Υλοποίηση Ομόχειρου Διαχειριστή Ουρών σε Γλώσσα Περιγραφής Υλικού
HDL implementation of a Pipelined Queue Manager

Αικατερίνη Πέτσα

Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

Χανιά, Ιούλιος 2011

Περίληψη

Η δομή των Ουρών, κατέχει αδιαλείπτως κεντρικό ρόλο στα συστήματα υλικού, παρά ή εξαιτίας της απλότητάς της, αλλά κυρίως λόγω της ιδιότητας αναπαράστασης χρονικά διατεταγμένων γεγονότων. Εξάλλου, η βελτιστοποίηση της κατάτμησης ενός μονοπατιού δεδομένων σε στάδια και παράλληλης εκτέλεσης εντολών ομόχειρα, είναι η πλέον διαδεδομένη και προφανής μέθοδος επιτάχυνσης. Στην παρούσα εργασία, ένας ομόχειρος διαχειριστής πολλαπλών ουρών σχεδιάστηκε εξαρχής και υλοποιήθηκε σε γλώσσα VHDL, ως παραμετροποιήσιμο σύστημα. Επαληθεύτηκε, σε αντιπαραβολή με μοντέλο σε γλώσσα python, παράγοντας, εκτελώντας και συγκρίνοντας αυτοματοποιημένα, εκτενή σύνολα ελέγχου.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Επί του θέματος ...

Η διαχείριση μνήμης, πολλές φορές ακόμα και σε περιπτώσεις εξειδικευμένων συστημάτων, συνήθως πραγματοποιείται από λογισμικό, σε ενσωματωμένους επεξεργαστές γενικού σκοπού. Δεν είναι ο μόνος, ούτε και ο αποδοτικότερος τρόπος. Η άλλη λύση, που αποδεικνύει κάθε φορά την ανωτερότητά της ως προς την απόδοση, είναι η συμπερίληψη κάποιου υποσυστήματος διαχείρισης της μνήμης στο κομμάτι της σχεδίασης του συστήματος, ιδίως αν πρόκειται για πολύ καλά ορισμένες λειτουργίες. Η σχεδίαση ενός συστήματος σε υλικό, το οποίο επιτελεί συγκεκριμένη λειτουργία, αλλά και στοχεύει σε συγκεκριμένη τεχνολογία, επιτρέπει την εκμετάλλευση τεχνικών βελτιστοποίησης, που είτε έχουν καθιερωθεί, είτε ανακύπτουν από τις ιδιαιτερότητες του προβλήματος. Η εφαρμογή των τεχνικών αυτών προϋποθέτει την ανάλυση του προβλήματος, για την προσαρμογή τους στην κάθε περίπτωση, αλλά και αναγνώριση των περιθωρίων εισαγωγής νέων ιδεών. Σε κάθε περίπτωση, η σχεδιαστική διαδικασία δεν μπορεί θεωρηθεί ολοκληρωμένη χωρίς μία συστηματική επιβεβαίωση των αποτελεσμάτων της.

Από τις προσομοιώσεις, στη διαδιεργασιακή επικοινωνία (InterProcess Communication, IPC), και από την χρονοδρομολόγηση των διεργασιών του επεξεργαστή σε συστήματα χρονομερισμού, μέχρι την δικτυακή κίνηση [3, 4], αλλά ακόμη και σε προβλήματα χωρίς χρονικά χαρακτηριστικά, όπως δρομολόγηση καλωδίων και απόδοση επιγραφής σε συστατικά εικόνας [1], το εύρος αλλά και η σημασία του φάσματος των εφαρμογών της δομής της Ουράς, παρά την απλότητά της, την τοποθετεί ανάμεσα στις σπουδαιότερες δομές.

Η τεχνική της ομοχειρίας, κινώντας το ερευνητικό ενδιαφέρον από την εποχή της εμφάνισης της, τη δεκαετία του 1960, και με επέκτασή της κάθε επόμενη δεκαετία σε νέο πεδίο ανάπτυξης των υπολογιστών, αρχικά από τους υπερυπολογιστές, φτάνοντας στους επιτραπέζιους και ενσωματωμένους επεξεργαστές, έχει καθιερωθεί σήμερα σε τέτοια έκταση που να αποτελεί, παρότι βελτιστοποίηση, την τυπική μέθοδο σχεδίασης επεξεργαστών, και υλικού κατ΄

επέκταση. Διδάσκεται, μάλιστα, ως βασικό προπτυχιακό υλικό, στα πανεπιστήμια και πολυτεχνεία.[5, 6]

Η διαδικασία για την εμπέδωση οποιασδήποτε έννοιας και αντικειμένου, υπόκειται, αν πράγματι διενεργείται συνειδητά και σκόπιμα, σε κανόνες της παιδαγωγικής και διδακτικής, οι οποίοι, στη συνέχεια, διαμορφώνουν κάποια συγκεκριμένη δομή και μορφή διδασκαλίας. Πράγματι, μία από τις μορφές που ακολουθείται συχνά, ιδίως όταν πρόκειται για την εισαγωγή νέων εννοιών, σε ανώριμα, σε γνωστικό επίπεδο, υποκείμενα, είναι εκείνη των ομόκεντρων κύκλων, δηλαδή, η διαδοχική επαναφορά στο ίδιο ζήτημα, αλλά με ευρύτερη ακτίνα εμβάθυνσης. Αν λοιπόν, η εισαγωγή των εννοιών της τεχνικής της ομοχειρίας, στο προπτυχιακό πρόγραμμα, πραγματοποιείται πάνω σε δεδομένη, δοκιμασμένη και επιβεβαιωμένη σχεδίαση, το επόμενο επίπεδο εμβάθυνσης, είναι μιά ολοκληρωμένη σχεδίαση από την αρχή. Η διεύρυνση της οπτικής, συμβάλλει στην πληρέστερη κατανόηση της τεχνικής, αλλά συντελεί και στην εξοικείωση με τη σχεδιαστική διαδικασία, η οποία αποτελεί, ένα γνωστικό πεδίο καθεαυτή. Άλλωστε, η ενίσχυση της θεμελίωσης της υπάρχουσας γνώσης, είναι εκ των ων ουκ άνευ, για όλη την περιπέτεια της μάθησης.

Στην παρούσα εργασία, σχεδιάζεται ένα γενικευμένο σύστημα διαχείρισης ουρών, σε γλώσσα περιγραφής υλικού VHDL, και με τεχνολογία στόχο τις συσκευές διάταξης πυλών προγραμματιζόμενου πεδίου (field-programmable gate array, FPGA). Παρουσιάζεται η διαδικασία της προσαρμογής της τεχνικής της ομοχειρίας στο συγκεκριμένο πρόβλημα, και άλλες επιμέρους σχεδιαστικές επιλογές. Ιδιαίτερη προσοχή δόθηκε στην δυνατότητα γενίκευσης του κώδικα, ώστε να προκύπτουν συστήματα διαφορετικών, αυθαίρετων διαστάσεων. Τελικά, το σύστημα υπο σχεδίαση περνά από τα εργαλεία σύνθεσης της Xilinx, και τόσο η αρχική εκδοχή, όσο και ο κώδικας που προκύπτει από την σύνθεση εκτελείται για επιβεβαίωση σε ένα ιδιότυπο σύστημα επαλήθευσης. Το σύστημα, γραμμένο σε μείγμα γλώσσας python, Tcl και εντολών του περιβάλλοντος προσομοίωσης ModelSim, ενσωματώνει το μοντέλο στο οποίο αντιπαραβάλλεται το σύστημα, και παράγει, βάσει επιλογών του χρήστη, σενάρια με αυθαίρετο αριθμό εντολών. Ακόμα, αναλαμβάνει την αυτόματη εκτέλεση του ίδιου συνόλου εντολών, τόσο από το σύστημα όσο και το μοντέλο, αποθηκεύοντας και συγκρίνοντας τις εξόδους τους.

1.2 Διάρθρωση κειμένου

Στο κεφάλαιο 2 δίνεται το θεωρητικό υπόβαθρο των δομών Ουράς και Στοίβας και της τεχνικής της ομοχειρίας. Στο κεφάλαιο 3 παρουσιάζεται η προσαρμογή της τεχνικής της ομοχειρίας στο συγκεκριμένο πρόβλημα, και στα κεφάλαια 4 και 5 αποσαφηνίζονται κάποιες λεπτομέρειες υλοποίησης και σχεδιαστικές επιλογές. Αντίστοιχα, στο κεφάλαιο 6 δίνονται οι γενικές γραμμές της μεθοδολογίας επαλήθευσης που ακολουθήθηκε, με πλήρη ανάλυσή της λειτουργικότητας του συστήματος που αναπτύχθηκε για το σκοπό αυτό, στο παράρτημα

Α΄. Τελικά, στο κεφάλαιο 7 παρουσιάζονται κάποιες σχετικές εργασίες, ενώ αντίστοιχα, προτάσεις για μελλοντικές, μαζί με μία συνοπτική παρουσίαση της εργασίας, στο κεφάλαιο 8.

Το κείμενο είναι γραμμένο σε δύο επίπεδα αφαίρεσης, και μπορεί να διαβαστεί με δύο τρόπους: είτε διαδοχικά, στην πλήρη έκτασή του, είτε παραλείποντας τα αναλυτικότερα κεφάλαια (4, 5, παρ. Α΄) για μία ταχύτερη εποπτεία.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

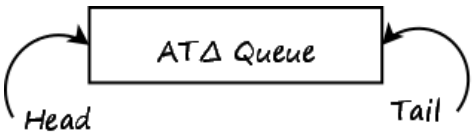
Πριν την σχεδίαση, την υλοποίηση σε γλώσσα περιγραφής υλικού, και την επαλήθευση του ομόχειρου διαχειριστή πολλαπλών Ουρών, είναι σκόπιμο να αναφερθούν συνοπτικά οι κυριότερες θεωρητικές συνιστώσες του θέματος. Αρχικά, μια αναφορά στις συγγενικές δομές δεδομένων, Ουρά και Στοίβα, και στη συνέχεια, μία επιγραμματική επισκόπηση των σημαντικότερων εννοιών που αφορούν την μέθοδο υλοποίησης, την τεχνική της ομοχειρίας.

2.1 Εισαγωγή στις δομές δεδομένων

Η Ουρά, διεθνώς Queue ή FIFO-List, είναι μία από τις πιο διαδεδομένες δομές δεδομένων, καθώς βασίζεται σε μιά στοιχειώδη, και ιδιαιτέρως διαισθητική διάρθρωση, ενώ ταυτόχρονα, οι χρονικής υφής ιδιότητές της, την καθιστούν ιδανική για πληθώρα εφαρμογών, με απαιτήσεις αναπαράστασης χαρακτηριστικών χρονικής ακολουθίας. Βέβαια, την ίδια απλότητα και χρηστικότητα, μοιράζεται, στον αντίποδα, και η δίδυμή της δομή, η Στοίβα, διεθνώς Stack ή LIFO-List.

Και οι δύο αυτές δομές, αποτελούν χρονικές ταξινομήσεις γεγονότων, τα οποία αποτυπώνονται στα δεδομένα των δομών, προσομοιώνοντας διαδικασίες αφίξεων και αναχωρήσεων, με διαφοροποίηση στην πολιτική της αναχώρησης, First-In, First-Out ή Last-In, First-Out. Στη μεν Ουρά, το προνομιακό χαρακτηριστικό είναι η παλαιότητα, ενώ στην Στοίβα η νεότητα, αναφορικά με τη χρονική στιγμή άφιξης κάθε γεγονότος, σχετικά με τα επόμενα, ή προηγούμενα αντίστοιχα, και το οποίο καθορίζει και την προτεραιότητα στις αναχωρήσεις, υπαγορεύοντας ουσιαστικά και την ιδιαιτερότητα της κάθε δομής, τόσο σε επίπεδο ορισμού και συμπεριφοράς, όσο και υλοποίησης.[2]

Κάθε δομή δεδομένων, περιγράφεται από τον Αφηρημένο Τύπο Δεδομένων (ΑΤΔ) της, διεθνώς Abstract Data Type, (ADT), ο οποίος είναι η απαρίθμηση των χαρακτηριστικών που την διακρίνουν από κάθε άλλη δομή δεδομένων, την καθορίζουν πλήρως, ως προς τη συμπεριφορά και τις λειτουργίες της, αποκρύπτοντας όμως τις λεπτομέρειες της εκάστοτε υλοποίησης. Παρόλο που



Σχήμα 2.1: Ο ΑΤΔ Ουρά, με τα σημεία πρόσβασής του.

οι παραλλαγές των υλοποιήσεων μπορεί να είναι ποικίλες, και με αντίκτυπο ακόμα και σε σημαντικότατα χαρακτηριστικά της δομής, όπως την χρονική και χωρική της πολυπλοκότητα,[1] δεν διαφοροποιούν τη δομή από τον βασικό της τύπο, τον ΑΤΔ.

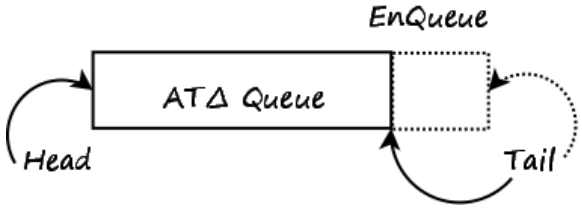
2.1.1 Ο ΑΤΔ Ουρά

Σε επίπεδο ΑΤΔ, επομένως, η δομή Ουρά ορίζεται ως η δομή εκείνη όπου η πρόσβαση στα δεδομένα γίνεται με σειρά παλαιότητας, δηλαδή, κάθε ανάκτηση δεδομένου αποδίδει εκείνο το οποίο είχε καταχωρηθεί στη δομή νωρίτερα, ή παλαιότερα, από *όλα* τα υπόλοιπα δεδομένα που είναι επίσης καταχωρημένα. Η απαίτηση αυτή έχει δύο προεκτάσεις : αφενός, η αποθήκευση των δεδομένων προσανατολίζεται σε διάρθρωση αντίστοιχη με την χρονική ακολουθία των αφίξεων. Αφετέρου, οποιεσδήποτε ενέργειες στη δομή πραγματοποιούνται αποκλειστικά στα στοιχεία με τα ακραία χρονικά χαρακτηριστικά. Οι εισαγωγές αφορούν τα νέα δεδομένα, μεν, όπως σε κάθε δομή, αλλά, οι εξαγωγές, και κάθε είδους περαιτέρω προσπέλαση των δεδομένων, περιορίζεται αυστηρά στο ακραίο εκείνο στοιχείο που αναπαριστά το παλαιότερο γεγονός, με μόνη ίσως εξαίρεση το πιό πρόσφατο όλων. Μάλιστα, η ανάκτηση επόμενων, νεότερων, στοιχείων είναι αδύνατη, αν τα προηγούμενά τους, παλαιότερα, δεν έχουν ήδη ανακτηθεί και διαγραφεί.

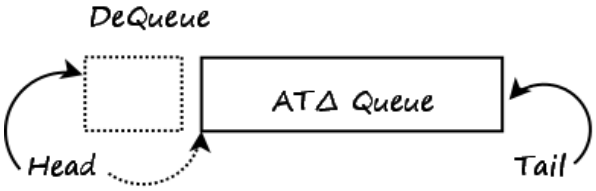
Η δομή της Ουράς, μπορεί λοιπόν να αναπαραστήσει τέλεια μία πραγματική ουρά αναμονής, σε ένα σταθμό εξυπηρέτησης πελατών, όπως ένα ταμείο, όπου οι οντότητες δεδομένων, τα στοιχεία ή τα γεγονότα, εδώ τα φυσικά πρόσωπα, εντάσσονται στην ουρά στο ένα άκρο της, εκείνο που απέχει από το σημείο εξυπηρέτησης, και αναχωρούν από το άλλο, μετά την εξυπηρέτησή τους. Βέβαια, επειδή η χρήση της δεν προορίζεται απαραίτητα και μόνο για τέτοιες διαδικασίες, σωστότερο και χρησιμότερο για την συγκεκριμένη περιγραφή, είναι ότι μία πραγματική ουρά αναμονής, προσομοιάζει σε πολλά χαρακτηριστικά της την δομή Ουρά, εξυπηρετώντας σε μια διαισθητικότερη αντίληψη της.

Ανεξάρτητα λοιπόν από την επιμέρους υλοποίηση της, η Ουρά, θα έχει δύο διακριτά σημεία πρόσβασης που, συμβατικά, ονομάζονται Tail, το σημείο αφίξεων, όπου εισάγονται τα νεότερα, και Head, το σημείο αναχωρήσεων, από όπου ανακτώνται τα παλαιότερα δεδομένα. (Σχήμα 2.1). Οι ενέργειες, κατά συνέπεια είναι, κυρίως, και πάλι με τα παραδοσιακά τους ονόματα, οι EnQueue(),

[1] αναλυτικότερα : 3.1.1



Σχήμα 2.2: Εισαγωγή στοιχείου στον ΑΤΔ Ουρά, EnQueue ().

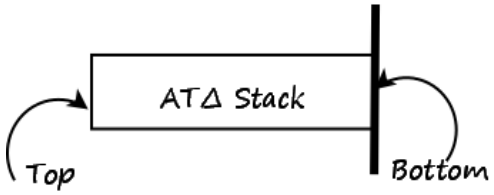


Σχήμα 2.3: Εξαγωγή στοιχείου από τον ΑΤΔ Ουρά, DeQueue ().

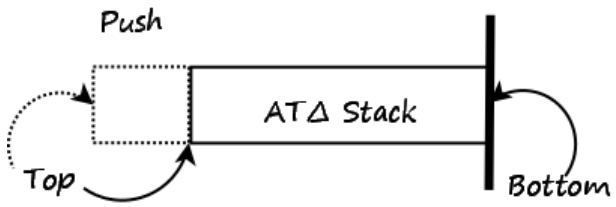
η άφιξη και εισαγωγή, και DeQueue(), η αναχώρηση και διαγραφή, αντίστοιχα. (Σχήματα 2.2 και 2.3). Δευτερεύουσες ενέργειες μπορούν να ανιχνεύσουν την κατάσταση της Ουράς ως προς το πλήθος των στοιχείων, IsEmpty() και IsFull(), να έχουν πρόσβαση στο στοιχείο στην αρχή, το Head , First(), ή το τέλος, το Tail, , Last(), χωρίς μεταβολή της κατάστασης της δομής, και φυσικά, να δημιουργήσουν, Initialize(), ή να διαγράψουν την Ουρά συνολικά, Delete().

2.1.2 Ο ΑΤΔ Στοίβα

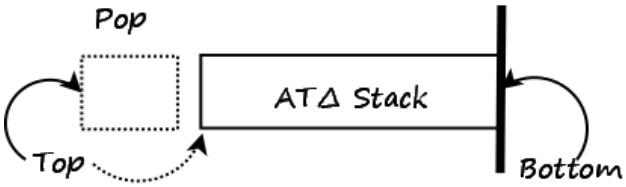
Αντίστοιχα, ο ΑΤΔ Στοίβα, ορίζεται, με την αντίστροφη πολιτική αναχωρήσεων, ως η δομή της οποίας τα δεδομένα ανακτώνται με σειρά νεότητας. Όπως και για την Ουρά, και στη Στοίβα αυτό έχει τόσο προεκτάσεις στον προσανατολισμό της διάρθρωσης, όσο και στις δυνατές ενέργειες που διαθέτει η δομή, όπως επίσης και το σημείο πρόσβασης των δεδομένων. Πλέον, εισάγονται και εξάγονται τα νεότερα των δεδομένων με φθίνουσα, χρονικά, σειρά αφίξεων, και η κατά χρονολογική σειρά ανάκτηση είναι η μόνη επιλογή.



Σχήμα 2.4: Ο ΑΤΔ Στοίβα, με το σημείο πρόσβασής του.



Σχήμα 2.5: Εισαγωγή στοιχείου στον ΑΤΔ Στοίβα, Push ().



Σχήμα 2.6: Εξαγωγή στοιχείου από τον ΑΤΔ Στοίβα, Pop ().

Η Στοίβα μπορεί να προσομοιώσει με μιά πραγματική στοίβα από δίσκους σερβιρίσματος στο φοιτητικό εστιατόριο, ή αντίστροφα, η εικόνα αυτή, με τις ομοιότητες της, να συμβάλλει στην θεώρηση της λειτουργίας της δομής. Κάθε νέο στοιχείο εισάγεται στο ίδιο άκρο από όπου γίνονται και οι ανακτήσεις, ενώ, η πρόσβαση στο άλλο άκρο, ή τα ενδιάμεσα στοιχεία, είναι απαγορευμένη. Το προσβάσιμο άκρο, ονομάζεται Top, και το μη προσβάσιμο, πάνω στο οποίο 'χτίζεται' η δομή, Bottom. (Σχήμα 2.4). Μάλιστα, από την συνήθη παρουσία ενός ελατηρίου, κάτω από τη βάση των πραγματικών διατάξεων που λειτουργούν παρόμοια με τη Στοίβα, προκύπτουν και τα ονόματα των κυριότερων ενεργειών στη δομή : Push() η εισαγωγή ενός νέου στοιχείου στην κορυφή, που αντικαθιστά το προηγούμενο Top, και Pop() η εξαγωγή ενός στοιχείου, πάντα από την κορυφή της Στοίβας, όπου το νέο Top προκύπτει απευθείας, από το αμέσως προηγούμενο στοιχείο. (Σχήματα 2.5 και 2.6). Σαφώς, και οι δευτερεύουσες ενέργειες υπάρχουν, από τις ερωτήσεις για το μέγεθος της δομής, IsEmpty() και IsFull(), μέχρι τις Initialize() και Delete(), για δημιουργία (αρχικοποίηση) και διαγραφή. Φυσικά, η πρόσβαση σε ακραία στοιχεία, χωρίς μεταβολή της δομής, αφορά πλέον το μοναδικό προσπελάσιμο, το Top, με την ομώνυμη λειτουργία Top().

2.2 Θεωρία της Ομοχειρίας

Η ομοχειρία, διεθνώς pipeline, ή, όπως εναλλακτικά συναντάται στη βιβλιογραφία, διοχέτευση, είναι μια τεχνική βελτιστοποίησης υλικού. Ιστορικά, η τεχνική εισήχθηκε στα τέλη του 1950, στη σχεδίαση του IBM 7030, χαϊδευτικά Stretch, με άνθιση της σχετικής ερευνητικής δραστηριότητας τις δεκαετίες 1970 και 1980, οπότε προέκυψε και το μεγαλύτερο μέρος της ορολογίας και των

συμπληρωματικών τεχνικών. Οι επεξεργαστές RISC, Reduced Instruction Set Computers, με την επικέντρωση της σχεδιαστικής μέριμνας στο σύνολο εντολών, και μην παραγνωρίζοντας τον αντίκτυπο του στο ίδιο το υλικό, πέτυχαν σχεδιάσεις προορισμένες για ομοχειρία, όπως του MIPS – αλλά, και για την κυριαρχία και των δύο αυτών τεχνικών, στο σύντομο μέλλον. Πλέον, οι περισσότεροι επεξεργαστές, είτε υλοποιούν εγγενώς ομοχειρία σε ένα κομψό σύνολο εντολών, είτε έμμεσα, μεταφράζοντας τις ήδη υπάρχουσες εντολές σε μικροκώδικα ανάλογο των RISC.

2.2.1 Γενικά χαρακτηριστικά

Η τεχνική της ομοχειρίας επιτυγχάνει βελτιστοποίηση των επιδόσεων του υλικού, εκμεταλλευόμενη την ήδη υπάρχουσα δυνατότητα των εντολών να παραλληλιστούν, ονομαζόμενη παραλληλισμός επιπέδου εντολών, διεθνώς instruction level parallelism, (ILP). Συγκεκριμένα, η κάθε εργασία, μπορεί να αντιστοιχισθεί σε ένα πεπερασμένο σύνολο εντολών, και, με τη σειρά τους, κάθε εντολή σε πεπερασμένο αριθμό απλών ενεργειών, ή βημάτων. Η αντιστοίχιση αυτή αποτελεί μέρος του σχεδιαστικού προβλήματος, και έτσι οι επιλογές που θα γίνουν στο κομμάτι αυτό θα καθορίσουν και την μετέπειτα υλοποίηση, στον βαθύτερό της πυρήνα. Στο βαθμό, λοιπόν, που η εκτελούμενη εργασία το επιτρέπει, οι απλές ενέργειες που απαρτίζουν τις εντολές, όχι μόνο είναι περιορισμένες στο πλήθος, αλλά και στο είδος. Αν το υποσύνολο των απλών βημάτων που αποτελείται κάθε εντολή ήταν τελείως διαφορετικό σε είδος, τότε οι εντολές θα μπορούσαν να παραλληλιστούν πλήρως. Συνήθως όμως, το υποσύνολο βημάτων είναι παραπλήσιο, σε είδος, και πιθανά αρκετά αντίστοιχο και στο πλήθος τους, συνεπώς, τα επιμέρους αυτά βήματα, μπορούν να εκτελεστούν παράλληλα, εξυπηρετώντας μακροσκοπικά και στην παράλληλη εκτέλεση των εντολών.

Για παράδειγμα, εάν θέλει κανείς, να πλύνει τα πιάτα και τα ρούχα του, μπορεί να το πραγματοποιήσει τελείως παράλληλα, αν βέβαια διαθέτει πλυντήριο, για μία τουλάχιστον από αυτές τις λειτουργίες. Αν όμως θέλει να μαγειρέψει ένα φιλόδοξο γεύμα, μπορεί είτε να περιμένει το κάθε επιμέρους τμήμα να ετοιμαστεί, για να ξεκινήσει το επόμενο, και το σύνολό τους, για να πλύνει τα μαγειρικά σκεύη, ή, όσο έχει διαθέσιμα σκεύη και εστίες, να τα παρασκευάζει παράλληλα, και να πλένει, επίσης, όταν δεν απασχολείται με το μαγείρεμα, τα σκεύη, όπως προκύπτουν. Καμία μέθοδος δεν είναι απορριπτέα, ως προς το αποτέλεσμα είναι ισότιμες -ως προς το χρόνο όμως για την ολοκλήρωσή

τους, μπορούν να διαφέρουν αρκετά. Είναι σαφές, επίσης, ότι η δεύτερη προσέγγιση απαιτεί ενδελεχή γνώση της διαδικασίας, και μία επιπλέον προσπάθεια συγχρονισμού. Κάτι ανάλογο, ισχύει και με την ομοχειρία.

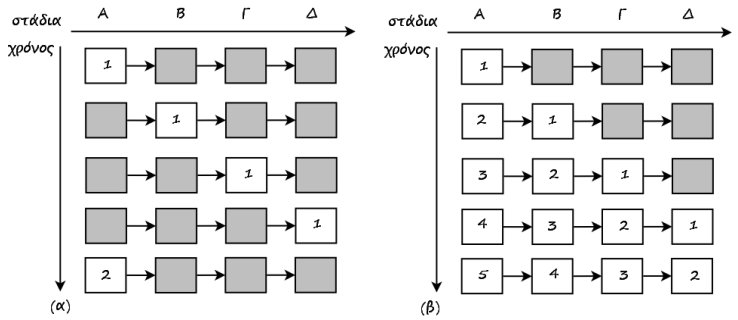
Σύμφωνα με τα παραπάνω, εκτός από την τμηματοποίηση της διαδικασίας, η παραλληλοποίηση των ενεργειών, έχει ως προϋπόθεση την διαφοροποίηση και επάρκεια πόρων, σε αντιστοιχία με τα βήματα, τους οποίους να μπορεί κάθε παράλληλο τμήμα να αξιοποιήσει ανεξάρτητα και ταυτόχρονα με τα υπόλοιπα. Οι πόροι αυτοί αποτελούν τις λειτουργικές μονάδες του υλικού, functional units, και τα βήματα στα οποία κάθε εντολή έχει κατατμηθεί, είναι τα στάδια της ομοχειρίας, pipeline stages. Αποδιδόμενο, κάθε στάδιο, συνήθως, σε μιά διαφορετική μονάδα υλικού, μπορεί να εκτελείται παράλληλα με τα υπόλοιπα.

Δημιουργείται, λοιπόν, ένα υπερσύνολο των βημάτων, από τις, κατά αντιστοιχία, απαιτούμενες λειτουργικές μονάδες, διατεταγμένες σε μία διαδρομή των δεδομένων, το οποίο θα εξυπηρετήσει όλες τις ενδεχόμενες εντολές του ομόχειρου συστήματος, το pipelined datapath. Το πλήθος αυτών των βημάτων αποτελεί το βάθος της ομοχειρίας, pipeline depth. Η κομψότητα της σχεδίασης, η ίδια η φύση της διαδικασίας, αλλά και η εκ των υστέρων γνώση των απαιτήσεων για επιτυχή ομοχειρία, επιβάλλει τα βήματα που απαρτίζουν την κάθε εντολή, να είναι σε μία δεδομένη διάταξη μεταξύ τους, ακόμα και για τις εντολές που κάποια στάδια λείπουν. Συνήθως, συγκεκριμένες δράσεις ακολουθούν άλλες, και χονδρικά, πρέπει πρώτα να ανακτηθούν οι πληροφορίες προς επεξεργασία, να εκτελεστεί η επεξεργασία αυτή και τέλος, αν πρόκειται για σύστημα με εσωτερική κατάσταση, όπως είναι τα περισσότερα με κάποιο νόημα, να αποθηκευθεί κάτι σχετικό με την συγκεκριμένη εντολή, που είχε αντίκτυπο στην εσωτερική αυτή κατάσταση. Φαίνεται αρχικά, λοιπόν, σαν να μην είναι και τόσο εφικτό να παραλληλιστούν τα βήματα, αν ακολουθούν την απολύτως ίδια σειρά. Πράγματι· αλλά, καθώς λένε, η ευγένεια αποδίδει: αν, αντί να εκκινηθούν ταυτόχρονα τόσες εντολές όσες τα στάδια της ομοχειρίας, που θα θέλουν όλες να χρησιμοποιήσουν την ίδια λειτουργική μονάδα, κάνοντας την σχεδίαση ως τώρα αποτυχημένη, ξεκινώντας μία μία τις εντολές, δίνεται το προβάδισμα στην κάθε προηγούμενη, ώστε να καταλαμβάνει ένα στάδιο βαθύτερα στην ομοχειρία, και έτσι, αισίως, κάθε εντολή να απασχολεί ένα διαφορετικό, μετά από το γέμισμα της αλυσίδας της ομοχειρίας.

2.2.2 Επίδραση στην επίδοση

Η αρχική κατάτμηση της εργασίας σε εντολές, και της κάθε εντολής σε επιμέρους βήματα, είναι απαραίτητη για την σχεδίαση ενός συστήματος, ανεξάρτητα από την εκμετάλλευση στη συνέχεια της πιθανής δυνατότητας παραλληλισμού. Το datapath, που προκύπτει, σε γενικές γραμμές, είναι αντίστοιχο και για τις δύο εκδοχές, είτε με ομοχειρία, είτε χωρίς, εκτός συγκεκριμένων δομών που απαιτούνται για την μετατροπή σε ομόχειρο σύστημα, και δεν αφορούν την εργασία του συστήματος καθεαυτή.

Στην εκδοχή, του ίδιου συστήματος, χωρίς ομοχειρία, (Σχήμα 2.7.α), κάθε



Σχήμα 2.7: Γενικευμένο μονοπάτι δεδομένων, ενεργό (α) χωρίς ομοχειρία και (β) με ομοχειρία.

νέα εντολή θα εκκινείται μετά την διέλευση από όλο το μονοπάτι των δεδομένων και ολοκλήρωση της προηγούμενής της. Στο χρόνο, αυτό μπορεί να αποτελεί έναν κύκλο ρολογιού, που να καταλαμβάνει τη διέλευση από άκρη σε άκρη, είτε πολλούς κύκλους, αν σε κάθε στάδιο του μονοπατιού αντιστοιχισθεί ένας κύκλος. Έτσι, μία εντολή στο απλό σύστημα εισάγεται ανά ένα, μεγάλο σε διάρκεια, κύκλο, είτε μετά από έναν αριθμό μικρότερων. Αντίστοιχα, σε κάθε, μεγάλο, κύκλο ολοκληρώνεται μία εντολή, στο άλλο 'άκρο' του μονοπατιού, ή μετά από τον καθορισμένο αριθμό, για την εκδοχή των πολλαπλών κύκλων.

Το ίδιο σύστημα, με ομοχειρία πλέον, (Σχήμα 2.7.β), θα εκτελεί πολλές εντολές, όσες και τα στάδια, ή το βάθος του, παράλληλα, εισάγοντας μία εντολή σε κάθε στάδιο ανά κύκλο, που είναι ο χρόνος που χρειάζεται για να ολοκληρώσει την επεξεργασία του το στάδιο. Μάλλον, επειδή όλα τα στάδια θα είναι ενεργά ταυτόχρονα, η διάρκεια του κύκλου θα αντιστοιχεί στο χρόνο που απαιτεί το αργότερο στάδιο για την ολοκλήρωσή του. Η χρονική εξισορρόπηση και αντιστοιχία των σταδίων, είναι, βέβαια, μέρος και της σχεδιαστικής διαδικασίας. Πάντως, κάθε εντολή πλέον εισάγεται ανά ένα κύκλο, και παρότι η καθεμία χρειάζεται τόσους κύκλους να ολοκληρωθεί όσο και το βάθος του μονοπατιού, στο άλλο 'άκρο' του μονοπατιού ολοκληρώνεται μία νέα εντολή, κάθε κύκλο.

Ιδανικά λοιπόν, φαίνεται πως η χρήση της ομοχειρίας, επιταχύνει το σύστημα κατά τόσες φορές όσο και το βάθος (έστω Β) της ομοχειρίας. Αν πρόκειται για το σύστημα ενός κύκλου, μία εντολή, και στις δύο εκδοχές, ολοκληρώνεται κάθε ένα κύκλο, αλλά ο κύκλος του απλού συστήματος έχει υποδιαιρεθεί κατά Β στο ομόχειρο, ενώ στην περίπτωση των πολλαπλών κύκλων, η διάρκεια του κύκλου, έστω, είναι ίδια, αλλά η ολοκλήρωση εντολών γίνεται συχνότερα, κατά τον παράγοντα Β, ανά έναν, αντί ανά Β κύκλους.

Βέβαια, τα πράγματα δεν είναι ποτέ ιδανικά, καθώς κάθε τι έχει και το ανάλογο κόστος. Καταρχάς, για να πραγματοποιηθεί η κατάτμηση του μονοπατιού δεδομένων στο χρόνο, και κάθε βήμα να εκτελείται σε ένα κύκλο, τόσο στην έκδοση του απλού συστήματος, όσο και του ομόχειρου, χρησιμοποιούν-

ται καταχωρητές, στοιχεία μνήμης που ενεργοποιούνται με τις μεταβάσεις των τιμών του ρολογιού, δημιουργώντας στην ενδιάμεση διάρκεια, χρονικά στεγανά. Οι απαιτήσεις των καταχωρητών για σωστή λειτουργία, αφορούν τόσο τη διατήρηση των σημάτων εισόδου τους σταθερά, πριν και μετά την μετάβαση του ρολογιού, διάρκεια των setup και hold, αλλά και μακροσκοπικά, ως προς το σύστημα στο σύνολό του, και οι μεταβάσεις του ρολογιού να γίνονται ταυτόχρονα σε κάθε τέτοιο στοιχείο, το οποίο δεν συμβαίνει λόγω απόκλισης, clock skew. Και τα δύο παραπάνω στοιχεία, προσθέτουν καθυστερήσεις σε κάθε κύκλο, και έτσι στην πραγματικότητα, ο αρχικός ενιαίος κύκλος είναι μικρότερος από τους B κύκλους του ομόχειρου συστήματος.

Σαν να μην ήταν αυτό αρκετό, ο παραλληλισμός με τις εγγενείς του απαιτήσεις, μειώνει, εν δυνάμει την βελτίωση των επιδόσεων περαιτέρω. Για να είναι εφικτός, όπως προαναφέρθηκε, είναι απαραίτητα τόσο μια επάρκεια στους πόρους, όσο και η ανεξαρτησία των επιμέρους ενεργειών. Αν αυτές δεν ικανοποιούνται σχεδιαστικά, είτε δεν αντιμετωπίζονται, καθώς σε ένα βαθμό είναι αναπόφευκτες, συνεισφέρουν με επιπλέον καθυστερήσεις, και μειώνουν την ιδανική επίδοση.

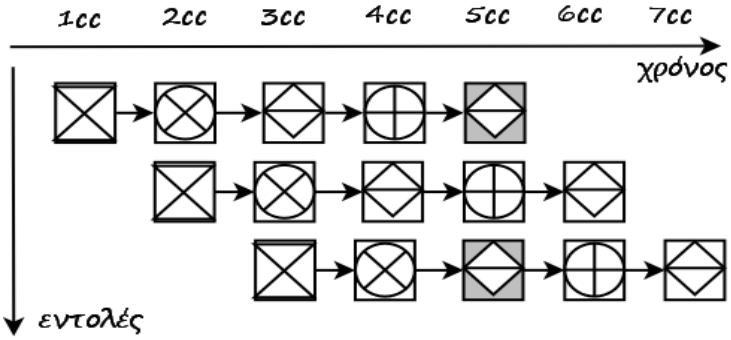
2.2.3 Κόστος βελτιστοποίησης

Εκτός, λοιπόν, από τους καταχωρητές της ομοχειρίας, το επιπλέον υλικό που απαιτείται για την ανάθεση του στα διαφορετικά στάδια, αλλά και τις ενισχυμένες επιδόσεις κάποιων μονάδων, όπως για παράδειγμα η απαίτηση για πολλαπλασιασμό του εύρους ζώνης των μνημών, ως προϋποθέσεις για τον παραλληλισμό των βημάτων, υπάρχει και επιπλέον κόστος σε υλικό, που ανακύπτει ως αποτέλεσμα του παραλληλισμού, αλλά και συνθήκες τέτοιες, που έχουν αρνητικό αντίκτυπο στις επιδόσεις. Το επιπλέον υλικό είναι αναγκαίο για την ανίχνευση αρχικά και στη συνέχεια την αντιμετώπιση αυτών των ανωμαλιών λόγω της ομοχειρίας, και αν δεν είναι εφικτή η αντιμετώπισή τους, τότε η έσχατη λύση είναι η προσωρινή αναίρεση του ίδιου του παραλληλισμού, εισάγοντας καθυστερήσεις στις εντολές, κατά έναν ή παραπάνω κύκλους, που ονομάζονται ανασχέσεις, stalls, ή, αναφορικά με τον τρόπο που διασχίζουν το datapath χωρίς να έχουν αποτέλεσμα, φυσαλίδες, bubbles.

Συγκεκριμένα, οι συνθήκες εκείνες που προκύπτουν από την ομοχειρία, ονομαζόμενες εξαρτήσεις, dependences, και εν δυνάμει, όταν πλέον αποτελούν κινδύνους, hazards, απαγορεύουν την σωστή εκτέλεση των εντολών αν δεν αντιμετωπιστούν, ανάγονται σε τρεις κατηγορίες:

δομικές , και αφορούν τους πόρους, ή τις λειτουργικές μονάδες που έχουν ανατεθεί για την εξυπηρέτηση των σταδίων του datapath.

δεδομένων , αφορούν τα δεδομένα τα οποία επεξεργάζονται από τις εντολές, τα οποία κατά περίπτωση αποτελούν εξόδους, αλλά και εισόδους αυτής της επεξεργασίας.



Σχήμα 2.8: Δομικός κίνδυνος.

ελέγχου , αφορούν τις εντολές που εισάγονται στο ομόχειρο datapath, όταν, εκτός των άλλων, υπάρχουν και εντολές που αλλάζουν τη ροή, επιλέγοντας μεταξύ εναλλακτικών μονοπατιών εκτέλεσης.

Οι δομικές εξαρτήσεις, είναι ουσιαστικά συγκρούσεις πόρων, που αποτελούν κινδύνους μόνο όταν έχει επιτραπεί σχεδιαστικά, πιθανά λόγω κόστους υλικού, η ανεπάρκεια των λειτουργικών μονάδων ώστε να μην είναι εφικτή η ταυτόχρονη εκτέλεση όλων των συνδυασμών των εντολών. (Σχήμα 2.8) Φυσικά, όταν οι επίμαχοι συνδυασμοί προκύπτουν, είναι συνηθισμένο να σταματάει η εκτέλεση, και να προκαλούνται ανασχέσεις έως ότου να είναι διαθέσιμοι οι πόροι.

Οι εξαρτήσεις δεδομένων, είναι εγγενής ιδιότητα των προγραμμάτων, καθώς είναι λογικό κάποιο δεδομένο που υπολογίζεται ή αποθηκεύεται να χρησιμοποιηθεί και στη συνέχεια ενός προγράμματος. Οι εξαρτήσεις αυτές, μετατρέπονται σε κινδύνους όταν οι εντολές που τις περιλαμβάνουν είναι αρκετά κοντά ώστε η υλοποίηση του μονοπατιού της ομοχειρίας να αλλάζει την σειρά προσβάσεων, αναγνώσεων ή εγγραφών, από την κανονική σειρά του προγράμματος. Οι εξαρτήσεις αυτές, ανάλογα με τις εμπλεκόμενες προσβάσεις, και την μεταξύ τους συσχέτιση, κατηγοριοποιούνται περαιτέρω, και αντιμετωπίζονται με τα κατάλληλα μέτρα. Οι επιμέρους εξαρτήσεις, και οι αντίστοιχοι κίνδυνοι, ονομασμένοι σύμφωνα με την σωστή σειρά εκτέλεσης, μπορεί να είναι:

- Ανάγνωση μετά την εγγραφή, Read after Write, RAW, όταν μία νεότερη εντολή προσπαθεί να διαβάσει κάποιο δεδομένο που παράγεται από μία προηγούμενη, αλλά δεν έχει ακόμα γραφεί, άρα, παίρνει λανθασμένη τιμή. Πρόκειται για πραγματική εξάρτηση, και για να αντιμετωπισθεί πρέπει να διατηρηθεί η σωστή σειρά των προσβάσεων.

- Εγγραφή μετά την εγγραφή, Write after Write, WAW, όταν δύο εντολές πρόκειται να γράψουν σε μία συγκεκριμένη θέση δεδομένων, συνήθως καταχωρητή, αλλά δεν αποκλείεται και η μνήμη. Αν η νεότερη γράψει πριν από την παλιότερη, θα αποδοθεί τελικά λανθασμένη τιμή, που δεν

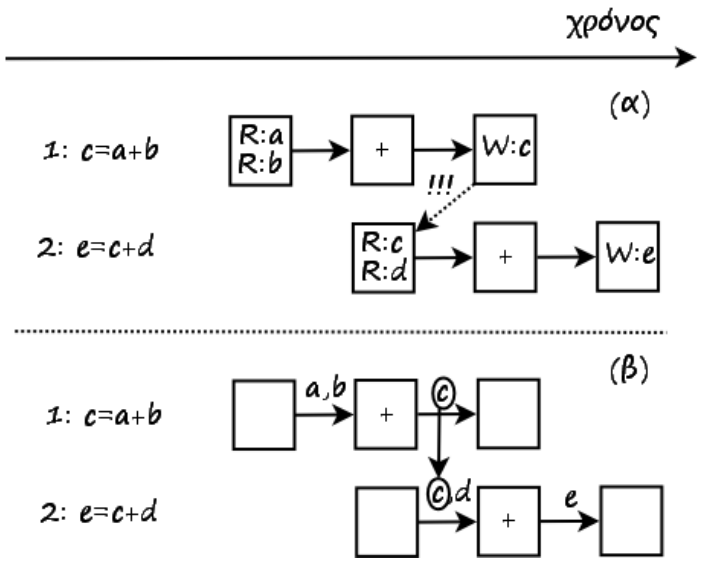
ισχύει πλέον. Ονομάζεται και εξάρτηση εξόδου, output dependence, και προκύπτει όταν αποτελέσματα γράφονται και σε βαθμίδες άλλες, εκτός της τελευταίας. Ουσιαστικά, δεν υπάρχει ανταλλαγή δεδομένων ανάμεσα στις δύο εντολές.

- Εγγραφή μετά την ανάγνωση, Write after Read, WAR, όταν μία νεότερη εντολή γράφει μία θέση δεδομένων προτού διαβαστεί από την παλιότερη, με αποτέλεσμα να διαβάσει λάθος δεδομένα. Και στην περίπτωση αυτή, που ονομάζεται και αντιεξάρτηση, antidependence, δεν υπάρχει, πραγματική, ανταλλαγή δεδομένων μεταξύ των εμπλεκόμενων εντολών.

Οι εξαρτήσεις, όταν αναγνωρίζονται ως κίνδυνοι πρέπει να αντιμετωπιστούν, για να διατηρηθεί η ορθότητα του προγράμματος. Γενικά, δύο είναι οι βασικές προσεγγίσεις για την αντιμετώπισή τους: είτε να διατηρηθεί η εξάρτηση, αλλά να αποφευχθεί ο κίνδυνος, είτε να εξαλειφθεί η εξάρτηση με μετασχηματισμό του κώδικα. Οι δύο τελευταίες εξαρτήσεις, που δεν αφορούν άμεση ανταλλαγή δεδομένων μεταξύ των εντολών, χαρακτηρίζονται και ως εξαρτήσεις ονόματος, name dependences, καθώς προκύπτουν από την αναφορά των εντολών σε μία ίδια θέση αποθήκευσης δεδομένων, που όμως δεν τις συσχετίζει άμεσα, αλλά έμμεσα. Αν λοιπόν αυτή η συσχέτιση λόγω ονόματος αναιρεθεί, αναιρείται και η εξάρτηση, συνεπώς οι κίνδυνοι αυτοί αντιμετωπίζονται με μετονομασία των θέσεων στις οποίες αναφέρονται, και αυτό μπορεί να γίνει είτε στατικά, κατά την μεταγλώττιση, είτε δυναμικά, κατά την εκτέλεση, από το υλικό.

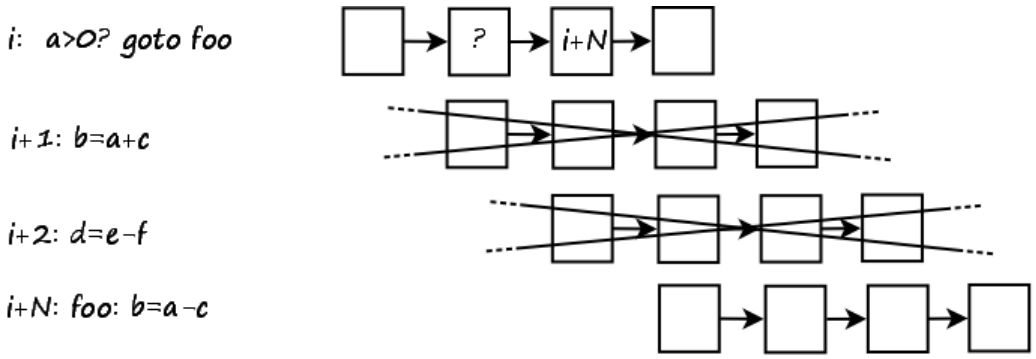
Το σπουδαιότερο είδος εξάρτησης δεδομένων, οι λεγόμενες πραγματικές εξαρτήσεις, αντιμετωπίζονται, σε πρώτο επίπεδο, αποφεύγοντας τον κίνδυνο, είτε με την τεχνική της προώθησης, ή προσπέρασης, των δεδομένων, διεθνώς forwarding, ή bypassing, αλλά και short-circuiting, αν το σημείο όπου υπολογίζονται προηγείται χρονικά εκείνο που θα καταναλωθούν, χωρίς να προηγείται προφανώς και η επίσημη καταγραφή στους ορατούς πόρους, είτε, αν δεν είναι χρονικά εφικτή η προώθηση, εισάγοντας ανασχέσεις. (Σχήμα 2.9) Βέβαια, οι ανασχέσεις είναι κάτι που με την σειρά του πρέπει να αποφευχθεί, καθώς μειώνει τις επιδόσεις της ομοχειρίας. Έτσι, χρησιμοποιούνται και προχωρημένες τεχνικές, τόσο υλικού, όσο και λογισμικού. Αναφορικά, ο δυναμικός χρονοπρογραμματισμός με πίνακα αποτελεσμάτων, scoreboard, ο δυναμικός χρονοπρογραμματισμός με μετονομασία καταχωρητών στον αλγόριθμο του Tomasulo, η κατ΄ εικασία εκτέλεση, η δυναμική αποσαφήνιση αναφορών μνήμης, στο επίπεδο του υλικού. Στο επίπεδο του λογισμικού, με ανάλυση εξαρτήσεων, βασικό χρονοπρογραμματισμό, χρονοπρογραμματισμό ίχνους και υποστήριξη της κατ΄ εικασία εκτέλεσης από τον μεταγλωττιστή. (Αναλυτικά, [5]).

Οι εξαρτήσεις ελέγχου, προκύπτουν στα ομόχειρα συστήματα που υλοποιούν σύνολα εντολών που περιλαμβάνουν εντολές για αλλαγή της ροής, είτε κατά συνθήκη, είτε χωρίς συνθήκη, τις ονομαζόμενες διακλαδώσεις. Και στις δύο περιπτώσεις, απαιτείται, συνήθως, ο υπολογισμός της διεύθυνσης προορισμού, ενώ πιθανά, και η εκτίμηση μίας συνθήκης, σε ένα υποσύνολο της



Σχήμα 2.9: (α) Κίνδυνος δεδομένων, λόγω αλλαγής του συσχετισμού εγγραφών και αναγνώσεων. (β) Αντιμετώπιση του κινδύνου, με προώθηση από τις εσωτερικές δομές.

πρώτης περίπτωσης, για να αποφασιστεί τελικά ποιά θα είναι η έγκυρη επόμενη διεύθυνση. Μέχρι τα προαπαιτούμενα αυτά μεγέθη, για την ολοκλήρωση της εντολής, να υπολογισθούν, και ανάλογα με τη διαμόρφωση του μονοπατιού της ομοχειρίας, είναι πιθανό να έχουν εισαχθεί στο datapath και νέες εντολές, οι οποίες τελικά να είναι άκυρες. (Σχήμα 2.10) Το βασικότερο, φυσικά, είναι να μη μεταβληθεί η εσωτερική κατάσταση του συστήματος από κάποια εντολή που δεν ισχύει, που όμως έχει εισέλθει και προχωρά στην ομοχειρία. Μία πρώτη προσέγγιση, λοιπόν, θα ήταν μέχρι να αποφασιστεί η επόμενη εντολή της διακλάδωσης, να μην εισάγονται οι επόμενες εντολές στο datapath, η οποία όμως θα είχε πολύ αρνητικές επιπτώσεις στην συνολική επίδοση, καθώς οι εντολές διακλάδωσης είναι αναπόσπαστο κομμάτι των γενικών συνόλων εντολών και των πραγματικών προγραμμάτων, μιάς και εκφράζουν την δυναμική διαμόρφωση του κώδικα, σε χρόνο εκτέλεσης. Για την βελτίωση των επιδόσεων, χρησιμοποιούνται απλές επεκτάσεις της λογικής του υλικού, προσέχοντας βέβαια να μην μεταβληθεί η εσωτερική κατάσταση του συστήματος, αποφεύγοντας τις ανασχέσεις και το άδειασμα της ομοχειρίας. Στη βασική τους μορφή μπορούμε να θεωρήσουμε όλες τις διακλαδώσεις ανεπιτυχείς, ή, αν είναι ήδη γνωστή η διεύθυνση προορισμού, επιτυχείς, και να εισάγονται οι αμέσως επόμενες εντολές, ή οι εντολές από τον προορισμό της διακλάδωσης, και αν η κατεύθυνση της διακλάδωσης τελικά είναι διαφορετική, οι εντολές που εισήχθησαν να ακυρωθούν. Εναλλακτικά, και με τη βοήθεια του μεταφραστή, αντί να εισαχθούν οι επόμενες, αναφορικά με οποιοδήποτε σημείο εκτέλεσης, οι εντολές που θα γεμίσουν το datapath έως ότου υπολογιστεί η διακλάδωση,



Σχήμα 2.10: Κίνδυνος ελέγχου : επιτυχής διακλάδωση, προκαλεί ανασχέσεις.

να είναι χρήσιμες εντολές, που θα εκτελεστούν κανονικά και δεν θα ακυρωθούν, το οποίο ονομάζεται καθυστερημένη διακλάδωση, delayed branch. Πιο προχωρημένες τεχνικές, περιλαμβάνουν τη δυναμική πρόβλεψη διακλαδώσεων και εκτέλεση κατ΄ εικασία στο υλικό, στατική πρόβλεψη διακλαδώσεων και αναδίπλωση βρόγχου στο λογισμικό.

Ένα ακόμα ευαίσθητο ζήτημα που προκύπτει από την αναδιάταξη των βημάτων, αλλά και των ίδιων των εντολών σε προχωρημένες τεχνικές, είναι η διατήρηση των εξαιρέσεων, interrupts ή exceptions, που είναι μία εξειδικευμένη μορφή κινδύνων ελέγχου, καθώς, αν η εξαίρεση μπορεί να αντιμετωπισθεί χωρίς τον τερματισμό της εκτέλεσης, δεν πρόκειται για κάτι διαφορετικό από την κλήση μίας διεργασίας, αυτής του χειρισμού εξαιρέσεων, που περιλαμβάνει, εκτός από την αποθήκευση των κατάλληλων διευθύνσεων και δεδομένων, την αλλαγή της ροής των εντολών. Το ζητούμενο είναι, αφενός, να παράγονται όσες εξαιρέσεις θα προέκυπταν σε μία σειριακή εκτέλεση των εντολών, ούτε περισσότερες, ούτε και λιγότερες, και αφετέρου, όσες προκύψουν από εντολές, να εξυπηρετούνται με εκείνη, την πραγματική σειρά του προγράμματος, και όχι με την χρονική σειρά, που στο ομόχειρο datapath ενδεχομένως διαφέρει.

Συνοπτικά, η ομοχειρία επιταχύνει την εκτέλεση, εκμεταλλευόμενη την δυνατότητα των εντολών να παραλληλισθούν, όχι μειώνοντας το χρόνο που απαιτείται για να ολοκληρωθεί μία εντολή, που ονομάζεται λανθάνων χρόνος, latency, αλλά το χρόνο που απαιτείται για να εκκινηθεί μία νέα εντολή, όπως αντίστοιχα, το χρόνο μεταξύ των εντολών που ολοκληρώνονται. Έτσι, ουσιαστικά, αυτό που βελτιώνεται είναι ο ρυθμός ολοκλήρωσης των εντολών, διεθνώς throughput. Ιδανικά, η επιτάχυνση αυτή θα έφτανε να επιταχύνει το σύστημα κατά ένα παράγοντα ίσο με το βάθος της ομοχειρίας, όμως αυτό το όριο δεν είναι εφικτό, καθώς προκύπτουν εξαρτήσεις και κίνδυνοι, που εισάγουν ανασχέσεις, μην επιτρέποντας τον πλήρη παραλληλισμό. Παρόλα αυτά, το μέγιστο του παραλληλισμού προσεγγίζεται με επιπλέον αντισταθμιστικές, των κινδύνων, προσεγγίσεις, τόσο σε υλικό, όσο και λογισμικό.

Κεφάλαιο 3

Υλοποίηση

Στο κεφάλαιο αυτό παρουσιάζεται η βασική σχεδίαση του υλοποιηθέντος συστήματος, παραλείποντας τις λεπτομέρειες της υλοποίησης, για τα αντίστοιχα κεφάλαια 5 και 4. Αρχικά, δίνεται η σειρά βημάτων που οδηγούν στην σύνθεση του κυρίως μονοπατιού δεδομένων, το οποίο, στη συνέχεια, μετατρέπεται σε ομόχειρο. Από αυτή τη μετατροπή, προκύπτουν τα χαρακτηριστικά ζητήματα, των συγκρούσεων πόρων και των εξαρτήσεων δεδομένων, καθώς και επανεξετάζεται το ήδη υπάρχον, της διαχείρισης της μνήμης. Ακολούθως, αναπτύσσεται η αντιμετώπισή τους.

3.1 Η Ουρά στην πράξη

Για την αποτύπωση του ζητούμενου συστήματος σε υλικό, είναι απαραίτητη η ενδελεχής ανάλυση του συνόλου των ενεργειών που απαρτίζουν τη συμπεριφορά της δομής της Ουράς. Αναγνωρίζοντας τις επιμέρους δράσεις (βήματα) της καθεμίας των ενεργειών, ως προς τις λειτουργικές τους απαιτήσεις, αλλά και τη χρονική τους ακολουθία, προκύπτουν τόσο οι δομικές μονάδες, του συστήματος συνολικά, όσο και η οργάνωσή τους στο μονοπάτι των δεδομένων. Το μονοπάτι που προκύπτει, με αυτόν τον τρόπο, είναι μη επικαλυπτόμενο, σε μορφή εκτέλεσης ενός, ή πολλαπλών, κύκλων. Συνεπώς, στην πρώτη αυτή 'μετάφραση' του συστήματος, δεν αντιμετωπίζεται κανένα από τα ζητήματα που προκύπτουν από την επικάλυψη, παρότι η προοπτική της εξυπηρέτηση της ομοχειρίας βρίσκεται πάντα στο πίσω μέρος της σκέψης.

3.1.1 Χώρος και χρόνος

Οι δύο συνιστώσες που αποτελούν την έννοια της απόδοσης της οποιασδήποτε επεξεργασίας, είναι ο χρόνος και ο χώρος μνήμης που καταναλώνεται. Οι ποσότητες του καθενός χαρακτηριστικού, που προκύπτουν είτε από αναλυτικές μεθόδους, είτε από μετρήσεις, ονομάζονται πολυπλοκότητα, χρονική και χωρική, αντίστοιχα, και αποτελούν κριτήρια βάσει των οποίων διαμορφώ-

νονται σχεδιαστικές αποφάσεις, σύμφωνα πάντα με τους περιορισμούς και τις ιδιαιτερότητες κάθε προβλήματος.[1]

Στο ίδιο νοηματικό πλαίσιο υπάρχουν και οι έννοιες της χρονικής και χωρικής τοπικότητας, που αντιπροσωπεύουν, αφενός, την ιδιότητα των προγραμμάτων να αναφέρονται σε θέσεις μνήμης που πρόσφατα χρησιμοποιήθηκαν, και αφετέρου, οι θέσεις μνήμης στις επόμενες αναφορές, να είναι κοντινές με εκείνες των προηγούμενων αναφορών.[7] Οι ιδιότητες αυτές είναι εγγενείς, μεν, στα προγράμματα, αλλά από πλευράς υλικού, η ύπαρξή τους οδηγεί, αλλά και απαιτεί, δομές και λειτουργίες που να επιτρέπουν την καλύτερη εξυπηρέτηση, ως και την εκμετάλλευσή τους, προς όφελος της απόδοσης. Συνεπώς, η αναγνώριση και αξιοποίησή τους, αποτελεί επιπλέον σχεδιαστική παράμετρο. Στην πράξη, οι συγκεκριμένες δομές δεδομένων, Ουρά και Στοίβα, είναι από τις προεξέχουσες στην αναπαράσταση, αλλά και ενίσχυση, της τοπικότητας σε πληθώρα προβλημάτων.

3.1.2 Εναλλακτικές υλοποιήσεις

Από τον ορισμό της Ουράς ως ΑΤΔ, προκύπτουν, όπως προαναφέρθηκε, οι βασικές της λειτουργίες και συμπεριφορά. Μπορεί ο ορισμός να μην επιβάλλει άμεσα κάποια μορφή υλοποίησης, αλλά σε συνδυασμό με τις εγγενείς ιδιότητες των δομικών μονάδων και στοιχειωδών ενεργειών, καθώς και τις απαιτήσεις κάθε εφαρμογής, τελικά, οι επιλογές καθοδηγούνται προς συγκεκριμένες λύσεις.

Είναι λογικό, βάσει της απλότητας της, η Ουρά να εξυπηρετείται ακόμα και από μια, σχετική, ανυπαρξία δομής, και να μπορεί να υλοποιηθεί χρησιμοποιώντας ένα δεσμευμένο σύνολο παρακείμενων θέσεων μνήμης, έναν πίνακα. Πράγματι, η εγγενής διευθυνσιοδότηση της μνήμης θα παρέχει τη σύνδεση μεταξύ του συνόλου των στοιχείων, και η αναφορά σε ένα επόμενο ή προηγούμενο, πραγματοποιείται άμεσα, με απλή αριθμητική δεικτών, προσθέτοντας ή αφαιρώντας μία μονάδα, από την τρέχουσα θέση. Η πρώτη διεύθυνση του πίνακα θα αναπαριστά το Head, και κάποια μεγαλύτερή της, όταν έχουν εισαχθεί στοιχεία, το Tail. Από άποψη χωρικής πολυπλοκότητας, πρόκειται για την οικονομικότερη εκδοχή υλοποίησης, καθώς, εκτός από το δεσμευμένο χώρο στοιχείων, ο οποίος είναι σταθερός (αλλά όχι απαραίτητα), δεν χρειάζεται επιπλέον χώρος δεικτών για τη συσχέτιση των στοιχείων.

Η υλοποίηση αυτή, απαιτεί σταθερό χρόνο, $\Theta(1)$, για κάθε εισαγωγή, αλλά γραμμικό, $\Theta(n)$, για κάθε εξαγωγή, ολισθαίνοντας, κάθε φορά, όλα τα στοιχεία κατά μία θέση, για να είναι πάντα το Head στην πρώτη. Χαλαρώνοντας αυτήν την απαίτηση, και επιτρέποντας το Head να κινείται κατά τις εξαγωγές στοιχείων, η χρονική πολυπλοκότητα της εξαγωγής μπορεί να μειωθεί σε σταθερό χρόνο, για κάποιον αριθμό ενεργειών, μέχρις ότου το Tail φτάσει τη μέγιστη επιτρεπόμενη διεύθυνση, και τότε, αν υπάρχει χώρος στην αρχή, επανέρχεται η ολίσθηση, ώστε το Head να βρεθεί στην πρώτη θέση. Και αυτό όμως, μπορεί να βελτιωθεί περαιτέρω, επιτυγχάνοντας σταθερή πολυπλοκότητα σε κάθε ε-

ξαγωγή, αν ο χώρος διευθύνσεων θεωρηθεί κυκλικός, αντί για γραμμικός, και αγνοηθεί το όριο της μικρότερης διεύθυνσης. Διατηρώντας την φορά της αύξησης, από μικρότερες σε μεγαλύτερες διευθύνσεις, και με μικρές παραλλαγές στον έλεγχο, μπορεί να διατίθεται μία θέση λιγότερη, ή και όλες, για πράξεις σταθερού, πλέον, χρόνου.

Υπό συγκεκριμένη οπτική, η απλότητα αυτής της εκδοχής, μετατρέπεται σε μειονέκτημα. Στην γενική περίπτωση, κυρίως στο λογισμικό, η δέσμευση συγκεκριμένου χώρου δεν θεωρείται αποδοτική, εκτός αν, ίσως, είναι το συνολικό μέγεθος γνωστό ή προβλέψιμο, και χωρίς ενδεχόμενο μεταβολής του. Ο χώρος, σε κάθε άλλη περίπτωση, και με την ανάλογη επιβάρυνση της χρονικής, αλλά και της χωρικής πολυπλοκότητας σε άλλο επίπεδο, δεσμεύεται δυναμικά. Χωρικά, το μέγεθος του κάθε στοιχείου αυξάνεται, αποθηκεύοντας την επιπλέον πληροφορία συσχέτισης του με τα υπόλοιπα, στη μορφή μιάς, ή και περισσότερων διευθύνσεων (δεικτών). Οπότε, η υλοποίηση της Ουράς γίνεται βάσει συνδεδεμένης λίστας, ενώ οι λειτουργικές της απαιτήσεις μεταφράζονται σε περιορισμούς προσπέλασής της. Χρονικά, παρόλο που κάθε ενέργεια καταναλώνει περισσότερες εντολές από ότι η εκδοχή του πίνακα, συμπεριλαμβάνοντας την δέσμευση ή απελευθέρωση μνήμης, αλλά και την σύνδεση των στοιχείων, θεωρείται σταθερής πολυπλοκότητας, τόσο για εισαγωγή, όσο και εξαγωγή στοιχείου.

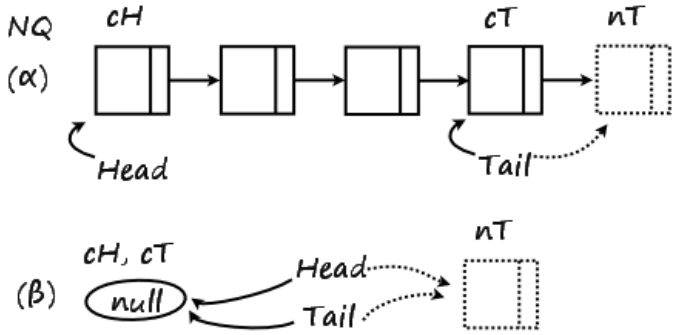
Στο υλικό, από την άλλη, όταν αναφερόμαστε σε μνήμη, δεν πρόκειται για κάποιον χώρο αορίστου μεγέθους, αλλά για κάποιο πολύ σαφών και γνωστών διαστάσεων στοιχείο. Έτσι και στο συγκεκριμένο σύστημα, η δυναμικότητα δεν εκφράζεται με δέσμευση χώρου ανάλογα με τις απαιτήσεις της εφαρμογής, γιατί ο χώρος μνήμης είναι σταθερός – το οποίο δεν αντιφάσκει, σε πρώτο επίπεδο, με την υλοποίηση με χρήση πίνακα. Αντίθετα όμως, ο χώρος διαμοιράζεται σε ένα ορισμένο πλήθος ουρών,[1] καθεμία δυναμικού μεγέθους, το οποίο μεταβάλλεται κατά την εκτέλεση. Ο μόνος περιορισμός βρίσκεται στο μέγεθος των ουρών αθροιστικά, που αντιστοιχεί στην διαθέσιμη μνήμη. Συνεπώς, η υλοποίηση με συνδεδεμένη λίστα είναι η καταλληλότερη για την συγκεκριμένη σχεδίαση.

3.1.3 Μέσες, ακραίες, αδύνατες περιπτώσεις

Αναλυτικότερα, μία ουρά αναπαρίσταται από δύο δείκτες (pointers), που αντιστοιχούν στο πρώτο στοιχείο, το Head, και στο τελευταίο, το Tail. Ανάμεσά τους μπορεί και να υπάρχουν και άλλα στοιχεία, οπότε και οι δύο αυτοί pointers έχουν διαφορετική τιμή, είτε να υπάρχει ένα μόνο στοιχείο στην ουρά, οπότε και είναι ισότιμοι μεταξύ τους. Αν η ουρά είναι άδεια, τότε έχουν τιμή μία σταθερά του κενού, που τυπικά συμβολίζεται με μηδέν, null const(ant).

Όσον αφορά τις ενέργειες επί της ουράς, όπως προαναφέρθηκε, είναι ουσιαστικά δύο : η εισαγωγή, EnQueue(), εδώ για συντομία NQ, και η εξαγωγή

[1] κάτι που θα ήταν εύκολα εφαρμόσιμο και με πίνακες, αν το μέγεθος της κάθε ουράς ήταν προκαθορισμένο και σταθερό.



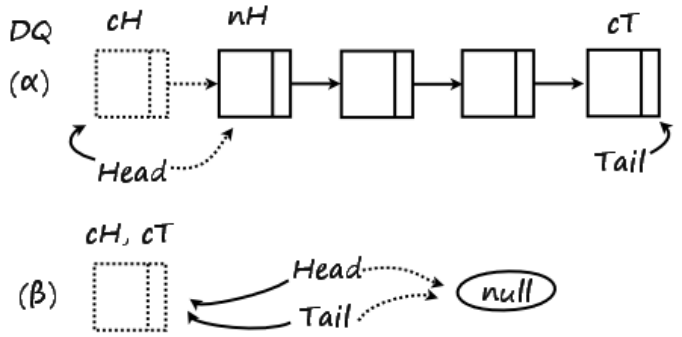
Σχήμα 3.1: (α) Μέση και (β) ακραία περίπτωση NQ.

στοιχείου, DeQueue(), για συντομία DQ. Γενικά, κατά μία εκτέλεση NQ, μία νέα θέση δεσμεύεται, στην οποία αποθηκεύονται τα δεδομένα εισόδου, και εισάγεται στο τέλος της ουράς, κάνοντας το δείκτη επόμενου, Next, του τρέχοντος τελευταίου στοιχείου, του currentTail, να δείχνει στο νέο στοιχείο, το οποίο θα είναι και το νέο Tail της ουράς, newTail, ενημερώνοντας και τον αντίστοιχο Tail pointer της δομής. Αντίστοιχα, κατά μία εκτέλεση DQ, ανακτάται το στοιχείο που δείχνει ο τρέχων (current)Head pointer, και αποδίδονται τα περιεχόμενά του δεδομένα στην έξοδο, ενώ από τον Next δείκτη προς το επόμενό του στοιχείο, προκύπτει και το νέο κορυφαίο στοιχείο, newHead, το οποίο αποθηκεύεται τελικά στον Head pointer της δομής. Το currentHead στοιχείο αποδεσμεύεται και η διεύθυνσή του θεωρείται και πάλι διαθέσιμη.

Στην πράξη, η καθεμία από τις ενέργειες, έχει τις εξής εκδοχές: τη μέση, την ακραία, και την αδύνατη περίπτωση. Η μέση περίπτωση, τόσο για NQ όσο και για DQ, είναι όταν η ουρά στην οποία αναφέρονται έχει ήδη κάποια στοιχεία, και αυτό παραμένει αμετάβλητο, και από την εκτέλεση της ενέργειας. Για την NQ, αυτό ισχύει αν στην ουρά βρίσκεται τουλάχιστον ένα στοιχείο, ενώ για την DQ, όταν υπάρχουν περισσότερα από ένα, δηλαδή, όχι μόνο ένα. Τότε, η μεν NQ, αναβαθμίζει μόνο τον Tail pointer, η δε DQ, μόνο τον Head pointer. (Σχήματα 3.1(α) και 3.2(α)).

Διαφορετικά, αν δηλαδή μία ουρά αρχικά, σε μία εντολή NQ, είναι άδεια (empty), ή αντίστοιχα, σε μία εντολή DQ, έχει ένα μόνο στοιχείο (one), πρόκειται για τις ακραίες περιπτώσεις της κάθε ενέργειας. (Σχήματα 3.1(β) και 3.2(β)). Και οι δύο, καθώς μετά την εκτέλεσή τους θα έχει αλλάξει η κατάσταση, state, της ουράς, αναβαθμίζουν και τον άλλο pointer της δομής, εκτός από εκείνον που γράφουν κατά την μέση περίπτωση. Η NQ, θα γράψει το newTail και στον Head pointer, και η DQ, θα γράψει και στον Tail pointer, τη μηδενική σταθερά.

Υπό ορισμένες συνθήκες, η εντολή δεν εκτελείται, αλλά ακυρώνεται. Αυτό, για την NQ συμβαίνει όταν δεν υπάρχουν διαθέσιμες θέσεις μνήμης για νέα στοιχεία, και ονομάζεται overflow, ενώ για την DQ, όταν η ουρά στην οποία



Σχήμα 3.2: (α) Μέση και (β) ακραία περίπτωση DQ.

αναφέρεται έχει αδειάσει, και δεν υπάρχει άλλο στοιχείο να εξαχθεί, underflow. Στην περίπτωση της NQ, πρέπει να εξεταστεί ο συνολικά διαθέσιμος χώρος, και η ολοκλήρωση της εντολής δεν εξαρτάται από την κατάσταση της ουράς, μίας και δεν υπάρχει άνω όριο στο πλήθος των στοιχείων που μπορεί κάθε ουρά να έχει, εφόσον υπάρχει διαθέσιμος χώρος. Αντίστροφα, για DQ, αν η κατάσταση της ουράς το επιτρέπει, είναι δεδομένο, σε πλαίσια σωστής λειτουργίας, ότι και ο διαθέσιμος χώρος θα επιτρέπει την ολοκλήρωσή της – επιστρέφοντας ένα στοιχείο που ήδη είχε επιτυχώς δεσμευτεί.

3.1.4 Αλγοριθμικά βήματα

Η παραπάνω περιγραφή της συμπεριφοράς του συστήματος, συγκεκριμένα της καθεμιάς των βασικών του ενεργειών, μπορεί να εκφραστεί συστηματικότερα με διάφορους τρόπους, ως προς το επίπεδο λεπτομέρειας ή αφαίρεσης της έκφρασης.[9, 10] Ξεκινώντας από μία αλγοριθμική καταγραφή διακριτών βημάτων σε φυσική γλώσσα, (Σχήμα 3.3), μπορεί στην συνέχεια να περιγραφεί λεπτομερέστερα σε μορφή RTL, Register Transfer Level, αναφέροντας τις μονάδες αποθήκευσης δεδομένων, και την ροή τους μεταξύ αυτών, ύστερα από τις απαραίτητες διεργασίες. (Σχήμα 3.4). Οι έννοιες μετατρέπονται, έτσι, σε δομικές μονάδες, ενέργειες επεξεργασίας και σήματα κατάστασης. Ακόμη, ανιχνεύονται τα κοινά μοτίβα, λειτουργικά και δομικά, μεταξύ των δύο εντολών, για να συντεθεί ένα συνολικό μονοπάτι δεδομένων, αλλά δομημένο σε επιμέρους στάδια, τα οποία στην συνέχεια θα λειτουργούν επικαλυπτόμενα.

Στο ανώτερο επίπεδο αφαίρεσης, οι δύο ενέργειες περιγράφονται σε φυσική γλώσσα, όπως είχαν παραπάνω περιγραφεί, (3.1.3), σε ακόμα αδρότερες γραμμές, ως εξής:

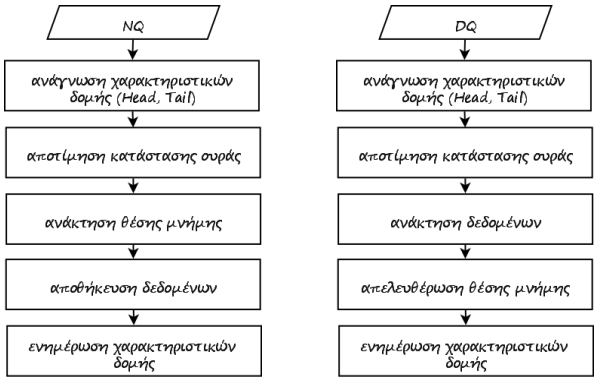
- NQ :

1. ανάγνωση των χαρακτηριστικών τιμών
2. αποτίμιση της κατάστασης της ουράς
3. ανάκτηση θέσης μνήμης
4. αποθήκευση των δεδομένων
5. ενημέρωση των χαρακτηριστικών της δομής

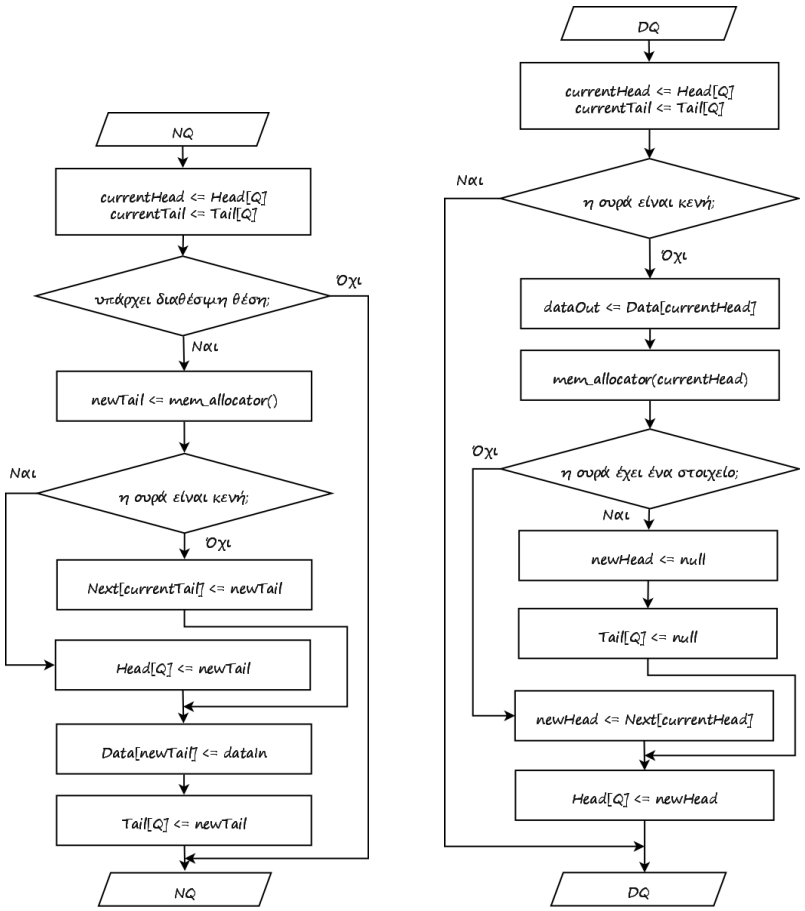
- DQ :

1. ανάγνωση των χαρακτηριστικών τιμών
2. αποτίμιση της κατάστασης της ουράς
3. ανάκτηση των δεδομένων
4. απελευθέρωση της θέσης μνήμης
5. ενημέρωση των χαρακτηριστικών της δομής

Στο επόμενο στάδιο αφαίρεσης της RTL, με περισσότερη λεπτομέρεια, αρχίζουν πλέον να αναδύονται τα βασικά λειτουργικά στοιχεία, όπως οι μνήμες των Head και Tail pointers, και των στοιχείων των ουρών, που το κάθε στοιχείο αποτελείται από ένα κομμάτι δεδομένων, Data, και ένα Next δείκτη προς το επόμενο. Ακόμα, αναγνωρίζονται καταχωτητές που διατηρούν τις τιμές που εισάγονται ή διαβάζονται από τις μνήμες, και από τους ελέγχους προκύπτουν σήματα που θα αναπαριστούν κατάσταση της ουράς, ή του συστήματος γενικότερα. Συγκεκριμένα το αν η ουρά είναι άδεια, empty, ή με ένα μόνο στοιχείο, one, κατάσταση κατά την αρχή της κάθε εντολής, και το αν υπάρχουν άλλες θέσεις στοιχείων στο διαθέσιμο χώρο. Η διαδικασία ανάκτησης ή απελευθέρωσης θέσεων μνήμης μοντελοποιείται πιό ελλειπτικά, από ότι το αντίστοιχο επίπεδο περιγραφής, σαν μία διεργασία αφηρημένη, η οποία απλώς αποδίδει αυτά που ζητούνται, κάπως αυτόματα. Δεν είναι τυχαίο· αποτελεί ιδιαίτερο κομμάτι της σχεδίασης (ενότητα 3.2.2).



Σχήμα 3.3: Αλγοριθμικά βήματα NQ και DQ .



Σχήμα 3.4: RTL βήματα NQ και DQ.

```
// NQ RTL
cH <= HeadMem(Qi);
cT <= TailMem(Qi);
nT <= mem_allocator();
if (nT == null) { //full
    STOP;
}else{
    if (cH == null){ //empty
        HeadMem(Qi) <= nT;
    }else{
        NextMem(cT) <= nT;
    }
    TailMem(Qi) <= nT;
    DataMem(nT) <= dataIn;
}
```

```
// DQ RTL
cH <= HeadMem(Qi);
cT <= TailMem(Qi);
if (cH == null) { //empty
    STOP;
}else{
    if (cH == cT){ //one
        TailMem(Qi) <= null;
        nH <= null;
    }else{
        nH <= NextMem(cH);
    }
    dataOut <= DataMem(cH);
    mem_allocator(cH);
    HeadMem(Qi) <= nH;
}
```

NQ

RTL	μνήμη	RE	AddrR	DataROUt	WE	AddrW	DataWIn
1. currentHead <= Head[Q]	Head	1	Qi	currentHead	-	-	-
2. currentTail <= Tail[Q]	Tail	1	Qi	currentTail	-	-	-
3. newTail <= mem_allocator()	???	1	???	newTail	-	-	-
4. αν δεν υπάρχει χώρος μνήμης, done=0 και τερματισμός εντολής	-	-	-	-	-	-	-
5. (not Empty) ? Next[currentTail] <= newTail	Next	-	-	-	not Empty	currentTail	newTail
6. Data[newTail] <= datain	Data	-	-	-	1	newTail	datain
7. Tail[Q] <= newTail	Tail	-	-	-	1	Qi	newTail
8. (Empty) ? Head[Q] <= newTail	Head	-	-	-	1	Qi	newTail
9. done = 1	-	-	-	-	-	-	-

DQ

RTL	μνήμη	RE	AddrR	DataROUt	WE	AddrW	DataWIn
1. currentHead <= Head[Q]	Head	1	Qi	currentHead	-	-	-
2. currentTail <= Tail[Q]	Tail	1	Qi	currentTail	-	-	-
3. (Empty) ? done = 0, τερματισμός εντολής	-	-	-	-	-	-	-
4. (not One) ? newHead <= Next[currentHead]	Next	not One	currentHead	newHead	-	-	-
5. dataOut <= Data[currentHead]	Data	1	currentHead	dataOut	-	-	-
6. mem_allocator(currentHead)	???	-	-	-	1	???	???
7. (One) ? Head[Q] <= null : Head[Q] <= newHead	Head	-	-	-	1	Qi	(One)?NULL : newHead
8. (One) ? Tail[Q] <= null	Tail	-	-	-	One	Qi	NULL
9. done = 1							

Σχήμα 3.5: Λειτουργίες μονάδων μνήμης NQ και DQ.

Έχοντας αναγνωρίσει τα σήματα κατάστασης και τα βασικά στοιχεία αποθήκευσης του συστήματος, πρέπει να διερευνηθούν οι συσχετίσεις των ενεργειών αυστηρότερα. Από την μία, πρέπει να εντοπισθούν τα κοινά, αλλά και τα διαφορετικά, αν υπάρχουν, δομικά στοιχεία, για να συγκεραστούν, στη συνέχεια, σε ένα σύνολο που να μπορεί να εξυπηρετεί και τις δύο εντολές, γενικεύοντας τες, σε ένα σύστημα κοινό. Από την άλλη, οι μονάδες αυτές, που θα έχουν εντοπιστεί για το γενικευμένο σύστημα, θα πρέπει να διαταχθούν, για να σχηματιστεί το μονοπάτι δεδομένων. Η διάταξη αυτή υπαγορεύεται από τη ροή της πληροφορίας μέσα στο σύστημα, τη χρονική αλληλουχία με την οποία τα σήματα παράγονται και καταναλώνονται, για να παραχθούν άλλα, και ούτω καθεξής, μάλιστα, πάλι προς την κοινή συνισταμένη, κατά το δυνατό, του συνόλου των εντολών του συστήματος. Επιπλέον, πρέπει οι συγκερασμοί και οι γενικεύσεις να γίνονται με γνώμονα την επακόλουθη επικάλυψη, άρα συμπεριλαμβάνοντας μία κατάτμηση σε στάδια, που θα είναι δομικά ανεξάρτητα μεταξύ τους, για να λειτουργήσουν παράλληλα, εκτελώντας διαφορετικές εντολές.

Σε καθεμία από τις εντολές, παρατηρούμε, (Σχήμα 3.5), ότι χρησιμοποιούνται όλες οι μονάδες μνήμης, που αναγνωρίστηκαν παραπάνω. Μάλιστα, οι μονάδες των Head και Tail pointers, σε κάθε εντολή, δύο φορές : μία φορά, απαραίτητα, για ανάγνωση των δεικτών της αναφερόμενης ουράς, και μία ακόμα, για εγγραφή των νέων δεικτών. Πάντοτε, η NQ γράφει τουλάχιστον στην TailMem, και αντίστοιχα, η DQ στην HeadMem, ενώ κατ΄ εξαίρεση, όπως προαναφέρθηκε, στις ακραίες τους περιπτώσεις, γράφουν και την άλλη μνήμη. Κατά την NQ, η τιμή που θα γραφτεί, είτε στη μία, είτε και στις δύο μνήμες, είναι του newTail, και αντίστοιχα, κατά την DQ, είναι του newHead, με τη διαφορά ότι, όταν γράφει και στις δύο μνήμες, η τιμή του θα είναι πάντα ίση με null. Ακόμα, η μνήμη των δεδομένων, Data, από την NQ γράφεται πάντα,

unit	NQ	DQ	παρατηρήσεις	πότε
H	v	v	διαβάζεται: το currHead	πάντα
T	v	v	διαβάζεται: το currTail	πάντα
T	v	-	γράφεται : το newTail	πάντα
H	v	-	γράφεται : το newTail	empty
H	-	v	γράφεται : το newHead	πάντα
T	-	v	γράφεται : το newHead (null)	one
D	v	-	γράφουμε: @newTail	πάντα
D	-	v	διαβάζουμε : @currHead	πάντα
N	v	-	γράφουμε : @currTail	NOT empty
N	-	v	διαβάζουμε : @currHead	πάντα
empty	v	v	χρησιμοποιείται	πάντα
full	v	-	χρησιμοποιείται	πάντα
one	-	v	χρησιμοποιείται	πάντα
null	-	v	χρησιμοποιείται	one

Σχήμα 3.6: Παρατηρήσεις λειτουργιών μονάδων μνήμης.

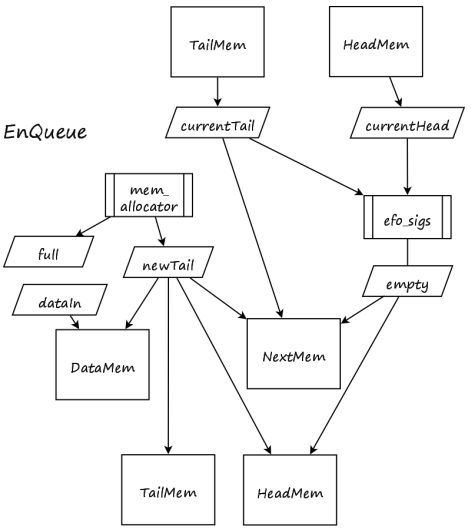
και στην διεύθυνση που δείχνει ο newTail, ενώ από την DQ διαβάζεται πάντα, στη διεύθυνση που δείχνει ο currentHead. Η μνήμη των δεικτών στα επόμενα στοιχεία, Next, επίσης, γράφεται από την NQ, και διαβάζεται από την DQ, αλλά, η NQ μόνο, γράφει σε διαφορετική διεύθυνση, αυτή του currentTail, για να γίνει η σύνδεση των στοιχείων, και μάλιστα κατά συνθήκη, αν υπάρχει τουλάχιστον ένα ακόμα στοιχείο για να συνδεθεί. Τα σήματα ελέγχου, από την άλλη, εμφανίζουν περισσότερη συσχέτιση με κάποια εντολή από τις δύο. Εκτός από το empty, που και οι δύο χρησιμοποιούν, η διαθεσιμότητα της μνήμης ενδιαφέρει μόνο την NQ, ενώ η ύπαρξη ενός εναπομείναντος στοιχείου, την DQ, [2]η οποία έχει και την αποκλειστικότητα στην χρήση της μηδενικής σταθεράς. (Σχήμα 3.6).

Όλες οι μονάδες, πρέπει να συνδεθούν με 'συνδετικό υλικό' τα σήματα που παράγουν και καταναλώνουν, για να προκύψει η χρονική τους διάταξη. (Σχήματα 3.7 και 3.8).

Και στις δύο εντολές, το κοινό χαρακτηριστικό είναι ότι ξεκινούν με ανάγνωση των HeadMem και TailMem, και καταλήγουν γράφοντας τις δύο αυτές μνήμες. Από κει και πέρα, στο μεταξύ, η καθεμία θα επενεργήσει στις DataMem και NextMem, αλλά και στη μονάδα που διαχειρίζεται την μνήμη των στοιχείων, αλλά με διαφορετικό τρόπο, όχι πια λειτουργικά, αλλά χρονικά.

Η NQ, ενδιάμεσα, θα επενεργήσει στην DataMem και τη NextMem, μόνο κατόπιν της λειτουργίας του διαχειριστή μνήμης, αλλά και χωρίς να εξαρτάται καμία άλλη ενέργεια από την ολοκλήρωσή εκείνων, χρονικά. Επίσης, για τις λειτουργίες στις μνήμες NextMem αλλά και HeadMem, πρέπει να έχει παραχθεί το σήμα empty, από τους αρχικούς δείκτες της ουράς. Βέβαια, το σήμα του διαχειριστή μνήμης ότι δεν διαθέτει άλλα στοιχεία θα είχε ήδη εμποδίσει την πρόοδο και ολοκλήρωση της εντολής. Έτσι, οι αναγνώσεις HeadMem,

[2]Βλέπε σελίδα 44



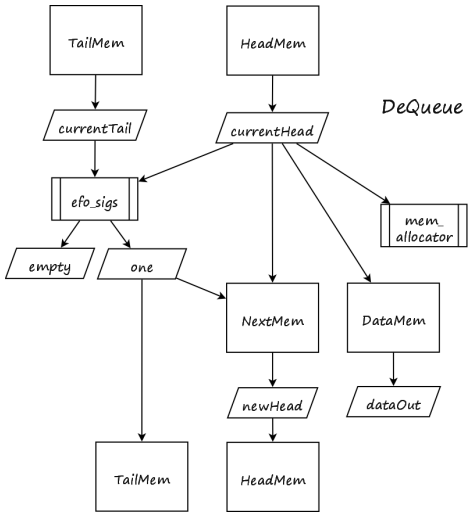
Σχήμα 3.7: Ροή σημάτων μεταξύ μονάδων, στην NQ.

TailMem, και ο διαχειριστής μνήμης είναι στην πρώτη χρονική ομάδα, και η DataMem, η παραγωγή του σήματος empty και η εγγραφή της TailMem, στην δεύτερη, ενώ η εγγραφή της HeadMem, και η NextMem, στην τρίτη.

Αντίστοιχα, η DQ, ενδιάμεσα, είναι καταρχάς σε θέση να αποδώσει το στοιχείο, αν αυτό υπάρχει και η ουρά δεν είναι empty, το οποίο είναι και εκείνο που εμποδίζει την ολοκλήρωση της διαφορετικά, και να επενεργήσει στην DataMem. Αντίθετα, για την εγγραφή της TailMem, απαιτείται η παραγωγή του σήματος one, ύστερα από την ανάγνωση των αρχικών δεικτών της ουράς. Είναι απαραίτητο και για την λειτουργία της NextMem, μετά την οποία, ακολουθεί η εγγραφή της HeadMem. Συνεπώς, οι αναγνώσεις HeadMem, TailMem, είναι στην πρώτη χρονική ομάδα, η παραγωγή των σημάτων empty και one, ακολουθούν, στην δεύτερη, μαζί με τον διαχειριστή μνήμης, και την DataMem. Σε μία τρίτη ομάδα βρίσκονται η εγγραφή του TailMem, και η NextMem, ενώ στην τέταρτη, η εγγραφή της HeadMem.

Έχοντας δώσει σε κάθε μονάδα, ανά εντολή, τον αριθμό βήματος που έχει στο μακρύτερο από τα μονοπάτια που οδηγούν σε αυτή, και μετά, αντιστοιχίζοντάς την με τον μέγιστο αριθμό που έλαβε από τις δύο κατατάξεις, ή και μεγαλύτερο αν χρειαστεί, προκύπτουν τα στάδια του datapath. (Σχήμα 3.9). Από το μέγιστο μήκος μονοπατιού στην εντολή DQ, προκύπτουν τέσσερα στάδια.

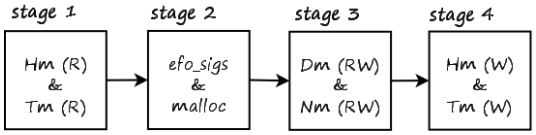
Στο πρώτο, και στο τέταρτο, βρίσκονται οι αναγνώσεις, και οι εγγραφές, αντίστοιχα, των HeadMem και TailMem. Έπειτα, παρεμβάλλονται δύο στάδια, στα οποία μοιράζονται οι υπόλοιπες μονάδες, κατά την σχετική τους ακολουθία, δηλαδή, στο δεύτερο η παραγωγή των σημάτων, και η διαχείριση της μνήμης, και στο τρίτο οι DataMem και NextMem, δίνοντας, την βασική διαμόρφωση του μονοπατιού δεδομένων. (Σχήματα 3.10 και 3.11).



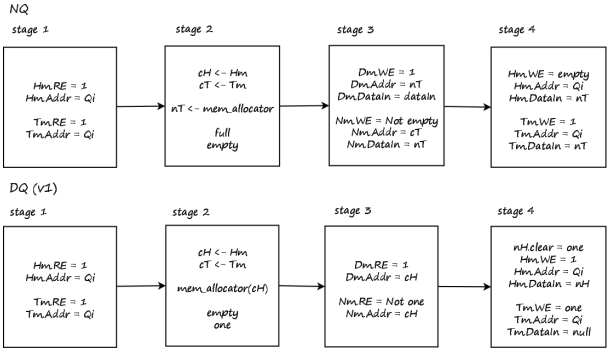
Σχήμα 3.8: Ροή σημάτων μεταξύ μονάδων, στην DQ.

	NQ	DQ	max++		NQ	DQ	max++
Hm.r	1	1	1	Dm	2	2	2++
Tm.r	1	1	1	Nm	3	3	3
efo_sigs	2	2	2	Hm.w	3	4	4
m_alloc	1	2	2	Tm.w	2	3	3++

Σχήμα 3.9: Βάρη μονάδων ανά εντολή, και συνολικά.



Σχήμα 3.10: Το συνολικό μονοπάτι δεδομένων, χωρισμένο σε στάδια.



Σχήμα 3.11: Το μονοπάτι δεδομένων, λειτουργίες ανά εντολή.

Η DataMem, δεν θα μπορούσε, όπως μοιάζει αρχικά, να είναι στο δεύτερο στάδιο, γιατί περιμένει την 'έγκριση λειτουργίας' από το δεύτερο στάδιο, είτε από το διαχειριστή μνήμης, είτε το (not) empty. Επίσης, η κατάσταση κάθε ουράς θα μπορούσε να διατηρείται και σε κάποια status bits, αλλά δεν προτιμάται, γιατί, φαινομενικά μόνο θα ήταν μια ανταλλαγή χρόνου με χώρο, χωρίς να εξοικονομείται πραγματικά χρόνος, καθώς τα currentHead και currentTail θα διαβάζονταν και πάλι από τη μνήμη, για τις άλλες τους χρήσεις, κι έτσι θα είχε άσκοπα σπαταληθεί ο εν λόγω χώρος.

3.2 Η διάσταση του υλικού

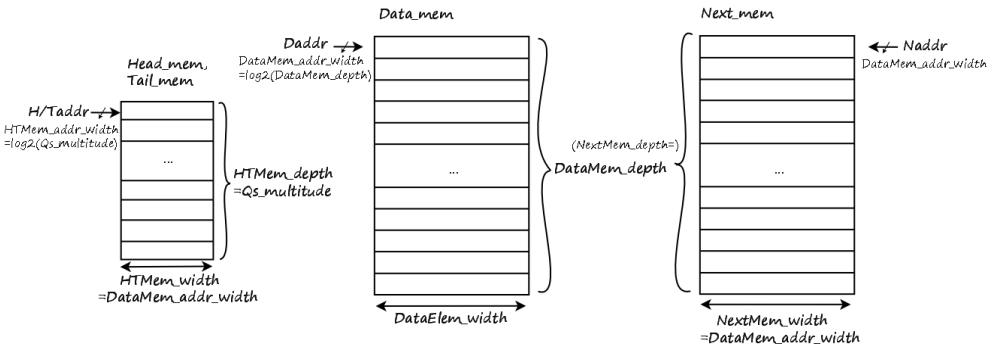
Στο υλικό, οι ψευδαισθήσεις που η παρεμβολή του λειτουργικού συστήματος επιτρέπει στον χρήστη, αλλά και τον προγραμματιστή, αίρονται, και όλα γίνονται απτά. Έτσι, χαρακτηριστικά όπως η μνήμη, γίνονται προβλήματα διαστασιοποίησης, και ζητήματα όπως η κατανομή μνήμης (memory allocation) είναι ανοιχτά, και χρήζουν σχεδιαστικών απαντήσεων. Στην περίπτωση του συγκεκριμένου συστήματος, πρέπει να διερευνηθεί ο τρόπος εκχώρησης και απελευθέρωσης μνήμης, για την εισαγωγή και κατά την εξαγωγή στοιχείων, αντίστοιχα, ύστερα από μία σύντομη αναφορά για τις διαστάσεις των βασικών μονάδων.

3.2.1 Διαστασιοποίηση μονάδων

Το πλήθος των στοιχείων είναι μία από τις παραμέτρους διαστασιοποίησης του συστήματος, και σε πρώτο επίπεδο μπορεί να οριστεί με τρόπο γενικό, και όχι συγκεκριμένα νούμερα. Κάθε στοιχείο αποτελείται από δύο αποθηκευόμενα κομμάτια πληροφορίας, το κομμάτι των δεδομένων που λαμβάνονται από την είσοδο, και ένα δείκτη Next, προς το επόμενο στοιχείο. Αυτά, καθώς αποτελούν, νοητά, μία ενότητα, θα είναι ισάριθμα σε πλήθος, όσα και το σύνολο των στοιχείων που θα επιτρέπει το σύστημα σε όλες τις ουρές, αθροιστικά, από όπου προκύπτει και το βάθος της αντίστοιχης φυσικής μνήμης όπου θα αποθηκεύονται.

Από την άλλη όμως, έχουν και μία βασική διαφορά : το πλάτος του κομματιού δεδομένων είναι μία αυθαίρετη ποσότητα, που εξαρτάται μόνο από την προσδοκώμενη χρήση του συστήματος, και μπορεί να ποικίλλει, από αρκετά bits, μέχρι ακόμα και ένα, ενώ το πλάτος των δεικτών Next καθορίζεται άμεσα από το συνολικό πλήθος των στοιχείων του συστήματος, μίας και πρόκειται ουσιαστικά για διευθύνσεις της μνήμης στοιχείων. Άρα, αν, όπως συνηθίζεται, και με αντίστροφη σχέση υπαιτιότητας, το βάθος της μνήμης είναι δύναμη του δύο, το πλάτος των δεικτών Next θα είναι ίσο με το λογάριθμό του, βάσει δύο.

Από αυτή τη διαφοροποίηση, προκύπτει ότι δεν είναι απαραίτητο τα δύο συστατικά κάθε στοιχείου να είναι σε μία κοινή μνήμη, αλλά μπορούν κάλλιστα, να ανήκουν σε διαφορετικές. Η επιλογή αυτή επιβεβαιώνεται και ενισχύεται από τις λειτουργικές απαιτήσεις του συστήματος, μιάς και, αντίθετα με την



Σχήμα 3.12: Σχετικές διαστάσεις μνημών, παραμετρικά.

DQ, που διαβάζει τα δεδομένα και τον δείκτη επόμενου του ίδιου στοιχείου, στη διεύθυνση currentHead, η NQ, γράφει δεδομένα στο νέο στοιχείο, στη διεύθυνση newTail, και την διεύθυνση του νέου στο μέχρι τότε τελευταίο, στην currentTail. Στην περίπτωση αυτή, η επιλογή κοινής μνήμης, με πλάτος το άθροισμα των δεδομένων και των δεικτών επομένου, ειδικά μιάς και πρόκειται για λειτουργία εγγραφής, θα εισήγαγε άσκοπη πολυπλοκότητα.

Άρα, όπως αναφερόταν και στην αλγοριθμική ανάλυση, οι μνήμες DataMem και NextMem, θα είναι δύο ξεχωριστές φυσικές μονάδες, με ίδιο βάθος, DataMem_depth, αυθαίρετο, και διαφορετικά πλάτη, η μεν πρώτη αυθαίρετο, DataElem_width, η δε δεύτερη συγκεκριμένο, NextMem_width = DataAddress_width.

Όσο αφορά τις μνήμες των χαρακτηριστικών της ουράς, HeadMem και TailMem, για τον αντίστοιχο λόγο, των κατά συνθήκη εγγραφών, θα είναι και κείνες διαφορετικές, παρότι το βάθος τους, HTmem_depth, είναι κοινό, ίσο με το αυθαίρετα επιτρεπόμενο πλήθος παράλληλων ουρών, Qs_multitude, και ότι διαβάζονται πάντα ταυτόχρονα, και στην ίδια διεύθυνση, όπως άλλωστε και οι εγγραφές τους. Το πλάτος τους, HTMem_width, και πάλι θα είναι δεδομένο, μίας και το περιεχόμενό τους είναι διευθύνσεις στοιχείων, άρα ίσο με το πλάτος των Next δεικτών. (Σχήμα 3.12).

3.2.2 Διερεύνηση διαχειριστών μνήμης

Στο λογισμικό, όπως προαναφέρθηκε, η διαχείριση μνήμης γίνεται με τρόπο 'μαγικό', από το λειτουργικό σύστημα, για λογαριασμό του προγραμματιστή. Αυτή είναι βέβαια η μισή αλήθεια, γιατί η μόνη 'μαγεία' βρίσκεται στην αποστασιοποίηση και την απόκρυψη πληροφορίας : σε επίπεδο λειτουργικού συστήματος η διαχείριση μνήμης, δηλαδή η κατανομή του χώρου, μαζί με την κατανομή του χρόνου (του επεξεργαστή), είναι θεμελιώδεις αρμοδιότητες, που εκπληρώνονται με χρήση πολύπλοκων και εκλεπτυσμένων στρατηγικών. [8] Άρα, η αρχική δήλωση είναι ανακριβής, γιατί η εμφανιζόμενη πολυπλοκότητα εξαρτάται πάντα από το επίπεδο στο οποίο αναφέρεται κανείς, ενώ στην

πραγματικότητα είναι σημαντική.

Συγκριτικά με τις στρατηγικές που υλοποιεί το λειτουργικό, για να διαχειριστεί και να κατανείμει την μνήμη, η αναγκαία λύση εδώ, είναι πολύ λιγότερο απαιτητική, χωρίς μηχανισμούς υποστήριξης εικονικών διευθύνσεων, και χωρίς ζητήματα αντικατάστασης ή κατακερματισμού. Πρόκειται για την κατανομή ενός χώρου φυσικών διευθύνσεων, ο οποίος είναι σε απόλυτη αντιστοιχία με τα μεγέθη που θα αποθηκεύει. Το μόνο που μένει, είναι η στοιχειώδης χαρτογράφηση και αναγνώριση του ελεύθερου έναντι του δεσμευμένου χώρου, με τους ανάλογους μηχανισμούς δέσμευσης και απελευθέρωσής του.

Μία τέτοια χαρτογράφηση, υπονοεί την διατήρηση μίας ένδειξης για τον αν κάθε θέση είναι κατειλημμένη ή όχι, την μετάφραση, τρόπον τινά, των ενδείξεων με κάποια πολιτική, και επιλογή μίας ελεύθερης θέσης, κατά τις αιτήσεις για κατανομή χώρου, όπως και την επισήμανση της θέσης, κατά την απελευθέρωση χώρου. Σε υλικό, μπορεί να υποστηριχθεί με χρήση κάποιας επιπλέον μνήμης, πλάτους ενός bit, και βάθους ίσου με το βάθος της μνήμης των στοιχείων, και ενός κωδικοποιητή προτεραιότητας.

Ο κωδικοποιητής προτεραιότητας (priority encoder) είναι ένα βασικό συνδυαστικό κύκλωμα, που λαμβάνοντας είσοδο πλάτους 2^n (ίσως και λιγότερου) δίνει στην έξοδο, πλάτους n, το δυαδικό κώδικα που αντιστοιχεί στις μεταβλητές εισόδου, σαν κάθε γραμμή της να ήταν ένας αριθμός από ένα διατεταγμένο σύνολο. Επιπλέον, δίνοντας διαφορετική προτεραιότητα σε κάθε γραμμή εισόδου έναντι των άλλων, δίνει μοναδικό αποτέλεσμα τον κώδικα της γραμμής με την μέγιστη προτεραιότητα, ακόμα και αν δεν είναι εκείνη η μοναδική ενεργή. [11]

Η μνήμη, με ένα bit κατάστασης για κάθε στοιχείο, θα ξεκινά με όλα τα στοιχεία διαθέσιμα και όλα τα bit στην ίδια ένδειξη, και θα αλλάζει κατάσταση στο bit του οποίου η διεύθυνση είναι ενεργή, είτε για δέσμευση είτε απελευθέρωση. Τα bit κατάστασης θα δίνονται ως είσοδοι στον κωδικοποιητή, που θα τα μεταφράζει σε ένα δυαδικό κώδικα, αντίστοιχο της διεύθυνσης της θέσης που έχει προκριθεί, ανάμεσα στις σημασμένες ως ελεύθερες, και μπορεί άμεσα να χρησιμοποιηθεί ως αναφορά προς τη θέση, τόσο στη μνήμη στοιχείων, όσο και στη μνήμη του διαχειριστή. Στην περίπτωση απελευθέρωσης, η διεύθυνση θα επιστέφεται από το τμήμα διαχείρισης των ουρών, και θα δίνεται σαν είσοδος στο διαχειριστή μνήμης, για την επισήμανση του στοιχείου ως ελεύθερου.

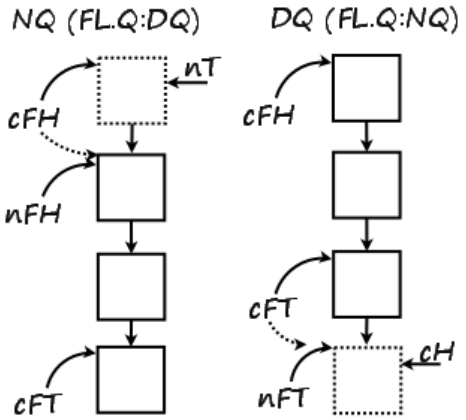
Λόγω της διαισθητικής της λειτουργικότητας, η λύση με κωδικοποιητή προτεραιότητας μοιάζει ελκυστική. Αντίθετα, όμως, είναι δαπανηρή, τόσο σε χώρο, όσο και σε χρόνο. Ο απαιτούμενος της χώρος συνίσταται στην μνήμη των bit κατάστασης, και στις πύλες του συνδυαστικού κομματιού. Μάλιστα, με την αύξηση των μνημών των στοιχείων, η μνήμη αυξάνεται γραμμικά, ακολουθώντας την πρώτη στη διάσταση του βάθους, ενώ και το κύκλωμα, αυξάνεται λογαριθμικά.[3] Παρότι, θεωρητικά κάθε λογική συνάρτηση μπορεί να υλοποιηθεί με δύο επίπεδα πυλών, στην πράξη, λόγω του ότι ο αριθμός εισόδων που κάθε

[3] πιθανά με βάση διαφορετική του 2, αλλά κάποιας άλλης δύναμης του, για παράδειγμα 4.

```
//FreeList.Queue

//operation NQ
//(FL.Q = DQ/R)
nT <= cFH;
nFH <= NextMem(cFH);
cFH <= nFH;

//operation DQ
//(FL.Q = NQ/W)
NextMem(cFT) <= cH;
cFT <= cH;
```



Σχήμα 3.13: Η FreeList υλοποιημένη ως Ουρά.

πύλη μπορεί να υποστηρίξει είναι περιορισμένος, η αύξηση των εισόδων και ο επακόλουθος πολλαπλασιασμός των πυλών του κυκλώματος, έχει αποτέλεσμα αύξηση των επιπέδων στα οποία διατάσσονται. Η επιβεβλημένη ανάπτυξη σε δεντρική δομή, εισάγει και τις αντίστοιχες χρονικές καθυστερήσεις ανά επίπεδο, προκαλώντας την επακόλουθη αύξηση του συνολικά απαιτούμενου χρόνου μέχρι την έξοδο του κωδικοποιητή.

Διαδεδομένη εναλλακτική, είναι η ενσωμάτωση της διαχείρισης μνήμης στην υπάρχουσα μνήμη των στοιχείων. Κανονικά, στις διευθύνσεις των δεσμευμένων στοιχείων υπάρχει χρήσιμη πληροφορία, τα δεδομένα εισόδου και οι δείκτες Next, αποτελώντας το σύνολο των ουρών του συστήματος, σε πεπλεγμένη κατάσταση. Στις μη δεσμευμένες διευθύνσεις, θα υπήρχε άχρηστη πληροφορία, από παλιές εγγραφές που απελευθερώθηκαν, ή από την αρχικοποίηση, πάντως τιμές άκυρες, που θεωρούνται 'σκουπίδια'. Αυτό δεν είναι απαραίτητο· αναβαθμίζοντας την άχρηστη πληροφορία σε χρήσιμη, οι μη δεσμευμένες διευθύνσεις μπορούνε οργανωμένα να συγκροτηθούν σε μία δομή λίστας, και έτσι να συνυπάρχουν το σύνολο των ουρών με τον διαχειριστή ουρών, χωρίς να σπαταλάται πλέον καμία θέση. Από το σύνολο των ελεύθερων θέσεων, αποτελούμενο από μία δομή, να κατατάσσονται στο σύνολο των δεσμευμένων, σε πολλαπλές δομές, και αντίστροφα.

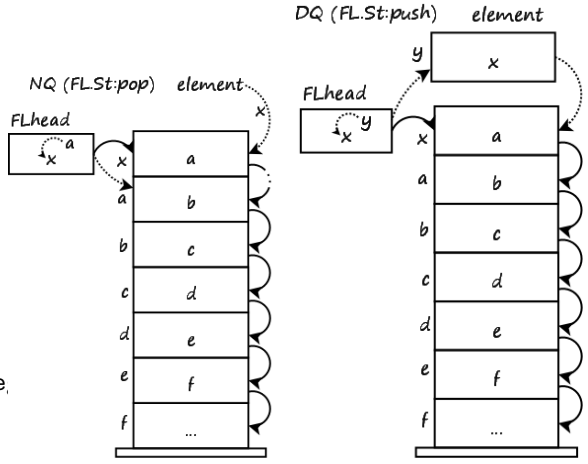
Η λίστα αυτή, συνήθως ονομαζόμενη Free List, περιέχει τις διαθέσιμες διευθύνσεις των στοιχείων, συνδεδεμένων μεταξύ τους, αλλά, χωρίς να φέρουν κάποιο κομμάτι πρόσθετης πληροφορίας. Μιάς και η μόνη αποθηκευμένη πληροφορία της FL είναι αναφορές σε στοιχεία, η μνήμη που αποθηκεύεται είναι η NextMem, χωρίς αντίστοιχο κομμάτι στη DataMem, η οποία θα συνεχίσει να περιέχει 'σκουπίδια'. Το παραπάνω όμως δεν αποτελεί πρόβλημα, καθώς, οι θέσεις των δεδομένων, αλλά και γενικά, διαβάζονται μονάχα αφού πρώτα δεσμευτούν και γραφτούν με κάτι γνωστό και χρήσιμο.

Η επιλογή της FL έχει μειωμένο κόστος στο υλικό, αφού, 'παρασιτεί' στον,

```
//FreeList.Stack

//operation NQ
//(FL.St = pop/R)
element <= headReg;
nFH <= Mem[headReg];
headReg <= nFH;

//operation DQ
//(FL.St = push/W)
Mem[element] <= headRe
headReg <= element;
```

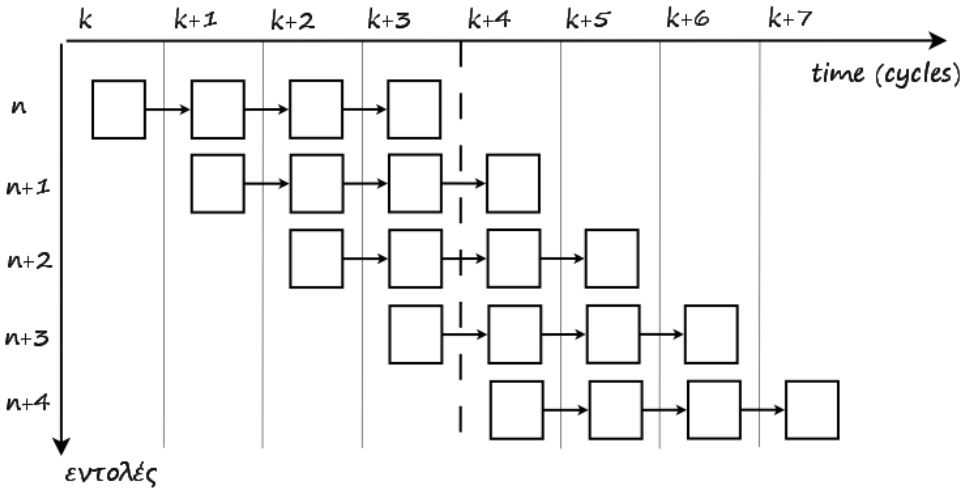


Σχήμα 3.14: Η FreeList υλοποιημένη ως Στοίβα.

αλλιώς, αδρανή χώρο της ήδη υπάρχουσας μνήμης, και από αυτήν την άποψη δεν επιβαρύνει καθόλου. Χρησιμοποιεί κάποιους καταχωρητές για τον εντοπισμό της, και επιπλέον, έχει ανάγκη από τις ανάλογες δομές ελέγχου. Καθώς η μνήμη στοιχείων αυξάνει σε μέγεθος, ο χώρος των δεικτών της παράσχεται άμεσα, και η μόνη αύξηση είναι στις διαστάσεις, δηλαδή το πλάτος και όχι το πλήθος, των εισόδων του κομματιού ελέγχου της. Η αύξηση αυτή, σε σχέση με την αύξηση του μεγέθους της μνήμης στοιχείων, είναι λογαριθμική, καθώς πρόκειται για τις ίδιες της τις διευθύνσεις, συνεπώς, ελάχιστη, ακόμα και για τα επιμέρους υποκυκλώματα του ελέγχου που επηρεάζει.

Η λίστα αποτελεί μία γενίκευση δομής, η οποία εξειδικεύεται με τον καθορισμό κάποιων λειτουργικών απαιτήσεων και πολιτικών, όπως της χρονικής, ή βάσει άλλου κριτηρίου, διάταξης των στοιχείων. Όπως προαναφέρθηκε, υπακούοντας στο κριτήριο κάποιας χρονικής διάταξης, προκύπτουν οι δομές της Ουράς και της Στοίβας. Η FL, μην έχοντας κανένα άλλο κριτήριο που θα μπορούσε να είναι σχετικό, μπορεί να υιοθετήσει την χρονική πολιτική κατάταξης των στοιχείων της, είτε με τον ένα, FIFO, είτε με τον άλλο, LIFO, τρόπο. (Σχήματα 3.13 και 3.14).

Οι επιλογές είναι αντίστοιχες, στα γενικά τους χαρακτηριστικά, καθώς οι δύο δομές δεν διαφέρουν στις απαιτήσεις τους και τον έλεγχό τους σημαντικά. Και οι δύο τους χρειάζονται καταχωρητές, δύο ή έναν, για τη διατήρηση των ακραίων στοιχείων, και κάποιον έλεγχο, για να εντοπιστεί το πότε αδειάζουν. Η βασική τους διαφορά, είναι στη χρονικά καθοριζόμενη πολιτική εισαγωγών και απελευθερώσεων στοιχείων, η οποία έχει τον αντίκτυπό της στην τοπικότητα της δομής. Η δομή της Στοίβας, επενεργώντας εντοπισμένα σε ένα μόνο άκρο της λίστας, αντί για δύο όπως η Ουρά, έχει ισχυρότερο τοπικό χαρακτήρα, ως προς και τις δύο εκδοχές, χωρικά και χρονικά. Αυτή είναι η ιδιότητα που προκρίνει την δομή της Στοίβας, γενικότερα στις εφαρμογές κατανομής



Σχήμα 3.15: Επικάλυψη τεσσάρων σταδίων.

μνήμης, όπως και στην συγκεκριμένη περίπτωση. Παρότι δεν προϋποτίθεται από κάποια συγκεκριμένη προδιαγραφή του συστήματος, ούτε και είναι άμεσα εμφανής, η εκμετάλλευσή της προκύπτει αργότερα στην τρέχουσα υλοποίηση, αλλά μπορεί να γίνει εντονότερη σε ενδεχόμενη επέκταση του συστήματος.

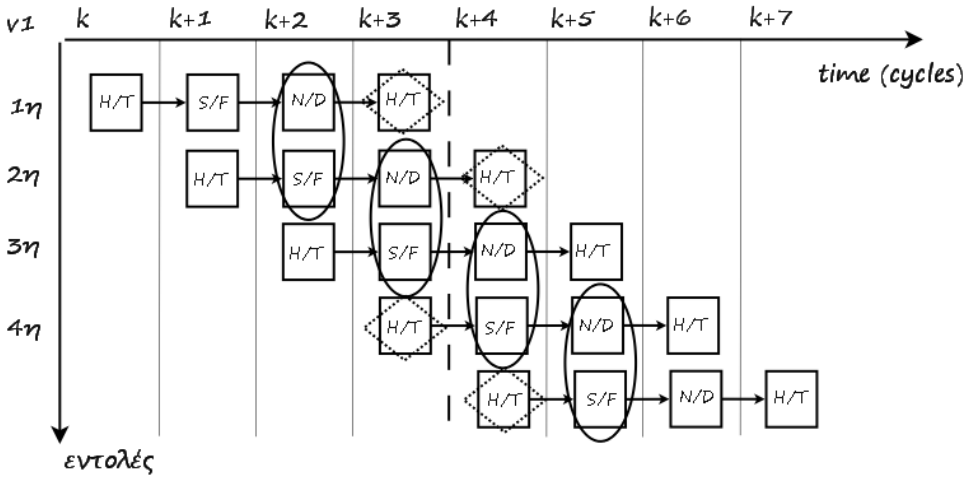
3.3 Συνθέτοντας την ομοχειρία

Το μονοπάτι δεδομένων που προέκυψε από την αρχική ανάλυση, όπως ήδη αναφέρθηκε, θα μπορούσε κάλλιστα να εκτελεί την λειτουργία του σε έναν ή πολλαπλούς κύκλους και χωρίς επικάλυψη, και όχι σε ομοχειρία. Υπό την οπτική της ομοχειρίας, απαιτούνται κάποιες διευκρινίσεις ή και προσθήκες στην αρχική σχεδίαση, τόσο για την υποστήριξη της επικάλυψης με την προσαρμογή των δομικών στοιχείων στην παράλληλη λειτουργία, όσο και για την αντιμετώπιση των ζητημάτων που ανακύπτουν από την χρονική αναδιάταξη των βημάτων, σε αντιδιαστολή προς την πλήρως σειριακή εκτέλεση τους. Έτσι, η σχεδίαση τροποποιείται αλλού και αλλού ενισχύεται με επιπλέον λογική. Οι λεπτομέρειες της υλοποίησης παρουσιάζονται στο ομώνυμο κεφάλαιο 5.

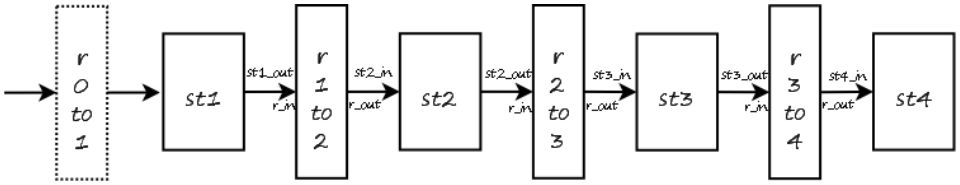
3.3.1 Επικάλυψη τώρα!

Ένα μονοπάτι δεδομένων κατά την ομόχειρή του λειτουργία, θα εκτελεί σε παραλληλισμό μία διαφορετική εντολή σε κάθε στάδιό του. Στην περίπτωσή μας, όταν πλέον έχει γεμίσει με δεδομένα, μετά από τους πρώτους τρεις κύκλους, θα επικαλύπτονται τέσσερις εντολές, σε κάθε δεδομένο κύκλο. (Σχήμα 3.15).

Στη γενική μορφή του, ως τώρα (βλ. Σχήμα 3.10), το μονοπάτι δεδομένων ουσιαστικά αποτελείται από τις τέσσερις μνήμες, Head, Tail, Next και Data, οι



Σχήμα 3.16: Επικαλυπτόμενη λειτουργία μνημών (v1).



Σχήμα 3.17: Καταχωρητές ομοχειρίας.

οποίες διαβάζονται και γράφονται από σχετικά απλή λογική που ενεργεί στις εισόδους και τις αναγνωσμένες τιμές, χωρίς καμία επεξεργασία πάνω τους, αλλά, βάσει απλών ελέγχων με κριτήρια τις τιμές αυτές ή, από τις ίδιες παραγόμενα, σήματα ελέγχου. Εκτός από τον, ελάχιστα πολυπλοκότερο, έλεγχο της Free List, οι υπόλοιποι έλεγχοι είναι κριτήρια ισότητας.

Οι μνήμες αυτές, για την λειτουργία χωρίς παραλληλισμό, παρόλο που για κάθε μία εντολή ενεργούν, εκτός της DataMem, σε περισσότερους από ένα κύκλους, πραγματοποιούν σε κάθε συγκεκριμένο κύκλο μία πρόσβαση, και έτσι είναι μίας θύρας. Στην ομόχειρη εκτέλεση (Σχήμα 3.16), όλες τους, και πάλι εκτός της DataMem, θα εξυπηρετούν, σε κάθε κύκλο, δύο στάδια, άρα είναι απαραίτητη η αναβάθμισή τους σε δίθυρες.

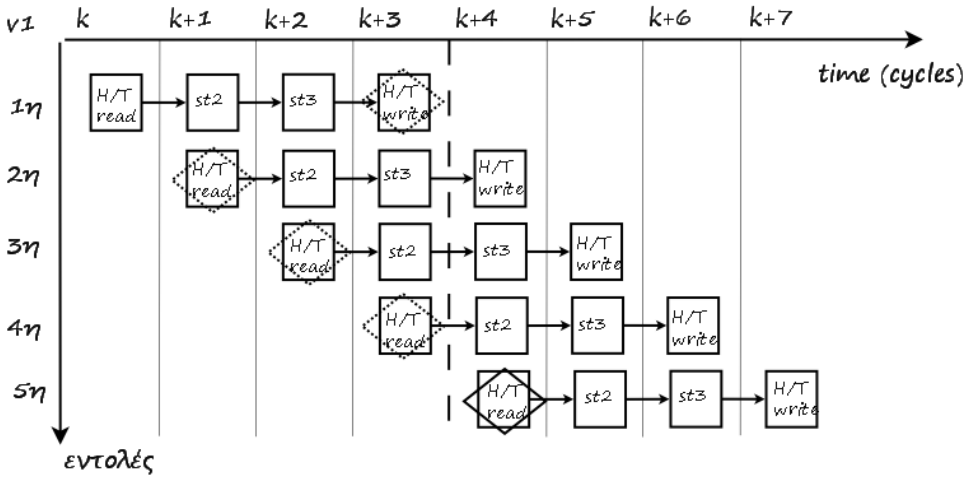
Τα σήματα, αντίστοιχα, που αποτελούνται είτε από εισόδους, είτε αναγνωσμένες τιμές ή τα παραγόμενα σήματα ελέγχου, που ρέουν ανάμεσα στα διαφορετικά στάδια, πρέπει να διατηρηθούν στους απαραίτητους καταχωρητές, τους επονομαζόμενους ομοχειρίας (Σχήμα 3.17). Ήδη, σε μία εκδοχή πολλαπλών κύκλων θα υπήρχε ένα μέρος τους, αλλά στην ομοχειρία είναι απαραίτητοι για κάθε τιμή που μεταβαίνει από ένα στάδιο προς οποιοδήποτε επόμενό της, καθώς κάθε εντολή περιορίζεται να 'ζεί' σε ένα στάδιο, άρα και πρέπει να διαθέτει μαζί της οποιαδήποτε τιμή την αφορά και θα της χρειαστεί, έστω και αργότερα.

3.3.2 Κίνδυνος στις μνήμες Head/Tail

Η επικαλυπτόμενη εκτέλεση των εντολών, αλλάζει την σχετική σειρά που εκτελούνται τα επιμέρους βήματα τους, καθώς μία νέα εντολή ξεκινά σε κάθε κύκλο, χωρίς να έχουν ολοκληρωθεί οι (β-1) προηγούμενες της (με β το βάθος του μονοπατιού). Σε αντίθεση με την σειριακή εκτέλεση, όπου κάθε νέα εντολή ξεκινά μετά την ολοκλήρωση της αμέσως προηγούμενής της, υπάρχει το ενδεχόμενο να απαιτείται να διαβαστεί μία τιμή, που κανονικά θα είχε γραφτεί στην εσωτερική κατάσταση του συστήματος από προηγούμενη εντολή. Στην περίπτωση που η εντολή εκείνη βρίσκεται ακόμη μέσα στο μονοπάτι, και μην έχοντας ολοκληρωθεί, κατά πάσα πιθανότητα δεν έχει καταγράψει τα αποτελέσματά της. Εφόσον μεταξύ των εντολών που συνυπάρχουν στο μονοπάτι υπάρχει εξάρτηση δεδομένων, εκείνων που χρειάζεται να διαβαστούν, από εκείνα που ακόμα δεν έχουν γραφεί, προκαλείται κίνδυνος δεδομένων, της κατηγορίας Ανάγνωση μετά την Εγγραφή (Read After Write, RAW). Ο κίνδυνος θα πρέπει να αντιμετωπιστεί, με σκοπό την αποκατάσταση της εξάρτησης, ώστε να μην διαφέρει η εκτέλεση σε ομοχειρία από την αντίστοιχή της σειριακή.

Γνωστή τακτική για την αποφυγή των άλλων κινδύνων, της Εγγραφής μετά την Εγγραφή (Write After Write, WAW), και της Εγγραφής μετά την Ανάγνωση (Write After Read, WAR), είναι η τοποθέτηση των αναγνώσεων κατά το δυνατόν νωρίτερα, προς την αρχή, και των εγγραφών αργότερα, προς το τέλος του μονοπατιού. Πράγματι, η εφαρμογή της εξαλείφει, σε πολλές περιπτώσεις, τους κινδύνους εκείνους, αντίθετα όμως, συνεισφέρει στην ενίσχυση του πρώτου, του RAW. Δεν πρόκειται, μεν, για τη γενεσιουργό του αιτία, που δεν είναι άλλη από την αιτία της τοπικότητας και των εξαρτήσεων, από την βάση κάθε επεξεργασίας, ότι κάτι θα διαβαστεί πρώτα, για να μετατραπεί, στη συνέχεια, σε κάτι το χρήσιμο, και ως τέτοιο θα αποθηκευτεί για περαιτέρω χρήση, δηλαδή να διαβαστεί (καταρχάς) ξανά, και μάλιστα σύντομα. Κατά συνέπεια, θα παρεμβληθούν μεταξύ ανάγνωσης και εγγραφής τα επιπλέον στάδια του μονοπατιού, επεκτείνοντας το εύρος των εντολών που ενδέχεται να έχουν επικίνδυνη εξάρτηση. Στην ακραία εκδοχή, των αναγνώσεων στην αρχή, και των εγγραφών στο τέλος, ο κίνδυνος μπορεί να αφορά τελικά όλες τις εντολές που συνυπάρχουν στο μονοπάτι.

Στο συγκεκριμένο σύστημα η παραπάνω κατάσταση εμφανίζεται σε σχέση με τις μνήμες Head και Tail. Πράγματι, κάθε εντολή θα διαβάσει και τις δύο μνήμες στο πρώτο στάδιο, και αν ολοκληρωθεί επιτυχώς, στο τέταρτο στάδιο θα γράψει τουλάχιστον την μία, ακόμα και τις δύο τους. Οι εν λόγω μνήμες αποθηκεύουν τους δείκτες που ορίζουν κάθε ουρά του συστήματος, οπότε κάθε επόμενη εντολή που αφορά την ίδια ουρά, εξαρτάται από την προηγούμενη. Σε περίπτωση που συμπέσει η παρουσία τους στο μονοπάτι, εμφανίζεται κίνδυνος δεδομένων, καθώς οι τιμές που χρειάζεται να διαβαστούν θα έχουν γραφεί μόνο για τις εντολές που τελικά δεν θα επικαλυφθούν! (Σχήμα 3.18). Συνεπώς, όσες εντολές αναφέρονται σε ουρά ίδια με κάποια από τις εντολές που ήδη υπάρχουν στο μονοπάτι, δεν μπορούν να διαβάσουν, έγκυρα, τις τιμές από τις



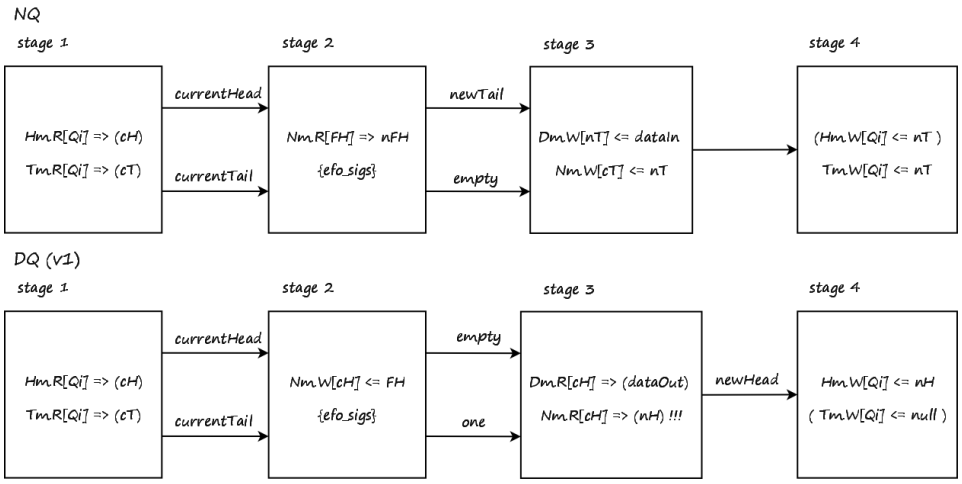
Σχήμα 3.18: Κίνδυνος δεδομένων στις μνήμες Head και Tail.

μνήμες, και θα πρέπει να μελετηθεί κατά πόσο είναι δυνατόν να τις αποκτήσουν με προώθηση (forwarding), από τις ενδιάμεσες δομές αποθήκευσης, και όχι την επίσημη κατάσταση του συστήματος, ή αν τελικά δεν είναι, και προκληθούν ανασχέσεις.

Μεταξύ των εντολών που συνυπάρχουν στο μονοπάτι δεδομένων, είναι δυνατόν να αναφέρονται στην ίδια ουρά από καμία, να είναι όλες διαφορετικές, μέχρι και τέσσερις, έστω 'συγγενικές'. Κατά αντιστοιχία της σειριακής εκτέλεσης, οσεσδήποτε κι αν είναι οι 'συγγενικές' της μέσα στο μονοπάτι, η κατάσταση που ενδιαφέρει κάθε νέα εντολή που εισάγεται, είναι εκείνη στην οποία θα άφηνε το σύστημα, με την ολοκλήρωσή της, η κοντινότερη από τις 'συγγενικές' της. Έτσι, οι συσχετίσεις που έχει νόημα να μελετηθούν, θα περιλαμβάνουν μόνο ανά δύο εντολές μέσα στο μονοπάτι, ενώ οι μεταξύ τους εντολές, αν υπάρχουν, θα αφορούν διαφορετικές ουρές.

Δεδομένου όμως ότι και κάποιες από τις προηγούμενες είναι πιθανόν να ήταν 'συγγενικές', τίθεται μία απαίτηση: να υπάρχει ένα σταθερό σημείο, από το οποίο και έπειτα όλες οι τιμές που συνοδεύουν την εντολή στις εσωτερικές δομές να θεωρούνται 'ασφαλείς', και να είναι εκείνες που πραγματικά αναπαριστούν τους δείκτες της ουράς που πρόκειται. Έτσι, από το σημείο εκείνο και έπειτα, κάθε εντολή θα θεωρείται ότι έχει τόσο έγκυρες τιμές όσο και αν μπορούσε και τις είχε διαβάσει κανονικά από τις μνήμες, περιορίζοντας τον έλεγχο της κάθε νέας εντολής που εισάγεται μόνο στο αν σχετίζεται με κάποια, και πιό πρόσφατα με ποιά, από τις υπόλοιπες εντολές, εξαλείφοντας την ανάγκη να ελέγξει αν εκείνη είναι 'συγγενής' με προηγούμενές της και έχει ενημερωθεί. Αν υπάρξει το 'ασφαλές σημείο', θα είναι αδιάφορο. Θα μπορεί να πάρει τις απαραίτητες τιμές από το 'ασφαλές σημείο' της πιο κοντινής 'συγγενικής' της, και να θεωρείται άπαξ τακτοποιημένη.

Η ύπαρξη του 'ασφαλούς σημείου' έχει δύο απαιτήσεις: καταρχάς, να είναι



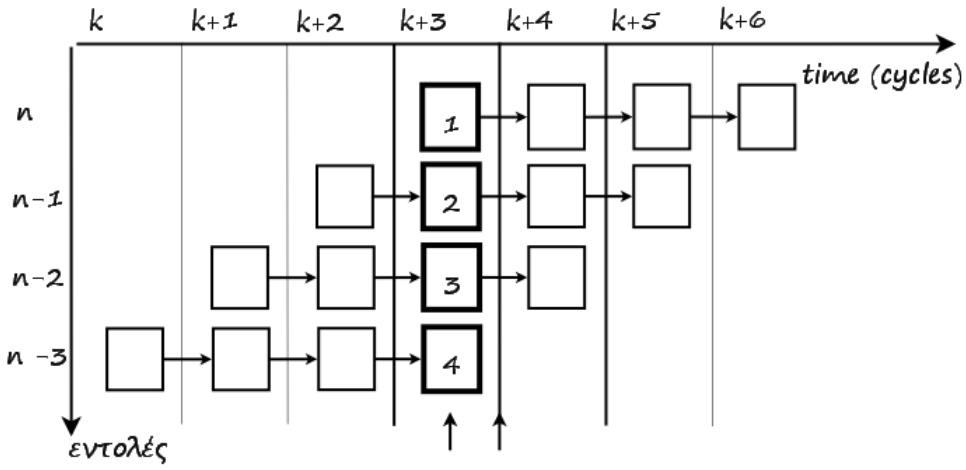
Σχήμα 3.19: Στάδια παραγωγής σημάτων (v1).

έτοιμες και διαθέσιμες όλες οι απαραίτητες τιμές, και σε κάθε δυνατή περίπτωση. Η άλλη, η πιο ουσιαστική, είναι, σε κάθε περίπτωση, να προωθούνται και οι δύο δείκτες, αποκόπτοντας έτσι την εξάρτηση από τις μνήμες τελείως, που σε μία ενδεχόμενη σειρά προωθήσεων δεν θα είχαν γραφεί. Αυτό θα μπορούσε, μεν, να αναλυθεί και να προβλεφθεί, αλλά θα χρειαζόταν επανεξέταση όλων των 'συγγενικών' εντολών, σε πλήρη αντίθεση με το επιδιωκόμενο.

Κάθε νέα εντολή που εισάγεται στο μονοπάτι, θα ελέγχει λοιπόν τις προηγούμενες, για να ανακαλύψει την πιο κοντινή 'συγγενική' της, με το μόνο ουσιαστικό στοιχείο που διαθέτει για τον εαυτό της, την ουρά στην οποία αναφέρεται. Αν υπάρχει κάποια τέτοια μέσα στο μονοπάτι, θα πρέπει να ελεγχθεί ακόμα, από τα δεδομένα εκείνης πλέον, το αν έχει τα εχέγγυα για να ολοκληρωθεί, και έτσι να πρόκειται να αλλάξει την κατάσταση του συστήματος, και άρα, να χρειαστεί οι τιμές της να προωθηθούν.

Για κάθε εντολή, είτε NQ είτε DQ, η οποία θα ολοκληρωθεί, υπάρχουν δύο περιπτώσεις, να γράφει τον ένα, ή και τους δύο δείκτες. Όπως αναλύθηκε παραπάνω,(Σχήμα 3.6), από την κατάσταση της ουράς στην οποία ενεργεί, είναι δεδομένο κάθε εντολή πόσους δείκτες θα επηρεάσει, αλλά και με ποιάς μεταβλητής την τιμή. Συνεπώς, ένας επιπλέον έλεγχος, για κάθε εντολή που πρόκειται να ολοκληρωθεί, θα δώσει το πόσους δείκτες θα γράψει. Αυτός, θα χρειαστεί στην επιλογή του κατάλληλου δείκτη, μεταξύ current και new, για προώθηση τόσο για τον Head όσο και τον Tail, που είναι απαραίτητο να λάβουν την τιμή, ακόμα και αν δεν αλλάζει, για την εξασφάλιση του μηχανισμού του 'ασφαλούς σημείου'.

Το 'ασφαλές σημείο', που θα πρέπει να εντοπιστεί σε συγκεκριμένο στάδιο του μονοπατιού, από την οπτική της εντολής που θα λάβει τις προωθημένες τιμές, θα πρέπει να είναι σε στάδιο τέτοιο ώστε, στην μεταξύ τους επικάλυψη, ακόμα και η κοντινότερη εντολή, να είναι σε θέση να παράσχει τις τιμές, προτού

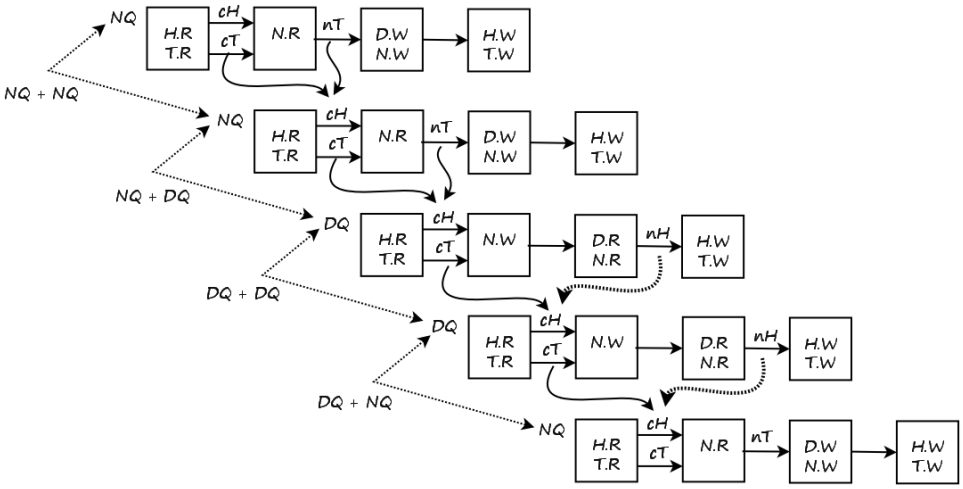


Σχήμα 3.20: 'Ασφαλές σημείο' προώθησης.

χρειαστεί να καταναλωθούν από την εκδιδόμενη. Από την αντίστροφη οπτική, των εντολών που παράγουν τις προωθούμενες τιμές, θα καταλήξει να βρεθεί στο νωρίτερο στάδιο όπου θα μπορούν να παρασχεθούν οι τιμές από το σημείο παραγωγής τους.

Για το σύστημά μας, το 'ασφαλές σημείο', θα έπρεπε να μπορεί να αποδώσει τις τιμές των δεικτών πριν το στάδιο που χρησιμοποιούνται, συγκεκριμένα πριν το δεύτερο, άρα, από την κοντινότερη επικαλυπτόμενη προκύπτει ότι θα έπρεπε να βρίσκεται μετά το δεύτερο στάδιο. (Σχήμα 3.20). Συνεπώς, είναι καταρχάς απαραίτητο να μελετηθεί το αν μέχρι εκεί είναι οι τιμές διαθέσιμες σε κάθε περίπτωση. Η διαθεσιμότητα συνίσταται από τη μία στο σημείο παραγωγής των δεικτών, και μάλιστα το να έχουν παραχθεί σε κάθε περίπτωση μέχρι τότε. Από την άλλη, στην δυνατότητα διαπίστωσης, βάσει των σημάτων κατάστασης, του αν η εντολή θα ολοκληρωθεί και οι τιμές θα αποτυπωθούν, τελικά, στην κατάσταση του συστήματος, όπως και σε τι κατάσταση ουράς επενεργεί, άρα και πόσους δείκτες θα επηρέαζε. Η δεύτερη παράμετρος, η ανάλυση για την εκτίμηση της ολοκλήρωσης και του τύπου της 'συγγενούς' εντολής, παραλείπεται για το κεφάλαιο 5, και θα μελετηθεί η πρώτη, με την παραδοχή ότι η εντολή έχει αναγνωριστεί ως προς την ολοκλήρωση και τον τύπο της. Άλλωστε το να είναι ολοκληρώσιμη είναι και πραγματική προϋπόθεση, για να παράγει τον δείκτη new της, και σε κάθε περίπτωση θα χρειαστεί η τιμή του, τουλάχιστον για τον ένα δείκτη.

Τα σήματα που θα προωθηθούν, είναι είτε ο δείκτης current, είτε ο δείκτης new, της 'συγγενούς', για καθέναν από τους δύο δείκτες, Head και Tail, της εκδιδόμενης εντολής. Στην γενική περίπτωση, στο μονοπάτι δεδομένων που προέκυψε παραπάνω, και οι δύο current διαβάζονται από τις μνήμες στο πρώτο στάδιο. Οι new, αντίστοιχα, παράγονται, για την NQ στο δεύτερο, για την DQ στο τρίτο στάδιο. (Σχήμα 3.19). Στην περίπτωση που η κοντινότερη



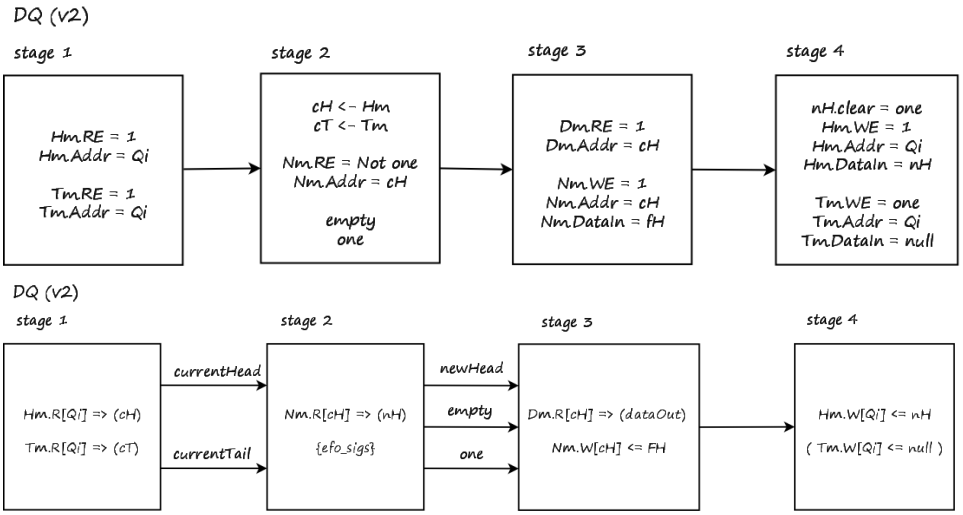
Σχήμα 3.21: Πρόβλημα : το newHead καθυστερεί (v1) .

'συγγενής', ήταν DQ, και αμέσως προηγούμενη της εκδιδόμενης, τότε η τιμή του newHead, δεν θα ήταν έτοιμη για προώθηση πριν το δεύτερο στάδιο της, όπως είναι σκόπιμο. (Σχήμα 3.21). Έτσι, όχι μόνο δεν θα ήταν δυνατή η εφαρμογή του 'ασφαλούς σημείου', αλλά θα χρειαζόταν και ανάσχεση ενός κύκλου, στην περίπτωση που η εντολή ολοκληρωνόταν χωρίς να αδειάζει η ουρά, που θα ήταν και το συχνότερο.

Η αναθεώρηση της σχεδίασης γίνεται επιτακτική, κατόπιν μίας παρατήρησης: συνδυαστικά, δύο σχεδιαστικές επιλογές, οδηγούν σε λανθασμένη συμπεριφορά, πέραν της παρεμπόδισης της προώθησης. Συγκεκριμένα, η επιλογή της δομής της Στοίβας,[4] για την Free List, και η ανάγνωση της NextMem στο τρίτο στάδιο. Ακολουθώντας την εγγραφή στη διεύθυνση currentHead, η ανάγνωση θα δίνει πάντα εσφαλμένο αποτέλεσμα, έχοντας ήδη πανωγραφεί το δεδομένο προτού διαβαστεί.(Σχήμα 3.19).

Το σφάλμα αυτό, από την άλλη, οδηγεί και ευκολότερα στην λύση του συνολικότερου προβλήματος : υπάρχει λόγος να μην γίνεται η εγγραφή κατόπιν της ανάγνωσης, σύμφωνα με την προαναφερθείσα τακτική; Πράγματι, όχι! Καταρχάς, στην εντολή DQ, η απαίτηση να ακολουθεί η ανάγνωση της NextMem, την παραγωγή των σημάτων κατάστασης, είναι υπερβολικά αυστηρή. Μπορεί μεν μία άδεια ουρά να μην είχε νόημα να διαβαστεί, αλλά, καθώς πρόκειται για ανάγνωση, από γνωστή διεύθυνση, ενός δεδομένου που στη συνέχεια θα αποτραπεί το να αλλάξει την κατάσταση του συστήματος, δεν είναι περισσότερο από μια ανώδυνη σπατάλη. Αντίθετα, η πρώιμη εγγραφή, έστω του null, θα είχε καταστροφικά αποτελέσματα. Η επιλογή για την τοποθέτηση της ανάγνωσης της NextMem στο τρίτο στάδιο, είχε σκοπό την ομοιομορφία του μονοπατιού,

[4] στην περίπτωση Ουράς, θα έγραφε σε άλλη διεύθυνση, χωρίς όμως να διευκολύνεται το ζήτημα χρονισμού για την προώθηση



Σχήμα 3.22: Νέα εκδοχή μονοπατιού δεδομένων, για την DQ (v2) .

λόγω επικράτησης του μεγαλύτερου βάρους της, από την περίπτωση της NQ. Όμως, και αυτό μπορεί να θεωρηθεί κοντόφθαλμη εκτίμηση.

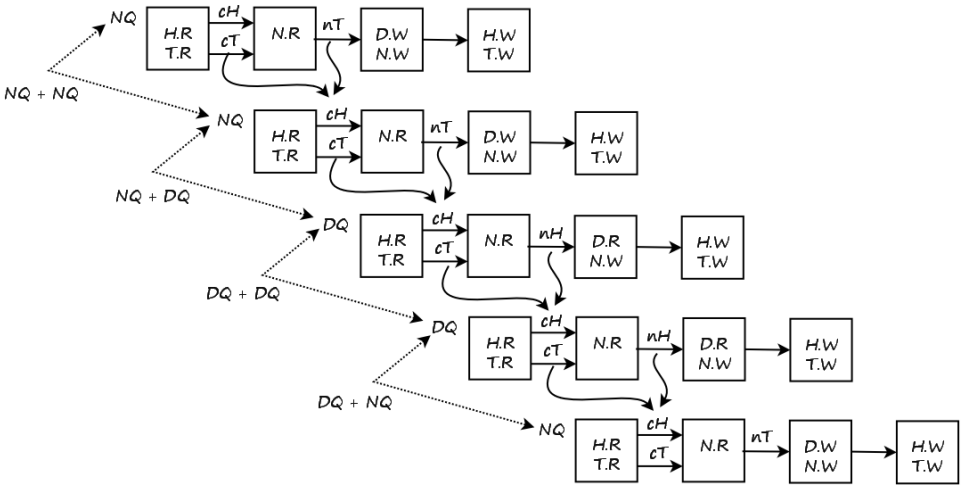
Η λειτουργική μονάδα και στις δύο ενέργειες, την εισαγωγή στοιχείου στην Free List και την ανάγνωση της διεύθυνσης του επόμενου στοιχείου της ουράς, είναι η NextMem. Μπορούν λοιπόν να αναδιαταχθούν οι δύο ενέργειες, με όλα τα δομικά στοιχεία να παραμένουν στα ίδια στάδια με πρώτα, χωρίς ουσιαστική επέμβαση στη σχεδίαση, παρά μάλλον στον τρόπο που αναγνωρίζεται η ομοιομορφία του μονοπατιού. Στη δεύτερη εκδοχή [5] του κοινού μονοπατιού, αντί να θεωρείται ότι στο δεύτερο στάδιο ανήκει η λειτουργία της Free List, και στο τρίτο η λειτουργία της NextMem, θα ισχύει το ουσιαστικότερο, να είναι η ανάγνωση της NextMem στο δεύτερο στάδιο, και η εγγραφή της στο τρίτο, το οποίο μεταφράζεται σε αντίστοιχες λειτουργίες κατά περίπτωση εντολής. Για την NQ, χωρίς αλλαγή, η λειτουργία της Free List στο δεύτερο στάδιο, και για την DQ, πλέον, στο τρίτο, και αντίστροφα η λειτουργία της NextMem για καθεμιά τους. (Σχήμα 3.22).

Η αναθεώρηση της δομής του μονοπατιού, λύνει και το πρόβλημα χρονισμού της προώθησης. (Σχήμα 3.23) Με την νωρίτερη παραγωγή του newHead, αποτρέπεται η εισαγωγή φυσαλίδων, ενώ επιτρέπεται η ύπαρξη του 'ασφαλούς σημείου', από την άποψη της παραγωγής των τιμών των δεικτών. Παραμένει η ανάλυση της ολοκλήρωσης, και κατάστασης της ουράς. (Κεφάλαιο 5).

3.3.3 Κίνδυνοι στη μνήμη Next

Οι κίνδυνοι RAW ελλοχεύουν όπου, εξαιτίας της επικάλυψης, υπάρχει χρονική αναδιάταξη ενεργειών ανάγνωσης και εγγραφής. Οι υπόλοιπες δομικές

[5] 2nd version (v2)



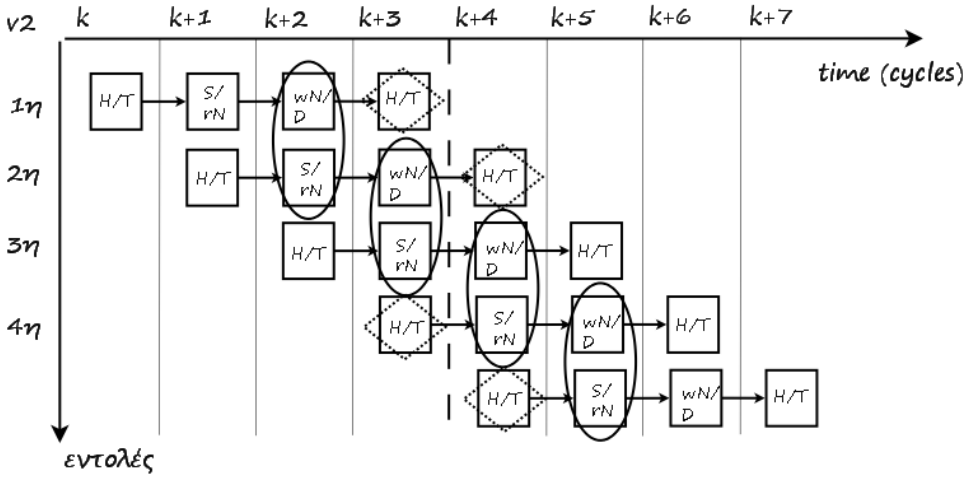
Σχήμα 3.23: Λύση : προώθηση χωρίς καθυστερήσεις (v2) .

μονάδες του συστήματος, εκτός από τα λίγα στοιχεία ελέγχου, είναι οι μνήμες των δεικτών και δεδομένων των στοιχείων των ουρών, η NextMem, που φιλοξενεί παράλληλα την δομή του διαχειριστή θέσεων, και η DataMem. Μετά την αναθεώρηση της σχεδίασης του μονοπατιού, η επικάλυψη έχει, μερικώς, διαφοροποιηθεί. (Σχήμα 3.24 ως προς 3.16).

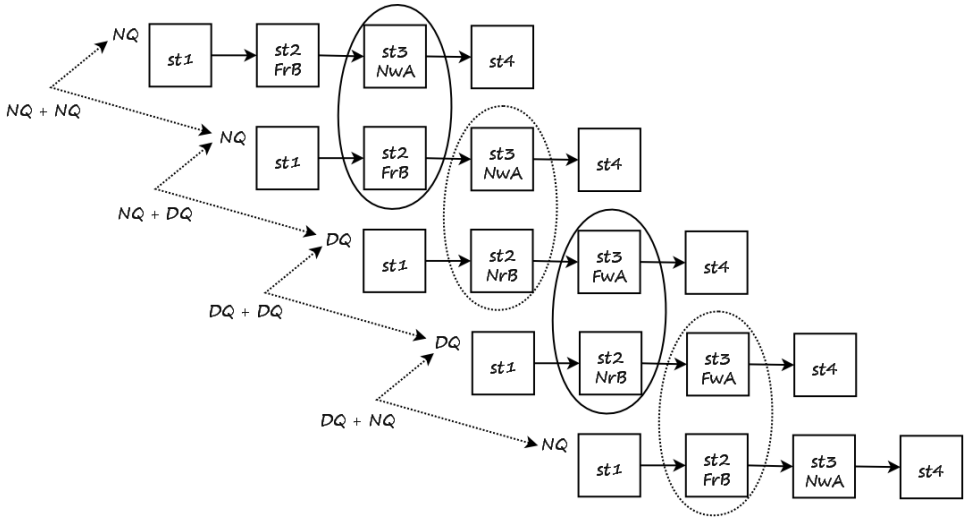
Για την DataMem δεν υπάρχει τέτοιο ζήτημα : καθώς η λειτουργία της περιορίζεται σε ένα μόνο στάδιο του μονοπατιού, σε κάθε κύκλο που περνά, κάθε νέα πρόσβαση θα ανήκει σε επόμενη εντολή. Συνεπώς, οι προσβάσεις γίνονται με την ίδια σειρά, αλλά διαφορετική χρονική απόσταση, που θα γινόταν και χωρίς την επικάλυψη, οπότε δεν υπάρχει χρονική αναδιάταξη των ενεργειών, ικανή να προκαλέσει την εμφάνιση κινδύνων. Για την NextMem, όμως, τα πράγματα είναι διαφορετικά.

Σε αντίθεση με τις μνήμες HeadMem και TailMem, των οποίων οι λειτουργίες στο μονοπάτι απέχουν κατά δύο στάδια, καταλήγοντας να εμπλέκουν το σύνολο των εντολών που συνυπάρχουν στο μονοπάτι σε περίπτωση εξάρτησης, οι λειτουργίες της NextMem βρίσκονται σε διπλανά στάδια, το δεύτερο και το τρίτο. Έτσι, κατά την επικάλυψη των εντολών, στις εντολές που μεταξύ τους παρεμβάλλεται έστω και μία εντολή, η παλαιότερη θα έχει ολοκληρώσει τις προσβάσεις της στη NextMem προτού η νεότερη τις ξεκινήσει, δηλαδή θα την προσπελάσουν με την κανονική σειρά. Οι μόνες εντολές που υπάρχει χρονική αναδιάταξη των βημάτων τους, σε σχέση με την σειριακή εκτέλεση, είναι κάθε δύο διαδοχικές, με την νεότερη να είναι στο δεύτερο της στάδιο, την στιγμή που η προηγούμενή της εκτελεί το τρίτο. Το τρίτο στάδιο της νεότερης θα είναι, στην σωστή σειρά προς την προηγούμενή της, αλλά θα εκτελείται ταυτόχρονα με το δεύτερο στάδιο της επόμενής της, και έτσι, όλες οι εντολές θα εμπλέκονται ανά δύο στον ενδεχόμενο κίνδυνο.

Κίνδυνος πρόκειται να εκδηλωθεί, εφόσον υπάρχει εξάρτηση. Όπως και



Σχήμα 3.24: Η επικάλυψη με το αναθεωρημένο μονοπάτι (v2) .



Σχήμα 3.25: Επικάλυψη δεύτερου και τρίτου σταδίου.

πριν, η εξάρτηση προκύπτει λόγω πρόσβασης στην ίδια διεύθυνση της μνήμης. Πρόκειται για την ίδια τιμή, μεν, αλλά αυτή την φορά, είναι δυνατόν να εντοπίζεται σε περισσότερες από μία μεταβλητές του συστήματος, απαιτώντας λεπτομερέστερη ανάλυση. Και πάλι θεωρώντας και τις δύο εντολές ολοκληρώσιμες, και έχοντας τα στοιχεία για να καθοριστεί η κατάσταση της ουράς τους, οι ανά δύο συνδυασμοί των δύο εντολών δίνουν τέσσερις βασικές περιπτώσεις. (Σχήμα 3.25).

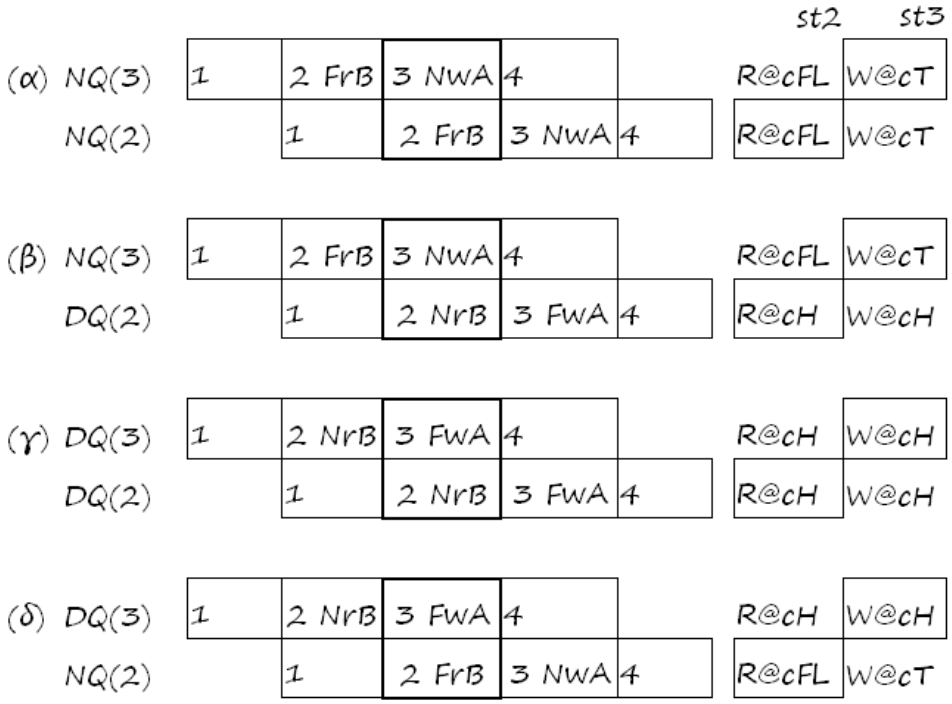
Δείχνοντας στην ίδια θέση μνήμης, σε αντίθεση με την σειριακή εκτέλεση, κατά την επικάλυψη των σταδίων που ενεργούν στην NextMem, θα ήταν αδύνατο να διαβάσει η νεότερη εντολή εκείνο που η προηγούμενή της γράφει, στον ίδιο κύκλο. Παρότι δίθυρες, οι μνήμες δεν γράφουν και διαβάζουν την ίδια διεύθυνση και στις δύο θύρες, κάτι που ονομάζεται σύγκρουση διευθύνσεων (address collision). Έτσι θα εμφανιζόταν κίνδυνος, που θα μπορούσε όμως εύκολα να αντιμετωπιστεί με προώθηση από κει και έπειτα. Αν η ανίχνευση των αναγκαίων συνθηκών για αυτή την προώθηση είναι δυνατή, και υπάρχει ανάγκη, τα δεδομένα για προώθηση είναι σίγουρα διαθέσιμα, αφού υπάρχουν ήδη έτοιμα για εγγραφή από την γηραιότερη των δύο εντολών, που βρίσκεται πάντα στο τρίτο στάδιο. Συνεπέστερα προς την ορολογία, πρόκειται ουσιαστικά για παράκαμψη, bypassing, παρά για προώθηση. Καθώς τα περιεχόμενα της μνήμης NextMem είναι δείκτες προς τον εαυτό της, η προωθούμενη τιμή θα είναι ένας ήδη υπάρχων δείκτης, από κάποια πρωτύτερη ανάγνωση στην παλαιότερη εντολή, προς την μεταβλητή new της νεότερης εντολής, που είναι οι προορισμοί των αναγνώσεων.

Στο δεύτερο στάδιο, που γίνονται αναγνώσεις στην NextMem, η μεταβλητή που δίνει τη διεύθυνση ανάγνωσης για εντολή NQ είναι η currentFreeList, δηλαδή η κορυφή της στοίβας FreeList, ενώ για εντολή DQ είναι η currentHead, το στοιχείο κεφαλής μίας συγκεκριμένης ουράς. Αντίστοιχα, στο τρίτο στάδιο των εγγραφών της NextMem, η μεταβλητή που δίνει τη διεύθυνση εγγραφής για εντολή NQ είναι η currentTail, δηλαδή το τελευταίο στοιχείο συγκεκριμένης ουράς, ενώ για εντολή DQ είναι η currentHead, και πάλι. (Σχήμα 3.26). Θα πρέπει να διερευνηθεί κατά πόσο είναι δυνατόν, και αν είναι, υπό ποιές προϋποθέσεις, να έχουν την ίδια τιμή, σε μία υποθετική σειριακή εκτέλεση.

Η πιθανότητα σύγκρουσης των διευθύνσεων ανάγνωσης και εγγραφής, ανά συνδυασμό των δύο διαδοχικών εντολών, εκτιμάται βάσει της υπόθεσης ότι κάτι τέτοιο ισχύει (συνοπτικά, Σχήμα 3.27) :

(α) για NQ(3)+NQ(2) θα έπρεπε να είναι cT(3)==cFL(2). Για να συμβαίνει αυτό θα έπρεπε το τελευταίο στοιχείο της ουράς της παλιότερης εντολής, να βρεθεί απελευθερωμένο στη FreeList στην νεότερη. Αυτό δεν είναι δυνατό να συμβεί, υπό οποιεσδήποτε συνθήκες, καθώς η παλαιότερη εκτελεί μία εισαγωγή στοιχείου, και όχι απελευθέρωση. Άρα, η υπόθεση είναι άτοπη, και δεν προκύπτει κίνδυνος.

(β) για NQ(3)+DQ(2) θα έπρεπε να είναι cT(3)==cH(2). Καθώς και οι δύο είναι δείκτες ουράς, και οι εντολές διαδοχικές, τίθεται η επιπλέον



Σχήμα 3.26: Συνδυασμοί εντολών ανά δύο, στην NextMem .

	σε επικάλυψη	υπόθεση	συμπέρασμα	προωθείται
(α)	NQ(3) + NQ(2)	cT(3) = cFL(2)	άτοπον	-
(β)	NQ(3) + DQ(2)	cT(3) = cH(2)	Nm_bypass**	nT(3)-->nH(2)
(γ)	DQ(3) + DQ(2)	cH(3) = cH(2)	άτοπον	-
(δ)	DQ(3) + NQ(2)	cH(3) = cFL(2)	FL_bypass	cH(3)-->nT(2)

** όταν Qi(3)==Qi(2) και one(3)

Σχήμα 3.27: Προωθήσεις στην NextMem.

προϋπόθεση, να αναφέρονται και οι δύο στην ίδια ουρά (Qi(3)==Qi(2)). Ακόμα και έτσι, γενικά, οι δύο αυτοί δείκτες διαφέρουν, εκτός από μία συγκεκριμένη διαδοχή καταστάσεων : για να είναι κάποιο στοιχείο τελευταίο σε μία εντολή εισαγωγής, και στην αμέσως επόμενή της να είναι πρώτο, σημαίνει ότι στην παλαιότερη ήταν το μοναδικό στοιχείο (cH(3)==cT(3) δηλαδή one(3)==1). Η νεότερη εντολή θα έχει για κορυφή της εκείνο το στοιχείο, και για ουρά της το στοιχείο στην θέση newTail της παλαιότερης εντολής. Αυτό, είναι το δεδομένο εγγραφής της NQ, και εκείνο που θέλει να αναγνώσει η DQ, ως δικό της newHead, αλλά δεν προλαβαίνει. Έτσι, όταν η ουρά και των δύο εντολών είναι κοινή, και για την πρώτη είχε ένα μόνο στοιχείο, τότε ο δείκτης nT(3) θα πρέπει να προωθηθεί, στο nH(2). [6]

(γ) για DQ(3)+DQ(2) θα έπρεπε να είναι cH(3)==cH(2). Κι εδώ, θα έπρεπε οι δύο εντολές να αναφέρονται στην ίδια ουρά. Για να είναι η κεφαλή της ουράς η ίδια, μεταξύ δύο εντολών DQ, θα πρέπει η πρώτη να αποτύχει. Σε αυτή την περίπτωση, όμως, ούτε και θα έγραφε στο τρίτο της στάδιο, άρα και πάλι, δεν θα υπήρχε ο κίνδυνος. Άρα, η υπόθεση είναι άτοπη.

(δ) για DQ(3)+NQ(2) θα έπρεπε να είναι cH(3)==cFL(2). Σε κάθε επιτυχημένη εντολή DQ, απελευθερώνεται η κεφαλή της ουράς, και γίνεται, για την οποιαδήποτε επόμενη εντολή, το στοιχείο στην κορυφή της FreeList. Αν η επόμενη εντολή είναι NQ, τότε ζητάει να διαβάσει την διεύθυνση αυτή ως nT, από τον καταχωρητή που δείχνει στην κεφαλή της στοίβας, αλλά και εκείνος, με την σειρά του, όντας ακμοπυροδότητος, δεν θα έχει λάβει στον ίδιο κύκλο την νέα τιμή του με εγγραφή από την προηγούμενη εντολή. Συνεπώς, πρέπει να λάβει το cH(3) με παράκαμψη, σε κάθε τέτοια ακολουθία εντολών, ανεξαρτήτως ουράς που αυτές μεταβάλλουν, σε πρώτο επίπεδο, καθώς τα δύο στάδια αναφέρονται και επιδρούν σε κοινή δομή, αυτή της FreeList. Μάλιστα η παράκαμψη μπορεί να εκτελεστεί απευθείας, χωρίς η FreeList να ενεργοποιηθεί για καμία από τις δύο εντολές, ματαιώνοντας την εγγραφή στο cH(3) για την σύνδεσή του στην κορυφή της στοίβας, άρα και την ανάγκη για ανάγνωση του επομένου του, δηλαδή της αρχικής κορυφής, ως νέας από την νεότερη εντολή.

Εξαιτίας της περίπτωσης (β), το σχήμα 3.6 καθίσταται ανακριβές ως προς τη χρήση του σήματος one, το οποίο τελικά χρειάζεται και χρησιμοποιεί και η εντολή NQ, αφήνοντας την κατάσταση της FreeList να είναι το σήμα κατάστασης που ενδιαφέρει μόνο μία από τις εντολές.

[6] Ως κοντινότερη 'συγγενής', θα το προωθήσει επίσης και στο cT της νεότερης, αλλά βάσει διαφορετικού, και πιο γενικευμένου, συνόλου ελέγχων, της προώθησης.

Κεφάλαιο 4

Η Free List λεπτομερέστερα

Η FreeList και η λειτουργία της, αποτελούν την καρδιά του συστήματος. Είναι στον πυρήνα κάθε εντολής, υλοποιώντας το σημαντικότερο βήμα, αλλά και το πολυπλοκότερο στον έλεγχό του, σε σχέση με την συνολική απλότητα του συστήματος. Υλοποιήθηκε σε πλήθος διαφορετικών εκδοχών, ανάλογα με τα στάδια της σχεδίασης του συστήματος. Στο μονοπάτι πολλαπλών κύκλων, χωρίς επικάλυψη, και στην αρχική εκδοχή του (v1), σε μονόθυρη μνήμη, ενεργοποιούνταν σε ένα στάδιο. Στην δεύτερη εκδοχή, σε μονόθυρη μνήμη μεν, αλλά να ενεργοποιείται σε δύο στάδια. Στο επικαλυπτόμενο μονοπάτι, σε δίθυρη μνήμη, και κατανεμημένη και στις δύο θύρες, ενεργοποιούμενη με πιο πολύπλοκο έλεγχο, ανάλογα με τον συνδυασμό των εντολών στα δύο στάδια της NextMem. Για διευκόλυνση της υλοποίησής της και της επαλήθευσής της, αναπτύχθηκε ανεξάρτητα από το υπόλοιπο σύστημα, σε μία ελλειπτική του μορφή (stripped). Μετά την μελέτη και υλοποίηση της προώθησης για την αντιμετώπιση του κίνδυνου των μνημών HeadMem και TailMem, αναθεωρήθηκε μία ακόμη φορά, εξαιτίας της επιλογής της λειτουργίας των μνημών (εκτός της DataMem) στην αρνητική ακμή του ρολογιού. Παρακάτω παρουσιάζεται σε αυτή, την τελική της εκδοχή.

4.1 Γενική περιγραφή

Η FL, αν λειτουργούσε αυτόνομα ως μία δομή Στοίβας, δεν θα χρειαζόταν τίποτα περισσότερο από τον έλεγχο που δείχνει το πότε έχει αδειάσει από στοιχεία. Μάλιστα, θα απαιτούνταν απλά μία σύγκριση ισότητας του δείκτη της κεφαλής της με μία σταθερή διεύθυνση, συνήθως μηδενική. Όμως, εξαιτίας της αρχικοποίησης της (ενότητα 4.3), αλλά και του ιδιαίτερου της ρόλου, δεν είναι τόσο απλοϊκός ο έλεγχός της.

Από την δεύτερη σχεδίαση του μονοπατιού δεδομένων, οι λειτουργίες της FL έχουν διαμοιραστεί στις δύο θύρες της μνήμης που φιλοξενείται. Ανάλογα τον συνδυασμό εντολών που εκτελούνται σε κάθε κύκλο στα στάδια της NextMem, η FL είναι πιθανό να ενεργεί στην μία ή την άλλη θύρα, αλλά και

σε κάποιες περιπτώσεις, να είναι ανενεργή. Χρειάζεται ένα σήμα για κάθε θύρα, που να δείχνει την ανάγκη ενεργοποίησης της, σε λειτουργία FL, για τον περαιτέρω έλεγχο του διαχειριστή, αλλά και την ενεργοποίηση των θυρών.

Κατά την ανάπτυξή της αποκομμένα από το υπόλοιπο σύστημα, εκτός από το άδειασμά της, ήταν αναγκαία και η ανίχνευση της πληρότητας, μιάς και δεν υπήρχε ρεαλιστικός τρόπος να σταματάει η επαναφορά στοιχείων, που δίνονταν ως είσοδοι, και όχι απελευθερωμένα στοιχεία από κάποια χρήση τους. Ο μηχανισμός υποστήριξης του ελέγχου αυτού, μία μηχανή πεπερασμένων καταστάσεων, fsm, οι οποίες εναλλασσόταν μετρώντας τα διαθέσιμα στοιχεία που απομένουν στην FL, θεωρήθηκε πιθανά χρήσιμος για την πρόβλεψη της κατάστασης της και της ολοκληρωσιμότητας των εντολών NQ. Τελικά, λόγω της εγκατάστασης του 'ασφαλούς σημείου' στο δεύτερο στάδιο, η εντολή είχε δείξει ήδη με επισημότερο τρόπο το ίδιο αποτέλεσμα, και άρα ήταν, ευτυχώς, και εκεί πλεονάζων.

Διατηρήθηκε στην μορφή εκείνη, παρόλα αυτά, λύνοντας ένα γενικότερο πρόβλημα από το απαιτούμενο, και με την ανάλογη επιβάρυνση στο υλικό του συστήματος. Έστω και υπερβολικά, είναι σε θέση να 'γνωρίζει' το κατά πόσο μπορεί να ανταποκριθεί στις αιτήσεις των εντολών, παρά την ενεργοποίησή της από το προαναφερθέν κομμάτι του ελέγχου. Συνεκτιμώντας τους δύο αυτούς παράγοντες, αποφασίζει την λειτουργία της, ή μη, αποδίδοντας ένα ενδεικτικό σήμα, επιπλέον της μεταβολής της εσωτερικής της κατάστασης και του ενδεχόμενου νέου στοιχείου που κατένειμε.

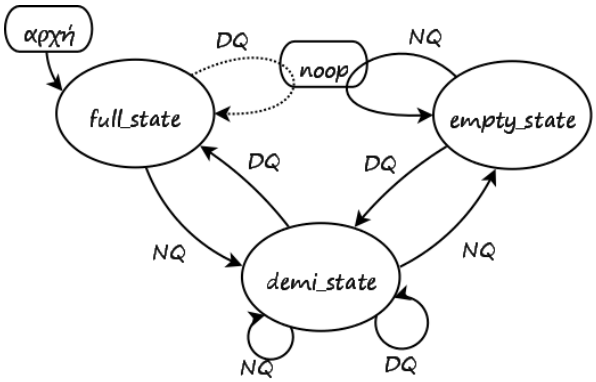
4.2 Έλεγχος

4.2.1 Εσωτερικές Καταστάσεις

Αναλυτικότερα, η FL, μέσω της fsm της, μπορεί να βρεθεί σε τρείς καταστάσεις, fl_state(s), την πλήρη, την μέση, και την αδειανή, full_state, demi_state, empty_state, όταν δεν έχει απελευθερώσει κανένα, ή αντίθετα, κάποια, μέχρι και όλα τα στοιχεία, αντίστοιχα. Οι μεταβάσεις από τις τρείς καταστάσεις, (Σχήμα 4.1), εκτός από το είδος της εντολής, έχουν κριτήριο και την ένδειξη του μετρητή, fl_elems ,όπως αυτός συγκρίνεται με μία σταθερά μεγίστου, limit_up, ίση με το σύνολο των διαθέσιμων θέσεων, αλλά για λόγους ελέγχου και αυθαίρετη, στις εισαγωγές προς την FL, είτε με την μηδενική σταθερά, στο limit_down, διαφορετικά.

Αιτήσεις για εξαγωγή στοιχείου, από NQ εντολή, αν υπάρχουν διαθέσιμα στοιχεία, θα μειώσουν τον μετρητή fl_elems, και θα μεταβάλλουν την κατάσταση, αν είναι full_state σε demi_state, ή, αν επρόκειτο για το τελευταίο διαθέσιμο στοιχείο σε empty_state, ένδειξη απαγορευτική για τις επόμενες αντίστοιχες αιτήσεις. Αντίστροφα, από την empty_state θα επανέλθει σε demi_state, αυξάνοντας και την τιμή του μετρητή, με εντολή DQ, μέχρι, ίσως, να επανακτήσει όλα τα στοιχεία, με το άδειασμα όλων των ουρών του συστήματος, και επαναφορά στην full_state. Ενδεχόμενες επιπλέον αιτήσεις εισαγωγής, θα

αρχική κατάσταση	με op = NQ	σχόλιο	με op = DQ	σχόλιο
full_state	demi_state	αρχική	NOP (full_state)	αδύνατη
demi_state	demi_state	μέση	demi_state	μέση
demi_state	empty_state	ακραία	full_state	ακραία
empty_state	NOP (empty_state)	αδύνατη	demi_state	αρχική



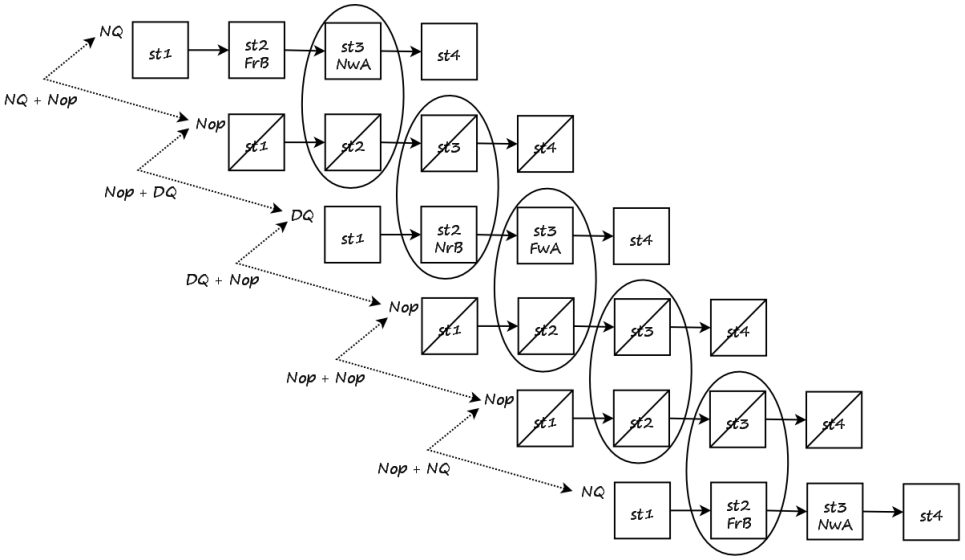
Σχήμα 4.1: Εναλλαγή εσωτερικών καταστάσεων της FreeList

απορριφθούν, παρόλο που δεν προκύπτει η ανάγκη, καθώς στο σύστημα θα έχει προηγηθεί η ανίχνευση της κενής ουράς. Βέβαια, η ένδειξη της κατάστασης full_state είναι περισσότερο περιεκτική σε πληροφορία, αφού είναι σίγουρο ότι όλες οι ουρές του συστήματος είναι κενές, σε αντίθεση με την ένδειξη μίας κενής ουράς, που δεν μπορεί να υπονοήσει τίποτα για την κατάσταση της FL, που μπορεί να είναι οποιαδήποτε.

4.2.2 Ενεργοποίηση

Η μνήμη NextMem περιλαμβάνεται σε δύο στάδια του μονοπατιού δεδομένων. Κατά την ομόχειρη εκτέλεση, όλα τα στάδια λειτουργούν σε επικάλυψη, και έτσι είναι απαραίτητη η χρήση δίθυρης μνήμης, για την ταυτόχρονη υποστήριξη του δεύτερου με το τρίτο[1] –όπως και του πρώτου με το τέταρτο, για τις HeadMem και TailMem. Η ανάθεση ενός συγκεκριμένου είδους πρόσβασης σε κάθε στάδιο, ανάγνωση στο δεύτερο και εγγραφή στο τρίτο, διασπά, δομικά, στη θύρα που η κάθε καθεμιά αντιστοιχεί, τις λειτουργίες του διαχειριστή μνήμης και του δεικτών Next. Ανάλογα με το συνδυασμό εντολών, σε κάθε κύκλο, είναι πιθανόν η FL, όπως και η διαχείριση των δεικτών Next, να ενεργεί στην μία ή την άλλη θύρα, είναι δυνατόν, όμως, και να μην ενεργοποιείται καθόλου. Τα σήματα en_FL_A_sig και en_FL_B_sig, δείχνουν αν στον κύκλο εκείνο η FL θα πρέπει να ενεργοποιηθεί στην αντίστοιχη θύρα και λει-

[1] st3//st2 στο εξής, για συντομία



Σχήμα 4.2: Επικάλυψη st3//st2, με παρουσία μη έγκυρων εντολών.

τουργία, αφήνοντας στον υπόλοιπο έλεγχό της να αποφασίσει το αν είναι σε κατάσταση να ανταποκριθεί.

Στο Σχήμα 3.25, παρουσιάζονται οι συνδυασμοί έγκυρων εντολών, στα επικαλυπτόμενα δεύτερο και τρίτο στάδιο. Όταν όμοιες εντολές επικαλύπτονται στα συγκεκριμένα στάδια, τότε, στον εν λόγω κύκλο, θα συνυπάρχει η διαχείριση της μνήμης στο ένα, με την λειτουργία των δεικτών Next στο άλλο στάδιο, έχοντας δηλαδή, την FL ενεργή σε μία από τις δύο θύρες. Αντίθετα, όταν οι εντολές στα st3//st2 είναι διαφορετικές, και οι δύο θύρες θα καταλαμβάνονται από μία ή την άλλη λειτουργία, προκαλώντας και τις επιπλοκές που πραγματεύεται η ενότητα 3.3.3. Η FL στις περιπτώσεις αυτές θα είναι ανενεργή, είτε λόγω της παράκαμψής της, FL bypass, για το συνδυασμό DQ(3)+NQ(2), που αλλιώς θα λειτουργούσε ταυτόχρονα και στις δύο θύρες, είτε λόγω ταυτόχρονων λειτουργιών σε δείκτες Next, για NQ(3)+DQ(2). Στις περιπτώσεις εμφάνισης μη έγκυρων εντολών, με valid==0, επικρατεί η λειτουργία της έγκυρης εντολής, αν τέτοια υπάρχει, ενεργοποιώντας την FL στην αντίστοιχη θύρα, όταν βρεθεί στο ανάλογο στάδιο, εκτός και αν δεν υπάρχει έγκυρη μεταξύ των δύο. (Σχήμα 4.2).

Τα σήματα που χαρακτηρίζουν κάθε εντολή, είναι το operation, για το είδος εντολής, το valid, για την εγκυρότητα, και το empty, που η καταφατική (1) τιμή του λειτουργεί ως valid==0 για τις εντολές DQ, ακυρώνοντάς τες, αλλά διατηρείται στην ανάλυση, για λόγους πληρότητας, και για τις NQ, παρότι δεν επηρεάζει την ολοκληρωσιμότητά τους. Από τα σήματα που προαναφέρθηκαν, και αντιστοιχώντας το καθένα σε πλάτος ενός bit, προκύπτουν 64 συνδυασμοί. Κατηγοριοποιώντας τις εντολές σε έγκυρες και μη, συμπεριλαμβάνοντας στις άκυρες τις (DQ, empty==1), προκύπτουν τρείς κατηγορίες, και από αυτές

περίπτωση	v3	e3	op3	v2	e2	op2	en_FL_A	en_FL_B	κωδικοποιεί
NQ(3) + NQ(2)	1	-	1	1	-	1	0	1	4
NQ(3) + DQ(2)	1	-	1	1	0	0	0	0	2
DQ(3) + DQ(2)	1	0	0	1	0	0	1	0	1
DQ(3) + NQ(2)	1	0	0	1	-	1	0	0	2
NQ(3) + NOP(2)	1	-	1	0	-	-	0	0	8
NQ(3) + NOP(2)	1	-	1	1	1	0	0	0	2
DQ(3) + NOP(2)	1	0	0	0	-	-	1	0	4
DQ(3) + NOP(2)	1	0	0	1	1	0	1	0	1
NOP(3) + NQ(2)	0	-	-	1	-	1	0	1	8
NOP(3) + NQ(2)	1	1	0	1	-	1	0	1	2
NOP(3) + DQ(2)	0	-	-	1	0	0	0	0	4
NOP(3) + DQ(2)	1	1	0	1	0	0	0	0	1
NOP(3) + NOP(2)	0	-	-	0	-	-	0	0	16
NOP(3) + NOP(2)	1	1	0	1	1	0	0	0	1
NOP(3) + NOP(2)	1	1	0	0	-	-	0	0	4
NOP(3) + NOP(2)	0	-	-	1	1	0	0	0	4
								6 bits -->	64

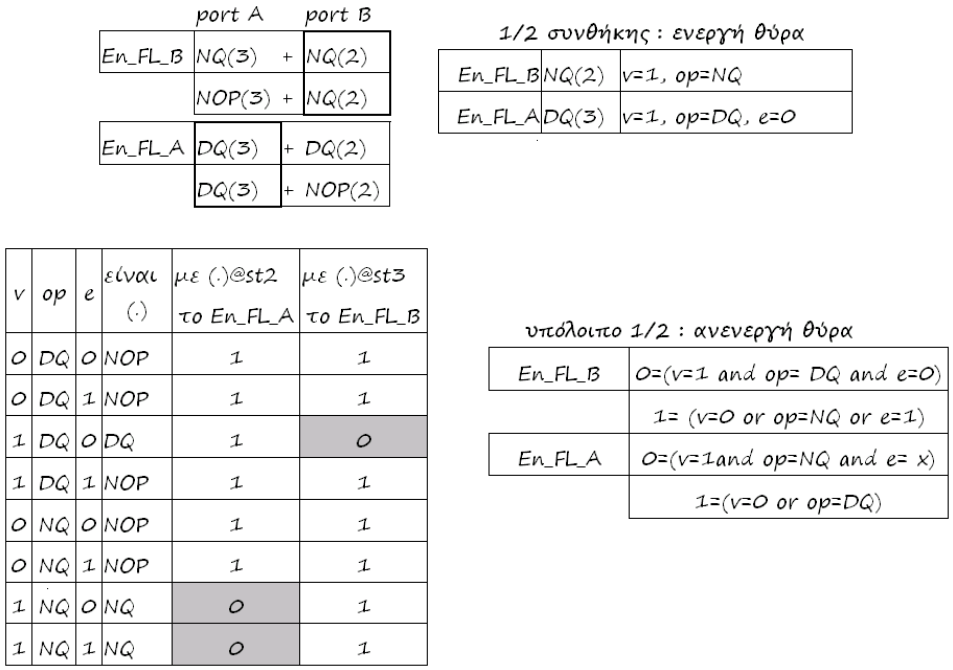
(- : don't care)

Σχήμα 4.3: Ανάλυση περιπτώσεων στην επικάλυψη st3//st2.

εννιά συνδυασμοί τους, που είναι γενικεύσεις των παραπάνω, θεωρώντας κάποιες τιμές ως αδιάφορες. Για valid==0, τα υπόλοιπα δύο σήματα, και για (valid==1, NQ) το empty, είναι όροι αδιαφορίας. (Σχήμα 4.3).

Διακρίνοντας τις περιπτώσεις που είναι ενεργή η κάθε θύρα, η ελαχιστοποίηση των συναρτήσεων των σημάτων en_FL_A_sig και en_FL_B_sig, απαιτεί, επιπλέον της περιγραφής του σταδίου στο οποίο ενεργοποιείται η θύρα, και το στάδιο που είναι εκείνο τον κύκλο σε επικάλυψη, αλλά ανενεργό ως προς την λειτουργία της FL. (Σχήμα 4.4). Όπως προαναφέρθηκε, πρόκειται για την ίδια εντολή ή μία άκυρη, που δεν δρουν ανταγωνιστικά προς την ενεργοποίηση του άλλου σταδίου. Η παράκαμψη για DQ(3)+NQ(2) έχει συμπεριληφθεί, σαν περίπτωση όπου και οι δύο θύρες είναι απενεργοποιημένες, καθώς η FL δεν χρειάζεται περισσότερη πληροφορία για αυτό το ζήτημα.

Η FL θα χρησιμοποιήσει τα En_FL_a/b_sig(s) για να διακρίνει αν και



Σχήμα 4.4: Σήματα ενεργοποίησης της FL στην καθεμία θύρα

ποιά λειτουργία αιτείται να επιτελέσει, την κατανομή μίας θέσης, ή την απελευθέρωσή της. Όπως κάθε αίτηση, μένει να επικυρωθεί και να εκτελεστεί, και αν στην περίπτωση της απελευθέρωσης είναι αναμενόμενο, στης κατανομής δεν είναι. Έτσι, χρειάζεται ένα σήμα, το fl_move, το οποίο να ανακοινώνει την απόφαση σε όλο τον κόσμο : και εσωτερικά, στην FL, για να ενημερώσει τις δομές της, και εξωτερικά, στο σύστημα, για να ληφθούν οι αντίστοιχες αποφάσεις συνέχισης της εκτέλεσης της εντολής.

Φυσικά, η επικαλυπτόμενη λειτουργία, διασπασμένη στα δύο στάδια και τις δύο θύρες, μαζί και με την παράκαμψη, εισάγουν και εδώ πολυπλοκότητα. Για την εσωτερική χρήση της FL, αρκεί ένα ενιαίο σήμα fl_move, αλλά όχι και για εξωτερικά. Εκεί, η κάθε εντολή πρέπει να μεταφέρει μαζί της, σε κάθε στάδιο τα σήματα που την αφορούν, και πρέπει να γίνει διάκριση σε ποιό στάδιο έγινε αυτή η επικύρωση, και όχι μόνο σε ποιόν κύκλο, καθώς σε καθένα κύκλο εκτελούνται όλα τα στάδια. Χρειάζονται διαφορετικά σήματα για το κάθε στάδιο λοιπόν, τα FL_move_out_3 και FL_move_out_2.

Στα στάδια που έχει ήδη εκτελεστεί η FL, παραλαμβάνεται η τιμή από τον αντίστοιχο καταχωρητή ομοχειρίας του προηγούμενου σταδίου, ενώ στις εντολές που εκτελούνται στον τρέχοντα κύκλο, δίνεται, στο σήμα του σταδίου που βρίσκεται, η τιμή του fl_move. Αυτό συμβαίνει σε όλες τις περιπτώσεις που είναι η FL ενεργή. Στην περίπτωση της παράκαμψής της, δεν ενεργοποιείται μεν, αλλά το σύστημα χρειάζεται να ενημερωθεί για το αντίθετο, ότι και οι δύο εντολές ικανοποίησαν τα αιτήματα τους για διαχείριση μνήμης, και έτσι, δίνεται

st3	st2	FL_move_out_3	FL_move_out_2
NQ(3)	+ NQ(2)	FL_move_in_3	fl_move
NQ(3)	+ DQ(2)	FL_move_in_3	0
DQ(3)	+ DQ(2)	fl_move	0
NQ(3)	+ NOP(2)	FL_move_in_3	0
NOP(3)	+ NQ(2)	0	fl_move
DQ(3)	+ NOP(2)	fl_move	0
NOP(3)	+ DQ(2)	0	0
NOP(3)	+ NOP(2)	0	0

DQ(3) + NQ(2)			FL_move	FL_move	
v3	e3	v2	_out_3	_out_2	ενέργεια
1	0	1	1	1	FL_bpss
1	1	1	0	fl_move	NQ
0	0	1	0	fl_move	NQ
0	1	1	0	fl_move	NQ
1	0	0	fl_move	0	DQ
1	1	0	0	0	-
0	0	0	0	0	-
0	1	0	0	0	-

Σχήμα 4.5: Σήματα επιτυχούς λειτουργίας της FL στα st3//st2

απευθείας τιμή 1 και στα δύο στάδια που συμμετέχουν. Στις υποπεριπτώσεις του συνδυασμού με operation(3)==DQ,operation(2)==NQ, αλλά διάφορες τιμές για valid(3),empty(3),valid(2), οι τιμές των FL_move_out_3/2_sig(s) διαμορφώνονται ανάλογα το ποιό στάδιο τελικά θα ενεργοποιηθεί. (Σχήμα 4.5).

4.2.3 Παράκαμψη

Είχε παραλειφθεί η ανάλυση για την ανιχνευσιμότητα των συνθηκών της παράκαμψης της FL (FL_bypass), η οποία συμβαίνει για το συνδυασμό των επικαλυπτόμενων εντολών DQ(3)+NQ(2). Η έκφραση αυτή δεν υπονοεί μόνο το είδος του operation, αλλά, όπως προαναφέρθηκε το να είναι και οι δύο έγκυρες, με valid==1, και ολοκληρώσιμες. Για την εντολή DQ η ολοκληρωσιμότητα είναι δεδομένη από την παρουσία στοιχείων στην ουρά που ενεργεί. Διαφορετικά, DQ, empty==1, ισοδυναμεί με valid==0. Αυτό, στο st3 είναι ήδη ξεκάθαρο, είτε εμπλέκεται η εντολή σε forwarding είτε όχι, από την σύγκριση των δεικτών currentHead και currentTail.

Στην γενική περίπτωση, η ολοκληρωσιμότητα της NQ εξαρτάται από την λειτουργία της FL, συνεπώς δεν είναι γνωστή παρά μετά την εκτέλεση του δευτέρου σταδίου. Παρόλα αυτά, ο συγκεκριμένος συνδυασμός εντολών, επιτρέπει την πρόγνωση : μία έγκυρη DQ, βάσει και empty==0, διαθέτει το στοιχείο εκείνο που θα καθιστούσε την NQ ολοκληρώσιμη, και είναι αυτό που θα προωθηθεί. Συνεπώς, η απάντηση είναι μία : η NQ δεν θα περιμένει την FL, για κανένα λόγο, αν η DQ μπορεί να προχωρήσει. Αντίθετα, αν μία από τις δύο εντολές είναι, με οποιονδήποτε τρόπο, άκυρη, τότε η FL θα ενεργοποιηθεί στην αντίστοιχη θύρα, και θα εκτελέσει την λειτουργία κανονικά, στο στάδιο που αναλογεί.

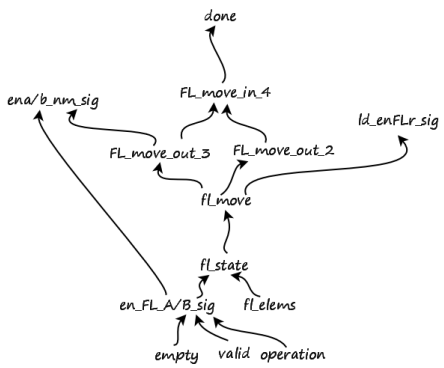
4.2.4 Σήματα

Το σύνολο των σημάτων της FL συμπληρώνουν, επιπλέον όσων προαναφέρθηκαν, τα σήματα του καταχωρητή της κεφαλής της στοίβας. Επίσης, με την λειτουργία της σχετίζεται ένα σύνολο σταθερών τιμών, είτε για τις συγκρίσεις του μετρητή fl_elems, είτε για την αρχικοποίηση των σχετικών δομών, του καταχωρητή κεφαλής και της NextMem. (Σχήμα 4.6). Οι αλληλοεξαρτήσεις των σημάτων ελέγχου, με τα σήματα στην κορυφή, να παράγονται σύμφωνα με τις τιμές των σημάτων στην αρχή των βελών (Σχήμα 4.7).

όνομα	τύπος	πλάτος	τιμή	λειτουργία
head_init	integer	-	1	η τιμή αρχικοποίησης του head_register της FL, σε 1, για αύξουσα αρχικοποίηση, παραλείποντας το στοιχείο στη διεύθυνση 0, που η τιμή του υπονοεί την κενή ουρά, και δεν γίνεται να χρησιμοποιηθεί.
limit_up	std_logic_vector	NextMem_addr_width	(<=) NextMem_depth	μέγιστο πλήθος διαθέσιμων θέσεων μνήμης (είτε το σύνολο -1, εκτός του 0 δηλαδή, είτε κάποια μικρότερη, αυθαίρετη τιμή).
limit_down	std_logic_vector	NextMem_addr_width	0	ελάχιστο πλήθος διαθέσιμων θέσεων μνήμης (εξ'ορισμού, μηδέν).
init_FLstack_Nmem_length	integer	-	NextMem_depth -2	το μήκος της αρχικοποίησης της μνήμης, σε σύνολο θέσεων που θα αρχικοποιηθούν. Είναι κατά δύο λιγότερα, από το μήκος της μνήμης, μείον την 0 και την τελευταία, που δεν διατίθεται, και δεν έχει επόμενο στοιχείο, αντίστοιχα.
fl_state_type	(ορισμός τύπου)	-	(full_state, demi_state, empty_state)	δυνατές καταστάσεις της FL σε πληρότητα στοιχείων. Είναι είτε γεμάτη, δεν έχει δώσει κανένα στοιχείο, είτε ενδιάμεσα, έχοντας δώσει μερικά, αλλά όχι όλα, είτε κενή, έχοντας τα δώσει όλα.

όνομα	τύπος	πλάτος	λειτουργία
fl_state	fl_state_type	(από τον μεταφραστή)	το σήμα που λαμβάνει τις τιμές των καταστάσεων πληρότητας της FL.
fl_elems	std_logic_vector	NextMem_addr_width	το πλήθος των διαθέσιμων στοιχείων της FL, εκείνη τη στιγμή.
fl_move	std_logic	1	αν η FL λειτούργησε, ή όχι, για οποιοδήποτε λόγο, εσωτερικό ή εξωτερικό.
en_FL_A_sig	std_logic	1	να λειτουργήσει η port a, ως FL.
en_FL_B_sig	std_logic	1	να λειτουργήσει η port b, ως FL.
nFL_sig	std_logic_vector	NextMem_Width	διεύθυνση του επόμενου στοιχείου στην κορυφή της FL.
ld_enFLr_sig	std_logic	1	να φορτωθεί η τιμή από την είσοδο του head_register της FL, στην έξοδό του.
clearFLr_sig	std_logic	1	να φορτωθεί η τιμή head_init, στην έξοδο του head_register της FL.
cFL_sig	std_logic_vector	NextMem_Width	διεύθυνση του τρέχοντος στοιχείου στην κορυφή της FL.

Σχήμα 4.6: Σταθερές και σήματα της FL



σήμα	εξαρτάται από
done	FL_move_in_4
FL_move_in_4	FL_move_out_2, FL_move_out_3
FL_move_out_2	fl_move + operation
FL_move_out_3	fl_move + operation
fl_move	fl_state, valid, operation
fl_state	fl_state, fl_elems, en_fl_sigs
ld_enFLr_sig	fl_move (πραγματικό enable)
en_FL_A/B_sig	valid, operation
ena/b_nm_sig	en_FL_A/B_sig, valid, empty, operation

αν πρέπει	και μπορεί	κινείται
valid, operation, en_FL_A/B_sig	fl_state (...)	fl_move

Σχήμα 4.7: Αλληλοεξαρτώμενα σήματα της FL

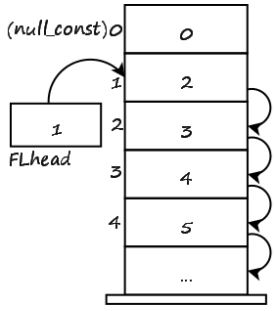
4.3 Αρχικοποίηση

Η αρχικοποίηση της NextMem είναι απαραίτητη, για την ενσάρκωση της στοίβας της FL, η οποία ξεκινά περιέχοντας το σύνολο των διαθέσιμων στοιχείων, και αδειάζει, και γεμίζει, κατά την εκτέλεση. Για το λόγο αυτό, πρέπει να αρχικοποιηθεί πριν την εκτέλεση οποιασδήποτε εντολής, αλλά και σε τυχόν επανεκκίνηση του συστήματος εν λειτουργία. Αντίστοιχα, οι μνήμες Head και Tail, πρέπει να είναι μηδενισμένες σε όλες τις θέσεις τους, για να ξεκινάνε με ουρές αδειανές, είτε αρχικά, είτε μετά από επανεκκίνηση. Η μνήμη DataMem δεν χρειάζεται και δεν λαμβάνει αρχικές τιμές.

Το σήμα reset δίνει την έναρξη της επανεκκίνησης, αλλά και της υποχρεωτικής πρώτης αρχικοποίησης, και το σύστημα λαμβάνει ένα εσωτερικό καθολικό σήμα κατάστασης αρχικοποίησης, init_state, το οποίο ανακοινώνεται και στο εξωτερικό του, με το σήμα initializing, για όλη τη διάρκεια της διαδικασίας. Όλα τα στάδια απενεργοποιούνται, και τα σήματα εισόδου αγνοούνται, εκτός από το σήμα inID, που επιτρέπεται να διασχίσει το σύστημα, όπως στην ενεργό λειτουργία, μέσω των καταχωρητών ομοχειρίας, μέχρι την έξοδο.

Η αναπαράσταση της στοίβας επιτυγχάνεται συνδέοντας όλα τα στοιχεία με δείκτες μεταξύ τους, δείχνοντας καθένα προς το επόμενό του, και έτσι, η αρχική διάταξη των στοιχείων κατά την απόδοσή τους στο σύστημα είναι αύξουσα. (Σχήμα 4.8). Καθώς η σταθερά που έχει επιλεγεί για να συμβολίζει την κενή ουρά είναι το μηδέν, η διεύθυνση αυτή πρέπει να εξαιρεθεί από τα διαθέσιμα στοιχεία, οπότε η αρχική τιμή της κεφαλής της FL είναι 1. Επιπλέον, το στοιχείο στην διεύθυνση μηδέν δεν χρειάζεται αρχικοποίηση, όπως και το στοιχείο στη μέγιστη διεύθυνση, που θα βρίσκεται αρχικά στην βάση της στοίβας.

Παρόλο που για την επιτάχυνση της αρχικοποίησης, και σε αντίθεση με τα στάδια, χρησιμοποιούνται και οι δύο θύρες για εγγραφή, η χρονική διάρκειά της παραμένει σημαντική, καθώς θα ισούται με το μισό του συνόλου των διαθέσιμων θέσεων, που συνήθως είναι αρκετές χιλιάδες, και η παράλειψη δύο θέσεων, της μηδενικής και της τελικής, την μειώνει μόνο κατά ένα κύκλο. Οι



init_Nm_index								step: +2 len: n		step: +1 len:(n-1)	
addrb	2	4	6	8	10	12	14	i+1	i	j & '0'	j & '0'
dinb	3	5	7	9	11	13	15	i+2	i+1	j & '1'	j & '1'
addra	1	3	5	7	9	11	13	i	i-1	(j & '0')-1	j_b & '1'
dina	2	4	6	8	10	12	14	i+1	i	j & '0'	j & '0'

i	2	4	6	8	10	12	14
j	1	2	3	4	5	6	7
j_b	0	1	2	3	4	5	6

$i = j \times 2$

$i - 1 = j\_b \times 2 + 1$

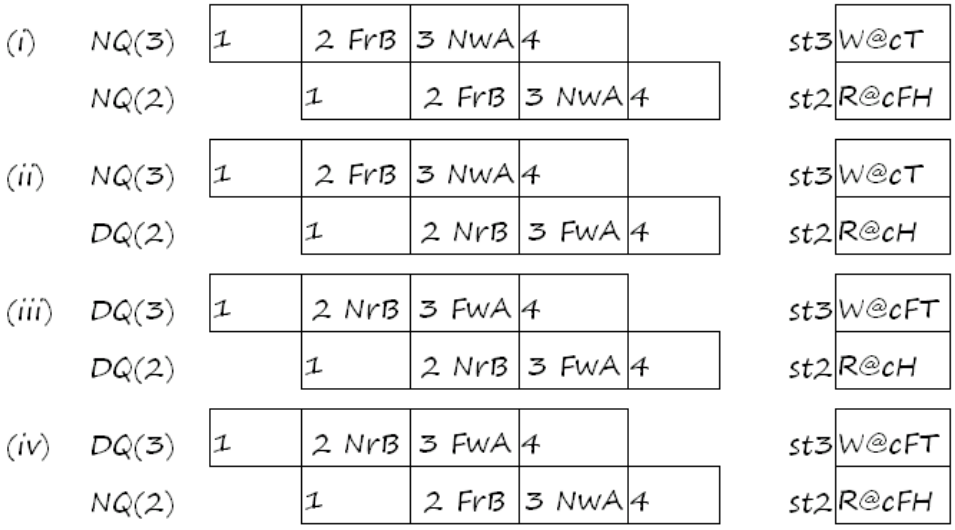
init_HTm_index									step: +2 len: n	step: +1 len:(n-1)
addrb	0	2	4	6	8	10	12	14	i	j &'0'
dinb	0	0	0	0	0	0	0	0	0	0
addra	1	3	5	7	9	11	13	15	i+1	j &'1'
dina	0	0	0	0	0	0	0	0	0	0

Σχήμα 4.8: Αρχικοποίηση των μνημών

αρχικοποιήσεις και των τριών μνημών γίνονται ταυτόχρονα, με την αναγκαία διάρκεια της NextMem να υπερκαλύπτει εκείνη των άλλων δύο. Τουλάχιστον, αυτό ισχύει στη γενική περίπτωση, που έχει αποδοθεί ολόκληρη προς χρήση, καθώς είναι δυνατή η αυθαίρετη επιλογή του πλήθους των διαθέσιμων στοιχείων προς το σύστημα, και σε μικρότερες τιμές από τη συνολική –όπου η αύξουσα αρχικοποίηση επιβάλλει, πλέον, την χρήση της fsm. (Κεφάλαιο Α΄.6). Πάντως, το σύστημα θα συνεχίσει την αρχικοποίηση για όσο είναι το μέγιστο ήμισυ διευθύνσεων.

Αρχικοποιώντας και στις δύο θύρες, η μία θα γράφει συνεχώς σε περιττές και η άλλη σε άρτιες διευθύνσεις, αυξανόμενες ανά δύο. Αντί να υπολογίζονται, όμως, με πρόσθεση του αριθμού 2, στη βασική τιμή, που συνεχώς θα άφηνε το τελευταίο bit ανενεργό, πράγμα που δεν αρέσει στο υλικό, μπορούν να κατασκευαστούν προσθέτοντάς της ένα, και συνενώνοντάς την νέα τιμή με μηδέν για τις άρτιες και ένα για τις περιττές τιμές. Έτσι, για τις μνήμες Hm/Tm εξοικονομείται και ο δεύτερος αθροιστής που θα έδινε την περιττή τιμή.

Η παράλειψη της πρώτης θέσης της μνήμης Next μεταβάλλει τα ζεύγη των διευθύνσεων που γράφονται ταυτόχρονα, αλλά, και πάλι, η κατασκευή των διευθύνσεων, αλλά και των περιεχομένων τους αυτή τη φορά, αντίθετα με τις άλλες, που απλά μηδενίζονται, πραγματοποιείται με αντίστοιχη τεχνική. Δεν εξοικονομεί τον δεύτερο αθροιστή, μόνο την εναλλαγή και του τελευταίου bit των μετρητών. (Σχήμα 4.8).



Σχήμα 4.9: Επικάλυψη σταδίων st3//st2, για FL βάσει Queue.

4.4 Εναλλακτικές

Η εναλλακτική δομή για την διάρθρωση της FL είναι η Ουρά. Αντί για τις επιμέρους, τετριμμένες, διαφορές, όπως τον εντοπισμό της με ένα δείκτη περισσότερο, τον Tail, τις συγκρίσεις για την ανίχνευση των καταστάσεων, και τα λοιπά, που δεν διαφέρουν από τις ουρές που υλοποιούνται στο υπόλοιπο σύστημα, ενδιαφέρει η μελέτη, και επακόλουθη σύγκριση, της συμπεριφοράς κατά την επικάλυψη και τις συγκρούσεις διευθύνσεων.

Καταρχάς, η FL, υλοποιημένη σε δομή Ουράς, σε εντολή του NQ θα αποδίδει στο σύστημα ένα στοιχείο, εκτελώντας μία ενέργεια DQ, και αντιστρόφως, για εντολή DQ θα δέχεται το στοιχείο, εκτελώντας NQ. Έτσι, για εντολή NQ, στο δεύτερο, θα διαβάζει στο currentFLHead, και για εντολή DQ, στο τρίτο στάδιο, θα γράφει στο currentFLTail. Είναι φυσικό η διάταξη των προσβάσεων στο μη επικαλυπτόμενο μονοπάτι να είναι ίδια με την δεύτερη σχεδίαση, μιάς και διαφέρει το σημείο πρόσβασης και όχι το είδος, μεταξύ Ουράς και Στοίβας.

Όπως και στην ανάλυση της ενότητας 3.3.3 για την εκδοχή της Στοίβας, (Σχήμα 3.27), βάσει της υπόθεσης ότι υπάρχει σύγκρουση διευθύνσεων, εκτιμάται ανά συνδυασμό εντολών (Σχήμα 4.10):

(i) για NQ(3)+NQ(2) θα έπρεπε να είναι cT(3)==cFH(2). Η υπόθεση είναι άτοπη, και δεν προκύπτει κίνδυνος, για τον ίδιο λόγο με την περίπτωση (α), ότι πρόκειται σε κάθε περίπτωση για διαφορετικά στοιχεία, διαφορετικών δομών.

(ii) για NQ(3)+DQ(2) θα έπρεπε να είναι cT(3)==cH(2). Πρόκειται για την ίδια περίπτωση με την (β), καθώς δεν υπάρχει εμπλοκή της FL, αλλά

FL.Q	σε επικάλυψη	υπόθεση	συμπέρασμα	προωθείται
(i)	NQ(3) + NQ(2)	cT(3) = cFH(2)	άτοπον	-
(ii)	NQ(3) + DQ(2)	cT(3) = cH(2)	Nm_bypass**	nT(3)-->nH(2)
(iii)	DQ(3) + DQ(2)	cFT(3) = cH(2)	άτοπον	-
(iv)	DQ(3) + NQ(2)	cFT(3) = cFH(2)	FL_bypass***	cH(3)-->nT(2)

** όταν Qi(3)==Qi(2) και one(3) *** όταν fl_one(3)

Σχήμα 4.10: Παρακάμψεις της NextMem, για FL βάσει Queue.

μόνο των ουρών του συστήματος, συνεπώς προκύπτει όταν αναφέρονται στην ίδια ουρά, και η κατάσταση στην NQ ήταν one, οπότε και απαιτείται παράκαμψη του nT(3) προς το nH(2).

(iii) για DQ(3)+DQ(2) θα έπρεπε να είναι cFT(3)==cH(2). Κάνοντας η παλαιότερη εντολή την απελευθέρωση του στοιχείου της γράφει σε ένα ήδη διαθέσιμο στοιχείο στην FL, και το στοιχείο όπου η νεότερη διαβάζει, ανήκει, εκείνη την στιγμή ακόμα, σε μία ουρά, άρα είναι διαφορετικά, σε κάθε περίπτωση, και άτοπη η υπόθεση.

(iv) για DQ(3)+NQ(2) θα έπρεπε να είναι cFT(3)==cFH(2). Όπως και στην περίπτωση (β)/(ii), είναι δυνατόν να αναφέρονται σε ίδια διεύθυνση, μίας και η ταύτιση της ουράς εδώ είναι δεδομένη, αν κατά την παλαιότερη εντολή η FL διέθετε ένα εναπομείναν στοιχείο, το οποίο, μετά την απελευθέρωση ενός ακόμα, από την ολοκλήρωσή της, συνέχισε να είναι, με τη μία από τις δύο του ιδιότητες προηγουμένως, στην κεφαλή. Έτσι, θα απαιτούνταν παράκαμψη του cH(3) στο nT(2), αλλά μόνο όταν έμενε ένα στοιχείο διαθέσιμο, και όχι στην γενική περίπτωση όπως για την αντίστοιχη περίπτωση (δ).

Η ομοιότητα αυτή των δύο εκδοχών, αλλά και η λεπτή τους διαφορά, δίνουν και πάλι το προβάδισμα στην δομή Στοίβας. Βάσει της ομοιότητάς τους, οι έλεγχοι που θα απαιτούνταν για την ανίχνευση της παράκαμψης για DQ(3)+NQ(2), είναι οι ίδιοι, ως προς τις εντολές, με επιπλέον έλεγχο της κατάστασης της FL, συνεπώς δεν θα συνεισέφεραν σε καμία απλοποίηση του συστήματος. Η διαφορά είναι ότι η παράκαμψη αυτή θα προέκυπτε σπανιότερα, πράγμα το οποίο, παρά την θετική του χροιά, παραμένει κατώτερο : με τους ίδιους, και περισσότερους ελέγχους, θα ενεργοποιεί τη NextMem για το συντριπτικό ποσοστό αυτού του συνδυασμού εντολών, σε αντίθεση με την Στοίβα, που έτσι εμφανίζεται ενεργειακά οικονομικότερη. Ο ίδιος ο συνδυασμός αυτός, δεν είναι καθόλου σπάνιος, αντιθέτως, μιάς και δεν υπόκειται στον περιορισμό της αναφοράς σε ίδια ουρά, προκύπτει σε κάθε εναλλαγή εντολών, από DQ σε

NQ, αν βέβαια είναι και οι δύο έγκυρες και ολοκληρώσιμες, που συμβαίνει, και πάλι, στο μεγαλύτερο ποσοστό.

Επιπλέον, υπάρχει το περιθώριο της βελτιστοποίησης με ιεραρχία μνημών, με τη χρήση κάποιου μικρού buffer, που θα απέτρεπε την συνεχή πρόσβαση στην μνήμη, μειώνοντας τον χρόνο του κύκλου των σταδίων που η FL δρά. Καθώς είναι το πιθανότερο τα στάδια αυτά, λόγω του ελέγχου που περιλαμβάνεται, να είναι τα πιό χρονοβόρα, η τοπική βελτίωση αυτή θα μπορούσε να επεκταθεί σε όλο το σύστημα, μειώνοντας τον συνολικό κύκλο, αν κάποιο από τα δύο στάδια ήταν το πιο αργό, και τον είχε καθορίσει. Παρότι ακόμα και η δομή της Ουράς θα μπορούσε να συνυπάρξει με μία τέτοια επέκταση της σχεδίασης, μιάς και χαρακτηρίζεται και αυτή από ισχυρή τοπικότητα, η οποία είναι όμως εντοπισμένη σε διαφορετικό χώρο ανάλογα με την πρόσβαση στη δομή, για ανάγνωση ή εγγραφή, η χρήση buffer ευνοείται κυρίως από την ανυπέρβλητη χωρική τοπικότητα της Στοίβας.

Κεφάλαιο 5

Το σύστημα λεπτομερέστερα

Το σύστημα, εκτός από το κομμάτι του διαχειριστή θέσεων μνήμης FL, είναι ένα σύνολο μνημών, οι οποίες γράφονται και διαβάζονται, βάσει μερικών πολύ απλών ελέγχων των εισόδων και των σημάτων κατάστασης. Όλοι οι έλεγχοι είναι λογικές συνθήκες ή ισότητας, ακόμα και για την παραγωγή των σημάτων κατάστασης, και κανενός είδους επεξεργασία δεν εφαρμόζεται εκτός αυτών, είτε στα σήματα που εισάγονται στο σύστημα, είτε σε εκείνα που διαβάζονται. Παρόλα αυτά, το σύστημα πρέπει να αποσαφηνιστεί λίγο περισσότερο από τις μέχρι εδώ γενικόλογες περιγραφές, μιάς και πρόκειται για μία συγκεκριμένη υλοποίηση. Επίσης, μένουν να αναλυθούν δύο σημαντικά κομμάτια της προώθησης, το αν, και πότε, η ολοκληρωσιμότητα και η κατάσταση της εντολής είναι ανιχνεύσιμες, και το ποιές δράσεις εφαρμόζουν την καθεαυτή μέθοδο στο σύστημα.

5.1 Διάρθρωση

5.1.1 Διεπαφές

Το σύστημα λαμβάνει ως εισόδους τα σήματα που δείχνουν το είδος και την εγκυρότητα της εντολής, operation και valid, τα δεδομένα που θα εγγραφούν, για εντολή NQ, dataIn, τον αριθμό της ουράς στην οποία θα εφαρμοστούν οι εντολές, Qi, το σήμα έναρξης της αρχικοποίησης ή επανεκκίνησης, reset, και φυσικά το ρολόι για τα ακμοπυροδότητα στοιχεία, μνήμες και καταχωρητές, clk. Ακόμα, παρέχεται ένας αύξων αριθμός που συνοδεύει κάθε εντολή, για διευκόλυνση του εντοπισμού των αποτελεσμάτων της, ο inID. Δεν επιτελεί κανένα λειτουργικό ρόλο, απλά δίνεται ως είσοδος, και διασχίζει όλο το μονοπάτι, μέσω των καταχωρητών ομοχειρίας, για να είναι συγχρονισμένος με την εντολή που συνοδεύει, μέχρι την έξοδο. Είναι αποτέλεσμα μετρητή, και καθώς έχει συγκεκριμένο πλάτος, η αρίθμηση επαναλαμβάνεται μετά από την

μεγιστοποίηση του.

Στην έξοδο, μετά από την εκτέλεση της εντολής στο μονοπάτι δεδομένων, λαμβάνεται καταρχάς ένα σήμα ένδειξης ολοκλήρωσης ή μη, της εντολής, done. Αν η εντολή ήταν DQ, θα φτάσουν στην έξοδο τα δεδομένα που είχαν παλιότερα αποθηκευτεί, ως dataOut πλέον. Στην έξοδο θα ξαναβγούν, ακόμα, τα σήματα Qi και inID, ως qout και outID, έχοντας διασχίσει όλο το μονοπάτι. Το μόνο σήμα που βγαίνει στην έξοδο και δεν είναι καθυστερημένο από τους καταχωρητές της ομοχειρίας είναι το initializing, που είναι ενεργό καθόλη τη διάρκεια της αρχικοποίησης των μνημών. Όπως προαναφέρθηκε, όσες είσοδοι δίνονται κατά την διάρκεια αυτή, αγνοούνται.

Το σύστημα αποτελείται από τέσσερα στάδια (stages) ομοχειρίας, st1, st2, st3, st4, τα οποία διαχωρίζουν, χρονικά, οι ανάλογοι καταχωρητές ομοχειρίας. Σύμφωνα με την σύμβαση, λαμβάνουν τα ονόματά τους από τα στάδια μεταξύ των οποίων παρεμβάλλονται, και έτσι είναι οι r1to2, r2to3, και r3to4. Υπάρχει ένας ακόμα καταχωρητής, ο r0to1, μεταξύ των εισόδων και του πρώτου σταδίου, παρέχοντας έτσι σταθερό χρονισμό εισόδου προς και το πρώτο στάδιο, εξομαλύνοντας τις σημαντικές αποκλίσεις εισόδου, ιδίως του Qi, προς τα αντίστοιχα σήματα εντός μονοπατιού (Qi_in_2,Qi_in_3,Qi_in_4), που εισήγαγε σφάλμα ως προς την παραγωγή και λειτουργία του σήματος nearest_relative.

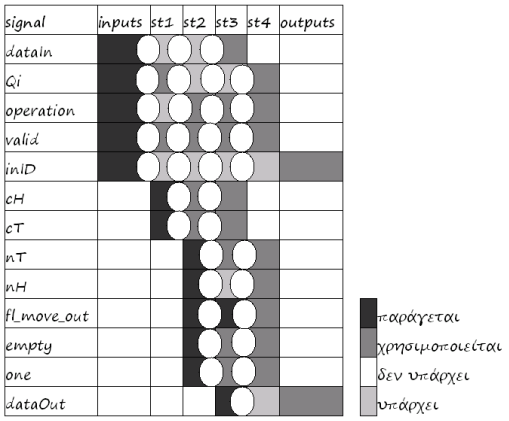
Στην πραγματικότητα, δεν πρόκειται για ενιαίους καταχωτητές, αλλά ομάδες καταχωτητών, αποτελούμενες από τόσους καταχωρητές, και τέτοιου πλάτους καθένας, όπως τα σήματα που μεταφέρονται. Είναι όμως διαισθητικά δόκιμη απεικόνιση, καθώς έχουν κατά ομάδες κοινά τα σήματα ενεργοποίησης, ld_en, και μηδένισης, clear [1]. Οι μοναδικοί καταχωρητές που έχουν διαφοροποιημένα τα σήματα αυτά, είναι εκείνοι που μεταφέρουν το inID, που είναι σκόπιμο να μεταφέρεται μέχρι την έξοδο ακόμα και όταν απενεργοποιείται η διέλευση των υπόλοιπων.

Το σύνολο των σημάτων που θα μεταφέρεται από κάθε ομάδα καταχωρητών ομοχειρίας, αποτελείται από τα σήματα που χρειάζεται η εντολή, σε οποιοδήποτε από τα επόμενα στάδια της ομοχειρίας. Τα μεταφέρει μαζί της από τη στιγμή που θα παραχθούν, σε κάποιο στάδιο, ή και από την είσοδο ακόμα, μέχρι το στάδιο που θα 'καταναλωθούν', δηλαδή θα χρησιμοποιηθούν, και μάλιστα για τελευταία φορά. Μέχρι τότε, μπορεί και να διασχίζουν και στάδια στα οποία δεν έχουν καμία χρήση.

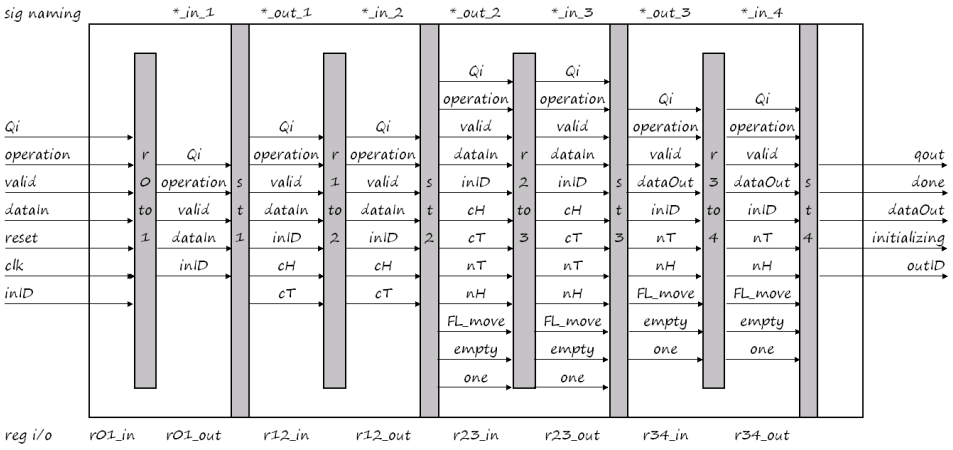
Η διάκριση των σημάτων που θα αποτελούν τους καταχωρητές ομοχειρίας, προκύπτει από τον εντοπισμό του σταδίου που κάθε σήμα θα παραχθεί και εκείνου όπου θα καταναλωθεί. Κάθε σήμα που καταναλώνεται σε διαφορετικό στάδιο από εκείνο που εμφανίστηκε, θα χρειαστεί έναν καταχωρητή, ο οποίος θα παρεμβάλλεται στα χρονικά όρια των σταδίων που το σήμα θα διαβεί. (Σχήματα 5.1 και 5.2).

Τα σήματα reset και clk δεν μεταφέρονται με καταχωρητές ομοχειρίας, μιάς

[1] όνομα που έχει αδόκιμα αποδοθεί και στο σήμα θέσης, κανονικά, του καταχωρητή κεφαλής της FL



Σχήμα 5.1: Η διάρκεια ζωής των σημάτων : από παραγωγή ως κατανάλωση



Σχήμα 5.2: Εξωτερικές και εσωτερικές διασυνδέσεις.

κατηγορία	βάση	(*) συνθετικό
είσοδοι	Qi, operation, inID, dataIn, valid	*_{in/out}_{1/2/3/4}
	reset, clk	-
έξοδοι	dataOut	*_in_4, *_out_3
	initializing, done, qOut, outID	-
αναγνώσεων/ κατάστασης	cH, cT, nH, nT, one, empty, FL_move, dataOut	*_{in/out}_{1/2/3/4}
μνημών	en, we, addr, din, dout	*{a/b}_{H/T/N/D}m_sig
καταχωρητών ομοχειρίας	ld_en, clear	{ID/r01/r12/r23 /r34}_regs_*_sig
FreeList	delayed_elemIn, fl_state, fl_elems, fl_move, en_FL_A/B_sig, nFL_sig, cFL_sig, ld_enFLr_sig, clearFLr_sig	-
αρχικοποίησης	init_Nm_index, init_Nm_index_b, init_Nm_length, init_HTm_index, init_HTm_length, init_Nm_state, init_HTm_state, init_state	-
αντίστροφο ρολόι	inv_clk	-
προώθησης	nearest_relative	-

Σχήμα 5.3: Ομάδες σημάτων.

και ο ρόλος τους είναι αρκετά ιδιαίτερος. Το reset επενεργεί στο κύκλωμα σε ένα ενδιάμεσο επίπεδο αφαίρεσης, το οποίο δεν γνωρίζει την οργάνωση του μονοπατιού σε στάδια, παρά μόνο την ύπαρξη μνημών στο σύστημα που θα αρχικοποιηθούν. Δρά ενεργοποιώντας το κύκλωμα για το σκοπό αυτό, ανεξάρτητα από το τι συμβαίνει σε επίπεδο εντολών και ουρών. Το ρολόι από την μεριά του είναι αυτό που πυροδοτεί τους καταχωρητές, και τις μνήμες, συνεπώς συνδέεται άμεσα σε όλα τα σύγχρονα στοιχεία του κυκλώματος. Το σύνολο των σημάτων, ομαδοποιημένα, Σχήμα 5.3.

5.1.2 Παραμετροποίηση

Το σύστημα, όπως και στην αρχική παρουσίαση των διαστάσεών του (ενότητα 3.2.1), δεν χαρακτηρίζεται από συγκεκριμένους αριθμούς, πράγμα εφικτό, καταρχάς, λόγω της ανάπτυξής του σε γλώσσα περιγραφής υλικού, και της προοπτικής μεταφοράς σε προγραμματιζόμενες συσκευές FPGA. Όλα του τα μεγέθη, εκτός από το αυθαίρετο σήμα μετρητή inID, είναι αλληλοσχετιζόμενα, και παράγονται βάσει τριών θεμελιωδών, και αυθαίρετων, διαστάσεων : του πλήθους των ουρών του συστήματος, Qs_multitude, του πλάτους του κομματιού δεδομένων των στοιχείων, DataElem_width, και του επιτρεπόμενου συνολικού πλήθους στοιχείων στο σύστημα, Data/NextMem_depth.

Αντί να δηλωθούν τα σήματα με συγκεκριμένες διαστάσεις στο σώμα του κώδικα σε VHDL, οι διαστάσεις δίνονται παραμετρικά, βάσει ενός συνόλου σταθερών που ορίζονται σε ένα διαφορετικό αρχείο βιβλιοθήκης *_libpack.vhd.

Οι σταθερές αυτές, τέτοιες ως προς τον ορισμό τους, λαμβάνουν είτε απευθείας κάποιες τιμές, αν πρόκειται για τις βασικές και αυθαίρετες, είτε υπολογίζουν την τιμή τους, αν είναι εξαρτώμενες παράμετροι. Επίσης, συμπεριλαμβάνονται τιμές που συμβάλλουν στην αναγνωσιμότητα του κώδικα, όπως για παράδειγμα, η std_logic := '1', που δηλώνεται ξεχωριστά, ως η NQ τιμή του σήματος operation, αλλά και ως η τιμή που θα λάβει ένα σήμα, μετά την θετική του ακμή. Με την πλήρη παραμετροποίηση των σημάτων είναι ευκολότερο να αλλάξει η διαστασιολόγηση του συστήματος άμεσα, μεταβάλλοντας μόνο τις βασικές τιμές στο αρχείο βιβλιοθήκης και χωρίς άλλη παρέμβαση στον κυρίως κώδικα. (Σχήμα 5.4).

5.1.3 Επιμέρους

Κάποιες λεπτομέρειες της υλοποίησης δεν είναι εμφανείς από τις σχεδιαστικές περιγραφές. Για παράδειγμα, οι μνήμες έχουν μόνο WE (WriteEnable) σήμα, το οποίο λειτουργεί ως RE (ReadEnable) για τιμή μηδέν, άρα, όταν θέλουμε ανάγνωση, δίνουμε γενικά την τιμή μηδέν, και όχι κάποια συνθήκη, που μπορεί κάποτε να αποτιμηθεί σε 1, και να γράψει, όπως η not one, για ανάγνωση του δείκτη Next σε εντολή DQ. Επίσης, υπάρχει ξεχωριστό σήμα EN (enable), το οποίο ενεργοποιεί και απενεργοποιεί την μνήμη, κάτι που έχει και ενεργειακό νόημα, σηματοδοτώντας, βάσει της κατάστασης, την έγκριση της εκτέλεσης της πρόσβασης που, βάσει της εντολής, έχει επιλεγεί.

Επίσης, το σύνολο των συστατικών (components) του συστήματος, βρίσκονται σε μία ενιαία αρχιτεκτονική, αντί για μία οντότητα ανά στάδιο, όπως φαίνεται λογικό, και ως δομή και ως διαδικασία ανάπτυξης και ελέγχου. Η καθεμία από τις μνήμες, τα βασικότερα συστατικά, εκτός της DataMem, συμμετέχει σε δύο στάδια. Θα ήταν πολυπλοκότερο το να διαχωριστούν σε υποσυστήματα, από το να συνυπάρχουν, άλλωστε, η μικρή έκταση του συστήματος δεν είναι απαγορευτική για κάτι τέτοιο.

5.2 Προώθηση

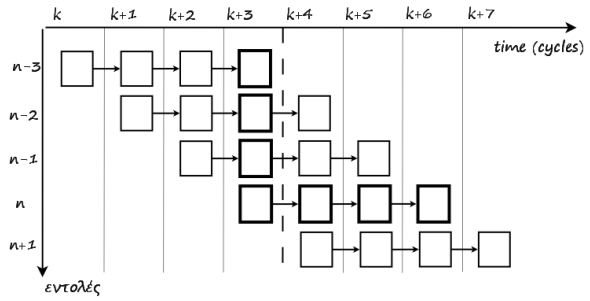
5.2.1 Εντοπισμός

Η αναθεώρηση της αρχικής σχεδίασης του μονοπατιού δεδομένων, και η αναδιάταξη των λειτουργιών της NextMem, κατέστησε δυνατή την έγκαιρη παραγωγή των νέων δεικτών, nT και nH, επιτρέποντας την εγκατάσταση του 'ασφαλούς σημείου' προώθησης μετά το δεύτερο στάδιο, για την κοντινότερη 'συγγενική', και για πριν το δεύτερο, για την εκδιδόμενη, στα πλαίσια της συγκεκριμένης συνιστώσας. Θεωρήθηκε δεδομένη η δυνατότητα ανίχνευσης της ολοκληρωσιμότητας και της κατάστασης της παλαιότερης εντολής, τα οποία είναι η άλλη συνιστώσα της ύπαρξης και εντοπισμού του 'ασφαλούς σημείου' προώθησης, και μένει να παρουσιαστούν, όπως και η επιλογή του δείκτη που θα προωθηθεί.

σταθερές					
όνομα	τύπος	τιμή		χρήση	έννοια
NQ	std_logic	1		operation	εντολή εισαγωγής στοιχείου
DQ	std_logic	0		operation	εντολή εξαγωγής στοιχείου
pos_edge	std_logic	1		register_gen/_bit(*) : edge	(τιμή μετά τη) θετική ακμή ρολογιού
neg_edge	std_logic	0		register_gen/_bit(*) : edge	(τιμή μετά την) αρνητική ακμή ρολογιού
head_init	integer	1		head_register : init	αρχική τιμή του FLhead
stage_n_0	std_logic_vector	"00"		nearest_relative	καμία σύγκρουση
stage_n_1	std_logic_vector	"01"		nearest_relative	σύγκρουση με το n-1
stage_n_2	std_logic_vector	"10"		nearest_relative	σύγκρουση με το n-2
stage_n_3	std_logic_vector	"11"		nearest_relative	σύγκρουση με το n-3
null_const	std_logic_vector	"00...00"		empty, one, (more)	μηδενική σταθερά
limit_down	std_logic_vector	"00...00"		fl_elems	ελάχιστο πλήθος στοιχείων στη FreeList

παράμετροι					
όνομα	τύπος	φόρμουλα		χρήση	έννοια
one_cycle	time	αυθαίρετη (100 ns)		χρονισμός εισόδων (testing)	περίοδος ρολογιού
half_cycle	time	one_cycle/2		χρονισμός εισόδων (testing)	ημιπερίοδος ρολογιού
unalignement_time	time	αυθαίρετη (80 ns)		χρονισμός εισόδων (testing)	εσοχή/μετατόπιση σημάτων προς την ακμή ρολογιού
inID_length	integer	αυθαίρετη (12)		in/outID sig	πλάτος σε bits του ID εντολών
Qs_multitude	integer	αυθαίρετη (16)		HTMem_depth, init_HTm_length	πλήθος ουρών συστήματος
HTMem_addr_width	integer	log2(Qs_multitude)		Qi(sigs), addra/b_H /Tm_sig(s), init_HTm_index/length	πλάτος διευθύνσεων HeadMem, TailMem
HTMem_depth	integer	Qs_multitude		-	βάθος μνημών των Head, και Tail pointers
HTMem_width	integer	DataMem_addr_width		dina/b_H/Tm_sig(s), cH, cT, nT, nH	πλάτος των Head, και Tail pointers
DataMem_addr_width	integer	log2(DataMem_depth)		addr_Dm_sig	πλάτος διεύθυνσης DataMem
DataMem_depth	integer	αυθαίρετη (16)		-	βάθος μνήμης στοιχείων (δεδομένων) DataMem
DataElem_width	integer	αυθαίρετη (16)		datain, dataOut, din/out_Dm_sig(s)	πλάτος αποθηκευόμενων στοιχείων
NextMem_addr_width	integer	DataMem_addr_width		addra/b_Nm_sig(s), fl_elems, init_Nm_index/length	πλάτος διεύθυνσης NextMem
NextMem_depth	integer	DataMem_depth		-	βάθος μνήμης στοιχείων (next pointers) NextMem
NextMem_width	integer	DataMem_addr_width		dina/b_Nm_sig, n/cFL_sig	πλάτος next pointers
limit_up	std_logic _vector	NextMem_depth-1		fl_elems	μέγιστο πλήθος στοιχείων στη FreeList
init_FLstack_ Nmem_length	integer	NextMem_depth -2		init_Nm_length	πλήθος θέσεων για αρχικοποίηση της FreeList

Σχήμα 5.4: Σταθερές και παράμετροι συστήματος.



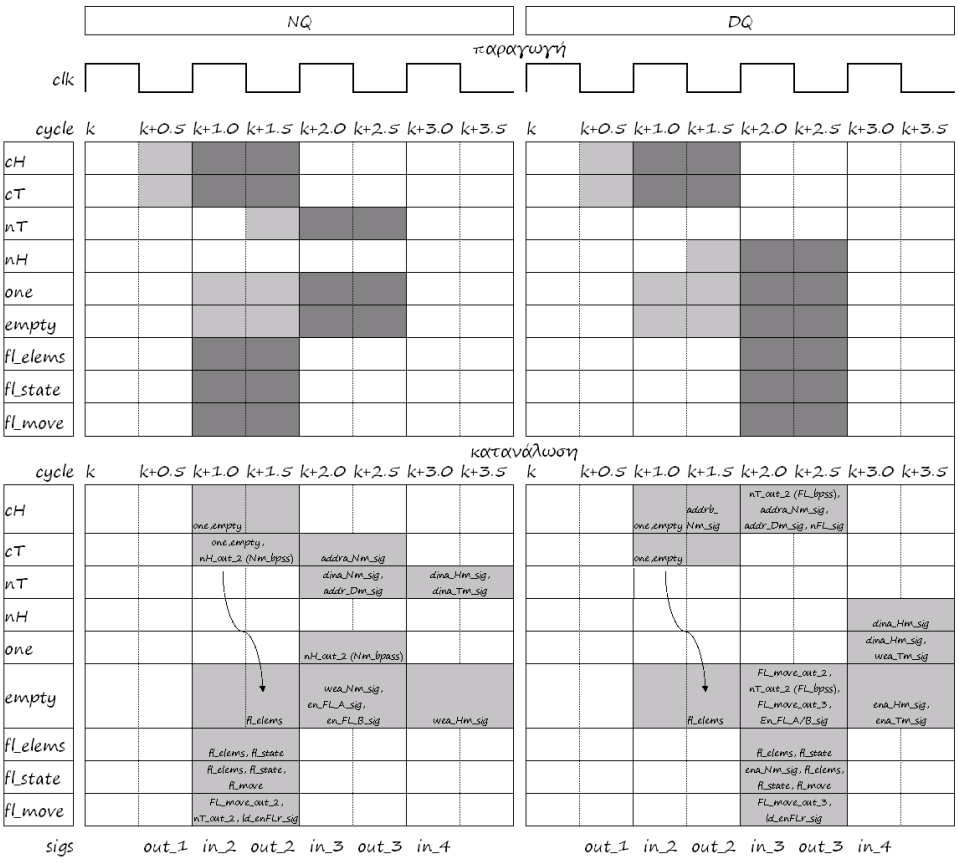
Σχήμα 5.5: Οπτική γωνία και ονοματολογία προώθησης.

Πριν από αυτό, θα πρέπει να διασαφηνιστούν κάποια σημεία που προαναφέρθηκαν. Καταρχάς, σε σημειολογικό επίπεδο, αξίζει να αναφερθεί, ότι η οπτική εξέτασης της προώθησης βρίσκεται, ή μάλλον, επιχειρείται να τοποθετηθεί, από την νεότερα εισαγόμενη στο μονοπάτι εντολή, προς τις εντολές που ήδη προϋπάρχουν αυτής. Έτσι προκύπτουν τα ονόματα των 'συγγενικών' εντολών, nearest_relative, όπως και της κατάστασης σύγκρουσης που ανιχνεύεται, ξεκινώντας με την νεότερη ως n, και τις όλο και παλαιότερες ως (n-1),(n-2),(n-3). (Σχήμα 5.5).

Επίσης, χρειάζεται να στοιχειοθετηθεί η απαίτηση της τοποθέτησης του 'ασφαλούς σημείου' πριν το δεύτερο στάδιο της εκδιδόμενης εντολής n. Βασίζεται στην ανάλυση της χρονικής αλληλουχίας παραγωγής και κατανάλωσης των σημάτων. Μελετάται το σύνολο των αλληλοσχετιζόμενων σημάτων, για να εντοπιστεί καταρχάς το σημείο που πρωτοχρησιμοποιούνται οι δείκτες cH και cT, και θα πρέπει να έχουν λάβει μέχρι τότε έγκυρη τιμή. Ακόμα, τα σήματα από τα οποία καθορίζεται η ολοκληρωσιμότητα και η κατάσταση της εντολής, είναι παράγωγα άλλων, συνεπώς η ανάλυση της αλληλεξάρτησής των σημάτων, θεμελιώνει την ανιχνευσιμότητά τους. Οι δείκτες cH, cT πρωτοχρησιμοποιούνται στην παραγωγή των σημάτων one, empty και της addrb_Nm_sig, διεύθυνσης ανάγνωσης της DQ, αλλά και για την παράκαμψη της NextMem, όλα κατά το δεύτερο στάδιο. (Σχήμα 5.6).

Η ολοκληρωσιμότητα κάθε εντολής, διαφέρει στα κριτήριά της. Ορίζεται βάσει της μη ολοκλήρωσης της εντολής, δηλαδή την άκυρη περίπτωσή της, κατά την ανάλυση περιπτώσεων της ενότητας 3.1.3. Για εντολές NQ αυτό συμβαίνει όταν δεν υπάρχουν διαθέσιμες θέσεις στοιχείων, για την εισαγωγή νέου, και για τις DQ όταν η ουρά στην οποία αναφέρονται δεν έχει κανένα στοιχείο. Σε σήματα κατάστασης, για την πρώτη είναι το να είναι η FL πλέον άδεια, σε fl_state empty, ενώ για την DQ όταν το σήμα empty==1. Αντίστροφα, προκύπτουν οι συνθήκες ολοκληρωσιμότητας, από την άρνηση των παραπάνω κριτηρίων.

Η NQ, λοιπόν, χρειάζεται την κατάσταση της FL, προτού φυσικά εκτελεστεί η ίδια. Ακόμα και ως η εγγύτερη 'συγγενής' (n-1), έχοντας τη λειτουργία αυτή στο δεύτερο στάδιο, θα την έχει ολοκληρώσει ώσπου η n χρειαστεί την



Σχήμα 5.6: Παραγωγή και κατανάλωση σημάτων.

προώθηση. Έτσι, το σήμα FL_move θα είναι σε θέση να δώσει την έκβαση της λειτουργίας. Αντίστοιχα, στη DQ, ακόμα και η (n-1), θα έχει, αν όλα συγκλίνουν, το empty της στο δεύτερο στάδιο. Συνεπώς, η ανίχνευση της ολοκληρωσιμότητας των εντολών είναι εφικτή, πριν το δεύτερο στάδιο της n.

Η κατάσταση της εντολής, το αν πρόκειται για μέση ή ακραία περίπτωση, έχει αντίκτυπο στο πλήθος των δεικτών που θα μεταβάλλει μετά την ολοκλήρωσή της. Βέβαια, και η μη ολοκλήρωση ουσιαστικά αποτελεί μία επιπλέον κατηγορία ενδιαφέροντος, καθώς και η γνώση της μη μεταβολής είναι σημαντική, λόγω της πολιτικής προώθησης και των δύο δεικτών σε κάθε περίπτωση. Τα σήματα που αποτελούν κριτήρια για τις ακραίες περιπτώσεις είναι, για την NQ το empty, και για την DQ το one, στην τιμή τους ίση με ένα. Όπως και πριν, και τα δύο παράγονται στο δεύτερο στάδιο, αν και η προώθηση το επιτρέψει, και συνεπώς, η κατάσταση μπορεί να εκτιμηθεί.

Η προώθηση, κανονικά, υπό αυτές τις συνθήκες φαίνεται να γίνεται προς το δεύτερο στάδιο της εκδιδόμενης. Στην πραγματικότητα όμως, γίνεται μέσα στο πρώτο της στάδιο, πριν τον καταχωρητή ομοχειρίας r12. Αυτό, είναι

παράγων	valid	operation	empty	one	FL_state	εκφράζει	τύπος
βάσει διάστασεων (bits)	1	1	1	1	2	64	2^(1+1+1+1+2)
δυνατές μορφές	2	2	3		3	36	2*2*3*3
δυνατές μορφές με αδιάφορους όρους	1 + 1	2	3		3	19	1+1*2*3*3

Σχήμα 5.7: Πλήθος περιπτώσεων εντολών.

εφικτό μετατρέποντας τις μνήμες σε αρνητικά ακμοπυροδότητες, για να δίνουν την έξοδό τους ως είσοδο στον καταχωρητή, που προηγουμένως δεν χρειαζόταν, καθώς λειτουργούν σαν να έχουν δικό τους καταχωρητή στην έξοδο. Αυτό, μπορεί να επιβραδύνει το σύστημα, μεν, αλλά απλοποιεί την προώθηση, δίνοντας μέσα στο δεύτερο στάδιο έγκυρες τιμές, που μπορούν να επαναπροωθηθούν, άμεσα, και χωρίς επιπλέον έλεγχο, όπως ήταν ο σκοπός. Επίσης, η αλλαγή εφαρμόστηκε και στην NextMem, με την ευχάριστη παρενέργεια της απλοποίησης του ελέγχου της FL, ως προς το χρονισμό, που χρησιμοποιούσε καταχωρητή αρνητικής ακμής.

5.2.2 Επιλογή

Αφού πλέον η προώθηση θεωρείται εφικτή, μένει να πραγματοποιηθεί. Όπως προαναφέρθηκε, είναι σκόπιμη η προώθηση και προς τους δύο δείκτες της εκδιδόμενης n, για να αποφεύγονται οι αναγνώσεις που θα απαιτούσαν εξέταση εξάρτησης περισσότερων από μία, την κοντινότερη, 'συγγενική' εντολή. Συνεπώς, οι τιμές που θα ληφθούν μπορεί, στην προωθούσα εντολή να ήταν και σε θέση current, άρα είναι απαραίτητη η εξέταση του πόσους και ποιούς δείκτες θα μετέβαλε, για την επιλογή μεταξύ current και new.

Από την ενότητα 3.1.3, είναι γνωστό ότι η μέση περίπτωση των NQ αναβαθμίζει μόνο τον δείκτη Tail, με τον nT, ενώ των DQ μόνο τον δείκτη Head, με τον nH. Στην ακραία περίπτωσή τους, και οι δύο εντολές γράφουν και στους δύο δείκτες, η NQ το nT, και η DQ το null. Έτσι, προστίθεται μία ακόμα υποψήφια στις μεταβλητές προς προώθηση, η σταθερά, όμως, null_const. Ακόμα, μιάς και current_in_4 δεν υπάρχει, θα διαβαστεί σε εκείνη την περίπτωση της (n-3) από την μνήμη, άλλωστε, δεν υπάρχει λόγος για κάτι το διαφορετικό, εκείνη δεν έχει παλαιότερες πλέον να της προωθήσουν, τα αποτελέσματά τους έχουν γραφεί.

Τα σήματα που καθορίζουν την κατάσταση και την ολοκληρωσιμότητα μίας εντολής, σε συνδυασμό και για τις δύο, είναι τα empty, one, fl_state και fl_move, σε συνδυασμό φυσικά με τα valid και operation. Συνολικά, εκφράζονται με 6 bits πλάτους σήματα, το οποίο δίνει 64 συνδυασμούς, ένα αρκετά μεγάλο αριθμό για να μελετηθεί. Στην πραγματικότητα, όμως, ο χώρος των περιπτώσεων δεν είναι τόσο εκτενής. Καταρχάς, η τιμή valid==0, καθιστά όλα τα υπόλοιπα σήματα αδιάφορα, άρα, άμεσα οι 32 περιπτώσεις εκφράζονται

a/a	valid	operation	empty	one	FL_state	FL_move	cH(n+i)	ομάδα	cT(n+i)	ομάδα	a/a
1	0	x	x	x	x	x	cH(n)	-	cT(n)	-	1
2	1	NQ	0	0	demi	1	cH(n)	a	nT(n)	i	3
3	1	NQ	0	0	empty	0	cH(n)	a	cT(n)	ii	4
4	1	NQ	0	1	demi	1	cH(n)	a	nT(n)	i	6
5	1	NQ	0	1	empty	0	cH(n)	a	cT(n)	ii	7
6	1	NQ	1	0	full	1	nT(n)	b	nT(n)	i	8
7	1	NQ	1	0	demi	1	nT(n)	b	nT(n)	i	9
8	1	NQ	1	0	empty	0	cH(n)	a	cT(n)	ii	10
9	1	DQ	0	0	demi	1	nH(n)	c	cT(n)	ii	12
10	1	DQ	0	0	empty	1	nH(n)	c	cT(n)	ii	13
11	1	DQ	0	1	demi	1	null	d	null	iii	15
12	1	DQ	0	1	empty	1	null	d	null	iii	16
13	1	DQ	1	0	full	0	cH(n)	a	cT(n)	ii	17
14	1	DQ	1	0	demi	0	cH(n)	a	cT(n)	ii	18
15	1	DQ	1	0	empty	0	cH(n)	a	cT(n)	ii	19

Σχήμα 5.8: Περιπτώσεις και υποομάδες για προώθηση.

από μόνο μία. Επιπλέον, τα empty και one, ενώ είναι 2 bit, μπορούν να βρεθούν μόνο στους τρείς από τους τέσσερις συνδυασμούς, καθώς είναι αμοιβαίως αποκλειόμενα. Παρομοίως και για το fl_state, που δίνει μόνο τρείς δυνατές καταστάσεις, όπως και να έχουν κωδικοποιηθεί. Συνεπώς, απομένουν έτσι 19 περιπτώσεις για ανάλυση.(Σχήμα 5.7).

Οι περιπτώσεις μειώνονται ακόμα περισσότερο από τις ασυμβατότητες των καταστάσεων των ανεξάρτητων σημάτων μεταξύ τους. Συγκεκριμένα, σε κατάσταση fl_state full, είναι νόμιμο μόνο όταν η κατάσταση της τρέχουσας ουράς, και ισχύει για οποιαδήποτε, είναι empty. Ο συνδυασμός της πληρότητας της FL, με την ένδειξη της ύπαρξης στοιχείου σε ουρά, δεν μπορεί να εμφανιστεί στα πλαίσια ορθής εκτέλεσης του συστήματος. Οι περιπτώσεις 2, 5, 11, 14 της αρχικής ομάδας των 19, που θα ήταν NQ/DQ με e=0,o=x και full, αποκλείονται. Έτσι, οι ελεγχόμενες περιπτώσεις μειώνονται περαιτέρω, σε 15.

Η ανάλυση των καταστάσεων, δίνει τους δείκτες που θα μεταβληθούν από κάθε δυνατή εκδοχή εντολής. Παρότι σε πρώτη όψη φαίνεται λογικό η κατηγοριοποίηση να γίνει βάσει του πλήθους, και ως προς το σύνολο και των δύο δεικτών, είναι εσφαλμένο. Κάθε current δείκτης στην εκδιδόμενη πρέπει να λάβει μία τιμή, και αυτό ανεξάρτητα από τον άλλο δείκτη, και έτσι η κατηγοριοποίηση γίνεται. Οι εντολές που θα προωθήσουν το nT τους προς το cH, είναι διαφορετικές από κείνες που θα προωθήσουν το ίδιο, προς το cT. (Σχήμα

5.8[2])

Οι υποομάδες των δεικτών, ελαχιστοποιούνται, σύμφωνα με την λογική ταυτότητα του επιμερισμού στη διάζευξη, xy + xz = x(y+z), και της ουδετερότητας των στοιχείων, καταρχάς x + 0 = x, και μετά x * 1 = x, που συνδυάζονται στο xy + xy' = x. Μετά από ελαχιστοποίηση των συναρτήσεων, προκύπτουν οι συνθήκες επιλογής δείκτη, εκπεφρασμένες με γενικευμένα σήματα, που ακόμα και αν υπάρχουν στο σύστημα, δεν αντιστοιχούν σε κάθε εντολή. (Σχήμα 5.9). Συνεπώς, ανάλογα με το ποιά είναι και η 'συγγενής', διαφοροποιούνται, με τα διαθέσιμα σήματα του αντίστοιχου σταδίου.

[2] η από τα δεξιά αρίθμηση των περιπτώσεων αντιστοιχεί στις αρχικές 19.

cH

a (cH): <2, 3, 4, 5, 8, 13, 14, 15>
=[NQ,e=0, o=0,fl_m=1 + NQ, e=0, o=0, fl_m=0 + NQ, e=0, o=1,fl_m=1 +
+ NQ, e=0, o=1,fl_m=0 + NQ, e=1, o=1, fl_m=0 + DQ, e=1, o=0,(fl_m=0)all]
=(με το μάτι) [fl_move=0 + NQ, e=0]
=[NQ,e=0 + NQ,fl_m=0 + DQ,e=1]

b (nT): <6, 7>
=[NQ, e=1, o=0, fl_m=1]
=[NQ, e=1,fl_m=1]

c (nH): <9, 10>
=[DQ, e=0, o=0,(fl_m=1)all]
=[DQ, e=0, o=0]

d (null) <11, 12>
=[DQ, e=0, o=1,(fl_m=1)all]
=[DQ,o=1]

cT

i (nT): <2, 4, 6, 7>
=[NQ,e=0,o=0,fl_m=1 + NQ,e=0,o=1,fl_m=1 + NQ, e=1,o=0,fl_m=1 + NQ, e=1,o=0,fl_m=1]
=[NQ,fl_m=1]

ii (cT): <3, 5, 8, 9, 10, 13, 14, 15 >
=[NQ, e=0,o=0,fl_m=0 + NQ,e=0,o=1,fl_m=0 + NQ,e=1,o=0,fl_m=0 +
+ DQ,e=0,o=0,(fl_m=1)all + DQ,e=1,o=0,(fl_m=0)all]
=[NQ,e=0,fl_m=0 + NQ,e=1,fl_m=0 + DQ,o=0]
=[NQ,fl_m=0 + DQ,o=0]

iii (null): <11, 12>
=[DQ, e=0, o=1, (fl_m=1)all]
=[DQ, o=1]

Σχήμα 5.9: Ελαχιστοποίηση συναρτήσεων προώθησης.

Κεφάλαιο 6

Επαλήθευση

Κατάληξη κάθε σχεδιαστικής προσπάθειας είναι η επαλήθευση του σχεδιασθέντος συστήματος. Αρχικά, παρουσιάζονται οι διάφορες μέθοδοι επαλήθευσης, η στοχοθεσία τους και τα χαρακτηριστικά τους. Ένα υποσύνολο των μεθόδων αυτών εφαρμόστηκε για το σύστημα του ομόχειρου διαχειριστή πολλαπλών Ουρών, οδηγώντας στην δημιουργία ενός αυτόματου συστήματος επαλήθευσης, το οποίο παρουσιάζεται αναλυτικότερα στο παράρτημα Α΄. Στα αποτελέσματα που παρατίθενται, εκτός από του ελέγχου, συμπεριλαμβάνονται μετρήσεις από την, σε επίπεδο εργαλείων, απεικόνιση της σχεδίασης σε προγραμματιζόμενη συσκευή αναδιατασσόμενης λογικής, FPGA .

6.1 Με σκοπό τον έλεγχο

Όπως σε κάθε σχεδιαστικό βήμα, από την μοντελοποίηση ως την υλοποίηση, έτσι και στο τελευταίο[1], της επαλήθευσης, πρέπει να ληφθούν σχεδιαστικές αποφάσεις. Συνεπώς, προϋποτίθεται μελέτη, και συγκεκριμένα, σε δύο κατευθύνσεις. Στην μία, έχουμε τις, ανεξάρτητες του προβλήματος, μεθοδολογικές επιλογές που πρέπει να διερευνηθούν, από αυστηρές μαθηματικές αναλύσεις, μέχρι ad hoc προσεγγίσεις. Στην άλλη, το ίδιο το πρόβλημα, που πρέπει να αναλυθεί υπό το πρίσμα των νέων αναγκών, για τον προσανατολισμό του ελέγχου αναλόγως, τόσο στο όλον, όσο και στο μέρος.

6.1.1 Μεθοδολογικό φάσμα

Η μόνη ουσιαστική ιδιότητα ενός συστήματος επεξεργασίας, είναι η ορθότητα. Παρότι γενικά στην επιστήμη αρκετά μεγάλα άλματα προόδου έχουν προκύψει από ευτυχείς συμπτώσεις και παρεκβάσεις, στην περιορισμένη οπτική ενός συγκεκριμένου προβλήματος, δεν έχει αξία να επιτυγχάνεται κάτι ακόμα και βέλτιστα, προς οποιοδήποτε κριτήριο απόδοσης, αν δεν λύνεται αυτό που επιδιώκεται από την αρχή! Συνεπώς, πρέπει πρωτίστως να επιβεβαιωθεί η λει-

[1] last, but not least!

τουργικότητα του συστήματος, αν δηλαδή εκτελούνται σωστά και πλήρως οι αναμενόμενες λειτουργίες.

Κατ΄ επέκταση της ορθότητας, και στον αντίποδα της προδιαγεγραμμένης λειτουργικότητας, το σύστημα δεν πρέπει να εμφανίζει απρόβλεπτες συμπεριφορές, διαφορετικές ή επιπλέον λειτουργίες από τις απαιτούμενες. Οτιδήποτε μπορεί να θεωρηθεί εσφαλμένη λειτουργία, πρέπει να αποκλειστεί. Επίσης, πρέπει να διασφαλιστεί ότι τυχόν ενδιάμεσες παρεμβάσεις στο σύστημα, έχουν μεν επιφέρει τις επιθυμητές διορθώσεις, αλλά δεν έχουν εισάγει σφάλματα εκεί που προηγουμένως δεν υπήρχαν.

Βέβαια, επιπλέον της τυπικής λειτουργικότητας, στα πραγματικά συστήματα είναι αναγκαίο να τηρούνται και κάποιες προδιαγραφές επιδόσεων και κόστους, που συνήθως αφορούν το χρόνο, το χώρο (μνήμης), το υλικό και την ενεργειακή κατανάλωση. Η διερεύνησή τους αποτελεί ακόμα μία διάσταση της επαλήθευσης.

Ένα σύστημα επεξεργασίας μπορεί να έχει διάφορους τρόπους έκφρασης.[2] Καταρχάς, εκφράζεται σε φυσική γλώσσα, συνοψίζοντας την επικοινωνία των εμπλεκομένων στην αναζήτηση της λύσης σε κάποιο συγκεκριμένο πρόβλημα. Έπειτα, μπορεί να εκφραστεί με διάφορους σχηματικούς τρόπους, είτε αυθαίρετους είτε πρότυπους, για παράδειγμα σε UML (Unified Modeling Language), για το λογισμικό, ή διαγράμματα ροής αλγοριθμικής μηχανής καταστάσεων (ASM, Algorithmic State Machine, flowcharts), για το υλικό. Έπειτα, αρχίζοντας να μορφοποιείται, μπορεί να περιγραφεί σε κάποια ψευδογλώσσα ή διάφορες πραγματικές γλώσσες, ακόμα και αν αυτές δεν αποτελούν την τελική μορφή του συστήματος. Στην περίπτωση λογισμικού η διαφορετική τελική μορφή θα ήταν, κατά πάσα πιθανότητα, μία άλλη γλώσσα, ενώ στην περίπτωση του υλικού, θα είναι σίγουρα η φυσική μορφή του κυκλώματος.

Στο υλικό, στην τελική του μορφή, χωρίς να εξαιρούνται όλοι οι προηγούμενοι στόχοι ελέγχου της λειτουργικής ορθότητας και προδιαγραφών, πρέπει να επιβεβαιωθούν και διάφορα εγγενή ζητήματα, όπως για παράδειγμα η απουσία ελαττωματικών κυκλωματικών στοιχείων, βραχυκυκλωμάτων, και λανθασμένων συνδέσεων. [12] Τότε, είναι λογικό να εφαρμόζονται κατάλληλες μέθοδοι ελέγχου, που δεν θα αναφερθούν παρακάτω.[3] Όμως, από ένα σημείο και έπειτα για το λογισμικό, ή σε κάποιο στάδιο πρωιμότερο της τελικής φυσικής υλοποίησης για το υλικό, μπορεί να έχει επιτευχθεί μία πλήρης αλγοριθμική έκφραση του συστήματος σε κάποια γλώσσα προγραμματισμού. Σε μορφή προγράμματος, εν πολλοίς, μοιράζονται αδιακρίτως τις ίδιες βασικές αρχές και μεθοδολογίες επαλήθευσης.

Δύο τρόποι υπάρχουν να επιβεβαιωθεί η ορθότητα ενός προγράμματος. Ο πρώτος είναι να αναλυθεί στατικά, με την μαθηματική μέθοδο που ονομάζεται Formal Verification. Φυσικά, απόδειξη επιτυγχάνεται αν η ίδια η ανάλυση είναι απαλλαγμένη από λάθη, το οποίο δεν είναι εξασφαλισμένο, μιάς και θα είναι

[2] Όπως στο κεφάλαιο 3.

[3] μιάς και το συγκεκριμένο σύστημα δεν φτάνει σε αυτό το στάδιο.

εκτενέστατη για οποιοδήποτε πρόγραμμα μη αμελητέου μεγέθους. Οπωσδήποτε, ακόμα και για την άψογη εφαρμογή της, είναι απαραίτητη η πρότερη εμπέδωση της μεθόδου! Λόγω των δυσκολιών αυτών, λίγα είναι τα προγράμματα που επαληθεύονται αυστηρά, τουλάχιστον άμεσα, χωρίς την βοήθεια εργαλείων που την εφαρμόζουν αυτοματοποιημένα.

Ο δεύτερος τρόπος είναι η δυναμική επαλήθευση, δηλαδή η εκτέλεση του προγράμματος, και η παρακολούθηση των αποτελεσμάτων. Ισχύς απόδειξης επιτυγχάνεται μόνο στην περίπτωση που θα χρησιμοποιηθεί το σύνολο των δυνατών εισόδων του συστήματος. Ακόμα και για τετριμμένα προγράμματα τα σύνολα αυτά είναι πολυάριθμα, καθιστώντας πρακτικά αδύνατη την εφαρμογή της μεθόδου εξαντλητικά.

Η λιγότερο αυστηρή εναλλακτική που ακολουθείται, είναι η επιλογή και εκτέλεση ενός υποσυνόλου των δυνατών εισόδων. Ακόμα και με τα αυστηρότερα κριτήρια, δεν μπορεί να αποτελέσει παρά μιά εκτίμηση, ή αρκετά ισχυρή ένδειξη, πάντως σίγουρα όχι απόδειξη ορθότητας. Βέβαια, υπάρχουν και άλλα χαρακτηριστικά, που εκτός από ελκυστική, την αναδεικνύουν και ως αποδεκτή προσέγγιση. Πλήθος απαιτήσεων διαβαθμίζουν την ισχύ της και καθορίζουν την επιλογή των δεδομένων που θα αποτελέσει το υποσύνολο εισόδων για τον έλεγχο.

Η επιλογή του υποσυνόλου των δεδομένων εισόδου, που θα αποτελέσει το σύνολο δοκιμών, επηρεάζεται και από την οπτική μας όσον αφορά το σύστημα υπό έλεγχο. Συχνά συμβαίνει στο υλικό, όταν γίνεται χρήση τμημάτων κώδικα άλλων, Intellectual Property Core(s), αλλά και ενίοτε ως αυθαίρετη προσέγγιση, να μην είναι γνωστή η εσωτερική διάρθρωση του συστήματος, να αποτελεί 'μαύρο κουτί'. [1] Παρά τον περιορισμό, υπό το πρίσμα των αναμενόμενων λειτουργιών του συστήματος, το μη εφαρμόσιμο πλήθος του πλήρους συνόλου εισόδων αποκαλύπτεται ότι δεν είναι τελείως ανομοιογενές και χαοτικό. Όσον αφορά την συμπεριφορά που θα προκαλέσουν στο σύστημα, δεν είναι όλα τα δεδομένα διαφορετικά. Υπάρχουν υποσύνολα, πολλές φορές μεγάλου εύρους, που τα περιεχόμενά τους θεωρούνται ισοδύναμα. Έτσι, αρκεί η αναγνώριση των διαφορετικών συμπεριφορών του συστήματος, ο διαχωρισμός του συνόλου εισόδων στα υποσύνολα που τις προκαλούν, και η συμπερίληψη δεδομένων από κάθε μία κλάση ισοδυναμίας στο σύνολο των δοκιμών.

Στην αντίθετη περίπτωση, το σύστημα αποτελεί 'λευκό κουτί', διαφανές για τον εκτελούντα τον έλεγχο. Η ευχέρεια της γνώσης των εσωτερικών δομών, επιβάλλει αυστηρότερες απαιτήσεις για επιμέρους ελέγχους, προσανατολίζοντας αναλόγως και για την επιλογή του συνόλου δοκιμών. Ένα πρόγραμμα αποτελείται από ένα σύνολο εντολών, που, συνήθως, δεν είναι ενιαίο, αλλά επιμερίζεται από εντολές διακλαδώσεων υπό συνθήκη. Διαμορφώνονται έτσι διάφορα μονοπάτια στον κώδικα, τα οποία σε κάθε εκτέλεση ακολουθώνται κατά περίπτωση, αφήνοντας κάποια τμήματα που δεν εκτελούνται. Η εκτέλεση μίας γραμμής κώδικα, αποκαλείται κάλυψή της (coverage), και μία απλοϊκή απαίτηση είναι η κάλυψη όλων των εντολών του προγράμματος. Αυστηρότερα, μπορεί επιπλέον να απαιτηθεί, να καλυφθούν όλες οι συνθήκες, είτε ως

συνολική απόφαση, είτε στις επιμέρους προτάσεις που τις συνθέτουν, με την λήψη όλων των δυνατών τιμών, και αντίστοιχα συνδυασμών, των όρων τους. Άλλη μορφή κάλυψης είναι των μονοπατιών, και ένα σύνολο δοκιμών που την πληροί, εκτελεί και κάλυψη εντολών και αποφάσεων, αλλά όχι απαραίτητα και προτάσεων, που είναι πιο απαιτητική. Οι απαιτήσεις κάλυψης, εκτός από οδηγό για την επιλογή των κατάλληλων εισόδων, αποτελούν κριτήριο πληρότητας και ισχυρότητας της μεθόδου, αλλά και ένδειξη ολοκλήρωσης της διαδικασίας ελέγχων.

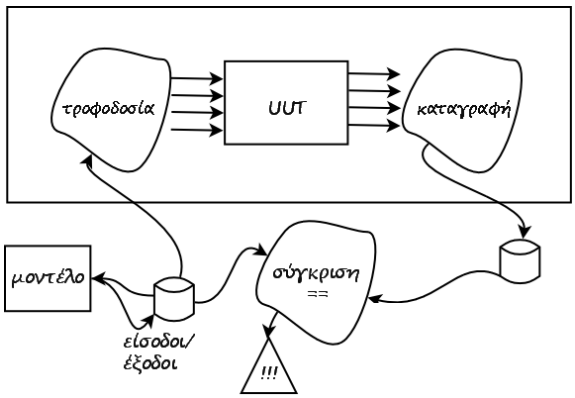
Η αποδοχή της δυναμικής επαλήθευσης επί περιορισμένου συνόλου εισόδων, θεμελιώνεται, έως ένα βαθμό, στην αναγνώριση των ιδιοτήτων των εισόδων και των προγραμμάτων, που μορφοποιούνται στις παραπάνω απαιτήσεις, λειτουργική κάλυψη και κάλυψη κώδικα. Ο βασικότερος παράγοντας είναι ότι για κάθε συνάρτηση δύο μεταβλητών αποδεικνύεται, πως ακόμα και με περιορισμένου αριθμού και τυχαία δείγματα εισόδων, επιτυγχάνεται αρκετά καλή αποτελεσματικότητα, μιάς και οι πρώτες δοκιμές θα ανιχνεύσουν με μεγάλη πιθανότητα σημαντικό τμήμα των σφαλμάτων. ([12], 11.5)

Εκτελώντας τυχαία δεδομένα, αλλά παραγόμενα με περιορισμούς (random constrained), που πρωτίστως τα καθιστούν έγκυρα, αλλά και αργότερα προσανατολίζουν τον έλεγχο προς περιοχές του χώρου του προβλήματος που δεν έχουν καλυφθεί, οι δύο κατευθύνσεις συνδυάζονται, για να προκύψει κάτι ελκυστικό, μαζί και αποτελεσματικό. Τα σύνολα δοκιμών λαμβάνουν τιμές είτε με δια χειρός απόδοση, είτε με αυτοματοποιημένη παραγωγή, ιδιαιτέρως για μεγάλο όγκο δεδομένων, τυχαίων ή μη.

Κατόπιν της επιλογής (και σχεδιασμού) της παραγωγής του κατάλληλου υποσυνόλου εισόδων, εκτελούνται οι δοκιμές, για να παρατηρηθεί η απόκριση του συστήματος. Για να επαληθευτεί η ορθότητα, που για κάθε επιμέρους περίπτωση είναι αποδεκτό, τα δεδομένα εξόδου θα πρέπει να συγκριθούν, είτε με το μάτι, είτε αυτόματα. Τα δεδομένα αναφοράς υπολογίζονται από κάποια άλλη λύση ή μοντελοποίηση του συστήματος, θεωρούμενη αξιόπιστη, και αποκαλούμενη golden model. Η αξιοπιστία του μοντέλου έναντι του αρχικού συστήματος, βασίζεται στην διαφοροποίησή τους. Άλλη προσέγγιση είναι η χρήση των εξόδων στο πρόβλημα που λύνουν, για παράδειγμα, δοκιμάζοντας αν οι ρίζες ενός πολυωνύμου δευτέρου βαθμού, το μηδενίζουν.

Ιδιαιτέρως στις γλώσσες περιγραφής υλικού, HDLs, επιπλέον των εξόδων, υπάρχει η δυνατότητα παραγωγής μηνυμάτων όταν κάποια συνθήκη δεν πληρείται. Ο μηχανισμός των assertions (και, χωρίς συνθήκη, reports), επιτρέπει την ευκολότερη συμπερίληψη των προδιαγραφών στον έλεγχο, με την ενσωμάτωση των επιθυμητών ιδιοτήτων στον κώδικα. Πράγματι, αποτελεί τόσο αποτελεσματικό εργαλείο, που εκτός από μέρος της δυναμικής επαλήθευσης, έχει αρχίσει να αποτελεί και βάση για τον αυτοματισμό των στατικών μεθόδων.[4]

[4] των οποίων η Formal Verification αποτελεί την μόνη αποδεικτική διαδικασία, αλλά υπάρχουν και επιμέρους ζητήματα που μπορούν να ελεγχθούν στατικά, για παράδειγμα, ζητήματα χρονισμού.



Σχήμα 6.1: Γενική δομή συστήματος ελέγχου.

Μεγάλη πρόοδος, τα τελευταία χρόνια, έχει γίνει στην τυποποίηση των μεθόδων της επαλήθευσης. Συμπεριλαμβάνοντας προϋπάρχουσες γλώσσες, όπως η VHDL, με επιρροές από άλλες, όπως η SystemC, αλλά κυρίως επεκτείνοντας ή διαμορφώνοντας νέα πρότυπα για κάποιες, όπως τα IEEE 1850 (Property Specification Language) και IEEE 1800 (SystemVerilog), η προσπάθεια αυτή έχει οδηγήσει στις OVM (Open Verification Methodology), και UVM (Universal Verification Methodology). Μένει να υιοθετηθούν ή να απορριφθούν ευρέως από την βιομηχανία του υλικού, πάντως είναι απαραίτητη η στροφή προς την εξάπλωση και συστηματοποίηση της επαλήθευσης, ως του σημαντικότερου σχεδιαστικού σταδίου, καθώς η έκταση και η πολυπλοκότητα των σχεδιάσεων δεν παύουν να αυξάνονται.

6.1.2 Επιλογή απόχρωσης

Από τις αυστηρές αποδεικτικές μεθόδους, καμία δεν ακολουθήθηκε, για τους λόγους που αυτό συνήθως συμβαίνει: στην πρώτη περίπτωση, άγνοια και απροθυμία για αναίρεσή της, και στην δεύτερη μη πρακτικότητα. Στο συγκεκριμένο σύστημα, το σύνολο των δυνατών εισόδων, και οι συνδυασμοί των τιμών τους, είναι ιδιαίτερα περιορισμένα σε αριθμό. Πρόκειται όμως για ακολουθιακό σύστημα, και έχοντας μνήμη, βρίσκεται σε διάφορες καταστάσεις που εξαρτώνται από τις προηγούμενες εισόδους. Το σύνολο όλων των δυνατών εισόδων ανάγεται έτσι, όχι σε συνδυασμούς, αλλά σε σύνολα ακολουθιών τιμών, και με απεριόριστα τα δυνατά μήκη τους το πλήθος αυξάνεται σημαντικά, καθιστώντας το αν όχι ανέφικτο, ανεφάρμοστο. Εφαρμόστηκε η λιγότερο αυστηρή δυναμική επαλήθευση, επί πεπερασμένου συνόλου εισόδων, τυχαίων υπό περιορισμούς.

Οι επιλεγμένες είσοδοι, αφού παραχθούν από το σχετικό υποσύστημα, τροφοδοτούν το σύστημα υπό έλεγχο (unit under test), αλλά και το μοντέλο. Αντί για την παροχή των εξόδων του μοντέλου στο σύστημα, και την επί τόπου σύγκριση, οι έξοδοι και των δύο θα αποθηκεύονται για να συγκριθούν στην συνέχεια, και να προκύψουν τυχόν ασυμφωνίες, ή αλλιώς, ταύτιση. (Σχήμα

6.1) Ειδικότερα όσον αφορά τις εισόδους δοκιμών, η ανάλυση στην οποία βασίζεται η παραγωγή τους, αλλά και οι συγκεκριμένες επιλογές, βρίσκονται στις ενότητες 6.1.3 και 6.3.2, αντίστοιχα.

Όλα μαζί τα παραπάνω μέρη ενοποιήθηκαν σε ένα αυτόματο σύστημα ελέγχου, αναφορικά (auto)Tester. Στον πυρήνα της λειτουργίας του βρίσκεται η παραγωγή των εισόδων δοκιμών, και πλαισιώνεται από την διαχείριση της λειτουργίας του μοντέλου, που αποτελεί κομμάτι του Tester, την αυτόματη κλήση της προσομοίωσης του συστήματος, την σύγκριση των εξόδων τους, και την καταγραφή των αποτελεσμάτων. (Λεπτομέρειες υλοποίησης Παράρτημα Α΄).

6.1.3 Διερεύνηση χώρου

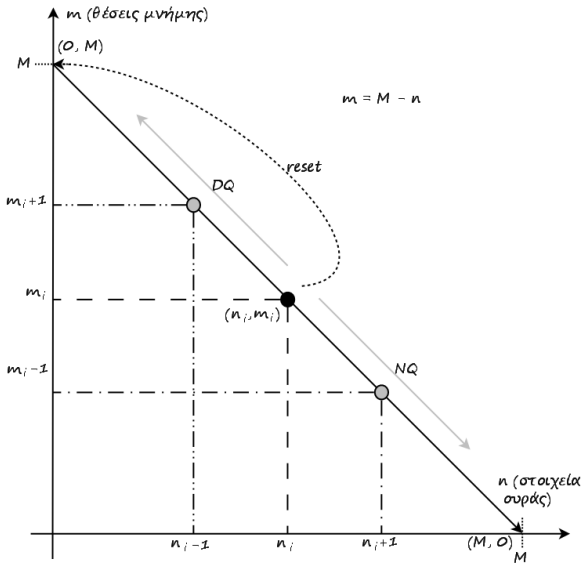
Το σύστημα έχει, χονδρικά, δύο χώρους για έλεγχο ανάλογα με την οπτική, μαύρου ή λευκού κουτιού. Στο πρώτο, αντιστοιχούν τα γενικά λειτουργικά χαρακτηριστικά, όπως η πληρότητα της μνήμης των διαθέσιμων θέσεων, και οι καταστάσεις των ουρών του συστήματος, ως προς το πλήθος στοιχείων της καθεμιάς. Στο δεύτερο, έχει σημασία να εξεταστεί και η συμπεριφορά των εγγενών στην υλοποίηση δομών και συμπεριφορών, όπως είναι η προώθηση (forwarding), και οι προσπεράσεις (bypassing) που εμφανίστηκαν.

Επιμένοντας στην θεώρηση μαύρου κουτιού, ο χώρος του προβλήματος, μπορεί να περιγραφεί ως συνάρτηση του πλήθους στοιχείων στην FreeList, και στις ουρές. Με μεταβλητές τις $n_i, i \in 0, \ldots, N-1$, ως το πλήθος στοιχείων στην i-οστή ουρά, όπου N το συνολικό πλήθος των ουρών, και m το τρέχον πλήθος των διαθέσιμων θέσεων στην FL, όπου M το συνολικό τους (μέγιστο), και θεωρώντας αρχικά ενιαίο το σύνολο των στοιχείων που βρίσκονται δεσμευμένα σε κάποια, n, ανεξάρτητα από το σε ποιά επιμέρους ουρά βρίσκονται, προκύπτει:

$$\sum_{i=0}^{N-1} n_i + m = M \Rightarrow \sum_{i=0}^{N-1} n_i = n = M - m \Rightarrow$$
$$m = M - n \qquad n, m \in [0, M], M > 0. \tag{6.1}$$

Ο χώρος του προβλήματος, όπως εκφράζεται παραπάνω, (6.1), είναι μία ευθεία ακεραίων σημείων στο καρτεσιανό επίπεδο, με το πλήθος δεσμευμένων στοιχείων n των ουρών ως ανεξάρτητη, και το πλήθος διαθέσιμων στοιχείων m στην FL ως εξαρτημένη μεταβλητή. (Σχήμα 6.2) Το σήμα reset επαναφέρει το σύστημα στο αρχικό σημείο, όπου κανένα στοιχείο δεν είναι δεσμευμένο, και όλα είναι διαθέσιμα στην FL. Οι εντολές μετακινούν το σημείο αναλόγως, αυξάνοντας τα στοιχεία των ουρών και μειώνοντας τα διαθέσιμα, η εντολή NQ, και το αντίστροφο, η εντολή DQ. Το αρχικό σημείο, είναι και ακραίο για την DQ, και δεν μπορεί να κινηθεί πλέον, ενώ για την NQ, ακραίο σημείο είναι εκεί όπου όλα τα στοιχεία έχουν δεσμευτεί στις ουρές, και η FL είναι άδεια.

Μην θεωρώντας ενιαίο το πλήθος των στοιχείων που έχουν δεσμευτεί σε όλες τις ουρές, αλλά διακρίνοντας το πλήθος στοιχείων n_i που ανήκουν σε μία



Σχήμα 6.2: Χώρος προβλήματος, σε δύο διαστάσεις.

ουρά επί της οποίας εκτελείται η τρέχουσα εντολή, δηλαδή qi $= i$, έναντι των υπολοίπων ουρών, με το άθροισμα των στοιχείων τους να αντιστοιχεί στην u_i, μεταφερόμαστε σε χώρο τριών διαστάσεων, ως εξής:

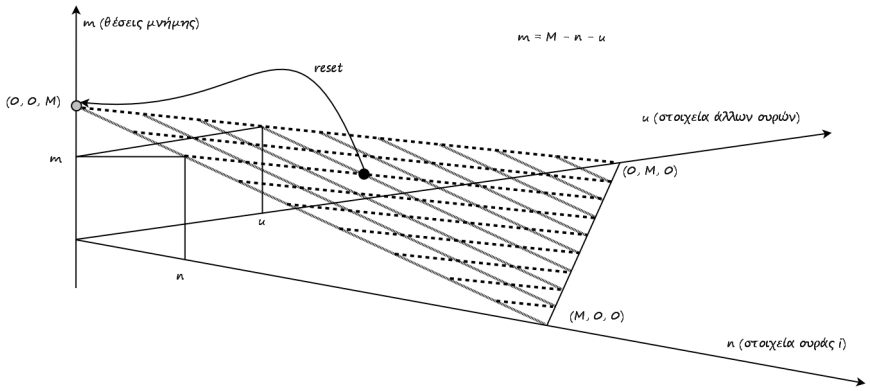
$$\text{αν} \qquad n = \sum_{j=0}^{N-1} n_j = n_i + \sum_{k\neq i}^{k\in[0,N-1]} n_k = n_i + u_i \qquad \stackrel{(6.1)}{\Longrightarrow}$$

$$n_i + u_i + m = M \qquad n_i, u_i, m \in [0, M], M > 0. \qquad (6.2)$$

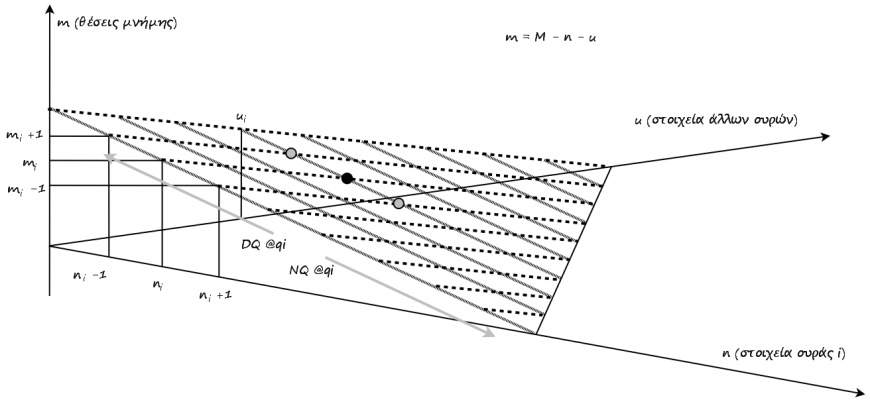
Το αρχικό σημείο, στο οποίο το reset επαναφέρει οποτεδήποτε, είναι εκεί όπου ούτε η τρέχουσα, ούτε καμία άλλη ουρά έχουν στοιχεία. (Σχήμα 6.3). Τα άκρα, είναι πλέον το σύνολο των σημείων όπου η διάσταση m μηδενίζεται, για την εντολή NQ, όπως και για την DQ είναι το σύνολο των σημείων όπου μηδενίζεται η διάσταση n, της εκάστοτε τρέχουσας ουράς.

Η κίνηση πραγματοποιείται αντίστοιχα με τον διδιάστατο χώρο, όσον αφορά μία συγκεκριμένη ουρά, αφού ο χώρος είναι ουσιαστικά πλέγμα γραμμών (ακεραίων σημείων) και όχι επιφάνεια. (Σχήμα 6.4) Σε κάθε δεδομένη στιγμή, γίνεται με σταθερό το πλήθος στοιχείων των άλλων ουρών, και περιορίζεται στην ευθεία που η διάσταση των υπολοίπων ουρών είναι u_i, μεταβάλλοντας μόνο τις τιμές των διαστάσεων n και m. Στην διάσταση u μπορεί να υπάρξει μεταβολή, αλλά όταν εκτελεστούν εντολές που αφορούν τις υπόλοιπες ουρές. (Σχήμα 6.5)

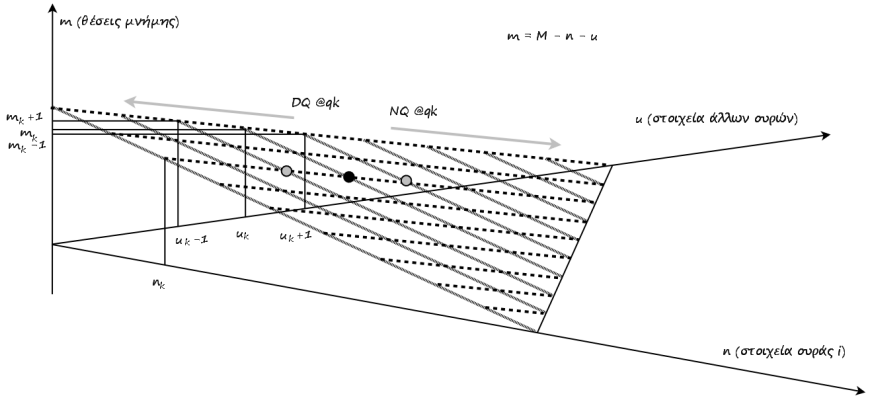
Σε πλήρη ανάλυση, κάθε μία ουρά θα είχε ένα σημείο στον τρισδιάστατο χώρο, και όλα τα σημεία, σε κάθε δεδομένη στιγμή, θα ανήκαν στο ίδιο επίπεδο ως προς τη διάσταση m, συνεπώς, σε κάθε εντολή, θα κινούνταν όλα. Για



Σχήμα 6.3: Χώρος προβλήματος, σε τρείς διαστάσεις. Αρχικό σημείο.

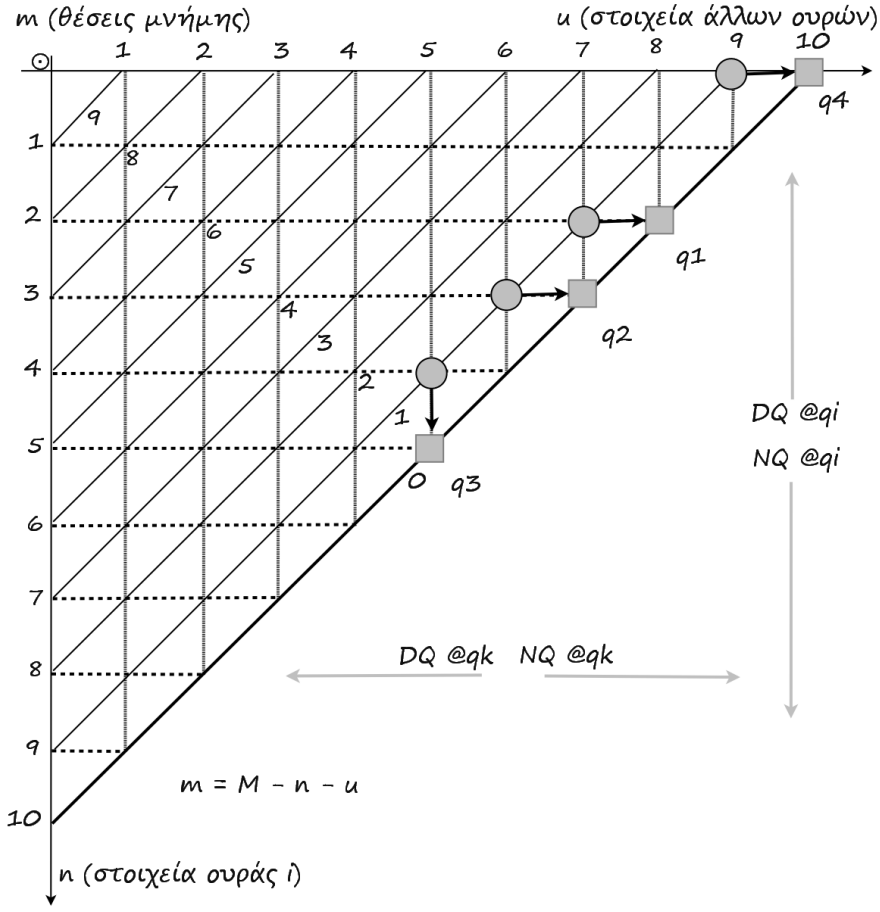


Σχήμα 6.4: Κίνηση σε τρείς διαστάσεις: τρέχουσα ουρά.



Σχήμα 6.5: Κίνηση σε τρείς διαστάσεις: υπόλοιπες ουρές.

παράδειγμα, σε ένα σύστημα με τέσσερις ουρές, γίνεται NQ στην τρίτη ουρά. (Σχήματα 6.6 και 6.7). Θα αλλάξει η τιμή της μόνο ως προς τη διάσταση n, και για τις άλλες ουρές μόνο ως προς u, και έτσι όλες μαζί, θα συνεχίσουν να βρίσκονται στο ίδιο επίπεδο ως προς m.



Σχήμα 6.6: Παράδειγμα κίνησης ουρών στον τρισδιάστατο χώρο.

M=10	n	u	m	n'	u'	m'
q1	2	7	1	2	8	0
q2	3	6	1	3	7	0
q3	4	5	1	5	5	0
q4	0	9	1	0	10	0

Σχήμα 6.7: Αρχικές και επόμενες συντεταγμένες ουρών.

εκδοχή	αριθμός εκδ.	υλοποίηση	περί μνημών
alfa	version_1_0	πολλαπλών κ. (2 c/st)	όλες δίθυρες, comp σε stages
beta	version_1_1	πολλαπλών κ. (1 c/st)	όλες μίας θύρας, στο d.path
gamma	version_2_0	πολλαπλών κ. (1 c/st)	όλες μίας θύρας
delta	version_2_1	ομόχειρο (+FLbpss-fwd)	3 δίθυρες, 1 μίας θύρας
epsilon	version_2_2	ομόχειρο+fwd(1,2)-fwd(3)+FLbpss-Nmbpss	nH, nT, nN/pFLH, pD, pregs
zhta	version_2_3	ομόχειρο+fwd(1,2,3)+FLbpss+Nmbpss (?)	nH, nT, nN/pFLH, pD, pregs
Hta	version_2_4	ομόχειρο+fwd(1,2,3)+FLbpss+Nmbpss	nH(16x8), nT(16x8), nN(2n8x8)/pFLH(FL=16), pD(2n8x16), pregs
T8eta	version_2_5	ομόχειρο+fwd(1,2,3)+FLbpss+Nmbpss	nH(16x8), nT(16x8), nN(2n8x8)/pFLH(FL=256), pD(2n8x16), pregs
giwta	version_2_6	ομόχειρο+fwd(1,2,3)+FLbpss+Nmbpss ΑΛΛΑ με configuration	nH(), nT(), nN()/pFLH(), pD(), pregs

Σχήμα 6.8: Εκδοχές της σχεδίασης.

6.2 Πιο χαμηλά

Κατά τα διάφορα στάδια της ανάπτυξης του συστήματος, προέκυψαν και οι ανάλογες εκδοχές του. (Σχήμα 6.8) Ξεκινώντας, με την alfa, από σχεδίαση πολλών κύκλων, σε version 1 και ύστερα 2, συνεχίζοντας σε επικαλυπτόμενη εκτέλεση, και προσθέτοντας προοδευτικά την υποστήριξη προώθησης και παρακάμψεων, φτάνοντας μέχρι την zhta.

Η γλώσσα ανάπτυξης, συγκεκριμένα η VHDL, έχει σκοπό τόσο την σύνθεση, όσο και την προσομοίωση. Έτσι, επιτρέπει περιγραφές που δεν είναι απεικονίσιμες στο φυσικό επίπεδο. Πλήρες μεν, λειτουργικά, το σύστημα δεν είχε αναπτυχθεί με σκεπτικό τελείως κοντά στο υλικό. Η διαδικασία της σύνθεσης, κατέληξε σε άρση μερικών χονδροειδών παρανοήσεων, που οδηγούσαν σε εμφάνιση ανεπιθύμητων μανδαλωτών (latches), δομών μη-συναγόμενων (non-inferable) σε κυκλωματικά στοιχεία, και προβλήματα χρονισμού. Η σύνθεση του συστήματος επιτρέπει, επιπλέον της προσομοίωσης και επαλήθευσης της περιγραφής συμπεριφοράς του (Behavioral/functional simulation), τον έλεγχο και σε χαμηλότερο επίπεδο, προσομοιώνοντας την απεικόνιση σε δομές

		O	W_N	D
ελάσσον	E	16	8	8
μείζον	M	32	16	16
άλλα	a	-	12	-
		-	11	-

Σχήμα 6.9: Ελάχιστες και μέγιστες τιμές των θεμελιωδών διαστάσεων.

συσκευών FPGA, (Stuctural/gate-level simulation).

Πρόσθετες εκδοχές δημιουργήθηκαν με αυτοσκοπό τον έλεγχο. Η Hta διαφοροποιούνταν από την προηγούμενή της μόνο χωρικά, στο σημείο αποθήκευσης, και για λόγους πρακτικούς. Η T8eta διέφερε στο πλήθος των στοιχείων της FL που θα γίνονταν τελικά διαθέσιμα, μέγεθος αυθαίρετο για την συγκεκριμένη σχεδίαση. Τελικά, αναπτύχθηκε μία ακόμη εκδοχή, gIwta, με αλλαγές ως προς την οργάνωση των αρχείων, αποδεικνύοντας τις προηγούμενες επιλογές εύλογες και αναγκαίες. (Περισσότερα, στο Α΄.6)

Πεδίο παρέμβασης και ελέγχου, εκτός από τις διάφορες εισόδους για λήψη ανάλογων αποκρίσεων, είναι οι διαστάσεις του συστήματος. Έχοντας ήδη αναπτυχθεί με παραμετρική διατύπωση, αλλά και με την απαραίτητη ευελιξία συνυφασμένη με τις δομές απεικόνισης (FPGAs), από την εκδοχή T8eta προέκυψε σειρά υπο-εκδοχών, για δοκιμές με ποικίλη διαστασιοποίηση.

Από τις διαστάσεις, όπως αναλύθηκε και στις ενότητες 3.2.1 και 5.1.2, μόνο τρείς είναι αυθαίρετες, και επιρρεάζουν όλες τις υπόλοιπες, τόσο στις μνήμες, όσο και τα σήματα.[5] Το πλήθος των ουρών, Qs_multitude, το πλήθος των στοιχείων στο σύστημα, Data/NextMem_Depth, και το πλάτος των δεδομένων, DataElem_width, ή, για συντομία, O, N, D, αντίστοιχα. Καθεμία δοκιμάστηκε για δύο τιμές, μία μέγιστη (M), και μία ελάχιστη (E). (Σχήμα 6.9).

Από τους συνδυασμούς των ελάχιστων και μέγιστων τιμών των θεμελιωδών διαστάσεων, προκύπτουν οκτώ συστήματα διαφόρων μεγεθών. (Σχήμα 6.10). Καθώς η διάσταση πλάτους W_N των διευθύνσεων DataMem, NextMem, καθορίζει το βάθος των μνημών αυτών, εκθετικά, αλλά και το πλάτος των μνημών HeadMem, TailMem, είναι η κυρίαρχη μεταξύ των τριών. Για το λόγο αυτό, δοκιμάστηκαν μόνο οι τέσσερις συνθέσεις που στις άλλες δύο διαστάσεις, είχαν από κοινού είτε τις ελάχιστες, είτε τις μέγιστες τιμές, και όχι τους μεταξύ τους συνδυασμούς. Μιάς και συνδυασμού διαστάσεων EEM ήταν οι προγενέστερες εκδόσεις, Hta και T8eta, ελέγχθηκε και αυτός. Για λόγους αδυναμίας εκτέλεσης των συνδυασμών των μεγίστων στα διαθέσιμα μηχανήματα, εκτενείς έλεγχοι εφαρμόστηκαν σε υβριδικές υπο-εκδοχές, EaE και Ea2E, με ελάχιστες τιμές διαστάσεων O και D, και τιμές μικρότερες της

[5] με επιπλέον αυθαίρετη διάσταση, που αφορά μόνο σήματα και καταχωρητές, το πλάτος του inID.

						H/T_mem	N_mem	D_mem	H/T_mem			N_mem			D_mem			Σύνολο	
κωδικός		σύμβλ.	O	W_N	D	OxW$_N$	2nW$_N$xW$_N$	2nW$_N$xD	p	bits	B	p	bits	pB	Bytes	p	bits	pB	Bytes	Bytes	kBytes
0	"000"	EEE	16	8	8	16x8	2n8x8	2n8x8	7	128	16	11	2048	8	256	11	2048	8	256	528	0,515625
1	"001"	EEM*	16	8	16	16x8	2n8x8	2n8x16	7	128	16	11	2048	8	256	12	4096	9	512	784	0,765625
2	"010"	EME	16	16	8	16x16	2n16x16	2n16x8	8	256	32	20	1048576	17	131072	19	524288	16	65536	196640	192,03125
3	"011"	EMM	16	16	16	16x16	2n16x16	2n16x16	8	256	32	20	1048576	17	131072	20	1048576	17	131072	262176	256,03125
4	"100"	MEE	32	8	8	32x8	2n8x8	2n8x8	8	256	32	11	2048	8	256	11	2048	8	256	544	0,53125
5	"101"	MEM	32	8	16	32x8	2n8x8	2n8x16	8	256	32	11	2048	8	256	12	4096	9	512	800	0,78125
6	"110"	MME	32	16	8	32x16	2n16x16	2n16x8	9	512	64	20	1048576	17	131072	19	524288	16	65536	196672	192,0625
7	"111"	MMM	32	16	16	32x16	2n16x16	2n16x16	9	512	64	20	1048576	17	131072	20	1048576	17	131072	262208	256,0625
		*Hta, T8eta																			
8	b12	EaE	16	12	8	16x12	2n12x12	2n12x8	-	192	24	-	49152	-	6144	-	32768	-	4096	10264	10,0234375
9	b11	Ea2E	16	11	8	16x11	2n11x11	2n11x8	-	176	22	-	22528	-	2816	-	16384	-	2048	4886	4,77148437

Σχήμα 6.10: Μεγέθη μνημών για τους συνδυασμούς των διαστάσεων.

κωδικός	0 = "000"	2 = "010"	5 = "101"	7 = "111"	8 = b12	9 = b11
σύμβολο(O,W_N,D)	EEE	EME	MEM	MMM	EaE	Ea2E
O // W_N // D	16// 8 // 8	16 // 16 //8	32 // 8 // 16	32 //16 // 16	16 // 12 // 8	16 // 11 // 8
report data- dummy_datapath						
Selected Device	4vsx25ff668-12	4vsx25ff668-12	4vsx25ff668-12	4vsx25ff668-12	4vsx25ff668-12	4vsx25ff668-12
Number of Slices	244 /10240 (2%)	338 /10240 (3%)	254 /10240 (2%)	355 /10240 (3%)	305 /10240 (2%)	298/10240 (2%)
Number of Slice Flip Flops	208 /20480 (1%)	303 /20480 (1%)	237 /20480 (1%)	335 /20480 (1%)	252 /20480 (1%)	246/20480 (1%)
Number of 4 input LUTs	455 /20480 (2%)	634 /20480 (3%)	464 /20480 (2%)	637 /20480 (3%)	572 /20480 (2%)	549/20480 (2%)
Number of IOs	54	54	72	72	54	54
Number of bonded IOBs	54 / 320 (16%)	54 /320 (16%)	72 /320 (22%)	72 / 320 (22%)	54 /320 (16%)	54 /320 (16%)
Number of GCLKs	1 /32 (3%)	1 / 32 (3%)	1 /32 (3%)	1 / 32 (3%)	1 /32 (3%)	1 /32 (3%)
Speed Grade	-12	-12	-12	-12	-12	-12
Min period	5.122ns	4.679ns	4.465ns	4.664ns	4.493ns	4.958ns
Max Frequency	195.253MHz	213.735MHz	223.949MHz	214.397MHz	222.574MHz	201.751MHz
Min input arrival time before clk	2.615ns	2.889ns	3.339ns	2.982ns	2.742ns	2.725ns
Max output required time after clk	4.796ns	5.103ns	4.859ns	4.774ns	4.999ns	4.915ns
Max combinational path delay	4.056ns	4.056ns	4.056ns	4.104ns	4.056ns	4.056ns

Σχήμα 6.11: Μετρήσεις από σύνθεση.

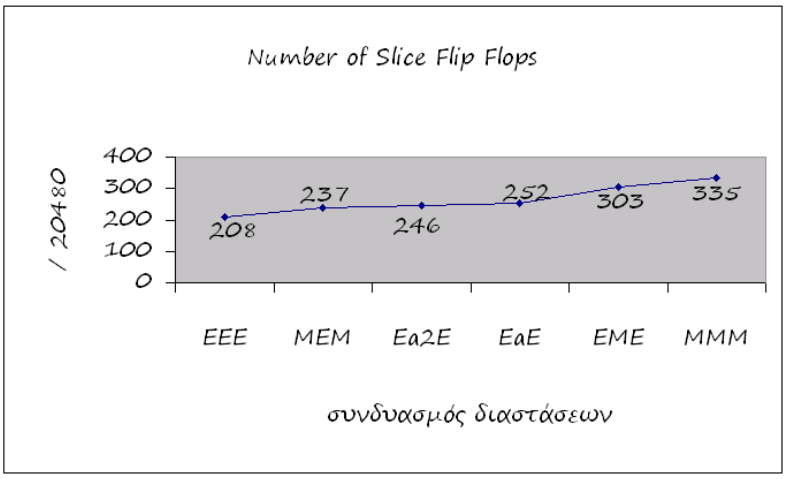
μέγιστης για την W_N. (Λεπτομέρειες στο Παράρτημα Α′).

Από την σύνθεση των παραπάνω έξι συνδυασμών, δηλαδή με εξαίρεση τον EEM, προέκυψαν στοιχεία για την χρήση της επιλεγμένης συσκευής FPGA, ως προς τα flip flops και τα Look Up Tables, LUTs, που δεσμεύτηκαν. Επίσης, στατικές εκτιμήσεις για τις επιδόσεις, εκφρασμένη ως μέγιστη επιτρεπτή συχνότητα ρολογιού, Max Frequency. (Σχήματα 6.11 έως 6.14). Επιβεβαιώνεται η κυρίαρχη επίδραση της διάστασης W_N.

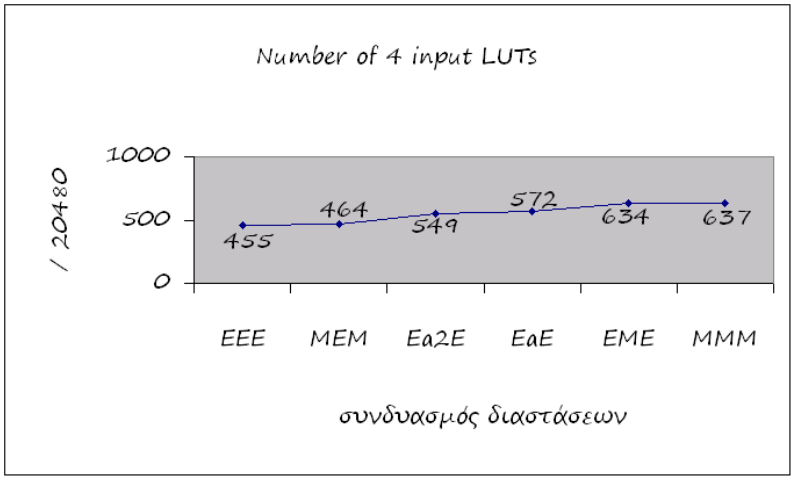
6.3 Έλεγχοι (testing)

6.3.1 Πρώιμα στάδια

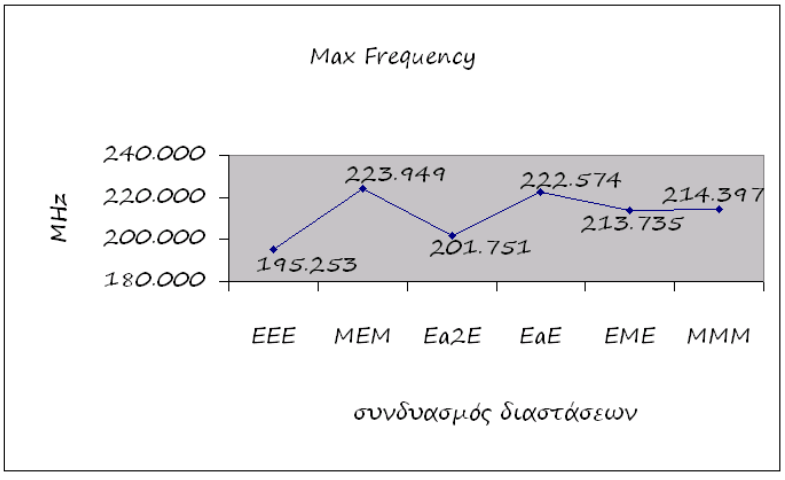
Δεν είναι απολύτως ακριβές ότι η επαλήθευση είναι το τελευταίο στάδιο της σχεδίασης, ακόμα και αν δεν χρειαζόντουσαν ποτέ διορθώσεις και παρεμβάσεις. Στην πραγματικότητα, πραγματοποιείται παράλληλα με την σχεδίαση και την ανάπτυξη όλων των υπόλοιπων σταδίων. Μαζί με αυτά εξελίσσεται, σε έκταση και πολυπλοκότητα, για να ενταθεί σε διάρκεια και βάθος, στο τέλος της διαδικασίας.



Σχήμα 6.12: Χρήση flip flops .



Σχήμα 6.13: Χρήση LUTs .



Σχήμα 6.14: Μέγιστη συχνότητα λειτουργίας.

είδος test	δημιουργία εισόδων	μέθοδος ελέγχου
στοιχειώδες	χέρι	με το μάτι
βασικό	looping process	με το μάτι
silver (common)	looping process	cmd fc
golden	autotester	autotester

Σχήμα 6.15: Επίπεδα και μέθοδοι επαλήθευσης.

είδος test	εκδοχή συστήματος	B.sim	T.sim	διάρκεια	αντικείμενο έλεγχου
στοιχειώδες	alfa, beta, gamma	v	-	30-130 cc	datapath (γενικά)
στοιχειώδες	alfa, beta, gamma, delta	v	-	30-130 cc	freelist (ως component ή stripped)
βασικό	delta, epsilon, zhta	v	-	1600 cc	datapath (γενικά)
βασικό	delta, epsilon, zhta	v	-	1600 cc	FL_stripped (γενικά)
βασικό	epsilon, zhta	v	-	1600 cc	περιπτώσεις προώθησης
silver (common)	delta <> epsilon	v	-	600 cc	datapth (out sigs: dataout, done)
silver (common)	delta <> epsilon	v	-	600 cc	FL_stripped (out sigs: nt_dummy, done)
silver (common)	epsilon <> zhta	v	-	600 cc	datapth (out sigs: dataout, done)
silver (common)	epsilon <> zhta	v	-	600 cc	FL_stripped (out sigs: nt_dummy, done)
βασικό	zhta	-	v	1600 cc	datapath (γενικά)
silver (common)	zhta(B) <> zhta(T)	v	v	600 cc	datapth (out sigs: dataout, done)
golden	Hta	v	-	1600 cc	forces (tssi2mti testbenches)
golden	Hta	-	v	1600 cc	forces (tssi2mti testbenches)
golden	Hta, Theta, EM*	v	-	3x10^5 cc	senarios (ειδικά , τυχαία)
golden	Hta, Theta, EM*	-	v	5x10^3 cc	senarios (ειδικά , τυχαία)
βασικό	EM(s{0,2,5,7})	-	v	1600 cc	FL_stripped (γενικά)
golden	Hta, Theta, EM*	v	-	10^6 cc	senarios (ειδικά , τυχαία)
golden	Hta, Theta, EM*	-	v	5x10^4 cc	senarios (ειδικά , τυχαία)

EM* = s0, s2, s5, s7, b11, b12

Σχήμα 6.16: Αντιστοιχία μεθόδων επαλήθευσης με εκδοχές του συστήματος.

Στα πιο πρώιμα στάδια, έγιναν στοχευμένοι έλεγχοι, πάνω στο εκάστοτε χαρακτηριστικό που βρισκόταν υπο ανάπτυξη. Μάλιστα, αλλά όχι απαραίτητα, τα σύνολα ελέγχου ήταν αρκετά περιορισμένα, και δημιουργούνταν χειρωνακτικά, ή με κάποιαν απλή αυτοματοποίηση, όπως μία διαδικασία VHDL σε βρόγχο (looping process). Η σύγκριση των αποτελεσμάτων γινόταν με το μάτι, και με νοητή εκτίμηση των σωστών εξόδων. Παρακάτω, όταν είχε επιτευχθεί κάποιο επαρκώς λειτουργικό κομμάτι, που θεωρούνταν αρκετά ασφαλές, χρησιμοποιήθηκε σαν πρώιμο μοντέλο (silver) (Α΄.2), για την αντιπαραβολή των εξόδων και την επιβεβαίωση ότι δεν εισάγονται σφάλματα εκεί που δεν υπήρχαν, με τις νέες προσθήκες (με την έννοια του regression testing). Με κάποιες τροποποιήσεις για καταγραφή των αποτελεσμάτων σε μορφή που να είναι επεξεργάσιμη από τον υπολογιστή, ευκολότερα από τις κυματομορφές τουλάχιστον, αυτοματοποιήθηκε στοιχειωδώς και η σύγκριση των εξόδων. (Σχήματα 6.15 και 6.16).

6.3.2 Αποτίμηση

Παρακάτω, παρουσιάζονται οι επιλογές με τις οποίες ο χρήστης κατευθύνει την παραγωγή συνόλων εισόδων, βάσει της συμπεριφοράς και επίδρασης κάθε σήματος εισόδου. Εξετάζεται η εκφραστικότητά τους και η συμβολή τους στον προσανατολισμό των ελέγχων στον χώρο του προβλήματος. Τελικά, δίνονται συνολικές μετρήσεις των δοκιμών που εκτελέστηκαν, επί συγκεκριμένων μηχανημάτων (Α΄.1.1).

Η βασικότερη λειτουργία του γενικευμένου συστήματος ελέγχου, όπως προαναφέρθηκε στην ενότητα 6.1.2, είναι η παραγωγή των συνόλων δεδομένων που θα εκτελεστούν από το σύστημα και το μοντέλο. Κάθε διαφορετική κατεύθυνση που μπορεί να καθοδηγηθεί ο Tester για την παραγωγή συνόλων εντολών, αποτελεί ένα σενάριο (senario), [6] που ορίζεται από το χρήστη, με ένα σύνολο (μία γραμμή) οδηγιών. Κάθε σενάριο, μπορεί να παράγει, ανάλογα με την αυστηρότητα των οδηγιών, από κάποιο μοναδικό σύνολο εντολών δοκιμών, μέχρι και διάφορα, που θεωρούνται λειτουργικά ισοδύναμα.

Από τον (auto)Tester θα παραχθούν, σε κάθε σύνολο δοκιμών, όλα τα σήματα εισόδου. Τα χαρακτηριστικά του κάθε σήματος εντοπίζονται, για τον προσδιορισμό του τρόπου παραγωγής καθενός, και για την απόδοση των ανάλογων επιλογών στον χρήστη. Η κυριότερη διάκριση γίνεται βάσει της επιρροής των σημάτων στην εσωτερική κατάσταση του συστήματος, και οι επιλογές του χρήστη, αφορούν τα σήματα αυτά, κατά κύριο λόγο. Η τυχαιότητα, η περιοδικότητα, και επιμέρους αναμενόμενες χρονικές διακυμάνσεις, στα πλάτους ενός bit σήματα, ενώ για μεγαλύτερου πλάτους σήματα, η σημασία των διαφόρων τιμών τους, και εξίσου, τα χρονικά μοτίβα εναλλαγών τους, είναι επιπλέον στοιχεία ενδιαφέροντος. (Σχήματα 6.17 και 6.18).

σήμα εισόδου	επιρροή κατάστασης	τυχαίο / αιτιοκρατικό	στοιχείο τυχαιότητας / συνταγή
clk	x	A	κάθε ημιπερίοδο {0 <-> 1}
reset	v	T	διάστημα εναλλαγής {0 <-> 1}
inid	x	A	αύξηση +1 κάθε νέο κύκλο
valid	v	T	διάστημα εναλλαγής {0 <-> 1}
operation	v	T	διάστημα εναλλαγής {0 <-> 1}
qi	x*	T	από [0, Qs_multitude -1]
datain	x	(T)	από [0, $dataMem\_width^2$ -1] (ή;)

Σχήμα 6.17: Χαρακτηριστικά σημάτων εισόδου: επιρροή, τυχαιότητα και πεδίο τιμών.

[6] κανονικά η λέξη γράφεται scenario.

σήμα	εξαρτάται / ορίζεται	επιλογές
clk	περίοδος, μονάδες χρόνου	(c)ycle_length, (t)ime_unit
reset	έναρξη, πολλαπλότητα, διάρκεια	(u)nal_reset, (r)eset_next, reset_(d)ur
inid	πλάτος σε bits	(y)nal_sigs
valid	εναλλασσόμενο, τυχαίο, σταθερό	(y)nal_sigs, (v)alid_fr, valid_(i)nit
operation	burst, εναλλασσόμενο, τυχαίο	(y)nal_sigs, (o)peration_fr, op(e)ration_(i)nit
qi	σταθερό, παράθυρο, πλήρες, τυχαίο	(y)nal_sigs, (q)i_code, qi_(f)req
datain	πλάτος σε bits (ή συνταγή;)	(y)nal_sigs

Σχήμα 6.18: Χαρακτηριστικά σημάτων εισόδου: ποικιλία συμπεριφορών, και αντίστοιχες επιλογές χρήστη.

Οι διάφορες είσοδοι του συστήματος, αν και όλες φέρουν κάποια πληροφορία, δεν είναι ισοδύναμες ως προς την επίδρασή τους. Από τα επτά σήματα, μόνο τα τέσσερα είναι εκείνα που επιρρεάζουν τις μεταβάσεις των καταστάσεων του συστήματος: τα reset, valid, operation, qi. Από τα υπόλοιπα τρία σήματα εισόδου, τα clk και inID έχουν πρακτικό ρόλο, ενώ τα dataIn δεν επιρρεάζουν το σύστημα, παρόλο που αποτελούν το ίδιο το αντικείμενό του.

Το σήμα clk είναι θεμελιώδες λειτουργικά σε οποιοδήποτε σύγχρονο ακολουθιακό σύστημα, όπως και το συγκεκριμένο. Έτσι, υπάρχουν οδηγίες σεναρίου για το clk, με σκοπό την ρύθμιση του προσομοιωτή, αλλά δεν επιρρεάζουν το παραγόμενο σενάριο ή τις εισόδους όπως θα τις λάβει το μοντέλο. Αυτό είναι εφικτό λόγω της σχεδιαστικής επιλογής για την απόδοση της επιθυμητής διάρκειας προσομοίωσης σε εντολές αντί σε χρόνο, σε αντίθεση με τους προσομοιωτές, όπου είναι το καθολικό εκφραστικό μέσο. Άλλωστε, στο συγκεκριμένο σύστημα κάθε εντολή αντιστοιχεί σε έναν κύκλο ρολογιού, τόσο σαν απαιτούμενος χρόνος στην είσοδο, έχοντας ακμοπυροδότητα στοιχεία, όσο και ως ρυθμός παραγωγής αποτελεσμάτων στην έξοδο. Οι επιλογές που ορίζουν ένα σήμα ρολογιού, δηλαδή μία περιοδική εναλλαγή από μηδέν σε ένα, σε δύο ίσης συνήθως διάρκειας ημιπεριόδους, είναι η περίοδος του, επιμερισμένη σε τιμή και υποδιαίρεση της μονάδας χρόνου. Για τον επιτυχή χρονισμό των ακμπυροδότητων στοιχείων, είναι απαραίτητο οι αλλαγές των εισόδων να μην γίνονται ταυτόχρονα, ή πολύ κοντά με τις ακμές του ρολογιού. Έτσι, μία επιπλέον επιλογή, είναι το διάστημα απόκλισης (unalignement) της αλλαγής των τιμών εισόδου από το ρολόι.

Παρότι οι ουρές δεν έχουν επιμέρους όριο στοιχείων, και δεν διαφοροποιείται η κατάσταση του συστήματος ως προς στην διάσταση δεσμευμένων στοιχείων, το qi καθορίζει το σημείο στον χώρο καταστάσεων που θα κινηθεί, ενδεχομένως, κάποιας ουράς άδειας έναντι άλλης με στοιχεία. Επίσης, κάθε νέα τιμή έχει σημασία στην συγκεκριμένη, ομόχειρη υλοποίηση, καθώς προκαλείται κίνδυνος δεδομένων όταν στο μονοπάτι συνυπάρχουν εντολές που δρούν στην ίδια ουρά ('συγγενικές'), που αποφεύγεται με τον μηχανισμό προ-

σήμα	επιρροή	πλάτος (bits)	διαφορετικές τιμές	εκδοχές
clk	x	1	2	1
reset	v	1	2	2^S
inid	x	(const)	$2^{(const)}$	1
valid	v	1	2	2^S
operation	v	1	2	2^S
qi	x*	$Q = log_2(Qs\_multitude)$	$2^Q = Qs\_multitude$	$x\ (2^{Q*S})$
datain	x	dataMem_width	$2^{dataMem\_width}$	$x\ (2^{dataMem\_width*S})$
		S = sim duration	σύνολο	$2^{3*S}\ (2^{[S*(3+Q+dmW)]})$

Σχήμα 6.19: Πλήθος διαφορετικών ακολουθιών σημάτων εισόδου.

ώθησης. Για την ανυπαρξία συμπτώσεων απαιτείται η διαφοροποίηση των αναφορών στις ουρές, εντός του 'παραθύρου' διαδοχικών εντολών, δηλαδή κάθε δυνατού υποσυνόλου με πλήθος όσο το βάθος του μονοπατιού μείον μίας θέσεων. Διαφορετικά, είναι δυνατή η εμφάνιση συμπτώσεων με εντολές μόνο σε μία συγκεκριμένη απόσταση, ή σε συνδυασμούς αποστάσεων, καθορισμένους ή και τυχαίους. Έτσι, επιλογές χρήστη αποτελούν η ύπαρξη και είδος και η επιθυμητή συχνότητα εμφάνισης συμπτώσεων.

Το reset, είναι μία ιδιάζουσα περίπτωση σήματος, καθώς η επίδρασή του δεν αφορά μόνο τον κύκλο στον οποίο εφαρμόζεται, ούτε και εμπλέκεται με την καθυστέρηση της ομοχειρίας. Για την επαναφορά του συστήματος στην αρχική του κατάσταση, το εισάγει σε μία ειδική φάση λειτουργίας που σηματοδοτείται στην έξοδο ταυτόχρονα και διαρκεί αρκετά. Η πρώτη εφαρμογή του σήματος είναι απαραίτητη, για την μορφοποίηση της μνήμης NextMem σε FL, αλλά από κει και έπειτα, είναι δυνατόν να εμφανιστεί κάποιες ή και μηδέν φορές. Εκτός από την εμφάνιση επόμενων reset, πέραν του απαιτούμενου, το οποίο παράγεται σε κάθε σενάριο, δίνεται η επιλογή καθορισμού της διάρκειας της καθεμίας εμφάνισης του σήματος (τιμή 1), μέχρι την απενεργοποίησή του.

Το valid είναι το συμπληρωματικό σήμα του operation. Και τα δύο από ένα bit, μαζί δίνουν τις τρείς δυνατές εντολές, NQ, DQ και noop. Μοιράζονται τις ίδιες, αλλά ανεξάρτητες, δυνατές επιλογές της συχνότητας εναλλαγής, περιοδικής ή τυχαίας, και της αρχικής τιμής.

Το inID αποτελεί μία ένδειξη βάσει μετρητή, για τον ευκολότερο εντοπισμό των αποτελεσμάτων κάθε εντολής. Για τον ίδιο σκοπό και χωρίς βλάβη της γενικότητας των ελέγχων, από θεωρητικά τυχαίο εξολοκλήρου, μέρος των bits του dataIn θα είναι αιτιοκρατικά. Και τα δύο σήματα παράγονται από τον Tester αυτόματα, χωρίς 'σεναριακή' παρέμβαση από τον χρήστη, αν εξαιρεθούν οι επιλογές χρονισμού και διάρκειας της προσομοίωσης.

Τα σήματα που έχουν τυχαίο χαρακτήρα είναι δυνατόν να εμφανίσουν, ανάλογα και με την ορισμένη διάρκεια προσομοίωσης, πλήθος διαφορετικών α-

επιλογή		σήμα	μεταβλητή	s/f	#	τύπος	default	πεδίο τιμών / περιορισμοί	εκδοχές
cycle_length	c		one_cycle	f	3	int	100	%10=0, >0	1 (10^14...)
time_unit	t	clk	time_unit	f	3	string	ns	{'s', 'ms', 'us', 'ns', 'ps', 'fs'}	6
ynal_sigs	y	! clk	unal_sigs	f	3	int	8	[1,9]	9
sim_dur	s	-	sim_dur	s	2	int	1600	>0	-
qi_code	q		qi_code	s	1	string	'7'	[0, ($2^{pipeline\_depth-1}$-1)10]	$2^{pipeline\_depth-1}$
qi_freq	f	qi	qi_freq	s	1	int	0	[0, sim_dur -1]	S
unal_reset	u		unal_res	f	3	int	7	[1,9]	9
reset_next	r		reset_next	s	1	int list	([])	[0, sim_dur -1], Σ <sim_dur	S/(rst_dur + init)
reset_dur	d	reset	reset_dur	s	2	int	1	[1,10]	10
valid_fr	v		valid_fr	s	1	int	0	[0, sim_dur -1]	S
valid_init	i	valid	valid_init	s	1	int	1	{0,1}	2
operation_fr	o		operation_fr	s	1	int	0	[0, sim_dur -1]	S
operation_init	e	operation	operation_init	s	1	int	1	{0,1}	2
						σύνολο εκδοχών :		19440 * ($2^{pipeline\_depth-1}$) * (S^4) / (rst_dur + init)	

= βαθμός διαμόρφωσης σεναρίου (μέγιστο 1), s/f=τι επιρρεάζει, senario ή forces, S = sim duration

Σχήμα 6.20: Πλήθος διαφορετικών σεναρίων.

κολουθιών, αν αυτό εκτιμηθεί ως προς τις τιμές τους αποκλειστικά. (Σχήμα 6.19). Τα σενάρια που μπορούν να περιγραφούν, είναι σημαντικά λιγότερα σε πλήθος, και είναι αναμενόμενο, καθώς αποτελούν γενικευμένες περιγραφές. (Σχήμα 6.20).

Για το ρολόι, πρέπει να αναφερθεί, ότι μία είναι η δυνατή εκδοχή του σε επίπεδο τιμών, ή ακριβέστερα δύο, γιατί περιορίζεται από την περιοδικότητά του. Η επιλογή cycle_length μπορεί να πάρει μεγάλο πλήθος διαφορετικών τιμών. Για παράδειγμα, με μέγιστο όριο διάρκειας κύκλου το 1 sec και μονάδες fs, πολλαπλάσια της δεκάδας, όπως απαιτεί ένας επιπλέον περιορισμός, είναι 10^{14} τιμές. Στην πραγματικότητα, οι τιμές αυτές δεν ορίζουν διαφορετικά σενάρια, μιάς και καμία τιμή της διάρκειας του κύκλου δεν αλλάζει τον χρονισμό του ρολογιού ως προς τα άλλα σήματα. Αυτό γιατί, επιπλέον της διάρκειας της προσομοίωσης, και η παραγωγή των σημάτων βασίζεται στην διάρκεια του κύκλου που θα επιλεγεί, με την ανελαστική λογική να ισοδυναμεί ένας κύκλος με μία παραγόμενη εντολή.

Μεγάλο μέρος των δυνατών εισόδων, εκείνες που δεν ακολουθούν κάποια περιοδικότητα, δεν περιγράφονται άμεσα από τις οδηγίες σεναρίου, αλλά είτε από την απουσία τους, ως default επιλογές, είτε με την ρητή επιλογή της τιμής που εκφράζει την τυχαιότητα. Ειδική περίπτωση αποτελεί το σήμα reset, στο οποίο, σχεδιαστικά, δεν έχει επιτραπεί τόση ελευθερία και αντικειμενικότητα. (Λεπτομέρειες στο παράρτημα Α′). Με την εξαίρεση αυτή, το σύνολο των ακολουθιών τιμών μπορεί τελικά να παραχθεί με τις λιγοστές οδηγίες σεναρίου, με την τυχαιότητα να καλύπτει το κενό της περιγραφής.

Ουσιαστικά, οι συνδυασμοί των οδηγιών του σεναρίου, κατά κύριο λόγο, καθοδηγούν τον έλεγχο προς την κατεύθυνση των εσωτερικών χαρακτηριστι-

test \ option(#1,2)	s	q2	f	r	d	v	i	o	e	παράδειγμα
full random (default)	N	111	0	-	-	0	1	0	1	-s N
fwd : (n-1)	N	001	0,1,...	-	-	(N-1)	-	-	-	-s N -q 1 -f 25 -v (N-1)
fwd : (n-2)	N	010	0,1,...	-	-	(N-1)	-	-	-	-s N -q 2 -f 2 -v (N-1)
fwd : (n-3)	N	100	0,1,...	-	-	(N-1)	-	-	-	-s N -q 4 -f 10
no fwd	N	000	-	-	-	-	-	-	-	-s N -q 0
fwd γενικό	N	011, 101, 110, 111	0,1,...	-	-	-	-	-	-	-s N -q 3 -v (N-1), -s N -q 5, -s N -q 6 -v (N-1), -s N -q 7
NextMem bypass	N	001	1	-	-	(N-1)	-	2	-	-s N -q 1 -f 1 -v (N-1) -o 2
FL bypass	N	-	-	-	-	!=1	-	1	-	-s N -v (N-1) -o 1
άδειασμα + γέμισμα	N	010	1	~	-	-	-	>Nm_depth	-	-s N -v (N-1) -q 2 -f 1 -o 280
σχεδόν γεμάτο πάντα	N	-	-	~	-	-	-	-	-	-
reset	N	-	-	[...]	~	-	-	-	-	-s N -r 0 -r 0 -d 2

Σχήμα 6.21: Περιγραφή σεναρίων από συνδυασμούς οδηγιών.

κών υλοποίησης, βάσει της προσέγγισης λευκού κουτιού. (Σχήμα 6.21) Την προσέγγιση μαύρου κουτιού, από την άποψη των λειτουργικών χαρακτηριστικών, θεωρείται ότι ικανοποιούν εκτενή σύνολα τυχαίας σύνθεσης, αλλά χωρίς να παράγονται κάποια πιό σύνθετου και περιγραφικά υψηλότερου επιπέδου σενάρια. Για αυτά, θα απαιτούνταν εκτενέστατη παρακολούθηση της παραγωγής εισόδων, ανάλογη μιάς εκτέλεσης του μοντέλου, ή ακόμα και η ριζική αναδιάρθρωση όλου του Tester, για παραγωγή των εισόδων παράλληλη με την προσομοίωση και αναπροσαρμοζόμενη βάσει ανατροφοδότησης.

Η λειτουργική κάλυψη θεωρείται πως εκπληρώνεται, σε μεγάλο βαθμό, με τους εξειδικευμένους ελέγχους της υλοποίησης και τις τυχαίες δοκιμές, λόγω της ιδιαίτερης απλότητας του συγκεκριμένου συστήματος. Κάθε σύνολο εντολών, θα διατρέξει όλο το χώρο του προβλήματος, σχήμα 6.2, κάποιες περιοχές με βεβαιότητα, και άλλες με ισχυρή πιθανότητα. Σίγουρα εμφανίζονται οι μέσες περιπτώσεις, ενώ από τις ακραίες, η αρχικοποίηση και η εισαγωγή σε άδεια ουρά, για όσες ουρές ενεργοποιηθούν. Επίσης, είναι πολύ πιθανό να ζητάται από άδεια ουρά στοιχείο, με πιθανότητα 0.5 επί των πρώτων ενεργών εντολών που θα λάβει κάθε ουρά. Έτσι, το σημείο $(0, M)$ και τα κοντινά του, καλύπτονται, σίγουρα για τις ακραίες και μέσες, και με μεγάλη πιθανότητα για τις αδύνατες περιπτώσεις. Στο άλλο άκρο του δισδιάστατου χώρου, το σημείο $(M, 0)$, όπου η FL έχει αδειάσει, δεν είναι τόσο εύκολα προβλέψιμο, μεν, με τις τυχαίες εισόδους ότι θα προσεγγιστεί, μπορεί όμως να περιγραφεί από συγκεκριμένες οδηγίες σεναρίου.[7] (Σχήμα 6.21) Έτσι, λόγω των μόνο δύο ειδών ενεργών εντολών, άμα στο σημείο, είναι εύκολο να επιτευχθούν και η αδύνατη κατάσταση και, με την επαναφορά του συστήματος, η ακραία περίπτωση.

[7] Η επιλογή qi_code =2, έχει σκοπό την διευκόλυνση του αδειάσματος, περιορίζοντας τις ενεργές ουρές.

Επί των προαναφερθέντων, διάφορων εκδοχών μεγέθους του συστήματος, εφαρμόστηκε πλήθος συνόλων εισόδων, που κατασκευάστηκαν από τον Tester, βάσει ειδικευμένων ή τυχαίων σεναρίων. Οι ίδιες είσοδοι εφαρμόστηκαν σε κατάλληλα παραμετροποιημένο μοντέλο, και συγκρίθηκαν αυτόματα. Οι δοκιμές που εκτελέστηκαν, δεν έδειξαν απόκλιση του συστήματος, από την αναμενόμενή του απόκριση, όπως αυτή αποδόθηκε από το μοντέλο. (Σχήματα 6.22, 6.23 και 6.24).

	Β	Τ			Β	Τ		
project	tests		Σ	%	εντολές		Σ	%
b11_Ea2E	5	5	10	3,85	5000000	250000	52500000	7,39
b12_EaE	50	7	57	21,92	19650000	350000	20000000	28,16
Hta	71	51	122	46,92	19947600	249000	20196600	28,43
Hta (?)	10		10	3,85	7510040		7510040	10,57
s0_EEE	6	7	13	5,00	5300000	350000	5650000	7,95
s2_EME	6	8	14	5,38	5300000	400000	5700000	8,02
s5_MEM	1	0	1	0,38	300000	0	300000	0,42
s7_MMM	1	3	4	1,54	50000	150000	200000	0,28
T8eta	24	5	29	11,15	6214000	11000	6225000	8,76
Σύνολο	164	86			61761600	1760000		
	250		260	100	63521600		71031640	100
ποσοστό %	63,08	33,08			86,95	2,48		
	96,15				89,43			

Σχήμα 6.22: Επιμέρους εκτελέσεις και πλήθη εντολών που εκτελέστηκαν.

	Β	Τ	Β	Τ	Β	Τ	Β	Τ		
project	διάρκεια παραγωγής		διάρκεια προσομοίωσης		διάρκεια μοντέλου		διάρκεια σύγκρισης		σύνολο χρόνου/prj	ποσοστό %χρόνου
b11_Ea2E	0:04:24	0:00:16	27:44:16	50:45:10	0:01:29	0:00:05	2:03:03	0:06:23	80:45:06	14,13
b12_EaE	0:16:54	0:00:26	86:01:38	67:43:58	0:05:44	0:00:05	5:43:11	0:09:09	160:01:05	28,00
Hta	0:30:48	0:01:00	82:36:51	69:35:53	0:09:35	0:00:06	9:05:35	0:08:01	162:07:49	28,37
Hta (?)	0:07:28		14:18:49		-		-		14:26:17	2,53
s0_EEE	0:04:54	0:00:23	25:47:14	55:37:33	0:01:40	0:00:05	2:19:25	0:06:16	83:57:30	14,69
s2_EME	0:05:03	0:00:31	28:24:50	108:15:28	0:01:38	0:00:04	2:21:09	0:08:46	31:02:01	5,43
s5_MEM	0:00:31		1:00:11		0:00:12		0:17:05		1:17:59	0,23
s7_MMM	0:00:06	0:00:13	0:09:18	0:00:00	0:00:01	0:00:00	0:03:35	0:00:00	0:13:13	0,04
T8eta	0:08:29	0:00:04	32:03:37	2:35:22	0:03:02	0:00:00	2:50:05	0:00:35	37:41:14	6,59
Σύνολο	1:11:09	0:02:53	283:47:55	246:17:56	0:23:21	0:00:25	24:43:08	0:39:10		
	1:21:30		544:24:40		0:23:46		25:22:18		571:32:14	100
ποσοστό %	0,24		95,25		0,07		4,44			
	100									

Σχήμα 6.23: Διάρκειες πραγματικών χρόνων εκτέλεσης ελέγχων.

project	B	T	B	T	project
	-q 0 -s 300000	-s 50000	-q 0 -s 300000	-q 0 -s 5000	
	-q 0 -s 300000 -v 299999		-q 0 -s 300000 -v 299999	-q 0 -s 5000 -v 4999	
	-q 1 -f 1 -s 300000 -v 299999 -o 2		-q 1 -f 1 -s 300000 -v 299999 -o 2	-q 1 -f 1 -s 5000 -v 4999 -o 2	
	-q 1 -s 300000		-q 1 -s 300000	-q 1 -s 5000	
	-q 1 -s 300000 -f 1		-q 1 -s 300000 -f 1	-q 1 -s 5000 -f 1	
	-q 1 -s 300000 -f 2		-q 1 -s 300000 -f 2	-q 1 -s 5000 -f 2	
	-q 1 -s 300000 -f 3		-q 1 -s 300000 -f 3	-q 1 -s 5000 -f 3	
	-q 1 -s 300000 -f 4		-q 1 -s 300000 -f 4	-q 1 -s 5000 -f 4	
	-q 1 -s 300000 -v 299999		-q 1 -s 300000 -v 299999	-q 1 -s 5000 -v 4999	
	-q 1 -s 300000 -v 299999 -f 1		-q 1 -s 300000 -v 299999 -f 1	-q 1 -s 5000 -v 4999 -f 1	
	-q 1 -s 300000 -v 299999 -f 2		-q 1 -s 300000 -v 299999 -f 2	-q 1 -s 5000 -v 4999 -f 2	
	-q 1 -s 300000 -v 299999 -f 3		-q 1 -s 300000 -v 299999 -f 3	-q 1 -s 5000 -v 4999 -f 3	
	-q 1 -s 300000 -v 299999 -f 4		-q 1 -s 300000 -v 299999 -f 4	-q 1 -s 5000 -v 4999 -f 4	
	-q 2 -s 300000		-q 2 -s 30000 -v 29999 -o 20	-q 2 -s 5000	
	-q 2 -s 300000 -f 1		-q 2 -s 300000	-q 2 -s 5000 -f 1	
	-q 2 -s 300000 -f 10		-q 2 -s 300000 -f 1	-q 2 -s 5000 -f 10	
	-q 2 -s 300000 -f 2		-q 2 -s 300000 -f 10	-q 2 -s 5000 -f 2	
	-q 2 -s 300000 -f 25		-q 2 -s 300000 -f 2	-q 2 -s 5000 -f 25	
	-q 2 -s 300000 -f 3		-q 2 -s 300000 -f 25	-q 2 -s 5000 -f 3	
	-q 2 -s 300000 -f 4		-q 2 -s 300000 -f 3	-q 2 -s 5000 -f 4	
	-q 2 -s 300000 -v 299999		-q 2 -s 300000 -f 4	-q 2 -s 5000 -v 4999	
b12_EaE	-q 2 -s 300000 -v 299999 -f 1		-q 2 -s 300000 -v 299999	-q 2 -s 5000 -v 4999 -f 1	
	-q 2 -s 300000 -v 299999 -f 2		-q 2 -s 300000 -v 299999 -f 1	-q 2 -s 5000 -v 4999 -f 2	
	-q 2 -s 300000 -v 299999 -f 3		-q 2 -s 300000 -v 299999 -f 2	-q 2 -s 5000 -v 4999 -f 3	
	-q 2 -s 300000 -v 299999 -f 4		-q 2 -s 300000 -v 299999 -f 3	-q 2 -s 5000 -v 4999 -f 4	
	-q 2 -s 300000 -v 299999 -o 4200		-q 2 -s 300000 -v 299999 -f 4	-q 2 -s 5000 -v 4999 -o 20	
	-q 4 -s 300000		-q 2 -s 300000 -v 299999 -o 20	-q 3 -s 5000	
	-q 4 -s 300000 -f 1		-q 3 -s 300000	-q 3 -s 5000 -v 4999	
	-q 4 -s 300000 -f 10		-q 3 -s 300000 -v 299999	-q 4 -s 5000	Hta
	-q 4 -s 300000 -f 2		-q 4 -s 300000	-q 4 -s 5000 -f 1	
	-q 4 -s 300000 -f 3		-q 4 -s 300000 -f 1	-q 4 -s 5000 -f 2	
	-q 4 -s 300000 -f 4		-q 4 -s 300000 -f 10	-q 4 -s 5000 -f 3	
	-q 4 -s 300000 -v 299999		-q 4 -s 300000 -f 2	-q 4 -s 5000 -v 4999	
	-q 4 -s 300000 -v 299999 -f 1		-q 4 -s 300000 -f 3	-q 4 -s 5000 -v 4999 -f 1	
	-q 4 -s 300000 -v 299999 -f 2		-q 4 -s 300000 -f 4	-q 4 -s 5000 -v 4999 -f 2	
	-q 4 -s 300000 -v 299999 -f 3		-q 4 -s 300000 -v 299999	-q 4 -s 5000 -v 4999 -f 3	
	-q 4 -s 300000 -v 299999 -f 4		-q 4 -s 300000 -v 299999 -f 1	-q 4 -s 5000 -v 4999 -f 4	
	-q 7 -s 300000		-q 4 -s 300000 -v 299999 -f 2	-q 5 -s 5000	
	-s 1000000		-q 4 -s 300000 -v 299999 -f 3	-q 5 -s 5000 -v 4999	
	-s 300000		-q 4 -s 300000 -v 299999 -f 4	-q 6 -s 5000	
	-s 300000 -r 0 -d 10		-q 5 -s 300000	-q 6 -s 5000 -v 4999	
	-s 300000 -r 0 -r 0 -r 0		-q 5 -s 300000 -v 299999	-q 7 -s 5000	
	-s 300000 -v 299999 -o 1		-q 6 -s 300000	-q 7 -s 5000 -v 4999	
	-s 50000		-q 6 -s 300000 -v 299999	-s 10000	
	-q 0 -s 300000	-s 2000	-q 7 -s 300000	-s 2000	
	-q 0 -s 300000 -v 299999	-s 5000	-q 7 -s 300000 -v 299999	-s 5000	
	-q 1 -f 1 -s 300000 -v 299999 -o 2		-s 1000000	-s 5000 -r 0 -d 10	
	-q 2 -s 300000 -v 299999 -o 280		-s 2000	-s 5000 -r 0 -r 0 -r 0	
	-q 3 -s 300000 -v 299999		-s 20000	-s 5000 -v 4999 -o 1	
	-q 5 -s 300000 -v 299999		-s 200000	(b/t?)	
	-q 6 -s 300000 -v 299999		-s 30000 -r 0 -d 10	-q 1 -f 1 -s 300000 -v 299999 -o 2	
	-q 7 -s 300000		-s 30000 -r 0 -r 0 -r 0	-q 4 -s 5000 -f 10	
T8eta	-q 7 -s 300000 -v 299999		-s 300000	-q 4 -s 5000 -f 4	
	-s 1000		-s 300000 -r 0 -d 10	-s 1000000	
	-s 1000000		-s 300000 -r 0 -r 0 -r 0	-s 200000	
	-s 2000		-s 300000 -v 299999 -o 1	-s 300000	
	-s 300000		-s 400000	-s 40 -q 1 -f 2	
	-s 300000 -r 0 -d 10		-s 1000000	-s 50000	b11_Ea2E
	-s 300000 -r 0 -r 0 -r 0		-s 1000000	-s 50000	s0_EEE
	-s 300000 -v 299999 -o 1		-s 300000		s2_EME
	-s 5000		-s 300000		s5_MEM
			-s 50000	-s 50000	s7_MMM

Σχήμα 6.24: Σενάρια που εκτελέστηκαν ανά διαφορετική εκδοχή.

Κεφάλαιο 7

Σχετικές εργασίες

Τίποτα δεν είναι αποκομμένο από τον περίγυρό του, και θεωρώντας χώρο το πεδίο αναφοράς και δράσης, υπάρχουν τόσο χωρικοί, όσο και χρονικοί συσχετισμοί. Μίας εργασίας έχουν προηγηθεί, χρονικά, άλλες, συναφούς θεματολογίας. Η ίδια, όπως αποκρυσταλλώνεται σε κάποια συγκεκριμένη περιγραφή, δεν είναι παρά ένα στιγμιότυπο των μορφών που μπορεί να λάβει, εξελισσόμενη καθαυτή. Στο μέλλον, νέες μελέτες θα διερευνήσουν το χώρο, επεκτείνοντάς τόσο τα αποτελέσματα, όσο και τον ίδιο τον ορισμό του. Παρακάτω, αναφέρονται, συγκριτικά, σχετικές εργασίες της βιβλιογραφίας, ενώ προτάσεις για μελλοντικές εργασίες συμπεριλαμβάνονται στην ενότητα 8.2. Κάποια σημεία που ήδη διαφαίνεται η ανάγκη βελτίωσής τους, αναφέρονται αναλυτικά στην ενότητα Α΄.6.

Συστήματα υλικού με πολλαπλές ουρές εμφανίζονται, εδώ και χρόνια, κατά κύριο λόγο σε μεταγωγείς δικτύου (switches / routers) υψηλών ταχυτήτων, που εκτελούν πολύπλοκο έλεγχο επί ροών με διαφορετικές απαιτήσεις προτεραιότητας και ποιότητας εξυπηρέτησης. Χαρακτηριστικά παραδείγματα, οι εργασίες των Χ. Κοζυράκη ([16], [17]), Α. Νικολογιάννη ([18], [19]), Δ. Καψάλη ([20]), Κορνάρου και άλλων [21]. (Σχήμα 7.1)

Στην πρώτη, ως υποσύστημα του ATM (Asynnchronous Tranfer Mode) μεταγωγέα ATLAS I, σχεδιάζεται ένας ομόχειρος διαχειριστής 54 ουρών, σε κοινό χώρο αποθήκευσης 256 κελιών, για υλοποίηση σε ολοκληρωμένο κύκλωμα VLSI, με συχνότητα λειτουργίας 80 MHz.

Ακόμα και σε επίπεδο λογικής οργάνωσης, η σχεδίαση διαφέρει, κατά το δυνατόν, από την παρούσα, και όχι μόνο λόγω της ενσωματωμένης υποστήριξης δικτυακών πρωτοκόλλων όπως για credit-based flow control και multicasting. Η καθεαυτή διαχείριση των ουρών, γίνεται σε τρία στάδια ομοχειρίας, από το κυρίως ομόχειρο μονοπάτι, με την ενεργοποίηση της FreeList στο πρώτο στάδιο των NQ και στο τρίτο των DQ, ενώ, υπερβαθμωτά, από ένα δεύτερο ομόχειρο μονοπάτι εκτελούνται εντολές NQ, στοιχείων ήδη αποθηκευμένων, ασύνδετα, στη μνήμη. Δευτερευόντως, ο αριθμός ουράς δεν δίνεται, αλλά υπολογίζεται εσωτερικά, ενώ οι δείκτες των ουρών αποθηκεύονται σε ενιαί-

α μνήμη, και μαζί τους συμπεριλαμβάνεται και ένα bit ενδεικτικό της άδειας ουράς, παρότι το ένα εναπομείναν στοιχείο, υπολογίζεται, κανονικά. Οι διαφορετικές ουρές, εξυπηρετώντας ροές διαφορετικής προτεραιότητας, μπορεί και να δικαιούνται διαφορετικού μεγέθους προορισμένα τμήματα του κοινού χώρου αποθήκευσης, εποπτευόμενα από μετρητή. Η FL είναι μία χαρτογράφηση των διαθέσιμων στοιχείων, επί συστοιχίας (256) καταχωρητών, πλάτους ενός bit, των οποίων προηγείται ένας αποκωδικοποιητής (decoder), για τις εγγραφές, και ακολουθεί ένας κωδικοποιητής προτεραιότητας (priority encoder), για τις αναγνώσεις. Ο κωδικοποιητής προτεραιότητας, στην συγκεκριμένη υλοποίηση, συντίθεται από ένα κύκλωμα επιλογής προτεραιότητας (priority enforcer), που διαλέγει την ανώτερη από τις ενεργές εισόδους του, επιτρέποντας την χρήση απλού κωδικοποιητή, στην συνέχεια. Ειδικότερα, η σχεδίαση του priority enforcer είναι ιδιαιτέρως εκλεπτυσμένη, εκμεταλλευόμενη πλήθους βελτιστοποιήσεων υλικού, με απλούστερη αυτών την ομόχειρη ανάπτυξη. Επίσης, η επαλήθευση πραγματοποιείται συγκρίνοντας δύο εκδοχές του συστήματος σε γλώσσα περιγραφής υλικού, μία πλήρους αντιστοιχίας με την υλοποίηση και μία αποκλειστικά λειτουργικής. Σε χρόνο εκτέλεσης, παράγονται και τροφοδοτούνται, είσοδοι τυχαίες αλλά με περιορισμούς, αλλά και το πλήρες σύνολο περιπτώσεων που απαιτούν παράκαμψη δεδομένων, και συγκρίνονται τόσο οι έξοδοι, όσο και η εσωτερική κατάσταση των ουρών.

Στην επόμενη ([18], [19]), ως τμήμα ενός μεταγωγέα ταχύτητας 10 Gbps, σχεδιάζεται και το υποσύστημα διαχείρισης ουρών, με δυνατότητα υποστήριξης πολλών χιλιάδων διαφορετικών ουρών, συγκεκριμένα 64 k, σε συχνότητα λειτουργίας 100 MHz.

Η επιλογή χρήσης μνημών τεχνολογίας Rambus, καθορίζει εν πολλοίς την σχεδίαση. Λόγω των εξορισμού καθυστερήσεων και πρόσθετων απαιτήσεων χρονισμού για την αποφυγή συγκρούσεων των τμημάτων (bank conflicts), οι προσβάσεις των εντολών στη μνήμη πραγματοποιούνται εκτός σειράς (out-of order), και οι κίνδυνοι που εμφανίζονται, αντιμετωπίζονται με την μετονομασία των τελεστέων (Tomasulo operand-renaming). Οι τιμές μεταξύ των εκκρεμών εντολών της ίδιας ουράς προωθώνται, και οι δομές του συστήματος ενημερώνονται μόνο για την τελευταία από αυτές. Οι εντολές NQ, DQ εκτελούνται ως δύο (από έξι συνολικά) ομόχειρες διαδικασίες η καθεμία, μεταξύ τους ασύγχρονες, ενός σταδίου η πρώτη, και δύο σταδίων η δεύτερη, με κάθε στάδιο να διαρκεί τέσσερις κύκλους ρολογιού. Οι δείκτες των Head, Tail των ουρών διατηρούνται σε διαφορετικές υπομονάδες μνήμης, αλλά αντίθετα, οι δείκτες Next διατηρούνται από κοινού με τα Data. Για να αποφευχθούν, κατά τα NQ, οι δύο προσβάσεις σε διαφορετικές διευθύνσεις μνήμης, στο τέλος κάθε ουράς διατηρείται ένα κενό στοιχείο, στο οποίο θα γραφτούν τόσο τα δεδομένα όσο και ο δείκτης next, συνδέοντας το επόμενο προδεσμευμένο στοιχείο (free buffer preallocation). Αντίστοιχα, για την αποφυγή των bank conflicts, η FL οργανώνεται σε 512 διαφορετικές ουρές, (αντί για μία και στοίβα, εδώ), εκτελώντας παράκαμψη της λειτουργίας της, κατά την ταυτόχρονη εκτέλεση NQ / DQ, αλλά μέσω μίας ουράς κενών στοιχείων, σε cache 10 θέσεων. Επιπλέον,

για την αναζήτηση της κατάλληλης εκκρεμούς πρόσβασης εκτός σειράς στη μνήμη, αλλά και μίας εκ των 512 ουρών της FL, χρησιμοποιούνται και πάλι CAMs και priority encoders. Το σύστημα, για λόγους επαλήθευσης περιγράφηκε σε Verilog, παράγοντας σύνολα εισόδων με κώδικα C, και συγκρίνοντας αρχεία εξόδων, αλλά σε μικρής μόνο έκτασης ελέγχους.

Στην εργασία [20] σχεδιάζεται ένας διαχειριστής 64 χιλιάδων ουρών, ως τμήμα ενός μεταγωγέα 1 Gbps, υλοποιημένο σε FPGA, με συχνότητα λειτουργίας 35 MHz.

Τόσο τα δεδομένα, όσο και οι δείκτες των ουρών αποθηκεύονται από κοινού σε εξωτερική SDRAM χωρίς κανένα παραλληλισμό στην εκτέλεση των εντολών, αναιρώντας την πιθανότητα εξαρτήσεων και κινδύνων. Η FL υλοποιείται ως Στοίβα στο εσωτερικό της FPGA και οι βελτιστοποιήσεις free list bypassing και buffer pre-allocation εμφανίζονται, αποσκοπώντας πλέον στην μείωση της διάρκειας εκτέλεσης των εντολών. Βασική ιδιαιτερότητα είναι η οργάνωση των ουρών, καθεμία από τις οποίες αντιστοιχεί σε μία ροή, σε 64 ομάδες ροών, εκτελώντας εισαγωγές προς συγκεκριμένη ροή, αλλά εξαγωγές από ομάδα ροής, με πολιτική round-robin. Η διατήρηση των κυκλικών λιστών των ομάδων, εισάγει μία διακύμανση στη διάρκεια εκτέλεσης των εντολών, που διπλασιάζεται στις ακραίες περιπτώσεις, δημιουργίας και αδειάσματος. Ακόμα, οι εντολές εξαγωγής επιστρέφουν και την διεύθυνση των δεδομένων, επιτρέποντας την επανάληψη ανάγνωσης και μετάδοσης, σε περίπτωση αποτυχίας. Τόσο για κάθε ουρά, όσο και για την FL διατηρούνται μετρητές, για εφαρμογή πρωτοκόλλου ελέγχου της συμπεριφοράς των ροών.

Στην εργασία [21], υλοποιείται ένα σύστημα διαχείρισης (32) χιλιάδων ουρών, με απόδοση 6.14 Gbps σε συχνότητα λειτουργίας 125 MHz, ως μεταφορά μίας ASIC σχεδίασης σε FPGA τεχνολογία.

Επιτεύχθηκε ένα σύστημα που, αφενός ξεπερνούσε σε απόδοση τις σύγχρονές του εκτελέσεις της διαχείρισης μνήμης με λογισμικό, και αφετέρου ήταν σε θέση να συνδεθεί με πλήθος διαφορετικών φυσικών διεπαφών και επεξεργαστών, επεξεργαζόμενο δεδομένα τόσο σταθερού, όσο και μεταβλητού μήκους, βάσει κοινού μεγέθους κατάτμησης, τα 64 bytes. Όλες οι προαναφερθείσες τεχνικές, συνυπάρχουν με πρόσθετες. Πέραν της ομοχειρίας, που χρησιμοποιείται στο σύνολο, εκτός μόνο από το υποσύστημα στην καρδιά, του διαχειριστή, οι προσβάσεις στις μνήμες αναδιατάσσονται, για αποφυγή εξαρτήσεων των δεδομένων, αλλά και μείωση των καθυστερήσεων από τις συγκρούσεις. Η αποφυγή των συγκρούσεων προσανατολίζει την διαίρεση της FL σε δύο λίστες, για ταυτόχρονη ελαχιστοποίηση των προσβάσεων. Καθεμία λίστα της FL αποτελείται από δύο δείκτες, τόσο προς πακέτο, όσο και τμήμα του, όπως είναι και η οργάνωση των ουρών του συστήματος. Η απώλεια δεδομένων από υπερχειλίσεις προλαμβάνεται με την έγερση σήματος backpressure, ενώ το εσωτερικό πρωτόκολλο επικοινωνίας του διαχειριστή με τους επεξεργαστές, στηρίζεται σε semaphores. Η επικοινωνία αυτή επιταχύνεται σημαντικά, αντικαθιστώντας ένα από τα προσφερόμενα buses της Xilinx με ένα ιδιότυπο. Μετρήσεις γίνονται ανεξάρτητα, με το ίδιο το σύστημα να λειτουργεί σε έναν

	Κοζυράκης	Νικολογιάννης	Καψάλης	Νικ. / Παπ. / Κορ. / Καχ.	Πέτσα
χρονολογία	1996	2000	2002	2004	2011
σύστημα	ATLAS i (ATM switch)	-	DiPOLO (ATM Sw.)	(NPMADE)	-
ονομασία	Queue Management Block	Datapath & Queue Management chip	Queue Manager	QMS (Queue Management System)	Pipelined Queue Manager
συχνότητα λειτουργίας	80 MHz	100 MHz	35 MHz	125 MHz	-
τεχνολογία στόχος	VLSI ASIC	VLSI ASIC	FPGA	FPFA	FPGA
πλήθος ουρών	54	64 k	64 k	32 k	παραμετροποιήσιμο
διαθέσιμες θέσεις	256	4M (2^22)	~4M	4 M	παραμετροποιήσιμο
διαχωρισμός θέσεων	ναι (βάσει μετρητών)	όχι	όχι	όχι	όχι
στάδια ομοχειρίας	3	6 pl processes: 1, 2, 5 (x4 cc)	-	-	4
επιλογή ουράς	Υπολογίζεται εσωτερικά	Υπολογίζεται εσωτερικά	είσοδος/rr	?	είσοδος
HeadMem	4-θυρη, 54x17 (H/T ενιαία)	64k x 24	256 MB	(32 k)	2-θυρη, (μεταβλ.)
TailMem		64k x 23			2-θυρη, (μεταβλ.)
DataMem	256x12, 256x16, 256x17	4M x 64-bytes (χρήση από κοινού) συν FL H/T Table: 512x45 FL Cache: 10x22		(4 M)	1-θυρη, (μεταβλ.)
NextMem	2-θυρη, 256x12			(4 M)	2-θυρη, (μεταβλ.), (χρήση από κοινού)
free list mem	2-θυρη, 256x1, CAM			?	
δομή FreeList	χαρτογράφηση ενός bit (dec + D-ff + pr.Enf+ enc)	512 διαφορετικές ουρές (ανά memory bank)	στοίβα	2 διπλές λίστες (ανά memory bank, δείκτες segment + packet)	στοίβα εντός μνήμης δεικτών Next
στάδια εκτέλεσης FL	st1(NQ), st3(DQ)	NQ, DQ execution process	-	-	st2(NQ), st3(DQ)
ιδιαιτερότητες / βελτιστοποιήσεις	~ χρήση ομόχειρων μνημών ~ valid bit σε H/T mem ~ υπερβαθμωτό NQ (2 p.l.) ~ "εσωτερικό" bpss μνημών ~ ομόχειρος++ priority enforcer	~εκτέλεση εκτός σειράς ~μετονομασία τελεστέων (Tomasulo) ~εγγραφή τελευταίου ~FL bypassing ~FL buffer preallocation ~search: CAM + pr.encoder ~διεπαφή μνήμης Rambus	~FL bypassing ~FL buffer preallocation ~μετρητές σε ουρές, FL ~ομαδοποίηση ροών/ουρών ~επανάληψη ανάγνωσης στοιχείου	~ομοχειρία ~pre-fetching ~αναδιάταξη / διαιτησία προσβάσεων ~backpressure sigs ~επικοινωνία μέσω semaphores ~ευελιξία	~ FL bypassing
επαλήθευση	~ 2 μοντέλα, σε Verilog ~ παραγωγή εισόδων κι έλεγχοι κατά την εκτέλεση ~ είσοδοι τυχαίες με περιορισμούς, και πλήρες σύνολο προϊόντησης ~ σύγκριση εξόδων, και κατάστασης ουρών	~μοντέλο σε Verilog ~ είσοδοι, έξοδοι και συγκρίσεις σε αρχεία ~ είσοδοι τυχαίες, με περιορισμούς, παραγωγή με κώδικα C ~περιορισμένη αποσφαλμάτωση	~Altera Max Plus ii ~είσοδοι φορτώνονται από CPU interface ~εκτεταμένη, on- board	~Xilinx Virtex Pro ii Platform FPGA + Microblazes + IBM PowerPC ~εκτεταμένοι έλεγχοι με μικροκώδικα ~σύγκριση με εκτέλεση σε λογισμικό	~ VHDL σύστημα, python μοντέλο ~ είσοδοι, έξοδοι και συγκρίσεις σε αρχεία ~ είσοδοι τυχαίες, με περιορισμούς ~ σύγκριση εξόδων

Σχήμα 7.1: Αντιπαραβολή σχετικών εργασιών.

ενδεικτικό επεξεργαστή δικτύου, τον NPMADE, αλλά και συγκριτικά, με την διαχείριση μνήμης να εκτελείται ως λογισμικό σε σύγχρονους επεξεργαστές, κατά το σύνηθες.

Κάθε νέα πρόοδος στις τεχνολογίες υλοποίησης καθιστά δυνατό τον εμπλουτισμό των σχεδιάσεων με δομές που προηγουμένως ήταν ανέφικτες, λόγω ανεπάρκειας διαθέσιμων λογικών στοιχείων και μνήμης, ή ταχύτητας. Σε πρόσφατες εργασίες, ουρές προστίθενται εκεί που προηγουμένως απουσίαζαν, και χωρίς καν την χρήση των πιο εκλεπτυσμένων τεχνικών, προσφέρουν σημαντικές βελτιώσεις επιδόσεων αλλά και μείωση πολυπλοκότητας.

Χαρακτηριστική περίπτωση, και πλέον γενικευμένη τάση, η χρήση επιμέρους ουρών σε κάθε σημείο διασύνδεσης (crosspoint) ενός μεταγωγέα τοπολογίας crossbar. ([22]) Η ύπαρξή τους, χαλαρώνει την σχέση εξάρτησης μεταξύ των επιλογών του χρονο-προγραμματιστή (scheduler), επιτρέποντας την ανε-

ξάρτητη επιλογή προέλευσης (εισόδου) των δεδομένων που θα εξαχθούν από κάθε θύρα εξόδου. Η ανεξαρτητοποίηση έχει αντίκτυπο στο χώρο και στο χρόνο. Χωρικά, απλοποιείται η δομή των χρονο-προγραμματιστών, επιταχύνοντας την λειτουργία τους. Σε χρονικό επίπεδο, δεν απαιτείται πλέον ο συγχρονισμός όλων των θυρών, καθιστώντας δυνατή την άμεση μεταγωγή πακέτων κυμαινόμενου μεγέθους (variable-size switching), χωρίς την κατάτμηση σε σταθερού μεγέθους τμήματα και επανένωσή τους (Segmentation And Reassembly, SAR) πριν την έξοδο. Αναιρείται έτσι, η ανάγκη εσωτερικής επιτάχυνσης (internal speedup), για την εξισορρόπηση των καθυστερήσεών τους. Συνεπώς, οι ήδη ταχύτεροι μεταγωγείς, μπορούν να ανταποκριθούν σε μεγαλύτερες από προηγουμένως εξωτερικές ταχύτητες, ίσες με εκείνες που λειτουργούσαν εσωτερικά. Μάλιστα, σε συνδυασμό με την έλλειψη εσωτερικής επιτάχυνσης, η απουσία SAR εξαλείφει τους buffers στις εξόδους, εξοικονομώντας σε υλικό. Υπερέχουν και σε ζητήματα δικαιοσύνης, έχοντας μεγαλύτερη διακριτική ικανότητα στην διαχείριση και τιμωρία επιβαρυντικών ροών.

Βέβαια, οι παλιοί προβληματισμοί επανακάμπτουν, απαιτώντας τις πεπατημένες λύσεις εξισορρόπησης: τα πακέτα μεταβλητού μεγέθους απαιτούν ελάχιστο χώρο όσο το μέγιστο αναμενόμενο. Όταν δεν κρίνεται διαθέσιμος, επανέρχεται η κατάτμηση, τουλάχιστον σε νέα πλαίσια, σε μεταβλητά και όχι σταθερά μεγέθη, όπως αρχικά. Για την αποφυγή επαναφοράς και της επανένωσης, προτείνεται ο περιορισμός της ανεξαρτησίας των χρονο-προγραμματιστών! Αλλά, η διευρυμένη οπτική έχει κατακτηθεί, παρά τα πισωγυρίσματα: τελικά, διαπιστώνεται ότι ακόμα και χωρίς buffers στα σημεία διασύνδεσης, οι μεταγωγείς μπορούν να χρονο-προγραμματιστούν ασύγχρονα, και άρα, να εξυπηρετήσουν πακέτα μεταβλητού μεγέθους. Για την προσφορά QoS, και πάλι, η απροθυμία παροχής τόσων ουρών όσες και οι κλάσεις προτεραιότητας, σε κάθε σημείο διασύνδεσης, αντισταθμίζεται με κάποιο μικρότερο αριθμό ουρών, και ανακατανομή των κλάσεων ανάλογα.

Πέραν από την δομική εισαγωγή επιπλέον ουρών σε μία σχεδίαση, πλεονεκτήματα φαίνεται να προσφέρει ([23]) και η αύξηση του αριθμού τους με καθαρά λογικές κατατμήσεις. Αν οι ουρές σε κάθε έξοδο θεωρηθούν μονοδιάστατες, διδιάστατες εκείνες στα σημεία διασύνδεσης, συνδέοντας ζεύγη εισόδου-εξόδου, τότε τρισδιάστατες, θα είναι οι διδιάστατες, χωρισμένες σε αυθαίρετο αριθμό τμημάτων. Η κατάτμηση αυτή, επιτρέπει το διαμοιρασμό του φορτίου (load-balancing) χωρίς αλλαγή της σειράς αναχώρησης των πακέτων σε περιπτώσεις απωλειών, που θεωρείται επιτρεπτό, αλλά αποφευκτέο. Άλλος τρόπος λογικής οργάνωσης του συνόλου των ουρών του μεταγωγέα, είναι η κατάτμηση σε κεφαλή, μέση και τέλος. Ομαδοποιώντας τα αναλόγως, μπορεί να εξαλειφθεί η απώλεια πακέτων από υπερχείλιση, φράσσοντας το αναγκαίο μέγεθος της μνήμης για το κεντρικό κομμάτι, εφαρμόζοντας στο αρχικό πολιτικές διαμόρφωσης ροής (traffic shaping).

Κεφάλαιο 8

Επίλογος

8.1 Σύνοψη

Ανακεφαλαιώνοντας, σχεδιάστηκε ένα σύστημα διαχείρισης ουρών σε γλώσσα περιγραφής υλικού VHDL. Από την ανάλυση του προβλήματος, το σύστημα χωρίστηκε σε τέσσερα στάδια ομοχειρίας, με τέσσερις διαφορετικές μνήμες να διατηρούν τους δείκτες και τα δεδομένα των δομών, τις Head, Tail, Next, Data. Η διαχείριση των δεικτών των διαθέσιμων θέσεων πραγματοποιείται από μία δομή Στοίβας, τη FreeList, που συνυπάρχει με τους κατειλημμένους δείκτες στην μνήμη Next, καθώς αποτελούν ασυμβίβαστα σύνολα. Η λειτουργία της επιταχύνεται με προώθηση κάποιων δεικτών, αντί καταγραφής τους, στην περίπτωση που εκτελούνται ταυτόχρονα εντολή εισαγωγής και εντολή εξαγωγής στοιχείου. Οι κίνδυνοι δεδομένων αντιμετωπίζονται με προώθηση, που δεν εισάγει καθυστερήσεις σε καμία περίπτωση.

Το αρχικό σύστημα μεταφράστηκε σε συνθέσιμο κώδικα για συσκευές αναδιατασσόμενης λογικής FPGA , με το εργαλείο σύνθεσης Xilinx ISE. Αξιοποιώντας την δυνατότητα παραμετροποίησης του κώδικα του συστήματος, συντέθηκαν εκδοχές διαφορετικής διαστασιοποίησης, που στην συνέχεια ελέγχθηκαν, όπως και οι αντίστοιχες λειτουργικές εκδοχές, με εκτενή σύνολα εισόδων, χιλιάδων ή και εκατομμυρίων, σε κάθε δοκιμή, ξεπερνώντας συνολικά τα 71 εκατομμύρια εντολές. Η απουσία σφαλμάτων στην αντιπαραβολή των εξόδων του συστήματος ως προς του μοντέλου, που εκτέλεσε το ίδιο σύνολο εισόδων, αποτελεί ισχυρή ένδειξη για την λειτουργική ορθότητα του συστήματος.

Οι δοκιμές εκτελέστηκαν πλήρως αυτοματοποιημένα, από ιδιότυπο συνοδευτικό σύστημα ελέγχου. Ανεπτυγμένο κυρίως σε γλώσσα python, λαμβάνει τις επιλογές του χρήστη, τις μεταφράζει σε ένα σύνολο εισόδων, καλεί την εκτέλεση της προσομοίωσης του συστήματος σε περιβάλλον ModelSim, εκτελεί το μοντέλο, καταγράφει τις εξόδους του, και συγκρίνει τα αρχεία των εξόδων συστήματος και μοντέλου. Η καταγραφή των εξόδων του συστήματος γίνεται από το ModelSim, σε εντολές περιβάλλοντος και γλώσσα Tcl. Οι μειωμέ-

νες επιδόσεις των διαθέσιμων μηχανημάτων, και ο φόρτος των εκτελέσεων, οδήγησαν στην αναζήτηση λύσεων επιτάχυνσης του συστήματος ελέγχου.

8.2 Μελλοντικές εργασίες

Δεδομένης της κατεύθυνσής της ως άσκησης, η παρούσα εργασία θα μπορούσε, σε πρώτο επίπεδο, να διορθωθεί και να συμπληρωθεί στο ίδιο σχεδιαστικό πλαίσιο, και σε δεύτερο, να επανασχεδιαστεί, σε μεγάλο βαθμό.

Καταρχάς, μετά την μετάβαση στην δεύτερη εκδοχή του (v2), το μονοπάτι δεδομένων θα μπορούσε να έχει συμπτυχθεί σε τρία στάδια, με το πρώτο ίδιο, στο δεύτερο την ανάγνωση της NextMem, μαζί με την πρόσβαση στη DataMem και τον υπολογισμό των σημάτων κατάστασης (empty, one), και στο τρίτο τις εγγραφές των HeadMem, TailMem μαζί με την εγγραφή στη NextMem. Βέβαια, πρέπει να παρατηρηθεί, ότι μία τέτοια μετατροπή δεν θα είχε επίδραση στον σχετικό χρονισμό των σταδίων που η NextMem λειτουργεί, άρα ούτε και στις παρακάμψεις που απαιτούνται. Ακόμα και για τις προωθήσεις των HeadMem, TailMem, το μόνο που θα άλλαζε θα ήταν το εύρος των εμπλεκόμενων εντολών, παρά οι έλεγχοι και ο χειρισμός τους. Στην ίδια σχεδίαση, των τεσσάρων σταδίων, θα μπορούσε να επιχειρηθεί η επαναφορά των μνημών σε λειτουργία θετικής ακμής, με ενδεχόμενες αλλαγές στο κομμάτι της προώθησης. Το σκοπό αυτό, θα εξυπηρετούσε και κάποια μετατόπιση του σημείο προορισμού αργότερα στο μονοπάτι, χρησιμοποιώντας στη θέση των empty, one τα ίδια τα cH, cT.

Στα πλαίσια της εξάσκησης, θα μπορούσε να υποστηρίζεται κάποια δυναμική αναδιάταξη εντολών, με σκοπό την αποφυγή των προωθήσεων. Καθώς ήδη δεν προκαλούνται καθυστερήσεις και ανασχέσεις, δεν θα υπήρχε μεγάλο χρονικό κέρδος, εκτός από μία ενδεχόμενη βελτίωση της διάρκειας του κύκλου, που θα έπρεπε να εκτιμηθεί, όπως και το συγκριτικό κόστος σε υλικό. Κι αυτά μόνο υποθετικά, αν το κομμάτι ελέγχου προώθησης εξαιρούνταν, πόσο μάλλον όταν η ανάγκη της δεν μπορεί να εξαλειφθεί τελείως, ακόμα και με αναδιάταξη. Στον αντίποδα, μικρή βελτιστοποίηση, θα ήταν η υποστήριξη παράκαμψης δεδομένων, δηλαδή να αποδίδονται άμεσα χωρίς να γραφούν, στις σπάνιες περιπτώσεις που θα ζητούνταν εισαγωγή σε μία άδεια ουρά και εξαγωγή από την ίδια, αν όχι στην επόμενη εντολή, αρκετά κοντά, ώστε η προηγούμενη να περιέχεται ακόμα στο μονοπάτι και να μην έχει μεταβάλλει την κατάσταση του συστήματος. Η εμφάνιση τέτοιων ακολουθιών εντολών, βέβαια, θα αποδυναμώνονταν από την αναδιάταξη, αν δεν εξαλείφονταν τελείως. Παρότι η αναδιάταξη δεν μοιάζει να ωφελεί ιδιαιτέρως τις επιδόσεις, ωστόσο είναι, μακροσκοπικά, μία πιθανή χρήση του συστήματος.

Το σύστημα, θα μπορούσε να υποστηρίζει και κάποιες εξαιρέσεις, αποτελούμενες μάλλον από εσωτερικές καταστάσεις, κυρίως την μη περαιτέρω διαθεσιμότητα θέσεων. Ανάγκη που θα προέκυπτε οπωσδήποτε, από την επέκταση για υποστήριξη εσωτερικής ιεραρχίας μνημών, ή θεώρηση του συνόλου των

μνημών του συστήματος ως cache, που μπορεί να επικοινωνήσει με εξωτερικούς πόρους.

Σε κάθε περίπτωση, με τις παραπάνω προσθήκες ή χωρίς, το επόμενο εύλογο βήμα για το σύστημα θα ήταν η βελτιστοποίησή του για την φόρτωση και εκτέλεσή σε FPGA, πρωτίστως ως προς τη διάρκεια του κύκλου και τις λοιπές απαιτήσεις χρονισμού, αλλά και για πιθανές προδιαγραφές ταχύτητας, χώρου ή ενεργειακής κατανάλωσης.

Ο λόγος μη ενσωμάτωσης, παρά μόνο επισήμανσης, των διορθώσεων για το σύστημα και τον Tester της ενότητας (Α΄.6), είναι ότι τα δεδομένα που καταγράφηκαν και παρουσιάζονται θα καθίσταντο ανακριβή. Ο απαιτούμενος επανέλεγχος (regression testing), αρχικά στα επιμέρους, αλλά και η εκτεταμένη επανεκτέλεση δοκιμών, είναι έτσι το πρώτο προτεινόμενο βήμα επεκτάσεων του Tester, ο οποίος έχει και μεγάλα περιθώρια επανασχεδίασης.

Το σημαντικότερο μειονέκτημα της σχεδίασης του Tester είναι η μεγάλη εξάρτηση από συγκεκριμένο περιβάλλον προσομοίωσης (ModelSim). Χωρίς να παραγνωρίζεται το δεδομένο της παρουσίας κάποιου εργαλείου για την εκτέλεση, θα μπορούσε να δοθεί έμφαση στην φορητότητα, λαμβάνοντας τις εισόδους του συστήματος μέσα από συνθετικό κώδικα VHDL, παράγοντας τις εξόδους, και μάλιστα μορφοποιημένες, εκμεταλλευόμενοι την ευελιξία των assertions, και γενικώς, αποφεύγοντας τη χρήση των δυνατοτήτων οποιουδήποτε συγκεκριμένου περιβάλλοντος, με εξαίρεση, ίσως, της γλώσσας Tcl. Σημαντική, προς αυτή την κατεύθυνση, θα ήταν η ενισχυμένη διακριτοποίηση του σημείου πρόσδεσης του προσομοιωτή στον Tester, ώστε να είναι εύκολα προσβάσιμο, για παραμετροποίηση ή ριζική μεταβολή.

Απόρροια μίας στενότερης σύνδεσης με την γλώσσα υλοποίησης, εκτός της φορητότητας, θα μπορούσε να είναι και η ευστάθεια. Για παράδειγμα, η ανάγνωση του αρχείου βιβλιοθήκης libpack και εξαγωγή από εκείνο των τιμών των παραμέτρων, θα κατέληγε σε εγκυρότερη κατασκευή του Qs_dictionary.

Ξεχωριστή προσοχή θα έπρεπε να δοθεί στα παραγόμενα μεταδεδομένα εξόδου του Tester, των μετρήσεων και συμπερασμάτων των εκτελέσεων και της σύγκρισής τους, ώστε να είναι εύχρηστα, τόσο στην συλλογή τους, όσο και την περαιτέρω αυτόματη επεξεργασία.

Τα σύνολα εισόδων προς εκτέλεση από το σύστημα και το μοντέλο, θα μπορούσαν να καλύπτουν πολυπλοκότερα σενάρια, ακόμα και με την συνολική αναδιάρθρωση της παραγωγής τους, και μία περισσότερο σεναριο-κεντρική επανασχεδίαση. Ενδιαφέρουσα θα ήταν η προσέγγιση αναμενόμενων συνόλων εισόδων από τις πιθανές χρήσεις του συστήματος, σαν golden σύνολα, μέθοδος που χρησιμοποιείται και στη βιομηχανία. Σε κάθε περίπτωση, θα έπρεπε να συμπληρώνεται από μία, προσπάθεια έστω, συντεταγμένης παρακολούθησης της κάλυψης, με κάποιο υποτυπώδες coverage model.

Η ίδια η μεθοδολογία επαλήθευσης, θα μπορούσε να προσεγγιστεί διαφορετικά, με επιλογή άλλης μεθόδου ή, άλλης εφαρμογής της ίδιας, ιδιαιτέρως βάσει των επίσημων προτύπων της βιομηχανίας ως προς τη γλώσσα και τη δομή, (SystemVerilog, SystemC, Specman/e και assertions, OVM/UVM, αν-

τίστοιχα), με ή χωρίς την χρήση των προσφερόμενων εργαλείων.

Επιστέγασμα των παραπάνω, θα μπορούσε να είναι κάποια γραφική διεπαφή (GUI) του Tester, που να ενισχύει όχι μόνο την φιλικότητα, αλλά και την λειτουργικότητα. Επιτρέποντας, για παράδειγμα, την άμεση επιλογή των φακέλων και των αρχείων από το σύστημα αρχειοθέτησης, δυναμικότερα κι ασφαλέστερα, ή παρέχοντας κάποια εύληπτη παρουσίαση των αποτελεσμάτων, μεμονωμένων και συγκεντρωτικών.

Στο πεδίο της έρευνας, ο ευρύτερος χώρος που κινείται η εργασία μπορεί να παρουσιάσει αρκετό ενδιαφέρον. Με στόχο τη διεύρυνση της αξιοποίησης των FPGAs, και γενικότερα των άμεσων λύσεων σε υλικό, με ποιοτικά όσο και ευπροσάρμοστα συστατικά, σκόπιμη είναι η ανάπτυξη σε γλώσσα περιγραφής υλικού παραμετροποιήσιμων δομών, που θα μεταφράζονται σε αποδεδειγμένα αποδοτικές υλοποιήσεις. Αφετέρου, για αυτοματοποιημένη και αξιόπιστη επαλήθευση των λύσεων υλικού, απαιτείται η δημιουργία εύχρηστων όσο και αποτελεσματικών εργαλείων λογισμικού, εξειδικευμένων αλλά κυρίως γενικών, συστηματοποιώντας την μελέτη, βάσει των νεο-εισαγόμενων αλλά και των κλασσικών μεθοδολογιών.

Παράρτημα Α΄

Ο Tester λεπτομερέστερα

Όταν πιά δεν συνοδεύει περιορισμένα την υλοποίηση, οι προδιαγραφές της επαλήθευσης, καθεαυτής, την εξυψώνουν σε ένα ξεχωριστό και σχεδιαστικά απαιτητικό στάδιο. Ειδικότερα, η επιλογή της κατά το δυνατό αυτοματοποίησης της, στη συγκεκριμένη εργασία, κατέληξε σε ένα δεύτερο, εκτενέστερο και πολυπλοκότερο από το μελετούμενο, σύστημα ελέγχου. Παρουσιάζεται στην αρχή αφαιρετικά η διάρθρωσή του, και στη συνέχεια αναλυτικότερα, σχολιάζοντας την υλοποίηση, κατ΄ αναλογία ενός εγχειριδίου. Επιπλέον, μία αναγκαία επέκταση της βασικής εκδοχής του Tester και δύο μελέτες επιδόσεων, λόγω των περιορισμών του εξοπλισμού όπου εκτελέστηκαν οι έλεγχοι. Εκτός από τα στοιχεία των συγκεκριμένων εργαλείων, συμπληρώνονται τα συμπεράσματα της σύνθεσης και διευκρινίζεται ο ημι-αυτοματοποιημένος έλεγχος (silver).

Α΄.1 Σύνθεση (Synthesis)

Α΄.1.1 Εργαλεία

Κάθε αλγοριθμική έκφραση μιάς λύσης, έτσι και ένα σύστημα εκφρασμένο σε γλώσσα περιγραφής υλικού, κρίνεται ως προς την αποδοτικότητά της βάσει αναλυτικών μεθόδων και ανεξάρτητα από την γλώσσα υλοποίησης και το υλικό όπου θα εκτελεστεί. Προσομοιώνοντας την έκφραση αυτή σε εργαλεία συγκεκριμένα, και μεταφράζοντάς την σε ορισμένη συσκευή FPGA, η γενικότητα αυτή νοθεύεται και αναπόφευκτα επηρεάζονται οι μετρήσεις επί του υλικού και η πραγματική διάρκεια της εκτέλεσης. Για τις τιμές που παρουσιάστηκαν, στα σχήματα 6.11 έως 6.14, και 6.23, τα εργαλεία που χρησιμοποιήθηκαν είναι:

Η προσομοίωση και ανάπτυξη του συστήματος έγινε σε ModelSim XE (Xilinx Edition) III/Starter 6.3c, Revision : 2007.09, Date : Sep 12 2007. [1]

[1] που διαθέτει προ-μεταφρασμένες τις βιβλιοθήκες της Xilinx, χρησιμοποιώντας εξαρχής Xilinx IP Cores.

Η απεικόνιση σε FPGA, συγκεκριμένα τη συσκευή xc4vsx25 [2] της οικογένειας Virtex4 της Xilinx, έγινε με το Xilinx CORE Generator, Block memory generator LogiCORE v2.8, για την παραγωγή των μνημών, και το Xilinix ISE για την μετάφραση, έκδοσης Release Version 10.1.03 (nt), Application Version K.39, Updates Installed: ISE 10.1 IP Update 1, 2, 3.

Η εκτέλεση των αυτοματοποιημένων ελέγχων πραγματοποιήθηκε σε δύο μηχανήματα. Αρχικά σε ένα φορητό σύστημα DELL Latitude C600, με Microsoft Windows XP Professional, Version 2002, Service Pack 2 σε Intel Pentium III, 1.00GHz, 256 MB RAM, και στη συνέχεια, σε ένα επιτραπέζιο, με Microsoft Windows XP Professional, Version 2002, Service Pack 3, σε AMD Athlon XP 2100+, 1.74 GHz, 256 MB RAM, μιάς και ο φορητός δεν μπορούσε να ικανοποιήσει τις απαιτήσεις σε ιδεατή μνήμη, έχοντας περιορισμένο χώρο στο δίσκο. Στο επιτραπέζιο σύστημα, για την εξυπηρέτηση των προσομοιώσεων των μεγάλων συνόλων ελέγχου, των 10^6 εντολών σε Behavioral και $5 \cdot 10^4$ εντολών σε gate-level, το ελάχιστο μέγεθος του αρχείου σελιδοποίησης (pagefile.sys) τέθηκε αναγκαστικά στο άνω επιτρεπτό όριο μείον το μέγεθος της φυσικής μνήμης, δηλαδή στα 3840MB, και το μέγιστο στα 4096MB(=4GB).

Επιπλέον, το σύστημα ελέγχου αναπτύχθηκε σε γλώσσα Python, 2.6.2, στο περιβάλλον IDLE, και με επιμέρους κομμάτια σε γλώσσα Tcl, 8.5. Το κείμενο με το σύστημα στοιχειοθεσίας LaTeX, διανομής MikTeX 2.8, στο περιβάλλον TEXnicCenter 1.0, Stable Release Candidate 1. Ανεκτίμητη ήταν η χρήση του Mozilla Firefox, στην ασφαλή περιήγηση των αρχείων κώδικα!

Α΄.1.2 Διαδικαστικά

Η διαδικασία της σύνθεσης αποκάλυψε και διόρθωσε λάθη και παραλείψεις, οφειλόμενα κυρίως σε άγνοια και απειρία. Λόγω διατύπωσης, μπορεί κάποιες δομές να μην αναγνωρίζονται, άλλες να εμφανίζονται, αλλά και κάποιες να περισσεύουν.

Αναλυτικότερα, όπως προαναφέρθηκε στην ενότητα 6.2, η μετάφραση ενός συστήματος αναπτυγμένου σε γλώσσα περιγραφής υλικού περιορίζει το εκφραστικό εύρος της γλώσσας, που περιλαμβάνει δομές ακατάληπτες για το υλικό, βασιζόμενες κυρίως σε πρότυπα γλωσσών λογισμικού. Ο πρακτικός κανόνας, για να είναι μία δομή αποδεκτή και από την σύνθεση, είναι να αντιστοιχίζεται άμεσα, νοητά, σε κάποιο γνωστό κυκλωματικό στοιχείο. Επίσης, δεν είναι δυνατή η απόδοση αρχικών τιμών σε σήματα κατά τον ορισμό τους, όπως η γλώσσα επιτρέπει, αλλά πρέπει να γίνεται με απτό κυκλωματικό τρόπο.

[2] σε συσκευασία ff668, και ταχύτητα -12, συνοπτικά xc4vsx25-12ff668, σειρά που διακόπτεται [13].

Αντιστρόφως, από ατυχείς περιγραφές μπορούν να προκύψουν δομές που δεν ορίστηκαν. Από την χρήση, και όχι την πρόθεση ή την ονοματολογία, κάθε σήμα με σημαίνουσες μεταβάσεις (ακμές), αναγνωρίζεται ως ρολόι. Τέτοιες λειτουργίες, εκτός των καθορισμένων πραγματικών ρολογιών του κυκλώματος, πρέπει να αποφεύγονται. Αν κριθεί απαραίτητο, μεταβάσεις μπορούν να ανιχνευθούν συγκρίνοντας τις εξόδους δύο καταχωρητών σε σειρά, απαλείφοντας την παρακολούθηση των ακμών. Οι συνοπτικές εκφράσεις των λογικών συνθηκών, μπορεί είναι αποδεκτές από την γλώσσα υλικού, αλλά αν δεν είναι και πραγματικά πλήρεις, στο υλικό παράγουν απρόβλεπτους, και εν γένει ανεπιθύμητους, μανδαλωτές (latches). Με επαναδιατύπωση των συνθηκών, βάσει λογικής ελαχιστοποίησης και όχι σημασιολογικής, εύκολα εξαλείφονται.

Σήματα που έχουν αφεθεί ασύνδετα, αποκόπτονται από τις διαδικασίες βελτιστοποίησης. Τέτοια ήταν όλα τα σήματα εξόδου των θυρών A των δίπορτων μνημών του συστήματος. Έχοντάς τους ανατεθεί οι εγγραφές στα στάδια st3, st4, ποτέ δεν διαβάζονταν, και παρέμεναν ασύνδετα. Αυτό, αφέθηκε τελείως στην ευχέρεια των εργαλείων, όπως και οι προειδοποιήσεις για αναγνώριση των μνημών σαν 'μαύρα κουτιά'. Δεν αγνοήθηκαν τα μηνύματα για την αδράνεια, λόγω πρόσθεσης τιμής 2, και επερχόμενη αποκοπή των τελευταίων bits των μετρητών της αρχικοποίησης, καταλήγοντας στην μέθοδο μέτρησης της ενότητας 4.3.

Εκτός των διορθώσεων της αλγοριθμικής έκφρασης του συστήματος με σκοπό την συνθεσιμότητά του, δεν έγιναν παρεμβάσεις ελέγχου ή βελτιστοποίησης της απεικόνισης στο υλικό. Κανένας επιπρόσθετος περιορισμός δεν καταγράφηκε στο αρχείο .ucf, (user constraints file), και διατηρήθηκαν όλες οι αρχικές ρυθμίσεις του εργαλείου, ακόμα και το παρωχημένο stepping level (major silicon revision) 1, αντί για 2. Με το μοντέλο για προσομοίωση που δημιουργήθηκε μετά την τοποθέτηση και δρομολόγηση (post Place and Route simulation model), χρησιμοποιήθηκαν οι τιμές καθυστερήσεων που αυτομάτως εκτιμήθηκαν και αποδόθηκαν στο αρχείο .sdf (standard delay format).

Α΄.1.3 FL_stripped

Η λογική του μονοπατιού δεν κατανεμήθηκε σε υποσυστήματα (VHDL components), με εξαίρεση τους καταχωρητές και τις μνήμες, αλλά αναπτύχθηκε σε ένα ενιαίο κομμάτι κώδικα. Η περιορισμένη έκτασή του το επέτρεπε, αλλά και η από κοινού χρήση κάθε μνήμης, εκτός της DataMem, από δύο στάδια, το επέβαλλε, για την αποφυγή άσκοπου όγκου συνδέσεων. Η περιορισμένη ευελιξία στην αποσφαλμάτωση αντισταθμίστηκε, αναπτύσσοντας την FreeList σε μία απογυμνωμένη (stripped) εκδοχή του μονοπατιού, διατηρώντας την διαίρεση σε στάδια ομοχειρίας, αλλά παραλείποντας τις υπόλοιπες μνήμες. Για αποκατάσταση αυτών των ελλείψεων, οι έξοδοι currentHead, currentTail των μνημών Head και Tail γίναν είσοδοι, όπως και έξοδος έγινε η newTail, που προκύπτει από τη FL. Στην μορφή αυτή έγινε σύνθεση και προέκυψε, όπως είναι αναμενόμενο, η κυρίαρχη επίδραση της λογικής ελέγχου της FL στο σύ-

report data- FL_stripped	EEE	EME	MEM	MMM
Selected Device	4vsx25ff668-12	4vsx25ff668-12	4vsx25ff668-12	4vsx25ff668-12
Number of Slices	133 /10240 (1%)	198 /10240 (1%)	135 /10240 (1%)	200 /10240 (1%)
Number of Slice Flip Flops	141 /20480 (0%)	218 /20480 (1%)	144 /20480 (0%)	221 /20480 (1%)
Number of 4 input LUTs	185 /20480 (0%)	267 /20480 (1%)	185 /20480 (0%)	267 /20480 (1%)
Number of IOs	62	86	64	88
Number of bonded IOBs	62 /320 (19%)	86 /320 (26%)	64 /320 (20%)	88 /320 (27%)
Number of GCLKs	1 /32 (3%)	1 /32 (3%)	1 /32 (3%)	1 /32 (3%)
Speed Grade	-12	-12	-12	-12
Min period	3.484ns	4.654ns	3.484ns	4.654ns
Max Frequency	287.010MHz	214.876MHz	287.010MHz	214.876MHz
Min input arrival time before clk	4.405ns	4.716ns	4.405ns	4.716ns
Max output required time after clk	4.305ns	4.464ns	4.305ns	4.464ns
Max combinational path delay	1.273ns	1.495ns	1.281ns	1.504ns

Σχήμα Α′.1: Μετρήσεις σύνθεσης FL_stripped.

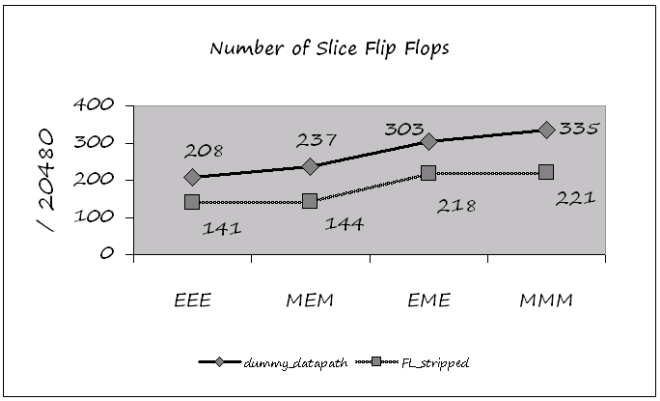
στημα, (67% των χρησιμοποιούμενων flip-flops και 41% των LUTs), μειώσεις σε υλικό και αυξήσεις σε επιδόσεις, λόγω της περιορισμένης έκτασης, αν και σημαντικές μόνο στις εκδοχές ελαχίστης διαστάσης βάθους μνήμης. (Σχήματα Α′.1, και Α′.2 έως Α′.4).

Α′.2 Silver έλεγχοι (testing)

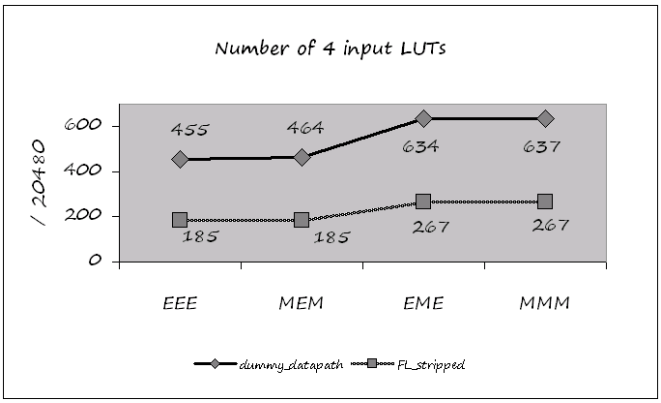
Χωρίς να αποτελούν ολοκληρωμένη λύση όπως ο Tester, οι ενδιάμεσοι έλεγχοι προοδευτικά αποκτούσαν χαρακτηριστικά αυτοματοποίησης. Μιά πλήρως αυτοματοποιημένη προσέγγιση, σε καμία περίπτωση δεν είναι απορριπτέα για τα πρώιμα στάδια, αλλά προϋποθέτει ενδελεχή γνώση των μεθοδολογιών επαλήθευσης, ιδίως των νεοεισαγόμενων συγκερασμών, και διαφοροποίηση της σχεδιαστικής ροής. Επόμενη των θεμελιωδών προσεγγίσεων, ημι-χειρωνακτικής απόδοσης εισόδων και παρακολούθησης των αποκρίσεων, η μέθοδος του common /silver testing, υπήρξε πρόδρομη μορφή του Tester.

Η αυτοματοποίηση των ελέγχων, τόσο η μερική, όπως εδώ, όπως και η πλήρης παρακάτω, απαιτεί διάφορα στοιχεία. Καταρχάς, την ύπαρξη ενός συστήματος αποδεκτού σε ρόλο μοντέλου αναφοράς. Έπειτα, τη διέγερση και των δύο συστημάτων από κοινή ακολουθία εισόδων και εκτέλεση της προσομοίωσης, και τελικά την παρακολούθηση και σύγκριση των αποτελεσμάτων, με την ελάχιστη δυνατή παρέμβαση.

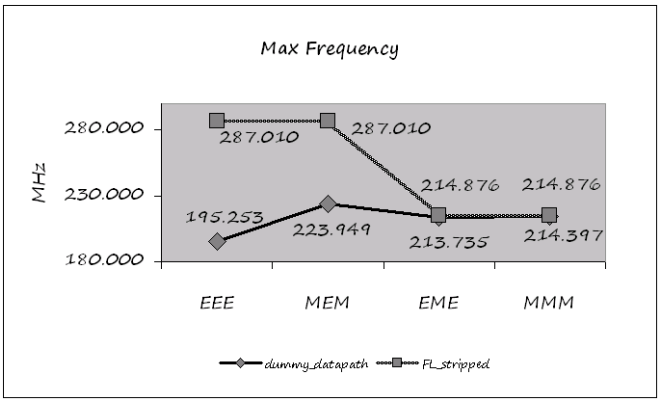
Σίγουρα όχι αδιαφιλονίκητο, πάντως ως επαρκές (silver) μοντέλο αναφοράς, χρησιμοποιήθηκε η προηγούμενη, κατά περίπτωση, εκδοχή του συστήματος. Η σχετική αρτιότητά τους είχε δοκιμαστεί με βασικούς ελέγχους, απόδοσης τιμών με το χέρι και νοητής επιβεβαίωσης, αλλά ευλόγως εκτενείς, δεδομένων και των συνολικά περιορισμένων χαρακτηριστικών. Σε αντιστοιχία με το εκάστοτε υπό ανάπτυξη χαρακτηριστικό, αποδίδονταν στοχευμένες κλάσεις ακολουθιών εισόδων, καλύπτοντας όλες τις αναμενόμενες εκφάνσεις του.



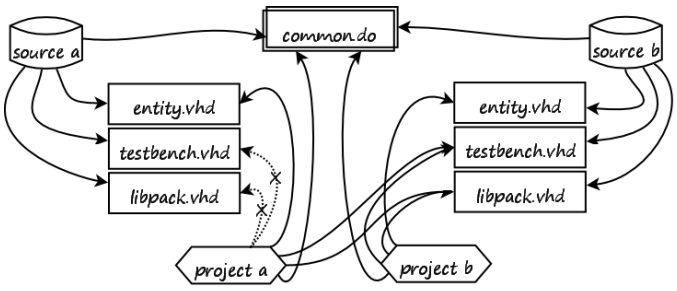
Σχήμα Α΄.2: Σύγκριση χρήσης flip flops.



Σχήμα Α΄.3: Σύγκριση χρήσης LUTs.



Σχήμα Α΄.4: Σύγκριση συχνοτήτων λειτουργίας.



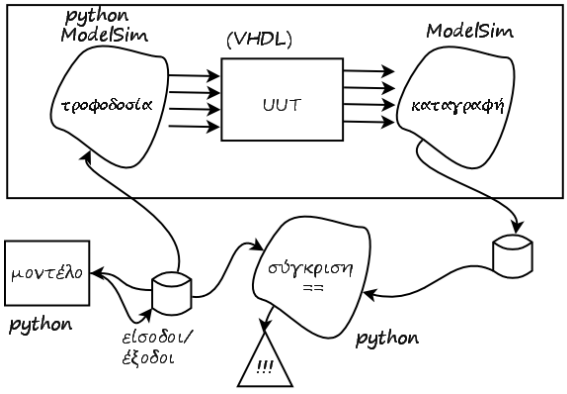
Σχήμα Α΄.5: Από κοινού αναφορά αρχείων στο silver testing.

Η νοητή παρακολούθηση μερικών εκατοντάδων εντολών ήταν εφικτή, λόγω της απλότητας της λειτουργίας του συστήματος, σε συνδυασμό με μία στοιχειώδη κωδικοποίηση των δεδομένων εισόδου (dataIn), που τα συσχέτιζε με τον αριθμό της ουράς (qi) και τον αριθμό εντολής (inID), συν κάποια χρονική πληροφορία. Οι έλεγχοι που εφαρμόζονταν σε αντιπαραβολή, αφορούσαν τα προηγούμενα, ήδη ελεγμένα χαρακτηριστικά, για επιβεβαίωση της διατήρησης της συμπεριφοράς τους στη νεότερη εκδοχή.

Αντιπαραβολή εξόδων νοείται, αν και μόνο αν οι εφαρμοζόμενες ακολουθίες διέγερσης είναι ταυτόσημες. Δεδομένης της ταύτισης των διεπαφών εισόδου και εξόδου, και αξιοποιώντας την δυνατότητα των εργαλείων ένα project να 'βλέπει' αρχεία που δεν βρίσκονται στον φάκελο του αλλά αναφερόμενο προς οποιαδήποτε θέση, και τα δύο αναφέρονταν σε ένα και μοναδικό αρχείο testbench -και κατ΄ επέκταση και στο αντίστοιχο αρχείο βιβλιοθήκης (libpack). Σε κάθε δοκιμή, μεταβάλλονταν με το χέρι και αποθηκεύονταν το ένα κοινό αρχείο, με αποτέλεσμα την παραγωγή και λήψη πανομοιότυπων εισόδων. (Σχήμα Α΄.5).

Το αρχείο απόδοσης των εισόδων δεν ήταν τίποτε άλλο από ένα 'περίβλημα' κώδικα VHDL, δηλαδή μία οντότητα (entity) χωρίς πραγματική λειτουργία, που περιέχει το σύστημα ως συστατικό υποσύστημά του (component), με διαδικασίες (processes) αφιερωμένες στην παραγωγή τιμών. Μετά τις μεταβολές, απαιτούνταν μετάφρασή του, και ενδεχομένως του αρχείου κώδικα του συστήματος. Όλες οι απαιτούμενες ενέργειες αυτοματοποιούνταν από τον εγγενή πλούτο εντολών περιβάλλοντος του ModelSim, την επέκταση της λειτουργικότητας με τη γλώσσα Tcl, και την δυνατότητα εκτέλεσης αρχείων μακροεντολών με περιεχόμενο συνδυασμό τους. ([14], κεφ.14, παρ.Α)

Συγκεκριμένα, τα δύο project που θα συγκρίνονταν, (σχήμα 6.16), περιελάμβαναν στο φάκελό τους, για ευχέρεια κλήσης, από ένα (όμοιο) αρχείο μακροεντολών του ModelSim, επέκτασης .do. Το μείγμα εντολών περιβάλλοντος και κώδικα Tcl, προκαλούσε τη μετάφραση των οσωνδήποτε μεταβληθέντων αρχείων κώδικα, την διαδοχική ενεργοποίηση των δύο project, εκτέλεση προσομοίωσης και καταγραφή σε αρχείο των τιμών των σημάτων dataOut και done, από το καθένα. Οι τιμές λαμβάνονταν από την προβολή List, σε μορφο-



Σχήμα Α΄.6: Εξειδικευμένη δομή συστήματος ελέγχου.

ποίηση tssi, ενώ μεταβλητή που δίνονταν ως είσοδος πριν την εκτέλεση, γινόταν συνθετικό των ονομάτων για την διάκριση των αρχείων μεταξύ εκτελέσεων. Η σύγκριση τους δεν περιλαμβάνονταν σε κώδικα Tcl στο .do, αλλά καλούνταν η σύγκριση αρχείων (fc) στην γραμμή εντολών των Windows (cmd).

Α΄.3 Δομή (Auto)Tester

Η στόχευση της silver μεθόδου ήταν η ταχεία εποπτεία της διατήρησης της ήδη εγκεκριμένης συμπεριφοράς, επί συγκεκριμένου συνόλου τιμών ελέγχου. Για τον Tester, από την άλλη, στόχος ήταν η διενέργεια αξιόπιστων ελέγχων επί εκτεταμένων συνόλων τιμών. Η μετατόπιση του ενδιαφέροντος, από το χρόνο στο χώρο, μετέβαλλε και τις απαντήσεις στα ζητήματα της αυτοματοποίησης. Το μοντέλο διαφοροποιείται εκφραστικά από το σύστημα υπό έλεγχο, και οι είσοδοι παράγονται δυναμικά, βάσει εισαγωγής επιλογών. Η παραγωγή των εισόδων, η προσομοίωση του συστήματος και του μοντέλου, και η σύγκριση εκτελούνται τελείως ενοποιημένα, και χωρίς παρέμβαση του χρήστη, πέραν της παραμετροποίησης.

Η γενική δομή που παρουσιάστηκε στο σχήμα 6.1, από πλευράς εργαλείων και έκφρασης, σε συγκεκριμένες γλώσσες, εξειδικεύεται στο σχήμα Α΄.6. Το σύστημα έχει αναπτυχθεί σε γλώσσα VHDL, η παραγωγή των εισόδων του συστήματος, η διατύπωση κι εκτέλεση του μοντέλου και οι συγκρίσεις των αποτελεσμάτων έγιναν σε γλώσσα python. Η τροφοδότηση των εισόδων στο σύστημα και η καταγραφή των εξόδων του έχει βασιστεί σε δυνατότητες του εργαλείου ModelSim της Mentor Graghics.

Η σημαντικότερη διαφορά από την silver μέθοδο, πέραν της μοντελοποίησης, είναι στην παραγωγή και κατανάλωση των εισόδων, τόσο από το σύστημα, όσο και το μοντέλο. Από τη μία, η διαφοροποίηση του μοντέλου, απαιτεί και προσαρμογή των εισόδων προς αυτό. Από την άλλη, η δυναμική παραγωγή εισόδων, μεταβάλλει την μέθοδο τροφοδότησης του ίδιου του συστήματος. Ε-

πιπλέον, πρέπει να διασφαλιστεί η ταυτότητα των συνόλων που εισάγονται στα δύο εκτελούμενα μέρη.

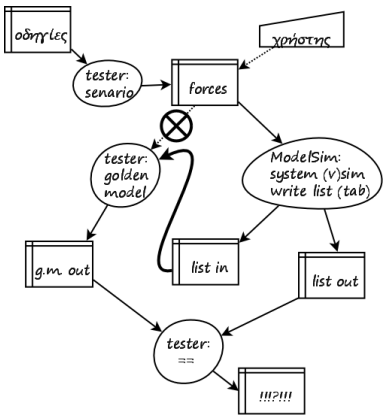
Εκτός από την προαναφερθείσα μέθοδο, μέσω μίας 'περιβάλλουσσας' οντότητας σε VHDL, υπάρχει δυνατότητα παραγωγής και απόδοσης τιμών στα σήματα άμεσα από το εργαλείο προσομοίωσης, με εντολές του περιβάλλοντος. Συγκεκριμένα η "force", σε κάθε γραμμή, αποδίδει, τουλάχιστον μία, τιμή σε κάποιο σήμα, αναφορικά προς αντίστοιχα χρονικά σημεία, σχετικά ή απόλυτα. Οι εντολές που επιρρεάζουν διαφορετικά σήματα, αλλά ακόμα και το ίδιο σε διαφορετικούς χρόνους, μπορούν να διαταχθούν ανεξάρτητα από τις υπόλοιπες. Έτσι, θεωρήθηκε καταλληλότερη για αυτοματοποιημένη σύνθεση προτάσεων, έναντι της VHDL. Το σύνολο των εισόδων, διατυπωμένων σε εντολές force, τροφοδοτείται στο σύστημα από το εργαλείο, κατά την εκτέλεση, μέσω αρχείου μακροεντολών .do.

Οι εντολές force είναι εκφραστικότατες και απλές στην παραγωγή τους, αλλά η μορφή τους δεν είναι η καταλληλότερη και για το μοντέλο, καθώς θα απαιτούνταν κάποια σχετικά πολυπλοκότερη συντακτική ανάλυση για την ανασύνθεση των εντεταλμένων τιμών. Ιδανικά, το μοντέλο θα έβλεπε τιμές που να αντιπροσωπεύουν μία εντολή, χωρίς κάποια χρονική πληροφορία, που δεν το αφορά. Ομοίως, οι έξοδοί του νοούνται ανά εντολή και όχι σε σχέση με ρολόι, που δεν υπάρχει στο μοντέλο[3]. Άρα, τόσο για τις εισόδους όσο και για τις εξόδους του μοντέλου, η απλούστερη μορφή θα ήταν κάποιος πίνακας, με τις στήλες να διαχωρίζονται με συμφωνημένους χαρακτήρες.

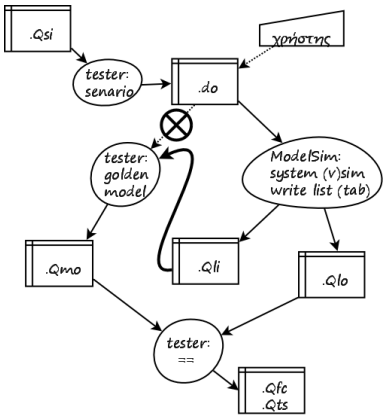
Εκτός από την tssi διατύπωση, η προβολή τιμών (View) List του ModelSim, διαθέτει και άλλες δύο, γεγονότων (events), και πίνακα (tabular). Παρότι η μορφή εξόδων του μοντέλου είναι απολύτως αυθαίρετη, η επιθυμητή μορφή του πίνακα ακολουθήθηκε για την καταγραφή των εξόδων του συστήματος, μέσω της tabular List. Μάλιστα, μέσω List υπάρχει δυνατότητα προβολής και καταγραφής οποιουδήποτε σήματος, άρα και των εισόδων. Ικανοποιούνται, λοιπόν, δύο απαιτήσεις. Αφενός, να δίνονται οι είσοδοι στο μοντέλο, όσο το δυνατόν απλούστερα, ως πίνακας, χωρίς την επισφάλεια της πολυπλοκότητας της μετάφρασης των force, και κυριότερα, να είναι απολύτως ταυτόσημες με τις εισόδους του συστήματος. Πράγματι, πρόκειται για ένα είδος bootstrapping: το σύστημα λαμβάνει τις εισόδους του σε εντολές force, που 'καταλαβαίνει', και είτε αποδόθηκε αυτό που υπονοούσε το σενάριο που προκάλεσε την παραγωγή τους, είτε όχι, το μοντέλο θα λάβει τις τιμές που 'είδε' και εκτέλεσε το σύστημα. (Σχήμα Α΄.7)

Γενικά, οι επιμέρους εργασίες που διενεργεί ο Tester, και έχουν αναπτυχθεί σε αντίστοιχο πλήθος τμημάτων κώδικα (modules) της python, επικοινωνούν μέσω, παραγόμενων από αυτά, αρχείων. Έχουν ήδη αναφερθεί τα (επέκτασης) .do, που, εδώ, χονδρικά περιλαμβάνουν εντολές force και εντολές για καταγραφή των εισόδων και εξόδων από το List. Τα (forces-) .do, μπορούν να δοθούν

[3] υπάρχει όμως μοντελοποίηση του βασικού χρονισμού του συστήματος, ιδίως σε σχέση με τα reset .



Σχήμα Α΄.7: Εξασφάλιση ταυτότητας εισόδων μοντέλου με συστήματος (bootstrapping).



Σχήμα Α΄.8: Bootstrapping με ροή αρχείων.

έτοιμα από το χρήστη, χωρίς τις υπόλοιπες εντολές καταγραφής, οι οποίες και θα συμπληρωθούν, ή να παραχθούν βάσει κάποιου σεναρίου. Γραμμές οδηγιών σεναρίου λαμβάνει ο Tester από αρχεία .Qsi (Qs-Tester senario input). Οι έξοδοι της προσομοίωσης του συστήματος καταγράφονται σε αρχείο .Qlo (Qs-System list output), και οι είσοδοί του σε .Qli (Qs-System/Model list input). Το τελευταίο, θα τροφοδοτηθεί στο μοντέλο, που θα δώσει τις δικές του εξόδους σε ένα .Qmo (Qs model output), και τελικά, τα .Qlo και .Qmo θα συγκριθούν, και τα αποτελέσματα της σύγκρισης θα καταγραφούν σε ένα .Qfc (Qs-Tester file compare). Έτσι, το σχήμα Α΄.7, με ροή των αρχείων, μετατρέπεται στο Α΄.8.

Εκτός της επικοινωνίας, και πέραν του κυρίου αποτελέσματος της σύγκρισης των εξόδων σε .Qfc, επιπλέον αρχεία παράγονται, για καταγραφή ποικίλλων λειτουργικών παρατηρήσεων. Διάφορες χρονικές στιγμές της εκτέλεσης του Tester γράφονται σταδιακά, σε ένα .Qts (Qs-Tester time stamp). Για τα

παραγόμενα .do, που το καθένα τους σηματοδοτεί μία δοκιμή (συνεδρία εκτέλεσης του Tester), καταγράφεται το όνομά τους σε αντιστοιχία με το αρχείο προέλευσης, σε .Qs_migrated, και .Qs_created, αν έχουν δοθεί από χρήστη, συμπληρωθεί και μετεγγραφεί, στην πρώτη περίπτωση, ή δημιουργηθεί βάσει σεναρίου, στην δεύτερη. Κάποιες παράμετροι του Tester, μπορούν να αποθηκευθούν σε αρχεία .pkl_Qd (pickled Qs_dictionary), για να φορτωθούν άμεσα. Εκτός των ιδιαίτερης μορφής (custom) αρχείων, το σύνολο των αρχείων κάθε επιμέρους δοκιμής, (συνοπτικά, Σχήμα Α΄.9), συμπιέζονταν, χειροκίνητα, σε .zip, και οι επισημάνσεις των αρχείων καταγραφής, όπως το είδος προσομοίωσης και οι διάρκειες εκτέλεσης των διάφορων σταδίων, συγκεντρώθηκαν τελικά σε αρχεία .csv (comma separated values). (Η μέθοδος αναλυτικά, ενότητα Α΄.5.3).

Η ροή των αρχείων (σχήμα Α΄.8) προϊδεάζει για την γενική συσχέτιση των modules του Tester (σχήμα Α΄.10). Όλα ενοποιούνται κάτω από ένα συνολικό, το Qs_tester, με το οποίο αλληλεπιδρά ο χρήστης και καλεί όλα τα επιμέρους στάδια. Ο χρήστης εισάγει διάφορες επιλογές, μεταξύ αυτών είναι ενδεχομένως και η φόρτωση αρχείου .pkl_Qd, κάποιου αρχείου γραμμών σεναρίου (.Qsi), ή με εντολές force του χρήστη (.do). Το Qs_dictionary θα αναλάβει την κατασκευή των απαραίτητων δομών μεταβλητών και σταθερών, είτε από default τιμές, είτε με νέες επιλογές. Τα ενδεχομένως δοθέντα αρχεία .do και .Qsi, θα αξιοποιηθούν για την παραγωγή νέων .do από το module Qs_senario, είτε με μετεγγραφή και ενίσχυση καθενός, είτε με σύνθεση κάθε γραμμής σεναρίου που περιέχεται, αντίστοιχα.

Τελικά, το κάθε νέο .do που έχει παραχθεί, στον αντίστοιχο φάκελό του, ορίζει μία νέα δοκιμή, και όλες οι δοκιμές διέρχονται από τα ίδια στάδια. Εκτέλεση της προσομοίωσης του συστήματος, με το Qs_hdl_sim, εκτέλεση του μοντέλου με τις εισόδους του συστήματος, το Qs_manager, και σύγκριση των αρχείων εξόδου, με το Qs_sVm_compare. Κάποια επιπρόσθετα modules συμπεριλαμβάνονται, είτε για βοήθεια στην αποσφαλμάτωση (katerinaPetsa), είτε ως επεκτάσεις της βασικής λειτουργικότητας του Tester (Qs_all_dirs, Qs_values, Qs_tester_embolima).

Α΄.4 Λειτουργία (Auto)Tester

Παρακάτω, τα modules αναλύονται διεξοδικότερα, ανά συνάρτηση (Σχήμα Α-΄.11), αντικαθιστώντας τα παρωχημένα σχόλια των αρχείων του κώδικα.

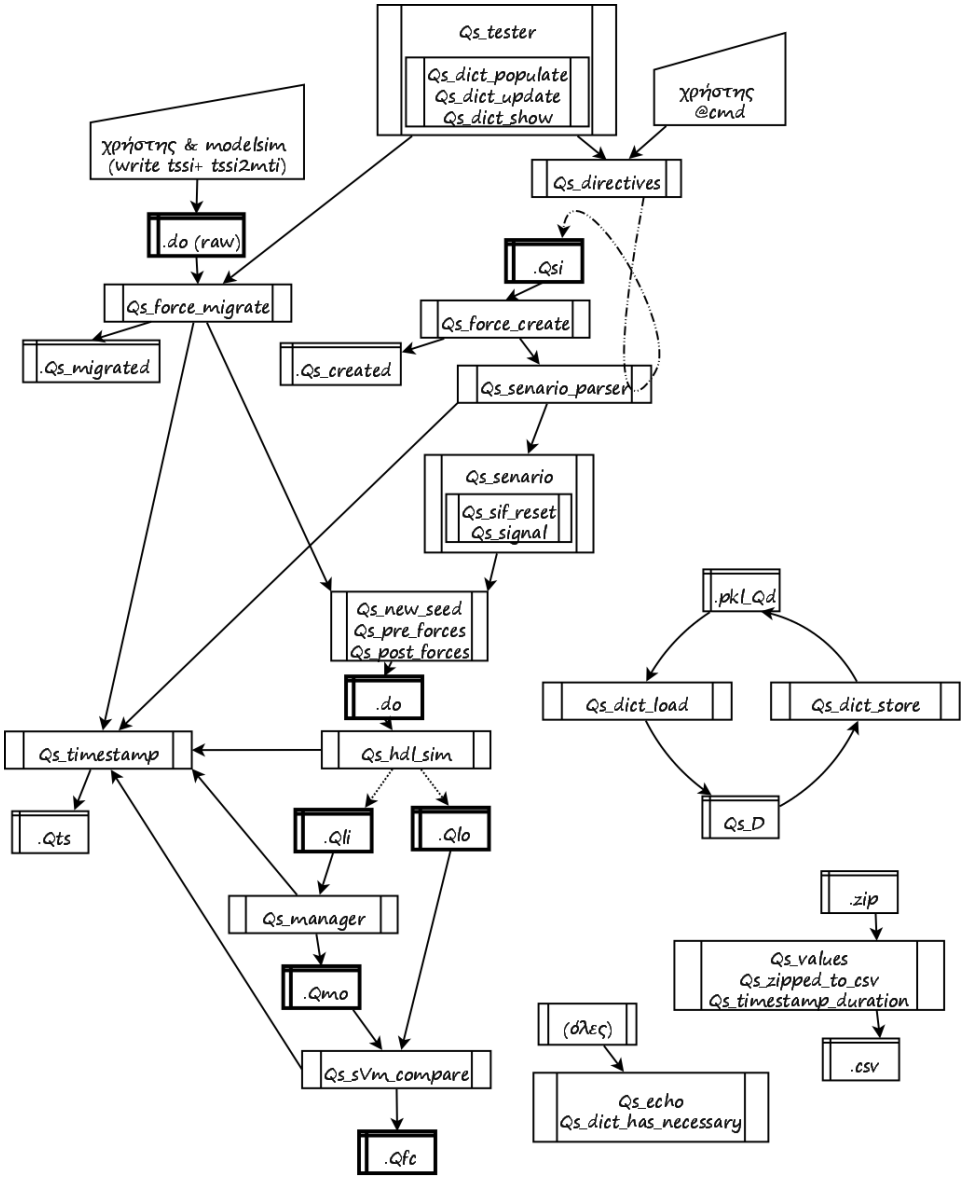
Α΄.4.1 Διεπαφή

Το μοναδικό σημείο πρόσβασης και αλληλεπίδρασης του χρήστη με τον Tester είναι το module Qs_tester, το οποίο καλεί από την γραμμή εντολών (cmd), και κείνο αναλαμβάνει ενοποιημένα τις υπόλοιπες εργασίες. Σε στάδιο ανάπτυξης, βέβαια, είναι θεμιτό και αναγκαίο να υπάρχει και άμεση πρόσβαση σε καθένα module ξεχωριστά.

επέκταση	α / χ	παράγεται από	περιέχει
.do	α / χ	modelsim (testbench.vhd-> write tssi-> tssi2mti), χρήστης (χειρονακτικά), tester (Qs_senario:Qs_senario)	κυρίως εντολές force, για απόδοση τιμών στα σήματα, αλλά και εντολές του περιβάλλοντος modelsim και κώδικα Tcl, για καταγραφή των σημάτων, και τερματισμό.
.Qsi	α (χ)	Qs_senario:Qs_directives (Qs_tester με επιλογή -i)	γραμμές οδηγιών παραγωγής σεναρίου
.Qs_migrated	α	Qs_senario:Qs_force_migrate	μήνυμα σε ποιόν φάκελο δοκιμής μετεγράφηκε (ενισχυμένο) το έτοιμο .do με το αντίστοιχο όνομα
.Qs_created	α	Qs_senario:Qs_force_create	μήνυμα σε ποιόν φάκελο δοκιμής δημιουργήθηκε αρχείο .do, και βάσει ποίας γραμμής οδηγιών σεναρίου, από το αρχείο με το αντίστοιχο όνομα
.pkl_Qd	α	Qs_dictionary:Qs_dict_store	Qs_dictionary, συνήθως με αλλαγμένες κάποιες τιμές, για να αναπαριστά διαφορετική διαστασιολόγηση
.Qli	α	modelsim (write list)	πίνακα με τις τιμές των εισόδων του συστήματος ανά κύκλο (=ανά εντολή)
.Qlo	α	modelsim (write list)	πίνακα με τις τιμές των εξόδων του συστήματος ανά κύκλο (=ανά εντολή)
.Qmo	α	Qs_manager:Qs_manager	γραμμές με τις τιμές των εξόδων του μοντέλου ανά εντολή
.Qfc	α	Qs_sVm_compare: Qs_sVm_compare	μηνύματα των τυχόν λαθών ή ειδοποιήσεων, και του γενικού αποτελέσματος της σύγκρισης των αρχείων εξόδων, συστήματος και μοντέλου (και το είδος της προσομοίωσης)
.Qts	α	Qs_hdl_sim:Qs_hdl_sim, Qs_manager:Qs_manager, Qs_senario:Qs_pre_forces, Qs_senario:Qs_post_forces, Qs_senario:Qs_force_create, Qs_senario:Qs_force_migrate, (Qs_senario:Qs_timestamp), Qs_sVm_compare: Qs_sVm_compare	μηνύματα χαρακτηριστικών της δοκιμής : από ποιο project, είδος προσομοίωσης, αρχικό αρχείο (.Qsi και οδηγία, ή .do), χρονικά σημεία έναρξης και λήξης σταδίων, μετρήσεις ταχύτητας
.zip	χ	-	τον φάκελο με αρχεία εκτέλεσης δοκιμής (.do, .Qli, .Qlo, .Qmo, .Qfc, .Qts) μετά από συμπίεση
.csv	α	Qs_values:Qs_zipped_to_csv	τιμές αποτελεσμάτων (από τα αρχεία .do, .Qfc, .Qts)

αυτόματα/χειρονακτικά

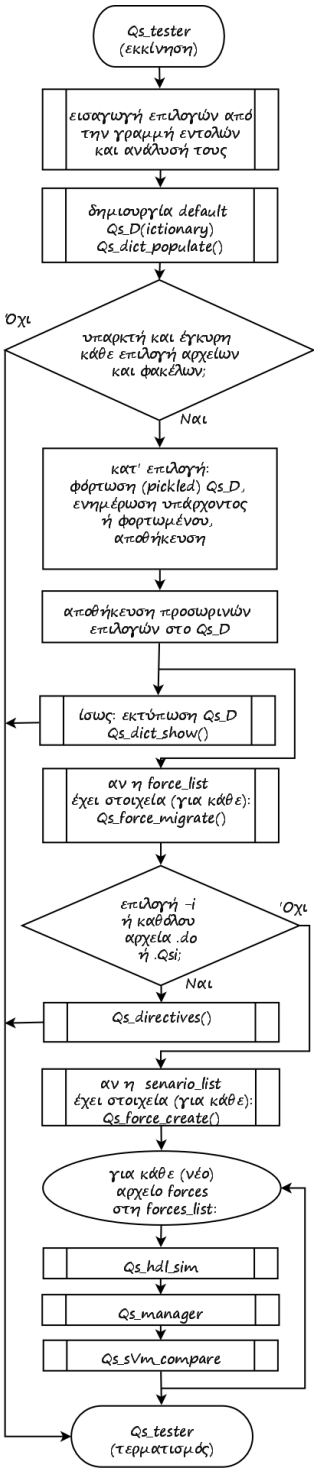
Σχήμα Α΄.9: Παραγόμενα αρχεία από τον Tester.



Σχήμα Α′.10: Ροή αρχείων μέσω συναρτήσεων.

όνομα	χρήση	συναρτήσεις
katerinaPetsa.py	αποσφαλμάτωση	-
Qs_all_dirs.py	συμπληρωματική	Qs_Bf_dir_search, Qs_Df_dir_search, Qs_all_dirs
Qs_values.py	συμπληρωματική	Qs_values, Qs_zipped_to_csv, Qs_timestamp_duration
Qs_tester_embolima.py	διόρθωση/επέκταση	Qs_tester_embolima, Qs_manager_embolima
Qs_dictionary.py	βασικό	Qs_dict_populate, Qs_dict_has_necessary, Qs_dict_update, Qs_dict_show, Qs_dict_load, Qs_dict_store, Qs_echo
Qs_hdl_sim.py	βασικό	Qs_hdl_sim
Qs_manager.py	βασικό	Qs_manager, (Qs_tester_embolima, Qs_manager_embolima)
Qs_senario.py	βασικό	Qs_senario, Qs_sif_reset, Qs_directives, Qs_senario_parser, Qs_pre_forces, Qs_post_forces, Qs_force_create, Qs_force_migrate, Qs_new_seed, Qs_signal, Qs_timestamp
Qs_sVm_compare.py	βασικό	Qs_sVm_compare
Qs_tester.py	βασικό	Qs_tester

Σχήμα Α΄.11: Τα modules και οι συναρτήσεις τους.



Σχήμα Α΄.12: Ροή εκτέλεσης του Qs_tester.

Α΄.4.1.1 katerinaPetsa.py

Το module υπάρχει για λόγους αποσφαλμάτωσης, και χρησιμοποιήθηκε κατά τη διάρκεια της ανάπτυξης στον διερμηνέα IDLE. Δίνοντας μία είσοδο ο χρήστης, για ελάχιστη διαδραστικότητα, ο κατάλογος εργασίας ανακατευθύνεται στο c:\python\bin, συμπεριλαμβάνεται στο sys.path, αν δεν είναι ήδη, και φορτώνονται όλα τα modules του tester, με σκοπό επιμέρους χειροκακτικές κλήσεις των συναρτήσεων.

Α΄.4.1.2 Qs_tester.py

Συνοπτικά, αποτελείται από το σύνολο των επιλογών που ο χρήστης διαθέτει για κλήση από την γραμμή εντολών και παραμετροποίηση της εκτέλεσης, τόσο του μοντέλου, όσο και του αρχικού συστήματος. Επίσης, οι αναγνωσμένες τιμές ελέγχονται και καλούνται όλες οι επιμέρους συναρτήσεις που αυτοματοποιούν την διαδικασία κατασκευής αρχείων εντολών force, εκτέλεσης προσομοιώσεων βάσει αυτών, και σύγκρισης των αρχείων εξόδου.

Αναλυτικότερα, (σχήμα Α΄.12), χρησιμοποιώντας το module βιβλιοθήκης optparse, δημιουργείται ένα αντικείμενο ανάλυσης επιλογών χρήστη τύπου γραμμής εντολών, το οποίο, αφού αρχικοποιηθεί με τις δυνατές επιλογές, αναλύει την γραμμή που εισάγεται από τον χρήστη. Οι επιλογές (σχήμα Α΄.13) έχουν ως εξής:

--ld-pkl / -p <f.pkl_Qd> Εισαγωγή διαδρομής αρχείου .pkl_Qd, pickle του Qs_dictionary, για άμεση φόρτωση διαφορετικών τιμών των default μεταβλητών.(Βλέπε Α΄.4.2).

--new-Qd / -n Επιλογή δημιουργίας Qs_dictionary με επιλογές χρήστη. Default τιμή, False.

--pr-Qd / -d Επιλογή εκτύπωσης του τρέχοντος Qs_dictionary, και τερματισμός. Default τιμή, False.

--ffile / -f <frc.do> Εισαγωγή διαδρομής ενός απλού[4] αρχείου .do.

--fcat / -c <frc_dir> Εισαγωγή διαδρομής φακέλου, και φόρτωση όσων απλών αρχείων .do περιέχει.

--senario / -s <s.Qsi> Εισαγωγή διαδρομής αρχείου γραμμών οδηγιών σεναρίου (.Qsi). Η κάθε γραμμή του θα παράγει ένα νέο forces.do, και την αντίστοιχη δοκιμή του.

--scat / -k <snr_dir> Εισαγωγή διαδρομής φακέλου, και φόρτωση όσων αρχείων γραμμών οδηγιών σεναρίου περιέχει.

[4]με εντολές force, αλλά όχι και καταγραφής των εξόδων.

επιλογή		μτβλ.	τύπος	dflt	πεδίο τιμών / περιορισμοί
ld-pkl	p	l_pkl	string	-	έγκυρη διαδρομή και επέκταση αρχείου (.pkl_Qd)
new-Qd	n	n_Qd	boolean	False	True/False
pr-Qd	d	p_Qd	boolean	False	True/False
ffile	f	ffile	string	-	έγκυρη διαδρομή και επέκταση αρχείου (.do)
fcat	c	fcat	string	-	έγκυρος φάκελος με αρχεία <force>.do
senario	s	sfile	string	-	έγκυρη διαδρομή και επέκταση αρχείου (.Qsi)
scat	k	scat	string	-	έγκυρος φάκελος με αρχεία <senario>.Qsi
inter	i	inter	boolean	False	True/False
Behav-Sim	B	b_or_t	int	0	{0,1}
echo-off	q	echo	int	0	{0,1}
shutdown	x	shtdn	boolean	False	True/False

Σχήμα Α′.13: Επιλογές χρήστη στον Qs_tester.

--inter / -i Έναρξη διαδραστικής σύνθεσης μορφοποιημένων γραμμών οδηγιών σεναρίου σε αρχείο .Qsi, και τερματισμός. Default τιμή, False.

--Behav-Sim / -B Επιλογή εκτέλεσης Behavioral προσομοίωσης. Default τιμή, Timing/gate-level.

--echo-off / -q Επιλογή σίγασης μηνυμάτων. Default τιμή, ενεργά.

--shutdown / -x Επιλογή τερματισμού λειτουργίας του υπολογιστή μετά την ολοκλήρωση της εκτέλεσης, για συστήματα μεWindows. Default τιμή, False.

Οι επιλογές συνδυάζονται όλες μεταξύ τους, σε επίπεδο ορισμού τουλάχιστον, καμία δεν αποκλείει την άλλη. Ειδικότερα, οι επιλογές -f, -c, -s, και -k, που αφορούν την φόρτωση αρχείων .do και .Qsi, δρούν προσθετικά, όταν δοθούν μαζί, σε οποιοδήποτε συνδυασμό. Οι -f και -s, δεν είναι επαναλαμβανόμενες, μπορούν να δοθούν μία φορά, και για την εκάστοτε κλήση του Tester, πολλαπλά αρχεία, κάθε είδους, παρέχονται από τις αντίστοιχες εντολές φόρτωσης από φάκελο, -s, -k. Όσα απλά αρχεία .do δίνονται, είτε με -f είτε με -s, θα φορτωθούν στην ίδια λίστα (force_list) για να ενισχυθούν, και ομοίως, όσα .Qsi, είτε με -s είτε με -k, στην ίδια λίστα των αρχείων σεναρίου (senario_list), για να παράγουν τα forces.do που περιγράφουν οι γραμμές οδηγιών τους. Βέβαια, κάποιοι συνδυασμοί δεν έχουν νόημα, ιδίως μαζί με τα -d και -i που προκαλούν τερματισμό, αλλά και πάλι, είναι νόμιμη η διατύπωση.

Εκφραστικά, η ποικιλομορφία προέρχεται περισσότερο από τα αρχεία (απλά .do, .Qsi, .pkl_Qd) που θα φορτωθούν και το είδος της προσομοίωσης που θα επιλεγεί, παρά τους συνδυασμούς των επιλογών. Συνηθέστεροι συνδυασμοί επιλογών είναι, ο

C:\python_bin>Qs_tester.py -n -d

για την παραγωγή των κατάλληλων αρχείων .pkl_Qd [5], για παραμετροποίηση του μοντέλου, αλλά και των συναρτήσεων παραγωγής forces, ο

C:\python_bin>Qs_tester.py -i

με ή και χωρίς την επιλογή, για κατασκευή αρχείων .Qsi, και ο

C:\python_bin>Qs_tester.py -s <s.Qsi> -p <p.pkl_Qd> -B -x

για την εκτέλεση προσομοιώσεων, βάσει μίας ή περισσότερων γραμμών οδηγιών σεναρίου, σε κάποια εκδοχή του συστήματος με διαστάσεις εκείνες που έχουν αποθηκευτεί στο pickle, με την επιλογή -B για Behavioral (αλλά ίσως και όχι, για Timing) προσομοίωση, και απενεργοποίηση του υπολογιστή μετά την ολοκλήρωση των εκτελέσεων. Οι επιλογές -f, -s χρησιμοποιήθηκαν κυρίως κατά την ανάπτυξη, πριν ολοκληρωθεί το Qs_senario.py, αλλά διατηρούνται, δίνοντας την δυνατότητα παροχής οποιασδήποτε επιθυμητής ακολουθίας εισόδων.

Στην συνέχεια, από τις δοθείσες, ελέγχονται οι επιλογές που δίνουν αρχεία και φακέλους, για το αν είναι υπαρκτά και του τύπου που αναλογεί σε κάθε επιλογή, αλλιώς εκτυπώνεται ένα μήνυμα και τερματίζει. Ειδικότερα για τις επιλογές φακέλων, θα τερματίσει ακόμα και αν ο φάκελος που δόθηκε υπάρχει αλλά δεν περιέχει κανένα κατάλληλο αρχείο.

Αν έχει ζητηθεί η φόρτωση του Qs_D(ictionary) (Α΄.4.2) από pickle, φορτώνεται, αλλιώς διατηρείται η ήδη κατασκευασμένη default εκδοχή. Το φορτωμένο, ενισχύεται με τρέχουσες τιμές για τα seed_name και echo, όπως και μία ένδειξη από πού προήλθαν οι τιμές του, για εκτύπωση μηνύματος παρακάτω. Ανεξάρτητα από το αν φορτώθηκε ή πρόκειται για default εκδοχή του Qs_D, αν έχει ζητηθεί (-n), μπορεί να ενημερωθούν μερικές τιμές επιτόπου από το χρήστη. Μετά από αυτό, τίθεται ερώτημα για την αποθήκευση του νέου Qs_D, στο pickling_path. Οι τιμές του στα κλειδιά force_list, echo, senario_list, b_or_t, δεν είναι μεταβλητές ως προς το Qs_D, αλλά αφορούν κάθε φορά την ενεργό κλήση, και αποθηκεύονται στο Qs_D μετά από κάθε παρέμβαση στο περιεχόμενό του, βάσει των επιλογών που έχουν δοθεί. Επίσης, δεν συμπεριλαμβάνονται στο .pkl_Qd οι νέες τιμές, αλλά οι default τους, που για τις force_list και senario_list είναι η κενή λίστα.

Αν έχει ζητηθεί η εκτύπωση του Qs_D, μετά από το σύνολο των παραπάνω παρεμβάσεων στα περιεχόμενά του, θα προβληθεί στην οθόνη, συμπεριλαμβανομένου και του τρέχοντος force_list, και τερματισμός.

Αλλιώς, αν υπάρχουν αιτήσεις φόρτωσης ήδη υπαρχόντων απλών .do, οι διαδρομές των οποίων θα βρίσκονται ήδη στην forces_list, καλείται για το

[5] το -d για να το εκτυπώνει και να τερματίζει, αντί να συνεχίζει και σε default κλήση του -i, ελλείψει αρχείων για παραγωγή εισόδων.

καθένα τους η Qs_force_migrate, που τα αντιγράφει, συμπληρωμένα, σε κατάλληλους φακέλους, και αντικαθιστά τις διαδρομές τους στην λίστα με των νέων.

Αν δεν υπάρχουν καθόλου αιτήσεις φόρτωσης, συνολικά για .do ή .Qsi, ή έχει δοθεί η επιλογή -i, καλείται η Qs_directives, για την κατασκευή και αποθήκευση νέων γραμμών οδηγιών σε αρχείο .Qsi, και τερματισμός.

Στην συνέχεια, αν υπήρξαν αιτήσεις κατασκευής αρχείων .do από σενάριο, καλείται για κάθε .Qsi στην senario_list η Qs_force_create, που κατασκευάζει ένα αρχείο .do για κάθε γραμμή οδηγιών του .Qsi, μέσα σε κατάλληλο φάκελο, καταχωρώντας τη διαδρομή κάθε νέου .do στην forces_list, η οποία τελικά, αποθηκεύει όλα τα αρχεία .do, που είναι έτοιμα, όπως και αν προέκυψαν, για εκτέλεση.

Πράγματι, τυπώνεται στην οθόνη το είδος της προσομοίωσης που έχει επιλεγεί και το σύνολο των περιεχομένων του Qs_D, και για κάθε αρχείο στην force_list καλούνται οι Qs_hdl_sim, Qs_manager, και Qs_sVm_compare. Μετά από μία ακόμα εκτύπωση των περιεχομένων του Qs_D, και αν αυτό έχει ζητηθεί, ο υπολογιστής απενεργοποιείται. (Σχήμα Α′.12)

Α′.4.2 Δεδομένα

Το Qs_dictionary εμπεριέχει, σε μία ενιαία δομή λεξικού (python dict), όλες τις απαραίτητες μεταβλητές και σταθερές που αφορούν μία συνεδρία του Tester. Από εκεί έχουν πρόσβαση οι συναρτήσεις στο απαιτούμενο υποσύνολο μεταβλητών, χωρίς το πέρασμα άλλων ορισμάτων εκτός από αυτό, επιτρέποντας και την έμμεση επικοινωνία τους. Ακόμα, είναι πιο κομψό σε περίπτωση αλλαγών και εισαγωγής νέων παραμέτρων.

Οι επιμέρους συναρτήσεις που αποτελούν το module Qs_dictionary είναι οι:

Qs_dict_populate(D, updated) Η συνάρτηση λαμβάνει δύο ορίσματα, το D, όπου αναμένεται ένα αντικείμενο κλάσης dict, και το updated, που αναμένεται ένας ακέραιος. Το updated σηματοδοτεί, για τιμή διαφορετική του 1, το να καλείται η συνάρτηση για την κατασκευή της default εκδοχής, και για 1, να καλείται μετά από επιλογή για ανανέωση των τιμών των στοιχείων εκείνων που έχουν χαρακτηριστεί μεταβλητά, συμπληρώνοντας τα σταθερά. Η ανανέωση γίνεται με την σειρά κλήσεων: Qs_tester, Qs_dict_update(D, metablhtes), Qs_dict_populate(D, 1). Η επανάκληση της Qs_dict_populate, δεν είναι σε πρώτη όψη απαραίτητη, εξασφαλίζει όμως, ακόμα και αν έχει φορτωθεί ένα παρωχημένο pickle, εκτός από τις μεταβλητές, το να υπάρχουν και όλες οι σταθερές του Qs_D, όπως ορίζονται στην τρέχουσα κατάσταση του Tester. Έτσι, αν ισχύει ότι είναι εκτός ανανέωσης, αρχικά κατασκευάζεται το κομμάτι των μεταβλητών, και διατηρούνται τα κλειδιά τους, για περαιτέρω χρήση, και στην συνέχεια, σε κάθε περίπτωση, το dictionary ενισχύεται με τις

σταθερές, που δίνονται τμηματικά και εκείνες, αλλά για λόγους αλληλοεξαρτήσεων τους. Η συνάρτηση επιστρέφει το dictionary και την λίστα των μεταβλητών κλειδιών.

Qs_dict_has_necessary(Qs_D, caller, nl) Στα ορίσματα της συνάρτησης, εκτός από το βασικό, το dictionary Qs_D, συμπεριλαμβάνεται το όνομα της καλούσας συνάρτησης, caller, και μια λίστα με τα κλειδιά των στοιχείων που η καλούσα χρησιμοποιεί, nl (necessary_list). Εξετάζονται για την ύπαρξή τους τα δοθέντα κλειδιά, αλλά και για κάθε υπάρχον κλειδί, αν έχει τιμή. Σε διαφορετική περίπτωση, τυπώνονται τα ανάλογα μηνύματα, και προσμετράται σε ένα αρνητικό σκόρ, το οποίο ανακοινώνεται στο τέλος, αν υπάρχει πρόβλημα. Η συνάρτηση έχει καθαρά επικουρικό ρόλο, αλλά έχει αποδειχτεί εξαιρετικά χρήσιμη στην αποσφαλμάτωση, κατά τις εισαγωγές νέων στοιχείων στο Qs_D.

Qs_dict_update(D, metablhtes) Λαμβάνει, εκτός από το ίδιο το dictionary, μία λίστα με τα κλειδιά του που θεωρούνται μεταβλητές. Ζητώνται από το χρήστη οι νέες τιμές, οι οποίες συγκρίνονται, βάσει του τύπου των παλαιών τιμών, για το αν είναι κατά το αναμενόμενο, οι αριθμοί, το ίδιο, και οι συμβολοσειρές, διαδρομές (paths) στο σύστημα αρχείων. Μπορεί και να διατηρηθεί η τιμή, δίνοντας κενή απάντηση (Enter). Σε περίπτωση λάθους, η ερώτηση επαναλαμβάνεται, μέχρι την σωστή ή κενή απάντηση. Ζητάται επιβεβαίωση, και επίσης προβάλλεται η τιμή που δόθηκε, πριν καταχωρηθεί. Οι συμβολοσειρές αριθμών, πρώτα μετατρέπονται σε int. Τελικά, το dictionary ανανεώνεται, καλώντας την Qs_dict_populate() με όρισμα updated=1, και επιστρέφεται το νέο dictionary που κατασκευάστηκε.

Qs_dict_show(D) Προβάλλει τα περιεχόμενα του dictionary, σε ζεύγη κλειδιού-τιμής, ανά γραμμή, και με τις συμβολοσειρές διευρυμένες στους 60 χαρακτήρες συμπληρώνοντας με κενά, για περισσότερη αναγνωσιμότητα.

Qs_dict_load(path, filename) Τα ορίσματα δίνουν το πλήρες μονοπάτι του αρχείου όπου είναι αποθηκευμένο το dictionary, σε pickle, από το οποίο φορτώνεται και επιστρέφεται.

Qs_dict_store(D) Αποθήκευση του dictionary, αφού μετατραπεί σε pickle. Καθώς είναι δεδομένης μορφής, εμπεριέχει το μονοπάτι όπου θα πρέπει να αποθηκευτεί, pickling_path, και τα συστατικά για την σύνθεση του ονόματος του αρχείου, το seed_name, και το pickling extension. Δεν επιστρέφεται κάποια τιμή.

Qs_echo(Qs_D, str) Εκτύπωση του μηνύματος str, για δύο τύπους Qs_D: για τύπο dict, ανάλογα με την τιμή στο κλειδί “echo”, αλλιώς, για τύπο int, με τιμή 1. Δεν επιστρέφεται κάποια τιμή.

Τα στοιχεία του Qs_D που θεωρούνται μεταβλητές, και μπορούν να λάβουν διαφορετικές από τις default τιμές με κλήση του Qs_dict_update, ουσιαστικά αποκρυσταλλώνουν την πολιτική παραμετροποίησης του Tester. Μερικά είναι συμβολοσειρές, που αφορούν διαδρομές όπου αποθηκεύονται τα αρχεία κώδικα, αλλά και τα παραγόμενα από τον Tester, ενώ τα υπόλοιπα είναι αριθμοί, που αντιστοιχούν στις θεμελιώδεις διαστάσεις του συστήματος (και του μοντέλου). Έτσι, είναι δυνατή η άμεση εκτέλεση δοκιμών, στο ίδιο ή άλλα μηχανήματα, με διαφορετική αρχειοθέτηση από ότι υπονοείται στα default, ή αντίστοιχα, για διάφορες εκδοχές του συστήματος, φορτώνοντας (-p) ένα κατασκευασμένο και αποθηκευμένο (-n) .pkl_Qd. Οι μεταβλητές αυτές, μαζί με τις αρχικές τους (default) τιμές, είναι οι:

test_path (C:\STAGE_TEST\TEST_py) Συμβολοσειρά διαδρομής του σημείου όπου αποθηκεύονται οι παραγόμενοι φάκελοι των δοκιμών.

pickling_path (C:\STAGE_TEST\TEST_py) Συμβολοσειρά διαδρομής του σημείου όπου αποθηκεύεται το παραγόμενο .pkl_Qd. Δεν αφορά και δεν περιορίζει τα σημεία από όπου μπορεί να φορτωθεί (-p). Επίσης, το αρχείο συνήθως μετονομάζεται, διαφορετικά από το αρχικό του βάσει seed_name.

prj_path (C:\STAGE_TEST\katerina\Hta) Συμβολοσειρά διαδρομής του σημείου όπου αποθηκεύονται τα αρχεία κώδικα του συστήματος. Κατά σύμβαση, αναμένεται να περιέχει συγκεκριμένη δομή φακέλων στο εσωτερικό του (βλέπε Α′.4.2.1).

Qs_multitude (16) Ακέραιος, πλήθος ουρών συστήματος.

dataMem_addr_width (8) Ακέραιος, πλάτος διευθύνσεων μνημών στοιχείων, καθορίζει το πλήθος των στοιχείων (βάθος) τους.

dataMem_width (16) Ακέραιος, πλάτος του κομματιού δεδομένων των στοιχείων.

mem_depth (16) Ακέραιος, (αυθαίρετα) διατιθέμενο πλήθος στοιχείων, μικρότερο ή ίσο με το βάθος των μνημών στοιχείων.

Οι σταθερές στο Qs_D, όχι με την έννοια της απαρέγκλιτης αποτίμησης τους, αλλά της δυνατότητας απόδοσης τιμής μόνο αλλάζοντας τον κώδικα του Tester, είναι οι:

custom_arg (None) Για γενικευμένη χρήση. Για παράδειγμα, μεταβίβαση δείκτη ενός ανοιχτού αρχείου, μεταξύ δύο συναρτήσεων (συγκεκριμένα, των Qs_senario, Qs_force_create).

lst_in_ext ("Qli") Ιδιότυπη επέκταση αρχείων με τιμές σημάτων εισόδου, του συστήματος και του μοντέλου. Στο ModelSim, η default επέκταση είναι .lst, και δεν γίνεται διάκριση μεταξύ σημάτων εισόδου και εξόδου.

lst_out_ext (`"Qlo"`) Ιδιότυπη επέκταση αρχείων με τιμές σημάτων εξόδου του συστήματος. (Η default επέκταση,όπως προαναφέρθηκε, είναι .lst).

Qm_out_ext (`"Qmo"`) Ιδιότυπη επέκταση αρχείων εξόδων μοντέλου.

cmp_out_ext (`"Qfc"`) Ιδιότυπη επέκταση αρχείων ευρημάτων σύγκρισης εξόδων συστήματος - μοντέλου.

pickling_ext (`"pkl_Qd"`) Ιδιότυπη επέκταση αρχείων pickle με περιεχόμενο μια εκδοχή του Qs_dictionary.

force_ext (`"do"`) Επέκταση αρχείων μακροεντολών του ModelSim . (Όντας σημαίνουσα, διατηρήθηκε).

migration_ext (`"Qs_migrated"`) Ιδιότυπη επέκταση αρχείων καταγραφής της μετεγγραφής αρχείου force.

senario_ext (`"Qsi"`) Ιδιότυπη επέκταση αρχείων με οδηγίες σεναρίου.

creation_ext (`"Qs_created"`) Ιδιότυπη επέκταση αρχείων καταγραφής της δημιουργίας αρχείου force.

timestamp_ext (`"Qts"`) Ιδιότυπη επέκταση αρχείων καταγραφής χρόνων και λοιπών στοιχείων.

seed_name (`strftime("%d%m%Y-%H%M%S", localtime())`) Ιδιότυπο όνομα φακέλου κάθε δοκιμής, από την χρονική στιγμή κατασκευής του, αλλά και των αντίστοιχων αρχείων που στη συνέχεια την απαρτίζουν. Για παράδειγμα, η συμβολοσειρά '14092001-162537', κωδικοποιεί την 14η Σεπτεμβρίου 2001 και ώρα 16:25:37. (Αναλυτικότερα Α΄.4.7)

echo (`1`) Επιλογή εκτύπωσης σχολίων στη γραμμή εντολών.

senario_list (`[]`) Λίστα αρχείων με σενάρια για δημιουργία αρχείου force. Δεν έχει default τιμή, αλλά όταν ελεγχθεί, με την Qs_dict_has_necessary, θα (πρέπει να) έχει λάβει.

senario_index (`0`) Δείκτης τρέχοντος αρχείου της λίστας αρχείων σεναρίου (senario_list).

senario_options (`""`) Επιλογές από ανάλυση γραμμής σεναρίου (αντικείμενο optparse.Values).

b_or_t (`1`) Επιλογή είδους προσομοίωσης, Behavioral (=0) ή Time (=1).

force_list (`[]`) Λίστα αρχείων force, αρχικά για μετεγγραφή, και τελικά για εκτέλεση, κατόπιν των αναγκαίων μετεγγραφών και παραγωγών.

force_index `( 0)` Δείκτης τρέχοντος αρχείου της λίστας αρχείων force (force_list).

ms_sep `("/")` Χαρακτήρας διαχωρισμού επιπέδων φακέλων στο σύστημα αρχείων κατά το ModelSim.

sys_dir `(os.path.basename(D["prj_path"])+"_sys")` Φάκελος με αρχεία ModelSim project του συστήματος, κατά σύμβαση (Α΄.4.2.1).

par_dir `(os.path.basename(D["prj_path"])+"_par")` Φάκελος με αρχεία κώδικα συστήματος, δομική περιγραφή από Place and Route, κατά σύμβαση (Α΄.4.2.1).

source_dir `(os.path.basename(D["prj_path"])+"_source")` Φάκελος με αρχεία κώδικα συστήματος, λειτουργική περιγραφή, κατά σύμβαση (Α΄.4.2.1).

Behav_arch `("Behavior")` Όνομα αρχιτεκτονικής λειτουργικής περιγραφής του συστήματος.

TimeSim_arch `("Structure")` Όνομα αρχιτεκτονικής δομικής περιγραφής του συστήματος.

TimeSim_tag `("_timesim")` Συνθετικό ονόματος αρχείου κώδικα δομικής περιγραφής, παραγόμενο από λειτουργικής.

system_entity `("dummy_datapath")` Όνομα VHDL οντότητας του συστήματος.

working_library `("work")` Βιβλιοθήκη κώδικα συστήματος, στο ModelSim project.

in_sigs `(["reset", "inid", "valid", "operation", "qi", "datain"])` Ονόματα σημάτων εισόδου.

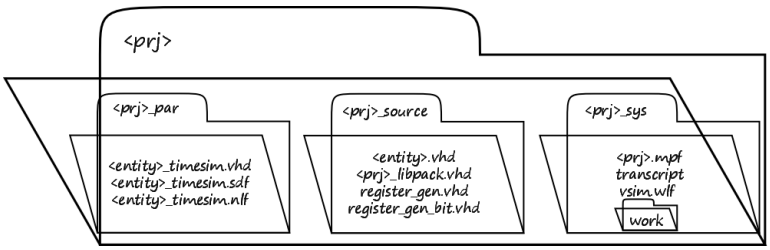
out_sigs `(["initializing", "outid", "done", "dataout", "qout"])` Ονόματα σημάτων εξόδου.

pipeline_depth `(4)` Βάθος ομοχειρίας.

max_positions `(D["mem_depth"] - 1)` Πλήθος (διαθέσιμων) θέσεων στοιχείων που χρειάζονται αρχικοποίηση.

Nm_init_length `(D["mem_depth"]/2 -1)` Διάρκεια αρχικοποίησης μνήμης NextMem, βάσει των αυθαίρετα διαθέσιμων θέσεων, και όχι του πραγματικού βάθους.

HTm_init_length `(D["Qs_multitude"] /2)` Διάρκεια αρχικοποίησης μνημών HeadMem, TailMem.



Σχήμα Α΄.14: Η κατά σύμβαση αρχειοθέτηση.

in_sigs_bits `([1, 12, 1, 1, int(math.log(D["Qs_multitude"], 2)), D["dataMem_width"]])` Πλάτος σε bits των σημάτων εισόδου.

init_max `(max(D["Nm_init_length"], D["HTm_init_length"]))` Μέγιστη διάρκεια αρχικοποίησης.

Α΄.4.2.1 Σύμβαση αρχειοθέτησης

Με σκοπό να είναι το prj_path η μόνη μεταβλητή που θα αλλάζει από το χρήστη, ενώ τα sys_dir, par_dir, source_dir, να προκύπτουν από αυτό, δημιουργείται μία σύμβαση για τη δομή των φακέλων του project: το όνομα του project να είναι μόνο του το όνομα ενός γονικού φακέλου, έστω S, και αυτός να περιέχει τους υποφακέλους S_sys, S_source, S_par με τα αντίστοιχα αρχεία. Ο υποφάκελος S_sys, περιέχει τα μεταγλωττισμένα αρχεία κώδικα και τα αρχεία του ModelSim project, τα οποία η εκτέλεση χρησιμοποιεί. Ο S_source τα αρχεία πηγαίου κώδικα του συστήματος, εκτός από το αρχείο με την δομική (Structural) περιγραφή του, που βρίσκεται στον φάκελο S_par. Οι δύο πρώτοι, προέρχονται από την πρακτική διαχωρισμού των αρχείων πηγαίου κώδικα από εκείνα των project, και κάθε εργαλείο να αναφέρεται σε αυτά, βλέποντας τον ίδιο κώδικα και το ModelSim και το ISE. Ο τρίτος, προκύπτει από την μετάφραση για FPGA του συστήματος, όπως κατασκευάστηκε από το ISE στο <ise_project_path>/netgen/par, αυτούσιος[6]. (Σχήμα Α΄.14)

Α΄.4.3 Είσοδοι

Η λειτουργία του module Qs_senario συνοψίζεται στην παραγωγή των αρχείων <forces>.do, είτε με αντιγραφή και ενίσχυση ενός ήδη υπάρχοντος δοθέντος, είτε με την δημιουργία νέου, βάσει των γραμμών οδηγιών δοσμένων από το χρήστη. Οι επιμέρους συναρτήσεις που το αποτελούν, και οι λειτουργίες τους, είναι οι:

Qs_senario(Qs_D) Δημιουργία εντολών force, βάσει γραμμής οδηγιών σεναρίου, η ανάλυση της οποίας διοχετεύεται μέσω της Qs_D[“senario_-options”], και γράφονται στο ανοιχτό, από την καλούσα Qs_force_create,

[6] μαζί με το αρχείο .sdf που χρησιμοποιείται και το .nlf που δεν χρησιμοποιείται.

αρχείο .do, που δίνεται μέσω της Qs_D["custom_arg"]. Μετά την αρχικοποίηση διαφόρων βοηθητικών μεταβλητών, βάσει των διαθέσιμων δεδομένων που τις αφορούν, συντίθεται καταρχάς το default reset, που παράγεται χωρίς συνθήκη. Στη συνέχεια, και στον εκάστοτε κύκλο ενός μεγάλου βρόγχου, βάσει αύξοντος μετρητή που αναπαριστά τον αριθμό εντολής, ελέγχονται οι επιλογές και αναλόγως παράγεται (ή όχι) με την κατάλληλη τιμή κάποιο force, για καθένα από τα σήματα εισόδου. (Αναλυτικότερα, Α΄.4.3.1.) Ανά ένα σταθερό διάστημα, και στο τέλος από όσο περισσεύει για μη ακέραια πολλαπλάσια, οι εντολές που έχουν παραχθεί, μαζί και με μία εντολή run του προσομοιωτή, καταγράφονται στο αρχείο. Η παραγωγή τους συνεχίζεται μέχρι την εξίσωση του μετρητή με την διάρκεια προσομοίωσης. Δεν επιστρέφεται κάποια τιμή.

Qs_sif_reset(Qs_D) Τα επόμενα reset είναι οι μόνες τιμές σήματος εισόδου που προϋπολογίζονται έξω από την συνάρτηση Qs_senario. Στις γραμμές οδηγιών σεναρίου, η επιλογή τους (-r), είναι επίσης η μόνη δυνητικά επαναλαμβανόμενη, σχηματίζοντας μία λίστα (reset_next) με επιτρεπόμενες τιμές θετικούς ακεραίους, για σταθερά διαστήματα, και το μηδέν, για τυχαίο διάστημα, ελεύθερα συνδυαζόμενες, υπό άλλους περιορισμούς. Η επιστρεφόμενη τιμή της συνάρτησης είναι μία λίστα από tuples, αποτελούμενα από την χρονική στιγμή (σε κύκλους, που ισούται με εντολές) που το σήμα reset αλλάζει τιμή, και την τιμή (0 / 1) την οποία λαμβάνει. Τα χρονικά σημεία των αλλαγών υπολογίζονται βάσει ενός σημείου αναφοράς, που είναι το τέλος της προκαλούμενης αρχικοποίησης από προηγούμενο reset, και το πρώτο σημείο αναφοράς είναι στο πέρας της αρχικοποίησης που προκάλεσε το default reset. Από εκεί, αν έχει δοθεί σταθερή απόσταση, προστίθεται άμεσα, ενώ για επιλογή τυχαίας παραγωγής, η απόσταση επιλέγεται τυχαία, από το ήδη υπολογισμένο εναπομείναν διάστημα. Μετά από αυτή τη διάρκεια αναμονής, είναι το σημείο που το σήμα λαμβάνει την τιμή 1, και από την επιλεγμένη διάρκεια του reset, προκύπτει το επόμενο σημείο, εκεί όπου το σήμα μηδενίζεται, και είναι τα δύο πρώτα στοιχεία από δύο tuples που θα εισαχθούν στη λίστα των επόμενων reset. Από το τελευταίο σημείο, του μηδενισμού, και βάσει της υπολογιζόμενης διάρκειας αρχικοποίησης, προκύπτει το επόμενο σημείο αναφοράς, και η ίδια διαδικασία επαναλαμβάνεται για όλα τα δοθέντα επόμενα reset. (Σχήμα Α΄.15). Επιστρέφεται η λίστα με τα υπολογισμένα tuples, για χρήση από την Qs_senario().

Qs_directives(Qs_D) Διαδραστική προσθήκη νέων γραμμών οδηγιών σε αρχεία .Qsi. Αρχικά, ζητείται να επιλεγεί αν θα αποθηκευθούν σε υπάρχον ή νέο αρχείο, και αναλόγως την επιλογή, ζητούνται επιπλέον διευκρινίσεις για το μονοπάτι και το όνομα του, και ελέγχονται για την ορθότητά τους. Αφού διευκρινιστεί το πλήρες όνομα του, ανοίγεται για επισύναψη των γραμμών στο τέλος, και καλείται η συνάρτηση του parser των επιλο-

γών σεναρίου, με όρισμα -h (help), για προβολή των επιλογών. Έπειτα, ζητείται και εισάγονται από το χρήστη σύμφωνα μορφοποιημένες γραμμές οδηγιών, που ελέγχονται πρώτα για την συντακτική ορθότητά τους, και αποθηκεύονται στο αρχείο, προβάλλοντας και τα ανάλογα μηνύματα, για διορθώσεις και επανάληψη της διαδικασίας, για νέα εισαγωγή, ή παύση. Η ύπαρξη ταυτόσημων γραμμών στο αρχείο δεν ελέγχεται, καθώς είναι συχνά επιθυμητό. Το αρχείο καταχωρείται, αν δεν υπάρχει ήδη, και στη λίστα των αρχείων σεναρίου του Qs_D, με σκοπό την παρακάτω εκτέλεσή τους (παρότι το -i προκαλεί τερματισμό). (Σχήμα Α΄.16)

Qs_senario_parser(Qs_D, line) Συντακτική ανάλυση των επιλογών που παραμετροποιούν την παραγωγή των forces, από τις γραμμές οδηγιών σεναρίου. Με το module βιβλιοθήκης optparse παράγεται ένας συντακτικός αναλυτής (parser), στον οποίο οι επιλογές εισάγονται, παρέχοντας όνομα, τύπο, αρχικές τιμές, και διευκρινίσεις για την λήψη και αποθήκευσή τους. Αντί να λαμβάνει είσοδο από την γραμμή εντολών, μπορεί να τροφοδοτηθεί και άμεσα με λίστα συμβολοσειρών, όπως θα προέκυπτε από διαχωρισμό στους κενούς χαρακτήρες μίας γραμμής οδηγιών. Το module θα εκτελέσει τους βασικούς ελέγχους, σύμφωνα με τον ορισμό κάθε επιλογής, κυρίως για την σύνταξη και τον τύπο. Για αυστηρότερους περιορισμούς, σχετικούς με την εφαρμογή, είναι αναγκαίοι επιπλέον έλεγχοι, οι οποίοι και πραγματοποιούνται. Αν βρεθεί κάποια ασυμφωνία και σε αυτούς, εγείρεται σφάλμα τύπου SystemExit, του ίδιου που παράγει το optparse [7], αλλιώς επιστρέφονται τα αντικείμενα με τις αναγνωρισμένες επιλογές.

Qs_pre_forces(Qs_D) Σύνολο γραμμών για την αρχή των αρχείων .do, αποτελούμενο από τις διάφορες άλλες αναγκαίες εντολές, εκτός των εισόδων, ώστε να δοθεί μοναδικό όρισμα αρχείου μακροεντολών στην εντολή προσομοίωσης vsim. Περιλαμβάνουν εντολές log, για την καταγραφή από το ModelSim των τιμών των σημάτων εισόδου κι εξόδου κατά την προσομοίωση, εντολές ορισμού και αρχικοποίησης μεταβλητών σε Tcl, για παραμετροποίηση των επόμενων εντολών περιβάλλοντος, εντολές για την μέτρηση της ταχύτητας της προσομοίωσης, το περιοδικό force του ρολογιού (clk) και τα forces για την αρχικοποίηση σε μηδέν όλων των σημάτων εισόδου.

Qs_post_forces(Qs_D) Σύνολο γραμμών για το τέλος των αρχείων .do, που περιλαμβάνει εντολές περιβάλλοντος ModelSim και Tcl για τον υπολογισμό της μέσης ταχύτητας εκτέλεσης της προσομοίωσης, μετά το πέρας της, τις εντολές για την ρύθμιση των παραμέτρων καταγραφής των τιμών των σημάτων εισόδου και εξόδου μέσω της προβολής List, αλλά

[7] το εκπέμπει η sys.exit(), η οποία καλείται εμμέσως, με όρισμα 2, που στο Unix σηματοδοτεί συντακτικό σφάλμα στη γραμμή εντολών.

και την καταγραφή καθαυτή, με παράγωγα της τα .Qli και .Qlo, όπως και εντολές για τον τερματισμό των εκτελέσεων του προσομοιωτή και του περιβάλλοντος.

Qs_force_create(Qs_D) Παραγωγή, για κάθε μία από τις γραμμές οδηγιών του τρέχοντος .Qsi αρχείου, τόσο του φακέλου για όλα τα αρχεία της αναμενόμενης δοκιμής, όσο και των αρχείων .do, .Qts, αλλά και .Qs_created. Αγνοώντας τις κενές γραμμές, ή όσες ξεκινούν με δίεση ('#'), που θεωρούνται σχόλια, το αρχείο εισόδου .Qsi διαβάζεται ανά γραμμή. Αρχικά παράγεται ο seed_named φάκελος, ονομασμένος διαφορετικά για κάθε γραμμή της επανάληψης, και στη συνέχεια καταγράφονται τα στοιχεία της προσομοίωσης σε ένα νέο .Qts, στον ίδιο φάκελο. Μετά τη μετάφραση της γραμμής οδηγιών, παράγονται οι αρχικές εντολές (pre), οι είσοδοι (forces), και οι τελικές εντολές (post), και αποθηκεύονται στο αρχείο .do, περιλαμβάνοντας και τη αρχική γραμμή οδηγιών σεναρίου. Η παραγωγή του φακέλου καταγράφεται σε ένα .Qs_created, στο ίδιο μονοπάτι και με ίδιο όνομα με το αρχικό .Qsi. Τελικά, το αρχείο .do καταχωρείται στην forces_list, και καταγράφεται η λήξη της δημιουργίας του .do στο .Qts, όπως είχε προηγηθεί και για την έναρξή της. (Σχήμα Α′.17).

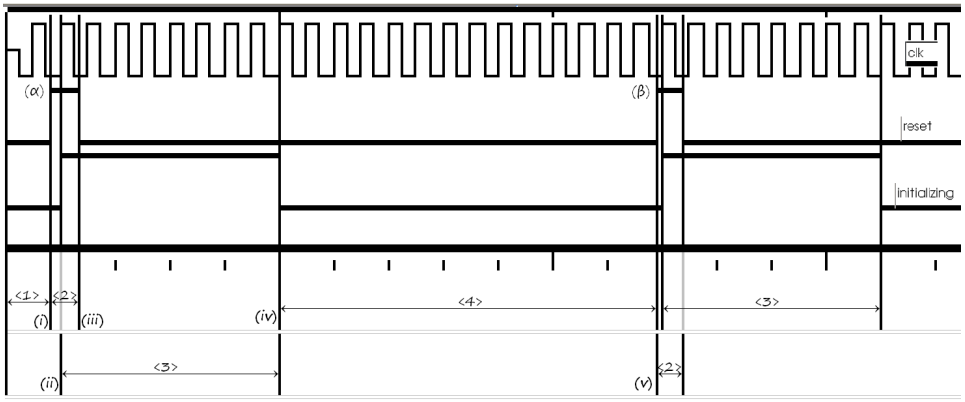
Qs_force_migrate(Qs_D) Παραγωγή, βάσει δοθέντος αρχείου με forces, ενός νέου, ενισχυμένου αρχείου .do, και του φακέλου που θα περιέχει τα αρχεία της συγκεκριμένης δοκιμής. Η τροφοδότηση με προκατασκευασμένα .do, με χρήση της εντολής tssi2mti και υπάρχοντα testbenches, σε αρχικά στάδια ανάπτυξης εξυπηρέτησε την εκτέλεση του Tester, απουσία της αυτόματης παραγωγής εισόδων. Διατηρείται, αντισταθμίζοντας τις όποιες εκφραστικές της αδυναμίες. Έχοντας διαχωρίσει την παραγωγή των αρχικών και τελικών εντολών, και αρχικοποιώντας το senario_options, με μόνη διαφορά την προέλευση των forces, ο χειρισμός είναι ενιαίος με την Qs_force_create(). Και εδώ, αρχικά κατασκευάζεται ο seed-named φάκελος, και δημιουργείται εκεί ένα νέο .Qts όπου καταγράφονται τα στοιχεία της προσεχούς προσομοίωσης και το χρονικό σημείο έναρξης της μετεγγραφής. Αρχικοποιείται το senario_options, με μία κενή γραμμή, για να λάβει τις default τιμές των μεταβλητών που παραμετροποιούν την παραγωγή των εντολών στο Qs_pre_forces. Παράγονται οι αρχικές εντολές (pre), αντιγράφονται οι είσοδοι (forces), παράγονται οι τελικές εντολές (post), και μαζί τα παραπάνω, αποθηκεύονται στο νέο .do, μέσα στο φάκελο. Τελικά, αφού καταγραφεί η μετεγγραφή σε ένα .Qs_migrated αρχείο με ίδιο όνομα και στο ίδιο μονοπάτι με το αρχικό, αποθηκεύεται στη θέση του αρχικού στη forces_list το πλήρες όνομα του νέου .do, και γράφεται στο .Qts το χρονικό σημείο λήξης της μετεγγραφής. (Σχήμα Α′.17).

Qs_new_seed(Qs_D) Παραγωγή νέου seed, με εξασφαλισμένα διαφορε-

τική τιμή από το αρχικό, ή και το κάθε προηγούμενο, ελέγχοντας και ανανεώνοντας την τιμή, σε περίπτωση ισότητας.

Qs_signal(num,length) Μετατροπή του δεκαδικού αριθμού num, είτε σε δυαδικό μήκους length, είτε σε δεκαεξαδικό, αν το length είναι πολλαπλάσιο του 4, και συμπληρώνοντας το μήκος του με μηδενικά, μέχρι length, ή length/4, αντίστοιχα. Πρώτα, έχει ελεγχθεί το μήκος, αν έχει κατάλληλη τιμή (θετική), και ότι η δυαδική τιμή του num δεν υπερχειλίζει.

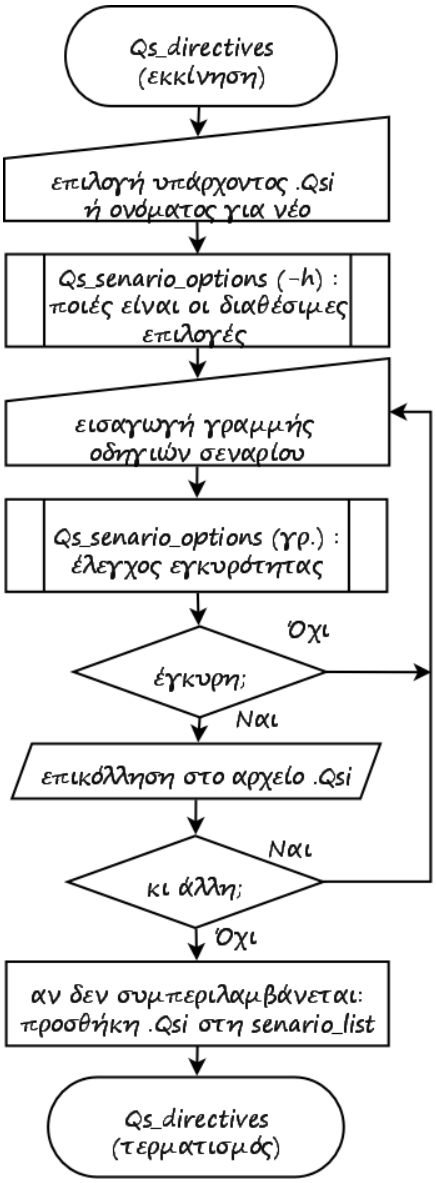
Qs_timestamp(filename, status) Επικόλληση στο τέλος του αρχείου filename, του μηνύματος status, συνοδευόμενου από την χρονική στιγμή, σε seed-ed μορφοποίηση.



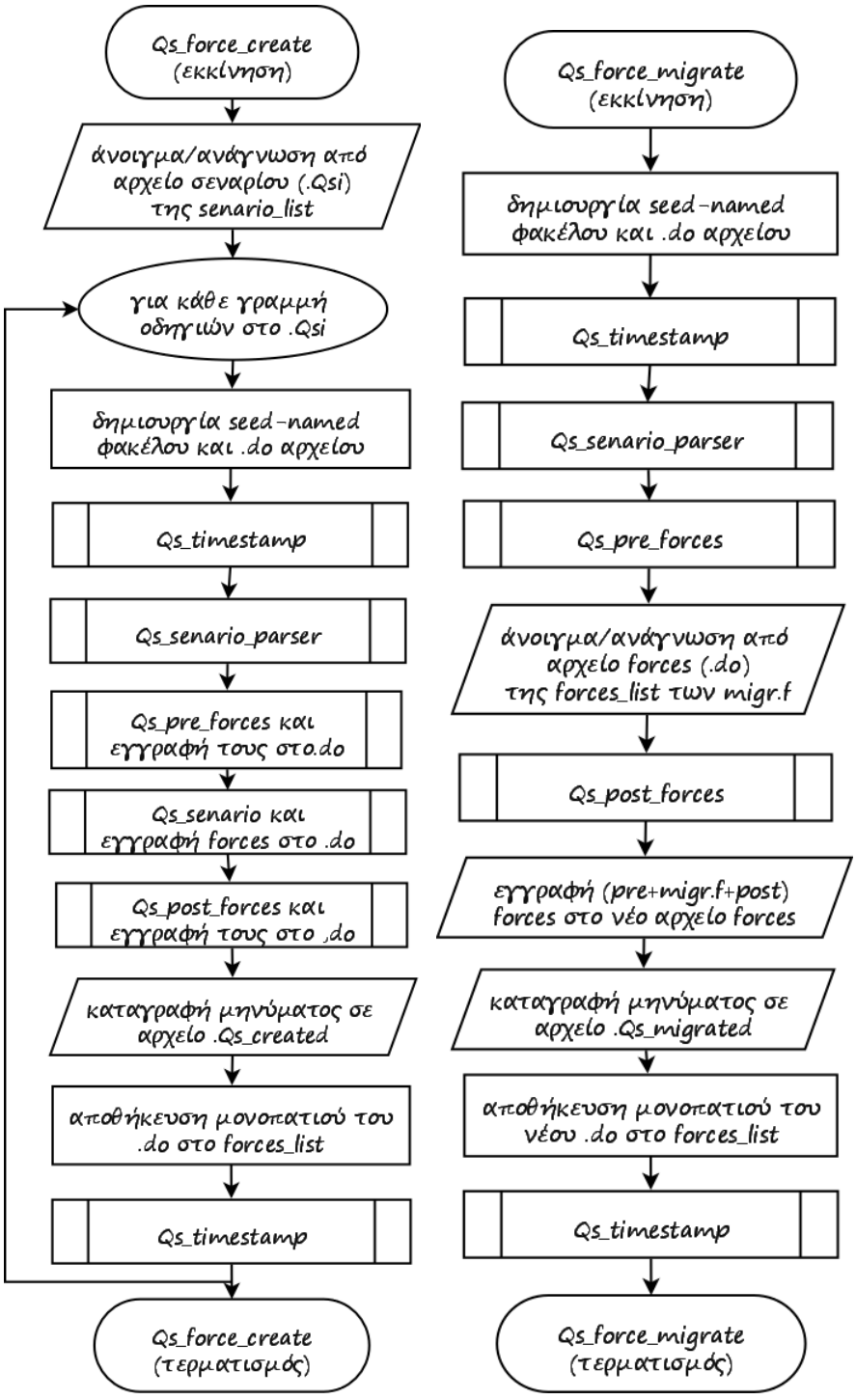
Σχήμα Α΄.15: Σημεία αναφοράς και διάρκειες στην παραγωγή επόμενων reset:

(i) αρχή default reset,
(ii) αρχή default αρχικοποίησης,
(iii) μηδενισμός default reset,
(iv) λήξη default αρχικοποίησης,
(v) αρχή επόμενου reset,

<1> απόσταση default reset από την αρχή του χρόνου, 1·<one_cycle> + <unal_res>,
<2> διάρκεια του reset, <reset_dur>,
<3> διάρκεια αρχικοποίησης, <init_max>,
<4> επιλεγμένη (σταθερή ή τυχαία) διάρκεια αναμονής επόμενου reset, <reset_next>



Σχήμα Α΄.16: Ροή εκτέλεσης της Qs_directives().



Σχήμα Α΄.17: Ροή εκτέλεσης των Qs_force_create() και Qs_force_migrate().

Α′.4.3.1 Παραγωγή forces

Η παραγωγή των εντολών πραγματοποιείται ανεξάρτητα στην Qs_senario(), με το αρχείο .Qsi να την απομονώνει από την σύνθεση των γραμμών σεναρίου, που θα έχει προηγηθεί, από τον χρήστη και την Qs_directives(). Η ανάλυση των, ήδη ελεγμένων και αποθηκευμένων, γραμμών που θα καθοδηγήσουν την Qs_senario(), επίσης γίνεται εξωτερικά, στην Qs_senario_parser(), και λαμβάνεται, μέσω Qs_D["senario_options"], η οντότητα με τις μεταβλητές των επιλογών, αρχικοποιημένες με τις τρέχουσες ή/και default τιμές τους, όπου δεν δόθηκαν.

Η σύνταξη της εντολής περιβάλλοντος force, είναι

> force [-freeze | -drive | -deposit] [-cancel <time>] [-repeat <time>] <object_name> <value> [<time>] [, <value> <time> ...]

Οι πρώτες (προαιρετικές) επιλογές καθοδηγούν την διατήρηση της τιμής, σε επόμενες αλλαγές, αλλά και την ακύρωση και επανάληψη της. Μετά από την απόδοση τιμής <value> στο σήμα <object_name>, μπορεί να οριστεί η χρονική στιγμή απόδοσης της τιμής, είτε σε σχετική απόσταση, είτε σε απόλυτο χρόνο προσομοίωσης, σε συνδυασμό με τον χαρακτήρα '@'. Χρησιμοποιήθηκε η απόλυτη αντιστοίχιση, για σαφέστερα αρχεία. Με την ίδια εντολή, μπορούν να αποδοθούν πολλαπλές τιμές, αλλά και να συμπεριλαμβάνεται, όπως και σε κάθε εντολή περιβάλλοντος του ModelSim, κώδικας Tcl. Τα σήματα πλάτους μεγαλύτερου 1 bit, μπορούν να λάβουν τιμές είτε ανά bit, είτε για το (πραγματικά) πλήρες τους πλάτος, και στις διάφορες συνήθεις βάσεις (2, 8, 10, 16).

Γενικά Εκτός του σήματος του ρολογιού, στην Qs_senario_parser(), ορίζονται και επιλογές γενικών παραμέτρων που αφορούν το χρονισμό της προσομοίωσης. (Σχήμα 6.20). Καταρχάς, η διάρκεια του κύκλου, one_cycle, που θα πρέπει να είναι θετικός ακέραιος και πολλαπλάσιο του δέκα[8], έχει default τιμή το 100. Η υποδιαίρεση της μονάδας χρόνου, time_unit, που αντιστοιχεί στην ορισμένη περίοδο, επιτρέπεται να είναι μόνο κάποια από τις : "s", "ms", "us", "ns", "ps", "fs". Η διάρκεια της προσομοίωσης, sim_dur, ορίζεται σε κύκλους ρολογιού, και πρέπει να είναι θετικός ακέραιος, με default τιμή το 1600. Αποτελεί άνω όριο των επιλογών που εκφράζουν συχνότητα, qi_freq, valid_fr, operation_fr, αλλά και του αύξοντος ακεραίου μετρητή εντολών, του βρόγχου της παραγωγής των forces. Δύο τιμές, unal_res για το reset, και unal_sigs για τα υπόλοιπα σήματα, μετατοπίζουν την ακμή των σημάτων σε σχέση με του ρολογιού, όπως απαιτείται για ορθό χρονισμό. Εκφραζόμενες σε δέκατα του κύκλου, με επιτρεπόμενες τιμές μεταξύ 1 και 9, (default τα 7 και 8, αντίστοιχα), μεταθέτουν τους ελέγχους περιορισμών στη διάρκεια του κύκλου. Η εντολή του σήματος του ρολογιού, clk, δεν κατασκευάζεται από

[8] για να είναι ακέραια και τα δέκατά του, τα unalignement

το Qs_senario(), αλλά δίνεται, τόσο στα παραγόμενα, όσο και στα έτοιμα .do, μεταξύ των αρχικών εντολών (pre), και έτσι, ο μετρητής βρόγχου των forces στην Qs_senario(), που αναπαριστά την τρέχουσα εντολή, <instr_num>, έχει αρχική τιμή 1. Στην αρχή του βρόγχου, υπολογίζεται, για την τρέχουσα επανάληψη, η χρονική στιγμή <time> αλλαγής τιμής των σημάτων, ίση με

(<instr_num>*<one_cycle>+<sig_offset>) <time_unit>,

όπου <sig_offset> είναι η προϋπολογισμένη απόκλιση των σημάτων, ίση με

<unal_sigs>*<one_cycle>/10.

Προηγουμένως, έχει κατασκευαστεί και το στοιχειώδες κομμάτι της εντολής, force_seed, ίσο με

"force /<system_entity>/".

reset Το reset αφορούν τρεις επιλογές σεναρίου: εκτός από το unal_res, για μετατόπιση της θετικής ακμής του σε σχέση με του ρολογιού, είναι και οι reset_dur και reset_next. Η πρώτη ορίζει τη διάρκεια ενεργοποίησης του σήματος σε κύκλους ρολογιού, με default τιμή το 1, και δυνατές τιμές τους ακεραίους από 1 ως 10. Το reset_next μπορεί να δίνεται επαναληπτικά, και οι τιμές του αντιστοιχούν σε διαστήματα αναμονής για την εφαρμογή του επόμενου reset, σε σχέση με το προηγούμενό του. Περιγράφουν τα επόμενα, μετά το πρώτο (default) reset που εμφανίζεται υποχρεωτικά, και σε συγκεκριμένο χρόνο. Η μηδενική τιμή αντιπροσωπεύει το τυχαίο διάστημα, με επιτρεπόμενη την ανάμειξη σταθερών και τυχαίων ορισμάτων. Αν δεν δοθεί κάποιο, η default τιμή της επιλογής είναι κενή λίστα. Το καθένα, χωριστά, οφείλει να είναι ακέραιος αριθμός, αλλά από κει και πέρα, ελέγχονται σε συσχετισμό όλα τα δοθέντα επόμενα reset.

Τα τυχαία ορίσματα καταμετρώνται, και τα σταθερά αθροίζονται. Προσθέτοντας το άθροισμα των σταθερών διαστημάτων, προσαυξημένο κατά ένα, όσο δηλαδή η διάρκεια αναμονής του default reset, σε τόσες διάρκειες αρχικοποίησης (init_max) όσα συνολικά τα reset, δηλαδή όλα τα προτεινόμενα επόμενα μαζί με ένα ακόμα, το default, προκύπτει το βασικό κόστος των reset. Αφαιρώντας το κόστος από την διάρκεια προσομοίωσης, και, αν έχουν δοθεί και επιλογές τυχαίων διαστημάτων, διαιρώντας με το πλήθος τους, προκύπτει το εναπομείναν διάστημα. Αυτό είναι που ελέγχεται, να είναι υπαρκτό, δηλαδή μεγαλύτερο του μηδενός, οπότε και ενσωματώνεται στην αρχή ([0]) της λίστας των reset_next, για να καθορίζει το δυνατό εύρος επιλογής τιμών τυχαίων διαστημάτων αναμονής στην Qs_sif_reset().

Σίγουρα, το σήμα reset ορίζεται αρκετά ανελαστικά. Το να πρέπει να έχει σταθερή διάρκεια σε κάθε του εμφάνιση, με κοινό reset_dur, και να μην επιτρέπεται να εμφανιστεί πριν την ολοκλήρωση της αρχικοποίησης που έχει προκαλέσει κάποιο προηγούμενο, δεν δίνει την δυνατότητα έκφρασης κάθε πιθανής συμπεριφοράς του, αλλά θεωρείται επαρκής για τις ανάγκες των δοκιμών.

Και ακόμα και αν δεν είναι, με τη δυνατότητα απόδοσης και εκτέλεσης αρχείων .do με έτοιμα forces και το Qs_force_migrate(), μπορεί να αναπληρωθούν οι εκφράσεις αυτές.

Όπως η ύπαρξη του ρολογιού και ο μηδενισμός όλων των σημάτων στις αρχικές εντολές, για κάθε συντακτικά έγκυρη γραμμή οδηγιών σεναρίου, είτε περιέχει επόμενα reset, είτε όχι, θα παραχθεί από την Qs_senario(), ως προαπαιτούμενο, το default reset, για να προκαλέσει την αρχικοποίηση της μνήμης Next σε μορφή Στοίβας, για τη FreeList. Έξωθεν του βρόγχου παραγωγής των υπολοίπων force, και χωρίς συνθήκη, δίνονται τα

force /<system_entity>/reset 1 @(<one_cycle>+<unal_res>) <time_unit>

force /<system_entity>/reset 0 @((<reset_dur>+1)*<o._c.> + <unal_res>) <time_unit>

Αν έχουν δοθεί και επιλογές για επόμενα reset, τα σημεία αλλαγής και οι νέες τιμές του σήματος, προϋπολογίζονται από την Qs_sif_reset() (σελίδα 124). Η λίστα που έχει επιστραφεί, διατρέχεται, παράγοντας τις εντολές των επόμενων reset. Όταν η υπολογισμένη χρονική στιγμή σε κύκλους (ή, εντολές) του τρέχοντος tuple ισούται με την τρέχουσα εντολή του βρόγχου, παράγεται μία force

force /<system_entity>/reset <value> @<time>

με την τιμή που συνοδεύει την χρονική στιγμή, χωρίς έλεγχο, και με χρόνο την γενική τιμή της συγκεκριμένης επανάληψης (<time>).

inID Το inID είναι ένας μετρητής, που σε κάθε επανάληψη αυξάνει κατά ένα και μηδενίζεται όταν φτάσει το μέγιστό του, σύμφωνα με το μήκος του σε bits. Δεν εξαρτάται από καμία επιλογή, μόνο μετατρέπεται σε δυαδική μορφή για τη force, μία εντολή σε κάθε επανάληψη του βρόγχου. Η πρώτη του αρχικοποίηση είναι σε 1, γιατί έχει ήδη τιμή μηδέν από τις αρχικές εντολές (pre).

dataIn Το dataIn δεν έχει επιλογές σεναρίου που να αφορούν την παραγωγή του, μιάς και στερείται επίδρασης στην κατάσταση του συστήματος. Από την άλλη όμως, είναι η σημαντικότερη ένδειξη ορθής λειτουργίας, και αυθαίρετα μεν, αλλά με σκοπό τον ευκολότερο εντοπισμό των τυχόν σφαλμάτων, η παραγωγή του υπόκειται σε μία 'συνταγή', αντί για πλήρη τυχαιότητα. Αν το μήκος του σε bits το επιτρέπει, όντας τουλάχιστον διπλάσιο από το 8 ή το 4, ένα ανάλογο κομμάτι, 8, 4, ή αλλιώς κανένα, από τα λιγότερο σημαντικά (LS) bits, θα παράγεται από μετρητή. Τα εναπομείναντα, που πάντα θα υπάρχουν, τουλάχιστον τα μισά του πλάτους του σήματος, περισσότερο σημαντικά (MS) bits, θα λαμβάνουν τυχαίες τιμές.

Αρχικά καθορίζεται το πλήθος των LS και MS. Αν των MS είναι πολλαπλάσιο του 4, όντας και των LS σε κάθε περίπτωση, ορίζεται η ανάλογη συνάρτηση μετατροπής σε δεκαεξαδικό, τα αντίστοιχα εισαγωγικά, και αναπροσαρμόζονται (/4) τα μήκη που θα καθορίζουν την συμπλήρωση με μηδενικά. Αλλιώς, θα παράγονται σε δυαδικό, με τις ανάλογες μεταβλητές για την μετατροπή τους επί τόπου, χωρίς την χρήση της Qs_signal(). Μέσα στον βρόγχο των εντολών, τα LS, αν υπάρχουν, παράγονται ως μετρητής, που μηδενίζεται στο μέγιστό του. Τα MS, είτε για να συνενωθούν, είτε μόνα τους, παράγονται από μία γεννήτρια τυχαίων δυαδικών ψηφίων, ορισμένου πλήθους, συγκεκριμένα του υπολογισμένου μήκους. Και τα δύο τμήματα, μετατρέπονται στην τελική τους μορφή, δυαδικά ή δεκαεξαδικά, συμπληρώνονται με μηδενικά στο επιθυμητό τους μήκος, και η τιμή εισάγεται για παραγωγή μίας force σε κάθε νέα εντολή.

qi Το σήμα qi καθορίζεται από δύο επιλογές: την qi_code και qi_freq. Η qi_code αναπαριστά τις επιθυμητές συμπτώσεις με έναν δεκαδικό αριθμό, του οποίου το δυαδικό αντίστοιχο κωδικοποιεί με κάθε ψηφίο του μία 'συγγενική' θέση στο μονοπάτι δεδομένων, συγκεκριμένα (MS:) [n-3],[n-2],[n-1] (:LS), για βάθος 4. Έτσι, για παράδειγμα, το $0_{10} = 000_2$ δηλώνει την επιθυμία ύπαρξης παραθύρου, χωρίς καθόλου συμπτώσεις, το $4_{10} = 100_2$, μόνο συμπτώσεις [n-3], και το $7_{10} = 111_2$, ότι όλες επιτρέπονται, που είναι και το default. Αυτό δεν είναι αρκετό, και η qi_freq ορίζει την επιθυμητή συχνότητα εμφάνισης των συμπτώσεων. Βέβαια, δεν είναι πάντα εφικτή, καθώς μία τιμή μπορεί να προκαλεί εμφάνιση άλλης σύμπτωσης από την επιθυμητή, άρα είναι μία μέγιστη συχνότητα. Η τιμή μηδέν συχνότητας, η default, κωδικοποιεί την αίτηση τυχαίων συχνοτήτων εμφάνισης των επιθυμητών συμπτώσεων. Αν ο κωδικός τους είναι 0, η επιλογή της συχνότητας δεν έχει καμία επίδραση.

Πριν ξεκινήσει η παραγωγή των forces μέσα στον βρόγχο, εξετάζονται οι επιλογές και αρχικοποιούνται οι αναγκαίες δομές και μεταβλητές. Δύο λίστες, η out_window, και η in_window, διατηρούν τα στοιχεία που υπάρχουν εκτός και εντός του μονοπατιού δεδομένων, αντίστοιχα. Διαφέρουν κυρίως, στο ότι η out_window απλά αποτελεί ένα σύνολο, μη διατεταγμένο, αλλά η in_window, υποδηλώνει και την διάταξη των αριθμών qi στα στάδια ομοχειρίας. Μάλιστα, η python εισάγει (append) στοιχεία στο τέλος μίας λίστας, δηλαδή μετά το μέγιστο δείκτη (i), αλλά αφαιρεί (remove) ξεκινώντας από τον μικρότερο ([0]). Έτσι, το στάδιο 4, και αντίστοιχα, για 'συγγένειες', η [n-3], θεωρείται ότι είναι στο δείκτη 0, και το στάδιο 1, δηλαδή το νέο στοιχείο, στο δείκτη 3 της λίστας in_window. (Σχήμα Α΄.18). Το in_window, αρχικοποιείται με τυχαίες, αλλά χωρίς συμπτώσεις, τιμές οι οποίες αφαιρούνται από το out_window, που αρχικά περιέχει όλο το εύρος διευθύνσεων για το πλήθος των ουρών του συστήματος.

Αν ο κωδικός qi_code είναι μη μηδενικός, επιτρέπει τις συμπτώσεις, οπότε μετατρέπεται σε δυαδικό (qi_cd_bin), για να εξαχθούν στην συνέχεια οι τιμές x των επιθυμητών 'συγγενειών' [n-x], (λίστα qi_ns). (Σχήμα Α΄.18). Ακόμα,

pipeline_depth

<p_d>		0	1	...	<p_d>-2
4	i	0	1		2
<p_d>,4	qi_cd_bin	0/1	0/1	...	0/1
<p_d>	nearest x	<p_d>-1	<p_d>-2	...	0
4	"(n- x)"	3	2		1
<p_d>	qi_ns =	<p_d>-0-1	<p_d>-1-1		<p_d> - (<p_d>-1) -1
4	<p_d>-i-1	4-0-1	4-1-1		4-2-1

Σχήμα Α′.18: Οι δείκτες στις δομές παραγωγής των qi.

αρχικοποιείται, σε 1, ένας μετρητής συχνότητας (curr_fr_qi), με όριό του (lim_fr_qi) τη σταθερά που δόθηκε στο qi_freq. Αν δόθηκε μηδέν, δηλαδή τυχαία συχνότητα, και το όριο αρχικοποιείται σε 1, για να υπάρξει από την αρχή παραγωγή εντολής, με την άμεση ισότητά τους, αλλά στην συνέχεια θα μεταβληθεί εκ νέου.

Μέσα στο βρόγχο, η παραγωγή διαφοροποιείται ανάλογα με τον κωδικό των επιθυμητών συμπτώσεων. Αν δόθηκε μηδέν, σε κάθε επανάληψη, λαμβάνεται το στοιχείο που βρίσκεται στο st4 (n4_Qi), και αφαιρείται από το in_window. Προτού η τιμή αυτή επαναπροστεθεί στο out_window, επιλέγεται από εκεί τυχαία μία νέα τιμή qi (n0_Qi), η οποία αφαιρείται, για να προστεθεί στο in_window, και τελικά οι δύο λίστες είναι ξένα σύνολα, καθεμία με μοναδικά στοιχεία. (Σχήμα Α′.19). Η νέα τιμή (n0) μετατρέπεται στην κατάλληλη βάση, και παράγεται η εντολή force, μία για κάθε κύκλο.

Όταν έχουν ζητηθεί συμπτώσεις, και ο μετρητής συχνότητας curr_fr_qi φτάσει στο καθορισμένο όριο lim_fr_qi, επιλέγεται τυχαία από την λίστα με τις επιθυμητές συμπτώσεις qi_ns η επόμενη πιθανή (αιτούμενη, requested), και υπολογίζεται βάσει αυτού το επόμενο qi (n0). Δεν είναι δεδομένο ότι θα εμφανιστεί με την εισαγωγή του υπολογιζόμενου n0 η αιτούμενη συγγένεια, καθώς είναι πιθανό να υπάρχουνε και κοντινότερες τιμές ίδιες, από προηγούμενες συμπτώσεις. Ελέγχεται λοιπόν η ύπαρξη της υποψήφιας τιμής στις νεότερες, από

i	0	1	2	3	0	1	2	3
in_window	n4_Qi	(n3_qi)	(n2_qi)	n0_qi (παλιό)	(n3_qi)	(n2_qi)	n0_qi (παλιό)	n0_qi
out_window	{oi, ..., oj , n0_Qi (επόμενο)}				{oi, ..., oj , n4_Qi}			

Σχήμα Α′.19: Τα σύνολα τιμών qi εντός και εκτός του μονοπατιού in_window, out_window.

την τρέχουσα αιτούμενη, θέσεις του μονοπατιού. Αν υπάρχει νεότερη ίση, η αίτηση αποτυγχάνει, ή στην περίπτωση που η συχνότητα δεν ήταν κατάλληλη, επιλέγεται διαφορετική νέα τιμή για n0, από το out_window. Τελικά, λαμβάνεται από την κεφαλή του in_window το παλαιότερο qi (n4), αλλά προτού εισαχθεί στο out_window, πρέπει να ελεγχθεί αν πράγματι δεν περιλαμβάνεται και πάλι σε άλλη θέση του μονοπατιού, μέσα στο in_window. Αν όχι, καταχωρείται στο out_window, και πάντως, το n0 στο τέλος του in_window. Η εντολή force παράγεται αν υπάρχει αλλαγή στην τιμή, δηλαδή αν δεν ήταν επιτυχής αίτηση για σύμπτωση 1, με το ακριβώς προηγούμενο, εκτός και αν πρόκειται για την πρώτη force, και άρα θα πρέπει να παραχθεί. Αν ήταν επιτυχής αίτηση, ο μετρητής συχνότητας επαναρχικοποιείται σε 1, ή, αν δεν ήταν καν η στιγμή, απλά αυξάνεται. Αν έχει ζητηθεί τυχαία συχνότητα, αλλάζει τιμή και το όριο. Η επιλογή γίνεται μέσα από το εύρος 1 έως 10, που θα δώσει καλύτερη συχνότητα εμφάνισης των συμπτώσεων, συγκριτικά με εκείνη που θα έδινε το εύρος του αριθμού υπόλοιπων εντολών, μέχρι το τέλος της διάρκειας προσομοίωσης[9]. Αν η αίτηση απέτυχε, ο μετρητής και το όριο της συχνότητας διατηρούνται, για να επιχειρηθεί σύμπτωση, αν κι όχι απαραίτητα η ίδια, στην επόμενη επανάληψη.

valid Το σήμα valid εκφράζεται με τις επιλογές valid_init και valid_fr. Το πρώτο καθορίζει την αρχική τιμή του σήματος, και το δεύτερο την συχνότητα εναλλαγής του, για σταθερά διαστήματα, ή με τιμή μηδέν, την τυχαία ακολουθία, το default. Πριν τον βρόγχο, αρχικοποιούνται ο μετρητής και το όριο συχνότητας με την δοθείσα τιμή. Ακόμα, μία δομή ουράς deque της python, με μέγιστο μήκος δύο, και αρχικές τιμές ένα 0 και ένα 1, με την καθορισμένη από την valid_init διάταξη. Οι deque με περιορισμό μήκους, έχουν την ιδιότητα να αποβάλλουν το στοιχείο στην θέση [0] κατά την εισαγωγή (append) ενός νέου στοιχείου, που θα τοποθετηθεί στο άλλο άκρο, για να διατηρήσουν το μήκος τους. Έτσι, μέσα στον βρόγχο, όταν ο μετρητής εξισωθεί με το όριο της συχνότητας, χρησιμοποιείται άμεσα το στοιχείο στη θέση [0], τόσο για την παραγωγή της force, χωρίς επιπλέον έλεγχο, όσο και για να εκτοπίσει με την εισαγωγή του στο τέλος, τον εαυτό του από την αρχή, προωθώντας την άλλη τιμή εκεί. Στην περίπτωση μεγιστοποίησης του μετρητή, επαναρχικοποιείται σε 1, διαφορετικά αυξάνεται. Αν έχει δοθεί μηδενική συχνότητα, η τυχαία ακολουθία παράγεται επιλέγοντας σε κάθε επανάληψη τυχαία μεταξύ των αριθμών 0 και 1. Οι έλεγχοι και η παραγωγή εντολών, αρχίζουν από την δεύτερη εντολή και μετά, εξασφαλίζοντας ότι το valid ενεργοποιείται μετά την εφαρμογή του default reset, χωρίς το οποίο δεν είναι, εξ ορισμού, δυνατή η ορθή λειτουργία του συστήματος.

operation Το σήμα operation ελέγχεται από τις επιλογές operation_init και operation_fr, και παράγεται πανομοιότυπα με το valid, πλην όμως ανεξάρ-

[9] υπολογίστηκαν 20% έναντι ενός αρχικού 6%.

τητα, όπως είναι αναγκαίο.

run Η απόδοση τιμών με εντολές force, όπως προαναφέρθηκε, είναι αρκετά ευέλικτη. Μπορεί, για παράδειγμα, να καθοριστεί μιά εκτενής ακολουθία για ένα σήμα, και στην συνέχεια, πλήρως κάποιο άλλο, ή να εναλλάσσονται τα σήματα-στόχοι των εντολών. Ακόμα, έχοντας απόλυτες χρονικές αναφορές, είναι δυνατό να δοθούν εντολές για το ίδιο σήμα, ακόμα και ανακατεμένα στο χρόνο, ή για χρόνους μεταγενέστερους από κάποια στιγμή στην οποία θα κληθεί να φτάσει η εκτέλεση. Μάλιστα, όλα τα παραπάνω, μη διακόπτοντας την προσομοίωση, για τις διαδοχικές εκτελέσεις ή την απόδοση επιπλέον τιμών! Παρόλα αυτά, καλύτερο είναι, σύμφωνα και με την πρακτική που ακολουθεί το ίδιο το εργαλείο και παρατηρείται στα παράγωγα εντολών tssi2mti, να παρεμβάλλεται μία εντολή προόδου της προσομοίωσης, run, μεταξύ μερικών εντολών force. Πράγματι, η απόδοση ευνοείται, και το πλήθος των εντολών για τις οποίες το όφελος είναι εμφανέστερο, εκτιμήθηκε περίπου στις 1500, αν και στα αρχεία του tssi2mti ήταν ακόμα λιγότερες.

Η επιλογή αυτή, προϋποθέτει, λογικά και όχι υποχρεωτικά, τα σήματα να λαμβάνουν εναλλασσόμενα τις τιμές τους, κάτι που επιτυγχάνεται με τον ενιαίο βρόγχο παραγωγής. Για την σταδιακή ενσωμάτωση εντολών run, σε κάθε εξίσωση του αριθμού εντολών με ένα ακέραιο πολλαπλάσιο των 1500, εισάγεται μία run, για εκείνο το χρονικό σημείο, κι επίσης, μία στο τέλος της διάρκειας προσομοίωσης, στις περιπτώσεις που υπάρχει υπόλοιπο εντολών. Εκτός από την εισαγωγή της εντολής run, για βελτίωση της απόδοσης της προσομοίωσης, στο ίδιο σημείο του βρόγχου, οι εντολές force που συντέθηκαν για το συγκεκριμένο χρονικό τμήμα, γράφονται στο αρχείο .do, βελτιώνοντας και την απόδοση της παραγωγής του.

Α΄.4.4 Προσομοίωση

Η Qs_hdl_sim(), από το ομώνυμο module, προετοιμάζει και καλεί τις εντολές μεταγλώττισης και εκτέλεσης της προσομοίωσης. Αρχικά, καταγράφεται η χρονική στιγμή έναρξης σε ένα .Qts αρχείο, όπως θα καταγραφεί και η αντίστοιχη χρονική στιγμή στο τέλος, θεωρώντας το σύνολο των ενεργειών ότι αποτελούν το στάδιο της προσομοίωσης. Αναλυτικότερα, αρχικά γίνεται μεταγλώττιση όλων των αρχείων του project με την εντολή

> vcom -work <working_library> -2002 -explicit <prj_path>/<source_dir>/*.vhd

Στη συνέχεια, μιάς και το ModelSim απαιτεί, σε περίπτωση ύπαρξης περισσότερων αρχιτεκτονικών, εκείνη που θα εκτελεστεί να έχει μεταγλωττιστεί τελευταία, προστίθεται μία ακόμα εντολή μεταγλώττισης. Ανάλογα με την επιλογή που έχει δοθεί στην Qs_tester() στην γραμμή εντολών, -B ή όχι, παράγεται η

vcom -work <working_library> -2002 -explicit <prj_path>/<source_dir>/<system_entity>.vhd

για Behavioral προσομοίωση, αρχικοποιώντας μία μεταβλητή architecture σε <Behav_arch>. Για επιλογή Timing προσομοίωσης, η εντολή

vcom -work <working_library> -2002 -explicit <prj_path>/<par_dir>/<system_entity>_<TimeSim_tag>.vhd .

Η architecture γίνεται ίση με <TimeSim_arch>, και ακόμα η μεταβλητή sdf_more, που αλλιώς μένει κενή συμβολοσειρά, γίνεται

-sdfmax /<system_entity>/=<prj_path>/<par_dir>/<system_entity>_timesim.sdf

περιγράφοντας το αρχείο με τις καθυστερήσεις, .sdf (standard delay format), για την εντολή εκτέλεσης της προσομοίωσης, δηλαδή την

vsim -t ps -do <force_list[force_index]> <sdf_more> <working_library>.<system_entity> <architecture> -c -quiet

και για τις δύο περιπτώσεις, και με τα διαχωριστικά των επιπέδων του συστήματος αρχειοθέτησης, μεταφρασμένα σύμφωνα με τις προσδοκίες του εργαλείου. Η εντολή λαμβάνει ως όρισμα, τόσο το όνομα της αρχιτεκτονικής προς εκτέλεση, όσο και το αρχείο .do, με τις εντολές απόδοσης τιμών στα σήματα εισόδου, και καταγραφής των τιμών όλων των σημάτων. Το όρισμα -c, επιλέγει την κατάσταση λειτουργίας στη γραμμή εντολών, μιάς από τις δύο καταστάσεις λειτουργίας του ModelSim, που δεν καλεί το γραφικό του περιβάλλον, από τις τρείς συνολικά που διαθέτει. Το μονοπάτι εργασίας μεταφέρεται στον φάκελο του project, στο <prj_path>\<sys_dir>, για να γίνουν άμεσα διαθέσιμες οι πληροφορίες του στο εργαλείο. Οι παραπάνω εντολές συγκεντρώνονται σε μία λίστα, η οποία διατρέχεται καλώντας την κάθε επόμενη εμμέσως στην γραμμή εντολών, έχοντας επιβεβαιώσει ότι η προηγούμενη ήταν επιτυχής.

Α΄.4.5 Μοντέλο

Η συνάρτηση Qs_manager(), από το ομώνυμο module, εκτελεί τη λειτουργία του μοντέλου, με εισόδους (από) το αρχείο .Qli, το οποίο προκύπτει από την αποθήκευση των τιμών των σημάτων εισόδου από την προβολή List του ModelSim, που ήδη εκτελέστηκαν από το σύστημα. Παράγει το αρχείο εξόδων του μοντέλου, .Qmo, το οποίο αργότερα θα συγκριθεί με το αντίστοιχο του συστήματος, .Qlo, για απόδειξη της ταύτισης των λειτουργιών, ή μη.

Αναλυτικότερα, μετά από τον έλεγχο ύπαρξης όλων των απαραίτητων στελεχών του Qs_dictionary, καταγράφεται στο .Qts η χρονική στιγμή της έναρξης εκτέλεσης του μοντέλου, το οποίο θα επαναληφθεί και στο τέλος, για την καταγραφή της ολοκλήρωσής της. Ανοίγει το αρχείο .Qli, από όπου διαβάζονται, ανά γραμμή, δηλαδή, ανά κύκλο, οι τιμές εισόδου που αντιστοιχούν

σε μία εντολή. Ακόμα, αρχικοποιούνται οι βασικές δομές του μοντέλου: μία λίστα από ουρές, (δομής deque της Python), όσες και το πλήθος των ουρών στο σύστημα, ένας μετρητής συνολικά διαθέσιμων θέσεων (positions_avail), που αντιστοιχεί στη λειτουργία του διαχειριστή μνήμης, και τα μεγέθη αρχικοποίησης, η σταθερά για το μέγιστο μήκος (max_init), και ο μετρητής της (init_length).

Επίσης, είναι αναγκαία μία επιπλέον δομή ουράς (deque), μήκους όσο το βάθος του μονοπατιού δεδομένων, για να προσομοιώνει την καθυστέρηση που υφίστανται οι τιμές κατά τη διάσχιση του, ώστε να συγχρονίζονται με το σήμα της αρχικοποίησης, initializing, το οποίο δεν υφίσταται καθυστέρηση. Παρόλο που η ίδια η αρχικοποίηση, από πλευράς μοντέλου δεν υφίσταται, τουλάχιστον όχι ως διεξοδική διαδικασία, είναι απαραίτητη η μοντελοποίησή της, καθώς η εμφάνισή της επηρεάζει την συμπεριφορά και των δύο, συστήματος και μοντέλου, καθορίζοντας, με την εμφάνισή της, τις εκτελέσιμες εισόδους. Διαφορετικά, το μοντέλο είναι σε θέση να εκτελέσει εισόδους τις οποίες το σύστημα αναγκαστικά αγνοεί, με αποτέλεσμα τη διαφορετική έκβαση. Άλλωστε, με τον τρόπο αυτό, το σύνολο των σημάτων εξόδου γίνεται διαθέσιμο, συμπεριλαμβάνοντας το initializing. Η δομή του amortized, αρχικοποιείται με μηδενικά, όπως θα είναι και οι τιμές του συστήματος αρχικά.

Στην συνέχεια, ανά γραμμή, διαβάζεται και εκτελείται το αρχείο εισόδου. Αφού αγνοηθούν οι αρχικές γραμμές, που το List γράφει τα ονόματα των σημάτων, η γραμμή χωρίζεται στις επιμέρους τιμές, και αρχικά ελέγχεται η τιμή του reset. Αν έχει δοθεί reset, τα υπόλοιπα σήματα εισόδου αγνοούνται, και το μοντέλο μπαίνει σε κατάσταση αρχικοποίησης, επαναφέροντας όλες τις δομές στις αρχικές τιμές, εκτός του μετρητή αρχικοποίησης. Για τον πρώτο κύκλο, για αντιστοιχία με τα αποτελέσματα της δειγματοληψίας του συστήματος, το initializing παραμένει 0. Για τους υπόλοιπους κύκλους τιμής reset 1, ή, όταν έχει πλέον μηδενιστεί, καθόλη τη διάρκεια της μετρώμενης, αντίστροφα, αρχικοποίησης, παράγεται initializing 1, αλλά done και dataout, 0.

Αν η τιμή του reset είναι 0, αλλά δεν θεωρείται ότι πρόκειται για διάρκεια αρχικοποίησης, λαμβάνονται υπόψιν και οι υπόλοιπες τιμές των σημάτων, και καταστάσεις των δομών, και εκτελείται η κανονική λειτουργία του μοντέλου, παράγοντας πάντα initializing 0. Και στις δύο περιπτώσεις, ελέγχεται ταυτόχρονα, ο τύπος και η εγκυρότητα της εντολής, τόσο βάσει του εξωτερικού σήματος valid, όσο και της εσωτερικής κατάστασης, όντας για την NQ η διαθεσιμότητα θέσης, και για DQ, η ύπαρξη στοιχείων στην εν λόγω ουρά. Αν όλα έχουν καλώς, συμπεριλαμβανόμενης της τιμής αριθμού αιτούμενης ουράς, εκτελείται στην συγκεκριμένη ουρά μία εισαγωγή ή εξαγωγή στοιχείου, και μεταβάλλεται αναλόγως ο μετρητής συνολικών θέσεων.

Παράγονται επίσης, τα σήματα done, 0 για noops και άκυρες εντολές, αλλιώς 1, και dataout, 0 στα NQ, ενώ στα DQ, ίσο με την τιμή που εξήχθηκε από την ουρά. Τελικά, ανακτάται η κορυφαία γραμμή του amortized, και μαζί με το τρέχον initializing, γράφονται ως μία γραμμή, στο αρχείο εξόδου .Qmo. Οι λοιπές τρέχουσες τιμές των σημάτων εξόδου, μορφοποιούνται σε μία γραμμή,

που περιλαμβάνει, εκτός των done, dataOut, αυτούσια τα inID και qi, όπως διαβάστηκαν από την είσοδο, για να βγούν μέχρι την έξοδο, όπως στο σύστημα, χωρίς επεξεργασία, ως outID, qout. Η τρέχουσα γραμμή των εξόδων του μονοπατιού, εισάγεται στο τέλος του amortized, για να καθυστερήσουν ανάλογα.

Για την μοντελοποίηση της αρχικοποίησης ειδικότερα, το init_length έχει δοθεί ίσο με max_init= (Qs_D['init_max'] + 1), και ο πρώτος κύκλος εμφάνισης reset ανιχνεύεται με την ισότητα τους, μιάς και στη δειγματοληψία δεν φαίνεται το initializing απευθείας. Στους επόμενους κύκλους reset, βγαίνει initializing 1, αλλά δεν αρχικοποιεί ακόμα, ενώ αφού απενεργοποιηθεί, αρχίζει και μετρά αντίστροφα η διάρκεια της αρχικοποίησης, βάσει του init_length (μειωμένου κατά 1 μετά την ανίχνευση του πρώτου κύκλου). Όταν το init_length μηδενιστεί, ή έχει ήδη επαναρχικοποιηθεί, επιτρέπει την κανονική λειτουργία του μοντέλου.

Αν έρθει εμβόλιμο reset, χωρίς να έχει τελειώσει η αρχικοποίηση που έχει αρχίσει, δεν επαναλαμβάνεται επί τόπου, όπως συμβαίνει στο σύστημα. Για το λόγο αυτό, έγινε δεύτερη εκδοχή του Qs_manager, η Qs_manager_embolima(), και η εκδοχή του Tester που την καλεί, Qs_tester_embolima(). Τα εμβόλιμα reset υποστηρίζονται ελέγχοντας την εμφάνιση του πρώτου ενεργού κύκλου reset ανεξάρτητα από την τιμή του init_length. Πλέον, μπορεί να έχει τιμή μικρότερη του max_init και να έρθει καινούριο 'πρώτο' reset. Τότε, το init_length επανέρχεται στην κανονική του τιμή, (max_init -1) και η αρχικοποίηση αρχίζει από την αρχή, το οποίο για το μοντέλο σημαίνει ουσιαστικά την διάρκεια. Ακόμα, ενώ στον πρώτο κύκλο νέου reset, για λόγους δειγματοληψίας, το initializing είναι 0, στον πρώτο κύκλο εμβόλιμου reset είναι 1, όπως και στο σύστημα. Κατά τα άλλα, η λειτουργία του μοντέλου είναι αμετάβλητη.

Α΄.4.6 Σύγκριση

Με την Qs_sVm_compare(), από το ομώνυμο module, πραγματοποιείται η σύγκριση των αρχείων εξόδου του συστήματος έναντι του μοντέλου, για την ανίχνευση των ενδεχόμενων διαφορών. Δεν έχουν όλα τα σήματα την ίδια βαρύτητα, και κατά περίπτωση, μπορεί να δίνουν σφάλμα (error) ή προειδοποίηση (warning). Η στοίχιση των γραμμών γίνεται βάσει των outIDs, και όχι του χρόνου, που βρίσκεται στο .Qlo, και θα μπορούσε να υπολογίζεται για το .Qmo, μεν, αλλά και πάλι θα απαιτούνταν ένα πιό εγγενές σημείο αναφοράς, για συγχρονισμό.

Καταρχάς καταγράφεται η χρονική στιγμή έναρξης της σύγκρισης, όπως και στο τέλος, της λήξης της. Ανοίγονται έπειτα τα αναγκαία αρχεία των εξόδων, .Qlo, .Qmo, που θα αποτελέσουν τις εισόδους του συγκριτή, και το αρχείο εξόδου του, το .Qfc, στο οποίο αρχικά καταγράφεται το είδος της προσομοίωσης που είχε εκτελεστεί. Αρχικοποιείται και μία λίστα συμβολοσειρών, για την κωδικοποίηση των διαφόρων αναμενόμενων σφαλμάτων και προειδοποιήσεων, και την σύνθεση μηνυμάτων παρακάτω. Διαβάζεται μία γραμμή από

	outID	>>	initializing	>>	done	>>	dataout	>>	qout
!=	στοίχιση		E		initializing = 1 => W		done = 0 => W		W
					initializing = 0 => E		done = 1 => E		

Σχήμα Α′.20: Προτεραιότητα σημάτων και βαρύτητα διαφορών κατά την σύγκριση.

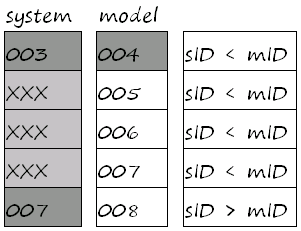
κάθε αρχείο, και ξεκινάει η σύγκριση σε βρόγχο, με την συνθήκη ότι από το αρχείο του συστήματος δεν έχει διαβαστεί κενή γραμμή, που ισοδυναμεί με το τέλος του (EOF, end of file).

Η γραμμή του αρχείου του συστήματος ελέγχεται, απαλείφοντας πρώτα τους χαρακτήρες '/', '_', '\n', για να αγνοηθεί αν πρόκειται για καταγραφή των ονομάτων των σημάτων από το List, διαβάζοντας την επόμενη γραμμή μόνο από το αρχείο του συστήματος. Όταν πρόκειται πλέον για γραμμή με τιμές σημάτων, πάλι η γραμμή του συστήματος, που ενδέχεται να εμφανίσει, ελέγχεται για την ύπαρξη απροσδιόριστων τιμών ('X' ή '?') σε οποιοδήποτε σήμα, το οποίο και καταγράφεται άμεσα στο αρχείο εξόδων. Η σύγκριση προχωράει σε επόμενες γραμμές, χωρίς αντιπαράθεση των επιμέρους τιμών, το οποίο θα είχε πρόσθετες δυσκολίες στη μετατροπή των τιμών, αλλά κυρίως δεν θα είχε νόημα.

Αν δεν εντοπιστούν απροσδιόριστες τιμές, ανακτώνται από τις γραμμές οι επιμέρους τιμές κάθε σήματος, που μετατρέπονται από string σε int, για να συγκριθούν ως αριθμοί, και κατά σειρά βαρύτητας (outID > initializing > done > dataout > qout). Κάθε διαφορά σε κάποια ανώτερης προτεραιότητας τιμή έχει ως αποτέλεσμα την παράκαμψη των ελέγχων των υπολοίπων, σημειώνοντας τον κωδικό που αντιστοιχεί στη λίστα, θεωρώντας 1 το δείκτη της πρώτης θέσης, και αυξάνοντας ένα μετρητή, των σφαλμάτων ή των προειδοποιήσεων. Μήνυμα προβάλλεται άμεσα, αν δεν έχει αποτραπεί κατ΄ επιλογήν (-q), και στην οθόνη. Τα διαφορετικά outID προκαλούν την διαδικασία της στοίχισης, ενώ διαφορετικές τιμές στις υπόλοιπες μεταβλητές, προκαλούν είτε σφάλμα, είτε προειδοποίηση.

Η ταύτιση όλων των τιμών σηματοδοτείται με τον κωδικό μηδέν, ενώ διαφορετικά qout προκαλούν προειδοποίηση. Ομοίως, και τα διαφορετικά dataout, όταν συνυπάρχουν με μηδενικό done, και τα διαφορετικά done, όταν το initializing είναι 1. Κάθε άλλη διαφορά, στα dataout με done 1, στα done με initializing 0, στα initializing γενικά, προσμετράται στα σφάλματα. (Σχήμα Α′.20)

Η στοίχιση, για τα διαφορετικά outIDs, διατηρείται για όσο οι τιμές τους διαφέρουν, ακόμα και αν διαβάζοντας παρακάτω, η μεταξύ τους ανίσωση αντιστραφεί. Κανονικά, η ανάγνωση προοδεύει μόνο στο αρχείο που έχει βρεθεί η μικρότερη μεταξύ τους τιμή. Αν εμφανιστεί απροσδιόριστη τιμή, από το αρχείο του συστήματος, η τρέχουσα τιμή του (sID) δεν θα αντικαθίσταται από την



Σχήμα Α΄.21: Προσπέραση κατά την στοίχιση.

ανακτηθείσα, διατηρώντας της σύγκριση εφικτή και την στοίχιση ενεργή, μέχρι την εμφάνιση έγκυρης τιμής. Σε περίπτωση που η στοίχιση, παρακάμπτοντας απροσδιόριστες τιμές sID, και φτάνοντας σε νέα ακέραια τιμή, έχει, εν δυνάμει, προσπεράσει την τιμή (mID) του μοντέλου, (σχήμα Α΄.21), η στοίχιση θα εξακολουθήσει, διαβάζοντας πιά από το άλλο αρχείο. Η κάθε απροσδιόριστη τιμή θα καταγραφεί άμεσα και στο αρχείο εξόδου. Επίσης, αν οποιοδήποτε αρχείο φτάσει στο τέλος, η στοίχιση και η σύγκριση, σταματούν. Όταν τα outIDs εξισωθούν, η σύγκριση συνεχίζει.

Η καταγραφή των σφαλμάτων και προειδοποιήσεων στο αρχείο εξόδου γίνεται στο τέλος κάθε επανάληψης του βρόγχου των συγκρίσεων, συμπεριλαμβάνοντας το είδος του συμβάντος και τις εμπλεκόμενες γραμμές των δύο αρχείων. Όταν η σύγκριση ολοκληρωθεί, συμπληρώνεται με ένα συμπερασματικό μήνυμα, με τις καταμετρημένες εμφανίσεις διαφορών, ή της απουσίας τους.

Αναμενόμενη είναι η εμφάνιση στα αποτελέσματα των Behavioral προσομοιώσεων, τριών σφαλμάτων απροσδιορίστων τιμών στο outID. Στο σύστημα, πράγματι δεν λαμβάνει για τους πρώτους κύκλους τιμή, και οι απροσδιόριστες τιμές διέρχονται ελεύθερα, μιάς και το load_enable των αντίστοιχων καταχωρητών είναι πάντα στο 1, και το clear στο 0. Επίσης, γενικά εμφανίζεται πληθώρα προειδοποιήσεων για διαφορά στα qout, και μερικές στα done, που ουσιαστικά αντιστοιχούν στη διάρκεια της αρχικοποίησης, όπου το σύστημα μηδενίζει τους καταχωρητές του, σε αντίθεση με το μοντέλο, που το σήμα qout παράγεται με άμεση διοχέτευση του qi που είχε διαβαστεί και υπάρχουν και τιμές από προηγούμενους κύκλους μέσα στο amortized. Λόγω της χαμηλότερης προτεραιότητας, οι διαφορές στα σήματα dataOut, κατά τη διάρκεια της αρχικοποίησης συνήθως θα υπερκαλυφθούν από των done, αλλιώς θα εμφανιστούν. Αρκετές προειδοποιήσεις για διαφορετικές τιμές στα dataOut, εμφανίζονται εκτός της διάρκειας αρχικοποίησης, μιάς και η μνήμη αντανακλά στην έξοδο ό,τι γράφει, αλλά και διατηρεί ό,τι διαβάζει, αν δεν διαβάσει κάτι άλλο. Εμφανίζονται έτσι, σε εντολές με done 0, παλιότερες έξοδοι της DataMem. Αντίθετα, σε έγκυρες και επιτυχείς εντολές NQ, η εμφάνιση διαφορετικών τιμών, και επακόλουθων σφαλμάτων, δεν αποφεύγεται λόγω των προτεραιοτήτων στις συγκρίσεις, αλλά γιατί το σύστημα έχει ρυθμιστεί να δίνει μηδενική έξοδο dataOut.

Α΄.4.7 Μηνύματα

Σε κάθε τύπο αρχείου γράφονται μηνύματα συγκεκριμένης μορφής και περιεχομένου.

.Qsi Περιέχουν τις γραμμές οδηγιών που παράγονται έμμεσα, με την Qs_directives(). Παρεμβάσεις μπορούν να γίνουν στο αρχείο, αφού είναι απλό κείμενο, και συνήθως συμβαίνει για την εισαγωγή του χαρακτήρα σχολιασμού, που η συνάρτηση δεν τοποθετεί.

.do Η μόνη δομική διαφορά των αρχείων .do που έχουν αντιγραφεί από αρχεία με έτοιμες εντολές force, από εκείνα που έχουν παραχθεί βάσει σεναρίου, είναι η πρώτη γραμμή. Τα μεν ενισχυμένα, αρχινούν με

#Apo to arxiko .do file : <arxiko.do> :

ενώ τα παραγόμενα, με την

#Apo to arxiko .Qsi file : <senario.Qsi> kai thn odhgia : <odhgia> ::

Και τα δύο όμως, συνεχίζουν με την ίδια δομή

<pre>
<forces>
<post>

Τα παραπάνω, είναι οι αρχικές εντολές, οι εντολές force, και οι τελικές εντολές. Οι αρχικές εντολές, <pre>, ορίζουν την παρακολούθηση των σημάτων, για να μπορέσουν να καταγραφούν αργότερα, ορίζουν μεταβλητές με τα σημαντικότερα μεγέθη, για να γίνεται η παραγωγή των εντολών στατικότερα, εντολές που βοηθούν τον υπολογισμό της ταχύτητας εκτέλεσης της προσομοίωσης (αναλυτικότερα Α΄.5.1), και επίσης, ορίζουν το σήμα του ρολογιού, και δίνουν αρχική μηδενική τιμή σε όλα τα άλλα σήματα εισόδου. Αναλυτικότερα είναι οι

add log sim:/<system_entity>/<sig_in(0)>
⋮
add log sim:/<system_entity>/<sig_in(n)>
add log sim:/<system_entity>/<sig_out(0)>
⋮
add log sim:/<system_entity>/<sig_out(n)>
##variables set
set one_cycle <senario_options.one_cycle>
set half_cycle <one_cycle/2>
set unal_sigs <(senario_options.unal_sigs)*(senario_options.one_cycle)/10>
set unal_res <(senario_options.unal_res)*(senario_options.one_cycle)/10>
set time_unit <senario_options.time_unit>

```
#arxh
set n_a $now ;
set ts_a [clock seconds];
set chan [open <test_path>/<seed_name>/<seed_name>.Qts
a];
puts $chan "(A) : edw_sim = $n_a , xronos = $ts_a , taxythta
= 0 $UserTimeUnit / sec.";
close $chan;
#taxythta initializing
when -label when_init_ends { /<system_entity>/initializing'event
and /<system_entity>/initializing='0'} {
if {$now != 0} {
set n_i $now ;
set ts_i [clock seconds];
set num_ui $n_i ;
set den_ui [expr ($ts_i- $ts_a)];
if {$den_ui==0} {set den_ui 1};
set chan [open <test_path>/<seed_name>/<seed_name>.Qts
a];
puts $chan "(I) : edw_sim = $n_i , xronos = $ts_i , taxythta
= [expr $num_ui/$den_ui] $UserTimeUnit / sec.";
close $chan;} }
#taxythta ekteleshs,
#c kyklous meta to init
set c <entoles_meta>;
set i_end <init_end>; #se kyklous
#set c_lim [expr ($i_end+ $c)*$one_cycle<conv_to_ps>] #o
typos
set c_high <c_lim_high_32>
set c_low <c_lim_low_32>
set c_lim [intToTime $c_high $c_low]
when -label when_after_i+c "\$now = $c_lim" {
set n_c $now;
set ts_c [clock seconds];
set num_uc [expr $n_c - $n_i];
set den_uc [expr $ts_c- $ts_i];
if {$den_uc==0} {set den_uc 1};
set chan [open <test_path>/<seed_name>/<seed_name>.Qts
a];
puts $chan "(C) : edw_sim = $n_c , xronos = $ts_c , taxythta
= [expr $num_uc/$den_uc] $UserTimeUnit / sec.";
close $chan;}
##pou pame xwris clk???
force clk 0 $half_cycle $time_unit, 1 $one_cycle $time_unit -
```

```
repeat $one_cycle $time_unit
##force @0
force /<system_entity>/<in_sig(0)> <zero> @0
⋮
force /<system_entity>/<in_sig(n)> <zero> @0
```

Οι μεταβλητές του τμήματος εντολών για την μέτρηση ταχύτητας εκτέλεσης, αντιστοιχούν στα

```
 entoles_meta = 100
init_end = 2 + <init_max>
c_lim_total = ( init_end + entoles_meta)*<senario_options.one_cycle>
/*υπολογισμένο σε ps, ό,τι time_unit και να έχει δοθεί*/
c_lim_64 = <c_lim_total> /*64 bits*/
c_lim_high_32 = <c_lim_64>  /*ανώτερα 32 bits*/
c_lim_low_32 = <c_lim_64>  /*κατώτερα 32 bits*/
```

Οι εντολές τέλους, <post>, βοηθούν τον υπολογισμό της μέσης ταχύτητας εκτέλεσης της προσομοίωσης, ορίζουν τους ρυθμούς δειγματοληψίας των σημάτων, την καταγραφή των τιμών των εισόδων και των εξόδων στα ανάλογα αρχεία, και τον τερματισμό της εκτέλεσης της προσομοίωσης και γενικότερα του περιβάλλοντος της. Αναλυτικά, είναι οι

```
#telos
set n_t $now ;
set ts_t [clock seconds];
set num_ut $n_t ;
set den_ut [expr $ts_t- $ts_a];
if {$den_ut==0} {set den_ut 1};
set chan [open <test_path>/<seed_name>/<seed_name>.Qts
a];
puts $chan "(T) : edw_sim = $n_t , xronos = $ts_t , taxythta
= [expr $num_ut/$den_ut] $UserTimeUnit / sec.";
close $chan;
configure list -delta none -strobeperiod "$one_cycle $time_unit"
-strobestart " [expr {$one_cycle + $unal_sigs}] $time_unit " -
usesignaltriggers {0} -usestrobe {1}
add list -hex sim:/<system_entity>/<sig_in(0)>
⋮
add list -hex sim:/<system_entity>/<sig_in(n)>
write list <test_path>/<seed_name>/<seed_name>.Qli
delete list *
configure list -delta none -strobeperiod "$one_cycle $time_unit"
-strobestart " [expr {$one_cycle + $half_cycle}] $time_unit "
```

-usesignaltriggers {0} -usestrobe {1}
add list -hex sim:/<system_entity>/<sig_out(0)>
⋮
add list -hex sim:/<system_entity>/<sig_out(m)>
write list <test_path>/<seed_name>/<seed_name>.Qlo
quit -sim
quit -f

Ειδικότερα, οι εντολές configure για την διαμόρφωση της προβολής List, είναι αναγκαίες μίας και η default καταγραφή είναι για κάθε αλλαγή τιμής του σήματος, σε κάθε κύκλο delta του προσομοιωτή. Κάτι τέτοιο δεν είναι επιθυμητό, και η εμφάνιση των σταθεροποιημένων τιμών ανά 'εντολή', απαιτεί δειγματοληψία των σημάτων σε συγκεκριμένα χρονικά σημεία. Αυτό είναι δυνατόν να οριστεί, και είναι ανά έναν κύκλο, όπως και η στιγμή έναρξης της δειγματοληψίας, που για τα σήματα εισόδου είναι η <one_cycle> + <unal_sigs> <time_unit>, που για τις default τιμές ισούται με 180 ns, ενώ για τα σήματα εξόδου, η έναρξη της δειγματοληψίας είναι στα <one_cycle> + <half_cycle> <time_unit>, δηλαδή, (default) 150 ns.

.Qs_migrated Κάθε φορά που το αρχικό αρχείο .do αντιγράφεται και ενισχύεται για να δώσει τις εισόδους μίας δοκιμής, γράφεται στο τέλος ενός αρχείου .Qs_migrated με το ίδιο όνομα, και στο ίδιο μονοπάτι αρχειοθέτησης με το αρχικό, μία γραμμή

To <arxiko>.do metafer8hke, me tis katallhles pros8hkes, sto <test_path>\<seed_name>\<seed_name>.do.

.Qs_created Αντίστοιχα, κάθε φορά που ένα αρχείο .Qsi διαβάζεται, και για κάθε μία γραμμή του που καθοδηγεί την παραγωγή ενός νέου .do, γράφεται στο τέλος ενός αρχείου .Qs_created με το ίδιο όνομα, και στο ίδιο μονοπάτι αρχειοθέτησης με το αρχικό, μία γραμμή

Apo thn odhgia : <line>, dhmiourgh8hke, me tis katallhles pros8hkes, to <test_path>\<seed_name>\<seed_name>.do.

.Qli Τα σήματα εισόδου καταγράφονται από το ModelSim στο αρχείο, μία εντολή, δηλαδή κύκλος, ανά γραμμή και σε στήλες που αντιστοιχούν στα

<time> <reset> <inid> <valid> <operation> <qi> <datain>

.Qlo Τα σήματα εξόδου καταγράφονται στο αρχείο από το ModelSim, μία εντολή, δηλαδή κύκλος, ανά γραμμή και σε στήλες που αντιστοιχούν στα

<time> <initializing> <outid> <done> <dataout> <qout>

.Qmo Το μοντέλο, καταγράφει τις εξόδους του, σε γραμμές της μορφής

<initalizing>___<outid>___<done>___<dataout>___<qout>

.Qfc Το αρχείο ξεκινά με την γραμμή

Parathrhseis ths <Behavioral/Time> prosomoiwshs :

Με error= ['E:Init', 'E:ID', 'E:done', 'E:data','E:value unknown', 'W:Qi', 'W:data', 'W:done'] να είναι η λίστα συμβολοσειρών των συμβάντων, και diff ο αριθμός διαφοράς που ενδεχομένως προέκυψε, για κάθε κύκλο που υπάρχει διαφορά, θα γραφτούν τα

<error[diff-1]>::
S: <simLine>
M: <mdLine>

Στο τέλος του αρχείου, είτε υπήρξαν διαφορές είτε όχι, θα σημειωθεί μέχρι ποιά outID έγινε σύγκριση, συμπεριλαμβάνοντας και την χρονική στιγμή από το αρχείο εξόδων του συστήματος

Sygkri8hkan mexri : sID(<s_time>) = <s_outid>, mID = <m_outid> .

Καμία από τις τιμές αυτές δεν είναι επαρκής για το μονοσήμαντο χαρακτηρισμό του σημείου. Το σήμα in/outID έχει περιορισμένο πλάτος, και είναι δυνατόν να επαναλαμβάνονται οι τιμές του, όπως και πράγματι συμβαίνει, όταν μεγιστοποιηθεί. Από την άλλη, η καταγραφή του χρόνου από την προβολή List παρουσιάζει μία ιδιομορφία: αν τα ψηφία της τιμής πρόκειται να ξεπεράσουν τα 10, το οποίο εύκολα συμβαίνει, σε μεγάλες δοκιμές και με την υποδιαίρεση του χρόνου στα ps, τότε κόβει το λιγότερο σημαντικό ψηφίο, και συνεχίζει με 10 ψηφία. Συγκεκριμένα, για κύκλο 100 ns, από τα 9,999,980,000 ps, δηλαδή την 99,999-ή εντολή, συνεχίζει την καταγραφή στα 1,000,008,000 ps, που είναι η 100,000-ή, αλλά θα φαινόταν (χωρίς το unalignement ειδικά) για 10,000-ή!

Τελικά, ανάλογα με τις συνολικές εμφανίσεις των συμβάντων, δηλαδή αν υπήρξαν κάποιες προειδοποιήσεις, με ή χωρίς σφάλματα, ή αν εμφανίστηκαν σφάλματα, ή αν δεν προέκυψε τίποτα από τα δύο, εμφανίζονται οι τις γραμμές

Warnings : <warn> mono! Kata ta alla...
Psyxraimia paidia! Einai mono <bugs> bug(s) kai <warn> warning(s)...
Einai ok!!!!! Kudos!

.Qts Στο αρχείο γράφουν όλα τα στάδια του Tester και συγκεντρώνει όλα τα στοιχεία της δοκιμής, εκτός των αποτελεσμάτων της σύγκρισης, σε ένα προοίμιο, λίγο διαφοροποιημένο ανάλογα με το αν οι είσοδοι αντιγράφηκαν ή κατασκευάστηκαν

/*από Qs_force_migrate */
Project: <prj_name>
Sim : <Behavioral/Timesim>
Forces.do : <arxiko.do>

/*από Qs_force_create */
Project : <prj_name>
Sim : <Behavioral/Timesim>
.Qsi : <senario.Qsi>
Odhgia : <odhgia>

Πριν από κάθε στάδιο, και στο τέλος του, γράφεται μία γραμμή

<status> : <seed_name>

Τα <status> λαμβάνουν διαδοχικά τις τιμές “Arxh prosomoiwshs”, “Telos prosomoiwshs”, “Arxh montelou”, “Telos montelou”, “Arxh syn8eshs <force>.do”, “Telos syn8eshs <force>.do”, (ή, αντίστοιχα για κατασκευή .do, “Arxh paragwghs <force>.do”, “Telos paragwghs <force>.do”,) “Arxh sygkrishs”, “Telos sygkrishs”. Αμέσως μετά την “Arxh prosomoiwshs” και πριν από τα υπόλοιπα, παρεμβάλλονται οι μετρήσεις ταχύτητας εκτέλεσης της προσομοίωσης

(A) : edw_sim = <now_arxh> , xronos = <ts_arxh> , taxythta = 0 <UserTimeUnit> / sec.
(I) : edw_sim = <now_init> , xronos = <ts_init> , taxythta = <u_init> <UserTimeUnit> / sec.
(C) : edw_sim = <now_c> , xronos = <ts_c> , taxythta = <u_c> <UserTimeUnit> / sec.
(T) : edw_sim = <now_t> , xronos = <ts_telous> , taxythta = <u_t> <UserTimeUnit> / sec.

seed_name Όπως προαναφέρθηκε (σελίδα 121), το seed_name δίνει μοναδικά ονόματα στους φακέλους των δοκιμών και τα αρχεία τους, μιάς και πρόκειται για ειδικά μορφοποιημένη συμβολοσειρά χρόνου, που περιλαμβάνει την πλήρη ημερομηνία και την ώρα της στιγμής που έλαβε τιμή. Εκτός από τα αρχεία της εκάστοτε δοκιμής, αρχικά τουλάχιστον, γιατί μπορεί και συνήθως μεταβάλλεται αυτό, ονομάζει και τα παραγόμενα αρχεία .pkl_Qd. Στα διάφορα μηνύματα, εμφανίζεται είτε ως όνομα αρχείου, στα .Qs_migrated, .Qs_created , είτε ως καθεαυτή χρονική επισήμανση, στα αρχεία .Qts, όπου λαμβάνει, βέβαια, και διάφορες τιμές για τις στιγμές που καταγράφει.

Α΄.5 Επεκτάσεις

Στη διαδικασία επαλήθευσης επιλέχθηκαν, χονδρικά, δύο διάρκειες προσομοίωσης, για μέτριας και μεγάλης έκτασης δοκιμές, των $3 \cdot 10^5$ και 10^6 εντολών

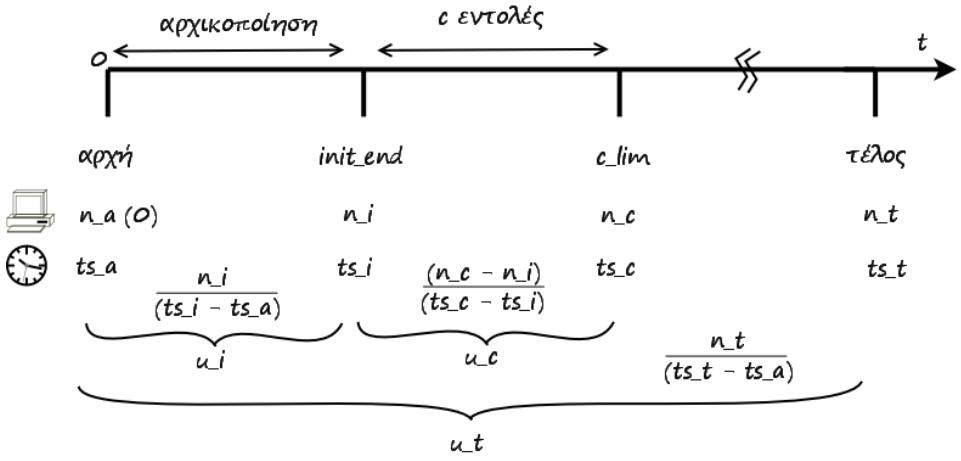
για Behavioral, και των $5 \cdot 10^3$ και $5 \cdot 10^4$ για Time προσομοίωση, αντίστοιχα. Στις μέτριες δοκιμές, με τα υπάρχοντα μηχανήματα, η εκτέλεση διαρκούσε κατά μέσο όρο μία ώρα, έχοντας ήδη εξασφαλίσει μία βελτίωση, ή εκλογίκευση, επιδόσεων με το σταδιακό γράψιμο των .do και τα ενδιάμεσα run. Στις μεγαλύτερης έκτασης δοκιμές, όπως προαναφέρθηκε (σελίδα 102), ακόμα και για τις συνθέσεις μνημών ελάχιστων διαστάσεων, απαιτήθηκε η επέκταση της εικονικής μνήμης στις μέγιστες, για το υποκείμενο λειτουργικό σύστημα και υλικό, επιτρεπόμενες τιμές. Οι συνθέσεις με τις μέγιστες διαστάσεις, για παράδειγμα η s7_MMM, φαινόταν απλά σαν να μην προχωρούν!

Το πλεονέκτημα της εκτέλεσης του προσομοιωτή στο παρασκήνιο, προς στιγμήν αποτέλεσε μειονέκτημα. Υπήρχε ανάγκη επικοινωνίας, κάποιας μορφής, της προόδου της εκτέλεσης. Το ModelSim επιτρέπει εποπτεία του τρέχοντος χρόνου προσομοίωσης, με τη μεταβλητή now, τόσο εμφανιζόμενη στο γραφικό περιβάλλον του, αλλά και άμεσα προσβάσιμη μέσω εντολών και κώδικα Tcl. Επίσης, διαθέτει εντολές περιβάλλοντος για παρακολούθηση συμβάντων στα σήματα. Έτσι, στις αρχικές και τελικές εντολές του .do, προστέθηκαν εντολές καταγραφής συγκεκριμένων χρονικών σημείων, και υπολογισμού της ταχύτητας εκτέλεσης της προσομοίωσης, με γνώση και του πλήθους εντολών που μεσολάβησαν. Από τις πρώτες μετρήσεις, έγιναν προβλέψεις για την διάρκεια εκτέλεσης των συνθέσεων με μέγιστες διαστάσεις για τις μεγάλης έκτασης δοκιμές. Πράγματι, παρέχοντας τον αναγκαίο χρόνο τελικά ολοκληρώθηκαν, εκτελώντας την ελαφρώς μικρότερη εκδοχή s2_EME, έχοντας αρχικά καταφύγει στις 'υβριδικές' b12_EaE, b11_Ea2E.

Μιάς και οι διάρκειες εκτέλεσης παρέμειναν αρκετά μεγάλες, μελετήθηκε επιπλέον το ενδεχόμενο βελτίωσης της απόδοσης με την σταδιακή καταγραφή και των τιμών των σημάτων στα αρχεία .Qli, .Qlo. Ακόμα, προστέθηκε ένα module για την εξαγωγή των αποτελεσμάτων των δοκιμών από τα αρχεία κειμένου στα οποία καταγράφονται, και συγκέντρωσή τους σε καταλληλότερη μορφή για μαζικό χειρισμό.

Α΄.5.1 Ταχύτητα

Η απλούστερη λύση, υπολογισμού μίας συνολικής ταχύτητας, απορρίπτεται, τουλάχιστον ως μοναδική, για δύο λόγους. Αφενός, ο υπολογισμός της ταχύτητας προέκυψε ως ανάγκη για την πρόβλεψη της διάρκειας εκτελέσεων που δεν διαφαινόταν η ολοκλήρωσή τους, άρα, είναι απαραίτητο να υπάρχουν καταγραφές νωρίτερα από το τέλος. Αφετέρου, είχε ήδη παρατηρηθεί ότι η ταχύτητα της εκτέλεσης δεν ήταν ίδια, ή ανάλογη, μεταξύ της κατάστασης αρχικοποίησης των μνημών και της κανονικής λειτουργίας του συστήματος. Συνεπώς, χρειάζεται ο υπολογισμός και των δύο ταχυτήτων, της αρχικοποίησης και της κανονικής εκτέλεσης, που καμία δεν αντιστοιχεί με την πραγματική ταχύτητα, μιάς και οι δύο φάσεις πάντα συνυπάρχουν. Για λόγους πληρότητας, ορίζονται και οι εντολές για τον υπολογισμό της μέσης ταχύτητας, αν η εκτέλεση ολοκληρωθεί.



Σχήμα Α΄.22: Σημεία μετρήσεων για υπολογισμό ταχυτήτων εκτέλεσης.

Για τον υπολογισμό της ταχύτητας προσομοίωσης, είναι αναγκαία η καταγραφή του χρόνου προσομοίωσης, που αποτελεί το μεταβαλλόμενο μέγεθος, και του πραγματικού χρόνου (wall clock time) μέσα στον οποίο υπήρξε η μεταβολή. Έτσι, η μονάδα ταχύτητας είναι χρόνος προσομοίωσης ανά πραγματικό χρόνο, και μιάς και η ανάλυση της προσομοίωσης ορίζεται (vsim -t ps ...) σε ps, ισούται με ps / sec. Τα σημεία καταγραφής των παραπάνω ταχυτήτων, αρχικοποίησης, κανονικής λειτουργίας και τελικής (μέσης), είναι η αρχή της εκτέλεσης, το πέρας της αρχικοποίησης, κάποιο σημείο προχωρημένης κανονικής εκτέλεσης, και το τέλος της εκτέλεσης. (Σχήμα Α΄.22)

Για την καταγραφή του χρόνου προσομοίωσης, αλλά και του πραγματικού χρόνου, χρησιμοποιήθηκε η εντολή περιβάλλοντος when [15], που επιτρέπει την εκτέλεση εντολών όταν συγκεκριμένες συνθήκες πληρούνται.

> when [[-label <label>] [-id <id#>] {<when_condition_expression>} {<command>}]

Η εντολή, εκτός από τα προαιρετικά στοιχεία αναγνώρισης της, label, id, αποτελείται από την συνθήκη, {<when_condition_expression>}, και το μπλόκ των εντολών {<command>}. Η γλώσσα της συνθήκης είναι μία περιορισμένη ψευδογλώσσα, που επιτρέπει μόνο συγκρίσεις ισότητας (==), διαφοράς (!=), ανισότητας (>, <, >=, <=), λογικό-και (&&, AND) και λογικό-ή (||, OR), με τελεστέους, αποκλειστικά και μόνο, σήματα, συμβάντα στα σήματα ('event) και σταθερές. Ο περιορισμός προκύπτει γιατί η συνθήκη αποτιμάται σε κάθε μεταβολή των τελεστέων. Αντίθετα, στο κομμάτι εντολών μπορεί να χρησιμοποιηθούν οποιεσδήποτε εντολές περιβάλλοντος και γλώσσας Tcl.

Τα σημεία της αρχής και του τέλους ανιχνεύονται άμεσα, χωρίς παρεμβολή εντολών when, εκτελώντας μόνο τις εντολές που περιλαμβάνονται, στις άλλες περιπτώσεις, στα αντίστοιχα μπλοκ. Με when ανιχνεύονται μόνο το σημείο

λήξης της αρχικοποίησης και το σημείο κανονικής εκτέλεσης, που ορίζεται να απέχει κάποιο σταθερό (c) πλήθος εντολών από το τέλος της αρχικοποίησης. Η συνθήκη εντοπίζει τα συγκεκριμένα σημεία, ανιχνεύοντας την αρνητική ακμή του σήματος initializing, και συγκρίνοντας την μεταβλητή now με μία υπολογισμένη σταθερά χρόνου (c_lim), αντίστοιχα.

Η σταθερά c_lim είναι το άθροισμα εντολών του προβλεπόμενου τέλους της default αρχικοποίησης, συν ένα σταθερό πλήθος c εντολών, μετασχηματισμένο σε χρόνο, ανάλυσης ps. Η ποσότητα προϋπολογίζεται από τον κώδικα python, και δίνεται έτοιμη στις εντολές (σελίδες 142, 144), γιατί το ModelSim εφαρμόζοντας[10] εσωτερική αναπαράσταση των αριθμών σε συμπλήρωμα ως προς 2 (2's complement), δίνει αρνητικά αποτελέσματα για μεγάλες τιμές, όπως εκείνες των συνθέσεων μνημών μεγίστων διαστάσεων (W_N16 bit). Διαθέτει, όμως, την εντολή intToTime, σύνθεσης δύο ποσοτήτων 32 bit σε μία των 64 bit, που είναι και το πλάτος των μεταβλητών χρόνου στον προσομοιωτή.

Μέσα στο μπλόκ εντολών καταγράφονται ο χρόνος προσομοίωσης, από την μεταβλητή περιβάλλοντος now, και ο πραγματικός χρόνος, με την εντολή Tcl [clock seconds]. Υπολογίζονται οι μεταβολές (διαφορές) και το κλάσμα της ταχύτητας, έχοντας πρώτα διορθώσει ενδεχόμενη εμφάνιση μηδέν στον παρονομαστή. Τελικά, ανοίγει το αρχείο .Qts, γράφονται οι μετρήσεις (σελίδα 147), και κλείνει, για να είναι διαθέσιμες, όσες έχουν γραφτεί, ακόμα και αν η εκτέλεση δεν προχωρήσει.

Από πέντε προσομοιώσεις κάθε είδους, Behavioral των 10^6 εντολών και Timesim $5 \cdot 10^4$ εντολών, για καθεμία από τις τέσσερις εκδοχές συνθέσεων μνημών, s0_EEE, b11_Ea2E, b12_EaE και s2_EME, συλλέχθηκαν, επεξεργάστηκαν και συγκρίθηκαν οι μετρήσεις ταχυτήτων και τα παράγωγά τους. (Σχήμα Α′.23)

Καθώς καμία από τις δύο βασικές ταχύτητες u_i, u_c, δεν είναι αρκετά αντιπροσωπευτική για το σύνολο της εκτέλεσης, μία ακόμη ταχύτητα, η u_m, δημιουργήθηκε από τον μέσο όρο τους, θεωρούμενες ήδη κανονικοποιημένες λόγω διαίρεσης. Οι τρεις αυτές, συγκρίθηκαν με την τελική (μέση) ταχύτητα, που αντιστοιχεί στην πραγματική ταχύτητα εκτέλεσης. (Σχήματα Α′.24 έως Α′.26).

Μιάς και η μονάδα ορισμού της διάρκειας προσομοίωσης είναι η εντολή, είναι διαισθητικότερο οι υπολογισμένες ταχύτητες, να εκφραστούν σε εντολές / sec. Καταγεγραμμένες στο .Qts σε ps / sec, μετατρέπονται διαιρώντας με το πλήθος ps που αντιστοιχεί σε έναν κύκλο, άρα και μία εντολή. Για κάθε υποκλάση δοκιμών, την οποία αποτελεί κάθε διαφορετική αρχιτεκτονική μίας εκδοχής του συστήματος, υπολογίζεται μία μέση τιμή για κάθε επιμέρους ταχύτητα. Από τα timestamps αρχής και τέλους της προσομοίωσης, υπολογίζεται η πραγματική διάρκεια της προσομοίωσης σε δευτερόλεπτα, και ο μέσος όρος αυτών, για κάθε υποκλάση. Προβλέψεις διάρκειας υπολογίζονται διαιρώντας το πλήθος των εντολών με την μέση ταχύτητα, και οι τέσσερις προβλέψεις, i, c, m και

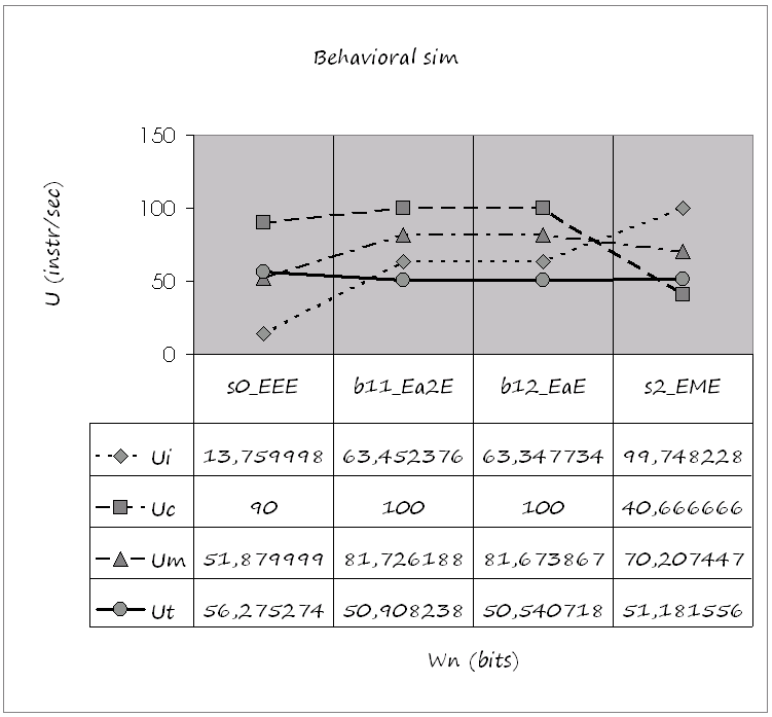
[10] αλλά, κατά περίπτωση!

		seed_name	Odhgia	Diarkeia_prosomoiwshs	διάρκεια προσ. (sec)	Ui (ps / sec)	Ui (i / sec)	i-περβλψ	Uc (ps / sec)	Uc (i / sec)	c-περβλψ	Um(ci) (i/sec)	m-περβλψ	Ut (ps / sec)	Ut (i / sec)	t-περβλψ
b11_Ea2E		b11_Ea2E	b11_Ea2E	b11_Ea2E	b11_Ea2E	b11_Ea2E	b11_Ea2E		b11_Ea2E	b11_Ea2E		11_Ea2E		b11_Ea2E	b11_Ea2E	
	Behavioral	06012010-005136	-s 1000000	4:46:09		7321428	73,21428	15759,85	10000000	100	10000,00	86,60714	12235,98	5851026	58,51026	19643,19
		06012010-134917	-s 1000000	6:06:33		4889952	48,80952		10000000	100		74,40476		4567248	45,67248	
		07012010-002446	-s 1000000	4:51:31	5,00	7321428	73,21428		10000000	100		86,60714		5742170	57,4217	
		18022010-194708	-s 1000000	5:59:45	32,00	4889952	48,80952		10000000	100		74,40476		4650292	46,50292	
		19022010-024105	-s 1000000	6:00:18	51,00	7321428	73,21428		10000000	100		86,60714		4643383	46,43383	
			1000000	5:32:51	14971,00	6345237,6	63,452376		10000000	100		81,726188		5090823,8	50,908238	
	Timdsim	10122009-143522	-s 50000	10:09:30		188092	1,88092	26312,05	188479	1,88479	31413,51	1,882855	28637,36	136811	1,36811	36510,63
		04012010-012255	-s 50000	10:09:01		198279	1,98279		118921	1,18921		1,586		137021	1,37021	
		04012010-211350	-s 50000	10:05:23	10,00	198279	1,98279		181625	1,81625		1,89452		137745	1,37745	
		17022010-141109	-s 50000	10:14:41	9,00	177047	1,77047		121822	1,21822		1,494345		135660	1,3566	
		18022010-022327	-s 50000	10:06:35	2,00	188438	1,88438		184989	1,84989		1,867135		137495	1,37495	
			50000	10:09:02	36542,00	190027	1,90027		159167,2	1,591672		1,745971		136946,4	1,369464	
b12_EaE		b12_EaE	b12_EaE	b12_EaE	b12_EaE	b12_EaE	b12_EaE		b12_EaE	b12_EaE		12_EaE		b12_EaE	b12_EaE	
	Behavioral	17122009-191645	-s 1000000	5:27:34		7880769	78,80769	15785,88	10000000	100	10000,00	89,403845	12243,82	5111685	51,11685	19786,03
		30122009-201838	-s 1000000	4:56:23		6209090	62,0909		10000000	100		81,04545		5648754	56,48754	
		31122009-022100	-s 1000000	5:55:47	5,00	4997560	49,9756		10000000	100		74,9878		4704327	47,04327	
		16022010-022933	-s 1000000	5:13:23	33,00	7588888	75,88888		10000000	100		87,94444		5306160	53,0616	
		16022010-234722	-s 1000000	6:13:18	41,00	4997560	49,9756		10000000	100		74,9878		4499433	44,99433	
			1000000	5:33:41	20021,00	6334773,4	63,347734		10000000	100		81,673867		5054071,8	50,540718	
	Timdsim	18122009-021236	-s 50000	10:55:37		175287	1,75287	27252,88	123323	1,23323	40342,17	1,49305	32515,95	127188	1,27188	38976,99
		11122009-025256	-s 50000	10:47:30		175287	1,75287		124865	1,24865		1,50076		128830	1,2883	
		29122009-231808	-s 50000	10:51:57	10,00	181176	1,81176		123323	1,23323		1,522495		127900	1,279	
		27012010-005517	-s 50000	10:49:44	50,00	193129	1,93129		124865	1,24865		1,58997		128360	1,2836	
		28012010-014146	-s 50000	10:45:44	6,00	193129	1,93129		123323	1,23323		1,58226		129126	1,29126	
			50000	10:50:06	34006,00	1836601,6	1,836016		123939,8	1,239398		1,537707		128280,8	1,282808	
s0_EEE		s0_EEE	s0_EEE	s0_EEE	s0_EEE	s0_EEE	s0_EEE		s0_EEE	s0_EEE		0_EEE		s0_EEE	s0_EEE	
	Behavioral	09012010-003111	-s 1000000	5:02:29		1290000	12,9	72674,43	10000000	100	11111,11	56,45	19275,25	5536786	55,36786	17769,74
		10012010-002806	-s 1000000	4:47:30		1290000	12,9		10000000	100		56,45		5824445	58,24445	
		11012010-000258	-s 1000000	4:49:09	4,00	1433333	14,33333		10000000	100		57,166665		5793065	57,93065	
		01032010-032306	-s 1000000	5:04:16	57,00	1433333	14,33333		5000000	50		32,166665		5505996	55,05996	
		26022010-182841	-s 1000000	5:05:46	50,00	1433333	14,33333		10000000	100		57,166665		5477345	54,77345	
			1000000	4:57:50	17870,00	1375998,8	13,75998		9000000	90		51,879999		5627527,4	56,275274	
	Timdsim	16122009-190449	-s 50000	9:12:18		134471	1,34471	35424,21	124884	1,24884	33571,56	1,246775	34473,01	150990	1,5099	32452,66
		10122009-012555	-s 50000	9:07:38		145047	1,45047		124884	1,24884		1,349655		152292	1,52292	
		07012010-150420	-s 50000	9:07:10	9,00	145047	1,45047		185013	1,85013		1,6503		152384	1,52384	
		19022010-162653	-s 50000	9:14:28	9,00	134471	1,34471		124884	1,24884		1,246775		150395	1,50395	
		25022010-022105	-s 50000	9:06:27	36,00	146696	1,46696		185013	1,85013		1,658545		152603	1,52603	
			50000	9:09:36	32976,00	141146,4	1,411464		148935,6	1,489356		1,45041		151732,8	1,517328	
s2_EME		s2_EME	s2_EME	s2_EME	s2_EME	s2_EME	s2_EME		s2_EME	s2_EME		2_EME		s2_EME	s2_EME	
	Behavioral	21012010-150826	-s 1000000	5:45:15		10673941	106,73941	10025,24	3333333	33,33333	24590,16	70,03637	14243,50	4847304	48,47304	19538,29
		22012010-001936	-s 1000000	4:50:21		11299655	112,99655		5000000	50		81,498275		5768337	57,68337	
		22012010-101038	-s 1000000	6:04:38	5,00	5759050	57,5905		5000000	50		53,79525		4586099	45,86099	
		23012010-010052	-s 1000000	5:02:14	24,00	11108135	111,08135		5000000	50		80,540675		5539853	55,39853	
		24012010-014551	-s 1000000	5:45:00	30,00	11033333	110,33333		2000000	20		65,166665		4849185	48,49185	
			1000000	5:29:30	19770,00	9974822,8	99,748228		4066666,6	40,666666		70,207447		5118155,6	51,181556	
	Timdsim	12012010-005718	-s 50000	19:05:09		82535	0,82535	61000,01	40942	0,40942	128026,23	0,617385	82625,64	72810	0,7281	69160,31
		14012010-004521	-s 50000	19:20:40		81616	0,81616		41281	0,41281		0,614485		71853	0,71853	
		14012010-233512	-s 50000	19:12:22	19,00	82002	0,82002		38721	0,38721		0,603615		72375	0,72375	
		16012010-014146	-s 50000	19:14:39	13,00	81798	0,81798		40609	0,40609		0,612035		72220	0,7222	
		17012010-021605	-s 50000	19:14:42	30,00	81885	0,81885		33750	0,3375		0,578175		72221	0,72221	
			50000	19:13:30	64210,00	81967,2	0,819672		39060,6	0,390606		0,605139		72295,8	0,722958	

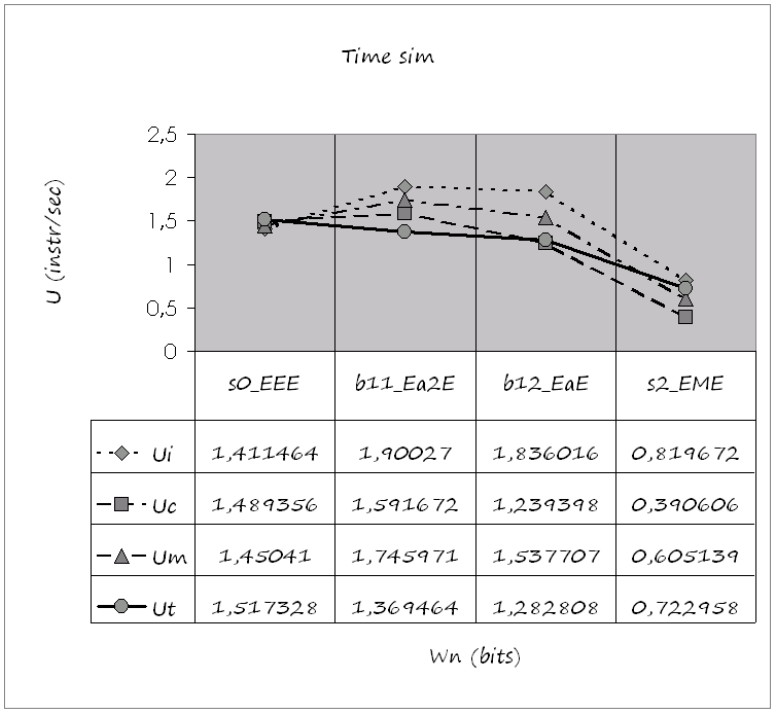
Σχήμα Α΄.23: Σύνολο μετρήσεων ταχύτητας και εκτιμήσεων.

	Wn	sim	Ui (i / sec)	Uc (i / sec)	Um(ci) (i/sec)	Ut (i / sec)
s0_EEE	8	B	13,759998	90	51,879999	56,2753
		T	1,411464	1,48936	1,45041	1,51733
b11_Ea2E	11	B	63,452376	100	81,726188	50,9082
		T	1,90027	1,59167	1,745971	1,36946
b12_EaE	12	B	63,347734	100	81,673867	50,5407
		T	1,8360016	1,2394	1,537707	1,28281
s2_EME	16	B	99,748228	40,6667	70,207447	51,1816
		T	0,819672	0,39061	0,605139	0,72296

Σχήμα Α΄.24: Τιμές ταχυτήτων προσομοίωσης.



Σχήμα Α′.25: Σύγκριση ταχυτήτων Behavioral προσομοίωσης.



Σχήμα Α′.26: Σύγκριση ταχυτήτων Time προσομοίωσης.

t μπορούν να συγκριθούν με την μέση πραγματική διάρκεια. (Σχήματα Α΄.27 έως Α΄.33).

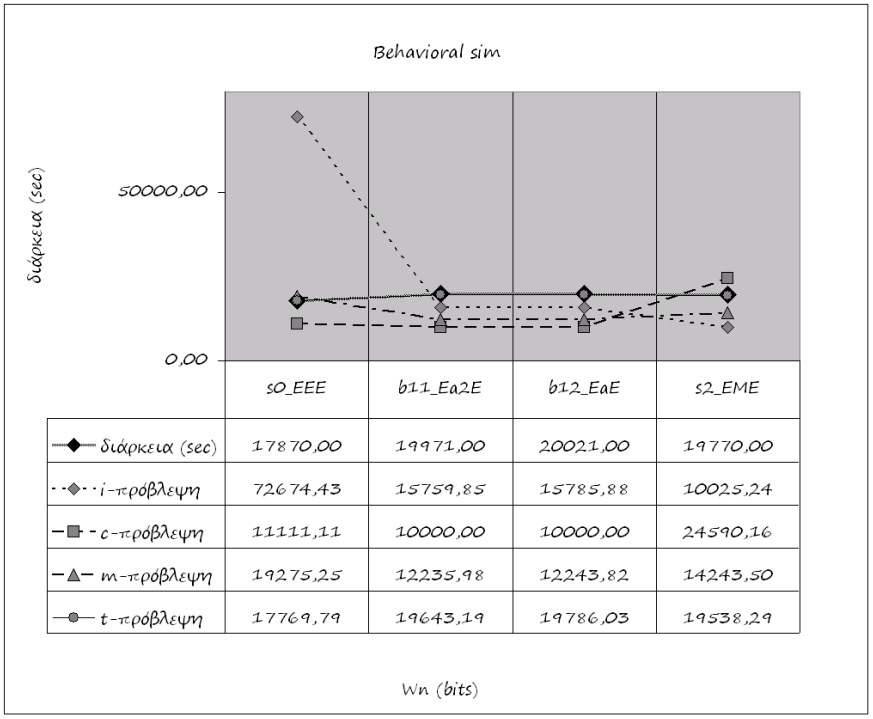
Καμία από τις ταχύτητες δεν εμφανίζεται ως γενικά καλύτερη προσέγγιση, και με διαφορετική συμπεριφορά ακόμα και μεταξύ των δύο τύπων προσομοίωσης, μπορούν να προσφέρουν μονάχα μία χονδρική αίσθηση, με τη χρήση και των τριών ταχυτήτων. Στις διάρκειες, ταύτιση εμφανίζεται μόνο με τις προβλέψεις της τελικής ταχύτητας, μιάς και δεν αποτελούν εκτιμήσεις, αλλά πραγματικά δεδομένα. Οι Time προσομοιώσεις εμφάνισαν καλύτερες προσεγγίσεις, κάτι που οφείλεται στις έντονες καθυστερήσεις και κατά την αρχικοποίηση, και όχι μόνο την κανονική λειτουργία, που απουσίαζαν στις Behavioral.

Η αποτυχία ικανοποιητικής προσέγγισης της πραγματικής ταχύτητας και διάρκειας από τις ταχύτητες αρχικοποίησης, κανονικής λειτουργίας, και ιδίως την μέση τιμή τους, μπορεί να αποδοθεί, εν μέρει, στο ότι η διάρκεια της προσομοίωσης δεν αποτελείται καθαρά από εκτέλεση αρχικοποίησης ή εντολών. Παρεμβάλλεται η τροφοδότηση εισόδων με τις εντολές force, ανά τακτά χρονικά διαστήματα. Οι 100 εντολές μετά την αρχικοποίηση ίσως να μην αρκούν για να επιβαρυνθεί η φυσική μνήμη σε βαθμό αντίστοιχο της μέσης κατάστασης της σε προχωρημένο σημείο εκτέλεσης, ώστε να ανακύψει ακριβέστερη εκτίμηση για την ταχύτητα της κανονικής λειτουργίας u_c. Εκτός αυτού, παρότι χρονοβόρες, με συνολική διάρκεια εκτέλεσης 363:39:26, και τα 354:40:34 αποκλειστικά από το στάδιο προσομοίωσης, οι πέντε δοκιμές ανά υποκλάση, ίσως δεν ήταν αρκετές για να εξαλείψουν ενδεχόμενες ιδιαιτερότητες των συνόλων εισόδων που εκτελέστηκαν. Βέβαια, επιλέχθηκε η default εκδοχή παραγωγής όλων των εισόδων, της μέγιστης τυχαιότητας και εμφανίσεων συμπτώσεων, καθώς από την δεδομένη υλοποίηση του Tester δεν μπορεί να κατασκευαστούν σύνολα που να θεωρούνται περισσότερο ισοδύναμα, εκτελούμενα σε διαφορετικές εκδοχές διαστασιοποίησης.

Α΄.5.2 Καταγραφή

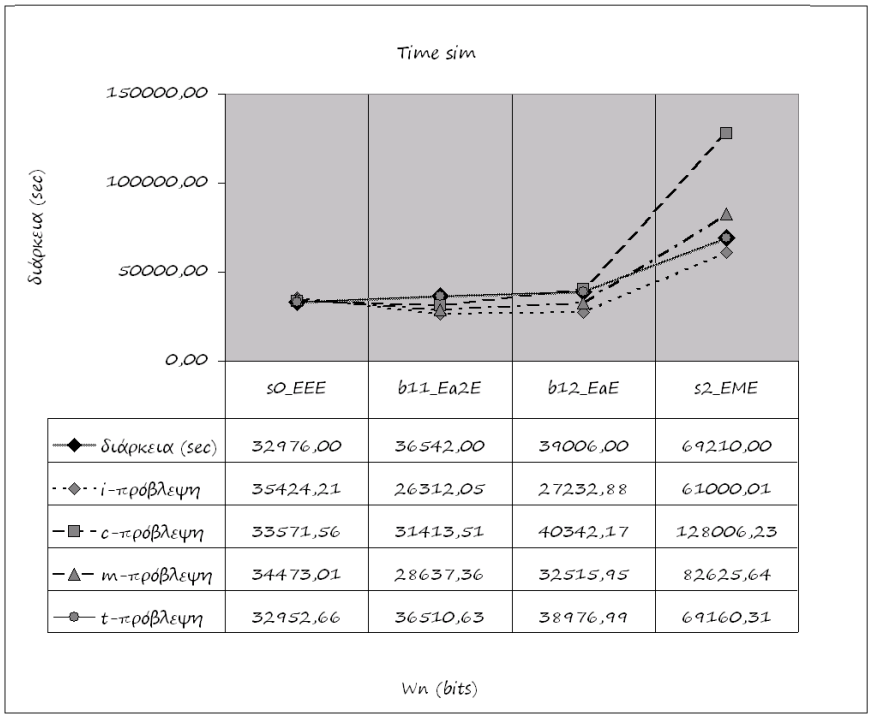
Σε κάθε προσομοίωση καταγράφονται επιλεγμένα δεδομένα σε ένα αρχείο .wlf (wave log format), με default όνομα vsim.wlf, το οποίο σε επόμενη εκτέλεση θα πανωγραφεί, αν δεν οριστεί διαφορετικά. Μπορεί να διατηρηθεί, και να χρησιμοποιηθεί για επισκόπηση σε μέλλοντα χρόνο, ορίζοντας διαφορετικό όνομα από το default στην εντολή vsim, με την επιλογή -wlf <filename>, είτε χρησιμοποιώντας τις εντολές dataset save ή dataset snapshot. Το σύνολο των δεδομένων που θα φορτωθούν για επισκόπηση από ένα .wlf αρχείο θα είναι ένα dataset, με όνομα ίδιο με το αρχείο, ή διαφορετικό, αν οριστεί κατά την φόρτωσή του. Το "sim:" είναι το όνομα της ενεργής προσομοίωσης, και μπορεί να συνυπάρξει με περισσότερα dataset, ή και να απουσιάζει. Η επιλογή των αντικειμένων που θα καταγραφούν γίνεται έμμεσα συμπεριλαμβάνοντας τα για επισκόπηση στις προβολές Ροής δεδομένων (add dataflow), Κυματομορφής (add wave) ή Λίστας (add list), ή άμεσα, με την εντολή add log.

Στον Tester, δεδομένης της καταγραφής των τιμών στα αρχεία .Qli/o, δεν



	s0_EEE	b11_Ea2E	b12_EaE	s2_EME
διάρκεια (sec)	17870,00	19971,00	20021,00	19770,00
i-πρόβλεψη	72674,43	15759,85	15785,88	10025,24
c-πρόβλεψη	11111,11	10000,00	10000,00	24590,16
m-πρόβλεψη	19275,25	12235,98	12243,82	14243,50
t-πρόβλεψη	17769,79	19643,19	19786,03	19538,29

Σχήμα Α′.27: Σύγκριση προβλέψεων διάρκειας Behavioral προσομοιώσεων.



	s0_EEE	b11_Ea2E	b12_EaE	s2_EME
διάρκεια (sec)	32976,00	36542,00	39006,00	69210,00
i-πρόβλεψη	35424,21	26312,05	27232,88	61000,01
c-πρόβλεψη	33571,56	31413,51	40342,17	128006,23
m-πρόβλεψη	34473,01	28637,36	32515,95	82625,64
t-πρόβλεψη	32952,66	36510,63	38976,99	69160,31

Σχήμα Α′.28: Σύγκριση προβλέψεων διάρκειας Time προσομοιώσεων.

	Wn	sim	διάρκεια (sec)	i-πρόβλεψη	c-πρόβλεψη	m-πρόβλεψη	t-πρόβλεψη
s0_EEE	8	B	17870,00	72674,43	11111,11	19275,25	17769,79
		T	32976,00	35424,21	33571,56	34473,01	32952,66
b11_Ea2	11	B	19971,00	15759,85	10000,00	12235,98	19643,19
		T	36542,00	26312,05	31413,51	28637,36	36510,63
b12_EaE	12	B	20021,00	15785,88	10000,00	12243,82	19786,03
		T	39006,00	27232,88	40342,17	32515,95	38976,99
s2_EME	16	B	19770,00	10025,24	24590,16	14243,50	19538,29
		T	69210,00	61000,01	128006,23	82625,64	69160,31

Σχήμα Α΄.29: Τιμές προβλέψεων διάρκειας προσομοιώσεων.

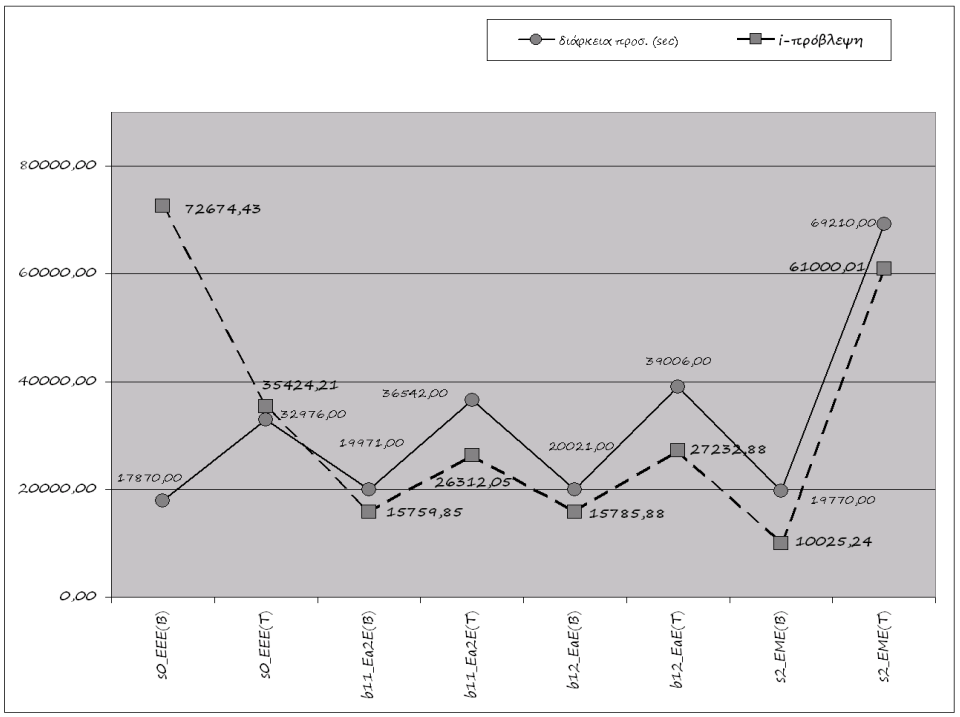
δόθηκε η επιλογή για την διατήρηση του αρχείου .wlf που προκύπτει πάλι σε κάθε εκτέλεση, στον αντίστοιχο φάκελο _sys, αλλά πανωγράφεται. Για την παραγωγή τους, χρησιμοποιείται η εντολή add log, για την μετέπειτα ανάθεση εισόδων και εξόδων σε διαφορετικές προβολές List ώστε να καταγραφούν σε διαφορετικά αρχεία. Η καταγραφή εκτελείται μονομιάς, μετά την ολοκλήρωση της προσομοίωσης, αν και δεν αποτελεί την μοναδική δυνατότητα. Κατ' αντιστοιχία της σταδιακής εκτέλεσης από εμβόλιμα run και της αποσπασματικής εγγραφής των αρχείων .do, εξετάστηκε η σταδιακή καταγραφή των τιμών των σημάτων ως σημείο πιθανής βελτίωσης της απόδοσης.

Έγινε μελέτη της επιλογής -wlftlim της εντολής vsim, για περιορισμό του μεγέθους του παραγόμενου αρχείου .wlf βάσει χρονικού ορίου, καταγράφοντας κάθε φορά τόσο χρόνο προς τα πίσω, όσο ορίζεται. Συμπεριλαμβάνοντας επαναλαμβανόμενα, ανάμεσα στις εντολές force, εντολές προβολής και καταγραφής List, σε αντιστοιχία με τα χρονικά διαστήματα που διατηρούνταν στο .wlf, παρατηρήθηκε βελτίωση στην χρήση της μνήμης, αλλά όχι και του χρόνου εκτέλεσης, όπως ήταν σκόπιμο. Ως ακατάλληλη λύση κρίνεται, εκτός της ανεπαρκούς απόδοσης, κυρίως γιατί δεν έγινε εφικτός ο καθορισμός μιάς χρονικής ποσότητας, ή μεθόδου, τέτοιας ώστε να αποφεύγεται μία κάποια απώλεια πληροφορίας –ελάχιστη, αλλά υπαρκτή και συνεπώς απαγορευτική.

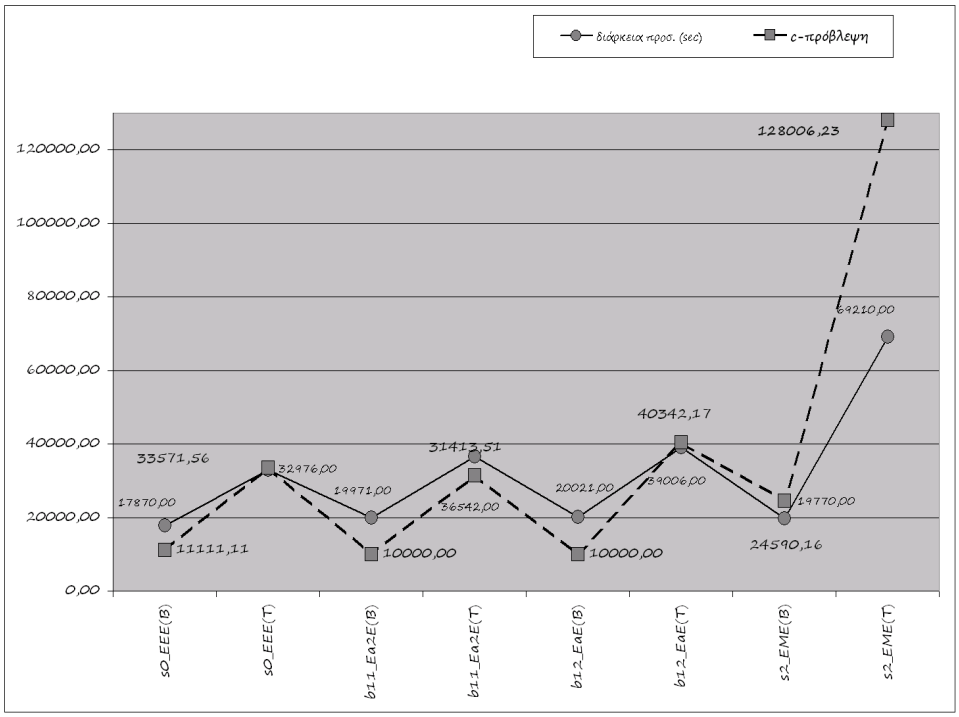
Δοκιμάστηκε και η εντολή dataset snapshot, η οποία στην μορφή

> dataset snapshot -dir <snap_dir_path> -file <filename>.wlf -filemode increment -mode sequential -time <snap_time> <time_unit>

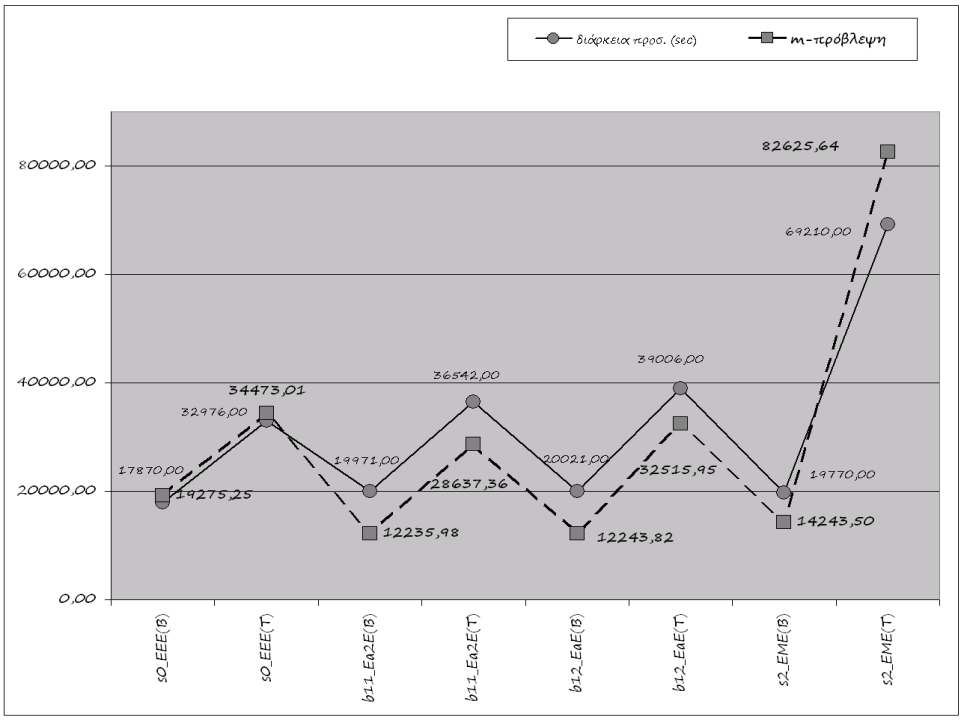
μπορεί να γράψει ανά ορισμένο χρονικό διάστημα (<snap_time>), αρχεία .wlf, ονομασμένα <filename> και με επίθεμα έναν αύξοντα αριθμό (increment), αδειάζοντας τα τρέχοντα δεδομένα μετά την καταγραφή (sequential). Πράγματι, με συμπληρωματικό κώδικα Tcl και μακροεντολών, μετά την εκτέλεση, τα επιμέρους .wlf φορτώνονταν, τα δεδομένα τους περιλαμβάνονταν, κατά περίπτωση, σε προβολή List και καταγράφονταν στα .Qli/o. Δυστυχώς, η εντολή write



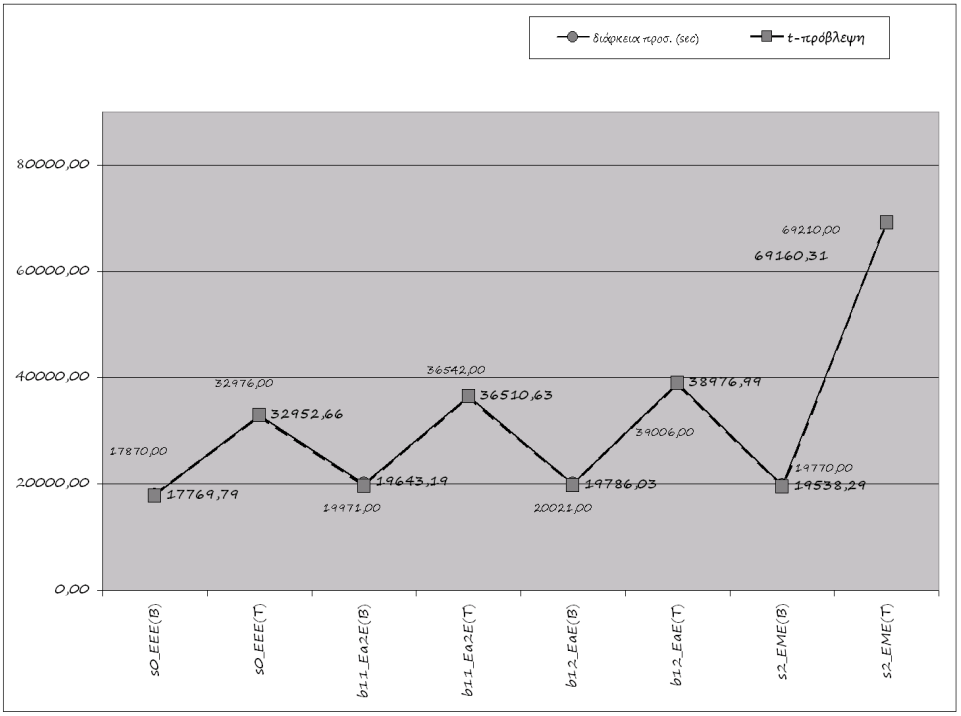
Σχήμα Α΄.30: Προβλέψεις διάρκειας βάσει ταχύτητας αρχικοποίησης u_i .



Σχήμα Α΄.31: Προβλέψεις διάρκειας βάσει κανονικής ταχύτητας u_c .



Σχήμα Α΄.32: Προβλέψεις διάρκειας βάσει μεικτής ταχύτητας u_m .



Σχήμα Α΄.33: Προβλέψεις διάρκειας βάσει τελικής (μέσης) ταχύτητας u_t .

list δεν γράφει σε αρχείο με επισύναψη στο τέλος (append mode), ακόμα κι αν πρόκειται για ήδη ανοιγμένο με αυτό τον τρόπο μέσω Tcl, άρα θα πρέπει να κατασκευαστούν ενδιάμεσες καταγραφές σε προσωρινά .Qli/o, και να μετεγγραφούν στην συνέχεια στα δύο συνολικά αρχεία. Έτσι, η κανονική καταγραφή βρέθηκε να υπερτερεί της σταδιακής, τόσο σε χρόνο, όσο και χρήση μνήμης.

Α΄.5.3 Αποτελέσματα

Κάθε δοκιμή μετά την ολοκλήρωσή της περιελάμβανε στον, αποκλειστικής χρήσης, φάκελό της, ένα σύνολο αρχείων κειμένου. Τα .Qli, .Qlo, .Qmo, ενδιάμεσα ως προς την παραγωγή και καθαρά βοηθητικά στη χρήση, και τα .do, .Qfc, .Qts, που συγκεντρώνουν τις πληροφορίες χαρακτηριστικών και αποτελεσμάτων της δοκιμής. Όσα τυχόν .Qsi, .Qs_migrated, .Qs_created, .pkl_Qd αφορούν κάποια δοκιμή, δεν περιλαμβάνονται στο seed-named φάκελό της, και οι βασικές τους πληροφορίες καταγράφονται και στα αρχεία αποτελεσμάτων. Τα αρχεία σε μορφή κειμένου, εξυπηρετούν μεν την επισκόπηση των αποτελεσμάτων αμέσως μετά την λήξη της εκάστοτε δοκιμής, αλλά δεν προσφέρονται για επεξεργασία και εποπτεία των αποτελεσμάτων και μεγεθών των δοκιμών συγκεντρωτικά. Ο κάθε φάκελος συμπιεσμένος ανεξάρτητα, για μεσοπρόθεσμη διατήρηση, λόγω του σημαντικού μεγέθους των παραγόμενων αρχείων, δεν είναι καν πρακτικό να αποσυμπιεστεί. Έτσι, για την αυτόματη αποσυμπίεση και εξαγωγή των σημαντικών πληροφοριών από τα αρχεία μέσα στο .zip κάθε δοκιμής, και προσθήκη τους σε ένα συγκεντρωτικό αρχείο .csv (comma separated values), επεξεργάσιμο από πρόγραμμα λογιστικών φύλλων (spreadsheet), δημιουργήθηκε το συμπληρωματικό module Qs_values.py, με τις εξής συναρτήσεις:

Qs_values() Η συνάρτηση μπορεί να κληθεί από τη γραμμή εντολών, και να προετοιμάσει, βάσει επιλογών, την κλήση της συνάρτησης που πραγματοποιεί την εξόρυξη των πληροφοριών, Qs_zipped_to_csv(). Οι επιλογές είναι τέσσερις: η -f <f.zip> για φόρτωση ενός μεμονωμένου συμπιεσμένου αρχείου, η -c <zcat_path> για φόρτωση όσων .zip περιλαμβάνει ο ορισμένος φάκελος, η -v <v.csv> για υπόδειξη ήδη υπάρχοντος αρχείου για συμπλήρωση των νέων τιμών, και η -x για την απενεργοποίηση του μηχανήματος μετά την ολοκλήρωση της εκτέλεσης. Η επιλογή του μεμονωμένου .zip ελέγχεται μόνο ως προς την ύπαρξη και την συμφωνία της επέκτασης του, ενώ στου φακέλου, αν είναι υπαρκτός, το αν περιέχει κατάλληλου είδους αρχεία, τα οποία θα προστεθούν στην ίδια λίστα με το ενδεχόμενο μεμονωμένο, καθώς οι επιλογές είναι επικαλυπτόμενες. Αν δεν είναι έγκυρος φάκελος, ή δεν περιέχει συμπιεσμένα αρχεία, ακόμα και αν δόθηκε κάποιο μεμονωμένο, η εκτέλεση θα τερματιστεί, όπως και αν δεν προέκυψε κάποιο αρχείο, με οποιονδήποτε τρόπο. Αν έχει δοθεί κάποιο αρχείο .csv, για αποθήκευση των νέων τιμών, ελέγχεται μόνο η ύπαρξη και η επέκτασή του, ενώ αν δεν έχει δοθεί, ζητάται μονοπάτι, και

όνομα, για την δημιουργία νέου. Για κάθε αρχείο .zip που ορίστηκε, καλείται η Qs_zipped_to_csv(), εντός της οποίας πραγματοποιείται η δημιουργία του νέου .csv, ή για ήδη υπαρκτό, επιπλέον έλεγχος της εσωτερικής δομής του. Ασυμφωνία με την επιθυμητή προκαλεί ματαίωση της επεξεργασίας, που όμως, δεν θα υπερσκελίσει την επιλογή απενεργοποίησης.

Qs_zipped_to_csv() Γενικά, σε ένα .csv, κάθε γραμμή του πίνακα είναι μία γραμμή κειμένου, χωρισμένη από την επόμενη με τον κατάλληλο χαρακτήρα νέας γραμμής. Οι οσεσδήποτε στήλες, εντός κάθε γραμμής, οριοθετούνται από τον χαρακτήρα ";", είτε έχουν, είτε όχι τιμή. Συγκεκριμένα, λοιπόν, συγκεντρώνονται αρχικά σε μία λίστα τα ονόματα των στηλών του πίνακα τιμών, και κατασκευάζεται μία γραμμή, που αποτελεί τον τίτλο, με μόνη διαφορά από τις υπόλοιπες, ότι θα προηγείται, και αντί για αριθμούς, αποτελείται από συμβολοσειρές. Αν δεν είχε δοθεί ήδη υπάρχον .csv, το νέο αρχείο δημιουργείται, και γράφεται η παραπάνω πρώτη του γραμμή, διαφορετικά, το αρχείο ανοίγεται, και η πρώτη του γραμμή διαβάζεται. Αν ο υπάρχων τίτλος διαφέρει από τον τρέχοντα, η συνάρτηση επιστρέφει, επισημαίνοντας την ασυμφωνία. Στη συνέχεια, ανοίγεται το .zip, και λαμβάνεται μία λίστα με τα περιεχόμενά σε αυτό αρχεία. Επίσης, λαμβάνονται οι πρώτες τιμές, από τα στοιχεία του ίδιου του αρχείου: η διαδρομή του, το χρονικό όνομά του (seed-name), το οποίο μετατρέπεται και σε δευτερόλεπτα από κάποιο χρονικό σημείο αναφοράς, για διευκόλυνση της χρονολογικής ταξινόμησης, και αποθηκεύονται στο csv_dict, μία δομή λεξικού, με κλειδιά τα ονόματα των αντίστοιχων στηλών.

Ανοίγονται τα αρχεία που περιέχουν τις πληροφορίες που χαρακτηρίζουν την κάθε δοκιμή, διαβάζονται οι γραμμές που ενδιαφέρουν, και συμπεριλαμβάνονται σε μία ενιαία λίστα. Στο .do ενδιαφέρει μόνο η πρώτη γραμμή από την αρχή, στο .Qfc, η πρώτη από την αρχή, και οι τρείς από το τέλος, και μόνο στο .Qts, όλες. Γενικά, τα αρχεία που έχουν απλώς ανοιχτεί και όχι εξαχθεί από ένα συμπιεσμένο αρχείο, από την βιβλιοθήκη zipfile της python, είναι στιγμιότυπα της ZipExtFile, που δεν διαθέτουν μέθοδο για άμεση μετάβαση σε κάποιο σημείο τους (seek). Ειδικά για το .Qfc, οι αναγκαίες τελευταίες γραμμές ανακτώνται με χρήση ενός amortizer, βάσει μίας Ουράς περιορισμένου μεγέθους, ίσο με 3. Εισάγοντας κάθε νέα γραμμή στο τέλος της, διαγράφεται αυτομάτως από την αρχή της η παλιότερη, διατηρώντας τις τρεις τελευταίες μετά το πέρας της ανάγνωσης.

Η εξαγωγή των τιμών από τις γραμμές κειμένου, δεδομένου ότι πρόκειται για ένα γνωστό, ως προς τη σύνταξή της καθεμίας, σύνολο συμβολοσειρών γίνεται με αντιπαράθεση μοτίβων (pattern matching) βάσει κανονικών εκφράσεων (regular expressions). Αντίστοιχα με τις γνωστές φόρμες σύνταξης των αναμενόμενων μηνυμάτων, σχηματίζεται ένα σύνολο από μοτίβα, που αφήνουν προς ταύτιση τις ζητούμενες τιμές, αλλά είναι κατά τα άλλα, σχεδόν πανομοιότυπο με το αρχικό, κατά το δυνατόν. Η λογική για το σκανάρισμα είναι κάθε μοτίβο, ή μάλλον, κάθε σαρωτής (scanner) που προκύπτει από την μετάφρασή

του, καθώς αντιστοιχεί σε ένα ή περιορισμένου πλήθους μηνύματα της ίδιας μορφής, να επιλέγεται άμεσα για την γραμμή μηνύματος για το οποίο προορίζεται, μέσω μιάς δομής λεξικού (dictionary) που για κλειδιά έχει τα δύο πρώτα γράμματα κάθε μηνύματος, τα οποία είναι μοναδικά. Έτσι, χρησιμοποιείται το li_sc, που, μετά την κατασκευή του, για την ταύτιση κάθε γραμμής, δεν χρειάζεται περαιτέρω ελέγχους. Κάθε γραμμή, με κλειδί τα δύο πρώτα της ψηφία, θα σκαναριστεί από τον δικό της σαρωτή, δίνοντας ένα αντικείμενο, που σε περίπτωση ταύτισης είναι διαφορετικό του None, και καταχωρείται σε μία λίστα matched.

Το module βιβλιοθήκης της python για την αναγνώριση κανονικών εκφράσεων (re), δίνει την δυνατότητα της ονομάτισης κάθε (υπο)μοτίβου, και από το αντικείμενο που προκύπτει για κάθε επιτυχή ταύτιση, μπορεί να εξαχθεί μία δομή λεξικού, έχοντας τα δοθέντα ονόματα για κλειδιά, και τις συμβολοσειρές που ταυτίστηκαν ως τιμές. Τα υπομοτίβα που ονοματίστηκαν στα μοτίβα των γραμμών είναι ακριβώς τα κομμάτια των αποτελεσμάτων που ενδιαφέρουν, και με ονόματα αντίστοιχα των στηλών που θα τοποθετηθούν. Έτσι, εξάγεται από το αντικείμενο ταύτισης το λεξικό και, είτε αυτούσια, είτε μετά από κάποια μορφοποίηση, για περιορισμένες περιπτώσεις, ενημερώνεται το συνολικό, csv_dict, με τα νέα ζεύγη κλειδιών και τιμών.

Χρησιμοποιώντας μία υπολίστα των ονομάτων των στηλών, με τις ομάδες 3 στηλών, αρχή-τέλος-διάρκεια, ανά στάδιο εκτέλεσης, υπολογίζονται οι διάρκειες των σταδίων όταν έχουν και οι δύο άλλες τιμές καταγραφεί, και συμπληρώνονται στο csv_dict. Τέλος, χρησιμοποιώντας ως κλειδιά την συνολική λίστα ονομάτων των στηλών, για να προκύπτουν με συγκεκριμένη διάταξη, λαμβάνονται οι τιμές που ανακτήθηκαν στο csv_dict και σχηματίζεται η γραμμή με τα ενδιάμεσα ";", και αποθηκεύεται στο επιλεγμένο αρχείο .csv.

Qs_timestamp_duration() Το module time, δίνει την δυνατότητα μορφοποίησης μίας δομής struct_time, επιστρεφόμενης (μεταξύ άλλων) από την συνάρτηση localtime(), παράγοντας επιθυμητές συμβολοσειρές χρόνου ορισμένες από συγκεκριμένα σύμβολα μορφοποίησης. Η διαδικασία είναι και αντιστρέψιμη, δηλαδή, δίνοντας την συμβολοσειρά της χρονικής καταγραφής, και εκείνη που περιέγραφε την μορφοποίησή της, επιστρέφεται δομή ανάλογη του αρχικού τύπου, που με την σειρά της είναι περαιτέρω επεξεργάσιμη. Έτσι, τα timestamps που διαβάζονται, επαναφέρονται στην αρχική τους δομή, και εκείνη μετατρέπεται σε δευτερόλεπτα, από ένα χρονικό σημείο αναφοράς. Αναγνωρίζοντας κάποιο ως αρχική και άλλο ως τελική τιμή, υπολογίζεται, με την διαφορά τους, η μεταξύ τους χρονική διάρκεια, που αντιστοιχεί σε κάποιο στάδιο εκτέλεσης του Tester, αναγόμενη σε μεγαλύτερες υποδιαιρέσεις (Ω:Λ:Δ).

Qs_all_dirs.py Βοηθητικό module για χρήση μαζί με το Qs_values, το οποίο δεν εκτελεί διάσχιση των καταλόγων που δίνονται σαν ορίσματά του, αγνοώντας τους εσωτερικούς τους υποφακέλους. Έτσι, λαμβάνεται η εσωτερική

δομή ορισμένου φακέλου εξαντλητικά, ώστε, αν είναι σκόπιμο, να δοθούν οι επιμέρους υποφάκελοι για επεξεργασία. Η συνάρτηση Qs_Bf_dir_search(dir_path) εκτελεί αναδρομική διάσχιση σε πλάτος του δέντρου των υποφακέλων του αρχικού φακέλου-ορίσματος, ενώ η Qs_Df_dir_search(dir_path) αναδρομική διάσχιση σε βάθος. Η ομώνυμη συνάρτηση, Qs_all_dirs(), την διαδραστική επιλογή από τη γραμμή εντολών του αρχικού φακέλου και είδους διάσχισης, και την εκτύπωση των αποτελεσμάτων. Η ανακατεύθυνση της εξόδου (>) έχει ως αποτέλεσμα την εμφάνιση των προτρεπτικών μηνυμάτων στην νέα ροή, αν και οι είσοδοι εξακολουθούν να αναμένονται και να λαμβάνονται, ως συνήθως, από την γραμμή εντολών.

Α΄.6 Διορθώσεις

Από την ολοκλήρωση της λειτουργικότητας του Tester και έπειτα, δηλαδή κατά τις εκτελέσεις των δοκιμών, αλλά και την καταγραφή της εργασίας στο παρόν κείμενο, διακόπηκε η εξέλιξή του, όπως είχε και του συστήματος, νωρίτερα. Σε καμία περίπτωση δεν θεωρήθηκαν πλήρη ή αλάνθαστα, και η παρατεινόμενη μελέτη τους με σκοπό την περιγραφή, ανέδειξε, ή επανέφερε, κάποια σημεία όπου η σχεδίαση και υλοποίηση υπήρξαν πλημμελείς. Οι σχεδιαστικές ροές επιτάσσουν αναδρομή σε προηγούμενα σχεδιαστικά στάδια, για εφαρμογή των παρατηρήσεων και διορθώσεων. Σε πραγματικές συνθήκες, ίσως ιδανικές, η ανάπτυξη θα συνεχιζόταν αξιοποιώντας τις παρακάτω αναφορές, των σημαντικότερων από όσα παρατηρήθηκαν.

Μορφή κώδικα Η μορφή του κώδικα, δεν ήταν πάντοτε σύμφωνη με τις προδιαγραφές, ή εκμεταλλευόμενη πλήρως τις δυνατότητες των γλωσσών ανάπτυξης, VHDL και python.

Καταρχάς, τα διαφορετικά projects των εκδοχών της σχεδίασης, θα έπρεπε να είναι διαφορετικές αρχιτεκτονικές (architecture) της ίδιας οντότητας (entity). Επίσης, η επαναχρησιμοποίηση του κώδικα, θα ενισχύονταν από την χρήση configurations, όμως, δεν υποστηρίζονται επαρκώς από το Xilinx ISE.

Συγκεκριμένα, βάσει και των εκτεταμένων δοκιμών στο gIwta project, στο ModelSim μόνο θα ήταν δυνατή η χρήση configuration declaration για πλήρη διαχωρισμό των αρχείων που διαστασιοποιούν το σύστημα, έχοντας σταθερό και το τμήμα κώδικα της αρχιτεκτονικής, για τις ποικίλες διαστασιοποιήσεις. Τα ανεξάρτητα μεγέθη από το libpack θα έπρεπε να είναι generic στο entity του dummy_datapath, και τα εξαρτώμενα να ορίζονται στο τμήμα δηλώσεων (declarative part) της αρχιτεκτονικής. Με μία οντότητα ως 'περιτύλιγμα' (wrapper) που θα περιείχε το κανονικό dummy_datapath ως συστατικό υποσύστημα component, θα δινόταν οι επιθυμητές διαστάσεις σε ένα configuration declaration, όπως επίσης, και το ουσιώδες της διαστασιοποίησης, οι διαφορετικού μεγέθους οντότητες (entities) για τα components των μνημών.

Για την περίπτωση της Xilinx, το ISE δεν υποστηρίζει το configuration

declaration, παρόλο που είναι βασική δομή της VHDL, παρά μόνο τα configuration specification, που δηλώνονται μέσα στο τμήμα δηλώσεων μίας αρχιτεκτονικής. Θα μπορούσε έτσι, να ορίζεται μέσα στο wrapper entity το generic map του κανονικού dummy_datapath, και να επιλέγεται η επιθυμητή αρχιτεκτονική του, αλλά μέχρι εκεί. Το configuration specification δεν μπορεί να υποστηρίζει ρυθμίσεις σε ιεραρχική ανάπτυξη, όπως το declaration, συνεπώς δεν θα μπορούσαν να επιλεγούν οι επιθυμητές οντότητες για τις μνήμες, χωρίς παρέμβαση στην αρχική αρχιτεκτονική -όπως και γίνεται, χωρίς την χρήση του wrapper, αφού η χρήση του δεν είναι αναγκαία, έχοντας δηλώσει όλες τις εξαρτώμενες και ανεξάρτητες παραμέτρους στο package libpack.

Ακόμα, η ονοματολογία στο σύστημα, έγινε με μερική χρήση κεφαλαίων, για διαχωρισμό, αντί αποκλειστικά '_', παρότι στην VHDL δεν υπάρχει διάκριση πεζών-κεφαλαίων. Αντίστοιχα, στην python, αγνοήθηκε το οντοκεντρικό μοντέλο ανάπτυξης, λόγω της επιτρεπόμενης χαλαρότητας στο θέμα αυτό, αλλά και δεν αξιοποιήθηκαν κάποιες εγγενείς δυνατότητες της γλώσσας, όπως για παράδειγμα, το with για άνοιγμα (και κλείσιμο) αρχείων, ή το in για το χειρισμό λίστας. Σημαντική, επίσης, για τον ενδιαφερόμενο αναγνώστη, ήταν η διατήρηση παρωχημένων σχολίων, και στο σύστημα, αλλά κυρίως στα docstrings του Tester, καθιστώντας το κείμενο την μόνη αξιόπιστη πηγή τεκμηρίωσης.

Παραλείψεις Σε αρκετά, δυστυχώς, σημεία, η σχεδίαση ήταν κοντόφθαλμη, ώστε υλοποιούμενη, ναι μεν δεν εισάγει σφάλματα, αλλά επιβαρύνει αδικαιολόγητα τις επιδόσεις, την χρηστικότητα, ακόμα και την κομψότητα του προγράμματος.

Χαρακτηριστικότερη όλων, η περίπτωση της FreeList, με αποτελέσματα δυσανάλογα πενιχρά, σε σχέση με την προσπάθεια που επενδύθηκε στην σχεδίαση και υλοποίησή της. Παρότι νωρίτερα αναφέρεται το αντίθετο (4.3), τελικά η χρήση fsm για τον έλεγχο των διαθέσιμων στοιχείων, δεν επιβάλλεται από τίποτε παραπάνω από την επιλογή της αρχικοποίησης σε αύξουσα διάταξη. Ακόμα και για απόδοση αυθαίρετου ποσοστού των θέσεων, στην φθίνουσα διάταξη αρκεί η αρχικοποίηση του head_register με την συγκεκριμένη τιμή αντί για 1, από το head_init, και έχοντας τη διεύθυνση μηδέν, στοιχείο που δεν αποδίδεται ποτέ, ως την σταθερή ένδειξη αδειάσματος. Το γέμισμα, καλώς εχόντων των πραγμάτων, σηματοδοτείται από τις άδειες ουρές στο σύστημα. Μάλιστα, για την μεταβολή σε φθίνουσα διάταξη, δεν χρειάζεται παρά να αντιστραφούν οι τιμές μεταξύ των διευθύνσεων και των δεδομένων κατά την αρχικοποίηση. Επίσης, η ύπαρξη δύο διαφορετικών σημάτων index για NextMem και Head/TailMem ξεχωριστά, είναι κι αυτή πλεονάζουσα.

Η υλοποίηση του συστήματος επιβαρύνεται από την χρήση δομών διακλάδωσης με συνθήκη if-then αντί για case, που στην σύνθεση μετατρέπονται σε κυκλώματα προτεραιότητας, αντί για (ταχύτερους) πολυπλέκτες, αντίστοιχα. Επίσης, διατηρήθηκε η χρήση του ιδιότυπου καταχωρητή όπως σχεδιάστηκε, έχοντας, το συνήθως προς αποφυγήν, ασύγχρονο reset, ενώ για τις μνήμες,

αγνοήθηκαν πρότυπες δομές βιβλιοθήκης (primitives) της Xilinx, με εγγενή την ανταπόκριση σε αρνητική ακμή ρολογιού.

Άκομψα και πλεονασματικά στοιχεία, παρατηρήθηκαν και στην ανάπτυξη του Tester. Για παράδειγμα, εκείνο που η μεταβλητή updated σηματοδοτεί στη συνάρτηση Qs_dict_populate(), θα μπορούσε να κωδικοποιηθεί από τον τύπο του άλλου ορίσματος (None έναντι dict), κι ακόμα, η τοπική συνάρτηση p(str) μέσα στην Qs_dict_has_necessary(), με λειτουργικότητα που ήδη καλύπτεται από την Qs_echo(). Στις αρχικές εντολές, στην Qs_pre_forces(), έχουν διατηρηθεί οι εντολές αρχικοποίησης (set) των μεταβλητών c, i_end, παρότι δεν θα χρησιμοποιηθούν. Οι συναρτήσεις Qs_force_create() και Qs_force_migrate() θα μπορούσαν, με μικρή παρέμβαση, να είναι μία ενιαία, όπως και η Qs_sif_reset() να έχει ενσωματωθεί στην Qs_senario(), όχι στον βρόγχο, όπως οι υπόλοιπες, αλλά στο κομμάτι προεργασίας και αρχικοποιήσεων. Επίσης, στην παραγωγή των valid και operation, δεδομένου ότι εναλλάσσονται μόνο δύο τιμές, 0 και 1, αντί του toggler θα αρκούσε μία δυαδική μεταβλητή (boolean). Στην Qs_senario_parser() η reset_dur θα πρεπε να επιλέγεται από λίστα τιμών (choices), εξαλείφοντας την ανάγκη περαιτέρω ελέγχων.

Γίνεται χρήση της συνάρτησης βιβλιοθήκης os.system(), παρότι στην τεκμηρίωσή της προτείνεται η χρήση του module "subprocess" στη θέση της. Στην Qs_timestamp_duration() η μετατροπή που θα μπορούσε να γίνει[11] με χρήση των ήδη υπαρχουσών συναρτήσεων, γίνεται αναλυτικά. Στην Qs_senario(), ο έλεγχος για την εμφάνιση συμπτώσεων κοντινότερων από την αιτούμενη στην συγκεκριμένη επανάληψη, δεν ελέγχει αν πρόκειται για άλλη επιτρεπόμενη. Στην Qs_dict_load(), θα έπρεπε να γίνεται μία ενημέρωση των σταθερών, με κλήση της Qs_dict_populate(), μην θεωρώντας ότι το pickle περιέχει απαραίτητα τις σωστές, ή καν, τις τιμές τους. Οι έλεγχοι της Qs_dict_has_necessary() μπορούν να εντοπίσουν κάποια τέτοια έλλειψη, αλλά όχι και να την διορθώσουν, το οποίο αποτελεί από μόνο του παράλειψη.

Διατηρήθηκε η εντολή τερματισμού μετά την κλήση της Qs_directives(), παρότι προσωρινή, όπως αποδεικνύει και η αποθήκευση του παραγόμενου .Qsi στην Qs_D['senario_list'], στο σώμα της συνάρτησης. Αν κλήθηκε βάσει της επιλογής -i, τα αρχεία .do ή/και .Qsi που ενδεχομένως φορτώθηκαν, αντί να εκτελεστούν θα αγνοηθούν, πράγμα μη αναμενόμενο. Αντίθετα, η default κλήση της, ελλείψει φόρτωσης αρχείων εισόδων, ανταποκρίνεται στη σύμβαση διαφόρων προγραμμάτων να προσφέρουν μία βασική λειτουργικότητα ακόμα και καλούμενα χωρίς ορίσματα. Στα .Qsi αρχεία, η νόμιμη συντακτικά κενή γραμμή, που αναπαριστά την αρχικοποίηση με τις default τιμές όλων των επιλογών, δεν εκτελείται ποτέ, γιατί οι κενές γραμμές παρακάμπτονται, όπως και οι σχολιασμένες. Ανάλογο αποτέλεσμα επιτυγχάνεται με γραμμές "-s <sim_dur>", μιάς και συνήθως εκεί χρειάζεται διαφορετική τιμή.

[11] ως dur = strftime('%H:%M:%S', gmtime(d_secs)).

Αστάθεια Στον αντίποδα των προηγούμενων, υπάρχουν αβλεψίες που, εν δυνάμει, μπορούν να εμφανίσουν σφάλματα στην εκτέλεση του Tester, καταλήγοντας σε αναπάντεχο τερματισμό της λειτουργίας του.

Όταν το στάδιο της προσομοίωσης δεν ολοκληρωθεί, το επόμενο στάδιο, εκτέλεσης του μοντέλου, επιχειρώντας το άνοιγμά του αδημιούργητου .Qli χωρίς καμία προφύλαξη, θα καταλήξει σε τερματική εξαίρεση, αναπόφευκτη, αλλά ουσιαστικά πιό απότομη από ότι θα μπορούσε. Παρόλα αυτά, μία σχετιζόμενη παράλειψη αποδεικνύεται χρήσιμη: μην ελέγχοντας η εντολή χρονικής καταγραφής Qs_timestamp την επιτυχή ολοκλήρωση της προσομοίωσης, θα καταγράψει το υποτιθέμενο τέλος του σταδίου, αποκαλύπτοντας την διάρκεια εκτέλεσης πριν την παύση της. Παρακάτω, μία καταγραφή ακόμα, της προς ακύρωση έναρξης εκτέλεσης του μοντέλου, δεν προσφέρει περισσότερη πληροφορία.

Στην Qs_sVm_compare(), τα επίφοβα σημεία είναι περισσότερα. Κατά την ανίχνευση των αρχικών γραμμών που περιέχουν τα ονόματα των σημάτων, δεν έγινε πρόβλεψη για ύπαρξη αριθμητικών ψηφίων, επιτρεπτών στην ονοματολογία, και του χαρακτήρα ':', που συνοδεύει το διακριτικό πρόθεμα των datasets. Ακόμα, εξαλείφεται ο χαρακτήρας νέας γραμμής, που δεν αναγνωρίζεται ως αλφαβητικός χαρακτήρας, αποκλείοντας τον εντοπισμό του ως χαρακτήρα κενού, κατά εκείνον τον έλεγχο, και έτσι οι κενές γραμμές επίσης δεν αναγνωρίζονται ως αρχικές. Τα ονόματα των σημάτων στους τρέχοντες ελέγχους δεν περιλαμβάνουν αριθμούς ή dataset prefix, και ούτε παρεμβάλλονται κενές γραμμές, αλλά η γενικότητα περιορίζεται, και σε διαφορετική περίπτωση, αρχικές ή κενές θα αναγνωρίζονταν ως γραμμές τιμών, προκαλώντας εξαιρέσεις και τερματισμό, κατά την προσπάθεια εξαγωγής τιμών.

Στην ανίχνευση απροσδιόριστων τιμών, η μέθοδος που χρησιμοποιήθηκε, σβήνοντας πρώτα τους κενούς χαρακτήρες, μετατρέποντας έπειτα τους 'X' και '?' σε κενά, σπάζοντας την συμβολοσειρά και μετρώντας αν προέκυψαν παραπάνω από ένα κομμάτια, δεν εντοπίζει απροσδιόριστες τιμές που θα εμφανίζονταν στις ακραίες θέσεις, από την αρχή ή το τέλος, αλλά ούτε και με πεζούς χαρακτήρες ('x'). Η παράλειψη αυτή, εύκολα διορθώνεται[12], αλλά προκύπτουν, εύλογα, αμφιβολίες για την αξιοπιστία των παρουσιαζόμενων αποτελεσμάτων λειτουργίας, χωρίς σφάλματα.

Τα αποτελέσματα παραμένουν αξιόπιστα, μιάς και οι ιδιαιτερότητες του συστήματος δεν θα επέτρεπαν την εμφάνιση των περιπτώσεων που είναι επικίνδυνες για διαφυγή. Σε κάθε γραμμή, η ακραία τιμή από την αρχή είναι του χρόνου προσομοίωσης, όπως καταγράφεται από το List, και από το τέλος, η τιμή του σήματος qout, που είναι αυτούσια η τιμή του qi όπως διέρχεται από το μονοπάτι, χωρίς επεξεργασία, άρα, αρκετά απίθανο να προκύψει απροσδιοριστία. Παρόλα αυτά, για επιβεβαίωση των παραπάνω, δόθηκαν προς σύγκριση

[12] με ενίσχυση της συνάρτησης μετατροπής, και σπάσιμο σε χαρακτήρα κενού, tr = string.maketrans('Xx?',' ');... if(len(simLine.translate(tr, ' ').split(' ')) >1): ..., ή, με τον πιό pythonic τρόπο if ('X' in simLine.upper())or('?' in simLine):... .

στην συνάρτηση και .Qlo επεξεργασμένα με το χέρι, έχοντας εισαγάγει απροσδιόριστες τιμές σε όλα τα σήματα. Πράγματι, θα ανιχνευθούν σε κάθε περίπτωση, είτε προβάλλοντας σφάλμα απροσδιόριστης τιμής, τα dataout, done, initializing, είτε προκαλώντας στοίχιση, το outID. Όσο για το σήμα qout, η ανίχνευση τελικά επιτυγχάνεται, από την προκαλούμενη εξαίρεση κατά την προσπάθεια μετατροπής του απροσδιόριστου χαρακτήρα σε ακέραιο.

Απροσδόκητα αποτελέσματα Τελικά, υπάρχουν και οι περιπτώσεις που δεν αναμένεται σφάλμα κατά την εκτέλεση, αλλά το να διαφέρουν ή να αποκρύπτουν τα αποτελέσματα που εμφανίζονται, από τι έχει πραγματικά συμβεί.

Η στοίχιση των γραμμών εξόδου βάσει του σήματος inID / outID είναι το επισφαλέστερο σημείο. Θα μπορούσε, φορτώνοντας έτοιμο .do, με το σήμα, στην απλούστερη εκδοχή, σε μία σταθερή τιμή, αντί στην συμβατική συμπεριφορά αύξουσας μέτρησης, η στοίχιση να παρακάμπτεται σε σημεία που να απαιτούνταν, με ενδεχόμενη συνέπεια σωστά αποτελέσματα να εκτιμώνται ως λανθασμένα, ή το αντίθετο. Ακόμα και αν οι τιμές εισόδου μεταβάλλονται κατά σύμβαση, υπάρχει η απίθανη περίπτωση μία στοίχιση να ταιριάξει διαφορετικές επαναλήψεις του πεπερασμένου κύκλου μέτρησης του σήματος. Σε κάθε περίπτωση, η πραγματοποίηση στοίχισης δεν καταγράφεται, οπότε, θα μπορούσαν να συγκαλυφθούν με τις προσπεράσεις και ενδεχόμενα σφάλματα, ή μάλλον, το σημείο εμφάνισής τους, μιάς και η κατάσταση του συστήματος ενδέχεται να επηρεάζεται μέχρι αρκετά αργότερα. Επίσης, είναι το μοναδικό από τα σήματα με μεταβλητό πλάτος, που δεν έχει οριστεί παραμετρικά στο Qs_dictionary ούτε και μπορεί να του αποδοθεί νέα τιμή εξωτερικά από το χρήστη. Η δυσκαμψία αυτή διατηρήθηκε και στο Qs_values, ορίζοντας μοτίβο τριών ψηφίων σε δεκαεξαδικό, για την εξαγωγή των τιμών του από το σχόλιο του .Qfc για τις τελευταίες εξόδους που συγκρίθηκαν.

Κατά τη μέτρηση της ταχύτητας, η παρουσία επόμενων reset θα απέδιδε λανθασμένα αποτελέσματα, ή και αδυναμία υπολογισμού κάποιων. Εντοπίζοντας το τέλος της αρχικοποίησης από την μεταβολή του σήματος initializing σε 0, για κάθε επόμενο reset θα καταγραφεί μία νέα ταχύτητα. Η κάθε επόμενη τιμή u_i όμως, θα περιλαμβάνει όλη τη μέχρι τότε εκτέλεση, έχοντας για αρχικό σημείο την αρχή της προσομοίωσης, και όχι την αντίστοιχη μεταβολή του σήματος initializing σε 1. Ακόμα, σε περίπτωση εμφάνισης εμβόλιμου reset, κατά την αρχικοποίηση του default reset αποκλειστικά, θα αποτραπεί και ο υπολογισμός της ταχύτητας κανονικής εκτέλεσης u_c, που χρειάζεται το σημείο τερματισμού της αρχικοποίησης. Αυτό δεν θα έχει, πιθανότατα, εμφανιστεί ακόμα, έχοντας προϋπολογίσει το όριο c_lim στατικά και απλοϊκά, αγνοώντας την πραγματική εκτέλεση αρχικοποίησης και τις τιμές του reset. Τις προβλέψεις διαψεύδει και το σήμα reset, χρησιμοποιώντας την επιλογή unal_res μόνο στην παραγωγή του default, ενώ τα ενδεχόμενα επόμενα αποδίδονται στην τιμή time του εκάστοτε κύκλου, υπολογισμένη με την επιλογή unal_sigs.

Ευχαριστώ...

Καταρχάς, τον επιβλέποντά μου, καθηγητή Διονύση Πνευματικάτο, για την έμπνευση, την εμπιστοσύνη, την υπομονή, την συνεργασία —και τις ερωτήσεις με φατσούλα ☺...

Επίσης, *το σύνολο* των συναδέλφων και συνοδοιπόρων, στην μακρά αυτή πορεία. Από την Χρ.Κ. και τον Κ.Αθ., συνεργάτες στα πρώτα μας εργαστήρια, στους τόσους άλλους, στα πολλά ακόμα που ακολούθησαν, μέχρι εκείνους, που γίναν σύντροφοι, στα δύσκολα ξενύχτια, τα ακαδημαϊκά, αλλά και τα άγρια ξυπνήματα, τα πολιτικά, και φυσικά τους φίλους, παντού και πάντα. Ιδιαιτέρως, αν και δεν είναι οι μόνοι, τους: Ευτυχία Γ., Βασίλη Π., Γιάννη J . Π., Ευτυχία Μ., Άρη Ν., Φανούλα Τ., τα Κοριτσάκια, Ίριδα Κ. και Χριστίνα Π., Walid Σ., Διαμαντή Μ., Πετρούλα Τ., Ελισάβετ Τ., Ειρήνη Σ., Θωμαή Δ., Εύη Κ., Νατάσα Π., Νίκο Γ., Νταίζη Χ. .

Την οικογένειά μου, τελικά, για τα πάντα ...

Βιβλιογραφία

[1] Sartaj Sahnii, Δομές δεδομένων, Αλγόριθμοι, και Εφαρμογές στη C++, Εκδόσεις Τζιόλα, 2004.

[2] Γεώργιος Φρ. Γεωργακόπουλος, Δομές δεδομένων, Έννοιες, τεχνικές και αλγόριθμοι, Πανεπιστημιακές Εκδόσεις Κρήτης, 2002.

[3] `http://en.wikipedia.org/wiki/Queue_(data_structure)`

[4] `http://en.wikipedia.org/wiki/Message_queue`

[5] John L. Hennessy, David A. Patterson, Αρχιτεκτονική Υπολογιστών, 3η έκδοση, Εκδόσεις Τζιόλα, 2005.

[6] David A. Patterson, John L. Hennessy, Οργάνωση και σχεδίαση υπολογιστών, Η διασύνδεση υλικού και λογισμικού, 3η έκδοση, Εκδόσεις Κλειδάριθμος, 2006.

[7] William Stallings, Λειτουργικά Συστήματα : Αρχές Σχεδίασης, Εκδόσεις Τζιόλα, 2003.

[8] Andrew S, Tannenbaum, Σύγχρονα Λειτουργικά Συστήματα, Εκδόσεις Κλειδάριθμος, 2002.

[9] Peter J. Ashenden, The Designer's Guide to VHDL, Morgan Kaufmann Publishers, Inc., 1996.

[10] James R. Armstrong, Chip-level modeling with VHDL, Prentice-Hall, Inc., 1989.

[11] M. Morris Mano, Ψηφιακή Σχεδίαση, Prentice-Hall,International, Inc., Παπασωτηρίου, 1992.

[12] Stephen D. Brown and Zvonko G. Vranesic, Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL, Εκδόσεις Τζιόλα, 2001.

[13] Product Discontinuation Notice - 22 June 2009, `http://www.xilinx.com/support/documentation/customer_notices/xcn09017.pdf`

[14] ModelSim User's Manual, Software Version 6.3c, September 2007, 1991-2007 Mentor Graphics Corporation.

[15] ModelSim Reference Manual, Software Version 6.3c, September 2007, 1991-2007 Mentor Graphics Corporation.

[16] Christoforos E. Kozyrakis, The Architecture, Operation and Design of the Queue Management Block in the ATLAS I ATM Switch, B.Sc. Thesis, University of Crete, Greece; Technical Report TR-172, Institute of Computer Science - Foundation for Research and Technology Hellas (ICS-FORTH), Ιούλιος 1996. URL: `file://ftp.ics.forth.gr/tech-reports/1996/1996.TR172.QueueManagement.ps.gz`.

[17] George Kornaros, Christoforos Kozyrakis, Panagiota Vatsolaki, and Manolis Katevenis, Pipelined Multi-Queue Management in a VLSI ATM Switch Chip with Credit-Based Flow-Control, Proc. of 17th Conf. on Advanced Research in VLSI (ARVLSI'97), Univ. of Michigan, Ann Arbor, USA, Sep. 1997. URL: `ftp://ftp.ics.forth.gr/tech-reports/1997/1997.AVLSI.Pipe_MultiQueue.ps.gz`

[18] A. Nikologiannis, Efficient Per-Flow Queueing in DRAM at OC-192 Line Rate using Out-of-Order Execution Techniques, Master of Science Thesis, University of Crete, Greece; Technical Report FORTH-ICS/TR-279, Institute of Computer Science, FORTH, Ηράκλειο, Κρήτη, Νοέμβριος 2000. URL: `http://archvlsi.forth.gr/muqpro/queueMgt.html`.

[19] Aristides Nikologiannis, Manolis Katevenis, Efficient Per-Flow Queueing in DRAM at OC-192 Line Rate using Out-of-Order Execution Techniques, Proc. IEEE Int. Conf. on Communications (ICC'2001), Helsinki, Finland, Ιούνιος 2001, σελ. 2048 - 2052. URL: `http://archvlsi.forth.gr/muqpro/queueMgt.html`

[20] Dimitrios S. Kapsalis, Design and Implementation of a Per-Flow Queue Manager for an ATM Switch using FPGA technology, Master of Science Thesis, University of Crete, Greece; Technical Report FORTH-ICS/TR-302, Institute of Computer Science, FORTH, Ηράκλειο, Κρήτη, Φεβρουάριος 2002. URL: `ftp://ftp.ics.forth.gr/tech-reports/2002/2002.TR302.Design_per_flow_queue_manager_FPGA_Technology.ps.gz`.

[21] A. Nikologiannis, I. Papaefstathiou, G. Kornaros, C. Kachris, An FPGA-based queue management system for high speed networking devices, Microprocessors and Microsystems 28, 2004, 223-236.

[22] Variable-Packet-Size IQ and CICQ (Buffered) Crossbar Arch. (FORTH-ICS). `http://archvlsi.ics.forth.gr/bufxbar/`

[23] S.A. Paredes, S. Taebi, T.J. Hall, Packet-loss-robust load balancing switch with distributed extended cross-point queues, IET Communications, 2009, Vol. 3, Iss. 1, pp. 123-134.