ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ
ΗΛΕΚΤΡΟΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Κωνσταντίνος Κυριακουλάκος
Α.Μ.:2000030020

ΔΙΠΛΩΜΑΤΙΚΉ ΕΡΓΑΣΊΑ

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ ΜΙΑΣ ΠΑΡΑΜΕΤΡΙΚΗΣ ΚΡΥΦΗΣ ΜΝΗΜΗΣ

Επιβλέπων Καθηγητής:

Διονύσιος Πνευματικάτος

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον κ. Πνευματικάτο Διονύσιο για την πολύτιμη βοήθεια που μου προσέφερε καθ'όλη τη διάρκεια της παρούσας εργασίας.
Επίσης ένα μεγάλο ευχαριστώ στα παιδιά του εργαστηρίου μικροεπεξεργαστών και υλικού του Π.Κ, καθώς και σε όλους τους συναδέλφους που μου παρείχαν βοήθεια για τη διεκπεραίωση της εργασίας.

ΠΡΟΛΟΓΟΣ

Η διπλωματική αυτή εργασία παρουσιάζει την υλοποίηση μιας παραμετρικής κρυφής μνήμης, γνωστής με τον αγγλικό όρο cache. Το γνώρισμα "παραμετρική" σχετίζεται με τη δυνατότητα του χρήστη να επιλέγει το επιθυμητό μέγεθος κρυφής μνήμης, μέγεθος μπλοκ και βαθμό συνολοσυσχετιστικότητας (associativity).

Η σχεδίαση έγινε για τρια ξεχωριστά συστήματα, ανάλογα με το τρόπο εγγραφής (write (no)allocate – write through/back), που στο τέλος ενοποιούνται. Κατά την σχεδίαση υποστηρίξαμε την λειτουργία της αρχικοποίησης και πετύχαμε απόδοση εξυπηρέτισης: 1 αίτηση/κύκλο (περίπτωση hit).
Η σύνθεση της κρυφής μνήμης, σε συσκευή FPGA, έγινε με το εργαλείο ISE 7.1 της Xilinx, από το οποίο συλλέξαμε πληροφορίες σχετικά με την συχνότητα λειτουργίας και τους πόρους της συσκευής που απαιτούνται. Τέλος η προσομοίωση του συστήματος έγινε στο περιβάλλον του ModelSim SE 6.0a, όπου διαπιστώθηκε η ορθή λειτουργία της κρυφής μνήμης.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Αρχικά θα δώσω μια εικόνα του τι είναι μια κρυφή μνήμη, για πιο σκοπό επινοήθηκε και πως αντιλαμβανόμαστε την απόδοσή της (Κεφ. 1). Σκοπός αυτής της εισαγωγής είναι να μπορέσει να παρακολουθήσει την συνέχεια της αναφοράς ακόμα και κάποιος με ελάχιστο υπόβαθρο στον τομέα της αρχιτεκτονικής υπολογιστών.
Ύστερα θα αναφέρω τα αυστηρά χαρακτηριστικά (προδιαγραφές)του μοντέλου που προσπάθησα να δημιουργήσω (Κεφ. 2). Τα χαρακτηριστικά αυτά αποτελούν τον μπούσουλα του σχεδιαστή γιατί τον οδηγούν από μια αφηρημένη εικόνα προς ένα σχέδιο με συγκεκριμένο σκοπό και περιορισμούς.
Το επόμενο κεφάλαιο (Κεφ. 3) αποτελεί την μελέτη που έκανα πριν τη σχεδίαση με σκοπό να βρω τα εύρη και τους περιορισμούς των μεταβλητών του προβλήματος.
Ακολουθεί η περιγραφή της σχεδίασης του μοντέλου με ιδιαίτερη έμφαση στην παραμετρικοποίηση, διότι ήταν το στοιχείο για το οποίο χρειαζόταν η περισσότερη προσχεδιαστική μελέτη (Κεφ. 4).
Στο τελευταίο κομμάτι (Κεφ. 5) περιγράφεται η υλοποίηση και η στρατηγική ελέγχου (testing) του μοντέλου. Πρέπει να σημειωθεί ότι έλεγχοι γίνονταν καθ' όλη την διάρκεια υλοποίησης ώστε να διασφαλίζεται η σωστή λειτουργία κάθε κομματιού που δημιουργούσαμε. Ωστόσο τα τελευταία τεστ είναι και τα πιο σημαντικά διότι είναι εκείνα που θα μας δώσουν πλήρη εικόνα για το πόσο το μοντέλο μας πέτυχε να εκπληρώσει τις προδιαγραφές του. Ύστερα δίνονται τα αποτελέσματα των δοκιμών ελέγχου και ακολουθούν τα συμπεράσματα που βγαίνουν από τα τεστ.
Η αναφορά κλείνει με τα τελικά σχόλια για την συγκεκριμένη εργασία και προτεινόμενες βελτιώσεις.

Σκοπός και συνεισφορά της διπλωματικής
Η διπλωματική αυτή εργασία έχει ως σκοπό την «Υλοποίηση και Προσομοίωση μιας Παραμετρικής Κρυφής Μνήμης». Με τον όρο παραμετρική εννοούμε ότι ο χρήστης του μοντέλου θα μπορεί να επιλέξει μέσα από ένα σύνολο τιμών κάποιο ή κάποια χαρακτηριστικά του συστήματος π.χ. το μέγεθος της μνήμης.
Η συνεισφορά της εργασίας είναι σημαντική στην εμπειρία που απέκτησα για το πως σχεδιάζουμε και υλοποιούμε ένα σύστημα ξεκινώντας από το μηδέν, δηλαδή χωρίς την καθοδήγηση που υπήρχε στις εργασίες που γίνονταν στα πλαίσια ενός μαθήματος. Ήρθα σε επαφή με εργαλεία καινούργια σε μένα και πιο σύνθετα από ότ,ι γνώριζα και εργάστηκα για πρώτη φορά με τρόπο συστηματικό για να ελέγξω και τελικά να ολοκληρώσω το σύστημα.

ΟΡΟΛΟΓΙΑ

cache	κρυφή μνήμη
cache hit	ευστοχία κρυφής μνήμης
cache miss	αστοχία κρυφής μνήμης
block	μπλοκ
virtual memory	εικονική μνήμη
page	σελίδα
page fault	σφάλμα σελίδας
memory stall cycles	κύκλοι σταματήματος μνήμης
miss penalty	ποινή αστοχίας
miss rate	ρυθμός αστοχίας
address trace	ίχνος διευθύνσεων
direct mapped	άμεσα αντιστοιχισμένη
fully associative	πλήρως συσχετιστική
n-way set associative	σύνολο-συσχετιστική ν-δρόμων
set	σύνολο
valid bit	bit εγκυρότητας
dirty bit	bit αλλοίωσης
least-recently used	λιγότερο προσφάτως χρησιμοποιημένος
random replacement	τυχαία αντικατάσταση
block address	διεύθυνση μπλοκ
block offset	απόκλιση μπλοκ
tag field	πεδίο ετικέτα
index	πεδίο δείκτης
write through	διεγγραφή
write back	ετεροχρονισμένη εγγραφή
write allocate	κατανομή εγγραφής
no-write allocate	κατανομή μη εγγραφής
instruction cache	κρυφή μνήμη εντολών
data cache	κρυφή μνήμη δεδομένων
unified cache	ενοποιημένη κρυφή μνήμη
write buffer	προσωρινός χώρος αποθήκευσης εγγραφής
average memory access time	μέσος χρόνος προσπέλασης μνήμης
hit time	χρόνος ευστοχίας
misses per instruction	αστοχίες ανά εντολή
write stall	σταμάτημα εγγραφής

Η παραπάνω ορολογία είναι σύμφωνη με την μεταφρασμένη στα ελληνικά έκδοση του βιβλίου “Computer Architecture, A Quantitative Approach, 3rd edition” των John L. Henessy και David A. Patterson.

ΚΕΦΑΛΑΙΟ 1

ΘΕΩΡΙΑ

Σημείωση:
Το κεφάλαιο αυτό είναι βασισμένο στο τετάρτο κεφαλαίο του βιβλίου "Computer Architecture, A Quantitative Approach, 3rd edition" των John L. Henessy και David A. Patterson, με μικρές εκ μέρους μου αλλαγές.

Η κρυφή μνήμη δημιουργήθηκε από την ανάγκη να κατασκευαστεί μία ιεραρχία μνήμης. Ο τρόπος που είναι οργανωμένη, ως προς το ποια δεδομένα κρατάει, απορρέει από την ικανοποίηση της αρχής της τοπικότητας.

Η αρχή της τοπικότητας
Αν παρατηρήσουμε τον κώδικα των προγραμμάτων μπορούμε να βγάλουμε χρήσιμες πληροφορίες από τις ιδιότητές τους. Η πιο σημαντική ιδιότητα την οποία χρησιμοποιούμε συχνά είναι η αρχή της τοπικότητας: Τα προγράμματα τείνουν να ξαναχρησιμοποιήσουν δεδομένα και εντολές που χρησιμοποίησαν πρόσφατα. Σύμφωνα με την ανάλυση που είναι γενικώς παραδεκτή ένα πρόγραμμα περνά το 90% του χρόνου εκτέλεσης σε μόνο 10% του κώδικα. Ένα αποτέλεσμα της τοπικότητας είναι ότι μπορούμε να προβλέψουμε με σχετική ακρίβεια ποιες εντολές και δεδομένα θα χρησιμοποιήσει ένα πρόγραμμα στο εγγύς μέλλον βασιζόμενοι στις προσβάσεις του στο πρόσφατο παρελθόν.
Η αρχή της τοπικότητας ισχύει και για προσβάσεις δεδομένων, αλλά όχι τόσο πολύ όσο στις προσβάσεις κώδικα. Δύο διαφορετικοί τύποι τοπικότητας έχουν παρατηρηθεί. Η χρονική τοπικότητα ορίζει ότι αντικείμενα που χρησιμοποιήθηκαν πρόσφατα έχουν μεγάλες πιθανότητες να ξαναχρησιμοποιηθούν στο εγγύς μέλλον. Η χωρική τοπικότητα λέει ότι αντικείμενα των οποίων οι διευθύνσεις είναι κοντά η μια στην άλλη τείνουν να αναφέρονται σε χρόνους που είναι κοντά ο ένας στον άλλο.
Σαν παράδειγμα εντολών που χρησιμοποιούνται πολύ περισσότερο από ότι οι υπόλοιπες εντολές του κώδικα ενός προγράμματος σκεφτείτε τους βρόγχους εντολών (loops). Είναι ένα εμφανέστατο δείγμα χρονικής τοπικότητας σε προσβάσεις κώδικα. Η χωρική τοπικότητα για εντολές είναι πασιφανής αφού όταν η εκτέλεση του κώδικα γίνεται σειριακά πάντα η επόμενη εντολή προς εκτέλεση είναι αυτή με την επόμενη διεύθυνση στη μνήμη. Εξαιρούνται φυσικά οι περιπτώσεις των αλμάτων(jumps). Όσο για τα δεδομένα χρονική τοπικότητα μπορούμε να παρατηρήσουμε σε ένα πρόγραμμα που έχει λίγες μεταβλητές και κάνει συνεχώς πράξεις μεταξύ τους, ενώ μια διάταξη(array) π.χ s που αρχικοποιείται σε ένα βρόγχο είναι δείγμα χωρικής τοπικότητας δεδομένων αφού το s[i] και το s[i+1] έχουν γειτονικές διευθύνσεις.

ΣΧΕΔΙΑΣΜΟΣ ΙΕΡΑΡΧΙΑΣ ΜΝΗΜΗΣ
«Ιδεωδώς κάποιος θα επιθυμούσε μια απεριόριστα μεγάλης χωρητικότητας μνήμη τέτοια που οποιαδήποτε συγκεκριμένη … λέξη θα μπορούσε να είναι άμεσα διαθέσιμη … Είμαστε … αναγκασμένοι να αναγνωρίσουμε τη δυνατότητα της κατασκευής μιας ιεραρχίας μνημών, καθεμία από τις οποίες έχει μεγαλύτερη χωρητικότητα από την προηγούμενη, αλλά η οποία είναι λιγότερο γρήγορα προσπελάσιμη.»
A.W. Burks, H. H. Goldstine, and J. von Neumann
Preliminary Discussion of the Logical Design of an Electronic Computing Instrument (1946)

Εισαγωγή

Οι πρωτοπόροι των υπολογιστών ορθώς προέβλεψαν ότι οι προγραμματιστές θα επιθυμούσαν απεριόριστες ποσότητες ταχείας μνήμης. Μια οικονομική λύση σε αυτή την επιθυμία είναι η ιεραρχία μνήμης, η οποία εκμεταλλεύεται την τοπικότητα και το κόστος απόδοσης των τεχνολογιών μνήμης. Η αρχή της τοπικότητας αναφέρεται στο ότι τα περισσότερα προγράμματα δεν προσπελαύνουν όλο τον κώδικα και τα δεδομένα με ομοιόμορφο τρόπο. Αυτή η αρχή σε συνδυασμό με την οδηγία ότι το μικρότερο υλικό είναι γρηγορότερο, οδήγησαν σε ιεραρχίες βασισμένες σε μνήμες διαφορετικών ταχυτήτων και μεγεθών. Το σχήμα 1 αποτυπώνει μια πολυεπίπεδη ιεραρχία μνήμης.

Εφόσον η ταχεία μνήμη είναι ακριβή, μια ιεραρχία μνήμης οργανώνεται σε πολλά επίπεδα –καθένα μικρότερο- γρηγορότερο και ακριβότερο ανά byte απ' ότι το αμέσως χαμηλότερο επίπεδο. Ο στόχος είναι να παρέχει ένα σύστημα μνήμης με κόστος σχεδόν τόσο χαμηλό όσο το φθηνότερο επίπεδο μνήμης, και ταχύτητα σχεδόν τόσο μεγάλη όσο το γρηγορότερο επίπεδο. Τα επίπεδα της ιεραρχίας συνήθως υποκαθιστούν το ένα το άλλο. Όλα τα δεδομένα σε ένα επίπεδο βρίσκονται επίσης στο παρακάτω επίπεδο και όλα τα δεδομένα σ' αυτό το χαμηλότερο επίπεδο, βρίσκονται και στο παρακάτω αυτού, και ούτω καθεξής μέχρι να φτάσουμε τον πυθμένα της ιεραρχίας.

Σημειώστε ότι κάθε επίπεδο χαρτογραφεί διευθύνσεις από μια πιο αργή, μεγαλύτερη μνήμη σε μια μικρότερη αλλά ταχύτερη μνήμη, υψηλότερα στην ιεραρχία. Ως μέρος της αντιστοίχησης των διευθύνσεων, η ιεραρχία μνήμης αναλαμβάνει την ευθύνη του ελέγχου των διευθύνσεων, έτσι θεωρούμε ότι τα σχήματα προστασίας για να ελέγχουν διευθύνσεις είναι επίσης τμήματα της ιεραρχίας της μνήμης.

Η σημασία της ιεραρχίας μνήμης έχει αυξηθεί με προόδους στην απόδοση των επεξεργαστών. Για παράδειγμα, το 1980 μικροεπεξεργαστές συχνά σχεδιάζονταν χωρίς κρυφές μνήμες, ενώ το 2001 πολλοί έχουν δύο επίπεδα κρυφών μνημών στο τσιπ.

«Κρυφή μνήμη (cache): ένα ασφαλές μέρος για να κρύβεις ή να αποθηκεύεις πράγματα.»
Λεξικό του Webster της Αμερικάνικης Γλώσσας
2η κολεγιακή έκδοση του 1976

Κρυφή μνήμη είναι το όνομα που δίνεται στο πρώτο επίπεδο της ιεραρχίας της μνήμης που συναντάται από τη στιγμή που η διεύθυνση αφήνει την ΚΜΕ (κεντρική μονάδα επεξεργασίας). Από την στιγμή που η αρχή της τοπικότητας ισχύει σε πολλά επίπεδα και το να εκμεταλλεύεσαι την τοπικότητα για να βελτιώσεις την απόδοση είναι δημοφιλές, ο όρος κρυφή μνήμη τώρα απευθύνεται οπουδήποτε γίνεται εκμετάλλευση της μνήμης για να ξαναχρησιμοποιηθούν στοιχεία που συχνά συναντώνται. Παραδείγματα περιλαμβάνουν κρυφές μνήμες αρχείων, κρυφές μνήμων ονομάτων κλπ.

Όταν η ΚΜΕ βρίσκει ένα αιτούμενο στοιχείο δεδομένων στη κρυφή μνήμη, αυτό ονομάζεται ευστοχία κρυφής μνήμης. Όταν η ΚΜΕ δεν βρίσκει ένα στοιχείο δεδομένων που χρειάζεται στην κρυφή μνήμη, συμβαίνει αστοχία κρυφής μνήμης.
Μια συγκεκριμένου μεγέθους συλλογή δεδομένων στην κρυφή μνήμη, που περιλαμβάνει την απαιτούμενη λέξη ονομάζεται μπλοκ, ανασύρεται από την κύρια μνήμη και τοποθετείται στην κρυφή μνήμη.
Η χρονική τοπικότητα μας λέει ότι μάλλον θα χρειαστούμε αυτή τη λέξη πάλι στο εγγύς μέλλον, έτσι είναι χρήσιμο να την τοποθετήσουμε στην κρυφή μνήμη όπου μπορεί να είναι γρήγορα προσπελάσιμη. Εξαιτίας της χωρικής τοπικότητας υπάρχει υψηλή πιθανότητα ότι τα άλλα δεδομένα μπλοκ θα χρειαστούν σύντομα.
Ο χρόνος που χρειάζεται να εξυπηρετηθεί η ΚΜΕ κατά την αστοχία κρυφής μνήμης εξαρτάται και από την καθυστέρηση και από το εύρος ζώνης της μνήμης. Η καθυστέρηση καθορίζει το χρόνο ανάκτησης της πρώτης λέξης του μπλοκ και το εύρος ζώνης καθορίζει το χρόνο ανάκτησης του υπόλοιπου μπλοκ. Μια αστοχία κρυφής μνήμης χειρίζεται το υλικό και αναγκάζει τους επεξεργαστές, οι οποίοι ακολουθούν εκτέλεση με σειρά να σταματήσουν μέχρι να είναι διαθέσιμα τα δεδομένα.

Λειτουργία της κρυφής μνήμης
Εξαιτίας της τοπικότητας και της μεγαλύτερης ταχύτητας των μικρότερων μνημών, μια ιεραρχία μνήμης μπορεί ουσιαστικά να βελτιώσει την απόδοση σημαντικά. Για να αξιολογήσουμε την απόδοση της κρυφής μνήμης υπολογίζουμε τον αριθμό των κύκλων, στη διάρκεια των οποίων η ΚΜΕ σταματά περιμένοντας για μια προσπέλαση μνήμης, τους οποίους ονομάζουμε κύκλους καθυστέρησης μνήμης. Η απόδοση τότε είναι το γινόμενο του χρόνου κύκλου ρολογιού και του αθροίσματος των κύκλων ΚΜΕ και των κύκλων καθυστέρησης μνήμης:

ΚΜΕ χρόνος εκτέλεσης = (ΚΜΕ κύκλοι ρολογιού + κύκλοι καθυστέρησης μνήμης)
$\times$ χρόνος κύκλου ρολογιού

Αυτή η ισότητα υποθέτει ότι οι κύκλοι ρολογιού ΚΜΕ περιλαμβάνουν το χρόνο να χειριστούν μια ευστοχία κρυφής μνήμης και ότι η ΚΜΕ σταματά στη διάρκεια αστοχίας μνήμης.
Ο αριθμός των κύκλων καθυστέρησης μνήμης εξαρτάται και από τον αριθμό των αστοχιών και από το κόστος ανά αστοχία, το οποίο ονομάζεται ποινή αστοχίας:

Κύκλοι καθυστέρησης μνήμης = Αριθμός αστοχιών $\times$ ποινή αστοχίας =

$$= IC \times \frac{\text{Αστοχίες}}{\text{Εντολή}} \times \text{ποινή αστοχίας} =$$

IC: αριθμός εντολών (instruction count)

$$= IC \times \frac{\text{προσπελάσεις μνήμης}}{\text{εντολή}} \times \text{ρυθμό αστοχίας}$$

$\times$ ποινή αστοχίας

Το πλεονέκτημα του τελευταίου τύπου είναι ότι τα συστατικά στοιχεία μπορούν εύκολα να μετρηθούν. Ο αριθμός εντολών μπορεί να μετρηθεί π.χ χρησιμοποιώντας τους μετρητές υλικού του επεξεργαστή ή μεταφράζοντας το πρόγραμμα σε επίπεδο εντολών και μετρώντας τις εντολές κατά τη διαδικασία αυτή. Το μέτρημα του αριθμού αναφορών μνήμης ανά εντολή μπορεί να γίνει με τον ίδιο τρόπο· κάθε εντολή απαιτεί μια προσπέλαση εντολής και μπορούνε εύκολα να αποφασίσουμε αν απαιτεί επίσης και

προσπέλαση δεδομένων. Το συστατικό στοιχείο ρυθμός αστοχίας είναι απλώς το κλάσμα των προσπελάσεων κρυφής μνήμης που καταλήγουν σε αστοχία (δηλαδή ο αριθμός των προσπελάσεων που αστοχούν διαιρεμένος δια του αριθμού των προσπελάσεων). Οι ρυθμοί αστοχίας μπορούν να μετρηθούν με προσομοιωτές κρυφής μνήμης οι οποίοι παίρνουν ένα ίχνος διευθύνσεων των αναφορών σε εντολές και δεδομένα, προσομοιώνουν την συμπεριφορά των κρυφών μνημών για να εντοπίσουν ποιες αναφορές ευστοχούν και ποιες αστοχούν και μετά αναφέρουν τα αθροίσματα των ευστοχιών και αστοχιών. Κάποιοι μικροεπεξεργαστές παρέχουν για να μετρούν τον αριθμό των αστοχιών και των αναφορών μνήμης, γεγονός το οποίο αποτελεί έναν πολύ ευκολότερο και γρηγορότερο τρόπο για τη μέτρηση του ρυθμού αστοχίας. Η ποινή αστοχίας δεν είναι στην πραγματικότητα ένας σταθερός αριθμός αλλά μπορούμε να χρησιμοποιήσουμε μια σταθερή τιμή για απλούστευση.

- Ο ρυθμός αστοχίας είναι μια από τις πιο σημαντικές παραμέτρους του σχεδιασμού κρυφών.

Τέσσερις ερωτήσεις ιεραρχίας μνήμης
Συνεχίζουμε την εισαγωγή μας στις κρυφές μνήμες απαντώντας στις τέσσερις συνήθεις ερωτήσεις για το πρώτο επίπεδο της ιεραρχίας μνήμης

Ε1: Που μπορεί να τοποθετηθεί ένα μπλοκ στο ανώτερο επίπεδο; (τοποθέτηση μπλοκ)
Ε2: Πως βρίσκεται ένα μπλοκ εάν είναι στο ανώτερο επίπεδο; (αναγνώριση μπλοκ)
Ε3: Ποιο μπλοκ θα μπορούσε να αντικατασταθεί σε μια αστοχία; (αντικατάσταση μπλοκ)
Ε4: Τι συμβαίνει σε μια εγγραφή; (στρατηγική εγγραφής)

Οι απαντήσεις σ' αυτές τις ερωτήσεις μας βοηθούν να καταλάβουμε διαφορετικά την αλληλεπίδραση των μνημών σε διαφορετικά επίπεδα μιας ιεραρχίας.

Ε1: Που μπορεί να τοποθετηθεί ένα μπλοκ σε μια κρυφή μνήμη;
Το σχήμα 2 δείχνει ότι οι περιορισμοί σχετικά με το πού τοποθετείται ένα μπλοκ δημιουργούν τρεις κατηγορίες οργάνωσης κρυφής μνήμης:

❖ Αν κάθε μπλοκ έχει μόνο μια θέση που μπορεί να εμφανίζεται στην κρυφή μνήμη, η κρυφή μνήμη ονομάζεται **«άμεσα αντιστοιχισμένη»**. Η αντιστοίχηση συνήθως ορίζεται ως

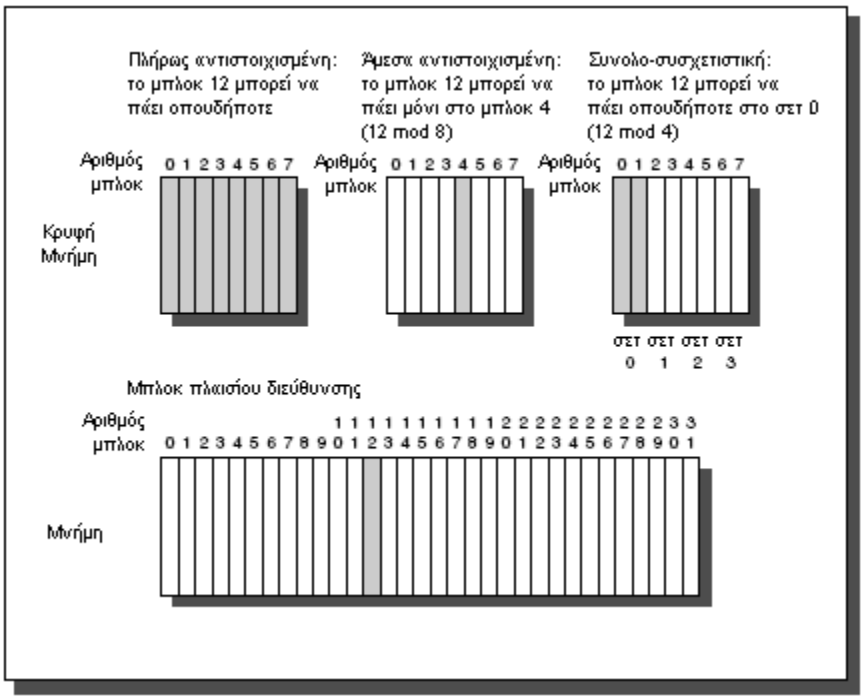
(διεύθυνση του μπλοκ) MOD (αριθμός μπλοκ στην κρυφή μνήμη)

❖ Αν ένα μπλοκ μπορεί να τοποθετηθεί οπουδήποτε στην κρυφή μνήμη, η κρυφή μνήμη ονομάζεται **«πλήρως συσχετιστική»**.
❖ Αν ένα μπλοκ μπορεί να τοποθετηθεί σε περιορισμένο αριθμό θέσεων στην κρυφή μνήμη, η κρυφή μνήμη είναι **«συνολοσυσχετιστική»**. Ένα σύνολο είναι μια ομάδα από μπλοκ στην κρυφή μνήμη. Ένα μπλοκ αρχικά αντιστοιχίζεται πάνω σε ένα σύνολο και μετά το μπλοκ μπορεί να τοποθετηθεί οπουδήποτε μέσα σ' αυτό το σύνολο. Το σύνολο συνήθως επιλέγεται σε επιλογή bit, δηλαδή

(διεύθυνση του μπλοκ) MOD (αριθμός συνόλων στην κρυφή μνήμη)

Αν υπάρχουν **n** μπλοκ σε ένα σύνολο, η τοποθέτηση στη κρυφή μνήμη ονομάζεται **«συνολοσυσχετιστική n δρόμων»**

Οι σύγχρονες κρυφές μνήμες περιέχουν χιλιάδες πλαίσια μπλοκ και οι σύγχρονες μνήμες περιέχουν εκατομμύρια μπλοκ. Η «συνολοσυσχετιστική» οργάνωση που έχει τέσσερα σύνολα με δύο μπλοκ ανά σύνολο, ονομάζεται σύνολο-συσχετιστική δύο δρόμων.



Σχήμα 2 Αυτό το παράδειγμα κρυφής μνήμης έχει οκτώ πλαίσια μπλοκ και μνήμη έχει 32 μπλοκ. Οι τρεις επιλογές για τις κρυφές μνήμες εκφράζονται από αριστερά προς τα δεξιά. Στην πλήρως συσχετιστική το μπλοκ 12 από το χαμηλότερο επίπεδο μπορεί να μετακινηθεί σε οποιοδήποτε από τα 8 πλαίσια μπλοκ της κρυφής μνήμης. Με άμεσα αντιστοιχισμένη το μπλοκ 12 μπορεί να τοποθετηθεί στο πλαίσιο του μπλοκ 4 (12 modulo 8). Η συνολοσυσχετιστική, η οποία έχει μερικά και από τα δύο χαρακτηριστικά, επιτρέπει στο μπλοκ να τοποθετηθεί οπουδήποτε στο σύνολο 0 (12 modulo 4). Με δύο μπλοκ ανά σύνολο, αυτό σημαίνει ότι το μπλοκ 12 μπορεί να τοποθετηθεί είτε στο μπλοκ 0 ή στο μπλοκ 1 της κρυφής μνήμης.

Το εύρος των κρυφών μνημών από την άμεσα αντιστοιχισμένη μέχρι την πλήρως συσχετιστική είναι στην πραγματικότητα μια συνέχεια από συνολοσυσχετισμό. Η άμεσα αντιστοιχισμένη είναι απλά συνολοσυσχετιστική ενός δρόμου και μια πλήρως συσχετιστική κρυφή μνήμη με **m** μπλοκ θα μπορούσε να ονομαστεί **«συνολοσυσχετιστική m- δρόμων»**. Ισοδύναμα η άμεσα αντιστοιχισμένη μπορεί να θεωρηθεί ότι έχει **m** σύνολα και η πλήρως συσχετιστική ότι έχει ένα σύνολο.
Η τεράστια πλειοψηφία των κρυφών μνημών επεξεργαστών σήμερα είναι άμεσα αντιστοιχισμένες, συνολοσυσχετιστικές 2 δρόμων ή 4 δρόμων.

Ε2: Πως βρίσκεται ένα μπλοκ αν είναι μέσα στην κρυφή μνήμη;
Οι κρυφές μνήμες έχουν μια ετικέτα διεύθυνσης σε κάθε πλαίσιο μπλοκ η οποία δίνει τη διεύθυνση του μπλοκ. Η ετικέτα κάθε μπλοκ κρυφής μνήμης που θα μπορούσε να περιέχει την επιθυμητή πληροφορία ελέγχεται για να δούμε αν ταιριάζει με την διεύθυνση του μπλοκ από την ΚΜΕ. Ως κανόνας, όλες οι πιθανές ετικέτες ελέγχονται παράλληλα γιατί η ταχύτητα είναι σημαντική υπόθεση.

Πρέπει να υπάρχει ένας τρόπος να γνωρίζουμε ότι ένα μπλοκ μνήμης δεν έχει έγκυρες πληροφορίες. Η συνηθέστερη διαδικασία είναι να προσθέσουμε ένα bit εγκυρότητας στην ετικέτα για να πούμε αν αυτή η εγγραφή περιέχει μια έγκυρη διεύθυνση. Αν το bit δεν έχει τεθεί δεν μπορεί να υπάρχει ταίριασμα σ' αυτή τη διεύθυνση.
Πριν προχωρήσουμε στην επόμενη ερώτηση, ας εξερευνήσουμε τη σχέση μιας διεύθυνσης ΚΜΕ με την κρυφή μνήμη. Το σχήμα 3 δείχνει πως διαιρείται μια διεύθυνση. Η πρώτη υποδιαίρεση είναι ανάμεσα στη διεύθυνση του μπλοκ και στο μπλοκ offset(απόκλιση μπλοκ). Η διεύθυνση του πλαισίου του μπλοκ μπορεί να διαιρεθεί περαιτέρω σε πεδία ετικέτας και πεδίο δείκτη. Το πεδίο μπλοκ offset επιλέγει το επιθυμητό δεδομένο από το μπλοκ, το πεδίο δείκτη επιλέγει το σύνολο και το πεδίο ετικέτας συγκρίνεται με αυτό για μια ευστοχία.

Μπλοκ διεύθυνσης		Απόκλιση μπλοκ
Ετικέτα	Δείκτης	

Σχήμα 3 Τα τρία τμήματα μιας διεύθυνσης σε μια σύνολο-συσχετιστική ή άμεσα αντιστοιχισμένη κρυφή μνήμη. Η ετικέτα χρησιμοποιείται για να ελέγξει όλα τα μπλοκ στο σύνολο και ο δείκτης χρησιμοποιείται για να επιλέξει το σύνολο. Η απόκλιση του μπλοκ (offset) είναι η διεύθυνση των επιθυμητών δεδομένων μέσα στο μπλοκ. Οι πλήρως συσχετιστικές κρυφές μνήμες δεν έχουν πεδίο δείκτη.

Αν και η σύγκριση μπορούσε να γίνει σε περισσότερα bits της διεύθυνση παρά μόνο στην ετικέτα, δεν υπάρχει τέτοια ανάγκη για τους παρακάτω λόγους:

- Το offset δεν θα έπρεπε να χρησιμοποιείται στη σύγκριση, αφού ολόκληρο το μπλοκ ή είναι παρόν ή όχι και συνεπώς όλα τα μπλοκ offset έχουν ως αποτέλεσμα ένα ταίριασμα εξ' ορισμού.
- Το να ελέγχουμε το δείκτη είναι πλεονάζον, αφού είχε χρησιμοποιηθεί για να επιλέξει το σύνολο που έπρεπε να ελεγχθεί. Μια διεύθυνση αποθηκεμένη στο σύνολο 0, για παράδειγμα, πρέπει να έχει 0 στο πεδίο δείκτη, αλλιώς δεν θα μπορούσε να αποθηκευτεί στο σύνολο 0. Το σύνολο 1 πρέπει να έχει τιμή δείκτη 1 κ.ο.κ. Αυτή η βελτιστοποίηση εξοικονομεί υλικό και ισχύ, μειώνοντας το πλάτος του μεγέθους μνήμης για την ετικέτα της κρυφής μνήμης.

Αν το συνολικό μέγεθος της κρυφής μνήμης έχει διατηρηθεί ίδιο, αυξάνοντας τη συσχετιστικότητα αυξάνεται και ο αριθμός των μπλοκ ανά σύνολο και κατ' αυτόν τον τρόπο μειώνοντας το μέγεθος του δείκτη και αυξάνοντας το μέγεθος της ετικέτας. Έτσι, το όριο ετικέτας-δείκτη στο σχήμα 3 κινείται προς τα δεξιά με αυξανόμενη συσχετιστικότητα, με τελικό σημείο πλήρως συσχετιστικές κρυφές μνήμες που δεν έχουν πεδίο δείκτη.

Ε3: Ποιο μπλοκ θα έπρεπε να αντικατασταθεί σε μια αστοχία κρυφής μνήμης;
Όταν συμβαίνει μια αστοχία, ο ελεγκτής της κρυφής μνήμης πρέπει να επιλέξει ένα μπλοκ να αντικατασταθεί με τα επιθυμητά δεδομένα. Ένα πλεονέκτημα της τοποθέτησης της άμεσης αντιστοίχησης είναι ότι οι επιλογές υλικού απλοποιούνται στην πραγματικότητα τόσο πολύ που δεν υπάρχει καμία επιλογή˙ μόνο ένα πλαίσιο του μπλοκ ελέγχεται για μια ευστοχία και μόνο αυτό το μπλοκ μπορεί να αντικατασταθεί. Με πλήρως συσχετιστική ή συνολοσυσχετιστική τοποθέτηση, υπάρχουν πολλά μπλοκ να επιλέξει κανείς σε περίπτωση αστοχίας.
Υπάρχουν τρεις βασικές στρατηγικές που χρησιμοποιούνται για την επιλογή του μπλοκ που αντικαθίσταται

- *Τυχαία.* Για να απλωθεί η κατανομή ομοιόμορφα, τα υποψήφια μπλοκ επιλέγονται τυχαία. Μερικά συστήματα παράγουν ψευδοτυχαίους αριθμούς μπλοκ για να πάρουν αναπαραγώγιμη συμπεριφορά, η οποία είναι ιδιαίτερα χρήσιμη όταν το υλικό εκκαθαρίζεται από σφάλματα.
- *Λιγότερο προσφάτως χρησιμοποιημένο (least-recently used, LRU).* Για να μειώσουμε την πιθανότητα να απαλείψουμε πληροφορία που θα την χρειαστούμε σύντομα, οι προσπελάσεις στα μπλοκ καταγράφονται. Στηριζόμενοι στο παρελθόν για να προβλέψουμε το μέλλον, το μπλοκ που αντικαθίσταται είναι αυτό που δεν έχει χρησιμοποιηθεί για το μακρύτερο χρονικό διάστημα. Το LRU στηρίζεται σ' πένα συμπέρασμα της τοπικότητας: αν πρόσφατα χρησιμοποιημένα μπλοκ είναι πιθανόν να χρησιμοποιηθούν πάλι, τότε ένας καλός υποψήφιος για απόρριψη είναι το λιγότερο προσφάτως χρησιμοποιημένο μπλοκ.
- *Πρώτο μέσα, πρώτο έξω (first in, first out, FIFO).* Επειδή το LRU μπορεί να είναι περίπλοκο να το υπολογίσει κανείς, αυτό προσεγγίζει το LRU αποφασίζοντας το παλαιότερο μπλοκ αντί του LRU.

Ένα πλεονέκτημα της τυχαίας αντικατάστασης είναι ότι είναι απλή να την υλοποιήσεις στο υλικό. Καθώς ο αριθμός των μπλοκ που παρακολουθούνται αυξάνεται το LRU γίνεται βαθμιαία ακριβό και συχνά μόνο προσεγγίζεται.

Ε4: Τι συμβαίνει σε μια εγγραφή;
Οι αναγνώσεις κυριαρχούν στις προσπελάσεις κρυφής μνήμης του επεξεργαστή. Όλες οι προσπελάσεις εντολών είναι αναγνώσεις και οι περισσότερες εντολές δεν γράφουν στη μνήμη. Από την κυκλοφορία δεδομένων κρυφής μνήμης, οι εγγραφές είναι πολύ λιγότερες από τις αναγνώσεις. Επιταχύνοντας τη συνηθισμένη περίπτωση σημαίνει να βελτιστοποιήσει κανείς κρυφές μνήμες για αναγνώσεις, ειδικά αφού οι επεξεργαστές παραδοσιακά περιμένουν τις αναγνώσεις να ολοκληρωθούν ενώ δεν χρειάζεται να περιμένουν για εγγραφές.
Ο νόμος του Amdahl μας υπενθυμίζει παρόλα αυτά ότι σχέδια υψηλής απόδοσης δεν μπορούν να παραμελούν την ταχύτητα των εγγραφών. Για όσους δεν θυμούνται ο νόμος του Amdahl υποστηρίζει ότι η βελτίωση της απόδοσης που επιτυγχάνεται με τη χρήση κάποιας πιο γρήγορης μεθόδου εκτέλεσης περιορίζεται από το κλάσμα του χρόνου στον οποίο μπορεί να χρησιμοποιηθεί η γρηγορότερη αυτή μέθοδος.
Ευτυχώς η συνηθισμένη περίπτωση είναι επίσης και η εύκολη περίπτωση να την επιταχύνεις. Το μπλοκ μπορεί να διαβαστεί από την κρυφή μνήμη την ίδια στιγμή που διαβάζεται και συγκρίνεται η ετικέτα, έτσι η ανάγνωση του μπλοκ ξεκινά αμέσως μόλις η διεύθυνση του μπλοκ είναι διαθέσιμη. Αν η ανάγνωση είναι εύστοχη, το αιτούμενο κομμάτι του μπλοκ περνιέται στην ΚΜΕ αμέσως. Εάν είναι μια αστοχία, δεν υπάρχει όφελος αλλά επίσης ούτε ζημιά στους υπολογιστές γραφείου και στους εξυπηρετητές˙ απλώς αγνοούν την τιμή που αναγνώσθηκε.
Η έμφαση των ενσωματωμένων στην ισχύ γενικά σημαίνει να αποφεύγει κανείς μη απαραίτητη δουλειά, η οποία θα μπορούσε να οδηγήσει το σχεδιαστή να χωρίσει τα δεδομένα ανάγνωσης από τον έλεγχο της διεύθυνσης, έτσι ώστε τα δεδομένα να μην διαβάζονται σε μια αστοχία.
Τέτοια αισιοδοξία δεν επιτρέπεται για εγγραφές. Η αλλαγή σ' ένα μπλοκ δεν μπορεί να ξεκινήσει αν δεν ελεγχθεί η ετικέτα για να δούμε αν η διεύθυνση είναι εύστοχη. Επειδή ο έλεγχος της ετικέτας δεν μπορεί να συμβαίνει παράλληλα, οι εγγραφές φυσιολογικά παίρνουν περισσότερο χρόνο απ' ότι οι αναγνώσεις. Μια άλλη επιπλοκή είναι ότι ο επεξεργαστής επίσης προσδιορίζει το μέγεθος της εγγραφής συνήθως ανάμεσα σε 1 και 8 bytes˙ μόνο αυτό το τμήμα του μπλοκ μπορεί να αλλαχθεί. Αντιθέτως οι αναγνώσεις

μπορούν να προσπελάσουν περισσότερα bytes απ' ότι είναι απαραίτητο άφοβα˙ άλλη μια φορά οι σχεδιαστές ενσωματωμένων επεξεργαστών θα έπρεπε να ζυγίσουν το κέρδος ισχύος των λιγότερων αναγνώσεων.
Οι πολιτικές εγγραφών συχνά διακρίνουν τα σχέδια κρυφών μνημών. Υπάρχουν δύο βασικές επιλογές όταν γράφεις στην κρυφή μνήμη:

- *Διεγγραφή (write through)*. Η πληροφορία γράφεται και στο μπλοκ στην κρυφή μνήμη και στο μπλοκ στην μνήμη χαμηλότερου επιπέδου.
- *Ετεροχρονισμένη εγγραφή (write back)*. Η πληροφορία γράφεται μόνο στο μπλοκ στην κρυφή μνήμη. Το αλλαγμένο μπλοκ κρυφής μνήμης γράφεται στην κύρια μνήμη μόνο όταν αντικαθίσταται.

Για να μειώσουμε την συχνότητα των ετεροχρονισμένα εγγεγραμμένων μπλοκ στην αντικατάσταση, συχνά χρησιμοποιείται ένα χαρακτηριστικό που ονομάζεται bit αλλοίωσης.
Αυτό το bit κατάστασης υποδεικνύει αν το μπλοκ είναι αλλοιωμένο (αλλαγμένο ενώ είναι στην κρυφή μνήμη) ή καθαρό (μη αλλαγμένο). Αν είναι καθαρό, το μπλοκ δεν εγγράφεται ετεροχρονισμένα στην περίπτωση αστοχίας αφού η ίδια πληροφορία με αυτή της κρυφής μνήμης βρίσκεται σε χαμηλότερα επίπεδα.
Τόσο η ετεροχρονισμένη εγγραφή όσο και η διεγγραφή έχουν τα δικά τους πλεονεκτήματα η καθεμία. Με την ετεροχρονισμένη εγγραφή, οι εγγραφές συμβαίνουν με την ταχύτητα της κρυφής μνήμης και οι πολλαπλές εγγραφές μέσα σε ένα μπλοκ απαιτούν μόνο μία εγγραφή στη μνήμη χαμηλότερου επιπέδου. Αφού κάποιες εγγραφές δεν πηγαίνουν στην κύρια μνήμη, η ετεροχρονισμένη εγγραφή χρησιμοποιεί λιγότερο εύρος ζώνης μνήμης, κάνοντας την ετεροχρονισμένη εγγραφή ελκυστική στους πολυεπεξεργαστές. Οι οποίοι είναι συχνοί στους εξυπηρετητές. Αφού η ετεροχρονισμένη εγγραφή χρησιμοποιεί το υπόλοιπο της ιεραρχίας μνήμης και τους διαύλους μνήμης λιγότερο από την διεγγραφή επίσης εξοικονομεί ισχύ, όντας πιο ελκυστική για ενσωματωμένες εφαρμογές.
Η διεγγραφή είναι ευκολότερη να υλοποιηθεί απ' ότι η ετεροχρονισμένη εγγραφή. Η κρυφή μνήμη είναι πάντα καθαρή, οπότε αντίθετα με τις αστοχίες ανάγνωσης στην περίπτωση της ετεροχρονισμένης εγγραφής ποτέ δεν έχουν ως αποτέλεσμα εγγραφές στο χαμηλότερο επίπεδο. Η διεγγραφή έχει επίσης το πλεονέκτημα ότι το αμέσως χαμηλότερο επίπεδο έχει το πιο πρόσφατο αντίγραφο των δεδομένων, το οποίο απλοποιεί την συμβατότητα των δεδομένων.
Όταν η ΚΜΕ πρέπει να περιμένει να ολοκληρωθούν οι εγγραφές στη διάρκεια της διεγγραφής, η ΚΜΕ λέγεται ότι είναι σε καθυστέρηση εγγραφής. Μια συνηθισμένη βελτιστοποίηση για να μειώσει κανείς τις καθυστερήσεις εγγραφών είναι μια μνήμη εγγραφών η οποία επιτρέπει στον επεξεργαστή να συνεχίσει αμέσως μόλις τα δεδομένα έχουν γραφεί στη μνήμη επιτυγχάνοντας έτσι παραλληλισμό στην εκτέλεση του επεξεργαστή με την ενημέρωση της μνήμης.
Καθώς τα δεδομένα δεν χρειάζονται σε μια εγγραφή, υπάρχουν δύο επιλογές σε μια αστοχία εγγραφής:

- Κατανομή εγγραφής (write allocate). Το μπλοκ κατανέμεται σε μια αστοχία εγγραφής ακολουθούμενο από τις παραπάνω ενέργειες εύστοχης εγγραφής. Σ' αυτή τη φυσική επιλογή, οι αστοχίες εγγραφών λειτουργούν ως αστοχίες αναγνώσεων.
- Κατανομή μη-εγγραφής (no-write allocate). Αυτή η προφανώς ασυνήθιστη εναλλακτική λύση είναι ότι οι αστοχίες εγγραφής δεν επηρεάζουν την κρυφή μνήμη. Αντιθέτως το μπλοκ αλλάζει μόνο στη μνήμη χαμηλότερου επιπέδου.

Έτσι, τα μπλοκ μένουν έξω από την κρυφή μνήμη στην κατανομή μη-εγγραφής μέχρι το πρόγραμμα να προσπαθήσει να διαβάσει τα μπλοκ αλλά ακόμα και τα μπλοκ που είναι μόνο γραμμένα θα είναι ακόμα στην κρυφή μνήμη με την κατανομή εγγραφής.

Η πολιτική της αστοχίας εγγραφής θα μπορούσε να χρησιμοποιηθεί είτε με διεγγραφή ή με ετεροχρονισμένη εγγραφή. Φυσιολογικά οι κρυφές μνήμες ετεροχρονισμένης εγγραφής χρησιμοποιούν κατανομή εγγραφής, ελπίζοντας ότι οι ακόλουθες εγγραφές σ' αυτό το μπλοκ θα συλληφθούν από την κρυφή μνήμη. Οι κρυφές μνήμες διεγγραφής συνήθως χρησιμοποιούν κατανομή μη-εγγραφής. Η λογική είναι ότι ακόμα και αν υπάρχουν ακόλουθες εγγραφές σ' αυτό το μπλοκ, οι εγγραφές πρέπει να πάνε σε ακόμα χαμηλότερο επίπεδο, έτσι τι θα κερδηθεί;

ΚΕΦΑΛΑΙΟ 2

ΠΡΟΔΙΑΓΡΑΦΕΣ ΚΡΥΦΗΣ ΜΝΗΜΗΣ

Το σύστημα που θα φτιάξουμε πρέπει να είναι σύμφωνο με ορισμένες προδιαγραφές. Οι προδιαγραφές αποτελούν πολλές φορές δύσκολα εμπόδια που πρέπει να ξεπεράσει ο μηχανικός προκειμένου να έχει το επιθυμητό αποτέλεσμα. Από μια άλλη πλευρά όμως οι προδιαγραφές είναι ένας οδηγός που βοηθούν το σχεδιαστή να διαμορφώσει και να ακολουθήσει ένα συγκεκριμένο μονοπάτι (χωρίς να σημαίνει ότι είναι το πλέον κατάλληλο) προς το στόχο του.
Στη συγκεκριμένη προσπάθεια η θετική όψη των προδιαγραφών έγινε ιδιαίτερα φανερή στην αρχή της σχεδίασης όπου δεν ήξερα από που και πως να ξεκινήσω (κάθε αρχή και δύσκολη). Όσο για την αρνητική όψη των προδιαγραφών θα γίνει αντιληπτή καθόλη την καταγραφή της προσπάθειας.
Αυτό που θέλουμε να φτιάξουμε είναι ένα σύστημα κρυφής μνήμης ενός επιπέδου. Το αμέσως χαμηλότερο επίπεδο μνήμης θα είναι δηλαδή η κύρια μνήμη. Αυτό από μόνο του δίνει κάποιες αυτονόητες προδιαγραφές όπως:
Κατά την εγγραφή, όταν δίνεται στο σύστημα μια διεύθυνση και μια τιμή δεδομένων πρέπει να ενημερώνεται το κατάλληλο πεδίο(εκείνο που αντιστοιχεί στη συγκεκριμένη διεύθυνση) της κρυφής μνήμης ή της κύριας μνήμης με τη τιμή αυτή.
Κατά την ανάγνωση, θα πρέπει όταν δίνεται στο σύστημα μια διεύθυνση πρέπει να δίνεται σαν απάντηση από το σύστημα η σωστή τιμή δεδομένων της διεύθυνσης αυτής.
Ένα χαρακτηριστικό που πρέπει να επισημάνουμε είναι ότι **το σύστημα που θα δημιουργηθεί θα είναι σύγχρονο**. Αυτό σημαίνει ότι **ο χρόνος θα μετριέται σε κύκλους ρολογιού**. Άρα:
Κάθε αίτηση για εγγραφή ή ανάγνωση θα έχει διάρκεια έναν κύκλο. Αυτό στην πράξη θα υλοποιηθεί με δύο σήματα εισόδου: ένα για ανάγνωση και ένα για εγγραφή.
Όπως είναι λογικό θα χρησιμοποιηθούν κάποια μοντέλα μνήμης που θα παίζουν το ρόλο του αποθηκευτικού μέρους της κρυφής μνήμης αλλά και την αναπαράσταση της κύριας μνήμης. Τα μοντέλα αυτά θα είναι επίσης σύγχρονα, δηλαδή η πρόσβαση σε αυτά κατά την ανάγνωση ή την εγγραφή θα γίνεται σε ένα κύκλο ρολογιού.
Από τη θεωρία γνωρίζουμε ότι μία αίτηση ανάγνωσης στην κρυφή μνήμη μπορεί να καταλήξει σε ευστοχία ή σε αστοχία ανάλογα με το αν η συγκεκριμένη εγγραφή υπάρχει εκείνη τη στιγμή στην κρυφή μνήμη. Σε περίπτωση ευστοχίας η απάντηση της κρυφής μνήμης μπορεί θεωρητικά να δοθεί το γρηγορότερο σε ένα κύκλο. Αυτή είναι και η πρώτη αυστηρή προδιαγραφή του μοντέλου μας. Δηλαδή όταν έχουμε ευστοχία τα δεδομένα που ζητούνται θα πρέπει να έρχονται ως έξοδος στο τέλος του κύκλου αίτησης. Αυτό αλλιώς μπορεί να περιγραφεί ως εξής:
Το μοντέλο θα πρέπει να ικανοποιεί αιτήσεις ανάγνωσης που έρχονται με ρυθμό μία αίτηση ανά εντολή, δεδομένου ότι όλες οι αιτήσεις καταλήγουν σε ευστοχία κρυφής μνήμης.
Κατά την αίτηση εγγραφής πρέπει πρώτα να γίνει μία ανάγνωση (για να εξερευνηθεί αν υπάρχει ή όχι το ζητούμενο μπλοκ στην κρυφή μνήμη) και κατόπιν μια εγγραφή. Αυτό σημαίνει ότι αφού όλα γίνονται σύγχρονα, όπως είπαμε πριν, μία αίτηση εγγραφής μπορεί θεωρητικά να εξυπηρετηθεί το γρηγορότερο σε δύο κύκλους εφόσον προκύψει ευστοχία. Αυτό θα μπορούσε να μας οδηγήσει στο συμπέρασμα ότι αρκεί το μοντέλο να ικανοποιεί αιτήσεις εγγραφής με ρυθμό εισόδου μία αίτηση ανά δύο κύκλους. Όμως μια πιο προσεκτική προσέγγιση μπορούμε να δούμε ότι: κατά την ανάγνωση αυτό που ελέγχεται αν υπάρχει είναι η διεύθυνση του μπλοκ, ενώ κατά την εγγραφή αυτό που γράφεται είναι τα δεδομένα του μπλοκ. Αν λοιπόν διαχωρίζαμε με κάποιο τρόπο τις

διευθύνσεις των μπλοκ(μνήμη ετικετών) από τα καθαυτό μπλοκ(μνήμη δεδομένων) θα μπορούσαμε να επιτύχουμε ταυτόχρονή πρόσβαση ανάγνωσης στην μνήμη των ετικετών και πρόσβαση εγγραφής στη μνήμη δεδομένων. Η πρώτη πρόσβαση θα ικανοποιεί το πρώτο στάδιο μιας αίτησης ενώ η δεύτερη πρόσβαση θα ικανοποιεί το δεύτερο στάδιο της προηγούμενης αίτησης. Παρόλο λοιπόν που μεμονωμένα μία αίτηση εγγραφής απαιτεί δύο κύκλους(στάδια) για να ικανοποιηθεί:
Το μοντέλο θα πρέπει να ικανοποιεί αιτήσεις εγγραφής που έρχονται με ρυθμό μία αίτηση ανά εντολή, δεδομένου ότι όλες οι αιτήσεις καταλήγουν σε ευστοχία κρυφής μνήμης.
Οι δύο παραπάνω προδιαγραφές πηγάζουν από την γενική αρχή «να δουλεύει σωστά και όσο πιο γρήγορα γίνεται» αφού «ο χρόνος είναι χρήμα».
Στη θεωρία είδαμε ότι υπάρχουν δύο βασικές επιλογές όταν γράφεις στην κρυφή μνήμη: η διεγγραφή και η ετεροχρονισμένη εγγραφή.
Επίσης είδαμε ότι υπάρχουν δύο στρατηγικές σε μία αστοχία εγγραφής: η κατανομή εγγραφής και η κατανομή μη εγγραφής.
Εμείς πρέπει να φτιάξουμε τα δύο «φυσιολογικά» μοντέλα δηλαδή αυτό της διεγγραφής με κατανομή μη εγγραφής και αυτό της ετεροχρονισμένης εγγραφής με κατανομή εγγραφής. Ακόμα πρέπει να φτιάξουμε και ένα μοντέλο διεγγραφής με κατανομή εγγραφής. Το τέταρτο μοντέλο που θα ήταν εκείνο της ετεροχρονισμένης εγγραφής με κατανομή μη εγγραφής δεν έχει λογική βάση αφού:
αν είχαμε μια αστοχία μνήμης δεν θα κατανέμαμε το μπλοκ στην κρυφή μνήμη και θα χάναμε πιθανές επόμενες ευστοχίες τις οποίες θέλει να αξιοποιήσει η ετεροχρονισμένη εγγραφή μειώνοντας τις εγγραφές στην κύρια μνήμη.
Άρα μπορεί πολλές φορές να μιλάμε για «το σύστημα» κρυφής μνήμης στην πραγματικότητα όμως θα εννοούμε τα τρία διαφορετικά συστήματα κρυφής μνήμης:
1. Διεγγραφής με κατανομή μη εγγραφής
2. Διεγγραφής με κατανομή εγγραφής
3. Ετεροχρονισμένης εγγραφής με κατανομή εγγραφής

Εκτός όμως από την κανονική λειτουργία της μνήμης που είναι το διάβασμα και το γράψιμο το σύστημά μας θα πρέπει να υποστηρίζει και μία ακόμα λειτουργία:
την **αρχικοποίηση**.
Περιγράψαμε ως εδώ τις σημαντικότερες προδιαγραφές του συστήματος κρυφής μνήμης. Μένει μόνο να πούμε που θα αναφέρεται ο όρος «παραμετρική». Τρεις θα είναι οι παράμετροι του συστήματος που ο χρήστης θα μπορεί να δίνει ως είσοδο προκειμένου να αποκτήσει την επιθυμητή κρυφή μνήμη:
Το μέγεθος της κρυφής μνήμης
Το μέγεθος του μπλοκ
Ο βαθμός συνολοσυσχετιστικότητας

Οι μόνες σταθερές που μας δίνονται από τις προδιαγραφές είναι:
Το μέγεθος της διεύθυνσης είναι 64 bits.
Το μέγεθος της λέξης(word) του συστήματος είναι 32 bits.
Η διεύθυνση του συστήματος είναι διεύθυνση byte όπως συνηθίζεται. Οι αιτήσεις ανάγνωσης και εγγραφής όμως θα αναφέρονται σε μία ολόκληρη λέξη.

ΚΕΦΑΛΑΙΟ 3

ΠΡΟΣΧΕΔΙΑΣΤΙΚΗ ΜΕΛΕΤΗ

Πριν να ξεκινήσει η σχεδίαση του συστήματος είναι χρήσιμο να μελετηθούν οι επιπτώσεις που έχει η εισαγωγή παραμέτρων σε αυτήν. Κατά την μελέτη αυτή προσδιορίσα πλήρως όλα τα μεγέθη που θα αποκτούσαν παραμετρικό χαρακτήρα ώστε να γίνει εφικτή η υποστήριξη των τριών παραμέτρων. Αφού βρήκα τις μαθηματικές σχέσεις που εμπλέκουν όλες τις μεταβλητές του προβλήματος, καθορίσα τα πεδία τιμών που θα επέτρεπα να πάρουν οι μεταβλητές αυτές. Πρέπει να σημειώσω ότι η μελέτη ήταν όσο πιο γενική γινόταν γι' αυτό και φαίνονται ως μεταβλητές ακόμα και το μήκος της διεύθυνσης σε bits, παρόλο που στην περίπτωση της εργασίας είναι γνωστή και σταθερή (64 bits). Όλα αυτά εξηγούνται αναλυτικά παρακάτω.

Ανεξάρτητες μεταβλητές
Οι ανεξάρτητες μεταβλητές του προβλήματος είναι:

ΟΝΟΜΑ ΜΕΤΑΒΛΗΤΗΣ	ΜΟΝΑΔΑ ΜΕΤΡΗΣΗΣ	ΜΑΘΗΜΑΤΙΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ
Μέγεθος διεύθυνσης	bits	x
Μέγεθος κρυφής μνήμης	bytes	z
Μέγεθος μπλοκ	bytes	y
Βαθμός συνολοσυσχετιστικότητας	-	k

Περιορισμοί
Αφού το μέγεθος της διεύθυνσης είναι x bits και κάθε μία διεύθυνση αναφέρεται σε ένα συγκεκριμένο byte, συμπεραίνουμε ότι το μέγεθος της κύριας μνήμης θα έχει μέγιστο μέγεθος 2^x bytes. Αν είχε μεγαλύτερο μέγεθος τότε κάποιο κομμάτι της κύριας μνήμης δεν θα μπορούσε να προσπελαστεί βάσει της διεύθυνσης. Η κρυφή μνήμη είναι από τη θεωρία ένα υποσύνολο της κύριας μνήμης άρα και για το μέγεθος της κρυφής μνήμης ισχύει η ανισότητα $z < 2^x$.
Το μέγεθος του μπλοκ με τη σειρά του δεν μπορεί να είναι μεγαλύτερο από το μέγεθος της κρυφής μνήμης και δεν μπορεί να είναι ούτε ίσο με αυτό. Κάτι τέτοιο δεν θα είχε νόημα. Άρα θα ισχύει η ανισότητα $y < z$.
Μια τρίτη ανισότητα πηγάζει από την διερεύνηση της έννοιας του βαθμού συνολοσυσχετιστικότητας. Η διάσπαση σε σύνολα μπορεί να αυξάνεται ώσπου να προκύψει μια πλήρως συσχετιστική κρυφή μνήμη όπου ο βαθμός συνολοσυσχετιστικότητας θα είναι ίσος με τον αριθμό των μπλοκ της μνήμης. Κατά συνέπεια αφού ο αριθμός των μπλοκ ισούται με z/y πρέπει να ισχύει η ανισότητα $k \le \frac{z}{y}$.

Εξαρτημένες μεταβλητές
Τώρα θα καταγραφούν όλες οι μεταβλητές που υπολογίζονται από τις παραπάνω.
Η διεύθυνση x χωρίζεται σύμφωνα με τη θεωρία στις διευθύνσεις:
1. Διεύθυνση μπλοκ
2. Απόκλιση μπλοκ

Αν η απόκλιση μπλοκ είναι m bits τότε πρέπει να ισχύει ο τύπος:

$$m = \log_2 y$$

Το μέγεθος της απόκλισης μπλοκ εξαρτάται μόνο από το μέγεθος του μπλοκ.
Στην πραγματικότητα ακριβέστερη σχέση θα ήταν αυτή με το άνω όριο του λογαρίθμου αφού στην περίπτωση που το μέγεθος μπλοκ δεν είναι δύναμη του δύο το αποτέλεσμα δεν θα ήταν ακέραιο. Επειδή όμως το m πρέπει να είναι ακέραιος αριθμός και ικανός για να διευθυνσιοδοτήσει κάθε byte του μπλοκ θα ορίζεται από το άνω όριο του $\log_2 y$. Παρόλα αυτά επειδή θέλω να απλοποιήσω τα πράγματα δεν χρησιμοποιώ το άνω όριο και αντ' αυτού εισάγω τον περιορισμό: το μέγεθος του μπλοκ πρέπει να είναι δύναμη του 2.

Αν η διεύθυνση μπλοκ είναι n bits τότε έχουμε τη σχέση:

$$n = x - m \Rightarrow n = x - \log_2 y$$

Το μέγεθος της διεύθυνσης μπλοκ εξαρτάται από το μέγεθος διεύθυνσης και από το μέγεθος του μπλοκ.
Η διεύθυνση μπλοκ με τη σειρά της χωρίζεται στα δύο μέρη:
1. Ετικέτα
2. Δείκτης

Από τη θεωρία γνωρίζουμε ότι αν ο δείκτης είναι p bits θα ισχύει ο τύπος:

$$2^p = \frac{z}{y \times k} \Rightarrow p = \log_2\left(\frac{z}{y \times k}\right)$$

Το μέγεθος του δείκτη εξαρτάται από το μέγεθος της κρυφής μνήμης, το μέγεθος του μπλοκ και το βαθμό συνολοσυσχετιστικότητας. Για να έχει νόημα το αποτέλεσμα θα πρέπει να ισχύει η σχέση $k \le \frac{z}{y}$, την οποία είχαμε και θεωρητικά υποστηρίξει προηγουμένως.
Ο τύπος αυτός δείχνει ξεκάθαρα ότι το p είναι δύναμη του 2.

Αν η ετικέτα είναι t bits τότε έχουμε τη σχέση:

$$t = n - p \Rightarrow t = (x - \log_2 y) - \left(\log_2\left(\frac{z}{y \times k}\right)\right) \Rightarrow t = x - \left(\log_2 y + \log_2\left(\frac{z}{y \times k}\right)\right)$$

$$\Rightarrow t = x - \log_2\left(y \times \frac{z}{y \times k}\right) \Rightarrow t = x - \log_2\left(\frac{z}{k}\right)$$

Το μέγεθος της ετικέτας εξαρτάται από το μέγεθος διεύθυνσης, το μέγεθος της κρυφής μνήμης και το βαθμό συνολοσυσχετιστικότητας. Δεν εξαρτάται από το μέγεθος του μπλοκ.

Από τον προηγούμενο τύπο προκύπτει η σχέση $\frac{z}{k} = 2^a$, που πρέπει να ισχύει ανάμεσα στο μέγεθος της κρυφής μνήμης και το βαθμό συνολοσυσχετιστικότητας.

Σύνοψη

Στον παρακάτω πίνακα δίνονται συνοπτικά όλες οι μεταβλητές του προβλήματος μαζί με τους περιορισμούς τους. Φυσικά αν μια μεταβλητή υπόκειται σε δύο περιορισμούς όπου ο ένας είναι υποσύνολο του άλλου, κρατάμε τον πιο γενικό-αυστηρό περιορισμό.

ΟΝΟΜΑ ΜΕΤΑΒΛΗΤΗΣ	ΜΟΝΑΔΑ ΜΕΤΡΗΣΗΣ	ΜΑΘΗΜΑΤΙΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ	ΜΑΘΗΜΑΤΙΚΟΙ ΠΕΡΙΟΡΙΣΜΟΙ	ΜΑΘΗΜΑΤΙΚΟΣ ΤΥΠΟΣ
Μέγεθος διεύθυνσης	bits	x	-	-
Μέγεθος κρυφής μνήμης	bytes	z	$z < 2^x$ $z = k \times 2^a$	-
Μέγεθος μπλοκ	bytes	y	$y < z$ $y = 2^a$	-
Βαθμός συνολο συσχετιστικότητας	-	k	$k \le \frac{z}{y}$ $k = \frac{z}{2^a}$	-
Απόκλιση μπλοκ	bits	m	-	$\log_2 y$
Διεύθυνση μπλοκ	bits	n	-	$x - \log_2 y$
Δείκτης	bits	p	-	$\log_2\left(\frac{z}{y \times k}\right)$
Ετικέτα	bits	t	-	$x - \log_2\left(\frac{z}{k}\right)$

Όπου a είναι ένας φυσικός αριθμός.

Ο πίνακας δεν δείχνει τους μαθηματικούς περιορισμούς που μπαίνουν στις εξαρτημένες μεταβλητές. Αυτοί προέρχονται εγγενώς από την εξάρτηση που έχουν από τις πρώτες 4 μεταβλητές. Επειδή όμως το σύστημα που θα υλοποιηθεί θα παίρνει ως είσοδο μόνο τις ανεξάρτητες μεταβλητές δεν χρειάζεται να καταγράψουμε τους εν λόγω περιορισμούς καθώς μια επιτρεπτή ομάδα μεταβλητών εισόδου θα δώσει εξ ορισμού και επιτρεπτές τιμές και στις υπόλοιπες μεταβλητές. Αυτό λοιπόν που πρέπει να κάνει αρχικά το σύστημα είναι να ελέγχει τις τιμές εισόδου. Αν ακολουθούν τους μαθηματικούς περιορισμούς, τότε θα προχωράει στον υπολογισμό των υπολοίπων μεταβλητών σύμφωνα με τους μαθηματικούς τύπους. Αλλιώς αν δεν ικανοποιεί έστω και μία μεταβλητή εισόδου τους περιορισμούς του πίνακα, τότε θα ειδοποιείται ο χρήστης ότι πρέπει να αλλάξει τις τιμές εισόδου προκειμένου να προχωρήσει η υλοποίηση.

ΚΕΦΑΛΑΙΟ 4

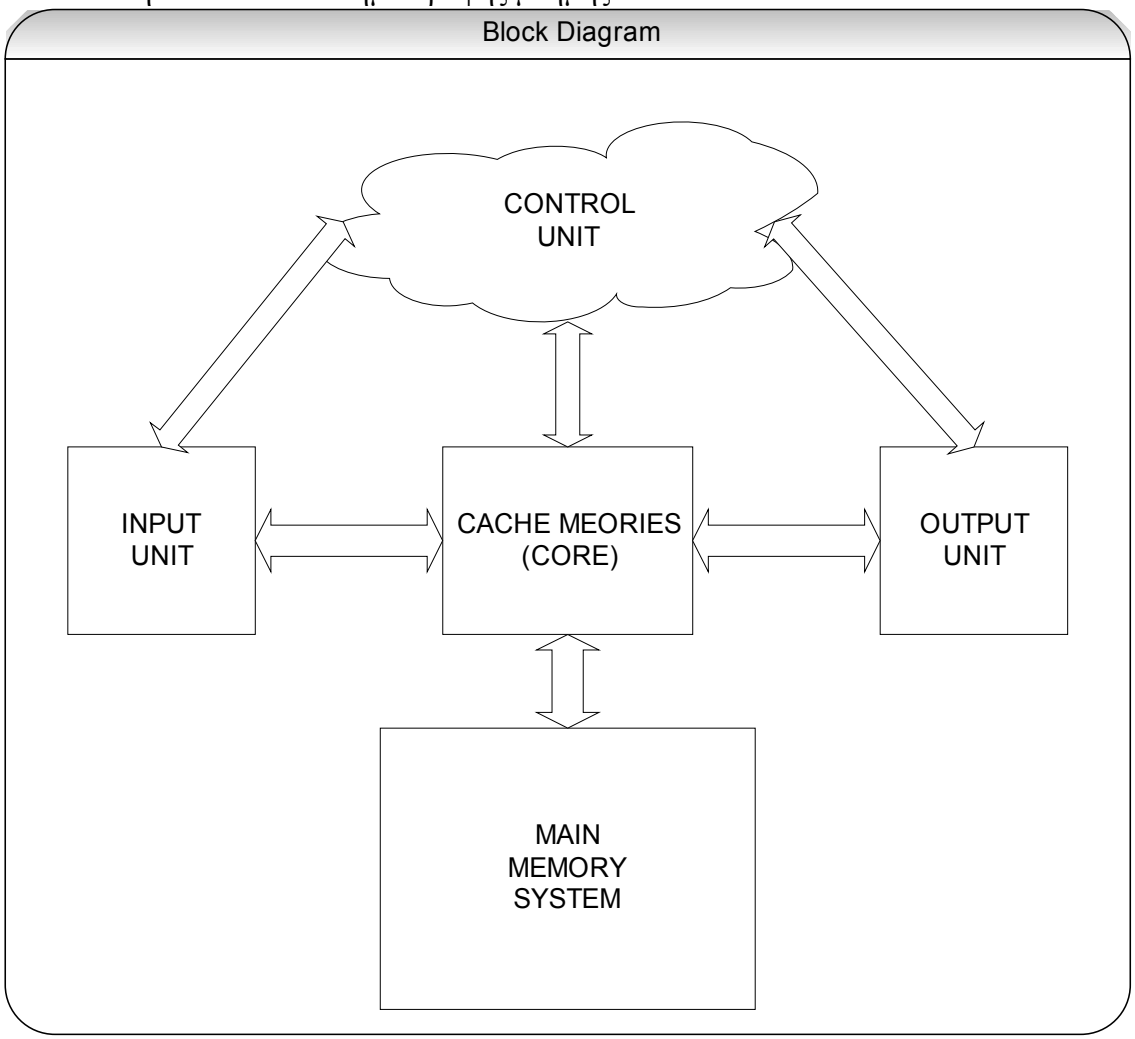
ΣΧΕΔΙΑΣΗ

Στο κεφάλαιο αυτό αναλύω το πιο δημιουργικό κομμάτι της εργασίας που είναι η σχεδίαση του συστήματος. Η παρουσίαση της σχεδίασης θα γίνει με μεταβάσεις από μια γενική εικόνα προς μια πιο ειδική. Εκτός από την περιγραφή των κυκλωμάτων της κρυφής μνήμης, κάποια θέματα θα μελετηθούν ξεχωριστά. Αυτά είναι:

Α. Η αρχικοποίηση
Β. Η υποστήριξη ετεροχρονισμένης εγγραφής
Γ. Ο έλεγχος ροής δεδομένων
Δ. Η δημιουργία ενός ολοκληρωμένου συστήματος

4.1 Διάγγραμμα μπλοκ

Στο παρακάτω σχήμα φαίνεται η γενική εικόνα του συστήματος που σχεδιάστηκε. Η μορφή είναι απλοποιημένη και απλώς δείχνει τα μεγάλα σύνολα που με τη κατάλληλη διασύνδεση δίνουν το σύστημα κρυφής μνήμης.



Στο κέντρο της σχεδίασης υπάρχει ο «πυρήνας» που είναι τα κομμάτια μνήμης που συγκροτούν το αποθηκευτικό χώρο του συστήματος. Όπως θα φανεί παρακάτω ο πυρήνας αποτελείται από σύνολα (sets), ο αριθμός των οποίων ορίζεται παραμετρικά. Το κουτί που προηγείται του πυρήνα είναι η μονάδα των εισόδων του συστήματος.

Έχει σαν σκοπό να παίρνει τα σήματα του έξω κόσμου (επεξεργαστής), να τα διαχειρίζεται με κατάλληλο τρόπο και να τα προωθεί προς τον πυρήνα. Αντίστοιχα η μονάδα εξόδων, που έπεται του πυρήνα, είναι υπεύθυνη να παράγει τα κατάλληλα σήματα απάντησης προς τον έξω κόσμο. Όπως φαίνεται και στο σχήμα όλα τα επιμέρους συτήματα ελέγχονται από μια μονάδα ελέγχου που στην πράξη είναι μια μηχανή πεπερασμένων καταστάσεων. Αν και σε ένα σύτημα κρυφής μνήμη δεν έχει θέση η κύρια μνήμη, εγώ την έχω προσθέση για να μπορεί να γίνει η μεταφορά δεδομένων από και προς αυτήν. Έτσι ο πυρήνας ζητά δεδομένα από την κύρια μνήμη σε μια αστοχία (miss) ενώ άλλες φορές στέλνει αυτός δεδομένα προς εγγραφή (write back).

4.2 Διεπαφή του συστήματος

Η κρυφή μνήμη επικοινωνεί σαν σύστημα με τον επεξεργαστή και με το επόμενο επίπεδο της ιεραρχίας του συστήματος μνήμης. Εγώ όπως έχω δείξει έχω συμπεριλάβει την κύρια μνήμη στο σύστημά μου, οπότε η μόνη εξωτερική διεπαφή είναι αυτή με τον επεξεργαστή και είναι η ακόλουθη:

Είσοδοι

Το σύστημα δέχεται τρία σήματα ελέγχου και δύο σήματα δεδομένων από τον επεξεργαστή:

1. Ένα σήμα αρχικοποίησης (reset) της κρυφής μνήμης.
2. Ένα σήμα που ειδοποιεί ότι γίνεται μια αίτηση ανάγνωσης (1 bit).
3. Ένα σήμα που ειδοποιεί ότι γίνεται μια αίτηση εγγραφής (1 bit).
4. Η διεύθυνση ανάγνωσης ή εγγραφής ανάλογα με τον τύπο της αίτησης (64 bits).
5. Τα δεδομένα που πρέπει να γραφτούν στην διεύθυνση εγγραφής (32 bits).

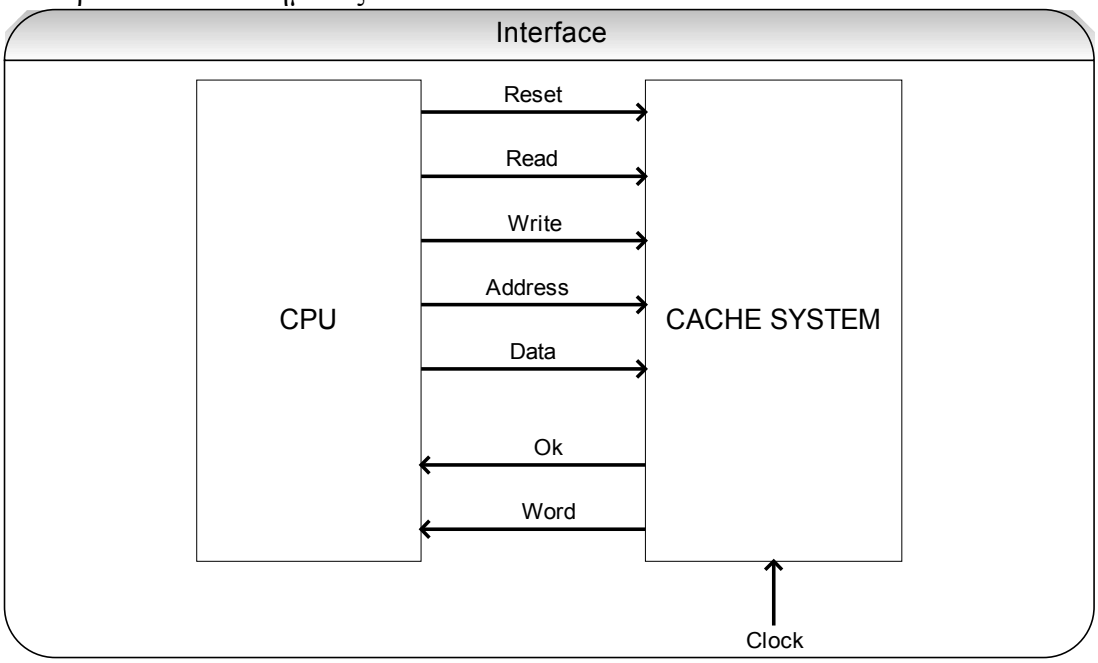
Έξοδοι

Το σύστημα στέλνει προς τον επεξεργαστή ένα σήμα ελέγχου και ένα σήμα δεδομένων:

1. Ένα σήμα που ειδοποιεί ότι ολοκληρώθηκε η εξυπηρέτιση της αίτησης (1 bit).
Ο επεξεργαστής μπορεί να κάνει μια νέα αίτηση μόνο αφού έχει δεχτεί αυτό το σήμα για την προηγούμενη.
2. Τα δεδομένα που διαβάστηκαν για την εξυπηρέτηση της αίτηση ανάγνωσης (32 bits).
Η τιμή τους οφείλει να είναι σωστή όταν το προηγούμενο σήμα είναι ενεργό.

Το σύστημα παίρνει εξωτερικά ένα ακόμα σήμα:

1. Το ρολόι του συστήματος.

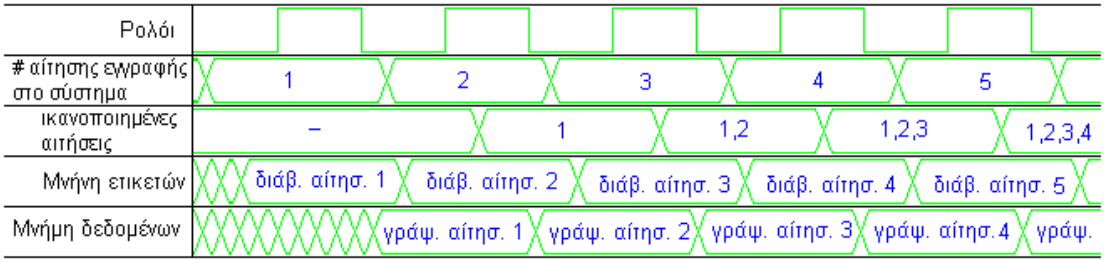


4.3 Διασύνδεση του πυρήνα

4.3.1 Εξυπηρέτηση αιτήσεων που καταλήγουν σε ευστοχία.

Τώρα θα δούμε με ποια συνδεσμολογία μπορεί ο πυρήνας να εξυπηρετήσει αιτήσεις ανάγνωσης ή εγγραφής με αναφορές σε διευθύνσεις που υπάρχουν στην κρυφή μνήμη (περίπτωση hit). Ο λόγος που χωρίσα τις ετικέτες από τα δεδομένα είναι η επίτευξη ρυθμού εξυπηρέρισης 1 αίτηση εγγραφής ανά κύκλο. Υποθέτουμε βέβαια ότι όλες οι αιτήσεις καταλήγουν σε ευστοχία για να έχουμε αυτήν την απόδοση. Η αναλυτικότερη περιγραφή του διαχωρισμού θα περιγραφεί στην ενότητα που παρουσιάζεται το εσωτερικό του πυρήνα.

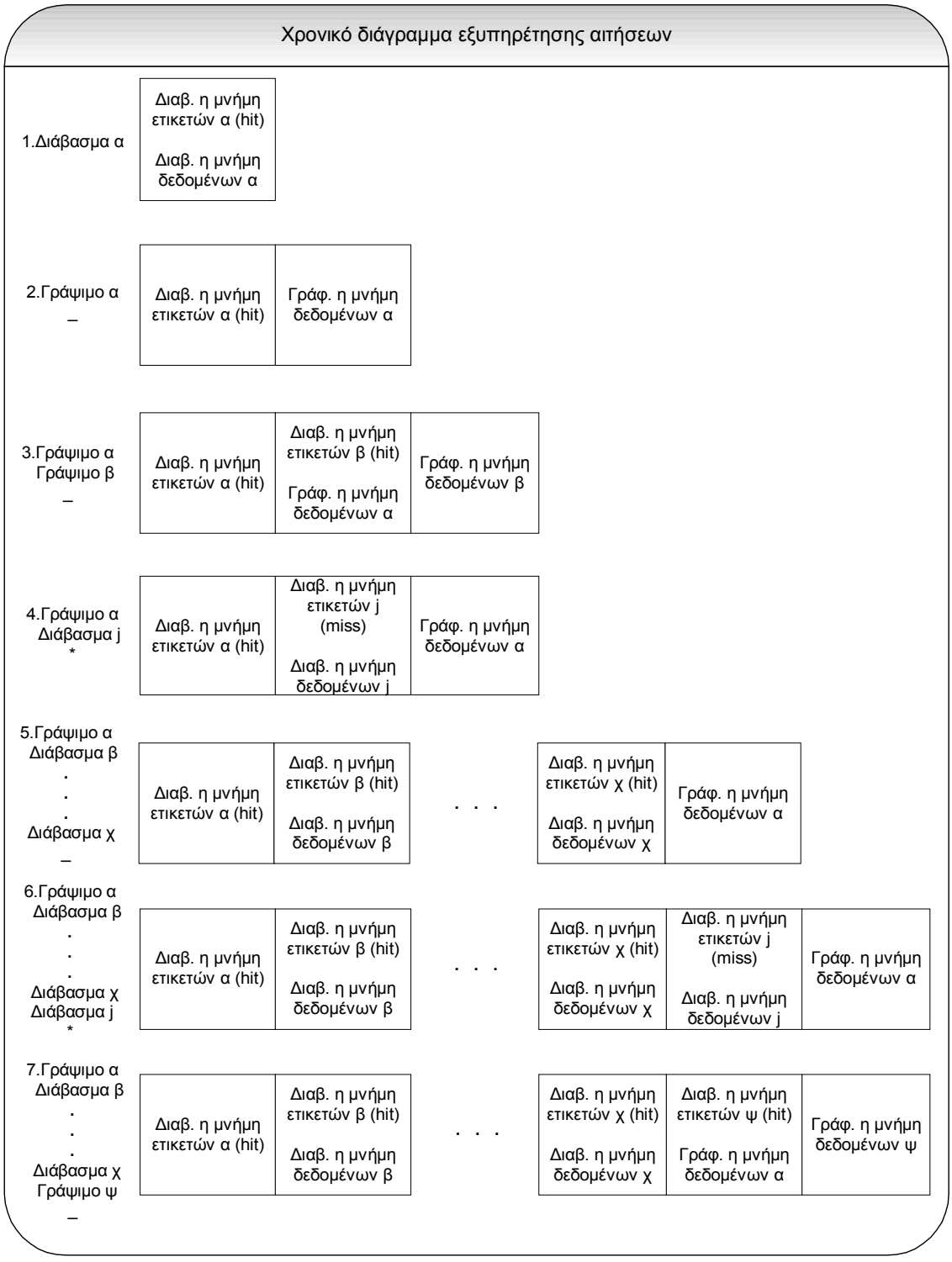
Για καλύτερη αναπαράσταση δίνουμε το ακόλουθο διάγραμμα χρονισμού που δείχνει πως πρέπει να ικανοποιούνται συνεχόμενες αιτήσεις εγγραφής.



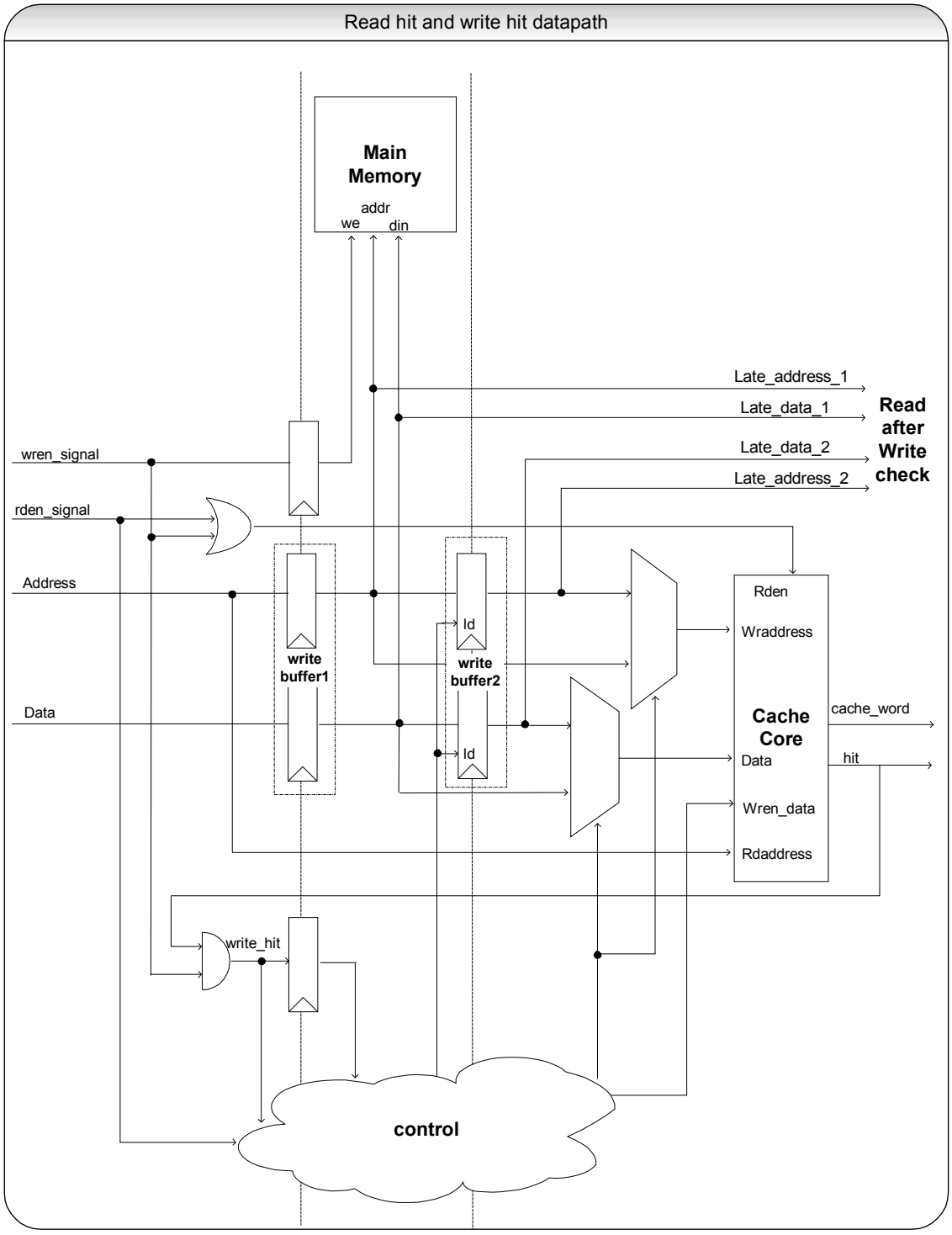
Για να επιτευχθεί η παραπάνω λειτουργικότητα χρειάζεται να κρατάμε αποθηκεμένα τη διεύθυνση εγγραφής και τα δεδομένα για ένα κύκλο, αφού το γράψιμο γίνεται στο δεύτερο κύκλο. Για το σκοπό αυτό χρησιμοποιούμε ένα καταχωρητή με είσοδο τη διεύθυνση αίτησης και έναν άλλο με είσοδο τα δεδομένα της αίτησης.

Υπάρχουν όμως και άλλες αληλουχίες αιτήσεων που χρειάζονται διαφορετική προσέγγιση. Για παράδειγμα αν ανάμεσα σε εγγραφές υπάρχει μια αίτηση εγγραφής, τότε υπάρχει πρόβλημα αφού όταν διαβάζουμε τη μνήμη δεδομένων για την αίτηση ανάγνωσης δεν μπορούμε ταυτόχρονα να κάνουμε την εγγραφή της προηγούμενης αίτησης εγγραφής (αφού η μνήμη είναι μονόπορτη). Αν είχαμε δίπορτη μνήμη δεδομένων αυτό θα μπορούσε να γίνει. Όμως επειδή οι δίπορτες μνήμες δεν ακολουθούν τον κανόνα της μινιμαλιστικής σχεδίασης, που πρέπει να δουλεύει και εκτός FPGA δεν ακολούθησα αυτή την επιλογή. Η ολοκλήρωση της εγγραφής θα γίνει λοιπόν στο πρώτο κύκλο που θα είναι διαθέσιμη η μνήμη δεδομένων. Αυτό εισάγει στη σχεδίαση ένα ακόμα σετ καταχωρητών που κρατάει την χρήσιμη πληροφορία για την εγγραφή που κάποια στιγμή πρέπει να γίνει.

Παρακάτω ακολουθεί ένα διάγραμμα χρονισμού με 7 διαφορετικά σενάρια αλληλουχιών αιτήσεων που δείχνει πως πρέπει το σύστημα να τις ικανοποιεί. Όλες οι αιτήσεις θεωρούμε ότι καταλήγουν σε ευστοχία εκτός από αυτές που το γράμμα τους ανήκει μόνο στο λατινικό αλφάβητο. Οι τρεις τελείες σημαίνουν ότι παρεμβάλονται καμία, μία ή περισσότερες ίδιες αιτήσεις με την προηγούμενη. Η παύλα σημαίνει ότι δεν ακολουθεί κάποια αίτηση, ενώ με τον αστερίσκο σηματοδοτείται οποιαδήποτε αίτηση μπορεί να υπάρξει στο σύστημα..



Για να ικανοποιήσουμε όλα τα παραπάνω κατέληξα στην σύδεση του πυρήνα με τα κατάλληλα σήματα έτσι όπως φαίνεται στο ακόλουθο σχήμα.

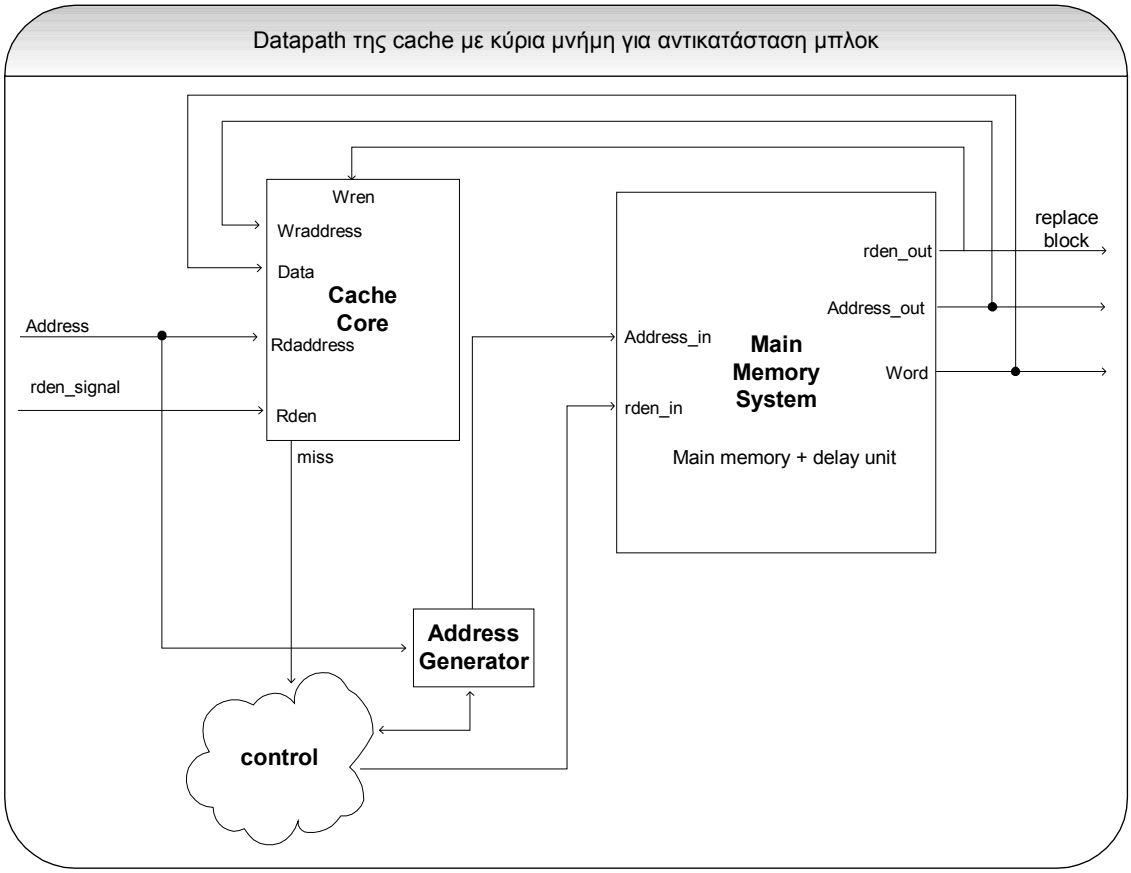


Η τοποθέτηση των καταχωρητών χωρίζουν το σχήμα σε τρεις ζώνες. Κάθε ζώνη δίνει στην κρυφή μνήμη τα κατάλληλα σήματα ανάλογα με την ενέργεια που πρέπει να γίνει. Η πρώτη ζώνη, στην οποία παίρνουμε τα σήματα που έρχονται από τον έξω κόσμο, δίνει στην κρυφή μνήμη το σήμα ανάγνωσης και την αντίστοιχη διεύθυνση ανάγνωσης. Υπενθυμίζω ότι ανάγνωση έχουμε και στις δύο τύπους αιτήσεων (ανάγνωσης ή εγγραφής). Η ανάγνωση γίνεται και στο τέλος του ίδιου κύκλου παίρνουμε από την κρυφή μνήμη τα αποτελέσματα. Έτσι μια εύστοχη αίτηση ανάγνωσης εξυπηρετείται σε ένα κύκλο. Αν είχαμε εύστοχη ανάγνωση σε μια αίτηση εγγραφής το σήμα hit συνδιάζεται με το σήμα που δηλώνει αίτηση εγγραφής ώστε να πάρουμε το σήμα που

δηλώνει ότι είμαστε έτοιμοι να γράψουμε στην κρυφή μνήμη στον επόμενο κύκλο. Η δεύτερη ζώνη είναι υπεύθυνη για να δώσει τα απαραίτητα σήματα για να γίνει αυτή η εγγραφή και να ολοκληρωθεί η εξυπηρέτηση της αίτησης (σενάρια 2 και 3). Αν όμως στον κύκλο αυτό χρειάζεται να γίνει μια ανάγνωση από τη μνήμη δεδομένων, η εγγραφή θα αναβληθεί για αργότερα (σενάρια 4-7). Η αναβολή αυτή γίνεται αντιληπτή αφού βρισκόμαστε ήδη στη δεύτερη χρονική ζώνη. Το σύστημα δεν γνωρίζει εκ των προτέρων αν υπάρχει αίτηση ανάγνωσης σε αναμονή, αλλά η αίτηση αυτή δίνεται από τον έξω κόσμο χωρίς προειδοποίηση. Αυτό έχει σαν αποτέλεσμα να μην μπορούμε να κρατήσουμε στο πρώτο write buffer τα δεδομένα για να τα χρησιμοποιήσουμε με την πρώτη ευκαιρία. Έτσι πρόσθεσα ένα δεύτερο write buffer που μας οδηγεί στη τρίτη ζώνη. Η ζώνη αυτή είναι υπεύθυνη για την ολοκλήρωση μιας εγγραφής που βρίσκεται σε αναμονή. Επειδή ο χρόνος αναμονής μπορεί να είναι ένας ή περισσότεροι κύκλοι οι δύο τελευταίοι καταχωρητές ελέγχονται από το control του συστήματος που μπορεί να παρακολουθήσει την κίνηση των αιτήσεων και να ρυθμίσει κατάλληλα το σήμα load_enable του buffer.
Ενώ η εγγραφή των δεδομένων σε μια εύστοχη αίτηση εγγραφής δεν είναι σίγουρο πότε θα συμβεί, η εγγραφή τους στην κύρια μνήμη (περίπτωση διεγγραφής) γίνεται πάντα στη δεύτερη χρονική ζώνη.
Τέλος αξίζει να σημιωθεί ότι τα σήματα από τους δύο buffers, εκτός από την κρυφή μνήμη, βγαίνουν και προς τα έξω παρακάπτοντάς την ώστε να αντιμετωπιστούν οι εξαρτήσεις read after write.

4.3.2 Εξυπηρέτηση αιτήσεων που καταλήγουν σε αστοχία.

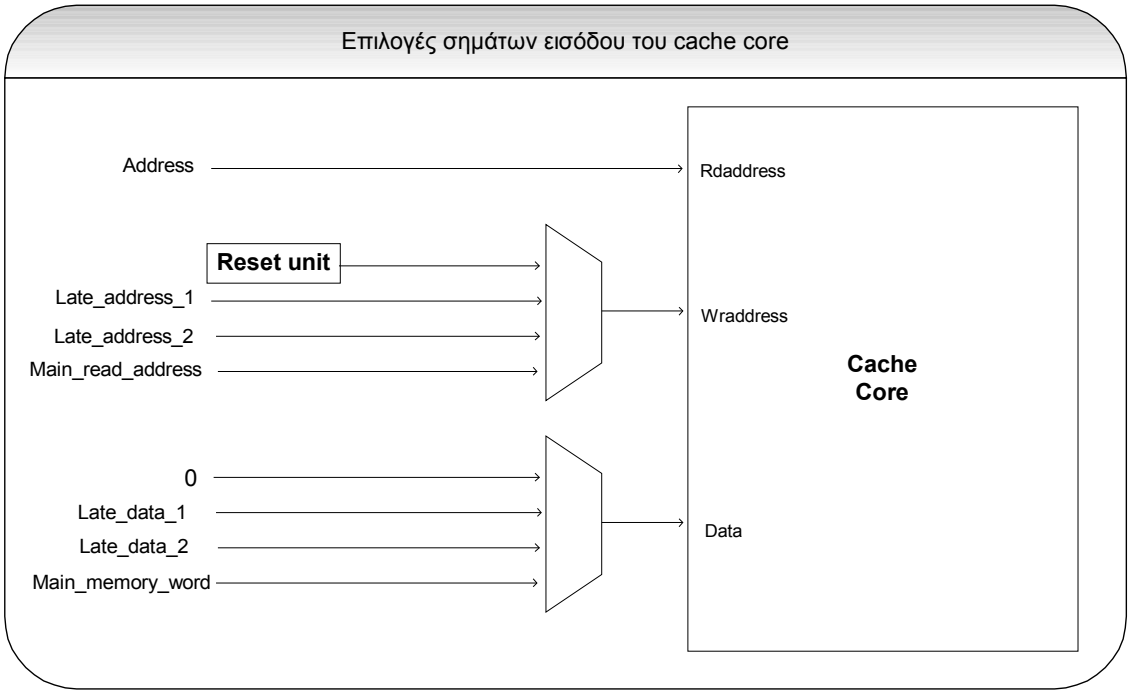
Τώρα θα δούμε με ποια συνδεσμολογία μπορεί ο πυρήνας να εξυπηρετήσει αιτήσεις ανάγνωσης ή εγγραφής με αναφορές σε διευθύνσεις που δεν υπάρχουν στην κρυφή μνήμη (περίπτωση miss). Όταν έχουμε αστοχία σε μια αίτηση ανάγνωσης πρέπει να γίνει αντικατάσταση μπλοκ. Το μπλοκ στο οποίο υπάρχει η επιθυμητή λέξη μεταφέρεται από την κύρια μνήμη στην κρυφή μνήμη. Σε μια άστοχη αίτηση εγγραφής το σύστηματα συμπεριφέρεται ανάλογα με τον τύπο του. Στο πρώτο σύστημα που δεν έχουμε κατανομή εγγραφής δεν συμβαίνει τίποτα πέρα από το να ενημερωθεί μόνο η κύρια μνήμη. Αντίθετα στα δύο άλλα συστήματα που έχουμε κατανομή εγγραφής αρχικά ενημερώνεται η κύρια μνήμη και ύστερα μεταφέρεται το αντίστοιχο μπλοκ στην κρυφή μνήμη.
Κατά την αντικατάσταση του μπλοκ διαβάζεται η κύρια μνήμη του συστήματος λέξη – λέξη, μέχρι να διαβαστεί όλο το μπλοκ. Οι διευθύνσεις ανάγνωσεις που χρειάζονται για να γίνει αυτή η διαδικασία παράγονται από μια ειδική μονάδα, που γεννά τις διευθύνσεις αυτές. Στη συνέχεια τα δεδομένα που απαντάει η κύρια μνήμη οδηγούνται στην κρυφή μνήμη, όπου γίνεται εγγραφή στη μνήμη δεδομένων με τις λέξεις του μπλοκ, αλλά και στη μνήμη ετικετών με τη νέα ετικέτα. Η όλη διαδικασία ελέγχεται από το control του συστήματος.
Παρακάτω δίνεται η διασύνδεση της κρυφής με την κύρια μνήμη.



Το σύστημα κύριας μνήμης όπως φαίνεται στο σχήμα αντιπροσωπεύει την ένωση της κύριας μνήμης με τη μονάδα καθυστέρισης, ώστε να πάρουμε μια μνήμη που αργεί να απαντήσει. Τα δεδομένα του συστήματος κύριας μνήμης πάνε και προς την έξοδο του συστήματος για την περίπτωση εξυπηρέτισης μιας αίτησης ανάγνωσης.
Αργότερα στην περιγαφή των καταστάσεων του συστήματος θα γίνει λεπτομερής ανάλυση της συμπεριφοράς του συστήματος.

4.3.3 Τελική επιλογή εισόδων του πυρήνα.

Αφού παρουσίασα όλες τις διαφορετικές διασυνδέσεις του πυρήνα της κρυφής μνήμης θεωρώ ότι είναι καλό να δείξω σε ένα σχήμα όλα τα σήματα που πάνε προς τον πυρήνα και επιλέγονται ανάλογα με τη κατάσταση του συστήματος. Η διεύθυνση ανάγνωσης είναι απλώς η διεύθυνση των αιτήσεων. Αντίθετα η διεύθυνση και τα δεδομένα εγγραφής μπορεί να μια από της τέσσερις επιλογές που φαίνονται στο παρακάτω σχήμα ανάλογα με το αν έχουμε:
Α) Αρχικοποίηση
Β) Εγγραφή στον δεύτερο κύκλο εξυπηρέτισης μιας αίτησης εγγραφής
Γ) Εγγραφή σε αναμονή εξυπηρέτισης μιας αίτησης εγγραφής
Δ) Αντικατάσταση ενός μπλοκ με ένα άλλο που προέχεται από την κύρια μνήμη

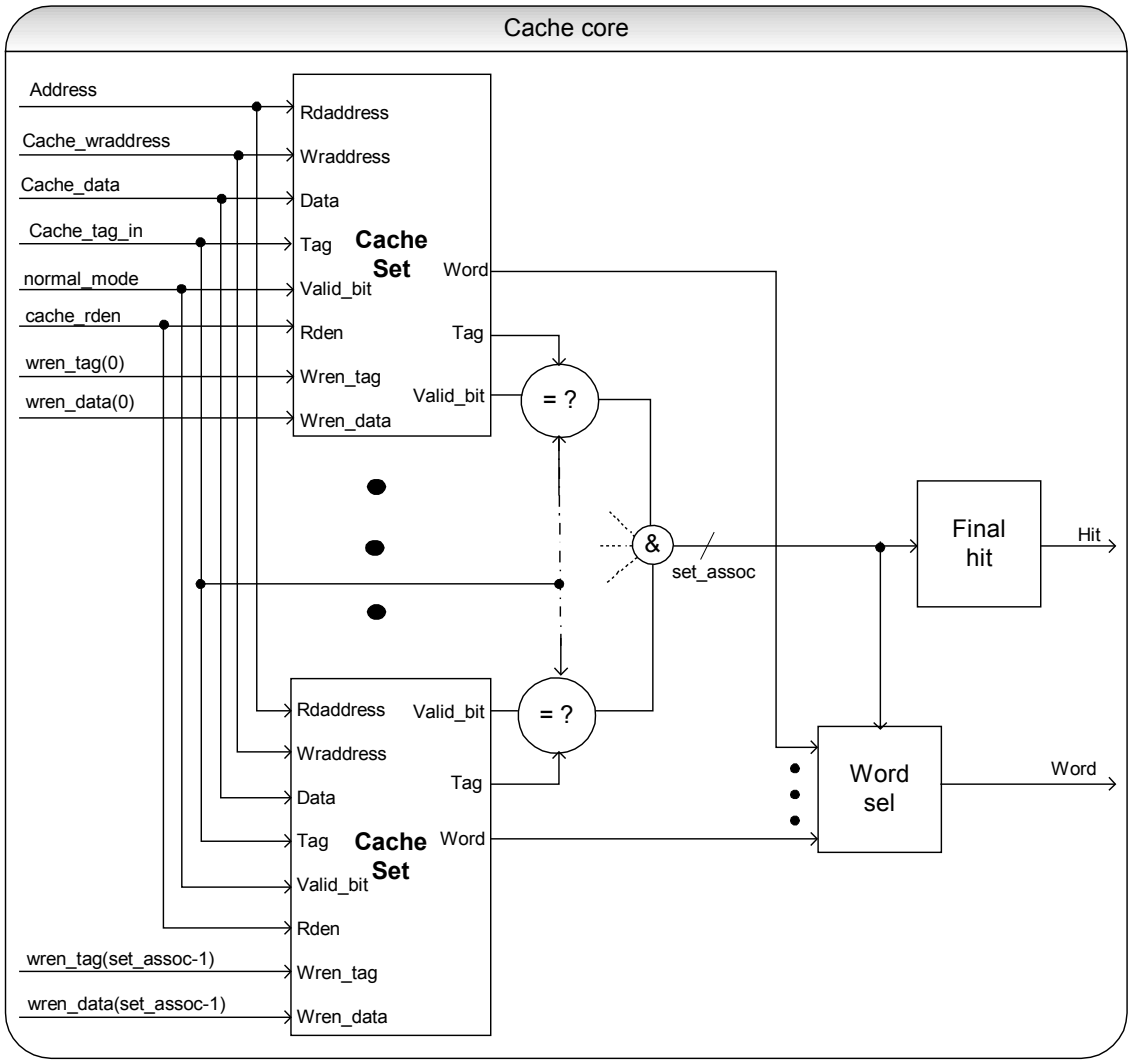


Οι διευθύνσεις αρχικοποίησης παράγονται από ειδική μονάδα.

4.4 Πυρήνας

Μπαίνουμε τώρα στο κομμάτι όπου θα αναπτυχθεί η σχεδίαση του συστήματος.
Αρχίζουμε από τον πυρήνα της σχεδίασης που δεν είναι άλλος από τα κομμάτια μνήμης, τον συγκριτή και τον πολυπλέκτη. Ο τελευταίος χρειάζεται στην περίπτωση της συνολοσυσχετιστικής κρυφής μνήμης όπου διαλέγουμε ποια λέξη από όλα τα σετ θα διαλέξουμε. Ο συγκριτής χρειάζεται για να συγκρίνουμε την τιμή του πεδίου ετικέτα της μνήμης με το αντίστοιχο της δοθείσας διεύθυνσης.
Ανάλογα με το αν έχουμε ένα ή περισσότερα σύνολα χρειαζόμαστε έναν ή περισσότερους συγκριτές που τοποθετούνται μετά τις μνήμες ετικετών και κάνουν τη σύγκριση της ετικέτας εισόδου με την ετικέτα που υπέδειξε ο δείκτης, ενώ παίρνουν επίσης σαν είσοδο το bit εγκυρότητας. Για περισσότερα από δύο σύνολα χρειάζεται επίσης και ένας πολυπλέκτης που όπως είπαμε διαλέγει μια λέξη ανάλογα με τα αποτελέσματα των συγκριτών.
Στο κώδικα της σχέδιασής μου έχω δημιουργήσει μια μονάδα που αποτελεί ένα σετ της κρυφής μνήμης. Ανάλογα με την παράμετρο τις συνολοσυσχετιστικότητας ένα ή περισσότερα σετ ενώνονται κατάλληλα και δίνουν τελικά τον πυρήνα της κρυφής μνήμης.

Ακολουθεί το σχήμα της σύνθεσης του πυρήνα.



Η ετικέτα από κάθε tag memory πάει σε ένα συγκριτή, ο οποίος παίρνοτας επιπλέον σαν είσοδο το μέρος «ετικέτα» της διεύθυνσης της αίτησης και το bit εγκυρότητας, δίνει ένα σήμα έξόδου. Αυτό το σήμα είναι το σήμα hit που βγαίνει από αυτό το σετ. Αν έχουμε πολλά σετ μόνο ένα μπορεί να δώσει hit με τιμή 1. Όλα τα επιμέρους hits ενόνωνται σε ένα σήμα το οποίο περιέχει την πληροφορία σε πιο σετ έγινε η ευστοχία, αν έγινε. Το σήμα αυτό με πλάτος όσο και η παράμετρος της συνολοσυσχετικότητας χρησιμεύει για να διαλέξουμε πιοα δεδομένα θα βγουν προς τα έξω. Παράλληλα από το σήμα αυτό παράγεται ένα σήμα ενός bit που είναι το τελικό hit του συστήματος.
Ας δούμε τώρα πως υλοποιούνται σε κώδικα οι δύο τελευταίες μονάδες final hit και word sel.

```
ENTITY final_hit IS
	GENERIC
	(
		n	:INTEGER :=4
	);
	PORT
	(
		Hit_bits	:IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
		Hit		:OUT STD_LOGIC
	);
```

```
END final_hit;

ARCHITECTURE behavioral OF final_hit IS
	SIGNAL zeros  :STD_LOGIC_VECTOR(n-1 DOWNTO 0);
BEGIN
	zeros<=(OTHERS=>'0');
	PROCESS(Hit_bits)
	BEGIN
		IF (Hit_bits>zeros) THEN
			Hit<='1';
		ELSE
			Hit<='0';
		END IF;
	END PROCESS;
END behavioral;


ENTITY word_sel IS
	GENERIC
	(
		n	:INTEGER :=2
	);
	PORT
	(
		Words  :IN word_array(n-1 DOWNTO 0);
		Sel    :IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
		Word   :OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
	);
END word_sel;

ARCHITECTURE behavioral OF word_sel IS
BEGIN
	PROCESS(Words,Sel)
	VARIABLE tempword    :STD_LOGIC_VECTOR(31 DOWNTO 0);
	BEGIN
		tempword:=(OTHERS=>'0');
		For_loop:FOR i IN 0 TO n-1 LOOP
			IF (Sel(i)='1') THEN
				tempword:=Words(i);
			END IF;
		END LOOP;
		Word<=tempword;
	END PROCESS;
END behavioral;
```

Το word_array είναι ένας τύπος που δημιούργησα για αναπαραστήσω ποιο εύκολα τα καλώδια που μεταφέρουν λέξεις (32 bits). Η δήλωσή του είναι η ακόλουθη.

```
TYPE word_array IS ARRAY (NATURAL RANGE <>) OF STD_LOGIC_VECTOR(31
DOWNTO 0);
```

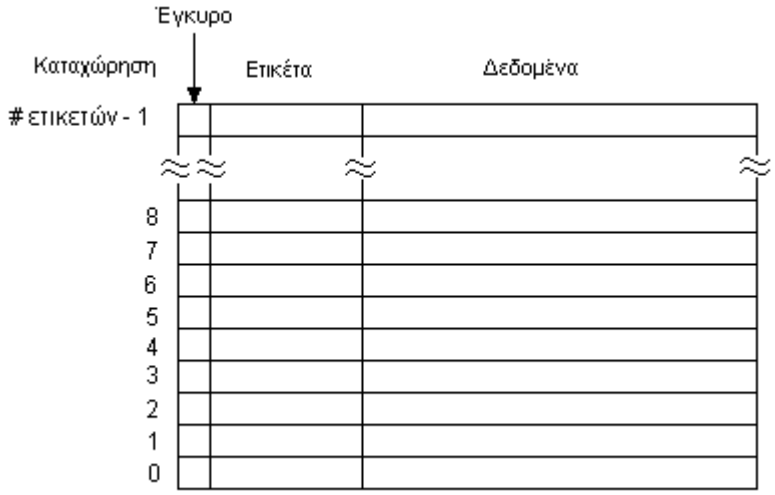
Δύο ήταν οι λόγοι που με οδήγησαν να μην χρησιμοποιήσω ένα κλασσικό πολυπλέκτη αλλά τη μονάδα word_sel. Ο πρώτος είναι ότι λόγω της παραμετρικής φύσης του προβλήματος δεν ξέρουμε πόσες εισόδους θα έχει ο πολυπλέκτης. Μπορεί με μεταβλητές του τύπου Generic να ελέγχεις το μέγεθος ενός σήματος (πόσα bits φαρδύ θα είναι) αλλά η γλώσσα δεν σου δίνει τη δυνατότητα να ελέγχεις πόσες εισόδους θες παραμετρικά. Μια λύση θα ήταν να φτιάξω πολλούς πολυπλέκτες με 2, 4, 8 εισόδους ο καθένας αντίστοιχα και να χρησιμοποιούσα με if statement το κατάλληλο κάθε φορά. Ο δεύτερος λόγος είναι ότι ο πολυπλέκτης παίρνει ως σήμα ελέγχου ένα σήμα που έχει

μέγεθος το $\log_2$ των εισόδων. Όμως από τις εξόδους των συγκριτών έχουμε ένα σήμα μεγέθους όσο οι είσοδοι και όχι το λογάριθμό τους. Θα έπρεπε λοιπόν να μετατρέψουμε το σήμα αυτό στο επιθυμητό κάτι που θα εισήγαγε παραπάνω λογική στο σύστημα, ίσως αχρείαστη. Με τη μονάδα word_sel αντιμετωπίζονται και τα δύο προβλήματα.

<u>Διαχωρισμός μνήμης ετικέτας – δεδομένων</u>

Τώρα θα δείξω με ποιο σκεπτικό διαχώρισα τις μνήμη ετικετών και δεδομένων και ποια μορφή έχει η μονάδα που αποτελεί ένα σετ του πυρήνα.

Ας πάρουμε πρώτα την απλή περίπτωση της άμεσα αντιστοιχισμένης κρυφής μνήμης. Θα χρειαστούμε μια μνήμη που κάθε της καταχώρηση θα έχει τα τρία πεδία που φαίνονται στο παρακάτω σχήμα.

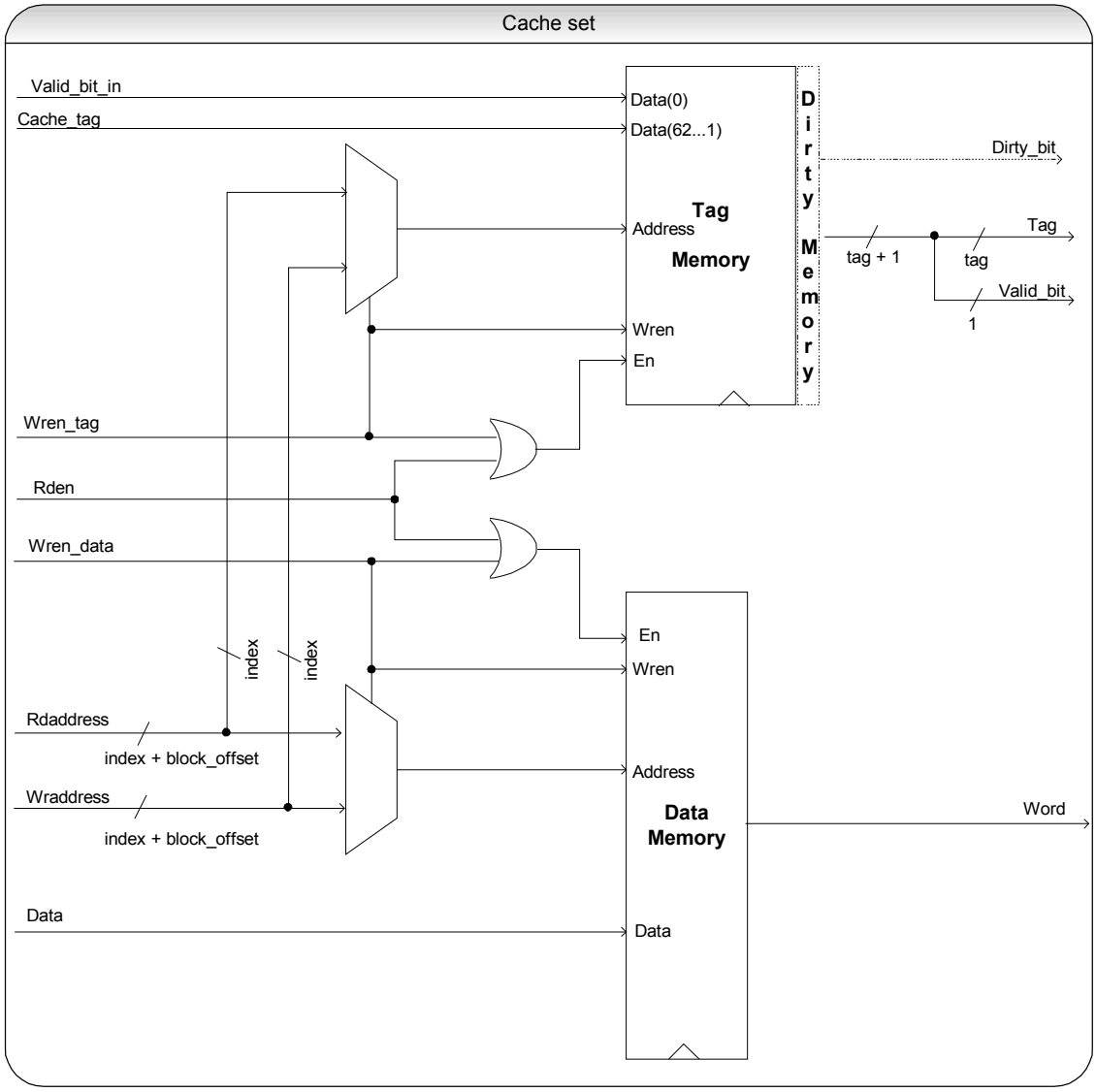


Το πεδίο έγκυρο είναι ένα μόνο bit. Όταν έχει τη τιμή 1 σημαίνει πως η συγκεκριμένη καταχώρηση είναι έγκυρη αλλιώς όχι.

Παρατηρώντας την παραπάνω δομή μπορούμε να καταλάβουμε πως έχουμε πρόσβαση σε μία ετικέτα μαζί με τα δεδομένα της. Σε μία αίτηση εγγραφής διαβάζουμε πρώτα την ετικέτα και εφόσον είναι αυτή που ψάχνουμε γράφουμε τα δεδομένα στον επόμενο κύκλο. Το γράψιμο χρειάζεται λοιπόν 2 κύκλους. Επειδή όμως όπως αναφέραμε και στις προδιαγραφές θέλουμε να πετύχουμε απόδοση συστήματος της τάξεως μια εγγραφή ανά κύκλο, κάνουμε το εξής:

Διαχωρίζουμε τις ετικέτες από τα δεδομένα και χρησιμοποιούμε ξεχωριστή μνήμη για το καθένα. Έτσι μπορούμε ταυτόχρονα (στον ίδιο κύκλο) να διαβάζουμε μια ετικέτα για να δούμε αν είναι η σωστή ενώ γράφουμε τα δεδομένα της προηγούμενης αίτησης.

Ακολουθεί το σχήμα της σύνθεσης ενός σετ.



Η διεπαφή του cache set είναι αυτή που μας επιτρέπει να εκτελέσουμε όλες τις απαραίτητες ενέργειες ώστε να γράψουμε και να διαβάσουμε από την κρυφή μνήμη. Ας εξηγήσουμε την χρησιμότητα κάθε εισόδου και εξόδου.

Είσοδοι

Rdaddress: Η διεύθυνση το δεδομένο της οποίας θέλουμε να διαβάσουμε.

Wraddress: Η διεύθυνση το δεδομένο της οποίας θέλουμε να γράψουμε.

Data: Η τιμή των δεδομένων που θέλουμε να γραφτούν στην διεύθυνση εγγραφής, στη μνήμη δεδομένων.

Tag_in: Η τιμή της ετικέτας που θέλουμε να γραφτεί, κατά τη διαδικασία μιας αντικατάστασης, στην διεύθυνση εγγραφής της μνήμης ετικετών.

Valid_bit_in: Η τιμή του bit εγκυρότητας που θέλουμε να γράψουμε μαζί με την αντίστοιχη ετικέτα.

Rden: Σήμα ανάγνωσης της μνήμης ετικετών και της μνήμης δεδομένων.

Wren_tag: Σήμα εγγραφής στη μνήμη ετικετών.

Wren_data: Σήμα εγγραφής στη μνήμη δεδομένων.

Clock: Ρολόι συστήματος που συγχρονίζει τις μνήμες.

Έξοδοι

Valid_bit: Η τιμή του bit εγκυρότητας που εξέρχεται από τη μνήμη ετικετών όταν θέλουμε να δούμε αν υπάρχει στη κρυφή μνήμη η εγγραφή που θέλουμε να διαβάσουμε ή να γράψουμε. Είναι το bit που στέλνεται μαζί με την αντίστοιχη ετικέτα στον συγκριτή.

Tag: Η τιμή της ετικέτας που εξέρχεται από τη μνήμη ετικετών όταν θέλουμε να δούμε αν υπάρχει στη κρυφή μνήμη η εγγραφή που θέλουμε να διαβάσουμε ή να γράψουμε. Είναι τα δεδομένα που στέλνεται μαζί με το αντίστοιχο bit εγκυρότητας στον συγκριτή.

Word: Η λέξη που διαβάζεται από τη μνήμη δεδομένων σύμφωνα με τη διεύθυνση ανάγνωσης.

Ο κώδικας που αντιστοιχεί στη δήλωση του cache set ως οντότητα είναι ο ακόλουθος.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY cache_set IS
	GENERIC
	(
		t		:INTEGER :=52;
		tag_addr	:INTEGER :=8;
		data_addr	:INTEGER :=10
	);
	PORT
	(
		Rdaddress		:IN STD_LOGIC_VECTOR(data_addr-1
DOWNTO 0);
		Wraddress		:IN STD_LOGIC_VECTOR(data_addr-1
DOWNTO 0);
		Data			:IN STD_LOGIC_VECTOR(31 DOWNTO 0);
		Tag_in		:IN STD_LOGIC_VECTOR(t-1 DOWNTO 0);
		Valid_bit_in	:IN STD_LOGIC;
		Rden			:IN STD_LOGIC;
		Wren_tag		:IN STD_LOGIC;
		Wren_data		:IN STD_LOGIC;
		Clock			:IN STD_LOGIC;
		Valid_bit		:OUT STD_LOGIC;
		Tag			:OUT STD_LOGIC_VECTOR(t-1 DOWNTO
0);
		Word			:OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
	);
END cache_set;
```

Οι τρεις μεταβλητές που χρησιμοποιούνται, και ίσως να μπερδεύουν λίγο τον αναγνώστη, χρησιμεύουν για να υποστηριχθεί η παραμετρική φύση του προβλήματος. Γι' αυτό και τα όρια πολλών σημάτων καθορίζονται από τις μεταβλητές αυτές. Για παράδειγμα η προτελευταία γραμμή του κώδικα χρησιμοποιεί τη μεταβλητή t για θέσει τα όρια στο tag_out. Το bit 0 παραλείπεται από την ετικέτα εξόδου αφού είναι το bit εγκυρότητας (βγαίνει ως valid_bit στην προηγούμενη γραμμή). Επιπλέον αξιοποιώντας το t που μας δείχνει πόσα bits είναι η ετικέτα σε μια δεδομένη υλοποίηση, παίρνουμε από το tag_out όσα bits χρειαζόμαστε. Με αυτόν τον τρόπο θα στείλουμε στον συγκριτή το σωστό μήκος ετικέτας που χρειάζεται και όχι όλη την εγγραφή της μνήμης ετικέτας που είναι μεγαλύτερη (θα δούμε στη συνέχεια το μήκος της).

Η μνήμη ετικετών και η μνήμη δεδομένων είναι μονόπορτες. Αυτό οδηγεί στη χρησιμοποίηση ενός πολυπλέκτη, όπως φαίνεται και στο σχήμα, που διαλέγει τη διεύθυνση που θα δοθεί σε αυτές ανάλογα με το αν θέλουμε να διαβάσουμε ή να γράψουμε. Η διεύθυνση ανάγνωσης (Rdaddress) του σχήματος είναι στην πράξη η διεύθυνση οποιαδήποτε αίτησης (ανάγνωσης ή εγγραφής) αφού αρχικά πρώτα γίνεται ανάγνωση για την διερεύνηση ευστοχίας. Η διεύθυνση εγγραφής (Wraddress) του σχήματος είναι η διεύθυνση της λέξης που θέλουμε να γραφτεί στη κύρια μνήμη. Αυτό συμβαίνει σε δύο περιπτώσεις. Πρώτον κατά την εγγραφή της λέξης σε μια επιτυχημένη εγγραφή (δεύτερο στάδιο εξυπηρέτης αίτησης), όπου η εγγραφή γίνεται μόνο στη μνήμη δεδομένων. Δεύτερον κατά την μεταφορά ενός ολόκληρου μπλοκ από την κύρια μνήμη προς την κρυφή μνήμη, όπου η εγγραφή γίνεται και στις δύο μνήμες για κάθε λέξη του μπλοκ. Όπως είναι εμφανές από την πολυπλοκότητα των εγγραφών, τα σήματα Wren_tag και Wren_data προέρχονται από ειδική μονάδα ελέγχου. Η μνήμη dirty χρησιμοποιείται μόνο στο τρίτο σύστημα όπου έχουμε ετεροχρονισμένη εγγραφή (write back), γι' αυτό και είναι γραμμοσκιασμένη με διαφορετικό τρόπο.

4.5 Αρχικοποίηση

Αφού σχεδιάσαμε τον πυρήνα της μνήμης, και πριν σκεφτούμε τι λογική χρειάζεται ώστε να λειτουργήσει σωστά, θα δείξουμε πως υποστηρίζεται η αρχικοποίηση.

Τι σημαίνει αρχικοποίηση; Αρχικοποίηση σημαίνει η λειτουργία κατά την οποία το σύστημα θέτεται σε μια τέτοια κατάσταση η οποία μας είναι γνωστή και κατάλληλη για να προχωρήσει το σύστημα στην κανονική του λειτουργία. Πρακτικά στην περίπτωση μας είναι να μηδενιστούν οι καταχωρητές και οι μνήμες που χρησιμοποιούνται. Οι καταχωρητές μπορούν να μηδενιστούν σε ένα κύκλο, αλλά κάθε μνήμη χρειάζεται για να μηδενιστεί τόσους κύκλους όσο είναι και το ύψος της. Αυτό γίνεται διότι δεν μπορούμε να γράψουμε παραπάνω από ένα κελί στον ίδιο κύκλο. Μιλάμε πάντα για μια απλή σύγχρονη μνήμη. Πρέπει λοιπόν να γράφουμε την τιμή μηδέν σε κάθε κελί της μνήμης και αυτό γίνεται κάνοντας συνέχεια εγγραφή με σταθερή είσοδο δεδομένων(μηδέν) και αλλάζοντας τη διεύθυνση εγγραφής ανά κύκλο. Ένας μετρητής θα μπορούσε να μας δώσει την επιθυμητή διεύθυνση. Ο μετρητής αυτός θα ξεκινάει από την τιμή μηδέν και κάθε κύκλο θα αυξάνεται κατά ένα. Η τελευταία του τιμή πριν μηδενιστεί θα είναι ίση με τη διεύθυνση του τελευταίου κελιού της μνήμης. Όταν μηδενιστεί πάλι θα σημαίνει ότι η αρχικοποίηση ολοκληρώθηκε και το σύστημα θα ειδοποιείται κατάλληλα.

Το μόνο που χρειάζεται να γνωρίζουμε λοιπόν είναι το ύψος της μνήμης που θέλουμε να αρχικοποιήσουμε. Αυτό όμως δεν είναι γνωστό κατά τη σχεδίαση λόγω της παραμετρικής φύσης του προβλήματος. Ένα ακόμα στοιχείο που έχει ενδιαφέρον είναι ότι δεν έχουμε μία μόνο μνήμη αλλά δύο. Μία με τις ετικέτες και μία με τα δεδομένα. Οι δύο αυτές μνήμες έχουν διαφορετικό ύψος. Γενικά ισχύει ότι η μνήμη των δεδομένων έχει μεγαλύτερο ή ίσο ύψος με τη μνήμη ετικετών. Αυτό οφείλεται στο ότι στη μνήμη δεδομένων κάθε κελί αποτελεί μία λέξη και όχι όλο το μπλοκ στο οποίο αναφέρεται μια ετικέτα. Ο λόγος που οργάνωσα έτσι τη μνήμη δεδομένων (κελί= λέξη και όχι μπλοκ) είναι για να αποφύγω ένα πολυπλέκτη μετά την έξοδο της μνήμης που θα επιλέγει τη κατάλληλη λέξη. Η δομή αυτή προτείνεται και στη βιβλιογραφία. Βέβαια είναι φυσικό ως αποτέλεσμα αυτής της απόφασης η μνήμη των δεδομένων να χρειάζεται κάτι παραπάνω από τα bits του δείκτη για να διευθυνσιοδοτηθεί. Αυτά είναι τα bits της απόκλισης μπλοκ.

Ας επιστρέψουμε όμως στην αρχικοποίηση. Τα ύψη των μνημών θα δίνονται παραμετρικά και επομένως θα είναι γνωστές οι μέγιστες τιμές των διευθύνσεων. Εφόσον το ύψος της μνήμης δεδομένων είναι μεγαλύτερο από αυτό της μνήμης

ετικετών μπορούμε να επικεντρωθούμε σε αυτήν αφού αυτή είναι που μας καθορίζει το μέγεθος της καθυστέρησης. Ενώ παράγουμε τις διευθύνσεις για τη μνήμη δεδομένων δίνουμε παράλληλα και προς την μνήμη εντολών την ίδια διεύθυνση αφαιρώντας ασφαλώς τα bits που αναφέρονται στην απόκλιση μπλοκ ώστε να έχει τη σωστή τιμή. Στο τέλος της διαδικασίας θα έχουμε και τις δύο μνήμες μηδενισμένες. Το μόνο μειονέκτημα είναι ότι κάθε κελί της μνήμης ετικετών θα γράφεται πάνω από μια φορά (περίπτωση όπου τα bits απόκλισης μπλοκ είναι > 0) με την τιμή μηδέν. Κάτι τέτοιο είναι περιττό. Όμως χρονικά δεν μας στοιχίζει τίποτα γιατί όπως είπαμε ο χρόνος καθορίζεται από τη μνήμη δεδομένων για την οποία η αρχικοποίηση γίνεται όσο πιο γρήγορα επιτρέπεται από τους περιορισμούς του συστήματος.
Ας υποθέσουμε ότι έχουμε αρχικά ένα σύστημα όπου οι μνήμες του έχουν τυχαίες τιμές. Η αρχικοποίηση είναι απαραίτητη ώστε να πάρουμε τα αποτελέσματα που περιμένουμε βάσει των αιτήσεων που κάνουμε προς τη μνήμη. Ωστόσο βάσει της θεωρίας αλλά και της σχεδίασης μας μπορούμε να παρατηρήσουμε ότι η κρυφή μνήμη κατά κάποιο τρόπο αρχικοποιείται αν πάρουν τη τιμή μηδέν όλα τα bits εγκυρότητας, αυτά δηλαδή που δείχνουν να μια καταχώρηση στη κρυφή μνήμη είναι έγκυρη. Τα bits αυτά βρίσκονται στη μνήμη ετικετών όπως έχει δειχτεί παραπάνω. Για να μηδενιστεί ένα τέτοιο bit είναι απαραίτητη η προσπέλαση και της ετικέτας. Επομένως αφού γίνεται η προσπέλαση και «χάνεται» ο κύκλος θα ήταν ανόητο να μην μηδενίσουμε και την ετικέτα. Προκύπτει όμως το εξής ερώτημα: Γιατί να μηδενίσουμε και τη μνήμη δεδομένων αφού ό,τι και να έχει μέσα δεν πρόκειται να χρησιμοποιηθεί ποτέ αφού δεν θα είναι έγκυρο. Το ερώτημα αυτό αποκτά ιδιαίτερη βαρύτητα όταν δούμε ότι για την δουλειά αυτή χάνουμε χρόνου. Πραγματικά αν μας ενδιαφέρει να μηδενίσουμε μόνο τη μνήμη ετικετών ο χρόνος αρχικοποίησης μειώνεται λογαριθμικά ανάλογα με το πόσες λέξεις έχει ένα μπλοκ στο εκάστοτε σύστημά μας. Η μόνη απάντηση που μπορώ να δώσω είναι ότι μπορεί να ισχύει αυτό λογικά όμως από την άλλη δεν θα πάρουμε ένα σύστημα με πλήρως γνωστή κατάσταση αφού δεν θα ξέρουμε τις τιμές της μνήμης δεδομένων κάτι που δεν είναι σωστό από άποψη ελέγχου του συστήματος. Με άλλα λόγια δεν θα υπακούγαμε στον «ορισμό» της αρχικοποίησης.
Βλέπουμε λοιπόν ότι έχουμε να διαλέξουμε ανάμεσα σε δύο αρχικοποιήσεις:
μια ολοκληρωτική αλλά αργή και μια μερική αλλά πιο γρήγορη.
Εγώ επέλεξα την πρώτη αλλά δείχνω πόσο εύκολα μπορούμε να πάμε στη δεύτερη.
Ας δούμε λοιπόν σε κώδικα τη μονάδα που είναι υπεύθυνη για την αρχικοποίηση της κρυφής μνήμης.

```
ENTITY reset_unit IS
	GENERIC
	(
--		tag_addr  :INTEGER :=8;			--
tag_addr=index=log(cache_size/(block_size*set_associativity))+7
		data_addr :INTEGER :=4			--
data_addr=index+block_offset=tag_addr+log(block_size)
	);
	PORT
	(
		Enable		:IN STD_LOGIC;
		Reset			:IN STD_LOGIC;
		Clock			:IN STD_LOGIC;
		Finish		:OUT STD_LOGIC;
		--Tag_address :OUT STD_LOGIC_VECTOR(tag_addr-1 DOWNTO
0);
		Data_address  :OUT STD_LOGIC_VECTOR(data_addr-1 DOWNTO
0)
```

```
	);
END reset_unit;

ARCHITECTURE structure OF reset_unit IS
	COMPONENT counter
		GENERIC
		(
			n	:INTEGER :=4
		);
		PORT
		(
			Resetn :IN STD_LOGIC;
			E		:IN STD_LOGIC;
			Clock		:IN STD_LOGIC;
			Q		:OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)
		);
	END COMPONENT;

	COMPONENT ANDn
		GENERIC
		(
			n	:INTEGER :=2
		);
		PORT
		(
			X	:IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
			F	:OUT STD_LOGIC
		);
	END COMPONENT ;

	SIGNAL temp_data_addr:STD_LOGIC_VECTOR(data_addr-1 DOWNTO 0) ;
BEGIN
	data_count_1:counter
		GENERIC MAP
		(
			n=>data_addr    -- n=>tag_addr
		)
		PORT MAP
		(
			Resetn=>Reset,
			E=>Enable,
			Clock=>Clock,
			Q=>temp_data_addr
		);

	andn_1:andn
		GENERIC MAP
		(
			n=>data_addr  -- n=>tag_addr
		)
		PORT MAP
		(
			X=>temp_data_addr,
			F=>Finish
		) ;

	Data_address<=temp_data_addr;
END structure;
```

Όπως βλέπουμε η μονάδα αποτελείται από ένα παραμετρικό μετρητή στον οποίο μπορούμε να καθορίσουμε το μέγεθος της εξόδου του και από μία παραμετρική πύλη

«και». Η τελευταία που είναι παραμετρική ως προς τον αριθμό των εισόδων της πάνω στις οποίες γίνεται η λογική πράξη «και» χρησιμεύει για να δημιουργήσει το σήμα finish το οποίο ειδοποιεί τον έξω κόσμο ότι η αρχικοποίηση ολοκληρώθηκε. Το πλάτος της διεύθυνσης της μνήμης δεδομένων δίνεται παραμετρικά μέσω της μεταβλητής data_address. Αν αντικαταστήσουμε τη μεταβλητή αυτή με εκείνη που λέγεται tag_address θα πάρουμε το σύστημα αρχικοποίησης που μηδενίζει μόνο τις ετικέτες. Ό,τι έχω μέσα σε σχόλιο είναι για να δείξω τις ελάχιστες αλλαγές που χρειάζονται. Παρακάτω δείχνω και πως υλοποιούνται οι δύο μονάδες.

```
ENTITY counter IS
	GENERIC
	(
		n	:INTEGER :=4
	);
	PORT
	(
		Resetn :IN STD_LOGIC;
		E		:IN STD_LOGIC;
		Clock		:IN STD_LOGIC;
		Q		:OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)
	);
END counter;

ARCHITECTURE behavioral OF counter IS
	SIGNAL count  :STD_LOGIC_VECTOR(n-1 DOWNTO 0);
BEGIN
	PROCESS(Resetn,E,Clock)
	BEGIN
		IF (Resetn='1') THEN
			count<=(OTHERS=>'0');
		ELSIF (Clock'EVENT AND Clock = '0') THEN
			IF (E='1') THEN
				count<=count+1;
			END IF;
		END IF;
	END PROCESS;
	Q<=count;
END behavioral;


ENTITY andn IS
	GENERIC
	(
		n	:INTEGER :=2
	);
	PORT
	(
		X	:IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
		F	:OUT STD_LOGIC
	);
END andn;

ARCHITECTURE behavioral OF andn IS
	SIGNAL ones   :STD_LOGIC_VECTOR(n-1 DOWNTO 0);
BEGIN
	ones<=(OTHERS => '1');
	F<=	'1' WHEN (X=ones) ELSE
		'0';
END behavioral;
```

4.6 Υποστήριξη διαφορετικών μεγεθών μνήμης

Μπορεί να είναι εύκολο να καθοριστεί η διεπαφή των μνημών ετικετών και δεδομένων δεν είναι όμως καθόλου εύκολο να καθοριστεί το μέγεθός τους. Σε ένα παραμετρικό πρόβλημα όπως αυτό ανακύπτουν στον σχεδιαστή οι ακόλουθες ερωτήσεις.
Α)Ποιο πρέπει να είναι το πλάτος της μνήμης ετικετών;
Β)Ποιο πρέπει να είναι το ύψος της μνήμης ετικετών;
Γ)Ποιο πρέπει να είναι το πλάτος της μνήμης δεδομένων;
Δ)Ποιο πρέπει να είναι το ύψος της μνήμης δεδομένων;

Μόνο σε ένα μπορεί να δοθεί άμεση απάντηση. Αφού η μνήμη δεδομένων θα είναι οργανωμένη σε λέξεις (μία λέξη ανά κελί), το πλάτος της θα είναι ίσο με μια λέξη. Στο πρόβλημα από τις προδιαγραφές ξέρουμε ότι η λέξη του συστήματος είναι 32 bits. Άρα:
Γ)Το πλάτος της μνήμης δεδομένων πρέπει είναι 32 bits.
Στις υπόλοιπες ερωτήσεις η άμεση απάντηση που μπορούμε να δώσουμε είναι ότι δεν γνωρίζουμε εκ των προτέρων. Τι κάνουμε λοιπόν; Τρεις στρατηγικές μπορώ να σκεφτώ.
Μια στρατηγική είναι η εξής:
Θα φτιάξω δυο μνήμες με τέτοια μεγέθη που να υποστηρίζουν τη χειρότερη περίπτωση που μπορεί να συμβεί. Ως χειρότερη περίπτωση εννοώ ένα συνδυασμό παραμέτρων από των χρήστη που να χρειάζονται πολύ μεγάλες μνήμες και ως προς το ύψος και ως προς το πλάτος (εξαιρώντας τη μνήμη δεδομένων για το πλάτος). Άρα με άλλα λόγια φτιάχνω δύο μεγάλες μνήμες και χρησιμοποιώ κάποιο μέρος τους ανάλογα με τις παραμέτρους.
Μια δεύτερη στρατηγική είναι:
Φτιάχνω αρκετές μνήμες που να ικανοποιούν όλους τους δυνατούς συνδυασμούς παραμέτρων. Όταν οι παράμετροι γίνουν γνώστες, το σύστημα με κατάλληλο κώδικα θα διαλέξει δύο από αυτές που θα έχουν το κατάλληλο μέγεθος.
Τρίτη στρατηγική:
Οι μνήμες δεν θα προϋπάρχουν αλλά θα κατασκευάζονται δυναμικά μέσω κώδικα οπότε και θα παίρνουν τις κατάλληλες διαστάσεις.
Αξιολογώντας τις τρεις στρατηγικές παρατηρούμε ότι πιο εύκολα υλοποιείται η πρώτη και πιο δύσκολα η τρίτη. Από τη σκοπιά όμως της ευελιξίας και της οικονομίας πόρων καλύτερη είναι η τρίτη στρατηγική ενώ η πρώτη είναι η πιο σπάταλη. Η δεύτερη στρατηγική συνδυάζει ένα μέρος της ευκολία και ένα μέρος της ευελιξίας που έχουν οι δύο άλλες στρατηγικές.
Επειδή δεν είχα την εμπειρία της κατασκευής δικής μου μνήμης και αφού μου προτάθηκε η χρησιμοποίηση των εργαλείων που παράγουν μνήμες απέρριψα την τρίτη στρατηγική. Ο σχεδιασμός που επέλεξα συνδυάζει κατά κάποιο τρόπο τις δύο πρώτες στρατηγικές:
Α) Το πλάτος της μνήμης ετικετών θα είναι το μέγιστο δυνατό πλάτος που θα επέτρεπε το πρόβλημα. Δηλαδή θα ήταν ο μέγιστος αριθμός bits που θα μπορούσε να έχει μια ετικέτα συν ένα bit ακόμα, αυτό της εγκυρότητας.
Β, Δ) Θα κατασκευαστούν μνήμες διαφόρων υψών, ώστε να ικανοποιούνται αρκετοί συνδυασμοί παραμέτρων. Όταν οι παράμετροι γίνουν γνώστες, το σύστημα με κατάλληλο κώδικα θα διαλέξει δύο από αυτές που θα έχουν το κατάλληλο μέγεθος.
Παρακάτω θα δείξω πως βρήκα το πλάτος της μνήμης ετικετών και ποιες μνήμες έφτιαξα τελικά με το εργαλείο που είχα.

4.6.1 Δημιουργία μνημών με χρήση εργαλείου.

Για την δημιουργία των μνημών της εργασίας χρησιμοποιήθηκε το εργαλείο:
Xilinx CORE Generator.
Στο εργαλείο αυτό δημιουργώντας πρώτα ένα project ο χρήστης μπορεί να επιλέξει, από την καρτέλα με τα functions, τον φάκελο Memories & Storage Elements και ύστερα τον υποφάκελο RAMs & ROMs. Εκεί εμφανίζονται τέσσερις διαφορετικές επιλογές για το τι είδους μνήμη θα φτιάξει το εργαλείο. Εγώ χρησιμοποίησα την επιλογή Single Port Block Memory 6.2 για όλες τις μνήμες του συστήματος.

Μνήμες ετικετών

Υπολογίζω αρχικά το πλάτος κάθε τέτοιας μνήμης. Όπως έχω αναφέρει η επιλογή μου είναι ότι το πλάτος της μνήμης ετικετών θα είναι το μέγιστο δυνατό. Τώρα θα ανατρέξω σε έναν υπολογισμό που είχα κάνει κατά την προσχεδιαστική μελέτη. Εκεί είχα δείξει τα εξής:
Αν η ετικέτα είναι t bits τότε έχουμε τη σχέση:

$$t = n - p \Rightarrow t = (x - \log_2 y) - \left(\log_2 \left(\frac{z}{y \times k} \right) \right) \Rightarrow t = x - \left(\log_2 y + \log_2 \left(\frac{z}{y \times k} \right) \right)$$

$$\Rightarrow t = x - \log_2 \left(y \times \frac{z}{y \times k} \right) \Rightarrow t = x - \log_2 \left(\frac{z}{k} \right)$$

όπου x:Μέγεθος διεύθυνσης, z:Μέγεθος κρυφής μνήμης και k:Βαθμός συνολοσυσχετιστικότητας
Το μέγεθος της ετικέτας εξαρτάται από το μέγεθος διεύθυνσης, το μέγεθος της κρυφής μνήμης και το βαθμό συνολοσυσχετιστικότητας. Δεν εξαρτάται από το μέγεθος του μπλοκ.
Ο τελευταίος τύπος αποκτά έναν γνωστό όρο αφού από τα δεδομένα του προβλήματος ξέρουμε ότι το μέγεθος της διεύθυνσης είναι 64 bits. Το 64 αναφέρεται σε διεύθυνση byte αλλά αφού οι αιτήσεις ανάγνωσης και εγγραφής αναφέρονται σε μία ολόκληρη λέξη και η λέξη είναι 32 bit (4 bytes), η τελική διεύθυνση λέξης έχει μέγεθος 62. Άρα στη σχέση μας θα βάλουμε όπου x την τιμή 62. Εφόσον ψάχνουμε το μέγιστο t αρκεί να βρούμε πότε μεγιστοποιείται ο λογάριθμος και ποια τιμή παίρνει. Για το k έχω βρει ότι ισχύει η ανισότητα $k \le \frac{z}{y}$. Αν τώρα θεωρήσω ότι το y παίρνει την τιμή 1, πράγμα που σημαίνει ότι το μέγεθος του μπλοκ είναι 1 byte (θεωρητικά εφικτή περίπτωση), τότε προκύπτει η ανισότητα $k \le z$. Για να μεγιστοποιήσω το λογάριθμο θα πάρω την περίπτωση της ισότητας, δηλαδή την περίπτωση $k = z$. Όλες αυτές οι υποθέσεις οδήγησαν στην πράξη στο να εκφυλιστεί η κρυφή μνήμη σε μια κανονική μνήμη, αφού δεν θα υπάρχει καθόλου το πεδίο δείκτης ούτε το πεδίο της απόκλισης μπλοκ. Σε αυτήν τη ακραία περίπτωση θα προκύψει η σχέση:

$$t = 62 - \log_2 \left(\frac{z}{k} \right) \Rightarrow t = 62 - \log_2 (1) \Rightarrow t = 62 - 0 \Rightarrow t = 62$$

Όπως έχει αναφερθεί στη μνήμη ετικετών μαζί με το πεδίο της ετικέτας θα κρατείται και το bit εγκυρότητας. Οπότε το μέγιστο πλάτος που θα χρειαστούμε ποτέ από μια μνήμη ετικετών είναι:
62+1=63
Αυτή είναι και η τιμή που έδωσα στην παράμετρο του πλάτους σε όσες μνήμες ετικετών έφτιαξα με το εργαλείο.
Το ύψος της μνήμης ετικετών και κατ' επέκταση της μνήμης δεδομένων σχετίζεται κυρίως με το μέγεθος της κρυφής μνήμης. Άρα το εύρος του μεγέθους κρυφής μνήμης

που θέλουμε να υποστηρίξουμε είναι αυτό που θα μας καθορίσει και τα ύψη των μνημών. Απευθυνόμενος στον επιβλέποντα καθηγητή μου προτάθηκε να δημιουργήσω ένα σύστημα που η κύρια μνήμη του θα είχε μέγεθος 256 KB ενώ η κρυφή μνήμη θα μπορούσε να έχει μέγεθος 8-32 KB. Φυσικά τα νούμερα είναι ενδεικτικά αλλά απαραίτητα στο να γνωρίζω τι πρέπει να φτιάξω.
Ας υποθέσουμε ότι θέλουμε να φτιάξουμε μια κρυφή μνήμη με μέγεθος 8 KB. Έστω ότι πρόκειται για μια άμεσα αντιστοιχισμένη μνήμη με μέγεθος μπλοκ μία λέξη (4 bytes). Αν διαιρέσουμε το μέγεθος της κρυφής μνήμης με το μέγεθος του μπλοκ θα πάρουμε το ύψος της μνήμης ετικετών. Στη συγκεκριμένη περίπτωση έχουμε:

$$\text{Ύψος μνήμης ετικετών} = \left(\frac{8 \times 2^{10}}{4} \right) = 2 \times 2^{10} = 2048$$

Αν αντί για 8 KB θέλαμε μέγεθος κρυφής μνήμης 32 KB τότε θα βρίσκαμε ύψος μνήμης ετικετών 8192. Αυτά είναι περίπου τα όρια των μνημών που πρέπει να υποστηρίξω. Βέβαια τα 8 KB μπορούν να επιτευχθούν και σαν μία συνολοσυσχετιστική μνήμη π.χ. 4 δρόμων όπου το ύψος κάθε μνήμης ετικετών θα είναι σε αυτή την περίπτωση 512.
Ακολουθεί ένας πίνακας που παρουσιάζει τις μνήμες που τελικά κατασκεύασα με το Xilinx CORE Generator και την επιλογή Single Port Block Memory 6.2.

Όνομα μνήμης	**Πλάτος μνήμης**	**Ύψος μνήμης**
tag_mem_8	63	256
tag_mem_9	63	512
tag_mem_10	63	1024
tag_mem_11	63	2048
tag_mem_12	63	4096
tag_mem_13	63	8192
main_memory	32	65536

Η τελευταία μνήμη είναι όπως δηλώνει και το όνομά της η κύρια μνήμη του συστήματος και έχει μέγεθος 256 KB. Το πλάτος είναι 32 bits που είναι το μέγεθος μιας λέξης.

Μνήμες δεδομένων

Το πλάτος των μνημών δεδομένων είναι γνωστό, 32 bits. Το ύψος τους σχετίζεται με αυτό των μνημών ετικετών. Όταν το μπλοκ έχει μέγεθος 1 λέξη τότε το ύψος της μνήμης δεδομένων είναι το ίδιο με αυτό της αντίστοιχης μνήμης ετικετών. Όταν το μπλοκ έχει μέγεθος 2 ή 4 λέξεις τότε το ύψος της μνήμης δεδομένων είναι διπλάσιο ή τετραπλάσιο από αυτό της μνήμης ετικετών αντίστοιχα. Επομένως δεν θα είχε νόημα να κατασκευάσουμε μια μνήμη δεδομένων με ύψος μικρότερο από το ύψος της μικρότερης μνήμης ετικετών. Επιπλέων το μέγεθος των 32 KB της κρυφής μνήμης ορίζει και το ύψος της μεγαλύτερης μνήμης δεδομένων που πρέπει να υποστηρίξω (ύψους 8192).
Σύμφωνα με τα παραπάνω δημιούργησα μνήμες δεδομένων αντίστοιχου ύψους με τις μνήμες ετικετών. Ακολουθεί ένας πίνακας που παρουσιάζει τις μνήμες που τελικά κατασκεύασα με το Xilinx CORE Generator και την επιλογή Single Port Block Memory 6.2

Όνομα μνήμης	Πλάτος μνήμης	Ύψος μνήμης
data_mem_8	32	256
data_mem_9	32	512
data_mem_10	32	1024
data_mem_11	32	2048
data_mem_12	32	4096
data_mem_13	32	8192

Η κύρια μνήμη, η μνήμη ετικετών και η μνήμη δεδομένων είναι απαραίτητες σε κάθε υλοποίηση κρυφής μνήμης. Υπάρχει όμως και ένας τέταρτος τύπος μνήμης που κατασκεύασα. Η βρώμικη μνήμη**.** Αυτή χρειάζεται για να υποστηριχθεί η κρυφή μνήμη ετεροχρονισμένης εγγραφής με κατανομή εγγραφής. Στην πράξη η ετεροχρονισμένη εγγραφή απαιτεί την υποστήριξη ενός bit για κάθε μπλοκ του οποίου η τιμή δηλώνει αν το μπλοκ έχει γραφτεί ή όχι. Η βρώμικη μνήμη έχει επομένως πλάτος 1 bit και ύψος ίδιο ακριβώς με τη μνήμη ετικετών. Οι βρώμικες μνήμες κατασκευάστηκαν όπως και οι υπόλοιπες μνήμες με το Xilinx CORE Generator και την επιλογή Single Port Block Memory 6.2 και φαίνονται στο παρακάτω πίνακα.

Όνομα μνήμης	Πλάτος μνήμης	Ύψος μνήμης
dirty_mem_8	1	256
dirty_mem_9	1	512
dirty_mem_10	1	1024
dirty_mem_11	1	2048
dirty _mem_12	1	4096
dirty _mem_13	1	8192

Κύρια μνήμη

Προηγουμένως ανέφερα πως δημιούργησα την κύρια μνήμη. Η κύρια μνήμη έχει σαφώς μεγαλύτερο μέγεθος από την κρυφή μνήμη. Πρέπει να έχει όμως και ένα ακόμα χαρακτηριστικό. Να είναι πιο αργή από την κρυφή μνήμη. Αυτό το χαρακτηριστικό που θεωρητικά καθορίζει και την καθυστέρηση που έχει μια αστοχία, το δημιούργησα βάζοντας στην έξοδο της κύριας μνήμης αρκετούς καταχωρητές. Έτσι μπορεί η μνήμη να βγάζει αμέσως τα δεδομένα, όμως τα τελευταία θα καθυστερήσουν για μερικούς κύκλους πριν πάνε οπουδήποτε μέσα στο σύστημα. Ακολουθεί ο αντίστοιχος κώδικας:

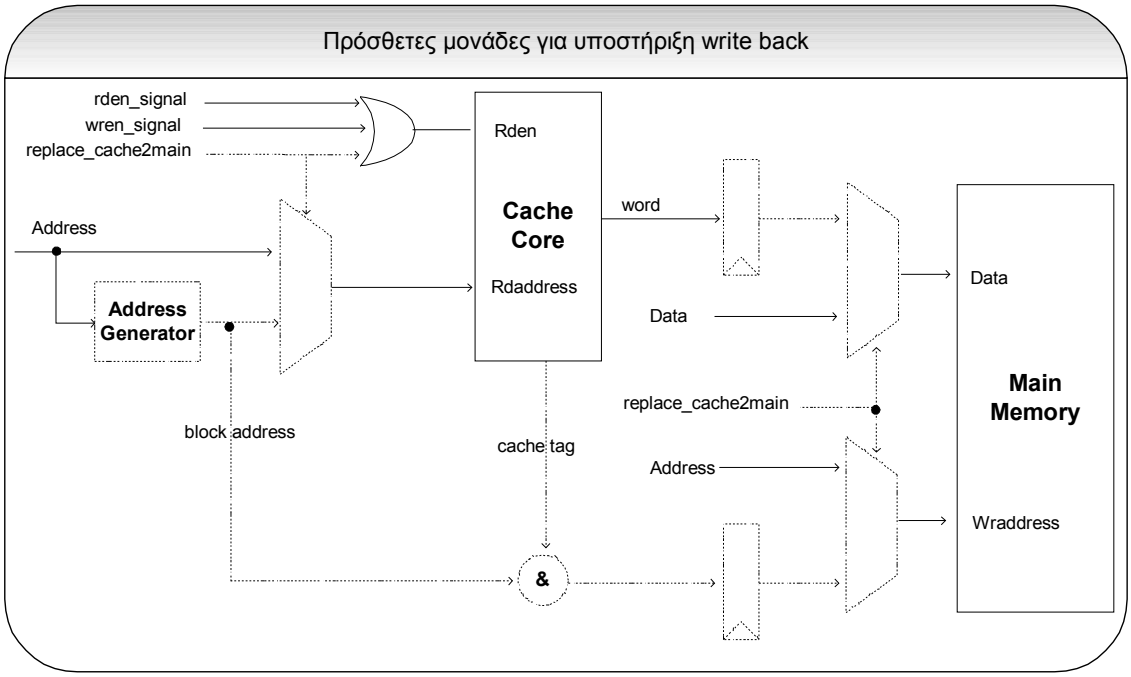
```
ENTITY delay_unit IS
	PORT
	(
		Mem_word_in		:IN STD_LOGIC_VECTOR(31 DOWNTO 0);
		LdEn			:IN STD_LOGIC;
		Reset			:IN STD_LOGIC;
		Clock			:IN STD_LOGIC;
		Mem_word_out	:OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
	);
END delay_unit;

ARCHITECTURE structure OF delay_unit IS
	COMPONENT reg_n_bit
		GENERIC
		(
			n	:INTEGER :=4
		);
```

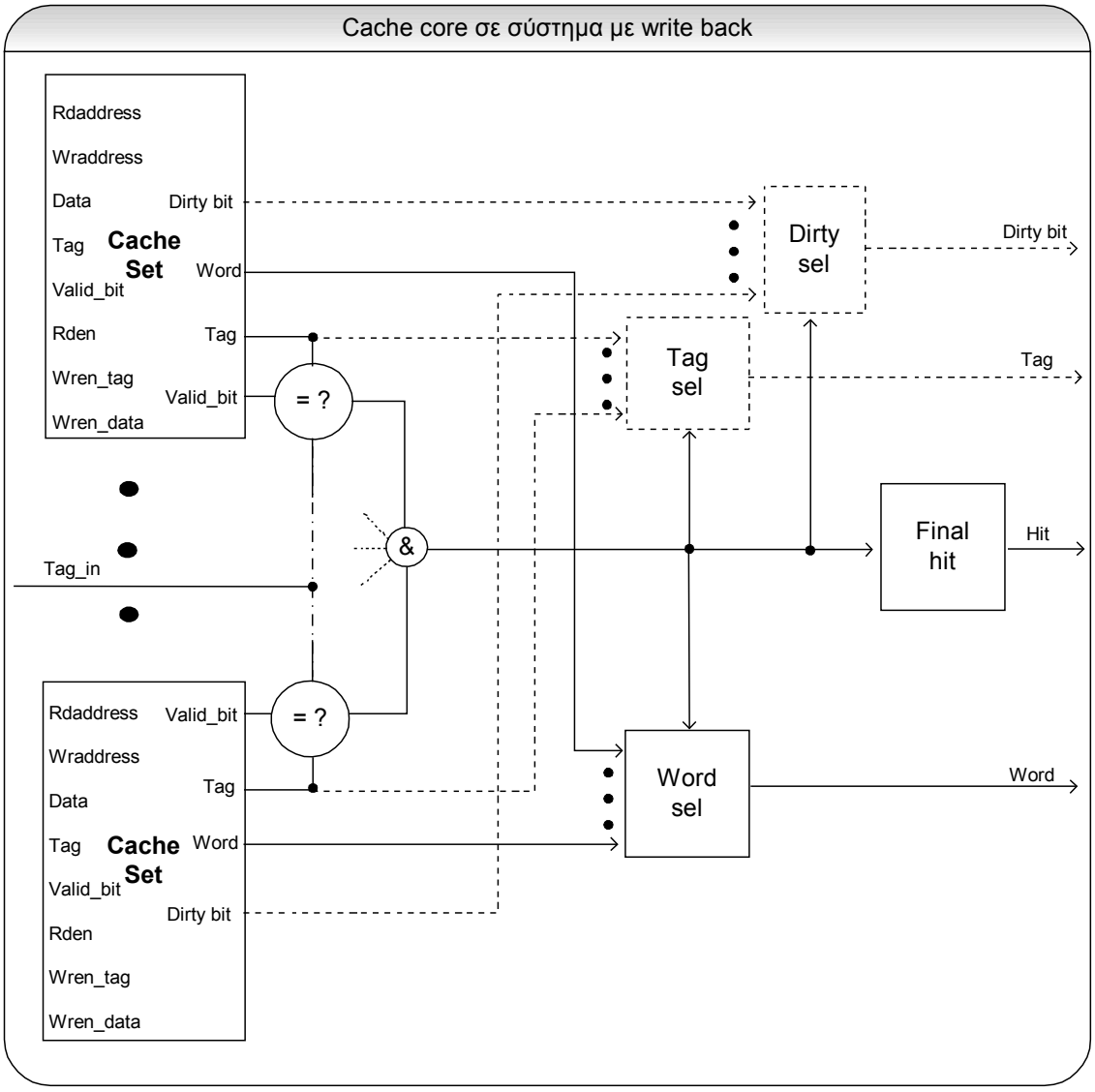
```
		PORT
		(
			X	:IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
			LdEn	:IN STD_LOGIC;
			Reset	:IN STD_LOGIC;
			Clk	:IN STD_LOGIC;
			Y	:OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)
		);
	END COMPONENT;

	SIGNAL
mem_word_2,mem_word_3,mem_word_4,mem_word_5,mem_word_6,mem_word_7,mem_
word_8,mem_word_9,mem_word_10	:STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN

	mem_res_reg_1:reg_n_bit
		GENERIC MAP
		(
			n=>32
		)
		PORT MAP
		(
			X=>Mem_word_in,
			LdEn=>LdEn,
			Reset=>Reset,
			Clk=>Clock,
			Y=>mem_word_2
		);

	mem_res_reg_2:reg_n_bit
		GENERIC MAP
		(
			n=>32
		)
		PORT MAP
		(
			X=>mem_word_2,
			LdEn=>LdEn,
			Reset=>Reset,
			Clk=>Clock,
			Y=>mem_word_3
		);

			.
			.
			.

	mem_res_reg_10:reg_n_bit
		GENERIC MAP
		(
			n=>32
		)
		PORT MAP
		(
			X=>mem_word_10,
			LdEn=>LdEn,
			Reset=>Reset,
			Clk=>Clock,
			Y=>Mem_word_out
		);
END structure;
```

Ο παραπάνω κώδικας είναι απλοποιημένος. Δεν παρουσιάζεται με την τελική μορφή του αλλά παρουσιάζει το κομμάτι της σχεδίασης που αναλύουμε τώρα. Τα στοιχεία που παραλείπονται θα δειχτούν αργότερα. Παρατηρούμε ότι οι καταχωρητές καθυστέρησης είναι δέκα. Αυτό είναι μια υπόθεση δική μου. Η αλλαγή του κώδικα για να προστεθούν ή να αφαιρεθούν καταχωρητές είναι πολύ απλή και γρήγορη υπόθεση.
Η κύρια μνήμη (main memory) μαζί με τη μονάδα καθυστέρησης (delay unit) αποτελούν το υποσύστημα κύριας μνήμης που θα δειχτεί και σε σχήμα αργότερα.

4.7 Υποστήριξη ετεροχρονισμένης εγγραφής

Το τρίτο σύστημα χρησιμοποιεί ετεροχρονισμένη εγγραφή (write back). Για να υποστηριχθεί κάτι τέτοι χρειάζεται ο πυρήνας της σχεδίασης αλλά και η διασύνδεσή του με την κύρια μνήμη να αλλάξει. Στην πράξη πρόσθεσα κάποιες μονάδες στη σχεδίαση που ήδη είχα κάνει.
Κάθε σετ του πυρήνα δίνει ως έξοδο ένα dirty bit. Μια μονάδα, ανάλογη αυτής που επιλέγει τη λέξη, οδηγεί το κατάλληλο bit προς την έξοδο του πυρήνα. Το bit αυτό πηγαίνει στη μηχανή πεπερασμένων καταστάσεων που ελέγχει το σύστημα. Εκτός όμως από το βρώμικο bit η ετεροχρονισμένη εγγραφή χρειάζεται και την ετικέτα του μπλοκ το οποίο είναι βρώμικο και θα μεταφερθεί στην κύρια μνήμη. Η ετικέτα αυτή είναι απαραίτητη στη δημιουργία της σωστής διεύθυνσης εγγραφής της κύριας μνήμης, κατά τη μεταφορά του μπλοκ. Έτσι μια ανάλογη μονάδα επιλογής ετικέτας προστίθεται στη σχεδίαση.
Οι διευθύνσεις ανάγνωσης του βρώμικού μπλοκ από τη κρυφή μνήμη παράγονται από μια μονάδα – γεννήτρια διευθύνσεων. Τα δεδομένα διαβάζονται από την κρυφή μνήμη και στον επόμενο κύκλο γράφονται στην κύρια μνήμη. Η τοποθέτηση καταχωρητών έγινε για να συγχρονιστεί σωστά αυτή η λειτουργία. Τα επιπλέον σήματα που εισήχθησαν κατέστησαν απαραίτητους και νέους πολυπλέκτες.
Όλα τα προηγούμενα φαίνονται στα σχήματα που ακολουθούν, όπου οι νέες μονάδες και σήματα έχουν διαφορετική γραμμοσκίαση.





4.8 Έλεγχος ροής δεδομένων – Καταστάσεις συστήματος

Αφού έδειξα τη σχεδίαση του πυρήνα της κρυφής μνήμης, θα αναλύσω πως θα λειτουργεί το σύστημα βάσει των εισόδων που θα παίρνει και από αυτό θα οδηγηθούμε στον έλεγχο που πρέπει να υποστηρίξουμε. Υπενθυμίζω ότι το σύστημα υποστηρίζει αιτήσεις ανάγνωσης και εγγραφής δεδομένων. Οι αιτήσεις αυτές έρχονται από τον επεξεργαστή και ικανοποιούνται σε σειρά. Μόλις δηλαδή ικανοποιηθεί η πρώτη αίτηση το σύστημα είναι έτοιμο να δεχτεί και να ικανοποιήσει τη δεύτερη κ.ο.κ.
Επειδή όπως έχω ήδη αναφέρει η σχεδίαση αφορά τρία διαφορετικά είδη κρυφής μνήμης η ανάλυση που θα ακολουθήσει θα αναφέρεται σε κάθε είδος ξεχωριστά.

1ο Σύστημα κρυφής μνήμης: Διεγγραφής με κατανομή μη εγγραφής

Όταν το σύστημα ξεκινάει (αρχική κατάσταση) καλό είναι να μην συμβαίνει τίποτα πέρα από την αρχικοποίηση όλων των καταχωρητών. Αυτή η κατάσταση θα κρατάει μόνο έναν κύκλο. Η αρχικοποίηση της κρυφής μνήμης αποτελεί μια ξεχωριστή κατάσταση στην οποία μεταβαίνει το σύστημα μόλις δεχτεί μια ανάλογη αίτηση (μέσω του σήματος Reset). Στην κατάσταση αυτή παραμένει μέχρι να ολοκληρωθεί η

αρχικοποίηση. Πόσους κύκλους διαρκεί αυτό; Δεν είναι σταθερή η τιμή, γι' αυτό όπως έχω δείξει ο τερματισμός της διαδικασίας γίνεται αντιληπτός μέσω ενός σήματος (Finish_reset). Αμέσως μετά μεταβαίνουμε στην αρχική κατάσταση όπου μηδενίζονται οι καταχωρητές.

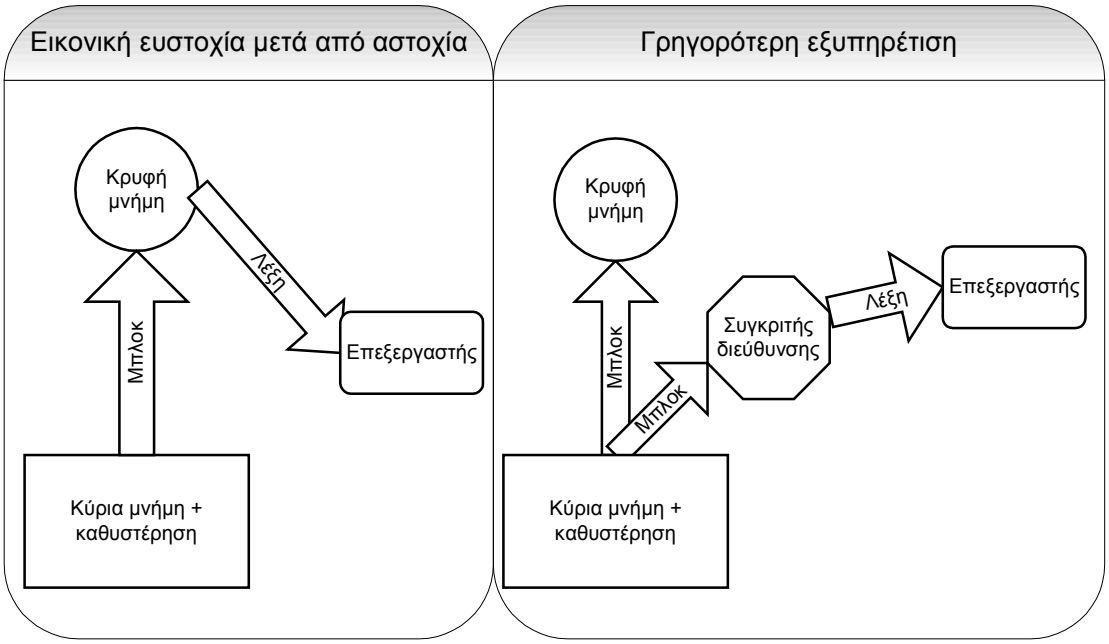
Τώρα προχωράμε στις καταστάσεις της κύριας λειτουργίας. Μετά την αρχική κατάσταση το σύστημα μεταβαίνει στη κατάσταση όπου ελέγχει κάποιες εισόδους για να δει αν έχει κάποια αίτηση. Ας ονομάσουμε τη κατάσταση αυτή ως κατάσταση ηρεμίας. Όσο δεν υπάρχει κάποια αίτηση το σύστημα παραμένει στην ίδια αυτή κατάσταση. Ο έλεγχος γίνεται για τις ακόλουθες αιτήσεις:

1. Αίτηση ανάγνωσης: Αφού το σύστημα καταλάβει ότι υπάρχει αίτηση εγγραφής ελέγχει ένα άλλο σήμα που ενημερώνει αν υπάρχει ευστοχία ή αστοχία. Η ροή των δεδομένων πρέπει να είναι τέτοια ώστε το συγκεκριμένο σήμα (Hit) να έχει τη σωστή τιμή τη στιγμή του ελέγχου. Για παράδειγμα αν έχουμε ευστοχία αλλά το σήμα Hit δεν έχει πάρει ακόμα τη τιμή 1 τη στιγμή του ελέγχου, τότε το σύστημα θα καταλάβει ότι έχουμε αστοχία.

Αν έχουμε ευστοχία το σύστημα δεν αλλάζει κατάσταση αλλά παραμένει στη κατάσταση ηρεμίας. Αν άλλαζε κατάσταση θα σήμαινε ότι δεν θα είχαμε ρυθμό εξυπηρέτησης μία αίτηση ανά κύκλο αλλά μικρότερο (υποθέτουμε πάντα εύστοχες αιτήσεις). Άρα αφού δεν υπάρχει αλλαγή κατάστασης σημαίνει πως η ροή των δεδομένων από μόνη της βγάζει προς την έξοδο του συστήματος τις κατάλληλες τιμές στα σήματα. Στην ουσία αυτό που γίνεται είναι να οδηγούνται τα δεδομένα που βρέθηκαν στην κρυφή μνήμη σε μια έξοδο και παράλληλα ένα σήμα που ειδοποιεί τον έξω κόσμο (επεξεργαστής) ότι τα δεδομένα είναι έτοιμα για να τα πάρει και ότι μπορεί να δώσει την επόμενη αίτηση. Όλα αυτά γίνονται σε ένα κύκλο.

Αν έχουμε αστοχία σημαίνει ότι τα ζητούμενα δεδομένα δεν υπάρχουν στην κρυφή μνήμη και πρέπει να ανακτηθούν από την κύρια μνήμη. Έτσι λοιπόν η επόμενη κατάσταση είναι αυτή της ανάγνωσης της κύριας μνήμης. Τα δεδομένα του ζητούμενου μπλοκ διαβάζονται από την κύρια μνήμη και στέλνονται στην κατάλληλη θέση της κρυφής μνήμης ενώ η λέξη που έχει ζητηθεί στέλνεται και προς τον έξω κόσμο. Το σύστημα μένει στην κατάσταση αυτή για όσο διαβάζεται η κύρια μνήμη. Ο χρόνος αυτός είναι ανάλογος με το μέγεθος του μπλοκ. Αν το μπλοκ είναι 2 λέξεις χρειάζονται δύο κύκλοι (ένας για κάθε λέξη) ενώ αν έχει μέγεθος 4 λέξεις χρειάζονται 4 κύκλοι αντίστοιχα. Επειδή πάλι δεν έχουμε σταθερό αριθμό κύκλο ελέγχουμε ένα σήμα που ειδοποιεί για λήξη της ανάγνωσης (Finish _read_main). Θα υπέθετε κανείς ότι η επόμενη κατάσταση θα είναι αυτή της ηρεμίας. Όμως κάτι τέτοιο θα σήμαινε ότι το σύστημα είναι έτοιμο να δεχτεί και να ικανοποιήσει μια επόμενη αίτηση. Όπως έχω όμως ήδη αναφέρει τα δεδομένα από την κύρια μνήμη δεν πάνε αμέσως οπουδήποτε αλλά περνάνε από μερικούς καταχωρητές. Έτσι υπάρχουν κάποιοι κύκλοι καθυστέρησης μέχρι να φθάσουν οι λέξεις του μπλοκ στην κρυφή μνήμη. Η κατάσταση αυτή ονομάζεται καθυστέρηση αστοχίας και τελειώνει μόλις το σήμα Finish_miss πάρει την τιμή 1. Ύστερα από αυτή τη κατάσταση επιστρέφουμε στην κατάσταση ηρεμίας. Τα ζητούμενα δεδομένα μπορούσαν να αναγνωσθούν από την κρυφή μνήμη ύστερα από την ανάκτησή τους, δημιουργώντας μια εικονική ευστοχία ύστερα από την αντίστοιχη αστοχία. Όμως εγώ επέλεξα να δίνονται κατευθείαν στον έξω κόσμο κατά τη μεταφορά τους από την κύρια προς την κρυφή μνήμη. Έτσι ελαχιστοποίησα το χρόνο εξυπηρέτησης μιας αίτησης αστοχίας. Ο έξω κόσμος παίρνει τα ζητούμενα δεδομένα πριν ολοκληρωθεί στο σύστημα η διαδικασία αντικατάστασης του μπλοκ. Έτσι έχουμε μεν ικανοποίηση της αίτησης αλλά όχι δυνατότητα διαβάσματος επόμενης αίτησης. Η επόμενη αίτηση θα μπορεί να εξυπηρετηθεί μόνο όταν πάμε στην κατάσταση ηρεμίας. Ένα άλλο σημείο που θέλω να υπογραμμίσω είναι ότι μπλοκ

διαβάζεται από την κύρια μνήμη με τη σειρά που έχουν οι διευθύνσεις των αντίστοιχων λέξεων. Π.χ σε ένα μπλοκ μεγέθους 4 διαβάζεται πρώτα η λέξη 00, μετά η λέξη 01 , η 10 και τέλος η 11. Άρα ήταν υπερβολή η προηγούμενη φράση για ελαχιστοποίηση του χρόνου εξυπηρέτησης της αίτησης. Ελαχιστοποίηση θα είχαμε αν διαβάζαμε πρώτα τη λέξη που μας έχει ζητηθεί και ύστερα το υπόλοιπο μπλοκ. Ο λόγος που δεν το έκανα ήταν για απλότητα της σχεδίασης.



Στην παραπάνω εικόνα φαίνονται οι δύο τρόποι με τους οποίους θα μπορούσε το σύστημα να εξυπηρετήσει τον επεξεργαστή. Εγώ επέλεξα τον δεύτερο, που για να υλοποιηθεί θέλει μια πρόσθετη μονάδα που κάνει έλεγχο διευθύνσεων, ώστε να μην δώσουμε στον επεξεργαστή όλο το μπλοκ αλλά μόνο τη λέξη που ζήτησε.

2. Αίτηση εγγραφής: Αφού το σύστημα καταλάβει ότι υπάρχει αίτηση εγγραφής ελέγχει πάλι το σήμα που ενημερώνει αν υπάρχει ευστοχία ή αστοχία.
Αν έχουμε ευστοχία το σύστημα δεν αλλάζει κατάσταση αλλά παραμένει στη κατάσταση ηρεμίας. Η ροή των δεδομένων που ήδη έχω δείξει είναι τέτοια ώστε να γίνει η εγγραφή στο πρώτο κύκλο που είναι διαθέσιμη η μνήμη δεδομένων.
Σε μια αστοχία πάλι η κρυφή μνήμη δεν επηρεάζετε αλλά τα δεδομένα γράφονται μόνο στη κύρια μνήμη αφού έχουμε διεγγραφή και κατανομή μη εγγραφής.

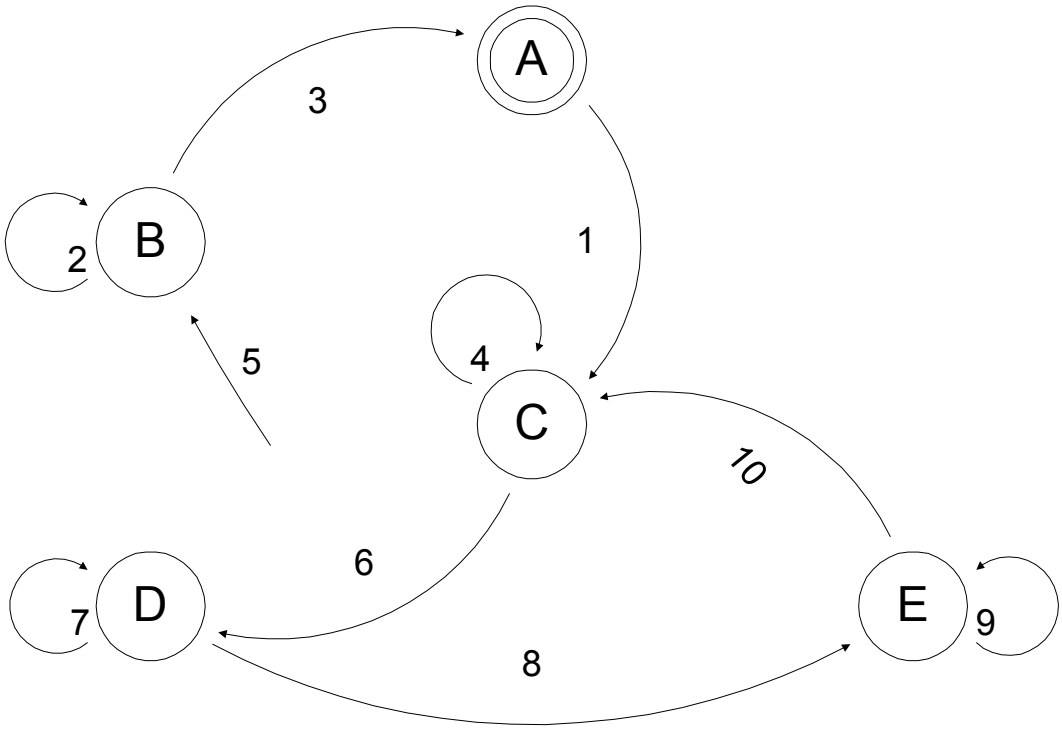
Ακολουθεί η μηχανή πεπερασμένων καταστάσεων που υλοποιεί την παραπάνω ανάλυση.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY nwa_cache_control IS
	PORT
	(
		Cache_rden		:IN STD_LOGIC;
		Hit			:IN STD_LOGIC;
		Reset			:IN STD_LOGIC;
```

```
		Finish_reset  :IN STD_LOGIC;
		Finish_read_main	:IN STD_LOGIC;
		Finish_miss		:IN STD_LOGIC;
		Clock			:IN STD_LOGIC;
		Start			:OUT STD_LOGIC;
		LdEn			:OUT STD_LOGIC;
		Scan			:OUT STD_LOGIC;
		Reset_state		:OUT STD_LOGIC;
		Replace_block :OUT STD_LOGIC;
		State			:OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
	);
END nwa_cache_control;

ARCHITECTURE behavioral OF nwa_cache_control IS
	TYPE State_type IS (A,B,C,D,E);
	SIGNAL y_present,y_next	:State_type;
BEGIN
	PROCESS
(Cache_rden,Hit,Reset,Finish_reset,Finish_read_main,Finish_miss,y_pres
ent)
	BEGIN
		CASE y_present IS
			WHEN A=>--Initialize
				y_next<=C;
			WHEN B=>--reset
				IF (Finish_reset='0') THEN
					y_next<=B;
				ELSE
					y_next<=A;
				END IF;
			WHEN C=>--KANONIKI THESI PULLING
				IF (Reset='1') THEN
					y_next<=B;
				ELSIF (Cache_rden='1') THEN
					IF (Hit='0') THEN
						y_next<=D;
					ELSE
						y_next<=C;
					END IF;
				ELSE
					y_next<=C;
				END IF;
			WHEN D=>--miss diabasma
				IF (Finish_read_main='0') THEN
					y_next<=D;
				ELSE
					y_next<=E;
				END IF;
			WHEN E=>--miss delay
				IF (Finish_miss='0') THEN
					y_next<=E;
				ELSE
					y_next<=C;
				END IF;
			END CASE;
	END PROCESS;

	PROCESS(Clock)
	BEGIN
		IF (Clock' EVENT AND Clock='0') THEN
			y_present<=y_next;
```

```
            END IF;
        END PROCESS;

        Start<='1' WHEN (y_present=A) ELSE
                     '0';

        LdEn<='0' WHEN (y_present=A OR y_present=B) ELSE
               '1';

        Scan<='1' WHEN (y_present=C) ELSE
               '0';

        Reset_state<='1' WHEN (y_present=B) ELSE
                             '0';

        Replace_block<='1' WHEN (y_present=D) ELSE
                             '0';

        State<="000" WHEN y_present=A ELSE
                      "001" WHEN y_present=B ELSE
                      "010" WHEN y_present=C ELSE
                      "011" WHEN y_present=D ELSE
                      "100" WHEN y_present=E ELSE
                      "101";
END behavioral;
```

Τα σήματα εξόδων μπορεί να μην είναι όλα προφανή. Πάνε προς σε μονάδες που θα αναλυθούν αργότερα.
Παρακάτω δίνεται το διάγραμμα ροής της μηχανής πεπερασμένων καταστάσεων το οποίο συνοψίζει τον παραπάνω κώδικα και είναι ευκολότερα αναγνώσιμο.



Καταστάσεις

A: αρχική κατάσταση
B: αρχικοποίηση (στην καταστάση αυτή μεταβαίνει το σύστημα από οποιαδήποτε άλλη κατάσταση εφόσον γίνει reset από τον έξω κόσμο)
C: ηρεμία
D: ανάγνωση κύριας μνήμης
E: καθυστέρηση αστοχίας

Μεταβάσεις

1: αυτόματη μετάβαση (η κατάσταση Α κρατάει πάντα έναν κύκλο)
2: `Finish_reset='0'`
3: `Finish_reset='1'`
4: `Reset='0' και ((Cache_rden='0') ή (Cache_rden='1' και Hit='1'))`
5: `Reset='1'`
6: `Cache_rden='1'και Hit='0'`
7: `Finish_read_main='0'`
8: `Finish_read_main='1'`
9: `Finish_miss='0'`
10: `Finish_miss='1'`

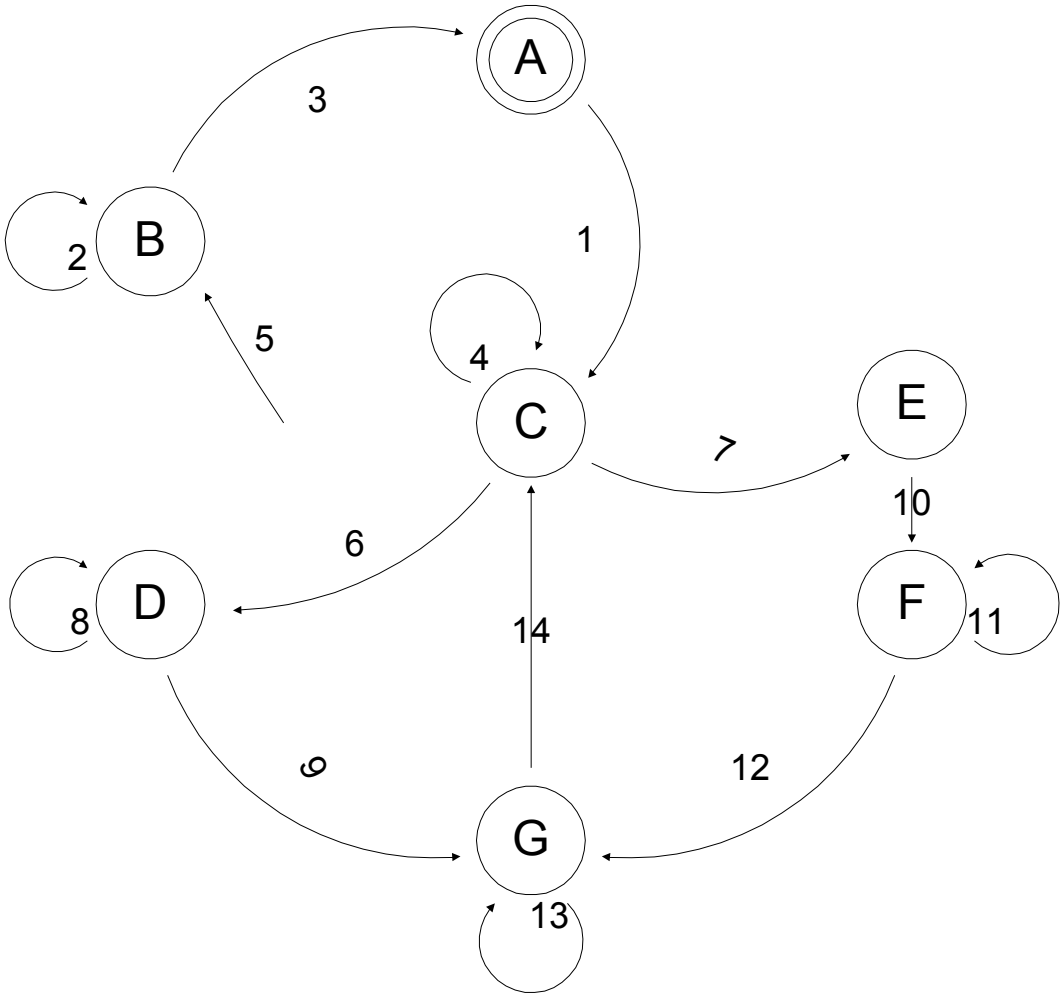
2ο Σύστημα κρυφής μνήμης: Διεγγραφής με κατανομή εγγραφής

Το δεύτερο σύστημα είναι παρόμοιο με το πρώτη. Συμπεριφέρεται φυσικά με τον ίδιο τρόπο για την αρχικοποίηση, αλλά και για τις αιτήσεις ανάγνωσης. Εκεί που διαφέρει είναι η εξυπηρέτηση των αιτήσεων εγγραφής και αυτό γιατί έχουμε κατανομή εγγραφής. Κατανομή εγγραφής σημαίνει ότι σε μια ανεπιτυχή αίτηση εγγραφής το μπλοκ που περιέχει την επιθυμητή λέξει μεταφέρεται στην κρυφή μνήμη και δεσμεύει κάποιον αντίστοιχο χώρο σε αυτήν, ακριβώς όπως συμβαίνει και με μια ανεπιτυχή ανάγνωση. Επομένως όταν το σύστημα εξυπηρετεί μια αίτηση εγγραφής ισχύει:
Αν η αίτηση είναι εύστοχη το σύστημα συνεχίζει να βρίσκεται στην κατάσταση ηρεμίας αφού η επιθυμητή εγγραφή γίνεται όπως και στο πρώτο σύστημα κρυφής μνήμης. Αν προκύψει αστοχία τότε αρχικά γράφουμε τη λέξη στην κατάλληλη θέση της κύριας μνήμης. Η κατάσταση αυτή ονομάζεται ενημέρωση κύριας μνήμης, κρατάει έναν κύκλο και αποτελεί το πρώτο βήμα της ικανοποίησης μιας εγγραφής που είχε αστοχία. Το δεύτερο βήμα είναι η ανάγνωση της κύριας μνήμης για τη μεταφορά του μπλοκ στην κρυφή μνήμη, όπως γίνεται και σε μια άστοχη αίτηση ανάγνωσης. Εδώ η διαφορά είναι ότι δεν δίνεται καμία λέξη προς τον έξω κόσμο. Πριν περάσουμε στην κατάσταση ηρεμίας μεσολαβεί πάλι η κατάσταση καθυστέρησης αστοχίας.
Τα παραπάνω δείχνονται με τον κώδικα που ακολουθεί καθώς και το αντίστοιχο σχήμα.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

```
ENTITY wa_cache_control IS
	PORT
	(
		Cache_rden		:IN STD_LOGIC;
		Cache_wren		:IN STD_LOGIC;
		Hit			:IN STD_LOGIC;
		Reset			:IN STD_LOGIC;
		Finish_reset	:IN STD_LOGIC;
		Finish_read_main	:IN STD_LOGIC;
		Finish_miss		:IN STD_LOGIC;
		Clock			:IN STD_LOGIC;
		Start			:OUT STD_LOGIC;
		LdEn			:OUT STD_LOGIC;
		Scan			:OUT STD_LOGIC;
		Mem_wren		:OUT STD_LOGIC;
		Reset_state		:OUT STD_LOGIC;
		Replace_block	:OUT STD_LOGIC;
		Read_miss		:OUT STD_LOGIC;
		State			:OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
	);
END wa_cache_control;

ARCHITECTURE behavioral OF wa_cache_control IS
	TYPE State_type IS (A,B,C,D,E,F,G);
	SIGNAL y_present,y_next	:State_type;
BEGIN
	PROCESS
(Cache_rden,Cache_wren,Hit,Reset,Finish_reset,Finish_read_main,Finish_
miss,y_present)
	BEGIN
		CASE y_present IS
			WHEN A=>--Initialize
				y_next<=C;
			WHEN B=>--reset
				IF (Finish_reset='0') THEN
					y_next<=B;
				ELSE
					y_next<=A;
				END IF;
			WHEN C=>--KANONIKI THESI PULLING
				IF (Reset='1') THEN
					y_next<=B;
				ELSIF (Cache_rden='1') THEN
					IF (Hit='0') THEN
						y_next<=D;
					ELSE
						y_next<=C;
					END IF;
				ELSIF (Cache_wren='1') THEN
					IF (Hit='0') THEN
						y_next<=E;--grafw meta apo
miss
					ELSE
						y_next<=C;--grafw meta apo
hit
					END IF;
				ELSE
					y_next<=C;
				END IF;
			WHEN D=>--miss diabasma
				IF (Finish_read_main='0') THEN
```

```
                                y_next<=D;
                        ELSE
                                y_next<=G;
                        END IF;
                WHEN E=>--miss write 1st step
                        y_next<=F;
                WHEN F=>--miss write continue
                        IF (Finish_read_main='0') THEN
                                y_next<=F;
                        ELSE
                                y_next<=G;
                        END IF;
                WHEN G=>--miss delay
                        IF (Finish_miss='0') THEN
                                y_next<=G;
                        ELSE
                                y_next<=C;
                        END IF;
        END CASE;
END PROCESS;

PROCESS(Clock)
BEGIN
        IF (Clock' EVENT AND Clock='0') THEN
                y_present<=y_next;
        END IF;
END PROCESS;

Start<='1' WHEN (y_present=A) ELSE
                '0';

LdEn<='0' WHEN (y_present=A OR y_present=B) ELSE
        '1';

Scan<='1' WHEN (y_present=C) ELSE
        '0';

Mem_wren<='1' WHEN (y_present=E) ELSE
                '0';

Reset_state<='1' WHEN (y_present=B) ELSE
                        '0';

Replace_block<='1' WHEN (y_present=D OR y_present=F) ELSE
                        '0';

Read_miss<='1' WHEN (y_present=D) ELSE
                '0';

State<="000" WHEN y_present=A ELSE
                "001" WHEN y_present=B ELSE
                "010" WHEN y_present=C ELSE
                "011" WHEN y_present=D ELSE
                "100" WHEN y_present=E ELSE
                "101" WHEN y_present=F ELSE
                "110" WHEN y_present=G ELSE
                "111";
END behavioral;
```



Καταστάσεις

A: αρχική κατάσταση
B: αρχικοποίηση (στην καταστάση αυτή μεταβαίνει το σύστημα από οποιαδήποτε άλλη κατάσταση εφόσον γίνει reset από τον έξω κόσμο).
C: ηρεμία
D: ανάγνωση κύριας μνήμης (αίτηση ανάγνωσης)
E: ενημέρωση κύριας μνήμης
F: ανάγνωση κύριας μνήμης (αίτηση εγγραφής)
G: καθυστέρηση αστοχίας

Μεταβάσεις

1: αυτόματη μετάβαση (η κατάσταση A κρατάει πάντα έναν κύκλο)
2: `Finish_reset='0'`
3: `Finish_reset='1'`
4: `Reset='0' και ((Cache_rden='0' και Cache_wren='0') ή (Cache_rden='1' και Hit='1') ή (Cache_wren='1' και Hit='1'))`
5: `Reset='1'`
6: `Cache_rden='1'και Hit='0'`
7: `Cache_wren='1'και Hit='0'`

8: `Finish_read_main='0'`
9: `Finish_read_main='1'`
10: αυτόματη μετάβαση (η κατάσταση Ε κρατάει πάντα έναν κύκλο)
11: `Finish_read_main='0'`
12: `Finish_read_main='1'`
13: `Finish_miss='0'`
14: `Finish_miss='1'`

3ο Σύστημα κρυφής μνήμης: Ετεροχρονισμένης εγγραφής με κατανομή εγγραφής

Το τρίτο σύστημα είναι το πιο πολύπλοκο από τα τρία γι' αυτό και έχει τις περισσότερες καταστάσεις. Πριν τις αναλύσουμε θα περιγραφεί τι συμβαίνει κατά την ετεροχρονισμένη εγγραφή.

Στην ετεροχρονισμένη εγγραφή ένα μπλοκ της κρυφής μνήμης μπορεί να αλλάζει χωρίς να αλλάζει το αντίστοιχό του στην κύρια μνήμη. Η ενημέρωση της κύριας μνήμης γίνεται μόνο όταν το συγκεκριμένο μπλοκ πρόκειται να αντικατασταθεί από κάποιο άλλο, δηλαδή σε περίπτωση που έχουμε αστοχία σε αίτηση ανάγνωσης ή εγγραφής. Το αν ένα μπλοκ έχει αλλοιωθεί και άρα χρειάζεται να ενημερωθεί η κύρια μνήμη, μας το λέει η βρώμικη μνήμη. Το συμπέρασμα είναι ότι κάθε φορά που πάμε να διώξουμε ένα μπλοκ από την κρυφή μνήμη πρέπει πρώτα να ελέγξουμε αν είναι βρώμικο και να πράξουμε αναλόγως.

Ένα άλλο σημείο που πρέπει να σταθούμε είναι ο τρόπος που επιλέγεται ποιο μπλοκ θα διωχθεί. Η επιλογή αυτή χρειάζεται όταν η κρυφή μνήμη είναι συνολοσυσχετιστική. Η στρατηγική που χρησιμοποιείται σε όλα τα συστήματα είναι αυτή της τυχαίας αντικατάστασης. Ο όρος «τυχαία» δεν είναι ο ακριβέστερος. Αργότερα θα δείξουμε την μονάδα που επιλέγει το σύνολο στο οποίο θα γίνει η αντικατάσταση. Πάντως αυτό που πρέπει να γίνει κατανοητό είναι ότι όταν προκύψει μια αστοχία δεν ξέρουμε εκ των προτέρων ποιο μπλοκ πρόκειται να αντικατασταθεί. Αυτό έχει ιδιαίτερη σημασία τώρα, γιατί πρέπει να βρούμε ποιο μπλοκ είναι αυτό για να ελέγξουμε το αντίστοιχο βρώμικο bit.

Με βάση τα παραπάνω προχωράμε στην ανάλυση των καταστάσεων. Για τις πρώτες τρεις καταστάσεις ισχύουν τα ίδια με το δεύτερο σύστημα κρυφής μνήμης. Από την κατάσταση ηρεμίας, όταν έχουμε μια άστοχη αίτηση το σύστημα πηγαίνει σε μία από τις επόμενες καταστάσεις:

1. *Κατάσταση ειδοποίησης αστοχίας για ανάγνωση*
   Στέλνεται κατάλληλο σήμα στην μονάδα που επιλέγει το σύνολο στο οποίο θα γίνει η αντικατάσταση.
2. *Κατάσταση ειδοποίησης αστοχίας για εγγραφή*
   Ό,τι και στην προηγούμενη. Επιπλέον γράφεται η λέξη στην κύρια μνήμη. Έτσι κερδίζεται ο κύκλος που σπαταλούσαμε στο δεύτερο σύστημα (διεγγραφής με κατανομή εγγραφής) για να κάνουμε τη συγκεκριμένη εγγραφή (κατάσταση Ε σε εκείνο το σύστημα)

Στις επόμενες καταστάσεις ελέγχουμε αν το μπλοκ που πρόκειται να αντικατασταθεί είναι βρώμικο:

1. *Κατάσταση ελέγχου βρώμικου μπλοκ για ανάγνωση*
2. *Κατάσταση ελέγχου βρώμικου μπλοκ για εγγραφή*

Παρατηρούμε μια πλήρη αντιστοίχηση στις καταστάσεις της ανάγνωσης με αυτές τις εγγραφής. Επειδή όμως διαφέρουν σε ελάχιστα σημεία οι καταστάσεις είναι ξεχωριστές.

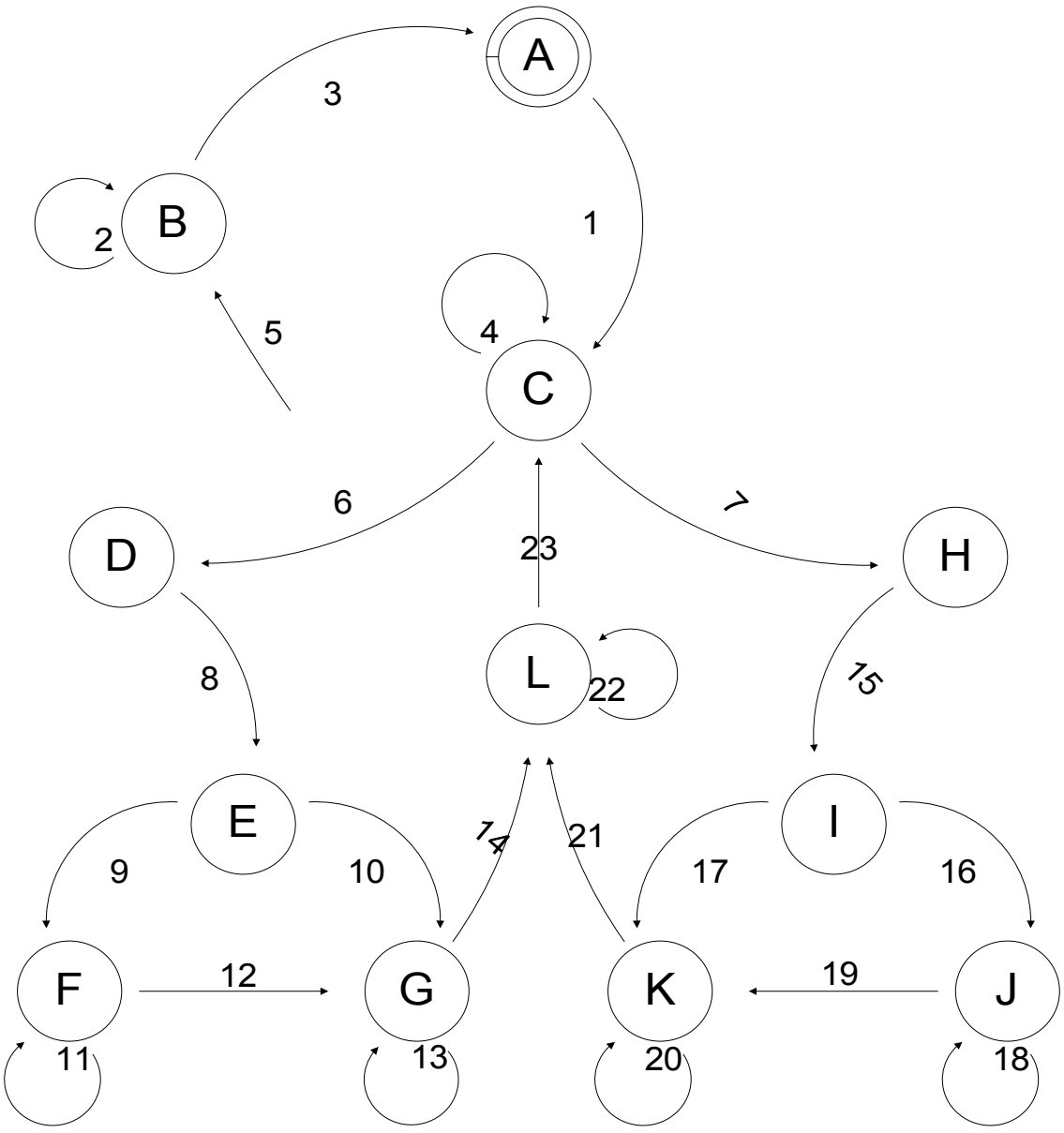
Αν το μπλοκ που ελέγχθει ήταν καθαρό περνάμε στις καταστάσεις αντικατάστασης του μπλοκ από την κύρια μνήμη στην κρυφή. Διαφορετικά, αν το μπλοκ βρέθηκε βρώμικο, κάνουμε πρώτα την κατάλληλη ενημέρωση της κύριας μνήμης μεταφέροντας σε αυτήν το βρώμικο μπλοκ. Η ενημέρωση αυτή αποτελεί μια ξεχωριστή κατάσταση που η διάρκειά της δεν είναι σταθερή αλλά εξαρτάται από το μέγεθος του μπλοκ, οπότε όπως και σε ανάλογες καταστάσεις φεύγουμε από αυτήν όταν ένα κατάλληλο σήμα εισόδου αλλάξει τιμή. Στη συνέχεια το σύστημα μεταβαίνει στην κατάσταση αντικατάστασης σαν επρόκειτο για καθαρό μπλοκ. Η τελευταία κατάσταση είναι όπως και στα δύο προηγούμενα συστήματα αυτή της καθυστέρησης αστοχίας. Πάλι η διαφορά των καταστάσεων αντικατάστασης ανάμεσα στην ανάγνωση και την εγγραφή είναι ότι κατά την ανάγνωση στέλνουμε την επιθυμητή λέξη και προς τον έξω κόσμο.
Τα παραπάνω δείχνονται με τον κώδικα που ακολουθεί καθώς και το αντίστοιχο σχήμα. Προσέξτε τον παραλληλισμό των καταστάσεων και τις μικρές διαφορές τους που φαίνονται στην ενημέρωση των σημάτων εξόδου.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY cache_control IS
	PORT
	(
		Cache_rden			:IN STD_LOGIC;
		Cache_wren			:IN STD_LOGIC;
		Hit				:IN STD_LOGIC;
		Reset				:IN STD_LOGIC;
		Finish_reset		:IN STD_LOGIC;
		Finish_read_main		:IN STD_LOGIC;
		Finish_read_cache		:IN STD_LOGIC;
		Finish_miss			:IN STD_LOGIC;
		Check_dirty			:IN STD_LOGIC;
		Clock				:IN STD_LOGIC;
		Start				:OUT STD_LOGIC;
		LdEn				:OUT STD_LOGIC;
		Scan				:OUT STD_LOGIC;
		Read_dirty			:OUT STD_LOGIC;
		Mem_wren			:OUT STD_LOGIC;
		Reset_state			:OUT STD_LOGIC;
		Cache2main_replace	:OUT STD_LOGIC;
		Future_replace		:OUT STD_LOGIC;
		Main2cache_replace	:OUT STD_LOGIC;
		Read_miss			:OUT STD_LOGIC;
		State				:OUT STD_LOGIC_VECTOR(3
DOWNTO 0)
	);
END cache_control;

ARCHITECTURE behavioral OF cache_control IS
	TYPE State_type IS (A,B,C,D,E,F,G,H,I,J,K,L);
	SIGNAL y_present,y_next	:State_type;
BEGIN
	PROCESS
(Cache_rden,Cache_wren,Hit,Reset,Finish_reset,Finish_read_main,Finish_
read_cache,Finish_miss,Check_dirty,y_present)
	BEGIN
		CASE y_present IS
			WHEN A=>--Initialize
				y_next<=C;
			WHEN B=>--reset
```

```
                        IF (Finish_reset='0') THEN
                                y_next<=B;
                        ELSE
                                y_next<=A;
                        END IF;
                WHEN C=>--KANONIKI THESI PULLING
                        IF (Reset='1') THEN
                                y_next<=B;
                        ELSIF (Cache_rden='1') THEN
                                IF (Hit='0') THEN
                                        y_next<=D;--diavazw meta apo
miss
                                ELSE
                                        y_next<=C;
                                END IF;
                        ELSIF (Cache_wren='1') THEN
                                IF (Hit='0') THEN
                                        y_next<=H;--grafw meta apo
miss
                                ELSE
                                        y_next<=C;
                                END IF;
                        ELSE
                                y_next<=C;
                        END IF;
                WHEN D=>--miss read 1st step: stelno sima oti exv
miss read gia dw poio tha replace
                        y_next<=E;
                WHEN E=>--miss read 2nd step:elegxv an auto pou
tha replace einai dirty
                        IF (Check_dirty='1') THEN
                                y_next<=F;
                        ELSE
                                y_next<=G;
                        END IF;
                WHEN F=>--miss read 3rd step: cache2main_replace
                        IF (Finish_read_cache='0') THEN
                                y_next<=F;
                        ELSE
                                y_next<=G;
                        END IF;
                WHEN G=>--miss read 4th step: main2cache_replace
                        IF (Finish_read_main='0') THEN
                                y_next<=G;
                        ELSE
                                y_next<=L;
                        END IF;
                WHEN H=>--miss write 1st step: stelno sima oti
exv miss write gia dv poio tha replace
                        y_next<=I;
                WHEN I=>--miss write 2nd step:elegxv an auto pou
tha replace einai dirty
                        IF (Check_dirty='1') THEN
                                y_next<=J;
                        ELSE
                                y_next<=K;
                        END IF;
                WHEN J=>--miss write 3rd step: cache2main_replace
                        IF (Finish_read_cache='0') THEN
                                y_next<=J;
                        ELSE
```

```
                                y_next<=K;
                        END IF;
                WHEN K=>--miss write 4th step: main2cache_replace
                        IF (Finish_read_main='0') THEN
                                y_next<=K;
                        ELSE
                                y_next<=L;
                        END IF;
                WHEN L=>--miss delay
                        IF (Finish_miss='0') THEN
                                y_next<=L;
                        ELSE
                                y_next<=C;
                        END IF;
        END CASE;
END PROCESS;

PROCESS(Clock)
BEGIN
        IF (Clock' EVENT AND Clock='0') THEN
                y_present<=y_next;
        END IF;
END PROCESS;

Start<='1' WHEN (y_present=A) ELSE
                '0';

LdEn<='0' WHEN (y_present=A OR y_present=B) ELSE
        '1';

Scan<='1' WHEN (y_present=C) ELSE
        '0';

Read_dirty<='1' WHEN (y_present=C OR y_present=D OR
y_present=H) ELSE
                '0';

Mem_wren<='1' WHEN (y_present=F OR y_present=H OR y_present=J)
ELSE
                '0';

Reset_state<='1' WHEN (y_present=B) ELSE
                        '0';

Cache2main_replace<='1' WHEN (y_present=E OR y_present=F OR
y_present=I OR y_present=J) ELSE
                                '0';

Future_replace<='1' WHEN (y_present=D OR y_present=H) ELSE
                        '0';

Main2cache_replace<='1' WHEN (y_present=G OR y_present=K) ELSE
                                '0';

Read_miss<='1' WHEN (y_present=G) ELSE
                '0';

State<="0000" WHEN y_present=A ELSE --0
                "0001" WHEN y_present=B ELSE --1
                "0010" WHEN y_present=C ELSE --2
                "0011" WHEN y_present=D ELSE --3
```

```
                    "0100" WHEN y_present=E ELSE --4
                    "0101" WHEN y_present=F ELSE --5
                    "0110" WHEN y_present=G ELSE --6
                    "0111" WHEN y_present=H ELSE --7
                    "1000" WHEN y_present=I ELSE --8
                    "1001" WHEN y_present=J ELSE --9
                    "1010" WHEN y_present=K ELSE --10
                    "1011" WHEN y_present=L ELSE --11
                    "1100";--12
END behavioral;
```



Καταστάσεις

A: αρχική κατάσταση

B: αρχικοποίηση (στην καταστάση αυτή μεταβαίνει το σύστημα από οποιαδήποτε άλλη κατάσταση εφόσον γίνει reset από τον έξω κόσμο)

C: ηρεμία

D: ειδοποίηση αστοχίας για ανάγνωση

E: έλεγχος βρώμικου μπλοκ για ανάγνωση

F: ανάγνωση κρυφής μνήμης – ενημέρωση κύριας μνήμης (αίτηση ανάγνωσης)
G: ανάγνωση κύριας μνήμης (αίτηση ανάγνωσης)
H: ειδοποίησης αστοχίας για εγγραφή και ενημέρωση κύριας μνήμης
I: έλεγχος βρώμικου μπλοκ για εγγραφή
J: ανάγνωση κρυφής μνήμης – ενημέρωση κύριας μνήμης (αίτηση εγγραφής)
K: ανάγνωση κύριας μνήμης (αίτηση εγγραφής)
L: καθυστέρηση αστοχίας

Μεταβάσεις

1: αυτόματη μετάβαση (η κατάσταση A κρατάει πάντα έναν κύκλο)
2: `Finish_reset='0'`
3: `Finish_reset='1'`
4: `Reset='0' και ((Cache_rden='0' και Cache_wren='0') ή (Cache_rden='1' και Hit='1') ή (Cache_wren='1' και Hit='1'))`
5: `Reset='1'`
6: `Cache_rden='1'και Hit='0'`
7: `Cache_wren='1'και Hit='0'`
8: αυτόματη μετάβαση (η κατάσταση D κρατάει πάντα έναν κύκλο)
9: `Check_dirty='1'`
10: `Check_dirty='0'`
11: `Finish_read_cache='0'`
12: `Finish_read_cache='1'`
13: `Finish_read_main='0'`
14: `Finish_read_main='1'`
15: αυτόματη μετάβαση (η κατάσταση H κρατάει πάντα έναν κύκλο)
16: `Check_dirty='1'`
17: `Check_dirty='0'`
18: `Finish_read_cache='0'`
19: `Finish_read_cache='1'`
20: `Finish_read_main='0'`
21: `Finish_read_main='1'`
22: `Finish_miss='0'`
23: `Finish_miss='1'`

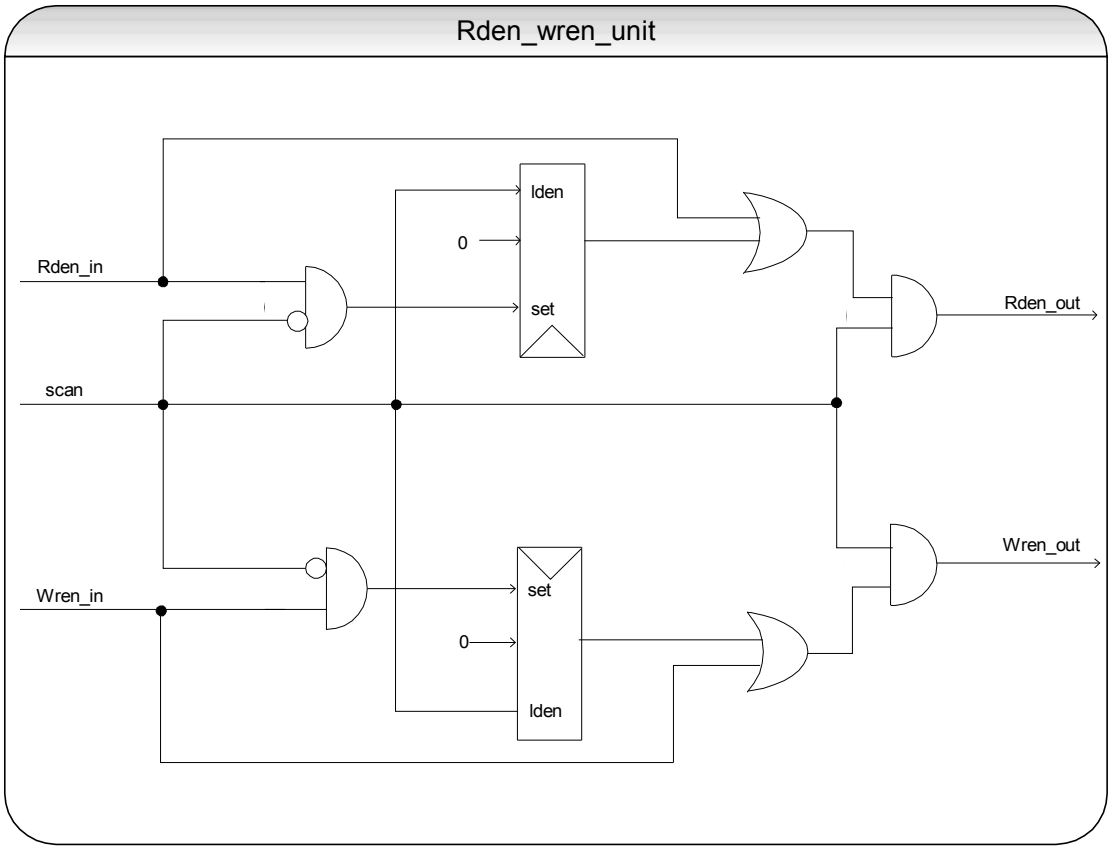
4.9 Δευτερεύουσες μονάδες – ολοκλήρωση του συστήματος

Έχοντας περιγράψει τον πυρήνα, τη διασύνδεσή του και την κεντρική μονάδα ελέγχου του συστήματος κρυφής μνήμης, θα προχωρήσω στην ανάλυση των υπόλοιπων μονάδων που χρειάστηκαν για να λειτουργήσει αυτό σωστά. Σύμφωνα με το αρχικό σχήμα θα περιγράψω τα μέρη που προηγούνται και που έπονται του πυρήνα, καθώς και κάποιες ειδικές μονάδες με συγκεκριμένο ρόλο στη λειτουργία του συστήματος. Ειδικά για τις μονάδες που αποτελούν μέρος του ελέγχου θα δώσω και τον κώδικα αφού τα σχήματα δεν αρκούν. Ο κώδικας όμως δείχνει πως λειτουργούν ακριβώς.

rden_wren_unit

Η μονάδα αυτή αποτελεί κομμάτι του μέρους που προηγείται του πυρήνα. Παράγει τα σήματα ανάγνωσης και εγγραφής που χρειάζεται το σύστημα εσωτερικά για να λειτουργήσει σωστά. Τι εννοούμε με αυτό;

Ας δούμε πως λειτουργεί ο έξω κόσμος ως προς το σύστημά μας (επεξεργαστής). Κάποια στιγμή στέλνει μια αίτηση προς την κρυφή μνήμη. Αν πρόκειται για ανάγνωση η αίτηση αυτή αποτελείται από ένα σήμα ελέγχου (το σήμα ανάγνωσης) και ένα σήμα δεδομένων (η διεύθυνση ανάγνωσης). Αν πρόκειται για εγγραφή τότε στα προηγούμενα προστίθεται και ένα ακόμα σήμα δεδομένων (τα δεδομένα προς εγγραφή). Εφόσον όλα συμβαίνουν σε ένα σύγχρονο περιβάλλον ένα ερώτημα είναι για πόσους κύκλους θα έχουν σταθερή τιμή τα σήματα αυτά. Για τα σήματα δεδομένων δεν χρειάζεται να είμαστε απόλυτοι. Μπορούν να μένουν σταθερά για ένα ή και για παραπάνω κύκλους εφόσον δεν υπάρχει νέα αίτηση. Το σήμα ελέγχου όμως είναι διαφορετικό. Η αποστολή ενός τέτοιου σήματος από τον έξω κόσμο προς την κρυφή μνήμη ισοδυναμεί με μια πράξη στο σύστημα. Αυτήν της ανάγνωσης ή της εγγραφής. Αν το σήμα παραμείνει ενεργό για δύο συνεχόμενους κύκλους θα σημαίνει ότι ο έξω κόσμος αιτεί δύο τέτοιες πράξεις. Άρα είναι λάθος να επιτρέψουμε για την ίδια αίτηση το σήμα ελέγχου να παραμείνει ενεργό για παραπάνω από ένα κύκλο.
Πότε ο έξω κόσμος μπορεί να στείλει μια νέα αίτηση; Αμέσως μόλις εξυπηρετηθεί η προηγούμενή του. Αυτό του γίνεται αντιληπτό μέσω ενός σήματος επιβεβαίωσης που στέλνει το σύστημά μας. Στην περίπτωση της ανάγνωσης το σήμα αυτό έρχεται στον ίδιο κύκλο που έρχονται και τα δεδομένα που αναγνώσθηκαν.
Τώρα ας θυμηθούμε πως λειτουργεί το σύστημά μας. Όταν ικανοποιείται μια αίτηση ανάγνωσης και είχαμε αστοχία, το σύστημα βγάζει προς τον έξω κόσμο την απάντηση (και μαζί της το σήμα επιβεβαίωσης) πριν ακόμα ολοκληρωθούν όλες οι διαδικασίες που χρειάζονται. Όταν δηλαδή ο έξω κόσμος πάρει την απάντησή του, το σύστημα δεν έχει έρθει ακόμα στην κατάσταση ηρεμίας, ώστε να είναι έτοιμο να εξυπηρετήσει μια νέα αίτηση. Αυτό είναι κόστος που πληρώνουμε προκειμένου να είμαστε γρήγοροι στην εξυπηρέτηση μιας αίτησης. Όταν ο έξω κόσμος έχει μόνο μια αίτηση και μετά μένει αδρανής, όλα λειτουργούν καλά αφού και η απάντηση του έρχεται γρήγορα και δεν πειράζει που το σύστημα ακόμα δουλεύει.
Αν συνυπολογίσουμε όμως πως λειτουργεί ο έξω κόσμος και το σύστημα κρυφής μνήμης σε ένα σενάριο όπου ο έξω κόσμος έχει πάντα μια αίτηση να κάνει προς το σύστημα καταλήγουμε στο εξής φαινόμενο: Ο έξω κόσμος παίρνει την απάντηση και την επιβεβαίωση μιας αίτησης ανάγνωσης και στον επόμενο κύκλο κάνει μια νέα αίτηση (είτε ανάγνωσης είτε εγγραφής). Το σήμα ελέγχου που στέλνει διαρκεί μόνο ένα κύκλο όπως επιβάλλεται. Κατά τον κύκλο όμως αυτό το σύστημα δεν έχει επανέλθει στη κατάσταση ηρεμίας, τη μόνη κατάσταση που ελέγχει για νέες αιτήσεις. Όταν μεταβεί στην κατάσταση ηρεμίας δεν θα δει ποτέ την αίτηση και απλώς θα περιμένει. Από την πλευρά του ο έξω κόσμος θα περιμένει άδικα να του έρθει μια επιβεβαίωση, έχοντας στην ουρά να περιμένουν άλλες αιτήσεις. Ανάλογες καταστάσεις παρουσιάζονται όταν το σύστημα δίνει επιβεβαίωση μιας εγγραφής πριν όμως η διαδικασία εγγραφής ολοκληρωθεί πραγματικά.
Αυτό το πρόβλημα έρχεται να λύσει η μονάδα `rden_wren_unit` που δίνεται στο παρακάτω σχήμα.



Το σήμα ανάγνωσης (ή εγγραφής αντίστοιχα) είναι η έξοδος του παραπάνω κυκλώματος. Εφόσον βρισκόμαστε σε κατάσταση ηρεμίας (χρήση της δεύτρης πύλης και), στην οποία μπορεί να γίνει αντιληπτή μια αίτηση, δίνουμε μια τέοια αίτηση στο σύστημα σε δύο περιπτώσεις. Είτε γιατί έχεται από τον έξω κόσμο, είται γιατί υπάρχει μια αίτηση σε αναμονή. Αυτό γίνεται μέσω της πύλης ή.
Το σήμα που γίνεται 1 όταν υπάρχει ανάγνωση σε αναμονή (`waiting_rden` στον κώδικα) παράγεται από έναν καταχωρητή, διαφορετικό από τους υπόλοιπους του συστήματος. Ο καταχωρητής αυτός δεν έχει reset αλλά set. Δίνεται παρακάτω

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY set_reg_1_bit IS
	PORT
	(
		X	:IN STD_LOGIC;
		LdEn	:IN STD_LOGIC;
		Set	:IN STD_LOGIC;
		Clk	:IN STD_LOGIC;
		Y	:OUT STD_LOGIC
	);
END set_reg_1_bit;

ARCHITECTURE behavioral OF set_reg_1_bit IS
BEGIN
	PROCESS(Clk,Set,LdEn,X)
	BEGIN
		IF (Set='1') THEN
			Y<='1';
		ELSIF (Clk'event AND Clk='0') THEN
```

```
                    IF (LdEn='1') THEN
                            Y<=X;
                    END IF;
                END IF;
        END PROCESS;
END behavioral;
```

Ο καταχωρητής αυτός όταν δουλεύει κανονικά (lden=1) παίρνει σαν είσοδο την σταθερά 0. Κανονικά λειτουργεί όταν το σύστημα βρίσκεται στην κατάσταση ηρεμίας στην οποία δεν χάνεται καμία αίτηση και έτσι δεν έχει νόημα να βρεθεί μια αίτηση σε αναμονή. Σε όλες τις άλλες καταστάσεις όμως ο καταχωρητής «παγώνει». Το μόνο που μπορεί να του αλλάξει την τιμή σε 1 είναι το σήμα set το οποίο παράγεται από την έκφραση:set = `phantom_rden<=Rden_in AND (NOT Scan);`
Δηλαδή αίτηση διαβάσματος σε κατάσταση μη ηρεμίας.
Έτσι ο καταχωρητής μόλις υπάρξει «χαμένη» αίτηση κρατάει την τιμή 1 μέχρι το σύστημα να μεταβεί σε κατάσταση ηρεμίας οπότε και ειδοποιείται μία φορά για την «χαμένη» αίτηση.

mux_vir_addr

Ο πολυπλέκτης αυτός δεν έχει δειχτεί σε προηγούμενα σχήματα γιατί δεν έχει μεγάλη σημασία για το datapath. Επιλέγει ανάμεσα στην εξωτερική διεύθυνση και την καθυστερημένη – παγωμένη ανάλογα με τη κατάσταση του συστήματος (ηρεμία ή μη). Την έξοδου αυτού του πολυπλέκτη παίρνουν στη πραγματικότητα όλες οι μονάδες. Δρα συμπληρωματικά με τη προηγούμενη μονάδα επιτρέποντας στο σύστημα την ανεξαρτητοποίησή του από τον έξω κόσμο. Αυτό γιατί η χρήση του μας επιτρέπει να αδιαφορούμε για το αν ο έξω κόσμος κρατάει σταθερή την διεύθυνση μιας αίτησης μέχρι αυτή να ικανοποιηθεί. Δίνω απλώς τον κώδικα αντιστοίχησης.

```
mux_vir_addr:mux_2to1_n
                GENERIC MAP
                (
                        n=>64
                )
                PORT MAP
                (
                        A=>late_address,
                        B=>Address,
                        Mux_sel=>scan_signal,
                        Out1=>virtual_address
                );
```

address_unit

Η μονάδα αυτή, που ήδη έχω αναφέρει δημιουργεί τις διευθύνσεις στις οποίες βρίσκεται το μπλοκ που θα μεταφερθεί από την κύρια μνήμη στην κρυφή μνήμη. Στην πράξη παίρνει την διεύθυνση που ήρθε από την αίτηση που ήταν άστοχη και της αφαιρεί το πεδίο της απόκλισης του μπλοκ. Αντ' αυτού, με χρήση ενός μετρητή παράγει τιμές που ξεκινούν από την πρώτη λέξη του μπλοκ έως την τελευταία. Π.χ αν το πεδίο απόκλισης μπλοκ είναι δύο bits τότε παράγονται οι διευθύνσεις που τελειώνουν σε 00, 01, 10 και 11. Κάθε τέτοια διεύθυνση εξέρχεται ανά κύκλο ενώ η διαδικασία ξεκινάει όταν τα σήματα enable και reset πάρουν τις κατάλληλες τιμές. Όταν η διαδικασία τελειώσει το σήμα Finish παίρνει τη τιμή 1 και ειδοποιεί τον έλεγχο του συστήματος. Το σήμα αυτό παράγεται από μια πύλη and. Η μονάδα αυτή μοιάζει

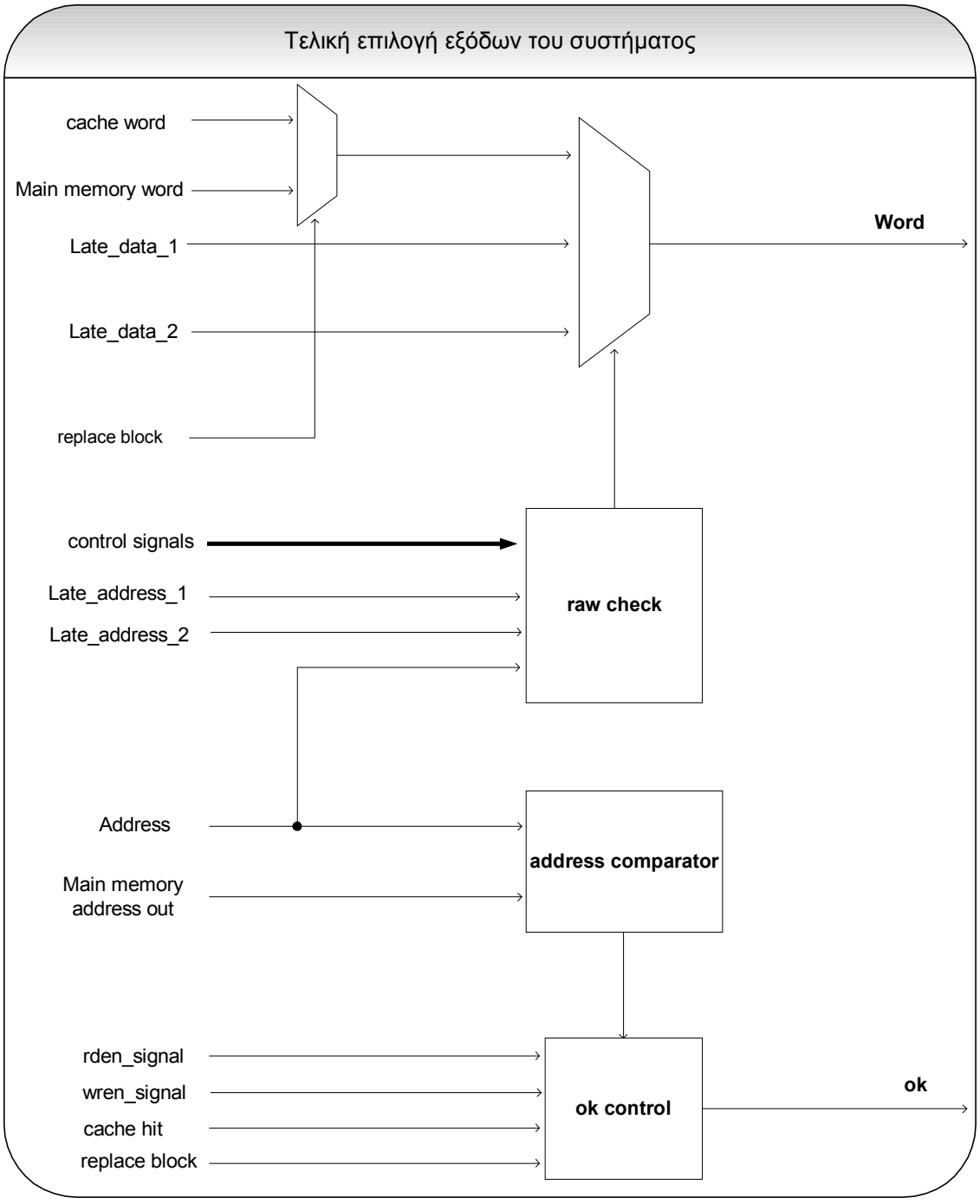
πολύ με τη μονάδα που παράγει τις διευθύνσεις αρχικοποίησης. Δίνεται ο κώδικας αντιστοίχησης.

```
address_unit_1:address_unit
	GENERIC MAP
	(
		block_offset=>block_offset,
		memory_address_width=>main_mem_addr_width
	)
	PORT MAP
	(
		Enable=>replace_main2cache,
		Reset=>not_replace_main2cache,
		Clock=>Clock,
		Address_in=>virtual_address(main_mem_addr_width+1 DOWNTO
block_offset+2),
		Finish=>finish_read_main_flag,
		Address_out=>main_rdaddress
	);
```

Τελική επιλογή εξόδων του συστήματος

Το κομμάτι που έπεται του πυρήνα είναι αυτό που δίνει στον επεξεργαστή τα σήματα – απαντήσεις μιας αίτησης. Οι τελικές έξοδοι είναι η λέξη που περιμένη ο έξω κόσμος για μια ανάγνωση και το σήμα επιβεβαίωσης. Το τελευταίο σε περίπτωση ανάγνωσης δείχνει ότι η λέξη που εξέρχεται εκείνο τον κύκλο είναι η σωστή. Επίσης το σήμα αυτό έχει σαν ρόλο να ειδοποιεί το σύστημα ότι έιναι έοιμο για να δεχτεί επόμενη αίτηση. Αυτό όπως έχω αναφέρει δεν σημαίνει ότι είναι ακριβές. Για την ακρίβεια στην περίπτωση αίτησης εγγραφής η επιβεβαίωσει σημαίνει ότι το σύστημα θα διευθετήσει την εγγραφεί και ο έξω κόσμος είναι ελεύθερος να αλλάξει τις εισόδους του, προς το σύστημα. Ακόμα και αν κάνει μια νέα αίτηση, αυτή θα χρειαστεί να περιμένει μέχρι να πάρει σειρά. Με αυτό το τρόπο επιτυγχάνουμε να έχουμε μέγιστη δυνατή απάντηση προς τον έξω κόσμο, αποκρύπτοντάς του την πολυπλοκότητα των λειτουργιών.

Ακολουθεί το σχήμα που δείχνει το μέρος της σχεδίασης που δίνει τις εξόδους και η ανάλυση των επί μέρους μονάδων.



address_comparator

Ένας απλός συγκριτής διευθύνσεων όπως υποδηλώνει και το όνομά του. Χρησιμοποιείται για να ελέγξει ποια λέξη από το μπλοκ, που μεταφέρεται από την κύρια στην κρυφή μνήμη, είναι αυτή που ζήτησε ο έξω κόσμος. Όταν η σύγκριση είναι επιτυχής παράγει κατάλληλο σήμα ελέγχου. Το σήμα αυτό είναι σημαντικό γιατί αυτό ειδοποιεί ότι τώρα περνάνε από το δίαυλο και τα ζητούμενα δεδομένα, ώστε να ειδοποιηθεί ο έξω κόσμος να τα πάρει. Δίνεται ο κώδικας αντιστοίχησης.

```
address_comparator_1:address_comparator
	GENERIC MAP
	(
		n=>block_offset
	)
	PORT MAP
```

```
        (
                Address_A=>virtual_address(block_offset+1 DOWNTO 2),
                        Address_B=>delayed_main_rdaddress(block_offset-1
       DOWNTO 0),
                Hit=>hit_compare_signal
        );
```

raw_check (read after wtrite check)
Δεδομένου ότι η εγγραφή (επιτυχής) γίνεται στην κρυφή μνήμη έναν ή περισσότερους κύκλους μετά την αίτηση του έξω κόσμου, υπάρχει μία εξάρτηση που δεν μπορεί να εξυπηρετήσει το σύστημα: Η ανάγνωση μιας λέξης που ακόμα δεν γράφτηκε. Το σύστημα είχε δώσει αμέσως την επιβεβαίωση της εγγραφής πριν αυτή γίνει πραγματικά για να μπορεί να ικανοποιήσει το ρυθμό μία αίτηση ανά κύκλο σε επιτυχημένες αιτήσεις. Σε αυτό το σενάριο όμως που ζητείται η ίδια η λέξη από τον έξω κόσμο. Υπάρχουν δύο τύποι της συγκεκριμένης εξάρτησης. Ο πρώτος είναι όταν η ζητούμενη λέξη βρίσκεται στο πρώτο write buffer. Αντίστοιχα ο δεύτερος τύπος είναι όταν έουμε ανάγνωση μιας λέξης που βρίσκεται στο δεύτερο write buffer σε αναμονή. Το πρόβλημα αυτό λύνεται με την συγκεκριμένη μονάδα. Έτσι μόλις υπάρξει raw εξάρτηση είτε τύπου 1 είται τύπου 2, δίνεται στον έξω κόσμο η λέξη που υπήρχε στον αντίστοιχο καταχωρητή και όχι κάποια λέξη που προέρχεται από την κρυφή μνήμη. Δίνεται ο κώδικας υλοποίησής του.

```
ENTITY raw_check IS
       GENERIC
       (
              n      :INTEGER :=4
       );
       PORT
       (
              Address_A   :IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
              Address_B   :IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
              Address_C   :IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
              Late_wren     :IN STD_LOGIC;
              Rden          :IN STD_LOGIC;
              Frozen_write  :IN STD_LOGIC;
              Hit           :OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
       );
END raw_check;

ARCHITECTURE behavioral OF raw_check IS
BEGIN

PROCESS(Address_A,Address_B,Address_C,Late_wren,Rden,Frozen_write)
       BEGIN
              IF ((Address_A=Address_B) AND (Late_wren='1') AND
(Rden='1')) THEN
                     Hit<="01";
              ELSIF ((Address_A=Address_C) AND (Frozen_write='1'))
THEN
                     Hit<="10";
              ELSE
                     Hit<="00";
              END IF;
       END PROCESS;
END behavioral;
```

word_mux1, word_mux2(τα ονόματα αυτά εμφανίζονται στον κώδιμα)
Τελικά προς τον έξω κόσμο μπορούν να βγουν δεδομένα από τέσσερις διαφορετικές πηγές κατά την ικανοποίηση μιας αίτησης ανάγνωσης, η επιλογή των οποίων γίνεται μέσα από δύο πολυπλέκτες.

1. Από την κρυφή μνήμη, αν ήταν μια εύστοχη αίτηση.
2. Από την κύρια μνήμη, αν ήταν μια άστοχη αίτηση.
3. Από τον write buffer 1, αν ήταν μια αίτηση ανάγνωση λέξης που μόλις γράφτηκε (raw εξάρτηση τύπου 1).
4. Από τον write buffer 2, αν ήταν μια αίτηση ανάγνωση λέξης που περιμένει να γραφτεί (raw εξάρτηση τύπου 2).

ok_control
Έχουμε πει ότι για κάθε αίτηση στέλνεται η επιβεβαίωση της προς τον έξω κόσμο. Αν πρόκειται για εγγραφή η επιβεβαίωση σημαίνει ότι το σύστημα έχει αναλάβει τη διαδικασία (χωρίς να έχει ολοκληρωθεί) και επομένως ο έξω κόσμος είναι ελεύθερος να κάνει μια νέα αίτηση. Αν επιβεβαίωση είναι για ανάγνωση εκτός από τα προηγούμενα σημαίνει ότι σε αυτόν το κύκλο εξέρχονται τα δεδομένα που αναγνώστηκαν και ο έξω κόσμος πρέπει να τα πάρει. Η μονάδα αυτή είναι υπεύθυνη για την παραγωγή του σήματος επιβεβαίωσης. Δίνεται ο κώδικας αντιστοίχησης και υλοποίησης.

```
ok_control_1:ok_control
	PORT MAP
	(
		Rden=>rden_signal,
		Wren=>wren_signal,
		Hit=>temp_final_hit,
		Read_miss=>delayed_plus_replace_block_signal,
		Hit_compare=>hit_compare_signal,
		Ok=>temp_ack_hit
	);
PROCESS(Rden,Wren,Hit,Read_miss,Hit_compare)
	BEGIN
		IF ((Rden='1' AND Hit='1') OR Wren='1') THEN
			Ok<='1';
		ELSIF (Read_miss='1' AND Hit_compare='1') THEN
			Ok<='1';
		ELSE
			Ok<='0';
		END IF;
	END PROCESS;
```

Όλες οι μονάδες που αναφέρθηκαν μέχρι τώρα υπάρχουν και στα τρία συστήματα. Μόνο τα σήματα ελέγχου ίσως να έχουν διαφορετικό όνομα.

<u>Διαφορές συστημάτων</u>
Τώρα θα αναφέρω τις μονάδες που υπάρχουν μεν και στα τρία συστήματα αλλά ανάλογα με την πολυπλοκότητα του καθενός έχουν κάποια πρόσθετα σήματα.

dm_write_enable και sa_write_enable (dm= direct mapped, sa= set associative)
Πρόκειται για την σημαντικότερη μονάδα ελέγχου του συστήματος μετά την μηχανή πεπερασμένων καταστάσεων. Είναι υπεύθυνη για τρεις διαφορετικές λειτουργίες, που αναλύονται παρακάτω.
Το σήμα που ενεργοποιεί την εγγραφή σε μια μνήμη (write enable signal), είναι ένα σήμα που θέλει ιδιαίτερη προσοχή. Ενώ δεν είναι κακό να διαβάζουμε από μια μνήμη

ακόμα και άχρηστα δεδομένα πρέπει να είμαστε προσεκτικοί τη στιγμή που γράφουμε σε αυτή. Ας ξεκινήσουμε με το πιο απλό σενάριο. Έχουμε το πρώτο σύστημα κρυφής μνήμης και παίρνουμε την περίπτωση η κρυφής μνήμη να είναι άμεσα αντιστοιχισμένη. Δύο είναι τα σήματα εγγραφής που πρέπει να ελέγξουμε. Το σήμα που γράφει την μνήμη δεδομένων και το σήμα που γράφει την μνήμη ετικετών. Όταν έχουμε ευστοχία σε μία εγγραφή γράφεται μόνο η μνήμη δεδομένων, ενώ όταν έχουμε αντικατάσταση μπλοκ γράφεται και η μνήμη ετικετών.
Το δεύτερο process είναι υπεύθυνο για την καταγραφή μιας εγγραφής που ήταν να γίνει αλλά αναβλήθηκε λόγω μιας αίτησης ανάγνωσης. Μόλις καταγραφεί κάτι τέτοιο ο write buffer 2 που ελέγχεται από αυτή τη μονάδα παγώνει. Παράλληλα στέλνεται ένα σήμα εγγραφής το οποίο θα τελικά θα μπει στον πυρήνα μόλις σταματήσουν οι αναγνώσεις. Τότε είναι που επανέρχεται το σύστημα στην κανονική του λειτουργία.
Η μονάδα αυτή παράγει και ένα ακόμα σήμα. Αυτό που ειδοποιεί τον κεντρικό έλεγχο πότε τελείωσε η αντικατάσταση του μπλοκ, ώστε να βγει από την κατάσταση καθυστέρησης αστοχίας και να μεταβεί στην κατάσταση ηρεμίας. Αυτό γίνεται στο τρίτο process. Ακολουθεί ο κώδικας της μονάδας. Τα γράμματα nwa_dm σημαίνουν **n**o **w**rite **a**llocate **d**irect **m**apped.

```
ENTITY nwa_dm_write_enable IS
	PORT
	(
		Replace		:IN STD_LOGIC;
		Read			:IN STD_LOGIC;
		Write_hit		:IN STD_LOGIC;
		Write_hit_late:IN STD_LOGIC;
		Miss			:IN STD_LOGIC;
		Reset_flag		:IN STD_LOGIC;
		Clock			:IN STD_LOGIC;
		Load			:OUT STD_LOGIC;
		Wren_hit		:OUT STD_LOGIC;
		Wren_data		:OUT STD_LOGIC;
		Wren_tag		:OUT STD_LOGIC;
		Finish		:OUT STD_LOGIC
	);
END nwa_dm_write_enable;

ARCHITECTURE behavioral OF nwa_dm_write_enable IS
BEGIN
	PROCESS(Clock,Replace,Reset_flag,Write_hit)
	BEGIN
		IF (Clock'EVENT AND Clock='0') THEN
			IF (Replace='1' OR Reset_flag='1') THEN
				Wren_data<='1';
				Wren_tag<='1';
			ELSIF (Write_hit='1') THEN
				Wren_data<='1';
				Wren_tag<='0';
			ELSE
				Wren_data<='0';
				Wren_tag<='0';
			END IF;
		END IF;
	END PROCESS;

	PROCESS(Clock,Read,Write_hit_late)
		VARIABLE freeze_write:STD_LOGIC :='0';
	BEGIN
```

```
		IF (Clock'EVENT AND Clock='0') THEN
			IF (Write_hit_late='1' AND Read='1') THEN
				freeze_write:='1';
				Load<='0';
				Wren_hit<='1';
			ELSIF (freeze_write='1') THEN
				IF (Read='0') THEN
					freeze_write:='0';
					Wren_hit<='0';
					Load<='1';
				ELSE
					Load<='0';
					Wren_hit<='1';
				END IF;
			ELSE
				Load<='1';
				Wren_hit<='0';
			END IF;
		END IF;
	END PROCESS;

	PROCESS(Miss,Replace)
	BEGIN
		IF (Miss='1') THEN
			Finish<='0';
		ELSIF (Replace'EVENT AND Replace='0') THEN
			Finish<='1';
		END IF;
	END PROCESS;
END behavioral;
```

Όταν όμως έχουμε συνολοσυσχετιστική κρυφή μνήμη το πρόβλημα γίνεται πιο σύνθετο, γιατί τώρα δεν έχουμε μόνο ένα σήμα ενεργοποίησης εγγραφής, αλλά περισσότερα (ένα για κάθε σύνολο). Κατά την αρχικοποίηση γράφουμε όλα τα σύνολα, αλλά σε περίπτωση αντικατάστασης πρέπει μόνο ένα σήμα να ενεργοποιηθεί. Εδώ είναι που παρουσιάζεται η πολιτική αντικατάστασης που χρησιμοποίησα. Η πιο εύκολη πολιτική είναι η τυχαία επιλογή αλλά επειδή δεν υπάρχει τίποτα εντελώς τυχαίο στον προγραμματισμό μου ζητήθηκε να διαλέγω κάθε φορά το επόμενο σύνολο από αυτό που επέλεξα στην προηγούμενη αντικατάσταση. Π.χ αν έχουμε τρία σύνολα α, β και γ στις δέκα πρώτες αστοχίες θα διαλέγονται τα σύνολα με αυτή τη σειρά :α, β, γ, α, β, γ, α, β, γ, α. Αυτό από πλευράς σημάτων με κατεύθυνε στο συμπέρασμα ότι πρέπει με κάποιο τρόπο να παράγω ένα σήμα που να αλλάζει κάπως έτσι:001, 010, 100, 001, 010, 100... για τρία σύνολα. Κάθε bit είναι το σήμα ενεργοποίησης εγγραφής για το αντίστοιχο σύνολο. Επομένως θέλουμε σε μια ακολουθία από μηδενικά να ολισθαίνουμε έναν άσσο προς τα αριστερά μέχρι να φθάσει στο τέρμα και ύστερα να τον ξαναβάλουμε τέρμα δεξιά κ.ο.κ. Αυτό το σήμα θα παραγόταν εύκολα αν είχαμε σταθερό μέγεθος σήματος, δηλαδή σταθερή συνολοσυσχετιστικότητα. Λόγω όμως της παραμετρικής φύσης του προβλήματος αναγκαζόμαστε να δουλέψουμε μεταβλητές. Έτσι για να παράγω το επιθυμητό αποτέλεσμα κατέληξα στο παρακάτω κώδικα.

```
IF (Replace='1') THEN
	IF (same_replace='0') THEN
		IF (counter=0) THEN
			counter:=counter+1;
		ELSIF (counter(n-1)='1') THEN
			counter:=(OTHERS=>'0');
				counter:=counter+1;
			ELSE
```

```
                        counter:=counter(n-2 DOWNTO 0) & '0';
                    END IF;
                    same_replace:='1';
                END IF;
                Wren_data<=counter;
                Wren_tag<=counter;
```

Ο παραπάνω κώδικας κάνει ακριβώς την ολίσθηση του άσσου που χρειάζεται σε ένα σήμα μεγέθους n. Η μεταβλητή same_replace μας αποτρέπει να αλλάξουμε σύνολο εγγραφής κατά την ίδια αντικατάσταση μπλοκ. Ο ολοκληρωμένος κώδικας της μονάδος όταν έχουμε συνολοσυχετιστική μνήμη ακολουθεί.

```
ENTITY nwa_sa_write_enable IS
        GENERIC
        (
                n       :INTEGER :=3
        );
        PORT
        (
                Temp_hit              :IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
                Temp_hit_late  :IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
                Replace        :IN STD_LOGIC;
                Read                  :IN STD_LOGIC;
                Write_hit             :IN STD_LOGIC;
                Write_hit_late:IN STD_LOGIC;
                Miss                  :IN STD_LOGIC;
                Reset_flag            :IN STD_LOGIC;
                Clock                 :IN STD_LOGIC;
                Load                  :OUT STD_LOGIC;
                Wren_hit              :OUT STD_LOGIC_VECTOR(n-1 DOWNTO
0);
                Wren_data             :OUT STD_LOGIC_VECTOR(n-1 DOWNTO
0);
                Wren_tag              :OUT STD_LOGIC_VECTOR(n-1 DOWNTO
0);
                Finish         :OUT STD_LOGIC
        );
END nwa_sa_write_enable;

ARCHITECTURE behavioral OF nwa_sa_write_enable IS
BEGIN
        PROCESS(Clock,Temp_hit,Replace,Reset_flag,Write_hit)
                VARIABLE counter      :STD_LOGIC_VECTOR(n-1 DOWNTO 0)
:=(OTHERS=>'0');
                VARIABLE same_replace:STD_LOGIC :='0';
        BEGIN
                IF (Clock'EVENT AND Clock='0') THEN
                        IF (Replace='1') THEN
                                IF (same_replace='0') THEN
                                        IF (counter=0) THEN
                                                counter:=counter+1;
                                        ELSIF (counter(n-1)='1') THEN
                                                counter:=(OTHERS=>'0');
                                                counter:=counter+1;
                                        ELSE
                                                counter:=counter(n-2 DOWNTO
0) & '0';
                                        END IF;
                                        same_replace:='1';
                                END IF;
                                Wren_data<=counter;
```

```
                        Wren_tag<=counter;
                    ELSE
                        same_replace:='0';
                        IF (Reset_flag='1') THEN
                            Wren_data<=(OTHERS=>'1');
                            Wren_tag<=(OTHERS=>'1');
                        ELSIF (Write_hit='1') THEN
                            Wren_data<=Temp_hit;
                            Wren_tag<=(OTHERS=>'0');
                        ELSE
                            Wren_data<=(OTHERS=>'0');
                            Wren_tag<=(OTHERS=>'0');
                        END IF;
                    END IF;
            END IF;
        END PROCESS;

        PROCESS(Clock,Temp_hit_late,Read,Write_hit_late)
            VARIABLE write_set   :STD_LOGIC_VECTOR(n-1 DOWNTO 0)
:=(OTHERS=>'0');
            VARIABLE freeze_write:STD_LOGIC :='0';
        BEGIN
            IF (Clock'EVENT AND Clock='0') THEN
                IF (Write_hit_late='1' AND Read='1') THEN
                    freeze_write:='1';
                    write_set:=Temp_hit_late;
                    Load<='0';
                    Wren_hit<=write_set;
                ELSIF (freeze_write='1') THEN
                    IF (Read='0') THEN
                        freeze_write:='0';
                        Wren_hit<=(OTHERS=>'0');
                        Load<='1';
                        write_set:=(OTHERS=>'0');
                    ELSE
                        Load<='0';
                        Wren_hit<=write_set;
                    END IF;
                ELSE
                    Load<='1';
                    Wren_hit<=(OTHERS=>'0');
                END IF;
            END IF;
        END PROCESS;

        PROCESS(Miss,Replace)
        BEGIN
            IF (Miss='1') THEN
                Finish<='0';
            ELSIF (Replace'EVENT AND Replace='0') THEN
                Finish<='1';
            END IF;
        END PROCESS;
END behavioral;
```

Παρατηρούμε ότι και στο δεύτερο process μπήκε ένα επιπλέον σήμα ώστε να θυμάται το σύστημα σε ποιο σετ πρέπει να γίνει η εγγραφή που είναι σε αναμονή. Όπως υποδηλώνεται και από το όνομά του το σήμα αυτό έχει υποστεί καθυστέριση ενός κύκλου. Αυτό γίνεται μέσω ενός καταχωρητή ο οποίος δεν είχε φανεί μέχρι τώρα για να μην μπλεχτεί ο αναγνώστης. Στον κώδιακα εμφανίζεται με το όνομα temp_hit_reg.

Τώρα θα πάμε στο δεύτερο σύστημα και θα δούμε τις διαφορές που έχουν οι δύο μονάδες που αναλύσαμε. Το μόνο διαφορετικό που έχουν είναι ότι παράγουν και το σήμα που ελέγχει την εγγραφή της κύριας μνήμης κατά την ευστοχία μιας αίτηση ς εγγραφής, κάτι που δεν χρειαζόταν στο πρώτο σύστημα. Ο κώδικας είναι λοιπόν σχεδόν ο ίδιος.
Το τρίτο σύστημα είναι αυτό με την ετεροχρονισμένη εγγραφή. Κατά την ανάλυση του ελέγχου είχα αναφέρει ότι κάποια στιγμή ελέγχουμε ποιο μπλοκ πρόκειται να αντικαταστήσουμε για να δούμε αν είναι βρώμικο. Επομένως πρέπει να γίνει η διαδικασία υπολογισμού του συνόλου που πρόκειται να γράψουμε χωρίς όμως να το γράψουμε εκείνη τη στιγμή. Για το λόγο αυτό έφτιαξα δύο σήματα. Ένα που δηλώνει ποιο σύνολο **θα γραφτεί** και ένα όπως και πριν που ελέγχει την εγγραφή των συνόλων. Δίνεται ο κώδικας και των δύο μονάδων. Δείχνεται μόνο το πρώτο process στο οποίο υπάρχει και διαφορά.

```
ENTITY dm_write_enable IS
	PORT
	(
		Replace		:IN STD_LOGIC;
		Read			:IN STD_LOGIC;
		Write_hit		:IN STD_LOGIC;
		Write_hit_late	:IN STD_LOGIC;
		Miss			:IN STD_LOGIC;
		Reset_flag		:IN STD_LOGIC;
		Clock			:IN STD_LOGIC;
		Load			:OUT STD_LOGIC;
		Wren_hit		:OUT STD_LOGIC;
		T_wren		:OUT STD_LOGIC;
		F_wren		:OUT STD_LOGIC;
		Tag_wren		:OUT STD_LOGIC;
		Finish		:OUT STD_LOGIC
	);
END dm_write_enable;

ARCHITECTURE behavioral OF dm_write_enable IS
BEGIN
	PROCESS(Clock,Replace,Reset_flag,Write_hit)
	BEGIN
		IF (Clock'EVENT AND Clock='0') THEN
			IF (Replace='1' OR Reset_flag='1') THEN
				T_wren<='0';
				F_wren<='1';
				Tag_wren<='1';
			ELSIF (Write_hit='1') THEN
				T_wren<='1';
				F_wren<='1';
				Tag_wren<='0';
			ELSE
				T_wren<='1';
				F_wren<='0';
				Tag_wren<='0';
			END IF;
		END IF;
	END PROCESS;

ENTITY sa_write_enable IS
	GENERIC
	(
		n	:INTEGER :=3
```

```
	);
	PORT
	(
		Temp_hit		:IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
		Temp_hit_late	:IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
		Will_replace	:IN STD_LOGIC;
		Replace		:IN STD_LOGIC;
		Read			:IN STD_LOGIC;
		Write_hit		:IN STD_LOGIC;
		Write_hit_late:IN STD_LOGIC;
		Miss			:IN STD_LOGIC;
		Reset_flag		:IN STD_LOGIC;
		Clock			:IN STD_LOGIC;
		Load			:OUT STD_LOGIC;
		Wren_hit		:OUT STD_LOGIC_VECTOR(n-1 DOWNTO
0);
		T_wren_s		:OUT STD_LOGIC_VECTOR(n-1 DOWNTO
0);
		F_wren_s		:OUT STD_LOGIC_VECTOR(n-1 DOWNTO
0);
		Tag_wren_s		:OUT STD_LOGIC_VECTOR(n-1 DOWNTO
0);
		Finish		:OUT STD_LOGIC
	);
END sa_write_enable;

ARCHITECTURE behavioral OF sa_write_enable IS
BEGIN

PROCESS(Clock,Temp_hit,Will_replace,Replace,Reset_flag,Write_hit)
		VARIABLE counter	:STD_LOGIC_VECTOR(n-1 DOWNTO 0)
:=(OTHERS=>'0');
	BEGIN
		IF (Clock'EVENT AND Clock='0') THEN
			IF (Will_replace='1') THEN
				IF (counter=0) THEN
					counter:=counter+1;
				ELSIF (counter(n-1)='1') THEN
					counter:=(OTHERS=>'0');
					counter:=counter+1;
				ELSE
					counter:=counter(n-2 DOWNTO 0) &
'0';
				END IF;
				T_wren_s<=counter;
				F_wren_s<=(OTHERS=>'0');
				Tag_wren_s<=(OTHERS=>'0');
			ELSIF (Replace='1') THEN
				T_wren_s<=(OTHERS=>'0');
				F_wren_s<=counter;
				Tag_wren_s<=counter;
			ELSE
				IF (Reset_flag='1') THEN
					T_wren_s<=(OTHERS=>'0');
					F_wren_s<=(OTHERS=>'1');
					Tag_wren_s<=(OTHERS=>'1');
				ELSIF (Write_hit='1') THEN
					T_wren_s<=temp_hit;
					F_wren_s<=temp_hit;
					Tag_wren_s<=(OTHERS=>'0');
				ELSE
```

```
                                    T_wren_s<=counter;
                                    F_wren_s<=(OTHERS=>'0');
                                    Tag_wren_s<=(OTHERS=>'0');
                                END IF;
                            END IF;
                    END IF;
            END PROCESS;
```

Το σήμα T_wren_s έχει το νόημα του temp wrens δηλαδή του σήματος που δείχνει ποιο σύνολο πρόκειται να αντικαταστήσουμε ενώ το F_wren_s είναι το final wrens δηλαδή το τελικό σήμα που πάει στην κρυφή μνήμη και ελέγχει την εγγραφή της. Ανάλογα το σήμα will_replace ειδοποιεί ότι το σύστημα θα προβεί σε αντικατάσταση ώστε να υπολογιστεί το σύνολο που θα αντικατασταθεί. Στη μονάδα της άμεσα αντιστοιχισμένης μονάδας ασφαλώς δεν χρειάζεται αφού έχουμε μόνο ένα σύνολο.

delay_unit

Είναι η μονάδα που υλοποιεί την καθυστέρηση που υπόκεινται τα δεδομένα που εξέρχονται από την κύρια μνήμη. Στην πράξη είναι μια διάταξη καταχωρητών σε σειρά. Εγώ επέλεξα να είναι δέκα, πολύ εύκολα όμως αυτό αλλάζει. Το σύστημα όμως είναι έτσι δομημένο που αυτήν την καθυστέρηση πρέπει να υπολογίζουν και κάποια σήματα ελέγχου.

Θα δώσω ένα παράδειγμα: Κατά την αντικατάσταση ενός μπλοκ δίνονται στην κύρια μνήμη ορισμένες διευθύνσεις και παίρνονται τα αντίστοιχα δεδομένα. Οι διευθύνσεις αυτές όμως είναι χρήσιμες και μετά το διάβασμα για δύο λόγους α) θα δοθούν αργότερα και στην κρυφή μνήμη ως διευθύνσεις εγγραφής του μπλοκ που ήρθε από την κύρια μνήμη, β) θα συγκριθούν ως προς τη διεύθυνση αίτησης για να επιλεχθεί η λέξη που θα δοθεί προς τον έξω κόσμο (περίπτωση άστοχης ανάγνωσης). Και στις δύο περιπτώσεις οι διευθύνσεις αυτές πρέπει να είναι συγχρονισμένες με τα δεδομένα που ήρθαν από την κύρια μνήμη. Για να γίνει αυτό περνάνε από αντίστοιχους καταχωρητές. Για τους ίδιους λόγους πρέπει να καθυστερήσει – συγχρονιστεί και το σήμα ελέγχου που δείχνει πότε ξεκινάει και πότε τελειώνει η αντικατάσταση του μπλοκ.

Άρα η μονάδα delay unit περιέχει όλους τους καταχωρητές που χρειάζονται για να λειτουργήσει σωστά το σύστημα πέραν των καταχωρητών που προσομοιώνουν την καθυστέρηση της κύριας μνήμης. Η μονάδα αυτή είναι ακριβώς η ίδια για το δεύτερο και τρίτο σύστημα, ενώ έχει λιγότερα σήματα εισόδου - εξόδου στο πρώτο σύστημα. Δίνονται οι δύο αντιστοιχίσεις:

Πρώτο σύστημα

```
nwa_delay_unit_1:nwa_delay_unit
        GENERIC MAP
        (
                memory_width=>main_mem_addr_width
        )
        PORT MAP
        (
                Mem_word_in=>main_word_out,
                Address_in=>main_address,
                Wren_in=>replace_main2cache,
                LdEn=>normal_mode,
                Reset=>reset_registers,
                Clock=>Clock,
                Mem_word_out=>delayed_main_word_out,
                Address_out=>delayed_main_rdaddress,
                Wren_out_1=>delayed_replace_block_signal,
                Wren_out_2=>delayed_plus_replace_block_signal
        );
```

Δεύτερο και τρίτο σύστημα

```
wa_delay_unit_1:wa_delay_unit
	GENERIC MAP
	(
		memory_width=>main_mem_addr_width
	)
	PORT MAP
	(
		Mem_word_in=>main_word_out,
		Address_in=>main_address,
		Wren_in=>replace_main2cache,
		Read_miss_in=>read_miss_signal,
		LdEn=>normal_mode,
		Reset=>reset_registers,
		Clock=>Clock,
		Mem_word_out=>delayed_main_word_out,
		Address_out=>delayed_main_rdaddress,
		Wren_out_1=>delayed_replace_block_signal,
		Wren_out_2=>delayed_plus_replace_block_signal,
		Read_miss_out=>delayed_read_miss_signal
	);
```

Εδώ τελειώνει η ανάλυση της σχεδίασης. Πρέπει να τονίσω πως ό,τι περιγράφτηκε με λόγια, σχήματα είτε κώδικα αποτελεί την τελική μορφή της σχεδίασης στην οποία έφτασα μετά από πολλές αλλαγές. Σχεδόν τίποτα δεν έμεινε ίδιο από την αρχική μου σχεδίαση και αυτό είναι λογικό γιατί μου διέφευγαν κάποιες παράμετροι. Ο έλεγχος του όλου συστήματος ακολουθεί στο επόμενο κεφάλαιο. Επιμέρους έλεγχοι πραγματοποιούνταν όμως καθόλη την εγγραφή του κώδικα, στις επιμέρους μονάδες.

ΚΕΦΑΛΑΙΟ 5

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΕΛΕΓΧΟΣ

5.1 Περιβάλλον σχεδίασης – υλοποίησης

Πριν προχωρήσω στην ανάλυση του ελέγχου θα παρουσιάσω εν συντομία τη γλώσσα και τα εργαλεία, που θα χρησιμοποίησα για την υλοποίηση της σχεδίασης. Θεωρητικά μπορεί να υποθέσει κανείς ότι η σχεδίαση έχει μια τέτοια αυτοτέλεια ώστε να εμφανίζεται ανεξάρτητη από τον τρόπο που θα υλοποιηθεί. Αυτό θα συνέβαινε πραγματικά αν είχαμε στη διάθεσή μας κατάλληλα εργαλεία και γλώσσες που θα μας επέτρεπαν να υλοποιούμε οποιαδήποτε κυκλώματα κι αν σχεδιάζαμε στο χαρτί. Επειδή όμως και οι γλώσσες αλλά κυρίως τα εργαλεία δεν είναι όσο ευέλικτα θα θέλαμε πρέπει η σχεδίαση που θα κάνουμε να συμβαδίζει με τις απαιτήσεις ή τους περιορισμούς αυτών. Μόνο έτσι αυτό που σκεφτήκαμε και σχεδιάσαμε στο χαρτί θα το δούμε κάποτε να υλοποιείται στην πράξη.
Δύο παραδείγματα είναι:
α) η μη υποστήριξη του for loop statement από εργαλεία, όπως το MAX + plus II και το Quartus της Altera και το ISE 7.1 της Xilinx, ενώ υπάρχουν στο συντακτικό της γλώσσας VHDL.
β)Ο περιορισμός του πλάτος, μιας δημιουργούμενης από τον wizard μνήμης, που πρέπει να είναι τουλάχιστον δύο bits. Αυτός ο περιορισμός υπάρχει στο εργαλείο MAX + plus II της Altera.

ΓΛΩΣΣΑ

Η γλώσσα που χρησιμοποίησα είναι η VHDL. Η VHDL είναι μια γλώσσα προγραμματισμού με την οποία περιγράφουμε την συμπεριφορά, την δομή και την υλοποίηση ενός ψηφιακού κυκλώματος ή συστήματος. Η γλώσσα αυτή συνιστά ένα εργαλείο CAD και έχει καθιερωθεί σαν ένα πρότυπο (standard) στη σχεδίασση ηλεκτρονικών κυκλωμάτων ASIC (Application Specific Integrated Circuits). Το γεγονός αυτό μας εγγυάται ότι οι νεότερες εκδόσεις εργαλείων σχεδίασης θα υποστηρίζουν το πρότυπο αυτό. Έτσι μια περιγραφή ενός κυκλώματος που αναπτύχθηκε με τα σημερινά εργαλεία σχεδίασης θα είναι μεταφέρσιμη μελλοντικά σε νέα εργαλεία σχεδίασης με ελάχιστες τροποποιήσεις. Επίσης οι περιγραφές κυκλωμάτων που αναπτύχθηκαν από διάφορους σσχεδιαστές θα είναι διαθέσιμες σε μια κοινή βάση δεδομένων η χρήση της οποίας θα μας διευκολύνει στην επίλυση παρόμοιων σχεδιαστικών προβλημάτων Η συγκεκριμένη γλώσσα είναι κατάλληλη για ανάπτυξη ψηφιακών κυκλωμάτων και είναι ιδιαίτερα διαδεδομένη τα τελευταία χρόνια. Επιλέχθηκε επίσης διότι ήταν η γλώσσα με την οποία είχα αρκετή τριβή, αφού αυτήν χρησιμοποίησα κατά την διεξαγωγή εργαστηριακών ασκήσεων ορισμένων προπτυχιακών μαθημάτων.

ΕΡΓΑΛΕΙΑ

Στην αρχή της εργασίας χρησιμοποίησα το εργαλείο MAX + plus II της Altera, το οποίο όμως εγκατέληψα σύντομα.
Η προσομοίωση και ο έλεγχος ορθής λειτουργίας του συστήματος έγιναν με το εργαλείο ModelSim SE 6.0a.
Η υλοποίηση του συστήματος (σύνθεση, τοθέτηση και διασύνδεση σε FPGA) έγινε με το εργαλείο ISE 7.1 της Xilinx.
Ο κώδικας σε γλώσσα VHDL αφού γραφόταν ελεγχόταν και από τα δύο εργαλεία.

5.2 Υλοποίηση

Αφού δεν είμαστε σε θέση να κατασκευάσουμε το πραγματικό σύστημα, χρησιμοποιούμε εργαλεία CAD που κάνουν την υλοποίηση με λογισμικό τρόπο. Χρησιμοποίησα το εργαλείο ISE 7.1i της Xilinx και συγκεκριμένα την λειτουργία Implement Design που υλοποιεί ένα project σε μια συσκευή (FPGA).Έκανα την υλοποίηση για μια σειρά συνόλων με διαφορετικές παραμέτρους. Από τα αποτελέσματα των υλοποιήσεων αυτών δημιούργησα τους παρακάτω πίνακες:
Όλες οι υλοποιήσεις έγιναν στην συσκευή **xc4vlx60** της οικογένειας **Virtex 4** και με **Speed Grade: -12**

1ο Σύστημα

Παράμετροι			Αποτελέσματα				
block size	set assoc	cache size	Ελάχιστη Περίοδος (ns)	Μέγιστη Συχνότητα (MHz)	# LUTs 4 - εισόδων	# αποσχολ. Slices	# RAMB16s
4	1	32	7.067	141.512	1,154	889	137
4	4	32	10.049	99.516	1,348	999	140
4	8	32	10.395	96.197	1,695	1,137	140
2	2	32	7.943	125.903	1,173	886	144
4	2	32	7.943	125.903	1,172	886	138
8	2	32	7.943	125.903	1,174	888	136
2	2	4	8.075	123.845	1,154	865	120
2	2	8	8.009	124.866	1,159	868	124
2	2	16	8.009	124.866	1,166	883	130
4	2	64	7.943	125.903	1,318	973	158
4	2	128	9.298	107.548	1,200	924	140

2ο Σύστημα

Παράμετροι			Αποτελέσματα				
block size	set assoc	cache size	Ελάχιστη Περίοδος (ns)	Μέγιστη Συχνότητα (MHz)	# LUTs 4 - εισόδων	# αποσχολ. Slices	# RAMB16s
4	1	32	7.213	138.647	1,168	904	137
4	4	32	9.701	103.085	1,370	993	140
4	8	32	10.409	96.073	1,711	1,158	140
2	2	32	9.040	110.617	1,186	902	144
4	2	32	9.040	110.617	1,187	901	138
8	2	32	7.987	125.207	1,187	902	136
2	2	4	9.172	109.025	1,169	884	120
2	2	8	9.106	109.815	1,173	885	124
2	2	16	9.106	109.815	1,180	899	130
4	2	64	9.040	110.617	1,332	991	158
4	2	128	9.426	106.085	1,215	940	140

3ο Σύστημα

Παράμετροι			Αποτελέσματα				
block size	set assoc	cache size	Ελάχιστη Περίοδος (ns)	Μέγιστη Συχνότητα (MHz)	# LUTs 4 - εισόδων	# αποσχολ. Slices	# RAMB16s
4	1	32	9.859	101.425	1,286	978	138
4	4	32	10.895	91.787	1,628	1,130	144
4	8	32	13.012	76.850	2,230	1,446	148
2	2	32	10.305	97.041	1,337	986	146
4	2	32	10.226	97.795	1,335	1,002	140
8	2	32	9.814	101.899	1,336	988	138
2	2	4	9.945	100.555	1,311	963	122
2	2	8	10.109	98.921	1,320	967	126
2	2	16	10.231	97.737	1,329	981	132
4	2	64	10.305	97.041	1,488	1,092	160
4	2	128	10.663	93.778	1,307	994	142
4	2	256	10.412	96.043	1,424	1,053	158

Παρατηρήσεις

Η ελάχιστη περίοδος και η μέγιστη συχνότητα είναι μεγέθη που χαρακτηρίζουν την ταχύτητα ρολογιού, άρα και την ταχύτητα του συστήματος.

Η ταχύτητα αυξάνεται καθώς: αυξάνεται το μέγεθος του μπλοκ, ενώ μειώνεται η συνολοσυσχετιστικότητα και το μέγεθος της κρυφής μνήμης.

Τα LUTs είναι οι πίνακες αναφοράς της FPGA συσκευής και έχει 4 εισόδους το καθένα, δηλαδή μπορεί να παράγει το αποτελέσμα μιας συνάρτησης 4 μεταβλητών.

Ο αριθμός των LUTs αυξάνεται καθώς: αυξάνεται η συνολοσυσχετιστικότητα ή μειώνεται το μέγεθος του μπλοκ.

Τα Slices είναι κομμάτια λογικές που σχηματιζούν ένα CLB που είναι η βασική μονάδα μιας FPGA. Κάθε slice περιέχει 2 LUTs. Στον πίνακα δίνεται ο αριθμός των απασχολούμενων slices.

Αυτός αυξάνεται καθώς: αυξάνεται η συνολοσυσχετιστικότητα.

Τέλος τα RAMB16s είναι τα μπλοκ μνήμης που χρησιμοποιεί η συσκευή για να υλοποιήση την κρυφή μνήμη.

Όπως είναι λογικό τα απαιτούμενα μπλοκ μνήμης αυξάνονται καθώς αυξάνεται η συνολοσυσχετιστικότητα και το μέγεθος της κρυφής μνήμης ή μειώνεται το μέγεθος του μπλοκ (μικρότερες μνήμες ετικετών).

Critical path:

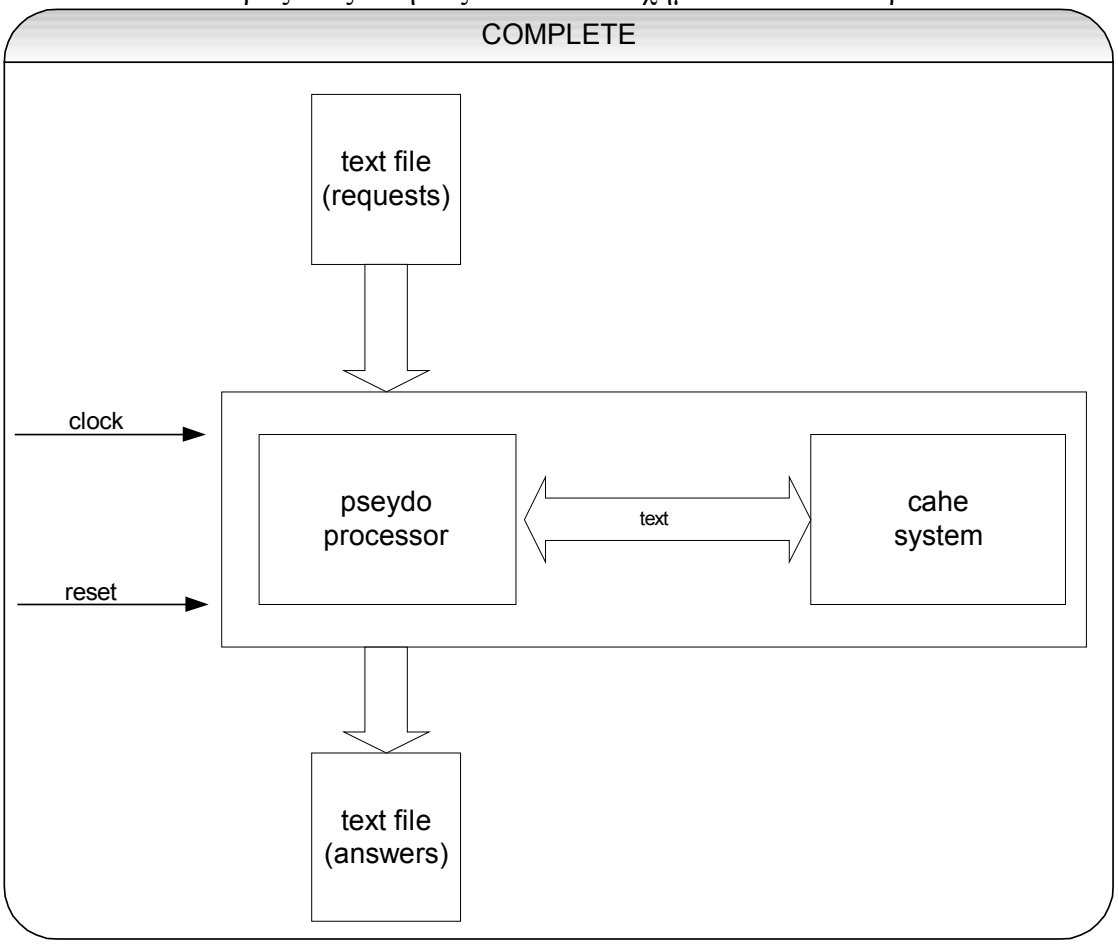
Η συχνότητα του συστήματος βρίσκεται από το μονοπάτι που έχει τη μεγαλύτερη καθυστέρηση. Στο σύστημα αυτό το μονοπάτι που καθορίζει τη συχνότητα είναι αυτό που ξεκινάει από τη μνήμη ετικετών περνάει από τον συγκριτή, ύστερα από τη μονάδα final hit και δίνει το παραγόμενο σήμα σε ένα καταχωρητή. Η χρονική διάρκεια μετάδοσης αυτού του σήματος ορίζει και την ελάχιστη ημιπερίοδο του ρολογιού. Ο λόγος που έχουμε ημιπερίοδο και όχι περίοδο είναι ότι η μνήμη λειτουργεί με τη θετική ακμή του ρολογιού, ενώ οι καταχωρητές του συστήματος λειτουργούν με την αρνητική ακμή του ρολογιού. Έτσι η μετάδοση του συγκεκριμένου σήματος πρέπει να γίνει μέσα σε μισό κύκλο.

5.3 Δημιουργία ενός ολοκληρωμένου συστήματος

Η ανάλυση της σχεδίασης έχει ολοκληρωθεί και έχουμε τρία συστήματα έτοιμα για έλεγχο. Ο ξεχώριστος έλεγχος όμως είναι λίγο κουραστικός. Έτσι γεννήθηκε η ιδέα να ενώσω τα τρία συστήματα σε ένα. Επειδή τα δύο πρώτα έμοιαζαν αρκετά ξεκίνησα με το ενώσω αυτά πρώτα. Έτσι δημιούργησα ένα σύστημα κρυφής μνήμης διεγγραφής, το οποίο έχει σαν επιλογή αν θέλουμε κατανομή εγγραφής ή όχι. Τέλος ένωσα και τα τρια δημιουργώντας ένα σύστημα κρυφής μνήμης στο οποίο μπορούμε να επιλέξουμε ποια από τις τρεις μορφές θέλουμε να υλοποιήσουμε. Η ένωση των τριών συστημάτων βασίστηκε στο if generate statement που μας παρέχει η γλώσσα VHDL. Έτσι στο τέλος κατέληξα με πέντε υλοποιήσεις, μία για κάθε σύστημα και δύο με ενώσεις αυτών.

Ένα άλλο θέμα ήταν το πως θα κάνω τα τεστ χωρίς να χρειάζεται να βάζω με το χέρι μία μία τις αιτήσεις στην είσοδο την κατάλληλη στιγμή (μόλις πάρω το ok). Έτσι δημιούργησα μια μονάδα που παίζει το ρόλο του έξω κόσμου, δηλαγή του επεξεργαστή. Ο ψευδοεπεξεργαστής αυτός διαβάζει αιτήσεις από ένα αρχείο και στέλνει τα κατάλληλα σήματα. Όταν πάρει την επιβεβαίωση παίρνει την απάντηση και στέλνει την επόμενη αίτηση. Αυτό που καταγράφεται σε καθε απάντηση που λαμβάνει είναι το αν είχαμε hit ή miss καθώς και την τιμή της λέξης σε περίπτωση αίτησης ανάγνωσης. Δεν δίνω το κώδικα γιατί δεν είναι κάτι το ιδιαίτερο. Το μόνο που χρειάστηκε ήταν να μάθω κάποιες εντολές και βιβλιοθήκες για την διαχείριση αρχείων στην VHDL.

Τελικά δημιουργήθηκε ένα αυτόνομο σύστημα που ονόμασα complete. Οι μόνες είσοδοι που χρειάζεται είναι το ρολόι και το reset, ενώ θέλει και ένα αρχείο κειμένου από το οποίο θα διαβάζει τις αιτήσεις. Δίνεται το σχήμα που το αναπαριστά.



Τέλος, έφτιαξα και ένα πρόγραμμα σε C που, βάσει των τριών παραμέτρων που δίνει ο χρήστης στο σύστημα, παράγει τις υπόλοιπες εξαρτημένες παραμέτρους που τυχόν μπορεί να δυσκολέψουν το χρήστη να υπολογίσει. Ο κώδικας ακολουθεί:

```
#include <stdio.h>
#include <math.h>
 int main()
{
        unsigned long x;
        unsigned long y;
        unsigned long z;
        unsigned long block_offset;
        unsigned long tag_addr;
        unsigned long tag;
        unsigned long data_addr;
        puts("Give block size (words)");
        scanf("%lu",&x);
        puts("Give set associativity");
        scanf("%lu",&y);
        puts("Give cache size (KBytes)");
        scanf("%lu",&z);
        block_offset=log10(x)/log10(2);
        tag_addr=log10(z/(x*y))/log10(2)+8;
        data_addr=tag_addr+block_offset;
        tag=62-data_addr;
        printf("\nBlock_offset:%lu\n",block_offset);
        printf("\nTag_addr:%lu\n",tag_addr);
        printf("\nData_addr:%lu\n",data_addr);
        printf("\nTag:%lu\n",tag);
}
```

5.4 Προσομοίωση

Μετά την σχεδίαση και υλοποίηση του συστήματος σε γλώσσα VHDL, προχώρησα στην προσομοίωση του. Η προσομοίωση είναι η διαδικασία κατά την οποία βλέπουμε το σύστημα να λειτουργεί, όχι στην πραγματικότητα αλλά εικονικά, Ανάλογα με το αποτέλεσμα της προσομοίωσης καταλαβαίνουμε αν κάτι δεν λειτουργεί σωστά. Ο έλεγχος των αποτελεσμάτων δεν είναι απλή υπόθεση. Μπορεί για παράδειγμα το αποτέλεσμα μιας προσομοίωσης να είναι σωστό, αλλά όχι γιατί το σύστημα δούλεψε έτσι όπως έπρεπε.

Στην περίπτωση του δικού μου συστήματος μπορώ να αναφέρω κάποια σενάρια τέτοιων προσομοιώσεων. Αν ελέγχω μόνο τη λέξη που θα μου βγάλει σαν έξοδο η κρυφή μνήμη σε κάθε ανάγνωση τότε θα μπορούσα αντί για αυτό που έφτιαξα, να χρησιμοποιούσα μια απλή μνήμη. Σίγουρα αυτή θα μου έδινε τις σωστές απαντήσεις!

Επίσης από την παρατήρηση και μόνο του αποτελέσματος δεν μπορώ να ελέγξω καθένα από τα τρία συστήματα ξεχωριστά αφού και τα τρία δίνουν τις ίδιες απαντήσεις.

Ένα τελευταίο παράδειγμα είναι ότι αν έχω ένα σύστημα μνήμης που όλα του τα δεδομένα έχουν την τιμή μηδέν, τότε αν διαβάσω μια διεύθυνση και πάρω το αποτέλεσμα μηδέν, δεν θα μπορώ ποτέ να είμαι σίγουρος ότι όλα πήγαν καλά. Η τιμή αυτή εμφανίζεται σαν έξοδο σε πάρα πολλές περιπτώσεις ανάλογα και με τα χαρακτηριστικά της προσομοίωσης. Αν όμως περιμένω να διαβαστεί η τιμή 123 και όντως αυτό συμβεί, θα έχω ελαχιστοποιήσει την πιθανότητα σωστού αποτελέσματος από τύχη.

Στην πράξη δεν ξεκίνησα τις προσομοιώσεις μετά την πλήρη σχεδίαση και υλοποίηση. Κάθε ένα κομμάτι που έφτιαχνα ελεγχόταν μέσω προσομοίωσης πριν προχωρήσω. Με αυτό τον τρόπο εντόπισα αρκετά μικρά και ανόητα λάθη που θα ήταν πολύ

δυσκολότερο να τα βρω στην τελική φάση της προσομοίωσης. Η καταγραφή όλων αυτών των σφαλμάτων θα ήταν κουραστική και τις περισσότερες φορές ανούσια. Απλώς αναφέρω ότι υπήρξαν για μην πάρει ο αναγνώστης τη λανθασμένη εντύπωση ότι όλα κύλησαν τόσο ομαλά όσο θα φανούν παρακάτω.

5.4.1 Στρατηγικές ελέγχου

Υπάρχουν διάφοροι τρόποι για να ελεγχθεί το σύστημα μέσω της προσομοίωσης. Το σύστημα μπορούμε να το δούμε ως ένα εξυπηρετητή αιτήσεων. Οπότε ένα εύλογο ερώτημα που προκύπτει είναι: πόσες αιτήσεις θα περιέχει ένα τεστ.
Αρχικά δημιουργούσα τεστ με λίγες αιτήσεις (5 με 10). Εκεί κοιτούσα αναλυτικά τη ροή των δεδομένων. Αυτό επιτυγχάνεται παρατηρώντας όσα περισσότερα σήματα γίνεται στην κυματομορφή της προσομοίωσης.
Ένας πιο ολοκληρωμένος έλεγχος απαιτεί πολλές αιτήσεις (δεκάδες ή και εκατοντάδες). Επιπλέον επειδή ο έλεγχος δεν μπορεί σε τέτοιο πλήθος αιτήσεων να γίνει από ανθρώπινο μάτι (κουραστικό και επικίνδυνο), χρειάζεται μια άλλη προσέγγιση. Έτσι κατέληξα στη παρακάτω στρατηγική:
Αρχικά χρειαζόμαστε ένα σύστημα που να μας δίνει με σιγουριά σωστά αποτελέσματα. Το σύστημα αυτό είναι απαραίτητο ώστε να παίρνουμε τις απαντήσεις με τις οποίες θα συγκρίνουμε εκείνες του συστήματος που ελέγχουμε. Στην προηγούμενη στρατηγική των μερικών αιτήσεων υπολόγιζα μόνος μου τις τιμές των απαντήσεων, όμως τώρα θέλουμε κάτι πιο γρήγορο και αυτόματο. Έτσι δημιούργησα το σύστημα αλλά αυτή τη φορά με καθαρή γλώσσα software. Το σύστημα αυτό θα το ονομάζω ως οδηγό. Ένα άλλο πρόγραμμα δημιουργεί τις αιτήσεις που χρειαζόμαστε για το τεστ, οι οποίες δίνονται στο σύστημα-οδηγό αλλά και στο σύστημα που ελέγχουμε. Τέλος συγκρίνουμε τις απαντήσεις των δύο συστημάτων μέσω ενός έτοιμου προγράμματος που συγκρίνει αρχεία. Φυσικά πρέπει να τροποποιήσουμε το αρχικό μας σύστημα ώστε να δίνει απαντήσεις με την ίδια μορφή που δίνει και το σύστημα-οδηγός.

5.4.2 Δημιουργία συστήματος οδηγού

Μια εύκολη και σίγουρη λύση ήταν η χρησιμοποίηση ενός έτοιμου και δοκιμασμένου προσομοιωτή κρυφής μνήμης που θα υποστήριζε τις επιθυμητές παραμέτρους. Ένα τέτοιο πρόγραμμα είναι το Dinero. Συγκεκριμένα κατέβασα από το διαδύκτιο το Dinero IV, release 7. Υπήρχε όμως ένα πρόβλημα. Ενώ για την άμεσα αντιστοιχισμένη κρυφή μνήμη όλα λειτουργούν όπως πρέπει, στην συνολοσυσχετιστική κρυφή μνήμη υπάρχει ασυμβατότητα ανάμεσα στο Dinero και το δικό μου σύστημα. Το πρόβλημα εντοπίζεται στο τρόπο που επιλέγεται ποιο σύνολο (set) θα αντικατασταθεί κατά «τυχαίο» τρόπο. Το Dinero χρησιμοποιεί διαφορετικό αλγόριθμο για να υποστηρίξει την τυχαία επιλογή από το δικό μου. Μπορεί επομένως σε ένα μεγάλο αριθμό αιτήσεων να είχαμε ίδια ή παρόμοια αποτελέσματα ευστοχίας (hit ratio), όμως αν παίρναμε κάθε αίτηση ξεχωριστά θα βρίσκαμε κάποιες διαφορές.
Έτσι έγραψα μόνος μου τον κώδικα του προσομοιωτή του συστήματος που ήδη είχα δημιουργήσει. Η γλώσσα που χρησιμοποίησα ήταν η C. Αυτό που κάνει το πρόγραμμα είναι να παίρνει αιτήσεις και να απαντάει αν έχουμε ευστοχία ή αστοχία. Επιπλέον δίνει τα δεδομένα της κατάλληλης διεύθυνσης αν πρόκειται για ανάγνωση.
Αρχικά χρειαζόμαστε ένα πρόγραμμα που θα παράγει τις αιτήσεις και να τις γράφει σε ένα αρχείο. Ο κώδικας που έγραψα δίνεται παρακάτω:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int n=300;//max: 65536;
	int m=20000;
	int temp=0,data=0,random=0;
	char c;
	int i;
	FILE *fp;
	fp=fopen("requests.txt","w");
	for(i=1;i<=50;i++)
	{
		random=((double)(rand())/RAND_MAX)*2;
		if(random==0)
		{
			c='r';
			temp=((double)(rand())/RAND_MAX)*n;
			fprintf(fp,"%c ",c);
			if(temp>=0x1000)
				fprintf(fp,"%X\n",temp);
			if(temp<0x1000 && temp>=0x100)
				fprintf(fp,"0%X\n",temp);
			if(temp<0x100 && temp>=0x10)
				fprintf(fp,"00%X\n",temp);
			if(temp<0x10)
				fprintf(fp,"000%X\n",temp);
		}
		else
		{
			c='w';
			temp=((double)(rand())/RAND_MAX)*n;
			data=((double)(rand())/RAND_MAX)*m;
			fprintf(fp,"%c ",c);
			if(temp>=0x1000)
				fprintf(fp,"%X ",temp);
			if(temp<0x1000 && temp>=0x100)
				fprintf(fp,"0%X ",temp);
			if(temp<0x100 && temp>=0x10)
				fprintf(fp,"00%X ",temp);
			if(temp<0x10)
				fprintf(fp,"000%X ",temp);
			if(data>=0x1000)
				fprintf(fp,"%X\n",data);
			if(data<0x1000 && data>=0x100)
				fprintf(fp,"0%X\n",data);
			if(data<0x100 && data>=0x10)
				fprintf(fp,"00%X\n",data);
			if(data<0x10)
				fprintf(fp,"000%X\n",data);
		}
	}
	fclose(fp);
}
```

Εδώ παράγω ενδεικτικά 50 αιτήσεις. Προφανώς με αλλαγή του αριθμού 50 στο loop έχουμε όσες αιτήσεις θέλουμε. Το n είναι η μέγιστη τιμή της διεύθυνσης που θέλουμε να παραχθεί, ενώ το m είναι το αντίστοιχο μέγεθος για τα δεδομένα. Τα πολλά if

μπήκαν μόνο για να έχω την επιθυμητή μορφή στο αρχείο που δημιουργείται. Τώρα δίνω τον κώδικα του προσομοιωτή της κρυφής μνήμης.

```
#include<stdio.h>
int alloc=1;
int set_as=2;
int indexo=256;
int block_size=16;//se bytes
int misses=0;
int hits=0;
double miss_rate=0;
int req=0;
int counter=0;
int tag_memory[2][256];//i prwti aggili periexei ton arithmo twn synolwn
                       //kai i deyteri ton arithmo twn block ana synolo
int memory[65536];
void read_cache(int address);
void write_allocate_cache(int address, int data);
void no_write_allocate_cache(int address, int data);
FILE *fo;

int main()
{
        int i,j,input,data;
        char c;
        FILE *mem;
        FILE *fp;
        fo=fopen("output2.txt","w");
        mem=fopen("mem.txt","w");
        fp=fopen("requests.txt","r");
        for(i=0;i<65536;i++)
        {
                memory[i]=i+1;
                fprintf(mem,"%X\n",memory[i]);
        }
        for(i=0;i<set_as;i++)
        {
                for(j=0;j<indexo;j++)
                {
                        tag_memory[i][j]=-1;
                }
        }
        while(1)
        {
                data=0;
                i=fscanf(fp,"%c %X",&c,&input);
                if(i==EOF)
                {
                        break;
                }
                if(c=='w')
                {
                        i=fscanf(fp,"%X",&data);
                }
                printf("Elava %c %X %X\n",c,input,data);
                if(c=='r')
                {
                        read_cache(input);
                }
                if(c=='w')
```

```
                {
                        if(alloc==0)
                        {
                                no_write_allocate_cache(input,data);
                        }
                        else
                        {
                                write_allocate_cache(input,data);
                        }
                }
                if(i==EOF)
                {
                        break;
                }
                i=fscanf(fp,"\n");
                if(i==EOF)
                {
                        break;
                }
        }
        fclose(fp);
        fclose(mem);
        fclose(fo);
}

void read_cache(int address)
{
        int p,k,hit,i,data;
        p=(address/block_size)%indexo;
        k=(address/block_size)*block_size;
        req++;
        hit=0;
        for(i=0;i<set_as;i++)
        {
                if(tag_memory[i][p]==k)
                {
                        printf("Hit!Data:%X\n",memory[address/4]);
                        fprintf(fo,"Hit ");
                        hits++;
                        hit=1;
                }
        }
        if(hit==0)
        {
                printf("Miss.Data:%X\n",memory[address/4]);
                fprintf(fo,"Miss ");
                misses++;
                tag_memory[counter][p]=k;
                counter=(counter+1)%set_as;
        }
        data=memory[address/4];
        if(data<0x100000000 && data>=0x10000000)
                fprintf(fo,"%X\n",data);
        if(data<0x10000000 && data>=0x1000000)
                fprintf(fo,"0%X\n",data);
        if(data<0x1000000 && data>=0x100000)
                fprintf(fo,"00%X\n",data);
        if(data<0x100000 && data>=0x10000)
                fprintf(fo,"000%X\n",data);
        if(data<0x10000 && data>=0x1000)
```

```
                fprintf(fo,"0000%X\n",data);
        if(data<0x1000 && data>=0x100)
                fprintf(fo,"00000%X\n",data);
        if(data<0x100 && data>=0x10)
                fprintf(fo,"000000%X\n",data);
        if(data<0x10)
                fprintf(fo,"0000000%X\n",data);
}
void write_allocate_cache(int address, int data)
{
        int p,k,hit,i;
        p=(address/block_size)%indexo;
        k=(address/block_size)*block_size;
        req++;
        hit=0;
        for(i=0;i<set_as;i++)
        {
                if(tag_memory[i][p]==k)
                {
                        printf("Hit!\n");
                        fprintf(fo,"Hit\n");
                        hits++;
                        hit=1;
                }
        }
        if(hit==0)
        {
                printf("Miss.\n");
                fprintf(fo,"Miss\n");
                misses++;
                tag_memory[counter][p]=k;
                counter=(counter+1)%set_as;
        }
        //write part
        memory[address/4]=data;
}
void no_write_allocate_cache(int address, int data)
{
        int p,k,hit,i;
        p=(address/block_size)%indexo;
        k=(address/block_size)*block_size;
        req++;
        hit=0;
        for(i=0;i<set_as;i++)
        {
                if(tag_memory[i][p]==k)
                {
                        printf("Hit!\n");
                        fprintf(fo,"Hit\n");
                        hits++;
                        hit=1;
                }
        }
        if(hit==0)
        {
                printf("Miss.\n");
                fprintf(fo,"Miss\n");
                misses++;
        }
        //write part
```

```
        memory[address/4]=data;
}
```

Στην αρχή του κώδικα αρχικοποιώ την κύρια μνήμη για να μην έχει μηδενικές τιμές. Παράλληλα δημιουργώ και ένα αρχείο με τις τιμές αυτές για να το χρησιμοποιήσω ως αρχείο αρχικοποίησης (mif) κατά την προσομοίωση της VHDL κρυφής μνήμης. Κάνω όλες τις τιμές της μνήμης ετικετών -1 για να μην υπάρξει λάθος στη σύγκριση. Ο υπόλοιπος κώδικας υλοποιεί τη λογική της κρυφής μνήμης. Πάλι κατά την εκτύπωση των αποτελεσμάτων στο αντίστοιχο αρχείο χρειάζεται η κατάλληλη μορφοποίηση (πολλά if – statements παρόμοια). Στον παραπάνω κώδικα εκτός από το αρχείο output2.txt η πληροφορία αποτυπώνεται και στην οθόνη. Αυτό ήταν για καθαρά δική μου βοήθεια. Οι γραμμές αυτές μπορούν να παραληφθούν. Μια σημαντική παρατήρηση είναι ότι ενώ διαχωρίζονται οι περιπτώσεις της κατανομής εγγραφής ή μη, δεν διαχωρίζεται αν έχουμε ετεροχρονισμένη εγγραφή ή όχι. Αυτό συμβαίνει γιατί οι πληροφορία που μας δίνεται ως απάντηση είναι η ίδια. Δηλαδή έχουμε τα ίδια hits και miss. Αυτό που διαφέρει είναι η εσωτερική λειτουργία της κρυφής μνήμης, κάτι που δεν φαίνεται στο σύστημα – οδηγό.
Πως είμαστε σίγουροι ότι ο προηγούμενος κώδικας δίνει σίγουρα τα σωστά αποτελέσματα; Δεν μπορούμε να είμαστε απόλυτα σίγουροι. Αυτό που έκανα ήταν να συγκρίνω τα αποτελέσματα του δικού μου προγράμματος με εκείνα του Dinero στην περίπτωση της άμεσα αντιστοιχισμένης κρυφής μνήμης. Τα αποτελέσματα ήταν πάντα τα ίδια. Όσο για την περίπτωση της συνολοσυσχετιστικής κρυφής μνήμης αρκέστηκα στα παρόμοια ποσοστά επιτυχίας που δίνουν μεταξύ τους, κάτι που είναι αναμενόμενο. Ακόμα η απλότητα του κώδικα, μου δίνει σχεδόν τη βεβαιότητα της σωστής λειτουργίας. Τέλος μια διαφοροποίηση των αποτελεσμάτων του οδηγού με το σύστημα που ελέγχουμε, μπορεί να αποτελέσει μια τελευταία ευκαιρία διόρθωσης του ίδιου του οδηγού, αν ελέγξουμε το αποτέλεσμα με τη λογική. Κάτι τέτοιο βέβαια αντιβαίνει στην υπόθεση του σωστού οδηγού.

5.4.3 Δημιουργία μη – τυχαίων αιτήσεων

Εκτός από την δημιουργία τυχαίων αιτήσεων θεώρησα καλό να φτιάξω ένα δικό μου τεστ το οποίο θα περιέχει όσο ποιο πολλά σενάρια «δύσκολων» καταστάσεων. Βασίστηκα στα σενάρια που ήδη είχα κταγράψει στη σχεδίαση. Θα δείξω το τεστ αυτό με επεξήγηση των σεναρίων.

w 0024 4444 r 0024	Διάβασμα μετά από γράψιμο.
w 03FD 1111 r 0029 r 03FD	Διάβασμα λέξης που είναι σε αναμονή για να γραφτεί
r 03FE w 56BA 2222 w 03F5 3333 w 03FA 5555 w 03F0 6666 r 56BA	Συνεχόμενες εύστοχες εγγραφές και διάβασμα μιας από αυτές
r 56B8 w D3C1 1234 r 56B3 r 56B6 r 56B9 r D3C1	Συνεχόμενες εύστοχες ανγνώσεις που δεν αφήνουν να γραφτεί μια λέξη η οποία και διαβάζεται

w 56BA 7777 w 03F5 FFFF w 031A DDDD w 03F0 AAAA r 56BA	Παρεμβολή μιας άστοχης εγγραφής ανάμεσα σε εύστοχες
w D3C1 5678 r 56B3 r 5FB6 r 56B9 r D3C1	Παρεμβολή μιας άστοχης ανάγνωσης ανάμεσα σε εύστοχες που κάνει δυνατή την εγγραφή της πρώτης αίτησης σε αναμονή
w 0091 9999 w 009C 8888 r 0091 r 009C	Εγγραφή και ανάγνωση γειτονικών διευθύνσεων
r 1237 w 1232 4568 w 123D 3622 r 123D r 123B r 1232 r 1237	Αλλαγές ανάμεσα σε ανάγνωση και εγγραφή γειτονικών διευθύνσεων
r 3555 r 4555 w 3555 9753 w 4555 8642	Αστοχία ανάγνωσης ή και εγγραφής ανάλογα με τη συνολοσυσχετιστικότητα
w 2435 ABCD r 2435 w 4435 DCBA r 4435	Αστοχίες εγγραφής και ανάγνωση της ίδιας διεύθυνσης
r 0406 w 0406 289D r F406 w F406 4852	Εγγραφή μετά ανάγνωση με αστοχία στην δεύτερη περίπτωση
r 5D10 w 5D10 17BF w 7D10 1000 r 7D10	Αστοχίες που παράγουν αντικατάσταση κατά την εγγραφή ή την ανάγνωση (κατανομή εγγραφής ή μη)
r 00FC r 10FC r 20FC r 30FC w 20FC 1551 w 10FC 4170 w 30FC 100B w 00FC 23C7	Συνεχείς αναγνώσεις από διευθύνσεις με ίδιο το τελευταίο μέρος τους και ύστερα εγγραφές σε αυτές. Έλεγχος ανάλογα με τη συνολοσυσχετιστικότητα

Στη συνέχεια έλεγξα με το παραπάνω τεστ διάφορες περιπτώσεις του συστήματός μου. Αυτό που πρέπει να σημιωθεί είναι τα αποτελέσματα ενός τεστ είναι θεωρητικά τα ίδια στο δεύτερο και το τρίτο σύστημα όπου έχουμε κατανομή εγγραφής. Η διαφορά τους που έχει να κάνει με την ετεροχρονισμένη ή όχι εγγραφή δεν μπορεί να φανεί από τα αποτελέσματα αλλά μόνο από τις κυματορφές. Παρ'όλα αυτά εγώ έκανα τα τεστ και

στα τρία συστήματα. Όλα λειτούργησαν σωστά σύμφωνα με τα αποτελέσματα του οδηγού. Δίνω τους αντίστοιχους πίνακες.

Παράμετροι	Σύστημα 1	Σύστημα 2	Σύστημα 3	Σύστημα 4	Σύστημα 5	Σύστημα 6
cahe size (KB)	8	8	4	4	2	2
block_size (words)	4	4	4	4	2	2
set_assoc	2	2	1	1	1	1
write_alloc	0	1	0	1	0	1
αιτήσεις	απαντήσεις	απαντήσεις	απαντήσεις	απαντήσεις	απαντήσεις	απαντήσεις
w 0024 4444	Miss	Miss	Miss	Miss	Miss	Miss
r 0024	Miss 00004444	Hit 00004444	Miss 00004444	Hit 00004444	Miss 00004444	Hit 00004444
w 03FD 1111	Miss	Miss	Miss	Miss	Miss	Miss
r 0029	Hit 0000000B	Hit 0000000B	Hit 0000000B	Hit 0000000B	Miss 0000000B	Miss 0000000B
r 03FD	Miss 00001111	Hit 00001111	Miss 00001111	Hit 00001111	Miss 00001111	Hit 00001111
r 03FE	Hit 00001111	Hit 00001111	Hit 00001111	Hit 00001111	Hit 00001111	Hit 00001111
w 56BA 2222	Miss	Miss	Miss	Miss	Miss	Miss
w 03F5 3333	Hit	Hit	Hit	Hit	Miss	Miss
w 03FA 5555	Hit	Hit	Hit	Hit	Hit	Hit
w 03F0 6666	Hit	Hit	Hit	Hit	Miss	Hit
r 56BA	Miss 00002222	Hit 00002222	Miss 00002222	Hit 00002222	Miss 00002222	Hit 00002222
r 56B8	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222
w D3C1 1234	Miss	Miss	Miss	Miss	Miss	Miss
r 56B3	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD	Miss 000015AD	Miss 000015AD
r 56B6	Hit 000015AE	Hit 000015AE	Hit 000015AE	Hit 000015AE	Hit 000015AE	Hit 000015AE
r 56B9	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222
r D3C1	Miss 00001234	Hit 00001234	Miss 00001234	Hit 00001234	Miss 00001234	Hit 00001234
w 56BA 7777	Hit	Hit	Hit	Hit	Hit	Hit
w 03F5 FFFF	Hit	Hit	Hit	Hit	Miss	Hit
w 031A DDDD	Miss	Miss	Miss	Miss	Miss	Miss
w 03F0 AAAA	Hit	Hit	Hit	Hit	Miss	Hit
r 56BA	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777
w D3C1 5678	Hit	Hit	Hit	Hit	Hit	Hit
r 56B3	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD
r 5FB6	Miss 000017EE	Miss 000017EE	Miss 000017EE	Miss 000017EE	Miss 000017EE	Miss 000017EE
r 56B9	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777
r D3C1	Hit 00005678	Hit 00005678	Hit 00005678	Hit 00005678	Hit 00005678	Hit 00005678
w 0091 9999	Miss	Miss	Miss	Miss	Miss	Miss
w 009C 8888	Miss	Hit	Miss	Hit	Miss	Miss
r 0091	Miss 00009999	Hit 00009999	Miss 00009999	Hit 00009999	Miss 00009999	Hit 00009999
r 009C	Hit 00008888	Hit 00008888	Hit 00008888	Hit 00008888	Miss 00008888	Hit 00008888
r 1237	Miss 0000048E	Miss 0000048E	Miss 0000048E	Miss 0000048E	Miss 0000048E	Miss 0000048E
w 1232 4568	Hit	Hit	Hit	Hit	Hit	Hit
w 123D 3622	Hit	Hit	Hit	Hit	Miss	Miss
r 123D	Hit 00003622	Hit 00003622	Hit 00003622	Hit 00003622	Miss 00003622	Hit 00003622

r 123B	Hit 0000048F	Hit 0000048F	Hit 0000048F	Hit 0000048F	Hit 0000048F	Hit 0000048F
r 1232	Hit 00004568	Hit 00004568	Hit 00004568	Hit 00004568	Hit 00004568	Hit 00004568
r 1237	Hit 0000048E	Hit 0000048E	Hit 0000048E	Hit 0000048E	Hit 0000048E	Hit 0000048E
r 3555	Miss 00000D56	Miss 00000D56	Miss 00000D56	Miss 00000D56	Miss 00000D56	Miss 00000D56
r 4555	Miss 00001156	Miss 00001156	Miss 00001156	Miss 00001156	Miss 00001156	Miss 00001156
w 3555 9753	Hit	Hit	Miss	Miss	Miss	Miss
w 4555 8642	Hit	Hit	Hit	Miss	Hit	Miss
w 2435 ABCD	Miss	Miss	Miss	Miss	Miss	Miss
r 2435	Miss 0000ABCD	Hit 0000ABCD	Miss 0000ABCD	Hit 0000ABCD	Miss 0000ABCD	Hit 0000ABCD
w 4435 DCBA	Miss	Miss	Miss	Miss	Miss	Miss
r 4435	Miss 0000DCBA	Hit 0000DCBA	Miss 0000DCBA	Hit 0000DCBA	Miss 0000DCBA	Hit 0000DCBA
r 0406	Miss 00000102	Miss 00000102	Miss 00000102	Miss 00000102	Miss 00000102	Miss 00000102
w 0406 289D	Hit	Hit	Hit	Hit	Hit	Hit
r F406	Miss 00003D02	Miss 00003D02	Miss 00003D02	Miss 00003D02	Miss 00003D02	Miss 00003D02
w F406 4852	Hit	Hit	Hit	Hit	Hit	Hit
r 5D10	Miss 00001745	Miss 00001745	Miss 00001745	Miss 00001745	Miss 00001745	Miss 00001745
w 5D10 17BF	Hit	Hit	Hit	Hit	Hit	Hit
w 7D10 1000	Miss	Miss	Miss	Miss	Miss	Miss
r 7D10	Miss 00001000	Hit 00001000	Miss 00001000	Hit 00001000	Miss 00001000	Hit 00001000
r 00FC	Miss 00000040	Miss 00000040	Miss 00000040	Miss 00000040	Miss 00000040	Miss 00000040
r 10FC	Miss 00000440	Miss 00000440	Miss 00000440	Miss 00000440	Miss 00000440	Miss 00000440
r 20FC	Miss 00000840	Miss 00000840	Miss 00000840	Miss 00000840	Miss 00000840	Miss 00000840
r 30FC	Miss 00000C40	Miss 00000C40	Miss 00000C40	Miss 00000C40	Miss 00000C40	Miss 00000C40
w 20FC 1551	Hit	Hit	Miss	Miss	Miss	Miss
w 10FC 4170	Miss	Miss	Miss	Miss	Miss	Miss
w 30FC 100B	Hit	Hit	Hit	Miss	Hit	Miss
w 00FC 23C7	Miss	Miss	Miss	Miss	Miss	Miss

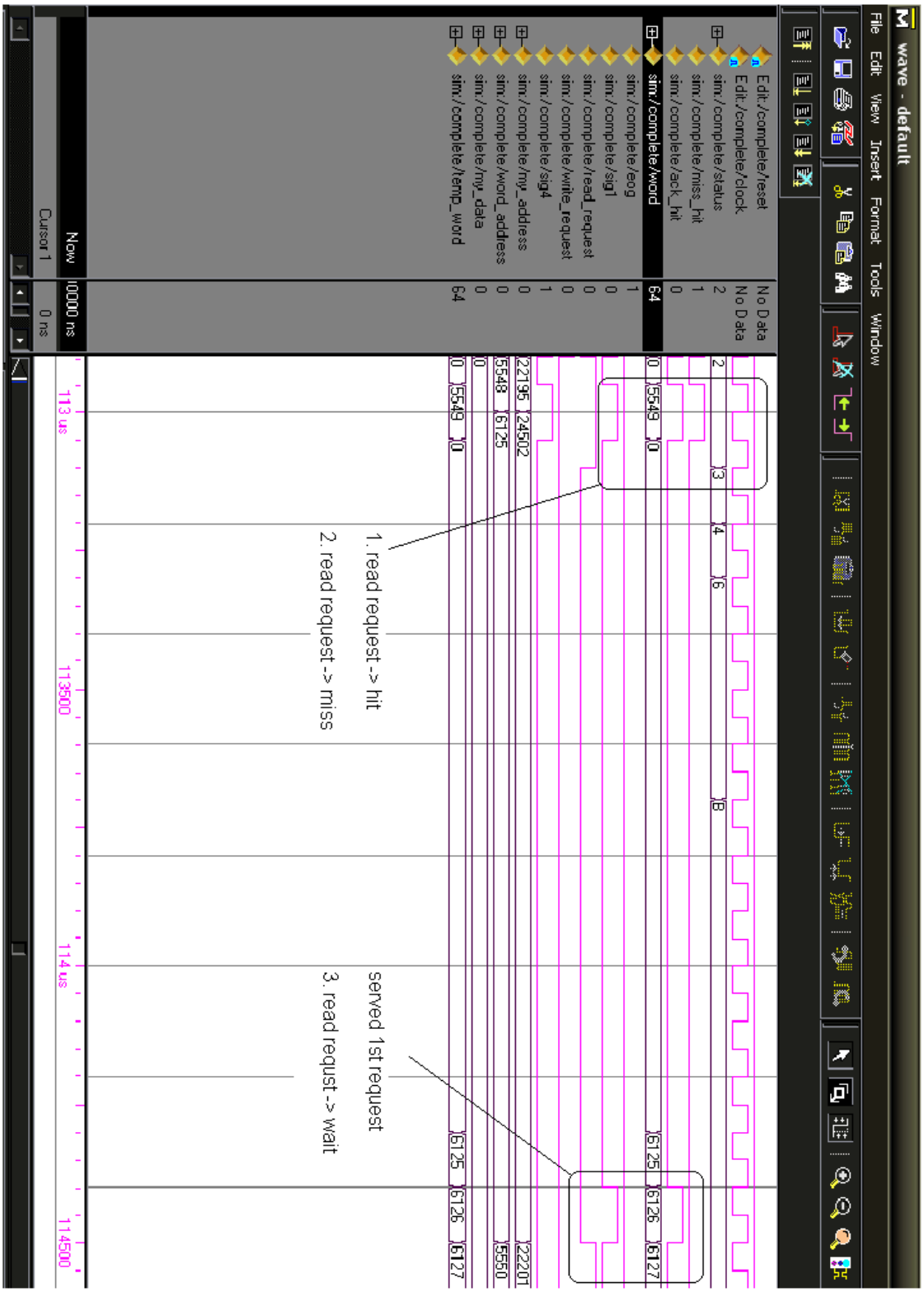
Παράμετροι	Σύστημα 7	Σύστημα 8	Σύστημα 9	Σύστημα 10	Σύστημα 11	Σύστημα 12
cahe size (KB)	32	32	256	256	8	8
block_size (words)	4	4	4	4	8	8
set_assoc	4	4	8	8	1	1
write_alloc	0	1	0	1	0	1
αιτήσεις	απαντήσεις	απαντήσεις	απαντήσεις	απαντήσεις	απαντήσεις	απαντήσεις
W 0024 4444	Miss	Miss	Miss	Miss	Miss	Miss
r 0024	Miss 00004444	Hit 00004444	Miss 00004444	Hit 00004444	Miss 00004444	Hit 00004444
W 03FD 1111	Miss	Miss	Miss	Miss	Miss	Miss
r 0029	Hit 0000000B	Hit 0000000B	Hit 0000000B	Hit 0000000B	Hit 0000000B	Hit 0000000B
r 03FD	Miss 00001111	Hit 00001111	Miss 00001111	Hit 00001111	Miss 00001111	Hit 00001111
r 03FE	Hit 00001111	Hit 00001111	Hit 00001111	Hit 00001111	Hit 00001111	Hit 00001111
W 56BA 2222	Miss	Miss	Miss	Miss	Miss	Miss
W 03F5 3333	Hit	Hit	Hit	Hit	Hit	Hit

W 03FA 5555	Hit	Hit	Hit	Hit	Hit	Hit
W 03F0 6666	Hit	Hit	Hit	Hit	Hit	Hit
r 56BA	Miss 00002222	Hit 00002222	Miss 00002222	Hit 00002222	Miss 00002222	Hit 00002222
r 56B8	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222
W D3C1 1234	Miss	Miss	Miss	Miss	Miss	Miss
r 56B3	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD
r 56B6	Hit 000015AE	Hit 000015AE	Hit 000015AE	Hit 000015AE	Hit 000015AE	Hit 000015AE
r 56B9	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222	Hit 00002222
r D3C1	Miss 00001234	Hit 00001234	Miss 00001234	Hit 00001234	Miss 00001234	Hit 00001234
W 56BA 7777	Hit	Hit	Hit	Hit	Hit	Hit
W 03F5 FFFF	Hit	Hit	Hit	Hit	Hit	Hit
w 031A DDDD	Miss	Miss	Miss	Miss	Miss	Miss
w 03F0 AAAA	Hit	Hit	Hit	Hit	Hit	Hit
r 56BA	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777
W D3C1 5678	Hit	Hit	Hit	Hit	Hit	Hit
r 56B3	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD	Hit 000015AD
r 5FB6	Miss 000017EE	Miss 000017EE	Miss 000017EE	Miss 000017EE	Miss 000017EE	Miss 000017EE
r 56B9	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777	Hit 00007777
r D3C1	Hit 00005678	Hit 00005678	Hit 00005678	Hit 00005678	Hit 00005678	Hit 00005678
W 0091 9999	Miss	Miss	Miss	Miss	Miss	Miss
W 009C 8888	Miss	Hit	Miss	Hit	Miss	Hit
r 0091	Miss 00009999	Hit 00009999	Miss 00009999	Hit 00009999	Miss 00009999	Hit 00009999
r 009C	Hit 00008888	Hit 00008888	Hit 00008888	Hit 00008888	Hit 00008888	Hit 00008888
r 1237	Miss 0000048E	Miss 0000048E	Miss 0000048E	Miss 0000048E	Miss 0000048E	Miss 0000048E
W 1232 4568	Hit	Hit	Hit	Hit	Hit	Hit
W 123D 3622	Hit	Hit	Hit	Hit	Hit	Hit
r 123D	Hit 00003622	Hit 00003622	Hit 00003622	Hit 00003622	Hit 00003622	Hit 00003622
r 123B	Hit 0000048F	Hit 0000048F	Hit 0000048F	Hit 0000048F	Hit 0000048F	Hit 0000048F
r 1232	Hit 00004568	Hit 00004568	Hit 00004568	Hit 00004568	Hit 00004568	Hit 00004568
r 1237	Hit 0000048E	Hit 0000048E	Hit 0000048E	Hit 0000048E	Hit 0000048E	Hit 0000048E
r 3555	Miss 00000D56	Miss 00000D56	Miss 00000D56	Miss 00000D56	Miss 00000D56	Miss 00000D56
r 4555	Miss 00001156	Miss 00001156	Miss 00001156	Miss 00001156	Miss 00001156	Miss 00001156
W 3555 9753	Hit	Hit	Hit	Hit	Hit	Hit
W 4555 8642	Hit	Hit	Hit	Hit	Hit	Hit
w 2435 ABCD	Miss	Miss	Miss	Miss	Miss	Miss
r 2435	Miss 0000ABCD	Hit 0000ABCD	Miss 0000ABCD	Hit 0000ABCD	Miss 0000ABCD	Hit 0000ABCD
w 4435 DCBA	Miss	Miss	Miss	Miss	Miss	Miss
r 4435	Miss 0000DCBA	Hit 0000DCBA	Miss 0000DCBA	Hit 0000DCBA	Miss 0000DCBA	Hit 0000DCBA
r 0406	Miss 00000102	Miss 00000102	Miss 00000102	Miss 00000102	Miss 00000102	Miss 00000102
W 0406 289D	Hit	Hit	Hit	Hit	Hit	Hit
r F406	Miss 00003D02	Miss 00003D02	Miss 00003D02	Miss 00003D02	Miss 00003D02	Miss 00003D02
W F406 4852	Hit	Hit	Hit	Hit	Hit	Hit
r 5D10	Miss 00001745	Miss 00001745	Miss 00001745	Miss 00001745	Miss 00001745	Miss 00001745

W 5D10 17BF	Hit	Hit	Hit	Hit	Hit	Hit
W 7D10 1000	Miss	Miss	Miss	Miss	Miss	Miss
r 7D10	Miss 00001000	Hit 00001000	Miss 00001000	Hit 00001000	Miss 00001000	Hit 00001000
r 00FC	Miss 00000040	Miss 00000040	Miss 00000040	Miss 00000040	Miss 00000040	Miss 00000040
r 10FC	Miss 00000440	Miss 00000440	Miss 00000440	Miss 00000440	Miss 00000440	Miss 00000440
r 20FC	Miss 00000840	Miss 00000840	Miss 00000840	Miss 00000840	Miss 00000840	Miss 00000840
r 30FC	Miss 00000C40	Miss 00000C40	Miss 00000C40	Miss 00000C40	Miss 00000C40	Miss 00000C40
W 20FC 1551	Hit	Hit	Hit	Hit	Hit	Hit
W 10FC 4170	Hit	Hit	Hit	Hit	Miss	Miss
W 30FC 100B	Hit	Hit	Hit	Hit	Hit	Miss
W 00FC 23C7	Hit	Hit	Hit	Hit	Miss	Miss

Μετά τα δικά μου τεστ ακολούθησαν αυτόματα τεστ με πολύ περισσότερες αιτήσεις (1000). Το σύστημα ανταποκρίθηκε σε όλες σωστά. Ίσως το πιο ολοκληρωμένο τεστ που έκανα είναι η ένωση όλων των προηγούμενων μαζί. Σε κάθε περίπτωση δεν υπάρχει το τέλειο τεστ. Απλώς εξάντλησα όσο μπορούσα κάποιες περιπτώσεις.

Παρακάτω η εικόνα από μια προσομοίωση. Στην εικόνα φαίνεται η εξυπηρέτηση μιας εύστοχης και μιας άστοχης αίτησης ανάγνωσης σε σύστημα με ετεροχρονισμένη εγγραφή .



Η πρώτη αίτηση ανάγνωσης εξυπηρετείται αμέσως αφού έχουμε ευστοχία. Η επόμενη αίτηση καθυστερεί 14 κύκλους και η λέξη – απάντηση δίνεται προς τον επεξεργαστεί πριν το σύστημα πάει στην κατάσταση ηρεμίας. Έτσι η επόμενη αίτηση πρέπει να περιμένει.

ΣΥΜΠΕΡΑΣΜΑΤΑ – ΒΕΛΤΙΩΣΕΙΣ

Η εκπόνηση της διπλωματικής εργασίας ήταν πραγματικά ένα καλό μάθημα και μια επένδυση για μελοντικές εργασίες. Αυτό γιατί ήταν η πρώτη φορά που δούλεψα ένα project μόνος μου από το μηδέν και ακολουθώντας μια πιο επιστημονική μέθοδο για την διεκπαιρέωσή του. Στην οργάνωση της εργασίας, παρόλο που θεωρητικά ήξερα ποια πρέπει να είναι, δεν κατάφερα να υπολογίσω καθυστερίσεις που προκύπτουν από προβλήματα που σε κάνουν να γυρίζω βήματα πίσω (π.χ αλλαγή εργαλίου). Έτσι έπεσα αρκετά έξω στα αρχικά μου χρονοδιαγράμματα. Ωστόσο το κέρδος από αυτήν την καθυστέριση ήταν η τριβή με κάθε λεπτομέρεια της εργασίας που μου έδειξε την διαφορά της θεωρίας από την πράξη.

Όσο αφορά το συγκεκριμένο θέμα της εργασίας διαπίστωσα ότι η υποστήριξη παραμέτρων αποτελεί μια σχεδιαστική πρόκλιση και η επίλυση του προβλήματος αυτού είναι μια ιδιαίτερα δημιουργική διαδικασία. Ιδιαίτερα δύσκολη είναι η επιλογή της καταλληλότερης μεθόδου, όταν υπάρχουν περισσότερες από μία ιδέες.

Φυσικά αν και το τελικό σύστημα δούλεψε σωστά μπορούν να γίνουν πάνω σε αυτό αρκετές βελτιώσεις ή προσθήκες. Τέτοιες είναι:

Α) Η υποστήριξη διαφορετικών αλγορίθμων αντικατάστασης των μπλοκ όπως ο αλγόριθμος LRU ή FIFO.

Β) Ευρύτερες δυνατότητες επιλογής των παραμέτρων.

Γ) Μνήμες που να παράγονται δυναμικά, ώστε να μην σπαταλούνται άδικα πόροι.

Δ) Τέλος μια επέκταση μπορεί να είναι η οργάνωση των μπλοκ σύμφωνα με την τεχνική sub-blocking. Δίνω μια γρήγορη περιγραφή της συγκεκριμένης αρχιτεκτονικής.

Αρχιτεκτονική κρυφής μνήμης sub-blocking

Το subblocking είναι μια τεχνική που επιτρέπει στα τμήματα ενός μπλοκ της κρυφής μνήμης να μαρκαριστούν ως έγκυρα ή μη έγκυρα. Σαν αποτέλεσμα, όταν το μπλοκ έχει κατανεμηθεί, και δεν έχουν όλα του τα bytes τροποποιηθεί ή διαβαστεί, τότε δεν χρειάζεται ολόκληρο το μπλοκ να κρατάει δεδομένα.

Σε αστοχία μιας εγγραφής, για παράδειγμα, ο επεξεργαστής μπορεί να μαρκάρει ολόκληρο το μπλοκ ως μη έγκυρο, εκτός από τα sub-blocks που ανταποκρίνονται στα bytes που έχουν γραφτεί μέσα στο μπλοκ. Σε μια αστοχία ανάγνωσης, μόνο τα sub-blocks που σχετίζονται με εκείνα που διαβάζονται μπορεί να χρειάζεται να φοτωθούν. Επιπλέον, αυτό μπορεί να χρησιμοποιηθεί για να μειώσει το κόστος της αποδέσμευσης των μπλοκ της κρυφής μνήμης, αφού μόνο τα sub-blocks που έχουν τροποποιηθεί χρειάζεται να πεταχτούν από τη μνήμη.

Η γενική θεωρία του subblocking είναι ότι μπορείς να κάνεις την ποσότητα της μνήμης που χρησιμοποιείς για μνήμες ετικετών μικρότερη (κάνοντας τα μπλοκ μεγαλύτερα), και μπορείς να μειώσεις τα προβλήματα που σχετίζονται με τις αστοχίες με υπερβολικά μεγάλα μπλοκ.

Βιβλιογραφία

1. John L Hennessy, David A. Patterson, Αρχιτεκτονική Υπολογιστών, Τρίτη εκδοση.
2. Stephen Brown, Zvonko Vranesic, Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL.
3. Πολυτεχνείο Κρήτης, Σημειώσεις μαθημάτων " Οργάνωση Υπολογιστών" και " Αρχιτεκτονική Υπολογιστών".