reg` είναι οι καταχωρητές δεδομένων της μετάδοσης και λήψης των μνημών FIFO, αντίστοιχα.

```
# Functional Part
1.   parameter=fifo_depth
2.   tableA=[0x0000,0xFFFF,0xFFFF,0x0000,…]
3.   tableB=[0xFFFF,0x0000,0x0000,0xFFFF,…]
4.   Initialize general registers
5.   Activate loopback mode

# Configuration Part
 # Apply test set #1
6.   for i =0 to fifo_depth
7.   tx_fifo_reg = tableA[i]
 # Capture Test Responses
8.   for i =0 to fifo_depth
9.   read rx_fifo_reg
 # Apply test set #2
10.  for i =0 to fifo_depth
11.  tx_fifo_reg = tableB[i]
 # Capture the responses
12.  for i =0 to fifo_depth
13.  read rx_fifo_reg
```

Εικόνα 3.21 Μπλοκ ψευδοκώδικα για τις μνήμες FIFO

3.9.3 Μπλοκ δοκιμής για την λογική χειρισμού των σφαλμάτων του πρωτοκόλλου επικοινωνίας

Το τρίτο μπλοκ δοκιμής στοχεύει την λογική χειρισμού των σφαλμάτων του πρωτοκόλλου επικοινωνίας. Η ντετερμινιστική μεθοδολογία, προκειμένου να στοχεύσει τη λογική αυτή, απενεργοποιεί εσκεμμένα τον βρόχο επιστροφής κατά τη διάρκεια της μετάδοσης ενός πακέτου ώστε να προκληθεί ένα σφάλμα μετάδοσης. Ανάλογα με το χρόνο και τη διάρκεια της απενεργοποίησης, διαφορετικοί τύποι σφαλμάτων μπορούν να εμφανιστούν. Στην υβριδική προσέγγιση, εφαρμόζουμε έναν εναλλακτικό μηχανισμό που είναι βασισμένος στη χρήση των διαφορετικών παραμέτρων του πρωτοκόλλου για τη μετάδοση και την λήψη του πακέτου δεδομένων. Αυτός ο μηχανισμός είναι ευκολότερος στο να αυτοματοποιηθεί και να ενσωματωθεί στη βιβλιοθήκη περιορισμών. Η βασική ιδέα είναι η τροποποίηση της τιμής του καταχωρητή ελέγχου (του οποίου το περιεχόμενο καθορίζει τις παραμέτρους του πρωτοκόλλου επικοινωνίας) αφότου έχει αρχίσει μια μετάδοση, περιμένοντας το τέλος της λήψης και διαβάζοντας έπειτα τον καταχωρητή που περιέχει την κατάσταση της γραμμής. Δεδομένου ότι η μετάδοση αρχίζει με μερικές παραμέτρους και η λήψη πραγματοποιείται σύμφωνα με άλλες παραμέτρους, ένα ή περισσότερα σφάλματα μπορούν να ενεργοποιηθούν.

Το τμήμα διαμόρφωσης του μπλοκ δοκιμής παρέχει τις παραμέτρους (`uGP_parameterA` και `uGP_parameterB`) που υπολογίζονται από τον εξελικτικό αλγόριθμο για τους διαφορετικούς τύπους σφαλμάτων, ενώ το τμήμα λειτουργίας περιγράφει τον τρόπο με τον οποίο ο μηχανισμός προκαλεί την εμφάνιση ενός σφάλματος. Το αντίστοιχο τρίτο μπλοκ δοκιμής της βιβλιοθήκης περιορισμών ακολουθεί τον ψευδοκώδικα που παρουσιάζεται στην Εικόνα 3.22, όπου οι καταχωρητές `status_regs` περιέχουν τις πληροφορίες για την κατάσταση του περιφερειακού και την κατάσταση της μετάδοσης και της λήψης των πακέτων δεδομένων. Η ρουτίνα αυτή επαναλαμβάνεται τόσες φορές όσος και ο αριθμός των πιθανών σφαλμάτων του περιφερειακού επικοινωνίας, όπως αυτό καθορίζεται από την παράμετρο `errors`.

```
# Functional Part
1.   errors = {framing_error,parity_error,…}
2.   valueA = uGP_parameterA
3.   valueB = uGP_parameterB

# Configuration Part
 # Initialize general registers
4.   Activate loopback mode
5.   Mode Register = valueA
     Write tx_fifo_reg
 # Change the Mode Register
6.   Mode Register = valueB
7.   Read status_regs
```

Εικόνα 3.22 Μπλοκ ψευδοκώδικα για τη λογική χειρισμού των σφαλμάτων του πρωτοκόλλου επικοινωνίας

3.9.4 Μπλοκ δοκιμής για την λογική διασύνδεσης διαύλου

Το τέταρτο μπλοκ δοκιμής στοχεύει στην δοκιμή της λογικής διασύνδεσης διαύλου. Όπως προαναφέραμε, η πλειοψηφία των ελαττωμάτων αυτής της μονάδας οφείλονται κυρίως στη *λογική αποκωδικοποίησης των διευθύνσεων* (*decoding logic*), δεδομένου ότι κάθε πυρήνας έχει ένα προκαθορισμένο διάστημα μνήμης. Εάν ο πυρήνας ενσωματώνει έναν ελεγκτή DMA, τα διαφορετικά διαστήματα μνήμης μπορούν να οριστούν ως περιοχή του ελεγκτή.

Κατά συνέπεια, υιοθετήσαμε τη λύση της ντετερμινιστικής προσέγγισης. Σύμφωνα με αυτή, ορίζονται οι διαφορετικές περιοχές της ενσωματωμένης μνήμης δεδομένων ως περιοχές του ελεγκτή DMA και για καθεμία από αυτές πραγματοποιούμε μια αποστολή δεδομένων. Στην υβριδική προσέγγιση το εξελικτικό εργαλείο ψάχνει για τα διαστήματα των περιοχών DMA που μεγιστοποιούν τις μετρικές κάλυψης της λογικής διασύνδεσης διαύλου. Το τμήμα διαμόρφωσης

του μπλοκ δοκιμής παρέχει τις παραμέτρους των δύο περιοχών του ελεγκτή DMA που υπολογίζονται από το εξελικτικό εργαλείο, ενώ το τμήμα λειτουργίας παρέχει τον κώδικα δοκιμής που καθορίζει τον τρόπο λειτουργίας του ελεγκτή DMA και πραγματοποιεί την αποστολή και λήψη των πακέτων δεδομένων. Το αντίστοιχο τέταρτο μπλοκ δοκιμής της βιβλιοθήκης περιορισμών ακολουθεί τον ψευδοκώδικα που περιγράφεται στην Εικόνα 3.23

```
# Functional Part
1.   RegionA = uGP_parameterA
2.   RegionB = uGP_parameterB

# Configuration Part
 # Define virtual DMA region
3.   Configure the DMA region [RegionA] of TX FIFO
4.   Configure the DMA region [RegionB] of RX FIFO
 # Apply Test Data
5.   Stores the transmit frame to memory
6.   Transmit frame
 # Capture Test Responses
7.   Load the receive frame from memory
```

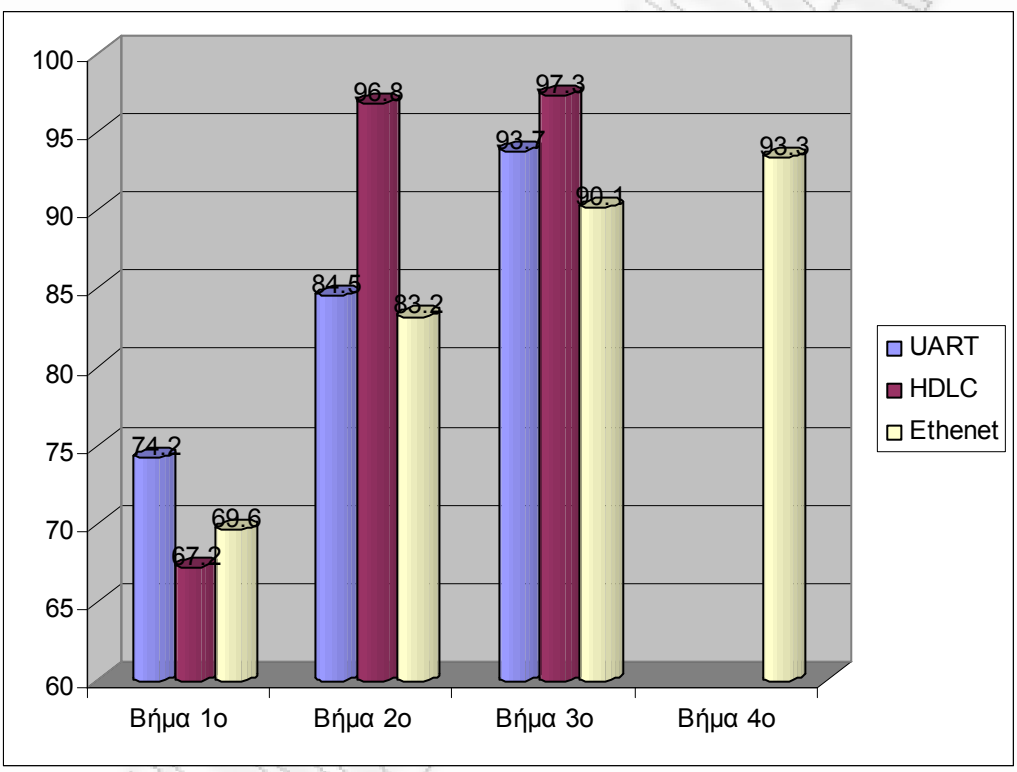
Εικόνα 3.23 Μπλοκ ψευδοκώδικα για την δοκιμή του διαύλου επικοινωνίας

3.10 Πειραματικά αποτελέσματα

Τα αποτελέσματα που παρουσιάζονται σε αυτήν την ενότητα συγκρίνουν τις τρεις μεθοδολογίες (αυτόματη [127], ντετερμινιστική [26], [27] και υβριδική [28]) ως προς το ποσοστό κάλυψης ελαττωμάτων, το χρόνο ανάπτυξης της δοκιμής, το μέγεθος του κώδικα δοκιμής και το χρόνο εφαρμογής της δοκιμής. Οι μεθοδολογίες αξιολογήθηκαν με πειράματα που έγιναν στο SoC που παρουσιάσαμε στην Ενότητα 3.8, το οποίο περιλαμβάνει τον ενσωματωμένο επεξεργαστή OpenRISC 1200 [24] και τρία περιφερειακά επικοινωνίας UART, HDLC και Ethernet [24].

Για την αυτόματη και την υβριδική μεθοδολογία οι ερευνητές ανέπτυξαν ένα εργαλείο, το οποίο εφαρμόζει τις δύο μεθόδους. Το εργαλείο έχει γραφεί σε γλώσσα προγραμματισμού C++ και ανέρχεται σε περίπου 50.000 γραμμές κώδικα. Όλα τα πειράματα έχουν εκτελεσθεί σε ένα ηλεκτρονικό υπολογιστή με επεξεργαστή Athlon XP 3000 και μνήμη 1GByte με λειτουργικό Linux. Για την προσομοίωση ελαττωμάτων χρησιμοποιήθηκε το εργαλείο TetraMax από την εταιρία Synopsys [134], ενώ για τον υπολογισμό των μετρικών υψηλού επιπέδου χρησιμοποιήθηκε το εργαλείο ModelSim από την εταιρία Mentor Graphics [135].

Πριν αναλύσουμε και συγκρίνουμε τα αποτέλεσμα των τριών μεθοδολογιών παρουσιάζουμε το ποσοστό βελτίωσης της κάλυψης ελαττωμάτων που επιφέρει κάθε βήμα της ντετερμινιστικής μεθοδολογίας. Τα αποτελέσματα του ποσοστού κάλυψης ελαττωμάτων συναρτήσει των βημάτων της ντετερμινιστικής μεθοδολογίας παρουσιάζονται στην Εικόνα 3.24.



Εικόνα 3.24 Βελτίωση ως προς το ποσοστό κάλυψης ελαττωμάτων ανά βήμα της ντετερμινιστικής μεθοδολογίας

Τα βήματα της ντετερμινιστικής μεθοδολογίας "εμβολιάστηκαν" στην αυτόματη μεθοδολογία με απώτερο στόχο την βελτίωση του ποσοστού κάλυψης ελαττωμάτων της τελευταίας και την αποδοτικότερη αναζήτηση του χώρου πιθανών λύσεων του εξελικτικού αλγόριθμου. Κατά το πρώτο βήμα της μεθοδολογίας το συνολικό ποσοστό κάλυψης ελαττωμάτων και για τα τρία περιφερειακά επικοινωνίας ανέρχεται περί το 70%. Το χαμηλό αυτό ποσοστό είναι απολύτως αναμενόμενο καθώς το πρώτο βήμα της μεθοδολογίας μοιάζει πάρα πολύ με τα βήματα που ακολουθούνται κατά την *επαλήθευση σχεδίασης* (*design verification*) και δεν εφαρμόζει κανένα ντετερμινιστικό διάνυσμα δοκιμής προς τα περιφερειακά επικοινωνίας. Κατά τα δύο επόμενα βήματα (βήμα 2^{ο} και βήμα 3^{ο}) παρατηρείται μία σταδιακή άνοδος του ποσοστού κάλυψης ελαττωμάτων και για τα τρία περιφερειακά. Σημαντική είναι η βελτίωση του ποσοστού κάλυψης ελαττωμάτων του ελεγκτή HDLC από το 67.2% (βήμα 1^{ο}) στο

96.8% (βήμα 2ο) σε ένα μόλις βήμα της ντετερμινιστικής μεθοδολογίας. Δεδομένου ότι το 2ο βήμα της ντετερμινιστικής μεθοδολογίας είναι σε θέση να επιτύχει ποσοστό κάλυψης ελαττωμάτων περισσότερο από 99% για οποιαδήποτε μνήμη FIFO ανεξάρτητα από το μέγεθος της, καθώς επίσης και ότι η εν λόγω μνήμη FIFO καταλαμβάνει το 87.13% του συνολικού αριθμού πυλών του ελεγκτή HDLC, γίνεται αντιληπτό ότι και το συνολικό ποσοστό κάλυψης ελαττωμάτων θα συμπαρασυρθεί επιτυγχάνοντας ποσοστό κάλυψης ελαττωμάτων κοντά στο 97% σε ένα μόλις βήμα της ντετερμινιστικής προσέγγισης.

Τέλος αξίζει να σημειωθεί ότι δεν μπορεί να υπάρξει βελτίωση του ποσοστού κάλυψης ελαττωμάτων τόσο για τον ελεγκτή UART όσο και για τον ελεγκτή HDLC κατά την εφαρμογή του 4ου βήματος της ντετερμινιστικής μεθοδολογίας. Το 4ο βήμα της μεθοδολογίας λαμβάνει μέρος μόνο εάν ενσωματώνεται ένας ελεγκτής άμεσης προσπέλασης μνήμης στο περιφερειακό επικοινωνίας. Οι ελεγκτές UART και HDLC δεν ενσωματώνουν ελεγκτή DMA, σε αντίθεση με τον πυρήνα Ethernet, με αποτέλεσμα να μην είναι αναγκαία η εφαρμογή του 4ου σταδίου και συνεπώς να μην βελτιώνεται περαιτέρω το ποσοστού κάλυψης ελαττωμάτων.

Ο Πίνακας 3.8 παρουσιάζει λεπτομερή αποτελέσματα προσομοίωσης ελαττωμάτων για ελαττώματα προσκόλλησης του ελεγκτή UART σύμφωνα με τις τρεις διαφορετικές προσεγγίσεις αυτοδοκιμής με λογισμικό. Η κάλυψη ελαττωμάτων των περισσότερων μονάδων που επιτυγχάνεται από την υβριδική μεθοδολογία βελτιώνεται σημαντικά έναντι της αυτόματης προσέγγισης. Επιπλέον η υβριδική προσέγγιση, επιτυγχάνει υψηλό ποσοστό κάλυψης ελαττωμάτων όμοιο με την ντετερμινιστική προσέγγιση, απαιτώντας πολύ λιγότερο χρόνο για την ανάπτυξη του κώδικα δοκιμής (Πίνακας 3.11). Οι περισσότερες μονάδες του ελεγκτή UART επιτυγχάνουν ποσοστό κάλυψης ελαττωμάτων προσκόλλησης περισσότερο από 90%, ενώ η συνολική κάλυψη ελαττωμάτων είναι 93.13%. Ειδικά για τις μνήμες FIFOs, που καταλαμβάνουν ένα σημαντικό μέρος του πυρήνα, τα προγράμματα δοκιμής επιτυγχάνουν πολύ υψηλή κάλυψη ελαττωμάτων (μεγαλύτερη από 99.5%).

Ο Πίνακας 3.9 παρουσιάζει λεπτομερή αποτελέσματα προσομοίωσης ελαττωμάτων για τον ελεγκτή HDLC. Η υβριδική μεθοδολογία επιτυγχάνει ποσοστό κάλυψης ελαττωμάτων προσκόλλησης 97.43% (το πολύ μεγαλύτερο βάθος της μνήμης FIFO του πυρήνα έχει θετικό αντίκτυπο στη συνολική κάλυψη ελαττωμάτων) που είναι ελαφρώς υψηλότερη από την κάλυψη ελαττωμάτων της ντετερμινιστικής προσέγγισης, αλλά σχεδόν 30% υψηλότερη σε σύγκριση με την αυτόματη προσέγγιση. Η αυτόματη προσέγγιση δεν είναι ικανή να επιτύχει καλύτερα αποτελέσματα ως προς το ποσοστό κάλυψης ελαττωμάτων καθώς ο συσχετισμός μεταξύ των υψηλού επιπέδου μετρικών της μνήμης FIFO και της αντίστοιχης κάλυψης ελαττωμάτων είναι μικρός. Για ακόμα μια φορά, ο χρόνος ανάπτυξης του προγράμματος δοκιμής έναντι της ντετερμινιστικής προσέγγισης είναι χαμηλότερος (Πίνακας 3.11).

Μονάδα	Ελαττώματα προσκόλλησης	Ποσοστό Κάλυψης Ελαττωμάτων (%)		
		Αυτόματη	Ντετερμινιστική	Υβριδική
Wishbone I/F	2,058	88.44	90.57	90.18
Registers	4,490	77.59	91.43	87.32
Tx Engine	1,560	96.23	95.20	96.59
Tx FIFO cntr	566	98.90	94.02	98.94
Tx FIFO	2,850	95.16	99.33	99.72
Rx Engine	3,066	77.21	91.23	88.13
Rx FIFO cntr	2,546	80.52	91.40	95.68
Rx FIFO	2,850	99.02	99.86	100.00
Debug	536	85.63	92.72	86.57
Σύνολο	**20,558**	**86.35**	**93.92**	**93.13**

Πίνακας 3.8 Αποτελέσματα προσομοίωσης ελαττωμάτων για τον ελεγκτή UART

Μονάδα	Ελαττώματα προσκόλλησης	Ποσοστό Κάλυψης Ελαττωμάτων (%)		
		Αυτόματη	Ντετερμινιστική	Υβριδική
Tx Synch	40	90.00	80.00	90.00
Tx Core	1,590	88.36	82.50	88.61
Rx Channel	1,216	82.98	79.17	80.17
Rx Synch	40	90.00	90.00	90.00
Wishbone I/F	1,812	67.11	84.81	85.26
Tx FIFO	25,118	99.24	99.24	99.24
Tx CRC	1,606	94.96	91.78	84.68
Rx FIFO	25,308	71.82	98.81	98.90
Rx CRC	1,118	12.97	93.11	93.29
Σύνολο	**57,868**	**68.65**	**97.34**	**97.43**

Πίνακας 3.9 Αποτελέσματα προσομοίωσης ελαττωμάτων για τον ελεγκτή HDLC

Ο Πίνακας 3.10 παρουσιάζει λεπτομερή αποτελέσματα προσομοίωσης ελαττωμάτων για τον πυρήνα Ethernet. Για αυτόν τον πυρήνα, περιγράψαμε επίσης στη βιβλιοθήκη περιορισμών το μηχανισμό πρόσβασης του ελεγκτή DMAτόσο για την αυτόματη όσο και για την υβριδική προσέγγιση. Η κάλυψη ελαττωμάτων σε αυτήν την περίπτωση είναι ελαφρώς χαμηλότερη (91.79%) από την ντετερμινιστική προσέγγιση (93.39%), δεδομένου ότι η ύπαρξη του ελεγκτή DMA προσθέτει πολυπλοκότητα, η οποία αποτελεί ένα εμπόδιο αυξάνοντας τον χώρο αναζήτησης της υβριδικής μεθοδολογία. Ωστόσο, η υβριδική προσέγγιση επιτυγχάνει πολύ καλύτερα αποτελέσματα σε σύγκριση με την αυτόματη προσέγγιση. Ο χρόνος ανάπτυξης του

κώδικα δοκιμής αυξάνεται λόγω της περιγραφής του μηχανισμού πρόσβασης του ελεγκτή DMA στη βιβλιοθήκη περιορισμών, αλλά σε κάθε περίπτωση είναι πολύ μικρότερος από αυτόν της ντετερμινιστικής προσέγγισης (Πίνακας 3.11).

Μονάδα	Ελαττώματα προσκόλλησης	Ποσοστό Κάλυψης Ελαττωμάτων (%)		
		Αυτόματη	Ντετερμινιστική	Υβριδική
Wishbone I/F	11,858	68.98	88.92	88.26
Registers	8,120	91.62	99.00	95.10
Tx Engine	5,002	72.53	81.38	75.31
Tx FIFO	20,465	98.50	98.80	98.81
Rx Engine	3,564	67.97	72.09	72.00
Rx FIFO	20,465	99.15	99.78	99.12
Management	1,868	97.20	95.59	97.22
Control	3,884	48.58	71.38	74.12
Status	1,306	36.08	77.62	38.31
Glue Logic	1,042	82.70	97.73	94.13
Σύνολο	**77,574**	**86.57**	**93.39**	**91.79**

Πίνακας 3.10 Αποτελέσματα προσομοίωσης ελαττωμάτων για τον πυρήνα Ethernet

Ο Πίνακας 3.11 παρουσιάζει το χρόνο ανάπτυξης του προγράμματος δοκιμής για τις τρεις διαφορετικές προσεγγίσεις. Είναι σημαντικό να τονίσουμε ότι στην περίπτωση της αυτόματης και της υβριδικής προσέγγισης ο χρόνος ανάπτυξης της δοκιμής αντιστοιχεί στο χρόνο που απαιτείται για την οργάνωση του περιβάλλοντος, συν το χρόνο που χρειάζεται από το εργαλείο για να παράγει τα προγράμματα δοκιμής. Ο χρόνος παραγωγής δοκιμής της υβριδικής προσέγγισης είναι ελαφρώς μεγαλύτερος από αυτόν της αυτόματης προσέγγισης, δεδομένου ότι η διαδικασία παραγωγής είναι ακριβέστερη και υπάρχουν περισσότερες παράμετροι που πρέπει βελτιστοποιούν. Στον αντίποδα, η υβριδική προσέγγιση είναι αποδοτικότερη ως προς το χρόνο παραγωγής του κώδικα δοκιμής έναντι της ντετερμινιστικής προσέγγισης.

Πυρήνας	Χρόνος Ανάπτυξης Δοκιμής (~Μέρες)		
	Αυτόματη	Ντετερμινιστική	Υβριδική
UART	1.0	30	1.5
HDLC	0.5	15	0.8
Ethernet	2.0	45	6.0

Πίνακας 3.11 Χρόνος ανάπτυξης κώδικα δοκιμής για τις τρεις διαφορετικές προσεγγίσεις

Ο Πίνακας 3.12 περιγράφει τους επιμέρους χρόνους που απαιτούνται, για την ανάπτυξη των πειραμάτων (κυρίως για την δημιουργία της βιβλιοθήκης περιορισμών), την εκτέλεση του εργαλείου και το συνολικό χρόνο για την αυτόματη και την υβριδική προσέγγιση. Ο χρόνος για την ανάπτυξη των πειραμάτων είναι ελαφρώς μεγαλύτερος από το χρόνο παραγωγής του κώδικα δοκιμής, τόσο για την αυτόματη όσο και για την υβριδική προσέγγιση. Δεδομένου ότι στην υβριδική προσέγγιση η βιβλιοθήκη περιορισμών είναι πιο σύνθετη, ο χρόνος που απαιτείται για την ανάπτυξη και για την παραγωγή δοκιμής είναι μεγαλύτερος απ' ότι στην αυτόματη προσέγγιση.

Πυρήνας	**Χρόνος Ανάπτυξης Δοκιμής (~Μέρες)**					
	Αυτόματη			**Υβριδική**		
	Χρόνος Ανάπτυξης	**Παραγωγή Κώδικα**	**Συνολικός Χρόνος**	**Χρόνος Ανάπτυξης**	**Παραγωγή Κώδικα**	**Συνολικός Χρόνος**
UART	0.6	0.4	1.0	0.8	0.7	1.5
HDLC	0.3	0.2	0.5	0.4	0.4	0.8
Ethernet	1.2	0.8	2.0	3.5	2.5	6.0

Πίνακας 3.12 Αναλυτικός χρόνος ανάπτυξης κώδικα δοκιμής για της αυτόματη και την υβριδική προσέγγιση

Ο Πίνακας 3.13 συνοψίζει τα ποσοστά κάλυψης ελαττωμάτων των τριών διαφορετικών προσεγγίσεων. Η τελευταία γραμμή του πίνακα εκθέτει τη συνολική κάλυψη ελαττωμάτων θεωρώντας το σύνολο των περιφερειακών πυρήνων επικοινωνίας. Η υβριδική μεθοδολογία επιτυγχάνει κάλυψη ελαττωμάτων σχεδόν τόσο υψηλή όσο και η ντετερμινιστική προσέγγιση με την τελευταία να επιτυγχάνει το υψηλότερο ποσοστό κάλυψης ελαττωμάτων από όλες τις προσεγγίσεις. Ωστόσο, τόσο η υβριδική όσο και αυτόματη προσέγγιση δεν απαιτούν την συνεχή συμμετοχή του μηχανικού δοκιμής για την δημιουργία του προγράμματος δοκιμής, όπως συμβαίνει στην περίπτωση της ντετερμινιστικής προσέγγισης.

Πυρήνας	**Ποσοστό Κάλυψης Ελαττωμάτων**		
	Αυτόματη	**Ντετερμινιστική**	**Υβριδική**
UART	86.35	93.92	93.13
HDLC	68.65	97.34	97.43
Ethernet	86.57	93.39	91.79
Σύνολο	**79.89**	**94.92**	**94.05**

Πίνακας 3.13 Ποσοστά κάλυψης ελαττωμάτων για τις τρεις διαφορετικές προσεγγίσεις

Ο Πίνακας 3.14 παρουσιάζει τα στατιστικά των προγραμμάτων δοκιμής που παράγονται από κάθε προσέγγιση. Η αυτόματη προσέγγιση έναντι της ντετερμινιστικής παράγει μεγαλύτερου μεγέθους προγράμματα δοκιμής για κάθε πυρήνα, σε αντίθεση με τον χρόνο εφαρμογής της δοκιμής που είναι πολύ χαμηλότερος. Κατά συνέπεια, η υβριδική προσέγγιση παράγει επίσης μεγαλύτερα προγράμματα δοκιμής σε σύγκριση με την ντετερμινιστική προσέγγιση, αλλά με ταχύτατο *χρόνο εφαρμογής της δοκιμής* (*test application time*). Αυτό είναι πολύ σημαντικό, δεδομένου ότι ο χρόνος εφαρμογής της δοκιμής πρέπει να είναι πολύ μικρός κατά την *δοκιμή κατασκευής* (*manufacturing testing*) του SoC. Στην περίπτωση του πυρήνα Ethernet η διαφορά μεταξύ του μεγέθους κώδικα της αυτόματης/υβριδικής προσέγγισης και του μεγέθους κώδικα της ντετερμινιστικής προσέγγισης είναι ακόμα μεγαλύτερη, δεδομένου ότι η παρουσία του ελεγκτή DMA προσθέτει περισσότερους περιορισμούς στη διαδικασία παραγωγής και το εργαλείο δεν είναι ικανό να παράγει αποδοτικά βελτιστοποιημένο κώδικα.

Πυρήνας	Αυτόματη		Ντετερμινιστική		Υβριδική	
	Κώδικας (λέξεις)	Χρόνος (msec)	Κώδικας (λέξεις)	Χρόνος (msec)	Κώδικας (λέξεις)	Χρόνος (msec)
UART	5,969	2.91	3,429	22.55	6,330	6.00
HDLC	1,816	1.95	623	3.21	1,823	2.04
Ethernet	15,190	9.38	2,358	59.30	12,129	29.21

Πίνακας 3.14 Στατιστικά προγράμματος δοκιμής για τις τρεις διαφορετικές προσεγγίσεις

3.11 Ανακεφαλαίωση

Στο κεφάλαιο 3 παρουσιάστηκαν,

- η αρχιτεκτονική των SoC καθώς επίσης και η αρχιτεκτονική των περιφερειακών επικοινωνίας,
- οι τεχνικές δοκιμής βασισμένες στο υλικό των συστημάτων αυτών,
- η σχετική ως προς την δοκιμή βιβλιογραφία για περιφερειακές μονάδες και οι αδυναμίες που παρουσιάζουν οι υπάρχουσες τεχνικές,
- η ντετερμινιστική προσέγγιση αυτοδοκιμής με λογισμικό για περιφερειακά επικοινωνίας,
- η υβριδική προσέγγιση αυτοδοκιμής με λογισμικό για περιφερειακά επικοινωνίας.

Κεφάλαιο 4

Δοκιμή Συμμετρικών Πολυεπεξεργαστών

Pure mathematics is, in its way, the poetry of logical ideas. One seeks the most general ideas of operation which will bring together in simple, logical and unified form the largest possible circle of formal relationships. In this effort toward logical beauty spiritual formulas are discovered necessary for the deeper penetration into the laws of nature.

Albert Einstein

4.1 Εισαγωγή

Σύμφωνα με τους περισσότερους ειδικούς ο πόλεμος των συνεχώς αυξανόμενων συχνοτήτων λειτουργίας και της συρρίκνωσης του μεγέθους των τρανζίστορ, που οδήγησε τον κόσμο των επεξεργαστών για περισσότερα από 20 χρόνια, ανήκει πλέον στο παρελθόν. Οι εταιρείες ανταγωνίζονταν η μία την άλλη σε συχνότητα λειτουργίας και οι βελτιώσεις σε απόδοση ήταν σημαντικές, με τις ταχύτητες να διπλασιάζονταν κάθε ένα με δύο χρόνια, από τα 4,77MHz του 1981 στα 3GHz του 2002. Όμως, από τη στιγμή που κατακτήθηκαν τα 3GHz, οι ταχύτητες έχουν προχωρήσει ελάχιστα παραπάνω.

Ένας λόγος είναι η κατανάλωση ενέργειας και η ανάλογη απαίτηση για πιο προηγμένες μορφές ψύξης. Η κατανάλωση ενέργειας ενός μικροεπεξεργαστή είναι ευθέως ανάλογη της συχνότητας ρολογιού. Συνεπώς, ακόμα και αν όλες οι άλλες παράμετροι είναι ίδιες ο διπλασιασμός της συχνότητας θα προκαλέσει διπλασιασμό της κατανάλωσης ισχύος. Επιπλέον η συρρίκνωση του μεγέθους των τρανζίστορ προκαλεί εκθετική αύξηση του *ρεύματος διαρροής* (*leakage power*). Κατά συνέπεια, αυτή η κατεύθυνση προς τη βελτίωση της απόδοσης στους μικροεπεξεργαστές, φαίνεται να έχει φτάσει σε αδιέξοδο για την τεχνολογία των ημιαγωγών.

Η είσοδος των πολυπύρηνων επεξεργαστών έχει ως στόχο να λύσει τα παραπάνω προβλήματα, αλλά αντίθετα με προηγούμενες βελτιώσεις στην ψηφιακή σχεδίαση, τα οφέλη που προσφέρει η παράλληλη επεξεργασία δεν είναι εγγυημένα. Επιπλέον, οι σημερινές εφαρμογές είναι ιδιαίτερα απαιτητικές σε υπολογιστική ισχύ. Για αυτούς τους λόγους, πολλοί κατασκευαστές επεξεργαστών παρουσιάζουν νέες γενιές οι οποίοι περιλαμβάνουν πολλούς πυρήνες σε έναν ενιαίο ολοκληρωμένο κύκλωμα, όπου αν και λειτουργούν σε χαμηλότερη συχνότητα λειτουργίας από τους προηγούμενης γενιάς επεξεργαστές, είναι ικανοί να προσφέρουν υψηλότερη επεξεργαστική ισχύ και παραλληλισμό μεταξύ των εφαρμογών.

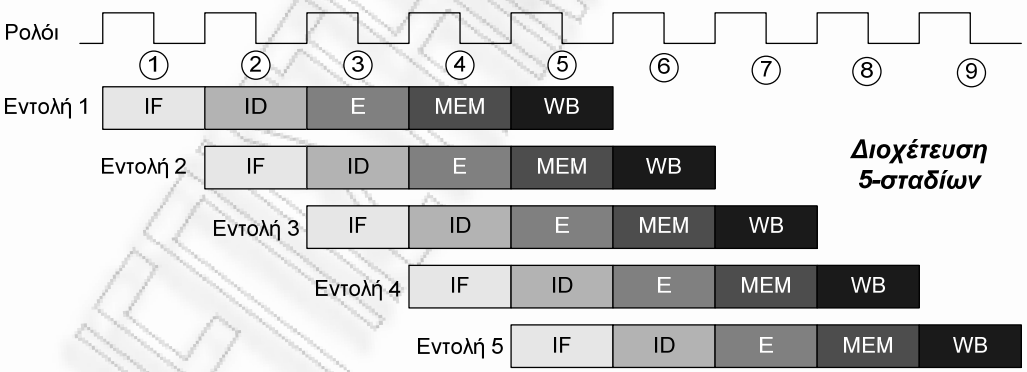
Σήμερα, οι χρήστες διαθέτουν τετραπύρηνους επεξεργαστές στους προσωπικούς υπολογιστές τους και οι ειδικοί αναμένουν τους μελλοντικούς επεξεργαστές να διαθέτουν εκατοντάδες πυρήνες. Μερικές εταιρείες ήδη διαθέτουν προϊόντα που χρησιμοποιούν επεξεργαστές με πολλούς πυρήνες. Η Intel Corporation [138] έχει ήδη επιδείξει έναν 80πύρηνο επεξεργαστή [140]. Η Sun Microsystems [136] χρησιμοποιεί έναν οκταπύρηνο επεξεργαστή UltraSPARC T2 στους τελευταίους της διακομιστές (servers) και άλλες εταιρείες παρέχουν ολοκληρωμένα με εκατοντάδες πυρήνες, τα οποία στοχεύουν εξειδικευμένες εφαρμογές, όπως κινητά τηλέφωνα. Η αγορά των προσωπικών ηλεκτρονικών υπολογιστών, όμως, δεν μένει πίσω. Τόσο η Intel Corporation [138], όσο και η Advanced Micro Devices (AMD) [139] αναμένεται να διαθέτουν οκταπύρηνους στο εγγύς μέλλον. Στην παρούσα διδακτορική διατριβή αναλύουμε την πολυπύρηνη τεχνολογία, τα κέρδη που προσφέρει και τη δοκιμή των συμμετρικών πολυεπεξεργαστών με στόχο την μείωση του συνολικού χρόνου εφαρμογής της δοκιμής.

4.2 Προσεγγίσεις στον παραλληλισμό

Παρόλο που η λέξη "πολυπύρηνος" κυριαρχεί τη δεδομένη στιγμή, δεν αντικατοπτρίζει το πρώτο βήμα της βιομηχανίας στην παράλληλη επεξεργασία. Η πολυπύρηνη προσέγγιση αποτελεί την πιο πρόσφατη ιδέα σε μια μεγάλη σειρά από τεχνικές που αποσκοπούν στο να κάνουν περισσότερα του ενός πράγματα ταυτόχρονα.

4.2.1 Επεξεργαστής με διοχέτευση

Η επιστροφή στις εντολές ενός κύκλου ήταν το αποτέλεσμα μίας τεχνικής που ονομάζεται *διοχέτευση* (*pipeline*) [144], η οποία αποτέλεσε το πρώτο αβέβαιο βήμα της βιομηχανίας στον παραλληλισμό. Η εκτέλεση μίας εντολής είναι μια διαδικασία πολλών βημάτων. Μια απλή μορφή διοχέτευσης, μπορεί να απαιτεί τα παρακάτω βήματα: την *προσκόμιση της εντολής* (*Instruction Fetch, IF*), την *αποκωδικοποίηση της εντολής* (*Instruction Decode, ID*), *την εκτέλεση* (*Execution, E*), τη *μεταφορά δεδομένων από και προς τη μνήμη* (*Memory, Mem*) και τέλος, την *εγγραφή των αποτελεσμάτων στο αρχείο καταχωρητών* (*Write Back, WB*). Καθένα από τα βήματα αυτά θα μπορούσε να διαρκέσει έναν κύκλο ρολογιού. Τα βήματα δεν είναι δυνατό να πραγματοποιηθούν παράλληλα για μία μοναδική εντολή, όμως, για παράδειγμα, μόλις ο αποκωδικοποιητής εντολών έχει ολοκληρώσει την εκτέλεση μιας εντολής, μπορεί να μεταβεί στην αποκωδικοποίηση της επόμενης, την ίδια στιγμή που ένα άλλο τμήμα της διοχέτευσης μεταφέρει τα δεδομένα που χρειάζεται η πρώτη εντολή. Ένα χαρακτηριστικό παράδειγμα εκτέλεσης πέντε εντολών σε ένα επεξεργαστή με υποστήριξη διοχέτευσης 5-σταδίων απεικονίζεται στην Εικόνα 4.1 όπου στον 5ο κύκλο ρολογιού παρατηρούνται πέντε εντολές στη διοχέτευση οι οποίες εκτελούνται παράλληλα.

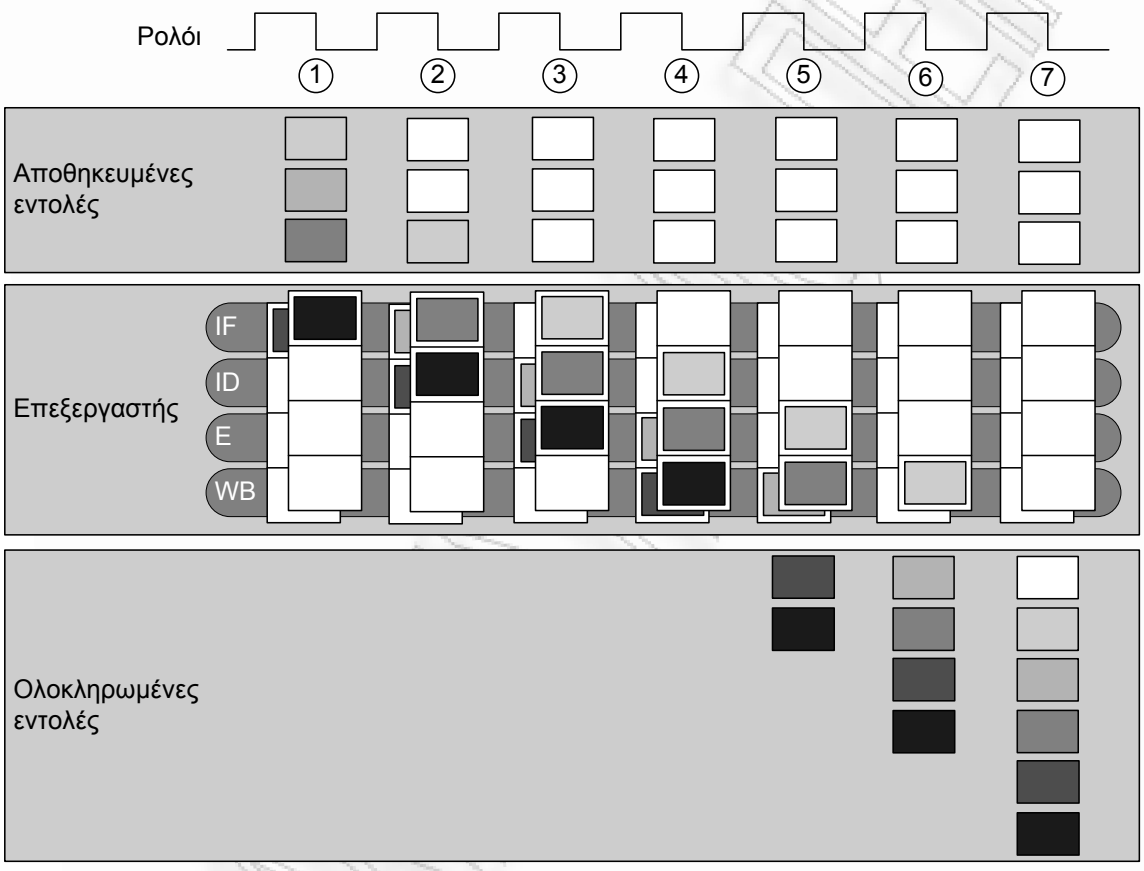


Εικόνα 4.1 Εκτέλεση εντολών σε ενσωματωμένο επεξεργαστή με διοχέτευση 5-σταδίων.

4.2.2 Υπερβαθμωτοί επεξεργαστές

Η επόμενη εξέλιξη στον παραλληλισμό ήταν η παροχή πολλαπλών μονάδων εκτέλεσης και πολλαπλών διοχετεύσεων στους *υπερβαθμωτούς επεξεργαστές* (*superscalar*) [142]. Ένας επεξεργαστής για παράδειγμα, μπορεί να διαθέτει δύο *αριθμητικές και λογικές μονάδες* (*Arithmetic Logic Unit, ALU*), δύο *μονάδες κινητής υποδιαστολής* (*Floating Point Unit, FPU*) και δύο μονάδες διοχέτευσης. Σε αυτή την περίπτωση, οι πολλαπλές μονάδες εκτέλεσης επιτρέπουν για πρώτη φορά την εκτέλεση περισσότερων της μίας εντολών σε έναν κύκλο ρολογιού. Όμως, η επιτάχυνση δεν παρουσιάζει την αναμενόμενη αύξηση. Το γεγονός αυτό

οφείλεται στο ότι, δεν είναι πάντα δυνατό να εκτελεστούν δύο συνεχόμενες εντολές παράλληλα, διότι η δεύτερη μπορεί να εξαρτάται από το αποτέλεσμα της πρώτης. Παράλληλα, η εκτέλεση πολλαπλών εντολών μπορεί να περιλαμβάνει πρόβλεψη για το αν θα γίνει διακλάδωση ή όχι. Αν αυτή η πρόβλεψη αποδειχθεί λανθασμένη, το αποτέλεσμα μίας από τις εντολές που εκτελέστηκαν παράλληλα θα πρέπει να απορριφθεί. Τέλος, εάν μόνο οι ALU και οι FPU διπλασιαστούν, υπάρχει ακόμη ανταγωνισμός για άλλους πόρους στο εσωτερικό του επεξεργαστή.



Εικόνα 4.2 Υπερβαθμωτός επεξεργαστής διοχέτευσης 4-σταδίων (στην ιδανική κατάσταση σε κάθε κύκλο ρολογιού μπορεί να εκτελέσει δύο διαφορετικές εντολές παράλληλα).

Οι υπερβαθμωτές αρχιτεκτονικές υποστηρίζουν επίσης την εκτέλεση *εντολών εκτός σειράς* (*out-of-order*) [144]. Αυτό σημαίνει ότι κάποιες εντολές που ίσως καθυστερούν σε κάποια από τα στάδια της διοχέτευσης του υπερβαθμωτού επεξεργαστή (όπως όταν γίνεται μια αναφορά στην κύρια μνήμη), παρακάμπτονται και συνεχίζεται η εκτέλεση κάποιων από τις επόμενες εντολές. Το τελικό όμως αποτέλεσμα θα εμφανίζει τις εντολές να ολοκληρώνονται με τη *σειρά που επιτάσσει το αρχικό πρόγραμμα* (*in-order-retirement*). Με τον τρόπο αυτό, διατηρείται η

ορθότητα της λογικής σειράς των εντολών και τα αποτελέσματά είναι τα ίδια με την περίπτωση που οι εντολές θα εκτελούνταν αυστηρά σειριακά, όπως εμφανίζονται στην Εικόνα 4.2. Στον επεξεργαστή αυτό υπάρχουν δυο ξεχωριστές διοχετεύσεις, η κάθε μια από τις οποίες έχει όλα τα στάδια της διοχέτευσης (στο συγκεκριμένο παράδειγμα τέσσερα στάδια). Σε κάθε στάδιο της διοχέτευσης μπορεί να εκτελείται μια εντολή σε κάθε κύκλο. Παρατηρείται ότι ενώ οι εντολές ακολουθούν δυο διαφορετικά μονοπάτια διοχέτευσης, στο τέλος της εκτέλεσης του προγράμματος η εντύπωση που δίνεται στο χρήση είναι ότι η σειρά τους διατηρήθηκε.

4.2.3 Πολυνημάτωση

Όλες οι προηγούμενες παράλληλες προσεγγίσεις παρουσιάζουν έναν κοινό παρονομαστή: η διαχείριση των παράλληλων πόρων πραγματοποιείται στο εσωτερικό του επεξεργαστή, με αποτέλεσμα να είναι διαφανής στο λειτουργικό σύστημα, καθώς και στις εφαρμογές. Κατά συνέπεια, η αλλαγή ενός επεξεργαστή χωρίς διοχέτευση με έναν που διαθέτει, ή η αναβάθμιση ενός επεξεργαστή που διαθέτει μία ALU με έναν που διαθέτει δύο ALU, θα παρουσίαζε άμεση αύξηση της απόδοσης με το ίδιο λογισμικό. Ωστόσο, αυτό δεν συμβαίνει με τους πολυπύρηνους επεξεργαστές. Για πρώτη φορά, οι παράλληλοι πόροι θα λειτουργήσουν μόνο με κατάλληλα γραμμένο λογισμικό, χρησιμοποιώντας την τεχνική *πολυνημάτωσης* (*multithreading*) [144].

Το *λογισμικό πολυνημάτωσης* (*multithreading software*) είναι απαραίτητο ώστε να επιτευχθεί η αξιοποίηση των πολλαπλών πυρήνων. Παρείχε κέρδη σε απόδοση προτού εμφανιστούν οι πολυπύρηνοι επεξεργαστές, χάρη στην τεχνολογία της *πολυνημάτωσης* (*multithreading*). Για τους σημερινούς πολυπύρηνους επεξεργαστές ισχύει ότι μπορούν να εκτελούν περισσότερα νήματα από το πλήθος των πυρήνων που διαθέτουν. Αυτό επιτυγχάνεται με τον διπλασιασμό των τμημάτων του επεξεργαστή στα οποία αποθηκεύει την αρχιτεκτονική του κατάσταση, τη διεύθυνση της εντολής που εκτελείται, τα περιεχόμενα των καταχωρητών, τις σημαίες κατάστασης κλπ, χωρίς να διπλασιάζονται τα κυκλώματα που πραγματικά εκτελούν τις εντολές. Αυτό έχει ως αποτέλεσμα, ο επεξεργαστής να εμφανίζεται στο λειτουργικό σύστημα ως δύο ξεχωριστοί επεξεργαστές ή πυρήνες, το οποίο επιτρέπει σε δύο νήματα να εκτελούνται παράλληλα. Η επιτάχυνση αυτής της προσέγγισης επιτυγχάνεται όταν κάποιες από τις μονάδες εκτέλεσης δεν χρησιμοποιούνται από το ένα νήμα ή όταν η διοχέτευση του επεξεργαστή έχει "κολλήσει", είτε επειδή η απαιτούμενη εντολή δεν βρίσκεται στην κρυφή μνήμη εντολών, είτε επειδή μία διακλάδωση προβλέφθηκε λανθασμένα ή μία εντολή *εκτός σειράς* (*out-of-order*) χρειάζεται τα αποτελέσματα μιας άλλης εντολής, που δεν έχει ακόμη εκτελεστεί. Υπό αυτές τις συνθήκες, η δυνατότητα αλλαγής στο δεύτερο νήμα επιτρέπει την αξιοποίηση των ανενεργών πόρων. Υπάρχουν τρείς κύριες προσεγγίσεις πολυνημάτωσης:

1. Η *λεπτή πολυνημάτωση* (*fine-grained multithreading*).

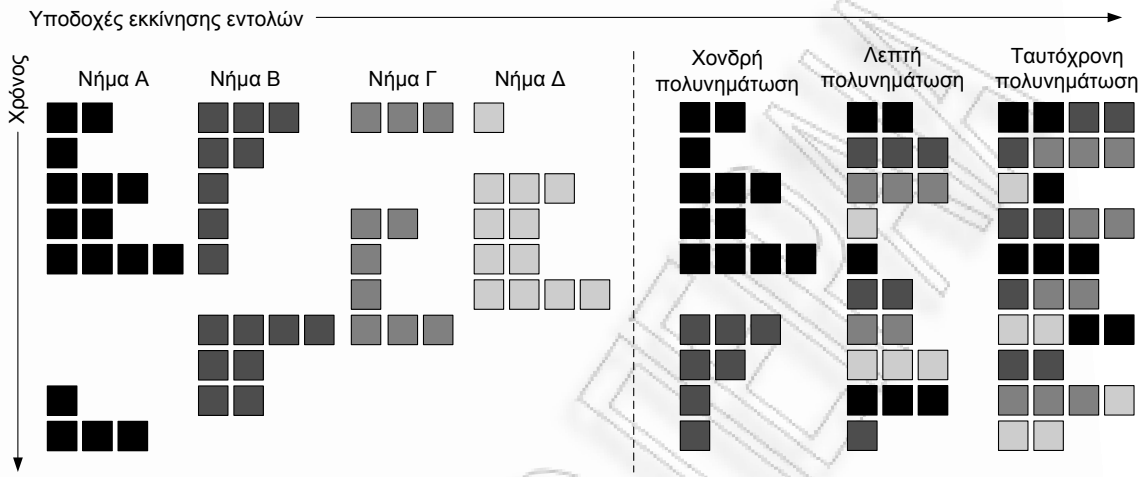
2. Η *χονδρή πολυνημάτωση* (*coarse-grained multithreading*).

3. Η *ταυτόχρονη πολυνημάτωση* (*simultaneous multithreading*).

Η λεπτή πολυνημάτωση εναλλάσσεται μεταξύ των νημάτων σε κάθε εντολή, οδηγώντας σε μια *"πλεκτή"* (*interleaved*) εκτέλεση πολλών νημάτων. Αυτή η πολύπλεξη πραγματοποιείται συχνά με μια μέθοδο *εκ περιτροπής* (*round robin*), αγνοώντας οποιαδήποτε νήματα βρίσκονται σε κατάσταση αδράνειας τη συγκεκριμένη χρονική στιγμή. Η χονδρή πολυνημάτωση προτάθηκε ως μία εναλλακτική προσέγγιση της λεπτής πολυνημάτωσης. Η χονδρή πολυνημάτωση αλλάζει νήματα μόνο όταν προκύπτουν μεγάλες καθυστερήσεις, όπως αστοχίες της κρυφής μνήμης δευτέρου επιπέδου. Αυτή η αλλαγή μετριάζει την ανάγκη εναλλαγής νημάτων και είναι λιγότερο πιθανό να επιβραδυνθεί η εκτέλεση ενός ανεξάρτητου νήματος, εφόσον οι εντολές από τα άλλα νήματα θα ξεκινήσουν μόνον όταν ένα νήμα αντιμετωπίζει μια δαπανηρή καθυστέρηση. Η ταυτόχρονη πολυνημάτωση είναι μια παραλλαγή της πολυνημάτωσης που χρησιμοποιεί τους επιπλέον πόρους ενός υπερβαθμωτού επεξεργαστή για την αξιοποίηση της παραλληλίας σε επίπεδο νήματος και εντολής ταυτόχρονα.

Στην Εικόνα 4.3 παρουσιάζονται εννοιολογικά οι διαφορές στην δυνατότητα ενός επεξεργαστή να αξιοποιήσει τους πόρους υπερβάθμωσης. Στο αριστερό μέρος του σχήματος απεικονίζεται ο τρόπος με τον οποίο θα εκτελούνταν ανεξάρτητα νήματα σε έναν υπερβαθμωτό επεξεργαστή χωρίς υποστήριξη πολυνημάτωσης, ενώ στο δεξί μέρος απεικονίζονται οι δυνατοί συνδυασμοί των τεσσάρων νημάτων ώστε να εκτελεστούν στον επεξεργαστή πιο αποδοτικά με τη χρήση των τριών προσεγγίσεων πολυνημάτωσης. Η οριζόντια διάσταση αντιπροσωπεύει την ικανότητα εκκίνησης εντολών σε κάθε κύκλο ρολογιού. Η κατακόρυφη διάσταση αντιπροσωπεύει μια ακολουθία κύκλων ρολογιού. Ένα άδειο τετράγωνο σημαίνει ότι η αντίστοιχη υποδοχή εκκίνησης εντολών δεν χρησιμοποιείται σε εκείνον τον κύκλο ρολογιού. Τόσο τα γκρίζα όσο και τα μαύρα τετράγωνα αντιστοιχούν σε τέσσερα διαφορετικά νήματα.

Παρά το γεγονός εμφάνισης τεσσάρων επεξεργαστών, στην πραγματικότητα δεν υπάρχουν δύο πλήρεις επεξεργαστές, όπως στην περίπτωση ενός διπύρηνου, αλλά ένας. Ο δεύτερος εικονικός απλώς επωφελείται από τις ανενεργές περιόδους του πρώτου. Ο πολυνηματισμός δεν αποτελεί πάντα εφικτή λύση για τη βελτίωση της απόδοσης ενός επεξεργαστή. Σε περιπτώσεις υλοποίησης με ένα νήμα όπου οι μονάδες εκτέλεσης αξιοποιούνται στο έπακρο, η εισαγωγή πολυνηματισμού δεν θα επιφέρει καμία επιπλέον επιτάχυνση στην απόδοση του επεξεργαστή. Στην τελευταία περίπτωση συγκαταλέγονται οι επεξεργαστές της AMD [139], όπου σε όλες τις υλοποιήσεις τους δεν έχει ενσωματωθεί αυτή η δυνατότητα και σύμφωνα με τα σχέδια της εταιρίας δεν αναμένεται η ενσωμάτωση στο προσεχές μέλλον. Η AMD δεν υιοθέτησε την ενσωμάτωση τεχνικών πολυνημάτωσης δεδομένου ότι οι μονάδες εκτέλεσης στους επεξεργαστές της αξιοποιούνται περισσότερο από 90% με την χρήση μόλις ενός νήματος.



Εικόνα 4.3 Αξιοποίηση λειτουργικών μονάδων ανά μονάδα χρόνου σε ένα επεξεργαστή, (α) χωρίς πολυνημάτωση και (β) με πολυνημάτωση

4.3 Πολυεπεξεργαστές

Η μετάβαση από τους μονοπύρηνους επεξεργαστές στους πολυεπεξεργαστές πραγματοποιήθηκε θέτοντας ως στόχο την βελτίωση της απόδοσης και του παραλληλισμού σε εφαρμογές που απαιτούν εντατικούς υπολογισμούς. Όμως η μετάβαση από έναν σε δυο ή τέσσερις πυρήνες σε συνεπάγεται διπλασιασμό ή τετραπλασιασμό αντίστοιχα της απόδοσης ενός επεξεργαστή. Επομένως, τι κέρδη σε απόδοση αναμένονται σε πραγματική χρήση; Μερικά προγράμματα κλιμακώνονται επαρκώς με αποτέλεσμα να παρατηρηθεί επιτάχυνση κοντά στον αριθμό των πυρήνων. Ενίοτε, αυτό μπορεί να επιφέρει τετραπλασιασμό της ταχύτητας σε τέσσερις πυρήνες, αλλά ακόμη και τριπλασιασμός της ταχύτητας σε τέσσερις πυρήνες κρίνεται αξιόλογη επιτάχυνση. Οτιδήποτε έχει σχετικά ανεξάρτητες λειτουργίες είναι δυνατόν να επιτύχει τέτοιες επιταχύνσεις, για παράδειγμα, αναζητήσεις σε βάσεις δεδομένων.

Η διαφορά μεταξύ ενός μονοπύρηνου και ενός πολυπύρηνου επεξεργαστή μπορεί να παρομοιαστεί σαν τη διαφορά μεταξύ ενός σπορ αυτοκινήτου και ενός σχολικού λεωφορείου. Το πρώτο μπορεί να πιάνει μεγάλες ταχύτητες αλλά μεταφέρει το πολύ δυο άτομα, το δεύτερο κινείται πιο αργά αλλά μεταφέρει περισσότερους ανθρώπους. Αναμφίβολα όμως σήμερα, οι νέοι πολυεπεξεργαστές είναι τόσο ισχυροί που στην ουσία αντιστοιχούν σε ένα λεωφορείο που μπορεί να αναπτύξει μεγάλη ταχύτητα. Για την εκμετάλλευση όμως των επιπλέον δυνατοτήτων που μας προσφέρει ο πολυπύρηνος επεξεργαστής, ώστε να βελτιωθεί η ταχύτητα εκτέλεσης των εντολών μας, απαιτείται ο επαναπρογραμματισμός του λογισμικού μας.

Η αύξηση των επιδόσεων από τη χρήση των πολυεπεξεργαστών απαιτεί τη συγγραφή λογισμικού για των εκάστοτε εφαρμογών ώστε να είναι δυνατόν να διαιρεθούν αυτόματα οι

διεργασίες του προγράμματος και να διαβιβαστούν σε κάθε πυρήνα ξεχωριστά, αλλά και να είναι εφικτό να συγχρονιστεί η παράλληλη επεξεργασία τους και να ελεγχθούν τα δεδομένα που μοιράζονται οι πυρήνες, έτσι ώστε να μην υπάρχουν *συγκρούσεις* (*conflicts*). Προσπαθούμε δηλαδή να γεμίσουμε όλες τις θέσεις στο σχολικό λεωφορείο. Την παρούσα στιγμή, πολλά προγράμματα δεν καταφέρνουν να αξιοποιήσουν του πολλαπλούς πυρήνες. Για παράδειγμα στην περίπτωση ενός τετραπύρηνου επεξεργαστή, γεμίζουν το 25% των διαθέσιμων "θέσεων", αφού αξιοποιούν μόνο τον ένα πυρήνα.

Συγκεκριμένα είδη λογισμικού είναι πιθανό να ωφεληθούν περισσότερο από την πολυπύρηνη προσέγγιση. Κάθε πρόγραμμα το οποίο επεξεργάζεται μεγάλο πλήθος δεδομένων, θα μπορούσε να βελτιστοποιηθεί ως προς τον παραλληλισμό. Αυτό το είδος παραλληλισμού αποκαλείται *παραλληλισμός δεδομένων* (*data-parallelism*). Τα προγράμματα που επεξεργάζονται φωτογραφίες, βίντεο ή επιστημονικά δεδομένα έχουν την τάση να εκμεταλλεύονται τον παραλληλισμό δεδομένων. Ένα άλλο είδος παραλληλισμού είναι ο *παραλληλισμός διεργασιών* (*task-parallelism*). Το είδος αυτό παρουσιάζει αυξημένο βαθμό δυσκολίας, η οποία έγκειται στον εντοπισμό των τμημάτων του κώδικα που μπορούν να εκτελεστούν ταυτόχρονα.

Ένας ιδιαίτερα σημαντικός νόμος στα παράλληλη συστήματα, είναι ο *νόμος του Amdahl* (*Amdahl Law*). Ο Νόμος του Amdahl είναι μία εξίσωση που υπολογίζει το κέρδος στην απόδοση ενός παράλληλου συστήματος: Αν όλος ο φόρτος εργασίας μπορεί να παραλληλιστεί, τότε το κέρδος είναι ανάλογο του αριθμού των επεξεργαστών. Αν ακόμη κι ένα μικρό ποσοστό του προγράμματος δεν μπορεί να εκτελεστεί παράλληλα, το επίπεδο της επιτάχυνσης μειώνεται δραματικά.

Παράδειγμα: Έστω ότι θέλουμε να επιτύχουμε επιτάχυνση ενός υπολογισμού 80 σε ένα σύστημα με 100 επεξεργαστές. Τι ποσοστό του αρχικού υπολογισμού μπορεί να είναι ακολουθιακό;

Απάντηση: Ο νόμος του Amdahl ορίζεται ως εξής:

$$\text{Speedup} = \frac{1}{\frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} + (1 - \text{Fraction}_{\text{enhanced}})}$$

Για την απλούστευση του παραδείγματος, θεωρείστε ότι το πρόγραμμα λειτουργεί σε δύο μόνο καταστάσεις: την παράλληλη όπου εκμεταλλεύεται όλους τους επεξεργαστές, ή αλλιώς *προηγμένη λειτουργία* (*enhanced mode*) και την ακολουθιακή που χρησιμοποιεί μόλις έναν επεξεργαστή. Κατά συνέπεια, η επιτάχυνση της προηγμένης κατάστασης είναι ο αριθμός των επεξεργαστών ($\text{Speedup}_{\text{enhanced}} = 100$), ενώ το ποσοστό της προηγμένης κατάστασης είναι ο

χρόνος που δαπανάτε στην παράλληλη κατάσταση ($Fraction_{enhanced}$ = $Fraction_{parallel}$). Εφαρμόζοντας τις τιμές του παραδείγματος, η παραπάνω εξίσωση διαμορφώνεται ως εξής:

$$80 = \frac{1}{\frac{Fraction_{parallel}}{100} + (1 - Fraction_{parallel})} \quad =>$$

$$80 = Fraction_{parallel} + 80 * (1 - Fraction_{parallel}) = 1 \quad =>$$

$$80 - 79.2 * Fraction_{parallel} = 1 \quad =>$$

$$Fraction_{parallel} = 0.9975$$

Για να επιτύχουμε επιτάχυνση 80 σε ένα σύστημα με 100 επεξεργαστές, μόνο το 0.25% του αρχικού υπολογισμού μπορεί να είναι ακολουθιακό. Για την επίτευξη γραμμικής επιτάχυνσης ισοδύναμης με τον αριθμό των επεξεργαστών, απαιτείται πλήρης παραλληλισμός του προγράμματος.

4.3.1 Συμμετρικοί πολυεπεξεργαστές

Ο όρος *συμμετρικοί πολυεπεξεργαστές* (*Symmetric Multiprocessors, SMP*) αναφέρεται τόσο στην αρχιτεκτονική του υλικού του υπολογιστή, όσο και στην συμπεριφορά του λειτουργικού συστήματος. Ένας συμμετρικός πολυεπεξεργαστής μπορεί να θεωρηθεί ως ένα μεμονωμένο σύστημα υπολογιστή με τα ακόλουθα χαρακτηριστικά:

1) Υπάρχουν δύο ή περισσότεροι όμοιοι επεξεργαστές.

2) Οι επεξεργαστές μοιράζονται την ίδια μνήμη και τις συσκευές εισόδου-εξόδου, καθώς επίσης και το κοινό μέσο επικοινωνίας, ούτως ώστε οι προσπελάσεις στην μνήμη να απαιτούν τον *ίδιο χρόνο προσπέλασης* (*Uniform Memory Access, UMA*) από όλους τους επεξεργαστές.

3) Όλοι οι επεξεργαστές διαμοιράζονται την πρόσβαση στις συσκευές εισόδου-εξόδου, είτε μέσω των ίδιων καναλιών επικοινωνίας, είτε μέσω διαφορετικών καναλιών που παρέχουν πρόσβαση στις αναφερόμενες συσκευές.

4) Όλοι οι επεξεργαστές είναι ικανοί να χειριστούν όλες τις πιθανές διεργασίες.

Ένα σύστημα ενός συμμετρικού πολυεπεξεργαστή έχει πολλά πλεονεκτήματα σε σχέση με τον μονοπύρηνο επεξεργαστή:

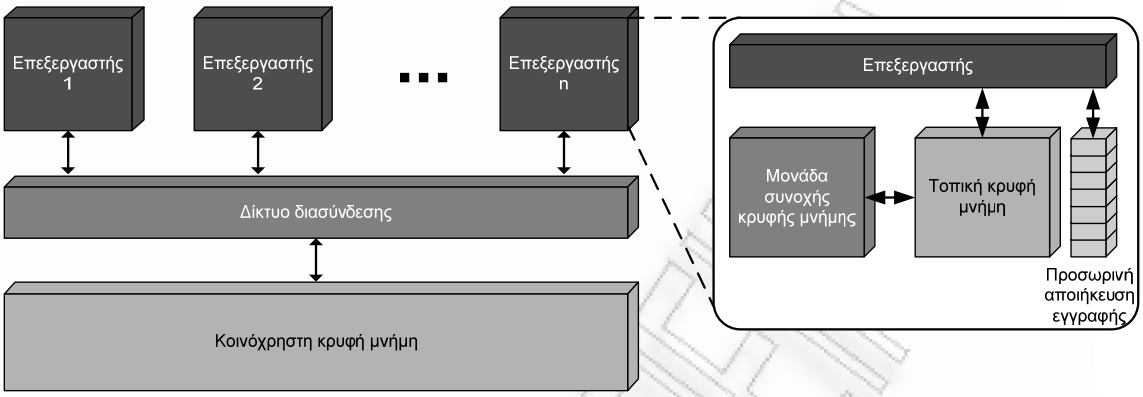
- *Απόδοση* (*performance*): Εάν μία εργασία είναι δυνατό να διαιρεθεί και κάποιες από αυτές τις υποδιαιρέσεις μπορούν να εκτελεστούν παράλληλα, τότε ένα σύστημα συμμετρικού πολυεπεξεργαστή είναι δυνατό να βελτιώσει το συνολικό χρόνο εκτέλεσης της εργασίας σε σχέση με ένα μονοπύρηνο επεξεργαστή (Εικόνα 4.4).

- *Διαθεσιμότητα* (*availability*): Δεδομένου ότι όλοι οι επεξεργαστές είναι δυνατό να εκτελέσουν τις ίδιες διεργασίες, η *αποτυχία* (*failure*) ενός από αυτούς δεν διακόπτει το σύστημα. Απεναντίας, το σύστημα είναι σε θέση να συνεχίσει την εκτέλεση της διεργασίας μειώνοντας ενδεχομένως την συνολική του απόδοση.

- *Κλιμάκωση* (*scaling*): Οι προμηθευτές συμμετρικών πολυεπεξεργαστών είναι δυνατόν να παρέχουν προϊόντα με διαφορετικά χαρακτηριστικά κόστους–απόδοσης που καθορίζονται από τον αριθμό των επεξεργαστών.

Εικόνα 4.4 Εκτέλεση διεργασιών (α) οι διεργασίες εκτελούνται η μία μετά την άλλη, (β) κάποιες από τις διεργασίες είναι δυνατό να εκτελεστούν παράλληλα.

Αξίζει να σημειωθεί ότι αυτά μπορεί να είναι πιθανά και όχι εγγυημένα πλεονεκτήματα για τους συμμετρικούς πολυεπεξεργαστές. Όπως προαναφέραμε, η επιτάχυνση ενός πολυεπεξεργαστή εξαρτάται από την ίδια την διεργασία. Κατά συνέπεια τόσο οι προγραμματιστές λογισμικού, όσο και τα λειτουργικά συστήματα, πρέπει να εκμεταλλεύονται τις έμφυτες ιδιότητες των πολυεπεξεργαστών ώστε να επιτύχουν μία αποδεκτή επιτάχυνση για κάθε δεδομένη διεργασία.



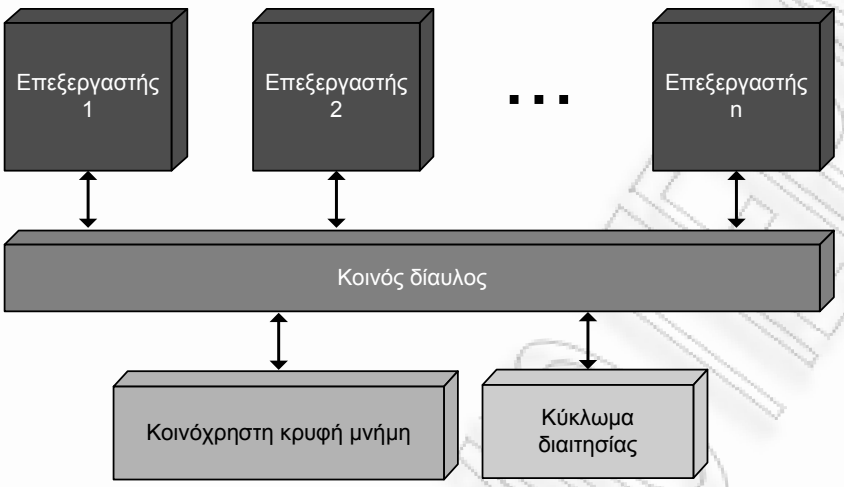
Εικόνα 4.5 Τυπικό διάγραμμα συμμετρικού πολυεπεξεργαστή

Η Εικόνα 4.5 παρουσιάζει το γενικό διάγραμμα της αρχιτεκτονικής των συμμετρικών πολυεπεξεργαστών που αποτελεί αντικείμενο μελέτης της παρούσας διδακτορικής διατριβής. Η αρχιτεκτονική αυτή αποτελείται από *n* ίδιους πυρήνες, οι οποίοι περιέχουν και την τοπική κρυφή μνήμη πρώτου ή δευτέρου επιπέδου. Κάθε επεξεργαστής αποκτά πρόσβαση στην κοινόχρηστη κρυφή μνήμη δευτέρου ή τρίτου επιπέδου του συστήματος, μέσω ενός *δικτύου διασύνδεσης* (*interconnection network*). Οι επεξεργαστές επικοινωνούν μεταξύ τους μέσω της κοινόχρηστης κρυφής μνήμης καθώς και μέσω της *κύριας μνήμης* (*main memory*) του συστήματος (η κύρια μνήμη δεν απεικονίζεται σε αυτή την εικόνα). Η *μνήμη προσωρινής αποθήκευσης της εγγραφής* (*write buffers*) κάθε *τοπικής κρυφής μνήμης* (*local cache memory*) διατηρεί προσωρινά τα δεδομένα εγγραφής προς την κοινόχρηστη κρυφή μνήμη επιταχύνοντας το υποσύστημα μνήμης. Η μονάδα *συνοχής κρυφής μνήμης* (*cache coherence unit*) διατηρεί την συνέπεια μεταξύ της τοπικής κρυφής μνήμης των επεξεργαστών, με την κοινόχρηστη κρυφή μνήμη του συστήματος. Η συνοχή της κρυφής μνήμης και ένας συγκεκριμένος αλγόριθμος συνοχής παρουσιάζονται εκτενέστερα στην Ενότητα 4.3.2. Στην αρχιτεκτονική των συμμετρικών πολυεπεξεργαστών οι δύο πιο κοινές αρχιτεκτονικές επιλογές για το δίκτυο διασύνδεσης είναι:

- Ο *κοινός δίαυλος* (*shared bus*).
- Ο *σταυρωτός μεταγωγέας* (*crossbar switch*).

Σύμφωνα με την αρχιτεκτονική που παρουσιάζεται στην Εικόνα 4.6, το σύστημα του πολυεπεξεργαστή χρησιμοποιεί ένα *κοινό δίαυλο επικοινωνίας* (*shared bus*), όπου ένα *κύκλωμα διαιτησίας* (*bus arbiter*) καθορίζει ποιος από τους *n* επεξεργαστες αποκτά πρόσβαση κάθε φορά προς την κοινόχρηστη κρυφή μνήμη. Το κύκλωμα διαιτησίας διανέμει συνήθως το δίαυλο επικοινωνίας στους επεξεργαστές, ακολουθώντας μία *εξυπηρέτηση εκ' περιτροπής* (*round robin*). Η αρχιτεκτονική αυτή αποτελεί την πιο κοινή οργάνωση δικτύου διασύνδεσης για *προσωπικούς επεξεργαστές* (*personal computers*), *σταθμούς εργασίας* (*workstations*) και *διακομιστές* (*servers*).

Η δομή και η *διεπαφή* (*interface*) είναι όμοιες με έναν μονοπύρηνο επεξεργαστή κοινού διαύλου.



Εικόνα 4.6 Αρχιτεκτονική συμμετρικού επεξεργαστή με κοινό δίαυλο επικοινωνίας

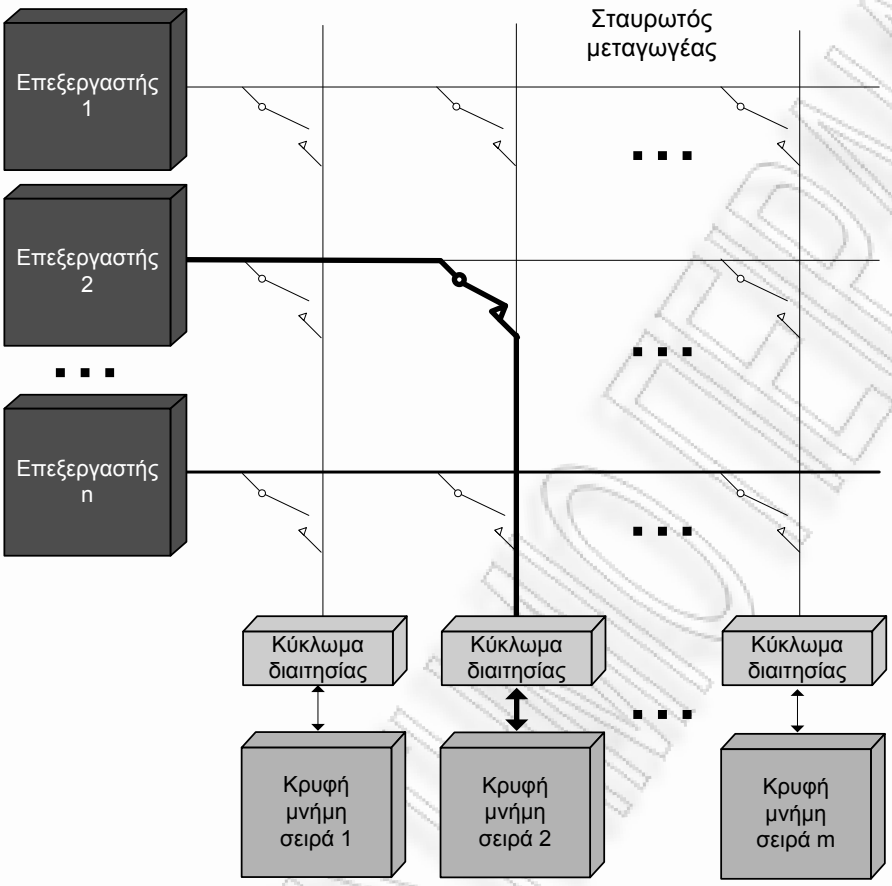
Η γενική αρχιτεκτονική κοινού διαύλου για ένα συμμετρικό πολυεπεξεργαστή προϋποθέτει ότι είναι εφικτός ο διαχωρισμός των μονάδων (επεξεργαστές και κοινόχρηστη κρυφή μνήμη) του διαύλου επικοινωνίας, προκειμένου να καθοριστεί η πηγή και ο προορισμός της εκάστοτε αίτησης εγγραφής ή ανάγνωσης. Επίσης, κάθε επεξεργαστής του συστήματος μπορεί να οριστεί προσωρινά ως *κύριος κάτοχος του διαύλους* (*master*). Όταν ένας επεξεργαστής κατέχει την κυριότητα του διαύλου, όλοι οι άλλοι επεξεργαστές είναι αδύνατον να προσπελάσουν την κοινόχρηστη κρυφή μνήμη. Ο καθορισμός του κατόχου του διαύλου πραγματοποιείται μέσω του *κυκλώματος διαιτησίας* (*bus arbiter*).

Η αρχιτεκτονική του κοινού διαύλου επικοινωνίας, παρουσιάζει πολλά πλεονεκτήματα. Το κύριο πλεονέκτημα είναι το μικρό κόστος υλοποίησης. Επιπλέον, η *φυσική διεπαφή* (*physical interface*), καθώς επίσης και η *λογική διευθυνσιοδότησης* (*addressing*) και *διαιτησίας* (*arbitration*) κάθε επεξεργαστή του συστήματος, παραμένουν τα ίδια με την περίπτωση του μονοπύρηνου επεξεργαστή. Παράλληλα, είναι ιδιαίτερα εύκολη η επέκταση του συστήματος, ενσωματώνοντας επιπλέον επεξεργαστές "πάνω" στον κοινό δίαυλο επικοινωνίας χωρίς να είναι απαραίτητη η τροποποίηση του διαύλου. Επίσης, ο κοινός δίαυλος αποτελεί ένα *παθητικό μέσο* (*passive medium*) με αποτέλεσμα η λειτουργική αποτυχία οποιασδήποτε μονάδας να μην επιφέρει την παύση της λειτουργίας ολόκληρου του συστήματος. Συνοψίζοντας, σε αυτή την αρχιτεκτονική είναι πιο εύκολο να υλοποιηθεί η μονάδα συνοχής της κρυφής μνήμης, δεδομένου ότι χρειάζεται να παρακολουθεί μόλις έναν δίαυλο επικοινωνίας.

Το κύριο μειονέκτημα του κοινού διαύλου επικοινωνίας είναι η ανεπαρκής απόδοση, η οποία μειώνεται ολοένα και περισσότερο καθώς αυξάνεται ο αριθμός των επεξεργαστών του συστήματος. Το πρόβλημα απορρέει από το γεγονός ότι όλες οι αναφορές προς την κοινόχρηστη κρυφή μνήμη δρομολογούνται μέσω ενός κοινού διαύλου με αποτέλεσμα η απόδοση του συστήματος να περιορίζεται από το εύρος ζώνης του ίδιου του διαύλου. Για παράδειγμα, αν δύο επεξεργαστές επιθυμούν την ίδια χρονική στιγμή να επικοινωνήσουν με την κοινόχρηστη κρυφή μνήμη, ο ένας από αυτούς αναγκάζεται να περιμένει έως ότου απελευθερωθεί ο κοινός δίαυλος από τον άλλο επεξεργαστή. Το πρόβλημα γίνεται εντονότερο στην περίπτωση όπου δέκα επεξεργαστές πραγματοποιούν ταυτόχρονες αιτήσεις προς τον δίαυλο. Ένα επιπλέον μειονέκτημα είναι ότι το σύστημα είναι ευαίσθητο σε βλάβες του κοινού διαύλου. Μία βλάβη στον κοινό δίαυλο συνεπάγεται την καταστροφή της επικοινωνίας του συστήματος και κατ' επέκταση την παύση λειτουργίας του πολυεπεξεργαστή.

Στον αντίποδα και σύμφωνα με την αρχιτεκτονική που απεικονίζεται στην Εικόνα 4.7, ο κοινός δίαυλος επικοινωνίας αντικαθίσταται από ένα *σταυρωτό μεταγωγέα* (*crossbar switch*) προκειμένου να πραγματοποιηθεί η σύνδεση μεταξύ των *n* επεξεργαστών με τις *m σειρές της κοινόχρηστης κρυφής μνήμης* (*shared cache banks*). Η κοινόχρηστη κρυφή μνήμη έχει διαιρεθεί σε πολλές διαφορετικές *σειρές* (*banks*) καθεμία από τις οποίες συνδέεται σε διαφορετικό διασυνδετικό δίαυλο του σταυρωτού μεταγωγέα. Κάθε σειρά της κοινόχρηστης κρυφής μνήμης ενσωματώνει ένα διαιτητή κρυφής μνήμης, ο οποίος διαχειρίζεται τις συναλλαγές από τους πυρήνες επεξεργαστών προς την σειρά αυτή. Ο σταυρωτός μεταγωγέας συνδέει κατ' εντολή τους επεξεργαστές με τις σειρές της κοινόχρηστης κρυφής μνήμης. Η Εικόνα 4.7 απεικονίζει επίσης με απλοποιημένο τρόπο την αρχή λειτουργίας ενός σταυρωτού μεταγωγέα. Ανάλογα με το ποιος διακόπτης θα κλείσει, συνδέεται ο αντίστοιχος επεξεργαστής με την αντίστοιχη σειρά της κοινόχρηστης κρυφής μνήμης του συστήματος (Processor 2 με Shared Cache Bank 2 στο συγκεκριμένο παράδειγμα).

Εσωτερικά, ο σταυρωτός μεταγωγέας διαθέτει μια δομή *μεταγωγής* (*switching fabric*), η οποία έχει τη δυνατότητα να μεταβιβάζει τα πακέτα που εισέρχονται από κάθε επεξεργαστή σε οποιαδήποτε σειρά της κοινόχρηστης κρυφής μνήμης. Μπορεί δηλαδή να αλλάζει δυναμικά τις διασυνδέσεις εισόδων / εξόδων ανάλογα με τις αιτήσεις εγγραφής και ανάγνωσης των επεξεργαστών. Οι συνδέσεις δημιουργούνται στιγμιαία σύμφωνα με τις ανάγκες του συστήματος και βάση, όχι ενός προκαθορισμένου στατικού τρόπου, αλλά με εντολές που προκαλούνται από τις πληροφορίες που εμπεριέχονται στα ίδια τα πακέτα των εντολών των επεξεργαστών που εισέρχονται από τις πόρτες του κόμβου του σταυρωτού μεταγωγέα.



Εικόνα 4.7 Αρχιτεκτονική συμμετρικού επεξεργαστή με σταυρωτό μεταγωγέα

Το σχέδιο διασύνδεσης του σταυρωτού μεταγωγέα αποτελεί την πιο κοινή επιλογή διασύνδεσης σύγχρονων συστημάτων πολυεπεξεργαστών υψηλών επιδόσεων, καθώς επιτρέπει την ταυτόχρονη πρόσβαση στις διαφορετικές σειρές της κοινόχρηστης κρυφής μνήμης, επιτυγχάνοντας υψηλότερο εύρος ζώνης επικοινωνίας μεταξύ των επεξεργαστών και της κοινόχρηστης κρυφής μνήμης. Συνεπώς, δύο επεξεργαστές είναι δυνατόν να επικοινωνήσουν με δύο διαφορετικές σειρές της κρυφής μνήμης ταυτόχρονα χωρίς να δημιουργείται ανταγωνισμός διαύλου μεταξύ των επεξεργαστών. Όμοια με την περίπτωση του κοινού διαύλου επικοινωνίας, ο σταυρωτός μεταγωγέας αποτελεί ένα παθητικό μέσο, με αποτέλεσμα η αποτυχία οποιασδήποτε μονάδας επεξεργαστή να μην προκαλέσει την παύση της λειτουργίας ολόκληρου του συστήματος.

Στα μειονεκτήματα αυτής της αρχιτεκτονικής συγκαταλέγεται το ακριβό κόστος υλοποίησης και η μεγάλη πολυπλοκότητα, ιδιαίτερα όσο αυξάνεται ο αριθμός των επεξεργαστών. Ωστόσο, είναι δυνατόν να κρατηθεί το κόστος σε σχετικά χαμηλά επίπεδα, μειώνοντας τις διαθέσιμες συνδέσεις μεταξύ επεξεργαστών και των σειρών της κοινόχρηστης

κρυφής μνήμης. Το γεγονός αυτό έχει αντίκτυπο στην συνολική απόδοση της επικοινωνίας του πολυεπεξεργαστή και για το λόγο αυτό δεν χρησιμοποιείται σε μεγάλο βαθμό. Επίσης, η πολυπλοκότητα υλοποίησης των μηχανισμών συνοχής της κρυφής μνήμης αυξάνεται δεδομένου ότι ο ελεγκτής συνοχής κρυφής μνήμης θα πρέπει να παρακολουθεί όλους του διασυνδετικούς διαύλους του σταυρωτού μεταγωγέα. Ένα ακόμα μειονέκτημα είναι ότι ο ανταγωνισμός του διαύλου επικοινωνίας εξακολουθεί να υφίσταται σε περιπτώσεις που δύο οι περισσότεροι επεξεργαστές πραγματοποιούν αιτήσεις εγγραφής ή ανάγνωσης στην ίδια σειρά ταυτόχρονα.

Ο Πίνακας 4.1 συνοψίζει τα χαρακτηριστικά των δύο διαφορετικών σχημάτων διασύνδεσης συμμετρικών πολυεπεξεργαστών. Ο κοινός δίαυλος, δεδομένου του περιορισμένου εύρους ζώνης που διαθέτει, είναι αδύνατο να υποστηρίξει περισσότερους από 32 πυρήνες συνδεδεμένους σε ένα μοναδικό δίαυλο επικοινωνίας. Αντίθετα, ο σταυρωτός μεταγωγέας δεδομένου ότι αυξάνει των αριθμό των διασυνδέσεων μεταξύ των επεξεργαστών και των σειρών της κοινόχρηστης κρυφής μνήμης, παρέχει μεγαλύτερο εύρος ζώνης και μπορεί να υποστηρίξει έως και 256 πυρήνες.

Χαρακτηριστικά	Κοινός δίαυλος	Σταυρωτός μεταγωγέας
Απόδοση	Χαμηλή	**Υψηλή**
Ανταγωνισμός διαύλου	Υψηλός	**Χαμηλός**
Κόστος υλοποίησης	**Χαμηλό**	Υψηλό
Πολυπλοκότητα μηχανισμού συνοχής κρυφής μνήμης	**Χαμηλή**	Υψηλή
Αριθμός επεξεργαστών	Έως 36	**Έως 256**

Πίνακας 4.1 Χαρακτηριστικά των αρχιτεκτονικών διασύνδεσης ενός συστήματος συμμετρικού πολυεπεξεργαστή

4.3.2 Συνοχή κρυφής μνήμης πολυεπεξεργαστών

Το πρόβλημα συνοχής μεταξύ κοινόχρηστης κρυφής μνήμης και τοπικής κρυφής μνήμης έγκειται στο γεγονός ότι οι μεταβλητές που αλλάζουν στην τοπική κρυφή μνήμη ενός επεξεργαστή δεν αλλάζουν και στις τοπικές κρυφές μνήμες των άλλων επεξεργαστών που ενδεχομένως αποθηκεύουν την ίδια μεταβλητή. Ως εκ τούτου ενδέχεται κάποια στιγμή η ίδια μεταβλητή να βρεθεί με διαφορετικές τιμές σε διαφορετικές τοπικές κρυφές μνήμες.

Η *μονάδα συνοχής κρυφής μνήμης* (*cache coherence unit*) στους συμμετρικούς πολυεπεξεργαστές εξασφαλίζει τη συνέπεια μεταξύ των κοινών πόρων, όπως οι τοπικές κρυφές μνήμες και η κοινόχρηστη κρυφή μνήμη. Η μονάδα συνοχής κρυφής μνήμης κάθε επεξεργαστή διατηρεί την συνέπεια μεταξύ της τοπικής κρυφής μνήμης με την κοινόχρηστη στην περίπτωση του κοινού διαύλου επικοινωνίας και με τις σειρές της κοινόχρηστης κρυφής μνήμης στην

περίπτωση του σταυρωτού μεταγωγέα. Τα πρωτόκολλα για την διατήρηση της συνοχής σε πολλούς επεξεργαστές ονομάζονται *πρωτόκολλα συνοχής κρυφής μνήμης* (*cache coherence protocols*).

Τα πρωτόκολλα συνοχής κρυφής μνήμης διαχωρίζονται σε δύο κατηγορίες. Η πρώτη κατηγορία περιλαμβάνει τα πρωτόκολλα *κατασκοπείας* (*snooping*) [144] και αποτελεί την δημοφιλέστερη προσέγγιση. Στα πρωτόκολλα κατασκοπείας, η μονάδα συνοχής κάθε επεξεργαστή παρατηρεί τον δίαυλο του συστήματος έως ότου πραγματοποιηθεί μια εγγραφή σε μια θέση μνήμης κοινού πόρου. Εάν η εγγραφή πραγματοποιηθεί σε μια θέση της κοινόχρηστης κρυφής μνήμης όπου η τοπική κρυφή μνήμη ενός επεξεργαστή έχει ένα αντίγραφο, τότε ο ελεγκτής της κρυφής μνήμης πρέπει να ακυρώσει όλα τα δεδομένα της θέσης αυτής. Η δεύτερη κατηγορία περιλαμβάνει τα πρωτόκολλα *καταλόγου* (*directory-based*) και δεν απαιτεί κάποιο κοινό μέσο παρακολούθησης. Όποιος επεξεργαστής θελήσει να τροποποιήσει κάποιον πόρο, πρέπει να επικοινωνήσει με κάποια κοινόχρηστη κρυφή μνήμη η οποία παρέχει κατάλόγο με τους επεξεργαστές που διαθέτουν τον πόρο στην τοπική κρυφή μνήμη τους, ώστε να τους ειδοποιεί αυτούς για τις επικείμενες. Το πρωτόκολλο MESI το οποίο θα αναλυθεί στην επόμενη υποενότητα εντάσσεται στα πρωτόκολλα παρακολούθησης.

Τα πρωτόκολλα παρακολούθησης χωρίζονται σε δύο κατηγορίες. Στην πρώτη κατηγορία ανήκουν τα πρωτόκολλα *εγγραφής-ενημέρωσης* (*write-update*) στα οποία όταν μια μεταβλητή τροποποιηθεί από έναν επεξεργαστή, η νέα της τιμή μεταδίδεται σε όλους τους άλλους επεξεργαστές, με αποτέλεσμα όσες τοπικές κρυφές μνήμες έχουν τη μεταβλητή να την ενημερώσουν αμέσως με τη νέα τιμή. Στη δεύτερη κατηγορία ανήκουν τα πρωτόκολλα *εγγραφής-ακύρωσης* (*write-invalidate*) στα οποία όταν μια μεταβλητή τροποποιηθεί, η νέα τιμή δεν μεταφέρεται στους υπόλοιπους επεξεργαστές. Ωστόσο, όποια τοπική κρυφή μνήμη περιέχει την μεταβλητή, την ακυρώνει, οπότε όταν τη ζητήσει ο επεξεργαστής θα προκληθεί αστοχία και θα προσκομιστεί από την κοινόχρηστη κρυφή μνήμη.

4.3.2.1 Το πρωτόκολλο MESI

Το πρωτόκολλο MESI είναι το πιο διαδεδομένο όσον αφορά τα πρωτόκολλα παρακολούθησης εγγραφής-ακύρωσης. Στο πρωτόκολλο αυτό, η βασική προϋπόθεση για την τροποποίηση μιας μεταβλητής είναι η κατοχή της *αποκλειστικότητας εγγραφής/ανάγνωσης* (*exclusive read/write*). Η ακύρωση μιας μεταβλητής στις υπόλοιπες τοπικές κρυφές μνήμες πραγματοποιείται τη στιγμή που κάποια τοπική κρυφή μνήμη ζητήσει την αποκλειστικότητα της μεταβλητής. Η αποκλειστικότητα παραμένει στη τοπική κρυφή μνήμη που τη ζήτησε, μέχρι κάποιος άλλος επεξεργαστής να ζητήσει την μεταβλητή. Όσο διατηρείται η αποκλειστικότητα, ο επεξεργαστής μπορεί να τροποποιήσει ανενόχλητα την μεταβλητή στην τοπική κρυφή μνήμη του, χωρίς να δημιουργήσει επιπλέον κίνηση στο δίαυλο. Το πρωτόκολλο περιλαμβάνει τις εξής τέσσερις καταστάσεις:

- *Άκυρο* (*Invalid*). Η τοπική κρυφή μνήμη δεν έχει έγκυρο αντίγραφο της μεταβλητής. Το έγκυρο αντίγραφο μπορεί να υπάρχει είτε στην κοινόχρηστη κρυφή μνήμη, είτε στη τοπική κρυφή μνήμη κάποιου άλλου επεξεργαστή.

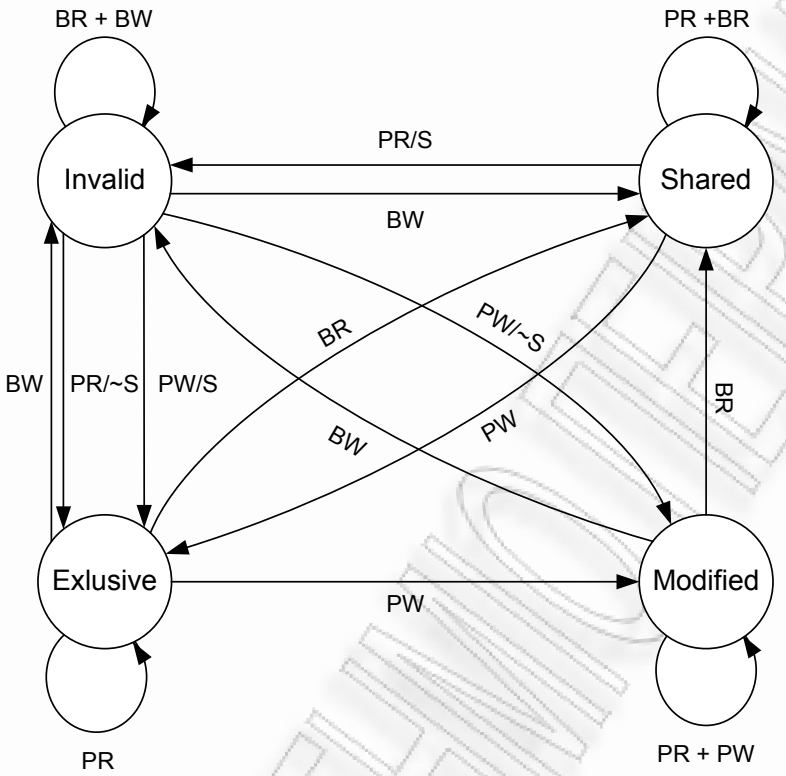
- *Αποκλειστικό* (*Exclusive*). Η μεταβλητή της συγκεκριμένης τοπικής κρυφής μνήμης είναι η πιο έγκυρη και πρόσφατη (τροποποιημένη από τον επεξεργαστή που ανήκει η τοπική κρυφή μνήμη). Το αντίγραφο στην κοινόχρηστη κρυφή μνήμη είναι επίσης το πιο πρόσφατο έγκυρο αντίγραφο της μεταβλητής. Οι υπόλοιπες τοπικές κρυφές μνήμες δεν διαθέτουν αντίγραφο της μεταβλητής.

- *Κοινό* (*Shared*). Η μεταβλητή της συγκεκριμένης τοπικής κρυφής μνήμης είναι η πιο έγκυρη και πρόσφατη. Οι υπόλοιποι επεξεργαστές, όπως και η κοινόχρηστη κρυφή μνήμη, μπορούν να κρατούν αντίγραφα της μεταβλητής καθώς κανένας επεξεργαστής δεν έχει τροποποιήσει ακόμα την μεταβλητή.

- *Τροποποιημένο* (*Modified*). Η μεταβλητή της συγκεκριμένης τοπικής κρυφής μνήμης είναι η πιο έγκυρη και πρόσφατη. Το αντίγραφο που έχει η κοινόχρηστη κρυφή μνήμη είναι παλαιό (λανθασμένο), ενώ κανένας άλλος επεξεργαστής δε διαθέτει αντίγραφο της μεταβλητής στην δική του τοπική κρυφή μνήμη.

Στην Εικόνα 4.8 παρουσιάζεται το διάγραμμα καταστάσεων για το πρωτόκολλο MESI. Οι μεταβάσεις PR και PW αναφέρονται σε αιτήσεις για αναγνώσεις και εγγραφές, αντίστοιχα, από τον ίδιο επεξεργαστή, ενώ οι μεταβάσεις BR και BW αναφέρονται σε αιτήσεις για αναγνώσεις και εγγραφές, αντίστοιχα, που παρατηρήθηκαν στο δίαυλο (δηλαδή από άλλους επεξεργαστές). Με S και ~S αναφέρονται οι περιπτώσεις όπου η μεταβλητή υπό εξέταση περιέχεται (ή δεν περιέχεται αντίστοιχα) σε κάποια τοπική κρυφή μνήμη ενός άλλου επεξεργαστή.

Ξεκινώντας από την άκυρη κατάσταση (invalid), αν ο επεξεργαστής ζητήσει να διαβάσει την μεταβλητή θα αποτύχει, οπότε θα ελέγξει τους υπόλοιπους επεξεργαστές για την ύπαρξη έγκυρων αντιγράφων στη τοπική κρυφή μνήμη τους. Αν βρει έγκυρο αντίγραφο, τότε το λαμβάνει στην τοπική κρυφή μνήμη του και η μεταβλητή μεταβαίνει στην κοινή κατάσταση (shared). Αν δεν βρει έγκυρο αντίγραφο, τότε λαμβάνει την μεταβλητή από την κοινόχρηστη κρυφή μνήμη και η μεταβλητή μεταβαίνει στην αποκλειστική κατάσταση (exclusive). Αν η μεταβλητή δεχθεί αίτηση εγγραφής ή ανάγνωσης από το δίαυλο, τότε παραμένει στην άκυρη κατάσταση (invalid). Αν η μεταβλητή ζητηθεί για εγγραφή από τον ίδιο επεξεργαστή, τότε αν η μεταβλητή δεν υπάρχει στις άλλες τοπικές κρυφές μνήμες, η μεταβλητή μεταβαίνει απευθείας στην τροποποιημένη κατάσταση (modified), ενώ αν υπάρχει, τότε η μεταβλητή μεταβαίνει στην αποκλειστική κατάσταση (exclusive).



Εικόνα 4.8 Διάγραμμα καταστάσεων για το πρωτόκολλο MESI

Όταν η μεταβλητή βρίσκεται στην κοινή κατάσταση (shares) και πραγματοποιηθεί αίτηση εγγραφής για τη συγκεκριμένη μεταβλητή από το δίαυλο, τότε μεταβαίνει στην άκυρη κατάσταση (invalid). Αν η μεταβλητή ζητηθεί για εγγραφή από τον ίδιο επεξεργαστή, τότε μεταβαίνει στην αποκλειστική κατάσταση (exclusive). Στην περίπτωση που η μεταβλητή ζητηθεί για ανάγνωση είτε από το δίαυλο, είτε από τον ίδιο τον επεξεργαστή, τότε η μεταβλητή παραμένει στην ίδια κατάσταση.

Όταν η μεταβλητή βρίσκεται στην αποκλειστική κατάσταση (exclusive) και πραγματοποιηθεί μια αίτηση εγγραφής από το δίαυλο, τότε η μεταβλητή μεταβαίνει στην άκυρη κατάσταση (invalid). Αν πραγματοποιηθεί αίτηση ανάγνωσης από το δίαυλο, τότε η μεταβλητή μεταβαίνει στην κοινή κατάσταση (shared). Στην περίπτωση που πραγματοποιηθεί αίτηση εγγραφής της μεταβλητής από τον ίδιο επεξεργαστή, τότε η μεταβλητή μεταβαίνει στην τροποποιημένη κατάσταση (modified). Επίσης, αν πραγματοποιηθεί αίτηση ανάγνωσης από το δίαυλο, τότε η μεταβλητή παραμένει στην αποκλειστική κατάσταση (exclusive).

Όταν η μεταβλητή βρίσκεται στην τροποποιημένη κατάσταση και δεχθεί μια αίτηση εγγραφής από το δίαυλο, τότε μεταβαίνει στην άκυρη κατάσταση (invalid). Αν δεχθεί μια αίτηση ανάγνωσης από το δίαυλο, τότε μεταβαίνει στην κοινή κατάσταση (shared). Επίσης, αν

δεχθεί μια αίτηση εγγραφής ή ανάγνωσης από τον ίδιο τον επεξεργαστή, τότε η μεταβλητή παραμένει στην τροποποιημένη κατάσταση (modified).

4.4 Ευρέως διαδεδομένοι πυρήνες συμμετρικών πολυεπεξεργαστών

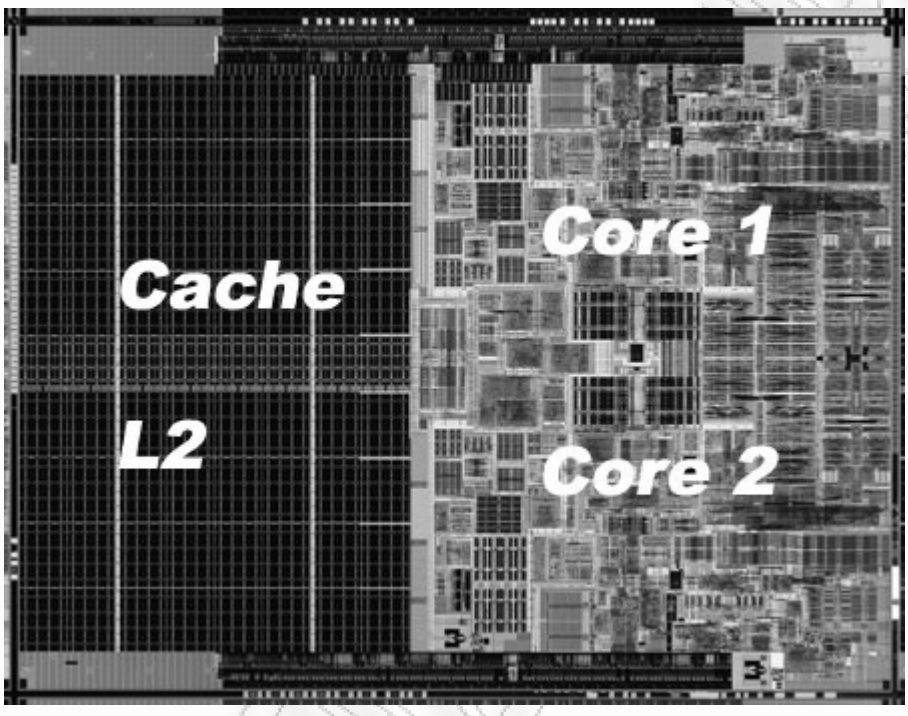
Σε αυτήν την ενότητα αναφέρουμε σύντομα δύο χαρακτηριστικές κατηγορίες συμμετρικών πολυεπεξεργαστών που κυριαρχούν στην αγορά και τα πεδία εφαρμογών τους. Στην πρώτη κατηγορία ανήκει ο συμμετρικός πολυεπεξεργαστής Core™ 2 [141] της εταιρίας Intel [138] που στοχεύει την αγορά των *προσωπικών υπολογιστών* (*Personal Computer, PC*). Στην δεύτερη κατηγορία ανήκει μια πιο εξειδικευμένη πρόταση στην σύγχρονη αγορά *διακομιστών* (*servers*), οι επεξεργαστές UltraSPARC T1 και UltraSPARC T2 της εταιρίας Sun Microsystems [136].

4.4.1 Οικογένεια επεξεργαστών Intel Core 2

Η ονομασία Core™ 2 [141] αναφέρεται σε μια σειρά επεξεργαστών της εταιρείας Intel με αρχιτεκτονική συνόλου εντολών x86-64, που απευθύνονται σε κάθε είδους χρήστες. Προσφέρεται σε μονοπύρηνες, διπύρηνες και τετραπύρηνες εκδόσεις αυτής της οικογένειας. Οι συχνότητες ρολογιού είναι χαμηλότερες από τους μονοπύρηνους επεξεργαστές της εταιρείας, καθώς δόθηκε βάρος τόσο στην σχεδίαση πιο αποδοτικών δομικών στοιχείων όσο και στην μείωση της κατανάλωσης ισχύος του κάθε πυρήνα. Οι επεξεργαστές αυτοί παρουσιάστηκαν στην αγορά τον Ιούλιο του 2006 ενώ από τότε έχουν κυκλοφορήσει διάφορες εκδόσεις οι οποίες παρουσιάζουν βελτιώσεις ή προσφέρουν εξειδικευμένες δυνατότητες. Θα γίνει σύντομη αναφορά στα χαρακτηριστικά της αρχιτεκτονικής, λαμβάνοντας υπόψη ότι η ποικιλία των εκδόσεων που υπάρχει διαθέσιμη στο εμπόριο είναι εξαιρετικά μεγάλη και είναι δυνατόν να υπάρχουν αποκλίσεις από έκδοση σε έκδοση. Στην Εικόνα 4.9 παρουσιάζεται η φυσική διάταξη του επεξεργαστή Core™ 2 Duo ο οποίος ενσωματώνει δύο πυρήνες επεξεργαστών εντός του ολοκληρωμένου κυκλώματος.

Η διοχέτευση αυτής της αρχιτεκτονικής περιλαμβάνει 14 στάδια. Σε κάθε πυρήνα υπάρχουν τρεις αριθμητικές και λογικές μονάδες, δύο μονάδες κινητής υποδιαστολής και τέσσερις αποκωδικοποιητές. Οι εντολές μπορούν να αναδιαταχθούν για την επίτευξη καλύτερης απόδοσης (υπάρχει δηλαδή, δυνατότητα εκτέλεσης *εκτός σειράς* (*out-of-order*)). Ο κάθε πυρήνας υλοποιεί εξελιγμένες *τεχνικές πρόβλεψης διακλάδωσης* (*branch prediction*), καθώς επίσης και εκτέλεσης εικασίας (*speculative execution*) υποστηρίζοντας κατά συνέπεια *δυναμική εκτέλεση* (*dynamic execution*). Σε κάθε κύκλο μπορούν να ολοκληρωθούν μέχρι και τέσσερις εντολές. Το σύστημα κρυφής μνήμης περιλαμβάνει δυο επίπεδα, ενώ ο δίαυλος δεδομένων μεταξύ των δύο επιπέδων είναι πιο ευρύς (256 bit). Η κρυφή μνήμη δευτέρου επιπέδου είναι

κοινόχρηστη μεταξύ των δύο πυρήνων και μπορεί να έχει μέγεθος από 2 MB έως και 12 MB. Κάθε πυρήνας περιλαμβάνει μία κρυφή μνήμη εντολών και μία δεδομένων (32 KB η κάθε μια συνήθως) πρώτου επιπέδου. Η συχνότητα του ρολογιού ανάλογα με τη έκδοση κυμαίνεται μεταξύ 1.06 και 3.2 GHz. Η ολοκλήρωση κατασκευής στις πρώτες εκδόσεις πραγματοποιούταν στα 65nm, ενώ στις τελευταίες στα 45 nm.



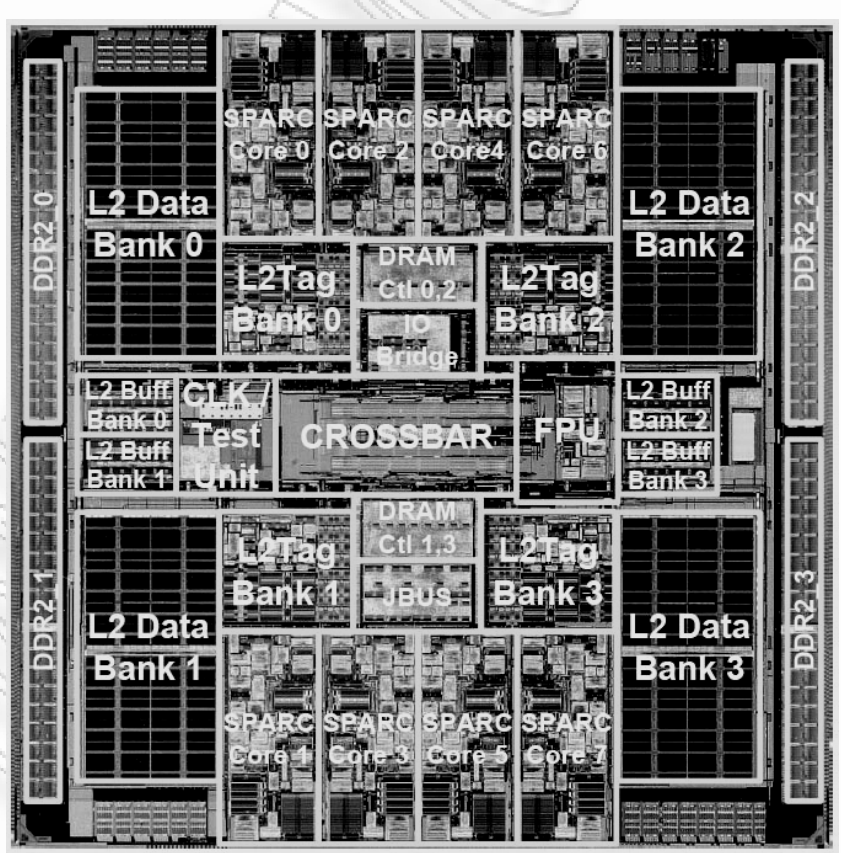
Εικόνα 4.9 Ο επεξεργαστής Intel Core 2 Quad

4.4.2 Επεξεργαστές UltraSPARC T1 και UltraSPARC T2 της Sun Microsystems

Η πρόταση της εταιρείας Sun Microsystems στον τομέα της παράλληλης επεξεργασίας είναι ο επεξεργαστής UltraSPARC T1 (κωδική ονομασία Niagara) [143], ο οποίος παρουσιάστηκε το 2005. Στόχος της αρχιτεκτονικής αυτού του επεξεργαστή ήταν η μέγιστη δυνατή απόδοση σε πολυνηματικές εφαρμογές (σε εξυπηρετητές του διαδικτύου, στον τομέα των βάσεων δεδομένων και άλλα). Ο επεξεργαστής αυτός παρέχει τη δυνατότητα εκτέλεσης 32 παράλληλων νημάτων (*strands*) (σε επίπεδο υλικού) με ιδιαίτερα υψηλές επιδόσεις και χαμηλές απαιτήσεις ισχύος (λειτουργεί στα 1.0 – 1.4 GHz με 72 W κατανάλωση ισχύος). Τα λειτουργικά συστήματα αντιμετωπίζουν τον πολυεπεξεργαστή ως 32 διαφορετικούς πυρήνες, κάθε ένας από

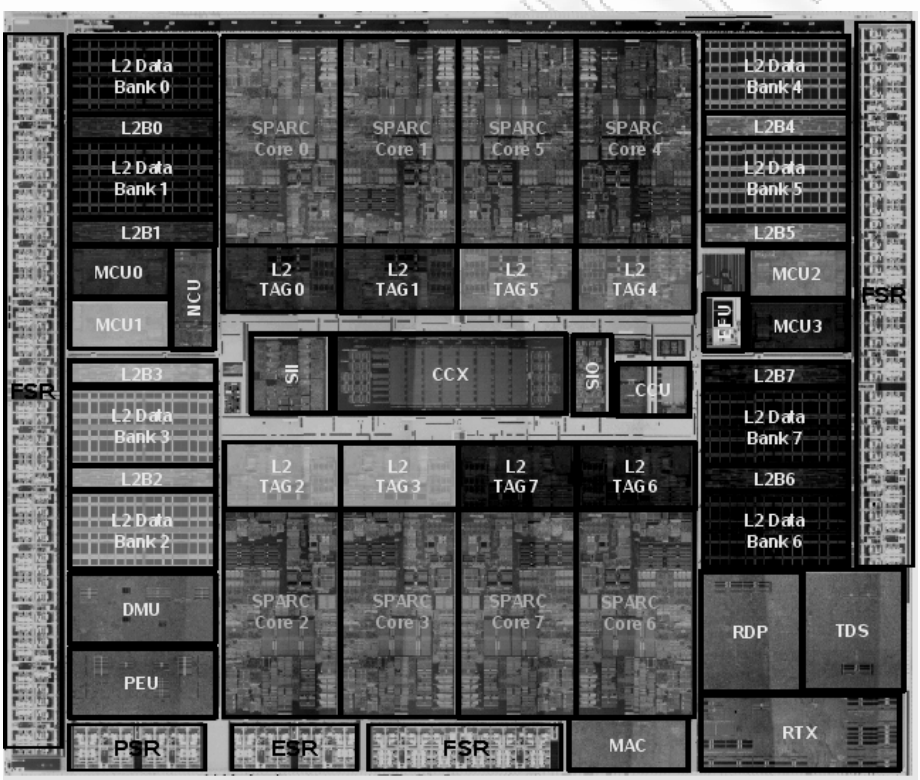
τους οποίους μπορεί να αναλάβει την εκτέλεση μιας διεργασίας. Η Εικόνα 4.10 παρουσιάζει τη φυσική διάταξη του συγκεκριμένου επεξεργαστή.

Ο επεξεργαστής Niagara, ουσιαστικά, αποτελείται από οκτώ πυρήνες με αρχιτεκτονική SPARC V9, καθένας εκ' των οποίων μπορεί να εκτελέσει μέχρι τέσσερα νήματα υλικού με τεχνολογία *λεπτής πολυνημάτωσης* (*fine-grained multithreading*). Κάθε πυρήνας περιλαμβάνει δυο κρυφές μνήμες πρώτου επιπέδου για εντολές (μεγέθους 16 KB) και δεδομένα (μεγέθους 8 KB), τις οποίες διαμοιράζονται τα τέσσερα νήματα. Η διοχέτευση αποτελείται από έξι στάδια: *προσκόμιση εντολής* (*instruction fetch*), *επιλογή νήματος* (*thread selection*), *αποκωδικοποίηση* (*decoding*), *εκτέλεση* (*execute*), *προσπέλαση μνήμης* (*memory access*) και *εγγραφή στο αρχείο καταχωρητών* (*write back*). Με την ομαδοποίηση των νημάτων ανά τετράδες (και της ανάθεσης κάθε μίας από τις τετράδες σε έναν πυρήνα), είναι δυνατόν να μειωθούν οι καθυστερήσεις κατά την πρόσβαση της μνήμης ή στη διοχέτευση του ενός νήματος, δεδομένου ότι πάντα θα υπάρχουν εντολές προς εκτέλεση ενός τουλάχιστον από τα τέσσερα νήματα. Η δρομολόγηση των νημάτων σε κάθε πυρήνα γίνεται με μηδενικό κόστος [143].



Εικόνα 4.10 Ο επεξεργαστής Sun UltraSPARC T1

Όλα τα νήματα μοιράζονται μια κρυφή μνήμη δευτέρου επιπέδου μεγέθους 3 MB, η οποία είναι οργανωμένη σε τέσσερις σειρές κάθε μια από τις οποίες συνδέεται με ένα κανάλι επικοινωνίας με τη μνήμη ή με κάποια περιφερειακή συσκευή. Η επικοινωνία των πυρήνων με την κρυφή μνήμη δευτέρου επιπέδου, τα περιφερειακά και τους άλλους κοινόχρηστους πόρους του επεξεργαστή πραγματοποιείται μέσω ενός δικτύου διασύνδεσης σταυρωτού μεταγωγέα, το οποίο προσφέρει μέχρι και 200 GB/s εύρους ζώνης [143]. Οι επεξεργαστές μοιράζονται μέσω του σταυρωτού μεταγωγέα μια μονάδα εκτέλεσης κινητής υποδιαστολής. Ο επεξεργαστής Niagara είναι ο πρώτος μιας σειράς επεξεργαστών της Sun Microsystems που στοχεύουν στην παράλληλη επεξεργασία νημάτων σε επίπεδο υλικού σε εμπορικά συστήματα. Οι σύγχρονες εφαρμογές του διαδικτύου αποτελούν την αγορά αυτών των επεξεργαστών, καθώς βελτιώνουν σημαντικά την απόδοση τέτοιου είδους εφαρμογών, έχοντας παράλληλα ιδιαίτερα χαμηλή κατανάλωση ισχύος.



Εικόνα 4.11 Ο επεξεργαστής Sun UltraSPARC T2

Ενάμιση χρόνο μετά (2007), η Sun Microsystems κυκλοφόρησε τον επεξεργαστή UltraSparc T2, με 8 πυρήνες SPARC και 8 νήματα ανά πυρήνα, υποστηρίζοντας κατά συνέπεια συνολικά 64 νήματα ανά ολοκληρωμένο κύκλωμα. Κάθε πυρήνας περιλαμβάνει δυο κρυφές μνήμες πρώτου επιπέδου για εντολές (μεγέθους 16 KB) και δεδομένα (μεγέθους 8 KB), οι

οποίες μοιράζονται στα οκτώ νήματα. Η διοχέτευση αποτελείται από οκτώ στάδια: *προσκόμιση εντολής* (*instruction fetch*), *προσπέλαση κρυφής μνήμης* (*cache*), *επιλογή νήματος* (*thread selection*), *αποκωδικοποίηση εντολής* (*instruction decode*), *εκτέλεση* (*execute*), *προσπέλασης μνήμης* (*memory access*), *παράκαμψη* (*bypass*) και *εγγραφή στο αρχείο καταχωρητών* (*write back*). Η Εικόνα 4.11 παρουσιάζει τη φυσική διάταξη του συγκεκριμένου επεξεργαστή.

Όλα τα νήματα μοιράζονται μια κρυφή μνήμη δευτέρου επιπέδου μεγέθους 4 MB, η οποία είναι οργανωμένη σε οκτώ σειρές. Η επικοινωνία των πυρήνων με την κρυφή μνήμη δευτέρου επιπέδου, τα περιφερειακά και τους άλλους κοινόχρηστους πόρους του επεξεργαστή πραγματοποιείται μέσω ενός δικτύου διασύνδεσης σταυρωτού μεταγωγέα,. Σε αντίθεση με τον επεξεργαστή UltraSPARC T1, οι επεξεργαστές ενσωματώνουν εντός του πυρήνα μια μονάδα εκτέλεσης κινητής υποδιαστολής. Επίσης, κάθε πυρήνας ενσωματώνει δύο αριθμητικές και λογικές μονάδες. Η κάθε αριθμητική και λογική μονάδα μοιράζεται ανά τέσσερα νήματα του πυρήνα.

Χαρακτηριστικά	**Intel Core™ 2**	**Sun UltraSPARC T1**	**Sun UltraSPARC T2**
Πυρήνες	2-4	8	8
Νήματα ανά πυρήνα	1	4	8
Κρυφή μνήμη δευτέρου επιπέδου	2-12 MB	3 MB	4 MB
Κρυφή μνήμη εντολών πρώτου επιπέδου	~ 32 KB	16 KB	16 KB
Κρυφή μνήμη δεδομένων πρώτου επιπέδου	~ 32 KB	8 KB	8 KB
Αριθμητικές και λογικές μονάδες	3	1	2
Μονάδες κινητής υποδιαστολής	2 + FP move	1 κοινή για όλους τους πυρήνες	8
Στάδια διοχέτευσης	14	6	8
Εκτέλεση εκτός σειράς	Ναι	Όχι	Όχι
Πρόβλεψη διακλάδωσης	Ναι	Όχι	Όχι
Εκτέλεση εικασίας	Ναι	Όχι	Όχι
Πολυνηματισμός	SMT	Λεπτή πολυνημάτωση	Λεπτή πολυνημάτωση
Ολοκλήρωση (nm)	45 nm, 65nm	90 nm	65 nm
Συχνότητα λειτουργίας (GHz)	1.06 – 3.2	1.0 – 1.4	1.4
Κατανάλωση ισχύος (Watt)	5.5 – 150	72	92

Πίνακας 4.2 Χαρακτηριστικά τριών συμμετρικών πολυεπεξεργαστών

Ο Πίνακας 4.2 συνοψίζει τα χαρακτηριστικά των τριών συμμετρικών πολυεπεξεργαστών που παρουσιάστηκαν σε αυτή την ενότητα. Ο επεξεργαστής της Intel Core™ 2 ενσωματώνει

περισσότερα χαρακτηριστικά απόδοσης εντός του πυρήνα με μικρότερο πλήθος επεξεργαστών. Η Sun Microsystems αντίθετα ακολουθεί μια πιο λιτή δομή για κάθε πυρήνα, ενσωματώνοντας όμως πολλούς πυρήνες με πολλαπά νήματα σε κάθε πυρήνα για να επιτύχει την προσδοκώμενη συνολική επιτάχυνση.

4.5 Δοκιμή συμμετρικών πολυεπεξεργαστών

Με τους πολλαπλούς πυρήνες στην αιχμή της τεχνολογίας των επεξεργαστών, η δοκιμή κατασκευής αντιμετωπίζει ολοένα περισσότερες νέες προκλήσεις όταν εφαρμόζεται στα ολοκληρωμένα κυκλώματα πολυεπεξεργαστών, είτε γενικότερα στα συστήματα πολυεπεξεργαστών. Η μεταφορά των τεχνικών δοκιμής από τους μονούς επεξεργαστές στις αρχιτεκτονικές των πολυεπεξεργαστών παρουσιάζει σημαντικά οφέλη καθώς επιτρέπει την εκμετάλλευση της υπάρχουσας τεχνογνωσίας. Ωστόσο, το εγχείρημα αυτό εμπεριέχει τον κίνδυνο να αυξηθεί εκθετικά η πολυπλοκότητα της δοκιμής με τον αριθμό των πυρήνων, γεγονός που μπορεί να οδηγήσει σε μεγάλη αύξηση τόσο του χρόνου ανάπτυξης όσο και του χρόνου εφαρμογής της δοκιμής στο άμεσο μέλλον. Δεδομένου ότι κάθε πυρήνας επεξεργαστή απαιτεί τα ίδια δεδομένα δοκιμής (πρόγραμμα και διανύσματα δοκιμής), είναι πιθανό ο χρόνος εφαρμογής της δοκιμής για ολόκληρο τον πολυεπεξεργαστή να είναι πολλαπλάσιος του αριθμού των πυρήνων. Η εφαρμογή και η επέκταση των υπαρχουσών τεχνικών δοκιμής για μονοπύρηνους επεξεργαστές, σε σύνθετες αρχιτεκτονικές πολυεπεξεργαστών, προϋποθέτει την επίλυση συγκεκριμένων προβλημάτων δρομολόγησης και εφαρμογής της δοκιμής.

4.5.1 Ανασκόπηση βιβλιογραφίας

Οι προσεγγίσεις της αυτοδοκιμής με λογισμικό έχουν αποτελέσει αντικείμενο μελέτης πολλών ερευνητικών ομάδων καθώς και ομάδων ανάπτυξης [82]-[97], όπως παρουσιάστηκε στο Κεφάλαιο 2. Οι προσεγγίσεις αυτές προτείνουν αποδοτικές μεθοδολογίες που επαληθεύονται σε πολλούς πυρήνες επεξεργαστών, από απλούς επεξεργαστές χαμηλής πολυπλοκότητας έως σύνθετους επεξεργαστές με τους προηγμένους μηχανισμούς απόδοσης.

Η δοκιμή των πολυεπεξεργαστών περιγράφηκε πρόσφατα στις ερευνητικές εργασίες [145] - [152]. Στις εργασίες [145] και [146] οι συγγραφείς παρουσίασαν τα χαρακτηριστικά γνωρίσματα δοκιμής του επεξεργαστή CELL ο οποίος ενσωματώνει έναν 64-bit επεξεργαστή PowerPC δύο νημάτων και οκτώ *συνεργατικών επεξεργαστικών στοιχείων* (*Synergistic Processor Elements, SPE*) τα οποία συνδέονται μέσω ενός *διαύλου επικοινωνίας* (*Element Interconnect Bus, EIB*). Στην εργασία παρουσιάζονται οι τεχνικές δοκιμής που εφαρμόζονται κατα την διάρκεια σχεδίασης του επεξεργαστή, με στόχο την μείωση των ακροδεκτών δοκιμής. Για την δοκιμή του επεξεργαστή τόσο για τις ενσωματωμένες μνήμες, όσο και για τη λογική αυτού, χρησιμοποιείται

ενσωματωμένη αυτοδοκιμή (*Built-In Self-Test, BIST*) ώστε να βελτιωθεί το συνολικό ποσοστό κάλυψης ελαττωμάτων και να επιτραπεί η δοκιμή κατά την πραγματική συχνότητα λειτουργίας.

Ο Makar και οι άλλοι ερευνητές [147] περιέγραψαν, μια αρχιτεκτονική δοκιμής σάρωσης που ονομάζεται Azul Scan (AZSCAN), για τη δοκιμή του πολυεπεξεργαστή Vega2 που ενσωματώνει 48 πυρήνες επεξεργαστών και επιπλέον εφεδρικούς πυρήνες. Ο κύριος στόχος της τεχνικής είναι η δημιουργία μιας αρχιτεκτονικής δοκιμής η οποία ανιχνεύει αποτελεσματικά τους ελαττωματικούς πυρήνες, δοκιμάζει αποδοτικά τις ενσωματωμένες μνήμες και τέλος, εφαρμόζει τα διανύσματα δοκιμής μέσω των αλυσίδων σάρωσης. Η μεθοδολογία επιτυγχάνει την μείωση των διανυσμάτων δοκιμής κατά 75%, διατηρώντας την ικανότητα διάγνωσης των ελαττωματικών πυρήνων.

Ο Parulkar και οι άλλοι ερευνητές [148] προτείνουν μια ιεραρχική αρχιτεκτονική *σχεδίασης για αυξημένη δοκιμαστικότητα* (*Design For Testability, DFT*), η οποία εκμεταλλεύεται τα υπάρχοντα χαρακτηριστικά γνωρίσματα δοκιμής και επιδιόρθωσης των πυρήνων, ώστε να επιτραπεί η δοκιμή των πολυεπεξεργαστών UltraSPARC της εταιρίας Sun Microsystems [136]. Η αρχιτεκτονική σχεδίασης για αυξημένη δοκιμαστικότητα επιτρέπει την δοκιμή κάθε επεξεργαστή ώστε να διατηρηθεί η δυνατότητα διάγνωσης των ελαττωματικών πυρήνων, ενώ παράλληλα επιτυγχάνει μείωση του συνολικού χρόνου εφαρμογής της δοκιμής και κατά συνέπεια του συνολικού κόστους της δοκιμής. Η προτεινόμενη μεθοδολογία εφαρμόζει *λειτουργικά διανύσματα δοκιμής* (*functional test patterns*), δεδομένου ότι είναι χρήσιμα για την *λειτουργική αποσφαλμάτωση* (*functional debug*) και το speed binning των πολυεπεξεργαστών. Κατά την εφαρμογή της λειτουργικής δοκιμής, αποθηκεύεται μόνο ένα αντίγραφο των διανυσμάτων δοκιμής στην μνήμη του εξωτερικού ελεγκτή, ώστε να μειωθούν οι απαιτήσεις σε μέγεθος μνήμης, τα οποία εφαρμόζονται σε κάθε πυρήνα επεξεργαστή.

Ο Tan και οι άλλοι [149] παρουσίασαν μια μεθοδολογία δοκιμής για τον μικροεπεξεργαστή UltraSPARC T1 της εταιρίας Sun Microsystems [136]. Οι ερευνητές εξέτασαν τις προκλήσεις της λειτουργικής δοκιμής των πολυεπεξεργαστικών και πολυνηματικών επεξεργαστών. Σύμφωνα με την μεθοδολογία αυτή, η επαλήθευση της συχνότητας λειτουργίας κάθε μεμονωμένου πυρήνα πραγματοποιείται μέσω λειτουργικών τεχνικών δοκιμής. Επιπλέον, παράγονται διανύσματα δοκιμής μέσω ενός εμπορικού αυτομάτου εργαλείου παραγωγής διανυσμάτων, τα οποία εφαρμόζονται με αλυσίδες σάρωσης έχοντας ως στόχο την δοκιμή των οκτώ επεξεργαστών. Οι κρυφές μνήμες πρώτου και δεύτερου επιπέδου, δοκιμάζονται μέσω του RAMTEST για την ανίχνευση των *στατικών ελαττωμάτων* (*static faults*) και της *ενσωματωμένης αυτοδοκιμής* (*Built-In Self-Test, BIST*) για την ανίχνευση των *δυναμικών ελαττωμάτων* (*dynamic faults*).

Ο Bayraktaroglou και οι άλλοι [150] περιέγραψαν μια μεθοδολογία αυτοδοκιμής με λογισμικό, όπου ο *κώδικας δοκιμής* (*test code*), τα *διανύσματα δοκιμής* (*test patterns*) και οι *αποκρίσεις* (*test responses*) αποθηκεύονται στην κρυφή μνήμη δευτέρου επιπέδου του

επεξεργαστή. Οι ερευνητές απέδειξαν ότι η αυτοδοκιμή με λογισμικό είναι δυνατόν να χρησιμοποιηθεί κατά τη διάρκεια της *δοκιμής κατασκευής* (*manufacturing testing*) του εμπορικού πολυεπεξεργαστή UltraSPARC T1 και ότι αυτή μπορεί να εφαρμοστεί από έναν *χαμηλού κόστους εξωτερικό ελεγκτή* (*low cost tester*). Τα προγράμματα και τα διανύσματα δοκιμής φορτώνονται στην κρυφή μνήμη δευτέρου επιπέδου μέσω του *πρωτοκόλλου πρόσβασης δοκιμής* (*test access protocol*). Στη συνέχεια, οι επεξεργαστές εκτελούν τα προγράμματα δοκιμής ακολουθιακά και οι αποκρίσεις αποθηκεύονται στην κρυφή μνήμη δευτέρου επιπέδου. Μετά την ολοκλήρωση της εκτέλεσης του προγράμματος, από τους επεξεργαστές, οι αποκρίσεις της δοκιμής συλλέγονται από την κοινόχρηστη κρυφή μνήμη και μεταφέρονται στον εξωτερικό ελεγκτή για την επαλήθευση της ορθής λειτουργίας του πολυεπεξεργαστή. Το κύριο μειονέκτημα της εν' λόγω προσέγγισης είναι ότι δεν εκμεταλλεύεται τα έμφυτα χαρακτηριστικά των πολυεπεξεργαστών. Αυτό συμβαίνει διότι οι πυρήνες εκτελούν ο ένας μετά τον άλλο (ακολουθιακά) τον κώδικα δοκιμής με αποτελέσματα να αυξάνεται ο συνολικός χρόνος εφαρμογής της δοκιμής και κατά συνέπεια το συνολικό κόστος της δοκιμής.

Ο Constantinides και οι άλλοι [151] προτείνουν μια μεθοδολογία για την ανίχνευση και διάγνωση των ελαττωμάτων που βασίζεται στο λογισμικό και εφαρμόζεται κατά την λειτουργία του επεξεργαστή. Βασίζεται στην επέκταση της *αρχιτεκτονικής του συνόλου εντολών* (*instruction set architecture*) του πολυεπεξεργαστή, με εντολές που παρέχουν δυνατότητα πρόσβασης και ρύθμισης της εσωτερικής του κατάστασης. Ειδικό υλικολογισμικό σταματάει περιοδικά την εκτέλεση του κανονικού προγράμματος στους επεξεργαστές και χρησιμοποιεί ειδικές εντολές για να εφαρμόσει τα διανύσματα δοκιμής στους επεξεργαστές και στις υπόλοιπες μονάδες του ολοκληρωμένου (DRAM controller, floating point unit, I/O bridge, CPU-Cache crossbar κτλ). Η προτεινόμενη μεθοδολογία παρέχει την δυνατότητα τροποποίησης του κώδικα δοκιμής στο πεδίο λειτουργίας του πολυεπεξεργαστή, ανάλογα με τις απαιτήσεις τόσο για την απόδοση του κανονικού προγράμματος του συστήματος όσο και της αξιοπιστίας του πολυεπεξεργαστή, χωρίς να απαιτείται καμία αλλαγή στο υλικό. Η προτεινόμενη προσέγγιση εφαρμόστηκε στον πολυεπεξεργαστή OpenSPARC T1 [137] επιτυγχάνοντας 99,22% ποσοστό κάλυψης ελαττωμάτων σε όλο τον πολυεπεξεργαστή. Η επιβάρυνση απόδοσης της μεθοδολογίας είναι 5,5%, ενώ απαιτεί 5,8% επιπλέον υλικό για την υλοποίηση των εντολών δοκιμής.

Ο Guzey και οι άλλοι [152] προτείνουν μια μεθοδολογία που βασίζεται *εξόρυξη δεδομένων* (*data mining*) για την βελτιστοποίηση των πεδίων δοκιμών. Η μεθοδολογία εξάγει *περιορισμούς εισόδου* (*input constraints*) για τα σήματα της μονάδας υπό δοκιμή, αναλύοντας το *ίχνος της εξομοίωσης* (*simulation trace*). Ο χρήστης έχει την δυνατότητα να επιλέξει ποιους από τους περιορισμούς θα χρησιμοποιήσει κατά τον επανακαθορισμό της εξομοίωσης. Όταν αυτοί οι περιορισμοί εισάγονται στην διαδικασία της εξομοίωσης, ενεργοποιούν και ανιχνεύουν τα ελαττώματα της μονάδας υπό δοκιμή. Η διαδικασία επαναλαμβάνεται είτε όταν επιτευχθεί η ανίχνευση των ελαττωμάτων του κυκλώματος, είτε όταν δεν είναι δυνατόν να παραχθούν

περιορισμοί για άλλα σήματα. Η μεθοδολογία εφαρμόστηκε στην μονάδα *επεξεργασίας συνεχούς ροής* (*stream processing unit*) και στην *μονάδα εκτέλεσης κινητής υποδιαστολής* (*floating point unit*) του πολυεπεξεργαστή OpenSPARC T1 [137].

Οι προηγούμενες ερευνητικές εργασίες οι οποίες εφάρμοσαν την προσέγγιση αυτοδοκιμής με λογισμικό για την δοκιμή των πολυεπεξεργαστών, όπως οι [148], [149], [150] και [151] επαναχρησιμοποιούν τα *λειτουργικά διανύσματα δοκιμής* (*functional test patterns*) τα οποία έχουν επαληθευτεί ως προς την αποτελεσματικότητά τους από τους επεξεργαστές μονού πυρήνα, ώστε να μειωθεί το κόστος ανάπτυξης του προγράμματος δοκιμής. Τα προγράμματα δοκιμής των επεξεργαστών εκτελούνται από όλους τους πυρήνες για να δοκιμαστεί η ορθή λειτουργία κάθε πυρήνα. Εφαρμόζεται παράλληλα ένα πλάνο επαναχρησιμοποίησης του προγράμματος ώστε να εξαλειφτεί η ανάγκη ύπαρξης πολλαπλών αντίγραφων κώδικα προγράμματος αυτοδοκιμής για κάθε έναν από τους πυρήνες. Επομένως δεν αυξάνονται οι απαιτήσεις χωρητικότητας της κοινόχρηστης κρυφής μνήμης με τον αριθμό των πυρήνων, για την αποθήκευση των προγραμμάτων δοκιμής. Ωστόσο, τα προγράμματα δοκιμής εκτελούνται διαδοχικά σε κάθε επεξεργαστή, επιτρέποντας κατά συνέπεια την δυνατότητα διάγνωσης στο επίπεδο των πυρήνων, απενεργοποιώντας όμως τον έμφυτο παραλληλισμό του πολυεπεξεργαστή. Έτσι, ο συνολικός χρόνος εφαρμογής της δοκιμής αυξάνεται συναρτήσει του αριθμού των πυρήνων.

Από τα παραπάνω συνεπάγεται ότι για να μειωθεί ο συνολικός χρόνος εφαρμογής της δοκιμής, οι επεξεργαστές θα πρέπει να εκτελέσουν τον κώδικα δοκιμής παράλληλα. Μια συστηματική πειραματική αξιολόγηση και ανάλυση της αρχιτεκτονικής των συμμετρικών πολυεπεξεργαστών αποκάλυψε ότι καθώς οι πολλαπλοί πυρήνες μοιράζονται τους ίδιους πόρους διασύνδεσης και μνήμης του συστήματος, ο *ανταγωνισμός του διαύλου επικοινωνίας* (*bus contention*) και οι ακυρώσεις (*invalidations*) των μηχανισμών συνοχής κρυφής μνήμης επιβραδύνουν και περιορίζουν τον παραλληλισμό κατά τη διάρκεια εφαρμογής της δοκιμής. Επιπλέον, οι μηχανισμοί διασύνδεσης συστήματος, όπως ο *κοινός δίαυλος* (*shared bus*) και ο *σταυρωτός μεταγωγέας* (*crossbar switch*), παρέχουν περιορισμένο εύρος ζώνης επικοινωνίας προς την κοινόχρηστη κρυφή μνήμη, προκαλώντας έναν ακόμη περιορισμό ως προς την αύξηση του παραλληλισμού εκτέλεσης των προγραμμάτων δοκιμής. Σύμφωνα με τα παραπάνω, οι ρουτίνες δοκιμής πρέπει να σχεδιαστούν κατάλληλα ώστε να μειωθεί ο αντίκτυπος της κοινής χρήσης των πόρων (δίκτυο επικοινωνίας και κοινόχρηστη κρυφή μνήμη) και να ελαχιστοποιηθούν τα διαστήματα κατά τα οποία παραμένουν ανενεργοί οι επεξεργαστές. Επιπλέον, η αποθήκευση του κώδικα και των διανυσμάτων δοκιμής στη κοινόχρηστη κρυφή μνήμη (κυρίως στη περίπτωση του σταυρωτού μεταγωγέα) πρέπει να πραγματοποιηθεί με τέτοιο τρόπο ώστε να μεγιστοποιηθεί η χρήση του διαθέσιμου εύρους ζώνης επικοινωνίας.

Στην παρούσα διδακτορική διατριβή προτείνεται μια αποδοτική γενική μεθοδολογία αυτοδοκιμής με λογισμικό [29], [30], [31] με στόχο την *κατανομή του κώδικα και των*

διανυσμάτων της δοκιμής (*test code and responses allocation*) στην κοινόχρηστη κρυφή μνήμη καθώς και την *δρομολόγηση των ρουτινών δοκιμής* (*test routines scheduling*) στους πυρήνες επεξεργαστών ώστε να επιτευχθεί σημαντική μείωση του χρόνου εκτέλεσης των προγραμμάτων δοκιμής. Η μεθοδολογία βασίζεται σε στατιστικά δεδομένα ως προς το *χρόνο εκτέλεσης* (*test application time*) και των αιτήσεων *προς την κοινόχρηστη κρυφή μνήμη* (*shared cache memory requests*) των ρουτινών αυτοδοκιμής. Οι πληροφορίες αυτές μπορούν να ληφθούν χωρίς χρονοβόρες προσομοιώσεις για τον πολυεπεξεργαστή.

Η προτεινόμενη μεθοδολογία [29], [31] αποδεικνύεται σε συμμετρικούς πολυεπεξεργαστές δύο, τεσσάρων και οκτώ επεξεργαστών που βασίζονται σε έναν επεξεργαστή RISC, τον OpenRISC 1200 [24] που έχει χρησιμοποιηθεί εκτενώς στην έρευνα της αυτοδοκιμής με λογισμικό στους μονοπύρηνους επεξεργαστές. Υλοποιήθηκαν συμμετρικοί πολυεπεξεργαστές με βάση τον OpenRISC 1200 ώστε να μελετηθεί η δυνατότητα εφαρμογής της προτεινόμενης μεθοδολογίας σε αρχιτεκτονικές συμμετρικών επεξεργαστών με διαφορετικό αριθμό επεξεργαστών και τύπο διασύνδεσης. Μελετώνται οι δύο πιο κοινές αρχιτεκτονικές διασύνδεσης: ο κοινός δίαυλος ο οποίος αξιολογείται σε συμμετρικούς πολυεπεξεργαστές δύο και τεσσάρων πυρήνων, και ο σταυρωτός μεταγωγέας ο οποίος αξιολογείται σε συμμετρικούς πολυεπεξεργαστές τεσσάρων και δύο διαφορετικών υλοποιήσεων οκτώ πυρήνων.

Παρουσιάζονται πειραματικά αποτελέσματα [29], [31] ως προς τον χρόνο εφαρμογής και το μέγεθος του κώδικα του προγράμματος δοκιμής καθώς και την αποτελεσματικότητα της προτεινόμενης μεθοδολογίας έναντι των εναλλακτικών λύσεων αυτοδοκιμής με λογισμικό. Παρέχονται επίσης αποτελέσματα προσομοίωσης ελαττωμάτων για ολόκληρη τη λογική των συμμετρικών πολυεπεξεργαστών (όχι μόνο της εσωτερικής λογικής των επεξεργαστών, αλλά και για της λογικής διαλειτουργικότητας μεταξύ των πυρήνων συμπεριλαμβανομένων των ελεγκτών κρυφής μνήμης και του δικτύου διασύνδεσης). Στη συνέχεια, η μεθοδολογία αξιολογείται με βάση ένα σύνολο *μετροπρογραμμάτων* (*benchmarks*) για επεξεργαστές του Πανεπιστημίου Stanford με διαφορετικά χαρακτηριστικά εκτέλεσης [25].

Στο δεύτερο μέρος των πειραματικών αποτελεσμάτων [30], παρουσιάζεται η εφαρμογή της μεθοδολογίας στον επεξεργαστή OpenSPARC T1 της Sun Microsystems. Τα αποτελέσματα προστέθηκαν με σκοπό την επαλήθευση της μεθοδολογίας σε μια πραγματική υλοποίηση σύγχρονου πολυεπεξεργαστή υψηλής απόδοσης. Για την εκτέλεση των πειραμάτων χρησιμοποιήσαμε ένα νήμα ανά πυρήνα, ώστε να συγκρίνουμε άμεσα τα αποτελέσματα με τους συμμετρικούς πολυεπεξεργαστές με τον πυρήνα OpenRISC 1200.

Η προτεινόμενη μεθοδολογία επιχειρεί την επίτευξη των ακόλουθων στόχων:

- Την επαναχρησιμοποίηση των προγραμμάτων αυτοδοκιμής από τους μονοπύρηνους επεξεργαστές, διατηρώντας υψηλά επίπεδα ως προς το ποσοστό κάλυψης ελαττωμάτων για τους μεμονωμένους πυρήνες.

- Την προσαρμογή των προγραμμάτων αυτοδοκιμής προκειμένου να ενεργοποιηθεί η παράλληλη εκτέλεση ενός ενιαίου αντιγράφου προγράμματος δοκιμής από όλους τους πυρήνες επεξεργαστών.

- Την ικανότητα προσδιορισμού των ελαττωματικών επεξεργαστών στο επίπεδο των πυρήνων παράγοντας διαφορετικές αποκρίσεις δοκιμής για κάθε πυρήνα.

Οι παραπάνω στόχοι επιχειρούν να διατηρήσουν το *κόστος της δοκιμής* (*test cost*) σε όσο το δυνατόν χαμηλότερα επίπεδα, εξασφαλίζοντας την ποιότητα και την αξία της δοκιμής.

4.6 Εφαρμογή της αυτοδοκιμής με λογισμικό στους συμμετρικούς πολυεπεξεργαστές

Κατά την εφαρμογή της αυτοδοκιμής με λογισμικό στους συμμετρικούς πολυεπεξεργαστές γίνονται οι ακόλουθες παραδοχές: Ο κώδικας αυτοδοκιμής αναπτύσσεται ώστε να μην επιτρέπεται κανένας κύκλος πρόσβασης προς την κύρια μνήμη, όπως ακριβώς υλοποιείται και στις ερευνητικές εργασίες [87], [150]. Πριν από την εφαρμογή της δοκιμής, οι ρουτίνες αυτοδοκιμής φορτώνονται στην κοινόχρηστη κρυφή μνήμη των επεξεργαστών μέσω του πρωτοκόλλου πρόσβασης κρυφής μνήμης. Υιοθετείται η λύση αυτή έναντι της χρησιμοποίησης των μεμονωμένων τοπικών κρυφών μνημών για την αποθήκευση του προγράμματος και των διανυσμάτων δοκιμής διότι:

- Οι τοπικές κρυφές μνήμες είναι πάντα μικρότερες από τις κοινόχρηστες, με αποτέλεσμα να μην είναι πάντοτε εφικτή η αποθήκευση του προγράμματος και των διανυσμάτων της αυτοδοκιμής.

- Η εκτέλεση από τις τοπικές κρυφές μνήμες προϋποθέτει μεγάλο αριθμό φορτώσεων του προγράμματος, ίσο με τον αριθμό των επεξεργαστών (συνεπώς υπερβολικού χρόνου φόρτωσης από έναν αργό εξωτερικό ελεγκτή), ενώ η εκτέλεση από την κοινόχρηστη κρυφή μνήμη απαιτεί μόλις μία.

- Η εκτέλεση των προγραμμάτων αυτοδοκιμής από τη κοινόχρηστη κρυφή μνήμη επιτρέπει τη δοκιμή της λογικής του *διαιτητή διαύλου επικοινωνίας* (*bus arbiter*).

- Η ίδια προσέγγιση, υιοθετείται και από την ερευνητική εργασία [150] για τη λειτουργική δοκιμή του επεξεργαστή UltraSPARC T1 της Sun Microsystems.

Στην παρούσα διδακτορική διατριβή, εξετάζεται η ικανότητα παράλληλης εκτέλεσης του προγράμματος δοκιμής σε συμμετρικούς πολυεπεξεργαστές με στόχο τη βελτιστοποίηση του

μεγέθους προγράμματος δοκιμής διατηρώντας την ικανότητα προσδιορισμού των ελαττωματικών πυρήνων. Στην ιδανική περίπτωση, το συνολικό μέγεθος μνήμης και ο χρόνος εκτέλεσης δοκιμής θα ήταν ίσος με αυτόν του αντίστοιχου μονοπύρηνου επεξεργαστή. Η περίπτωση όμως αυτή δεν είναι εφικτή, όπως παρατηρείται σε επόμενη ενότητα, λόγω των περιορισμών παραλληλισμού που επιβάλλονται από το υποσύστημα κρυφής μνήμης, Τα πειραματικά αποτελέσματα που παρουσιάζονται στην ενότητα 4.8, αποδεικνύουν ότι η προτεινόμενη μέθοδος οδηγεί σε τιμές χρόνου εκτέλεσης πολύ κοντά στην περίπτωση του μονοπύρηνου επεξεργαστή.

Προτού περιγραφεί η προτεινόμενη προσέγγιση, εξετάζονται οι διαφορετικές εναλλακτικές λύσεις για μεταφορά των υπαρχόντων προγραμμάτων αυτοδοκιμής από τους μονοπύρηνους επεξεργαστές στους συμμετρικούς πολυεπεξεργαστές και συγκρίνονται ως προς τρεις διαφορετικές παραμέτρους:

- Το συνολικό μέγεθος μνήμης συμπεριλαμβανομένου του προγράμματος δοκιμής και των διανυσμάτων δοκιμής.

- Το συνολικό χρόνο εφαρμογής της δοκιμής συμπεριλαμβανομένου του χρόνου φόρτωσης του κώδικα δοκιμής, του χρόνου εκτέλεσης της δοκιμής και του χρόνου συλλογής των αποκρίσεων.

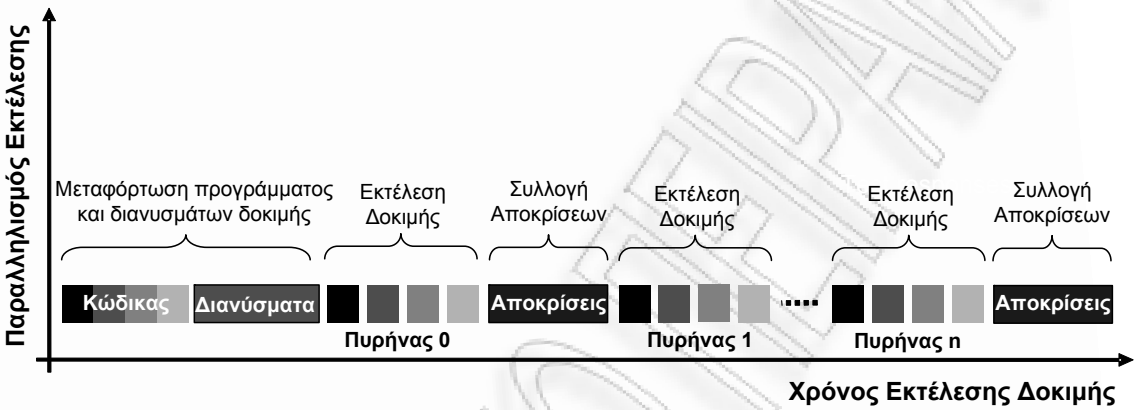
- Την ικανότητα διάγνωσης στο επίπεδο των πυρήνων.

Στη συνέχεια περιγράφονται οι τρεις διαφορετικές εναλλακτικές προσεγγίσεις της αυτοδοκιμής με λογισμικό για τους συμμετρικούς πολυεπεξεργαστές.

4.6.1 Ενιαίο Αντίγραφο - Διαδοχική Εκτέλεση (ΕΑ-ΔΕ)

Σύμφωνα με την προσέγγιση αυτή, ένα ενιαίο αντίγραφο του προγράμματος αυτοδοκιμής αποθηκεύεται στην κοινόχρηστη κρυφή μνήμη και ένας από τους πυρήνες επεξεργαστών το εκτελεί κάθε φορά. Η εναλλακτική αυτή προσέγγιση αποτελεί την απλούστερη λύση και έχει ήδη υιοθετηθεί από προηγούμενες ερευνητικές προσεγγίσεις [148], [149]. Ο χρόνος φόρτωσης του προγράμματος δοκιμής ισούται με την περίπτωση του μονοπύρηνου επεξεργαστή (ενιαία φόρτωση). Κατά τη διάρκεια της αυτοδοκιμής, ένας μόνο επεξεργαστής πραγματοποιεί αιτήσεις εξυπηρέτησης προς το μέσο επικοινωνίας. Οι αποκρίσεις δοκιμής κάθε πυρήνα αποθηκεύονται στην κοινόχρηστη κρυφή μνήμη και συλλέγονται από τον εξωτερικό ελεγκτή πριν ξεκινήσει την εκτέλεση του προγράμματος ο επόμενος πυρήνας. Η προσέγγιση αυτή δεν παρουσιάζει ανταγωνισμό του διαύλου επικοινωνίας καθώς επίσης και ακυρώσεις δεδομένων της κρυφής μνήμης, δεδομένου ότι δεν υπάρχει παράλληλη εκτέλεση του προγράμματος. Το κύριο μειονέκτημα της προσέγγισης είναι ότι δεν εκμεταλλεύεται τα έμφυτα χαρακτηριστικά

παραλληλίας των συμμετρικών πολυεπεξεργαστών. Ο χρόνος των διαφορετικών σταδίων αυτής της προσέγγισης απεικονίζεται στην Εικόνα 4.12.



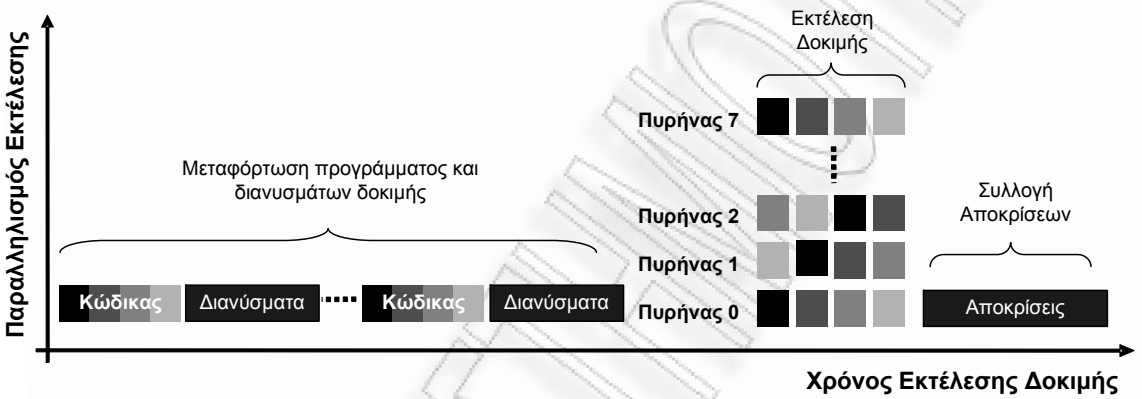
Εικόνα 4.12 Στάδια προσέγγισης ενιαίου αντιγράφου – διαδοχικής εκτέλεσης.

4.6.2 Πολλαπλά Αντίγραφα - Παράλληλη Εκτέλεση (ΠΑ-ΠΕ)

Η εναλλακτική προσέγγιση που εξαλείφει τα μειονεκτήματα της προηγούμενης προσέγγισης και επιτρέπει την παράλληλη εκτέλεση του προγράμματος δοκιμής, υποθέτει ότι πολλαπλά αντίγραφα του προγράμματος δοκιμής αποθηκεύονται στην κοινόχρηστη κρυφή μνήμη και κάθε επεξεργαστής εκτελεί το δικό του αντίγραφο παράλληλα με τους υπόλοιπους. Στην αρχιτεκτονική του κοινού διαύλου, η κοινόχρηστη κρυφή μνήμη διαιρείται σε *n* περιοχές, όπου *n* είναι ο αριθμός των επεξεργαστών. Κάθε περιοχή αντιστοιχεί σε έναν επεξεργαστή και για κάθε περιοχή αποθηκεύεται ένα αντίγραφο του προγράμματος δοκιμής και οι αντίστοιχες αποκρίσεις του επεξεργαστή αυτού. Η εφαρμογή της αυτοδοκιμής μπορεί να επιβραδυνθεί σημαντικά λόγω του ανταγωνισμού του διαύλου (E_{con}, Πίνακας 4.3) καθώς όλοι οι επεξεργαστές χρησιμοποιούν τον κοινό δίαυλο επικοινωνίας για να αποκτήσουν πρόσβαση στην κοινόχρηστη κρυφή μνήμη.

Στην περίπτωση του σταυρωτού μεταγωγέα, *n* αντίγραφα του προγράμματος δοκιμής αποθηκεύονται στις *m* διαφορετικές σειρές της κοινόχρηστης κρυφής μνήμης και κάθε σειρά της κοινόχρηστης κρυφής μνήμης διαιρείται σε *n/m* περιοχές. Η επιβράδυνση λόγω του ανταγωνισμού του σταυρωτού μεταγωγέα (E_{con}, Πίνακας 4.3) υπολογίζεται χωριστά για κάθε σειρά της κοινόχρηστης κρυφής μνήμης και είναι μικρότερη από την επιβράδυνση λόγω ανταγωνισμού στην περίπτωση του κοινού διαύλου. Το φαινόμενο αυτό παρατηρείται λόγω του ότι ο σταυρωτός μεταγωγέας μπορεί να υποστηρίξει ταυτόχρονες προσβάσεις προς τις σειρές της κοινόχρηστης κρυφής μνήμης από διαφορετικούς επεξεργαστές.

Ο χρόνος φόρτωσης του προγράμματος δοκιμής και ο χρόνος συλλογής των αποκρίσεων είναι *n* φορές μεγαλύτεροι από τους αντίστοιχους χρόνους του μονοπύρηνου επεξεργαστή. Επιπλέον, είναι εφικτός ο προσδιορισμός των ελαττωματικών επεξεργαστών, δεδομένου ότι κάθε επεξεργαστής αποθηκεύει τις αποκρίσεις δοκιμής σε διαφορετικές περιοχές της κοινόχρηστης κρυφής μνήμης του συστήματος. Το κύριο μειονέκτημα της προσέγγισης αυτής είναι οι υπερβολικές απαιτήσεις αποθήκευσης της κρυφής μνήμης δεδομένου των πολλαπλών προγραμμάτων δοκιμής και η μεγάλη χρονική αύξηση του χρόνου φόρτωσης ο οποίος ολοένα αυξάνεται με την ενσωμάτωση όλο και περισσότερων επεξεργαστών στο σύστημα. Ο παραλληλισμός αυτής της προσέγγισης για κάθε στάδιο της δοκιμής απεικονίζεται στην Εικόνα 4.13.



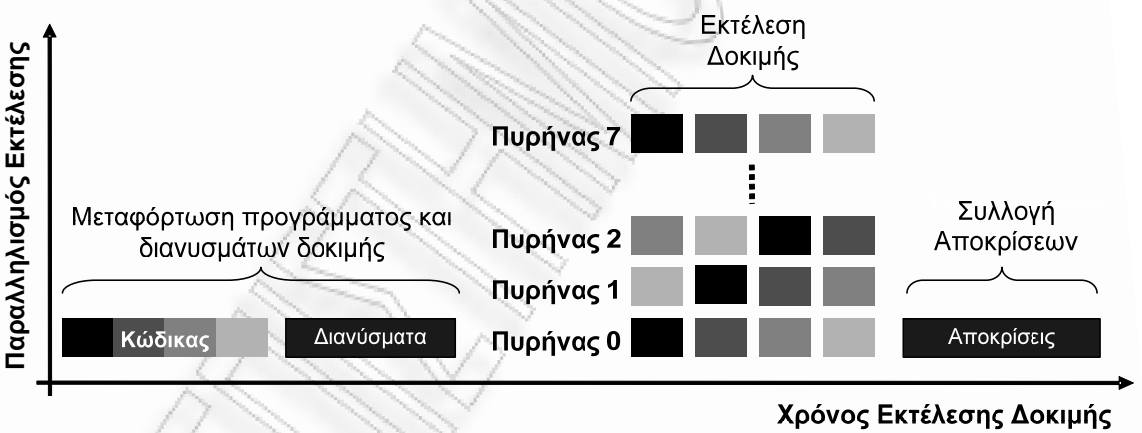
Εικόνα 4.13 Στάδια προσέγγισης πολλαπλών αντιγράφων– παράλληλης εκτέλεσης.

4.6.3 Ενιαίο Αντίγραφο - Παράλληλη Εκτέλεση (ΕΑ-ΠΕ)

Η τρίτη εναλλακτική προσέγγιση που συνδυάζει τα πλεονεκτήματα των δύο προηγούμενων προσεγγίσεων, υποθέτει ότι ένα ενιαίο αντίγραφο του προγράμματος αυτοδοκιμής αποθηκεύεται στη κοινόχρηστη κρυφή μνήμη του συστήματος και εκτελείται παράλληλα από όλους τους επεξεργαστές. Για να επιτευχθεί η παράλληλη εκτέλεση από έναν ενιαίο κώδικα αυτοδοκιμής, απαιτείται ο μετασχηματισμός των ρουτινών δοκιμής ώστε να επιτραπεί η αποθήκευση των αποκρίσεων δοκιμής από τους πυρήνες στις ίδιες μεταβλητές (οι μεταβλητές που χρησιμοποιούνται για να αποθηκεύσουν τις αποκρίσεις δοκιμής είναι κοινές δεδομένου ότι όλοι οι επεξεργαστές εκτελούν το ίδιο αντίγραφο προγράμματος). Αυτό επιτυγχάνεται με την εκτέλεση ρουτινών *σύμπτυξης* (*compaction*) των αποκρίσεων της δοκιμής όλων των επεξεργαστών, έναντι της αποθήκευσης των αποκρίσεων σε διαφορετικές θέσεις μνήμης. Συνεπώς, οι αποκρίσεις κάθε επεξεργαστή συμπιέζονται με τις αποκρίσεις των υπολοίπων

επεξεργαστών δημιουργώντας μία μοναδική υπογραφή για κάθε *n* αντίστοιχα αντίγραφα αποκρίσεων των προηγούμενων προσεγγίσεων.

Ο χρόνος φόρτωσης ισούται με τον χρόνο φόρτωσης του μονοπύρηνου επεξεργαστή, προσθέτοντας έναν επιπλέον χρόνο για να φορτωθεί ο κώδικας που εφαρμόζει τη σύμπτυξη των αποκρίσεων. Στην προσέγγιση αυτή, οι επεξεργαστές ανταγωνίζονται για τους πόρους διασύνδεσης του συστήματος ώστε να αποκτήσουν πρόσβαση στη κοινόχρηστη κρυφή μνήμη επιβραδύνοντας σημαντικά την εκτέλεση του προγράμματος δοκιμής. Επιπλέον, οι ακυρώσεις συνοχής κρυφής μνήμης επιβραδύνουν σημαντικά την εκτέλεση του προγράμματος δοκιμής, δεδομένου ότι οι επεξεργαστές χρησιμοποιούν κοινές διευθύνσεις μνήμης για να επιτύχουν συμπίεση των αποκρίσεων δοκιμής. Ο χρόνος συλλογής των αποκρίσεων δοκιμής ισούται με αυτόν του μονοπύρηνου επεξεργαστή. Κύριο μειονέκτημα αυτής της προσέγγισής είναι ότι δεν υποστηρίζει τη διάγνωση των ελαττωματικών πυρήνων μειώνοντας έτσι την συνολική εσοδεία του προϊόντος.



Εικόνα 4.14 Στάδια προσέγγισης ενιαίου αντίγραφου– παράλληλης εκτέλεσης.

Ο Πίνακας 4.3 συνοψίζει τα χαρακτηριστικά των τριών εναλλακτικών προσεγγίσεων αυτοδοκιμής με λογισμικό για τους συμμετρικούς πολυεπεξεργαστές. Με απώτερο στόχο την παρουσίαση μιας καλύτερης ποσοτικής ανάλυσης και σύγκρισης αυτών των εναλλακτικών λύσεων αυτοδοκιμής με λογισμικό, πραγματοποιήθηκε ένα σύνολο πειραμάτων για κάθε προσέγγιση σε μια υλοποίηση πολυεπεξεργαστή αποτελούμενο από τέσσερις πυρήνες. Η υλοποίηση του συμμετρικού πολυεπεξεργαστή περιέχει τέσσερις επεξεργαστές OpenRISC 1200 [24] οι οποίοι συνδέονται με την κοινόχρηστη κρυφή μνήμη δευτέρου επιπέδου μέσω ενός κοινού δίαυλου ανοικτού κώδικα Wishbone [24]. Περαιτέρω ανάλυση του συμμετρικού πολυεπεξεργαστή παρατίθεται στην Ενότητα 4.8.

Εναλλακτικές Προσεγγίσεις	Χρόνος Φόρτωσης Προγράμματος	Χρόνος Εκτέλεσης Προγράμματος	Χρόνος Συλλογής Αποκρίσεων	Απαιτήσεις Μνήμης	Δυνατότητα Διάγνωσης
Μονοπύρηνος	D	E	U	S	n.a.
ΕΑ-ΔΕ	D	$n \times E$	$n \times U$	S	Yes
ΠΑ-ΠΕ	$n \times D$	$E + E_{con}$	$n \times U$	$n \times S$	Yes
ΕΑ-ΠΕ	$D + D_{acc}$	$E + E_{acc} + E_{con} + E_{inv}$	U	$S + S_{acc}$	No

Πίνακας 4.3 Σύγκριση εναλλακτικών προσεγγίσεων (n=αριθμός ενσωματωμένων πυρήνων, E_{acc}=επιπρόσθετος χρόνος για την συσσώρευση των αποκρίσεων, E_{con}=επιπρόσθετος χρόνος λόγω ανταγωνισμού του διαύλου, E_{inv}=επιπρόσθετος χρόνος λόγω ακυρώσεων της κρυφής μνήμης, D_{acc}=χρόνος φόρτωσης του κώδικα συσσώρευσης, S_{acc}=μέγεθος κώδικα συσσώρευσης.

Ο Πίνακας 4.4 παρουσιάζει τα συγκριτικά αποτελέσματα των πειραμάτων για τις τρεις εναλλακτικές προσεγγίσεις αυτοδοκιμής με λογισμικό, ως προς το μέγεθος του κώδικα, των αποκρίσεων και του χρόνου εκτέλεσης του προγράμματος δοκιμής, με αυτά του μονοπύρηνου επεξεργαστή OpenRISC 1200. Αναφορικά με το μέγεθος του κώδικα δοκιμής, οι προσεγγίσεις ΕΑ-ΔΕ και ΕΑ-ΠΕ παρουσιάζουν συγκρίσιμα αποτελέσματα με την περίπτωση του μονοπύρηνου επεξεργαστή. Η εναλλακτική ΕΑ-ΠΕ έχει ελαφρώς μεγαλύτερο μέγεθος προγράμματος λόγω του πρόσθετου κώδικα για τη σύμπτυξη των αποκρίσεων της δοκιμής. Το μέγεθος του κώδικα δοκιμής της προσέγγισης ΠΑ-ΠΕ είναι τέσσερις φορές μεγαλύτερο από το αντίστοιχο του μονοπύρηνου επεξεργαστή. Η απαίτηση για υψηλή χωρητικότητα για την αποθήκευση του κώδικα δοκιμής κλιμακώνεται με τον αριθμό πυρήνων και αποτελεί το σημαντικότερο μειονέκτημα της προσέγγισης ΠΑ-ΠΕ. Ο τετραπλάσιος αριθμός αποκρίσεων δοκιμής είναι το τίμημα που πληρώνουν οι προσεγγίσεις ΕΑ-ΔΕ και ΠΑ-ΠΕ για να διατηρήσουν την ικανότητα διάγνωσης στο επίπεδο των πυρήνων.

Ο χρόνος εφαρμογής της δοκιμής των παράλληλων προσεγγίσεων θα αναμενόταν να συγκλίνει προς τον χρόνο εκτέλεσης του μονοπύρηνου επεξεργαστή. Η προσέγγιση ΠΑ-ΠΕ επιτυγχάνει το συντομότερο χρόνο εκτέλεσης δοκιμής ο οποίος, ωστόσο, τείνει να είναι διπλάσιος από τον αντίστοιχο χρόνο του μονοπύρηνου επεξεργαστή. Αυτό αποδεικνύει ότι ο ανταγωνισμός του δίαυλου επικοινωνίας μπορεί να επιβραδύνει σημαντικά την εκτέλεση της δοκιμής λόγω των ταυτόχρονων προσβάσεων στους κοινούς πόρους της κοινόχρηστης κρυφής μνήμης. Στην περίπτωση της προσέγγισης ΕΑ-ΠΕ, οι αλλεπάλληλες ακυρώσεις συνοχής της κρυφής μνήμης επιβραδύνουν περαιτέρω την εκτέλεση της δοκιμής, η οποία διαρκεί περίπου το ίδιο με την ακολουθιακή προσέγγιση. Το φαινόμενο αυτό παρατηρείται κυρίως λόγω των ταυτόχρονων προσβάσεων των κοινών μεταβλητών που χρησιμοποιούνται από τους διαφορετικούς πυρήνες για να συμπιέσουν τις αποκρίσεις της δοκιμής.

Εναλλακτικές Προσεγγίσεις	Μέγεθος Κώδικα (Λέξεις)	Αποκρίσεις Δοκιμής (Λέξεις)	Χρόνος Εκτέλεσης (Κύκλοι Ρολογιού)
Μονοπύρηνος	3271	1791	87533
ΕΑ-ΔΕ	3271	7164	350132
ΠΑ-ΠΕ	13084	7164	198904
ΕΑ-ΠΕ	3543	1791	342742

Πίνακας 4.4 Σύγκριση των εναλλακτικών προσεγγίσεων ως προς το μέγεθος του κώδικα, των αριθμό των αποκρίσεων και το συνολικό χρόνο εκτέλεσης της αυτοδοκιμής.

4.7 Προτεινόμενη μεθοδολογία βελτιστοποίησης απόδοσης αυτοδοκιμής σε συμμετρικούς πολυεπεξεργαστές

Από την παραπάνω ανάλυση συμπεραίνεται ότι τα σημαντικά ζητήματα για την αποτελεσματική εφαρμογή μιας προσέγγισης αυτοδοκιμής με λογισμικό σε μια αρχιτεκτονική συμμετρικών πολυεπεξεργαστών είναι:

- Η μείωση των απαιτήσεων αποθήκευσης στην κοινόχρηστη κρυφή μνήμη με τη χρήση ενός ενιαίου αντιγράφου προγράμματος δοκιμής.
- Η μείωση του χρόνου εκτέλεσης αυξάνοντας τον παραλληλισμό κατά τη διάρκεια της εκτέλεσης του προγράμματος δοκιμής.
- Η ελαχιστοποίηση του ανταγωνισμού του μέσου επικοινωνίας και των ακυρώσεων δεδομένων λόγω συνοχής της κρυφής μνήμης, τα οποία και επιβραδύνουν την εκτέλεση του κώδικα δοκιμής.
- Η παραγωγή αποκρίσεων σε διαφορετικές θέσεις της κοινόχρηστης κρυφής μνήμης για κάθε επεξεργαστή, με σκοπό να επιτραπεί η διάγνωση στο επίπεδο των πυρήνων.

Η προτεινόμενη μεθοδολογία [29], [30], [31] στοχεύει στην επίλυση των ανωτέρω ζητημάτων. Αποτελείται από τις τρεις φάσεις οι οποίες περιγράφονται στις ακόλουθες ενότητες.

4.7.1 Φάση Α: Ισοκατανομή του κώδικα αυτοδοκιμής στην κοινόχρηστη κρυφή μνήμη

Η πρώτη φάση της μεθοδολογίας εφαρμόζεται μόνο στην περίπτωση των συμμετρικών πολυεπεξεργαστών που χρησιμοποιούν την αρχιτεκτονική του σταυρωτού μεταγωγέα και στοχεύει στην μεγιστοποίηση του διαθέσιμου εύρους ζώνης επικοινωνίας μεταξύ των πυρήνων και της κοινόχρηστης κρυφής μνήμης κατά τη διάρκεια της εκτέλεσης του προγράμματος

δοκιμής. Η βασική ιδέα είναι η ισοκατανομή των διαφορετικών τμημάτων του ενιαίου αντιγράφου του προγράμματος στις διαφορετικές σειρές της κοινόχρηστης κρυφής μνήμης, ούτως ώστε οι πολλαπλοί πυρήνες να μπορέσουν να εκμεταλλευτούν τις πιθανές ελεύθερες συνδέσεις του σταυρωτού μεταγωγέα για να εκτελεστούν παράλληλα οι ρουτίνες του προγράμματος δοκιμής. Το βήμα αυτό δεν είναι δυνατό να εφαρμοστεί στην περίπτωση υλοποίησης των συμμετρικών πολυεπεξεργαστών με την χρήση κοινού διαύλου, δεδομένου ότι δεν υποστηρίζονται οι ταυτόχρονες συναλλαγές με την κοινόχρηστη κρυφή μνήμη.

Έστω ότι ένα πρόγραμμα δοκιμής αποτελείται από διαφορετικές ρουτίνες δοκιμής. Εάν αυτό δεν ισχύει τότε το πρόγραμμα δοκιμής πρέπει να χωριστεί σε πολλαπλές ρουτίνες. Η Φάση Α βασίζεται σε στατιστικά δεδομένα εκτέλεσης του προγράμματος δοκιμής, τα οποία συλλέγονται από την προσομοίωση του προγράμματος δοκιμής σε ένα μονοπύρηνο σύστημα με παρόμοια οργάνωση μνήμης. Τα στατιστικά αυτά αφορούν το χρόνο εκτέλεσης της δοκιμής καθώς και τον αριθμό των αιτήσεων προς την κοινόχρηστη κρυφή μνήμη κάθε ρουτίνας αυτοδοκιμής. Η φάση αυτή κατανέμει τις ρουτίνες δοκιμής στις σειρές της κοινόχρηστης κρυφής μνήμης, με απώτερο στόχο να διανείμει εξίσου το συνολικό χρόνο εκτέλεσης και το συνολικό αριθμό των αιτήσεων κρυφής μνήμης σε όλες τις σειρές της κοινόχρηστης κρυφής μνήμης. Μεγιστοποιώντας με αυτό τον τρόπο την αξιοποίηση του διαθέσιμου εύρους ζώνης επικοινωνίας, αυξάνεται ο παραλληλισμός της εκτέλεσης του προγράμματος δοκιμής και ταυτόχρονα ελαχιστοποιείται ο ανταγωνισμός σε κάθε σειρά της κοινόχρηστης κρυφής μνήμης.

Θεωρήστε ένα συμμετρικό πολυεπεξεργαστή που χρησιμοποιεί σταυρωτό μεταγωγέα με n πυρήνες επεξεργαστών και m σειρές κοινόχρηστης κρυφής μνήμης. Ο κώδικας αυτοδοκιμής από τον αντίστοιχο μονοπύρηνο επεξεργαστή αποτελείται από v ρουτίνες δοκιμής $R_1, R_2, \ldots, R_v$, ενώ κάθε ρουτίνα R_i όταν εκτελείται σε ένα μονοπύρηνο σύστημα ολοκληρώνεται σε t_i κύκλους ρολογιού και προκαλεί r_i αιτήσεις προς την κοινόχρηστη κρυφή μνήμη.

Ο στόχος της Φάσης Α είναι η ισοκατανομή του συνολικού χρόνου $t_{total}=\sum t_i$ και των αιτήσεων στην κοινόχρηστη κρυφή μνήμη $r_{total}=\sum r_i$ όλων των ρουτινών δοκιμής $R_1,R_2,\ldots,R_v$, μεταξύ των m σειρών της κοινόχρηστης κρυφής μνήμης. Προκειμένου να επιτευχθεί η ισοκατανομή του κώδικα δοκιμής, πραγματοποιείται εξαντλητική αναζήτηση όλων των πιθανών συνδυασμών ρουτινών αυτοδοκιμής στις σειρές της κοινόχρηστης κρυφής μνήμης και υπολογίζεται για κάθε σειρά k, $k=1,2\ldots,m$ ο συνολικός χρόνος εκτέλεσης δοκιμής $t_{bank\text{-}k}$ και οι συνολικές αιτήσεις $r_{bank\text{-}k}$ των ρουτινών δοκιμής. Στη συνέχεια, υπολογίζεται η απόκλιση των τιμών $\mathrm{dev}(t_{bank\text{-}k})$ και $\mathrm{dev}(r_{bank\text{-}k})$ σύμφωνα με τις μέσες τιμές του χρόνου εκτέλεσης δοκιμής $t_{mean}=t_{total}/m$ και των αιτήσεων στη κοινόχρηστη κρυφή μνήμη $r_{mean} = r_{total}/m$. Τέλος, επιλέγεται η κατανομή με την ελάχιστη συνολική απόκλιση τόσο για το χρόνο εκτέλεσης, όσο και για τις αιτήσεις προς την κοινόχρηστη κρυφή μνήμη.

4.7.2 Φάση Β: Μείωση των ακυρώσεων συνοχής κρυφής μνήμης

Σύμφωνα με το πρωτόκολλο συνοχής κρυφής μνήμης που αναπτύχθηκε στην Ενότητα 4.3.2, κάθε φορά που μια λέξη εγγράφεται από έναν επεξεργαστή στην κοινόχρηστη κρυφή μνήμη, όλοι οι άλλοι επεξεργαστές ακυρώνουν τα τοπικά αντίγραφα της λέξης αυτής. Επομένως, η εκτέλεση των προγραμμάτων αυτοδοκιμής μπορεί να επιβραδυνθεί σημαντικά, λόγω του μεγάλου αριθμού ακυρώσεων κρυφής μνήμης οδηγώντας στην επαναπροσκόμιση δεδομένων στην τοπική κρυφή μνήμη από την κοινόχρηστη, αυξάνοντας το συνολικό χρόνο εφαρμογής της δοκιμής. Οι στόχοι της φάσης Β είναι οι εξής:

- Η επίτρεψη της διάγνωσης στο επίπεδο των πυρήνων.
- Η μείωση των ακυρώσεων συνοχής της κρυφής μνήμης.

Από τις εναλλακτικές προσεγγίσεις που περιγράφηκαν στην Ενότητα 4.6, οι προσεγγίσεις ΕΑ-ΑΕ και ΠΑ-ΠΕ υποστηρίζουν την διάγνωση στο επίπεδο των πυρήνων. Ωστόσο, η πρώτη δεν εκμεταλλεύεται τον έμφυτο παραλληλισμό των πολυεπεξεργαστών ενώ η δεύτερη απαιτεί πολλαπλά αντίγραφα του προγράμματος δοκιμής αυξάνοντας υπερβολικά τις απαιτήσεις μεγέθους της κοινόχρηστης κρυφής μνήμης. Η τρίτη εναλλακτική λύση, ΕΑ-ΠΕ, υπερνικά τα παραπάνω μειονεκτήματα, αλλά δεν υποστηρίζει την διάγνωση στο επίπεδο των πυρήνων και χρονοτριβεί λόγω των ακυρώσεων συνοχής της κρυφής μνήμης.

Μια γενική λύση για την ελαχιστοποίηση των ακυρώσεων συνοχής της κρυφής μνήμης απαιτεί την αποθήκευση των αποκρίσεων της δοκιμής σε διαφορετικές περιοχές της κοινόχρηστης κρυφής μνήμης. Εάν κάθε επεξεργαστής αποθηκεύει τις αποκρίσεις της δοκιμής σε διαφορετικές θέσεις μνήμης, ο ελεγκτής κρυφής μνήμης δεν θα ακυρώσει καμία λέξη κατά τη διάρκεια εκτέλεσης της αυτοδοκιμής. Η προτεινόμενη προσέγγισή η οποία αποτελεί βελτίωση της προσέγγισης ΕΑ-ΠΕ προσαρμόζει το πρόγραμμα δοκιμής με τέτοιο τρόπο ώστε κάθε επεξεργαστής να αποθηκεύει τις αποκρίσεις της αυτοδοκιμής σε διαφορετικές περιοχές της κοινόχρηστης κρυφής μνήμης. Κατά συνέπεια, η διάγνωση στο επίπεδο των πυρήνων επιτρέπεται, ενώ παράλληλα δεν πραγματοποιείται καμία ακύρωση κατά τη διάρκεια εφαρμογής της δοκιμής.

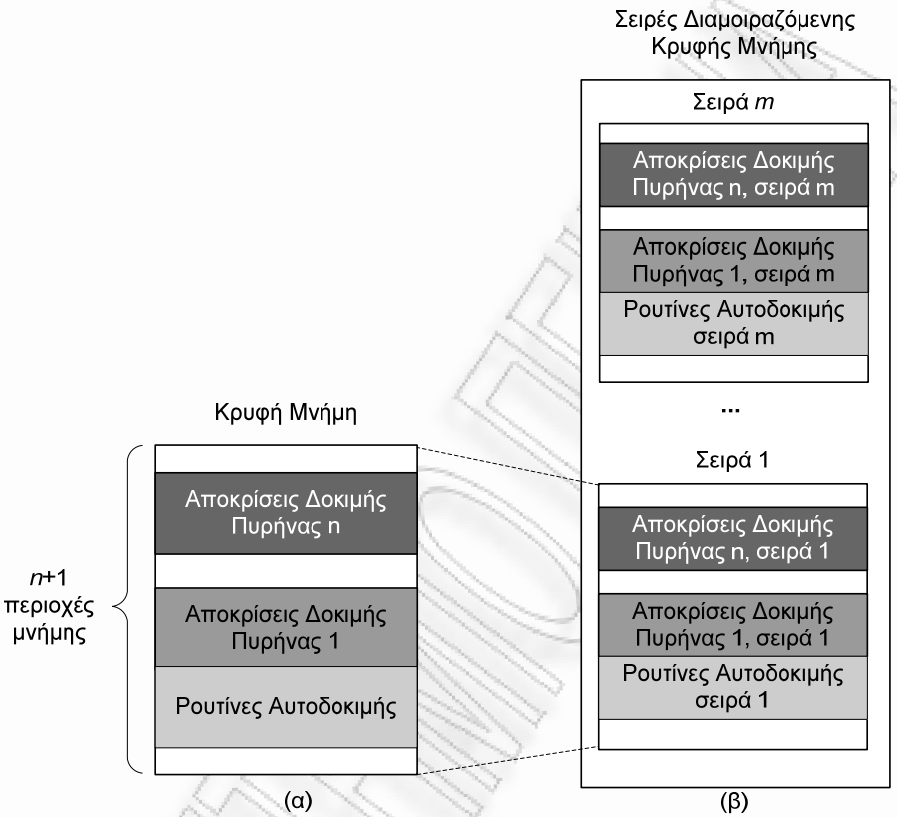
Στην προτεινόμενη προσέγγιση, το πρόγραμμα αυτοδοκιμής φορτώνεται μόνο μια φορά, ενώ προστίθεται ένα μικρό τμήμα κώδικα στην αρχή κάθε ρουτίνας αυτοδοκιμής το οποίο επαναπροσδιορίζει την τιμή του *δείκτη διεύθυνσης βάσης* (*Base Address Pointer, BAP*) με βάση το μοναδικό *κωδικό ταυτότητας* (*core id*) των πυρήνων. Η μεταβλητή του δείκτη διεύθυνσης βάσης χρησιμοποιείται από κάθε ρουτίνα αυτοδοκιμής για να προσδιορίσει τη θέση μνήμης όπου οι αποκρίσεις των επεξεργαστών θα αποθηκευτούν. Το επιπλέον τμήμα κώδικα διαβάζει τον κωδικό ταυτότητας του επεξεργαστή που εκτελεί τη συγκεκριμένη ρουτίνα αυτοδοκιμής και ορίζει μια νέα τιμή για κάθε δείκτη διεύθυνσης βάσης, έτσι ώστε οι αποκρίσεις της δοκιμής να

αποθηκευτούν στις διαφορετικές περιοχές της κοινόχρηστης κρυφής μνήμης που αφιερώνονται στον εκάστοτε επεξεργαστή. Ο ψευδοκώδικας που προστίθεται σε κάθε ρουτίνα δοκιμής παρουσιάζεται στην Εικόνα 4.15:

```
1.   Self_test_routine_Ri()
2.   Begin
3.   Self_test_routine_Ri()
# BAP_Ri : Base Address Pointer for routine Ri
# Assume that core Cj executes the routine
# and that it stores its responses in
# a memory region starting from address ADD_Cj
# Processor Identification
4.   Read core_id
# Apply Test Data
5.   Find ADD_Cj
# Change the Base Address Pointer (BAP)
6.   BAP_Ri ← F[ADD_Cj,BAP_Ri]
# Here is the test code …
7.   End
```

Εικόνα 4.15 Ψευδοκώδικας αποθήκευσης αποκρίσεων δοκιμής

Η Εικόνα 4.16 περιγράφει τον διαχωρισμό της κοινόχρηστης κρυφής μνήμης και της κατανομής του προγράμματος και των αποκρίσεων δοκιμής για κάθε πυρήνα, όπως περιγράφηκε στις φάσεις Α και Β της προτεινόμενης μεθοδολογίας. Στην αρχιτεκτονική του κοινού διαύλου επικοινωνίας, η κοινόχρηστη κρυφή μνήμη διαιρείται σε n+1 περιοχές, όπως φαίνεται στην Εικόνα 4.16 (α). Η πρώτη περιοχή της κοινόχρηστης κρυφής μνήμης χρησιμοποιείται για την αποθήκευση των ρουτινών αυτοδοκιμής, ενώ οι επόμενες n περιοχές χρησιμοποιούνται για την αποθήκευση των αντίστοιχων αποκρίσεων δοκιμής των n επεξεργαστών. Στην αρχιτεκτονική διασύνδεσης του σταυρωτού μεταγωγέα (Εικόνα 4.16 (β)) εφαρμόζεται ο διαχωρισμός της φάσης Β σε κάθε σειρά της κοινόχρηστης κρυφής μνήμης. Κατά συνέπεια, κάθε σειρά της κοινόχρηστης κρυφής μνήμης, η οποία αποθηκεύει ένα διαφορετικό τμήμα του προγράμματος αυτοδοκιμής σύμφωνα με τη φάση Α της μεθοδολογίας, διαιρείται σε n+1 περιοχές. Η πρώτη περιοχή χρησιμοποιείται για την αποθήκευση των ρουτινών αυτοδοκιμής, ενώ οι επόμενες n περιοχές χρησιμοποιούνται για να αποθηκεύσουν τις αποκρίσεις της δοκιμής των n επεξεργαστών για τις αντίστοιχες ρουτίνες αυτοδοκιμής.



Εικόνα 4.16 Διαχωρισμός κοινόχρηστης κρυφής μνήμης σύμφωνα με την προτεινόμενη μεθοδολογία (α) κοινός δίαυλος, (β) σταυρωτός μεταγωγέας

Ο χρόνος φόρτωσης της προτεινόμενης προσέγγισης ισούται με το αντίστοιχο χρόνο φόρτωσης του μονοπύρηνου επεξεργαστή, προσθέτοντας ένα επιπλέον μικρό χρόνο για τη φόρτωση του κώδικα για την κατανομή των αποκρίσεων δοκιμής. Ο χρόνος συλλογής των αποκρίσεων είναι *n* φορές μεγαλύτερος από αυτόν του μονοπύρηνου επεξεργαστή. Ο ελεγκτής συνοχής κρυφής μνήμης δεν θα ακυρώσει κανένα από τα δεδομένα κατά την διάρκεια εφαρμογής της δοκιμής. Η μόνη επιβράδυνση κατά την εκτέλεση του προγράμματος δοκιμής οφείλεται στον ανταγωνισμό του μέσου επικοινωνίας. Στην φάση Γ που ακολουθεί εξετάζεται και επιλύεται το πρόβλημα αυτό.

4.7.3 Φάση Γ: Μείωση του ανταγωνισμού του διαύλου επικοινωνίας

Η φάση Γ στοχεύει στην μείωση του ανταγωνισμού του διαύλου επικοινωνίας, ο οποίος εμφανίζεται όταν δύο ή περισσότεροι επεξεργαστές ζητούν ταυτόχρονη πρόσβαση σε έναν κοινό πόρο (είτε στην κοινόχρηστη κρυφή μνήμη στην αρχιτεκτονική του κοινού διαύλου επικοινωνίας, είτε στις σειρές της κοινόχρηστης κρυφής μνήμης στην αρχιτεκτονική του σταυρωτού μεταγωγέα). Η προτεινόμενη μεθοδολογία δρομολογεί τις ρουτίνες αυτοδοκιμής

μεταξύ των επεξεργαστών με διαφορετική διάταξη ώστε ο μέσος αριθμός των επεξεργαστών που ζητούν πρόσβαση από την κοινόχρηστη κρυφή μνήμη να ελαχιστοποιείται.. Η μεθοδολογία βασίζεται στα στατιστικά δεδομένα εκτέλεσης του προγράμματος δοκιμής (τα οποία χρησιμοποιούνται και στη φάση Α):

- το συνολικό απαιτούμενο αριθμό αιτήσεων προς την κοινόχρηστη κρυφή μνήμη κάθε ρουτίνας αυτοδοκιμής και
- το συνολικό χρόνο εκτέλεσης κάθε ρουτίνας στην περίπτωση του μονοπύρηνου επεξεργαστή.

Οι αιτήσεις ανάγνωσης/εγγραφής από τους επεξεργαστές προς την κοινόχρηστη κρυφή μνήμη μέσω του δικτύου επικοινωνίας είναι δυνατόν να μοντελοποιηθούν ως ένα σύστημα ουράς αναμονής [153]. Στο σύστημα αυτό γίνονται οι ακόλουθες υποθέσεις: (α) ο αριθμός των επεξεργαστών είναι πεπερασμένος και (β) κάθε επεξεργαστής εισάγεται στην ουρά αναμονής όταν υποβάλλει ένα αίτημα προς την κοινόχρηστη κρυφή μνήμη. Οι βασικές παράμετροι για την μελέτη ενός συστήματος ουράς αναμονής είναι:

n : Αριθμός πυρήνων του συστήματος πολυεπεξεργαστών.

m : Αριθμός σειρών κοινόχρηστης κρυφής μνήμης.

v : Αριθμός ρουτινών του προγράμματος αυτοδοκιμής που αποθηκεύονται στην κοινόχρηστη κρυφή μνήμη.

P_κ : Πιθανότητα k επεξεργαστές ($0 \le k \le n$) να βρίσκονται στην ουρά αναμονής περιμένοντας να εξυπηρετηθούν από την κοινόχρηστη κρυφή μνήμη.

λ_κ : Μέσος ρυθμός αιτήσεων άφιξης από τους επεξεργαστές, δεδομένου ότι k επεξεργαστές είναι στην ουρά αναμονής (κατάσταση k).

μ_κ : Μέσος ρυθμός εξυπηρέτησης κοινόχρηστης κρυφής μνήμης, δεδομένου ότι k επεξεργαστές είναι στην ουρά αναμονής (κατάσταση k).

$\overline{\mu}$: Μέσος χρόνος εξυπηρέτησης (μέσος χρόνος πρόσβασης ανάγνωσης – εγγραφής της κοινόχρηστης κρυφής μνήμης σε κύκλους ρολογιού).

λ : Μέσος ρυθμός αιτήσεων άφιξης από τους επεξεργαστές.

μ : Μέσος ρυθμός εξυπηρέτησης της κοινόχρηστης κρυφής μνήμης.

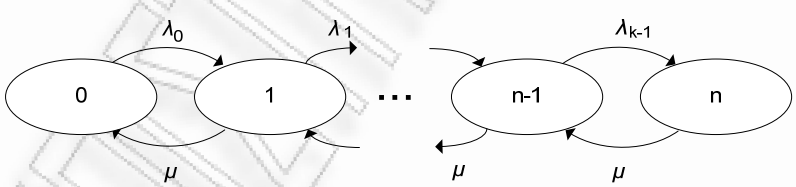
L_q : Μέσος αριθμός επεξεργαστών στην ουρά αναμονής του κοινού διαύλου επικοινωνίας.

$\overline{L}_q$: Μέσος αριθμός επεξεργαστών στην ουρά αναμονής του σταυρωτού μεταγωγέα.

Παρατηρείστε ότι χρησιμοποιούμε διαφορετική παράμετρο για τον υπολογισμό του μέσου αριθμού επεξεργαστών στην ουρά αναμονής για τον κοινό δίαυλο επικοινωνίας (L_q) και για τον σταυρωτό μεταγωγέα ($\overline{L}_q$). Ο μέσος αριθμός επεξεργαστών στην ουρά αναμονής σε ένα

σύστημα πολυεπεξεργαστών όπου το μέσο διασύνδεσης υλοποιείται με έναν κοινό δίαυλο, υπολογίζεται σύμφωνα με τον αριθμό των επεξεργαστών που αναμένουν να αποκτήσουν πρόσβαση στον μοναδικό αυτό δίαυλο. Στην περίπτωση του σταυρωτού μεταγωγέα, κάθε ουρά αναμονής αντιστοιχεί σε μία σειρά της κοινόχρηστης κρυφής μνήμης. Δεδομένου ότι ο κώδικας δοκιμής θα κατανεμηθεί στις διαφορετικές σειρές της κοινόχρηστης κρυφής μνήμης, σύμφωνα με την Φάση Α της προτεινόμενης μεθοδολογίας, η ουρά αναμονής υπολογίζεται από το μέσο όρο των ουρών αναμονής του συστήματος. Κατά συνέπεια, δημιουργούνται *m* ουρές αναμονής από τις οποίες και υπολογίζεται ο μέσος αριθμός επεξεργαστών στην ουρά αναμονής.

Το διάγραμμα μεταβάσεων ενός συστήματος αναμονής απεικονίζεται στην Εικόνα 4.17. Στην κατάσταση *0* δεν συμβαίνει καμία αίτηση πρόσβασης προς την κοινόχρηστη κρυφή μνήμη με αποτέλεσμα να μην υπάρχει επεξεργαστής στην ουρά αναμονής. Σε όλες τις άλλες καταστάσεις *1,2,...,n*, ο συνολικός αριθμός των επεξεργαστών στην ουρά αναμονής είναι *1,2,...,n* αντίστοιχα. Στο σύστημα αυτό, όλοι οι επεξεργαστές δρουν ανεξάρτητα από τους άλλους και ο μέσος ρυθμός άφιξης λ_κ ποικίλει ανάλογα με την εκάστοτε κατάσταση *k* της ουράς αναμονής και τον επεξεργαστή που εξυπηρετείται στην κατάσταση αυτή. Μολονότι, η κοινόχρηστη κρυφή μνήμη εξυπηρετεί τις αιτήσεις των επεξεργαστών με σταθερό ρυθμό, ο μέσος ρυθμός εξυπηρέτησης της κοινόχρηστης κρυφής μνήμης (μ_κ) στην κατάσταση *k* είναι $\mu_\kappa=\mu$.



Εικόνα 4.17 Διάγραμμα καταστάσεων ενός συστήματος ουράς αναμονής

Οι αιτήσεις πρόσβασης από τους επεξεργαστές προς την κοινόχρηστη κρυφή μνήμη μέσω του διαύλου επικοινωνίας μπορούν να μοντελοποιηθούν ως ένα σύστημα ουράς αναμονής. Όπως προαναφέρθηκε, η προτεινόμενη προσέγγιση δρομολογεί τις ρουτίνες αυτοδοκιμής μεταξύ των πυρήνων με στόχο την ελαχιστοποίηση του αριθμού των επεξεργαστών που περιμένουν να αποκτήσουν πρόσβαση στη κοινόχρηστη κρυφή μνήμη. Δηλαδή, την ελαχιστοποίηση του L_q ή του $\overline{L}_q$ για τον κοινό δίαυλο και τον σταυρωτό μεταγωγέα, αντίστοιχα. Ο μέσος ρυθμός αιτήσεων άφιξης από τους επεξεργαστές λ_k, προσεγγίζεται υπό την υπόθεση ότι ο μέσος ρυθμός άφιξης λ ακολουθεί κατανομή Poisson [153]. Ο μέσος ρυθμός εξυπηρέτησης είναι σταθερός (μ) και ο μέσος χρόνος εξυπηρέτησης της κοινόχρηστης κρυφής μνήμης δίνεται από τον ακόλουθο τύπο:

$$\overline{\mu} = \frac{(R_r \cdot S_r) + (R_w \cdot S_w)}{R_r + R_w} \qquad (1)$$

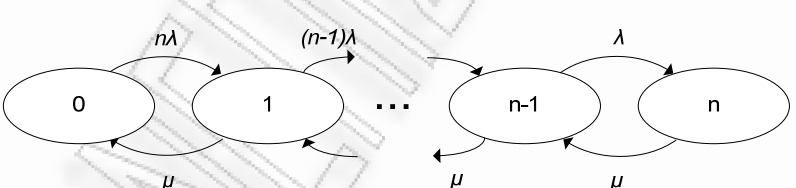
όπου τα R_r και R_w είναι ο συνολικός αριθμός αιτήσεων ανάγνωσης και εγγραφής των ρουτινών αυτοδοκιμής, αντίστοιχα, όταν αυτές εκτελούνται από τον μονοπύρηνο επεξεργαστή. Τα S_r και S_w είναι ο χρόνος πρόσβασης, σε κύκλους ρολογιού, της κοινόχρηστης κρυφής μνήμης για τις αιτήσεις ανάγνωσης και εγγραφής, αντίστοιχα. Υποθέτοντας ότι η *μνήμη προσωρινής αποθήκευσης των αιτήσεων εγγραφής* (*write buffers*) δεν είναι ποτέ πλήρης, τότε ισχύει $R_w=0$ και $\overline{\mu} = S_r$.

Σύμφωνα με τις παραπάνω υποθέσεις, το σύστημα ουράς αναμονής για τους συμμετρικούς πολυεπεξεργαστές είναι το *M/M/1/n* [153], όπου το "*M/M*" υποδηλώνει ότι οι αφίξεις και οι αναχωρήσεις ακολουθούν κατανομή Poisson, το "*1*" υποδηλώνει τον ενιαίο εξυπηρετητή του συστήματος αναμονής (την κοινόχρηστη κρυφή μνήμη), και το "*n*" είναι ο αριθμός των επεξεργαστών. Το διάγραμμα καταστάσεων μετάβασης του συστήματος *M/M/1/n* απεικονίζεται στην Εικόνα 4.18 και οι μεταβλητές λ_k, μ_k, P_0, L_q και $\overline{L_q}$ δίνονται από τους ακόλουθους τύπους [153]:

$$\lambda_k = \lambda(n-k),\ 0 \le k \le n \qquad L_q = n - \frac{\lambda+\mu}{\lambda}(1-P_0) \qquad (2)$$

$$P_0 = \frac{1}{\sum_{k=0}^{n} \frac{n!}{(n-k)!}\left(\frac{\lambda}{\mu}\right)^k} \qquad (3)$$

$$\mu_k = \mu,\ 0 < k \le n \qquad \bar{L}_q = \frac{\sum (n - \frac{\lambda+\mu}{\lambda}(1-P_0))}{m} \qquad (2')$$



Εικόνα 4.18 Διάγραμμα καταστάσεων ενός M/M/1/n συστήματος ουράς αναμονής

Η προτεινόμενη μεθοδολογία δρομολογεί τις ρουτίνες αυτοδοκιμής στους επεξεργαστές σύμφωνα με τους συνδυασμούς ρουτινών που επιτυγχάνουν τη μικρότερη μέση τιμή επεξεργαστών στη ουρά αναμονής. Οι υπολογισμοί βασίζονται στη ρουτίνα αυτοδοκιμής με το μικρότερο χρόνο εκτέλεσης (σε κύκλους ρολογιού). Τα t_i και r_i, $1 \le i \le v$ αποτελούν τον συνολικό χρόνο εκτέλεσης σε κύκλους ρολογιού του μονοπύρηνου επεξεργαστή καθώς και τις συνολικές αιτήσεις πρόσβασης της κοινόχρηστης κρυφής μνήμης ανά ρουτίνα δοκιμής *i*, αντίστοιχα.

Για κάθε συνδυασμό ρουτινών αυτοδοκιμής, οι ρουτίνες ταξινομούνται ως προς τον χρόνο εκτέλεσης $t_1 < t_2 < ... < t_n$ και υπολογίζεται ο συνολικός αριθμός αιτήσεων στην κοινόχρηστη κρυφή μνήμη, ο συνολικός ρυθμός εξυπηρέτησης και οι αντίστοιχες πιθανότητες αναμονής σε ένα διάστημα χρόνου t_1 κύκλων ρολογιού (ο χρόνος εκτέλεσης της μικρότερης ρουτίνας).

$$\lambda = \frac{r_1 + \sum_{i=2}^{n} \frac{r_i \cdot t_1}{t_i}}{n} \quad (4) \qquad \mu = \frac{t_1}{=} \quad (5)$$

Ο αλγόριθμος που δρομολογεί τις ρουτίνες στους επεξεργαστές είναι ο ακόλουθος:

Βήμα 1: Υπολογίστε τον μέσο χρόνο εξυπηρέτησης ($\overline{\mu}$) βάση της εξίσωσης (1) και τον αριθμό των συνδυασμών των ρουτινών δοκιμής.

Βήμα 2: Για κάθε συνδυασμό ρουτινών, προσδιορίστε τη ρουτίνα αυτοδοκιμής με τον μικρότερο χρόνο εκτέλεσης.

Βήμα 3: Υπολογίστε τον μέσο ρυθμό αιτήσεων άφιξης (λ) και τον μέσο χρόνο εξυπηρέτησης της κοινόχρηστης κρυφής μνήμη (μ) για κάθε συνδυασμό ρουτινών βάση των εξισώσεων (4), (5).

Βήμα 4: Υπολογίστε την πιθανότητα P_o (η ουρά αναμονής να είναι άδεια), καθώς και τον μέσο αριθμό επεξεργαστών στην ουρά αναμονής (L_q ή $\overline{L}_q$ για τον κοινό δίαυλο και για τον σταυρωτό μεταγωγέα, αντίστοιχα) σύμφωνα με τις εξισώσεις (2) ή (2') και (3).

Βήμα 5: Επιλέξτε τον συνδυασμό ρουτινών με το μικρότερο μέσο αριθμό επεξεργαστών (L_q ή $\overline{L}_q$) στην ουρά αναμονής και δρομολογήστε τις ρουτίνες στους πυρήνες επεξεργαστών.

Βήμα 6: Προσδιορίστε την ρουτίνα με το μικρότερο χρόνο εκτέλεσης για κάθε συνδυασμό. Αντικαταστήστε αυτή την ρουτίνα με την ρουτίνα που όταν συνδυάζεται με τις εκτελούμενες, η τιμή του μέσου αριθμού επεξεργαστών στην ουρά αναμονής (L_q ή $\overline{L}_q$) ελαχιστοποιείται.

Βήμα 7: Επιστρέψτε στο βήμα 6 μέχρι να ολοκληρώσετε την εκτέλεση όλων των αναγκαίων ρουτινών δοκιμής από τους επεξεργαστές.

Βήμα 8: Εάν στο τέλος της δρομολόγησης μια ή περισσότερες ρουτίνες έχουν δρομολογηθεί παράλληλα για τους επεξεργαστές, τότε ακυρώστε την τρέχουσα δρομολόγηση. Οι ρουτίνες αυτές δρομολογούνται πρώτες σύμφωνα με τον συνδυασμό που ελαχιστοποιεί την τιμή του L_q ή του $\overline{L}_q$. Επιστρέψτε στο βήμα 6 για να δρομολογήσετε τις υπόλοιπες ρουτίνες αυτοδοκιμής.

Ο αλγόριθμος μπορεί να αυτοματοποιηθεί ώστε να οδηγήσει γρήγορα σε μια αποτελεσματική δρομολόγηση των ρουτινών αυτοδοκιμής βασιζόμενος στις γνωστές τιμές αιτήσεων των ρουτινών από την προσομοίωση του μονοπύρηνου επεξεργαστή. Στην συνέχεια, παρουσιάζεται ένα παράδειγμα εφαρμογής του αλγόριθμου.

Παράδειγμα: Έστω ένας συμμετρικός πολυεπεξεργαστής που αποτελείται από n=3 επεξεργαστές συνδεδεμένους με ένα κοινό δίαυλο επικοινωνίας και ένα πρόγραμμα αυτοδοκιμής αποτελούμενο από $v=4$ ρουτίνες. Ο Πίνακας 4.5 παρουσιάζει τις αιτήσεις πρόσβασης προς την κοινόχρηστη κρυφή μνήμη δευτέρου επιπέδου, καθώς και το συνολικό χρόνο εκτέλεσης των ρουτινών αυτοδοκιμής σε κύκλους ρολογιού.

Ρουτίνες Δοκιμής	Χρόνος Εκτέλεσης	Αιτήσεις Ανάγνωσης	Αιτήσεις Εγγραφής	Συνολικές Αιτήσεις
R_1	40	4	-	4
R_2	60	3	2	5
R_3	80	4	-	4
R_4	60	6	-	6
Σύνολο	**240**	R_r **= 17**	R_w**=2**	**19**

Πίνακας 4.5 Στατιστικά εκτέλεσης ρουτινών παραδείγματος

Στο σύστημα της μνήμης του παραδείγματος, θεωρείστε ότι ο χρόνος εξυπηρέτησης της κρυφής μνήμης δευτέρου επιπέδου για ανάγνωση (S_r) είναι 4 κύκλοι ρολογιού, ενώ ο αντίστοιχος χρόνος εγγραφής (S_w) είναι 1 κύκλος ρολογιού υποθέτοντας ότι η προσωρινή μνήμη εγγραφής δεν είναι ποτέ πλήρης.

Βήματα 1–4: Υπολογισμός των τιμών των λ, μ, P_0 and L_q για όλους τους πιθανούς συνδυασμούς ρουτινών. Ο Πίνακας 4.6 παρουσιάζει τα σχετικά αποτελέσματα.

Συνδυασμοί Ρουτινών	Ελάχιστος Χρόνος	λ	μ	P_0	L_q
[R_1,R_2 R_3]	40	3.11	10.85	0.40	0.31
[R_1,R_2,R_4]	40	3.78	10.85	0.33	0.40
[R_1,R_3,R_4]	40	3.33	10.85	0.38	0.34
[R_2,R_3,R_4]	60	4.67	16.28	0.40	0.31

Πίνακας 4.6 Τιμές των παραμέτρων της M/M/1//n ουράς αναμονής που υπολογίζονται στα τέσσερα πρώτα βήματα του αλγορίθμου

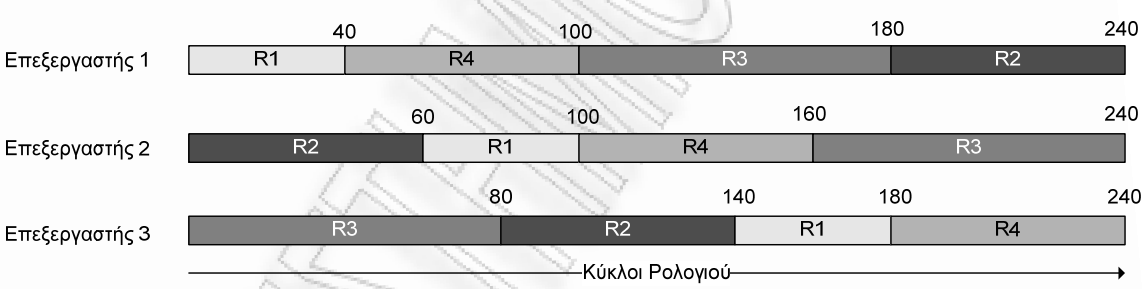
Βήμα 5: Οι συνδυασμοί ρουτινών με την μικρότερη τιμή του μέσου αριθμού επεξεργαστών στην ουρά αναμονής (L_q) είναι οι [R_1,R_2,R_3] και [R_2,R_3,R_4]. Επιλέγεται τυχαία ο συνδυασμός [R_1,R_2,R_3].

Βήμα 6: Η ταχύτερη ως προς τον χρόνο εκτέλεσης ρουτίνα είναι η R_1. Οι επεξεργαστές 2 και 3 συνεχίζουν την εκτέλεση των ρουτινών R_2 και R_3, αντίστοιχα. Αναζητείται ο συνδυασμός ο οποίος περιλαμβάνει τις ρουτίνες R_2 και R_3, αντικαθιστά την ρουτίνα R_1 με κάποια άλλη και δίνει την μικρότερη τιμή του μέσου αριθμού επεξεργαστών στην ουρά αναμονής (L_q). Ο συνδυασμός αυτός είναι ο [R_2,R_3,R_4].

Βήμα 7: Επανάληψη του βήματος 6 μέχρις ότου όλοι οι επεξεργαστές να εκτελέσουν όλες τις ρουτίνες δοκιμής.

Βήμα 8: Δεν εκτελείται στο συγκεκριμένο παράδειγμα.

Η δρομολόγηση των ρουτινών αυτοδοκιμής όπως προέκυψε από την προτεινόμενη μεθοδολογία, απεικονίζεται στην Εικόνα 4.19.



Εικόνα 4.19 Παράδειγμα – Δρομολόγηση ρουτινών

Συνοψίζοντας, κρίνεται απαραίτητο να αναφερθεί ότι κάθε επεξεργαστής έχει τον δικό του *δρομολογητή δοκιμής* (*test scheduler*) ο οποίος καθορίζει την σειρά εκτέλεσης των ρουτινών σε κάθε επεξεργαστή. Ο δρομολογητής δοκιμής αποτελεί ένα μικρό τμήμα κώδικα το οποίο τοποθετείται στην συνάρτηση *main* κάθε επεξεργαστή και *καλεί* (*call*) τις ρουτίνες δοκιμής την μία μετά την άλλη σύμφωνα με την σειρά εκτέλεσης που καθορίστηκε από την Φάση Γ της προτεινόμενης μεθοδολογίας.

4.8 Πειραματικά αποτελέσματα

Η αποτελεσματικότητα της προτεινόμενης μεθοδολογίας [29], [31] αποδεικνύεται σε διπύρηνους, τετραπύρηνους και οκταπύρηνους πολυεπεξεργαστές οι οποίοι βασίζονται στον επεξεργαστή ανοιχτού κώδικα OpenRISC 1200 [24]. Στην συνέχεια, η μεθοδολογία [30] εφαρμόζεται σε μια υλοποίηση εμπορικού συμμετρικού πολυεπεξεργαστή υψηλής απόδοσης από την εταιρία Sun Microsystems [136], τον επεξεργαστή OpenSPARC T1 [137].

4.8.1 Συμμετρικοί πολυεπεξεργαστές με βάση τον πυρήνα OpenRISC 1200

Αναφορικά με την αρχιτεκτονική του κοινού διαύλου επικοινωνίας υλοποιήσαμε δύο συμμετρικούς πολυεπεξεργαστές. Ο πρώτος αποτελείται από δύο πυρήνες και ονομάζεται OpenRISC Duo ενώ ο δεύτερος από τέσσερις πυρήνες και ονομάζεται OpenRISC Quad. Όσον αφορά την αρχιτεκτονική του σταυρωτού μεταγωγέα υλοποιήσαμε τρεις συμμετρικούς πολυεπεξεργαστές. Ένας με τέσσερις πυρήνες και δύο σειρές κοινόχρηστης κρυφής μνήμης ο οποίος ονομάζεται OpenRISC Quad 2 και δύο με οκτώ πυρήνες που ονομάζονται OpenRISC Octo 2 και OpenRISC Octo 4 και ενσωματώνουν δύο και τέσσερις σειρές κοινόχρηστης κρυφής μνήμης, αντίστοιχα. Ο Πίνακας 4.7 απεικονίζει τα χαρακτηριστικά των πολυεπεξεργαστών, ενώ ο Πίνακας 4.8 απεικονίζει τον συνολικό αριθμό πυλών του επεξεργαστή και τον αριθμό των ελαττωμάτων ανά μονάδα. Όλοι οι επεξεργαστές έχουν προκύψει από σύνθεση με την χρήση μιας τεχνολογικής βιβλιοθήκης των 0.18 um.

Χαρακτηριστικό	**Duo**	**Quad**	**Quad 2**	**Octo 2**	**Octo 4**
Πυρήνες	**2**	**4**	**4**	**8**	**8**
Τοπική κρυφή μνήμη L1	8 KB I-Cache + 8 KB D-Cache				
Πολιτική εγγραφής (L1)	Απευθείας εγγραφή (Write Through)				
Μέγεθος μπλοκ	16-Bytes				
Διασύνδεση	Κοινός δίαυλος		Σταυρωτός μεταγωγέας		
Τύπος διασύνδεσης	Wishbone				
Συνοχή κρυφής μνήμης	Κατασκοπεία (Snooping)				
Κοινόχ. κρυφή μνήμη L2 (KB)	128	256	256	512	512
Πολιτική εγγραφής (L2)	Απευθείας εγγραφή (Write Through)				
Αριθμός σειρών κρυφ. μνήμης	-	-	2	2	4

Πίνακας 4.7 Χαρακτηριστικά πολυεπεξεργαστών

Μονάδα	**Duo**	**Quad**	**Quad 2**	**Octo 2**	**Octo 4**
Επεξεργαστές	167320	334640	334600	669280	669280
Ελεγκ. κρυφής μνήμης εντολών	2476	4952	4952	9904	9904
Ελεγκ. κρυφής μνήμης δεδομ.	3396	6792	8496	16992	21192
Καταχωρ. προσωρ. αποθήκ.	15180	30360	30360	60720	60720
Διαιτητής διαύλου	1212	3156	14668	33992	67984
Συνολικά ελαττώματα	**189584**	**379900**	**393076**	**790888**	**829080**
Συνολικές πύλες	**80897**	**162189**	**164226**	**329491**	**337343**

Πίνακας 4.8 Συνολικό πλήθος πυλών και ελαττωμάτων προσκόλλησης των πολυεπεξεργαστών

Ο Πίνακας 4.9 παρουσιάζει ένα σύνολο ρουτινών που δοκιμάζουν αποτελεσματικά τον πυρήνα OpenRISC 1200 και κατά συνέπεια κάθε μεμονωμένο επεξεργαστή των συμμετρικών πολυεπεξεργαστών. Οι στήλες 3 έως 5 παραθέτουν τα στατιστικά δεδομένα εκτέλεσης των ρουτινών αυτοδοκιμής: τον χρόνο εκτέλεσης σε κύκλους ρολογιού, καθώς και τις αιτήσεις εγγραφής και ανάγνωσης προς την κρυφή μνήμη δευτέρου επιπέδου. Το πρόγραμμα αυτοδοκιμής αποτελείται από έξι ντετερμινιστικές ρουτίνες δοκιμής οι οποίες προτάθηκαν στην ερευνητική εργασία [97] για τη δοκιμή του επεξεργαστή OpenRISC 1200 (Comparator, Register Bank, Control Logic, Shifter, Multiplier and ALU) και τρεις ψευδοτυχαίες ρουτίνες δοκιμής (Multiplier-LFSR, Shifter-LFSR and Adder-LFSR). Το σύνολο των ρουτινών αυτών δοκιμάζει αποτελεσματικά όλες τις λειτουργικές μονάδες και τις μονάδες ελέγχου του επεξεργαστή OpenRISC 1200. Η προσθήκη των ψευδοτυχαίων ρουτίνων δοκιμής πραγματοποιήθηκε για την αύξηση της ποικιλίας του συνόλου των ρουτινών δοκιμής.

Ρουτίνες δοκιμής	**Χρόνος εκτέλεσης (Κύκλοι)**	**Κρυφή Μνήμη Δευτέρου Επιπέδου**	
		Αιτήσεις Ανάγνωσης	**Αιτήσεις Εγγραφής**
Comparator (**cp**)	17488	209	104
Multiplier – LFSR (**mll**)	11949	12	400
Shifter – LFSR (**sftl**)	11460	10	200
Adder – LFSR (**adl**)	11459	10	200
Register Bank (**reg**)	11452	123	256
Control Logic (**cnt**)	6329	56	67
Shifter (**sft**)	5636	52	60
Multiplier (**mul**)	5438	13	256
ALU (**alu**)	4536	29	70

Πίνακας 4.9 Στατιστικά δεδομένα ρουτινών αυτοδοκιμής

Στους Πίνακες που ακολουθούν (Πίνακας 4.10 έως Πίνακας 4.14) παρουσιάζεται η δρομολόγηση των ρουτινών δοκιμής (φάση Γ) και η κατανομή του κώδικα δοκιμής στις σειρές της κοινόχρηστης κρυφής μνήμης για τους πολυεπεξεργαστές που χρησιμοποιούν τον σταυρωτό μεταγωγέα σύμφωνα με την φάση Α της προτεινόμενης μεθοδολογίας για τα πέντε συστήματα των συμμετρικών πολυεπεξεργαστών. Παρατηρήστε ότι στους συμμετρικούς πολυεπεξεργαστές με σταυρωτό μεταγωγέα, οι ρουτίνες δοκιμής έχουν κατανεμηθεί στις διαφορετικές σειρές της κοινόχρηστης κρυφής μνήμης, ενώ παράλληλα κάθε χρονική στιγμή οι ρουτίνες εκτελούνται από διαφορετικές ομάδες επεξεργαστών. Για παράδειγμα, στον συμμετρικό πολυεπεξεργαστή Octo 4 εμφανίζονται τέσσερις ομάδες επεξεργαστών (P0-P1, P2-P3, P4-P5 και P6-P7) οι οποίες εκτελούν τις ρουτίνες δοκιμής από τις σειρές 0 έως 3 της κοινόχρηστης κρυφής μνήμης.

Επιπλέον, είναι σημαντικό να αναφερθεί ότι οι πίνακες αυτοί παρουσιάζουν μόνο τη σειρά εκτέλεσης των ρουτινών δοκιμής και δεν συμπεριλαμβάνουν οποιαδήποτε χρονική πληροφορία σχετικά με την έναρξη και την λήξη της εκτέλεσης των ρουτινών. Η παρατήρηση αυτή είναι σημαντική, ιδιαίτερα για τον πολυεπεξεργαστή Octo 4, όπου η σειρά 0 της κρυφής μνήμης εμφανίζεται ότι προσπελαύνεται ταυτόχρονα σε κάποιες χρονικές στιγμές από πολλούς επεξεργαστές. Τέλος, για την αρχιτεκτονική διασύνδεσης του σταυρωτού μεταγωγέα κρίνεται απαραίτητη η διαμέριση ορισμένων ρουτινών δοκιμής για την αποδοτικότερη κατανομή τους στην κοινόχρηστη κρυφή μνήμη σύμφωνα με τη φάση Α της μεθοδολογίας. Στις περιπτώσεις των συμμετρικών πολυεπεξεργαστών με σταυρωτό μεταγωγέα, στο κάτω μέρος των πινάκων απεικονίζεται το "σπάσιμο" των ρουτινών δοκιμής καθώς και το πλήθος των αποκρίσεων δοκιμής κάθε υπορουτίνας. Για παράδειγμα, στο πολυεπεξεργαστή Octo 4, η ρουτίνα adl χωρίζεται σε δύο υπορουτίνες, οι οποίες ονομάζονται adl_1 και adl_2 και παράγουν 41 και 159 αποκρίσεις δοκιμής, αντίστοιχα.

Πυρήνας 0	adl	sftl	mll	mul	alu	cp	sft	reg	cnt
Πυρήνας 1	alu	Cp	sft	reg	cnt	adl	sftl	mll	mul

Πίνακας 4.10 Δρομολόγηση ρουτινών για τον πολυεπεξεργαστή OpenRISC Duo

Πυρήνας 0	adl	sftl	mul	mll	reg	alu	Cnt	cp	sft
Πυρήνας 1	mll	Cp	alu	cnt	sft	reg	Adl	sftl	mul
Πυρήνας 2	reg	mll	alu	adl	sftl	mul	Sft	cnt	cp
Πυρήνας 3	alu	cnt	sft	reg	cp	adl	Sftl	mul	mll

Πίνακας 4.11 Δρομολόγηση ρουτινών για τον πολυεπεξεργαστή OpenRISC Quad

	Σειρά 0					**Σειρά 1**				
Πυρήνας 0	adl	mul	$sftl_{[1]}$	alu	cp	mll	$sftl_{[2]}$	Sft	reg	cnt
Πυρήνας 1	alu	cp	adl	mul	$sftl_{[1]}$	sft	reg	Cnt	mll	$sftl_{[2]}$
	Σειρά					**Σειρά 0**				
Πυρήνας 2	mll	$sftl_{[2]}$	sft	reg	cnt	adl	mul	$sftl_{[1]}$	alu	cp
Πυρήνας 3	sft	reg	cnt	mll	$sftl_{[2]}$	alu	cp	addl	mul	$sftl_{[1]}$
$sftl_{[1]}$=77 διανύσματα						$sftl_{[2]}$=123 διανύσματα				

Πίνακας 4.12 Δρομολόγηση ρουτινών για τον πολυεπεξεργαστή OpenRISC Quad 2

	Σειρά 0					Σειρά 1				
Πυρήνας 0	adl	mul	$mll_{[1]}$	alu	cp	sftl	$mll_{[2]}$	Cnt	sft	reg
Πυρήνας 1	cp	adl	mul	$mll_{[1]}$	alu	$mll_{[2]}$	cnt	Reg	sftl	sft
Πυρήνας 2	alu	cp	adl	mul	$mll_{[1]}$	reg	sft	$mll_{[2]}$	sft_l	cnt
Πυρήνας 3	mul	$mll_{[1]}$	alu	cp	adl	cnt	sft	Sftl	seg	$mll_{[2]}$
	Σειρά 1					**Σειρά 0**				
Πυρήνας 4	sftl	$mtl_{[2]}$	cnt	sft	reg	adl	mul	$mll_{[1]}$	alu	cp
Πυρήνας 5	$mll_{[2]}$	cnt	reg	sftl	sft	cp	adl	Mul	$mll_{[1]}$	alu
Πυρήνας 6	reg	sft	$mll_{[2]}$	sftl	cnt	alu	cp	Adl	mul	$mll_{[1]}$
Πυρήνας 7	cnt	sft	sftl	reg	$mll_{[2]}$	mul	$mull_{[1]}$	Alu	cp	adl
	$mtl_{[1]}$=64 διανύσματα					$mtl_{[2]}$=135 διανύσματα				

Πίνακας 4.13 Δρομολόγηση ρουτινών για τον πολυεπεξεργαστή OpenRISC Octo 2

	Σειρά 0						Σειρά 1		Σειρά 2			Σειρά 3	
Πυρήνας 0	adl_1	mll_1	$sftl_1$	mul	cp_1	alu	adl_2	cp_2	mll_2	sft	cnt	sfl_2	reg
Πυρήνας 1	cp_1	alu	adl_1	mll_1	$sftl_1$	mul	cp_2	adl_2	sft	cnt	mll_2	reg	sfl_2
	Σειρά 1		**Σειρά 2**			**Σειρά 3**		**Σειρά 0**					
Πυρήνας 2	adl_2	cp_2	mll_2	sft	cntr	sfl_2	reg	adl_1	mll_1	$sftl_1$	mul	cp_1	alu
Πυρήνας 3	cp_2	adl_2	sft	cnt	mll_2	reg	sfl_2	cp_1	alu	adl_1	mll_1	$sftl_1$	mul
	Σειρά 2			**Σειρά 3**		**Σειρά 0**						**Σειρά 1**	
Πυρήνας 4	mll_2	sft	cnt	sfl_2	reg	adl_1	mll_1	$sftl_1$	mul	cp_1	alu	adl_2	cp_2
Πυρήνας 5	sft	cnt	mll_2	reg	sfl_2	cp_1	alu	adl_1	mll_1	$sftl_1$	mul	cp_2	adl_2
	Σειρά 3		**Σειρά 0**						**Σειρά 1**		**Σειρά 2**		
Πυρήνας 6	sfl_2	reg	adl_1	mll_1	$sftl_1$	mul	cp_1	alu	adl_2	cp_2	mll_2	sft	cnt
Πυρήνας 7	reg	sfl_2	cp_1	alu	adl_1	mll_1	$sftl_1$	mul	cp_2	adl_2	sft	cnt	mll_2
adl_1=41 διανύσματα			mll_1=50 διανύσματα			sfl_1=38 διανύσματα			cp_1=35 διανύσματα				
adl_2=159 διανύσματα			mll_2=150 διανύσματα			sfl_2=162 διανύσματα			cp_2=165 διανύσματα				

Πίνακας 4.14 Δρομολόγηση ρουτινών για τον πολυεπεξεργαστή OpenRISC Octo 4

Στους επόμενου πίνακες (Πίνακας 4.15 έως Πίνακας 4.19) παρουσιάζονται τα αποτελέσματα προσομοίωσης ελαττωμάτων προσκόλλησης για τους πέντε πολυεπεξεργαστές. Για την προσομοίωση των ελαττωμάτων χρησιμοποιήθηκε το εργαλείο Tetramax της εταιρίας Synopsys [134]. Παρέχονται αναλυτικά αποτελέσματα για την εσωτερική λειτουργία κάθε μεμονωμένου πυρήνα, για το υποσύστημα μνήμης (ελεγκτές κρυφής μνήμης, μνήμες προσωρινής αποθήκευσης αιτήσεων εγγραφής), καθώς και για την διεπαφή διασύνδεσης (κύκλωμα διαιτησίας). Το ποσοστό κάλυψης ελαττωμάτων διαφέρει μερικώς μεταξύ των

πυρήνων καθώς οι ρουτίνες δοκιμής εκτελούνται με διαφορετική σειρά. Η σειρά εκτέλεσης των ρουτινών δοκιμής επηρεάζει την κατάσταση της διοχέτευσης, την συμπεριφορά της κρυφής μνήμης και κατά συνέπεια το ποσοστό κάλυψης ελαττωμάτων των επεξεργαστών σύμφωνα με τους παρακάτω τρόπους:

- Όταν οι πρώτες εντολές μιας ρουτίνας δοκιμής εισέρχονται στην διοχέτευση του επεξεργαστή, η ακολουθία των εντολών που βρίσκονται στα τελευταία στάδια της διοχέτευσης ανήκει στην προηγούμενη ρουτίνα δοκιμής. Η διαφορετική σειρά εκτέλεσης παράγει ανόμοιες καταστάσεις διοχέτευσης μεταξύ των επεξεργαστών, προκαλώντας διαφορετικές *συνθήκες κινδύνου* (*hazard condition*) και ενεργοποίηση διαφορετικών *μονοπατιών προώθησης* (*forwarding paths*).

- Οι εντολές του δρομολογητή δοκιμής (ενότητα 4.7.3) για κάθε πυρήνα επεξεργαστή αποθηκεύονται σε διαφορετικές διευθύνσεις μνήμης. Αυτό έχει ως αποτέλεσμα, οι τιμές της διεύθυνσης των εντολών (που χρησιμοποιούνται για παράδειγμα στον ελεγκτή κρυφής μνήμης εντολών) κατά την εκτέλεση του δρομολογητή δοκιμής να είναι διαφορετικές για κάθε πυρήνα.

- Η σειρά εκτέλεσης των ρουτινών δοκιμής επηρεάζει τις *αστοχίες* (*miss*) της τοπικής κρυφής μνήμης εντολών και δεδομένων των επεξεργαστών.

Η μεθοδολογία επιτυγχάνει υψηλό ποσοστό κάλυψης ελαττωμάτων προσκόλλησης, μεγαλύτερο του 90% σε όλους τους συμμετρικούς πολυεπεξεργαστές.

Μονάδα	**Ποσοστό κάλυψης ελαττωμάτων (%)**		
	Πυρήνας 1	**Πυρήνας 2**	**Σύνολο**
Επεξεργαστές	90.79	90.81	90.80
Ελεγκτής κρυφής μνήμης εντολών	91.63	91.57	91.60
Ελεγκτής κρυφής μνήμης δεδομένων	90.21	90.18	90.20
Μνήμη προσωρινής αποθήκευσης	94.15	94.10	94.13
Κύκλωμα διαιτησίας	90.35		90.35
Σύνολο			**91.06**

Πίνακας 4.15 Ποσοστό κάλυψης ελαττωμάτων για τον επεξεργαστή OpenRISC Duo

Μονάδα	Ποσοστό κάλυψης ελαττωμάτων (%)				
	Πυρήνας 1	Πυρήνας 2	Πυρήνας 3	Πυρήνας 4	Σύνολο
Επεξεργαστές	90.78	90.79	90.82	90.80	90.80
Ελεγκτής κρυφής μνήμης εντολών	91.59	91.62	91.43	91.69	91.58
Ελεγκτής κρυφής μνήμης δεδομένων	90.16	90.03	90.16	90.23	90.15
Μνήμη προσωρινής αποθήκευσης	94.12	93.65	94.22	94.31	94.08
Κύκλωμα διαιτησίας	90.43				90.43
Σύνολο					**91.05**

Πίνακας 4.16 Ποσοστό κάλυψης ελαττωμάτων για τον πολυεπεξεργαστή OpenRISC Quad

Μονάδα	Ποσοστό κάλυψης ελαττωμάτων (%)				
	Πυρήνας 1	Πυρήνας 2	Πυρήνας 3	Πυρήνας 4	Σύνολο
Επεξεργαστές	90.79	90.83	90.81	90.78	90.80
Ελεγκτής κρυφής μνήμης εντολών	91.57	91.49	91.53	91.64	91.55
Ελεγκτής κρυφής μνήμης δεδομένων	90.20	90.19	90.23	90.22	90.21
Μνήμη προσωρινής αποθήκευσης	94.26	93.84	94.29	94.32	94.17
Κύκλωμα διαιτησίας	90.16 (σειρά 0), 90.40 (σειρά 1)				90.28
Σύνολο					**91.04**

Πίνακας 4.17 Ποσοστό κάλυψης ελαττωμάτων για τον πολυεπεξεργαστή OpenRISC Quad 2

Μονάδα	Ποσοστό κάλυψης ελαττωμάτων (%)								
	Πυρ. 1	Πυρ. 2	Πυρ. 3	Πυρ. 4	Πυρ. 5	Πυρ. 6	Πυρ. 7	Πυρ. 8	Σύνολο
Επεξεργαστές	90.85	90.87	90.79	90.79	90.80	90.82	90.80	90.77	90.81
Ελεγ. κρυφής μνήμης εντολ.	91.45	91.57	91.62	91.54	91.55	91.61	91.59	91.48	91.55
Ελεγ. κρυφής μνήμης δεδομ.	90.38	90.31	90.28	90.35	90.28	90.31	90.24	90.21	90.29
Μνήμη προσ. αποθήκευσης	94.29	94.27	94.19	94.54	94.17	94.26	94.29	94.27	94.28
Κύκλωμα διαιτησίας	87.01 (σειρά 0), 87.57 (σειρά 1)								87.29
Σύνολο									**90.92**

Πίνακας 4.18 Ποσοστό κάλυψης ελαττωμάτων για τον πολυεπεξεργαστή OpenRISC Octo 2

Μονάδα	Ποσοστό κάλυψης ελαττωμάτων (%)								
	Πυρ. 1	Πυρ. 2	Πυρ. 3	Πυρ. 4	Πυρ. 5	Πυρ. 6	Πυρ. 7	Πυρ. 8	Σύνολο
Επεξεργαστές	90.83	90.75	90.83	90.81	90.76	90.79	90.78	90.82	90.79
Ελεγ. κρυφής μνήμης εντολ.	91.61	91.57	91.55	91.53	91.65	91.57	91.62	91.59	91.58
Ελεγ. κρυφής μνήμης δεδομ.	90.26	90.25	90.18	90.19	90.17	90.24	90.21	90.27	90.22
Μνήμη προσ. αποθήκευσης	94.31	94.14	94.19	94.28	94.21	94.01	94.29	94.17	94.2
Κύκλωμα διαιτησίας	89.15 (σειρά 0), 88.83 (σειρά 1), 88.96 (σειρά 2), 89.38 (σειρά 3)								89.08
Σύνολο									**90.75**

Πίνακας 4.19 Ποσοστό κάλυψης ελαττωμάτων για τον πολυεπεξεργαστή OpenRISC Octo 4

Ο Πίνακας 4.20 και ο Πίνακας 4.21 συγκρίνουν την προτεινόμενη μεθοδολογία με τις εναλλακτικές προσεγγίσεις αυτοδοκιμής με λογισμικό ως προς το μέγεθος του κώδικα δοκιμής και των αποκρίσεων δοκιμής. Η υπεροχή της προτεινόμενης προσέγγισης συγκριτικά με την προσέγγιση ΠΑ-ΠΕ ως προς το συνολικό μέγεθος του κώδικα δοκιμής είναι εμφανής. Αναφορικά με τις δύο εναλλακτικές προσεγγίσεις που χρησιμοποιούν ενιαίο αντίγραφο του προγράμματος δοκιμής παρατηρούμε τα ακόλουθα: (α) η προτεινόμενη προσέγγιση αυξάνει ελάχιστα το μέγεθος του προγράμματος δοκιμής ως προς την προσέγγιση ΕΑ-ΑΕ, λόγω του πρόσθετου κώδικα που εισήχθη στην Φάση Β της μεθοδολογίας στην αρχή κάθε ρουτίνας αυτοδοκιμής, (β) εμφανίζει περισσότερες αποκρίσεις σε σύγκριση με την προσέγγιση ΕΑ-ΠΕ (παρουσιάζει μικρότερο μέγεθος κώδικα δοκιμής, αλλά παράγει περισσότερες αποκρίσεις ώστε να υποστηρίξει την διάγνωση στο επίπεδο των πυρήνων). Εδώ, αξίζει να σημειωθεί ότι ο κατά πολύ μικρότερος αριθμός αποκρίσεων της προσέγγισης ΕΑ-ΠΕ οφείλεται στην σύμπτυξη των αποκρίσεων της δοκιμής. Το κύριο μειονέκτημα της εναλλακτικής αυτής προσέγγισης είναι ότι δεν υποστηρίζει τη διάγνωση στο επίπεδο των πυρήνων, μειώνοντας την εσοδεία παραγωγής των πολυεπεξεργαστών, κυρίως όταν το σύστημα περιλαμβάνει πολλούς πυρήνες.

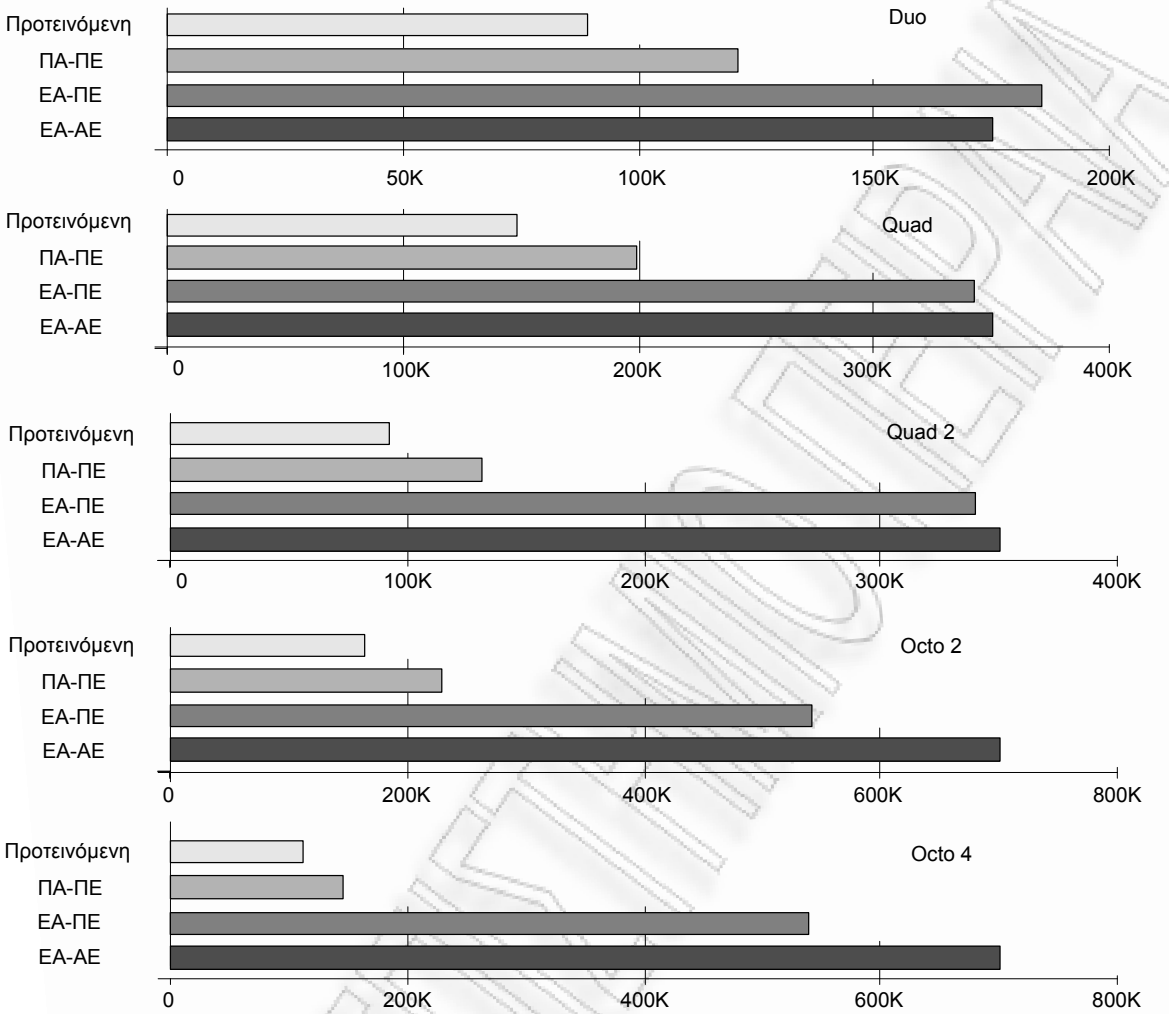
Προσεγγίσεις	Duo	Quad	Quad 2	Qcto 2	Octo 4
ΕΑ-ΑΕ	3271	3271	3271	3271	3271
ΕΑ-ΠΕ	3543	3543	3543	3543	3543
ΠΑ-ΠΕ	6566	13084	13084	26128	26128
Προτεινόμενη	3327	3406	3416	3539	3635

Πίνακας 4.20 Σύγκριση μεγέθους κώδικα δοκιμής (λέξεις)

Προσεγγίσεις	Duo	Quad	Quad 2	Qcto 2	Octo 4
ΕΑ-ΑΕ	3582	7164	7164	14328	14328
ΕΑ-ΠΕ	1791	1791	1791	1791	1791
ΠΑ-ΠΕ	3582	7164	7164	14328	14328
Προτεινόμενη	3582	7164	7164	14328	14328

Πίνακας 4.21 Σύγκριση μεγέθους αποκρίσεων δοκιμής (λέξεις)

Η Εικόνα 4.20 συγκρίνει το χρόνο εκτέλεσης του προγράμματος δοκιμής (σε κύκλους ρολογιού) των διαφορετικών προσεγγίσεων. Η υπεροχή της προτεινόμενης προσέγγισης έναντι της προσέγγισης ΕΑ-ΑΕ είναι εμφανής. Ωστόσο, η αποτελεσματικότητα της προτεινόμενης μεθοδολογίας αποδεικνύεται από την σύγκριση με τις εναλλακτικές προσεγγίσεις παράλληλης εκτέλεσης. Μια περαιτέρω ανάλυση των αποτελεσμάτων έδειξε ότι στην περίπτωση της προσέγγισης ΕΑ-ΠΕ το 70% της μείωσης του χρόνου εκτέλεσης του προγράμματος δοκιμής οφείλεται στην εξάλειψη των ακυρώσεων του ελεγκτή συνοχής κρυφής μνήμης, ενώ το υπόλοιπο 30% οφείλεται στη μείωση των συγκρούσεων από τον ανταγωνισμό του διαύλου επικοινωνίας. Στην προσέγγιση ΠΑ-ΠΕ, η βελτίωση του χρόνου εκτέλεσης οφείλεται μόνο στην μείωση του ανταγωνισμού του διαύλου επικοινωνίας, καθώς δεν πραγματοποιούνται ακυρώσεις της κρυφής μνήμης. Το κύριο μειονέκτημα της προσέγγισης ΠΑ-ΠΕ είναι ο υπερβολικός χρόνος φόρτωσης των πολλαπλών αντιγράφων του κώδικα δοκιμής στην κοινόχρηστη κρυφή μνήμη. Να σημειωθεί ότι για να είναι δίκαιη η σύγκριση με την εναλλακτική προσέγγιση ΠΑ-ΠΕ, τα πολλαπλά αντίγραφα του κώδικα δοκιμής δεν αποθηκεύτηκαν σε μία μόνο σειρά της κοινόχρηστης κρυφής μνήμης, αλλά φορτώθηκαν σε όλες τις σειρές ώστε να αυξηθεί η αξιοποίηση του διαθέσιμου εύρους ζώνης του σταυρωτού μεταγωγέα.



Εικόνα 4.20 Σύγκριση του χρόνου εκτέλεσης του προγράμματος δοκιμής

Για την περαιτέρω επαλήθευση της μεθοδολογίας πραγματοποιήθηκαν πειράματα με *προγράμματα μέτρησης απόδοσης επεξεργαστών* (*benchmarks*) (bubblesort, the eight queens, και άλλα) από την σουίτα Stanford Small Benchmark Suite (SSB) [25]. Ο Πίνακας 4.22 παρέχει μια σύντομη περιγραφή των προγραμμάτων μέτρησης απόδοσης, ενώ ο Πίνακας 4.23 παρουσιάζει τα στατιστικά εκτέλεσης των προγραμμάτων ως προς τον χρόνο εκτέλεσης, τις αιτήσεις προσκόμισης εντολών και τις αιτήσεις ανάγνωσης και εγγραφής δεδομένων στην κοινόχρηστη κρυφή μνήμη.

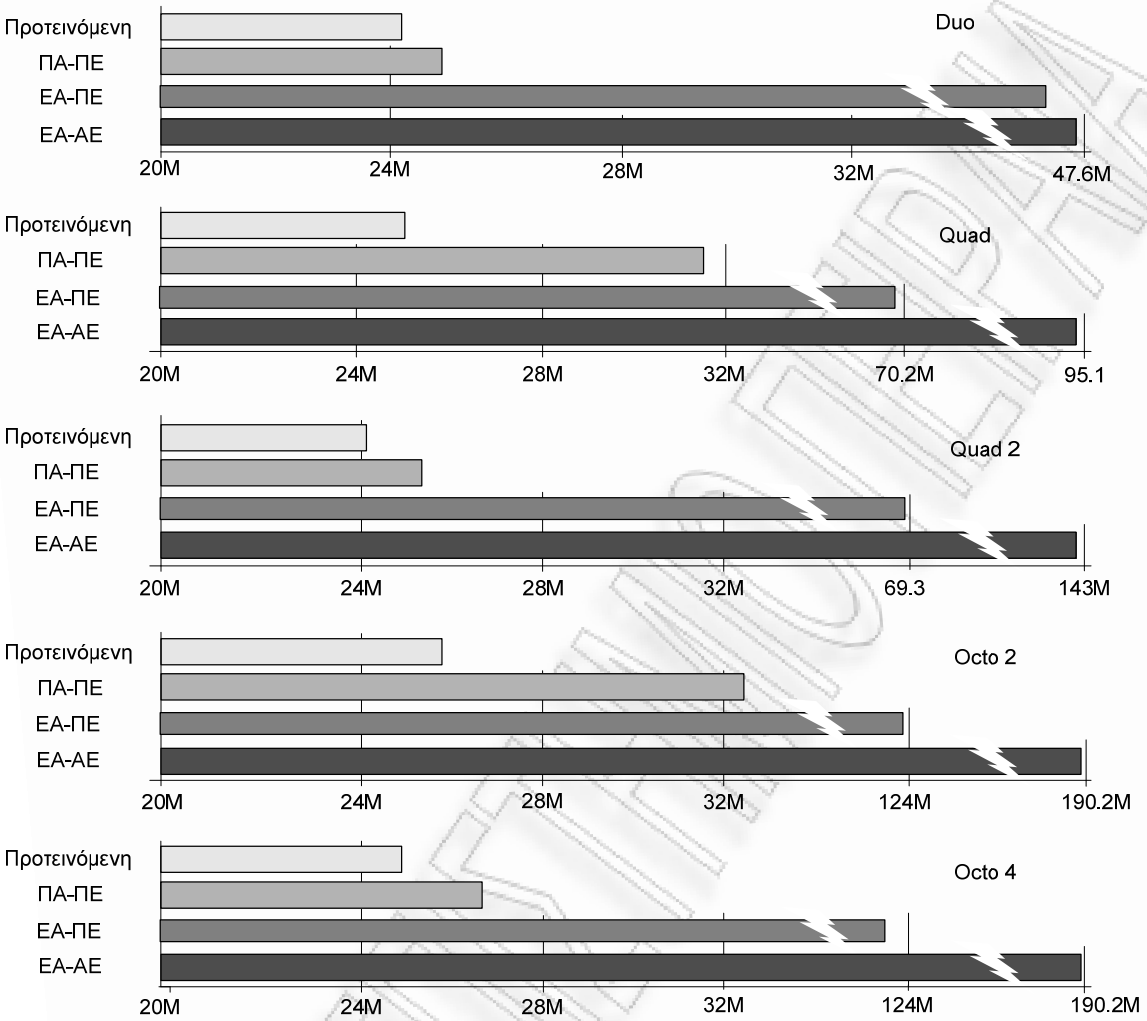
Μετροπρογράμματα	Περιγραφή
Puzzle	Compute-bound program.
Towers	Program solving the Towers of Hanoi problem.
Intmm	Program multiplying two integer matrices.
Queens	Program solving the Eight Queens problem 50 times.
Bubble	Program sorting an array using Bubblesort.

Πίνακας 4.22 Περιγραφή των προγραμμάτων μέτρησης απόδοσης

Μετροπρογράμματα	Χρόνος εκτέλεσης (Κύκλοι)	Αιτήσεις ανάγνωσης εντολών (L2)	Αιτήσεις ανάγνωσης δεδομένων (L2)	Αιτήσεις εγγραφής δεδομένων (L2)
Puzzle	11227968	201	131321	70233
Towers	5561262	66	223984	442542
Intmm	1905769	43	77900	75221
Queens	2376631	51	51251	162751
Bubble	2700740	33	80360	128175

Πίνακας 4.23 Στατιστικά δεδομένα των προγραμμάτων μέτρησης απόδοσης

Η Εικόνα 4.21 παρουσιάζει τον χρόνο εκτέλεσης των προγραμμάτων μέτρησης απόδοσης των εναλλακτικών προσεγγίσεων αυτοδοκιμής με λογισμικό για όλους τους συμμετρικούς πολυεπεξεργαστές. Η προτεινόμενη προσέγγιση επιτυγχάνει περίπου δύο, τέσσερις και οκτώ φορές μικρότερο χρόνο εκτέλεσης για τους διπύρηνους, τετραπύρηνους και οκταπύρηνους συμμετρικούς επεξεργαστές, αντίστοιχα σε σύγκριση με την εναλλακτική προσέγγιση ΕΑ-ΑΕ. Παράλληλα επιτυγχάνει πολύ καλύτερα αποτελέσματα συγκρινόμενη με την εναλλακτική προσέγγιση ΕΑ-ΠΕ. Ο αυξανόμενος χρόνος εκτέλεσης της προσέγγισης ΕΑ-ΠΕ οφείλεται στο μεγάλο αριθμό ακυρώσεων του ελεγκτή συνοχής κρυφής μνήμης, προκαλώντας την συχνή προσκόμιση μπλοκ δεδομένων από την κοινόχρηστη στην τοπική κρυφή μνήμη δεδομένων. Όταν το διαθέσιμο εύρος ζώνης ανά πυρήνα είναι υψηλό, όπως στις περιπτώσεις των πολυεπεξεργαστών Duo, Quad 2 και Octo 4, ο χρόνος εκτέλεσης της προσέγγισης ΠΑ-ΠΕ τείνει στο χρόνο εκτέλεσης της προτεινόμενης μεθοδολογίας. Ωστόσο, όταν το διαθέσιμο εύρος ζώνης ανά πυρήνα είναι χαμηλό, ή με άλλα λόγια η αναλογία των ταυτόχρονων διασυνδέσεων που υποστηρίζονται από το μέσο διασύνδεσης προς των αριθμό των πυρήνων μειώνεται, όπως στους πολυεπεξεργαστές Quad και Octo 2, ο χρόνος εκτέλεσης της προσέγγισης ΠΑ-ΠΕ αυξάνεται καθώς δεν αντιμετωπίζει το πρόβλημα του ανταγωνισμού στον δίαυλο επικοινωνίας. Στις παραπάνω περιπτώσεις, η προτεινόμενη προσέγγιση αντιμετωπίζει αποτελεσματικά και αυτό το πρόβλημα.



Εικόνα 4.21 Σύγκριση του χρόνου εκτέλεσης των προγραμμάτων μέτρησης απόδοσης

4.8.2 Ο επεξεργαστής OpenSPARC T1

Ο επεξεργαστής OpenSPARC T1 [137] αποτελεί την ανοικτή έκδοση του πηγαίου κώδικα του επεξεργαστή UltraSPARC T1 της εταιρίας Sun Microsystems [136] που παρουσιάστηκε στην Ενότητα 4.4.2. Στον επεξεργαστή αυτό ο μηχανισμός συνοχής κρυφής μνήμης διατηρεί τη συνέπεια μεταξύ του περιεχομένου των τοπικών κρυφών μνημών και της κοινόχρηστης κρυφής μνήμης. Η κοινόχρηστη κρυφή μνήμη τηρεί τη συνοχή των δεδομένων διατηρώντας ένα αντίγραφο όλων των ετικετών των τοπικών κρυφών μνημών σε μια *δομή καταλόγου* (*directory structure*). Ο Πίνακας 4.24 συνοψίζει τις βασικές παραμέτρους του επεξεργαστή OpenSPARC T1.

Χαρακτηριστικό	Τιμή
Πυρήνες	8
Νήματα ανά πυρήνα	4
Τοπική κρυφή μνήμη εντολών L1 (μέγεθος γραμμής)	16 Kbytes (32 bytes)
Τοπική κρυφή μνήμη δεδομένων L1 (μέγεθος γραμμής)	8 Kbytes (16 bytes)
Πολιτική τοπικής κρυφής μνήμης (L1)	4-ways set associative
Πολιτική εγγραφής (L1)	Write-through
Κοινόχρηστη κρυφή μνήμη	3 Mbytes (64 bytes)
Αριθμός σειρών	4
Πολιτική κοινόχρηστης κρυφής μνήμης (L2)	12-ways set associative
Πολιτική εγγραφής (L2)	Write Back
Τύπος διασύνδεσης	Crossbar Switch
Συνοχή κρυφής μνήμης	Directory based

Πίνακας 4.24 Χαρακτηριστικά του πολυεπεξεργαστή OpenSPARC T1

Για την εφαρμογή της αυτοδοκιμής με λογισμικό στον επεξεργαστή OpenSPARC T1 πραγματοποιήθηκαν οι ακόλουθες υποθέσεις [30]:

- Επιλέχθηκε ένα σύνολο ρουτινών που έχει ως στόχο την δοκιμή των λειτουργικών μονάδων: αριθμητική και λογική μονάδα (Arithmetic and Logic Unit, ALU), ολισθητής shifter, πολλαπλασιαστής (multiplier), διαιρέτης (divider), μονάδα κινητής υποδιαστολής (Floating Point Unit, FPU). Ο Πίνακας 4.25 παρουσιάζει τα βασικά χαρακτηριστικά των ρουτινών αυτοδοκιμής: αριθμός διανυσμάτων δοκιμής και χρόνος εκτέλεσης σε κύκλους ρολογιού. Ο Πίνακας 4.26 περιέχει τις αιτήσεις εγγραφής και ανάγνωσης τόσο των εντολών όσο και των δεδομένων. Δεδομένου ότι ο κύριος στόχος της παρούσας διδακτορικής διατριβής είναι η αποδοτική εκμετάλλευση της παράλληλης εκτέλεσης του προγράμματος αυτοδοκιμής σε συμμετρικούς πολυεπεξεργαστές με στόχο τη μείωση του χρόνου εφαρμογής της δοκιμής, οι ρουτίνες δεν αναπτύχθηκαν με σκοπό την επίτευξη υψηλού ποσοστού κάλυψης ελαττωμάτων. Αντίθετα, χρησιμοποιήθηκαν είτε διαθέσιμες ρουτίνες αυτοδοκιμής από τις προηγούμενες ερευνητικές εργασίες για άλλους πυρήνες επεξεργαστών [95], [97] οι οποίες προσαρμόστηκαν στην αρχιτεκτονική του συνόλου εντολών του επεξεργαστή SPARC V9 (περιπτώσεις των ρουτινών ALU, shifter, multiplier), είτε έτοιμες ρουτίνες επαλήθευσης της σχεδίασης από την εταιρία Sun Microsystems [136] (περιπτώσεις των ρουτινών Divider, FPU, Various).

- Οι παραπάνω ρουτίνες αυτοδοκιμής εκτελούνται από κάθε επεξεργαστή με σκοπό να

δοκιμαστούν οι τοπικές λειτουργικές μονάδες. Εδώ, πρέπει να σημειωθεί ότι αν και οι λειτουργικές μονάδες κάθε πυρήνα μοιράζονται μεταξύ των τεσσάρων νημάτων, τα προγράμματα δοκιμής εκτελούνται μόνο από ένα νήμα. Ο χρόνος εκτέλεσης όπως ο Πίνακας 4.25 παρουσιάζει, αφορά την εκτέλεση των ρουτινών αυτοδοκιμής από ένα νήμα.

- Οι ρουτίνες αυτοδοκιμής φορτώνονται στην κοινόχρηστη κρυφή μνήμη δευτέρου επιπέδου του επεξεργαστή OpenSPARC T1. Οι κύκλοι εκτέλεσης υπολογίστηκαν εκτελώντας τις ρουτίνες αυτοδοκιμής από τη κοινόχρηστη κρυφή μνήμη.

Ρουτίνα	Διανύσματα δοκιμής	Χρόνος εκτέλεσης (Κύκλοι)
ALU	124	2182
Shifter	62	1268
Multiplier	256	7577
Divider	32	5784
FPU	32	1980
Various	20	518

Πίνακας 4.25 Αριθμός διανυσμάτων δοκιμής και χρόνος εκτέλεσης των ρουτινών αυτοδοκιμής

Ρουτίνα	Αιτήσεις Κρυφής Μνήμης Δευτέρου Επιπέδου		
	Εντολές	Αναγνώσεις Δεδομένων	Εγγραφές Δεδομένων
ALU	18	0	124
Shifter	32	0	62
Multiplier	8	0	256
Divider	8	4	129
FPU	10	4	64
Various	12	1	20

Πίνακας 4.26 Στατιστικά αιτήσεων ρουτινών αυτοδοκιμής για τον επεξεργαστή OpenSPARC T1

Για να επιτραπεί η εκτέλεση του προγράμματος αυτοδοκιμής από τη κοινόχρηστη κρυφή μνήμη στο περιβάλλον προσομοίωσης του επεξεργαστή OpenSPARC T1 πραγματοποιήθηκαν οι ακόλουθες ενέργειες:

1. Απενεργοποίηση των τοπικών κρυφών μνημών πρώτου επιπέδου χρησιμοποιώντας τον καταχωρητή ελέγχου ASI_LSU_CONTROL_REG.

2. Αποθήκευση του προγράμματος αυτοδοκιμής στην κύρια μνήμη του συστήματος.

3. Εκτέλεση του προγράμματος και υπολογισμός του χρόνου εκτέλεσης, έστω *t1*. Κατά την ολοκλήρωση της πρώτης εκτέλεσης, ο κώδικας και τα διανύσματα δοκιμής αποθηκεύονται στην κοινόχρηστη κρυφή μνήμη, εξασφαλίζοντας ότι κατά την διάρκεια της δεύτερης εκτέλεσης δεν θα πραγματοποιηθεί καμία αστοχία από την κοινόχρηστη κρυφή μνήμη. Στην περίπτωση που προκληθεί αστοχία, τότε το πρόγραμμα δοκιμής θα πρέπει να τροποποιηθεί κατάλληλα ώστε να αποφθεχθεί το παραπάνω φαινόμενο.

4. Ενεργοποίηση των τοπικών κρυφών μνημών και επανεκτέλεση του κώδικα δοκιμής. Υπολογισμός του συνολικού χρόνου εκτέλεσης (τόσο για την πρώτη, όσο και την δεύτερη εκτέλεση) έστω *t2*. Έλεγχος για πρόσβαση προς την κύρια μνήμη του συστήματος κατά την δεύτερη εκτέλεση.

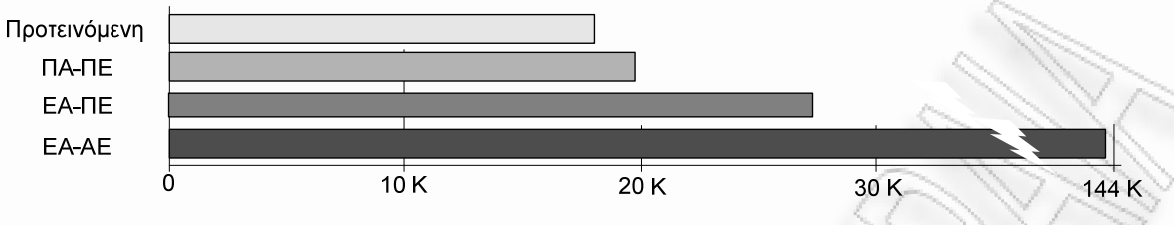
5. Υπολογισμός του χρόνου εκτέλεσης από την κοινόχρηστη κρυφή μνήμη: *t2-t1*.

Ο Πίνακας 4.27 συγκρίνει τα αποτελέσματα της προτεινόμενης μεθοδολογίας με τις εναλλακτικές προσεγγίσεις αυτοδοκιμής με λογισμικό ως προς το μέγεθος του προγράμματος δοκιμής (κώδικας και διανύσματα δοκιμής), και την ικανότητα διάγνωσης στο επίπεδο των πυρήνων. Η προτεινόμενη προσέγγιση αυξάνει ελάχιστα το μέγεθος του προγράμματος αυτοδοκιμής σε σύγκριση με την προσέγγιση ΕΑ-ΑΕ, λόγω του επιπλέον κώδικα που προστίθεται κατά την Φάση Β της μεθοδολογίας.

Προσέγγιση αυτοδοκιμής	Μέγεθος κώδικα δοκιμής (Λέξεις)	Ικανότητα διάγνωσης των πυρήνων
ΕΑ-ΑΕ	749	Yes
ΕΑ-ΠΕ	981	No
ΠΑ-ΠΕ	4869	Yes
Προτεινόμενη	773	Yes

Πίνακας 4.27 Σύγκριση εναλλακτικών προσεγγίσεων αυτοδοκιμής με λογισμικό στον επεξεργαστή OpenSPARC T1

Η Εικόνα 4.22 συγκρίνει τον χρόνο εκτέλεσης (σε κύκλους ρολογιού) των εναλλακτικών προσεγγίσεων αυτοδοκιμής με λογισμικό, με τον χρόνο εκτέλεσης της προτεινόμενης προσέγγισης για τον πολυεπεξεργαστή OpenSPATC T1. Τα πειραματικά αποτελέσματα απέδειξαν ότι η προτεινόμενη προσέγγιση αυξάνει τον παραλληλισμό κατά την εκτέλεση του προγράμματος δοκιμής, επιτυγχάνοντας σημαντική μείωση στο χρόνο εφαρμογής της δοκιμής. Αξίζει να σημειωθεί ότι η προσέγγιση ΠΑ-ΠΕ παρόλο που επιτυγχάνει χρόνο εκτέλεσης κοντά στην προτεινόμενη προσέγγιση, απαιτεί μεγάλο χρόνο για την φόρτωση των οκτώ διαφορετικών αντιγράφων του κώδικα αυτοδοκιμής στην κοινόχρηστη κρυφή μνήμη.



Εικόνα 4.22 Σύγκριση χρόνου εκτέλεσης για τον επεξεργαστή OpenSPARC T1

4.9 Ανακεφαλαίωση

Στο κεφάλαιο 4 παρουσιάστηκαν,

- οι τεχνικές παραλληλίας στο επίπεδο των εντολών στους μονοπύρηνους επεξεργαστές,
- οι αρχιτεκτονικές των πολυεπεξεργαστών και κυρίως των συμμετρικών πολυεπεξεργαστών,
- οι δύο κύριες αρχιτεκτονικές διασύνδεσης των συμμετρικών πολυεπεξεργαστών,
- περιπτώσεις ευρέως διαδεδομένων συμμετρικών πολυεπεξεργαστών,
- η βιβλιογραφία για την δοκιμή των συμμετρικών πολυεπεξεργαστών καθώς και οι αδυναμίες που παρουσιάζουν οι υπάρχουσες τεχνικές,
- η προτεινόμενη μεθοδολογία αυτοδοκιμής με λογισμικό που στοχεύει στη μείωση του συνολικού χρόνου εφαρμογής της δοκιμής για συμμετρικούς πολυεπεξεργαστές.

Κεφάλαιο 5

Συμπεράσματα και Μελλοντική Έρευνα

The most beautiful experience we can have is the mysterious. It is the fundamental emotion that stands at the cradle of true art and true science...

Albert Einstein

Σε αυτό το καταληκτικό κεφάλαιο, συνοψίζονται τα πρωτότυπα στοιχεία της παρούσας διατριβής και καταγράφονται ερευνητικές κατευθύνσεις για μελλοντική έρευνα. Η παρούσα διδακτορική διατριβή ασχολήθηκε με το πρόβλημα της δοκιμής των ενσωματωμένων περιφερειακών επικοινωνίας σε ένα σύστημα σε ένα ολοκληρωμένο με στόχο την επίτευξη υψηλού ποσοστού κάλυψης ελαττωμάτων προσκόλλησης των περιφερειακών αυτών, καθώς και με την δοκιμή των συμμετρικών πολυεπεξεργαστών υψηλής απόδοσης με στόχο την μείωση του συνολικού χρόνου εφαρμογής της αυτοδοκιμής με λογισμικό.

Στο πρώτο μέρος της διατριβής, παρουσιάστηκε η μελέτη δοκιμαστικότητας των περιφερειακών μονάδων επικοινωνίας σε συστήματα σε ένα ολοκληρωμένο, η οποία και οδήγησε στην ανάπτυξη των τεσσάρων βημάτων της μεθοδολογίας καθένα από τα οποία δοκιμάζει μια μονάδα του περιφερειακού επικοινωνίας. Τα τέσσερα αυτά βήματα εκφράζουν υπό ποίες συνθήκες τα διανύσματα δοκιμής μπορούν να ανιχνεύσουν ελαττώματα προσκόλλησης στο περιφερειακό επικοινωνίας. Στο δεύτερο μέρος της διδακτορικής διατριβής, παρουσιάστηκαν τα κριτήρια για την ανάπτυξη λογισμικού δοκιμής ώστε να είναι κατάλληλο για τη δοκιμή κατά τη κατασκευή συμμετρικών πολυεπεξεργαστών. Οι τρεις φάσεις της

προτεινόμενης μεθοδολογίας προσπαθούν να εκμεταλλευτούν και να μεγιστοποιήσουν τον έμφυτο παραλληλισμό των συμμετρικών πολυεπεξεργαστών ώστε να μειωθεί ο συνολικός χρόνος εφαρμογής της δοκιμής.

Τα συμπεράσματα που προκύπτουν από το πρώτο μέρος της διατριβής είναι πως η ενσωμάτωση των περιφερειακών μονάδων επικοινωνίας σε ένα σύστημα σε ένα ολοκληρωμένο επηρεάζει αρνητικά τη δοκιμαστικότητά τους. Η παρατήρηση αυτή προκύπτει λόγω της περιορισμένης ελεγξιμότητας και παρατηρισιμότητας των εσωτερικών καταχωρητών, αλλά και τον ακροδεκτών του περιφερειακού επικοινωνίας από τους ακροδέκτες του συστήματος σε ένα ολοκληρωμένο. Χαρακτηριστικό παράδειγμα αποτελεί η δοκιμή της λογικής του παραλήπτη δεδομένου ότι μπορεί να επιτευχθεί εάν και μόνο εάν μπορέσουμε να οδηγήσουμε τη γραμμή λήψης του περιφερειακού. Ομοίως, η δοκιμή της λογικής του αποστολέα μπορεί να επιτευχθεί εάν και μόνο εάν μπορέσουμε να παρατηρήσουμε την γραμμή αποστολής του περιφερειακού.

Ένα επιπλέον συμπέρασμα είναι ότι σε όλες τις κλασικές προσεγγίσεις δοκιμής για ένα σύστημα σε ένα ολοκληρωμένο, είτε οι πυρήνες, είτε ο ίδιος ο επεξεργαστής έπρεπε να ενσωματώσουν χαρακτηριστικά δοκιμής (scan chains, DFT, IEEE 1500, instruction-level DFT) τα οποία διευκόλυναν την διαδικασία δοκιμής. Ενσωματώνοντας όμως τα εν λόγω χαρακτηριστικά κληρονομούμε αναπόφευκτα και όλα τα μειονεκτήματα αυτών των προσεγγίσεων, όπως η αύξηση της κατανάλωσης ισχύος, η επιβάρυνση της επιφάνειας του ολοκληρωμένου, η μείωση της εσοδείας, η υπερδοκιμή και η υποβάθμιση της απόδοσης του συστήματος σε ένα ολοκληρωμένο. Επιπλέον, διάφορες τροποποιήσεις πρέπει να εφαρμοστούν στους πυρήνες προκειμένου αυτοί να καταστούν "έτοιμοι" για αυτού του είδους τις προσεγγίσεις ώστε να αποφευχθούν οι αντιστάσεις των τυχαίων διανυσμάτων και οι συγκρούσεις στον δίαυλο επικοινωνίας.

Ένα άλλο συμπέρασμα που προκύπτει είναι πως η χαμηλή ταχύτητα μετάδοσης - λήψης έναντι της συχνότητας του ενσωματωμένου επεξεργαστή αναγκάζει τις ρουτίνες αυτοδοκιμής να περιμένουν κάποια χρονικά διαστήματα πριν από την επόμενη ακολουθία δοκιμής. Επομένως, ο χρόνος προσομοίωσης ελαττωμάτων και οι πραγματικός χρόνος εφαρμογής της δοκιμής αυξάνονται ανάλογα. Η ανάπτυξη ρουτινών αυτοδοκιμής για τις περιφερειακές μονάδες επικοινωνίας επιβάλει την προσεκτική εκτέλεση αυτών, ώστε να αποφευχθεί ο υπερβολικός χρόνος εφαρμογής της δοκιμής.

Το κυριότερο συμπέρασμα που προκύπτει από το δεύτερο μέρος της διατριβής είναι πως το λογισμικό αυτοδοκιμής θα πρέπει να διαθέτει και επιπλέον χαρακτηριστικά εκτός από την επίτευξη υψηλού ποσοστού κάλυψης ελαττωμάτων, για να εφαρμοστεί επιτυχώς για την αυτοδοκιμή σε συμμετρικούς πολυεπεξεργαστές. Τα χαρακτηριστικά αυτά αφορούν τον ενιαίο κώδικα δοκιμής για όλους του επεξεργαστές, την παράλληλη εκτέλεση του κώδικα δοκιμής, το μικρό χρόνο εκτέλεσης και τη δυνατότητα διάγνωσης στο επίπεδο των πυρήνων. Από τις τρεις εναλλακτικές προσεγγίσεις αυτοδοκιμής με λογισμικό για συμμετρικούς πολυεπεξεργαστές, ως

προς το μέγεθος του κώδικα δοκιμής, οι προσεγγίσεις, ΕΑ-ΔΕ και ΕΑ-ΠΕ, παρουσιάζουν αντίστοιχα αποτελέσματα με την περίπτωση του μονοπύρηνου επεξεργαστή. Η εναλλακτική ΕΑ-ΠΕ έχει ελαφρώς μεγαλύτερο μέγεθος προγράμματος λόγω του πρόσθετου κώδικα για τη σύμπτυξη των αποκρίσεων της δοκιμής. Εντούτοις, το μέγεθος του κώδικα δοκιμής της προσέγγισης ΠΑ-ΠΕ είναι *n* φορές μεγαλύτερο από το μέγεθος του κώδικα δοκιμής του μονοπύρηνου επεξεργαστή, όπου *n* ο αριθμός των επεξεργαστών. Η μεγάλη αυτή απαίτηση ως προς την αποθήκευση του κώδικα δοκιμής κλιμακώνεται με τον αριθμό πυρήνων και αποτελεί το σημαντικότερο μειονέκτημα αυτής της προσέγγισης. Όσον αφορά τις αποκρίσεις της δοκιμής, ο αριθμός των αποκρίσεων είναι *n* φορές μεγαλύτερος σε σχέση με τον μονοπύρηνο επκεργαστή και είναι το τίμημα που οι προσεγγίσεις ΕΑ-ΔΕ και ΠΑ-ΠΕ επιβάλουν ώστε να διατηρήσουν την ικανότητα διάγνωσης στο επίπεδο των πυρήνων. Όλες οι εναλλακτικές προσεγγίσεις αυτοδοκιμής, εκτός της ΕΑ-ΑΕ, εκμεταλλεύονται τον έμφυτο παραλληλισμό των συμμετρικών επεξεργαστών.

Άλλο ένα συμπέρασμα που προκύπτει είναι ότι ο αναμενόμενος χρόνος εφαρμογής της δοκιμής των παράλληλων προσεγγίσεων θα περίμενε κάποιος να είναι αρκετά κοντά στο χρόνο εκτέλεσης του μονοπύρηνου επεξεργαστή. Τα πειράματα έδειξαν πως η προσέγγιση ΠΑ-ΠΕ επιτυγχάνει τον πιο σύντομο χρόνο δοκιμής ο οποίος, εντούτοις, είναι σχεδόν διπλάσιος από τον χρόνος δοκιμής του μονοπύρηνου επεξεργαστή. Αυτό αποδεικνύει το κατά πόσο το ανταγωνισμός του δίαυλου επικοινωνίας μπορεί να επιβραδύνει το χρόνο εκτέλεσης δοκιμής λόγω των ταυτόχρονων προσβάσεων στους κοινούς πόρους της κρυφής μνήμης. Στην περίπτωση της προσέγγισης ΕΑ-ΠΕ, οι πολλές ακυρώσεις συνοχής της κρυφής μνήμης επιβραδύνουν επιπλέον το χρόνο εκτέλεσης της δοκιμής, που είναι σχεδόν ίσος με το χρόνο εκτέλεσης δοκιμής της ακολουθιακής προσέγγισης. Το εν' λόγω φαινόμενο παρουσιάζεται κυρίως λόγω των ταυτόχρονων προσβάσεων των κοινών μεταβλητών που χρησιμοποιούνται από τους διαφορετικούς πυρήνες για να συμπιέσουν τις αποκρίσεις τους.

Το κυριότερο συμπέρασμα είναι ότι τα ζητήματα για την αποτελεσματική εφαρμογή μιας προσέγγισης αυτοδοκιμής με λογισμικό σε μια αρχιτεκτονική συμμετρικών πολυεπεξεργαστών είναι (α) η μείωση των απαιτήσεων αποθήκευσης στην κοινόχρηστη κρυφή μνήμη με τη χρήση ενός ενιαίου αντιγράφου προγράμματος, (β) η μείωση του χρόνου εκτέλεσης δοκιμής αυξάνοντας το παραλληλισμό κατά τη διάρκεια της εκτέλεσης των προγραμμάτων δοκιμής, (γ) η ελαχιστοποίηση του ανταγωνισμού του μέσου επικοινωνίας και των ακυρώσεων δεδομένων λόγω συνοχής της κρυφής μνήμης, τα οποία και επιβραδύνουν την εκτέλεση του κώδικα δοκιμής και (δ) η παραγωγή αποκρίσεων σε διαφορετικές θέσεις της κοινόχρηστης κρυφής μνήμης για κάθε επεξεργαστή, ώστε να επιτραπεί η διάγνωση στο επίπεδο των πυρήνων. Η προτεινόμενη προσέγγιση προσπαθεί να επιλύσει τα ανωτέρω προβλήματα.

Τα πειραματικά αποτελέσματα έδειξαν η προτεινόμενη προσέγγιση συγκριτικά με την εναλλακτική ΠΑ-ΠΕ χρησιμοποιεί πολύ μικρότερο μέγεθος κώδικα. Αναφορικά με τις δύο

εναλλακτικές προσεγγίσεις που χρησιμοποιούν ενιαίο αντίγραφο του προγράμματος δοκιμής παρατηρούμε τα εξής. Η προτεινόμενη προσέγγιση αυξάνει ελαφρώς το μέγεθος του προγράμματος δοκιμής ως προς την προσέγγιση ΕΑ-ΑΕ, δεδομένου του επιπλέον κώδικα που εισάγαμε κατά την Φάση Β την μεθοδολογίας μας στην αρχή κάθε ρουτίνας αυτοδοκιμής, ενώ εμφανίζει αρκετά περισσότερες αποκρίσεις εν' συγκρίσει με την προσέγγιση ΕΑ-ΠΕ (εμφανίζει μικρότερο μέγεθος κώδικα δοκιμής, αλλά παράγει περισσότερες αποκρίσεις ούτως ώστε να υποστηρίξει την διάγνωση στο επίπεδο των πυρήνων). Εδώ πρέπει να σημειώσουμε ότι ο πολύ μικρότερος αριθμός αποκρίσεων της προσέγγισης ΕΑ-ΠΕ οφείλεται στο γεγονός της σύμπτυξης των αποκρίσεων της δοκιμής. Το κύριο μειονέκτημα όμως αυτής την εναλλακτικής προσέγγισης είναι η μη υποστήριξη διάγνωσης στο επίπεδο των πυρήνων μειώνοντας με αυτόν τον τρόπο την εσοδεία παραγωγής των πολυεπεξεργαστών, ιδίως όταν το σύστημα περιλαμβάνει πολλούς πυρήνες επεξεργαστών.

Ως προς τον χρόνο εκτέλεσης της αυτοδοκιμής συμπεραίνουμε ότι η προτεινόμενη προσέγγιση επιτυγχάνει περίπου δύο, τέσσερις και οκτώ φορές μικρότερο χρόνο εκτέλεσης για τους διπύρηνους, τετραπύρηνους και οκταπύρηνους συμμετρικούς επεξεργαστές αντίστοιχα, εν' συγκρίσει με την εναλλακτική προσέγγιση ΕΑ-ΑΕ. Επιτυγχάνει επίσης πολύ καλύτερα αποτελέσματα εν' συγκρίσει με την εναλλακτική προσέγγιση ΕΑ-ΠΕ. Ο μεγάλος χρόνος εκτέλεσης της προσέγγισης ΕΑ-ΠΕ οφείλεται στο μεγάλο αριθμό ακυρώσεων του ελεγκτή συνοχής κρυφής μνήμης, προκαλώντας την συχνή προσκόμιση μπλοκ δεδομένων από την κοινόχρηστη κρυφή μνήμη στην τοπική κρυφή μνήμη δεδομένων. Σημαντικά είναι επίσης να αναφερθεί ότι όταν το διαθέσιμο εύρος ζώνης ανά πυρήνα είναι υψηλό, όπως στις περιπτώσεις των πολυεπεξεργαστών Duo, Quad 2 και Octo 4, η προσέγγιση ΠΑ-ΠΕ επιτυγχάνει χρόνο εκτέλεσης κοντά στον χρόνο εκτέλεσης της προτεινόμενης προσέγγισης. Εντούτοις, όταν το διαθέσιμο εύρος ζώνης ανά πυρήνα είναι χαμηλό, ή με άλλα λόγια η αναλογία των υποστηριζόμενων ταυτόχρονων διασυνδέσεων από το μέσο διασύνδεσης προς των αριθμό των πυρήνων μειώνεται, όπως στους πολυεπεξεργαστές Quad και Octo 2, ο χρόνος εκτέλεσης της προσέγγισης ΠΑ-ΠΕ αυξάνεται καθώς δεν επιλύει τον ανταγωνισμό του διαύλου επικοινωνίας. Στις εν' λόγω περιπτώσεις η προτεινόμενη προσέγγιση υπερνικάει και αυτό το πρόβλημα.

Όσον αφορά τη μελλοντική έρευνα που προκύπτει από το πρώτο μέρος της διδακτορικής διατριβής, αυτή αφορά την επέκταση της μεθοδολογίας αυτοδοκιμής με λογισμικό ώστε να καλυφτούν περισσότεροι ενσωματωμένοι πυρήνες, όπως ο *προσαρμογέας γραφικών οθόνης* (*Video Graphics Adapter, VGA*) και οι κάρτες ήχου. Επιπλέον θα πρέπει να αναπτυχθούν μεθοδολογίες όπου επιτρέπουν την ανίχνευση των ελαττωμάτων *κατά την λειτουργία* (*on-line testing*) του συστήματος σε ένα ολοκληρωμένο. Άλλη μία ερευνητική κατεύθυνση με ιδιαίτερη σημασία αφορά την επέκταση της μεθοδολογίας ώστε να καλυφθούν πολυπλοκότερα μοντέλα ελαττωμάτων, όπως το μοντέλο καθυστέρησης μετάβασης και το μοντέλο καθυστέρησης μονοπατιού.

Σχετικά με τη μελλοντική έρευνα που προκύπτει από το δεύτερο μέρος της διατριβής, αυτή επικεντρώνεται στην επέκταση της μεθοδολογίας ώστε να καλυφθούν επιπλέον αρχιτεκτονικές πολυεπεξεργαστών. Ιδιαίτερο ενδιαφέρον παρουσιάζουν οι πολυεπεξεργαστές συνδεδεμένοι μέσω ενός δικτύου οι οποίοι εντάσσονται στην γενική κατηγορία της *ανομοιόμορφης προσπέλασης μνήμης* (*Non Uniform Memory Access, NUMA*) ή της *ανομοιόμορφης προσπέλασης μνήμης με συνοχή κρυφής μνήμης* (*Cache Coherent Non Uniform Memory Access, CC-NUMA*). Στην πρώτη κατηγορία ο χρόνος προσπέλασης μνήμης ποικίλει ανάλογα με την διεύθυνση της μνήμης, ενώ στην δεύτερη κατηγορία έχουμε ένα πολυεπεξεργαστή NUMA με μηχανισμό που εξασφαλίζει την συνοχή της κρυφής μνήμης. Η μετάβαση σε αυτού του είδους τις αρχιτεκτονικές προκύπτει από την πολυπλοκότητα και το ιδιαίτερα μεγάλο κόστος όταν το μέσο διασύνδεσης υλοποιείται με την χρήση σταυρωτού μεταγωγέα. Επιπροσθέτως οι σχεδιάσεις κοινού διαύλου είναι ελκυστικές άλλα έχουν περιορισμούς επειδή τα τρία επιθυμητά χαρακτηριστικά του διαύλου είναι ασύμβατα: υψηλό εύρος ζώνης, χαμηλός λανθάνων χρόνος και μεγάλο μήκος. Επίσης το διαθέσιμο εύρος ζώνης μιας μοναδικής μονάδας κοινόχρηστης μνήμης επιβάλει πρακτικούς περιορισμούς στον μέγιστο αριθμό επεξεργαστών που μπορούν να συνδεθούν στον κοινό δίαυλο. Αν και τα οι δύο φάσεις (φάση Α, φάση Β) της παρούσας διδακτορικής διατριβής είναι σε θέση να καλύψουν και αυτές τις αρχιτεκτονικής πολυεπεξεργαστών, η φάση Γ απαιτεί την μετατροπή του συστήματος ουράς αναμονής σε ένα σύστημα συμβατό με την εκάστοτε αρχιτεκτονική πολυεπεξεργαστών.

Επίσης, αξίζει να μελετηθεί η ανάπτυξη λογισμικού αυτοδοκιμής σε πολυεπεξεργαστές όπου ο κάθε πυρήνας είναι ένας υπερβαθμωτός μικροεπεξεργαστές. Το λογισμικό αυτοδοκιμής σε αυτές τις αρχιτεκτονικές θα πρέπει να είναι κατάλληλα σχεδιασμένο ώστε να εφαρμόζει διανύσματα δοκιμής σε όλες τις πολλαπλές μονάδες κάθε επεξεργαστή αλλά και να φροντίζει για τη διάδοση τυχόν εσφαλμένων τιμών στις εξόδους του πολυεπεξεργαστή. Επίσης η ανάπτυξη της *πολυνηματικής επεξεργασίας σε επίπεδο ολοκληρωμένου κυκλώματος* (*Chip MultiThreading - CMT*) σηματοδοτεί ένα νέο ερευνητικό πεδίο για την ανάπτυξη του λογισμικού αυτοδοκιμής. Το λογισμικό αυτοδοκιμής μπορεί να αναπτυχθεί σε επιμέρους τμήματα τα οποία θα εκτελούνται από πολλά *νήματα* (*threads*) στους πολλαπλούς επεξεργαστές, ώστε να μειώνονται τα τμήματα μη απασχόλησης των νημάτων που οφείλονται είτε σε προσπελάσεις από την κοινόχρηστη κρυφή μνήμη, είτε από εντολές που απαιτούν πολλούς κύκλους ρολογιού, με τελικό στόχο τη μείωση του συνολικού χρόνου εκτέλεσης της δοκιμής.

Βιβλιογραφία

[1] M. Annaratone, *Digital CMOS Circuit Design*, 2nd edition , Kluwer Academic Publishers, 1986.

[2] M. M. Mano, *Digital Design*, 4rd edition, Prentice Hall, 2006.

[3] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design: A Systems Perspective*, 3rd edition, Addison-Wesley Publishing Company, 1994.

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

[5] Microprocessors poster. Intel. [Online]. Available: http://download.intel.com/museum/Moores_Law/Printed_Materials/Microprocessor_Poster_24_36.pdf

[6] International technology roadmap for semiconductors - itrs. [Online]. Available http://www.itrs.net/Links/2007ITRS/Home2007.htm

[7] Statistical analysis of floating point flaw. Intel. [Online]. Available: http://support.intel com/support/processors/pentium/fdiv/wp/

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

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

[10] Intel "Components Quality Reliability Handbook" Intel Corporation: Santa Clara, 1987.

[11] M. Sachdev, *Defects Oriented Testing of CMOS Analog and Digital Circuits*, Kluwer Academic, Norwell, MA, 1998.

[12] R. Eldred, "Test Routines Based on Symbolic Logical Statements," in *Journal of the ACM*, vol. 6, no. 1, pp. 33–37, 1959.

[13] International Technology Roadmap for Semiconductors (ITRS), Test and Test Equipment 2007 Edition, http://public.itrs.net

[14] G.Aldrich and B. Cory, "Improving Test Quality and Reducing Escapes" in *Fabless Forum*, Fabless Semiconductor Assoc., 2003, pp. 34-35.

[15] S. Cook, "Rethinking test at 130 nanometers and below", in *EE Design*, September 2003.

[16] M. Abramovici, M.A. Breuer, A.D. Friedman, *Digital system testing and testable design*, Revised edition, Wiley-IEEE Press,, 1994.

[17] N. Jha and S. Gupta, *Testing of Digital Systems*, Cambridge University Press, 2003.

[18] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures Design for Testability (Systems on Silicon)*, Morgan Kaufmann, 2006.

[19] P. Nigh, W. Needham, K. Butler, P. Maxwell, and R. Aitken, "An experimental study comparing the relative effectiveness of functional, scan, iddq and delay- fault testing," in *IEEE VLSI Test Symposium,* 1997, pp. 459–464.

[20] P. Gelsinger, "Discontinuities driven by a billion connected machines," in *IEEE Design and Test of Computers*, vol. 17, no. 1, pp. 7–15, 2000.

[21] J. Gatej, L. Song, C. Pyron, R. Raina, and T. Munns, "Evaluating ATE features in terms of test escape rates and other cost of test culprits," in *IEEE International Test Conference*, pp. 1040-1048, 2002.

[22] D. Gizopoulos, A. Paschalis, and Y. Zorian, *Embedded Processor-Based Self-Test*, Kluwer Academic Publishers, 2004.

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

[24] Opencores Projects www.opencores.org/projects/ (OpenRISC, UART, HDLC, Ethernet and Wishbone models).

[25] Stanford Small Benchmark Suite, www.stanford.edu

[26] A. Apostolakis, M. Psarakis, D. Gizopoulos, A. Paschalis, "A Functional Self-Test Approach for Peripheral Cores in Processor-Based SoCs", in *IEEE International On-Line Testing Symposium*, pp. 271 - 276, July 2007.

[27] A. Apostolakis, M. Psarakis, D. Gizopoulos, A. Paschalis, "Functional Processor-Based Testing of Communication Peripherals in Systems-on-Chip", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 971 - 975, August 2007.

[28] A. Apostolakis, D. Gizopoulos, M. Psarakis, D. Ravotto, M. Sonza Reorda, "Test Program Generation for Communication Peripherals in Processor-Based System-on-Chips", in *IEEE Design and Test of Computers*, pp. 52 - 63, March-April, 2009.

[29] A. Apostolakis, M. Psarakis, D. Gizopoulos, A. Paschalis, "Functional Self-Testing for Bus-Based Symmetric Multiprocessors", in *Design Automation and Test in Europe*, pp. 1304 – 1309, March 2008.

[30] A. Apostolakis, D. Gizopoulos, M. Psarakis, I. Parulkar, "Exploiting Execution Parallelism in Functional Self-Testing of Chip Multiprocessors", in *Design for Reliability Workshop*, January 2009.

[31] A. Apostolakis, M. Psarakis, D. Gizopoulos, A. Paschalis, "Software-Based Self-Testing of Symmetric Shared-Memory Multiprocessors", in *IEEE Transactions on Computers* (accepted).

[32] T.W. Williams, and N.C. Brown, "Defect level as a function of fault coverage" in *IEEE Transactions on Computers*, pp. 987-988, December 1981.

[33] Agrawal et al, "Fault coverage requirements in production testing of LSI circuits" in *IEEE Journal of Solid-State Circuits*, pp. 57-61, February 1982.

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

[35] P.C. Maxwell, and R.C. Aitken "Test sets and reject rates: all fault coverage are not created equal" in *IEEE Design and Test of Computers*, pp. 42-51, March 1993.

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

[37] K. Singer, "VTS 97 Keynote: The Future of Test and DFT", in *IEEE Design & Test*, pp. 11-14, July-Sep. 1997.

[38] J. Grason "TMEAS-a testability measurement program", in *IEEE Design Automation Conference*, pp. 156-161, June 1979.

[39] M.A. Breuer, and A.D. Friedman "TEST/80-a proposal for an advanced automatic test generation system", in *IEEE Autatestcom*, pp. 205-312, 1979.

[40] L.M. Goldstein, and E.L. Thigan "SCOAP: Sandia controllability and observability analysis program", in *IEEE Design Automation Conference*, pp. 190-196, 1980.

[41] R.G. Bennetts, C.M. Mauder, G.D. Robinson, "COMELOT: a Computer-aided measure for logic testability" in *IEE Proceedings E. Computers and Digital Techniques*, pp. 177-189, November 1981.

[42] I.M. Ratiu, A. Sangiovanni-Vingentelli, D.O. Peterson "VICTOR: a fast VLSI testability analysis programme", in *IEEE International Conference on Test*, pp. 397-401, 1982.

[43] W.C. Berg, R.D. Hess "COMET: a testability analysis and design modification package", in *IEEE International Conference on Test*, pp. 364-378, 1982.

[44] V. Agrawal and A. Fung, "Multiple Fault Testing of Large Circuits by Single Fault Test Sets" in *IEEE Transactions Computers*, vol. 30, no. 11, pp. 855–865, November 1981.

[45] Jacob and N. N. Biswas, "GTBD Faults and Lower Bounds on Multiple Fault Coverage of Single Fault Test Sets," in *IEEE International Test Conference*, 1987, pp. 849–855.

[46] J. Roth, "Diagnosis of automata failures: a calculus and a method," in *IBM Journal of Research and Development*, vol. 10, no. 4, pp. 278–291, 1990.

[47] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," in *IEEE Transactions Computers*, vol. 30, no. 3, pp. 215–222, March 1981.

[48] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," in *IEEE Transactions Computers*, vol. 32, no. 12, pp. 1137–1144, 1983.

[49] K.T. Cheng, "Transition fault testing for sequential circuits" in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1971-1983, December 1993.

[50] Y. Levendel, P.R. Menon "Transition faults in combinational circuits: Input transition test generation and fault simulation" in *International Fault Tolerant Computing Symposioum*, pp. 278-283, 1986.

[51] M.H. Schulz, F. Brglez "Accelerated transition fault simulation" in *IEEE Design Automation Conference*, pp. 237-243, 1987.

[52] J.A. Waicukauski, E. Lindbloom, B. Rosen, V. Iyengar "Transition fault simulation" in *IEEE Transactions of Computers*, vol. 4, no. 2, pp. 32-38, April 1987.

[53] R.L. Wadsack, J.M. Soden, R.K. Treece, M.R. Taylor, C.F. Hawkins, "CMOS IC stuck-open fault electrical effects and design considerations", in *IEEE International Test Conference*, pp 423-430, August 1989.

[54] J. Carter, V. Iyengar, B. Rosen, "Efficient test coverage determination for delay faults", in *IEEE International Test Conference*, pp. 418-427, 1987.

[55] G.L Smith "Model for delay faults based upon paths", in *IEEE International Test Conference*, pp 342-349, 1985.

[56] W.N. Li, S.M. Reddy, S.K. Sahni, "On path Selection in combinational logic circuits", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8 no. 1, pp. 56-63, 1989.

[57] A.K. Majhi, J. Jacob, L.M. Patnail, V.D. Agrawal, "On test coverage of path delay faults" in *IEEE International Conference on VLSI Design*", pp. 418-421, January 1996.

[58] Y.K. Malaiya, R. Narayanaswamy, "Modeling and testing for timing faults in synchronous sequential circuits", in *IEEE Design and Test of Computers*, pp. 62-74, 1984.

[59] E.S. Park, B. Underwood, T.W. Williams, M.R. Mercer, "Delay testing quality in timing-optimized designs", in *IEEE International Test Conference*, pp 897-905, 1991.

[60] A. Kristic, K.T. Cheng, "Resynthesis of combinational circuits for path count reduction and path delay fault testability", in *European Design and Test Conference*, pp. 486-490, March 1996.

[61] I. Pomeranz, S.M. Reddy, "On synthesis-for-testability of combinational logic circuits", in *IEEE Design Automation Conference*, pp. 126-132, 1995.

[62] R. Rajsuman *Digital hardware testing: transistor-level fault modeling and testing*, Artect House Publishers, 1992.

[63] P.K. Lala, *Fault tolerant and fault testable hardware design*, Prentice Hall, 1984.

[64] G. Russell, *Computer aided tools for VLSI system design*, Peter Peregrinus, 1987.

[65] M.A. Breuer, A.D. Friedman, *Diagnosis and reliable design of digital systems*, Computer Science Press, 1976.

[66] Y.H. Levendel, P.R. Menon, "Fault simulation methods – extension and comparison", in *Bell Systems Technical Journal*, vol. 60, pp.2235-2258, November 1981.

[67] P. Goel, "Test generation cost analysis and projection", in *IEEE Design Automation Conference*, pp. 77-84, 1980.

[68] D.B. Armstrong, "A deductive method of simulating faults in logic circuits", in *IEEE Transactions* Computers, vol. C-21, no. 5, pp. 464-471, 1972.

[69] E.G. Ulrich, T.G. Baker, "Concurrent simulation of nearly identical digital networks", in *IEEE Design Automation Conference*, pp. 318-323, 1988.

[70] J.A. Waicukauski, E.B. Eichelburger, D.O. Forlenza, E. Lindbloom, T. McCarthy, "Fault simulation for structural VLSI", in VLSI Systems Design, pp. 20-32, 1985.

[71] T.W. Williams, *VLSI Testing*, North-Holland, 1986.

[72] R. T. Maniwa, "Design for Test Tools", in *Integrated System Design*, pp. 60-74, Sep. 1996.

[73] B. T. Murray, J. P. Hayes, "Testing ICs: Getting to the Core of the Problem", in *IEEE Computer*, vol. 29, no. 11, pp. 33-38, November 1996.

[74] M.L. Bushnell, V.D. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, Kluwer Academic Publisers, 2000.

[75] E.B. Eichrlbrger, E. Lindbloom, J.A. Waicukauski, T.W. Willinams, *Structured logic testing*, Englewood Cliffs, New Jersey: Prentice-Hall, 1991.

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

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

[78] 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. 1, pp. 73-82, March, 1993.

[79] 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.

[80] P.H. Bardell, W.H. McAnney, J. Savir, *Built-In test for VLSI: Pseudorandom techniques*, John Wiley and Sons, New Work, 1987.

[81] R.A. Frohwerk, "Signature analysis: A new digital field service method", in *Hewlett-Packard Journal*, vol. 28, no. 9, pp. 2-8, May 1977.

[82] S. Thatte and J. Abraham, "Test Generation for Microprocessors" in *IEEE Transactions on Computers*, vol. 29, no. 6, pp. 429–441, 1980.

[83] R. Tupuri and J. Abraham, "A Novel Functional Test Generation Method for Processors using Commercial ATPG," in *IEEE International Test Conference*, pp. 743–752, November 1997.

[84] 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.

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

[86] 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.

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

[88] L. Chen and S. Dey, "Software-based self-testing methodology for processor cores," in *IEEE Transactions Computer-Aided Design Integrated Circuits and Systems*, vol. 20, no. 3, pp. 369–380, March 2001.

[89] L. Chen, S. Ravi, A. Raghunathan, and S. Dey, "A scalable software-based self-testing methodology for programmable processors" in *IEEE Design Automation Conference*, pp. 548–553, June 2003.

[90] C. H.-P. Wen, L.-C. Wang, and K.-T. Cheng, "Simulation-based functional test generation for embedded processors" in *IEEE Transactions on Computers*, vol. 55, no. 11, pp. 1335–1343, 2006.

[91] S. Gurumurthy, S. Vasudevan, and J. Abraham, "Automated mapping of pre-computed module-level test sequences to processor instructions," in *IEEE International Test Conference*, pp. 303-313, November 2005.

[92] S. Guramurthy, S. Vasudevan, J.A. Abraham, "Automatic generation of instruction sequences targeting hard-to-detect structural faults in a processor," in *IEEE International Test Conference*, pp. 294–303, 2006.

[93] 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.

[94] 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.

[95] 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.

[96] 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.

[97] 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

[98] D. Gizopoulos, *Advances in Electronic Testing: Challenges and Methodologies*, Springer, November 2006.

[99] Ricardo Reis, Jochen Jess, *Design of System-on-Chip: Devices & Components*, Springer, July 2004.

[100] Erik Jan Marinissen and Yervant Zorian, "Challenges in Testing Core- Based System ICs", in *IEEE Communications Magazine*, vol. 37, no. 6, pp. 104-109, June 1999.

[101] IEEE 1450, "Standard Test Interface Language (STIL)", http://grouper.ieee.org/groups/1450.

[102] IEEE 1450.6, "Standard for Core Test Language (CTL)", http://grouper.ieee.org/groups/ctl.

[103] G.Hetherigton, T.Fryars, N.Tamarapalli, M.Kassab, A.Hassan, J.Rajski, "Logic BIST for Large Industrial Designs: Real Issues and Case Studies", in *IEEE International Test Conference*, pp. 358–367, 1999.

[104] Y. Zorian, E.J. Marinissen, S. Dey, "Testing embedded-core-based system chips", in *IEEE Computer*, vol. 32, no. 6, 1999.

[105] E.J. Marinissen, R. Kapur, M. Lousberg, T. McLaurin, M. Ricchetti, Y. Zorian, "On IEEE P1500's Standard for Embedded Core Test", in *Springer Journal of Electronic Testing*, vol. 18, no. 4-5, 2002.

[106] E.J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, L. Whetsel, "Towards a standard for embedded core test: an example", in *IEEE International Test Conference*, pp. 616-627, 1999.

[107] F. DaSilva, Y. Zorian, L. Whetsel, K.Arabi, R. Kapur, "Overview of the IEEE P1500 standard", in *IEEE International Test Conference*, pp. 988-997, 2003.

[108] B. Vermeulen, C. Hora, B. Kruseman, E.J. Marinissen, R. van Rijsinge, "Trends in testing integrated circuits", in *IEEE International Test Conference*, pp. 688-697, 2003.

[109] S . Koranne, "Design of reconfigurable access wrappers for embedded core based SoC test", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 955-960, October 2003.

[110] A. Sehgal, S.K. Goel, E.J. Marinissen, K. Chakrabarty, "IEEE P1500-compliant test wrapper design for hierarchical cores", in *IEEE International Test Conference*, pp. 1203 – 1212, October 2004.

[111] M.G. Wahl, S. Bhawmik, K. Zarrineh, P. Ghosh, S. Davidson, P. Harrod, "The P1500 DFT disclosure document: a standard to communicate mergeable core DFT data" in *IEEE International Test Conference*, pp.998 – 1007, September - October 2003.

[112] E.J. Marinissen, T. Waayers, "Infrastructure for modular SOC testing", in *IEEE Custom Integrated Circuits Conference*, pp. 671 - 678, October 2004.

[113] A.M. Amory, K. Goossens, E.J. Marinissen, M. Lubaszewski, F. Moraes, "Wrapper Design for the Reuse of Networks-on-Chip as Test Access Mechanism" in *IEEE European Test Symposium*, pp. 213 – 218, May 2006.

[114] V. lyengar; K. Chakrabarty E.J. Marinissen, "Test access mechanism optimization, test scheduling, and tester data volume reduction for system-on-chip", in *IEEE Transactions on Computers*, Volume 52, Issue 12, 2003.

[115] D. Appello, P. Bernardi, M. Grosso, M.S. Reorda, "System-in-package testing: problems and solutions", in *IEEE Design & Test of Computers*, vol. 23, no.3, pp. 203 – 211, May – June 2006.

[116] T. Waayers, R. Morren, R. Grandi, "Definition of a robust modular SOC test architecture; resurrection of the single TAM daisy-chain", in *IEEE International Test Conference*, pp. 619 – 629, November 2005.

[117] Y. Zorian, A. Yessayan, "IEEE 1500 utilization in SOC design and test", in *IEEE International Test Conference*, 2005.

[118] A. Sehgal, S.K. Goel, E.J. Marinissen, K. Chakrabarty, "Hierarchy-Aware and Area-Efficient Test Infrastructure Design for Core-Based System Chips", in *IEEE Design, Automation and Test in Europe*, 2006.

[119] C. Wu H. Wang, S. Yang, "A P1500-compliant wrapper and TAM controller co-design scheme", in *IEEE International Conference On ASIC*, pp. 709 – 713, 2005.

[120] E.J. Marinissen, M. Lousberg, "The role of test protocols in testing embedded-core-based system ICs", in *IEEE European Test Workshop*, pp. 70 – 75, 1999.

[121] D. Appello, V. Tancorre, P. Bernardi, M. Grosso, M. Rebaudengo,; M.S. Reorda, "On the automation of the test flow of complex SoCs", in *IEEE VLSI Test Symposium*, 2006.

[122] L. Kuen-Jong, C. Chia-Yi, H. Yu-Ting "An embedded processor based SOC test platform", in *IEEE International Symposium on Circuits and Systems*, pp. 2983 – 2986, May 2005.

[123] S. Hwang, J.A. Abraham, "Reuse of Addressable System Bus for SoC Testing", in *IEEE International ASIC/SoC Conference*, pp. 215–219, 2001.

[124] J-R. Huang, M.K. Iyer, K-T. Cheng, "A Self-Test Methodology for IP Cores in Bus-Based Programmable SoCs", in *IEEE VLSI Test Symposium*, pp. 198 – 203, 2001.

[125] W-C. Lai, K-T. Cheng, "Instruction-Level DFT for Testing Processor and IP Cores in System-on-a-Chip", in *IEEE Design Automation Conference*, pp. 59 – 64, 2001.

[126] K. Jayaraman, V.M. Vedula, J.A. Abraham, "Native Mode Functional Self-Test Generation for Systems-on-Chip", in *International Symposium on Quality Electronic Design*, pp. 280–285, 2002.

[127] L. Bolzani, E. Sanchez, M. Schillaci, M. Sonza Reorda, G. Squillero, "An Automated Methodology for Cogeneration of Test Blocks for Peripheral Cores», in *IEEE International On-Line Testing Symposium*, pp. 265-270, July 2007.

[128] A. Krstic, W.C. Lai, K.T Cheng, L. Chen, S. Dey, "Embedded Software-Based Self-Test for Programmable Core-Based Designs", in *IEEE Design & Test of Computers*, vol. 19, no. 4, pp.18–27, 2002.

[129] A. Van de Goor, Y. Zorian, "Effective march algorithms for testing single-order addressed memories", in *Journal of Electronic Testing: Theory and Applications*, pp. 499 – 505, February 1994.

[130] A.J. Van de Goor, I. Schanstra, Y. Zorian, "Functional test for shifting-type FIFOs", in *European Design and Test Conference*, pp. 133 - 138, 1995.

[131] A.J. Van de Goor, I. Schanstra, Y. Zorian, **"**Fault models and tests for Ring Address Type FIFOs", in *IEEE VLSI Test Symposium*, pp. 300 – 305, 1994.

[132] CoreConnect Bus Architecture, http://www.ibm.com/

[133] Advanced Microcontroller Bus Architecture (AMBA), http://www.arm.com

[134] Synopsys, http://www.synopsys.com

[135] Mentor Graphics, http://www.mentor.com

[136] Sun Microsystems, http://www.sun.com

[137] OpenSPARC Processor Cores, http://www.opensparc.net

[138] Intel® Corporation, http://www.intel.com

[139] Adavanced Micro Devices, AMD, http://www.amd.com

[140] Intel® Research Advances "Era of Tera" Intel Corporation: Santa Clara, 2007.

[141] Intel® Microarchitecture (Core 2), Intel Corporation: Santa Clara.

[142] T. Agerwala, J. Cocke, "High performance reduced instruction set processors", Technical report RC12434 (#55845. Yorktown, NY: IMB J. Thomas Watson Research Center, January 1987.

[143] Poonacha Kongetira, Kathirgamar Aingaran, Kunle Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor», in *IEEE Conference on Microarchitecture*, vol. 25, no. 2, pp. 21-29, Mar/Apr, 2005.

[144] J. Hennessy and D. Patterson, *Computer Architecture. A Quantitative Approach*, 4th Edition, Morgan Kauffman, 2006.

[145] M.Riley, L.Bushard, N.Chelstrom, N.Kiryu, S.Ferguson, "Testability Features of the First-Generation Cell Processor", in *IEEE International Test Conference*, pp. 119 - 128, November 2005.

[146] L.Bushard, N.Chelstrom, S.Ferguson, B.Keller, "DFT of the Cell Processor and its Impact on EDA Test Software", in *IEEE Asian Test Symposium*, pp. 369 – 374, November 2006.

[147] S.Makar, T.Altinis, N.Patkar, J.Wu, "Testing of Vega2, a chip multi-processor with spare processors", in *IEEE International Test Conference*, pp. 1-10, October 2007.

[148] I.Parulkar, T.Ziaja, R.Pendurkar, A.D'Souza, A.Majumdar, "A scalable, low cost design-for-test architecture for UltraSPARC Chip multi-processors" in *IEEE International Test Conference*, pp. 726 - 735, 2002.

[149] P.J.Tan, T.Le, K.-H.Ng, P.Mantri, J.Westfall, "Testing of UltraSPARC T1 Microprocessor and its Challenges", in *IEEE International Test Conference*, pp. 1 - 10, October 2006.

[150] I.Bayraktaroglu, J.Hunt, D.Watkins, "Cache Resident Functional Microprocessor Testing: Avoiding High Speed IO Issues", in *IEEE International Test Conference*, pp. 1-7, October 2006.

[151] K.Constantinides, O.Mutlu, T.Austin, V.Bertacco, "Software-Based Online Detection of Hardware Defects: Mechanisms, Architectural Support, and Evaluation", in *IEEE Conference on Microarchitecture*, pp. 97 – 108, Deceber 2007

[152] O.Guzey, L.-C.Wang, J.Bhadra, "Enhancing signal controllability in functional test-benches through automatic constraint extraction" in *IEEE International Test Conference*, pp. 1-10, October 2007.

[153] R.B. Cooper, Introduction to Queuing Theory, Elsevier 1981.