ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ
«Επιστήμη και Τεχνολογία των Υπολογιστών»

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων καθηγητής: Δ. Λιούπης

Θέμα:

Σχεδίαση και ανάπτυξη συστήματος κατανεμημένης διαμοιραζόμενης μνήμης για πολυεπεξεργαστή του ενός ολοκληρωμένου(CMP)

Αδαμίδης Ανδρέας

Πάτρα, Οκτώβριος 2008

Πίνακας περιεχομένων

Κεφάλαιο 1:.....9
Η αρχιτεκτονική SiScape.....9
1.1.Εισαγωγή.....9
1.2.Βασική δομή της αρχιτεκτονικής SiScape.....10
1.2.1. Τοπολογία.....11
1.2.2. Επεξεργαστικοί πυρήνες.....12
1.2.3. Μονάδες μνήμης.....13
1.2.4. Μοντέλο μνήμης: συνοχή και συνέπεια.....14
1.2.5. Υποστήριξη μεταφοράς μηνυμάτων.....14
1.2.6. Κατανομή φόρτου εργασίας.....15
1.3.Μοντέλο αρχιτεκτονικής SiScape.....16
1.3.1. Σύντομη παρουσίαση της βιβλιοθήκης GRLIB.....16
1.3.2. Στοχευόμενη υλοποίηση.....19
Κεφάλαιο 2:.....21
Το πρωτόκολλο AMBA και η λειτουργία των συσκευών master και slave.....21
Εισαγωγή στην προδιαγραφή AMBA.....21
Ορολογία.....21
Κατάλογος σημάτων του πρωτοκόλλου AMBA AHB.....22
2.4.Το πρωτόκολλο AMBA AHB.....25
Ένας τυπικός μικροελεγκτής βασισμένος στον δίαυλο AMBA AHB.....25
Διασύνδεση διαύλου.....25
Περίληψη λειτουργίας του AMBA AHB.....26
Βασική μεταφορά.....27
Τύποι μεταφοράς.....30
Λειτουργία ακολουθίας μεταφορών.....31
Σήματα ελέγχου.....32
Αποκωδικοποίηση διευθύνσεων.....33
Απαντήσεις συσκευών slave.....34
Δίαυλοι δεδομένων.....37
Διαιτησία (Arbitration).....39
Σήμα reset.....44
Slave διαύλου AHB.....44
Master διαύλου AHB.....45
Arbiter διαύλου AHB.....45
Αποκωδικοποιητής διαύλου AHB.....46
Κεφάλαιο 3:.....47
Το πρόβλημα συγχρονισμού των διαύλων και η λειτουργία της FIFO.....47
Εισαγωγή.....47
Η διπλή λειτουργία της FIFO.....48
Τεχνικές σχεδιασμού FIFO.....49
Βασική μορφή FIFO.....49
Μετρητής κώδικα Gray.....51
Ανίχνευση <γεμάτης> και <άδειας> κατάστασης.....52
Μετασταθερότητα (metastability) και συγχρονιστές.....55
Παραγωγή συγχρονισμένων σημάτων γεμάτης και άδειας κατάστασης.....58
Γεγονότα ενδιαφέροντος.....59
Χρονικά κρίσιμα μονοπάτια των σημάτων γεμάτης και άδειας κατάστασης.....61
Θέματα ασύγχρονης λειτουργίας.....62

Κεφάλαιο 4:....65
Λειτουργία μιας τυπικής κρυφής μνήμης....65
Εισαγωγή....65
Τοποθέτηση ενός μπλοκ στην κρυφή μνήμη....65
Εύρεση ενός μπλοκ στην κρυφή μνήμη....67
Αντικατάσταση ενός μπλοκ σε περίπτωση αστοχίας στην κρυφή μνήμη....68
Πολιτικές εγγραφής....69
Κεφάλαιο 5:....71
Υλοποίηση σχεδιασμού....71
Εισαγωγή....71
Cache.vhd....73
ahb_slv_cache.vhd....75
addr_check.vhd....78
cache_ram.vhd....80
fifo.vhd....82
ahb_mst_cache.vhd....85
buffer.vhd....87
lfsr.vhd....89
transfer_logic.vhd....90
multiplexor.vhd....93
fetch_list.vhd....94
Κεφάλαιο 6:....97
Σύνθεση και εξομοίωση....97
Σύνθεση....97
Εξομοίωση....98
Παράρτημα:....103
Κώδικας VHDL....103
Cache.vhd....103
ahb_slv_cache.vhd....112
addr_check.vhd....115
cache_ram.vhd....119
fifo.vhd....121
ahb_mst_cache.vhd....125
buffer.vhd....131
lfsr.vhd....135
transfer_logic.vhd....136
multiplexor.vhd....140
fetch_list.vhd....141
Βιβλιογραφία....143

Ευρετήριο εικόνων

Εικόνα 1: Βασική δομή της αρχιτεκτονικής SiScape....10
Εικόνα 2: α) Δισδιάστατο πλέγμα SiScape β) Μια τρισδιάστατη υλοποίηση πλάτη-με-πλάτη..11
Εικόνα 3: Άμεση επικοινωνία με οκτώ γειτονικούς επεξεργαστές....13
Εικόνα 4: Μητρώο σύνθεσης AMBA....18
Εικόνα 5: Διασύνδεση των κόμβων στο βασικό πλακίδιο SiScape....19
Εικόνα 6: Ένα τυπικό σύστημα AMBA....25
Εικόνα 7: Διασύνδεση με πολυπλέκτες....26
Εικόνα 8: Απλή μεταφορά....28
Εικόνα 9: Μεταφορά με στάδια αναμονής....29

Εικόνα 10: Πολλαπλές μεταφορές....29
Εικόνα 11: Παραδείγματα τύπων μεταφοράς....31
Εικόνα 12: Σήματα επιλογής slave....33
Εικόνα 13: Μεταφορά με απάντηση RETRY....36
Εικόνα 14: Απάντηση error....37
Εικόνα 15: Παραχώρηση ελέγχου διαύλου χωρίς στάδια αναμονής....41
Εικόνα 16: Παραχώρηση ελέγχου διαύλου με στάδια αναμονής....41
Εικόνα 17: Μεταφορά ελέγχου του διαύλου δεδομένων....42
Εικόνα 18: Μεταφορά ελέγχου του διαύλου έπειτα από μια ακολουθία μεταφορών....42
Εικόνα 19: Σήματα παραχώρησης ελέγχου σε master του διαύλου....43
Εικόνα 20: Διεπαφή συσκευής slave του διαύλου AHB....44
Εικόνα 21: Διεπαφή συσκευής master του διαύλου AHB....45
Εικόνα 22: Διεπαφή arbiter του διαύλου AHB....46
Εικόνα 23: Διεπαφή αποκωδικοποιητή του διαύλου AHB....46
Εικόνα 24: Μπλοκ διάγραμμα FIFO....50
Εικόνα 25: Μετρητής κώδικα Gray....51
Εικόνα 26: Η FIFO τείνει να γεμίσει γιατί ο δείκτης εγγραφής (wptr) βρίσκεται ένα τεταρτημόριο πίσω από τον δείκτη ανάγνωσης (rptr)....53
Εικόνα 27: Η FIFO τείνει να αδειάσει γιατί ο δείκτης ανάγνωσης (rptr) βρίσκεται ένα τεταρτημόριο πίσω από τον δείκτη εγγραφής (wptr)....54
Εικόνα 28: Κύκλωμα ανίχνευσης κατεύθυνσης FIFO....55
Εικόνα 29: Ασύγχρονα ρολόγια και αποτυχία συγχρονισμού....56
Εικόνα 30: Ένα μετασταθερό σήμα μπορεί να διαδώσει μη έγκυρες τιμές στον υπόλοιπο σχεδιασμό....57
Εικόνα 31: Συγχρονιστής δύο-flip-flop....58
Εικόνα 32: Ασύγχρονη σύγκριση δεικτών για την παραγωγή των σημάτων <γεμάτης> και <άδειας> κατάστασης....59
Εικόνα 33: Συγχρονιστές σημάτων <γεμάτης> και <άδειας> κατάστασης....60
Εικόνα 34: Χρονικά κρίσιμα μονοπάτια για την παραγωγή των σημάτων γεμάτης και άδειας κατάστασης....62
Εικόνα 35: Τα τρία είδη οργάνωσης μιας κρυφής μνήμης....66
Εικόνα 36: Τα τρία τμήματα της διεύθυνσης σε μια συνολοσυσχετιζόμενη ή άμεσα αντιστοιχιζόμενη κρυφή μνήμη....68
Εικόνα 37: Διασύνδεση των τμημάτων του σχεδιασμού και τα αρχεία που τα περιγράφουν....73
Εικόνα 38: Ένας απλός συγκριτής, όπως παράγεται από το εργαλείο σύνθεσης....80
Εικόνα 39: Οι τέσσερις μνήμες ram, όπως παράγονται από το εργαλείο σύνθεσης....82
Εικόνα 40: Λογική μετατροπής δυαδικού αριθμού σε αναπαράσταση Gray....84
Εικόνα 41: Η λογική ενός LFSR, όπως παράγεται από το εργαλείο σύνθεσης....90
Εικόνα 42: Η σειρά πολυπλεκτών που αποφασίζει ποιος δείκτης γραμμής θα οδηγηθεί στην έξοδο, όπως παράγεται από το εργαλείο σύνθεσης....94
Εικόνα 43: Η διάταξη οδήγησης της εξόδου match, όπως παράγεται από το εργαλείο σύνθεσης. .96

Ευρετήριο πινάκων

Πίνακας 1: Κατάλογος σημάτων AMBA AHB....24
Πίνακας 2: Σήματα διαιτησίας....24
Πίνακας 3: Κωδικοποίηση τύπων μεταφοράς....31
Πίνακας 4: Κωδικοποίηση μεγέθους....32
Πίνακας 5: Κωδικοποίηση απαντήσεων....35
Πίνακας 6: Ενεργές λωρίδες byte για little-endian δίαυλο δεδομένων των 32-bit....39
Πίνακας 7: Ενεργές λωρίδες byte για big-endian δίαυλο δεδομένων των 32-bit....39
Πίνακας 8: Σύγκριση τεχνικών LRU, Τυχαίας και FIFO αντικατάστασης σε αριθμό αστοχιών ανά 1000 εντολές, για διαφορετικά μεγέθη και συσχετισμούς κρυφής μνήμης....69
Πίνακας 9: Αριθμός πυλών ανά τμήμα της κρυφής μνήμης....97
Πίνακας 10: Μέγιστες συχνότητες των ρολογιών του σχεδιασμού....98

Πρόλογος

Αντικείμενο της παρούσας μεταπτυχιακής εργασίας είναι ο σχεδιασμός και η ανάπτυξη συστήματος κατανεμημένης διαμοιραζόμενης μνήμης ως τμήμα της αρχιτεκτονικής πολυεπεξεργαστικού συστήματος SiScape. Λόγω των ιδιαιτεροτήτων της αρχιτεκτονικής αυτής, το σύστημα μνήμης της κρίθηκε απαραίτητο να σχεδιαστεί και να αναπτυχθεί από το μηδέν, προκειμένου να ανταποκριθεί στις απαιτήσεις της.

Η αρχιτεκτονική SiScape αποτελεί ένα πολυεπεξεργαστικό σύστημα, με την ιδιαιτερότητα ότι δύο οποιοιδήποτε γειτονικοί επεξεργαστές έχουν τη δυνατότητα να επικοινωνήσουν άμεσα μέσω εγγραφής και ανάγνωσης στην ίδια διαμοιραζόμενη μνήμη. Η διάταξη που επιτρέπει αυτού του είδους την επικοινωνία είναι ένα δισδιάστατο πλέγμα, στο οποίο εναλλάσσονται μνήμη και επεξεργαστής. Κάθε επεξεργαστής συνδέεται φυσικά με τέσσερις μνήμες μέσω διαφορετικών level-2 caches, και αντίστοιχα κάθε μνήμη συνδέεται με τέσσερις επεξεργαστές. Έτσι, κάθε επεξεργαστής μπορεί να επικοινωνήσει πολύ γρήγορα με άλλους τρεις γειτονικούς επεξεργαστές μέσω μιας γειτονικής μνήμης και με οκτώ συνολικά γειτονικούς επεξεργαστές μέσω των τεσσάρων γειτονικών μνημών του. Η επικοινωνία μεταξύ απομακρυσμένων επεξεργαστών μπορεί να πραγματοποιηθεί με μεταφορά δεδομένων από μνήμη σε μνήμη, την οποία αναλαμβάνουν οι ενδιάμεσοι επεξεργαστές.

Όπως είναι προφανές, η διαφορετική αυτή προσέγγιση στα πολυεπεξεργαστικά συστήματα έχει ιδιαίτερες απαιτήσεις από το σύστημα μνήμης. Στα πλαίσια της εργασίας αυτής σχεδιάστηκε η λειτουργία του συστήματος μνήμης και περιγράφηκε στη γλώσσα περιγραφής υλικού VHDL η συσκευή που καθιστά δυνατή τη λειτουργία του συστήματος αυτού: η κρυφή μνήμη δευτέρου επιπέδου (level-2 cache). Η κρυφή μνήμη αυτή αναλαμβάνει την επικοινωνία μεταξύ κάθε επεξεργαστή και οποιασδήποτε από τις μνήμες του και, όπως θα περιγραφεί αναλυτικότερα παρακάτω, συνδέεται από τη μια πλευρά με τον δίαυλο του επεξεργαστή (P-bus) και από την άλλη με τον δίαυλο της μνήμης (M-bus). Ως εκ τούτου, η λειτουργία της είναι τετραπλή: α) Από την πλευρά του P-bus, η κρυφή μνήμη θα πρέπει να λειτουργεί ως slave συσκευή, να δέχεται τις αιτήσεις του επεξεργαστή και να τις εξυπηρετεί. β) Από την άλλη, στο δίαυλο της μνήμης, οφείλει να λειτουργεί ως master συσκευή, να συμμετέχει στο απαραίτητο arbitration για απόκτηση ελέγχου του διαύλου και να προωθεί τις απαραίτητες αιτήσεις στην κύρια μνήμη (slave συσκευή). γ) Η τρίτη πολύ βασική λειτουργία της κρυφής μνήμης δευτέρου επιπέδου είναι ο συγχρονισμός των δύο διαύλων. Καθώς είναι αδύνατη η ομοιόμορφη κατανομή του ίδιου ακριβώς σήματος χρονισμού σε ένα πλέγμα μεσαίου ή μεγάλου μεγέθους, οι δύο δίαυλοι είναι αναπόφευκτο να λειτουργούν με λίγο ή πολύ διαφορετικό ρολόι, ενώ η κρυφή μνήμη λειτουργεί σαν γέφυρα μεταξύ τους. δ) Τέλος, η κρυφή μνήμη θα πρέπει να πραγματοποιεί όλες τις κλασικές λειτουργίες μιας κρυφής μνήμης, όπως η εξυπηρέτηση των αιτήσεων του επεξεργαστή και η μεταφορά δεδομένων από την κύρια μνήμη στην κρυφή.

Κλείνοντας, κρίνεται απαραίτητη η απόδοση ευχαριστιών στον καθηγητή Δ. Λιούπη για τη συμπαράσταση του κατά την εκπόνηση της εργασίας αυτής.

Αδαμίδης Ανδρέας,
Οκτώβριος 2008

Κεφάλαιο 1:
Η αρχιτεκτονική SiScape

Στο κεφάλαιο αυτό γίνεται μια σύντομη παρουσίαση της αρχιτεκτονικής SiScape, ούτως ώστε να γίνουν φανεροί οι λόγοι για την ανάγκη ανάπτυξης του συστήματος μνήμης και ορισμένες από τις προκλήσεις που αυτό καλείται να αντιμετωπίσει.

1.1. Εισαγωγή

Οι σύγχρονες τεχνολογικές εξελίξεις υπόσχονται εκατομμύρια τρανζίστορ στα μελλοντικά ολοκληρωμένα, επιτρέποντας έτσι την υλοποίηση ολόκληρων συστημάτων σε ένα μόνο ολοκληρωμένο. Τα πολυεπεξεργαστικά συστήματα ενός ολοκληρωμένου (Chip MultiProcessors – CMP) με λίγους επεξεργαστές είναι ήδη πραγματικότητα, ενώ το κοντινό μέλλον υπόσχεται πολυεπεξεργαστικά συστήματα με μεγάλο αριθμό επεξεργαστικών κόμβων. Η προσέγγιση των CMPs προβάλλει ως η πιο κατάλληλη αρχιτεκτονική υπολογιστικών συστημάτων για την εκμετάλλευση του αυξημένου αριθμού τρανζίστορ, σε μια εποχή που οι αυξήσεις στη συχνότητα λειτουργίας συρρικνώνονται, ενώ οι καθυστερήσεις καλωδίων και η πολυπλοκότητα αυξάνονται. Η παράλληλη εκτέλεση, ωστόσο, απαιτεί συχνή επικοινωνία μεταξύ των επεξεργαστών. Τα περισσότερα από τα προτεινόμενα πολυεπεξεργαστικά συστήματα ενός ολοκληρωμένου στηρίζονται στη χρήση κάποιου είδους διασυνδετικού δικτύου (InterConnection Network – ICN), προκειμένου να εξασφαλίσουν τη διασύνδεση μεταξύ των επεξεργαστικών κόμβων.

Δεδομένων των χαρακτηριστικών της υλοποίησης σε ένα ολοκληρωμένο, το δίκτυο αυτό μπορεί να είναι δισδιάστατο και να παρέχει υψηλούς ρυθμούς μετάδοσης προς τη μνήμη και τους άλλους επεξεργαστές. Εφόσον συζητάμε για ένα μόνο, αλλά μεγάλο σε μέγεθος ολοκληρωμένο, το διασυνδετικό δίκτυο οφείλει να εκμεταλλεύεται τους διαθέσιμους πόρους για να παρέχει μεγάλο εύρος ζώνης, χωρίς όμως να αυξάνει την κατανάλωση ισχύος, και να ταιριάζει στην χρησιμοποιούμενη αρχιτεκτονική. Επιπρόσθετα, η δομή του διασυνδετικού δικτύου θα πρέπει να συνίσταται από μικρότερες υπομονάδες και να είναι βαθμωτή (scalable), επιδεικνύοντας σταθερές καθυστερήσεις καλωδίων, καθώς ο αριθμός των επεξεργαστικών πυρήνων αυξάνεται.

Ανεξάρτητα από την εκάστοτε αρχιτεκτονική του πολυεπεξεργαστικού συστήματος, μέχρι στιγμής κάθε κόμβος περιλαμβάνει έναν επεξεργαστικό πυρήνα και κάποιου είδους τοπική μνήμη – ένα τοπικό αρχείο καταχωρητών ή μια κρυφή μνήμη δευτέρου επιπέδου. Διασυνδετικά δίκτυα χρησιμοποιούνται για τη μεταφορά κώδικα και δεδομένων από τη μία μονάδα μνήμης στην άλλη ή από μια μονάδα μνήμης σε μια διεπαφή μονάδας I/O ή εξωτερικής DRAM.

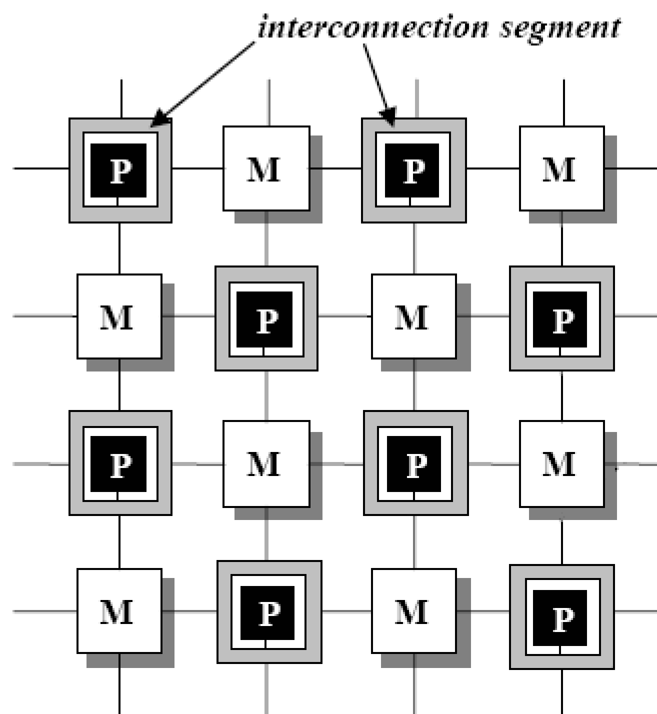
Η κεντρική ιδέα πίσω από την αρχιτεκτονική SiScape είναι η εξής. Εάν η αρχιτεκτονική του πολυεπεξεργαστικού συστήματος έχει σχεδιαστεί κατάλληλα και αν ένας επεξεργαστικός πυρήνας έχει πρόσβαση σε περισσότερες από μία τοπικές μνήμες, τότε η μεταφορά δεδομένων ανάμεσα σε αυτές τις μονάδες μνημών μπορεί να πραγματοποιείται από τον ίδιο τον επεξεργαστή. Η όλη ιδέα εξαρτάται από τη δομή της αρχιτεκτονικής, η οποία, δεδομένου ότι ο επεξεργαστής έχει επαρκή επεξεργαστική ισχύ (superscalar ή/και SMT), μπορεί να στηρίζεται στις διεπαφές επεξεργαστών/μνημών για τη μεταφορά κώδικα και δεδομένων από τη μία μονάδα μνήμης στην άλλη. Η προσέγγιση αυτή όχι μόνο απλοποιεί την αρχιτεκτονική του πολυεπεξεργαστικού

συστήματος, αλλά εξασφαλίζει ακόμα μεγαλύτερο ρυθμό μετάδοσης δεδομένων τοπικά σε σχέση με τα δίκτυα που χρησιμοποιούνται μέχρι στιγμής στις παράλληλες αρχιτεκτονικές υπολογιστών.

Η ιδέα της χρήσης διαμοιραζόμενης μνήμης στη θέση ενός εξειδικευμένου διασυνδετικού δικτύου είχε μελετηθεί και παλαιότερα στο Πανεπιστήμιο Πατρών, ωστόσο η μελέτη αυτή είχε γίνει για σύστημα πολλαπλών ολοκληρωμένων, όπου μνήμες και επεξεργαστές ήταν διακριτά στοιχεία. Με τη σημερινή τεχνολογία, όπου τα πάντα μπορούν να χωρέσουν σε ένα μόνο ολοκληρωμένο, το μοντέλο αυτό δείχνει ιδιαίτερα ταιριαστό. Εκμεταλλευόμενοι τη δισδιάστατη φύση της επιφάνειας του ολοκληρωμένου, οι επεξεργαστές μπορούν πολύ εύκολα να συνδεθούν με τέσσερις γειτονικές μνήμες, παρέχοντας έτσι τη δομή για μεταφορά δεδομένων μέσω διαμοιραζόμενης μνήμης. Η παρουσία μιας αρχιτεκτονικής πλακιδίων (tiled architecture) στην οποία επεξεργαστές και μνήμες εναλλάσσονται καθιστά σε μεγάλο βαθμό περιττή τη χρήση ενός χωριστού διασυνδετικού δικτύου. Θα πρέπει να σημειωθεί ωστόσο πως η ίδια η αρχιτεκτονική δεν αποκλείει την παρουσία ενός τέτοιου διασυνδετικού δικτύου.

1.2. Βασική δομή της αρχιτεκτονικής SiScape

Η βασική δομή της αρχιτεκτονικής SiScape φαίνεται στην Εικόνα 1.



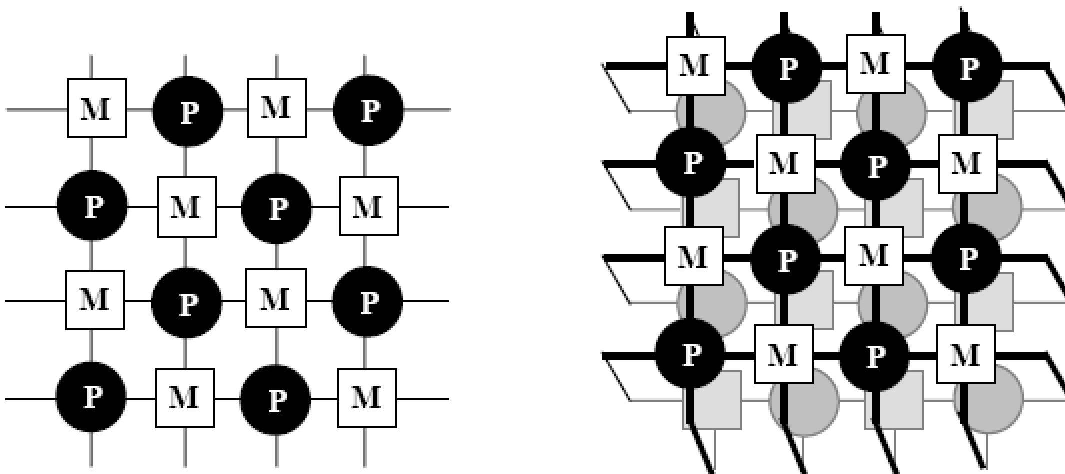
Εικόνα 1: Βασική δομή της αρχιτεκτονικής SiScape

Κάθε επεξεργαστικός πυρήνας συνδέεται με τέσσερις γειτονικές μνήμες (βόρεια, νότια, ανατολικά και δυτικά). Αντίστοιχα, κάθε μονάδα μνήμης είναι συνδεμένη με τέσσερις επεξεργαστές. Αυτό το βασικό σχήμα διασύνδεσης προσφέρει αυξημένη συνδεσιμότητα, καθώς κάθε πλακίδιο διασυνδέεται με τα τέσσερα γειτονικά του πλακίδια. Οι επεξεργαστές και οι μονάδες μνήμης που βρίσκονται στα άκρα περιλαμβάνουν διασυνδέσεις προς το εξωτερικό του

ολοκληρωμένου, οι οποίες μπορούν να χρησιμοποιηθούν είτε για επικοινωνία με εξωτερική μνήμη είτε για διασύνδεση με περισσότερα ίδια ολοκληρωμένα σε οριζόντια ή κάθετη διάταξη.

1.2.1. Τοπολογία

Η δομή της αρχιτεκτονικής, όπως φαίνεται και από την Εικόνα 2α, είναι παρόμοια με αυτή ενός δισδιάστατου πλέγματος. Η διαφορά έγκειται στο γεγονός ότι οι κόμβοι της αρχιτεκτονικής SiScape αποτελούνται εναλλάξ από έναν επεξεργαστή ή μια μονάδα μνήμης, σε αντίθεση με ένα δισδιάστατο πλέγμα με διασυνδετικό δίκτυο, στο οποίο κάθε κόμβος περιλαμβάνει ένα επεξεργαστικό στοιχείο και μια μονάδα μνήμης.



Εικόνα 2: α) Δισδιάστατο πλέγμα SiScape β) Μια τρισδιάστατη υλοποίηση πλάτη-με-πλάτη

Η δρομολόγηση δεν περιορίζεται σε δρομολόγηση από το x στο y, αλλά μπορεί να ακολουθήσει οποιοδήποτε μονοπάτι στο οποίο κάθε μετακίνηση προς οποιαδήποτε κατεύθυνση μας φέρνει πιο κοντά στον προορισμό. Ανεξάρτητα από ποιο μονοπάτι θα επιλεχθεί, ο αριθμός των ενδιάμεσων μνημών που προσπερνώνται παραμένει ο ίδιος.

Ένας γενικός τύπος που περιγράφει την *απόσταση* μεταξύ δύο επεξεργαστών (ο όρος απόσταση αναφέρεται στον αριθμό των μονάδων μνήμης που πρέπει να χρησιμοποιηθούν για την επικοινωνία δύο επεξεργαστών) φαίνεται παρακάτω:

M = (διαφορά_στήλης + διαφορά_γραμμής)/2 , όπου M ο αριθμός των μνημών που προσπερνώνται.

Η τοπολογία της αρχιτεκτονικής SiScape ως δισδιάστατο πλέγμα είναι η ευκολότερη στην υλοποίησή της, παρουσιάζει όμως μια ασυμμετρία στα άκρα. Οι επεξεργαστές δηλαδή που βρίσκονται στα άκρα δεν συνδέονται με τέσσερις μονάδες μνήμης. Όπως αναφέρθηκε παραπάνω, οι ακριανοί κόμβοι σε αυτήν την περίπτωση αποτελούν τη διεπαφή προς τον εξωτερικό κόσμο (είτε για επέκταση είτε για σύνδεση με εξωτερική μνήμη). Εναλλακτικά, θα μπορούσε να δημιουργηθεί μια δισδιάστατη τοπολογία torus εάν ενώναμε τις συνδέσεις των άκρων με τους κόμβους της άλλης πλευράς, χρησιμοποιώντας στρώματα μετάλλων υψηλότερου επιπέδου για τις

συνδέσεις. Φυσικά, η ασυμμετρία σε αυτήν την περίπτωση θα μεταφραζόταν σε ασυμμετρία *καθυστέρησης* λόγω της προσπέλασης μνημών μέσω καλωδίων μεγάλου μήκους.

Μια πιο ενδιαφέρουσα προσέγγιση είναι η χρήση τρισδιάστατης τεχνολογίας σιλικόνης για να τοποθετηθούν δύο dies της αρχιτεκτονικής SiScape είτε πλάτη-με-πλάτη είτε πρόσωπο-με-πρόσωπο. Στην τοποθέτηση πλάτη-με-πλάτη, που είναι πιο συνηθισμένη, για κάθε επεξεργαστή σε ένα επίπεδο υπάρχει μια μονάδα μνήμης στο άλλο (Εικόνα 2β). Συνδέσεις υπάρχουν μόνο στην περιφέρεια των dies και συνδέουν τους ακριανούς επεξεργαστές του ενός επιπέδου με τις ακριανές μνήμες του άλλου. Στην περίπτωση αυτή η τοπολογία είναι τέτοια ώστε κάθε επεξεργαστής, ανεξάρτητα από τη θέση του, συνδέεται με τέσσερις μονάδες μνήμης, εκτός από τους τέσσερις συνολικά γωνιακούς επεξεργαστές των δύο επιπέδων, οι οποίοι συνδέονται με τρεις μόνο ξεχωριστές μνήμες ο καθένας.

Ο γενικός τύπος για την επικοινωνία μεταξύ επεξεργαστών διαφορετικού επιπέδου είναι:

*Μ = (διαφορά_στήλης*_min + *διαφορά_γραμμής*_min*)/2* , όπου Μ ο αριθμός των μνημών που προσπερνώνται, ενώ οι διαφορές γραμμών και στηλών είναι οι ελάχιστες διαφορές για μετακίνηση από των κόμβο πηγής στον κόμβο προορισμού. Ας σημειωθεί ότι το άθροισμα των ελάχιστων διαφορών γραμμών και στηλών ανάμεσα σε δύο επεξεργαστές είναι πάντα ζυγός αριθμός, τόσο στην τοπολογία πλέγματος όσο και στην τοπολογία torus.

Στην τοποθέτηση πρόσωπο-με-πρόσωπο, η οποία είναι εφικτή με τη σημερινή τεχνολογία αλλά ακριβή και δύσκολη, μπορούν να συνδεθούν δύο dies επεξεργαστής με επεξεργαστή και μνήμη με μνήμη. Αυτό δίνει τη δυνατότητα κάθε επεξεργαστής να έχει πρόσβαση έως και σε οκτώ μνήμες (τέσσερις στο επίπεδό του και τέσσερις στο άλλο επίπεδο) με χρήση κάθετων vias. Παρόλο που αυτή η προσέγγιση αποτελεί μια ενδιαφέρουσα τοπολογία, δεν έχει εξεταστεί παραπέρα.

1.2.2. Επεξεργαστικοί πυρήνες

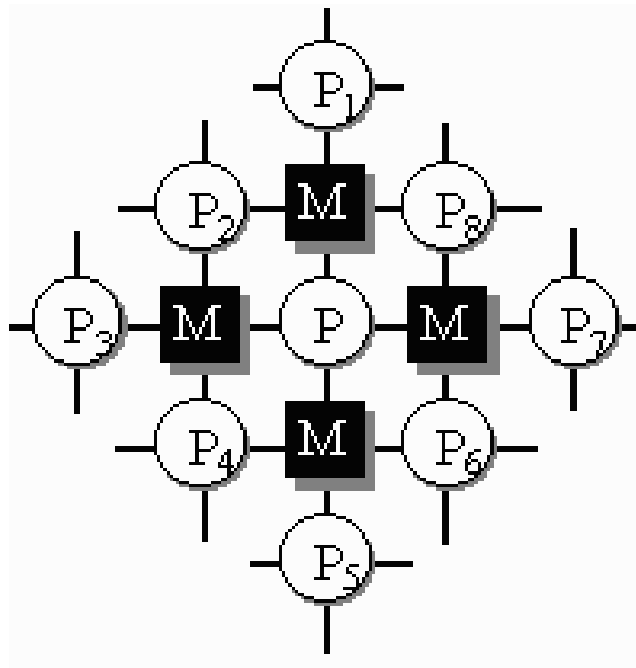
Οι επεξεργαστικοί πυρήνες μπορούν να εκτελούν πολλαπλά νήματα (threads) παράλληλα, πάρ' όλο που αυτό δεν είναι υποχρεωτικό (όπως για παράδειγμα σε ένα σύστημα σχεδιασμένο να τρέχει λογισμικό συγκεκριμένου σκοπού, το οποίο δεν απαιτεί παράλληλη εκτέλεση νημάτων ή σχετικές υψηλού-επιπέδου υπηρεσίες λειτουργικού συστήματος). Αυτή η δυνατότητα επιτρέπει την ανάθεση περισσότερων από ένα νημάτων σε κάθε πυρήνα, όταν τα νήματα που δημιουργεί κάποια εφαρμογή είναι περισσότερα από τους επεξεργαστικούς πυρήνες. Επιπρόσθετα, τα νήματα συστήματος που εφαρμόζουν το μοντέλο μνήμης που απαιτεί η εφαρμογή μπορούν να τρέχουν παράλληλα με τα νήματα της εφαρμογής, και να φροντίζουν για την απομακρυσμένη (multi-hop) επικοινωνία με μνήμες, τον συγχρονισμό και τη συνέπεια μεταξύ των διαμοιραζόμενων δεδομένων. Το μοντέλο του επεξεργαστή βασίζεται σε μια απλή εν-σειρά (in-order) αρχιτεκτονική, η οποία έχει επεκταθεί για να υποστηρίζει πολλαπλά νήματα. Οι επεκτάσεις αυτές επηρεάζουν μόνο τα στάδια φόρτωσης (fetch) και έκδοσης (issue) και απαιτούν ξεχωριστή αρχιτεκτονική κατάσταση (αρχείο καταχωρητών και bits κατάστασης) για κάθε νήμα που υποστηρίζεται από το υλικό. Οι πυρήνες μπορούν επίσης να υποστηρίξουν μικρές και γρήγορες πρώτου-επιπέδου κρυφές μνήμες εντολών και δεδομένων, ώστε να ελαφρύνουν τον φόρτο εργασίας των τοπικών μνημών (κύρια μνήμη ή/και κρυφές μνήμες δευτέρου επιπέδου).

1.2.3. Μονάδες μνήμης

Οι μονάδες μνήμης παίζουν θεμελιώδη ρόλο στην αρχιτεκτονική SiScape. Όχι μόνο παρέχουν τη δυνατότητα τοπικής αποθήκευσης, αλλά αποτελούν επίσης το μέσο επικοινωνίας μεταξύ των επεξεργαστών. Αντίθετα με ένα CMP που χρησιμοποιεί διασυνδετικό δίκτυο, και όπου κάθε κόμβος έχει τη δική του τοπική μνήμη, η αρχιτεκτονική SiScape έχει διαμοιραζόμενες τοπικές μνήμες, σε καθεμία από τις οποίες έχουν πρόσβαση τέσσερις επεξεργαστές. Ας σημειωθεί ωστόσο ότι οι διαμοιραζόμενες μονάδες μνήμης της SiScape και οι τοπικές μνήμες ενός CMP με διασυνδετικό δίκτυο έχουν τις ίδιες απαιτήσεις σε εύρος ζώνης. Παρ' όλο που οι μονάδες μνήμης της SiScape διαμοιράζονται ανάμεσα σε τέσσερις επεξεργαστές, κατά μέσο όρο λαμβάνουν μόνο το ένα τέταρτο από τις προσπελάσεις κάθε επεξεργαστή (τα υπόλοιπα τρία τέταρτα κατανέμονται ανάμεσα στις υπόλοιπες τρεις μονάδες μνήμης του επεξεργαστή).

Εφόσον οι προσπελάσεις στις διαμοιραζόμενες μονάδες μνήμης δεν τετραπλασιάζονται στην αρχιτεκτονική SiScape, αλλά είναι περίπου ίσες με τις συνολικές προσπελάσεις ενός μόνο επεξεργαστή, οι μονάδες μνήμες δεν έχουν ιδιαίτερες απαιτήσεις στην υλοποίησή τους. Δεν υπάρχει λόγος να χρησιμοποιηθούν μνήμες πολλαπλών ports, πράγμα που θα μείωνε αδικαιολόγητα την πυκνότητά τους (και συνεπώς θα αύξανε το μέγεθός τους). Η πρόσβαση των τεσσάρων επεξεργαστών σε μια διαμοιραζόμενη μνήμη ρυθμίζεται από μια κρυφή μνήμη δεύτερου επιπέδου, όπως περιγράφεται παρακάτω.

Η φιλοσοφία της προσέγγισης αυτής είναι να παρέχεται η γρηγορότερη δυνατή επικοινωνία από-γείτονα-σε-γείτονα, βάσει του συλλογισμού ότι αυτή είναι η πιο συνηθισμένη μορφή επικοινωνίας στις στοχευόμενες εφαρμογές. Στην αρχιτεκτονική SiScape, η επικοινωνία μεταξύ δύο γειτονικών επεξεργαστών συνίσταται σε μια απλή εγγραφή και ανάγνωση μιας διαμοιραζόμενης μεταβλητής σε κάποια φυσικά διαμοιραζόμενη μονάδα μνήμης. Με τον τρόπο αυτό, κάθε επεξεργαστής μπορεί να επικοινωνήσει άμεσα με άλλους οκτώ επεξεργαστές (Εικόνα 2).



Εικόνα 3: Άμεση επικοινωνία με οκτώ γειτονικούς επεξεργαστές

Με εξαίρεση την επικοινωνία μέσω καταχωρητών ή την άμεση επικοινωνία (μέσω εντολών για παράδειγμα), αυτός είναι ο γρηγορότερος τρόπος επικοινωνίας μέσω διαμοιραζόμενης μνήμης. Αντίθετα, σε ένα πολυεπεξεργαστικό σύστημα με διασυνδετικό δίκτυο, η επικοινωνία μεταξύ δύο γειτόνων απαιτεί την εγγραφή μιας διαμοιραζόμενης μεταβλητής στην τοπική μνήμη του επεξεργαστή-πηγή, τη μεταφορά της τιμής της μέσω του δικτύου (η οποία περιλαμβάνει διεπαφές δικτύου, switches και links), την εγγραφή της τιμής στην τοπική μνήμη του επεξεργαστή-προορισμού και την ανάγνωσή της από εκεί. Ωστόσο, ενώ η επικοινωνία μεταξύ απομακρυσμένων επεξεργαστών σε έναν πολυεπεξεργαστή με διασυνδετικό δίκτυο σημαίνει απλώς μια λίγο μεγαλύτερη παραμονή στο δίκτυο, στην SiScape τα δεδομένα πρέπει να αντιγραφούν μέσω των ενδιάμεσων μνημών υπό τον έλεγχο του λογισμικού συστήματος.

1.2.4. Μοντέλο μνήμης: συνοχή και συνέπεια

Η αρχιτεκτονική SiScape υποστηρίζει ένα μοντέλο εικονικής διαμοιραζόμενης μνήμης. Η διαμοιραζόμενη μνήμη αντιστοιχίζεται στις τοπικές μνήμες με χρήση σελίδων μεταβλητού μεγέθους. Η συνέπεια των MMU και TLB των πυρήνων εξασφαλίζεται από το λειτουργικό σύστημα. Ανάλογα με το πόσο δυναμική είναι η κατανομή μιας εφαρμογής πάνω στην αρχιτεκτονική (δηλαδή η κατανομή των δεδομένων στις μνήμες και των νημάτων στους επεξεργαστές), η επιβάρυνση για την ανανέωση των πληροφοριών των MMU και TLB μπορεί να κυμαίνεται από αμελητέα έως σημαντική.

Όπως αναφέρθηκε, κάθε επεξεργαστής μπορεί να προσπελάσει άμεσα ολόκληρο το τμήμα της διαμοιραζόμενης μνήμης που αντιστοιχεί στις τέσσερις γειτονικές του μονάδες μνήμης, κι έτσι είναι ικανός να επικοινωνήσει άμεσα (μέσω κοινών μνημών) με οκτώ γειτονικούς επεξεργαστές. Ωστόσο, προκειμένου να προσπελαθεί διαμοιραζόμενη μνήμη που αντιστοιχεί σε κάποια απομακρυσμένη μονάδα μνήμης, θα πρέπει να χρησιμοποιηθεί ένα πρωτόκολλο συνοχής, το οποίο είναι συγκεκριμένο για κάθε εφαρμογή. Το πρωτόκολλο αυτό υλοποιείται σε λογισμικό και μπορεί να είναι όσο απλό ή όσο σύνθετο απαιτεί η εκάστοτε εφαρμογή. Για παράδειγμα, το πρωτόκολλο αυτό θα μπορούσε να δημιουργήσει ένα αντίγραφο της απομακρυσμένης σελίδας διαμοιραζόμενης μνήμης (και να εξασφαλίζει τη συνέπειά του χρησιμοποιώντας έναν συνδυασμό μεταφοράς-μηνυμάτων και ασθενών πρωτοκόλλων συνέπειας, όπως στο [4]), ή να την ξανα-αναθέσει και να τη μεταφέρει σε κάποια τοπική μνήμη (να αλλάξει δηλαδή τον «οικείο» της κόμβο). Σε οποιαδήποτε περίπτωση, οι ενδιάμεσες αντιγραφές από μνήμη σε μνήμη πραγματοποιούνται διαφανώς από το πρωτόκολλο, έως ότου η σελίδα φτάσει στη μνήμη-προορισμό.

Μεμονωμένα, μια μονάδα μνήμης και οι τέσσερις επεξεργαστές που συνδέονται σε αυτή υποστηρίζουν σειριακή συνέπεια. Ωστόσο, το μοντέλο συνέπειας μνήμης για ολόκληρο το σύστημα που απαιτείται από την εφαρμογή ορίζεται και υλοποιείται σε υψηλότερο επίπεδο με τη συμβολή της εφαρμογής και του λογισμικού συστήματος.

Η αρχιτεκτονική SiScape επιζητά απλότητα στην συνηθισμένη περίπτωση διαμοιραζόμενων δεδομένων που είναι τοποθετημένα σε μια τοπική μνήμη, ενώ αναθέτει όλη τη σύνθετη λειτουργικότητα για απομακρυσμένη επικοινωνία στο λογισμικό.

1.2.5. Υποστήριξη μεταφοράς μηνυμάτων

Παρά το γεγονός ότι η SiScape είναι αρχιτεκτονική εικονικής διαμοιραζόμενης μνήμης, υποστηρίζει επίσης μεταφορά μηνυμάτων μέσω ουρών μηνυμάτων στις μονάδες μνήμης. Τα μηνύματα που προορίζονται για απομακρυσμένους κόμβους (είτε επεξεργαστές είτε μνήμες) προωθούνται προς τον προορισμό τους μέσω ενδιάμεσων μνημών. Για κάθε επεξεργαστή ορίζεται μια ουρά μηνυμάτων σε καθεμία από τις τέσσερις γειτονικές του μνήμες. Οι οκτώ γειτονικοί του επεξεργαστές αποθηκεύουν σε αυτές τις ουρές μηνύματα που έχουν ως παραλήπτη τον συγκεκριμένο επεξεργαστή. Κατά τη λειτουργία του επεξεργαστή, ένα νήμα χειρισμού μηνυμάτων εξυπηρετεί τα εκκρεμή μηνύματα.

Τα μηνύματα προωθούνται προς τον προορισμό τους μέσω *αποθήκευσης&προώθησης* από μνήμη σε μνήμη. Κάθε μήνυμα περιλαμβάνει μια επικεφαλίδα η οποία περιγράφει τον τύπο του μηνύματος, τη θέση του επεξεργαστή-προορισμού στο πλέγμα, καθώς και την πηγή του (αν αυτό είναι απαραίτητο). Η δρομολόγηση των μηνυμάτων μέσα από το πλέγμα επεξεργαστών είναι προκαθορισμένη. Θα μπορούσε επίσης να εφαρμοστεί δυναμική δρομολόγηση, αλλά προς το παρόν δεν έχει μελετηθεί. Μόλις ένας επεξεργαστής παραλαμβάνει ένα μήνυμα, είτε το προωθεί προς τον προορισμό του, είτε το καταναλώνει ο ίδιος.

Ο runtime πυρήνας της αρχιτεκτονικής SiScape προσπαθεί να ελαχιστοποιήσει την καθυστέρηση μεταφοράς μηνυμάτων, είτε αποφεύγοντάς την είτε αποκρύπτωντάς την. Όταν τα δεδομένα και οι διεργασίες κατανέμονται στους πόρους του συστήματος, το πλεονέκτημα της φυσικής επικοινωνίας μέσω κοινών γειτονικών μνημών χρησιμοποιείται στον μέγιστο δυνατό βαθμό. Έχει δειχθεί στα [7] και [5] ότι η προσέγγιση αυτή ανταποκρίνεται καλά σε ένα μεγάλο εύρος παράλληλων εφαρμογών και η προσθήκη ενός γενικού δικτύου επικοινωνίας δε βελτιώνει σημαντικά την απόδοση. Η κατηγορία εφαρμογών τις οποίες στοχεύει η αρχιτεκτονική περιλαμβάνει εφαρμογές που προσφέρονται για μια τέτοια κατανομή, και περιλαμβάνει πολλές ενσωματωμένες (embedded) εφαρμογές, ειδικά επεξεργασία πολυμέσων/ροών.

1.2.6. Κατανομή φόρτου εργασίας

Η ανάθεση νημάτων με διαμοιραζόμενους πόρους σε γειτονικούς επεξεργαστές είναι υπευθυνότητα του αλγορίθμου κατανομής (distribution algorithm). Η αποτελεσματική κατανομή των διεργασιών δεν είναι πάντοτε εφικτή. Ορισμένες φορές ο αλγόριθμος κατανομής δεν έχει αρκετές πληροφορίες ώστε να πραγματοποιήσει μια βέλτιστη κατανομή φόρτου, ενώ άλλες φορές η παράλληλη εφαρμογή δεν επιτρέπει την κατανομή του συνόλου των δεδομένων της σε διαφορετικές μνήμες. Στην αρχική θεώρηση της αρχιτεκτονικής, η ανάθεση νημάτων και η κατανομή δεδομένων πραγματοποιείται στατικά χρησιμοποιώντας πληροφορίες profiling.

Η στατική τοποθέτηση κώδικα και δεδομένων είναι κατάλληλη για εφαρμογές επεξεργασίας ροών, μια κατηγορία προγραμμάτων η οποία αποτελεί πρωτεύοντα στόχο των πολυεπεξεργαστικών συστημάτων SiScape. Εκ των πραγμάτων, μια εφαρμογή επεξεργασίας ροών χρησιμοποιεί το μοντέλο παραγωγού-καταναλωτή με ουρές δεδομένων που διασυνδέουν τα διάφορα επεξεργαστικά τμήματα. Η στατική κατανομή των εφαρμογών επεξεργασίας ροών θα είναι ιδιαίτερα ευνοϊκή όσον αφορά την τοπικότητα, καθώς γειτονικοί επεξεργαστές μπορούν να αναλαμβάνουν ζεύγη διεργασιών παραγωγού-καταναλωτή, ενώ οι ουρές δεδομένων θα τοποθετούνται σε κοινές τους μνήμες. Νέα εργαλεία και γλώσσες όπως η StreamIt φαίνονται κατάλληλες για αυτού του είδους την προγραμματιστική προσέγγιση.

Από την άλλη πλευρά, εφαρμογές που ακολουθούν το παράδειγμα της διαμοιραζόμενης μνήμης μπορούν να κατανεμηθούν δυναμικά στο σύστημα SiScape με τη βοήθεια του πυρήνα. Το πόσο αποτελεσματική θα είναι αυτή η δυναμική κατανομή εξαρτάται σε μεγάλο βαθμό από τον τρόπο προσπέλασης δεδομένων και επικοινωνίας διεργασιών που απαιτεί η κάθε εφαρμογή, αλλά

προηγούμενη μελέτη [5] έχει δείξει ότι η αποτελεσματική δυναμική κατανομή είναι εφικτή για ένα μεγάλο εύρος παράλληλων εφαρμογών.

Όπως σε όλα τα πολυεπεξεργαστικά συστήματα ενός ολοκληρωμένου με φυσικά κατανεμημένη μνήμη, οι απομακρυσμένες προσπελάσεις στην αρχιτεκτονική SiScape είναι μη αποδοτικές, σε σχέση με τις προσπελάσεις σε γειτονικές μνήμες. Το γεγονός αυτό είναι ιδιαίτερα εμφανές στην SiScape, όπου οι απομακρυσμένες προσπελάσεις σε μη-γειτονικές μνήμες δεν υποστηρίζονται άμεσα από το υλικό. Όλες οι αιτήσεις για απομακρυσμένες προσπελάσεις ανατίθενται και εξυπηρετούνται από νήματα συστήματος. Το κόστος των απομακρυσμένων προσπελάσεων θα πρέπει συνεπώς να αποφευχθεί, να αποκρυφθεί ή να αναπληρωθεί με κάποιον άλλον τρόπο προς το συνολικό σύστημα.

1.3. Μοντέλο αρχιτεκτονικής SiScape

Στο Πανεπιστήμιο Πατρών γίνεται μια προσπάθεια περιγραφής ενός μοντέλου της αρχιτεκτονικής SiScape στη γλώσσα περιγραφής υλικού VHDL (IEEE 1164) και συγκεκριμένα στο συνθέσιμο υποσύνολο της γλώσσας, ούτως ώστε ο σχεδιασμός να μπορεί να μεταφραστεί σε πραγματικό υλικό. Η χρήση της γλώσσας αυτής επιτρέπει λεπτομερή προσομοίωση του σχεδιασμού με ακριβείς χρονικούς περιορισμούς, καθώς και την υιοθέτηση διαθέσιμων, δοκιμασμένων open-source εξαρτημάτων. Ένα ακόμα σημαντικό πλεονέκτημα της επιλογής αυτής είναι εύκολης υλοποίησης του τελικού σχεδιασμού σε κάποιο πραγματικό σύστημα που περιέχει FPGA, πράγμα που επιτρέπει τη γρήγορη απόκτηση μετρήσεων, βασισμένων σε πρακτικά workloads. Η περιγραφή του σχεδιασμού βασίζεται στη χρήση open-source μοντέλων, κυρίως από τη βιβλιοθήκη IP cores GRLIB της Gaisler Research (http://www.gaisler.com).

1.3.1. Σύντομη παρουσίαση της βιβλιοθήκης GRLIB

Η βιβλιοθήκη IP GRLIB είναι ένα σύνολο επαναχρησιμοποιήσιμων IP cores, σχεδιασμένων για την ανάπτυξη *συστημάτων-σε-ολοκληρωμένο* (system-on-chip – SOC). Οι πυρήνες IP βασίζονται στη χρήση ενός κοινού on-chip διαύλου, και χρησιμοποιούν μια δεδομένη μέθοδο για προσομοίωση και σύνθεση. Η βιβλιοθήκη είναι ανεξάρτητη από τους παροχείς IP, υποστηρίζει διαφορετικά εργαλεία CAD και στοχεύει σε διάφορες τεχνολογίες. Χρησιμοποιεί επίσης μια ειδική μέθοδο plug&play για να ρυθμίζει και να διασυνδέει τους πυρήνες IP χωρίς την ανάγκη τροποποίησης γενικευμένων πόρων.

Οργάνωση βιβλιοθήκης

Η GRLIB είναι οργανωμένη σε βιβλιοθήκες VHDL, καθώς μια ξεχωριστή ονομασία βιβλιοθήκης ανατίθεται σε κάθε σημαντική IP (ή παροχέα IP). Η χρήση χωριστών βιβλιοθηκών εξασφαλίζει την αποφυγή συγκρούσεων στις ονομασίες των πυρήνων IP και αποκρύπτει μη απαραίτητες λεπτομέρειες υλοποίησης από τον τελικό χρήστη. Κάθε βιβλιοθήκη VHDL περιέχει τυπικά έναν αριθμό πακέτων, τα οποία περιγράφουν τους εξαγόμενους IP πυρήνες και των τύπο των διεπαφών τους. Τα scripts για την προσομοίωση και τη σύνθεση δημιουργούνται αυτόματα από ένα γενικό makefile. Η προσθήκη και η αφαίρεση βιβλιοθηκών και πακέτων πραγματοποιείται χωρίς την τροποποίηση καθολικών αρχείων, εξασφαλίζοντας έτσι ότι η τροποποίηση της βιβλιοθήκης ενός παροχέα δεν επηρεάζει άλλους παροχείς.

Δίαυλος on-chip

Η βιβλιοθήκη GRLIB έχει σχεδιαστεί ώστε να είναι «διαυλο-κεντρική», υποθέτει δηλαδή ότι οι περισσότεροι πυρήνες IP θα διασυνδέονται μέσω ενός διαύλου on-chip. Ως ο κεντρικός δίαυλος on-chip έχει επιλεγεί ο δίαυλος AMBA-2.0 AHB/APB, λόγω της ευρείας διάδοσής τους στο εμπόριο (επεξεργαστές ARM) και επειδή είναι καλά τεκμηριωμένος και μπορεί να χρησιμοποιηθεί ελεύθερα χωρίς την ανάγκη άδειας.

Αποκωδικοποίηση κατανεμημένων διευθύνσεων

Η προσθήκη ενός πυρήνα IP στον δίαυλο AHB δε γίνεται δυστυχώς συνδέοντας απλά τα σήματα του διαύλου. Η αποκωδικοποίηση διευθύνσεων στο AHB είναι κεντρικοποιημένη, οπότε πρέπει να τροποποιείται ένας αποκωδικοποιητής διαμοιραζόμενων διευθύνσεων και ένας πολυπλέκτης διαύλου κάθε φορά που ένας πυρήνας IP προστίθεται ή αφαιρείται. Προκειμένου να αποφευχθούν οι εξαρτήσεις από καθολικούς πόρους, η αποκωδικοποίηση κατανεμημένων διευθύνσεων έχει προστεθεί στους πυρήνες GRLIB και τους ελεγκτές AMBA AHB/APB.

Χειρισμός interrupts

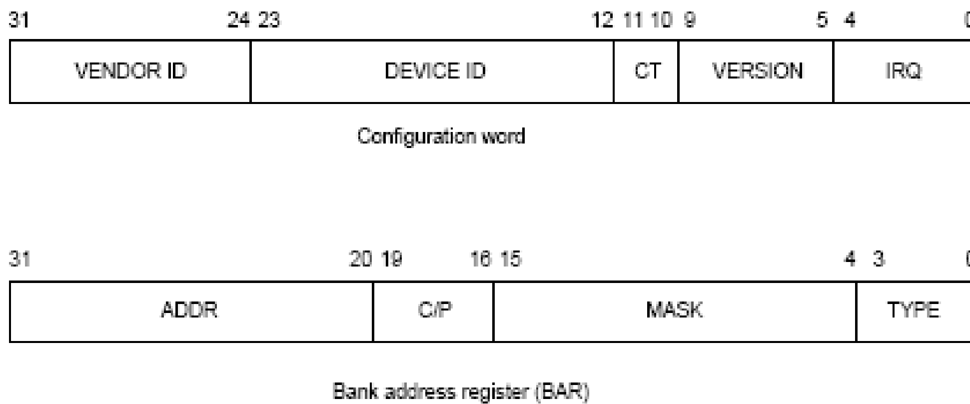
Η βιβλιοθήκη GRLIB παρέχει ένα ενοποιημένο σχήμα διαχείρισης σημάτων διακοπής (interrupt signals) μέσω της προσθήκης 32 σημάτων διακοπής στους διαύλους AHB και APB. Μια υπομονάδα AMBA μπορεί να οδηγήσει οποιοδήποτε από τα σήματα διακοπής, ενώ η υπομονάδα που υλοποιεί τον ελεγκτή σημάτων διακοπής μπορεί να παρακολουθεί το ενιαίο διάνυσμα διακοπών και να παράγει το κατάλληλο σήμα διακοπής προς κάποιον επεξεργαστή. Με τον τρόπο αυτό, τα σήματα διακοπής μπορούν να δημιουργηθούν ανεξάρτητα από το ποιος επεξεργαστής ή ελεγκτής interrupts χρησιμοποιείται στο σύστημα και δε χρειάζεται να δρομολογούνται προς κάποιον καθολικό πόρο. Το σχήμα αυτό επιτρέπει τη διαμοίραση των interrupts σε πολλούς πυρήνες και τον διαχωρισμό τους από το λογισμικό.

Δυνατότητα Plug&Play

Μια ευρεία ερμηνεία του όρου "plug&play" είναι η δυνατότητα ανίχνευσης του υλικού του συστήματος μέσω του λογισμικού. Μια τέτοια δυνατότητα καθιστά δυνατή τη χρήση εφαρμογών λογισμικού ή λειτουργικών συστημάτων τα οποία αυτορυθμίζονται για να ανταποκριθούν στο υπάρχον υλικό. Αυτό απλοποιεί ιδιαίτερα την ανάπτυξη εφαρμογών λογισμικού, εφόσον δε χρειάζεται να τροποποιούνται για κάθε πιθανή σύνθεση του υλικού.

Στη βιβλιοθήκη GRLIB, οι πληροφορίες για plug&play αποτελούνται από τρία στοιχεία: μια μοναδική ταυτότητα για κάθε πυρήνα IP, το memory mapping στο AHB/APB, και το διάνυσμα σημάτων διακοπής που χρησιμοποιείται. Οι πληροφορίες αυτές στέλνονται ως ένα σταθερό διάνυσμα στον arbiter/αποκωδικοποιητή του διαύλου, όπου και αποθηκεύονται σε μια μικρή περιοχή μόνο-για-ανάγνωση στην κορυφή του χώρου διευθύνσεων. Κάθε συσκευή master στο AHB μπορεί να αναγνώσει τη διαμόρφωση του συστήματος κατά τους κανονικούς κύκλους ρολογιού, και ένα λειτουργικό σύστημα plug&play μπορεί να υποστηριχθεί.

Προκειμένου οι πληροφορίες plug&play των υπομονάδων AMBA να παρέχονται με ομαλό τρόπο, έχει οριστεί ένα μητρώο σύνθεσης (configuration record) για τις συσκευές AMBA (Εικόνα 4). Το μητρώο σύνθεσης αποτελείται από 8 λέξεις των 32 bits, από τις οποίες οι τέσσερις περιέχουν πληροφορίες που καθορίζουν τον τύπο του πυρήνα και τη δρομολόγηση των interrupts, ενώ οι άλλες τέσσερις περιέχουν τους λεγόμενους «καταχωρητές διευθύνσεων» (bank address registers – BAR), οι οποίοι καθορίζουν το memory mapping.



Εικόνα 4: Μητρώο σύνθεσης AMBA

Η λέξη που καθορίζει τη σύνθεση του πυρήνα για κάθε συσκευή περιλαμβάνει μια ταυτότητα παροχέα (vendor ID), μια ταυτότητα συσκευής (device ID), αριθμό έκδοσης (version number) και πληροφορίες δρομολόγησης των interrupts. Παρέχεται επίσης ένας δείκτης τύπου σύνθεσης (configuration type – CT) για μελλοντική χρήση σε περίπτωση επέκτασης της λέξης σύνθεσης. Οι καταχωρητές BAR περιέχουν την αρχική διεύθυνση για τον χώρο διευθύνσεων που αντιστοιχεί στη συσκευή, μια μάσκα που καθορίζει το μέγεθος του χώρου, πληροφορίες σχετικά με το αν ο χώρος αυτός μπορεί να αποθηκευθεί σε κρυφή μνήμη ή να γίνει pre-fetch, και μια δήλωση τύπου που προσδιορίζει τον χώρο ως χώρο διευθύνσεων μνήμης AHB, ως χώρο εισόδου/εξόδου (I/O) AHB ή ως χώρο εισόδου/εξόδου (I/O) APB. Το μητρώο σύνθεσης μπορεί να περιέχει έως και τέσσερις καταχωρητές BAR και συνεπώς στον πυρήνα μπορούν να αντιστοιχηθούν έως και τέσσερις διαφορετικοί χώροι διευθύνσεων.

Μεταφερσιμότητα

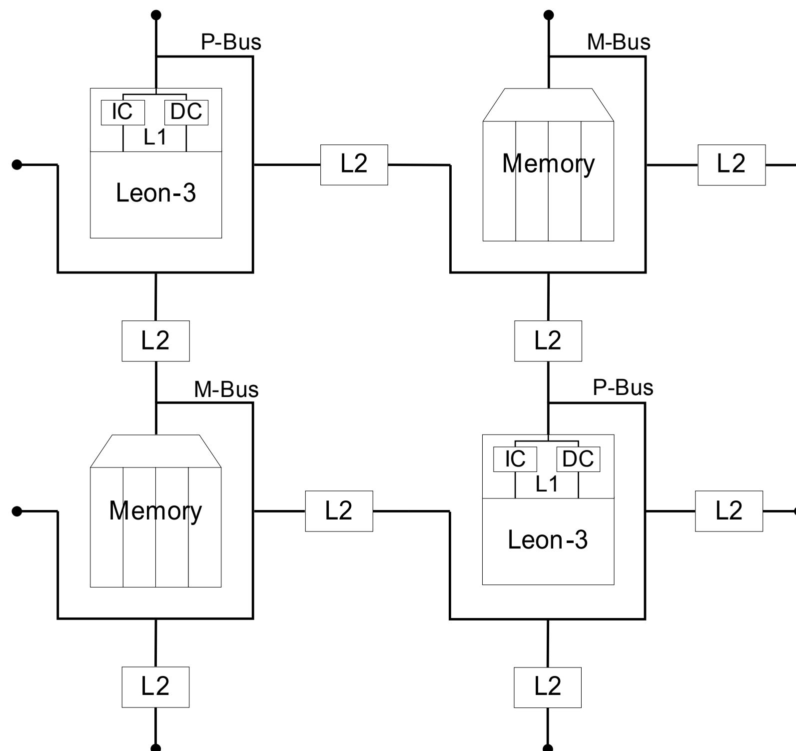
Η βιβλιοθήκη GRLIB έχει σχεδιαστεί ώστε να είναι ανεξάρτητη από την τεχνολογία και να υλοποιείται εύκολα τόσο σε τεχνολογίες ASIC όσο και σε FPGA. Η υποστήριξη μεταφερσιμότητας παρέχεται σε συσκευές όπως η RAM ενός port, η RAM δύο ports, η RAM διπλού port, η ROM ενός port, γεννήτριες ρολογιού και pads. Η μεταφερσιμότητα υλοποιείται με χρήση εικονικών συσκευών οι οποίες περιέχουν ένα generic της γλώσσας VHDL με το οποίο επιλέγεται το αντίστοιχο macro από την επιλεγμένη βιβλιοθήκη της εκάστοτε τεχνολογίας. Για τις μονάδες RAM, χρησιμοποιούνται generics και για τον καθορισμό πλάτους της διεύθυνσης και των δεδομένων, και τον αριθμό των ports.

Διαθέσιμοι πυρήνες IP

Η βιβλιοθήκη περιλαμβάνει πυρήνες για έλεγχο των AMBA AHB/APB, για τον επεξεργαστή SPARC LEON3, μια μονάδα κινητής υποδιαστολής IEEE-754, έναν ελεγκτή SDRAM PC133 των 32 bits, μια γέφυρα PCI με DMA των 32 bits, Ethernet MAC των 32 bits, έναν ελεγκτή CAN-2.0, έναν σύνδεσμο USB-2.0 για αποσφαλμάτωση, ελεγκτή για PROM/SRAM των 8/16/32 bits, έναν ελεγκτή SSRAM των 32 bits, έναν ελεγκτή DDR των 32 bits, ένα port GPIO των 32 bits, μονάδα χρονισμού, ελεγκτή interrupts, διεπαφή PS/2, ελεγκτή VGA και διάφορους άλλους πυρήνες. Γεννήτριες μνημών είναι διαθέσιμες για μοντέλα Actel, Altera, Atmel, Lattice, UMC, Virage και Xilinx. Θα πρέπει να σημειωθεί ότι ένα μέρος της βιβλιοθήκης δεν είναι open-source, αλλά εμπορικό, και συνεπώς η πρόσβαση σε ορισμένους πυρήνες είναι δυνατή μόνο μετά από αγορά ολόκληρης της βιβλιοθήκης.

1.3.2. Στοχευόμενη υλοποίηση

Μέχρι στιγμής δεν έχει γίνει αναφορά στον τρόπο διασύνδεσης των επεξεργαστών με τις γειτονικές μονάδες μνήμης. Στην Εικόνα 5 φαίνονται τα δομικά στοιχεία και ο τρόπος με τον οποίο πραγματοποιείται η διασύνδεσή τους στη στοχευόμενη υλοποίηση.



Εικόνα 5: Διασύνδεση των κόμβων στο βασικό πλακίδιο SiScape

Ο επεξεργαστής που χρησιμοποιείται είναι μια παραλλαγή του επεξεργαστή Leon-3 που συμπεριλαμβάνεται στη βιβλιοθήκη SiScape. Ο Leon-3 είναι επεξεργαστής των 32 bits, συμβατός με την αρχιτεκτονική IEEE-1754 (SPARC V-8). Περιλαμβάνει προαιρετικά μια Μονάδα Κινητής Υποδιαστολής (Floating Point Unit – FPU) και ακολουθεί την αρχιτεκτονική Harvard, δηλαδή χρησιμοποιεί ξεχωριστές κρυφές μνήμες για εντολές και δεδομένα, όπως φαίνεται και στην εικόνα. Η μονάδα αριθμητικής λογικής (Arithmetic-Logic Unit – ALU) χρησιμοποιεί pipeline 7 σταδίων. Η παραλλαγή του Leon-3 που θα χρησιμοποιηθεί τελικά στην υλοποίηση έχει τροποποιηθεί ώστε να υποστηρίζει την παράλληλη εκτέλεση πολλαπλών νημάτων. Για την υποστήριξη αυτή, ο αριθμός των λειτουργικών μονάδων παραμένει ο ίδιος, ενώ το αρχείο καταχωρητών, ο δείκτης προγράμματος και οι καταχωρητές κατάστασης πολλαπλασιάζονται επί ν και εφαρμόζεται μια γρήγορη αλλαγή περιβάλλοντος (context switching) μεταξύ των νημάτων.

Ο κάθε επεξεργαστής επικοινωνεί μέσω ενός διαύλου AMBA AHB (περιλαμβάνεται επίσης στη βιβλιοθήκη GRLIB) με τέσσερις κρυφές μνήμες δεύτερου επιπέδου. Ο δίαυλος αυτός ονομάζεται P-bus (Processor bus). Στη συνέχεια, καθεμία από τις κρυφές μνήμες δεύτερου επιπέδου επικοινωνεί μέσω ενός δεύτερου διαύλου AMBA AHB (M-bus – Memory bus) με μια μονάδα κύριας μνήμης.

Το σύστημα μνήμης της SiScape λοιπόν, αποτελείται από τις κρυφές μνήμες πρώτου επιπέδου του επεξεργαστή (διατίθενται με τον Leon-3 στη βιβλιοθήκη GRLIB), την κύρια μνήμη

(διάφορες υλοποιήσεις μπορούν να βρεθούν στην GRLIB) και τις κρυφές μνήμες δεύτερου επιπέδου.

Η υλοποίηση αυτών των μνημών δεύτερου επιπέδου είναι το θέμα της παρούσας μεταπτυχιακής εργασίας. Οι μνήμες αυτές δεν υπάρχουν υλοποιημένες σε καμία μορφή (όχι μόνο στη βιβλιοθήκη GRLIB αλλά και γενικότερα), οπότε έπρεπε να σχεδιαστούν και να αναπτυχθούν από το μηδέν. Αποτελούν το μέσο που καθιστά δυνατή τη λειτουργία του συστήματος μνήμης και η λειτουργία που οι ίδιες καλούνται να εκπληρώσουν είναι τετραπλή:

1. Από την πλευρά του διαύλου του επεξεργαστή (P-bus) η κρυφή μνήμη δεύτερου επιπέδου θα πρέπει να λειτουργεί ως συσκευή slave. Εφόσον ο επεξεργαστής αποτελεί τη master συσκευή στον δίαυλο αυτό, είναι απαραίτητη η υλοποίηση μιας διεπαφής slave προς τον δίαυλο AHB, έτσι ώστε η κρυφή μνήμη να λαμβάνει τις αιτήσεις του επεξεργαστή και να προχωρά στην εξυπηρέτησή τους.
2. Από την πλευρά του διαύλου της κύριας μνήμης (M-bus) η κρυφή μνήμη δεύτερου επιπέδου θα πρέπει να λειτουργεί ως συσκευή master. Η κρυφή μνήμη πρέπει εσωτερικά να παράγει τις κατάλληλες αιτήσεις προς την κύρια μνήμη (σε απάντηση φυσικά των αιτήσεων του επεξεργαστή) και στη συνέχεια να τις προωθεί στην κύρια μνήμη (η οποία λειτουργεί ως slave) μέσω μιας διεπαφής master, η οποία πρέπει επίσης να υλοποιηθεί.
3. Μια ιδιαίτερα βασική λειτουργία που πρέπει να εκπληρώσουν οι κρυφές μνήμες δευτέρου επιπέδου είναι ο συγχρονισμός των δύο διαύλων (P-bus και M-bus). Για υλοποιήσεις της αρχιτεκτονικής SiScape με ελάχιστους κόμβους είναι εύκολο να διαμοιραστεί το ίδιο ρολόι σε όλους. Καθώς όμως ο αριθμός των κόμβων αυξάνεται, ο διαμοιρασμός του ίδιου παλμού χρονισμού γίνεται δυσκολότερος και τελικά αδύνατος. Συνεπώς θα πρέπει να θεωρηθεί ότι οι δύο δίαυλοι λειτουργούν με διαφορετικό ρολόι, ενώ ο συγχρονισμός των δύο διαύλων και ο συντονισμός της επικοινωνίας τους πρέπει να πραγματοποιείται από την κρυφή μνήμη. Η υλοποίηση του μηχανισμού συγχρονισμού δεν κάνει καμία παραδοχή για τη σχέση μεταξύ των δύο ρολογιών, συνεπώς οι δύο δίαυλοι μπορούν να λειτουργήσουν με εντελώς διαφορετικά ρολόγια, χωρίς περιορισμούς στη συχνότητα ή τη φάση τους.
4. Τέλος, η κρυφή μνήμη δεύτερου επιπέδου οφείλει να πραγματοποιεί την κλασική λειτουργία μιας κρυφής μνήμης, δηλαδή την εξυπηρέτηση των αιτήσεων του επεξεργαστή όταν τα δεδομένα είναι διαθέσιμα στην κρυφή μνήμη, τη μεταφορά αιτηθέντων μπλοκς από την κύρια μνήμη στην κρυφή, καθώς και την ανανέωση της κύριας μνήμης.

Μελλοντικά, μπορεί να υλοποιηθεί ένας μηχανισμός για εξασφάλιση της συνέπειας μεταξύ των τεσσάρων κρυφών μνημών δεύτερου επιπέδου που έχουν πρόσβαση σε μια κύρια μνήμη, αυτό όμως δεν περιλαμβάνεται στους στόχους της παρούσας εργασίας. Στα επόμενα κεφάλαια θα αναλυθούν εκτενέστερα οι λειτουργίες που πρέπει να επιτελούν οι κρυφές μνήμες δεύτερου επιπέδου. Πρώτα παρουσιάζεται το πρωτόκολλο AMBA AHB και ο τρόπος λειτουργίας των συσκευών master και slave που συνδέονται στον δίαυλο. Έπειτα παρουσιάζεται το πρόβλημα του συγχρονισμού δύο διαύλων διαφορετικών ρολογιών και ο τρόπος επίλυσής του και ακολούθως αναλύονται οι βασικές λειτουργίες μιας κρυφής μνήμης. Τέλος παρουσιάζεται η τελική υλοποίηση της κρυφής μνήμης, με ανάλυση του κώδικα VHDL και παρουσίαση του υλικού που παράγεται από τη σύνθεση.

Κεφάλαιο 2:

Το πρωτόκολλο AMBA και η λειτουργία των συσκευών master και slave

Στο κεφάλαιο αυτό παρουσιάζεται το πρωτόκολλο AMBA και συγκεκριμένα το AMBA AHB, καθώς και ο τρόπος λειτουργίας των συσκευών master και slave. Έχει δοθεί λιγότερη έμφαση σε στοιχεία του πρωτοκόλλου που δεν είναι απαραίτητα για τον υπό ανάλυση σχεδιασμό.

2.1.Εισαγωγή στην προδιαγραφή AMBA

Η προδιαγραφή της Προηγμένης Αρχιτεκτονικής Διαύλου Μικροελεγκτών (Advanced Microcontroller Bus Architecture – AMBA) ορίζει ένα πρότυπο επικοινωνίας on-chip για τον σχεδιασμό ενσωματωμένων ελεγκτών υψηλής απόδοσης.

Τρεις ξεχωριστοί δίαυλοι ορίζονται στην προδιαγραφή AMBA:

- ο Προηγμένος Δίαυλος Υψηλής Απόδοσης (Advanced High-performance Bus – AHB)
- ο Προηγμένος Δίαυλος Συστήματος (Advanced System Bus – ASB)
- ο Προηγμένος Δίαυλος Περιφερειακών (Advanced Peripheral Bus – APB)

Στον παρόντα σχεδιασμό χρησιμοποιείται ο Προηγμένος Δίαυλος Υψηλής Απόδοσης (Advanced High-performance Bus – AHB).

Ο AMBA AHB προορίζεται για μονάδες συστήματος που απαιτούν υψηλή απόδοση και μεγάλη συχνότητα ρολογιού.

Ο δίαυλος AHB λειτουργεί ως ο υψηλής απόδοσης βασικός δίαυλος του συστήματος. Υποστηρίζει την αποτελεσματική διασύνδεση επεξεργαστών, on-chip μνημών και διεπαφών με εξωτερικές off-chip μνήμες. Ο AHB έχει επίσης οριστεί ώστε να εξασφαλίζει ευκολία χρήσης κατά τη σχεδιαστική διαδικασία που χρησιμοποιεί σύνθεση και αυτόματες τεχνικές ελέγχου.

2.2.Ορολογία

Οι ακόλουθοι όροι χρησιμοποιούνται στην προδιαγραφή.

Κύκλος διαύλου Ένας κύκλος διαύλου είναι η βασική μονάδα της μιας περιόδου του ρολογιού διαύλου και στην περιγραφή των πρωτοκόλλων AMBA AHB ή APB ορίζεται από θετική-ακμή μέχρι θετική ακμή. Ο κύκλος διαύλου του ASB ορίζεται από αρνητική-ακμή μέχρι αρνητική-ακμή. Ο χρονισμός των σημάτων του διαύλου γίνεται με βάση το ρολόι που παράγει τους κύκλους διαύλου.

Μεταφορά στον δίαυλο Μια μεταφορά στον δίαυλο AMBA AHB ή ASB είναι μια

λειτουργία εγγραφής ή ανάγνωσης ενός αντικειμένου δεδομένων, η οποία μπορεί να διαρκέσει έναν ή περισσότερους κύκλους διαύλου. Η μεταφορά τερματίζεται από μια απάντηση *ολοκλήρωσης* από τη συσκευή slave στην οποία απευθύνεται.
Τα μεγέθη μεταφορών που υποστηρίζονται από τον δίαυλο AMBA ASB είναι ενός byte (8-bit), μισής λέξης (16-bit) και λέξης (32-bit). Ο AMBA AHB υποστηρίζει επίσης μεταφορές μεγαλύτερου μεγέθους, περιλαμβανομένων των μεταφορών 64-bit και 128-bit. Μια μεταφορά διαύλου AMBA APB είναι μια λειτουργία εγγραφής ή ανάγνωσης ενός αντικειμένου δεδομένων, η οποία απαιτεί πάντα δύο κύκλους διαύλου.

Λειτουργία κατά ριπές Η λειτουργία κατά ριπές ορίζεται ως μία ή περισσότερες συναλλαγές δεδομένων, οι οποίες εκκινούνται από μια συσκευή master, έχουν σταθερό πλάτος συναλλαγής και πραγματοποιούνται σε μία περιοχή του χώρου διευθύνσεων, οι διευθύνσεις για την οποία αυξάνονται σταδιακά. Το βήμα αύξησης για κάθε συναλλαγή καθορίζεται από το πλάτος της μεταφοράς (byte, μισή λέξη, λέξη). Ο δίαυλος APB δεν υποστηρίζει λειτουργία κατά ριπές.

2.3.Κατάλογος σημάτων του πρωτοκόλλου AMBA AHB

Όλα τα σήματα AMBA έχουν ονομαστεί έτσι ώστε το πρώτο γράμμα του ονόματος να δηλώνει με ποιον δίαυλο συσχετίζεται το συγκεκριμένο σήμα. Συνεπώς, τα σήματα που παρατίθενται εδώ και τα οποία χρησιμοποιούνται στον σχεδιασμό έχουν όλα ονόματα που ξεκινούν με το γράμμα H, για να δηλώνουν ότι σχετίζονται με έναν δίαυλο AHB. Ένα μικρό n στο όνομα του σήματος δηλώνει ότι το σήμα είναι ενεργό LOW, σε αντίθετη περίπτωση όλα τα γράμματα σε ένα όνομα σήματος είναι πάντα κεφαλαία. Ως παράδειγμα αναφέρεται το σήμα HREADY, το οποίο χρησιμοποιείται για να δηλώσει ότι το τμήμα δεδομένων μιας μεταφοράς AHB μπορεί να ολοκληρωθεί, και είναι ενεργό HIGH.

Στο υποκεφάλαιο αυτό παρουσιάζεται μια περίληψη των σημάτων του διαύλου AMBA AHB. Η πλήρης περιγραφή καθενός από τα σήματα μπορεί να βρεθεί σε παρακάτω υποκεφάλαια.

Ονομασία	**Πηγή**	**Περιγραφή**
HCLK Ρολόι διαύλου	Πηγή ρολογιού	Το ρολόι αυτό χρονίζει όλες τις μεταφορές στον δίαυλο. Όλοι οι χρονισμοί των σημάτων γίνονται αναφορικά με τη θετική ακμή του **HCLK**.
HRESETn Επανεκκίνηση	Ελεγκτής επανεκκίνησης	Το σήμα reset του διαύλου είναι ενεργό LOW και χρησιμοποιείται για να επανεκκινήσει το σύστημα και τον δίαυλο. Αυτό είναι και το μοναδικό σήμα που είναι ενεργό LOW.
HADDR[31:0] Δίαυλος διευθύνσεων	Master	Ο δίαυλος διευθύνσεων του συστήματος, μεγέθους 32-bit.
HTRANS[1:0] Τύπος μεταφοράς	Master	Δηλώνει τον τύπο της τρεχούμενης μεταφοράς, η οποία μπορεί να είναι Μη-Ακολουθιακή (NONSEQUENTIAL), Ακολουθιακή (SEQUENTIAL), Αδρανής (IDLE) ή Απασχολημένη (BUSY).

HWRITE Κατεύθυνση μεταφοράς	Master	Όταν είναι HIGH το σήμα αυτό δηλώνει μια μεταφορά εγγραφής, ενώ όταν είναι LOW δηλώνει μια μεταφορά ανάγνωσης.
HSIZE[2:0] Μέγεθος Μεταφοράς	Master	Δηλώνει το μέγεθος της μεταφοράς, το οποίο είναι τυπικά ένα byte (8-bit), μισή λέξη (16-bit) ή μία λέξη (32-bit). Το πρωτόκολλο επιτρέπει και μεταφορές μεγαλύτερου μεγέθους με μέγιστο μέγεθος τα 1024 bits.
HBURST[2:0] Τύπος ριπής	Master	Δηλώνει εάν η μεταφορά αποτελεί μέρος μιας μεταφοράς κατά ριπές. Υποστηρίζονται μεταφορές κατά ριπές τεσσάρων, οκτώ και δεκαέξι βημάτων και η μεταφορά κατά ριπές μπορεί να είναι είτε αυξανόμενη είτε περιτύλιξης.
HPROT[3:0] Έλεγχος προστασίας	Master	Τα σήματα ελέγχου προστασίας παρέχουν επιπλέον πληροφορίες για μια πρόσβαση στον δίαυλο και προορίζονται πρωταρχικά για χρήση από οποιαδήποτε υπομονάδα επιθυμεί να υλοποιήσει κάποιο επίπεδο προστασίας. Τα σήματα δηλώνουν αν η μεταφορά είναι μια ανάκτηση κωδικού εντολής (opcode fetch) ή προσπέλαση δεδομένων, όπως επίσης και αν η μεταφορά είναι προσπέλαση προνομιούχου τύπου ή τύπου χρήστη. Για τις συσκευές master του διαύλου που περιλαμβάνουν μια μονάδα διαχείρισης μνήμης τα σήματα αυτά δηλώνουν επίσης αν η τρεχούμενη μεταφορά μπορεί να αποθηκευθεί σε κρυφή μνήμη ή σε buffer.
HWDATA[31:0] Δίαυλος δεδομένων εγγραφής	Master	Ο δίαυλος δεδομένων εγγραφής χρησιμοποιείται για τη μεταφορά δεδομένων από τον master προς τους slaves του διαύλου κατά τη διάρκεια λειτουργιών εγγραφής. Προτείνεται ένα ελάχιστο μέγεθος διαύλου δεδομένων των 32 bits. Ωστόσο, αυτό μπορεί πολύ εύκολα να επεκταθεί ώστε να επιτρέπεται λειτουργία μεγαλύτερου ρυθμού μεταφοράς δεδομένων.
HSELx Επιλογή συσκευής slave	Decoder	Κάθε συσκευή slave του AHB έχει το δικό της σήμα επιλογής slave και το σήμα αυτό δηλώνει ότι η τρέχουσα μεταφορά προορίζεται για την επιλεγμένη συσκευή slave. Το σήμα αυτό είναι απλώς μια συνδυαστική αποκωδικοποίηση του διαύλου διευθύνσεων.
HRDATA[31:0] Δίαυλος δεδομένων ανάγνωσης	Slave	Ο δίαυλος δεδομένων ανάγνωσης χρησιμοποιείται για τη μεταφορά δεδομένων από τις συσκευές slave του διαύλου προς τη συσκευή master κατά τη διάρκεια μιας λειτουργίας ανάγνωσης. Προτείνεται ένα ελάχιστο μέγεθος διαύλου δεδομένων των 32 bits. Ωστόσο, αυτό μπορεί πολύ εύκολα να επεκταθεί ώστε να επιτρέπεται λειτουργία μεγαλύτερου ρυθμού μεταφοράς δεδομένων.
HREADY Ολοκλήρωση μεταφοράς	Slave	Όταν είναι HIGH το σήμα **HREADY** δηλώνει ότι μια μεταφορά έχει ολοκληρωθεί στον δίαυλο. Το σήμα αυτό μπορεί να οδηγηθεί σε LOW για να επεκτείνει μια μεταφορά. Σημείωση: Οι συσκευές slave του διαύλου χρειάζονται το **HREADY** και ως σήμα εξόδου και ως σήμα εισόδου.

HRESP[1:0] Απάντηση μεταφοράς	Slave	Η απάντηση μεταφοράς παρέχει επιπλέον πληροφορίες για την κατάσταση μιας μεταφοράς. Παρέχονται τέσσερις δυνατές απαντήσεις: Εντάξει (OKAY), Σφάλμα (ERROR), Επαναπροσπάθεια (RETRY) και Διαχωρισμός (SPLIT).

Πίνακας 1: Κατάλογος σημάτων AMBA AHB

Ο δίαυλος AMBA AHB έχει επίσης έναν αριθμός σημάτων που είναι απαραίτητα για την υποστήριξη πολλαπλών master. Πολλά από αυτά τα σήματα διαιτησίας (arbitration signals) είναι αποκλειστικές συνδέσεις από σημείο σε σημείο και η κατάληξη x στον Πίνακα 2 δηλώνει ότι το σήμα προέρχεται από την υπομονάδα X.

Ονομασία	**Πηγή**	**Περιγραφή**
HBUSREQx Αίτηση απόκτησης διαύλου	Master	Ένα σήμα από τον master x του διαύλου προς τον διαιτητή διαύλου, το οποίο δηλώνει ότι ο master ζητά τον δίαυλο. Υπάρχει ένα σήμα **HBUSREQx** για κάθε συσκευή master στο σύστημα, με μέγιστο 16 συσκευές master σε έναν δίαυλο.
HLOCKx Κλειδωμένες μεταφορές	Master	Όταν είναι HIGH το σήμα αυτό δηλώνει πως η συσκευή master ζητά κλειδωμένη πρόσβαση στον δίαυλο και καμία άλλη συσκευή master δεν μπορεί να αποκτήσει τον δίαυλο μέχρι το σήμα αυτό ξαναγίνει LOW.
HGRANTx Παραχώρηση διαύλου	Arbiter	Το σήμα αυτό δηλώνει ότι ο master x είναι επί του παρόντος ο master με τη μεγαλύτερη προτεραιότητα. Η ιδιοκτησία των σημάτων διεύθυνσης/ελέγχου αλλάζει στο τέλος μιας μεταφοράς όταν το σήμα **HREADY** είναι HIGH, συνεπώς ένας master αποκτά πρόσβαση στον δίαυλο όταν και το **HREADY** και το **HGRANTx** είναι HIGH.
HMASTER[3:0] Αριθμός master	Arbiter	Τα σήματα αυτά προέρχονται από τον διαιτητή (arbiter) του διαύλου, δηλώνουν ποιος master πραγματοποιεί επί του παρόντος μια μεταφορά και χρησιμοποιούνται από τις συσκευές slave που υποστηρίζουν μεταφορές SPLIT ώστε να καθορίζουν ποιος master επιχειρεί προσπέλαση. Ο χρονισμός του σήματος **HMASTER** είναι ο ίδιος με τον χρονισμό των σημάτων διεύθυνσης και ελέγχου.
HMASTLOCK Κλειδωμένη ακολουθία	Arbiter	Δηλώνει ότι ο τρέχων master πραγματοποιεί μια κλειδωμένη ακολουθία μεταφορών. Το σήμα αυτό έχει τον ίδιο χρονισμό με το σήμα **HMASTER**.
HSPLITx[15:0] Αίτηση ολοκλήρωσης Διαχωρισμένης (SPLIT) μεταφοράς	Slave (που υποστηρίζει μεταφορές SPLIT)	Αυτός ο δίαυλος Διαχωρισμού (SPLIT) των 16 bits χρησιμοποιείται από έναν slave για να δηλώσει στον arbiter σε ποιον master θα πρέπει να επιτραπεί να ξαναπροσπαθήσει μια διαχωρισμένη συναλλαγή. Κάθε bit αυτού του διαύλου διαχωρισμού αντιστοιχεί σε έναν master του διαύλου.

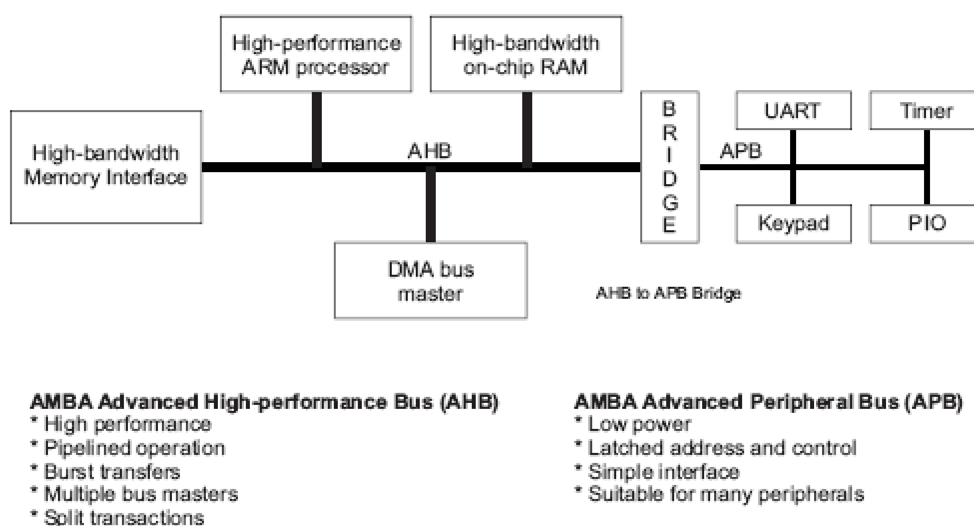
Πίνακας 2: Σήματα διαιτησίας

2.4.Το πρωτόκολλο AMBA AHB

Ο δίαυλος AHB αποτελεί μια νέα γενιά στους διαύλους AMBA, η οποία προορίζεται για να ικανοποιήσει τις απαιτήσεις των συνθέσιμων σχεδιασμών υψηλής απόδοσης. Ο AMBA AHB είναι ένα νέο επίπεδο διαύλου το οποίο τοποθετείται πάνω από τον APB και υλοποιεί τα χαρακτηριστικά που απαιτούνται από συστήματα υψηλής απόδοσης, με υψηλή συχνότητα ρολογιού.

2.4.1.Ένας τυπικός μικροελεγκτής βασισμένος στον δίαυλο AMBA AHB

Ένας μικροελεγκτής βασισμένος σε δίαυλο AMBA αποτελείται τυπικά από έναν κεντρικό δίαυλο συστήματος υψηλής απόδοσης, ικανό να υποστηρίξει το εύρος ζώνης μιας εξωτερικής μνήμης, πάνω στον οποίο τοποθετούνται η CPU και άλλες συσκευές DMA (Direct Memory Access), όπως επίσης και από μια γέφυρα προς έναν στενότερο δίαυλο APB πάνω στον οποίο τοποθετούνται περιφερειακές συσκευές μικρότερου εύρους ζώνης. Η Εικόνα 6 δείχνει και τον δίαυλο AHB και τον APB σε ένα τυπικό σύστημα AMBA.

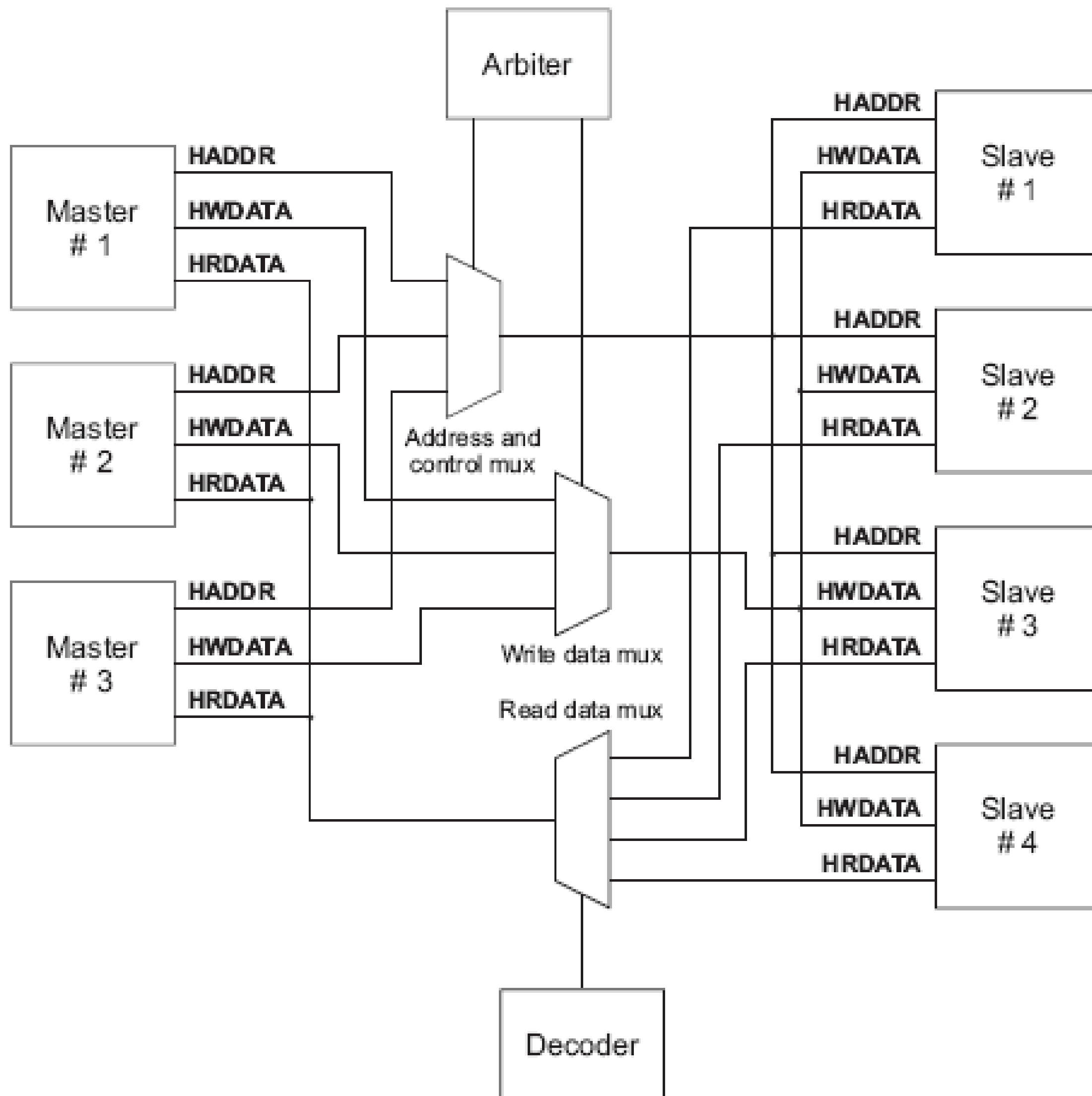


Εικόνα 6: Ένα τυπικό σύστημα AMBA

2.4.2.Διασύνδεση διαύλου

Το πρωτόκολλο διαύλου AMBA AHB σχεδιάστηκε για να λειτουργεί βάσει ενός κεντρικού σχήματος διασύνδεσης με πολυπλέκτες. Χρησιμοποιώντας το σχήμα αυτό, όλες οι συσκευές master οδηγούν τα σήματα διεύθυνσης και ελέγχου που περιγράφουν τη μεταφορά που θέλουν να πραγματοποιήσουν και ο arbiter ποια συσκευή master θα προωθήσει τα σήματα ελέγχου και διεύθυνσής της προς όλους τους slaves. Ένας κεντρικός αποκωδικοποιητής απαιτείται επίσης για να ελέγχει τον πολυπλέκτη σημάτων δεδομένων ανάγνωσης και απάντησης, ο οποίος επιλέγει τα κατάλληλα σήματα από τη συσκευή slave που συμμετέχει στη μεταφορά.

Η Εικόνα 7 παρουσιάζει τη δομή που απαιτείται για την υλοποίηση ενός σχεδιασμού AMBA AHB με τρεις master και τέσσερις slave συσκευές.



Εικόνα 7: Διασύνδεση με πολυπλέκτες

2.4.3.Περίληψη λειτουργίας του AMBA AHB

Προτού μπορέσει να ξεκινήσει μια μεταφορά AMBA AHB ο master πρέπει πρώτα να αποκτήσει πρόσβαση στον δίαυλο. Η διαδικασία αυτή εκκινείται όταν ο master οδηγεί στο HIGH ένα σήμα αίτησης του διαύλου, το οποίο διαβάζεται από τον arbiter. Στη συνέχεια ο arbiter δηλώνει αν ο master θα αποκτήσει τον έλεγχο του διαύλου.

Ένας master που έχει αποκτήσει τον έλεγχο του διαύλου εκκινεί μια μεταφορά AMBA AHB οδηγώντας τα σήματα διεύθυνσης και ελέγχου. Τα σήματα αυτά παρέχουν πληροφορίες για τη διεύθυνση, την κατεύθυνση (ανάγνωση/εγγραφή) και το πλάτος της μεταφοράς, καθώς επίσης και αν η μεταφορά αποτελεί μέρος μιας κατά ριπές ακολουθίας μεταφορών. Επιτρέπονται δύο διαφορετικά είδη ακολουθίας μεταφορών κατά ριπές:

- αυξανόμενη ακολουθία, η οποία δεν πραγματοποιεί περιτύλιξη στα όρια διευθύνσεων
- ακολουθία περιτύλιξης, η οποία πραγματοποιεί περιτύλιξη στα όρια διευθύνσεων.

Για τη μεταφορά δεδομένων από τον master σε έναν slave χρησιμοποιείται ένας δίαυλος δεδομένων εγγραφής, ενώ για τη μεταφορά δεδομένων από έναν slave προς τον master χρησιμοποιείται ένας δίαυλος δεδομένων ανάγνωσης.

Κάθε μεταφορά αποτελείται από:

- έναν κύκλο διευθύνσεων και ελέγχου
- έναν ή περισσότερους κύκλους για τα δεδομένα.

Η διεύθυνση δεν μπορεί να επεκταθεί και συνεπώς όλες οι συσκευές slave πρέπει να αναγνώσουν τη διεύθυνση στον χρόνο αυτό. Τα δεδομένα, ωστόσο, μπορούν να επεκταθούν χρησιμοποιώντας το σήμα **HREADY**. Όταν είναι LOW, το σήμα αυτό προκαλεί την εισαγωγή σταδίων αναμονής (διάρκειας ενός κύκλου το καθένα) στη μεταφορά και παρέχει στη συσκευή slave επιπλέον χρόνο ώστε να οδηγήσει ή να αναγνώσει τα δεδομένα.

Κατά τη διάρκεια μιας μεταφοράς η συσκευή slave δείχνει την κατάσταση χρησιμοποιώντας τα σήματα απάντησης, **HRESP[1:0]**:

OKAY	Η απάντηση OKAY χρησιμοποιείται για να δηλώσει ότι η μεταφορά προχωράει κανονικά και όταν το σήμα **HREADY** γίνει HIGH, τότε η μεταφορά έχει ολοκληρωθεί επιτυχώς.
ERROR	Η απάντηση ERROR δηλώνει ότι έχει συμβεί ένα σφάλμα μεταφοράς και η μεταφορά είναι ανεπιτυχής.
RETRY και SPLIT	Τόσο η απάντηση RETRY όσο και η SPLIT δηλώνει ότι η μεταφορά δεν μπορεί να ολοκληρωθεί άμεσα, αλλά η συσκευή master θα πρέπει να συνεχίσει να επιχειρεί τη μεταφορά.

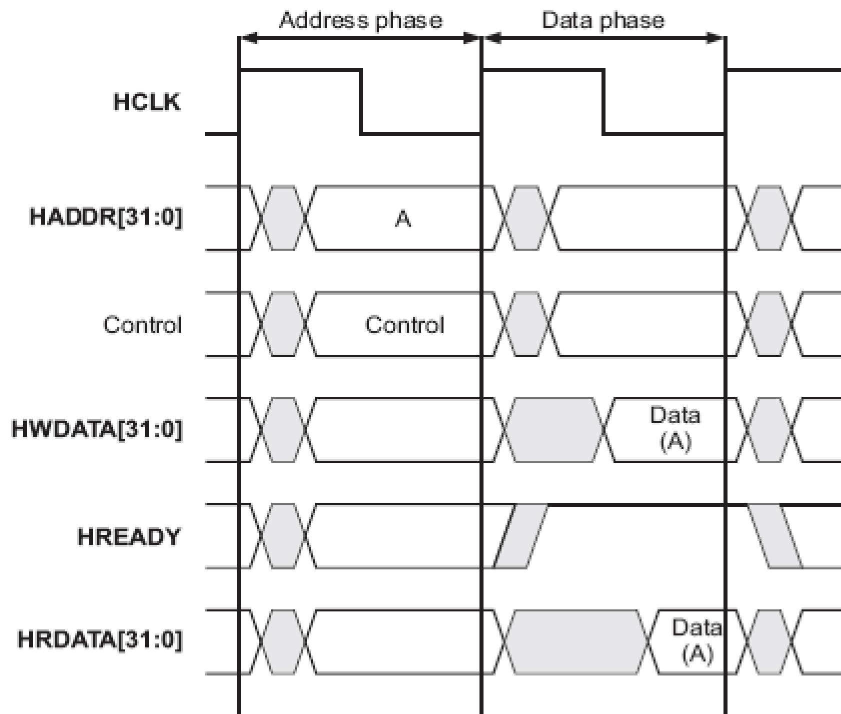
Στην κανονική λειτουργία, επιτρέπεται σε έναν master να ολοκληρώσει όλες τις μεταφορές σε μια ακολουθία μεταφορών κατά ριπές προτού ο arbiter παραχωρήσει τον δίαυλο σε κάποιον άλλο master. Ωστόσο, προκειμένου να αποφευχθούν υπερβολικές καθυστερήσεις διαιτησίας, είναι δυνατό ο arbiter να διακόψει την ακολουθία μεταφορών και σε μια τέτοια περίπτωση ο master θα πρέπει να ξαναζητήσει τον δίαυλο προκειμένου να ολοκληρώσει τις εναπομείνασες μεταφορές της ακολουθίας.

2.4.4.Βασική μεταφορά

Μια μεταφορά AHB αποτελείται από δύο διακριτές φάσεις:

- Τη φάση διεύθυνσης, η οποία διαρκεί έναν μόνο κύκλο.
- Τη φάση δεδομένων, η οποία μπορεί να απαιτεί περισσότερους από έναν κύκλους. Αυτό επιτυγχάνεται χρησιμοποιώντας το σήμα **HREADY**.

Η Εικόνα 8 δείχνει την απλούστερη μεταφορά, χωρίς κανένα στάδιο αναμονής.



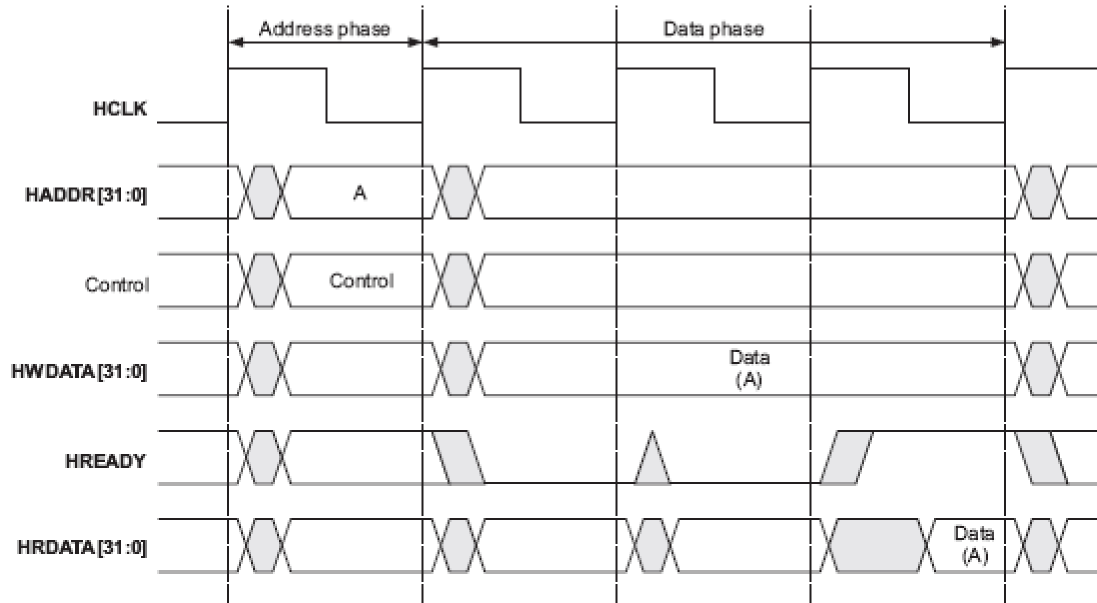
Εικόνα 8: Απλή μεταφορά

Σε μια απλή μεταφορά χωρίς στάδια αναμονής:

- Ο master οδηγεί τα σήματα διεύθυνσης και ελέγχου στον δίαυλο μετά τη θετική ακμή του ρολογιού **HCLK**.
- Ο slave δειγματοληπτεί τις πληροφορίες διεύθυνσης και ελέγχου στην επόμενη θετική ακμή του ρολογιού.
- Αφού ο slave έχει δειγματοληπτήσει τα σήματα διεύθυνσης και ελέγχου, μπορεί να ξεκινήσει να οδηγεί την κατάλληλη απάντηση, η οποία δειγματοληπτείται από τον master στην Τρίτη θετική ακμή του ρολογιού.

Αυτό το απλό παράδειγμα επιδεικνύει πως οι φάσεις διεύθυνσης και δεδομένων της μεταφοράς πραγματοποιούνται κατά τη διάρκεια διαφορετικών περιόδων του ρολογιού. Ουσιαστικά, η φάση διεύθυνσης μιας μεταφοράς συμβαίνει κατά τη φάση δεδομένων της προηγούμενης μεταφοράς. Αυτή η επικάλυψη διευθύνσεων και δεδομένων είναι θεμελιώδης για την λειτουργία pipeline του διαύλου και επιτρέπει τη λειτουργία υψηλής απόδοσης, ενώ παράλληλα παρέχει αρκετό χρόνο για να οδηγήσει ένας slave την κατάλληλη απάντηση σε μια μεταφορά.

Μια συσκευή slave μπορεί να εισάγει στάδια αναμονής σε οποιαδήποτε μεταφορά, όπως φαίνεται στην Εικόνα 9, τα οποία επεκτείνουν τη μεταφορά προσφέροντας επιπλέον χρόνο για την ολοκλήρωσή της.



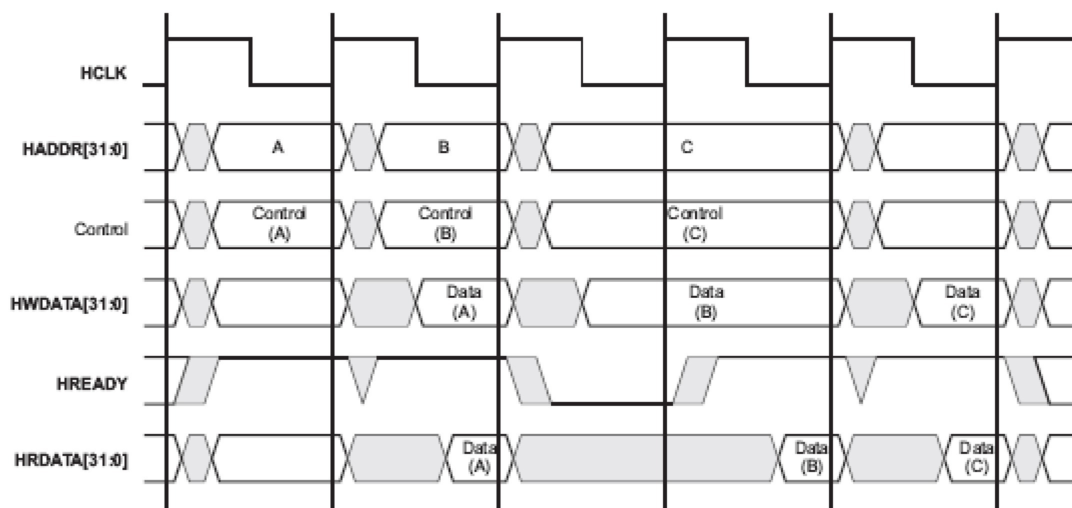
Εικόνα 9: Μεταφορά με στάδια αναμονής

Σημείωση

Για μεταφορές εγγραφής ο master του διαύλου θα πρέπει να κρατάει τα δεδομένα σταθερά κατά τη διάρκεια των κύκλων επέκτασης της μεταφοράς.

Για μεταφορές ανάγνωσης ο slave δε χρειάζεται να παρέχει έγκυρα δεδομένα μέχρι τον κύκλο στον οποίο θα ολοκληρωθεί η μεταφορά.

Όταν μια μεταφορά επεκτείνεται με τον τρόπο αυτό, αναπόφευκτα επεκτείνεται και η φάση διεύθυνσης της επόμενης μεταφοράς. Αυτό φαίνεται στην Εικόνα 10, η οποία απεικονίζει τρεις μεταφορές σε ασυσχέτιστες διευθύνσεις A, B και C.



Εικόνα 10: Πολλαπλές μεταφορές

Στην Εικόνα 10:
- οι μεταφορές στις διευθύνσεις A και C είναι και οι δύο μηδενικών σταδίων αναμονής
- η μεταφορά στη διεύθυνση B έχει ένα στάδιο αναμονής
- η επέκταση της φάσης δεδομένων της μεταφοράς στη διεύθυνση B έχει ως αποτέλεσμα την επέκταση της φάσης διεύθυνσης της μεταφοράς στη διεύθυνση C.

2.4.5.Τύποι μεταφοράς

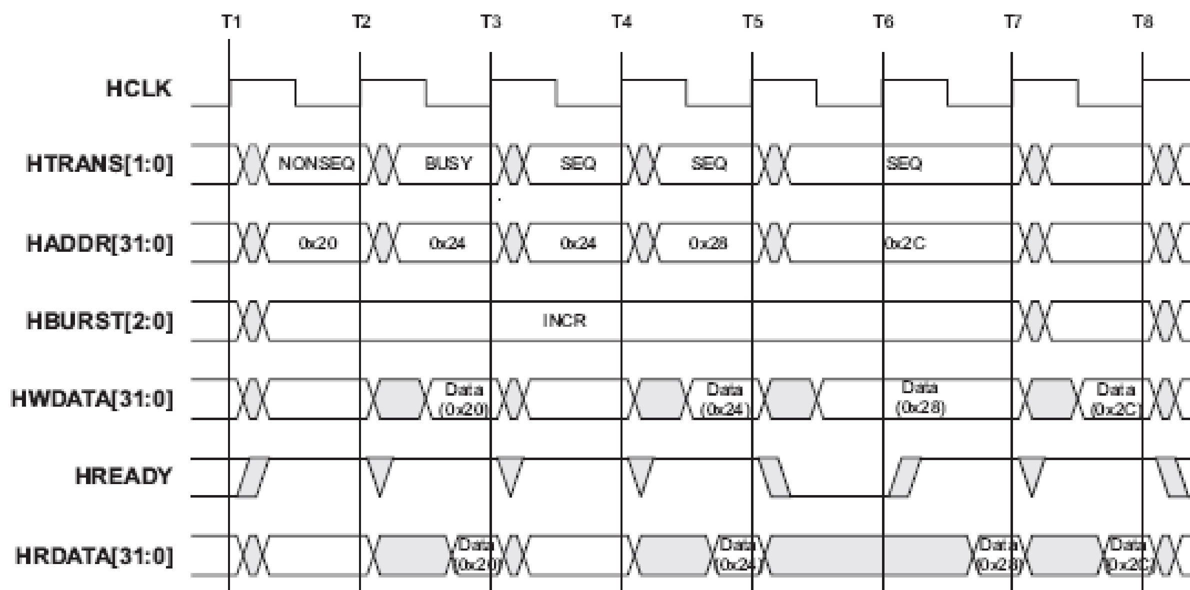
Κάθε μεταφορά μπορεί να ταξινομηθεί σε έναν από τέσσερις τύπους, όπως αυτοί δηλώνονται από τα σήματα **HTRANS[1:0]**, που φαίνονται στον Πίνακα 3.

HTRANS[1:0]	Τύπος	Περιγραφή
00	IDLE	Δηλώνει ότι δεν απαιτείται καμία μεταφορά. Ο τύπος μεταφοράς IDLE χρησιμοποιείται όταν μία συσκευή master παίρνει τον έλεγχο του διαύλου, αλλά δεν επιθυμεί να πραγματοποιήσει κάποια μεταφορά δεδομένων. Οι συσκευές slave πρέπει πάντα να δίνουν μια απάντηση OKAY χωρίς στάδια αναμονής στις μεταφορές IDLE και η μεταφορά θα πρέπει να αγνοείται από τον slave.
01	BUSY	Ο τύπος μεταφοράς BUSY επιτρέπει στις συσκευές master να εισάγουν κύκλους IDLE στο ενδιάμεσο μιας ακολουθίας μεταφορών κατά ριπές. Αυτός ο τύπος μεταφοράς δηλώνει ότι ο master του διαύλου προχωρά κανονικά με μια ακολουθία μεταφορών κατά ριπές, αλλά η επόμενη μεταφορά δεν μπορεί να πραγματοποιηθεί άμεσα. Όταν ένας master χρησιμοποιεί τον τύπο μεταφοράς BUSY, τα σήματα διεύθυνσης και ελέγχου πρέπει να ανταποκρίνονται στην επόμενη προς εκτέλεση μεταφορά της ακολουθίας. Η μεταφορά αυτή θα πρέπει να αγνοείται από τον slave. Οι συσκευές slave πρέπει πάντα να δίνουν μια απάντηση OKAY χωρίς στάδια αναμονής, όπως και στην περίπτωση των μεταφορών IDLE.
10	NONSEQ	Υποδεικνύει την πρώτη μεταφοράς μιας ακολουθίας ή μία μεμονωμένη μεταφορά. Τα σήματα διεύθυνσης και ελέγχου είναι ανεξάρτητα από αυτά της προηγούμενης μεταφοράς. Οι μεμονωμένες μεταφορές στον δίαυλο αντιμετωπίζονται ως ακολουθίες μεταφορών μήκους ένα και συνεπώς ο τύπος μεταφοράς τους είναι NONSEQUENTIAL (Μη-ακολουθιακή)
11	SEQ	Οι εναπομείνασες μεταφορές σε μία ακολουθία είναι SEQUENTIAL (Ακολουθιακές) και η διεύθυνσή τους σχετίζεται με αυτή της προηγούμενης μεταφοράς. Οι πληροφορίες ελέγχου είναι οι ίδιες με την προηγούμενη μεταφορά. Η διεύθυνση ισούται με τη διεύθυνση της προηγούμενης μεταφοράς συν το μέγεθος (σε bytes). Στην

	περίπτωση μιας μεταφοράς περιτύλιξης η διεύθυνση της μεταφοράς περιτυλίγεται στα όρια διευθύνσεων που ισούνται με το μέγεθος (σε bytes) πολλαπλασιασμένο με τον αριθμό των σταδίων της ακολουθίας (4, 8, ή 16).

Πίνακας 3: Κωδικοποίηση τύπων μεταφοράς

Η Εικόνα 11 δείχνει μια σειρά μεταφορών διαφορετικών τύπων.



Εικόνα 11: Παραδείγματα τύπων μεταφοράς

Στη Εικόνα 11:

- Η πρώτη μεταφορά είναι η αρχή μιας ακολουθίας και συνεπώς είναι τύπου NONSEQUENTIAL (Μη-ακολουθιακή)
- Ο master δεν μπορεί να πραγματοποιήσει τη δεύτερη μεταφορά της ακολουθίας αμέσως και συνεπώς χρησιμοποιεί μια μεταφορά BUSY για να καθυστερήσει την έναρξη της επόμενης μεταφοράς. Στο παράδειγμα αυτό ο master απαιτεί έναν μόνο επιπλέον κύκλου προτού είναι έτοιμος να ξεκινήσει την επόμενη μεταφορά της ακολουθίας, η οποία ολοκληρώνεται χωρίς στάδια αναμονής.
- Ο master πραγματοποιεί την τρίτη μεταφορά της ακολουθίας αμέσως, αλλά αυτή τη φορά ο slave δεν μπορεί να την ολοκληρώσει και χρησιμοποιεί το σήμα **HREADY** για να εισάγει ένα στάδιο αναμονής.
- Η τελική μεταφορά της ακολουθίας ολοκληρώνεται χωρίς στάδια αναμονής.

2.4.6.Λειτουργία ακολουθίας μεταφορών

Στο πρωτόκολλο AMBA AHB ορίζονται ακολουθίες μεταφορών τεσσάρων, οκτώ και δεκαέξι σταδίων, καθώς επίσης και ακολουθίες μη-ορισμένου μήκους και μεμονωμένες μεταφορές. Το πρωτόκολλο υποστηρίζει αυξανόμενες ακολουθίες και ακολουθίες περιτύλιξης.

Οι πληροφορίες σχετικά με την ακολουθία παρέχονται μέσω των σημάτων **HBURST[2:0]** και υπάρχουν οκτώ δυνατοί τύποι. Οι τύποι αυτοί δεν αναλύονται εδώ καθώς η μόνη απαραίτητη μορφή ακολουθίας μεταφορών για το υπό σχεδίαση σύστημα είναι η ακολουθία ενός σταδίου, δηλαδή μία μεμονωμένη NONSEQUENTIAL μεταφορά.

2.4.7.Σήματα ελέγχου

Εκτός από τον τύπο μεταφοράς και τον τύπο ακολουθίας, κάθε μεταφορά έχει έναν αριθμό από σήματα ελέγχου τα οποία παρέχουν επιπλέον πληροφορίες για τη μεταφορά. Αυτά τα σήματα ελέγχου έχουν ακριβώς τον ίδιο χρονισμό με τον δίαυλο δεδομένων. Θα πρέπει ωστόσο να παραμένουν σταθερά κατά τη διάρκεια μιας ακολουθίας.

Κατεύθυνση μεταφοράς

Όταν το **HWRITE** είναι HIGH, το σήμα αυτό δηλώνει μια μεταφορά εγγραφής και ότι ο master θα μεταδώσει δεδομένα στον δίαυλο δεδομένων εγγραφής **HWDATA[31:0]**. Όταν είναι LOW, θα πραγματοποιηθεί μια μεταφορά ανάγνωσης και ο slave θα πρέπει να οδηγήσει τα απαραίτητα δεδομένα στον δίαυλο δεδομένων ανάγνωσης **HRDATA[31:0]**.

Μέγεθος μεταφοράς

Τα σήματα HSIZE[2:0] δηλώνουν το μέγεθος της μεταφοράς, όπως φαίνεται στον Πίνακα 4.

HSIZE[2]	**HSIZE[1]**	**HSIZE[0]**	**Μέγεθος**	**Περιγραφή**
0	0	0	8 bits	Byte
0	0	1	16 bits	Μισή λέξη
0	1	0	32 bits	Λέξη
0	1	1	64 bits	-
1	0	0	128 bits	Γραμμή 4 λέξεων
1	0	1	256 bits	Γραμμή 8 λέξεων
1	1	0	512 bits	-
1	1	1	1024 bits	-

Πίνακας 4: Κωδικοποίηση μεγέθους

Το μέγεθος χρησιμοποιείται σε συνδυασμό με τα σήματα HBURST[2:0] προκειμένου να καθοριστούν τα όρια διεύθυνσης για ακολουθίες περιτύλιξης. Στον παρόντα σχεδιασμό χρησιμοποιείται επίσης για τον καθορισμό των bits εγγραφής στην RAM της κρυφής μνήμης, όπως θα περιγραφεί στο κεφάλαιο της υλοποίησης.

Έλεγχος προστασίας

Τα σήματα ελέγχου προστασίας HPROT[3:0] παρέχουν επιπλέον πληροφορίες σχετικά με μια πρόσβαση στον δίαυλο και προορίζονται πρωταρχικά για χρήση από οποιαδήποτε υπομονάδα επιθυμεί να υλοποιήσει κάποιο επίπεδο προστασίας.

Δεν είναι όλες οι συσκευές master ικανές να δημιουργήσουν ακριβείς πληροφορίες προστασίας, συνεπώς προτείνεται οι συσκευές slave να μη χρησιμοποιούν τα σήματα **HPROT**, εκτός και αν είναι απολύτως απαραίτητο. Στον υπό περιγραφή σχεδιασμό η χρήση των σημάτων αυτών δεν είναι απαραίτητη, συνεπώς δεν θα αναλυθούν παραπέρα.

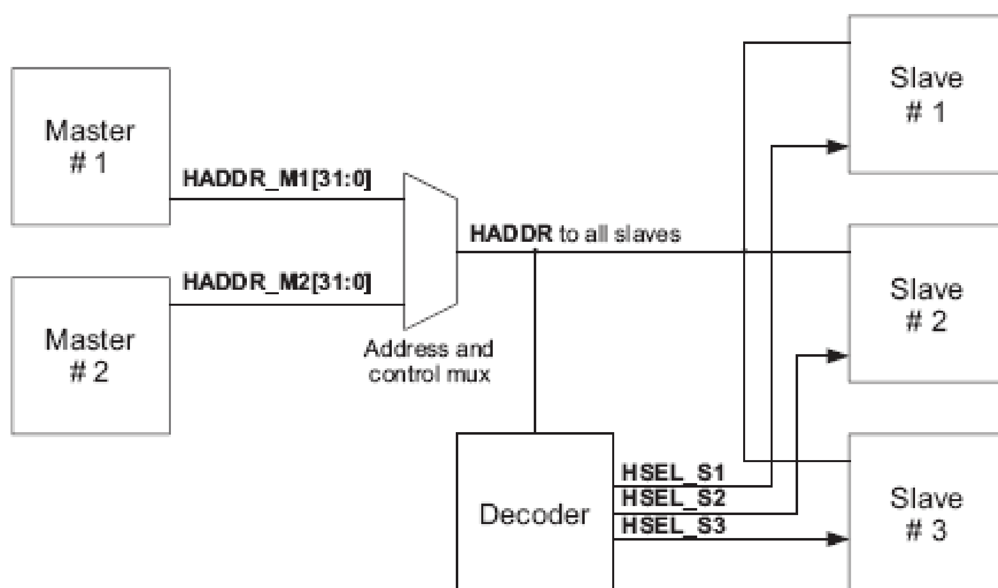
2.4.8.Αποκωδικοποίηση διευθύνσεων

Προκειμένου να παρέχει ένα σήμα επιλογής **HSELx** για κάθε slave στον δίαυλο, χρησιμοποιείται ένας κεντρικός αποκωδικοποιητής διευθύνσεων. Το σήμα επιλογής αποτελεί μια συνδυαστική αποκωδικοποίηση των πιο σημαντικών bits των σημάτων διεύθυνσης, ενώ ενθαρρύνεται η χρήση απλών σχημάτων αποκωδικοποίησης διευθύνσεων ώστε να αποφευχθούν σύνθετες λογικές αποκωδικοποίησης και να εξασφαλιστεί λειτουργία υψηλών ταχυτήτων.

Μια συσκευή slave χρειάζεται να δειγματοληπτήσει τα σήματα διεύθυνσης και ελέγχου και το **HSELx** μόνο όταν το **HREADY** είναι HIGH, πράγμα που δηλώνει ότι η τρέχουσα μεταφορά ολοκληρώνεται. Υπό ορισμένες συνθήκες, είναι δυνατόν το σήμα **HSELx** να γίνει HIGH ενώ το **HREADY** είναι LOW, αλλά ο επιλεγμένος slave θα έχει αλλάξει μέχρι να ολοκληρωθεί η τρέχουσα μεταφορά.

Ο ελάχιστος χώρος διευθύνσεων που μπορεί να αντιστοιχηθεί σε μια συσκευή slave είναι 1kB. Στην περίπτωση που σε κάποιον σχεδιασμό συστήματος δεν πραγματοποιείται πλήρης αντιστοίχιση του διαθέσιμου χώρου διευθύνσεων στις συσκευές slave, θα πρέπει να υλοποιείται ένας επιπλέον προκαθορισμένος slave ο οποίος θα παρέχει απάντηση όταν πραγματοποιείται προσπέλαση σε κάποια μη-υπαρκτή διεύθυνση. Εάν επιχειρείται NONSEQUENTIAL ή SEQUENTIAL μεταφορά σε μια μη-υπαρκτή διεύθυνση, τότε ο προκαθορισμένος slave θα πρέπει να δίνει απάντηση ERROR. Μεταφορές IDLE ή BUSY σε μη-υπαρκτές διευθύνσεις θα πρέπει να έχουν ως αποτέλεσμα μια απάντηση OK χωρίς στάδια αναμονής. Τυπικά, η λειτουργικότητα του προκαθορισμένου slave υλοποιείται ως τμήμα του κεντρικού αποκωδικοποιητή διευθύνσεων.

Η Εικόνα 12 δείχνει ένα τυπικό σύστημα αποκωδικοποίησης διευθύνσεων, καθώς και τα σήματα επιλογής slave.



Εικόνα 12: Σήματα επιλογής slave

2.4.9.Απαντήσεις συσκευών slave

Αφού ένας master εκκινήσει μια μεταφορά, ο slave αποφασίζει πώς η μεταφορά αυτή θα πρέπει να προχωρήσει. Η προδιαγραφή AHB δεν δίνει τη δυνατότητα σε έναν master να ματαιώσει μια μεταφορά αφότου αυτή έχει ξεκινήσει.

Όποτε πραγματοποιείται προσπέλαση σε κάποιον slave, θα πρέπει αυτός να δώσει μία απάντηση η οποία δηλώνει την κατάσταση της μεταφοράς. Το σήμα **HREADY** χρησιμοποιείται για να επεκταθεί μια μεταφορά και λειτουργεί σε συνδυασμό με τα σήματα απάντησης **HRESP[1:0]**, τα οποία φανερώνουν την κατάσταση της μεταφοράς.

Η συσκευή slave μπορεί να ολοκληρώσει μια μεταφορά με πολλούς τρόπους. Μπορεί:

- να ολοκληρώσει τη μεταφορά αμέσως
- να εισάγει ένα ή περισσότερα στάδια αναμονής ώστε να προλάβει να ολοκληρώσει τη μεταφορά
- να σηματοδοτήσει σφάλμα για να δηλώσει ότι η μεταφορά απέτυχε
- να καθυστερήσει την ολοκλήρωση της μεταφοράς, αλλά να επιτρέψει στον master και τον slave να ελευθερώσουν τον δίαυλο, ώστε να είναι διαθέσιμος για άλλες μεταφορές.

Ολοκλήρωση μεταφοράς

Το σήμα **HREADY** χρησιμοποιείται για να επεκτείνει το τμήμα δεδομένων μιας μεταφοράς AHB. Όταν είναι LOW το σήμα **HREADY** δηλώνει ότι η μεταφορά πρέπει να επεκταθεί και όταν είναι HIGH δηλώνει ότι η μεταφορά μπορεί να ολοκληρωθεί.

Απάντηση μεταφοράς

Μια τυπική συσκευή slave χρησιμοποιεί το σήμα **HREADY** για να εισαγάγει τον κατάλληλο αριθμό σταδίων αναμονής και μετά η μεταφορά ολοκληρώνεται με το **HREADY** να είναι HIGH και την απάντηση να είναι OKAY, που δηλώνει την επιτυχημένη ολοκλήρωση της μεταφοράς.

Ένας slave χρησιμοποιεί την απάντηση ERROR για να δηλώσει κάποιου είδους κατάσταση σφάλματος για τη σχετική μεταφορά. Τυπικά αυτό χρησιμοποιείται για κάποιο σφάλμα προστασίας, όπως για παράδειγμα μια προσπάθεια εγγραφής σε μια τοποθεσία μνήμης που είναι μόνο για ανάγνωση.

Οι συνδυασμοί απαντήσεων SPLIT και RETRY επιτρέπουν στους slaves να καθυστερούν την ολοκλήρωση μιας μεταφοράς, αλλά και να ελευθερώνουν τον δίαυλο για να μπορεί να χρησιμοποιηθεί από άλλους masters. Αυτοί οι συνδυασμοί απαντήσεων χρησιμοποιούνται συνήθως από slaves που έχουν μεγάλη καθυστέρηση προσπέλασης, οπότε μπορούν να χρησιμοποιήσουν τις απαντήσεις αυτές για να εξασφαλίσουν ότι δε θα απαγορεύεται σε άλλους masters να προσπελάσουν τον δίαυλο για μεγάλα χρονικά διαστήματα.

Μια πλήρης περιγραφή των λειτουργιών SPLIT και RETRY μπορεί να βρεθεί παρακάτω στο υποκεφάλαιο *SPLIT και RETRY*.

Η κωδικοποίηση των **HRESP[1:0]**, των σημάτων απάντησης μεταφοράς, και μια περιγραφή για κάθε μεταφορά φαίνεται στον Πίνακα 5.

HRESP[1]	**HRESP[0]**	**Απάντηση**	**Περιγραφή**
0	0	OKAY	Όταν το σήμα **HREADY** είναι HIGH, η απάντηση αυτή δηλώνει ότι η μεταφορά έχει ολοκληρωθεί επιτυχώς. Η απάντηση OKAY χρησιμοποιείται επίσης για οσουσδήποτε επιπλέον κύκλους εισάγονται, με το **HREADY** σε LOW, ακόμα και προτού δοθεί μία από τις τρεις άλλες απαντήσεις.
0	1	ERROR	Η απάντηση αυτή δείχνει ότι έχει συμβεί κάποιο σφάλμα. Η κατάσταση σφάλματος θα πρέπει να σηματοδοτείται στον master του διαύλου ώστε να γνωρίζει ότι η μεταφορά ήταν ανεπιτυχής. Απαιτείται μια απάντηση δύο-κύκλων για να δηλωθεί μια κατάσταση σφάλματος.
1	0	RETRY	Η απάντηση RETRY δηλώνει ότι η μεταφορά δεν έχει ολοκληρωθεί ακόμα, οπότε ο master θα πρέπει να ξαναεπιχειρήσει τη μεταφορά. Ο master πρέπει να συνεχίσει να ξαναεπιχειρεί τη μεταφορά μέχρι αυτή να ολοκληρωθεί. Απαιτείται απάντηση RETRY δύο-κύκλων.
1	1	SPLIT	Η μεταφορά δεν έχει ακόμα ολοκληρωθεί επιτυχώς. Ο master του διαύλου πρέπει να ξαναεπιχειρήσει τη μεταφορά όταν ξανα-αποκτήσει πρόσβαση στην δίαυλο. Ο slave θα ζητήσει πρόσβαση στον δίαυλο εκ μέρους του master όταν η μεταφορά θα μπορεί να ολοκληρωθεί. Απαιτείται απάντηση SPLIT δύο-κύκλων.

Πίνακας 5: Κωδικοποίηση απαντήσεων

Όταν είναι απαραίτητο για έναν slave να εισαγάγει έναν αριθμό σταδίων αναμονής προτού αποφασίσει ποια απάντηση θα δοθεί, τότε θα πρέπει να οδηγεί τα σήματα απάντησης σε OKAY.

Απάντηση δύο-κύκλων

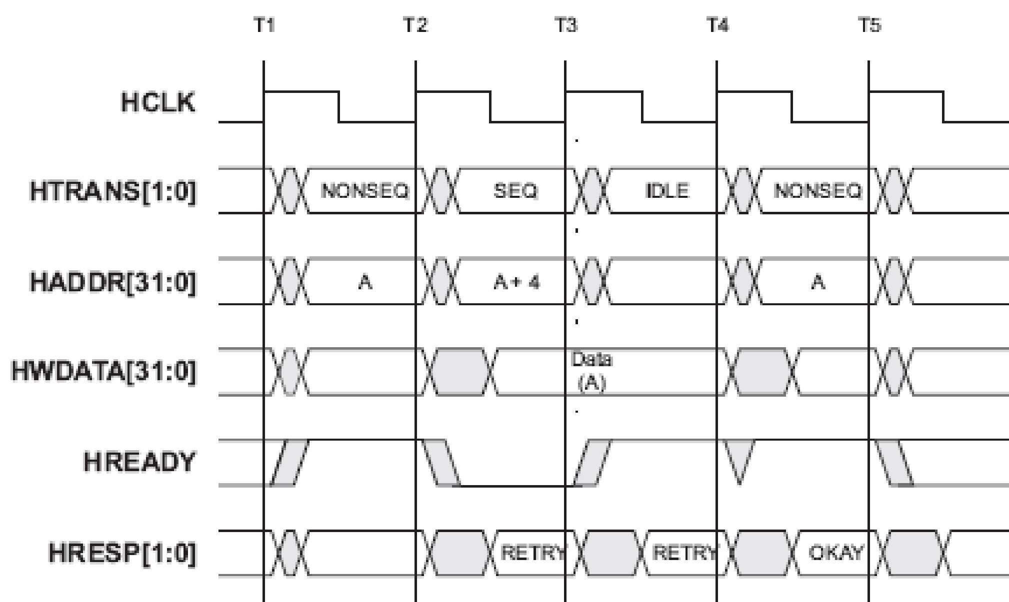
Μόνο μια απάντηση OKAY μπορεί να δοθεί σε έναν μόνο κύκλο. Οι απαντήσεις ERROR, SPLIT και RETRY απαιτούν τουλάχιστον δύο κύκλους. Για να δοθεί οποιαδήποτε από αυτές τις απαντήσεις θα πρέπει στον προτελευταίο κύκλο ο slave να οδηγήσει τα σήματα **HRESP[1:0]** σε ERROR, RETRY ή SPLIT, ενώ θα οδηγεί το **HREADY** σε LOW για να επεκτείνει τη μεταφορά για έναν επιπλέον κύκλο. Στον τελευταίο κύκλο το **HREADY** οδηγείται σε HIGH, ενώ το **HRESP[1:0]** εξακολουθεί να οδηγείται σε ERROR, RETRY ή SPLIT.

Εάν ένας slave χρειάζεται περισσότερο από δύο κύκλους για να δώσει μια απάντηση ERROR, SPLIT ή RETRY, τότε μπορούν να εισαχθούν στάδια αναμονής στην αρχή της μεταφοράς. Κατά τους κύκλους αυτούς το σήμα **HREADY** θα είναι LOW και η απάντηση θα πρέπει να είναι OKAY.

Η απάντηση δύο-κύκλων είναι απαραίτητη λόγω της λειτουργίας pipeline του διαύλου. Μέχρι τη στιγμή που ένας slave θα αρχίσει να οδηγεί μια απάντηση ERROR, SPLIT ή RETRY, τότε η διεύθυνση για την επόμενη μεταφορά θα έχει ήδη αρχίσει να μεταδίδεται στον δίαυλο. Η απάντηση δύο-κύκλων δίνει αρκετό χρόνο στον master για να ακυρώσει τη διεύθυνση αυτή και να οδηγήσει τα σήματα **HTRANS[1:0]** σε IDLE πριν την έναρξη της επόμενης μεταφοράς.

Για την απάντηση SPLIT ή RETRY η επόμενη μεταφορά πρέπει να ακυρωθεί γιατί δεν πρέπει να πραγματοποιηθεί πριν ολοκληρωθεί η τρέχουσα μεταφορά. Ωστόσο, για την απάντηση ERROR, στην περίπτωση της οποίας η τρέχουσα μεταφορά δεν επαναλαμβάνεται, η ολοκλήρωση της επόμενης μεταφοράς είναι προαιρετική.

Στην Εικόνα 13 φαίνεται ένα παράδειγμα μιας λειτουργίας RETRY.

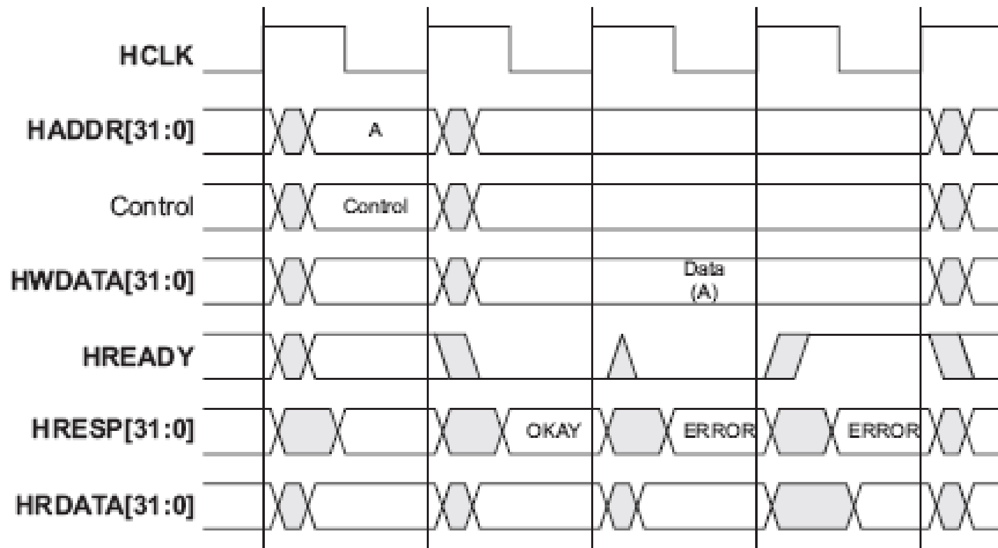


Εικόνα 13: Μεταφορά με απάντηση RETRY

Απεικονίζονται τα ακόλουθα συμβάντα:

- Ο master ξεκινά με μια μεταφορά στη διεύθυνση A.
- Πριν ληφθεί η απάντηση για τη μεταφορά αυτή ο master αυξάνει τη διεύθυνση σε A + 4.
- Ο slave στη διεύθυνση A δεν είναι σε θέση να ολοκληρώσει τη μεταφορά αμέσως και συνεπώς δίνει μια απάντηση RETRY. Η απάντηση αυτή δηλώνει στον master ότι η μεταφορά στη διεύθυνση A δεν μπορεί να ολοκληρωθεί και συνεπώς η μεταφορά στη διεύθυνση A + 4 ακυρώνεται και αντικαθίσταται από μια μεταφορά IDLE.

Στην Εικόνα 14 φαίνεται μια μεταφορά στην οποία ο slave χρειάζεται έναν κύκλο για να αποφασίσει τι απάντηση θα δώσει (κατά τη διάρκεια του οποίου τα σήματα **HRESP** δηλώνουν OKAY) και μετά ο slave τερματίζει τη μεταφορά με μια απάντηση ERROR δύο-κύκλων.



Εικόνα 14: Απάντηση error

SPLIT και RETRY

Οι απαντήσεις SPLIT και RETRY παρέχουν ένα μηχανισμό για να μπορούν οι slaves να απελευθερώνουν τον δίαυλο όταν δεν είναι σε θέση να παρέξουν άμεσα δεδομένα για μια μεταφορά. Και οι δύο μηχανισμοί επιτρέπουν τον τερματισμό της μεταφοράς στον δίαυλο και συνεπώς επιτρέπουν σε κάποιον master μεγαλύτερης προτεραιότητας να αποκτήσει πρόσβαση στον δίαυλο.

Η μεταφορά SPLIT απαιτεί επιπλέον πολυπλοκότητα τόσο για τον slave όσο και για τον arbiter, αλλά έχει το πλεονέκτημα ότι ελευθερώνει πλήρως τον δίαυλο για χρήση από άλλους masters, ενώ στην περίπτωση του RETRY επιτρέπεται πρόσβαση στον δίαυλο μόνο σε masters μεγαλύτερης προτεραιότητας.

Σημείωση

Στον υπό συζήτηση σχεδιασμό χρησιμοποιείται η απάντηση RETRY ως μια από τις απαντήσεις στις αιτήσεις του επεξεργαστή (και όχι η SPLIT), καθώς ο επεξεργαστής είναι ο μόνος master στον δίαυλό του και η επιπλέον πολυπλοκότητα που θα επέβαλλε η χρήση SPLIT δεν θα προσέφερε κανένα απολύτως πλεονέκτημα.

Ένας master του διαύλου πρέπει να αντιμετωπίζει τις μεταφορές SPLIT και RETRY με τον ίδιο τρόπο. Θα πρέπει να συνεχίσει να ζητά τον δίαυλο και να επιχειρεί τη μεταφορά έως ότου αυτή ολοκληρωθεί επιτυχώς ή τερματιστεί με μία απάντηση ERROR.

2.4.10.Δίαυλοι δεδομένων

Προκειμένου να είναι δυνατή η υλοποίηση ενός συστήματος AHB χωρίς χρήση οδηγών (drivers) τριών καταστάσεων, απαιτείται η ύπαρξη χωριστών διαύλων δεδομένων για εγγραφή και ανάγνωση. Το ελάχιστο πλάτος ενός διαύλου δεδομένων ορίζεται ως 32 bits, αλλά το πλάτος των

διαύλων μπορεί να αυξηθεί αν αυτό είναι απαραίτητο. Καθώς το χαρακτηριστικό αυτό δε βρίσκει εφαρμογή στον παρόντα σχεδιασμό, δεν θα αναλυθεί περαιτέρω.

HWDATA[31:0]

Ο δίαυλος δεδομένων εγγραφής οδηγείται από τον master του διαύλου κατά τη διάρκεια των μεταφορών εγγραφής. Εάν η μεταφορά επεκταθεί, τότε ο master θα πρέπει να διατηρήσει τα δεδομένα σταθερά μέχρι την ολοκλήρωσή της, όπως αυτή δηλώνεται όταν το σήμα **HREADY** γίνεται HIGH.

Όλες οι μεταφορές πρέπει να είναι ευθυγραμμισμένες με το όριο διεύθυνσης που ισούται με το μέγεθος της μεταφοράς. Για παράδειγμα, οι μεταφορές λέξης πρέπει να είναι ευθυγραμμισμένες προς τα όρια διευθύνσεων λέξης (δηλαδή A[1:0] = 00), ενώ οι μεταφορές μισής λέξης πρέπει να είναι ευθυγραμμισμένες προς τα όρια διευθύνσεων μισής λέξης (δηλαδή A[0] = 0).

Για τις μεταφορές που έχουν μικρότερο μέγεθος από το πλάτος του διαύλου, για παράδειγμα μια μεταφορά των 16-bit σε έναν δίαυλο των 32-bit, ο master του διαύλου χρειάζεται να οδηγήσει μόνο τις κατάλληλες λωρίδες byte. O slave είναι υπεύθυνος για την επιλογή των δεδομένων εγγραφής από τις σωστές λωρίδες byte. Οι Πίνακες 6 και 7 των επόμενων σελίδων δείχνουν ποιες λωρίδες δεδομένων είναι ενεργές για ένα σύστημα little-endian και big-endian αντίστοιχα. Αν είναι απαραίτητο, οι πληροφορίες αυτές μπορούν να επεκταθούν και για υλοποιήσεις διαύλων δεδομένων μεγαλύτερου μεγέθους. Οι ακολουθίες μεταφορών με μέγεθος μικρότερο από το πλάτος του διαύλου δεδομένων θα έχουν διαφορετικές ενεργές λωρίδες byte για κάθε στάδιο της ακολουθίας.

Η ενεργή λωρίδα byte είναι ανεξάρτητη από το σχήμα endianess του συστήματος, ωστόσο το πρωτόκολλο AHB δεν καθορίζει το απαραίτητο endianess. Ως εκ τούτου, είναι σημαντικό όλοι οι masters και όλοι οι slaves του συστήματος να χρησιμοποιούν το ίδιο σχήμα endianess.

HRDATA[31:0]

Ο δίαυλος δεδομένων ανάγνωσης οδηγείται από τον κατάλληλο slave κατά τη διάρκεια των μεταφορών ανάγνωσης. Εάν ο slave επεκτείνει τη μεταφορά ανάγνωσης κρατώντας το σήμα **HREADY** σε LOW, τότε ο slave χρειάζεται να παρέχει έγκυρα δεδομένα μόνο στο τέλος του τελικού κύκλου της μεταφοράς, όπως αυτός υποδεικνύεται όταν το **HREADY** είναι HIGH.

Για τις μεταφορές που έχουν μικρότερο μέγεθος από το πλάτος του διαύλου, ο slave χρειάζεται να παρέχει έγκυρα δεδομένα μόνο στις ενεργές λωρίδες byte, όπως αυτές παρουσιάζονται στους Πίνακες 6 και 7. Ο master του διαύλου είναι υπεύθυνος για την επιλογή των δεδομένων από τις σωστές λωρίδες byte.

Ένας slave χρειάζεται να παρέχει έγκυρα δεδομένα μόνο όταν η μεταφορά ολοκληρώνεται με μια απάντηση OKAY. Οι απαντήσεις SPLIT, RETRY και ERROR δεν απαιτούν έγκυρα δεδομένα ανάγνωσης.

Μέγεθος μεταφοράς	**Offset διεύθυνσης**	**DATA[31:24]**	**DATA[23:16]**	**DATA[15:8]**	**DATA[7:0]**
Λέξη	0	√	√	√	√
Μισή λέξη	0	-	-	√	√
Μισή λέξη	2	√	√	-	-
Byte	0	-	-	-	√
Byte	1	-	-	√	-

Byte	2	-	√	-	-
Byte	3	√	-	-	-

Πίνακας 6: Ενεργές λωρίδες byte για little-endian δίαυλο δεδομένων των 32-bit

Μέγεθος μεταφοράς	Offset διεύθυνσης	DATA[31:24]	DATA[23:16]	DATA[15:8]	DATA[7:0]
Λέξη	0	√	√	√	√
Μισή λέξη	0	√	√	-	-
Μισή λέξη	2	-	-	√	√
Byte	0	√	-	-	-
Byte	1	-	√	-	-
Byte	2	-	-	√	-
Byte	3	-	-	-	√

Πίνακας 7: Ενεργές λωρίδες byte για big-endian δίαυλο δεδομένων των 32-bit

Σχήμα endianess

Προκειμένου ένα σύστημα να λειτουργεί σωστά, είναι βασικό όλες οι υπομονάδες να χρησιμοποιούν το ίδιο σχήμα endianess, καθώς επίσης όλες οι δρομολογήσεις δεδομένων και οι γέφυρες να είναι ίδιου endianess. Για τον παρουσιαζόμενο σχεδιασμό έχει χρησιμοποιηθεί σχήμα little-endian.

2.4.11.Διαιτησία (Arbitration)

Ο μηχανισμός διαιτησίας (arbitration mechanism) χρησιμοποιείται για να εξασφαλιστεί ότι ανά πάσα στιγμή μόνο ένας master έχει πρόσβαση στον δίαυλο. Ο arbiter εκτελεί αυτή τη λειτουργία παρακολουθώντας έναν αριθμό διαφορετικών αιτήσεων για χρήση του διαύλου και αποφασίζοντας ποιος από τους master που ζητάνε τον δίαυλο τη συγκεκριμένη στιγμή έχει τη μεγαλύτερη προτεραιότητα. Ο arbiter λαμβάνει επίσης αιτήσεις από slaves που επιθυμούν να ολοκληρώσουν μεταφορές SPLIT.

Περιγραφή σημάτων

Τα σήματα που χρησιμοποιούνται από τον μηχανισμό διαιτησίας περιγράφονται παρακάτω:

HBUSREQx Το σήμα αίτησης του διαύλου χρησιμοποιείται για να ζητήσει ένας master πρόσβαση στον δίαυλο. Κάθε master του διαύλου έχει το δικό του σήμα **HBUSREQx** προς τον arbiter και μπορούν να υπάρχουν μέχρι 16 ξεχωριστοί masters σε κάθε σύστημα.

HLOCKx Το σήμα κλειδώματος τίθεται από έναν master μαζί με το σήμα αίτησης του διαύλου. Αυτό δηλώνει στον arbiter ότι ο master θα πραγματοποιήσει μια σειρά αδιάσπαστων μεταφορών και ο arbiter δεν πρέπει να

παραχωρήσει τον δίαυλο σε οποιονδήποτε άλλο master αφότου ξεκινήσει η πρώτη μεταφορά της ακολουθίας κλειδωμένων μεταφορών. Το σήμα **HLOCKx** πρέπει να τεθεί τουλάχιστον έναν κύκλο πριν τη διεύθυνση στην οποία αναφέρεται, έτσι ώστε να απαγορεύσει στον arbiter να αλλάξει τα σήματα παραχώρησης του διαύλου.

HGRANTx Τα σήματα παραχώρησης δημιουργούνται από τον arbiter και δηλώνουν ότι ο κατάλληλος master είναι τη στιγμή αυτή ο master με τη μεγαλύτερη προτεραιότητα που ζητάει τον δίαυλο, λαμβάνοντας υπ'όψιν κλειδωμένες μεταφορές και μεταφορές SPLIT.
Ένας master αποκτά τον έλεγχο του διαύλου διευθύνσεων όταν το **HGRANTx** είναι HIGH και το **HREADY** είναι HIGH κατά τη θετική ακμή του **HCLK**.

HMASTER[3:0] Ο arbiter δηλώνει σε ποιον master παραχωρείται τη στιγμή αυτή ο δίαυλος χρησιμοποιώντας τα σήματα **HMASTER[3:0]** και αυτό μπορεί να χρησιμοποιηθεί για τον έλεγχο του κεντρικού πολυπλέκτη διευθύνσεων και ελέγχου. Ο αριθμός του master χρησιμοποιείται από slaves που μπορούν να πραγματοποιούν μεταφορές SPLIT ώστε να δηλώσουν στον arbiter ποιος master μπορεί να ολοκληρώσει μια μεταφορά SPLIT.

HMASTLOCK Ο arbiter δηλώνει ότι η τρέχουσα μεταφορά αποτελεί τμήμα μιας κλειδωμένης ακολουθίας θέτοντας το σήμα **HMASTLOCK**, το οποίο έχει τον ίδιο χρονισμό με τα σήματα διεύθυνσης και ελέγχου.

HSPLIT[15:0] Ο 16-bit δίαυλος Ολοκλήρωσης Μεταφοράς SPLIT χρησιμοποιείται από έναν ικανό για μεταφορές SPLIT slave για να δηλώσει ποιος master μπορεί να ολοκληρώσει μια μεταφορά SPLIT. Την πληροφορία αυτή τη χρειάζεται ο arbiter για να παραχωρήσει τον έλεγχο του διαύλου στον εν λόγω master έτσι ώστε να ολοκληρώσει τη μεταφορά.

Αίτηση πρόσβασης στον δίαυλο

Ένας master χρησιμοποιεί το σήμα **HBUSREQx** για να ζητήσει τον έλεγχο του διαύλου και αυτό μπορεί να το κάνει κατά τη διάρκεια οποιουδήποτε κύκλου. Ο arbiter θα δειγματοληπτήσει την αίτηση αυτή στη θετική ακμή του ρολογιού και έπειτα θα χρησιμοποιήσει έναν εσωτερικό αλγόριθμο προτεραιότητας προκειμένου να αποφασίσει ποιος master θα αποκτήσει στη συνέχεια τον έλεγχο του διαύλου.

Κανονικά ο arbiter θα παραχωρήσει τον έλεγχο του διαύλου σε έναν διαφορετικό master μόνο κατά την ολοκλήρωση μιας ακολουθίας μεταφορών. Ωστόσο, αν απαιτείται, ο arbiter μπορεί να τερματίσει νωρίτερα μια ακολουθία ώστε να παραχωρήσει τον έλεγχο του διαύλου σε έναν master υψηλότερης προτεραιότητας.

Αν ένας master ζητάει κλειδωμένες προσπελάσεις, θα πρέπει να θέσει επίσης το σήμα **HLOCKx** για να δηλώσει στον arbiter ότι ο δίαυλος δεν θα πρέπει να παραχωρηθεί σε κανέναν άλλο master.

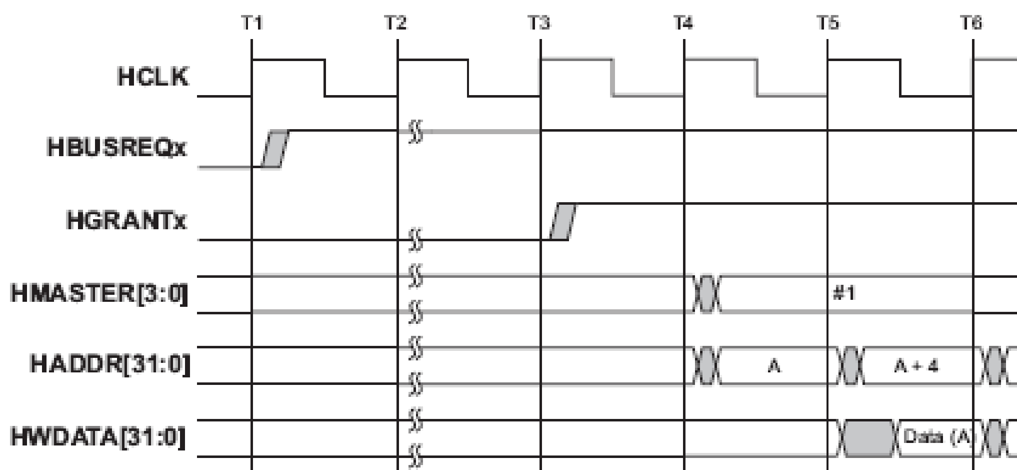
Για ακολουθίες αόριστου μήκους ο master πρέπει να συνεχίζει να θέτει το σήμα αίτησης έως ότου ξεκινήσει την τελευταία μεταφορά. Ο arbiter δεν μπορεί να προβλέψει πότε να αλλάξει τη διαιτησία στο τέλος μιας ακολουθίας αόριστου μήκους.

Είναι δυνατόν ένας master να αποκτήσει πρόσβαση στον δίαυλο ενώ δεν τον ζητάει. Αυτό μπορεί να συμβεί όταν κανένας master δε ζητάει τον δίαυλο και ο arbiter δίνει τον έλεγχο στον προκαθορισμένο master. Συνεπώς, είναι σημαντικό αν ένας master δε ζητάει τον έλεγχο του διαύλου να οδηγεί το σήμα **HTRANS** έτσι ώστε να δηλώνει μεταφορά IDLE.

Παραχώρηση ελέγχου του διαύλου

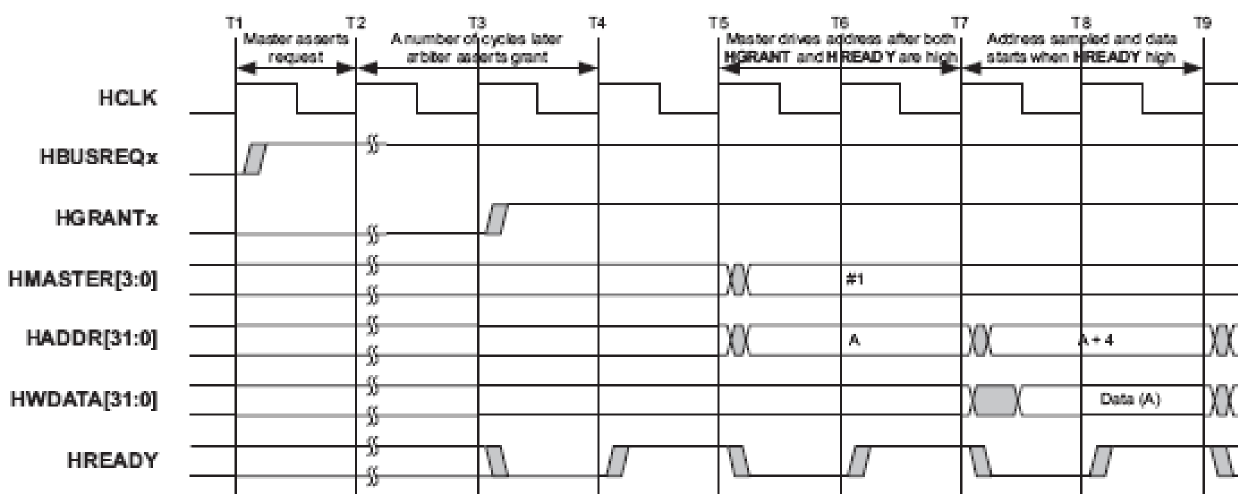
Ο arbiter δηλώνει ποιος master από αυτούς που ζητούν τον δίαυλο έχει αυτή τη στιγμή τη μεγαλύτερη προτεραιότητα θέτοντας το κατάλληλο σήμα **HGRANTx**. Όταν η τρέχουσα μεταφορά ολοκληρωθεί, γεγονός που θα φανεί όταν το **HREADY** γίνει HIGH, τότε ο master θα πάρει τον έλεγχο του διαύλου και ο arbiter θα αλλάξει τα σήματα **HMASTER[3:0]** ώστε να δηλώνουν τον αριθμό του master του διαύλου.

Η Εικόνα 15 δείχνει την παραπάνω διαδικασία όταν όλες οι μεταφορές πραγματοποιούνται χωρίς στάδια αναμονής και το σήμα **HREADY** είναι HIGH.



Εικόνα 15: Παραχώρηση ελέγχου διαύλου χωρίς στάδια αναμονής

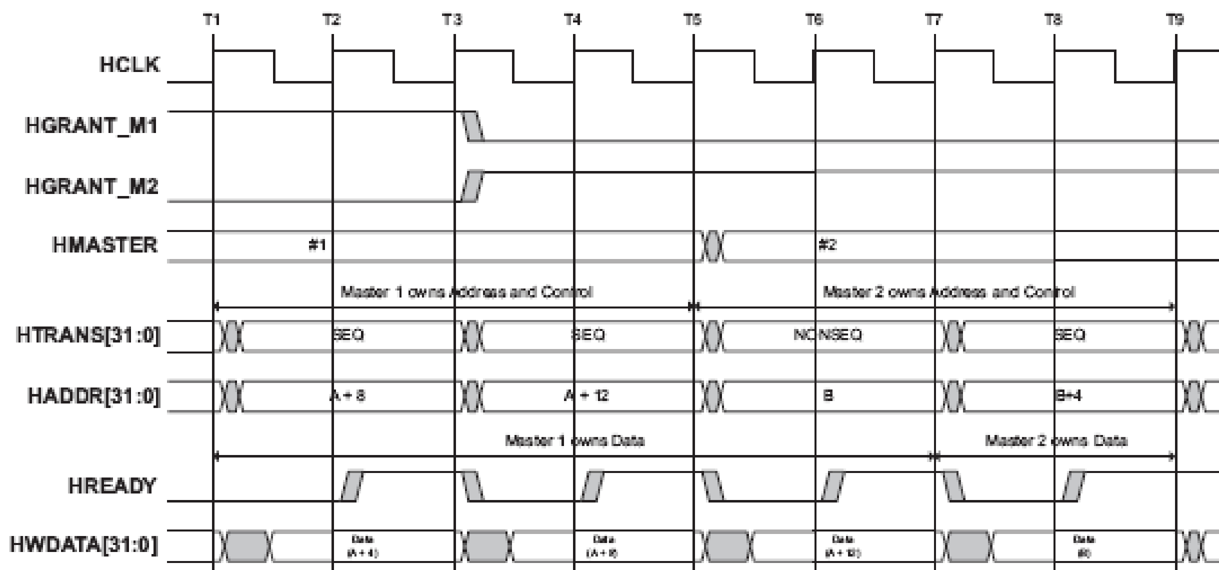
Η Εικόνα 16 δείχνει το αποτέλεσμα που έχουν τα στάδια αναμονής στη διαδικασία παραχώρησης του διαύλου.



Εικόνα 16: Παραχώρηση ελέγχου διαύλου με στάδια αναμονής

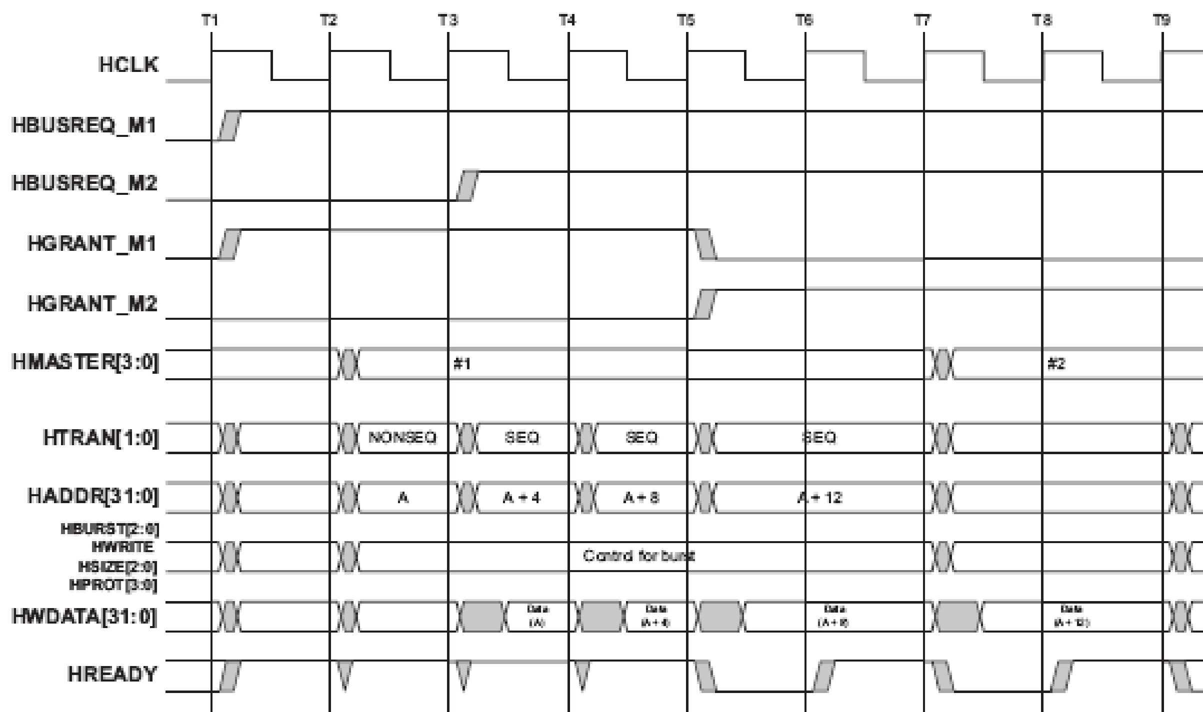
Η παραχώρηση του διαύλου δεδομένων πραγματοποιείται με καθυστέρηση σε σχέση με την παραχώρηση του διαύλου διευθύνσεων. Όταν ολοκληρώνεται μια μεταφορά, δηλαδή έχει

τεθεί το **HREADY** σε HIGH, τότε ο master που έχει τον έλεγχο του διαύλου διευθύνσεων θα μπορεί να χρησιμοποιήσει τον δίαυλο δεδομένων και θα συνεχίσει να έχει τον έλεγχο του διαύλου δεδομένων έως ότου ολοκληρωθεί η μεταφορά. Η Εικόνα 17 δείχνει πώς πραγματοποιείται η μεταφορά του ελέγχου του διαύλου δεδομένων όταν μεταφέρεται ο έλεγχος του διαύλου από τον έναν master στον άλλο.



Εικόνα 17: Μεταφορά ελέγχου του διαύλου δεδομένων

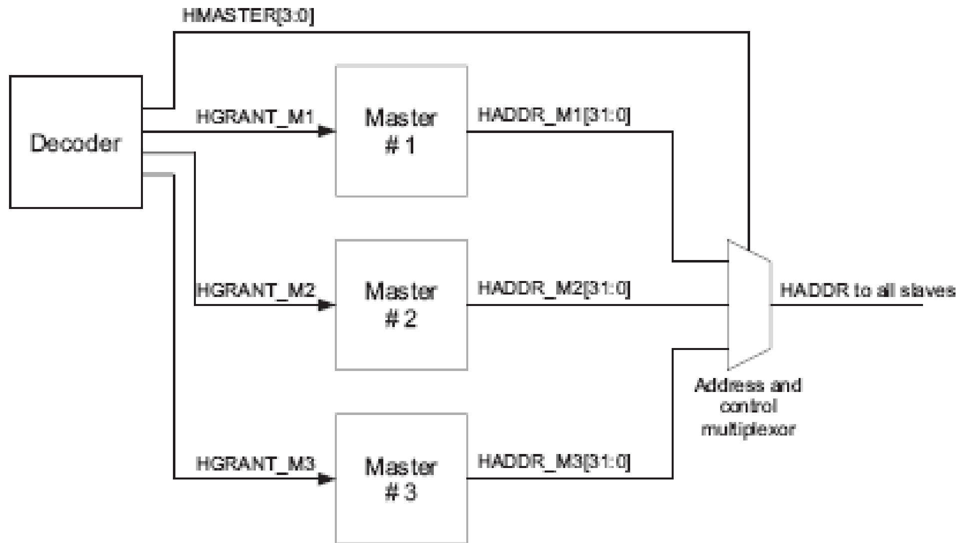
Η Εικόνα 18 δίνει ένα παράδειγμα του πώς ο arbiter μπορεί να αλλάξει τον έλεγχο του διαύλου στο τέλος μιας ακολουθίας μεταφορών.



Εικόνα 18: Μεταφορά ελέγχου του διαύλου έπειτα από μια ακολουθία μεταφορών

Ο arbiter αλλάζει τα σήματα **HGRANTx** όταν έχει δειγματοληπτηθεί η προτελευταία διεύθυνση. Οι νέες πληροφορίες **HGRANTx** θα δειγματοληπτηθούν την ίδια στιγμή που θα δειγματοληπτηθεί και η τελευταία διεύθυνση.

Στην Εικόνα 19 φαίνεται πώς χρησιμοποιούνται τα σήματα **HGRANTx** και **HMASTER** σε ένα σύστημα.



Εικόνα 19: Σήματα παραχώρησης ελέγχου σε master του διαύλου

Σημείωση

Επειδή χρησιμοποιείται ένας κεντρικός πολυπλέκτης, κάθε master μπορεί να οδηγήσει τη διεύθυνση της μεταφοράς που θέλει να πραγματοποιήσει αμέσως και δε χρειάζεται να περιμένει έως ότου αποκτήσει τον έλεγχο του διαύλου. Το σήμα **HGRANTx** χρησιμοποιείται από τον master μόνο για να καθορίσει πότε έχει τον έλεγχο του διαύλου και συνεπώς πότε θα πρέπει να θεωρήσει ότι η διεύθυνση έχει δειγματοληπτηθεί από τον κατάλληλο slave.

Μια καθυστερημένη έκδοση του **HMASTER** χρησιμοποιείται για να ελέγχει τον πολυπλέκτη του διαύλου δεδομένων εγγραφής.

Καθώς το παρουσιαζόμενο σύστημα δε χρησιμοποιεί κλειδωμένες μεταφορές ή μεταφορές SPLIT, η μεταφορά του ελέγχου του διαύλου στις περιπτώσεις αυτές δε θα συζητηθεί.

Προκαθορισμένος master του διαύλου

Κάθε σύστημα πρέπει να περιλαμβάνει έναν προκαθορισμένο master διαύλου ο οποίος θα παίρνει τον έλεγχο του διαύλου όταν όλοι οι υπόλοιποι master αδυνατούν να χρησιμοποιήσουν το δίαυλο. Όταν πάρει τον έλεγχο, ο προκαθορισμένος master πρέπει να πραγματοποιεί μόνο μεταφορές IDLE.

Αν κανένας master δε ζητά τον έλεγχο του διαύλου ο arbiter μπορεί είτε να παραδώσει τον έλεγχο στον προκαθορισμένο master ή εναλλακτικά μπορεί να δώσει τον έλεγχο στον master που θα επωφεληθεί περισσότερο από τη γρήγορη πρόσβαση στον δίαυλο.

Ο έλεγχος του διαύλου πρέπει επίσης να δοθεί στον προκαθορισμένο master όταν όλοι οι υπόλοιποι master αναμένουν την ολοκλήρωση μεταφορών SPLIT.

2.4.12.Σήμα reset

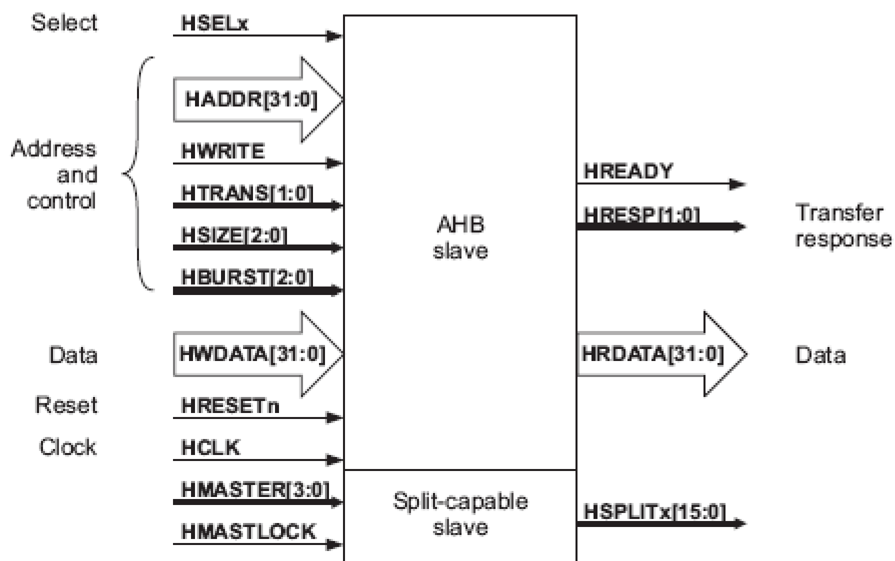
Το σήμα επανεκκίνησης, **HRESETn**, είναι το μόνο ενεργό LOW σήμα στην προδιαγραφή AMBA AHB και αποτελεί το βασικό σήμα επανεκκίνησης για όλα τα στοιχεία του διαύλου. Το σήμα reset μπορεί να τεθεί σε LOW ασύγχρονα, αλλά ξανατίθεται σε HIGH σύγχρονα μετά τη θετική ακμή του **HCLK**. Στον υπό συζήτηση σχεδιασμό αυτό δεν είναι απαραίτητο και το σήμα reset μπορεί να επιστρέψει σε κατάσταση HIGH ασύγχρονα.

Κατά τη διάρκεια του reset όλοι οι master πρέπει να διασφαλίζουν ότι τα σήματα διεύθυνσης και ελέγχου είναι σε έγκυρα επίπεδα και ότι τα σήματα **HTRANS[1:0]** δηλώνουν μεταφορά IDLE.

2.4.13.Slave διαύλου AHB

Ένας slave διαύλου AHB απαντά σε μεταφορές που εκκινούνται από τους masters του συστήματος. Ο slave χρησιμοποιεί ένα σήμα επιλογής **HSELx** από τον αποκωδικοποιητή προκειμένου να καθορίσει πότε πρέπει να απαντήσει σε μία μεταφορά. Τα υπόλοιπα σήματα που απαιτούνται για τη μεταφορά, όπως τα σήματα διεύθυνσης και ελέγχου, δημιουργούνται από τον master.

Στην Εικόνα 20 απεικονίζεται η διεπαφή μιας συσκευής slave του διαύλου AHB.

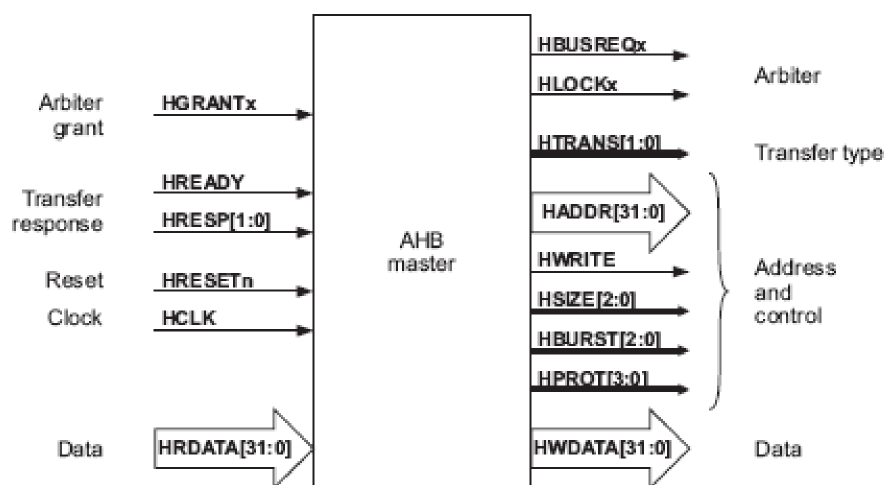


Εικόνα 20: Διεπαφή συσκευής slave του διαύλου AHB

2.4.14.Master διαύλου AHB

Οι συσκευές master του διαύλου AHB έχουν την πιο σύνθετη διεπαφή σε ένα σύστημα AMBA. Τυπικά, ένας σχεδιαστής συστήματος AMBA θα χρησιμοποιούσε προσχεδιασμένους masters και συνεπώς δε θα χρειαζόταν να ασχοληθεί με τις λεπτομέρειες της διεπαφής ενός master. Στην περίπτωση του παρουσιαζόμενου συστήματος, ωστόσο, η χρήση μιας έτοιμης διεπαφής δεν ήταν δυνατή, καθώς η μόνη διαθέσιμη διεπαφή master βασιζόταν σε έναν ελεγκτή DMA για την παραγωγή των διευθύνσεων και δεν ανταποκρινόταν στις ανάγκες του συστήματος, ενώ απουσίαζαν άλλες απαραίτητες λειτουργίες. Έτσι, σχεδιάστηκε μια διεπαφή master από την αρχή, ούτως ώστε να ανταποκρίνεται επακριβώς στις ανάγκες του συστήματος και να αποτελεί έναν συμπαγή και αποδοτικό σχεδιασμό.

Στην Εικόνα 21 απεικονίζεται η διεπαφή μιας συσκευής master του διαύλου AHB.

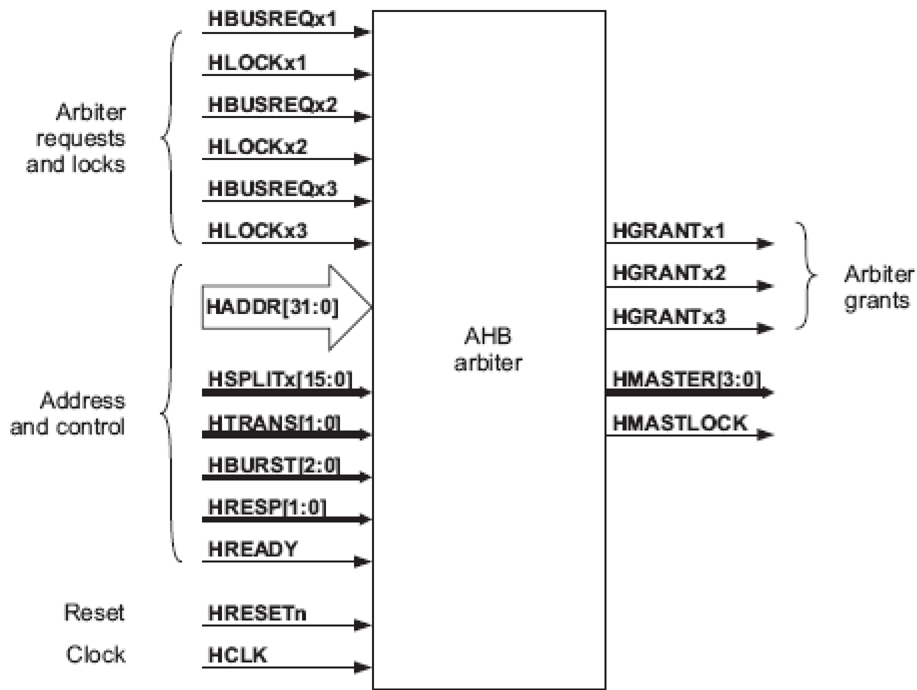


Εικόνα 21: Διεπαφή συσκευής master του διαύλου AHB

2.4.15.Arbiter διαύλου AHB

Ο ρόλος του arbiter σε ένα σύστημα AMBA είναι ο έλεγχος του ποιος master έχει πρόσβαση στον δίαυλο. Κάθε master έχει μια διεπαφή ΑΙΤΗΣΗΣ/ΠΑΡΑΧΩΡΗΣΗΣ με τον arbiter και ο arbiter χρησιμοποιεί ένα σχήμα προτεραιότητας για να αποφασίσει ποιος master από όσους ζητούν τον δίαυλο έχει τη μεγαλύτερη προτεραιότητα.

Στην Εικόνα 22 φαίνεται η διεπαφή ενός arbiter του διαύλου AHB.

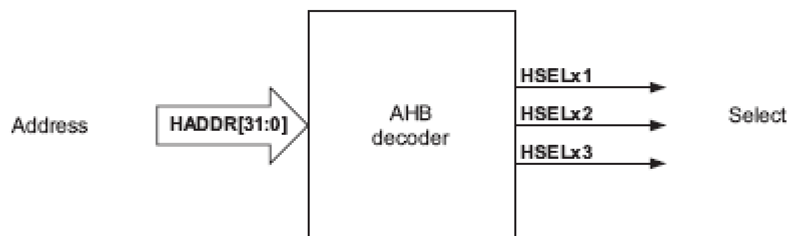


Εικόνα 22: Διεπαφή arbiter του διαύλου AHB

2.4.16.Αποκωδικοποιητής διαύλου AHB

Ο αποκωδικοποιητής ενός συστήματος AMBA χρησιμοποιείται για να εκτελεί μια κεντρικοποιημένη λειτουργία αποκωδικοποίησης, η οποία βελτιώνει τη μεταφερσιμότητα των περιφερειακών, καθιστώντας τα ανεξάρτητα από την αντιστοίχιση χώρου διευθύνσεων του συστήματος.

Στην Εικόνα 23 φαίνεται ένας αποκωδικοποιητής διαύλου AHB.



Εικόνα 23: Διεπαφή αποκωδικοποιητή του διαύλου AHB

Κεφάλαιο 3:

Το πρόβλημα συγχρονισμού των διαύλων και η λειτουργία της FIFO

Στο παρών κεφάλαιο παρουσιάζεται το πρόβλημα του συγχρονισμού των δύο διαύλων (επεξεργαστή και κύριας μνήμης), καθώς λειτουργούν υπό διαφορετικά ρολόγια. Παρουσιάζεται επίσης η λειτουργία της βασικής υπομονάδας που χρησιμοποιείται για την αντιμετώπιση του προβλήματος.

3.1.Εισαγωγή

Τα σύγχρονα συστήματα αποτελούσαν πάντα την επικρατέστερη μέθοδο υλοποίησης στον σχεδιασμό ψηφιακών ηλεκτρονικών κυκλωμάτων. Σχεδόν όλα τα σημερινά συστήματα εφαρμόζουν τεχνικές σύγχρονης σχεδίασης. Ωστόσο, προκειμένου να επιτευχθεί αυτό, πρέπει να παρέχεται επακριβώς ένα γενικό σήμα χρονισμού σε όλες τις περιοχές του κυκλώματος την ίδια ακριβώς στιγμή. Το μέγεθος των τρανζίστορ ολοένα και μικραίνει, ενώ τόσο οι συχνότητες ρολογιών όσο και οι δυνατότητες των εργαλείων αυτοματοποίησης του σχεδιασμού αυξάνονται. Αυτό οδηγεί σε μεγαλύτερες υλοποιήσεις συστημάτων υψηλών ταχυτήτων. Δυστυχώς, η καθυστέρηση των καλωδίων διασύνδεσης δεν έχει καταφέρει να επιδείξει τις ίδιες αυξήσεις στην απόδοση, καθιστώντας έτσι τη διανομή ενός καθολικού σήματος χρονισμού ένα από τα μεγαλύτερα προβλήματα στον σύγχρονο σχεδιασμό συστημάτων.

Μια λύση για την αντιμετώπιση του προβλήματος αυτού είναι η χρήση *αυτοχρονιζόμενων* ή *ασύγχρονων* κυκλωμάτων. Ωστόσο, πολλαπλά προβλήματα εμποδίζουν μια γενική μετάβαση σε αυτήν τη μέθοδο σχεδιασμού, συμπεριλαμβανομένης της έλλειψης ώριμων εργαλείων σχεδιασμού για σύγχρονα συστήματα καθώς και την απροθυμία της βιομηχανίας να αναλάβει το κόστος και το ρίσκο της απομάκρυνσης από ένα σχεδιαστικό στυλ που έχει αποδειχθεί τόσο επιτυχές στο παρελθόν. Μια εναλλακτική προσέγγιση είναι η δημιουργία συστημάτων που αναμειγνύουν σύγχρονες και ασύγχρονες τεχνικές σχεδιασμού. Αυτό το στυλ σχεδιασμού αναφέρεται ως Καθολικά Ασύγχρονος Τοπικά Σύγχρονος (Globally Asynchronous Locally Synchronous – GALS) σχεδιασμός. Σε αυτή τη σχεδιαστική προσέγγιση τα τοπικά μπλοκς δημιουργούνται χρησιμοποιώντας παραδοσιακές τεχνικές σύγχρονου σχεδιασμού, αλλά αυτά τα σύγχρονα μπλοκ δεν μοιράζονται κάποια καθολική πληροφορία χρονισμού αλλά λειτουργούν ασύγχρονα το ένα σε σχέση με το άλλο.

Δυστυχώς, ενώ είναι συχνά πολύ βολικός ο διαχωρισμός ενός συστήματος σε διάφορα υποτμήματα, δεν είναι πολύ πιθανό αυτά τα υποτμήματα να είναι εντελώς αυτόνομα. Συνεπώς, απαιτείται γενικά μεταφορά δεδομένων μεταξύ τοπικών σύγχρονων μπλοκς. Η ασφαλής (δηλαδή χωρίς αλλοίωση ή απώλεια δεδομένων) και αποδοτική (δηλαδή ελαχιστοποιώντας το μέγεθος και την απώλεια ενέργειας) επίτευξη αυτού του στόχου είναι μία από τις πρωταρχικές προκλήσεις στους σχεδιασμούς τόπου GALS.

Μια δομή που παρουσιάζεται ιδιαίτερα κατάλληλη για το καθήκον αυτό είναι η δομή μνήμης First-In First-Out (FIFO). Όπως δηλώνεται και από το όνομα, τα αντικείμενα δεδομένων

μετακινούνται μέσα στη δομή με έναν προκαθορισμένο τρόπο, σύμφωνα με τον οποίο το πρώτο αντικείμενο δεδομένων που θα εισαχθεί στη δομή θα είναι και το πρώτο αντικείμενο δεδομένων που θα εξαχθεί από τη δομή. Μια βασική αρχιτεκτονική FIFO μπορεί να τροποποιηθεί ούτως ώστε να λειτουργεί χρησιμοποιώντας δύο ανεξάρτητα μεταξύ τους σήματα χρονισμού. Τα δεδομένα που διέρχονται μέσα από τη FIFO θα εισέρχονται χρησιμοποιώντας το ένα ρολόι και θα εξέρχονται χρησιμοποιώντας το άλλο. Με τον τρόπο αυτό, τα δεδομένα μπορούν να περάσουν ασφαλώς από μία περιοχή ρολογιού (clock domain) σε άλλη.

3.2.Η διπλή λειτουργία της FIFO

Ο συγχρονισμός των δύο διαύλων θα μπορούσε να αντιμετωπιστεί με χρήση μιας FIFO μίας μόνο θέσης, δηλαδή έναν καταχωρητή και την απαραίτητη περιφερειακή λογική συγχρονισμού. Σε αυτόν τον καταχωρητή θα γραφόταν οποιαδήποτε αίτηση απευθυνόταν προς την κύρια μνήμη (είτε αίτηση για μεταφορά ενός μπλοκ στην κρυφή μνήμη είτε αίτηση για εγγραφή/ανανέωση της κύριας μνήμης) σύγχρονα με το ρολόι του διαύλου του επεξεργαστή (P-bus). Ακολούθως, όταν η διεπαφή master της κρυφής μνήμης δευτέρου επιπέδου θα ήταν σε θέση να εξυπηρετήσει την αίτηση αυτή, θα τη διάβαζε σύγχρονα με το ρολόι του διαύλου της μνήμης (M-bus) και θα την εξυπηρετούσε, ελευθερώνοντας τον καταχωρητή για εγγραφή της επόμενης αίτησης.

Η προσέγγιση όμως αυτή παρουσιάζει αρκετά προβλήματα/περιορισμούς. Από τη μια πλευρά, η κατάσταση του καταχωρητή θα μεταβαλλόταν σε κάθε ανάγνωση ή εγγραφή από <άδειος> σε <γεμάτος>. Όπως θα περιγραφεί παρακάτω, η ενημέρωση των περιφερειακών υπομονάδων της κρυφής μνήμης για καθεμία από αυτές τις καταστάσεις χρειάζεται δύο κύκλους ρολογιού, πράγμα που θα επιβάρυνε χρονικά το σύστημα, αφού θα απαιτούσε πάντα δύο κύκλους καθυστέρησης, είτε για εγγραφή στον καταχωρητή είτε για ανάγνωσή του. Ένας ακόμα πιο βασικός περιορισμός όμως προκύπτει από το γεγονός ότι στοχεύουμε στη χρήση επεξεργαστή που υποστηρίζει την παράλληλη εκτέλεση πολλαπλών νημάτων. Έτσι, εάν υποθέσουμε ότι κάποιο νήμα περιμένει για την εκτέλεση κάποιας λειτουργίας (μεταφορά από την κύρια μνήμη ή εγγραφή στην κρυφή και κύρια μνήμη), τα υπόλοιπα νήματα που θα χρειαζόταν τη χρήση του καταχωρητή είναι υποχρεωμένα να περιμένουν πριν καν δηλώσουν την πρόθεσή τους. Ακόμα και μία εγγραφή δε θα μπορούσε να πραγματοποιηθεί, εφόσον η τακτική write-through που ακολουθείται στην κρυφή μνήμη (βλ. κεφάλαιο 5) θα απαιτούσε τη χρήση του καταχωρητή για να προωθηθεί η αίτηση εγγραφής στην κύρια μνήμη. Η χρήση ωστόσο μιας ολοκληρωμένης FIFO αμβλύνει σε μεγάλο βαθμό τα προβλήματα αυτά.

Για να γίνει πιο κατανοητή η χρησιμότητα της FIFO (πέρα από τη βασική λειτουργία του συγχρονισμού), ας θεωρήσουμε την παρακάτω περίπτωση. Έστω ότι ο επεξεργαστής υποστηρίζει την παράλληλη εκτέλεση 5 νημάτων και ένας scheduler καθορίζει τη σειρά εκτέλεσής τους. Έστω επίσης ότι το νήμα 1 απαιτεί προσπέλαση στην κύρια μνήμη (ανάγνωση ή εγγραφή), οπότε η αίτησή του αυτή αποθηκεύεται στον καταχωρητή, αναμένοντας την εξυπηρέτησή της όταν ολοκληρωθεί η προηγούμενη λειτουργία στον δίαυλο μνήμης και αποκτηθεί και πάλι ο έλεγχός του. Ο scheduler θα παραχωρούσε ακολούθως τη χρήση των επεξεργαστικών στοιχείων στο δεύτερο νήμα. Έστω επίσης ότι τα νήματα 2 και 3 χρειάζονται επίσης κάποια μεταφορά ή εγγραφή. Στην περίπτωση ενός απλού καταχωρητή θα έπαιρναν μια απάντηση που θα τους σηματοδοτούσε απλώς να ξαναπροσπαθήσουν αργότερα και η εκτέλεση θα προχωρούσε στα νήματα 4 και 5, τα οποία δεν απαιτούν πρόσβαση στη μνήμη και συνεπώς εκτελούνται για όλο το διάστημα που τους αναλογεί. Αν στο διάστημα αυτό, η προηγούμενη λειτουργία που πραγματοποιούνταν στον δίαυλο M-bus είχε ολοκληρωθεί, τότε στο επόμενο «πέρασμα» του

scheduler από το νήμα 2, αυτό θα αποθήκευε απλώς την αίτησή του στον καταχωρητή και η εκτέλεση θα προχωρούσε στο νήμα 3, το οποίο θα ήταν και πάλι υποχρεωμένο να περιμένει χωρίς καμία πρόοδο.

Ας θεωρήσουμε τώρα την αντίστοιχη περίπτωση με χρήση μιας ολοκληρωμένης FIFO. Στην περίπτωση αυτή, από το πρώτο κιόλας «πέρασμα» του scheduler, τα νήματα 1, 2 και 3 θα μπορούσαν να αποθηκεύσουν τις αιτήσεις τους στη FIFO της κρυφής μνήμης. Στη συνέχεια, παράλληλα με την εκτέλεση των νημάτων 4 και 5, θα πραγματοποιούνταν η κατά σειρά εξυπηρέτηση των αιτήσεων αυτών. Στο επόμενο «πέρασμα» του scheduler, είναι πιθανό η εξυπηρέτηση των αιτήσεων των νημάτων 1, 2 και 3 (ή έστω κάποιων από αυτές) να έχει ολοκληρωθεί και η εκτέλεσή τους θα μπορεί να συνεχιστεί χωρίς καμία επιπλέον καθυστέρηση. Σε οποιαδήποτε περίπτωση, τα νήματα δε θα ήταν υποχρεωμένα να περιμένουν τον χρόνο ενός ολόκληρου «περάσματος» του scheduler προκειμένου να αποθηκεύσουν στον καταχωρητή μία και μόνο αίτηση.

Το παραπάνω πρόβλημα θα εμφανιζόταν όλο και χειρότερο με την αύξηση του αριθμού των νημάτων. Για τους λόγους αυτούς υλοποιήθηκε μια πλήρως λειτουργική FIFO, η οποία επιτελεί διπλή λειτουργία. Από τη μια πλευρά, πραγματοποιεί τον συγχρονισμό των δύο διαύλων (P-bus και M-bus), με λογική ανάλογη με αυτή που θα χρησιμοποιούνταν και στην περίπτωση ενός καταχωρητή. Από την άλλη, λειτουργεί ως buffer για τις αιτήσεις των νημάτων, ούτως ώστε να αποφεύγονται πολυάριθμοι «νεκροί» κύκλοι ρολογιού ανάμεσα στην εξυπηρέτηση μιας αίτησης και την αποθήκευση της επόμενης.

Οι βασικές αρχές λειτουργίας της FIFO αυτής παρουσιάζονται παρακάτω. Η υλοποίηση σε γλώσσα VHDL της ακόλουθης λογικής περιγράφεται στο κεφάλαιο 5.

3.3.Τεχνικές σχεδιασμού FIFO

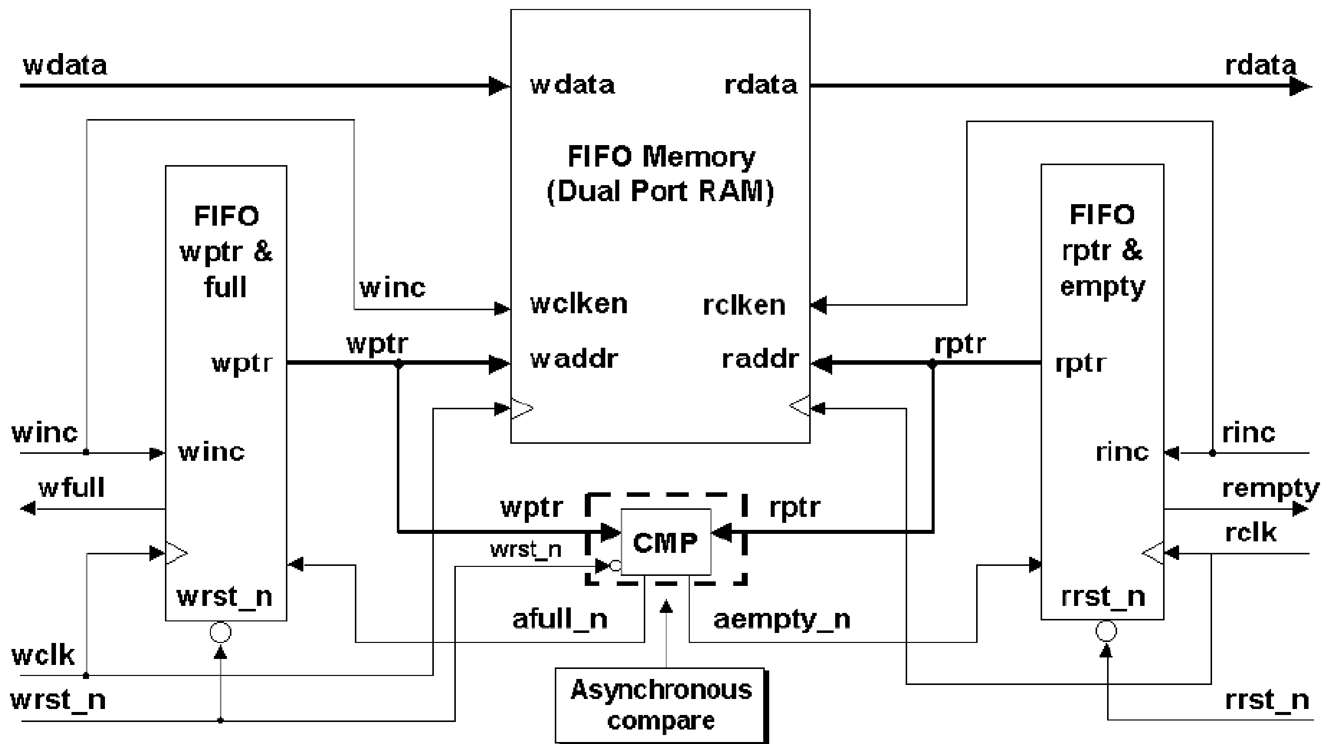
Ασύγχρονη FIFO είναι ένας σχεδιασμός FIFO στον οποίο τα δεδομένα γράφονται σειριακά σε έναν buffer FIFO χρησιμοποιώντας ένα ρολόι, ενώ διαβάζονται σειριακά από τον ίδιο buffer FIFO χρησιμοποιώντας κάποιο άλλο ρολόι, ενώ τα δύο ρολόγια είναι ασύγχρονα το ένα προς το άλλο.

Μία διαδεδομένη τεχνική σχεδιασμού μιας ασύγχρονης FIFO είναι η χρήση δεικτών Gray, συγχρονισμένων στο αντίθετο ρολόι, για την παραγωγή σύγχρονων σημάτων κατάστασης <γεμάτη> ή <άδεια>. Μια διαφορετική αλλά πολύ ενδιαφέρουσα προσέγγιση για τη δημιουργία των σημάτων <γεμάτη> και <άδεια> είναι να γίνεται πρώτα μια ασύγχρονη σύγκριση των δεικτών και επακόλουθα να τίθενται σύγχρονα τα κατάλληλα bits κατάστασης <γεμάτη> ή <άδεια>.

Αυτή η τελευταία προσέγγιση χρησιμοποιείται στον παρόντα σχεδιασμό και περιγράφεται παρακάτω.

3.4.Βασική μορφή FIFO

Το μπλοκ διάγραμμα της FIFO που χρησιμοποιείται φαίνεται στην Εικόνα 24.



Εικόνα 24: Μπλοκ διάγραμμα FIFO

Το μπλοκ FIFO memory αποτελεί τον buffer αποθήκευσης της FIFO, ο οποίος προσπελάζεται και από τα δύο πεδία ρολογιού, δηλαδή γράφεται με το ένα ρολόι και διαβάζεται με το άλλο. Δεικτοδοτείται από δύο δείκτες, έναν για εγγραφή και έναν για ανάγνωση, οι οποίοι διατρέχουν κυκλικά όλες τις θέσεις αποθήκευσης του buffer.

Η λογική ασύγχρονης σύγκρισης παράγει σήματα που χρησιμοποιούνται για τη ρύθμιση των ασύγχρονων bits <γεμάτης> και <άδειας> κατάστασης. Περιέχει μόνο συνδυαστική λογική σύγκρισης, δεν περιλαμβάνει καθόλου ακολουθιακή λογική.

Το μπλοκ δείκτη ανάγνωσης και άδειας κατάστασης (FIFO rptr&empty) είναι σύγχρονο προς το ρολόι ανάγνωσης και περιέχει τη λογική για μεταβολή του δείκτη ανάγνωσης και της σημαίας <άδειας> κατάστασης (βάσει του αποτελέσματος του συγκριτή). Το σήμα aempty_n (input προς το μπλοκ αυτό) τίθεται στο 1 σύγχρονα με το ρολόι ανάγνωσης, εφόσον μπορεί να τεθεί μόνο με την αύξηση του δείκτη ανάγνωσης, αλλά τίθεται στο 0 όταν αυξάνεται ο δείκτης εγγραφής, πράγμα που γίνεται ασύγχρονα με το ρολόι ανάγνωσης.

Το μπλοκ δείκτη εγγραφής και γεμάτης κατάστασης (FIFO wptr&full) είναι σύγχρονο προς το ρολόι εγγραφής και περιέχει τη λογική για μεταβολή του δείκτη εγγραφής και της σημαίας <γεμάτης> κατάστασης (βάσει του αποτελέσματος του συγκριτή). Το σήμα aempty_n (input προς το μπλοκ αυτό) τίθεται στο 1 σύγχρονα με το ρολόι εγγραφής, εφόσον μπορεί να τεθεί μόνο με την αύξηση του δείκτη εγγραφής, αλλά τίθεται στο 0 όταν αυξάνεται ο δείκτης ανάγνωσης, πράγμα που γίνεται ασύγχρονα με το ρολόι εγγραφής.

Δύο βασικά προβλήματα θα πρέπει να αντιμετωπιστούν για τη σωστή λειτουργία της παραπάνω λογικής. Το πρώτο πρόβλημα είναι η ασύγχρονη σύγκριση των δεικτών εγγραφής και ανάγνωσης,

3.5.Μετρητής κώδικα Gray

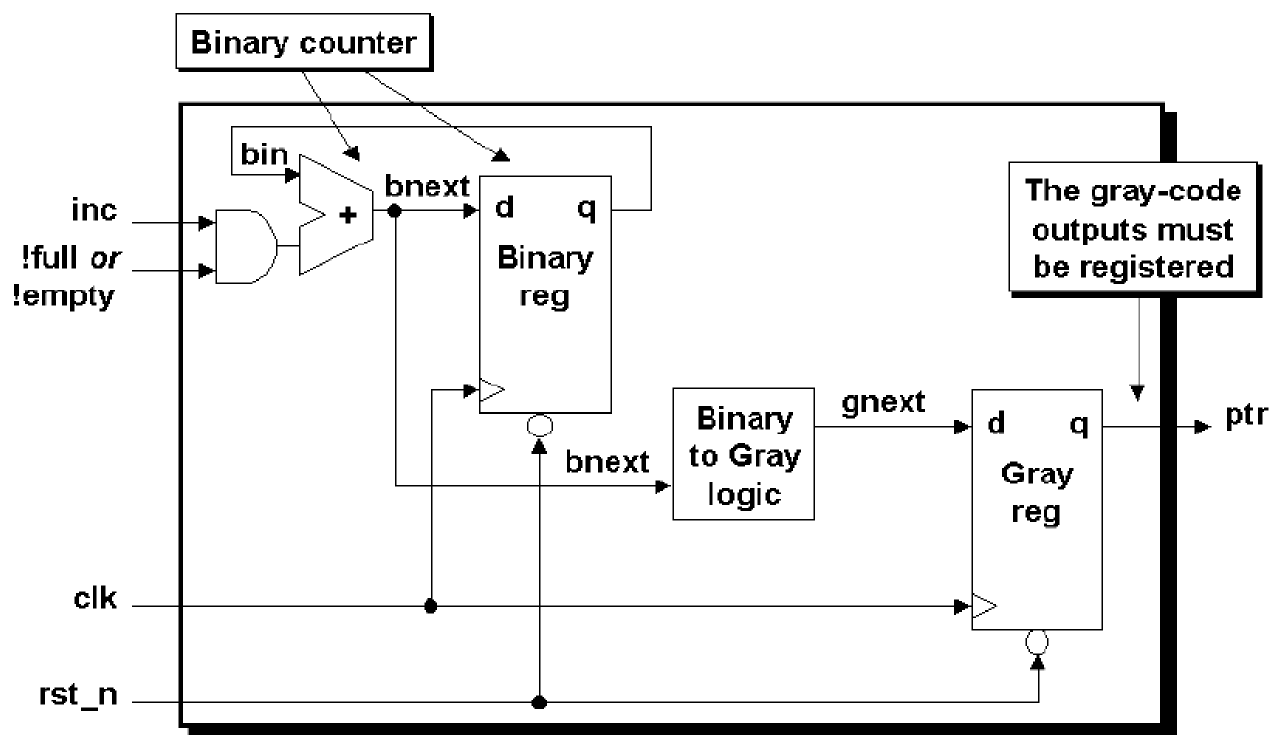
Η σύγκριση των δύο ασύγχρονων δυαδικών δεικτών μπορεί να οδηγήσει σε ανακριβή στιγμιαία αποτελέσματα αποκωδικοποίησης όταν ένας ή και οι δύο μετρητές μεταβάλλουν περισσότερα από ένα bits πάνω-κάτω ταυτόχρονα. Η λύση που εφαρμόζεται στο υπό συζήτηση σύστημα είναι η χρήση ενός μετρητή κώδικα Gray, στον οποίο μόνο ένα δυαδικό ψηφίο μεταβάλλεται από κάθε αριθμό στον επόμενο. Κάθε αποκωδικοποιητής ή συγκριτής θα μεταβαίνει τότε από το ένα έγκυρο αποτέλεσμα εξόδου στο επόμενο, χωρίς τον κίνδυνο ψευδών παρασιτικών σημάτων.

Μια μορφή μετρητή κώδικα Gray χρησιμοποιεί ένα σετ από flip-flops για τον καταχωρητή κώδικα Gray και την απαραίτητη περιφερειακή λογική για μετατροπή από Gray σε δυαδικό σύστημα, αύξηση της δυαδικής αναπαράστασης και μετατροπή από δυαδικό σύστημα σε κώδικα Gray.

Μια δεύτερη μορφή μετρητή κώδικα Gray, η οποία και χρησιμοποιείται στον παρόντα σχεδιασμό, χρησιμοποιεί δύο σετ καταχωρητών, ένα για τον δυαδικό μετρητή και ένα δεύτερο για αποθήκευση της μετατρεπόμενης από-δυαδικό-σε-Gray τιμής. Σκοπός της μορφής αυτής είναι να χρησιμοποιηθεί η δομή δυαδικού κρατουμένου, να απλοποιηθεί η μετατροπή από-Gray-σε-δυαδικό, να περιοριστεί η συνδυαστική λογική και να αυξηθεί η μέγιστη συχνότητα λειτουργίας του μετρητή κώδικα Gray.

Ο δυαδικός μετρητής αυξάνει υπό συνθήκη τη δυαδική τιμή, η οποία οδηγείται τόσο στην είσοδο του δυαδικού καταχωρητή ως η επόμενη δυαδική τιμή, όσο και στην απλή λογική μετατροπής από-δυαδικό-σε-Gray, που αποτελείται από μια πύλη XOR δύο εισόδων για κάθε bit. Η έξοδος της λογικής μετατροπής αποτελεί την επόμενη τιμή Gray και οδηγεί την είσοδο του καταχωρητή κώδικα Gray.

Η Εικόνα 25 δείχνει το μπλοκ διάγραμμα για των n-bit μετρητή κώδικα Gray που μόλις περιγράφηκε.



Εικόνα 25: Μετρητής κώδικα Gray

Η υλοποίηση αυτή απαιτεί τον διπλάσιο αριθμό flip-flops, αλλά μειώνει τη συνδυαστική λογική και μπορεί να λειτουργήσει σε υψηλότερες συχνότητες. Σε σχεδιασμούς FPGA, η διαθεσιμότητα επιπλέον flip-flops σπάνια αποτελεί πρόβλημα αφού τα FPGA τυπικά περιλαμβάνουν πολύ περισσότερα flip-flops από όσα θα χρησιμοποιήσει ποτέ κάποιος σχεδιασμός. Σε σχεδιασμούς FPGA, η μείωση της συνδυαστικής λογικής συχνά έχει ως αποτέλεσμα σημαντικές βελτιώσεις σε ταχύτητα.

Σημείωση

Εφόσον το πιο σημαντικό bit ενός δυαδικού αριθμού είναι πάντα το ίδιο με το πιο σημαντικό bit της αναπαράστασής του σε κώδικα Gray, ο σχεδιασμός μπορεί να απλοποιηθεί παραπέρα αν χρησιμοποιηθεί το flip-flop του πιο σημαντικού δυαδικού bit ως το flip-flop του πιο σημαντικού bit του κώδικα Gray. Η βελτιστοποίηση αυτή πραγματοποιήθηκε από το εργαλείο Leonardo Spectrum κατά τη σύνθεση του σχεδιασμού.

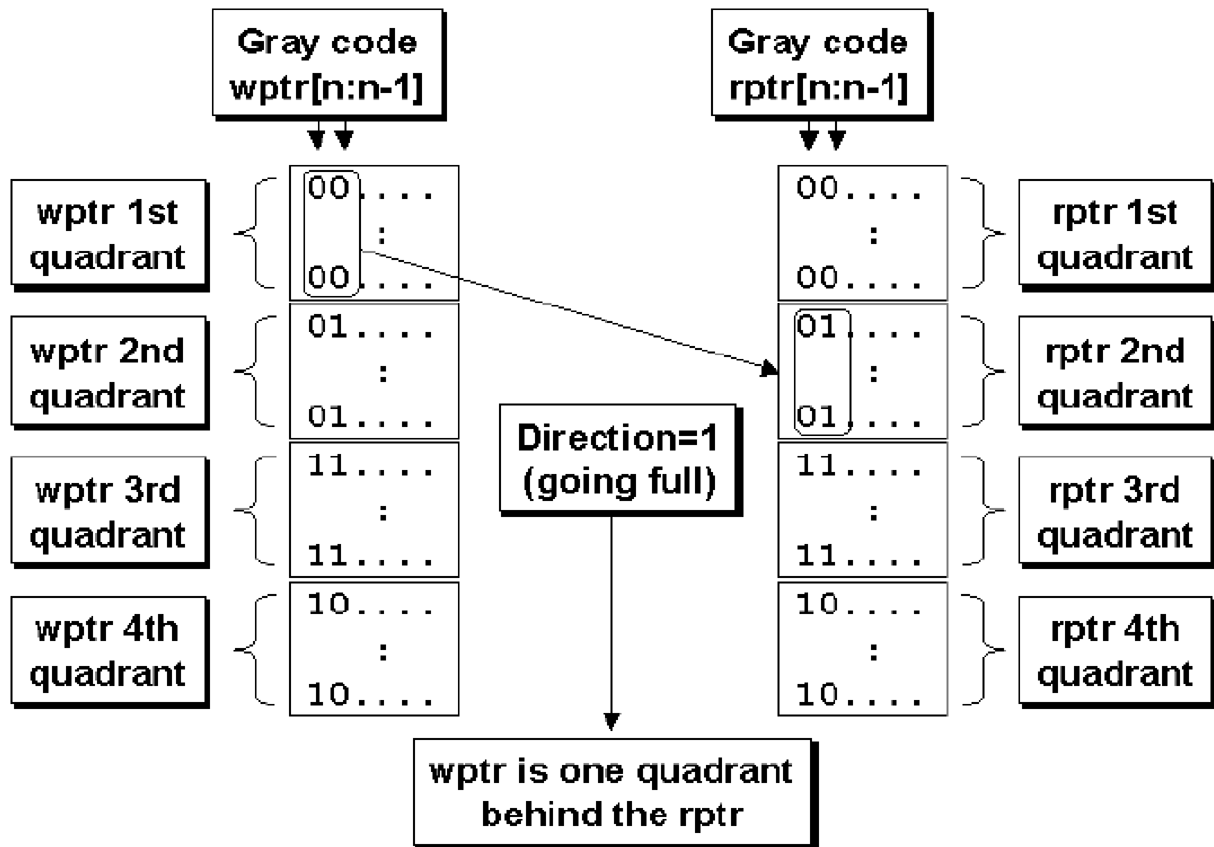
Ο buffer της FIFO δεικτοδοτείται από τη δυαδική αναπαράσταση των δεικτών, ενώ η λογική σύγκρισης δέχεται ως είσοδο την αναπαράσταση σε κώδικα Gray έτσι ώστε να μην παράγει ψευδή παρασιτικά σήματα, όταν πολλαπλά δυαδικά ψηφία των δεικτών αλλάζουν σχεδόν ταυτόχρονα.

3.6.Ανίχνευση <γεμάτης> και <άδειας> κατάστασης

Όπως σε κάθε σχεδιασμό FIFO, η σωστή παραγωγή των σημάτων <γεμάτης> και <άδειας> κατάστασης είναι ένα από τα δυσκολότερα τμήματα του σχεδιασμού.

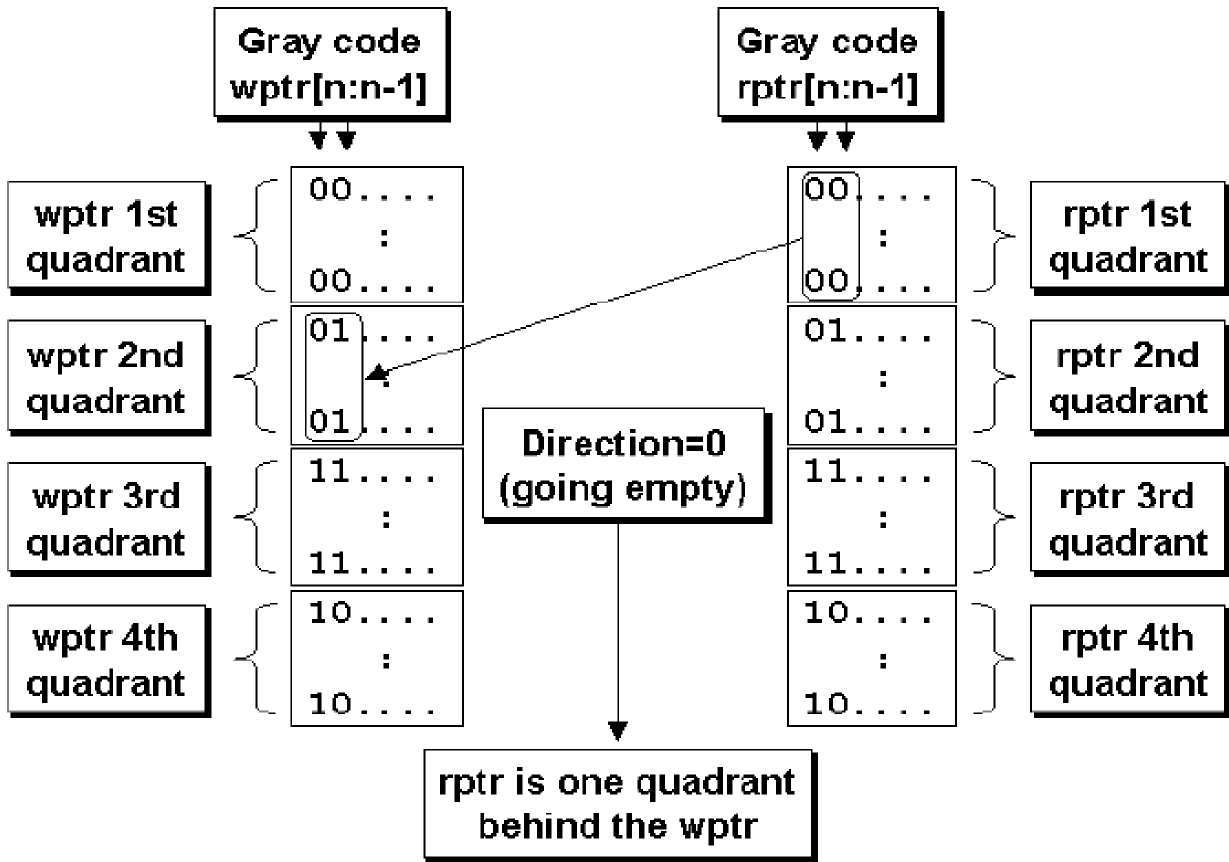
Ένα πρόβλημα που παρουσιάζεται είναι ότι και οι δύο καταστάσεις υποδηλώνονται από το γεγονός ότι οι δύο δείκτες (ανάγνωσης και εγγραφής) είναι ίσοι. Συνεπώς, θα πρέπει να χρησιμοποιηθεί κάποιος άλλος τρόπος για τον διαχωρισμό των δύο καταστάσεων. Σύμφωνα μα μια γνωστή λύση στο πρόβλημα αυτό, προστίθεται ένα επιπλέον bit στους δύο δείκτες και ακολούθως γίνεται σύγκριση των επιπλέον bits για ισότητα (<άδεια> FIFO) ή ανισότητα (<γεμάτη> FIFO), παράλληλα με τη σύγκριση των υπολοίπων bits των δεικτών.

Μια άλλη λύση, που χρησιμοποιείται κι εδώ, είναι να χωριστεί ο χώρος διευθύνσεων των δύο δεικτών σε τέσσερα τεταρτημόρια και να αποκωδικοποιούνται τα δύο πιο σημαντικά bits των δεικτών προκειμένου να αποφασιστεί αν η FIFO έτεινε να γίνει <γεμάτη> ή <άδεια> όταν οι δύο δείκτες έγιναν ίσοι.



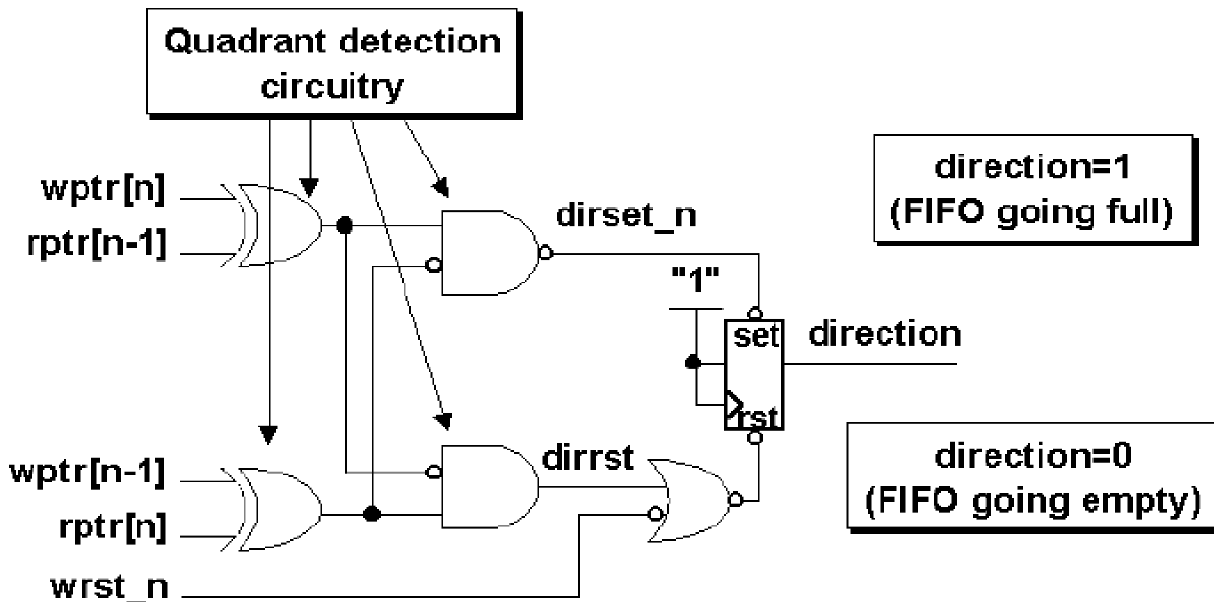
Εικόνα 26: Η FIFO τείνει να γεμίσει γιατί ο δείκτης εγγραφής (wptr) βρίσκεται ένα τεταρτημόριο πίσω από τον δείκτη ανάγνωσης (rptr)

Στην Εικόνα 26 φαίνεται η περίπτωση στην οποία ο δείκτης εγγραφής (wptr) βρίσκεται ένα τεταρτημόριο πίσω από τον δείκτη ανάγνωσης (rptr), πράγμα που δηλώνει ότι η FIFO «τείνει να γεμίσει». Η συνθήκη για να ισχύει αυτό είναι
dirset_n := (w_gray(n-1) xor r_gray(n-2)) and not (r_gray(n-1) xor w_gray(n-2));
, όπου w_gray και r_gray οι αναπαραστάσεις των δεικτών σε κώδικα Gray και n ο αριθμός των bits που χρησιμοποιούν. Όταν συμβαίνει η περίπτωση που μόλις περιγράφηκε, το latch **κατεύθυνσης** της Εικόνας 28 τίθεται στο 1.



Εικόνα 27: Η FIFO τείνει να αδειάσει γιατί ο δείκτης ανάγνωσης (rptr) βρίσκεται ένα τεταρτημόριο πίσω από τον δείκτη εγγραφής (wptr)

Στην Εικόνα 27 φαίνεται η περίπτωση στην οποία ο δείκτης ανάγνωσης (rptr) βρίσκεται ένα τεταρτημόριο πίσω από τον δείκτη εγγραφής (wptr), πράγμα που δηλώνει ότι η FIFO «τείνει να αδειάσει». Η συνθήκη για να ισχύει αυτό είναι
dirreset_n := (not (w_gray(L-1) xor r_gray(L-2)) and (r_gray(L-1) xor w_gray(L-2))) or not RES;
, όπου w_gray και r_gray οι αναπαραστάσεις των δεικτών σε κώδικα Gray, n ο αριθμός των bits που χρησιμοποιούν και RES το σήμα επανεκκίνησης (reset). Όταν συμβαίνει η περίπτωση που μόλις περιγράφηκε, το latch **κατεύθυνσης** της Εικόνας 28 τίθεται στο 0.



Εικόνα 28: Κύκλωμα ανίχνευσης κατεύθυνσης FIFO

Όταν η FIFO επανεκκινείται (reset), το latch **κατεύθυνσης** τίθεται και πάλι στο 0 για να δηλώσει ότι η FIFO «τείνει να αδειάσει» (ουσιαστικά, *είναι* άδεια όταν και οι δυο δείκτες μηδενίζονται σε περίπτωση reset). Η ρύθμιση του latch **κατεύθυνσης** δεν είναι σημαντική χρονικά (timing-critical), ενώ το latch **κατεύθυνσης** επιλύει το πρόβλημα της αβεβαιότητας στη σύγκριση των δύο δεικτών όταν αυτοί είναι ίσοι.

Τα τελικά σήματα aempty_n και afull_n που παράγονται από τη λογική σύγκρισης είναι ασύγχρονα αποκωδικοποιημένα. Το σήμα aempty_n τίθεται στο 1 με μια θετική ακμή του ρολογιού ανάγνωσης, αλλά επιστρέφει στο 0 με μια θετική ακμή του ρολογιού εγγραφής. Αντίστοιχα, το σήμα afull_n τίθεται στο 1 με το ρολόι εγγραφής και στο 0 με το ρολόι ανάγνωσης.

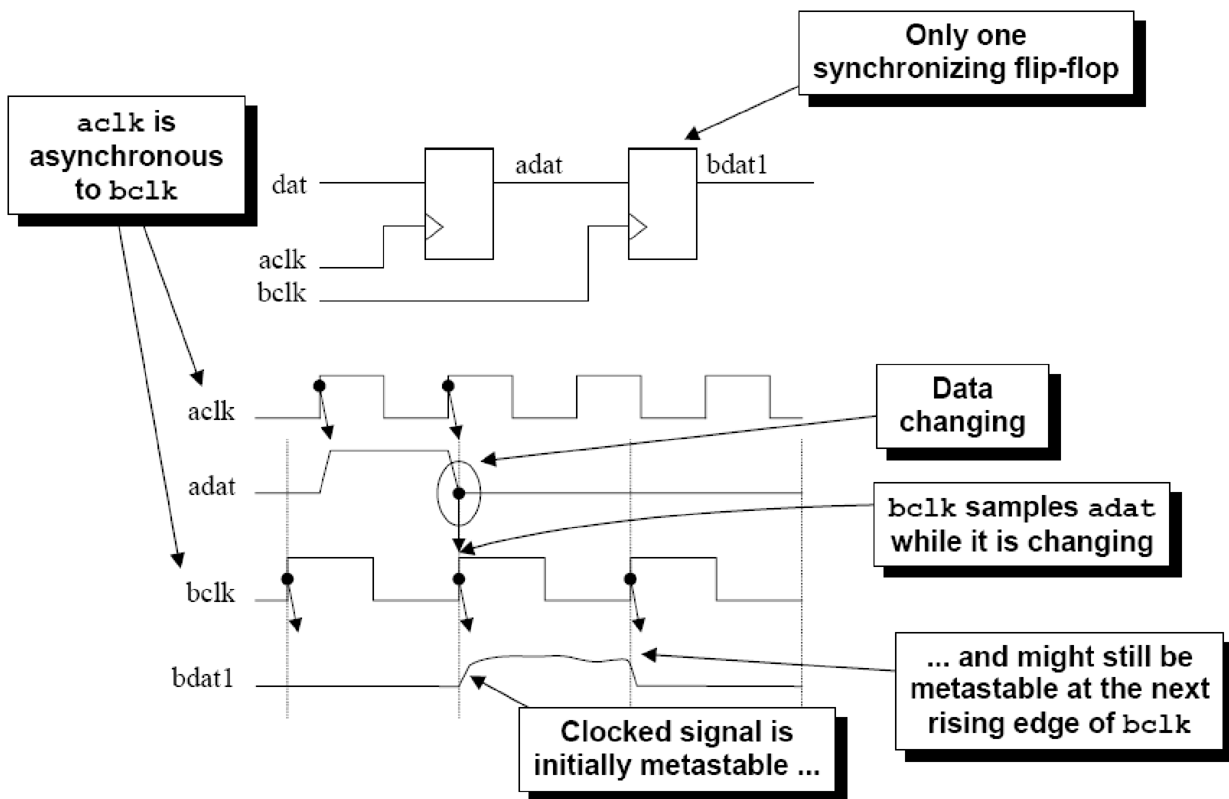
Το τελικό σήμα <άδειας> κατάστασης θα χρησιμοποιηθεί για να αποτρέψει μια μελλοντική λειτουργία ανάγνωσης (αφού η FIFO θα είναι άδεια), και η θετική ακμή του σήματος aempty_n είναι ορθώς σύγχρονη με το ρολόι ανάγνωσης, αλλά η αρνητική ακμή θα πρέπει με κάποιον τρόπο να συγχρονιστεί και αυτή με το ρολόι ανάγνωσης.

Αντίστοιχα, το τελικό σήμα <γεμάτης> κατάστασης θα χρησιμοποιηθεί για να αποτρέψει μια μελλοντική λειτουργία εγγραφής (αφού η FIFO θα είναι γεμάτη), και η θετική ακμή του σήματος afull_n είναι ορθώς σύγχρονη με το ρολόι εγγραφής, αλλά η αρνητική ακμή θα πρέπει με κάποιον τρόπο να συγχρονιστεί και αυτή με το ρολόι εγγραφής.

Αυτός ο συγχρονισμός των τελικών σημάτων <άδειας> και <γεμάτης> κατάστασης είναι και το δεύτερο από τα δύο προβλήματα που αναφέρθηκαν προηγουμένως. Ακολουθεί μια σύντομη περιγραφή του προβλήματος και της λύσης του.

3.7.Μετασταθερότητα (metastability) και συγχρονιστές

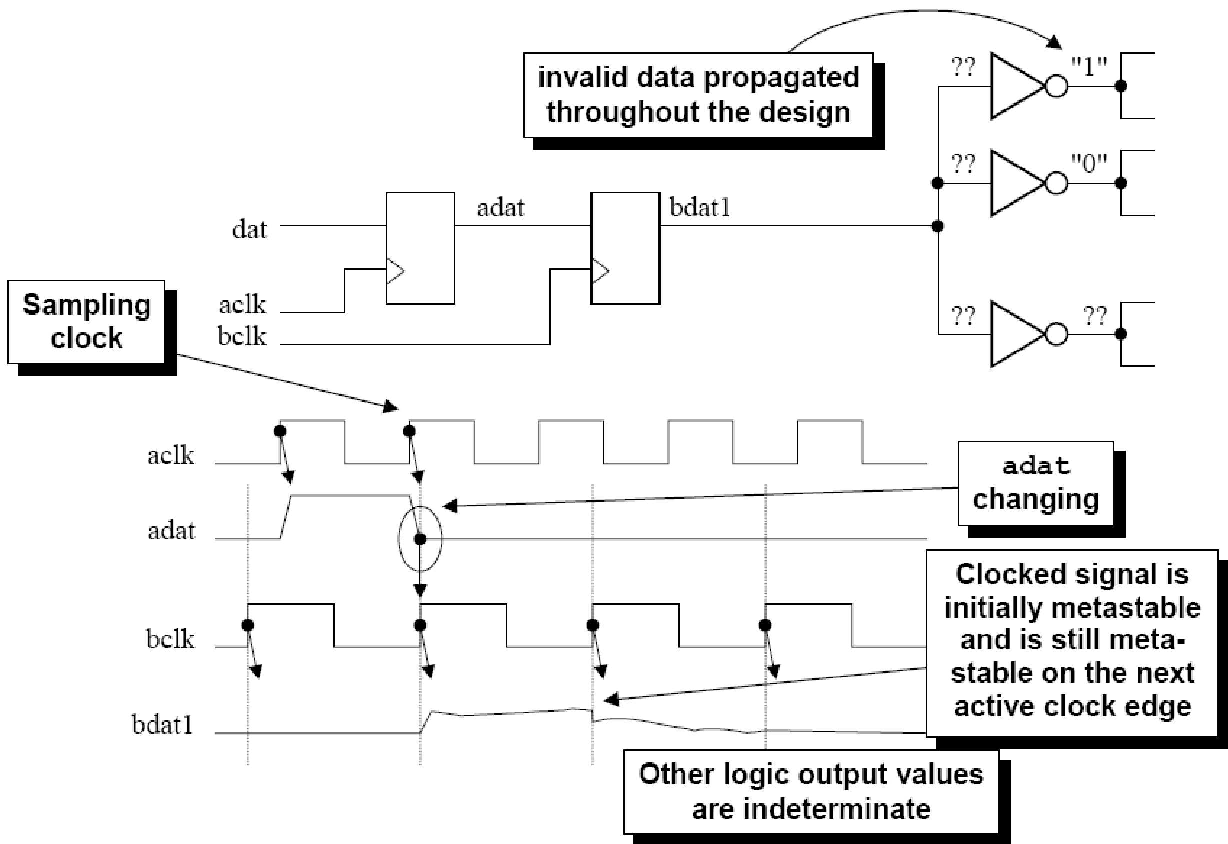
Στην Εικόνα 29 φαίνεται μια αποτυχία συγχρονισμού (synchronization failure) που συμβαίνει όταν ένα σήμα το οποίο δημιουργείται σε ένα πεδίο χρόνου δειγματοληπτείται πολύ σύντομα μετά την αλλαγή του από τη θετική ακμή ενός ρολογιού από άλλο πεδίο χρόνου.



Εικόνα 29: Ασύγχρονα ρολόγια και αποτυχία συγχρονισμού

Η αποτυχία συγχρονισμού προκαλείται όταν ένα σήμα εξόδου μπαίνει σε μετασταθερή κατάσταση και δεν συγκλίνει προς μια έγκυρη τιμή μέχρι τη στιγμή που θα δειγματοληπτηθεί ξανά. Στην Εικόνα 2 φαίνεται πώς ένα σήμα εξόδου μπορεί να προκαλέσει τη διάδοση μη έγκυρων σημάτων στον υπόλοιπο σχεδιασμό.

Κάθε flip-flop που χρησιμοποιείται σε έναν σχεδιασμό έχει συγκεκριμένο χρόνο setup και hold, ή αλλιώς χρόνο στον οποίο τα δεδομένα εισόδου δεν πρέπει να αλλάξουν πριν και μετά από μια θετική ακμή ρολογιού. Αυτό το χρονικό παράθυρο καθορίζεται ως παράμετρος του σχεδιασμού του flip-flop ακριβώς προκειμένου να αποτρέψει την αλλαγή ενός σήματος δεδομένων πολύ κοντά σε ένα σήμα συγχρονισμού, πράγμα που θα προκαλέσει την εισαγωγή του σήματος εξόδου σε μετασταθερή κατάσταση.



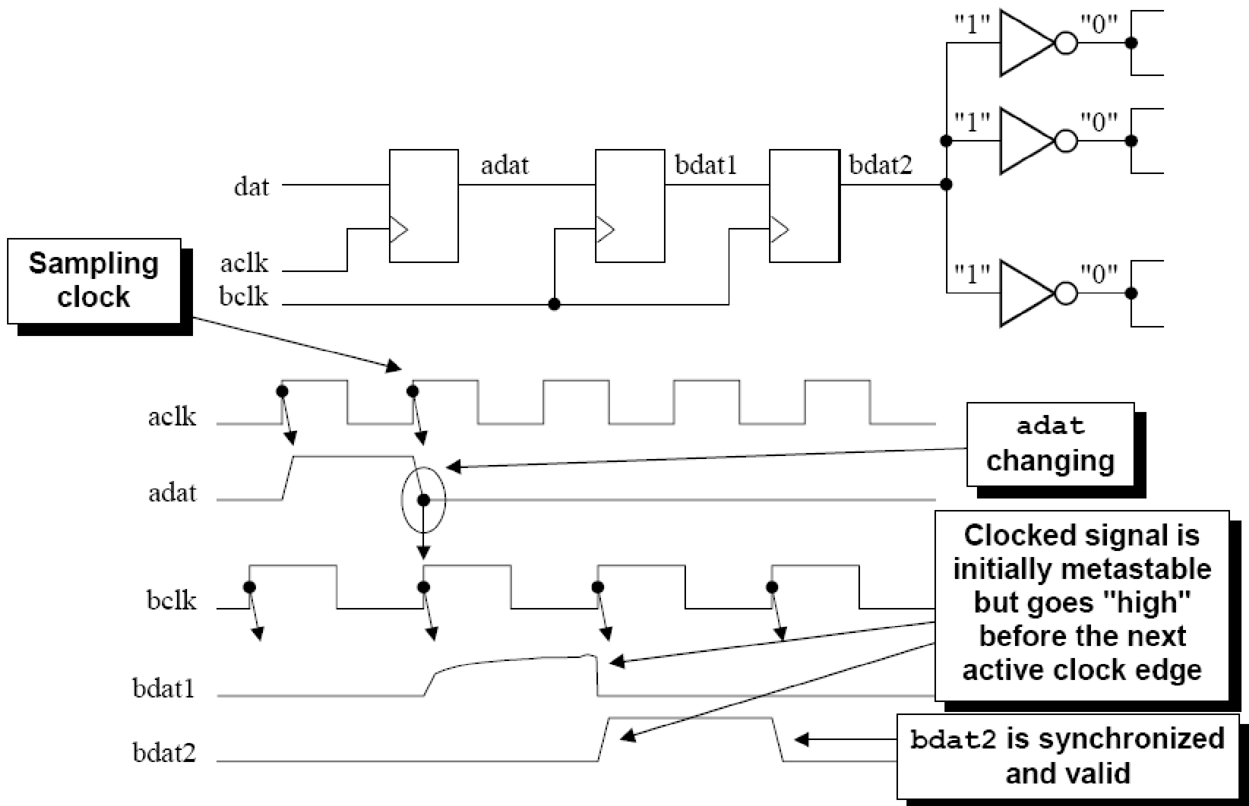
Εικόνα 30: Ένα μετασταθερό σήμα μπορεί να διαδώσει μη έγκυρες τιμές στον υπόλοιπο σχεδιασμό

Το παραπάνω πρόβλημα επιλύεται με τη χρήση διατάξεων που ονομάζονται *συγχρονιστές*. Ο πιο διαδεδομένος συγχρονιστής είναι ο συγχρονιστής δύο-flip-flop, ο οποίος φαίνεται στην Εικόνα 31.

Το πρώτο flip-flop δειγματοληπτεί το ασύγχρονο σήμα εισόδου προκειμένου αυτό να εισαχθεί στο νέο πεδίο χρόνου και περιμένει έναν ολόκληρο κύκλο ρολογιού ώστε να εξασθενήσει οποιαδήποτε μετασταθερότητα στο σήμα εξόδου πρώτου-σταδίου (στο flip-flop bdat1). Έπειτα το σήμα πρώτου σταδίου δειγματοληπτείται με το ίδιο ρολόι από ένα flip-flop δεύτερου-σταδίου, με το σκεπτικό ότι το σήμα δευτέρου-σταδίου είναι τώρα ένα σταθερό και έγκυρο σήμα, συγχρονισμένο με το νέο πεδίο χρόνου.

Θεωρητικά, είναι δυνατόν το σήμα πρώτου-σταδίου να είναι αρκετά μετασταθερό έως τη στιγμή που θα δειγματοληπτηθεί από το δεύτερο στάδιο, ώστε να προκαλέσει την είσοδο και του σήματος δεύτερου-σταδίου σε μετασταθερή κατάσταση. Ο υπολογισμός της πιθανότητας του χρόνου μεταξύ αποτυχιών συγχρονισμού (Mean Time Between Failures – MTBF) αποτελεί συνάρτηση πολλών παραγόντων, μεταξύ των οποίων και οι συχνότητες ρολογιών που χρησιμοποιούνται για να δημιουργήσουν το σήμα εισόδου και να χρονίσουν τα flip-flops συγχρονισμού.

Για τις περισσότερες εφαρμογές συγχρονισμού, ο κλασικός συγχρονιστής δύο flip-flop είναι αρκετός για να αφαιρέσει οποιαδήποτε μετασταθερότητα. Σε αντίθετη περίπτωση μπορεί απλώς να χρησιμοποιηθεί μια διάταξη περισσότερων flip-flops στη σειρά.

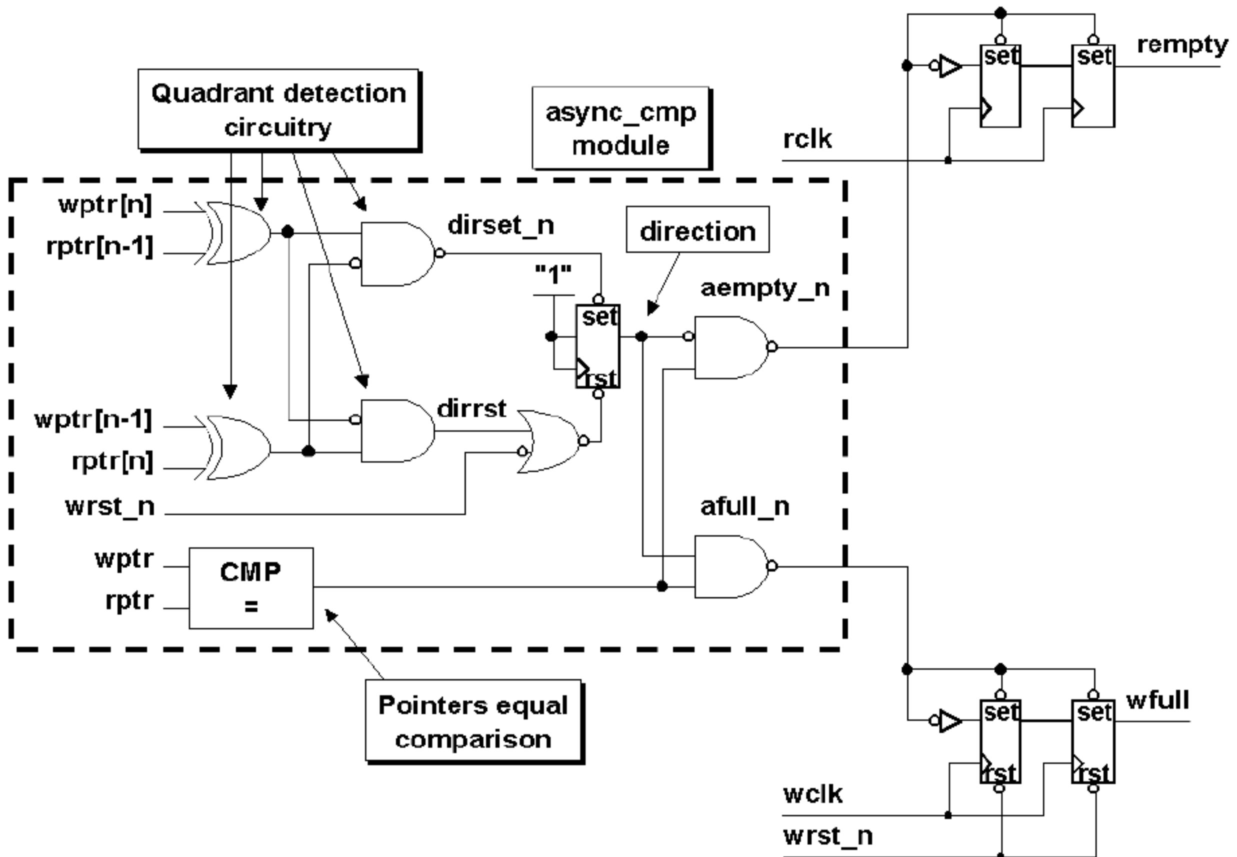


Εικόνα 31: Συγχρονιστής δύο-flip-flop

3.8.Παραγωγή συγχρονισμένων σημάτων γεμάτης και άδειας κατάστασης

Όπως έγινε φανερό από το προηγούμενο υποκεφάλαιο, είναι απαραίτητη η χρήση συγχρονιστών για την παραγωγή των τελικών σημάτων <γεμάτης> και <άδειας> κατάστασης από τα ασύγχρονα σήμα afull_n και aempty_n που παράγει η λογική σύγκρισης. Όταν το σήμα afull_n γίνεται 1, σύγχρονα με το ρολόι εγγραφής (όπως ήδη αναφέρθηκε), δεν υπάρχει ανάγκη συγχρονισμού και τα δύο flip-flops του συγχρονιστή τίθενται (set) ασύγχρονα στο 1. Όταν γίνει μηδέν, σύγχρονα με το ρολόι ανάγνωσης, υπάρχει ανάγκη συγχρονισμού με το ρολόι εγγραφής και συνεπώς το σήμα afull_n εισάγεται ως είσοδος στη διάταξη του συγχρονιστή, προκειμένου να αφαιρεθεί οποιαδήποτε μετασταθερότητα. Και στις δύο περιπτώσεις, το τελικό σήμα <γεμάτης> κατάστασης παράγεται σύγχρονα με το ρολόι εγγραφής. Ακριβώς αντίστοιχα λειτουργεί και η διάταξη συγχρονισμού του aempty_n, για παραγωγή του τελικού συγχρονισμένου σήματος <άδειας> κατάστασης.

Η ολοκληρωμένη διάταξη παραγωγής των σημάτων <γεμάτης> και <άδειας> κατάστασης απεικονίζεται στην Εικόνα 32.



Εικόνα 32: Ασύγχρονη σύγκριση δεικτών για την παραγωγή των σημάτων <γεμάτης> και <άδειας> κατάστασης

3.9.Γεγονότα ενδιαφέροντος

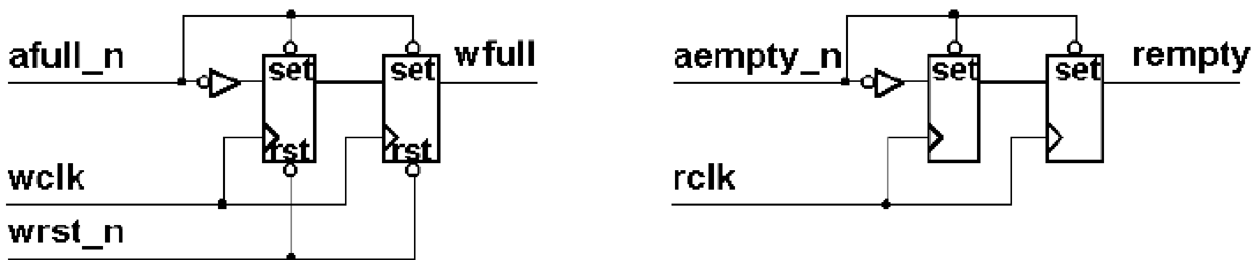
Ακολούθως περιγράφονται διάφορα συνηθισμένα γεγονότα ενδιαφέροντος στη λειτουργία της FIFO, προκειμένου αυτή να γίνει περισσότερο κατανοητή.

Το πρώτο γεγονός της FIFO που παρουσιάζει ενδιαφέρον συμβαίνει με την επανεκκίνηση (reset) της FIFO. Όταν η FIFO επανεκκινείται, συμβαίνουν τέσσερις σημαντικές ενέργειες:

1. Το σήμα επανεκκίνησης θέτει τη σημαία γεμάτης κατάστασης στο 0. Η σημαία άδειας κατάστασης δεν τίθεται στο 0.
2. Το σήμα επανεκκίνησης θέτει στο 0 και τους δύο δείκτες της FIFO, οπότε ο συγκριτής δεικτών αποφασίζει ότι οι δείκτες είναι ίσοι.
3. Το σήμα επανεκκίνησης θέτει στο 0 το bit κατεύθυνσης.
4. Εφόσον οι δείκτες είναι ίσοι και το bit κατεύθυνσης είναι στο 0, το σήμα aempty_n γίνεται 1, πράγμα που θέτει τη σημαία άδειας κατάστασης στο 1.

Το δεύτερο λειτουργικό γεγονός ενδιαφέροντος της FIFO συμβαίνει όταν πραγματοποιείται μια λειτουργία εγγραφής στη FIFO και αυξάνεται ο δείκτης εγγραφής. Στο σημείο αυτό, οι δείκτες της FIFO δεν είναι πλέον ίσοι, οπότε το σήμα aempty_n γίνεται 0, σταματώντας να θέτει στο 1 τα flip-flops του συγχρονιστή του σήματος άδειας κατάστασης. Μετά

από δύο θετικές ακμές του ρολογιού ανάγνωσης, η FIFO θα οδηγήσει το σήμα άδειας κατάστασης στο 0. Επειδή η αρνητική ακμή του aempty_n συμβαίνει με τη θετική ακμή του ρολογιού εγγραφής κι επειδή το σήμα άδειας κατάστασης χρονίζεται από το ρολόι ανάγνωσης, απαιτείται ο συγχρονιστής δύο-flip-flop που φαίνεται και στην Εικόνα 33 προκειμένου να αφαιρέσει ματασταθερότητα που μπορεί δημιουργηθεί στο πρώτο flip-flop άδειας κατάστασης.



Εικόνα 33: Συγχρονιστές σημάτων <γεμάτης> και <άδειας> κατάστασης

Το τρίτο λειτουργικό γεγονός ενδιαφέροντος της FIFO συμβαίνει όταν ο δείκτης εγγραφής αυξάνεται και εισέρχεται στο δεύτερο τεταρτημόριο του κώδικα Gray, ενώ ο δείκτης ανάγνωσης βρίσκεται ακόμα στο πρώτο. Το bit κατεύθυνσης τίθεται στο 0 (αλλά βρισκόταν ήδη σε αυτή την κατάσταση μετά την επανεκκίνηση).

Το τέταρτο λειτουργικό γεγονός ενδιαφέροντος της FIFO συμβαίνει όταν ο δείκτης εγγραφής βρίσκεται ένα τεταρτημόριο πριν τον δείκτη ανάγνωσης. Όταν συμβαίνει αυτό, το bit κατεύθυνσης τίθεται στο 1, για να δηλώσει ότι η FIFO «τείνει να γεμίσει». Αυτό σημαίνει ότι το bit κατεύθυνσης τίθεται πολύ πριν η FIFO γεμίσει και η μεταβολή αυτή δεν είναι χρονικά σημαντική σε σχέση με το σήμα afull_n.

Το πέμπτο λειτουργικό γεγονός ενδιαφέροντος της FIFO συμβαίνει όταν ο δείκτης εγγραφής φτάσει τον δείκτη ανάγνωσης (και το bit κατεύθυνσης είναι φυσικά ακόμα στο 1). Όταν συμβαίνει αυτό, το σήμα afull_n θέτει στο 1 τα δύο flip-flops του συγχρονιστή του σήματος γεμάτης κατάστασης. Το σήμα afull_n οδηγείται στο 1 από μια λειτουργία εγγραφής και είναι σύγχρονο με τη θετική ακμή του ρολογιού εγγραφής. Συνεπώς, το τελικό σήμα γεμάτης κατάστασης γίνεται 1 σύγχρονα με το ρολόι εγγραφής.

Το έκτο λειτουργικό γεγονός ενδιαφέροντος της FIFO συμβαίνει όταν πραγματοποιείται μια λειτουργία ανάγνωσης στη FIFO και ο δείκτης ανάγνωσης αυξάνεται. Στο σημείο αυτό, οι δείκτες της FIFO δεν είναι πλέον ίσοι, οπότε το σήμα afull_n οδηγείται στο 0, σταματώντας να θέτει στο 1 τα flip-flops του συγχρονιστή του σήματος γεμάτης κατάστασης. Έπειτα από δύο θετικές ακμές του ρολογιού εγγραφής, η FIFO θα οδηγήσει το σήμα γεμάτης κατάστασης στο 0. Επειδή η αρνητική ακμή του afull_n συμβαίνει με τη θετική ακμή του ρολογιού ανάγνωσης κι επειδή το σήμα γεμάτης κατάστασης χρονίζεται από το ρολόι εγγραφής, απαιτείται ο συγχρονιστής δύο-flip-flop που φαίνεται και στην Εικόνα 33 προκειμένου να αφαιρέσει ματασταθερότητα που μπορεί δημιουργηθεί στο πρώτο flip-flop γεμάτης κατάστασης, το οποίο αποθηκεύει το ασύγχρονα παραγόμενο σήμα afull_n.

Κατά τη λειτουργία, το σήμα γεμάτης κατάστασης παράγεται σύγχρονα με το ρολόι εγγραφής, όπως και το σήμα άδειας κατάστασης παράγεται σύγχρονα με το ρολόι ανάγνωσης. Το σήμα afull_n οδηγείται στο 1 έπειτα από μια θετική ακμή του ρολογιού εγγραφής, και η θετική του ακμή είναι φυσικά σύγχρονη με το ρολόι εγγραφής. Η αρνητική ακμή ωστόσο προκαλείται από το ρολόι ανάγνωσης, οπότε πρέπει να συγχρονιστεί όπως αναφέρθηκε με το ρολόι εγγραφής. Τα ίδια χρονικά θέματα που σχετίζονται με τη ρύθμιση της σημαίας γεμάτης κατάστασης ισχύουν και για τη ρύθμιση της σημαίας άδειας κατάστασης.

Το έβδομο λειτουργικό γεγονός ενδιαφέροντος της FIFO συμβαίνει όταν ο δείκτης ανάγνωσης αυξάνεται και εισέρχεται στο επόμενο τεταρτημόριο του κώδικα Gray, ενώ ο δείκτης εγγραφής βρίσκεται ακόμα στο προηγούμενο. Το bit κατεύθυνσης τίθεται στο 1 (αν και βρισκόταν ήδη στο 1).

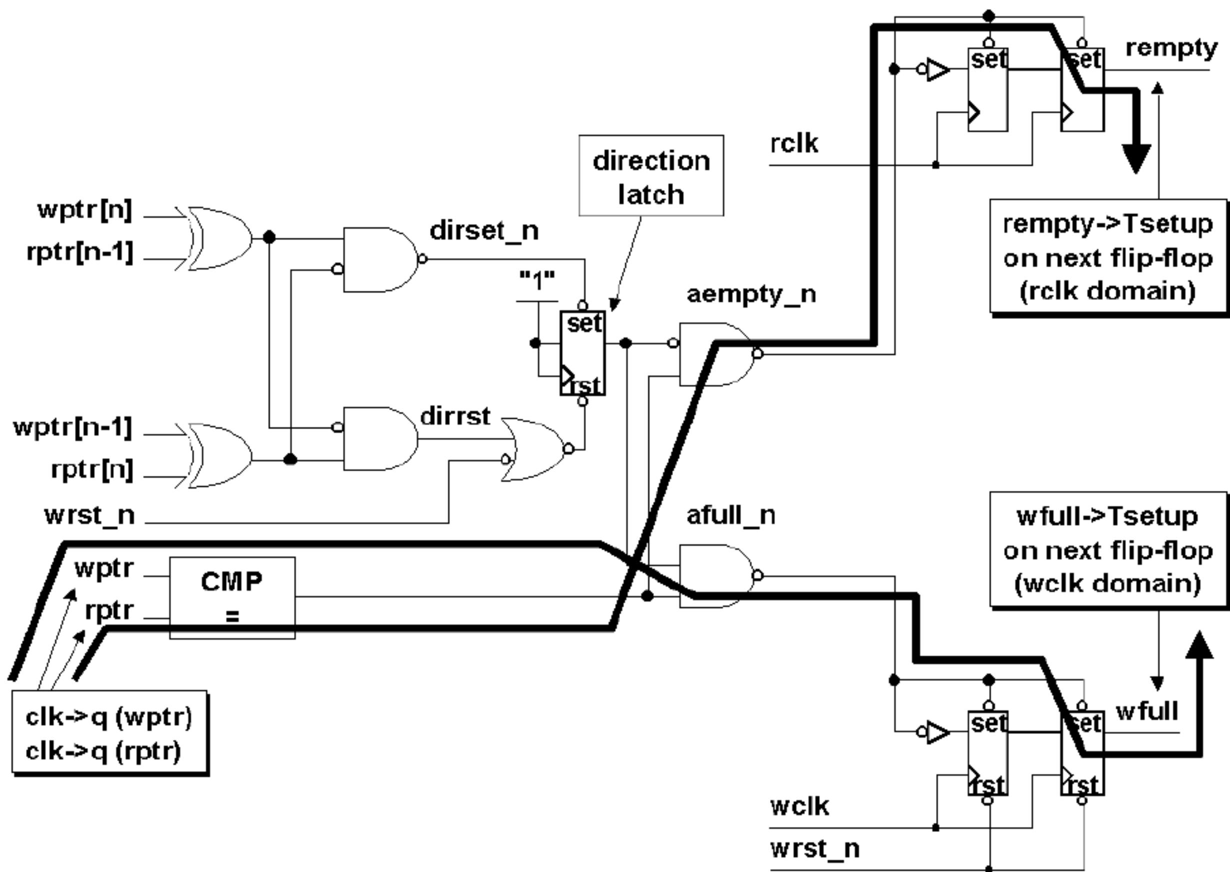
Το όγδοο λειτουργικό γεγονός ενδιαφέροντος της FIFO συμβαίνει όταν ο δείκτης ανάγνωσης βρίσκεται ένα τεταρτημόριο πριν τον δείκτη εγγραφής. Όταν συμβαίνει αυτό, το bit κατεύθυνσης τίθεται στο 0, για να δηλώσει ότι η FIFO «τείνει να αδειάσει». Αυτό σημαίνει ότι το bit κατεύθυνσης τίθεται στο 0 πολύ πριν η FIFO αδειάσει και η μεταβολή αυτή δεν είναι χρονικά σημαντική σε σχέση με το σήμα aempty_n.

Το ένατο λειτουργικό γεγονός ενδιαφέροντος της FIFO συμβαίνει όταν ο δείκτης ανάγνωσης φτάσει τον δείκτη εγγραφής (και το bit κατεύθυνσης είναι φυσικά ακόμα στο 0). Όταν συμβαίνει αυτό, το σήμα aempty_n θέτει στο 1 τα δύο flip-flops του συγχρονιστή του σήματος άδειας κατάστασης. Το σήμα aempty_n οδηγείται στο 1 από μια λειτουργία ανάγνωσης και είναι σύγχρονο με τη θετική ακμή του ρολογιού ανάγνωσης. Συνεπώς, το τελικό σήμα άδειας κατάστασης γίνεται 1 σύγχρονα με το ρολόι ανάγνωσης.

Έτσι, η FIFO έχει φτάσει και πάλι σε άδεια κατάσταση. Η μόνη λειτουργία που μπορεί τώρα να πραγματοποιηθεί είναι μια λειτουργία εγγραφής, στην οποία περίπτωση επαναλαμβάνεται το πρώτο λειτουργικό γεγονός ενδιαφέροντος. Οποιαδήποτε άλλη περίπτωση από εκεί και πέρα καλύπτεται από τα γεγονότα ενδιαφέροντος που ήδη αναλύθηκαν.

3.10.Χρονικά κρίσιμα μονοπάτια των σημάτων γεμάτης και άδειας κατάστασης

Το χρονικά κρίσιμο μονοπάτι (critical path) του σήματος άδειας κατάστασης, που φαίνεται στην Εικόνα 34, αποτελείται από (1) τη σύγχρονη με τη θετική ακμή του ρολογιού ανάγνωσης αύξηση του δείκτη ανάγνωσης, (2) τη λογική σύγκρισης των δύο δεικτών, (3) το συνδυασμό του αποτελέσματος του συγκριτή με την έξοδο του latch κατάστασης για την παραγωγή του αασύγχρονου σήματος aempty_n, (4) την οδήγηση του σήματος άδειας κατάστασης, (5) οποιαδήποτε λογική οδηγείται από το σήμα άδειας κατάστασης, και (6) τα επακόλουθα σήματα, που θα πρέπει να ικανοποιούν τον χρόνο setup οποιωνδήποτε παραπέρα flip-flops, τα οποία χρονίζονται μέσα στο πεδίο του ρολογιού ανάγνωσης. Αυτό το χρονικά κρίσιμο μονοπάτι έχει ένα συμμετρικά αντίστοιχο χρονικά κρίσιμο μονοπάτι για την παραγωγή του σήματος γεμάτης κατάστασης, όπως επίσης φαίνεται στην Εικόνα 34.



Εικόνα 34: Χρονικά κρίσιμα μονοπάτια για την παραγωγή των σημάτων γεμάτης και άδειας κατάστασης

3.11.Θέματα ασύγχρονης λειτουργίας

Παρακάτω παρουσιάζονται διάφορα θέματα που σχετίζονται με την ασύγχρονη λειτουργία της δημιουργίας των σημάτων γεμάτης ή άδειας κατάστασης και συζητούνται περιπτώσεις στις οποίες η ορθή λειτουργία της παραπάνω λογικής μπορεί να μην είναι άμεσα εμφανής.

Η δημιουργία του σήματος ελέγχου aempty_n είναι απλή. Όποτε ο δείκτης ανάγνωσης ισούται με τον δείκτη εγγραφής, και το bit κατεύθυνσης είναι στο 0, η FIFO είναι κενή.

Η σημαία άδειας κατάστασης χρησιμοποιείται μόνο στο πεδίο του ρολογιού ανάγνωσης και εφόσον ο δείκτης ανάγνωσης, ο οποίος αυξάνεται με το ρολόι ανάγνωσης, προκαλεί την οδήγηση της σημαίας άδειας κατάστασης στο 1, αυτό γίνεται πάντα σύγχρονα με το πεδίο του ρολογιού ανάγνωσης. Εφόσον η σημαία άδειας κατάστασης ικανοποιεί τις απαιτήσεις του χρονικά κρίσιμου μονοπατιού της σημαίας άδειας κατάστασης που περιγράφηκε στο προηγούμενο υποκεφάλαιο, δεν υπάρχει κανένα πρόβλημα συγχρονισμού με την οδήγηση της σημαίας άδειας κατάστασης στο 1.

Η οδήγηση της σημαίας άδειας κατάστασης στο 0 προκαλείται από την αύξηση του δείκτη εγγραφής σύμφωνα με το ρολόι εγγραφής, και συνεπώς δε σχετίζεται με το ρολόι ανάγνωσης. Η οδήγηση του aempty_n στο 0 θα πρέπει συνεπώς να συγχρονιστεί μέσω ενός συγχρονιστή δύο-flip-flop, ο οποίος χρονίζεται από το ρολόι ανάγνωσης. Το πρώτο flip-flop μπορεί να υπόκειται σε μετασταθερότητα, αλλά το δεύτερο flip-flop περιλαμβάνεται για να περιμένει να υποχωρήσει αυτή η μετασταθερότητα, όπως συμβαίνει και σε οποιονδήποτε άλλο συγχρονιστή.

Εφόσον το aempty-n τίθεται στο 1 από το ένα ρολόι και στο 0 από το άλλο, έχει ακαθόριστη διάρκεια, και μπορεί να είναι ακόμα και στιγμιαίος παλμός (runt pulse). Ένας στιγμιαίος σταθμός είναι μια μετάβαση σήματος Low-High-Low στον οποίο η μετάβαση σε High μπορεί να περάσει ή όχι από το κατώφλι του λογικού "1" της λογικής οικογένειας που χρησιμοποιείται.

Αν το σήμα ελέγχου aempty_n είναι ένας στιγμιαίος παλμός, υπάρχουν τέσσερα πιθανά σενάρια που θα πρέπει να συζητηθούν:

a. Ο στιγμιαίος παλμός δεν αναγνωρίζεται από τα flip-flops του συγχρονιστή άδειας κατάστασης και η σημαία άδειας κατάστασης δεν τίθεται στο 1. Αυτό δεν αποτελεί πρόβλημα.
b. Ο στιγμιαίος παλμός μπορεί να θέσει στο 1 το πρώτο flip-flop του συγχρονιστή, αλλά όχι το δεύτερο. Αυτό είναι πολύ απίθανο, αλλά θα είχε ως αποτέλεσμα έναν περιττό, αλλά ορθά συγχρονισμένο, θετικό παλμό του σήματος άδειας κατάστασης, ο οποίος θα εμφανιζόταν στην έξοδο του δεύτερου flip-flop έναν κύκλο ρολογιού μετά. Αυτό δεν αποτελεί πρόβλημα.
c. Ο στιγμιαίος παλμός μπορεί να θέσει στο 1 το δεύτερο flip-flop του συγχρονιστή, αλλά όχι το πρώτο. Αυτό είναι πολύ απίθανο, αλλά θα είχε ως αποτέλεσμα έναν περιττό, αλλά ορθά συγχρονισμένο, θετικό παλμό του σήματος άδειας κατάστασης (εφόσον καλύπτονται οι απαιτήσεις του κρίσιμου μονοπατιού), ο οποίος θα εμφανιστεί στην έξοδο του δεύτερου flip-flop μέχρι τον επόμενο θετικό παλμό του ρολογιού ανάγνωσης, οπότε και θα καθαριστεί από το 0 του πρώτου flip-flop. Αυτό δεν αποτελεί πρόβλημα.
d. Η πιο πιθανή περίπτωση είναι ο στιγμιαίος παλμός να θέσει στο 1 και τα δύο flip-flops, δημιουργώντας έτσι έναν σωστά συγχρονισμένο θετικό παλμό του σήματος άδειας κατάστασης, ο οποίος θα διαρκέσει δύο περιόδους του ρολογιού ανάγνωσης. Αυτή η εκτεταμένη διάρκεια προκαλείται από τον συγχρονιστή δύο-flip-flop. Αυτό δεν αποτελεί πρόβλημα.

Ο στιγμιαίος παλμός δεν μπορεί να επηρεάσει την είσοδο δεδομένων του συγχρονιστή, εφόσον ένας στιγμιαίος παλμός στο aempty_n μπορεί να εμφανιστεί μόνο αμέσως μετά από μια θετική ακμή του ρολογιού ανάγνωσης, οπότε πολύ πριν από την επόμενη θετική ακμή του ρολογιού ανάγνωσης (εφόσον καλύπτονται οι απαιτήσεις του κρίσιμου μονοπατιού.

Το σήμα aempty_n μπορεί επίσης να παραμείνει στο High για μεγαλύτερο χρονικό διάστημα και να γίνει Low οποτεδήποτε, ακόμα και ταυτόχρονα με την επόμενη θετική ακμή του ρολογιού ανάγνωσης. Αν γίνει Low πριν το χρόνο setup του πρώτου flip-flop του συγχρονιστή, το αποτέλεσμα είναι παρόμοιο με την περίπτωση (4) που συζητήθηκε παραπάνω. Αν γίνει Low μετά το διάστημα setup, ο συγχρονιστής θα επεκτείνει απλώς τον παλμό του σήματος άδειας κατάστασης για μία ακόμα περίοδο του ρολογιού ανάγνωσης.

Αν το aempty_n γίνει Low κατά τον υποκείμενο σε μετασταθερότητα χρόνο setup, η έξοδος του πρώτου flip-flop του συγχρονιστή θα είναι ακαθόριστη για μερικά nanoseconds, αλλά έπειτα θα γίνει είτε High είτε Low. Σε οποιαδήποτε περίπτωση, η έξοδος του δεύτερου flip-flop του συγχρονιστή θα δημιουργήσει την απαραίτητη συγχρονισμένη έξοδο του σήματος άδειας κατάστασης.

Το επόμενο ζήτημα είναι, τι συμβαίνει αν το ρολόι εγγραφής οδηγήσει στο 0 το σήμα aempty_n ταυτόχρονα με μια θετική ακμή του ρολογιού ανάγνωσης; Το πρώτο flip-flop του συγχρονιστή θα μπορούσε να εισέλθει σε κατάσταση μετασταθερότητας, αλλά για αυτόν τον λόγο υπάρχει ένα δεύτερο flip-flop σον συγχρονιστή.

Μόνο που η αφαίρεση του σήματος που έθετε το δεύτερο flip-flop σε High θα παραβίαζε τον χρόνο ανάκτησης (recovery time) του δεύτερου flip-flop. Θα μπορούσε αυτό να προκαλέσει την εισαγωγή του δεύτερου flip-flop σε μετασταθερή κατάσταση; Όχι, γιατί το σήμα που

αφαιρέθηκε έθετε ασύγχρονα το flip-flop σε High κατάσταση, ενώ η είσοδος του ίδιου flip-flop είναι ήδη High, οπότε δεν υπάρχει λόγος να μεταβληθεί η κατάσταση του flip-flop και να εισέλθει σε μετασταθερή κατάσταση.

Ένα τελευταίο ζήτημα είναι αν ένας στιγμιαίος παλμός, του οποίου η αρνητική ακμή προκαλείται από το ρολόι εγγραφής, θέσει το δεύτερο flip-flop του συγχρονιστή στο1 πολύ κοντά σε μια θετική ακμή του ρολογιού ανάγνωσης, θα μπορούσε αυτό να παραβεί τον χρόνο ανάκτησης του σήματος preset του flip-flop και να προκαλέσει μετασταθερότητα στο δεύτερο flip-flop; Η απάντηση είναι όχι, αρκεί να ικανοποιούνται οι απαιτήσεις του χρονικά κρίσιμου μονοπατιού. Αν συμβαίνει αυτό, η οδήγηση σε Low του σήματος aempty_n θα γίνει πολύ σύντομα έπειτα από έναν θετικό παλμό του ρολογιού ανάγνωσης και πολύ πριν από τη θετική ακμή του δεύτερου flip-flop, οπότε οι στιγμιαίοι παλμοί μπορούν να συμβούν μόνο πολύ πριν από τη θετική ακμή του ρολογιού ανάγνωσης.

Για όλες τις παραπάνω περιπτώσεις, συμμετρικά αντίστοιχα σενάρια ισχύουν για τη δημιουργία της σημαίας γεμάτης κατάστασης.

Κεφάλαιο 4:
Λειτουργία μιας τυπικής κρυφής μνήμης

Στο κεφάλαιο αυτό γίνεται μια σύντομη περιγραφή των λειτουργιών μιας κρυφής μνήμης, λειτουργίες που οφείλουν να πραγματοποιούνται και από την παρουσιαζόμενη κρυφή μνήμη δεύτερου επιπέδου, ανεξάρτητα από τις υπόλοιπα ιδιαίτερα καθήκοντα της.

4.1.Εισαγωγή

Κρυφή μνήμη ονομάζεται οποιοδήποτε τμήμα της ιεραρχίας μνήμης ενός υπολογιστικού συστήματος βρίσκεται ανάμεσα στους καταχωρητές του επεξεργαστή και την κύρια μνήμη. Τα διάφορα επίπεδα κρυφής μνήμης χρησιμοποιούνται για να καλύψουν το κενό ανάμεσα στις ταχύτητες λειτουργίας του επεξεργαστή και της κύριας μνήμης. Αποτελούν μνήμες μικρότερου μεγέθους από την κύρια μνήμη, αλλά μεγαλύτερης ταχύτητας, έτσι ώστε να ικανοποιούν γρήγορα τις περισσότερες από τις αιτήσεις του επεξεργαστή (βάσει των αρχών της χωρικής και χρονικής τοπικότητας των αναφορών), χωρίς την ανάγκη προσφυγής στην κύρια μνήμη, με τη μεγάλη καθυστέρηση που αυτή συνεπάγεται. Προκειμένου να περιγραφεί συνοπτικά η λειτουργία μιας κρυφής μνήμης, αναλύονται σύντομα τέσσερα βασικά θέματα που αφορούν στον μηχανισμό της.

4.2.Τοποθέτηση ενός μπλοκ στην κρυφή μνήμη

Μπλοκ ονομάζεται ένα σύνολο δεδομένων. Όταν ένας επεξεργαστής επιχειρεί προσπέλαση σε μια γραμμή της κύριας μνήμης, και η γραμμή αυτή δεν υπάρχει σε κάποιο ενδιάμεσο επίπεδο κρυφής μνήμης, τότε αντιγράφεται στην κρυφή μνήμη όχι μόνο η συγκεκριμένη γραμμή, αλλά και ένας ορισμένος αριθμός γραμμών που βρίσκονται γύρω από αυτήν στην κύρια μνήμη. Όπως αναφέρθηκε, αυτό γίνεται βάσει της αρχής της χωρικής τοπικότητας των αναφορών, βάσει δηλαδή της θεώρησης ότι εφόσον ο επεξεργαστής χρειάστηκε της συγκεκριμένη γραμμή δεδομένων, είναι πιθανόν να χρειαστεί στο κοντινό μέλλον πληροφορίες που περιέχονται στις γειτονικές της γραμμές, οπότε αυτές μεταφέρονται σε κατώτερο επίπεδο μνήμης, προκειμένου να είναι πιο άμεσα διαθέσιμες. Αυτό το σύνολο γραμμών που αντιγράφεται συνιστά το μπλοκ.

Η Εικόνα 35 δείχνει ότι οι περιορισμοί του πού μπορεί να τοποθετηθεί ένα μπλοκ δημιουργούν τρεις βασικές κατηγορίες οργάνωσης της κρυφής μνήμης:

- Αν κάθε μπλοκ μπορεί να τοποθετηθεί σε μία μόνο θέση μέσα στην κρυφή μνήμη, τότε η κρυφή μνήμη λέγεται ότι είναι *άμεσα αντιστοιχιζόμενη (direct mapped)*. Ο υπολογισμός της θέσης δίνεται συνήθως από τον τύπο

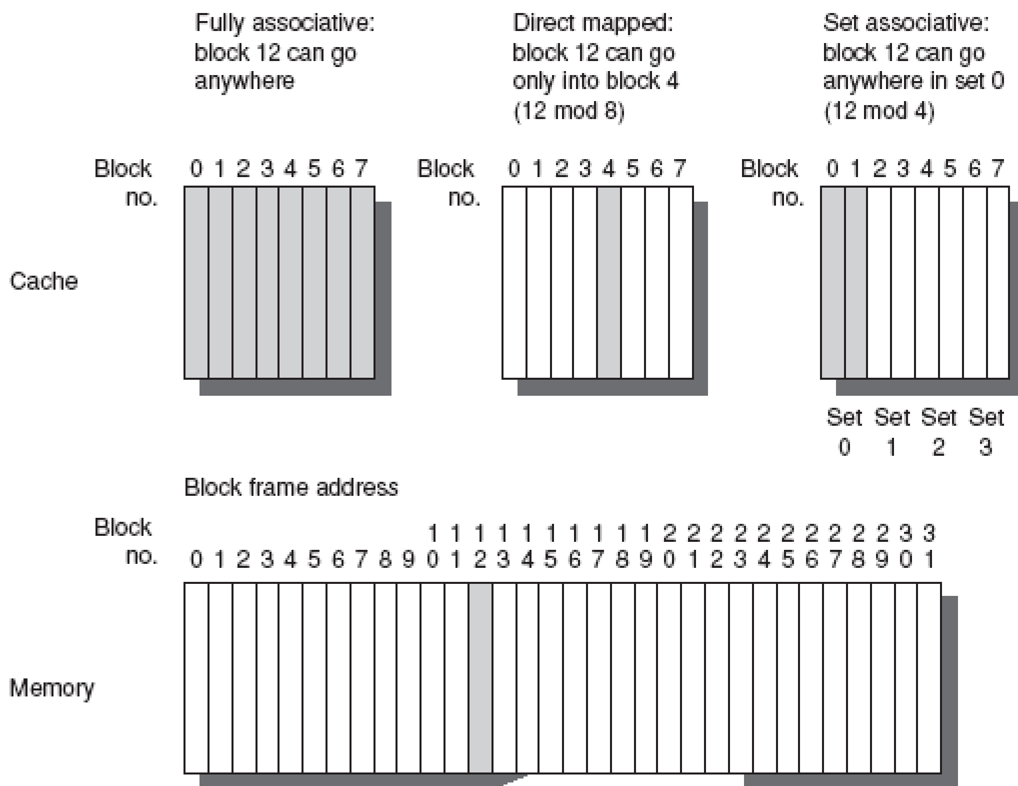
  (Διεύθυνση μπλοκ) MOD (Αριθμός μπλοκ στην κρυφή μνήμη)

- Αν ένα μπλοκ μπορεί να τοποθετηθεί οπουδήποτε στην κρυφή μνήμη, τότε η κρυφή μνήμη λέγεται ότι είναι *πλήρως συσχετιζόμενη (fully associative)*.

- Αν ένα μπλοκ μπορεί να τοποθετηθεί σε ένα περιορισμένο σύνολο θέσεων της κρυφής μνήμης, τότε η κρυφή μνήμη είναι *συνολοσυσχετιζόμενη (set associative)*. Ένα *σύνολο* είναι μια ομάδα μπλοκ στην κρυφή μνήμη. Ένα μπλοκ πρώτα αντιστοιχίζεται σε ένα σύνολο, και έπειτα το μπλοκ μπορεί να τοποθετηθεί σε οποιαδήποτε θέση μέσα στο σύνολο. Το σύνολο συνήθως επιλέγεται βάσει κάποιων *bits επιλογής*, δηλαδή

  (Διεύθυνση μπλοκ) MOD (Αριθμός συνόλων στην κρυφή μνήμη)

  Αν υπάρχουν n μπλοκ σε ένα σύνολο, η τοποθέτηση στην κρυφή μνήμη ονομάζεται *συνολοσυσχετιζόμενη n-τρόπων (n-way set associative).*



Εικόνα 35: Τα τρία είδη οργάνωσης μιας κρυφής μνήμης

Το εύρος των κρυφών μνημών από άμεσα αντιστοιχιζόμενες έως πλήρως συσχετιζόμενες αποτελούν ουσιαστικά μια συνεχή σειρά επιπέδων συνολοσυσχετισμού. Μια άμεσα αντιστοιχιζόμενη κρυφή μνήμη είναι απλώς συνολοσυσχετιζόμενη 1-τρόπου, ενώ μια πλήρως συσχετιζόμενη κρυφή μνήμη με m μπλοκ μπορεί να περιγραφεί ως συνολοσυσχετιζόμενη m-τρόπων. Ισοδύναμα, η άμεσα αντιστοιχιζόμενη κρυφή μνήμη μπορεί να θεωρηθεί ότι έχει m σύνολα, και η πλήρως συσχετιζόμενη κρυφή μνήμη μπορεί να θεωρηθεί ότι έχει ένα σύνολο.

Για να γίνουν πιο κατανοητά τα παραπάνω, ας δούμε το παράδειγμα της Εικόνας 35. Η κρυφή μνήμη του παραδείγματος αυτού έχει 8 θέσεις μπλοκ και η κύρια μνήμη έχει 32 θέσεις μπλοκ. Οι τρεις επιλογές οργάνωσης της κρυφής μνήμης φαίνονται από αριστερά προς τα δεξιά. Στην πλήρως συσχετιζόμενη κρυφή μνήμη, το μπλοκ 12 του κατώτερου επιπέδου μπορεί να τοποθετηθεί σε οποιαδήποτε από τις 8 θέσεις μπλοκ της κρυφής μνήμης. Στην άμεσα αντιστοιχιζόμενη κρυφή μνήμη, το μπλοκ 12 μπορεί να τοποθετηθεί στη θέση μπλοκ 4 (12 modulo 8). Σε μια συνολοσυσχετιζόμενη κρυφή μνήμη, που περιλαμβάνει χαρακτηριστικά και από τις δυο άλλες οργανώσεις, το μπλοκ αυτό μπορεί να τοποθετηθεί οπουδήποτε στο σύνολο 0 (12 modulo

4). Αν έχουμε δύο θέσεις μπλοκ ανά σύνολο, αυτό σημαίνει ότι το μπλοκ 12 μπορεί να τοποθετηθεί είτε στο μπλοκ 0 είτε στο μπλοκ 1 της κρυφής μνήμης. Αυτή η συνολοσυσχετιζόμενη οργάνωση έχει 4 σύνολα με 2 μπλοκ ανά σύνολο και συνεπώς ονομάζεται *συνολοσυσχετιζόμενη 2-τρόπων*. Ας υποτεθεί ότι η κρυφή μνήμη είναι άδεια και ότι η διεύθυνση μπλοκ ορίζει το μπλοκ 12 στη μνήμη κατώτερου επιπέδου.

Οι πραγματικές κρυφές μνήμες περιέχουν χιλιάδες θέσεις μπλοκ, ενώ οι πραγματικές κύριες μνήμες μπορεί να περιέχουν εκατομμύρια μπλοκ. Η συντριπτική πλειοψηφία των κρυφών μνημών σήμερα είναι άμεσα αντιστοιχιζόμενες, συνολοσυσχετιζόμενες 2-τρόπων ή συνολοσυσχετιζόμενες 4-τρόπων.

Η παρουσιαζόμενη κρυφή μνήμη δεύτερου επιπέδου είναι πλήρως παραμετροποιήσιμη. Αυτό σημαίνει ότι το μέγεθός της και το μέγεθος γραμμής της μπορούν να οριστούν από τον χρήστη, ενώ η οργάνωσή της μπορεί να επιλεχθεί να είναι από άμεσα αντιστοιχιζόμενη έως πλήρως συσχετιζόμενη με οποιοδήποτε ενδιάμεσο στάδιο.

4.3.Εύρεση ενός μπλοκ στην κρυφή μνήμη

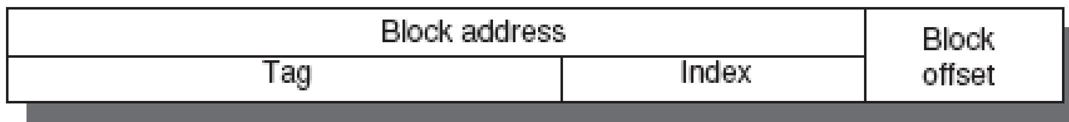
Οι κρυφές μνήμες έχουν μια ετικέτα διεύθυνσης (address tag) για κάθε θέση μπλοκ η οποία δηλώνει τη διεύθυνση του αποθηκευμένου μπλοκ. Η ετικέτα κάθε θέσης μπλοκ της κρυφής μνήμης που μπορεί να περιέχει την επιθυμητή πληροφορία ελέγχεται για να αποφασιστεί αν ταιριάζει με τη διεύθυνση μπλοκ που ορίστηκε από τον επεξεργαστή. Γενικά, όλες οι πιθανές ετικέτες ελέγχονται παράλληλα για λόγους ταχύτητας.

Πρέπει να υπάρχει ένας τρόπος να δηλώνεται ότι μια θέση μπλοκ δεν περιέχει έγκυρες πληροφορίες. Η πιο συνηθισμένη διαδικασία είναι να προστίθεται ένα *bit εγκυρότητας (valid bit)* στην ετικέτα, το οποίο δηλώνει αν η θέση περιέχει μια έγκυρη διεύθυνση. Αν το bit αυτό δεν είναι 1, τότε δεν μπορεί να θεωρηθεί ότι η ετικέτα ταιριάζει στη διεύθυνση μπλοκ που ζητήθηκε.

Πριν προχωρήσουμε παρακάτω, ας εξετάσουμε τη σχέση που έχει μια διεύθυνση του επεξεργαστή με την κρυφή μνήμη. Η Εικόνα 36 δείχνει πώς χωρίζεται μια διεύθυνση. Η πρώτη διαίρεση είναι ανάμεσα στη διεύθυνση του μπλοκ και τη μετατόπιση (offset) στο μπλοκ. Η διεύθυνση του μπλοκ μπορεί να χωριστεί παραπέρα στο *πεδίο ετικέτας (tag field)* και στο *πεδίο δείκτη (index field)*. Η μετατόπιση στο μπλοκ επιλέγει τα επιθυμητά δεδομένα από το μπλοκ (ουσιαστικά καθορίζει ποια γραμμή μέσα στο μπλοκ είναι αυτή που ζητήθηκε), το πεδίο δείκτη επιλέγει το σύνολο, και το πεδίο ετικέτας συγκρίνεται με τις ετικέτες του συνόλου αυτού για να βρεθεί αν ταιριάζει με κάποια από αυτές. Είναι φανερό ότι οι πλήρως συσχετιζόμενες κρυφές μνήμες δεν έχουν πεδίο δείκτη, εφόσον περιέχουν ουσιαστικά μόνο ένα σύνολο (αντίστοιχα, αν κάθε μπλοκ περιέχει μόνο μία γραμμή, τότε είναι περιττός ο ορισμός κάποιας μετατόπισης). Παρόλο που η σύγκριση θα μπορούσε να γίνει σε μεγαλύτερο μέρος της διεύθυνσης από το πεδίο ετικέτας, δεν υπάρχει λόγος λόγω των παρακάτω:

- Η μετατόπιση δεν πρέπει να χρησιμοποιείται στη σύγκριση, εφόσον είτε ολόκληρο το μπλοκ είναι παρών είτε όχι, και συνεπώς οποιοσδήποτε ορισμός μετατόπισης θα έχει εξ’ορισμού το ίδιο αποτέλεσμα (είτε όλες οι γραμμές θα είναι παρούσες είτε καμία).
- Ο έλεγχος του δείκτη είναι πλεονάζων, αφού χρησιμοποιήθηκε για την επιλογή του συνόλου της κρυφής μνήμης που ελέγχεται. Μια διεύθυνση που είναι αποθηκευμένη στο σύνολο 0 για παράδειγμα, πρέπει να έχει 0 στο πεδίο δείκτη, αλλιώς δε θα μπορούσε να βρίσκεται αποθηκευμένη στο σύνολο 0. Το σύνολο 1 πρέπει να έχει δείκτη 1, το σύνολο 2 πρέπει να έχει δείκτη 2 και ούτω καθεξής.

Αυτή η βελτιστοποίηση εξοικονομεί υλικό και ισχύ μειώνοντας το πλάτος σε μέγεθος μνήμης των ετικετών της κρυφής μνήμης.



Εικόνα 36: Τα τρία τμήματα της διεύθυνσης σε μια συνολοσυσχετιζόμενη ή άμεσα αντιστοιχιζόμενη κρυφή μνήμη

Αν το μέγεθος της κρυφής μνήμης παραμένει το ίδιο, η αύξηση του συσχετισμού (associativity) αυξάνει τον αριθμό των μπλοκ ανά σύνολο, και συνεπώς μειώνει το μέγεθος του δείκτη και αυξάνει το μέγεθος της ετικέτας. Το όριο δηλαδή μεταξύ ετικέτας και δείκτη στην Εικόνα 36 μετακινείται προς τα δεξιά καθώς αυξάνεται ο συσχετισμός, έως την πλήρη απουσία πεδίου δείκτη στην ακραία περίπτωση των πλήρως συσχετιζόμενων κρυφών μνημών.

4.4.Αντικατάσταση ενός μπλοκ σε περίπτωση αστοχίας στην κρυφή μνήμη

Όταν συμβαίνει αστοχία, ο ελεγκτής της κρυφής μνήμης πρέπει να επιλέξει ένα μπλοκ, το οποίο θα αντικατασταθεί από τα επιθυμητά δεδομένα. Ένα από τα κέρδη της άμεσα αντιστοιχιζόμενης οργάνωσης είναι το γεγονός ότι οι αποφάσεις που πρέπει να ληφθούν από το υλικό είναι απλοποιημένες. Είναι ουσιαστικά τόσο απλές, που δεν υπάρχει επιλογή: Μόνο μία θέση μπλοκ ελέγχεται για ταίριασμα ετικέτας, και μόνο το μπλοκ εκείνης της θέσης μπορεί να αντικατασταθεί. Στις πλήρως συσχετιζόμενες ή τις συνολοσυσχετιζόμενες κρυφές μνήμες, υπάρχουν πολλά μπλοκ, ένα από τα οποία θα πρέπει να επιλεγεί για αντικατάσταση σε περίπτωση αστοχίας. Υπάρχουν τρεις βασικές τεχνικές για την επιλογή του μπλοκ το οποίο θα αντικατασταθεί:

- *Τυχαία (Random)* – Προκειμένου να γίνεται ομοιόμορφα η κατανομή, επιλέγεται τυχαία ένα από τα υποψήφια μπλοκ. Κάποια συστήματα παράγουν ψευδοτυχαίους αριθμούς μπλοκ ώστε να έχουν συμπεριφορά που μπορεί να αναπαραχθεί, πράγμα πολύ χρήσιμο κατά την αποσφαλμάτωση του υλικού.
- *Λιγότερο-πρόσφατα Χρησιμοποιημένο (Least-recently Used – LRU)* – Προκειμένου να μειωθεί η πιθανότητα της αντικατάστασης πληροφοριών που θα χρειαστούν σύντομα, καταγράφονται οι προσπελάσεις στα μπλοκ. Χρησιμοποιώντας το παρελθόν για να προβλεφθεί το μέλλον, το μπλοκ που επιλέγεται να αντικατασταθεί είναι αυτό που δεν έχει χρησιμοποιηθεί για το μεγαλύτερο χρονικό διάστημα. Η τεχνική LRU βασίζεται σε ένα επακόλουθο της τοπικότητας: Αν τα πρόσφατα χρησιμοποιημένα μπλοκ είναι πιθανόν να χρησιμοποιηθούν ξανά, τότε ένας καλός υποψήφιος για αντικατάσταση είναι το μπλοκ που έχει χρησιμοποιηθεί λιγότερο πρόσφατα.
- *Πρώτο μέσα, πρώτο έξω (First in, first out – FIFO*) – Καθώς η τεχνική LRU απαιτεί σύνθετους υπολογισμούς για την υλοποίησή της, η τεχνική αυτή

προσεγγίζει την LRU καθορίζοντας το παλαιότερο μπλοκ, αντί για το λιγότερο πρόσφατα χρησιμοποιημένο.

Ένα από τα οφέλη της τυχαίας αντικατάστασης είναι η ευκολία υλοποίησής της σε υλικό. Καθώς ο αριθμός των μπλοκ που πρέπει να παρακολουθούνται αυξάνεται, η τεχνική LRU γίνεται ολοένα και πιο ακριβή στην υλοποίησή της και συχνά απλώς προσεγγίζεται. Ο Πίνακας 8 δείχνει τις διαφορές στον ρυθμό αστοχιών ανάμεσα στις τεχνικές LRU, Τυχαίας και FIFO αντικατάστασης.

	Συσχετισμός (Associativity)								
	2-τρόπων			4-τρόπων			8-τρόπων		
Μέγεθος	**LRU**	**Τυχαία**	**FIFO**	**LRU**	**Τυχαία**	**FIFO**	**LRU**	**Τυχαία**	**FIFO**
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Πίνακας 8: Σύγκριση τεχνικών LRU, Τυχαίας και FIFO αντικατάστασης σε αριθμό αστοχιών ανά 1000 εντολές, για διαφορετικά μεγέθη και συσχετισμούς κρυφής μνήμης

Οι μετρήσεις του Πίνακα 8 ελήφθησαν για μέγεθος μπλοκ των 64 bytes στην αρχιτεκτονική Alpha, χρησιμοποιώντας 10 SPEC2000 benchmarks. Όπως είναι φανερό, υπάρχει πολύ μικρή διαφορά ανάμεσα στην LRU και την τυχαία τεχνική αντικατάστασης για τα μεγαλύτερα μεγέθη κρυφής μνήμης, ενώ η LRU αποδίδει καλύτερα από τις άλλες τεχνικές για μικρότερες κρυφές μνήμες.

Λόγω των πολύ καλών επιδόσεων σε κρυφές μνήμες μεγαλύτερων μεγεθών (οι οποίες αποτελούν και τον στόχο στον παρόντα σχεδιασμό), σε συνδυασμό με την απλότητα υλοποίησής της, επιλέχθηκε η τεχνική Τυχαίας αντικατάστασης για την παρουσιαζόμενη κρυφή μνήμη δεύτερου επιπέδου.

4.5.Πολιτικές εγγραφής

Οι πολιτικές εγγραφής συχνά διαχωρίζουν τον σχεδιασμό τον κρυφών μνημών. Υπάρχουν δύο βασικές επιλογές για τις περιπτώσεις εγγραφής στην κρυφή μνήμη:

- *Write through* – Η πληροφορία γράφεται τόσο στο μπλοκ της κρυφής μνήμης όσο και στο μπλοκ της μνήμης κατώτερου επιπέδου.
- *Write back* – Η πληροφορία γράφεται μόνο στο μπλοκ της κρυφής μνήμης. Το τροποποιημένο μπλοκ της κρυφής μνήμης γράφεται στην κύρια μνήμη μόνο όταν αντικαθίσταται.

Προκειμένου να ελαττωθεί η συχνότητα εγγραφής μπλοκ στην κύρια μνήμη όταν αυτά αντικαθίστανται, χρησιμοποιείται τυπικά η τεχνική του *βρώμικου bit (dirty bit)*. Αυτό το bit κατάστασης δηλώνει αν το μπλοκ είναι *βρώμικο* (αν έχει τροποποιηθεί όσο βρισκόταν στην κρυφή μνήμη) ή *καθαρό* (αν δεν έχει τροποποιηθεί). Αν είναι καθαρό, το μπλοκ δε γράφεται στην κύρια μνήμη (ή σε κρυφή μνήμη χαμηλότερου επιπέδου) σε περίπτωση αστοχίας, εφόσον η ίδια ακριβώς πληροφορία που περιέχει η κρυφή μνήμη βρίσκεται και σε χαμηλότερα επίπεδα.

Και οι δύο τεχνικές έχουν τα πλεονεκτήματά τους. Στην περίπτωση της write back, οι εγγραφές πραγματοποιούνται όποτε αποφασίζει η κρυφή μνήμη και πολλαπλές εγγραφές στο ίδιο

μπλοκ απαιτούν μόνο μία εγγραφή στη μνήμη κατώτερου επιπέδου. Αφού κάποιες εγγραφές δεν προωθούνται στην κύρια μνήμη, αυτή η τεχνική χρησιμοποιεί μικρότερο εύρος ζώνης, καθιστώντας την επιθυμητή για πολυεπεξεργαστές. Εφόσον η τεχνική write back χρησιμοποιεί το υπόλοιπο της ιεραρχίας και διασύνδεσης μνήμης λιγότερο από την write through, εξοικονομεί και ισχύ, καθιστώντας την επιθυμητή για ενσωματωμένες εφαρμογές.

Από την άλλη πλευρά, η τεχνική write through υλοποιείται ευκολότερα από την write back. Η κρυφή μνήμη είναι πάντοτε καθαρή, οπότε αντίθετα με την write back, οι αστοχίες ανάγνωσης δεν έχουν ποτέ ως αποτέλεσμα εγγραφές στην μνήμη κατώτερου επιπέδου. Η τεχνική write through έχει επίσης το πλεονέκτημα ότι το επόμενο κατώτερο επίπεδο ανανεώνεται πολύ γρηγορότερα, πράγμα που απλοποιεί τη συνέπεια δεδομένων, καθιστώντας την επιθυμητή για πολυεπεξεργαστές. Η τεχνική αυτή είναι επίσης κατάλληλη για κρυφές μνήμες μεγαλύτερου επιπέδου, εφόσον οι εγγραφές χρειάζεται να προωθηθούν μόνο στο αμέσως επόμενο επίπεδο και όχι σε όλα τα επίπεδα μέχρι την κύρια μνήμη.

Οι επεξεργαστές και οι συσκευές I/O είναι λοιπόν αναποφάσιστοι: Θέλουν write back για τις κρυφές μνήμες προκειμένου να μειώσουν την κίνηση στις μνήμες και write through για να ενημερώνονται γρηγορότερα τα κατώτερα επίπεδα.

Όταν ο επεξεργαστής πρέπει να περιμένει την ολοκλήρωση της εγγραφής στην τεχνική write through, λέμε ότι έχουμε *καθυστέρηση εγγραφής (write stall)*. Μια διαδεδομένη βελτιστοποίηση για την αντιμετώπιση των καθυστερήσεων εγγραφής είναι η χρήση ενός *buffer εγγραφών*, ο οποίος επιτρέπει στον επεξεργαστή να συνεχίσει την εκτέλεση αμέσως μόλις τα δεδομένα γραφτούν στον buffer, πραγματοποιώντας έτσι παράλληλα την εκτέλεση του επεξεργαστή και την ανανέωση των μνημών κατώτερου επιπέδου.

Για την κρυφή μνήμη δεύτερου επιπέδου που παρουσιάζεται, επιλέχθηκε η τεχνική write through, έτσι ώστε η κύρια μνήμη να ανανεώνεται γρηγορότερα. Ένας ακόμα λόγος είναι η ύπαρξη της FIFO, η οποία μπορεί να παίξει ταυτόχρονα και τον ρόλο του buffer εγγραφής, οπότε η ήδη υπάρχουσα υποδομή προσφερόταν για την υλοποίηση της τεχνικής write through με ελάχιστη επιβάρυνση από πλευράς υλικού.

Εφόσον τα δεδομένα δεν είναι απαραίτητα σε μια εγγραφή, υπάρχουν δύο επιλογές σε περίπτωση αστοχίας εγγραφής:

- *Διανομή εγγραφής (Write allocate)* – Το μπλοκ μεταφέρεται στην κρυφή μνήμη σε περίπτωση αστοχίας εγγραφής, ακολουθούμενο από τις ενέργειες ευστοχίας εγγραφής που αναφέρθηκαν παραπάνω. Σε αυτή τη φυσική επιλογή, οι αστοχίες εγγραφής δρουν όπως και οι αστοχίες ανάγνωσης.
- *Χωρίς διανομή εγγραφής (No-write allocate)* – Σε αυτή την ασυνήθιστη επιλογή, οι αστοχίες εγγραφών *δεν* επηρεάζουν την κρυφή μνήμη. Αντίθετα, το μπλοκ τροποποιείται μόνο στη μνήμη κατώτερου επιπέδου.

Συνεπώς, τα μπλοκ μένουν έξω από την κρυφή μνήμη στην τεχνική No-write allocate έως ότου ο επεξεργαστής επιχειρήσει να τα διαβάσει, αλλά ακόμα και τα μπλοκ που έχουν απλώς εγγραφεί θα βρίσκονται στην κρυφή μνήμη με την τεχνική Write allocate. Στον παρόντα σχεδιασμό χρησιμοποιείται η τεχνική Διανομής εγγραφής (Write allocate).

Κεφάλαιο 5:
Υλοποίηση σχεδιασμού

Στο κεφάλαιο αυτό παρουσιάζεται αναλυτικά η υλοποίηση του σχεδιασμού στο συνθέσιμο τμήμα της γλώσσας περιγραφής υλικού VHDL. Με μια κοντινή ματιά στον κώδικα της υλοποίησης γίνονται φανεροί οι τρόποι με τους οποίους υλοποιούνται οι λειτουργίες της κρυφής μνήμης δεύτερου επιπέδου που περιγράφηκαν στα προηγούμενα κεφάλαια. Παρουσιάζεται επίσης η διασύνδεση των διαφόρων τμημάτων, καθώς και το υλικό που παράγεται από το εργαλείο σύνθεσης. Προκειμένου να είναι το κείμενο πιο ευανάγνωστο, ο κώδικας VHDL δεν παρατίθεται εδώ, αλλά είναι διαθέσιμος στο Παράρτημα.

5.1.Εισαγωγή

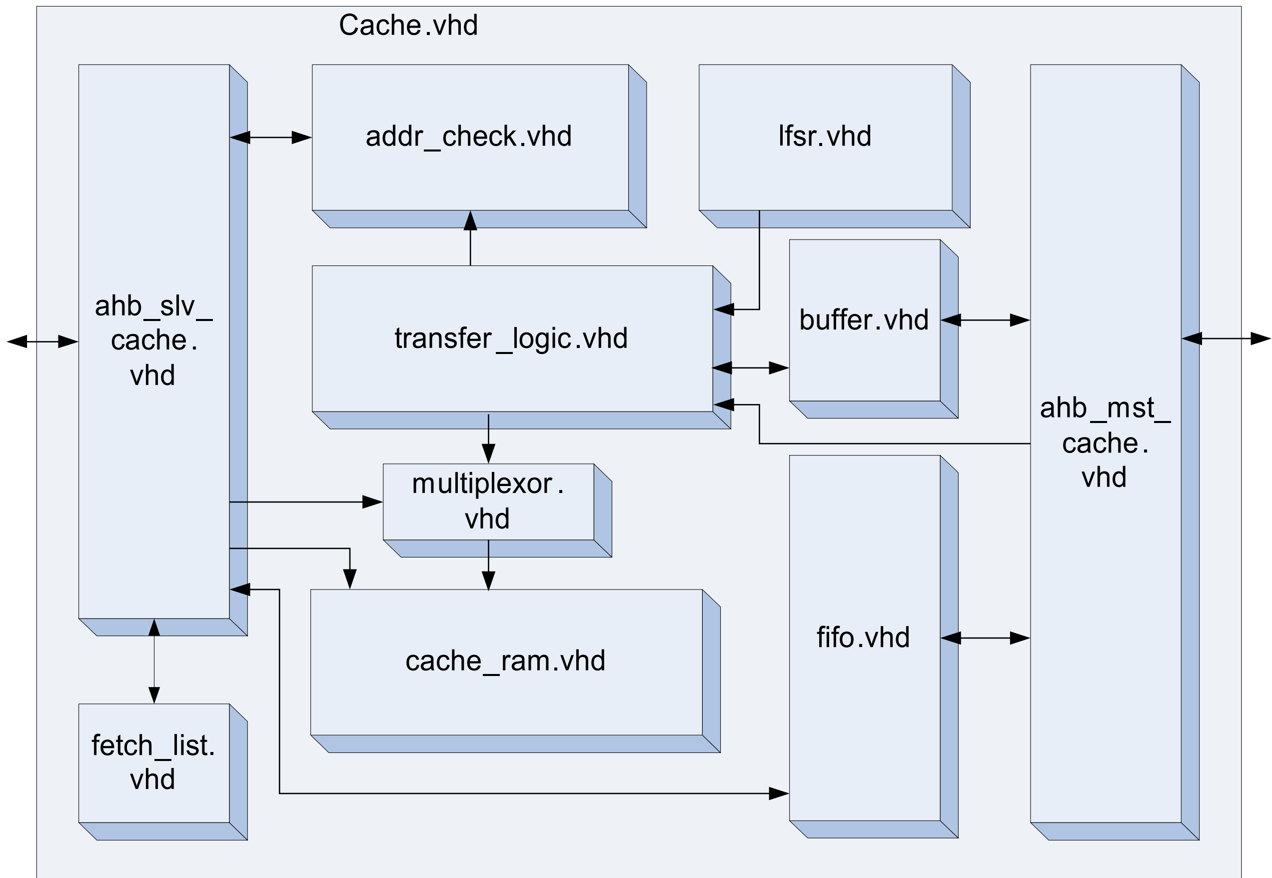
Ο σχεδιασμός της κρυφής μνήμης δεύτερου επιπέδου αποτελείται από έντεκα συνολικά τμήματα, το καθένα από τα οποία περιέχεται στο αντίστοιχο αρχείο VHDL. Τα τμήματα αυτά είναι:

1) **Cache.vhd** : Στο αρχείο αυτό περιέχεται η οντότητα περιτύλιξης (wrapper entity) του σχεδιασμού. Περιέχει δηλαδή την οντότητα που χρησιμοποιεί τα υπόλοιπα τμήματα του σχεδιασμού, δημιουργεί τα instances για το καθένα από αυτά και πραγματοποιεί τη μεταξύ τους διασύνδεση. Παράλληλα υπολογίζει κάποιες τιμές αρχικοποίησης βάσει επιλογών του χρήστη, και τις περνάει ως παραμέτρους στα υπόλοιπα τμήματα ώστε να συντεθεί το επιθυμητό υλικό. Με την τρόπο αυτό, ο τελικός χρήστης χρειάζεται μόνο να συμπεριλάβει την οντότητα περιτύλιξης στον σχεδιασμό του και να της δηλώσει τις επιλογές του για τη διαρρύθμιση των επιμέρους τμημάτων.
2) **ahb_slv_cache.vhd** : Στο αρχείο αυτό περιλαμβάνεται η υλοποίηση της διεπαφής της κρυφής μνήμης ως slave συσκευή του διαύλου AMBA AHB. Είναι το τμήμα που συνδέεται με τον δίαυλο AHB του επεξεργαστή (P-bus).
3) **addr_check.vhd** : Στο αρχείο αυτό υλοποιείται η μνήμη αποθήκευσης των ετικετών (tags) των μπλοκ της κρυφής μνήμης, καθώς και των bits εγκυρότητας (valid bits). Είναι το τμήμα που είναι υπεύθυνο για τον έλεγχο της εισερχόμενης διεύθυνσης και της απόφανσης αν το αιτηθέν μπλοκ περιέχεται στην κρυφή μνήμη ή όχι (αν έχουμε δηλαδή hit ή miss). Είναι φυσικά υπεύθυνο και για την ανανέωση των tags και των valid bits, όταν κάποιο μπλοκ μεταφέρεται στην κρυφή μνήμη.
4) **cache_ram.vhd :** Εδώ υλοποιείται η μνήμη αποθήκευσης των μπλοκ της κρυφής μνήμης. Αποτελεί δηλαδή την «καρδιά» της κρυφής μνήμης, όπου βρίσκονται αποθηκευμένες οι επιθυμητές πληροφορίες.
5) **fifo.vhd** : Το τμήμα αυτό, όπως περιγράφηκε στο Κεφάλαιο 3, είναι υπεύθυνο για δύο βασικές λειτουργίες. Πραγματοποιεί τον συγχρονισμό μεταξύ των δύο διαύλων AHB επεξεργαστή και μνήμης (σε συνδυασμό με το τμήμα που περιγράφεται στο αρχείο buffer.vhd) και λειτουργεί επίσης ως buffer αποθήκευσης των αιτήσεων που απευθύνονται στην κύρια μνήμη (εγγραφής ή ανάγνωσης), ούτως ώστε να

διευκολύνει την κατά το δυνατό απρόσκοπτη λειτουργία των νημάτων (threads) του επεξεργαστή.

6) **ahb_mst_cache.vhd** : Στο αρχείο αυτό περιλαμβάνεται η υλοποίηση της διεπαφής της κρυφής μνήμης ως master συσκευή του διαύλου AMBA AHB. Είναι το τμήμα που συνδέεται με τον δίαυλο AHB της κύριας μνήμης (M-bus).
7) **buffer.vhd** : Στο αρχείο αυτό υλοποιείται μια μνήμη προσωρινής αποθήκευσης, στην οποία αποθηκεύεται γραμμή-γραμμή το εκάστοτε μπλοκ που μεταφέρεται από την κύρια μνήμη. Όταν ο buffer γεμίσει (περιέχει δηλαδή ολόκληρο το μπλοκ), τότε το μπλοκ μπορεί να αποθηκευθεί στην κυρίως μνήμη της κρυφής μνήμης και να ανανεωθούν τα tags. Συμβάλλει επίσης στον συγχρονισμό των διαύλων και λειτουργεί παρόμοια με τη FIFO.
8) **lfsr.vhd** : Εδώ υλοποιείται η λογική που είναι υπεύθυνη για την παραγωγή ενός τυχαίου αριθμού, ο οποίος χρησιμοποιείται για την τυχαία επιλογή ενός μπλοκ μέσα από ένα σύνολο (set) της κρυφής μνήμης. Το μπλοκ αυτό είναι εκείνο που αντικαθίσταται όταν ένα νέο μπλοκ αντιγράφεται στην κρυφή μνήμη.
9) **transfer_logic.vhd** : Στο αρχείο αυτό υλοποιείται η λογική που είναι υπεύθυνη για τη μεταφορά ενός μπλοκ που έχει αποθηκευθεί στη μνήμη προσωρινής αποθήκευσης (που υλοποιείται στο αρχείο buffer.vhd) στην κυρίως μνήμη της cache. Όταν οι συνθήκες είναι κατάλληλες, πραγματοποιεί τη μεταφορά του μπλοκ και σηματοδοτεί τη λογική του αρχείου addr_check.vhd για ανανέωση των tags.
10) **multiplexor.vhd** : Στο αρχείο αυτό υλοποιείται ένας πολυπλέκτης ο οποίος είναι υπεύθυνος για την οδήγηση των εισόδων της κεντρικής μνήμης. Σε κανονική λειτουργία, οδηγεί στις εισόδους τις μνήμης τις εντολές που προέρχονται από τη slave διεπαφή της κρυφής μνήμης. Όταν πραγματοποιείται ωστόσο μεταφορά ενός μπλοκ από τον buffer στην μνήμη, ο πολυπλέκτης οδηγεί στις εισόδους της μνήμης τις επιλογές της λογικής μεταφοράς (που περιέχεται στο αρχείο transfer_logic.vhd).
11) **fetch_list.vhd** : Εδώ υλοποιείται ένα αρχείο καταχωρητών στο οποίο αποθηκεύονται οι διευθύνσεις των μπλοκ για τα οποία υπάρχει στη fifo αίτηση μεταφοράς από την κύρια μνήμη. Το αρχείο καταχωρητών αυτό χρησιμοποιείται από τη διεπαφή slave ahb_slv_cache, έτσι ώστε να μην εκχωρεί στη fifo αίτηση μεταφοράς του ίδιου μπλοκ δεύτερη φορά (πράγμα που υπό ορισμένες συνθήκες θα δημιουργούσε προβλήματα ασυνέπειας).

Στην Εικόνα 37 φαίνεται ο τρόπος που διασυνδέονται τα τμήματα λογικής που περιγράφονται από τα παραπάνω αρχεία. Τα ακριβή δεδομένα που ανταλλάσσουν τα διάφορα τμήματα λογικής δεν φαίνονται εδώ, αλλά περιγράφονται εκτενέστερα στην ξεχωριστή περιγραφή του κάθε τμήματος. Το σχήμα αυτό παρατίθεται εδώ για να δώσει μια γενική ιδέα της δομής του σχεδιασμού και για να προσφέρει ένα σημείο αναφοράς στη διασύνδεση των επιμέρους τμημάτων που θα διευκολύνει την επιμέρους περιγραφή τους.



Εικόνα 37: Διασύνδεση των τμημάτων του σχεδιασμού και τα αρχεία που τα περιγράφουν

5.2.Cache.vhd

Σκοπός του αρχείου αυτού είναι να ορίσει μία μόνο οντότητα, που ονομάζεται Cache, η οποία θα λειτουργεί σαν μαύρο κουτί για τον τελικό χρήστη. Ο χρήστης δηλαδή χρειάζεται μόνο να συμπεριλάβει την οντότητα αυτή στον σχεδιασμό του και να της δηλώσει τις επιλογές του, χωρίς να ασχοληθεί με τις λεπτομέρειες της εσωτερικής λογικής. Μέσα στην οντότητα Cache δημιουργούνται και διασυνδέονται αδιάφανα αντίγραφα της λογικής που περιγράφεται σε όλα τα υπόλοιπα αρχεία, βάσει πάντα των επιλογών του χρήστη.

Τα generics είναι επιλογές που δηλώνονται από τον χρήστη κατά την υλοποίηση ενός instance (ένα υλοποιημένο αντίγραφο της λογικής ενός σχεδιασμού) κάποιας οντότητας, προκειμένου να ρυθμιστούν διάφορες παράμετροι του σχεδιασμού. Καθώς η οντότητα Cache αποτελεί την οντότητα υψηλότερου επιπέδου του σχεδιασμού, ένα από τα καθήκοντά της είναι ο υπολογισμός διαφόρων παραμέτρων βάσει των επιλογών του χρήστη. Οι παράμετροι αυτές περνιούνται στη συνέχεια ως generics στα instances των επιμέρους τμημάτων του σχεδιασμού, έτσι ώστε κάθε instance να υλοποιεί την απαραίτητη λογική, όπως αυτή έχει επιλεγεί από τον χρήστη. Τα generics που χρειάζεται να ορισθούν από τον χρήστη για την οντότητα περιτύλιξης Cache, και τα οποία δηλώνουν τις επιλογές του σχετικά με τη διαμόρφωση της κρυφής μνήμης, είναι:

1) kbytes : Ακέραιος που δηλώνει το μέγεθος της κύριας μνήμης σε Kbytes.
2) cache_size : Ακέραιος που δηλώνει το μέγεθος της κρυφής μνήμης σε Kbytes.
3) block_size : Ακέραιος που δηλώνει το μέγεθος ενός μπλοκ σε bytes.
4) line_size : Ακέραιος που δηλώνει το μέγεθος μιας γραμμής της κρυφής μνήμης σε bits.
5) set_number : Ακέραιος που δηλώνει τον αριθμό των συνόλων (sets) που περιέχει η κρυφή μνήμη.
6) fifo_length : Ακέραιος που δηλώνει το επιθυμητό μήκος της fifo.

Μέσω των επιλογών αυτών ο χρήστης μπορεί να επιλέξει την ακριβή μορφή της κρυφής μνήμης, από το μέγεθος της μέχρι τον τρόπο οργάνωσής της (δηλαδή το associativity που χρησιμοποιεί).

Μια δεύτερη σειρά generics χρησιμεύει για την ταυτοποίηση της παρουσιαζόμενης συσκευής στους δύο διαύλους (επεξεργαστή και κύριας μνήμης). Οι παράμετροι με το πρόθεμα slv ταυτοποιούν την κρυφή μνήμη δεύτερου επιπέδου στον δίαυλο του επεξεργαστή ως συσκευή slave και ορίζουν το χώρο διευθύνσεών της (μέσω της αρχικής διεύθυνσης slv_haddr και της μάσκας slv_hmask). Οι παράμετροι με πρόθεμα mst ταυτοποιούν την κρυφή μνήμη δεύτερου επιπέδου ως συσκευή master στον δίαυλο της κύριας μνήμης.

Χρησιμοποιώντας τις παραπάνω επιλογές διαμόρφωσης, η οντότητα αυτή υπολογίζει όπως αναφέρθηκε προηγουμένως τις παραμέτρους υλοποίησης των επιμέρους τμημάτων. Οι παράμετροι αυτοί είναι:

1) abits : Ο αριθμός των bits διευθυνσιοδότησης της κύριας μνήμης. Υπολογίζεται βάσει του μεγέθους της κύριας μνήμης.
2) block_number : Ο αριθμός των θέσεων μπλοκ που περιέχει η κρυφή μνήμη. Υπολογίζεται βάσει του μεγέθους της κρυφής μνήμης και του μεγέθους ενός μπλοκ.
3) set_assoc : Ο τρόπος οργάνωσης της κρυφής μνήμης. Αν το set_assoc είναι 1, η κρυφή μνήμη είναι άμεσα αντιστοιχιζόμενη, αν είναι k (<n), τότε είναι συνολοσυσχετιζόμενη k-τρόπων, ενώ αν είναι n (όπου n ο αριθμός των μπλοκ), τότε είναι πλήρως συσχετιζόμενη. Υπολογίζεται από τον αριθμό των μπλοκ και τον αριθμό των συνόλων.
4) lines_number : Ο συνολικός αριθμός γραμμών που περιέχει η κρυφή μνήμη. Υπολογίζεται από το μέγεθος της κρυφής μνήμης και το μέγεθος γραμμής.
5) lines_block : Ο αριθμός των γραμμών ανά μπλοκ. Υπολογίζεται από τον συνολικό αριθμό γραμμών και των αριθμό των μπλοκ.
6) offset_bits : Ο αριθμός των bits στη διεύθυνση του επεξεργαστή, τα οποία καθορίζουν τη μετατόπιση (offset) μέσα σε ένα μπλοκ. Υπολογίζεται βάσει του αριθμού των γραμμών σε ένα μπλοκ.
7) index_bits : Ο αριθμός των bits στη διεύθυνση του επεξεργαστή, τα οποία καθορίζουν το σύνολο (set) στο οποίο θα πρέπει να αναζητηθεί ένα μπλοκ. Υπολογίζεται βάσει του αριθμού των συνόλων.

Οι παράμετροι αυτοί υπολογίζονται στο κομμάτι του κώδικα που ονομάζεται CONSTANT declaration.

Στο τμήμα του κώδικα που αναγράφεται COMPONENT declaration δηλώνονται τα τμήματα λογικής που περιλαμβάνονται στα υπόλοιπα αρχεία του σχεδιασμού. Στο τμήμα του κώδικα που αναγράφεται SIGNAL declaration ορίζονται οι γραμμές που χρησιμοποιούνται για τη διασύνδεση των επιμέρους τμημάτων. Τέλος, στο κυρίως σώμα της αρχιτεκτονικής της οντότητας Cache δημιουργούνται τα instances των επιμέρους τμημάτων του σχεδιασμού, υλοποιείται δηλαδή ένα αντίγραφο του υλικού που αυτά ορίζουν, βάσει των παραμέτρων που υπολογίστηκαν. Τα instances αυτά συνδέονται μεταξύ τους με τις γραμμές που έχουν οριστεί, ώστε να δημιουργήσουν ολόκληρο το σώμα (υλικό) της κρυφής μνήμης δεύτερου επιπέδου.

5.3.ahb_slv_cache.vhd

Η λογική του αρχείου αυτού υλοποιεί τη διεπαφή της κρυφής μνήμης δεύτερου επιπέδου με τον δίαυλο AMBA AHB του επεξεργαστή (P-bus). Στον δίαυλο αυτό, η κρυφή μνήμη οφείλει να λειτουργεί ως συσκευή slave, εφόσον ο επεξεργαστής είναι ο μοναδικός master. Η διεπαφή αυτή είναι λοιπόν υπεύθυνη για τη λήψη των αιτήσεων του επεξεργαστή (αν φυσικά αυτές αντιστοιχούν στον χώρο διευθύνσεών της), την ορθή απόκριση της κρυφής μνήμης στις αιτήσεις αυτές (OK ή RETRY), την κατάλληλη ενημέρωση της κεντρικής μνήμης της κρυφής μνήμης, την ενημέρωση της λίστας των μπλοκ προς μεταφορά, και τέλος την αποθήκευση στη fifo των αιτήσεων που θα πρέπει να προωθηθούν στην κύρια μνήμη.

Οι είσοδοι της οντότητας αυτής είναι:

- **hit** : Από addr_check. Δηλώνει αν η αιτηθείσα διεύθυνση υπάρχει στην κρυφή μνήμη. Υπολογίζεται ασύγχρονα και είναι διαθέσιμη στον ίδιο κύκλο με τη διεύθυνση.
- **line_pointer** : Από addr_check. Αποτελεί έναν ακέραιο που δείχνει τη γραμμή της κεντρικής μνήμης της κρυφής μνήμης, η οποία αντιστοιχεί στη διεύθυνση που ζητήθηκε. Αν η γραμμή αυτή δεν υπάρχει στην κρυφή μνήμη, η είσοδος line_pointer δείχνει στην τελευταία γραμμή του συνόλου στο οποίο θα αντιστοιχούσε το μπλοκ αν ήταν διαθέσιμο, και απλώς δε χρησιμοποιείται. Υπολογίζεται ασύγχρονα και είναι διαθέσιμη στον ίδιο κύκλο με τη διεύθυνση.
- **rst** : Από τον δίαυλο του επεξεργαστή. Είναι το σήμα reset.
- **clk** : Από τον δίαυλο του επεξεργαστή. Είναι το ρολόι του διαύλου του επεξεργαστή.
- **ahbsi** : Από τον δίαυλο του επεξεργαστή. Είναι το σύνολο των πληροφοριών που δέχεται η κρυφή μνήμη από τον δίαυλο του επεξεργαστή, όπως η αιτηθείσα διεύθυνση, τα δεδομένα προς εγγραφή, το σήμα επιλογής της κρυφής μνήμης (από τον decoder) κ.ά.
- **ramdata** : Από cache_ram. Είναι η γραμμή που περιέχει τα δεδομένα που διαβάζονται από την κεντρική μνήμη σε μία ανάγνωση. Η οντότητα ahb_slv_cache απλώς τα προωθεί στον δίαυλο.
- **fifo_full** : Από fifo. Δηλώνει ότι η fifo είναι γεμάτη και συνεπώς δεν μπορούν να γίνουν άλλες εγγραφές.
- **fifo_near_full** : Από fifo. Δηλώνει ότι η fifo έχει μόνο μία θέση ακόμα κενή και με μια ακόμα εγγραφή θα γεμίσει. Η χρησιμότητα της εισόδου αυτής περιγράφεται παρακάτω.
- **transfer_mode** : Από transfer_logic. Δηλώνει ότι η κρυφή μνήμη πραγματοποιεί μεταφορά ενός μπλοκ από τον buffer στην κεντρική μνήμη. Στο διάστημα της μεταφοράς δεν μπορούν να ικανοποιηθούν αιτήσεις για εγγραφή ή ανάγνωση, οπότε η διεπαφή ahb_slv_cache πρέπει να δίνει απάντηση RETRY στον επεξεργαστή.
- **fetch_match** : Από fetch_list. Δηλώνει ότι υπάρχει ήδη στη fifo μια αίτηση για μεταφορά του αιτηθέντος μπλοκ από την κύρια στην κρυφή μνήμη (fetch), οπότε δεν πρέπει να ξαναμπεί. Υπολογίζεται ασύγχρονα και είναι διαθέσιμη στον ίδιο κύκλο με τη διεύθυνση.

Οι έξοδοι της οντότητας αυτής είναι:

- **ahbso** : Προς τον δίαυλο του επεξεργαστή. Είναι το σύνολο των πληροφοριών που εξάγει η κρυφή μνήμη στον δίαυλο του επεξεργαστή, όπως τα αναγνωσμένα από την κρυφή μνήμη δεδομένα, οι απαντήσεις της κρυφής μνήμης (OK ή RETRY) κ.ά.
- **ramaddr** : Προς multiplexor (και εν τέλει προς cache_ram). Αποτελεί έναν ακέραιο που δηλώνει στην κεντρική μνήμη τη γραμμή που πρέπει να προσπελαστεί.

- **write_data** : Προς multiplexor (και εν τέλει προς cache_ram). Είναι τα δεδομένα προς εγγραφή στην κεντρική μνήμη.
- **write** : Προς multiplexor (και εν τέλει προς cache_ram). Είναι 4 bits που δηλώνουν ποια bytes της επιλεχθείσας γραμμής θα πρέπει να εγγραφούν.
- **fifo_write_en** : Προς fifo. Είναι η γραμμή ενεργοποίησης εγγραφής. Αν στη θετική ακμή του ρολογιού η γραμμή αυτή οδηγείται στο 1, η fifo θα πρέπει να πραγματοποιήσει μια εγγραφή.
- **fifo_entry** : Προς fifo. Είναι τα δεδομένα προς εγγραφή στη fifo, δηλαδή οι αιτήσεις που θα πρέπει να προωθηθούν στην κύρια μνήμη.
- **fifo_write_data** : Προς fifo. Σε περίπτωση αποθήκευσης στη fifo αίτησης εγγραφής στην κύρια μνήμη, αποθηκεύονται επίσης τα δεδομένα προς εγγραφή στην κύρια μνήμη. Τα δεδομένα αυτά βρίσκονται στη γραμμή αυτή.
- **fetch_data_in** : Προς fetch_list. Είναι τα δεδομένα προς αποθήκευση στη μνήμη fetch_list κάθε φορά που αποθηκεύεται στη fifo μια αίτηση για ανάγνωση από την κύρια μνήμη. Πρόκειται μόνο για το τμήμα της διεύθυνσης που καθορίζει το μπλοκ (χωρίς offset και άλλες πληροφορίες).
- **fetch_write_en** : Προς fetch_list. Αν στη θετική ακμή του ρολογιού η γραμμή αυτή οδηγείται στο 1, η μνήμη fetch_list θα πρέπει να πραγματοποιήσει μια εγγραφή.

Στην αρχή της περιγραφής της υλοποίησης, θα πρέπει να αναφερθούν λίγα πράγματα για τη μορφή του κώδικα. Ο κώδικας της αρχιτεκτονικής της παρούσας οντότητας, αποτελείται από δύο διεργασίες (processes). Στην πρώτη διεργασία (comb) περιγράφεται το συνδυαστικό τμήμα της λογικής. Αυτό το τμήμα είναι που λαμβάνει τις «αποφάσεις». Οι αποφάσεις όμως αυτές χρειάζονται πληροφορίες σχετικά με το τι συνέβη και στον προηγούμενο κύκλο. Για τον λόγο αυτό ορίζεται ένας τύπος δεδομένων reg_type, ο οποίος αποτελείται ουσιαστικά από ένα σύνολο υπο-τύπων, οι οποίοι αναπαριστούν τα στοιχεία που πρέπει να αποθηκευθούν από κύκλο σε κύκλο. Οι μεταβολές που πραγματοποιεί η διεργασία comb κάθε φορά που εκτελείται (όποτε αλλάζει μια είσοδός της) αποθηκεύονται σε μια μεταβλητή τύπου reg_type, ενώ στο τέλος κάθε εκτέλεσής της η μεταβλητή αυτή ανατίθεται σε ένα σήμα τύπου reg_type που ονομάζεται c. Στη δεύτερη διεργασία (reg), σε κάθε θετική ακμή του ρολογιού το σήμα c ανατίθεται στο σήμα r. Συνεπώς το σήμα r περιέχει τις απαραίτητες πληροφορίες για τον προηγούμενο κύκλο, και ως εκ τούτου κάθε υποσήμα του σήματος r υλοποιείται σε υλικό ως καταχωρητής. Η λειτουργία λοιπόν της οντότητας έχει ως εξής: Κατά τη διάρκεια ενός κύκλου ρολογιού η συνδυαστική λογική βασίζεται στις εισόδους και τα δεδομένα των καταχωρητών (πληροφορίες προηγούμενου κύκλου) για να οδηγήσει ασύγχρονα τα απαραίτητα σήματα, μεταξύ των οποίων και οι νέες είσοδοι των καταχωρητών. Με μια θετική ακμή ρολογιού, οι είσοδοι αυτοί αποθηκεύονται στους καταχωρητές και αρχίζει η διαδικασία «λήψης αποφάσεων» για τον επόμενο κύκλο. Αυτός ο τρόπος περιγραφής της λειτουργίας χρησιμοποιείται και σε άλλα αρχεία του σχεδιασμού, όπως η διεπαφή master του ahb_mst_cache.

Λόγω της pipelined λειτουργίας του πρωτοκόλλου AHB, υπάρχει η ανάγκη αποθήκευσης της διεύθυνσης που αιτήθηκε στον προηγούμενο κύκλο. Αυτή μπορεί να χρησιμοποιηθεί σε δύο περιπτώσεις:

1) Η πιο συνηθισμένη είναι η απλή λειτουργία εγγραφής. Σε μια λειτουργία εγγραφής, η διεύθυνση οδηγείται στον δίαυλο έναν κύκλο πριν από τα δεδομένα. Ωστόσο στην κεντρική μνήμη της κρυφής μνήμης, η διεύθυνση και τα δεδομένα πρέπει να μπουν στον ίδιο κύκλο, έτσι ώστε με την επόμενη θετική ακμή να πραγματοποιηθεί η εγγραφή.
2) Η δεύτερη περίπτωση είναι ανάγνωση αμέσως μετά από μια αίτηση εγγραφής. Καθώς σε μια εγγραφή αποθηκεύεται η διεύθυνση εγγραφής και εισάγεται στην κεντρική μνήμη μαζί με τα δεδομένα του επόμενου κύκλου, αν ο επεξεργαστής επιχειρήσει

ανάγνωση στον κύκλο αυτό (η οποία κανονικά πραγματοποιείται άμεσα σε έναν κύκλο), τότε η διεύθυνση ανάγνωσης δεν μπορεί να εισαχθεί στην κεντρική μνήμη γιατί οδηγείται ήδη από τη διεύθυνση εγγραφής. Σε αυτήν την περίπτωση, η διεύθυνση αποθηκεύεται για να εξυπηρετηθεί στον επόμενο κύκλο, ενώ παράλληλα εισάγεται και ένα στάδιο καθυστέρησης, οδηγώντας στον επόμενο κύκλο το σήμα HREADY σε Low, όπως περιγράφηκε στο Κεφάλαιο 2. Έτσι ο επεξεργαστής θα επεκτείνει την επόμενη λειτουργία του, έτσι ώστε να προλάβει να ολοκληρωθεί η προηγούμενη λειτουργία ανάγνωσης.

Αν ισχύει μία από τις παραπάνω περιπτώσεις ο αριθμός επιλογής γραμμής που εισάγεται στην κεντρική μνήμη είναι αυτός που είχε αποθηκευθεί στον προηγούμενο κύκλο. Αυτή είναι η λειτουργία που επιτελεί η πρώτη δήλωση if της αρχιτεκτονικής.

Στην δεύτερη δήλωση if απλώς ανανεώνονται κάποιες μεταβλητές (μεταξύ των οποίων και ο δείκτης γραμμής ή το bit εγγραφής ή ανάγνωσης) σύμφωνα με τις νέες εισόδους, αρκεί το HREADY να είναι High. Με τον επόμενο θετικό παλμό, οι πληροφορίες αυτές θα αποθηκευθούν στους καταχωρητές.

Στην τρίτη δήλωση if τίθενται τα bits εγγραφής - τα οποία όπως περιγράφηκε επιλέγουν ποια από τα bytes της κεντρικής μνήμης θα εγγραφούν- , αν στον προηγούμενο κύκλο είχε φτάσει μια εύστοχη αίτηση εγγραφής, αρκεί φυσικά ο κύκλος εκείνος να μην ήταν μέρος μιας απάντησης RETRY ή στάδιο αναμονής. Τα bits εγγραφής τίθενται βάσει του μεγέθους της εγγραφής (όπως ορίζεται από το σήμα HSIZE του Κεφαλαίου 3) και τα δύο πρώτα (λιγότερο σημαντικά) bits της διεύθυνσης. Εδώ αποφασίζεται επίσης, λαμβάνοντας υπ'όψιν και το είδος της αιτηθείσας προσπέλασης του παρόντος κύκλου (εγγραφή ή ανάγνωση), αν στον επόμενο κύκλο θα πρέπει να εισαχθεί στάδιο αναμονής (η περίπτωση 2 που αναφέρθηκε παραπάνω).

Η τέταρτη δήλωση if αποφασίζει αν στον επόμενο κύκλο πρέπει αν δοθεί απάντηση RETRY. Η απάντηση RETRY δίνεται στις εξής περιπτώσεις:

1) Αν έχουμε οποιαδήποτε προσπέλαση (εγγραφή ή ανάγνωση), και αστοχία στην κρυφή μνήμη, οπότε θα πρέπει το αιτηθέν μπλοκ να μεταφερθεί από την κύρια μνήμη. Η προσπέλαση θα πρέπει φυσικά να μην είναι IDLE και ο παρών κύκλος να μην είναι στάδιο αναμονής.
2) Αν έχουμε εγγραφή, ευστοχία στην κρυφή μνήμη, αλλά η fifo είναι γεμάτη ή έχει μόνο μία θέση κενή. Αν έχει μόνο μία θέση κενή, η θέση αυτή μπορεί να εγγραφεί στον παρόντα κύκλο (από μια αίτηση εγγραφής του προηγούμενου κύκλου), οπότε προληπτικά θα πρέπει να δοθεί απάντηση RETRY και να αποτραπεί άλλη εγγραφή στη fifo. Η ανάγκη για αυτό το μέτρο πρόληψης προκύπτει από την pipelined λειτουργία του διαύλου AHB, η οποία καθορίζει ότι θα πρέπει να δοθεί απάντηση μετά από τον πρώτο κύκλο (κύκλο διεύθυνσης) μιας λειτουργίας εγγραφής, ενώ στον ίδιο κύκλο μπορεί να ολοκληρώνεται μια προηγούμενη εγγραφή. Τα παραπάνω φυσικά ισχύουν αν η προσπέλαση δεν είναι IDLE και ο παρών κύκλος δεν είναι στάδιο αναμονής.
3) Αν η κρυφή μνήμη πραγματοποιεί μια μεταφορά μπλοκ από τον buffer στην κεντρική μνήμη, οπότε η απάντηση σε όλες τις αιτήσεις του επεξεργαστή είναι RETRY. Μόνη εξαίρεση είναι μια αίτηση IDLE μεταφοράς, στην οποία περίπτωση η απάντηση πρέπει να είναι OK.

Η πέμπτη δήλωση if απλώς εισάγει τον δεύτερο κύκλο της απάντησης RETRY αν ο παρών κύκλος είναι ο πρώτος κύκλος μιας απάντησης RETRY.

Η έκτη δήλωση if αποφασίζει αν στον επόμενο θετικό παλμό θα πρέπει να γίνει εγγραφή στην fifo και στη μνήμη fetch_list. Η εγγραφή στη fifo και στη μνήμη fetch_list πραγματοποιείται πάντα στον δεύτερο κύκλο μιας προσπέλασης, ανεξάρτητα αν αυτή είναι εγγραφή ή ανάγνωση. Στην περίπτωση εγγραφής είναι απαραίτητο να γίνει στον δεύτερο κύκλο, εφόσον μόνο σε αυτόν είναι διαθέσιμα τα προς εγγραφή δεδομένα, τα οποία πρέπει να αποθηκευθούν μαζί με τη διεύθυνση στη fifo για να γίνει η εγγραφή και στην κύρια μνήμη. Στην περίπτωση ανάγνωσης, η

εγγραφή στη fifo γίνεται στον δεύτερο κύκλο, για να αποφεύγονται συγκρούσεις με τις περιπτώσεις εγγραφής, άλλο ένα επακόλουθο της pipelined λειτουργίας του διαύλου AHB. Εγγραφή στη fifo πραγματοποιείται στις παρακάτω περιπτώσεις:

1) Αν ο προηγούμενος κύκλος ήταν μια έγκυρη εύστοχη αίτηση εγγραφής. Στην περίπτωση αυτή στη fifo αποθηκεύεται ένα bit (στο λογικό 1) που δηλώνει αίτηση εγγραφής, το μέγεθος της εγγραφής, η διεύθυνση της εγγραφής, καθώς και τα δεδομένα προς εγγραφή. Η αίτηση εγγραφής δεν είναι έγκυρη αν η αίτηση προσπέλασης ήταν τύπου IDLE, αν βρισκόμαστε στο μέσο μιας απάντησης RETRY (οπότε η αίτηση που εισάγεται στον πρώτο κύκλο πρέπει να αγνοηθεί), αν η κρυφή μνήμη πραγματοποιεί μεταφορά μπλοκ από τον buffer στην κεντρική μνήμη, αν η fifo είναι γεμάτη ή έχει μία θέση κενή (για λόγους που περιγράφηκαν παραπάνω), ή αν ο προηγούμενος κύκλος ήταν στάδιο αναμονής (οπότε η αίτηση αγνοείται και επαναλαμβάνεται από τον επεξεργαστή στον παρόντα κύκλο). Η περίπτωση αυτή υλοποιεί ουσιαστικά τη λειτουργία write-though της κρυφής μνήμης.
2) Αν ο προηγούμενος κύκλος ήταν μια έγκυρη άστοχη αίτηση προσπέλασης. Στην περίπτωση αυτή στη fifo αποθηκεύεται ένα bit (στο λογικό 0) που δηλώνει αίτηση μεταφοράς μπλοκ από την κύρια μνήμη (fetch) και η διεύθυνση της προσπέλασης. Το μέγεθος της προσπέλασης, καθώς και το τμήμα της fifo που χρησιμοποιείται για την αποθήκευση δεδομένων προς εγγραφή αφήνονται κενά, καθώς οι πληροφορίες αυτές είναι περιττές για τη μεταφορά ενός μπλοκ. Η αίτηση προσπέλασης δεν είναι έγκυρη αν ήταν τύπου IDLE, αν βρισκόμαστε στο μέσο μιας απάντησης RETRY (οπότε η αίτηση που εισάγεται στον πρώτο κύκλο πρέπει να αγνοηθεί), αν η κρυφή μνήμη πραγματοποιεί μεταφορά μπλοκ από τον buffer στην κεντρική μνήμη, αν η fifo είναι γεμάτη, ή αν ο προηγούμενος κύκλος ήταν στάδιο αναμονής (οπότε η αίτηση αγνοείται και επαναλαμβάνεται από τον επεξεργαστή στον παρόντα κύκλο). Εγγραφή στη fifo δεν πραγματοποιείται επίσης αν υπάρχει ήδη στη fifo μια αίτηση για μεταφορά του αιτηθέντος μπλοκ από την κύρια μνήμη στην κρυφή. Αυτός ο έλεγχος πραγματοποιείται στο fetch_list και φαίνεται από την είσοδο fetch_match.

Εγγραφή στη μνήμη fetch_list πραγματοποιείται μόνο στη δεύτερη από τις παραπάνω περιπτώσεις, εφόσον σκοπός της μνήμης αυτής είναι να κρατά τις αιτήσεις για μεταφορά μπλοκ από την κύρια μνήμη που υπάρχουν στη fifo, ώστε να μην επαναλαμβάνονται. Στη μνήμη fetch_list αποθηκεύεται μόνο το τμήμα της διεύθυνσης που καθορίζει το μπλοκ (χωρίς το offset και άλλες πληροφορίες).

Η τελευταία δήλωση if αποτελεί απλώς την αρχικοποίηση της λογικής σε περίπτωση reset.

5.4.addr_check.vhd

Η λογική του αρχείου αυτού περιλαμβάνει την υλοποίηση της μνήμης αποθήκευσης των ετικετών (tags) της κρυφής μνήμης, καθώς και των bits εγκυρότητας (valid bits). Κάθε φορά που ο επεξεργαστής εισάγει μια καινούρια διεύθυνση στον δίαυλο P-bus, η λογική αυτή εξετάζει ασύγχρονα αν το μπλοκ το οποίο στοχεύει η προσπέλαση περιέχεται στην κρυφή μνήμη ή όχι, και ενημερώνει τη διεπαφή slave με τον δίαυλο P-bus (ahb_slv_cache.vhd), ούτως ώστε εκείνη με τη σειρά της να πραγματοποιήσει ή όχι την προσπέλαση και να ενημερώσει κατάλληλα τη fifo, αν βέβαια χρειάζεται. Η λογική αυτή είναι βέβαια υπεύθυνη και για την ενημέρωση των tags και των valid bits.

Οι είσοδοι της οντότητας αυτής είναι:

- **clk** : Από τον δίαυλο του επεξεργαστή. Είναι το ρολόι του διαύλου του επεξεργαστή.

- **rst** : Από τον δίαυλο του επεξεργαστή. Είναι το σήμα reset.
- **ahbsi_haddr** : Από τον δίαυλο του επεξεργαστή. Πρόκειται για τη διεύθυνση που επιχειρεί να προσπελάσει ο επεξεργαστής.
- **tag_in** : Από transfer_logic. Είναι το νέο tag που πρέπει να εγγραφεί όταν ένα μπλοκ ξεκινά να μεταφέρεται από τον buffer στην κεντρική μνήμη.
- **tag_write_en** : Από transfer_logic. Στην θετική ακμή του ρολογιού, αν η είσοδος αυτή είναι 1, πρέπει να γίνει εγγραφή ενός νέου tag. Τίθεται από το transfer_logic όταν ξεκινά μεταφορά από τον buffer στην κύρια μνήμη.
- **block_num** : Από transfer_logic. Ακέραιος που δηλώνει τη θέση μπλοκ στην οποία αντιστοιχεί το tag που θα γραφτεί. Αποτελεί ουσιαστικά τη γραμμή στη μνήμη των ετικετών (tags) στην οποία θα γίνει η εγγραφή.

Οι έξοδοι της οντότητας αυτής είναι:

- **line_pointer** : Προς ahb_slv_cache. Ακέραιος που δηλώνει τη γραμμή της κεντρικής μνήμης της κρυφής μνήμης την οποία επιχειρεί να προσπελάσει ο επεξεργαστής. Αν η γραμμή που επιχειρεί να προσπελάσει ο επεξεργαστής δεν βρίσκεται στην κρυφή μνήμη, ο ακέραιος αυτός δείχνει την τελευταία γραμμή του συνόλου της κρυφής μνήμης στο οποίο θα βρισκόταν το αιτηθέν μπλοκ αν ήταν διαθέσιμο, και απλώς δε χρησιμοποιείται από τη λογική του ahb_slv_cache.vhd.
- **hit** : Προς ahb_slv_cache. Δηλώνει αν η γραμμή που προσπαθεί να προσπελάσει ο επεξεργαστής βρίσκεται στην κρυφή μνήμη ή όχι. Δηλώνει δηλαδή αν έχουμε ευστοχία ή αστοχία κρυφής μνήμης.

Η λογική του αρχείου αυτού ορίζει ένα νέο component, το οποίο ονομάζεται comparator. Αποτελεί έναν απλό συγκριτή, η λειτουργία του οποίου είναι να συγκρίνει το tag της αιτηθείσας διεύθυνσης με το tag ενός αποθηκευμένου στην κρυφή μνήμη μπλοκ. Με χρήση της εντολής generate της VHDL (η οποία δημιουργεί υπό συνθήκη ή επαναληπτικά αντίγραφα κάποιας υλοποιημένης λογικής), υλοποιούνται τόσοι comparators, όσος είναι ο συνολοσυσχετισμός της μνήμης, δηλαδή όσα μπλοκ υπάρχουν σε ένα σύνολο της μνήμης.

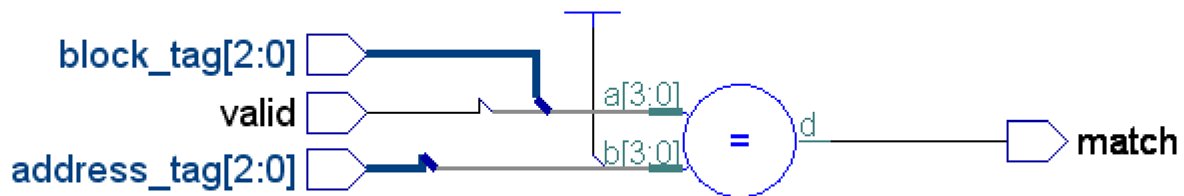
Μια επιπλέον ιδιαιτερότητα που εμφανίζεται για πρώτη φορά στο αρχείο αυτό είναι η χρήση της εντολής generate για να υλοποιήσει υπό συνθήκη διαφορετικές εκδοχές της ίδιας διεργασίας (process) VHDL. Η λογική της οντότητας addr_check απαιτεί την ανάγνωση των index bits (τα bits που επιλέγουν το σύνολο της κρυφής μνήμης) ή των offset bits (τα bits που επιλέγουν τη μετατόπιση μέσα σε ένα μπλοκ, δηλαδή ποια γραμμή του μπλοκ θα προσπελαστεί) της διεύθυνσης. Το πρόβλημα που υπάρχει εδώ είναι ότι ανάλογα με τις επιλογές του χρήστη, τα bits αυτά μπορεί να μην ορίζονται. Αν για παράδειγμα η κρυφή μνήμη έχει μόνο ένα σύνολο (είναι δηλαδή πλήρως συσχετιζόμενη), δεν ορίζονται bits επιλογής συνόλου. Αντίστοιχα, αν κάθε μπλοκ αποτελείται από μία μόνο γραμμή, δεν ορίζονται bits επιλογής μετατόπισης. Έχουν λοιπόν γραφτεί τέσσερις ελαφρώς παραλλαγμένες εκδοχές του τμήματος της λογικής που χρησιμοποιεί τα Bits αυτά (η διεργασία dex), μία για κάθε συνδυασμό των παραπάνω περιπτώσεων. Ανάλογα με τις επιλογές του χρήστη, μέσω της εντολής generate της VHDL, το εργαλείο σύνθεσης υλοποιεί την κατάλληλη εκδοχή της διεργασίας dex.

Ο κώδικας είναι δομημένος ως εξής. Η διεργασία tag_update είναι υπεύθυνη μόνο για την εγγραφή ενός νέου tag όταν ξεκινά να μεταφέρεται ένα μπλοκ από τον buffer στην κεντρική μνήμη.

Η διεργασία dex επιλέγει βάσει των index bits το σύνολο στο οποίο θα βρίσκεται το αιτηθέν μπλοκ αν αυτό βρίσκεται στην κρυφή μνήμη, και οδηγεί τα tags που αντιστοιχούν στα μπλοκ του συνόλου αυτού στους συγκριτές (comparators) που αναφέρθηκαν παραπάνω. Σε υλικό αυτό μεταφράζεται σε ένα σύνολο πολυπλεκτών, οι οποίοι, οδηγούμενοι από τα index bits της

διεύθυνσης, επιλέγουν τις κατάλληλες γραμμές της μνήμης ετικετών (τα tags δηλαδή που αντιστοιχούν στο επιλεγμένο σύνολο) και τις οδηγούν στις εισόδους των συγκριτών.

Ο καθένας από τους συγκριτές (Εικόνα 38) ελέγχει αν το tag με το οποίο οδηγείται ισούται με το tag του μπλοκ που ζήτησε ο επεξεργαστής και οδηγεί την έξοδό του στο 1 αν υπήρξε ταίριασμα.



Εικόνα 38: Ένας απλός συγκριτής, όπως παράγεται από το εργαλείο σύνθεσης

Τέλος, η διεργασία res ελέγχει τα αποτελέσματα των συγκριτών και αν κάποιο από αυτά είναι 1, σημαίνει ότι υπήρξε ευστοχία στην κρυφή μνήμη και οδηγεί την έξοδο hit στο 1 για να δηλώσει το γεγονός αυτό. Σε πραγματικό υλικό, οι έξοδοι των συγκριτών περνούν από μια σειρά πυλών OR δύο εισόδων και το τελικό αποτέλεσμα οδηγεί την έξοδο hit. Η διεργασία αυτή ελέγχει επίσης αν και ποιος συγκριτής είχε ευστοχία και χρησιμοποιεί την πληροφορία αυτή (μαζί με το επιλεγμένο σύνολο και το offset που προκύπτουν από τη διεργασία dex) για να υπολογίσει τον ακέραιο που δηλώνει τη γραμμή της κρυφής μνήμης την οποία προσπαθεί να προσπελάσει ο επεξεργαστής. Ο ακέραιος αυτός οδηγείται στην έξοδο line_pointer που περιγράφηκε παραπάνω.

5.5.cache_ram.vhd

Στη λογική του αρχείου αυτού υλοποιείται η κεντρική μνήμη της κρυφής μνήμης δεύτερου επιπέδου. Αποτελεί δηλαδή τη μνήμη στην οποία αποθηκεύονται τα μπλοκ που μεταφέρονται από την κύρια μνήμη και τα οποία μπορεί να προσπελάσει ο επεξεργαστής χωρίς να χρειαστεί να προσφύγει σε μνήμη κατώτερου επιπέδου.

Οι είσοδοι της οντότητας αυτής είναι:

- **clk** : Από τον δίαυλο του επεξεργαστή. Είναι το ρολόι του διαύλου του επεξεργαστή.
- **rst** : Από τον δίαυλο του επεξεργαστή. Είναι το σήμα reset.
- **line_pointer** : Από multiplexor (και εμμέσως είτε από ahb_slv_cache, είτε από transfer_logic). Ακέραιος που δηλώνει τη γραμμή της κεντρικής μνήμης της κρυφής μνήμης που πρέπει να προσπελαστεί. Η γραμμή αυτή είτε αποτελεί τη γραμμή που επιθυμεί να προσπελάσει ο επεξεργαστής (σε κανονική λειτουργία), είτε αποτελεί τη γραμμή στην οποία θα γραφτεί η επόμενη γραμμή του buffer (σε λειτουργία μεταφοράς ενός μπλοκ από τον buffer στην παρούσα κεντρική μνήμη). Στην πρώτη περίπτωση ο multiplexor περνάει στη μνήμη ram τον δείκτη γραμμής που ορίζει η οντότητα ahb_slv_cache, ενώ στη δεύτερη τον δείκτη γραμμής που ορίζει η οντότητα transfer_logic.
- **data_in** : Από multiplexor (και εμμέσως είτε από ahb_slv_cache, είτε από transfer_logic). Είναι τα δεδομένα προς εγγραφή στην μνήμη. Προέρχονται είτε από ahb_slv_cache σε

κανονική κατάσταση λειτουργίας της κρυφής μνήμης, είτε από transfer_logic (και εμμέσως από τον buffer) σε κατάσταση μεταφοράς ενός μπλοκ από τον buffer στην κεντρική μνήμη.

- **write_ar** : Από multiplexor (και εμμέσως είτε από ahb_slv_cache, είτε από transfer_logic). Ένα σύνολο τεσσάρων bits που ελέγχουν την εγγραφή σε καθεμία από τις τέσσερις single_ram που περιγράφονται παρακάτω. Προέρχονται είτε από ahb_slv_cache σε κανονική κατάσταση λειτουργίας της κρυφής μνήμης, είτε από transfer_logic σε κατάσταση μεταφοράς ενός μπλοκ από τον buffer στην κεντρική μνήμη.

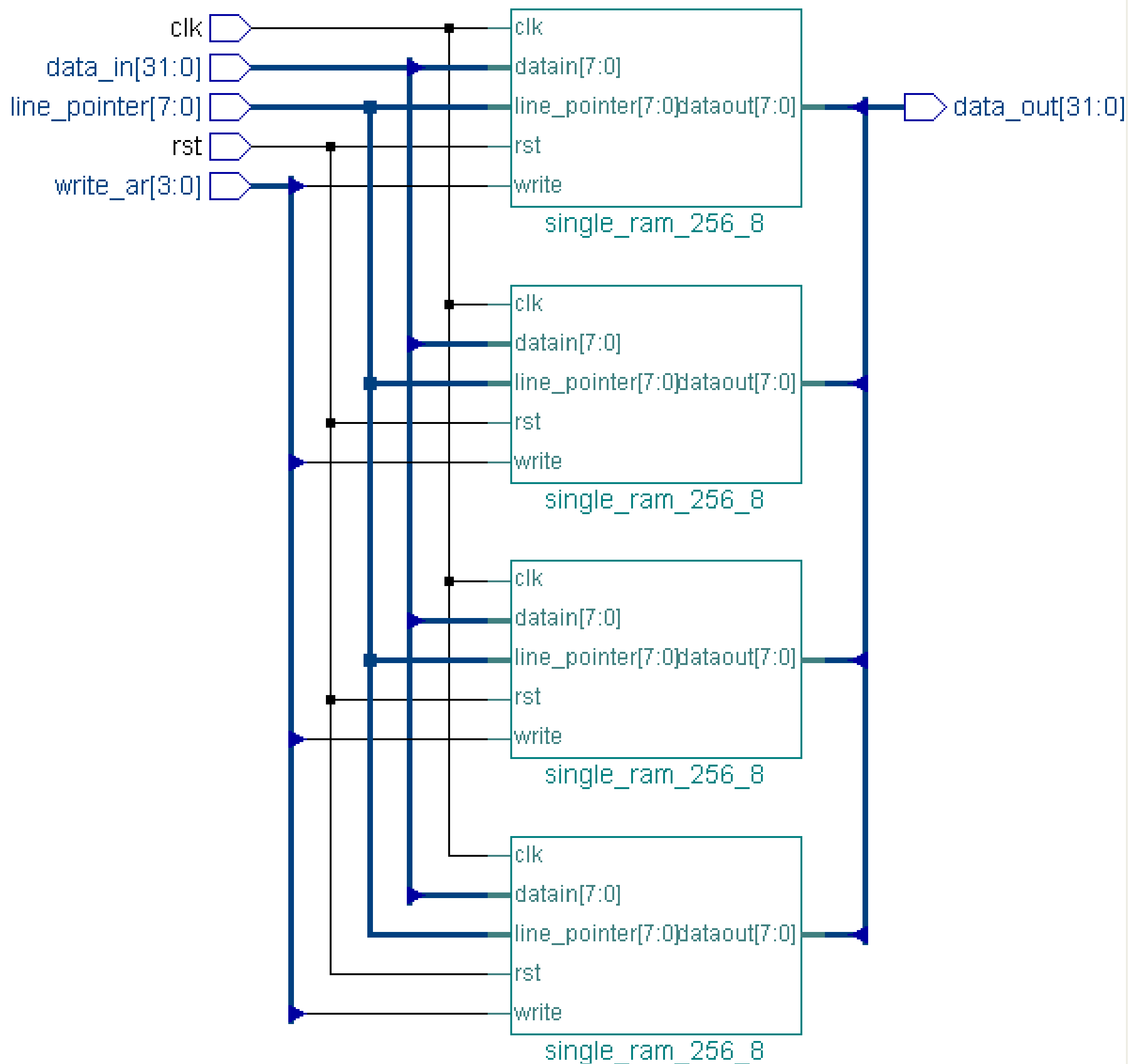
Η οντότητα αυτή έχει μία μόνο έξοδο:

- **data_out** : Προς ahb_slv_cache. Είναι τα δεδομένα που διαβάζονται από την κεντρική μνήμη. Αποτελούν το σύνολο των δεδομένων που διαβάζονται από καθεμία από τις μνήμες single_ram που περιγράφονται παρακάτω.

Στο αρχείο cache_ram.vhd ορίζεται μια ξεχωριστή οντότητα, η οποία ονομάζεται single_ram. Αποτελεί μια απλή μνήμη ram με ασύγχρονο reset. Η ιδιαιτερότητα της μνήμης ram αυτής είναι ότι έχει μέγεθος λέξης το ένα τέταρτο από αυτό που ορίζει ο χρήστης. Στη συνέχεια, μέσα στο σώμα της κυρίως οντότητας cache_ram χρησιμοποιείται η εντολή generate για να δημιουργήσει τέσσερα αντίγραφα της single_ram, για συνολικό μήκος λέξης μνήμης αυτό που όρισε ο χρήστης.

Αυτό γίνεται για να είναι ικανός ο χρήστης να γράψει οποιαδήποτε από τις τέσσερις υπο-λέξεις μια λέξης της κρυφής μνήμης χωριστά. Αυτό γίνεται για εξοικονόμηση ενέργειας κατά την εγγραφή στην κρυφή μνήμη. Αν χρειάζεται δηλαδή να εγγραφεί μόνο ένα byte από μια λέξη των 32 bits, δεν υπάρχει λόγος να πραγματοποιηθεί η εγγραφή και στα 32 bits. Η επιλογή γίνεται - όπως περιγράφηκε προηγουμένως - στο αρχείο ahb_slv_cache, βάσει του μεγέθους της προσπέλασης (δηλώνεται από τις γραμμές HSIZE του διαύλου AHB) και των δύο λιγότερο σημαντικών bits της διεύθυνσης (χρησιμοποιούνται για να επιλέξουν ποια από τις τέσσερις υπο-λέξεις θα γραφτεί, ή ανά δύο, οι δύο λιγότερο σημαντικές υπο-λέξεις ή οι δύο περισσότερο σημαντικές υπο-λέξεις). Κάθε ένα από τα τέσσερα bits εγγραφής της εισόδου write αποτελεί το bit εγγραφής μια μνήμης single_ram. Τα δεδομένα εγγραφής για καθεμιά από τις single_ram είναι το ένα τέταρτο των συνολικών δεδομένων εγγραφής.

Το εργαλείο σύνθεσης υλοποιεί απλώς τέσσερις μνήμες ram με ασύγχρονο reset.



Εικόνα 39: Οι τέσσερις μνήμες ram, όπως παράγονται από το εργαλείο σύνθεσης

5.6.fifo.vhd

Όπως περιγράφηκε εκτενέστερα στο Κεφάλαιο 3, η λογική του αρχείου αυτού είναι υπεύθυνη για δύο λειτουργίες. Η πρώτη λειτουργία είναι ο συγχρονισμός της επικοινωνίας μεταξύ του διαύλου του επεξεργαστή και του διαύλου της μνήμης (οι οποίοι είναι υποχρεωτικό να λειτουργούν με ελαφρώς ή πολύ διαφορετικά ρολόγια σε φάση και συχνότητα). Η δεύτερη λειτουργία είναι η λειτουργία της ως buffer για την προσωρινή αποθήκευση των αιτήσεων που απευθύνονται στην κύρια μνήμη, πράγμα που διευκολύνει την ομαλή εκτέλεση των διαφορετικών νημάτων του επεξεργαστή, εξασφαλίζοντας πολύ λιγότερους κύκλους αναμονής έως ότου πραγματοποιηθεί η επιθυμητή μεταφορά. Και οι δύο παραπάνω λειτουργίες (και το υλικό που τις υλοποιεί) περιγράφηκαν εκτενώς στο Κεφάλαιο 3. Εδώ παρουσιάζεται μια πιο κοντινή ματιά στον κώδικα που περιγράφει το παραπάνω υλικό.

Οι είσοδοι της οντότητας αυτής είναι:

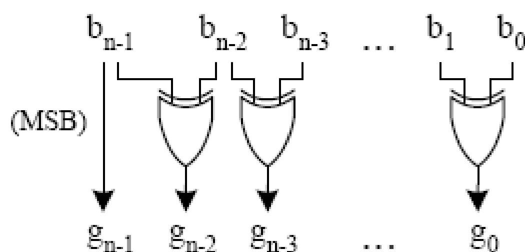
- **D** : Από ahb_slv_cache. Είναι τα δεδομένα που περιγράφουν την αίτηση προσπέλασης της κύριας μνήμης. 1 bit δηλώνει αν η αίτηση αφορά εγγραφή ή μεταφορά μπλοκ, 2 bits δηλώνουν το μέγεθος της μεταφοράς σε περίπτωση εγγραφής (όπως περιγράφεται από το σήμα HSIZE του διαύλου AHB του επεξεργαστή), ενώ τα υπόλοιπα bits είναι η προς προσπέλαση διεύθυνση.
- **write_data_in** : Από ahb_slv_cache. Σε περίπτωση αίτησης εγγραφής, στην είσοδο αυτή οδηγούνται τα προς εγγραφή δεδομένα. Σε περίπτωση αίτησης ανάγνωσης, η είσοδος αυτή δεν ανανεώνεται και δε γράφονται δεδομένα στην αντίστοιχη μνήμη.
- **RES** : Από τον δίαυλο του επεξεργαστή. Είναι το σήμα reset. Το σήμα αυτό μπορεί να ληφθεί και από τον δίαυλο της μνήμης.
- **WRCLK** : Από τον δίαυλο του επεξεργαστή. Είναι το ρολόι του διαύλου του επεξεργαστή. Το ρολόι αυτό λειτουργεί ως ρολόι εγγραφής, δηλαδή όλες οι εγγραφές στη fifo πραγματοποιούνται σύγχρονα με αυτό.
- **RDCLK** : Από τον δίαυλο της κύριας μνήμης. Είναι το ρολόι του διαύλου της κύριας μνήμης. Το ρολόι αυτό λειτουργεί ως ρολόι ανάγνωσης, δηλαδή όλες οι αναγνώσεις από τη fifo πραγματοποιούνται σύγχρονα με αυτό.
- **WREN** : Από ahb_slv_cache. Είναι το σήμα ενεργοποίησης εγγραφής. Στη θετική ακμή του ρολογιού εγγραφής, αν το σήμα αυτό είναι 1, η fifo πρέπει να πραγματοποιήσει μια εγγραφή.
- **RDEN** : Από ahb_mst_cache. Είναι το σήμα ενεργοποίησης ανάγνωσης. Στη θετική ακμή του ρολογιού ανάγνωσης, αν το σήμα αυτό είναι 1, η fifo πρέπει να πραγματοποιήσει μια ανάγνωση και να οδηγήσει στην έξοδο τα νέα δεδομένα.

Οι έξοδοι της οντότητας αυτής είναι:

- **Q** : Προς ahb_mst_cache. Είναι τα δεδομένα που περιγράφουν την αίτηση προσπέλασης της κύριας μνήμης. Σε κάθε ανάγνωση, η γραμμή αυτή οδηγείται και η διεπαφή master της κρυφής μνήμης με τον δίαυλο της κύριας μνήμης αναλαμβάνει να πραγματοποιήσει τη ζητούμενη μεταφορά.
- **write_data_out** : Προς ahb_mst_cache. Είναι τα δεδομένα που πρέπει να γραφτούν στην κύρια μνήμη σε μια αίτηση εγγραφής. Σε κάθε ανάγνωση από τη fifo, η γραμμή αυτή οδηγείται. Αν η αίτηση, ωστόσο, που διαβάζεται παράλληλα είναι αίτηση ανάγνωσης, τα δεδομένα αυτά απλά αγνοούνται.
- **F** : Προς ahb_slv_cache. Δηλώνει αν η fifo είναι γεμάτη, ώστε να αποτραπούν περαιτέρω εγγραφές. Η έξοδος αυτή είναι συγχρονισμένη προς το ρολόι εγγραφής (αφού αφορά μόνο το πεδίο ρολογιού εγγραφής). Είναι δηλωμένη ως buffer, που σημαίνει ότι μπορεί επίσης να διαβαστεί εσωτερικά, από τη λογική της fifo.
- **NF** : Προς ahb_slv_cache. Δηλώνει αν η fifo έχει μόνο μία θέση κενή, για λόγους που περιγράφηκαν στην ανάλυση της λογικής του αρχείου ahb_slv_cache.vhd. Η έξοδος αυτή είναι συγχρονισμένη προς το ρολόι εγγραφής (αφού αφορά μόνο το πεδίο ρολογιού εγγραφής). Είναι δηλωμένη ως buffer, που σημαίνει ότι μπορεί επίσης να διαβαστεί εσωτερικά, από τη λογική της fifo.
- **E** : Προς ahb_mst_cache. Δηλώνει αν η fifo είναι άδεια, ώστε να αποτραπούν περαιτέρω αναγνώσεις. Η έξοδος αυτή είναι συγχρονισμένη προς το ρολόι ανάγνωσης (αφού αφορά μόνο το πεδίο ρολογιού ανάγνωσης). Είναι δηλωμένη ως buffer, που σημαίνει ότι μπορεί επίσης να διαβαστεί εσωτερικά, από τη λογική της fifo.

Στη λογική της fifo υλοποιούνται δύο αρχεία καταχωρητών, ίδιου μήκους. Το ένα, rammemory, χρησιμοποιείται για την αποθήκευση των δεδομένων που περιγράφουν μια αίτηση προς την κύρια μνήμη. Το δεύτερο, ramwritedata, χρησιμοποιείται για την αποθήκευση των δεδομένων προς εγγραφή σε περίπτωση αποθήκευσης αίτησης εγγραφής. Και τα δύο αρχεία

δεικτοδοτούνται από δύο δείκτες, τον δείκτη εγγραφής και τον δείκτη ανάγνωσης, και γράφονται σύγχρονα με το ρολόι εγγραφής (με τη διαφορά ότι στο ramwritedata, δεν πραγματοποιείται κάποια εγγραφή σε περίπτωση αποθήκευσης αίτησης ανάγνωσης, απλά αυξάνεται ο δείκτης) και διαβάζονται σύγχρονα με το ρολόι ανάγνωσης.
Ο κώδικας της fifo είναι διαρθρωμένος σε τέσσερις διεργασίες. Η πρώτη από αυτές είναι η διεργασία Read_Process. Σε αυτή υλοποιείται ο καταχωρητής r_bin, ο οποίος περιέχει τη δυαδική τιμή του δείκτη ανάγνωσης. Το περιεχόμενο του καταχωρητή αυτού αυξάνεται με κάθε ανάγνωση, ώστε ο δείκτης να δείχνει την επόμενη θέση. Σε έναν δεύτερο καταχωρητή r_gray αποθηκεύεται η αντίστοιχη τιμή του δείκτη σε κώδικα Gray. Η προσαυξημένη είσοδος του καταχωρητή r_bin περνάει μέσα από μια λογική μετατροπής δυαδικού αριθμού σε κώδικα Gray και το αποτέλεσμα είναι η τιμή του κώδικα Gray που θα αποθηκευθεί στον καταχωρητή r_gray με την επόμενη ανάγνωση. Η αναπαράσταση σε κώδικα Gray ενός δυαδικού αριθμού προκύπτει αν σε κάθε bit του κώδικα Gray αποθηκεύσουμε την πράξη xor του αντίστοιχου δυαδικού bit με το αμέσως σημαντικότερο bit, ενώ το πιο σημαντικό bit και των δύο αναπαραστάσεων είναι το ίδιο (Εικόνα 40). Έτσι, οι δύο καταχωρητές αυξάνονται ταυτόχρονα. Η λογική που μόλις περιγράφηκε είναι αυτή της Εικόνας 25 του Κεφαλαίου 3, δηλαδή ένας μετρητής κώδικα Gray. Η δυαδική τιμή του δείκτη χρησιμοποιείται για δεικτοδότηση ανάγνωσης των αρχείων καταχωρητών, ενώ η αναπαράσταση σε κώδικα Gray χρησιμοποιείται για τη σύγκριση των δεικτών. Στην ίδια διεργασία περιγράφεται επίσης και η λειτουργία ανάγνωσης των αρχείων καταχωρητών.



Εικόνα 40: Λογική μετατροπής δυαδικού αριθμού σε αναπαράσταση Gray

Η διεργασία Write_Process περιγράφει αντίστοιχη λογική. Υλοποιεί δηλαδή έναν καταχωρητή w_bin, στον οποίο αποθηκεύεται η δυαδική τιμή του δείκτη εγγραφής, και έναν καταχωρητή w_gray, στον οποίο αποθηκεύεται η αναπαράσταση σε κώδικα Gray του δείκτη εγγραφής. Οι δύο καταχωρητές διασυνδέονται αντίστοιχα με τους καταχωρητές r_bin και r_gray που περιγράφηκαν παραπάνω, ενώ αυξάνονται με κάθε εγγραφή στη fifo. Η διαφορά στη λογική που περιγράφει η διεργασία αυτή είναι η ύπαρξη ενός τρίτου καταχωρητή, του w_gray_nf, στον οποίο αποθηκεύεται η αναπαράσταση σε κώδικα Gray της προηγούμενης τιμής του δείκτη ανάγνωσης. Ο καταχωρητής αυτός χρησιμοποιείται για σύγκριση με τον δείκτη εγγραφής, προκειμένου να αποφασιστεί αν η fifo έχει μόνο μία θέση κενή και να τεθεί η έξοδος NF. Στην είσοδο του καταχωρητή αυτού οδηγείται φυσικά το περιεχόμενο του καταχωρητή w_bin προσαυξημένο κατά 2 (ώστε να βρίσκεται πάντα μια θέση μπροστά από τον δείκτη εγγραφής) και διερχόμενο μέσα από τη λογική μετατροπής δυαδικού αριθμού σε αναπαράσταση Gray. Στην διεργασία αυτή περιγράφεται επίσης και η λειτουργία εγγραφής των αρχείων καταχωρητών, καθώς και η ασύγχρονη αρχικοποίησή τους με το σήμα reset.

Στη διεργασία Imd_E_F_Process περιγράφεται το τμήμα της Εικόνας 32 που περιβάλλεται από διακεκομμένες γραμμές. Πρόκειται για το κύκλωμα παραγωγής των ενδιάμεσων ασύγχρονων σημάτων άδειας και γεμάτης κατάστασης, και αποτελείται από το κύκλωμα ανίχνευσης κατεύθυνσης της Εικόνας 28 και τη λογική σύγκρισης των δεικτών. Στις πρώτες γραμμές της

διεργασίας (έως και την πρώτη δήλωση if) περιγράφεται το κύκλωμα ανίχνευσης κατεύθυνσης, όπως αυτό παρουσιάστηκε στο Κεφάλαιο 3. Ακολούθως πραγματοποιείται η σύγκριση των αναπαραστάσεων Gray των δεικτών εγγραφής και ανάγνωσης και σε περίπτωση ισότητας, ελέγχεται το bit κατεύθυνσης και παράγεται το κατάλληλο ασύγχρονο σήμα άδειας ή γεμάτης κατάστασης. Στην τελευταία δήλωση if της διεργασίας, πραγματοποιείται σύγκριση της αναπαράστασης Gray του δείκτη ανάγνωσης με την αναπαράσταση σε κώδικα Gray του αυξημένου κατά 1 δείκτη εγγραφής, και σε περίπτωση ισότητας, αποφασίζεται ότι η fifo έχει μόνο μία θέση κενή και τίθεται το κατάλληλο ασύγχρονο σήμα.

Στην τελευταία διεργασία E_F_Update υλοποιούνται οι συγχρονιστές που παρουσιάστηκαν στο Κεφάλαιο 3. Υλοποιούνται 3 συνολικά συγχρονιστές (καθένας από τους οποίους αποτελείται από δύο σε σειρά flip-flops), δύο για τα σήματα F και NF (συγχρονισμένα προς το ρολόι εγγραφής), και ένας για το σήμα E (συγχρονισμένο προς το ρολόι ανάγνωσης).

5.7.ahb_mst_cache.vhd

Η λογική του αρχείου αυτού υλοποιεί τη διεπαφή της κρυφής μνήμης δεύτερου επιπέδου με τον δίαυλο AMBA AHB της μνήμης (M-bus). Στον δίαυλο αυτό, η κρυφή μνήμη οφείλει να λειτουργεί ως συσκευή master. Η διεπαφή αυτή είναι λοιπόν υπεύθυνη για τη συμμετοχή στη διαδικασία διαιτησίας (arbitration) του διαύλου και την εξαγωγή και εξυπηρέτηση αιτήσεων από τη fifo. Όποτε λοιπόν η fifo δεν είναι άδεια, η διεπαφή αυτή πρέπει να διαβάζει μια αίτηση, να ζητάει τον έλεγχο του διαύλου, και αν η αίτηση είναι εγγραφή, να εκκινεί μια μεταφορά εγγραφής στην κύρια μνήμη. Αν η αίτηση είναι ανάγνωση, πρέπει να ξεκινάει μια σειρά μεταφορών αναγνώσεων από την κύρια μνήμη, μία για κάθε γραμμή του μπλοκ που ζητείται, και να αποθηκεύει κάθε γραμμή που διαβάζεται στον buffer της κρυφής μνήμης.

Οι είσοδοι της οντότητας αυτής είναι:

- **rst** : Από τον δίαυλο της κύριας μνήμης. Είναι το σήμα reset. Το σήμα αυτό μπορεί να προέρχεται και από τον δίαυλο του επεξεργαστή.
- **clk** : Από τον δίαυλο της κύριας μνήμης. Είναι το ρολόι του διαύλου της μνήμης.
- **ahbi** : Από τον δίαυλο της κύριας μνήμης. Είναι το σύνολο των πληροφοριών που δέχεται η κρυφή μνήμη από τον δίαυλο της κύριας μνήμης, όπως πληροφορίες για το arbitration, δεδομένα ανάγνωσης από την κύρια μνήμη κ.ά.
- **fifo_empty** : Από fifo. Δηλώνει ότι η fifo είναι άδεια και συνεπώς δεν μπορούν να γίνουν άλλες αναγνώσεις.
- **fifo_data** : Από fifo. Είναι τα δεδομένα που περιγράφουν την αίτηση προσπέλασης στην κύρια μνήμη. Ένα bit δηλώνει αν η αίτηση αφορά εγγραφή γραμμής ή μεταφορά μπλοκ, δύο δηλώνουν το μέγεθος της μεταφοράς (για εγγραφή) και τα υπόλοιπα είναι η διεύθυνση της γραμμής που επιχείρησε να προσπελάσει ο επεξεργαστής.
- **fifo_write_data** : Από fifo. Είναι τα δεδομένα που πρέπει να εγγραφούν στην κύρια μνήμη αν η αίτηση μεταφοράς που διαβάστηκε είναι αίτηση εγγραφής. Αν η αίτηση είναι αίτηση μεταφοράς μπλοκ από την κύρια μνήμη, τα δεδομένα της εισόδου αυτής απλά αγνοούνται.
- **buffer_empty** : Από buffer. Δηλώνει ότι ο buffer έχει αδειάσει και συνεπώς μπορεί να αποθηκευθεί σε αυτόν ένα νέο μπλοκ.

Οι έξοδοι της οντότητας αυτής είναι:

- **ahbo** : Προς τον δίαυλο της κύριας μνήμης. Είναι το σύνολο των πληροφοριών που εξάγει η κρυφή μνήμη στον δίαυλο της κύριας μνήμης, όπως αιτήσεις απόκτησης ελέγχου του

διαύλου, οι διευθύνσεις για προσπέλαση της κύριας μνήμης, τα προς εγγραφή δεδομένα (σε περίπτωση μεταφοράς εγγραφής) κ.ά.

- **fifo_read_en** : Προς fifo. Είναι η γραμμή ενεργοποίησης ανάγνωσης. Αν στη θετική ακμή του ρολογιού η γραμμή αυτή οδηγείται στο 1, η fifo θα πρέπει να πραγματοποιήσει μια ανάγνωση.
- **buffer_write_en** : Προς buffer. Είναι η γραμμή ενεργοποίησης εγγραφής. Αν στη θετική ακμή του ρολογιού η γραμμή αυτή οδηγείται στο 1, ο buffer θα πρέπει να πραγματοποιήσει μια εγγραφή.
- **buffer_data_in** : Προς buffer. Είναι η γραμμή που θα αποθηκευθεί στον buffer στην επόμενη εγγραφή (ουσιαστικά η τελευταία γραμμή που διαβάστηκε από την κύρια μνήμη).
- **tag_addr_in** : Προς transfer_logic. Είναι η διεύθυνση της μεταφοράς από την κύρια μνήμη. Στην αρχή κάθε μεταφοράς ενός μπλοκ προωθείται στη λογική transfer_logic, ώστε την κατάλληλη στιγμή εκείνη να εξάγει και να προωθήσει την ετικέτα (tag) του μπλοκ στη λογική addr_check (όπου αποθηκεύονται οι ετικέτες).
- **tag_addr_write_en** : Προς transfer_logic. Είναι η γραμμή ενεργοποίησης αποθήκευσης της διεύθυνσης της μεταφοράς από την κύρια μνήμη. Στη θετική ακμή του ρολογιού της μνήμης, αν η γραμμή αυτή είναι 1, η λογική transfer_logic αποθηκεύει τη διεύθυνση. Η αποθήκευση αυτή γίνεται στο ξεκίνημα μιας μεταφοράς ενός μπλοκ από την κύρια μνήμη.

Η δομή της λογικής του τμήματος αυτού είναι παρόμοια με αυτή του τμήματος ahb_slv_cache, δηλαδή της διεπαφής slave της κρυφής μνήμης. Όπως και στο αρχείο εκείνο, έτσι και εδώ, ο κώδικας αποτελείται από δύο διεργασίες. Η μία διεργασία, που ονομάζεται comb, περιγράφει τη συνδυαστική λογική του κυκλώματος. Βασίζεται δηλαδή στις εξωτερικές εισόδους της λογικής και σε ορισμένες αποθηκευμένες από τον προηγούμενο κύκλο, προκειμένου να οδηγήσει κατάλληλα τις εξόδους της και να υπολογίσει τις νέες τιμές που πρέπει να αποθηκευθούν στους καταχωρητές με τον επόμενο θετικό παλμό. Είναι το τμήμα που λαμβάνει τις «αποφάσεις» για το τι θα πρέπει να κάνει το τμήμα αυτό σε κάθε κύκλο ρολογιού. Οι μεταβολές πραγματοποιούνται συνήθως σε μεταβλητές, οι οποίες στο τέλος κάθε εκτέλεσης της διεργασίας, ανατίθενται σε ένα σύνθετο (αποτελούμενο από υπο-σήματα) σήμα rin. Το σήμα αυτό οδηγεί τις εισόδους των καταχωρητών της λογικής. Η δεύτερη διεργασία, η οποία ονομάζεται regs, περιγράφει το ακολουθιακό τμήμα της λογικής. Η λειτουργία που περιγράφει είναι πολύ απλή, ουσιαστικά σε κάθε παλμό ρολογιού αποθηκεύει τις εισόδους των καταχωρητών (το σήμα rin) μέσα στους καταχωρητές (έχουν δηλωθεί ως σύνθετο σήμα r). Το εργαλείο σύνθεσης υλοποιεί δηλαδή όλα τα υπο-σήματα του σήματος r ως καταχωρητές.

Όπως και στο αρχείο addr_check.vhd, έτσι κι εδώ, χρησιμοποιείται η εντολή generate της VHDL, προκειμένου να περιγραφούν δύο διαφορετικές εκδόσεις της διεργασίας comb. Αυτό γίνεται γιατί η διεργασία χρησιμοποιεί τα bits μετατόπισης (offset) της διεύθυνσης, τα οποία όμως μπορεί να μην ορίζονται, αν κάθε μπλοκ αποτελείται από μία μόνο γραμμή. Η μία έκδοση λοιπόν περιλαμβάνει τη χρήση των bits αυτών, ενώ η άλλη θεωρεί ότι δεν ορίζονται. Έτσι, η εντολή generate χρησιμοποιεί τις επιλογές του χρήστη, προκειμένου να αποφασίσει ποια από τις εκδόσεις της διεργασίας θα υλοποιηθεί. Αντίστοιχα, το εργαλείο σύνθεσης υλοποιεί τη συνδυαστική λογική του τμήματος αυτού χρησιμοποιώντας πάντα τη μία από τις δύο εκδόσεις.

Η δομή του κώδικα έχει ως εξής. Στην πρώτη δήλωση if της διεργασίας comb αποφασίζεται αν οι συνθήκες είναι κατάλληλες ώστε να εξαχθεί (pop) ένα νέο καθήκον από τη fifo. Αυτό γίνεται όταν ο buffer είναι κενός, η fifo δεν είναι κενή και η εξυπηρέτηση της προηγούμενης αίτησης έχει ολοκληρωθεί. Αν όντως οι συνθήκες είναι κατάλληλες, τίθεται το σήμα ενεργοποίησης ανάγνωσης από τη fifo και η διεπαφή master αρχίζει να ζητά τον έλεγχο του διαύλου. Η λογική έχει περιγραφεί έτσι ώστε η κρυφή μνήμη να μην κλειδώνει τον δίαυλο για τις μεταφορές της, αλλά να ζητά τον δίαυλο για κάθε μεταφορά (είτε εγγραφή είτε ανάγνωση) χωριστά.

Στη δεύτερη δήλωση if πραγματοποιούνται τα παρακάτω. Αν στον προηγούμενο κύκλο αναγνώστηκε ένα νέο καθήκον από τη fifo και αυτό αφορούσε εγγραφή, τότε εισάγεται η διεύθυνση στον δίαυλο. Τα δεδομένα αποθηκεύονται σε έναν καταχωρητή και εισάγονται στον δίαυλο έναν κύκλο μετά τη διεύθυνση, όπως ορίζει το πρωτόκολλο AHB. Αν το καθήκον ήταν μεταφορά μπλοκ από την κύρια μνήμη, τότε στον δίαυλο εισάγεται η διεύθυνση της πρώτης γραμμής του μπλοκ (μηδενίζοντας τα bits μετατόπιση, προφανώς μόνο στην έκδοση της διεργασίας που αυτά ορίζονται) και αρχικοποιούνται δύο μετρητές, ο μετρητής διεύθυνσης και ο μετρητής ανάγνωσης. Ο μετρητής διεύθυνσης μετράει πόσες διευθύνσεις έχουν οδηγηθεί στον δίαυλο και έχουν διαβαστεί από τη συσκευή slave (την κύρια μνήμη), έτσι ώστε να σταματήσει η διεπαφή να ζητά τον έλεγχο του διαύλου μόλις διαβαστεί και η τελευταία διεύθυνση. Ο μετρητής ανάγνωσης μετράει πόσες σωστές απαντήσεις έχουν ληφθεί (δηλαδή πόσες γραμμές έχουν αναγνωσθεί από την κύρια μνήμη), έτσι ώστε να γνωρίζει η διεπαφή ότι η εξυπηρέτηση της αίτησης ολοκληρώθηκε μόλις διαβαστεί και η τελευταία γραμμή του μπλοκ. Αν στον προηγούμενο κύκλο δεν αναγνώσθηκε νέο καθήκον από τη fifo, τότε απλώς ανανεώνεται η διεύθυνση που οδηγείται στον δίαυλο με μια νέα τιμή, η οποία μπορεί να έχει προσαυξηθεί (αν πραγματοποιείται ανάγνωση και η προηγούμενη διεύθυνση διαβάστηκε επιτυχώς από την κύρια μνήμη) ή όχι.

Η τρίτη δήλωση if ελέγχει αν αναγνώστηκε μια γραμμή από την κρυφή μνήμη και αν ναι, θέτει το σήμα ενεργοποίησης αποθήκευσής της στον buffer και μειώνει τον καταχωρητή ανάγνωσης.

Η τέταρτη δήλωση if ελέγχει αν η κρυφή μνήμη έχει αποκτήσει τον έλεγχο του διαύλου και αν θα πρέπει να οδηγήσει μια νέα διεύθυνση.

Η πέμπτη δήλωση if ελέγχει αν η διεύθυνση που οδηγείται στον δίαυλο αναγνώστηκε επιτυχώς από την κύρια μνήμη και αν ναι, υπολογίζει τη νέα διεύθυνση που πρέπει να οδηγηθεί (σε περίπτωση ανάγνωσης) και μειώνει τον μετρητή διεύθυνσης.

Η έκτη δήλωση if είναι υπεύθυνη για την πραγματοποίηση μεταφορών IDLE σε περίπτωση που η διεπαφή έχει πάρει τον έλεγχο του διαύλου χωρίς να τον έχει ζητήσει. Με τον τρόπο αυτό, κάθε κρυφή μνήμη δεύτερου επιπέδου μπορεί να επιτελέσει τη λειτουργία του default master που περιγράφηκε στο Κεφάλαιο 2, μπορεί δηλαδή να είναι η συσκευή master στην οποία παραδίδεται ο έλεγχος του διαύλου όταν κανείς δεν τον ζητά.

Η έβδομη δήλωση if ελέγχει αν έχει διαβαστεί η τελευταία διεύθυνση μιας μεταφοράς ενός μπλοκ από την κύρια μνήμη και αν ναι, σταματά να ζητά τον έλεγχο του διαύλου.

Η τελευταία δήλωση if πραγματοποιεί απλώς την αρχικοποίηση της διεπαφής σε περίπτωση σήματος reset.

5.8.buffer.vhd

Η λειτουργία της λογικής του αρχείου αυτού είναι ουσιαστικά διπλή. Από τη μία πλευρά υλοποιεί την ενδιάμεση μνήμη αποθήκευσης (buffer) των μπλοκ που μεταφέρονται από την κύρια μνήμη, πριν αυτά εγγραφούν κανονικά στην κεντρική μνήμη της κρυφής μνήμης. Από την άλλη, πραγματοποιεί εμμέσως τον συγχρονισμό ανάμεσα στο ρολόι του διαύλου της κύριας μνήμης, με το οποίο γίνονται προφανώς οι αναγνώσεις από αυτή, και στο ρολόι του επεξεργαστή, με το οποίο λειτουργεί η κεντρική μνήμη της κρυφής μνήμης. Είναι δηλαδή υπεύθυνος για τον συγχρονισμό της αντίστροφης κατεύθυνσης επικοινωνίας μεταξύ των δύο πεδίων ρολογιού, σε σχέση με τη fifo. Για τον λόγο αυτό, η λογική του buffer είναι παρόμοια με αυτή της fifo.

Οι είσοδοι της οντότητας αυτής είναι:

- **D** : Από ahb_mst_cache. Είναι τα δεδομένα που διαβάζονται από την κύρια μνήμη, δηλαδή μια γραμμή που αναγνώσθηκε από την κύρια μνήμη.
- **RES** : Από τον δίαυλο της κύριας μνήμης. Είναι το σήμα reset. Το σήμα αυτό μπορεί να προέρχεται και από τον δίαυλο του επεξεργαστή.
- **WRCLK** : Από τον δίαυλο της κύριας μνήμης. Είναι το ρολόι του διαύλου της κύριας μνήμης. Το ρολόι αυτό λειτουργεί ως ρολόι εγγραφής, δηλαδή όλες οι εγγραφές στον buffer πραγματοποιούνται σύγχρονα με αυτό.
- **RDCLK** : Από τον δίαυλο του επεξεργαστή. Είναι το ρολόι του διαύλου του επεξεργαστή. Το ρολόι αυτό λειτουργεί ως ρολόι ανάγνωσης, δηλαδή όλες οι αναγνώσεις από τον buffer πραγματοποιούνται σύγχρονα με αυτό.
- **WREN** : Από ahb_mst_cache. Είναι το σήμα ενεργοποίησης εγγραφής. Στη θετική ακμή του ρολογιού εγγραφής, αν το σήμα αυτό είναι 1, ο buffer πρέπει να πραγματοποιήσει μια εγγραφή.
- **RDEN** : Από transfer_logic. Είναι το σήμα ενεργοποίησης ανάγνωσης. Στη θετική ακμή του ρολογιού ανάγνωσης, αν το σήμα αυτό είναι 1, ο buffer πρέπει να πραγματοποιήσει μια ανάγνωση και να οδηγήσει στην έξοδο τα νέα δεδομένα.

Οι έξοδοι της οντότητας αυτής είναι:

- **Q** : Προς transfer_logic. Είναι τα δεδομένα που διαβάζονται από τον buffer, δηλαδή μία γραμμή του buffer. Η λογική transfer_logic αναλαμβάνει στη συνέχεια να τα εγγράψει στην κατάλληλη θέση της κεντρικής μνήμης της κρυφής μνήμης.
- **trl_F** : Προς transfer_logic. Δηλώνει αν ο buffer είναι γεμάτος, ώστε να μπορέσει να ξεκινήσει η διαδικασία μεταφοράς του μπλοκ στην κεντρική μνήμη. Η έξοδος αυτή είναι συγχρονισμένη προς το ρολόι ανάγνωσης (αφού αφορά μόνο το πεδίο ρολογιού ανάγνωσης). Είναι δηλωμένη ως buffer, που σημαίνει ότι μπορεί επίσης να διαβαστεί εσωτερικά, από τη λογική του buffer**.**
- **trl_E** : Προς transfer_logic. Δηλώνει αν ο buffer είναι άδειος, ώστε να σταματήσει η διαδικασία μεταφοράς του μπλοκ στην κεντρική μνήμη. Η έξοδος αυτή είναι συγχρονισμένη προς το ρολόι ανάγνωσης (αφού αφορά μόνο το πεδίο ρολογιού ανάγνωσης). Είναι δηλωμένη ως buffer, που σημαίνει ότι μπορεί επίσης να διαβαστεί εσωτερικά, από τη λογική του buffer**.**
- **mst_E** : Προς ahb_mst_cache. Δηλώνει αν ο buffer είναι άδειος, ώστε να μπορέσει να ξεκινήσει μια νέα διαδικασία μεταφοράς από την κύρια μνήμη. Η έξοδος αυτή είναι συγχρονισμένη προς το ρολόι εγγραφής (αφού αφορά μόνο το πεδίο ρολογιού εγγραφής). Είναι δηλωμένη ως buffer, που σημαίνει ότι μπορεί επίσης να διαβαστεί εσωτερικά, από τη λογική του buffer**.**

Όπως αναφέρθηκε παραπάνω, η λογική του buffer είναι πολύ παρόμοια με τη λογική της fifo. Μια βασική διαφορά στη λειτουργία τους είναι ότι ο buffer πρώτα γράφεται ολόκληρος γραμμή-γραμμή χρησιμοποιώντας το ένα ρολόι, και μετά αδειάζει πάλι ολόκληρος (όταν το μπλοκ αντιγράφεται, πάλι γραμμή-γραμμή, στην κεντρική μνήμη) χρησιμοποιώντας το άλλο ρολόι. Αυτό ωστόσο δεν επηρεάζει τη λογική του buffer, γιατί θα πρέπει και πάλι να πραγματοποιείται η ανίχνευση άδειας-γεμάτης κατάστασης και ο συγχρονισμός των εξωτερικών σημάτων που σηματοδοτούν τις καταστάσεις αυτές.

Η λογική του buffer αποτελείται από τέσσερις διεργασίες, οι οποίες είναι αντίστοιχες των διεργασιών της fifo. Η πρώτη, Read_Process, υλοποιεί τους καταχωρητές του δείκτη ανάγνωσης και της αναπαράστασής του σε κώδικα Gray, τη λογική προσαύξησης του δείκτη, τη λογική μετατροπής από δυαδική αναπαράσταση σε κώδικα Gray, καθώς και τη λογική ανάγνωσης από τον buffer.

Η δεύτερη, Write_Process, υλοποιεί τους αντίστοιχους καταχωρητές και την περιφερειακή λογική τους, καθώς και τη λογική εγγραφής και αρχικοποίησης του buffer.

Η τρίτη διεργασία, Imd_E_F_Process, υλοποιεί τη λογική ανίχνευσης κατεύθυνσης, τη σύγκριση των δεικτών και την οδήγηση των ασύγχρονων σημάτων άδειας και γεμάτης κατάστασης.

Η τελευταία διεργασία, E_F_Update, υλοποιεί τους συγχρονιστές που οδηγούν τα τελικά, συγχρονισμένα σήματα άδειας και γεμάτης κατάστασης. Υλοποιούνται τρεις συνολικά συγχρονιστές, ο καθένας από τους οποίους αποτελείται από δύο flip-flop. Δύο για να οδηγούν τα σήματα άδειας και γεμάτης κατάστασης για τη λογική transfer_logic, συγχρονισμένα προς το ρολόι του διαύλου του επεξεργαστή, και ένας για να οδηγεί το σήμα άδειας κατάστασης για τη λογική ahb_mst_cache, συγχρονισμένο προς το ρολόι του διαύλου της κύριας μνήμης.

5.9.lfsr.vhd

Η λογική του αρχείου αυτού είναι υπεύθυνη για την παραγωγή μιας ψευδοτυχαίας ακολουθίας αριθμών. Η ψευδοτυχαία αυτή ακολουθία (η οποία προχωρά ένα βήμα με κάθε παλμό ρολογιού), σε συνδυασμό με τον ουσιαστικά τυχαίο κύκλο ρολογιού στον οποίο ξεκινά μια μεταφορά ενός μπλοκ από τον buffer στην κεντρική μνήμη της κρυφής μνήμης, προσφέρει τον τυχαίο ακέραιο που καθορίζει τη θέση μπλοκ μέσα σε ένα σύνολο της κρυφής μνήμης, στην οποία θα τοποθετηθεί το καινούριο μπλοκ. Κάθε φορά δηλαδή που μεταφέρεται ένα μπλοκ από τον buffer στην κρυφή μνήμη, η λογική μεταφοράς transfer_logic διαβάζει έναν τυχαίο αριθμό από τη λογική lfsr για να καθορίσει σε ποια θέση του συνόλου θα τοποθετηθεί το μπλοκ.

Οι είσοδοι της οντότητας αυτής είναι:

- **clk** : Από τον δίαυλο του επεξεργαστή. Είναι το ρολόι του διαύλου του επεξεργαστή.
- **rst** : Από τον δίαυλο του επεξεργαστή. Είναι το σήμα reset.

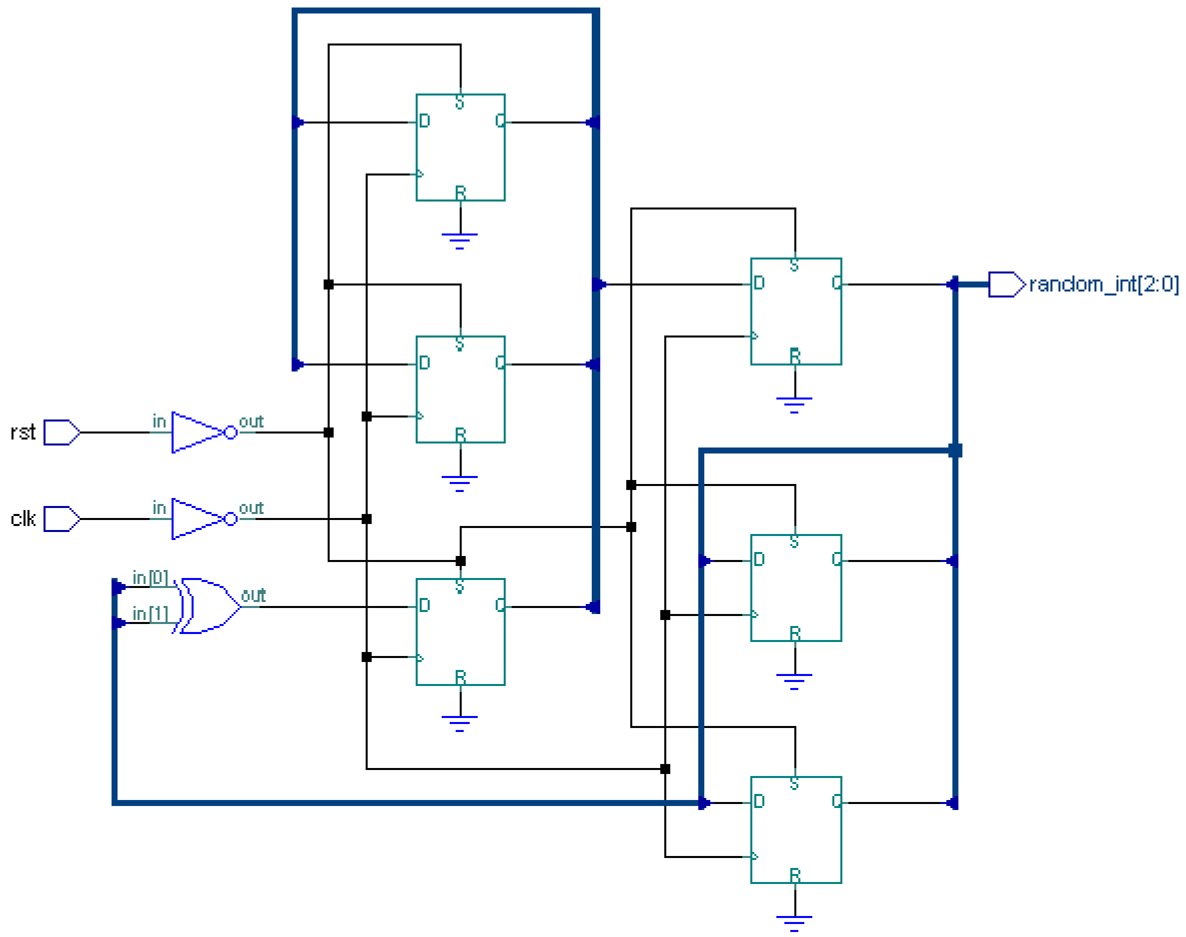
Η οντότητα αυτή έχει μία μόνο έξοδο:

- **random_int** : Προς transfer_logic. Τυχαίος ακέραιος, ο οποίος θα χρησιμοποιηθεί για να καθοριστεί σε ποια θέση μέσα στο σύνολο θα τοποθετηθεί το πρόσφατα διαβασμένο από την κύρια μνήμη μπλοκ.

Η λογική του αρχείου αυτού υλοποιεί έναν απλό LFSR (Linear Feedback Shift Register). Ένας LFSR παράγει μια ψευδοτυχαία ακολουθία ακεραίων ως εξής. Έστω ένας καταχωρητής μεγέθους n, όπου n δύναμη του 2. Για κάθε n, υπάρχει ένας συνδυασμός ορισμένων bits, τα οποία αν τροφοδοτούνται σε πύλες xor και το αποτέλεσμα ολισθαίνεται στην αριστερή θέση του καταχωρητή, τότε ο καταχωρητής θα παράγει μια ψευδοτυχαία ακολουθία ακεραίων από 1 έως 2^{n-1}.

Ένας πίνακας από constants (σταθερές τιμές) της VHDL, καθορίζει αυτές τις θέσεις bits για n από 4 έως 23. Το εργαλείο υλικού συμβουλεύεται αυτόν τον πίνακα για να υλοποιήσει την απλή λογική του καταχωρητή και των 2 έως 4 πυλών xor που απαιτούνται. Καθώς δεν παράγεται ποτέ ο αριθμός 0, αγνοούνται τα 3 πιο σημαντικά bits του καταχωρητή και χρησιμοποιούνται τα λιγότερο σημαντικά, μέσα στα οποία θα περιέχεται ο αριθμός 0 και θα εμφανίζεται με την ίδια σχεδόν συχνότητα με τους υπόλοιπους αριθμούς. Το είδος συνολοσυσχετισμού που υποστηρίζεται είναι συνεπώς από 2 έως 2^{20} (και πολύ εύκολα επεκτάσιμο παραπέρα, κάτι που θεωρήθηκε όμως περιττό για της ανάγκες της κρυφής μνήμης).

Αν φυσικά η κρυφή μνήμη είναι άμεσα αντιστοιχιζόμενη (συνολοσυσχετισμός 1), η λογική αυτή παράγει πάντα τον αριθμό 0, αφού το μπλοκ θα πρέπει να γράφεται πάντα στη μοναδική θέση που αντιστοιχεί στο εκάστοτε σύνολο. Αυτό επιτυγχάνεται και πάλι με χρήση της εντολής generate, η οποία, σε περίπτωση που ο συνολοσυσχετισμός οριστεί ως 1, αντικαθιστά την εκδοχή της διεργασίας που παράγει την ακολουθία των ψευδοτυχαίων αριθμών με μια εκδοχή που παράγει πάντα το 0.



Εικόνα 41: Η λογική ενός LFSR, όπως παράγεται από το εργαλείο σύνθεσης

5.10.transfer_logic.vhd

Η λογική του αρχείου αυτού είναι υπεύθυνη για τη μεταφορά ενός μπλοκ από τον buffer στην κεντρική μνήμη της κρυφής μνήμης. Στη λογική αυτή κρατείται επίσης αποθηκευμένη η διεύθυνση που ζητήθηκε από τον επεξεργαστή και προωθήθηκε στην κύρια μνήμη, έτσι ώστε όταν αρχίζει η μεταφορά του μπλοκ από τον buffer να μπορεί να υπολογίσει σε ποια θέση (σύνολο και μπλοκ) της κεντρικής μνήμης θα πρέπει να τοποθετηθεί και να σηματοδοτήσει και τη λογική addr_check, ώστε να ενημερώσει κατάλληλα το αρχείο των ετικετών (tags).

Οι είσοδοι της οντότητας αυτής είναι:

- **rst** : Από τον δίαυλο του επεξεργαστή. Είναι το σήμα reset.

- **clk** : Από τον δίαυλο του επεξεργαστή. Είναι το ρολόι του διαύλου του επεξεργαστή.
- **clk_mst** : Από τον δίαυλο της κύριας μνήμης. Είναι το ρολόι του διαύλου της κύριας μνήμης.
- **not_idle** : Από τον δίαυλο του επεξεργαστή. Είναι το πιο σημαντικό bit του σήματος τύπου μεταφοράς, από το οποίο μπορεί να καθοριστεί αν η μεταφορά είναι IDLE ή όχι.
- **ahbsi_hsel** : Από τον δίαυλο του επεξεργαστή. Είναι τα bits επιλογής συσκευών slave, όπως παράγονται από τον arbiter.
- **random_int_in** : Από lfsr. Είναι ένας τυχαίος ακέραιος που χρησιμοποιείται για να επιλεχθεί μια θέση μπλοκ σε ένα σύνολο, ώστε να τοποθετηθεί σε αυτή το νέο μπλοκ.
- **buffer_empty** : Από buffer. Δηλώνει αν ο buffer είναι γεμάτος, ώστε να μπορέσει να ξεκινήσει η διαδικασία μεταφοράς του μπλοκ στην κεντρική μνήμη.
- **buffer_full** : Από buffer. Δηλώνει αν ο buffer είναι άδειος, ώστε να σταματήσει η διαδικασία μεταφοράς του μπλοκ στην κεντρική μνήμη.
- **buffer_data** : Από buffer. Είναι τα δεδομένα που διαβάζονται από τον buffer, δηλαδή μία γραμμή του buffer. Η λογική transfer_logic αναλαμβάνει στη συνέχεια να τα εγγράψει στην κατάλληλη θέση της κεντρικής μνήμης της κρυφής μνήμης.
- **addr_in** : Από ahb_mst_cache. Είναι η διεύθυνση ανάγνωσης που ζητήθηκε από τον επεξεργαστή και προωθήθηκε στην κύρια μνήμη. Οδηγείται στην είσοδο της λογικής transfer_logic όταν αρχίζει η μεταφορά ενός μπλοκ από την κύρια μνήμη. Όταν αρχίζει η μεταφορά ενός μπλοκ από τον buffer στην κεντρική μνήμη, η λογική transfer_logic χρησιμοποιεί τη διεύθυνση αυτή για να υπολογίσει πού θα τοποθετηθεί το μπλοκ και να οδηγήσει την ετικέτα (tag) και τη θέση εγγραφής στη λογική addr_check, ώστε να ανανεωθεί το αρχείο των ετικετών (tags).
- **addr_write_en** : Από ahb_mst_cache. Είναι το σήμα ενεργοποίησης εγγραφής της διεύθυνσης. Στη θετική ακμή του ρολογιού του διαύλου της κύριας μνήμης, αν το σήμα αυτό είναι 1, η λογική transfer_logic πρέπει να αποθηκεύσει τη διεύθυνση.

Οι έξοδοι της λογικής αυτής είναι:

- **tag** : Προς addr_check. Είναι η ετικέτα (tag) του μπλοκ που ξεκινά να μεταφέρεται από τον buffer στην κεντρική μνήμη.
- **tag_write_en** : Προς addr_check. Είναι το σήμα ενεργοποίησης εγγραφής της ετικέτας (tag). Στη θετική ακμή του ρολογιού του επεξεργαστή, αν το σήμα αυτό είναι 1, η λογική addr_check πρέπει να ανανεώσει το αρχείο των ετικετών με τη νέα ετικέτα.
- **block_num** : Προς addr_check. Είναι η θέση στην οποία τοποθετείται το μπλοκ. Προωθείται στη λογική addr_check μαζί με την ετικέτα για να δηλώσει τη θέση στην οποία θα γραφτεί το νέο μπλοκ, και συνεπώς την ετικέτα που πρέπει να ανανεωθεί.
- **line_pointer** : Προς multiplexor. Δηλώνει τη γραμμή της κεντρικής μνήμης στην οποία πρέπει να πραγματοποιηθεί εγγραφή. Προωθείται από τον multiplexor στην κεντρική μνήμη κατά τη λειτουργία μεταφοράς μπλοκ από τον buffer στην κεντρική μνήμη.
- **write_data** : Προς multiplexor. Είναι τα δεδομένα προς εγγραφή στην κεντρική μνήμη. Προωθούνται από τον multiplexor στην κεντρική μνήμη κατά τη λειτουργία μεταφοράς μπλοκ από τον buffer στην κεντρική μνήμη.
- **write_ar** : Προς multiplexor. Είναι τέσσερα bits που δηλώνουν ποια bytes της επιλεχθείσας γραμμής θα πρέπει να εγγραφούν. Προωθούνται από τον multiplexor στην κεντρική μνήμη κατά τη λειτουργία μεταφοράς μπλοκ από τον buffer στην κεντρική μνήμη.
- **transfer_m** : Προς multiplexor και ahb_slv_cache. Δηλώνει αν η κρυφή μνήμη βρίσκεται σε κανονική κατάσταση λειτουργίας ή σε λειτουργία μεταφοράς μπλοκ από τον buffer στην κεντρική μνήμη.

- **buffer_read_en** : Προς buffer. Είναι το σήμα ενεργοποίησης ανάγνωσης από τον buffer. Στη θετική ακμή του ρολογιού του διαύλου του επεξεργαστή, αν το σήμα αυτό είναι 1, ο buffer πρέπει να οδηγήσει στην έξοδό του μια καινούρια γραμμή.

Η πρώτη διεργασία του αρχείου έχει γραφτεί σε δύο εκδόσεις, μία για πλήρως συσχετιζόμενη μνήμη, και μία για τις υπόλοιπες περιπτώσεις, επειδή χρησιμοποιεί τα index bits της διεύθυνσης (bits επιλογής συνόλου), τα οποία δεν ορίζονται στην πρώτη περίπτωση. Με χρήση της generate επιλέγεται η κατάλληλη έκδοση. Η πρώτη διεργασία λαμβάνει και αποθηκεύει τη διεύθυνση που προωθείται από τη λογική ahb_mst_cache, κάθε φορά που ξεκινά η μεταφορά ενός μπλοκ από την κύρια μνήμη στην κρυφή. Από τη διεύθυνση αυτή και με χρήση και του τυχαίου ακεραίου από τη λογική lfsr, υπολογίζει την ετικέτα (tag) που πρέπει να εγγραφεί, καθώς και τη θέση στην οποία πρέπει να εγγραφεί. Η προώθηση ωστόσο των πληροφοριών αυτών στη λογική addr_check δεν λαμβάνει χώρα μέχρι να αρχίσει το μπλοκ να μεταφέρεται από τον buffer στην κεντρική μνήμη.

Η δεύτερη διεργασία, Transfer_Pr, είναι αυτή που πραγματοποιεί τη μεταφορά αυτή. Είναι δομημένη ως εξής.

Στην πρώτη δήλωση if ανανεώνεται στη θετική ακμή του ρολογιού του διαύλου του επεξεργαστή το bit που δηλώνει αν η μεταφορά είναι IDLE ή όχι, καθώς και το bit που δηλώνει αν η συγκεκριμένη συσκευή slave έχει επιλεχθεί ή όχι.

Στη δεύτερη δήλωση if ελέγχεται αν οι συνθήκες είναι κατάλληλες και αν ναι, ξεκινά τη μεταφορά του μπλοκ, θέτοντας το bit που δηλώνει κατάσταση λειτουργίας μεταφοράς μπλοκ από τον buffer στην κεντρική μνήμη και σηματοδοτώντας τη λογική addr-check να αποθηκεύσει τη νέα ετικέτα. Οι συνθήκες θεωρούνται κατάλληλες όταν ο buffer είναι γεμάτος (το μπλοκ δηλαδή βρίσκεται ολόκληρο σε αυτόν) και πραγματοποιείται μια IDLE μεταφορά από τον επεξεργαστή, ώστε να μη διακοπεί κάποια προηγούμενη λειτουργία, ή πραγματοποιείται από τον επεξεργαστή προσπέλαση σε άλλη συσκευή slave (όπως κάποια από τις άλλες κρυφές μνήμες). Η IDLE μεταφορά μπορεί να πραγματοποιηθεί είτε επειδή ο επεξεργαστής δε θέλει να πραγματοποιήσει καμία προσπέλαση, ή επειδή η κρυφή μνήμη έδωσε μια απόκριση RETRY και σε απάντηση ο επεξεργαστής πραγματοποίησε μια μεταφορά IDLE (σύμφωνα με το πρωτόκολλο AHB).

Η τρίτη δήλωση if ελέγχει αν η κρυφή μνήμη βρίσκεται σε λειτουργία μεταφοράς και αν ναι, οδηγεί στο 1 το bit ενεργοποίησης ανάγνωσης από τον buffer. Έτσι, όσο η κρυφή μνήμη βρίσκεται σε λειτουργία μεταφοράς, σε κάθε θετική ακμή του ρολογιού του διαύλου του επεξεργαστή θα διαβάζεται μία γραμμή από τον buffer.

Η τέταρτη δήλωση if υλοποιεί έναν μετρητή, ο οποίος αρχικοποιείται ώστε να δηλώνει την πρώτη γραμμή του μπλοκ που θα μεταφερθεί. Σε λειτουργία μεταφοράς, ο μετρητής αυξάνεται σε κάθε αρνητική ακμή. Επίσης, στην ίδια δήλωση if, αν η κρυφή μνήμη είναι σε λειτουργία μεταφοράς, στην αρνητική ακμή τίθενται τα bits εγγραφής στην κεντρική μνήμη. Συνεπώς, σε κάθε (επόμενη) θετική ακμή του ρολογιού πραγματοποιείται μια εγγραφή στην κεντρική μνήμη (της γραμμής που διαβάστηκε από τον buffer στην προηγούμενη θετική ακμή). Στη θετική ακμή φυσικά που ο buffer διαβάζει την πρώτη γραμμή του μπλοκ, δεν πραγματοποιείται εγγραφή.

Στην τελευταία δήλωση if πραγματοποιείται η εξής λειτουργία. Επειδή ο μετρητής που προαναφέρθηκε αυξάνεται σε κάθε αρνητική ακμή (ακόμα και πριν την πρώτη ανάγνωση γραμμής από τον buffer, οπότε δεν πραγματοποιείται εγγραφή), η πραγματική τιμή της διεύθυνσης που στέλνεται στην κεντρική μνήμη πρέπει να είναι η τιμή του μετρητή μειωμένη κατά 1. Αυτός ο υπολογισμός της σωστής διεύθυνσης γίνεται εδώ.

Στην τελευταία τέλος διεργασία, αποθηκεύεται σε έναν καταχωρητή το bit που δηλώνει αν η κρυφή μνήμη είναι σε λειτουργία μεταφοράς, ώστε να μπορεί να χρησιμοποιηθεί η πληροφορία αυτή στον επόμενο κύκλο.

5.11.multiplexor.vhd

Η λειτουργία που εκτελεί η λογική αυτή είναι πολύ απλή. Απλώς επιλέγει αν στην κεντρική μνήμη τροφοδοτηθούν δεδομένα από τη διεπαφή slave της κρυφής μνήμης (ουσιαστικά αιτήσεις του επεξεργαστή) ή από τη λογική μεταφοράς transfer_logic (μεταφορά ενός μπλοκ από τον buffer).

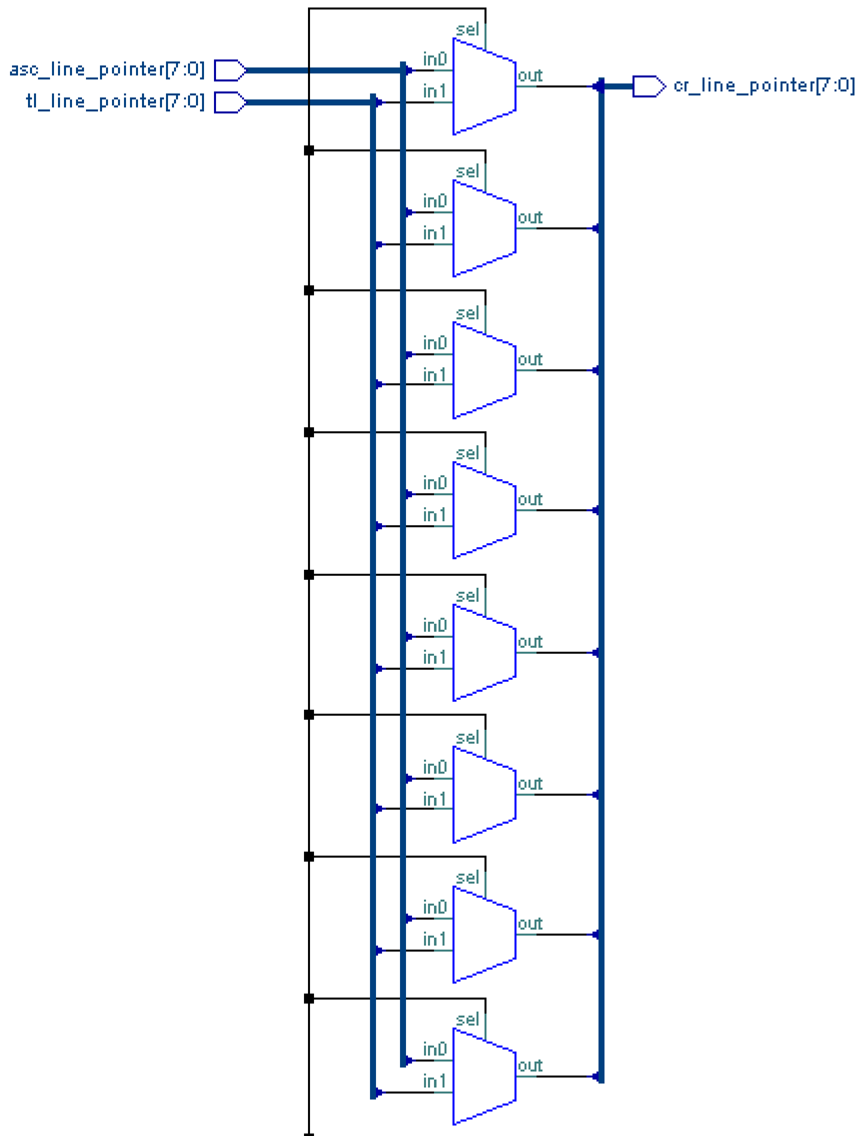
Οι είσοδοι της οντότητας αυτής είναι:

- **asc_line_pointer** : Από ahb_slv_cache. Ακέραιος που δηλώνει σε ποια γραμμή της κεντρικής μνήμης πρέπει να πραγματοποιηθεί προσπέλαση.
- **asc_write_data** : Από ahb_slv_cache. Δεδομένα προς εγγραφή στην κεντρική μνήμη.
- **asc_write_ar** : Από ahb_slv_cache. Τέσσερα bits που δηλώνουν σε ποια από τα τέσσερα τμήματα της μνήμης θα πραγματοποιηθεί εγγραφή (ένα bit για κάθε μνήμη).
- **trl_line_pointer** : Από transfer_logic. Ακέραιος που δηλώνει σε ποια γραμμή της κεντρικής μνήμης πρέπει να πραγματοποιηθεί προσπέλαση.
- **trl_write_data** : Από transfer_logic. Δεδομένα προς εγγραφή στην κεντρική μνήμη.
- **trl_write_ar** : Από transfer_logic. Τέσσερα bits που δηλώνουν σε ποια από τα τέσσερα τμήματα της μνήμης θα πραγματοποιηθεί εγγραφή (ένα bit για κάθε μνήμη).
- **transfer_mode** : Από transfer_logic. Δηλώνει αν η κρυφή μνήμη βρίσκεται σε κανονική κατάσταση λειτουργίας ή σε λειτουργία μεταφοράς μπλοκ από τον buffer στην κεντρική μνήμη.

Οι έξοδοι της οντότητας αυτής είναι:

- **cr_line_pointer** : Προς cache_ram. Ακέραιος που δηλώνει σε ποια γραμμή της κεντρικής μνήμης πρέπει να πραγματοποιηθεί προσπέλαση. Είναι είτε ο ακέραιος που δηλώνει η διεπαφή slave είτε αυτός που δηλώνει η λογική μεταφοράς.
- **cr_write_data** : Προς cache_ram. Δεδομένα προς εγγραφή στην κεντρική μνήμη. Είναι είτε τα δεδομένα που οδηγεί η διεπαφή slave είτε αυτά που οδηγεί η λογική μεταφοράς.
- **cr_write_ar** : Προς cache_ram. Τέσσερα bits που δηλώνουν σε ποια από τα τέσσερα τμήματα της μνήμης θα πραγματοποιηθεί εγγραφή (ένα bit για κάθε μνήμη). Είναι είτε τα bits εγγραφής που οδηγεί η διεπαφή slave είτε αυτά που οδηγεί η λογική μεταφοράς.

Η λογική αυτή υλοποιεί έναν απλό πολυπλέκτη (αποτελούμενο από μια σειρά μικρότερων πολυπλεκτών, όπως φαίνεται στην Εικόνα 42). Τα δεδομένα εισόδου είναι ο δείκτης γραμμής, τα δεδομένα προς εγγραφής και τα bits ενεργοποίησης εγγραφής τόσο της διεπαφής slave της κρυφής μνήμης όσο και της λογικής μεταφοράς μπλοκ από τον buffer στην κεντρική μνήμη. Στις εξόδους του πολυπλέκτη οδηγούνται είτε τα μεν είτε τα δε. Η γραμμή επιλογής είναι το bit που δηλώνει αν η κρυφή μνήμη βρίσκεται σε κανονική λειτουργία (οπότε προωθούνται οι πληροφορίες της διεπαφής slave) ή σε λειτουργία μεταφοράς μπλοκ (οπότε προωθούνται οι πληροφορίες της λογικής μεταφοράς).



Εικόνα 42: Η σειρά πολυπλεκτών που αποφασίζει ποιος δείκτης γραμμής θα οδηγηθεί στην έξοδο, όπως παράγεται από το εργαλείο σύνθεσης

5.12.fetch_list.vhd

Η λογική αυτή αποθηκεύει τις διευθύνσεις των μπλοκ για τα οποία υπάρχει στη fifo αίτηση μεταφοράς από την κύρια μνήμη στην κρυφή. Με τον τρόπο αυτό, η διεπαφή slave της κρυφής μνήμης μπορεί να αποφεύγει την αποθήκευση στη fifo της ίδιας αίτησης δύο φορές, πράγμα που μπορεί να οδηγήσει σε προβλήματα ασυνέπειας (καθώς θα υπήρχαν δύο αντίγραφα του ίδιου μπλοκ στην κρυφή μνήμη). Το τμήμα της διεύθυνσης που αποθηκεύεται εδώ είναι μόνο εκείνο που ορίζει το μπλοκ (ετικέτα και bits επιλογής συνόλου), χωρίς άλλες πληροφορίες (offset ή bits επιλογής byte), οπότε η μνήμη μπορεί να είναι μικρή σε μέγεθος.

Οι είσοδοι της οντότητας αυτής είναι:

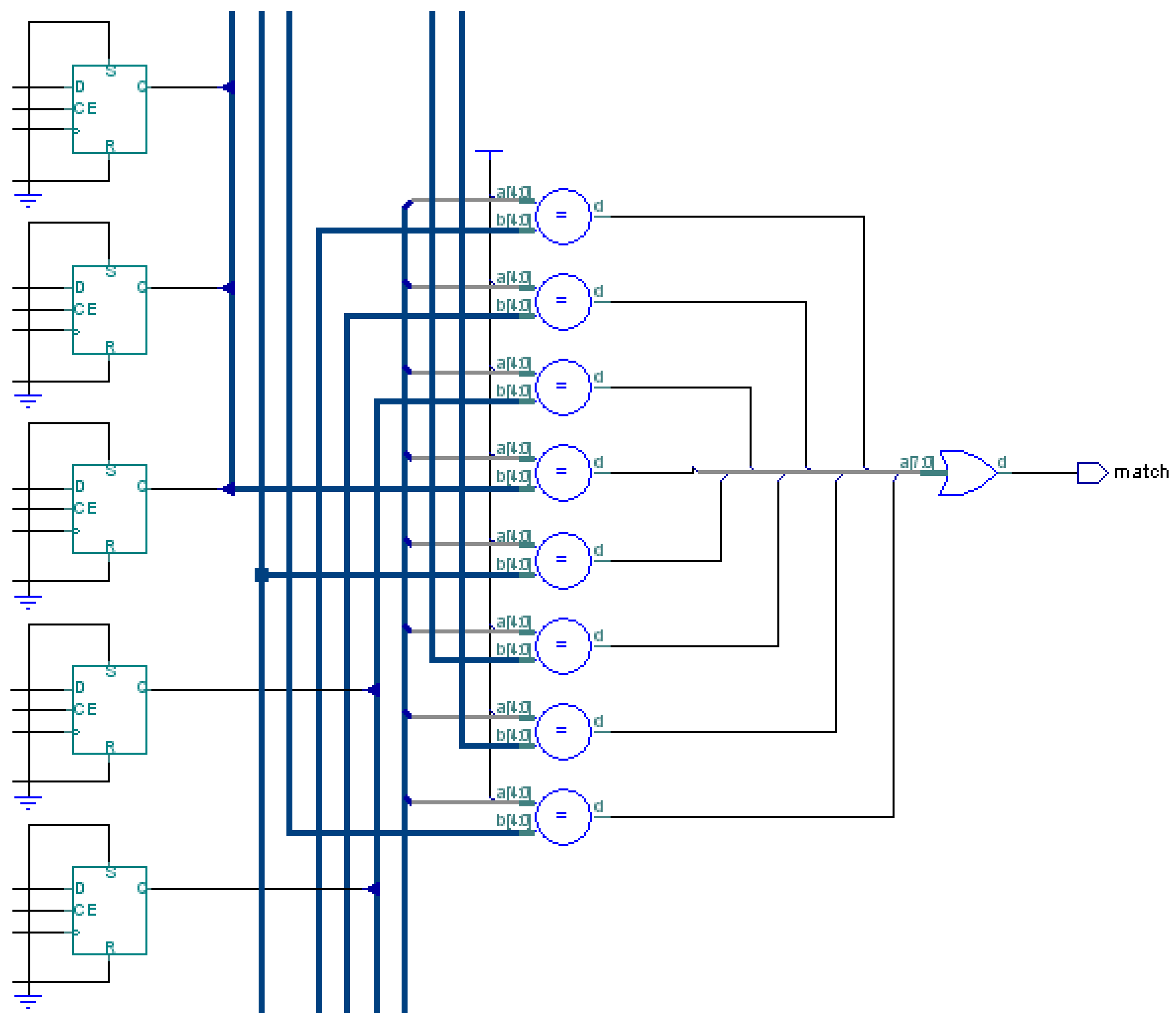
- **D** : Από ahb_slv_cache. Είναι τα δεδομένα που πρέπει να αποθηκευθούν στο αρχείο fetch_list κάθε φορά που γράφεται μια αίτηση μεταφοράς μπλοκ στη fifo. Αποτελούνται μόνο από το τμήμα της διεύθυνσης που ορίζει το μπλοκ, χωρίς άλλες πληροφορίες.
- **rst** : Από τον δίαυλο του επεξεργαστή. Είναι το σήμα reset.
- **clk** : Από τον δίαυλο του επεξεργαστή. Είναι το ρολόι του διαύλου του επεξεργαστή.
- **wren** : Από ahb_slv_cache. Είναι το σήμα ενεργοποίησης εγγραφής. Στη θετική ακμή ακμή του ρολογιού, αν το σήμα αυτό είναι 1, η λογική fetch_list πρέπει να πραγματοποιήσει μια εγγραφή, να θέσει το bit που δηλώνει εν αναμονή μεταφορά (αίτηση μεταφοράς που δεν έχει εξυπηρετηθεί) στο 1 και να αυξήσει τον δείκτη εγγραφής.
- **rden** : Από transfer_logic. Είναι το σήμα ενεργοποίησης ανάγνωσης, και τίθεται από τη λογική transfer_logic, κάθε φορά που ξεκινά η μεταφορά ενός μπλοκ από τον buffer στην κεντρική μνήμη. Αποτελεί ουσιαστικά το σήμα ενεργοποίησης εγγραφής μιας νέας ετικέτας στο αρχείο της λογικής addr_check, αφού και οι δύο λειτουργίες εκτελούνται ταυτόχρονα. Στη θετική ακμή ακμή του ρολογιού, αν το σήμα αυτό είναι 1, η λογική fetch_list πρέπει να πραγματοποιήσει μια ανάγνωση, να θέσει δηλαδή το bit που δηλώνει εν αναμονή μεταφορά (αίτηση μεταφοράς που δεν έχει εξυπηρετηθεί) στο 0 και να αυξήσει τον δείκτη ανάγνωσης.

Η οντότητα αυτή έχει μία μόνο έξοδο:
- **match** : Προς ahb_slv_cache. Δηλώνει αν μια αίτηση μεταφοράς ενός μπλοκ που επιθυμεί να εγγράψει η διεπαφή slave στη fifo βρίσκεται ήδη εκεί.

Η λογική αυτή υλοποιεί ένα αρχείο καταχωρητών στο οποίο αποθηκεύονται οι διευθύνσεις των μπλοκ για τα οποία υπάρχει στη fifo αίτηση μεταφοράς από την κύρια μνήμη στην κρυφή. Ο αριθμός θέσεων του αρχείου αυτού είναι ίδιος με τον αριθμό θέσεων της fifo (αφού μπορεί όλες οι θέσεις τις fifo να είναι γεμάτες με αιτήσεις μεταφοράς μπλοκ), ενώ χρησιμοποιούνται κι εδώ χωριστοί δείκτες εγγραφής και ανάγνωσης.

Με κάθε εγγραφή στη fifo μιας αίτησης μεταφοράς, γράφεται και εδώ η αντίστοιχη διεύθυνση του μπλοκ, μαζί με το bit εγκυρότητας που αναφέρθηκε παραπάνω, ενώ ο δείκτης εγγραφής προχωρά μια θέση. Η ανάγνωση έχει εδώ διαφορετική έννοια, εφόσον δεν διαβάζεται πραγματικά κάτι, αλλά ακυρώνονται τα δεδομένα μιας θέσης, όταν η αίτηση του αντίστοιχου μπλοκ έχει εξυπηρετηθεί. Κάθε φορά που ένα μπλοκ είναι στον buffer και ξεκινά η μεταφορά του στην κεντρική μνήμη, το bit εγκυρότητας της θέσης στην οποία δείχνει ο δείκτης ανάγνωσης τίθεται στο 0 και ο δείκτης ανάγνωσης προχωρά μια θέση. Επιπλέον έλεγχοι δεν απαιτούνται, εφόσον όλοι οι απαραίτητοι έλεγχοι (όπως ο έλεγχος για πλήρη μνήμη) πραγματοποιούνται από τη fifo, αντίστοιχα με την οποία γίνονται και εδώ οι εγγραφές και οι αναγνώσεις. Η λογική που μόλις περιγράφηκε υλοποιείται στην πρώτη διεργασία του κώδικα, η οποία ονομάζεται fl_update.

Στη δεύτερη διεργασία του κώδικα περιγράφεται η λογική ελέγχου για ταίριασμα. Όπως περιγράφηκε στη λογική ahb_slv_cache, προτού η διεπαφή slave εγγράψει μια νέα αίτηση μεταφοράς μπλοκ στη fifo, οδηγεί πρώτα τη διεύθυνση στην παρούσα λογική για να ελέγξει ότι η ίδια αίτηση δεν περιέχεται ήδη εκεί. Το τμήμα αυτό παίρνει το απαραίτητο κομμάτι της διεύθυνσης και το εισάγει σε μια σειρά συγκριτών, ο καθένας από τους οποίους το συγκρίνει με τη διεύθυνση του μπλοκ που βρίσκεται σε μια θέση του αρχείου καταχωρητών (ενώ παράλληλα ελέγχει και αν το αντίστοιχο bit εγκυρότητας της θέσης είναι 1). Αν κάποιος συγκριτής βρει ότι η νέα διεύθυνση ισούται με την αποθηκευμένη διεύθυνση, και αν το αντίστοιχο bit εγκυρότητας είναι 1, τότε η έξοδος match οδηγείται στο 1 προκειμένου να ενημερωθεί η λογική της διεπαφής slave ότι η αίτηση που επιθυμεί να εγγράψει στη fifo υπάρχει ήδη. Η έξοδος match δηλαδή οδηγείται από το λογικό or των εξόδων των συγκριτών.



Εικόνα 43: Η διάταξη οδήγησης της εξόδου match, όπως παράγεται από το εργαλείο σύνθεσης

Κεφάλαιο 6:
Σύνθεση και εξομοίωση

6.1.Σύνθεση

Για τη σύνθεση του σχεδιασμού χρησιμοποιήηκε το εργαλείο σύνθεσης Leonardo Spectrum, έκδοση 2006a.59, της Mentor Graphics. Η σύνθεση του σχεδιασμού που παρουσιάζεται εδώ πραγματοποιήθηκε σε τεχνολογία ASIC, χρησιμοποιώντας τη βιβλιοθήκη SCL05u του Leonardo. Πραγματοποιήθηκαν δε δύο «περάσματα» βελτιστοποίησης, ένα για περιοχή και ένα για καθυστέρηση.

Τα αποτελέσματα που παρουσιάζονται εδώ αφορούν μια κρυφή μνήμη δεύτερου επιπέδου, τα χαρακτηριστικά της οποίας ορίστηκαν ως εξής. Το μέγεθος της κρυφής μνήμης επιλέχθηκε να είναι 1 kB, το μέγεθος ενός μπλοκ 64 bytes, και μέγεθος γραμμής 32 bits. Τα παραπάνω σημαίνουν ότι η κρυφή μνήμη αυτή περιέχει 256 γραμμές των 32 bits, κατανεμημένες σε 16 συνολικά μπλοκ με κάθε μπλοκ να περιλαμβάνει 16 γραμμές. Ο αριθμός των συνόλων επιλέχθηκε να είναι 2, πράγμα που σημαίνει 8 μπλοκ ανά σύνολο, δηλαδή η κρυφή μνήμη αυτή θα είναι οργανωμένη σε συνολοσυσχετισμό 8-τρόπων (8-way set-associative). Τέλος, το μέγεθος της fifo (και συνεπώς και του αρχείου καταχωρητών fetch_list) ορίστηκε σε 8 θέσεις. Είναι αυτονόητο ότι σχεδόν οποιαδήποτε αλλαγή στις παραπάνω επιλογές θα έχει διαφορετικά αποτελέσματα σύνθεσης.

Ο αριθμός των πυλών που χρησιμοποιήθηκαν για την υλοποίηση κάθε τμήματος φαίνεται παρακάτω (Πίνακας 9).

Τμήμα	**Αριθμός πυλών**
ahb_slv_cache	515
addr_check	1280
cache_ram (4*single_ram)	159208 (4* 39802)
fifo	7048
ahb_mst_cache	974
buffer	8497
lfsr	66
transfer_logic	403
multiplexor	289
fetch_list	918
Σύνολο	179198

Πίνακας 9: Αριθμός πυλών ανά τμήμα της κρυφής μνήμης

Απαιτούνται δηλαδή συνολικά 179198 πύλες. Η συντριπτική πλειοψηφία από αυτές χρησιμοποιείται στην υλοποίηση της κεντρικής μνήμης της κρυφής μνήμης (4*39802=159208), καθώς το Leonardo υλοποιεί τις μνήμες σε ASIC, χρησιμοποιώντας πύλες για την υλοποίηση των flip-flops. Πράγματι, μία μόνο από τις μνήμες της κεντρικής μνήμης (που περιέχει 256 γραμμές των 8 bits) απαιτεί 39802 πύλες. Αντίστοιχα, ο buffer της κρυφής μνήμης, ο οποίος αποτελείται

από 16 γραμμές των 32 bits, δηλαδή το ένα τέταρτο του αριθμού bits σε σχέση με μία από τις κεντρικές μνήμες, απαιτεί περίπου το ένα τέταρτο και σε αριθμό πυλών, δηλαδή 8497 πύλες.

Οι μέγιστες συχνότητες για τα δύο ρολόγια του σχεδιασμού υπολογίστηκαν ως εξής:

Ρολόι	**Συχνότητα**
Διαύλου επεξεργαστή	101.7 MHz
Διαύλου κύριας μνήμης	157 MHz

Πίνακας 10: Μέγιστες συχνότητες των ρολογιών του σχεδιασμού

Το ρολόι δηλαδή του διαύλου του επεξεργαστή υπολογίζεται πιο αργό σε σχέση με το ρολόι του διαύλου της κύριας μνήμης, γεγονός πολύ λογικό, εφόσον η πλειοψηφία του σχεδιασμού λειτουργεί με το πρώτο ρολόι και συνεπώς τα κρίσιμα μονοπάτια του ρολογιού αυτού είναι μεγαλύτερα.

Ως το κρίσιμο μονοπάτι με τη μεγαλύτερη καθυστέρηση υποδείχθηκε το μονοπάτι μεταφοράς μιας γραμμής από τον buffer στην κεντρική μνήμη, που περιλαμβάνει την ανάγνωση της γραμμής από τον buffer (με την πρώτη θετική ακμή), τον υπολογισμό του δείκτη γραμμής για την κεντρική μνήμη (στην αρνητική ακμή), την είσοδο των δεδομένων στον πολυπλέκτη και τελικά την εγγραφή στην κεντρική μνήμη (στην επόμενη θετική ακμή).

6.2.Εξομοίωση

Για την εξομοίωση του σχεδιασμού χρησιμοποιήθηκε το εργαλείο εξομοίωσης ModelSim, έκδοση SE 6.2c, της Mentor Graphics, ένα από τα πλέον διαδεδομένα προγράμματα εξομοίωσης. Τα απαραίτητα αρχεία της βιβλιοθήκης GRLIB (όπως αυτά που περιέχουν τα πακέτα που περιγράφουν τους τύπους των εισόδων/εξόδων μιας συσκευής AMBA AHB) έγιναν compile σε μια βιβλιοθήκη χρήστη του ModelSim, η οποία ονομάστηκε nlib και περιλαμβάνεται στον κώδικα των τμημάτων του σχεδιασμού.

Η ορθή λειτουργία του σχεδιασμού επιβεβαιώθηκε με μια σειρά από testebench, γραμμένων επίσης στη γλώσσα VHDL. Ο κώδικας VHDL των αρχείων αυτών έχει τη διαφορά ότι η λογική που περιγράφεται δεν είναι ανάγκη να είναι συνθέσιμη. Ο σκοπός των testbench αυτών είναι να εισάγουν στον σχεδιασμό τις κατάλληλες εισόδους, προκειμένου να ελεγχθεί η ορθή απόκρισή του σε αυτές.

Στα testbench αυτά δημιουργείται ένα αντίγραφο (instance) του ολοκληρωμένου σχεδιασμού, σύμφωνα πάντα με τις επιλογές του χρήστη. Στη συνέχεια, μια ομάδα διεργασιών αναλαμβάνει να οδηγεί τις εισόδους του σχεδιασμού με τα κατάλληλα δεδομένα, προκειμένου να βεβαιωθεί ότι η συμπεριφορά του σχεδιασμού (η οποία μπορεί να παρατηρηθεί βήμα-βήμα μέσα από το πρόγραμμα εξομοίωσης) είναι η αναμενόμενη. Φυσικά, η ίδια η λογική των διεργασιών αυτών θα πρέπει να έχει τις σωστές αποκρίσεις στις εξόδους του σχεδιασμού, όπως αυτές περιγράφοναι από το πρωτόκολλο AMBA AHB.

Ο κώδικας ενός τέτοιου testbench παρατίθεται ενδεικτικά παρακάτω:

```
library ieee;
use ieee.std_logic_1164.all;
library nlib;
use nlib.amba.all;
use nlib.stdlib.all;
use nlib.devices.all;
```

```
entity test_Cache is
end;

architecture only of test_Cache is

component Cache is
        generic (
                kbytes  : integer := 1; --Size of main memory in kbytes
                cache_size : integer := 1024; -- Cache size in Kilobytes (default 1mb)
                block_size : integer := 64; -- Block size in bytes
                line_size : integer := 32; -- Line size (data size) in bits (16 or 32)
                set_number : integer := 1; -- Number of sets
                fifo_length : integer := 8;
                slv_hindex  : integer := 0;
                slv_haddr   : integer := 0;
                slv_hmask   : integer := 16#fff#;
                mst_hindex  : integer := 0;
                mst_venid   : integer := 0;
                mst_devid   : integer := 0;
                mst_version : integer := 0;
                mst_chprot  : integer := 3
        );
        port (
                rst     : in  std_ulogic;

                clk1     : in  std_ulogic;
                ahbsi   : in  ahb_slv_in_type;
                ahbso   : out ahb_slv_out_type;

                clk2  : in  std_ulogic;
                ahbi : in  ahb_mst_in_type;
                ahbo : out ahb_mst_out_type
        );
end component;

constant s_kbytes : integer := 1;
constant s_cache_size : integer := 1;
constant s_block_size : integer := 64;
constant s_line_size : integer := 32;
constant s_set_number : integer := 2;
constant s_fifo_length : integer := 8;

signal s_rst : std_ulogic := '1';
signal s_clk1, s_clk2 : std_ulogic := '0';
signal s_ahbsi : ahb_slv_in_type;
signal s_ahbso : ahb_slv_out_type;
signal s_ahbi  : ahb_mst_in_type;
signal s_ahbo  : ahb_mst_out_type;

type reg_type_arb is record
  hbusreq : std_ulogic;
  hgrant : std_ulogic;
end record;

type reg_type_mem is record
  haddr : std_logic_vector(31 downto 0);
end record;

type reg_type_proc is record
```

```
  nextaddr : std_logic_vector(31 downto 0);
  htrans : std_logic_vector(1 downto 0);
end record;

signal r_arb, rin_arb : reg_type_arb;
signal r_mem, rin_mem : reg_type_mem;
signal r_proc, rin_proc : reg_type_proc;


begin

mCache : Cache
   GENERIC MAP (
            kbytes  => s_kbytes,
            cache_size => s_cache_size,
            block_size => s_block_size,
            line_size => s_line_size,
            set_number => s_set_number,
            fifo_length => s_fifo_length)
   PORT MAP (
            rst     => s_rst,
            clk1    => s_clk1,
            clk2       => s_clk2,

            ahbsi   => s_ahbsi,
            ahbso   => s_ahbso,

            ahbi => s_ahbi,
            ahbo => s_ahbo);


proc : PROCESS(s_ahbso, s_clk1, s_rst)
   variable v : reg_type_proc;
   begin

   s_ahbsi.hsel <= (others => '0'); s_ahbsi.hsel(0) <= '1';
   s_ahbsi.haddr <= r_proc.nextaddr; s_ahbsi.hwrite <= '0'; s_ahbsi.htrans <= r_proc.htrans;
   s_ahbsi.hsize <= "010"; s_ahbsi.hburst <= "000"; s_ahbsi.hwdata <= "10101010101010101010101010101010";
   s_ahbsi.hprot <= "0001"; s_ahbsi.hready <= '1'; s_ahbsi.hmaster <= "0001"; s_ahbsi.hmastlock <= '0';
   s_ahbsi.hmbsel <= "0000"; s_ahbsi.hcache <= '0'; s_ahbsi.hirq <= "00000000000000000000000000000000";
   s_ahbsi.testen <= '0'; s_ahbsi.testrst <= '0'; s_ahbsi.scanen <= '0';

   if s_ahbso.hresp = "00" then
            if r_proc.nextaddr = "01010101010101010101010101111101" then
                        v.nextaddr := "01010101010101010101010101000001";
            else
                        v.nextaddr := r_proc.nextaddr + "100";
            end if;
            v.htrans := "10";
   elsif s_ahbso.hresp = "10" and s_ahbso.hready = '0' then
            v.nextaddr := r_proc.nextaddr;
            v.htrans := "00";
   else
            v.nextaddr := r_proc.nextaddr;
            v.htrans := "10";
   end if;

   rin_proc <= v;

end PROCESS proc;
```

```
arb : PROCESS(s_ahbo, s_clk2)
   variable v : reg_type_arb;
   begin

   v := r_arb; v.hbusreq := s_ahbo.hbusreq;

   s_ahbi.hgrant <= "0000000000000000";

   if r_arb.hbusreq = '1' then
      s_ahbi.hgrant(0) <= '1';
   end if;

   rin_arb <= v;

end PROCESS arb;


mem : PROCESS(s_ahbo, s_clk2)
   variable v : reg_type_mem;
   begin

   v := r_mem; v.haddr := s_ahbo.haddr;

   s_ahbi.hready <= '1'; s_ahbi.hrdata <= r_mem.haddr;
   s_ahbi.hresp <= "00"; s_ahbi.hcache <= '0'; s_ahbi.hirq <= "00000000000000001000000000000000";
   s_ahbi.testen <= '0'; s_ahbi.testrst <= '0'; s_ahbi.scanen <= '0';

   rin_mem <= v;

end PROCESS mem;



vr : PROCESS(s_clk1, s_clk2, s_rst)
   begin

   if s_rst = '0' then
      r_proc.nextaddr <= "01010101010101010101010101000001";
      r_proc.htrans <= "10";
   elsif rising_edge(s_clk1) then
      r_proc <= rin_proc;
   end if;

   if rising_edge(s_clk2) then
      r_arb <= rin_arb;
      r_mem <= rin_mem;
   end if;

end PROCESS vr;


clock1 : PROCESS
   begin
   wait for 10 ns; s_clk1  <= not s_clk1;
end PROCESS clock1;

clock2 : PROCESS
   begin
```

```
  wait for 11 ns; s_clk2  <= not s_clk2;
end PROCESS clock2;

stimulus : PROCESS
  begin
  wait for 5 ns; s_rst  <= '0';
  wait for 4 ns; s_rst  <= '1';
  wait;
end PROCESS stimulus;

end only;
```

Η λογική του αρχείου αυτού περιλαμβάνει μια σειρά διεργασιών, καθεμία από τις οποίες εξομοιώνει τη λειτουργία μιας από τις συσκευές με τις οποίες επικοινωνεί η κρυφή μνήμη. Έτσι, η διεργασία proc αναλαμβάνει το ρόλο του επεξεργαστή, η διεργασία mem το ρόλο της κρυφής μνήμης, ενώ η διεργασία arb λειτουργεί ως arbiter του διαύλου της κύριας μνήμης.

Η διεργασία proc ζητάει από την κρυφή μνήμη την ανάγνωση όλων των γραμμών ενός μπλοκ με τη σειρά και όταν αυτές τελειώσουν, ξεκινά από την αρχή. Όταν φυσικά λαμβάνει από τον σχεδιασμό μια απάντηση RETRY, δεν ζητά μια νέα γραμμή, αλλά εισάγει μια μεταφορά IDLE, όπως ορίζεται από το πρωτόκολλο, και συνεχίζει μετά. Στην πρώτη αίτηση της πρώτης γραμμής του μπλοκ συμβαίνει προφανώς αστοχία στην κρυφή μνήμη (εφόσον είναι άδεια) και το μπλοκ πρέπει να διαβαστεί γραμμη-γραμμή από την κύρια μνήμη, δηλαδή από τη διεργασία mem. Οι επόμενες αιτήσεις για τις άλλες γραμμές του μπλοκ δεν εκκινούν φυσικά μεταφορά από την κύρια μνήμη, γιατί στη fifo θα υπάρχει ήδη μια αίτηση μεταφοράς για το ίδιο μπλοκ.

Η διεργασία mem δέχεται τις αιτήσεις που της στέλνει η κρυφή μνήμη και για κάθε διεύθυνση που της ζητείται, δίνει σαν αναγνωσμένα δεδομένα την ίδια αυτή διεύθυνση.

Η διεργασία arb αναλαμβάνει να δώσει στην κρυφή μνήμη τον έλεγχο του διαύλου της κύριας μνήμης, όταν αυτή τον ζητά.

Οι τελευταίες τρεις διεργασίες παράγουν τα ρολόγια του διαύλου του επεξεργαστή και της κύριας μνήμης (η συχνότητά τους μπορεί φυσικά να ρυθμιστεί) και δίνουν έναν παλμό reset στην αρχή της εξομοίωσης για αρχικοποίηση του συστήματος.

Το τελικό αποτέλεσμα του παραπάνω testbench είναι η κρυφή μνήμη να μεταφέρει από την κύρια μνήμη το μπλοκ και στη συνέχεια να εξυπηρετεί τις αιτήσεις ανάγνωσης που δέχεται. Τα δεδομένα που διαβάζονται από κάθε γραμμή είναι φυσικά η διεύθυνση της γραμμής αυτής στην κύρια μνήμη.

Δόθηκε προσοχή ώστε τα testbench που χρησιμοποιήθηκαν να καλύπτουν κάθε πιθανή περίπτωση που μπορεί να προκύψει κατά τη λειτουργία της κρυφής μνήμης. Οι περιπτώσεις αυτές περιλαμβάνουν τον έλεγχο κάθε δυνατού συνδυασμού προσπελάσεων στην κρυφή μνήμη (ευστοχία ανάγνωσης μετά από ευστοχία εγγραφής, αστοχία ανάγνωσης μετά από ευστοχία εγγραφής, ευστοχία εγγραφής μετά από αστοχία ανάγνωσης κ.λπ.), έλεγχο των ακραίων συνθηκών της fifo (άδεια, γεμάτη, σχεδόν γεμάτη), τη συχνή απώλεια του διαύλου της κύριας μνήμης, τη συνεχή απόδοση του διαύλου ακόμα και όταν δε ζητείται, ίσες ή πολύ διαφορετικές συχνότητες των δύο ρολογιών κ.ά. Δοκιμές πραγματοποιήθηκαν και στα επιμέρους τμήματα του σχεδιασμού, ακόμα και για περιπτώσεις οι οποίες σύμφωνα με το πρωτόκολλο AHB και τη συνολική λειτουργία της κρυφής μνήμης είναι αδύνατο να εμφανιστούν. Ο λεπτομερής αυτός έλεγχος του σχεδιασμού προσφέρει κάθε βεβαιότητα για την ορθή λειτουργία του, ακόμα και στις πιο ασυνήθιστες συνθήκες.

Παράρτημα:
Κώδικας VHDL

Στο παρόν παράρτημα παρατίθεται ο κώδικας σε γλώσσα VHDL του σχεδιασμού.

A1.Cache.vhd

Cache.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library nlib;
use nlib.amba.all;
use nlib.stdlib.all;
use nlib.devices.all;



entity Cache is
	generic (
		kbytes  : integer := 1;
		cache_size : integer := 1024;
		block_size : integer := 64;
		line_size : integer := 32;
		set_number : integer := 1;
		fifo_length : integer := 8;
		slv_hindex  : integer := 0;
		slv_haddr   : integer := 0;
		slv_hmask   : integer := 16#fff#;
		mst_hindex  : integer := 0;
		mst_venid   : integer := 0;
		mst_devid   : integer := 0;
		mst_version : integer := 0;
		mst_chprot  : integer := 3
	);
	port (
		rst     : in  std_ulogic;

		clk1     : in  std_ulogic;
		ahbsi   : in  ahb_slv_in_type;
		ahbso   : out ahb_slv_out_type;

		clk2  : in  std_ulogic;
		ahbi : in  ahb_mst_in_type;
		ahbo : out ahb_mst_out_type
	);
end;

architecture main of Cache is
```

```
-- COMPONENT declaration

component ahbram is
  generic (
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#fff#;
    abits   : integer;
    lines_number : integer;
    line_size : integer := 32
  );
  port (
    --addr_check ports
    hit     : in std_ulogic;
    line_pointer : in integer range 0 to lines_number-1;

    --amba bus ports
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;

    --cache_ram ports
    ramdata  : in std_logic_vector(line_size-1 downto 0);
    ramaddr  : out integer range 0 to lines_number-1;
    write_data : out std_logic_vector(line_size-1 downto 0);
    write    : out std_logic_vector(3 downto 0);

    --fifo ports
    fifo_full : in std_ulogic := '0';
    fifo_near_full : in std_ulogic := '0';
    fifo_write_en : out std_ulogic := '0';
    fifo_entry: out std_logic_vector(abits+4 downto 0);
    fifo_write_data: out std_logic_vector(31 downto 0);

    --transfer_logic ports
    transfer_mode : in std_ulogic;

    --fetch_list ports
    fetch_data_in : out std_logic_vector(abits+1 downto 0);
    fetch_write_en : out std_ulogic;
    fetch_match : in std_ulogic
  );
end component;

component ahbmst is
  generic (
    hindex  : integer := 0;
    venid   : integer := 0;
    devid   : integer := 0;
    version : integer := 0;
    chprot  : integer := 3;
    abits   : integer;
    offset_bits : integer;
    lines_block : integer;
    line_size : integer );
   port (
      rst  : in  std_ulogic;
      clk  : in  std_ulogic;
```

```
    ahbi : in  ahb_mst_in_type;
    ahbo : out ahb_mst_out_type;

    fifo_empty : in std_ulogic := '1';
    fifo_read_en : out std_ulogic := '0';
    fifo_data : in std_logic_vector(abits+4 downto 0);
    fifo_write_data : in std_logic_vector(line_size-1 downto 0);

    buffer_empty    : in std_ulogic := '1';
    buffer_write_en : out std_ulogic := '0';
    buffer_data_in  : out std_logic_vector(line_size-1 downto 0);

    tag_addr_in  : out std_logic_vector(abits-1 downto 0);
    tag_addr_write_en : out std_ulogic
    );
end component;

component addr_check is
        generic (
                abits : integer;
                offset_bits : integer;
                index_bits : integer;
                block_number : integer;
                set_assoc : integer;
                set_number : integer;
                lines_block : integer;
                lines_number : integer
        );
        port (
                clk :   in std_ulogic;
                rst :       in std_ulogic;
                ahbsi_haddr :       in std_logic_vector(31 downto 0);
                line_pointer :       out integer range 0 to lines_number-1;
                hit :       out std_ulogic;

                tag_in    : in std_logic_vector(abits-offset_bits-index_bits-1 downto 0);
                tag_write_en : in std_ulogic;
                block_num : in integer range 0 to block_number-1
        );
end component;

component cache_ram is
  generic (
    lines_number : integer;
    line_size : integer := 32
  );
  port (
    clk      : in std_ulogic;
    rst      : in std_ulogic;
    line_pointer  : in integer range 0 to lines_number-1;
    data_in   : in std_logic_vector(line_size-1 downto 0);
    data_out  : out std_logic_vector(line_size-1 downto 0);
    write_ar    : in std_logic_vector(3 downto 0)
  );
end component;

component dff_N_W is
      generic(
            N: INTEGER := 8;
            line_size : integer;
```

```
            abits  : integer
          );
          port(
                    D: in std_logic_vector(abits+4 downto 0);
                    write_data_in: in std_logic_vector(line_size-1 downto 0);
                    Q: out std_logic_vector(abits+4 downto 0);
                    write_data_out: out std_logic_vector(line_size-1 downto 0);
                    RES, RDCLK, WRCLK, RDEN, WREN: in std_ulogic;
                    F: buffer std_ulogic :='0'; --full
                    E: buffer std_ulogic :='1'; --empty
                    NF: buffer std_ulogic :='0' --near full
          );
end component;

component buf is
      generic(
            line_size : INTEGER;
            N: INTEGER := 8
                    );
          port(
                    D: in std_logic_vector(line_size-1 downto 0);
                    Q: out std_logic_vector(line_size-1 downto 0);
                    RES, RDCLK, WRCLK, RDEN, WREN: in std_ulogic;
                    trl_F: buffer std_ulogic :='0';
                    trl_E: buffer std_ulogic :='1';
                    mst_E:buffer std_ulogic :='1'
          );
end component;

component transfer_logic is
  generic (
    hindex  : integer := 0;
    abits  : integer;
    offset_bits : integer;
    index_bits : integer;
    line_size : integer := 32;
    set_number : integer := 1;
    block_number : integer;
    lines_number : integer;
    set_assoc : integer ;
    lines_block : integer
  );
  port (
    --addr_check ports
    tag     : out std_logic_vector(abits-offset_bits-index_bits-1 downto 0);
    tag_write_en : out std_ulogic;
    block_num : out integer range 0 to block_number-1;

    --multiplexor (and cache_ram) ports (transfer_m goes to ahb_slv_cache too)
    line_pointer  : out integer range 0 to lines_number-1;
    write_data   : out std_logic_vector(line_size-1 downto 0);
    write_ar    : out std_logic_vector(3 downto 0);
    transfer_m  : out std_ulogic;

    --amba bus ports
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    clk_mst : in  std_ulogic;
    not_idle: in  std_ulogic;
    ahbsi_hsel : in std_logic_vector(0 to NAHBSLV-1);
```

```
    --lfsr ports
    random_int_in : in integer range 0 to set_assoc-1;

    --buffer ports
    buffer_empty    : in std_ulogic := '1';
    buffer_full     : in std_ulogic := '0';
    buffer_read_en : out std_ulogic := '0';
    buffer_data  : in std_logic_vector(line_size-1 downto 0);

    --ahb_mst_cache ports
    addr_in  : in std_logic_vector(abits-1 downto 0);
    addr_write_en : in std_ulogic
  );
end component;

component multiplexor is
  generic (
    line_size : integer := 32;
    lines_number : integer
  );
  port (

    --transfer_logic ports
    tl_line_pointer  : in integer range 0 to lines_number-1;
    tl_write_data   : in std_logic_vector(line_size-1 downto 0);
    tl_write_ar    : in std_logic_vector(3 downto 0);
    transfer_mode  : in std_ulogic;

    --ahb_slv_chache ports
    asc_line_pointer  : in integer range 0 to lines_number-1;
    asc_write_data   : in std_logic_vector(line_size-1 downto 0);
    asc_write_ar    : in std_logic_vector(3 downto 0);

    --cache_ram ports
    cr_line_pointer  : out integer range 0 to lines_number-1;
    cr_write_data   : out std_logic_vector(line_size-1 downto 0);
    cr_write_ar    : out std_logic_vector(3 downto 0)
  );
end component;

component lfsr is
  generic (
    set_assoc: integer
  );
  port (
    clk : in std_ulogic;
    rst : in std_ulogic;
    random_int : out integer range 0 to set_assoc-1
  );
end component;

component fetch_list is
        generic(
                N: INTEGER := 8;
                abits  : integer := 8;
                offset_bits : integer
               );
               port(
                       D: in std_logic_vector(abits+1 downto 0);
```

```
                        rst, clk, rden, wren: in std_ulogic;
                        match : out std_ulogic
            );
end component;


-- CONSTANT declaration

constant abits : integer := log2(kbytes) + 8;
constant block_number : integer := (cache_size*1024)/block_size; --Number of blocks in cache
constant set_assoc : integer := block_number/set_number; --Number of blocks in set (set-associativity)
constant lines_number : integer := (cache_size*1024*8)/line_size; --Number of lines in cache
constant lines_block : integer := lines_number/block_number; --Number of lines in block
constant offset_bits : integer := log2(lines_block); --Number of bits for offset
constant index_bits : integer := log2(set_number); --Number of bits for index


-- SIGNAL declaration

-- ahb_slv_cache signals
signal adch_slv_hit : std_ulogic;
signal adch_slv_line_pointer : integer range 0 to lines_number-1;
signal ram_slv_ramdata : std_logic_vector(line_size-1 downto 0);
signal slv_mux_ramaddr : integer range 0 to lines_number-1;
signal slv_mux_write_data : std_logic_vector(line_size-1 downto 0);
signal slv_mux_write : std_logic_vector(3 downto 0);
signal fifo_slv_fifo_full, fifo_slv_fifo_near_full : std_ulogic;
signal slv_fifo_fifo_write_en : std_ulogic;
signal slv_fifo_fifo_entry : std_logic_vector(abits+4 downto 0);
signal slv_fifo_fifo_write_data : std_logic_vector(line_size-1 downto 0);
signal trl_transfer_mode : std_ulogic;
signal slv_fl_data : std_logic_vector(abits+1 downto 0);
signal slv_fl_write_en : std_ulogic;
signal fl_slv_match : std_ulogic;

-- ahb_mst_cache signals
signal fifo_mst_fifo_empty : std_ulogic;
signal mst_fifo_fifo_read_en : std_ulogic;
signal fifo_mst_fifo_data : std_logic_vector(abits+4 downto 0);
signal fifo_mst_fifo_write_data : std_logic_vector(line_size-1 downto 0);
signal buf_mst_buffer_empty : std_ulogic;
signal mst_buf_buffer_write_en : std_ulogic;
signal mst_buf_buffer_data_in : std_logic_vector(line_size-1 downto 0);
signal mst_trl_addr_in : std_logic_vector(abits-1 downto 0);
signal mst_trl_addr_write_en : std_ulogic;

-- add_check signals
signal trl_adch_tag_in : std_logic_vector(abits-offset_bits-index_bits-1 downto 0);
signal trl_adch_tag_write_en : std_ulogic;
signal trl_adch_block_num : integer range 0 to block_number-1;

-- cache_ram signals
signal mux_ram_line_pointer : integer range 0 to lines_number-1;
signal mux_ram_write_data : std_logic_vector(line_size-1 downto 0);
signal mux_ram_write_ar : std_logic_vector(3 downto 0);

-- buffer signals
signal buf_trl_buffer_data : std_logic_vector(line_size-1 downto 0);
signal trl_buf_buffer_read_en : std_ulogic;
signal buf_trl_buffer_full : std_ulogic;
```

```
signal buf_trl_buffer_empty : std_ulogic;

-- transfer_logic signals
signal trl_mux_line_pointer : integer range 0 to lines_number-1;
signal trl_mux_write_data : std_logic_vector(line_size-1 downto 0);
signal trl_mux_write_ar : std_logic_vector(3 downto 0);
signal lfsr_trl_random_int_in : integer range 0 to set_assoc-1;

begin

ahb_slv_cache : ahbram
  GENERIC MAP (
        hindex  => slv_hindex,
        haddr   => slv_haddr,
        hmask   => slv_hmask,
        abits   => abits,
        lines_number => lines_number,
        line_size => line_size)
  PORT MAP (
        hit     => adch_slv_hit,
        line_pointer => adch_slv_line_pointer,

        rst     => rst,
        clk     => clk1,
        ahbsi   => ahbsi,
        ahbso   => ahbso,

        ramdata  => ram_slv_ramdata,
        ramaddr  => slv_mux_ramaddr,
        write_data => slv_mux_write_data,
        write    => slv_mux_write,

        fifo_full => fifo_slv_fifo_full,
        fifo_near_full => fifo_slv_fifo_near_full,
        fifo_write_en => slv_fifo_fifo_write_en,
        fifo_entry => slv_fifo_fifo_entry,
        fifo_write_data => slv_fifo_fifo_write_data,

        transfer_mode => trl_transfer_mode,

        fetch_data_in => slv_fl_data,
        fetch_write_en => slv_fl_write_en,
        fetch_match => fl_slv_match);

ahb_mst_cache : ahbmst
  GENERIC MAP (
        hindex => mst_hindex,
        venid => mst_venid,
        devid => mst_devid,
        version => mst_version,
        chprot => mst_chprot,
        abits => abits,
        offset_bits => offset_bits,
        lines_block => lines_block,
        line_size => line_size )
  PORT MAP (
        rst => rst,
        clk => clk2,
        ahbi => ahbi,
        ahbo => ahbo,
```

```
            fifo_empty => fifo_mst_fifo_empty,
            fifo_read_en => mst_fifo_fifo_read_en,
            fifo_data => fifo_mst_fifo_data,
            fifo_write_data => fifo_mst_fifo_write_data,

            buffer_empty => buf_mst_buffer_empty,
            buffer_write_en => mst_buf_buffer_write_en,
            buffer_data_in => mst_buf_buffer_data_in,

            tag_addr_in => mst_trl_addr_in,
            tag_addr_write_en => mst_trl_addr_write_en);

addrcheck : addr_check
  GENERIC MAP (
            abits => abits,
            offset_bits => offset_bits,
            index_bits => index_bits,
            block_number => block_number,
            set_assoc => set_assoc,
            set_number => set_number,
            lines_block => lines_block,
            lines_number => lines_number) -- Line size (data size) in bits (16 or 32)
  PORT MAP (
            clk => clk1,
            rst => rst,
            ahbsi_haddr => ahbsi.haddr,
            line_pointer => adch_slv_line_pointer,
            hit => adch_slv_hit,

            tag_in => trl_adch_tag_in,
            tag_write_en => trl_adch_tag_write_en,
            block_num => trl_adch_block_num);

cacheram : cache_ram
  GENERIC MAP (
            lines_number => lines_number,
            line_size => line_size)
  PORT MAP (
            clk => clk1,
            rst => rst,
            line_pointer => mux_ram_line_pointer,
            data_in => mux_ram_write_data,
            data_out => ram_slv_ramdata,
            write_ar => mux_ram_write_ar);

fifo : dff_N_W
  GENERIC MAP (
            N => fifo_length,
            line_size => line_size,
            abits => abits)
  PORT MAP (
            D => slv_fifo_fifo_entry,
            write_data_in => slv_fifo_fifo_write_data,
            Q => fifo_mst_fifo_data,
            write_data_out => fifo_mst_fifo_write_data,
            RES => rst,
            RDCLK => clk2,
            WRCLK => clk1,
            RDEN => mst_fifo_fifo_read_en,
```

```
            WREN => slv_fifo_fifo_write_en,
            F => fifo_slv_fifo_full,
            E => fifo_mst_fifo_empty,
            NF => fifo_slv_fifo_near_full);

bufferm : buf
  GENERIC MAP (
            line_size => line_size,
            N => lines_block)
  PORT MAP (
            D => mst_buf_buffer_data_in,
            Q => buf_trl_buffer_data,
            RES => rst,
            RDCLK => clk1,
            WRCLK => clk2,
            RDEN => trl_buf_buffer_read_en,
            WREN => mst_buf_buffer_write_en,
            trl_F => buf_trl_buffer_full,
            trl_E => buf_trl_buffer_empty,
            mst_E => buf_mst_buffer_empty);

transferlogic : transfer_logic
  GENERIC MAP (
            hindex => 0,
            abits => abits,
            offset_bits => offset_bits,
            index_bits => index_bits,
            line_size => line_size,
            set_number => set_number,
            block_number => block_number,
            lines_number => lines_number,
            set_assoc => set_assoc,
            lines_block => lines_block)
  PORT MAP (
            --addr_check ports
            tag => trl_adch_tag_in,
            tag_write_en => trl_adch_tag_write_en,
            block_num => trl_adch_block_num,

            --multiplexor (and cache_ram) ports (transfer_m goes to ahb_slv_cache too)
            line_pointer => trl_mux_line_pointer,
            write_data => trl_mux_write_data,
            write_ar => trl_mux_write_ar,
            transfer_m => trl_transfer_mode,

            --amba bus ports
            rst => rst,
            clk => clk1,
            clk_mst => clk2,
            not_idle => ahbsi.htrans(1),
            ahbsi_hsel => ahbsi.hsel,

            --lfsr ports
            random_int_in => lfsr_trl_random_int_in,

            --buffer ports
            buffer_empty => buf_trl_buffer_empty,
            buffer_full => buf_trl_buffer_full,
            buffer_read_en => trl_buf_buffer_read_en,
            buffer_data => buf_trl_buffer_data,
```

```
        --ahb_mst_cache ports
        addr_in => mst_trl_addr_in, --just the address, without the byte selecting bits, so abits-1 downto 0
        addr_write_en => mst_trl_addr_write_en);

mux : multiplexor
  GENERIC MAP (
        line_size => line_size,
        lines_number => lines_number)
  PORT MAP (
        --transfer_logic ports
        tl_line_pointer => trl_mux_line_pointer,
        tl_write_data => trl_mux_write_data,
        tl_write_ar => trl_mux_write_ar,
        transfer_mode => trl_transfer_mode,

        --ahb_slv_chache ports
        asc_line_pointer => slv_mux_ramaddr,
        asc_write_data => slv_mux_write_data,
        asc_write_ar => slv_mux_write,

        --cache_ram ports
        cr_line_pointer => mux_ram_line_pointer,
        cr_write_data => mux_ram_write_data,
        cr_write_ar => mux_ram_write_ar);

lfsrm : lfsr
  GENERIC MAP (
        set_assoc => set_assoc)
  PORT MAP (
        clk => clk1,
        rst => rst,
        random_int => lfsr_trl_random_int_in);

fetchlist : fetch_list
  GENERIC MAP (
        N => fifo_length,
        abits => abits,
        offset_bits => offset_bits)
  PORT MAP (
        D => slv_fl_data,
        rst => rst,
        clk => clk1,
        rden => trl_adch_tag_write_en,
        wren => slv_fl_write_en,
        match => fl_slv_match);


end;
```

A2.ahb_slv_cache.vhd

ahb_slv_cache.vhd

```
library ieee;
use ieee.std_logic_1164.all;
```

```
library nlib;
use nlib.amba.all;
use nlib.stdlib.all;
use nlib.devices.all;

entity ahbram is
  generic (
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#fff#;
    abits   : integer;
    lines_number : integer;
    line_size : integer := 32
  );
  port (
    --addr_check ports
    hit     : in std_ulogic;
    line_pointer : in integer range 0 to lines_number-1;

    --amba bus ports
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;

    --cache_ram ports
    ramdata  : in std_logic_vector(line_size-1 downto 0);
    ramaddr  : out integer range 0 to lines_number-1;
    write_data : out std_logic_vector(line_size-1 downto 0);
    write    : out std_logic_vector(3 downto 0);

    --fifo ports
    fifo_full : in std_ulogic := '0';
    fifo_near_full : in std_ulogic := '0';
    fifo_write_en : out std_ulogic := '0';
    fifo_entry: out std_logic_vector(abits+4 downto 0);
    fifo_write_data: out std_logic_vector(line_size-1 downto 0);

    --transfer_logic ports
    transfer_mode : in std_ulogic;

    --fetch_list ports
    fetch_data_in : out std_logic_vector(abits+1 downto 0);
    fetch_write_en : out std_ulogic;
    fetch_match : in std_ulogic
  );
end;

architecture rtl of ahbram is

constant hconfig : ahb_config_type := (
  0 => ahb_device_reg ( 1, 1, 0, abits+2, 0),
  4 => ahb_membar(haddr, '1', '1', hmask),
  others => zero32);


type reg_type is record
  hwrite : std_ulogic;
  hready : std_ulogic;
  hsel   : std_ulogic;
```

```
  addr   : std_logic_vector(abits+1 downto 0);
  size   : std_logic_vector(1 downto 0);
  line_p : integer range 0 to lines_number-1;
  hit         : std_ulogic;
  resp  : std_logic_vector(1 downto 0);
  fifo_NF : std_ulogic;
  wait_c : std_ulogic;
end record;

signal r, c : reg_type;
signal ramsel : std_ulogic;

begin

  comb : process (ahbsi, r, hit, line_pointer, rst, ramdata, fifo_full, fifo_near_full, transfer_mode, fetch_match)
  variable bs : std_logic_vector(3 downto 0);
  variable v : reg_type;
  variable line_point  : integer range 0 to lines_number-1;
  variable fetch_write_var :std_ulogic;
  begin
    v := r; v.hready := '1'; v.resp := "00"; bs := (others => '0'); v.fifo_nf := fifo_near_full;
    v.wait_c := not r.hready; fetch_write_var := '0';

    if (r.hwrite = '1' and r.hit = '1' and r.resp/="10") or (r.hready = '0') then line_point := r.line_p;
    else
          line_point := line_pointer; bs := (others => '0');
    end if;

    if ahbsi.hready = '1' then
      v.hit := hit;
      v.line_p := line_pointer;
      v.hsel := ahbsi.hsel(hindex) and ahbsi.htrans(1);
      v.hwrite := ahbsi.hwrite and v.hsel;
      v.addr := ahbsi.haddr(abits+1 downto 0);
      v.size := ahbsi.hsize(1 downto 0);
    end if;

    if (r.hwrite = '1' and r.hit = '1' and r.resp/="10" and r.wait_c = '0') then
      case r.size(1 downto 0) is
      when "00" => bs (conv_integer(r.addr(1 downto 0))) := '1';
      when "01" => bs := r.addr(1) & r.addr(1) & not (r.addr(1) & r.addr(1));
      when others => bs := (others => '1');
      end case;
      v.hready := not (v.hsel and not ahbsi.hwrite);
      v.hwrite := v.hwrite and v.hready;
    end if;

    if (((((v.hit = '0') or (v.hwrite = '1' and v.hit = '1' and (fifo_full = '1' or v.fifo_NF = '1'))) and r.hready = '1' and v.hsel
= '1') or (transfer_mode = '1')) and ahbsi.htrans(1) = '1') then
          v.resp := "10";
          v.hready := '0';
    end if;

    if r.resp = "10" and r.hready = '0' then
          v.resp := r.resp;
          v.hready := '1';
    end if;

     if (r.hwrite = '1' and r.hit = '1') and (r.hsel = '1') and not (r.resp="10" and r.hready='1') and transfer_mode = '0' and
r.fifo_NF = '0' and fifo_full = '0' and r.wait_c = '0' then
```

```
            fifo_entry <= '1' & r.size & r.addr;
            fifo_write_en <= '1';
     elsif (r.hit = '0') and (r.hsel = '1') and not (r.resp="10" and r.hready='1') and transfer_mode = '0' and fifo_full = '0'
and r.wait_c = '0' and fetch_match = '0' then
            fifo_entry <= "000" & r.addr;
            fifo_write_en <= '1';
            fetch_write_var := '1';
    else
            fifo_entry <= "000" & r.addr;
            fifo_write_en <= '0';
    end if;

    if rst = '0' then v.hwrite := '0'; v.hready := '1'; end if;
    fifo_write_data <= ahbsi.hwdata(line_size-1 downto 0); fetch_write_en <= fetch_write_var; fetch_data_in <= r.addr;
    write <= bs; ramsel <= v.hsel or r.hwrite; ahbso.hready <= r.hready; ahbso.hresp <= r.resp;
    ramaddr <= line_point; c <= v; write_data <= ahbsi.hwdata(line_size-1 downto 0);
    ahbso.hrdata(31 downto line_size) <= (others => '0'); ahbso.hrdata(line_size-1 downto 0) <= ramdata;

  end process;

  ahbso.hsplit  <= (others => '0');
  ahbso.hirq    <= (others => '0');
  ahbso.hcache  <= '1';
  ahbso.hconfig <= hconfig;
  ahbso.hindex  <= hindex;

  reg : process (clk)
  begin
    if rising_edge(clk ) then
            r <= c;
    end if;
  end process;

end;
```

A3.addr_check.vhd

addr_check.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;



entity comparator is
            generic (
                        tag_size : integer
            );
            port (
                        block_tag :           in std_logic_vector(tag_size-1 downto 0);
                        valid :    in std_ulogic;
                        address_tag :        in std_logic_vector(tag_size-1 downto 0);
                        match :   out std_ulogic
            );
end;
```

```
architecture comp_arc of comparator is

begin

        cp : process(block_tag, valid, address_tag)
        begin

                if ( (block_tag&valid) = (address_tag&'1') ) then
                        match <= '1';
                else
                        match <= '0';
                end if;
        end process;
end;





library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity addr_check is
        generic (
                abits : integer;
                offset_bits : integer;
                index_bits : integer;
                block_number : integer;
                set_assoc : integer;
                set_number : integer;
                lines_block : integer;
                lines_number : integer
        );
        port (
                clk :   in std_ulogic;
                rst :           in std_ulogic;
                ahbsi_haddr :           in std_logic_vector(31 downto 0);
                line_pointer :          out integer range 0 to lines_number-1;
                hit :           out std_ulogic;

                tag_in    : in std_logic_vector(abits-offset_bits-index_bits-1 downto 0);
                tag_write_en : in std_ulogic;
                block_num : in integer range 0 to block_number-1
        );
end;


architecture rtl of addr_check is

component comparator is
        generic (
                tag_size : integer
        );
        port (
                block_tag :             in std_logic_vector(tag_size-1 downto 0);
```

```
                        valid :     in std_ulogic;
                        address_tag :       in std_logic_vector(tag_size-1 downto 0);
                        match :   out std_ulogic
            );
end component;


type tags_array_type is array(block_number-1 downto 0) of std_logic_vector(abits-offset_bits-index_bits-1 downto 0);
type block_array is array(set_assoc-1 downto 0) of std_logic_vector(abits-offset_bits-index_bits-1 downto 0);

signal tags_array : tags_array_type;

signal valid_bits : std_logic_vector(block_number-1 downto 0);
signal block_tags_to_cmp : block_array;
signal valid_bits_to_cmp : std_logic_vector(set_assoc-1 downto 0);
signal match_results : std_logic_vector(set_assoc-1 downto 0);
signal chosen_set : integer range 0 to set_number-1;
signal offset : integer range 0 to lines_block-1;
signal cacheaddr : std_logic_vector(abits-1 downto 0);


begin

            cg : for i in 0 to set_assoc-1 generate
                        cmp :  comparator generic map (abits-offset_bits-index_bits) port map (
                                    block_tags_to_cmp(i),           valid_bits_to_cmp(i),           cacheaddr(abits-1           downto
offset_bits+index_bits), match_results(i));
            end generate;

            cacheaddr <= ahbsi_haddr(abits+1 downto 2);

            gen1: if set_number = 1 and lines_block /= 1 generate
                        dex : process(cacheaddr, tags_array, valid_bits)

                        variable chosen_set_var : integer range 0 to set_number-1;
                        variable offset_var : integer range 0 to lines_block-1;

                        begin

                                    chosen_set_var := 0;

                                    offset_var := to_integer(unsigned(cacheaddr(offset_bits-1 downto 0)));

                                    for i in set_assoc-1 downto 0 loop
                                                block_tags_to_cmp(i) <= tags_array(chosen_set_var*set_assoc +i);
                                                valid_bits_to_cmp(i) <= valid_bits(chosen_set_var*set_assoc +i);
                                    end loop;

                                    chosen_set <= chosen_set_var;
                                    offset <= offset_var;

                        end process dex;
            end generate gen1;



            gen2: if lines_block = 1 and set_number /= 1 generate
                        dex : process(cacheaddr, tags_array, valid_bits)

                        variable chosen_set_var : integer range 0 to set_number-1;
```

```
		variable offset_var : integer range 0 to lines_block-1;

		begin

			chosen_set_var := to_integer(unsigned(cacheaddr(offset_bits+index_bits-1 downto
offset_bits)));

			offset_var := 0;

			for i in set_assoc-1 downto 0 loop
				block_tags_to_cmp(i) <= tags_array(chosen_set_var*set_assoc +i);
				valid_bits_to_cmp(i) <= valid_bits(chosen_set_var*set_assoc +i);
			end loop;

			chosen_set <= chosen_set_var;
			offset <= offset_var;

		end process dex;
	end generate gen2;



	gen3: if set_number = 1 and lines_block = 1 generate
		dex : process(cacheaddr, tags_array, valid_bits)

		variable chosen_set_var : integer range 0 to set_number-1;
		variable offset_var : integer range 0 to lines_block-1;

		begin

			chosen_set_var := 0;

			offset_var := 0;

			for i in set_assoc-1 downto 0 loop
				block_tags_to_cmp(i) <= tags_array(chosen_set_var*set_assoc +i);
				valid_bits_to_cmp(i) <= valid_bits(chosen_set_var*set_assoc +i);
			end loop;

			chosen_set <= chosen_set_var;
			offset <= offset_var;

		end process dex;
	end generate gen3;



	gen4: if set_number /= 1 and lines_block /= 1 generate
		dex : process(cacheaddr, tags_array, valid_bits)

		variable chosen_set_var : integer range 0 to set_number-1;
		variable offset_var : integer range 0 to lines_block-1;

		begin

			chosen_set_var := to_integer(unsigned(cacheaddr(offset_bits+index_bits-1 downto
offset_bits)));

			offset_var := to_integer(unsigned(cacheaddr(offset_bits-1 downto 0)));
```

```
                    for i in set_assoc-1 downto 0 loop
                            block_tags_to_cmp(i) <= tags_array(chosen_set_var*set_assoc +i);
                            valid_bits_to_cmp(i) <= valid_bits(chosen_set_var*set_assoc +i);
                    end loop;

                    chosen_set <= chosen_set_var;
                    offset <= offset_var;

            end process dex;
    end generate gen4;



    res: process(match_results, chosen_set, offset)

    variable block_hit_num : integer range 0 to set_assoc-1;
    variable hit_v : std_ulogic;

    begin

            hit_v := '0';  block_hit_num := 0;
            for i in set_assoc-1 downto 0 loop
                    if match_results(i)='1' then
                            hit_v :='1';
                            block_hit_num := i;
                    end if;
            end loop;


            line_pointer <= (chosen_set*set_assoc+block_hit_num)*lines_block + offset;
            hit <= hit_v;

    end process res;


    tag_update: process(clk, rst)
    begin
            if rst = '0' then
                    valid_bits <= (others => '0');
                    tags_array <= (others => (others => '0'));
            elsif (rising_edge(clk)) then
                    if tag_write_en = '1' then
                            tags_array(block_num) <= tag_in;
                            valid_bits(block_num) <= '1';
                    end if;
            end if;

    end process tag_update;


end;
```

A4.cache_ram.vhd

cache_ram.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity single_ram is
  generic (
            lines_number : integer;
            sr_line_size : integer := 8
  );
  port (
    clk      : in std_ulogic;
    rst          : in std_ulogic;
    line_pointer  : in integer range 0 to lines_number-1;
    datain   : in std_logic_vector(sr_line_size-1 downto 0);
    dataout  : out std_logic_vector(sr_line_size-1 downto 0);
    write    : in std_ulogic
  );
end;

architecture behavioral of single_ram is

  type mem is array(lines_number-1 downto 0)
            of std_logic_vector(sr_line_size-1 downto 0);
  signal memarr : mem;
  signal line_p : integer range 0 to lines_number-1;

begin

  main : process(clk, rst)
  begin
    if rst = '0' then
      memarr <= (OTHERS=>(OTHERS=>'0'));
      line_p <= 0;
    elsif rising_edge(clk) then
      if write = '1' then
        memarr(line_pointer) <= datain;
      end if;
      line_p <= line_pointer;
    end if;
  end process;

  dataout <= memarr(line_p);

end;



library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cache_ram is
  generic (
    lines_number : integer;
    line_size : integer := 32
  );
  port (
    clk      : in std_ulogic;
    rst      : in std_ulogic;
```

```
    line_pointer  : in integer range 0 to lines_number-1;
    data_in   : in std_logic_vector(line_size-1 downto 0);
    data_out  : out std_logic_vector(line_size-1 downto 0);
    write_ar    : in std_logic_vector(3 downto 0)
  );
end;

architecture main of cache_ram is

component single_ram is
  generic (
            lines_number : integer;
            sr_line_size : integer := 8
  );
  port (
    clk      : in std_ulogic;
    rst      : in std_ulogic;
    line_pointer  : in integer range 0 to lines_number-1;
    datain   : in std_logic_vector(sr_line_size-1 downto 0);
    dataout  : out std_logic_vector(sr_line_size-1 downto 0);
    write    : in std_ulogic
  );
end component;

begin

  crg : for i in 0 to 3 generate
    aram :  single_ram generic map (lines_number, line_size/4) port map (
            clk, rst, line_pointer, data_in(i*(line_size/4)+((line_size/4) -1) downto i*(line_size/4)),
            data_out(i*(line_size/4)+((line_size/4) -1) downto i*(line_size/4)), write_ar(3-i));
  end generate;

end;
```

A5.fifo.vhd

fifo.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
library nlib;
use nlib.stdlib.all;


-- dff_N_W Entity Description
entity dff_N_W is
        generic(
               N: INTEGER := 8; --length of FIFO
               line_size  : integer;
               abits  : integer
              );
              port(
                        D: in std_logic_vector(abits+4 downto 0);
                        write_data_in: in std_logic_vector(line_size-1 downto 0);
                        Q: out std_logic_vector(abits+4 downto 0);
```

```
                    write_data_out: out std_logic_vector(line_size-1 downto 0);
                    RES, RDCLK, WRCLK, RDEN, WREN: in std_ulogic;
                    F: buffer std_ulogic :='0'; --full
                    E: buffer std_ulogic :='1'; --empty
                    NF: buffer std_ulogic :='0' --near full
          );
end dff_N_W;

-- dff_N_W Architecture Description
architecture rtl of dff_N_W is
          constant L: integer:=log2(N); --ceiling log2(N)

          type rammemory is array (N-1 downto 0) of std_logic_vector(abits+4 downto 0);
          signal ram: rammemory;

          type ramwritedata is array (N-1 downto 0) of std_logic_vector(line_size-1 downto 0);
          signal writedataarray: ramwritedata;

          signal r_bin_next : integer range 0 to N-1; --std_logic_vector(L-1 downto 0);
          signal r_bin : integer range 0 to N-1; --std_logic_vector(L-1 downto 0);
          signal r_gray_next : std_logic_vector(L-1 downto 0);
          signal r_gray : std_logic_vector(L-1 downto 0);

          signal w_bin_next : integer range 0 to N-1; --std_logic_vector(L-1 downto 0);
          signal w_bin : integer range 0 to N-1; --std_logic_vector(L-1 downto 0);
          signal w_gray_next : std_logic_vector(L-1 downto 0);
          signal w_gray : std_logic_vector(L-1 downto 0);
          signal w_gray_nf : std_logic_vector(L-1 downto 0);

          signal empty, full, near_full: std_ulogic;

begin
          Read_Process: process(RES, RDCLK, RDEN, E, r_bin)
          variable r_bin_next_slv : std_logic_vector(L-1 downto 0);
          variable r_inc : std_ulogic;
          variable r_bin_next_var : integer range 0 to N-1;
          variable r_bin_var : integer range 0 to N-1;
          variable r_gray_next_var : std_logic_vector(L-1 downto 0);
          variable r_gray_var : std_logic_vector(L-1 downto 0);
          begin

                    r_inc := RDEN and not E;

                    if (r_inc = '1') then
                              if (r_bin_var = N-1) then
                                        r_bin_next_var := 0;
                              else
                                        r_bin_next_var := r_bin_var + 1;
                              end if;
                    else
                              r_bin_next_var := r_bin_var;
                    end if;

                    r_bin_next_slv := std_logic_vector(to_unsigned(r_bin_next_var,L));
                    r_gray_next_var(L-1) := r_bin_next_slv(L-1);

                    for i in L-2 downto 0 loop
                              r_gray_next_var(i) := r_bin_next_slv(i+1) xor r_bin_next_slv(i);
                    end loop;
```

```
		if (RES='0') then
			r_bin_var:=0;
			r_gray_var:=(OTHERS=>'0');
			Q<=(OTHERS=>'0');
			write_data_out<=(OTHERS=>'0');
		elsif (rising_edge(RDCLK)) then
			if (RDEN='1' and E/='1') then
				Q<=ram(r_bin_var);
				write_data_out<=writedataarray(r_bin_var);
			end if;

			r_bin_var := r_bin_next_var;
			r_gray_var := r_gray_next_var;

		end if;

		r_bin_next <= r_bin_next_var;
		r_bin <= r_bin_var;
		r_gray_next <= r_gray_next_var;
		r_gray <= r_gray_var;


	end process Read_Process;

	Write_Process: process(RES, WRCLK, WREN, F, w_bin)
	variable w_bin_next_slv : std_logic_vector(L-1 downto 0);
	variable w_bin_nf_next_slv : std_logic_vector(L-1 downto 0);
	variable w_inc : std_ulogic;
	variable w_bin_next_var : integer range 0 to N-1;
	variable w_bin_nf_next_var : integer range 0 to N-1;
	variable w_bin_var : integer range 0 to N-1;
	variable w_gray_next_var : std_logic_vector(L-1 downto 0);
	variable w_gray_nf_next_var : std_logic_vector(L-1 downto 0);
	variable w_gray_var : std_logic_vector(L-1 downto 0);
	variable w_gray_nf_var : std_logic_vector(L-1 downto 0);
	begin

		w_inc := WREN and not F;

		if (w_inc = '1') then
			if (w_bin_var = N-1) then
				w_bin_next_var := 0;
			else
				w_bin_next_var := w_bin_var + 1;
			end if;
		else
			w_bin_next_var := w_bin_var;
		end if;

		if (w_bin_next_var = N-1) then
			w_bin_nf_next_var := 0;
		else
			w_bin_nf_next_var := w_bin_next_var + 1;
		end if;

		w_bin_next_slv := std_logic_vector(to_unsigned(w_bin_next_var,L));
		w_gray_next_var(L-1) := w_bin_next_slv(L-1);

		for i in L-2 downto 0 loop
			w_gray_next_var(i) := w_bin_next_slv(i+1) xor w_bin_next_slv(i);
```

```
		end loop;

		w_bin_nf_next_slv := std_logic_vector(to_unsigned(w_bin_nf_next_var,L));
		w_gray_nf_next_var(L-1) := w_bin_nf_next_slv(L-1);

		for i in L-2 downto 0 loop
			w_gray_nf_next_var(i) := w_bin_nf_next_slv(i+1) xor w_bin_nf_next_slv(i);
		end loop;

		if (RES='0') then
			w_bin_var:=0;
			w_gray_var:=(OTHERS=>'0');
			ram<=(OTHERS=>(OTHERS=>'0'));
			writedataarray<=(OTHERS=>(OTHERS=>'0'));
		elsif (rising_edge(WRCLK)) then
			if (WREN='1' and F/='1') then
				ram(w_bin_var)<=D;
				writedataarray(w_bin_var)<=write_data_in;
			end if;

			w_bin_var := w_bin_next_var;
			w_gray_var := w_gray_next_var;
			w_gray_nf_var := w_gray_nf_next_var;

		end if;

		w_bin_next <= w_bin_next_var;
		w_bin <= w_bin_var;
		w_gray_next <= w_gray_next_var;
		w_gray <= w_gray_var;
		w_gray_nf <= w_gray_nf_var;

	end process Write_Process;

	Imd_E_F_Process:process(RES, r_gray, w_gray, w_gray_nf)
	variable dir_set, dir_reset, direction: std_ulogic;
	begin
		dir_set := (w_gray(L-1) xor r_gray(L-2)) and not (r_gray(L-1) xor w_gray(L-2));
		dir_reset := (not (w_gray(L-1) xor r_gray(L-2)) and (r_gray(L-1) xor w_gray(L-2))) or not RES;

		if (dir_reset = '1') then
			direction := '0';
		elsif (dir_set = '1') then
			direction := '1';
		end if;

		if (r_gray=w_gray) then
			if direction='0' then
				empty <= '1';
				full <= '0';
			else
				full <= '1';
				empty <= '0';
			end if;
		else
			empty <= '0';
			full <= '0';
		end if;

		if (r_gray=w_gray_nf) then
```

```
                        near_full <= '1';
                else
                        near_full <= '0';
                end if;

        end process Imd_E_F_Process;

        E_F_Update: process(RDCLK, WRCLK, empty, full, near_full, RES)
        variable rempty, rempty2, wfull, wfull2, wnearfull, wnearfull2: std_ulogic;
        begin
                if (RES = '0') then
                        rempty := '1';
                        rempty2 := '1';
                elsif (empty = '1') then
                        rempty := '1';
                        rempty2 := '1';
                elsif rising_edge(RDCLK) then
                        rempty := rempty2;
                        rempty2 := empty;
                end if;

                if (RES = '0') then
                        wfull := '0';
                        wfull2 := '0';
                elsif (full = '1') then
                        wfull := '1';
                        wfull2 := '1';
                elsif rising_edge(WRCLK) then
                        wfull := wfull2;
                        wfull2 := full;
                end if;

                if (RES = '0') then
                        wnearfull := '0';
                        wnearfull2 := '0';
                elsif (near_full = '1') then
                        wnearfull := '1';
                        wnearfull2 := '1';
                elsif rising_edge(WRCLK) then
                        wnearfull := wnearfull2;
                        wnearfull2 := near_full;
                end if;

                E<=rempty;
                F<=wfull;
                NF<=wnearfull;

        end process E_F_Update;

end rtl;
```

A6.ahb_mst_cache.vhd

ahb_mst_cache.vhd
library ieee;

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library nlib;
use nlib.amba.all;
use nlib.stdlib.all;
use nlib.devices.all;

entity ahbmst is
  generic (
    hindex  : integer := 0;
    venid   : integer := 0;
    devid   : integer := 0;
    version : integer := 0;
    chprot  : integer := 3;
    abits   : integer;
    offset_bits : integer;
    lines_block : integer;
    line_size : integer );
   port (
      rst  : in  std_ulogic;
      clk  : in  std_ulogic;
      ahbi : in  ahb_mst_in_type;
      ahbo : out ahb_mst_out_type;

      fifo_empty : in std_ulogic := '1';
      fifo_read_en : out std_ulogic := '0';
      fifo_data : in std_logic_vector(abits+4 downto 0);
      fifo_write_data : in std_logic_vector(line_size-1 downto 0);

      buffer_empty    : in std_ulogic := '1';
      buffer_write_en : out std_ulogic := '0';
      buffer_data_in  : out std_logic_vector(line_size-1 downto 0);

      tag_addr_in  : out std_logic_vector(abits-1 downto 0);
      tag_addr_write_en : out std_ulogic
      );
end;

architecture rtl of ahbmst is

constant hconfig : ahb_config_type := (
  0 => ahb_device_reg ( venid, devid, 0, version, 0),
  others => zero32);

type reg_type is record
  grant   : std_ulogic;
  active  : std_ulogic;
  load    : std_ulogic;
  write_data : std_logic_vector(line_size-1 downto 0);
  addr_cntr : integer range 0 to lines_block-1;
  read_cntr : integer range 0 to lines_block-1;
  incaddr : std_logic_vector(abits+1 downto 0);
  hbusreq : std_ulogic;
  task    : std_ulogic;
  htrans  : std_logic_vector(1 downto 0);
end record;

signal r, rin : reg_type;

begin
```

```
gen1: if lines_block = 1 generate
        comb : process(ahbi, rst, r, clk, fifo_empty, fifo_data, fifo_write_data, buffer_empty)
        variable v       : reg_type;
        variable hwrite  : std_ulogic;
        variable hprot   : std_logic_vector(3 downto 0);
        variable xhirq    : std_logic_vector(NAHBIRQ-1 downto 0);
        variable size    : std_logic_vector(1 downto 0);

        variable fifo_read_var : std_ulogic;
        variable haddr : std_logic_vector(abits+1 downto 0);
        variable buffer_write_var : std_ulogic;

        variable tag_addr_var : std_logic_vector(abits-1 downto 0);
        variable tag_addr_write_var : std_ulogic;

        begin

                v := r; v.load := '0'; fifo_read_var := '0'; buffer_write_var := '0';
                hprot := std_logic_vector(to_unsigned(chprot, 4));
                xhirq := (others => '0');

                tag_addr_var := (others => '0');
                tag_addr_write_var := '0';
                v.write_data := fifo_write_data;
                hwrite  := fifo_data(abits+4);
                size   := fifo_data(abits+3 downto abits+2);


                if (buffer_empty = '1') and (fifo_empty = '0') and (r.load = '0') and (r.read_cntr = 0) and (r.task = '0')
then
                        v.load := '1';
                        fifo_read_var := '1';
                        v.hbusreq := '1';
                        v.task := '1';
                end if;

                if r.load = '1' then
                        if hwrite = '1' then
                        haddr := fifo_data(abits+1 downto 0);
                        v.addr_cntr := 0;
                        v.read_cntr := 0;
                        else
                                haddr := fifo_data(abits+1 downto 0);
                                tag_addr_var := haddr(abits+1 downto 2);
                                tag_addr_write_var := '1';
                                v.addr_cntr := lines_block - 1;
                                v.read_cntr := lines_block - 1;
                        end if;
                else
                        haddr := r.incaddr;
                end if;

                if r.active = '1' then
                        if ahbi.hready = '1' then
                        if hwrite = '0' then
                                buffer_write_var := '1';
                        end if;
                        if r.read_cntr /= 0 then
                                v.read_cntr := r.read_cntr - 1;
```

```
			else
				v.read_cntr := r.read_cntr; hwrite := '0'; v.task := '0';
			end if;
			else
				v.read_cntr := r.read_cntr;
			end if;
		end if;

		if ahbi.hready = '1' then
			v.grant := ahbi.hgrant(hindex);
			v.active := r.grant;
		end if;

		if v.active = '1' and ahbi.hready = '1' then
			if hwrite = '0' then
			v.incaddr := haddr + "100";
			else
				v.incaddr := haddr;
			end if;
			if r.addr_cntr /= 0 then
			v.addr_cntr := r.addr_cntr - 1;
			else
			v.addr_cntr := r.addr_cntr; -- hwrite := '0';
			end if;
		else
			v.incaddr := haddr;
		end if;

		if ahbi.hgrant(hindex) = '1' and r.hbusreq = '0' then
			v.htrans := HTRANS_IDLE;
			v.grant  := '0';
		else
			v.htrans := HTRANS_NONSEQ;
		end if;

		if v.grant = '1' and v.addr_cntr = 0 then
			v.hbusreq := '0';
		end if;

		if rst = '0' then v.active := '0'; v.addr_cntr := 0; v.read_cntr := 0; v.task := '0'; end if;

		rin <= v;

		ahbo.haddr(abits+1 downto 0) <= haddr;
		ahbo.haddr(31 downto abits+2) <= (others => '0');
		ahbo.htrans  <= r.htrans;
		ahbo.hbusreq <= v.hbusreq;
		ahbo.hwdata(31 downto line_size) <= (others => '0');
		ahbo.hwdata(line_size-1 downto 0) <= r.write_data;
		ahbo.hconfig <= hconfig;
		ahbo.hlock   <= '0';
		ahbo.hwrite  <= hwrite;
		ahbo.hsize   <= '0' & size;
		ahbo.hburst  <= HBURST_SINGLE;
		ahbo.hprot   <= hprot;
		ahbo.hirq    <= xhirq;
		ahbo.hindex  <= hindex;

		fifo_read_en  <= fifo_read_var;
```

```
			buffer_write_en  <= buffer_write_var;
			buffer_data_in   <= ahbi.hrdata(line_size-1 downto 0);

			tag_addr_write_en <= tag_addr_write_var;
			tag_addr_in   <= tag_addr_var;


		end process;
  end generate gen1;



  gen2: if lines_block /= 1 generate
		comb : process(ahbi, rst, r, clk, fifo_empty, fifo_data, fifo_write_data, buffer_empty)
		variable v       : reg_type;
		variable hwrite  : std_ulogic;
		variable hprot   : std_logic_vector(3 downto 0);
		variable xhirq    : std_logic_vector(NAHBIRQ-1 downto 0);
		variable size    : std_logic_vector(1 downto 0);

		variable fifo_read_var : std_ulogic;
		variable haddr : std_logic_vector(abits+1 downto 0);
		variable buffer_write_var : std_ulogic;

		variable tag_addr_var : std_logic_vector(abits-1 downto 0);
		variable tag_addr_write_var : std_ulogic;

		begin

			v := r; v.load := '0'; fifo_read_var := '0'; buffer_write_var := '0';
			hprot := std_logic_vector(to_unsigned(chprot, 4));
			xhirq := (others => '0');

			tag_addr_var := (others => '0');
			tag_addr_write_var := '0';
			v.write_data := fifo_write_data;
			hwrite  := fifo_data(abits+4);
			size   := fifo_data(abits+3 downto abits+2);


			if (buffer_empty = '1') and (fifo_empty = '0') and (r.load = '0') and (r.read_cntr = 0) and (r.task = '0')
then
				v.load := '1';
				fifo_read_var := '1';
				v.hbusreq := '1';
				v.task := '1';
			end if;

			if r.load = '1' then
				if hwrite = '1' then
				haddr := fifo_data(abits+1 downto 0);
				v.addr_cntr := 0;
				v.read_cntr := 0;
				--	v.transfer := '1';
				else
					haddr := fifo_data(abits+1 downto 0);
					haddr(offset_bits+1 downto 2) := (others => '0');
					tag_addr_var := haddr(abits+1 downto 2);
					tag_addr_write_var := '1';
					v.addr_cntr := lines_block - 1;
```

```
                    v.read_cntr := lines_block - 1;
          end if;
else
          haddr := r.incaddr;
end if;

if r.active = '1' then
          if ahbi.hready = '1' then
          if hwrite = '0' then
                    buffer_write_var := '1';
          end if;
          if r.read_cntr /= 0 then
                    v.read_cntr := r.read_cntr - 1;
          else
                    v.read_cntr := r.read_cntr; hwrite := '0'; v.task := '0';
          end if;
          else
                    v.read_cntr := r.read_cntr;
          end if;
end if;

if ahbi.hready = '1' then
          v.grant := ahbi.hgrant(hindex);
          v.active := r.grant;
end if;

if v.active = '1' and ahbi.hready = '1' then
          if hwrite = '0' then
          v.incaddr := haddr + "100";
          else
                    v.incaddr := haddr;
          end if;
          if r.addr_cntr /= 0 then
          v.addr_cntr := r.addr_cntr - 1;
          else
          v.addr_cntr := r.addr_cntr; -- hwrite := '0';
          end if;
else
          v.incaddr := haddr;
end if;

if ahbi.hgrant(hindex) = '1' and r.hbusreq = '0' then
          v.htrans := HTRANS_IDLE;
          v.grant  := '0';
else
          v.htrans := HTRANS_NONSEQ;
end if;

if v.grant = '1' and v.addr_cntr = 0 then
          v.hbusreq := '0';
end if;

if rst = '0' then v.active := '0'; v.addr_cntr := 0; v.read_cntr := 0; v.task := '0'; end if;

rin <= v;

ahbo.haddr(abits+1 downto 0) <= haddr;
ahbo.haddr(31 downto abits+2) <= (others => '0');
ahbo.htrans  <= r.htrans;
ahbo.hbusreq <= v.hbusreq;
```

```
            ahbo.hwdata(31 downto line_size) <= (others => '0');
            ahbo.hwdata(line_size-1 downto 0) <= r.write_data;
            ahbo.hconfig <= hconfig;
            ahbo.hlock   <= '0';
            ahbo.hwrite  <= hwrite;
            ahbo.hsize   <= '0' & size;
            ahbo.hburst  <= HBURST_SINGLE;
            ahbo.hprot   <= hprot;
            ahbo.hirq    <= xhirq;
            ahbo.hindex  <= hindex;

            fifo_read_en  <= fifo_read_var;

            buffer_write_en  <= buffer_write_var;
            buffer_data_in   <= ahbi.hrdata(line_size-1 downto 0);

            tag_addr_write_en <= tag_addr_write_var;
            tag_addr_in   <= tag_addr_var;


      end process;
  end generate gen2;



  regs : process(clk, rst)
  begin

    if rising_edge(clk) then
      r <= rin;
    end if;

    if (rst = '0') then
      r.active <= '0';
      r.grant <= '0';
      r.task <= '0';
      r.hbusreq <= '0';
      r.htrans <= "00";
      r.incaddr <= (others => '0');
    end if;

  end process;

end;
```

A7.buffer.vhd

buffer.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
library nlib;
use nlib.stdlib.all;

-- buf Entity Description
```

```
entity buf is
	generic(
		line_size : INTEGER;
		N: INTEGER := 8
		);
		port(
			D: in std_logic_vector(line_size-1 downto 0);
			Q: out std_logic_vector(line_size-1 downto 0);
			RES, RDCLK, WRCLK, RDEN, WREN: in std_ulogic;
			trl_F: buffer std_ulogic :='0';
			trl_E: buffer std_ulogic :='1';
			mst_E:buffer std_ulogic :='1'
		);
end buf;

-- buf Architecture Description
architecture rtl of buf is

		constant L: integer:=log2(N);

		type rammemory is array (N-1 downto 0) of std_logic_vector(line_size-1 downto 0);
		signal ram: rammemory;

		signal r_bin_next : integer range 0 to N-1;
		signal r_bin : integer range 0 to N-1;
		signal r_gray_next : std_logic_vector(L-1 downto 0);
		signal r_gray : std_logic_vector(L-1 downto 0);

		signal w_bin_next : integer range 0 to N-1;
		signal w_bin : integer range 0 to N-1;
		signal w_gray_next : std_logic_vector(L-1 downto 0);
		signal w_gray : std_logic_vector(L-1 downto 0);

		signal empty, full: std_ulogic;

begin
		Read_Process: process(RES, RDCLK, RDEN, empty, r_bin)
		variable r_bin_next_slv : std_logic_vector(L-1 downto 0);
		variable r_inc : std_ulogic;
		variable r_bin_next_var : integer range 0 to N-1;
		variable r_bin_var : integer range 0 to N-1;
		variable r_gray_next_var : std_logic_vector(L-1 downto 0);
		variable r_gray_var : std_logic_vector(L-1 downto 0);
		begin

			r_inc := RDEN and not empty;

			if (r_inc = '1') then
				if (r_bin_var = N-1) then
					r_bin_next_var := 0;
				else
					r_bin_next_var := r_bin_var + 1;
				end if;
			else
				r_bin_next_var := r_bin_var;
			end if;

			r_bin_next_slv := std_logic_vector(to_unsigned(r_bin_next_var,L));
			r_gray_next_var(L-1) := r_bin_next_slv(L-1);
```

```
		for i in L-2 downto 0 loop
			r_gray_next_var(i) := r_bin_next_slv(i+1) xor r_bin_next_slv(i);
		end loop;

		if (RES='0') then
			r_bin_var:=0;
			r_gray_var:=(OTHERS=>'0');
			Q<=(OTHERS=>'0');
		elsif (rising_edge(RDCLK)) then
			if (RDEN='1' and empty/='1') then
				Q<=ram(r_bin_var);
			end if;

			r_bin_var := r_bin_next_var;
			r_gray_var := r_gray_next_var;

		end if;

		r_bin_next <= r_bin_next_var;
		r_bin <= r_bin_var;
		r_gray_next <= r_gray_next_var;
		r_gray <= r_gray_var;


	end process Read_Process;

	Write_Process: process(RES, WRCLK, WREN, full, w_bin)
	variable w_bin_next_slv : std_logic_vector(L-1 downto 0);
	variable w_inc : std_ulogic;
	variable w_bin_next_var : integer range 0 to N-1;
	variable w_bin_var : integer range 0 to N-1;
	variable w_gray_next_var : std_logic_vector(L-1 downto 0);
	variable w_gray_var : std_logic_vector(L-1 downto 0);
	begin

		w_inc := WREN and not full;

		if (w_inc = '1') then
			if (w_bin_var = N-1) then
				w_bin_next_var := 0;
			else
				w_bin_next_var := w_bin_var + 1;
			end if;
		else
			w_bin_next_var := w_bin_var;
		end if;

		w_bin_next_slv := std_logic_vector(to_unsigned(w_bin_next_var,L));
		w_gray_next_var(L-1) := w_bin_next_slv(L-1);

		for i in L-2 downto 0 loop
			w_gray_next_var(i) := w_bin_next_slv(i+1) xor w_bin_next_slv(i);
		end loop;

		if (RES='0') then
			w_bin_var:=0;
			w_gray_var:=(OTHERS=>'0');
			ram<=(OTHERS=>(OTHERS=>'0'));
		elsif (rising_edge(WRCLK)) then
			if (WREN='1' and full/='1') then
```

```
                        ram(w_bin_var)<=D;
                end if;

                w_bin_var := w_bin_next_var;
                w_gray_var := w_gray_next_var;

        end if;

        w_bin_next <= w_bin_next_var;
        w_bin <= w_bin_var;
        w_gray_next <= w_gray_next_var;
        w_gray <= w_gray_var;

end process Write_Process;

Imd_E_F_Process:process(RES, r_gray, w_gray)
variable dir_set, dir_reset, direction: std_ulogic;
begin
        dir_set := (w_gray(L-1) xor r_gray(L-2)) and not (r_gray(L-1) xor w_gray(L-2));
        dir_reset := (not (w_gray(L-1) xor r_gray(L-2)) and (r_gray(L-1) xor w_gray(L-2))) or not RES;

        if (dir_reset = '1') then
                direction := '0';
        elsif (dir_set = '1') then
                direction := '1';
        end if;

        if (r_gray=w_gray) then
                if direction='0' then
                        empty <= '1';
                        full <= '0';
                else
                        full <= '1';
                        empty <= '0';
                end if;
        else
                empty <= '0';
                full <= '0';
        end if;

end process Imd_E_F_Process;

E_F_Update: process(RDCLK, WRCLK, empty, full, RES)
variable mstempty, mstempty2, trlfull, trlfull2, trlempty, trlempty2: std_ulogic;
begin
        if (RES = '0') then
                mstempty := '1';
                mstempty2 := '1';
        elsif (empty = '0') then
                mstempty := '0';
                mstempty2 := '0';
        elsif rising_edge(WRCLK) then
                mstempty := mstempty2;
                mstempty2 := empty;
        end if;

        if (RES = '0') then
                trlfull := '0';
                trlfull2 := '0';
        elsif (full = '0') then
```

```
                    trlfull := '0';
                    trlfull2 := '0';
            elsif rising_edge(RDCLK) then
                    trlfull := trlfull2;
                    trlfull2 := full;
            end if;

            if (RES = '0') then
                    trlempty := '0';
                    trlempty2 := '0';
            elsif (empty = '1') then
                    trlempty := '1';
                    trlempty2 := '1';
            elsif rising_edge(RDCLK) then
                    trlempty := trlempty2;
                    trlempty2 := empty;
            end if;

            mst_E <= mstempty;
            trl_F <= trlfull;
            trl_E <= trlempty;

    end process E_F_Update;

end rtl;
```

A8.lfsr.vhd

lfsr.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library nlib;
use nlib.stdlib.all;

entity lfsr is
  generic (
    set_assoc: integer
  );
  port (
    clk : in std_ulogic;
    rst : in std_ulogic;
    random_int : out integer range 0 to set_assoc-1
  );
end;

architecture main of lfsr is

type lfsr_taps_type is array (1 to 4) of integer range 0 to 23;
type lfsr_taps_table_type is array (4 to 23) of lfsr_taps_type;

constant lfsr_taps_table : lfsr_taps_table_type := (
(4, 3, 0, 0), (5, 3, 0, 0), (6, 5, 0, 0), (7, 6, 0, 0),
(8, 6, 5, 4), (9, 5, 0, 0), (10, 7, 0, 0), (11, 9, 0, 0),
(12, 6, 4, 1), (13, 4, 3, 1), (14, 5, 3, 1), (15, 14, 0, 0),
```

```
(16, 15, 13, 4), (17, 14, 0, 0), (18, 11, 0, 0), (19, 6, 2, 1),
(20, 17, 0, 0), (21, 19, 0, 0), (22, 21, 0, 0), (23, 18, 0, 0));

constant lfsr_length : integer := log2(set_assoc)+3;

begin

        gen1: if set_assoc = 1 generate
                mf: process(rst, clk)

                variable random_int_var : integer range 0 to set_assoc-1;

                begin

                        random_int_var := 0;

                        random_int <= random_int_var;

                end process mf;
        end generate gen1;



        gen2: if set_assoc /= 1 generate
                mf: process(rst, clk)

                variable lfsr_var : std_logic_vector(1 to lfsr_length);
                variable random_int_var : integer range 0 to set_assoc-1;
                variable feedback : std_ulogic;

                begin

                        if rst = '0' then
                                lfsr_var := (others => '1');
                        elsif falling_edge(clk) then
                                feedback := '0';
                                for tap in 1 to 4 loop
                                        if lfsr_taps_table(lfsr_length)(tap) /= 0 then
                                                feedback := feedback xor lfsr_var(lfsr_taps_table(lfsr_length)
(tap));
                                        end if;
                                end loop;
                                lfsr_var := feedback & lfsr_var(1 to lfsr_length-1);
                        end if;

                        random_int_var := to_integer(unsigned(lfsr_var(4 to lfsr_length)));
                        random_int <= random_int_var;

                end process mf;
        end generate gen2;

end;
```

A9.transfer_logic.vhd

transfer_logic.vhd

```
library ieee;
use ieee.math_real.all;
use ieee.numeric_std.all;
use IEEE.std_logic_1164.all;
library nlib;
use nlib.amba.all;

entity transfer_logic is
  generic (
    hindex  : integer := 0;
    abits  : integer;
    offset_bits : integer;
    index_bits : integer;
    line_size : integer := 32;
    set_number : integer := 1;
    block_number : integer;
    lines_number : integer;
    set_assoc : integer ;
    lines_block : integer
  );
  port (
    --addr_check ports
    tag     : out std_logic_vector(abits-offset_bits-index_bits-1 downto 0);
    tag_write_en : out std_ulogic;
    block_num : out integer range 0 to block_number-1;

    --multiplexor (and cache_ram) ports (transfer_m goes to ahb_slv_cache too)
    line_pointer  : out integer range 0 to lines_number-1;
    write_data   : out std_logic_vector(line_size-1 downto 0);
    write_ar    : out std_logic_vector(3 downto 0);
    transfer_m  : out std_ulogic;

    --amba bus ports
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    clk_mst : in  std_ulogic;
    not_idle: in  std_ulogic;
    ahbsi_hsel : in std_logic_vector(0 to NAHBSLV-1);

    --lfsr ports
    random_int_in : in integer range 0 to set_assoc-1;

    --buffer ports
    buffer_empty    : in std_ulogic := '1';
    buffer_full     : in std_ulogic := '0';
    buffer_read_en : out std_ulogic := '0';
    buffer_data  : in std_logic_vector(line_size-1 downto 0);

    --ahb_mst_cache ports
    addr_in  : in std_logic_vector(abits-1 downto 0);
    addr_write_en : in std_ulogic
  );
end transfer_logic;

architecture main of transfer_logic is

signal r_transfer_mode, rin_transfer_mode : std_ulogic;
```

```
signal chosen_set : integer range 0 to set_number-1;
signal random_block : integer range 0 to set_assoc-1;

begin

    gen1: if set_number = 1 generate
        Read_Process: process(rst, clk_mst)

        variable chosen_set_var : integer range 0 to set_number-1;
        variable tag_var : std_logic_vector(abits-offset_bits-index_bits-1 downto 0);
        variable random_block_var : integer range 0 to set_assoc-1;
        variable address : std_logic_vector(abits-1 downto 0);

        begin
            if (rst='0') then
                address:=(OTHERS=>'0');
            elsif (falling_edge(clk_mst)) then
                if addr_write_en = '1' then
                    address:=addr_in;
                    random_block_var := random_int_in;
                end if;
            end if;

            tag_var := address(abits-1 downto offset_bits+index_bits);

            chosen_set_var := 0;

            chosen_set <= chosen_set_var;
            random_block <= random_block_var;

            tag <= tag_var;
            block_num <= chosen_set_var*set_assoc + random_block_var;

        end process Read_Process;
    end generate gen1;



    gen2: if set_number /= 1 generate
        Read_Process: process(rst, clk_mst)

        variable chosen_set_var : integer range 0 to set_number-1;
        variable tag_var : std_logic_vector(abits-offset_bits-index_bits-1 downto 0);
        variable random_block_var : integer range 0 to set_assoc-1;
        variable address : std_logic_vector(abits-1 downto 0);

        begin
            if (rst='0') then
                address:=(OTHERS=>'0');
            elsif (falling_edge(clk_mst)) then
                if addr_write_en = '1' then
                    address:=addr_in;
                    random_block_var := random_int_in;
                end if;
            end if;

            tag_var := address(abits-1 downto offset_bits+index_bits);

            chosen_set_var     :=     to_integer(unsigned(address(offset_bits+index_bits-1     downto
offset_bits)));
```

```
				chosen_set <= chosen_set_var;
				random_block <= random_block_var;

				tag <= tag_var;
				block_num <= chosen_set_var*set_assoc + random_block_var;

		end process Read_Process;
end generate gen2;



Transfer_Pr: process(clk, r_transfer_mode, buffer_empty, buffer_full, buffer_data, not_idle)

variable v_transfer_mode : std_ulogic;
variable buffer_read_var : std_ulogic;
variable tag_write_var   : std_ulogic;
variable write_ar_var    : std_logic_vector(3 downto 0);
variable line_p_var      : integer range 0 to lines_number;
variable data            : std_logic_vector(line_size-1 downto 0);
variable not_idle_var    : std_ulogic;
variable not_selected    : std_ulogic;

begin
		v_transfer_mode := r_transfer_mode; tag_write_var := '0';

		if rising_edge(clk) then
				not_idle_var := not_idle;
				not_selected := ahbsi_hsel(hindex);
		end if;

		if buffer_full = '1' and (not_idle_var = '0' or not_selected = '0') and r_transfer_mode = '0' then
				v_transfer_mode := '1';
				tag_write_var := '1';
		elsif buffer_empty = '1' then
				v_transfer_mode := '0';
		end if;

		if v_transfer_mode = '1' then
				buffer_read_var := '1';
		else
				buffer_read_var := '0';
		end if;

		if falling_edge(clk) then
				if r_transfer_mode = '0' then
						line_p_var := (chosen_set*set_assoc+random_block)*lines_block;
						write_ar_var := "0000";
				else
						line_p_var   := line_p_var +1;
						write_ar_var := "1111";
				end if;

		end if;

		buffer_read_en <= buffer_read_var;
		data := buffer_data;

		if (line_p_var = 0) then
				line_pointer <= 0;
```

```
                else
                        line_pointer <= line_p_var -1;
                end if;

                write_data <= data;
                write_ar <= write_ar_var;
                transfer_m <= v_transfer_mode or r_transfer_mode;

                tag_write_en <= tag_write_var;

                rin_transfer_mode <= v_transfer_mode;

        end process Transfer_Pr;

        regs : process(clk)
        begin if rising_edge(clk) then r_transfer_mode <= rin_transfer_mode; end if; end process;

end;
```

A10.multiplexor.vhd

multiplexor.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity multiplexor is
  generic (
    line_size : integer := 32;
    lines_number : integer
  );
  port (

    --transfer_logic ports
    tl_line_pointer  : in integer range 0 to lines_number-1;
    tl_write_data   : in std_logic_vector(line_size-1 downto 0);
    tl_write_ar    : in std_logic_vector(3 downto 0);
    transfer_mode  : in std_ulogic;

    --ahb_slv_chache ports
    asc_line_pointer  : in integer range 0 to lines_number-1;
    asc_write_data   : in std_logic_vector(line_size-1 downto 0);
    asc_write_ar    : in std_logic_vector(3 downto 0);

    --cache_ram ports
    cr_line_pointer  : out integer range 0 to lines_number-1;
    cr_write_data   : out std_logic_vector(line_size-1 downto 0);
    cr_write_ar    : out std_logic_vector(3 downto 0)
  );
end;

architecture rtl of multiplexor is

begin
```

```
        main_function:    process(tl_line_pointer,    tl_write_data,    tl_write_ar,    transfer_mode,    asc_line_pointer,
asc_write_data, asc_write_ar)
        begin
                if transfer_mode = '0' then
                        cr_line_pointer <= asc_line_pointer;
                        cr_write_data   <= asc_write_data;
                        cr_write_ar     <= asc_write_ar;
                else
                        cr_line_pointer <= tl_line_pointer;
                        cr_write_data   <= tl_write_data;
                        cr_write_ar     <= tl_write_ar;
                end if;
        end process main_function;

end;
```

A11.fetch_list.vhd

fetch_list.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fetch_list is
    generic(
        N: INTEGER := 8;
        abits  : integer := 8;
        offset_bits : integer
        );
        port(
                D: in std_logic_vector(abits+1 downto 0);
                rst, clk, rden, wren: in std_ulogic;
                match : out std_ulogic
        );
end fetch_list;


architecture main of fetch_list is

        type rammemory is array (N-1 downto 0) of std_logic_vector(abits-offset_bits downto 0);
        signal ram: rammemory;

        signal r_point : integer range 0 to N-1;
        signal w_point : integer range 0 to N-1;



begin

        fl_update: process(rst, clk, rden, wren)
        variable r_point_var : integer range 0 to N-1;
        variable w_point_var : integer range 0 to N-1;
        begin

                if (rst = '0') then
                        ram <= (others=>(others=>'0'));
```

```
                r_point_var := 0;
                w_point_var := 0;
        elsif (rising_edge(clk)) then
                if (wren = '1') then
                        ram(w_point_var) <= D(abits+1 downto offset_bits+2)&'1';
                        if (w_point_var = N-1) then
                                w_point_var := 0;
                        else
                                w_point_var := w_point_var + 1;
                        end if;
                elsif (rden = '1') then
                        ram(r_point_var)(0) <= '0';
                        if (r_point_var = N-1) then
                                r_point_var := 0;
                        else
                                r_point_var := r_point_var + 1;
                        end if;
                end if;
        end if;

        r_point <= r_point_var;
        w_point <= w_point_var;

    end process fl_update;

    match_pr: process(D, ram)
    variable match_var : std_ulogic;
    begin
        match_var := '0';

        for i in N-1 downto 0 loop
                if (D(abits+1 downto offset_bits+2)&'1' = ram(i)) then
                        match_var := '1';
                end if;
        end loop;

        match <= match_var;

    end process match_pr;

end main;
```

Βιβλιογραφία

[1] Dimitrios Lioupis, Andreas Adamidis, Nikolaos Theoharis, "*Simulating SiScape: A Parallel CMP Architecture*", NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2008), June 2008

[2] Dimitrios Lioupis, Stefanos Kaxiras, Vagelis Kalampalikhs, Andreas Adamidis, Nikos Theoharis, "*Regenerative Chip-Multiprocessor for Space Applications*", Data Systems In Aerospace (DASIA) conference 2007

[3] D. Lioupis, N. Kanellopoulos, "CHESS Multiprocessor A Processor-Memory Grid for Parallel Programming", Cache and Interconnect Architectures in Multiprocessors, edited by Michael Dubois and Shreekant Thakkar. Kluwer Academic Publishers, pp 245-257, June 1990.

[4] D. Lioupis, N. Kanellopoulos and M. Stefanidakis, "The Memory Hierarchy of the CHESS Computer", Microprocessors and Microprogramming (JSA) Vol 38, pp 99-107, Sep 1993.

[5] D. Lioupis, M. Stefanidakis, "Dynamic Load Balancing on a Virtually-Shared Memory Parallel Computer System", Proceedings of the 6th International PARLE Conference, pp 813-819, (PARLE '94 Parallel Architectures and Languages Europe), Athens 1994.

[6] D. Lioupis, "PiSMA: Parallel Architecture for Video on Demand", Scientific Computing, issue 28, pp 10-11, May 1997.

[7] D. Lioupis, A. Pipis, M. Stefanidakis, "PiSMA: An Upgradeable Fault Tolerant Approach to Parallel Processing", Proceedings of the fourth IEEE International Conference on High Performance Computing (HiPC97), pp 277-283, December 1997, Bangalore India.

[8] A. Pipis, G.Theodoropoulos, M. Stefanidakis, D.Lioupis "Efficient Modeling and Simulation of a Virtually Shared Memory Parallel Architecture", in the 3rd International Congress of the Federation of EUROpean SIMulation Societies (Eurosim98), April 1998, Helsinki, Finland.

[9] D Lioupis, A. Pipis, M. Smirli, M. Stefanidakis, "PiSMA: A Parallel VSM Architecture", ACM Crossroads Magazine, Spring 1999 - issue 5.3, pp 11-14.

[10] Dimitris Lioupis, Andreas Pipis, Michael Stefanidakis, "A Low Overhead Run Time Support System for a Parallel VSM Architecture", in the 7th International Conference on High Performance Computing and Networking Europe (HPCN Europe 99), April 12-14, 1999, Amsterdam, The Netherlands.

[11] Dimitris Lioupis, Andreas Pipis, Michael Stefanidakis, "A Run Time Support System for a Parallel Multimedia Server", Proceedings of the 3th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA99), pp 151-156, October 1999, Nassau, The Bahamas.

[12] Andreas Pipis, George Theodoropoulos, Michael Stefanidakis, Dimitris Lioupis, "A Hybrid Approach for the Modelling and Simulation of a Virtually Shared Memory Parallel Computer Architecture", Mathematics and Computers in Simulation Journal, Transactions of the International Association for Mathematics and Computers in Simulation (IMACS), Vol 57 (1-2), August 2001, pp 81-93, Published by Elsevier Science B.V.

[13] John L. Hennessy, David A. Patterson, “Computer Architecture: A Quantitative Approach”, Second Edition, 1996, Morgan Kaufmann Publishers, Inc., San Francisco, California.

[14] AMBA Specification (Rev 2.0), http://www.arm.com

[15] Jiri Gaisler, Edvin Catovic, Marko Isomäki, Kristoffer Glembo, Sandi Habinc, “GRLIB IP Core User’s Manual”, Gaisler Research, 2007.

[16] Jiri Gaisler, Sandi Habinc, Edvin Catovic, “GRLIB IP Library User’s Manual”, Gaisler Research, 2007.
[17] D. M. Chapiro, “Globally-Asynchronous Locally-Synchronous Systems”, Ph.D. thesis, Stanford University, Stanford, CA, USA, 1984.
[18] I. Sutherland and S. Fairbanks, “GasP: A minimal FIFO control”, in Advanced Research in Asynchronous Circuits and Systems, Mar. 2001, pp. 46-53.
[19] J. T. Yantchev, C. G. Huang, M. B. Josephs, and I. M. Nedelchev, “Low-latency asynchronous FIFO buffers”, in Proc. Asynchronous Design Methodologies, May 1995.
[20] M. Pechoucek, “Anamolous response times of input synchronizers”, in IEEE Journal of Solid-State Circuits, Feb. 1976, vol. 25, pp. 133-139.
[21] I. Soderquist, “Globally updated mesochronous design style”, in IEEE Journal of Solid-State Circuits, July 2003, pp. 1242-1249.
[22] R. Ginosar and R. Kol, “Adaptive synchronization”, in IEEE International Conference on Computer Design, Oct. 1998, pp. 188 -189.
[23] M. Bolton, “A guided tour of 35 years of metastability research”, in Western Electronic Show and Convention (WESCON), Program Session 16, 1987.
[24] T. J. Chaney and C. E. Molnar, “Anomalous behavior of synchronizer and arbiter circuits”, in IEEE Transactions on Computers, Apr. 1973, pp. 421-422.
[25] L. R. Marino, “General theory of metastability”, in IEEE Transactions on Computers, 1981, pp. 107-115.
[26] J. H. Hohl, W. R. Larsen, and L.C. Schooley, “Prediction of error probabilities for integrated digital synchronizers”, in IEEE Journal of Solid-State Circuits, Apr. 1984, vol. 19, pp. 236-244.
[27] Uming Ko and P. T. Balsara, “High-performance energy-efficient D-flip-flop circuits”, in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Feb. 2000, pp. 94-98.
[28] D. J. Kinniment, A. Bystrov, and A. V. Yakovlev, “Synchronization circuit performance”, in IEEE Journal of Solid-State Circuits, Feb. 2002, pp. 202-209.
[29] R. Ginosar, “Fourteen ways to fool your synchronizer”, in Advanced Research in Asynchronous Circuits and Systems, May 2003, pp. 89-96.
[30] J. N. Seizovic, “Pipeline synchronization”, in Advanced Research in Asynchronous Circuits and Systems, Nov. 1994, pp. 87-96.
[31] J. Muttersbach, T. Villiger, and W. Fichtner, “Practical design of globally-asynchronous locally-synchronous systems”, in Advanced Research in Asynchronous Circuits and Systems, Apr. 2000, pp. 52-59.
[32] C. J. Myers and A. E. Sjogren, “Interfacing synchronous and asynchronous modules within a high-speed pipeline”, in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Oct. 2000, pp. 573-583.
[33] R. Dobkin, R. Ginosar, and C. P. Sotiriou, “Data synchronization issues in GALS SoCs”, in Advanced Research in Asynchronous Circuits and Systems, Apr. 2004, pp. 170-180.
[34] T. Chelcea and S. M. Nowick, “A low-latency FIFO for mixed-clock systems”, in IEEE Computer Society Workshop on VLSI, Apr. 2000, pp. 119-126.
[35] L.-S. Kim and R. W. Dutton, “Metastability of CMOS latch/flip-flop”, in IEEE Journal of Solid-State Circuits, Aug. 1990, pp. 942-951.
[36] H.-S. Jung and M.-K. Lee, “Analysis and implementation of interface for heterogeneous system”, in Asia Pacific Conference on ASICs, Aug. 2000, pp. 21{26.
[37] T. Chelcea and S. M. Nowick, “Robust interfaces for mixed-timing systems with application to latency-insensitive protocols”, in Design Automation Conference, June 2001, pp. 147-150.
[38] Clifford E. Cummings, “Simulation and Synthesis Techniques for Asynchronous FIFO Design,” *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User*

Papers, March 2002, Section TB2, 2nd paper. Also available at www.sunburst-design.com/papers.

[39] Clifford E. Cummings and Peter Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, March 2002, Section TB2, 3rd paper. Also available at www.sunburst-design.com/papers.

[40] K.C. Chang, "Digital Design and Modeling with VHDL and Synthesis", Wiley-IEEE Computer Society Pr, 1st edition (October 4, 1997).

[41] Douglas L. Perry, "VHDL: Programming by Example", McGraw-Hill, 4th edition, 2002.

[42] Peter J. Ashenden, "The Designer's Guide to VHDL", Morgan Kaufmann Publishers, 1st edition (December 1995).

[43] Ben Cohen, "VHDL Coding Styles and Methodologies", Springer; 2nd edition (March 31, 1999).

[44] Alan Jay Smith, "Cache Memories", Computing Surveys, Vol 14, No. 3, September 1982.

[45] Bruce Jacob, "Cache Design for Embedded Real-Time Systems", Embedded Systems Conference, Summer 1999. Danvers MA, June 1999.

[46] B. L. Jacob and T. N. Mudge. "Virtual memory: Issues of implementation", *IEEE Computer*, vol. 31, no. 6, pp. 33–43, June 1998.

[47] N. P. Jouppi. "Cache write policies and performance", In *Proc. 20th Annual International Symposium on Computer Architecture (ISCA-20)*, May 1993, pp. 191–201.

[48] Bruce Jacob, Spencer W. Ng, and David T. Wang, with contributions by Samuel Rodriguez, "Memory Systems: Cache, DRAM, Disk", Morgan Kaufmann Publishers, September 2007.

[49] Steven A. Przybylski, "Cache and Memory Hierarchy Design: A Performance Directed Approach", Morgan Kaufmann; 1 edition (May 1, 1990).

[50] Jim Handy, "The Cache Memory Book", Morgan Kaufmann; 2 edition (January 15, 1998).