2ή Εργαστηριακή Άσκηση

ΔΥΝΑΜΙΚΗ ΟΜΟΧΕΙΡΙΑ ΣΕ ΠΥΡΗΝΑ ΕΠΕΞΕΡΓΑΣΤΗ - ΑΛΓΟΡΙΘΜΟΣ TOMASULO

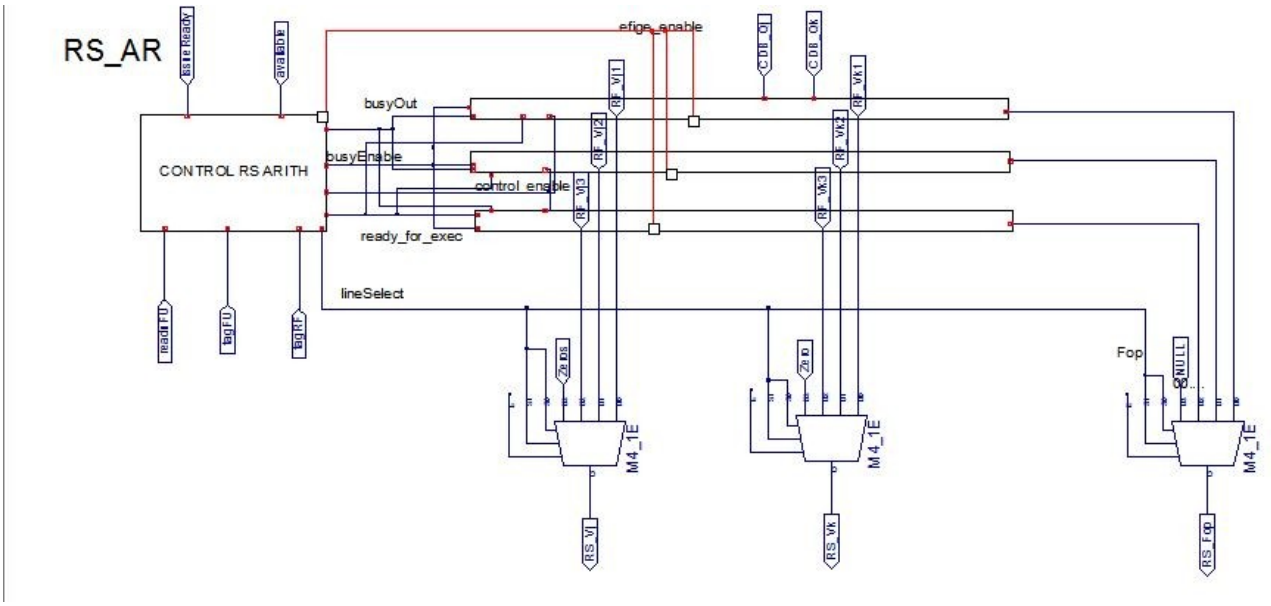
Ομάδα LAB41538988

ΧΡΗΣΤΟΣ ΖΗΣΚΑΣ ΑΜ 2014030191
ΜΑΝΩΛΗΣ ΠΕΤΡΑΚΟΣ ΑΜ 2014030009

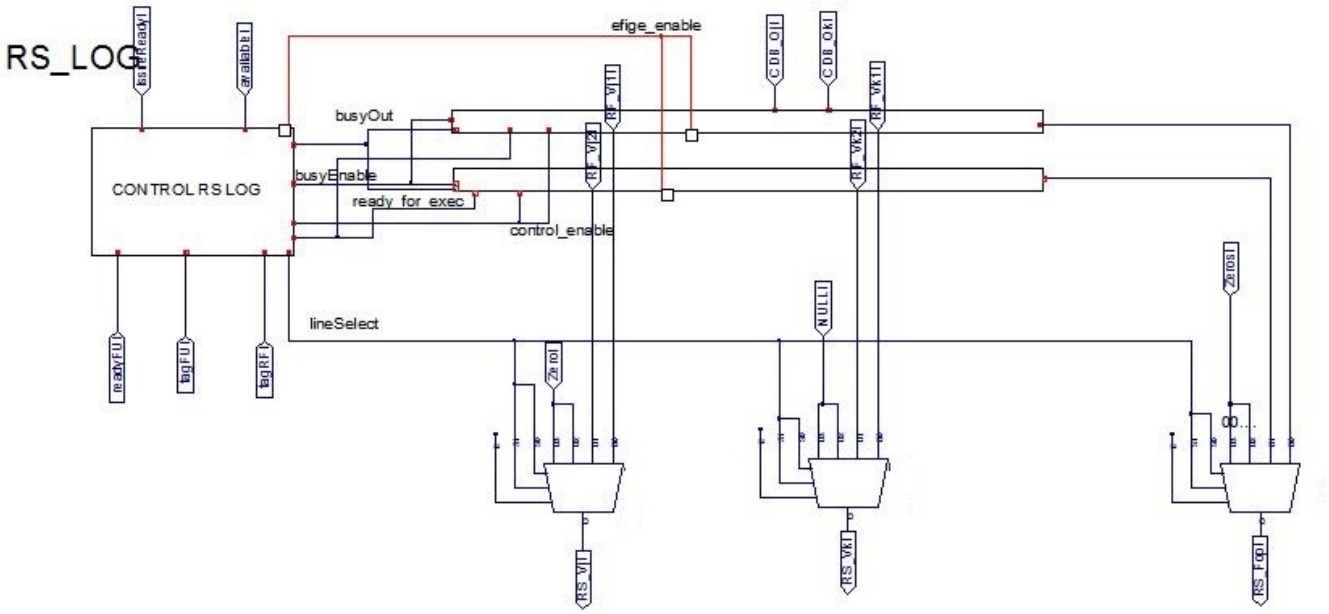
Σχηματικά διαγράμματα και χρονισμοί

Οι αλλαγές φαίνονται με κόκκινο χρώμα.

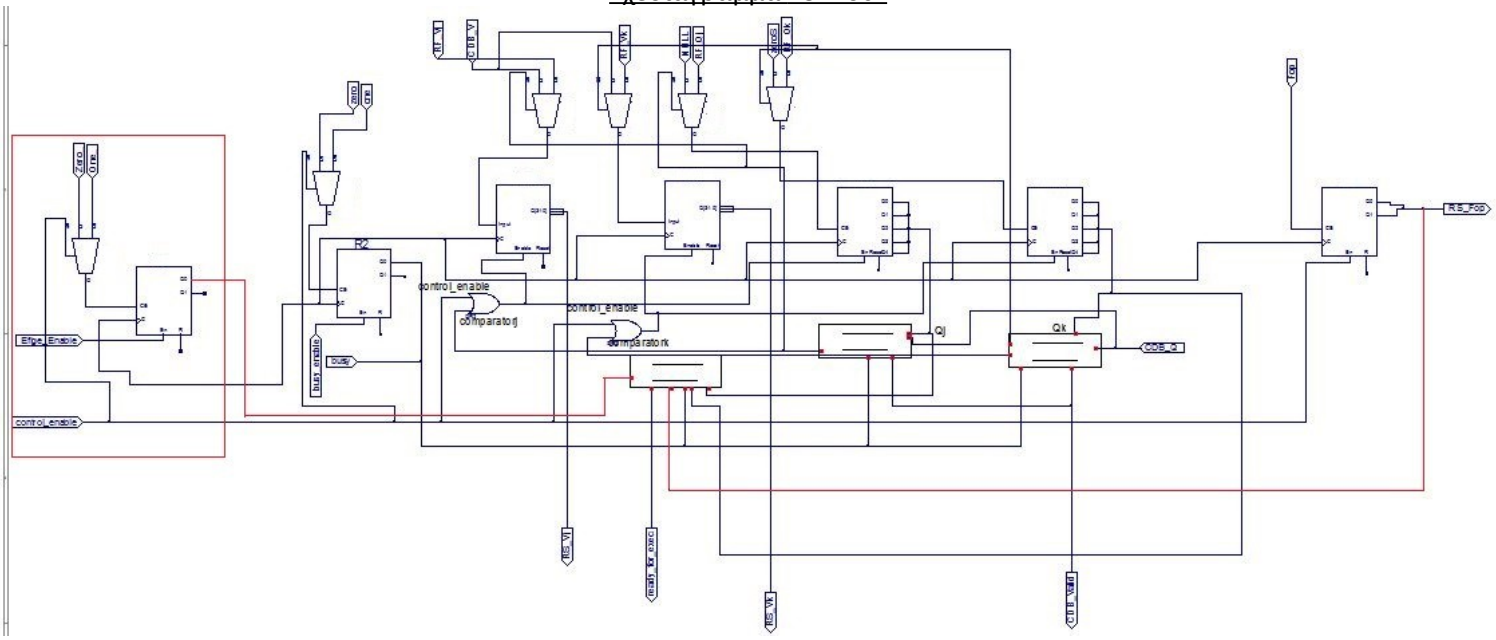
ΒΑΘΜΙΔΑ Reservation Station-arithmetic



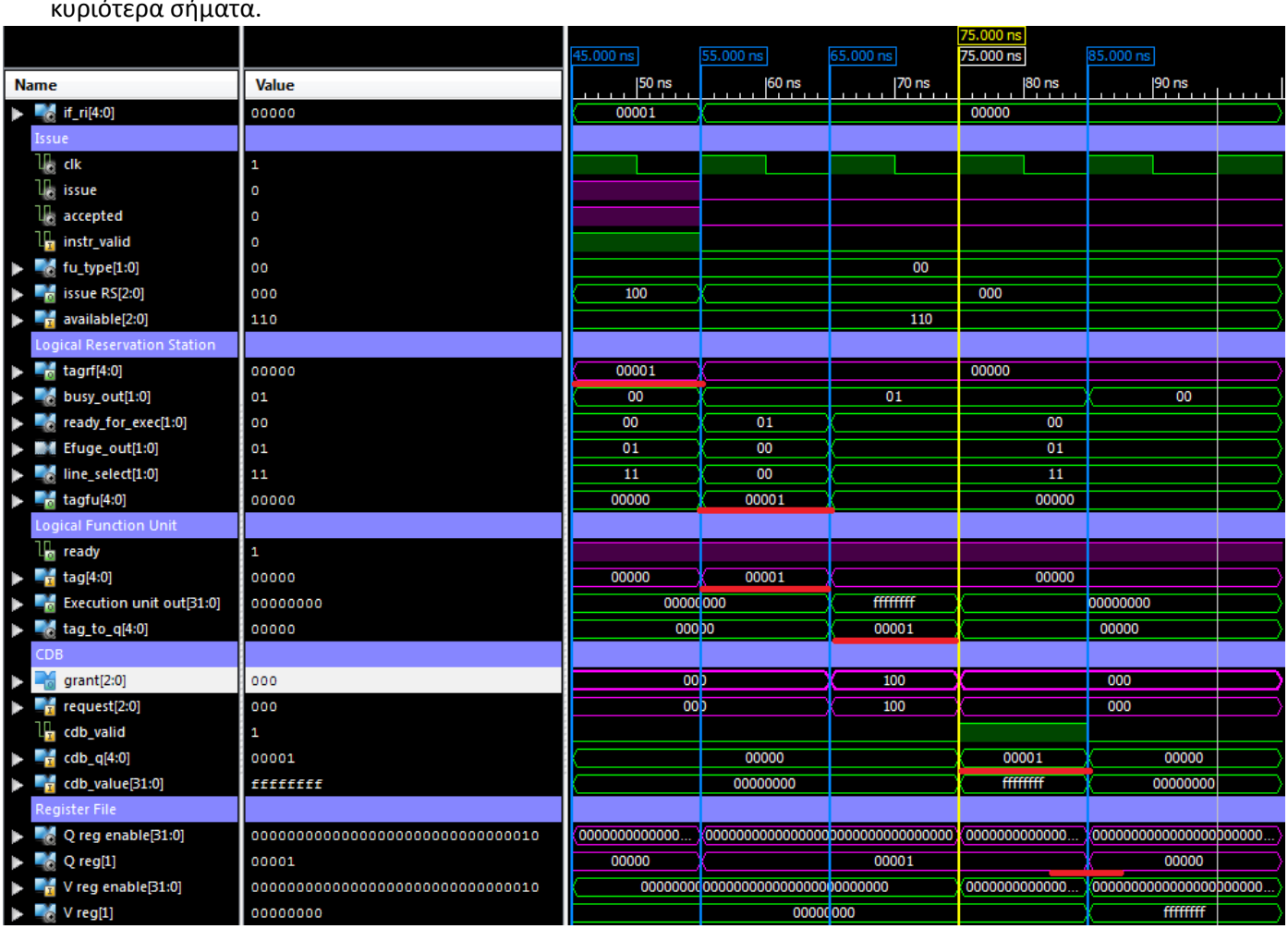
ΒΑΘΜΙΔΑ Reservation Station-logical



Σχεδιάγραμμα RS Block



Δεν έχουν γίνει αλλαγές στους χρονισμούς των μονάδων ή του συνολικού συστήματος, αλλά σε αυτήν την αναφορά θα την αναλύσουμε με τα πιο σημαντικά σήματα ελέγχου. Στο παρακάτω χρονοδιάγραμμα παρουσιάζεται η πορεία μιας λογικής εντολής χωρίς εξαρτήσεις. Δείχνουμε πως αλληλεπιδράν τα κυριότερα σήματα.



- 1ος κύκλος (45ns)

Έκδοση εντολής

Με

$$(issue=1)\wedge(fy_{type}=00)\wedge(available[2]=1)\Rightarrow$$

Στον ίδιο κύκλο

$$(accepted=1)\wedge(instr_{valid}=1)\wedge(issueRS=100)\wedge(Qreg[ri]=tagrf)$$

Στον επόμενο (καθυστερούν λόγο καταχωρητών)

$$(busy_{out}[tagrf]=1)$$

- 2ος κύκλος (55ns)

Αποστολή εντολής στο FU

Με

$$(busy_{out}=01)\wedge(ready=1)\Rightarrow$$

Στον ίδιο κύκλο

$$(ready_{forexc}[0]=1)\wedge(line_{select}=00)\wedge(tagfu=tag[line_{select}])$$

Στον επόμενο

$$(Efuge_{out}[0]=1)$$

Ξεκίνημα πράξης από το FU

Με

$$(tag!=00000)\Rightarrow$$

Στον επόμενο

$$(tag_{toQ}=tag)$$

- 3ος κύκλος(65 ns)

Δέσμευση RS line μέχρι να έρθει το αποτέλεσμα της εντολής

Με

$$(Efuge_{out}[0]=1)\Rightarrow$$

Στον ίδιο κύκλο

$$(ready_{forexc}[0]=0)$$

Διαιτησία ελέγχου CDB

Με

$$(tag_{toQ}!=00000)\Rightarrow$$

Στον ίδιο κύκλο

$$request[2]=1$$

Με

$$(request_{cdb}=100)\Rightarrow$$

Στον ίδιο κύκλο

$$grant=100$$

Με

$$grant=100$$

Στον επόμενο κύκλο

$$(cdb_{valid}=1)\wedge(cdb_{q}=tag_{toQ})\wedge(cdb_{value}=execUnit_{out})$$

- 4ος κύκλος (75 ns)

Κοινοποίηση αποτελεσμάτων στο RS

Στον επόμενο κύκλο

$$(cdb_{valid}=1)\wedge(cdb_q=tagRS)\Rightarrow$$

$$busy_{out}[tag]=0$$

(μηδενίζεται στο επόμενο, αλλά μπορεί να μπει εντολή στον ίδιο κύκλο γιατί το σήμα ελέγχου θα γίνει forward)

Κοινοποίηση αποτελεσμάτων στη RF

$$(cdb_{valid}=1)\wedge(\exists i \rightarrow (cdb_q=Q[i]))\Rightarrow$$

Στον ίδιο κύκλο

$$(Qreg_{enable}[i]=1)\wedge(Vreq_{enable}[i]=1)$$

Στον επόμενο

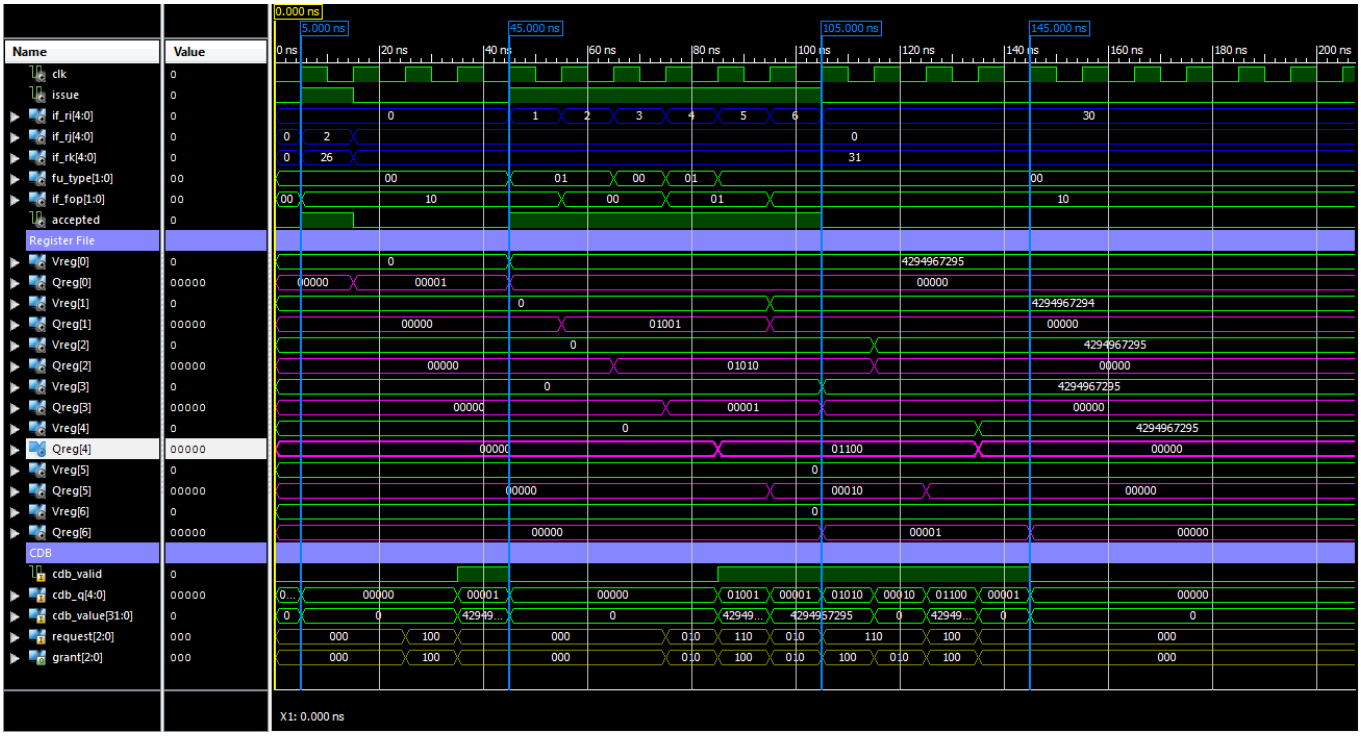
$$(Qreg[i]=00000)\wedge(Vreg[i]=CDB_{value})$$

Περιγραφή αλλαγών

Γίνανε δύο αλλαγές στα block των RS. Ο έλεγχος για το αν η εντολή είναι έτοιμη για εκτέλεση δέχεται μια παραπάνω είσοδο, το Fop. Εφόσον αυτό υποδεικνύει shift ή not, ο έλεγχος αδιαφορεί για τον δεύτερο τελεστέο. Έτσι, αν μόνο το Qk περιμένει το αποτέλεσμα μιας άλλης πράξης (!= “00000”), ο έλεγχος θα στείλει την πράξη προς εκτέλεση πριν έρθει το αποτέλεσμα, εφόσον υπάρχουν διαθέσιμοι πόροι.
Η δεύτερη αλλαγή αφορά την διαδικασία αφαίρεσης εντολών από τα RS. Πριν, ένα process έλεγχε πότε στέλνονται οι εντολές προς το FU και πότε αυτές σβήνονται από τα blocks. Τώρα, η κάθε διαδικασία έχει δικό της process, είναι ανεξάρτητες και δεν συγκρούονται μεταξύ τους. Όμως, χρειάζεται ένα παραπάνω bit μνήμης ανά block όπου υποδεικνύει αν η εντολή έχει σταλθεί για εκτέλεση. Όταν το αποτέλεσμα της εντολής κοινοποιηθεί από το CDB, η εντολή σβήνεται και στον ίδιο κύκλο μπορεί να έρθει στην θέση της μία άλλη. Στις βαθμίδες RS συνδέθηκαν τα καινούργια σήματα από τα blocks με το control

Κυματομορφές

Επειδή όλοι οι καταχωρητές ξεκινάνε με τιμή 0, Στα test έχουμε αρχικοποιήσει τον καταχωρητή 0 με μια not ώστε να έχουμε κάποια διαφορετική τιμή και να φαίνονται καλύτερα τα αποτελέσματα. Αφού τελειώσει η εντολή ξεκινάμε κανονικά το test.(5 -45ns)



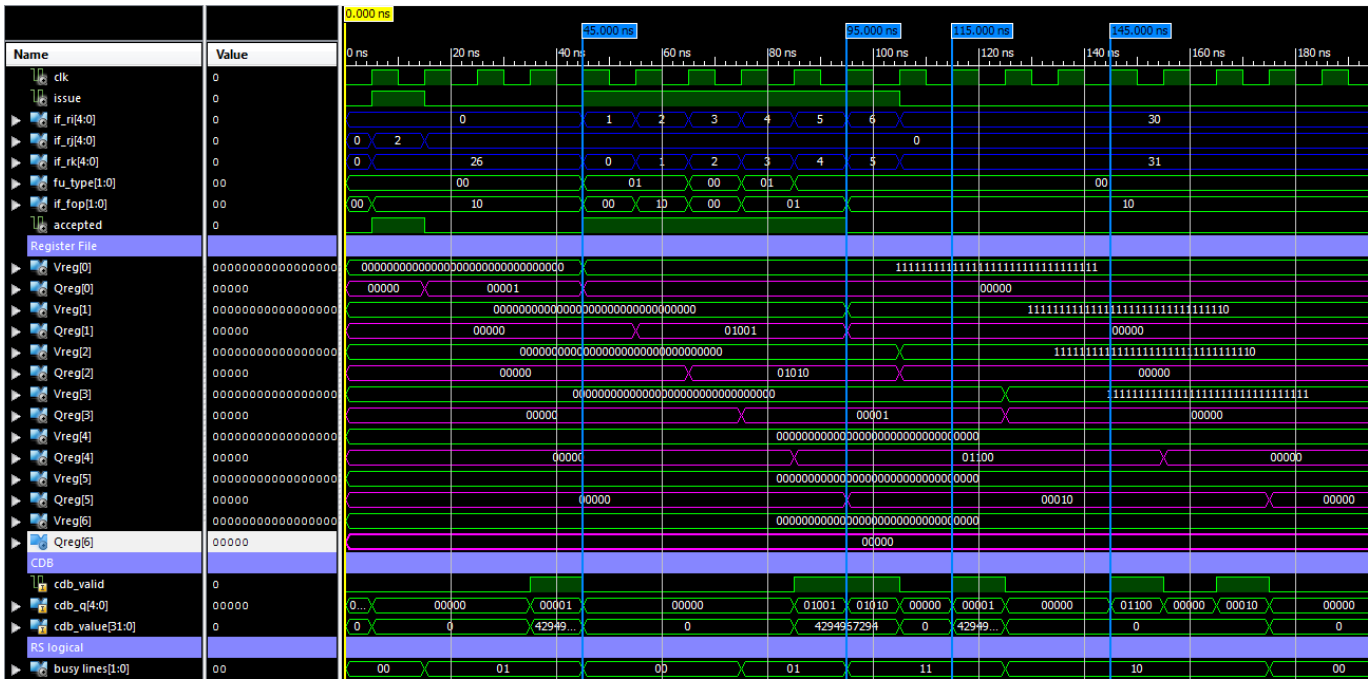
Σε αυτό το τεστ βάζουμε 6 τυχαίες εντολές χωρίς εξαρτήσεις και των δύο τύπων, ώστε να απασχολούνται και τα δύο reservation stations. Επίσης φαίνεται και η διαιτησία πρόσβασης στο CDB όταν ζητάνε τον έλεγχο του δύο μονάδες ταυτόχρονα.

Κανονική λειτουργία χωρίς εξαρτήσεις

Στα 45 ns αρχίζουν να έρχονται οι εντολές. Η πρώτη εντολή γράφει στον καταχωρητή 1, η δεύτερη στον 2 κτλ. Από τους καταχωρητές Q φαίνεται ότι το renaming γίνεται διαδοχικά. Η έκδοση εντολών γίνεται με την σειρά που τις στέλνει το IF stage. Στα 105 ns σταμάτησαν οι εντολές και φαίνεται ότι εγκρίθηκαν όλες, αφού το σήμα accepted ήταν συνέχεια ‘1’.
Στα 85 με 145 ns εμφανίζονται τα αποτελέσματα τους στον CDB. Έχουν καθυστέρηση 4 κύκλους γιατί η πρώτη εντολή είναι αριθμητική. Η διαιτησία έχει γίνει ένα κύκλο πριν, καθώς τα σήματα request και grant ενεργοποιούνται ένα κύκλο πριν τα σήματα CDB_Q, CBD_V και CDB_valid. Επίσης, στον επόμενο κύκλο μετά την κοινοποίηση, το αποτέλεσμα γράφεται στον καταχωρητή V με το ίδιο Q το οποίο μηδενίζεται. Παρατηρώντας την σειρά με την οποία αποθηκεύονται τα αποτελέσματα, διαπιστώνεται ότι οι εντολές δεν υλοποιούνται με την σειρά που εκδόθηκαν.

Συγκρούσεις CDB

Στο παραπάνω τεστ συμβαίνουν 3 συγκρούσεις. Η πρώτη γίνεται στη δεύτερη διαιτησία(85 ns) αφού request = “110” και τον έλεγχο παίρνει η λογική(grant = “100). Πράγματι, στον επόμενο κύκλο το Q που κοινοποιείται είναι “00001” όπου προέρχεται από το λογικό RS. Η εντολή που δεν πήρε τον έλεγχο θα τον πάρει στον επόμενο κύκλο.
Οι άλλες δυο συγκρούσεις συμβαίνουν η μία μετά την άλλη(105-125ns). Για δύο κύκλους το request είναι “110”. Επειδή στον κύκλο πριν τη σύγκρουση είχε περάσει αριθμητική, πήρε πρώτο τον έλεγχο η λογική εντολή(grant =”100”). Στον επόμενο κύκλο θα πάρει προτεραιότητα η αριθμητική(grant =”010”) .
Αυτό συμβαίνει λόγο του round-robin αλγόριθμου που υλοποιεί το control του CDB.



Τέλος, επειδή δεν υπάρχουν εξαρτήσεις οι εντολές διαρκούν τους αναμενόμενους κύκλους και το CPI είναι 1 από τότε που αρχίσουν να εκδίδονται τα αποτελέσματα. Επίσης οι συγκρούσεις του CDB δεν επηρεάζουν το CPI.
Σε αυτό το τεστ βάζουμε 6 εντολές όπου η κάθε μία εξαρτάται από την προηγούμενη. Πέρα από τις εξαρτήσεις RAW, κοιτάμε την αδιαφορία του δεύτερου τελεστή από τις εντολές not/shift, το forward στο RS από το CDB και ότι ο RS γεμίζει αν βάλουμε πολλές εντολές που πρέπει να περιμένουν άλλες.

Εξαρτήσεις RAW

Οι εντολές είναι ίδιες με του προηγούμενου τεστ εκτός από τους καταχωρητές πηγής Rk. Το rename γίνεται και πάλι διαδοχικά αλλά η κοινοποίηση των αποτελεσμάτων είναι αρκετά διαφορετική. Αυτό συμβαίνει γιατί οι εντολές περιμένουν τα αποτελέσματα της προηγούμενης. Εξαιρείται το δεύτερο αποτέλεσμα, το οποίο κοινοποιείται κατευθείαν μετά το πρώτο(95ns), γιατί προέρχεται από shift εντολή όπου αδιαφορεί για τις εξαρτήσεις του καταχωρητή Rk.
Για την τρίτη κοινοποίηση περιμέναμε δύο καθυστερήσεις γιατί η εντολή είναι λογική, αλλά εμφανίζεται μόνο μία. Αυτό συμβαίνει γιατί το αποτέλεσμα του CDB γίνεται forward μέσα στα RS, ώστε στον ίδιο κύκλο με την κοινοποίηση να μπορεί να ξεκινήσει η επόμενη εντολή που το περιμένει. Αντίστοιχα, στην επόμενη κοινοποίηση(145ns) παρατηρούμε δύο καθυστερήσεις αντί των αναμενόμενων τριών, επειδή η εντολή είναι αριθμητική.
Τέλος επειδή η κάθε εντολή περιμένει την προηγούμενη της, αυτές εκτελούνται με την σειρά που ήρθαν και με τον ίδιο τρόπο κοινοποιούνται τα αποτελέσματα τους.

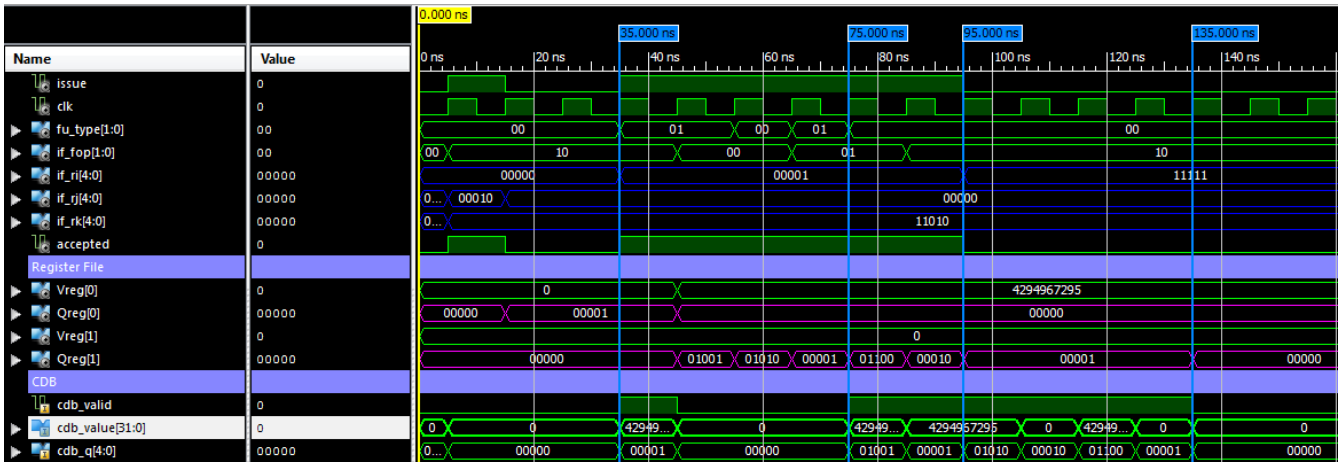
Γέμισμα RS

Παρατηρώντας το πεδίο accepted στα 95ns φαίνεται πως το σύστημα απορρίπτει την τελευταία εντολή. Αυτό συμβαίνει γιατί η εντολή είναι λογική και η κατάλληλη μονάδα δεν έχει διαθέσιμους πόρους. Πράγματι, το πεδίο busy lines του λογικού RS έχει την τιμή “11”, δηλαδή και οι δύο γραμμές του είναι απασχολημένες. Αν είχε υλοποιηθεί το front end του επεξεργαστή η εντολή θα είχε ξανασταλθεί.

Εμφανώς το CPI είναι μικρότερο της μονάδας, καθώς υπάρχουν καθυστερήσεις λόγω των εξαρτήσεων αλλά και της απόρριψης εντολών.

Στο συγκεκριμένο τεστ, εκτελούμε συνέχεια εντολές που αφορούν το αριθμητικό RS(fu_type = “01”). Βλέπουμε ότι κάθε τέταρτη εντολή απορρίπτεται από το σύστημα(accepted = ‘0’). Αυτό συμβαίνει επειδή το αριθμητικό RS έχει τρία slot για εντολές αλλά το αντίστοιχο FU έχει τρεις κύκλους καθυστέρηση. Όταν η τέταρτη εντολή στέλνει αίτημα(75ns) για έκδοση στο Issue η πρώτη βρίσκεται στην τελευταία καθυστέρηση του FU. Δηλαδή και τα τρία slots έχουν εντολές που εκτελούνται και αν γινόταν το Issue, θα αντικαθιστούσε μια άλλη όπου δεν έχει κοινοποιήσει τα αποτελέσματα της. Τότε θα είχε το ίδιο tag με την παλιά, και μόλις η πρώτη κοινοποιούσε το αποτέλεσμα της, το σύστημα θα νόμιζε ότι ήταν και τον δύο και θα αποθηκευόντουσαν λάθος τιμές στη καινούργια. Αυτό πρέπει να το αποφύγουμε για να εγγυηθούμε την ορθότητα του συστήματος.
Για να επιβεβαιώσουμε την παραπάνω λειτουργία, κάθε slot του RS έχει ένα bit μνήμης για να θυμάται αν έχει μέσα έγκυρη εντολή όπου αποτυπώνονται στο πεδίο busy lines. Πράγματι, όταν γίνεται “111” δεν έχει διαθέσιμο χώρο και το σήμα available γίνεται ‘0’. Όμως, στους επόμενους 3 κύκλους παραμένει το busy lines “111” αλλά το RS λέει πως έχει διαθέσιμο χώρο. Αυτό συμβαίνει γιατί τελείωσε μια εντολή που ήταν μέσα και στον ίδιο κύκλο μπορεί να μπει στη θέση της μια άλλη, καθώς η μονάδα είναι pipe-lined. Δηλαδή, μια εντολή απορρίπτεται μόνο εάν είναι όλα τα block απασχολημένα και καμία από τις αποθηκευμένες εντολές δεν ολοκληρώνεται τον ίδιο κύκλο.

Το CPI στο τεστ είναι 3/4. Αν εγίνει το τεστ στην λογική μονάδα, κάθε τρίτη εντολή απορρίπτεται γιατί έχει δύο θέσεις για εντολές και δύο καθυστερήσεις στην μονάδα εκτέλεσης της. Τότε το CPI είναι 2/3.



Εξαρτήσεις WAW

Στο παραπάνω τεστ εκτελούμε εντολές χωρίς εξαρτήσεις με τον ίδιο καταχωρητή προορισμού. Βλέπουμε ότι πεδίο Qreg[1] αλλάζει συνέχεια tag. Οι εντολές που έχουν τα παλιά tags υλοποιούνται και κοινοποιούν τα αποτελέσματά τους στο CDB αλλά δεν αποθηκεύονται στο αρχείο καταχωρητών. Αν κάποια εντολή περιμένει αυτά τα αποτελέσματα, θα τα λάβει και θα συνεχίσει η λειτουργία κανονικά. Αλλά δε υπάρχει λόγος να γραφτούν στο RS. Στα 135ns που κοινοποιείται η τελευταία εντολή, το αποτέλεσμα της γράφεται στο RF και το Q μηδενίζεται.

Forward RF

Επίσης έχουμε ξεκινήσει τις εντολές στον ίδιο κύκλο(35ns) που κοινοποιείται το αποτέλεσμα της πρώτης εντολής not που αρχικοποιεί τον καταχωρητή 0. Τότε πρέπει να γίνει το αποτέλεσμα forward από τον RF ώστε να το χρησιμοποιήσει η πρώτη εντολή. Πράγματι κοιτώντας στα 75ns βλέπουμε ότι το αποτέλεσμα της νέας εντολής έχει παραχθεί από τα δεδομένα της προηγούμενης.

Το CPI είναι καθώς δεν επηρεάζεται από τις εξαρτήσεις WAW.

Συνοπτικά τα παραπάνω τεστ ελέγχουν πέρα από το pipe-lining: Εξαρτήσεις RAW/ WAW, συγκρούσεις στο CDB, forward στο RF και τα RS, γέμισμα των λειτουργικών μονάδων καθώς και περιπλοκές μεταξύ τους. Τέλος παρουσιάζεται και το CPI ώστε να φαίνεται η αποδοτικότητα του συστήματος σε αυτές τις περιπτώσεις.

Στρατηγική και αποτελέσματα του ελέγχου

Η στρατηγική του ελέγχου εγκυρότητας του συστήματος αποτελείται από τρία στάδια.

- Unit tests: Τα τεστ του προηγούμενου εργαστηρίου έλεγχαν κάθε μονάδα ξεχωριστά. Σκοπός τους είναι να εγγυηθούν ότι η εσωτερική λειτουργία των μονάδων είναι σωστή και ολοκληρωμένη. Υποθέτουν πως όλες οι άλλες μονάδες είναι ολοκληρωμένες και ορθές καθώς οι είσοδοι αλλάζουν χειροκίνητα από το τεστ. Χαμηλού επιπέδου τεστ σαν και αυτά, επιτρέπουν ενδελεχή και συγκεντρωμένο έλεγχο των modules που μας ενδιαφέρουν χωρίς να μας απασχολεί το περιβάλλον τους και πως αυτό τα επηρεάζει. Έτσι, μπορούμε εύκολα να παρατηρήσουμε την συμπεριφορά τους σε μικρές αλλαγές καθώς και τα εσωτερικά σήματα τους. Αφού είναι επιτυχής και μπορούμε να εγγυηθούμε την ορθότητα των ξεχωριστών μονάδων, ο έλεγχος μπορεί να προχωρήσει στο επόμενο στάδιο.

- Απλά integrity tests: Μετά την σύνδεση των ξεχωριστών μονάδων φτιάξαμε απλά τεστ, πχ. μια εντολή. Σκοπός τους είναι να ελέγξουν την σύνδεση των modules καθώς και την επικοινωνία μεταξύ τους. Επίσης, παρατηρείται εύκολα ο χρονισμός του συστήματος, καθώς η απλότητα του ελέγχου αποφεύγει τις συγκρούσεις. Ακόμα, επιτρέπουν να μελετήσουμε μεμονωμένες ενδιαφέρον περιπτώσεις χωρίς συγκρούσεις και θόρυβο. Τέλος, λόγο της μικρής πολυπλοκότητας του test, σε περίπτωση λάθους εντοπίζεται εύκολα η μονάδα που ευθύνεται και μπορούμε εύκολα να γυρίσουμε στο προηγούμενο επίπεδο. Αν όλα φαίνονται σωστά, ο έλεγχος προχωράει στο τελευταίο στάδιο.

- Σύνθετα integrity tests: Τα τεστ που αναλύονται στην παρούσα αναφορά είναι αυτού του τύπου. Αποτελούνται από μικρά προγράμματα 5-15 εντολών με διάφορων τύπων συγκρούσεις και εξαρτήσεις(RAW, WAW). Σκοπός τους είναι να μελετήσουν ακραίες περιπτώσεις(πχ. Συνέχεια εντολές ίδιου τύπου), την ορθή συνύπαρξη διάφορων καταστάσεων (πχ. Issue ενώ έχει γεμίσει το RS επειδή μπλόκαρε το FU καθώς δεν πήρε τον έλεγχο του CDB.) και το pipe-lining του επεξεργαστή. Πέρα από την ορθότητα του συστήματος, τα τεστ αυτού του είδους μπορούν να ανακαλύψουν σχεδιαστικά λάθη όπου δεν εμφανίζονται στα προηγούμενα επίπεδα, καθώς αυτά ελέγχουν κυρίως την υλοποίηση.

Γενικά, οι έλεγχοι δεν επικεντρώνονται στα αποτελέσματα των πράξεων και δεν είναι εξαντλητικοί. Αν μια εντολή κάθε τύπου έχει σωστά αποτελέσματα και αυτά αποθηκευτούν στους καταχωρητές που υποδεικνύουν, τότε μπορούμε να υποθέσουμε ότι όλες οι πράξεις θα γίνονται σωστά. Επίσης, είναι αδύνατον να ελεγχθούν όλοι οι δυνατοί συνδυασμοί πράξεων και προγραμμάτων. Για αυτό, επιλέχθηκαν μερικά αντιπροσωπευτικά, ώστε να καλύπτεται ένα ικανοποιητικά μεγάλο εύρος περιπτώσεων. Τέλος, τα τεστ που παρουσιάζονται επικεντρώνονται στη συμπεριφορά των σημάτων ελέγχου και το χρονισμό του συστήματος. Εφόσον τα σήματα είναι ορθά σύμφωνα με την σχεδίαση, η αποθήκευση των αποτελεσμάτων γίνεται σωστά και οι χρονισμοί είναι βέλτιστοι σύμφωνα με τις προδιαγραφές · μπορούμε να υποθέσουμε ότι το σύστημα είναι λειτουργικό και αποδοτικό.

Τα αποτελέσματα του ελέγχου δείχνουν ότι το σύστημα είναι πλήρως λειτουργικό και αντιδράει σωστά σε ακραίες περιπτώσεις. Επίσης, είναι pipe-lined σε όλα τα στάδια, αφού δεν μένουν μονάδες ανεκμετάλλευτες εάν υπάρχει διαθέσιμη δουλειά για αυτές.

Σημειώσεις

Μέσα στον φάκελο με τον κώδικα υπάρχουν αρχεία .wcfg για τα test που παρουσιάζουμε. Επίσης ,σε δικούς τους φακέλους, επισυνάπτονται οι κυματομορφές και τα σχεδιαγράμματα.

Github Link