ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση σε systemC ενός μικροεπεξεργαστή με ομοχειρία 5 επιπέδων.

Γιαννακού Χ. Χριστόδουλος

Επιτροπή: Αναπληρωτής Καθηγητής Δ. Πνευματικάτος (επιβλέπων)

Καθηγητής Γ. Σταυρακάκης

Διδάσκων Π.Δ. 407/80 Ι. Παπαευσταθίου

ΧΑΝΙΑ 2006

Θα ήθελα να εκφράσω τις θερμες μου ευχαριστίες στον Διδάσκων Π.Δ. 407/80 κ. Παπαευσταθίου Ιωάννη για τη πολύτιμη βοήθεια που μου προσέφερε καθ'όλη τη διάρκεια της παρούσας εργασίας.

Ένα μεγάλο ευχαριστώ επίσης και στο κ. Κιμιωνή Μάρκο, καθώς και στα παιδιά του εργαστηρίου μικροεπεξεργαστών και υλικού του Π.Κ και ιδιαίτερα τους κ. Κοζανίτη Χρήστο, Χρυσό Γρηγόριο, για την αμερόληπτη βοήθεια που μου παρείχαν για τη διεκπεραίωση της εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΗ – ΓΕΝΙΚΑ ΣΤΟΙΧΕΙΑ ΓΙΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΠΕΞΕΡΓΑΣΤΩΝ

1.1 Γενικά στοιχεία - Ιστορική ανασκόπηση

Η χρήση των ηλεκτρονικών υπολογιστών (Η/Υ) σε όλους τους τομείς της ανθρώπινης δραστηριότητας είναι ουσιώδους σημασίας. Όλα βασίζονται στη χρησιμοποίηση Η/Υ, από τις πιο ασήμαντες υπηρεσίες μέχρι υπηρεσίες υψίστης ασφαλείας. Από την εμφάνιση των πρώτων υπολογιστικών συστημάτων μέχρι και σήμερα, υπάρχει αλματώδης ανάπτυξη των τεχνολογιών υλοποίησης. Η ανάπτυξη αυτή συνεχίζεται με γοργούς ρυθμούς.

Ο πρώτος ηλεκτρονικός υπολογιστής γενικού σκοπού, κατασκευάστηκε πριν το $2^{ο}$ παγκόσμιο πόλεμο, από τους J. Presper Eckert και John Mauchly στο Moore school του πανεπιστημίου της Πελσυλβάνιας[1]. Η μηχανή αυτή με την επωνυμία ENIAC (Electronic Numerical Integrator and Calculator), χρηματοδοτήθηκε και χρησιμοποιήθηκε για στρατιωτικούς σκοπούς από τις Η.Π.Α. Ήταν βασισμένος σε συσσωρευτές και αποτελούνταν από 20 καταχωρητές των 10 ψηφίων, 18000 ηλεκτρονικές λυχνίες και είχε τεράστιες διαστάσεις. Η διαφοροποίηση του σε σχέση με παλαιότερους υπολογιστές έγκειται στο γεγονός ότι μπορούσε να προγραμματιστεί. Ακολούθησε το 1944, η υλοποίηση ενός υπολογιστή αποθηκευμένων προγραμμάτων που εισάγονταν σαν αριθμοί. Το μηχάνημα αυτό ονομάστηκε EDVAC (Electronic Discrete Variable Automatic Computer).

Το 1946, ο Maurice Wilkes βασιζόμενος στις μέχρι τότε υλοποιήσεις, εξέλιξε ακόμη περισσότερο τη τεχνολογία κατασκευάζοντας το πρώτο πλήρη λειτουργικό υπολογιστή με αποθήκευση προγραμμάτων. Η αρχιτεκτονική του μηχανήματος αυτού βασιζόταν σε συσσωρευτή, τεχνική που ήταν δημοφιλής έως τις αρχές της δεκαετίας του 70. Παράλληλα με τις προσπάθειες αυτές, αναπτύχθηκαν και μηχανήματα για ειδικούς σκοπούς όπως κρυπτανάλυση. Τέτοια μηχανήματα προωθούνταν για στρατιωτικούς σκοπούς τόσο στις Η.Π.Α όσο και στη Μεγάλη Βρετανία.

Εκτός όμως από την στρατιωτική εκμετάλλευση των Η/Υ, δημιουργήθηκαν και εταιρείες κατασκευής υπολογιστών για εμπορικούς σκοπούς. Έτσι το 1947 δημιουργήθηκε η εταιρεία Eckert και Mauchly. Το πρώτο τους μηχάνημα, ο BINAC, κατασκευάστηκε για την εταιρεία Northrop το 1949. Ακολούθησε η δημιουργία του UNIVAC που αν και κόστιζε $250000, πούλησε 48 κομμάτια. Μετά το 1950 δραστηριοποιείται στον τομέα αυτό και η IBM με τη κατασκευή του IBM 701 ο οποίος πούλησε 19 κομμάτια. Στη συνέχεια αν και υπήρξαν αρκετές απαισιόδοξες προβλέψεις για το μέλλον των «εξειδικευμένων» μηχανών, η IBM εξελίχθηκε στη πιο πετυχημένη εταιρεία υπολογιστών. Το 1956 κατασκευάστηκε ο πρώτος υπολογιστής καταχωρητή γενικού τύπου, ο Pegasus. Αργότερα παρουσιάστηκαν υπολογιστές που λάμβαναν υπόψη την εξισορρόπηση όλων των παραγόντων λογισμικού και υλικού. Οι σχεδιάσεις των μηχανημάτων βασίζονταν σε μια αρχιτεκτονική στοίβας. Η αρχιτεκτονική αυτή απέδιδε καλή πυκνότητα κώδικα, αλλά παρείχε μόνο δύο περιοχές αποθήκευσης με μεγάλη ταχύτητα.

Ο όρος αρχιτεκτονική υπολογιστή επινοήθηκε από την IBM στις αρχές της δεκαετίας του 1960, όταν οι σχεδιαστές ήθελαν να αναφερθούν σε μέρος του συνόλου εντολών του IBM 360 που ήταν ορατό από τον προγραμματιστή. Ο ορισμός για την αρχιτεκτονική ήταν:
… η δομή ενός υπολογιστή την οποία πρέπει να καταλάβει ο προγραμματιστής της γλώσσας μηχανής για να γράψει ένα σωστό πρόγραμμα για τη μηχανή αυτή.

Στις αρχές της δεκαετίας του 1970, όταν το κόστος του λογισμικού μεγάλωνε γρηγορότερα από ότι το κόστος του υλικού, άρχισε να δημιουργείται μια κρίση λογισμικού. Ο Tanenbaum το 1978, μετά από μια έρευνα διαπίστωσε ότι τα περισσότερα προγράμματα είναι απλά. Αυτό έπρεπε να λαμβάνεται υπόψη από τις αρχιτεκτονικές. Επίσης οι αρχιτεκτονικές έπρεπε να βελτιστοποιούν το μέγεθος του προγράμματος καθώς και την ευκολία της μεταγλώττισης. Έτσι αναπτύχθηκε η αρχιτεκτονική VAX.

Κατά τη δεκαετία του 1980, η κατεύθυνση της αρχιτεκτονικής υπολογιστών άρχισε να απομακρύνεται από την παροχή υλικού το οποίο να υποστηρίζει υψηλού επιπέδου για γλώσσες. Η ανάγκη της υιοθέτησης απλούστερων αρχιτεκτονικών, οδήγησε στην παρουσίαση των υπολογιστών περιορισμένου συνόλου εντολών RISC [Patterson και Ditzel 1980]. Οι απλοί υπολογιστές φόρτωσης-αποθήκευσης όπως ο MIPS ονομάζονται κοινώς αρχιτεκτονικές RISC (Βλέπε Παράρτημα Α). Στη συνέχεια υλοποιήθηκαν τρία RISC ερευνητικά προγράμματα πάνω στα οποία βασίστηκε η βιομηχανία υπολογιστών. Από τότε, με εξαίρεση ένα σύνολο εντολών από υπολογιστές με πολύπλοκο σύνολο εντολών (CISC), η αρχιτεκτονική

RISC είναι η βάση σχεδίασης υπολογιστικών συστημάτων. Οι μετάφραση από CISC σε RISC γίνεται με τη κατασκευή μικροεπεξεργαστών που υλοποιούν τη μετάφραση εσωτερικά.

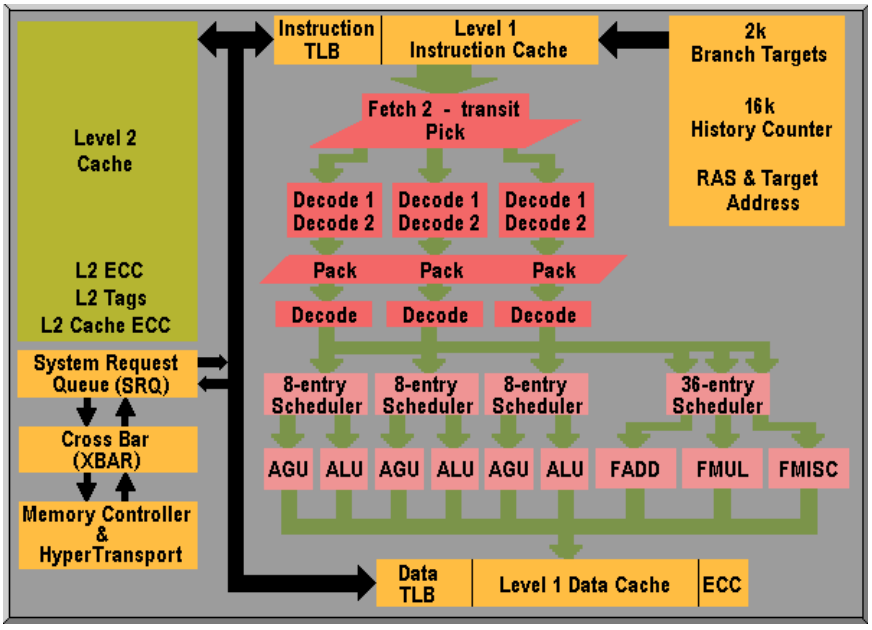
Σήμερα, η παραγωγή και διάθεση υπολογιστών γενικής χρήσης έχει αυξηθεί δραματικά. Η εξέλιξη και αναβάθμιση των αρχιτεκτονικών των σημερινών επεξεργαστών είναι πάρα πολύ μεγάλη. Δύο είναι οι εταιρείες που δεσπόζουν στο τομέα αυτό: Η Intel με τους επεξεργαστές Pentium, Pentium M, Celeron και η AMD με τους επεξεργαστές Athlon, Duron και Athlon M. Συγκεκριμένα η εταιρεία Intel[2] παρέχει επεξεργαστές Pentium 4 με τεχνολογίες Hyper threading, απλούς Pentium 4, Pentium 4 – M για ενσωματωμένες (embedded) τεχνολογίες, καθώς και επεξεργαστές 64-bit. Οι επεξεργαστές αυτοί χρησιμοποιούνται σε εφαρμογές δικτύων, τηλεπικοινωνιακών και αποθηκευτικών εργαλείων, σύνθετης αλληλεπίδρασης πελατών (sophisticated interactive clients), σε βιομηχανικές λύσεις αυτοματισμού, ψηφιακής εποπτείας και παρακολούθησης συστημάτων ασφαλείας, για ψηφιακή επεξεργασία σήματος και εικόνας, καθώς και σαν απλές παιγνιδομηχανές ή για εφαρμογές σπιτιού. Όλες οι τεχνολογίες που υποστηρίζουν βασίζονται στη ομοχειρία με τους επεξεργαστές Hyper threading να διπλασιάζουν το μήκος της ομοχειρίας συγκρινόμενο με τους επεξεργαστές Pentium 3.

Παράλληλα με τους πιο πάνω επεξεργαστές το 1999 ανακοινώθηκε από την Intel η 64-bit αρχιτεκτονική[15]. Το πλεονέκτημα της σε σχέση με τη 32-bit αρχιτεκτονική είναι το μέγεθος της μνήμης που μπορεί να διευθυνσιοδοτηθεί. Συγκεκριμένα ενώ οι 32-bit πλατφόρμες παρέχουν 4GB διευθυνσιοδοτημένης μνήμης, οι 64-bit πλατφόρμες υποστηρίζουν 16 TB μνήμης (4 εκατομμύρια φορές περισσότερη μνήμη από ότι οι 32-bit πλατφόρμες). Οι πλατφόρμες αυτές συνδυάζουν παραλληλισμό και πρόβλεψη που εκμεταλλεύονται καλύτερα τις τεχνολογίες RISC και CISC. Η αυξανόμενη απόδοση πετυχαίνεται με την ελαχιστοποίηση του κόστους των διακλαδώσεων και την ελαχιστοποίηση των επιπτώσεων της καθυστέρησης της διεπικοινωνίας μνήμης – επεξεργαστή.

Σε ότι αφορά τους επεξεργαστές της AMD[15], η εταιρεία επέκτεινε την αρχιτεκτονική x86 έτσι ώστε να υποστηρίξει 64-bit επεξεργαστές παράλληλα με υποστήριξη των ήδη 32-bit υλοποιημένων εφαρμογών. Έτσι η σχεδίαση AMD x86-64 γνωστή και ως «Sledgehammer» ήταν σε θέση να ανιχνεύει ποια από τις δύο σχεδιάσεις απαιτείται για τις ανάγκες κάποιας εφαρμογής. Τα πλεονεκτήματα των 64-bit AMD επεξεργαστών σε σχέση με τους αντίστοιχους της Intel, θεωρούνται τα εξής:

- Πλήρη υποστήριξη και των 64-bit και των 32-bit εφαρμογών.
- Λιγότερη κατανάλωση ισχύος με συνέπεια μεγαλύτερες συχνότητες λειτουργίας
- Η ενσωμάτωση και των δύο x86-64 επεξεργαστών σε ένα chip
- Είναι ανεξάρτητοι νέων πολύπλοκων τεχνολογιών μεταγλώτισης.
- Λιγότερο κόστος.

Οι αλλαγές στο ομόχειρο μονοπάτι δεδομένων του επεξεργαστή αυτού, από τον Hammer, έγκεινται στο γεγονός ότι το front-end των ομόχειρων σταδίων ανάκτησης εντολής και αποκωδικοποίησης σπάζουν έτσι ώστε να παραδίδουν μεγαλύτερα πακέτα εντολών από τους αποκωδικοποιητές στο στάδιο εκτέλεσης εντολής. Αυτό έχει ως αποτέλεσμα να ξανά οριστούν τα ομόχειρα στάδια έτσι ώστε να επιφέρουν μεγαλύτερο βαθμό σταθερότητας συχνότητας. Για να γίνει αυτό προστίθενται ακόμα δύο στάδια ομοχειρίας. Το σχηματικό διάγραμμα του μικροεπεξεργαστή αυτού φαίνεται στο σχήμα 1.

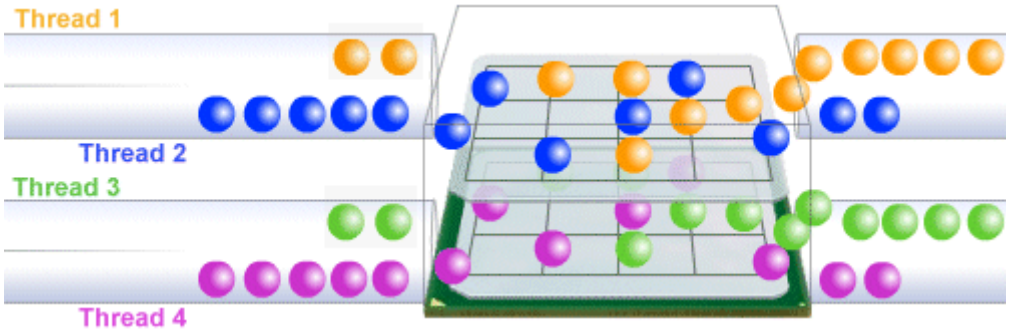


Σχήμα 1

Διάγραμμα των βαθμίδων του ομόχειρου επεξεργαστή AMD 64-bit.

Εκτός από τους προαναφερθέντες επεξεργαστές με μόνο ένα πυρήνα, αναπτύχθηκαν και διπύρηνοι επεξεργαστές (dual core). Τον Οκτώβριο του 1989, τέσσερις μηχανικοί της Intel, σε άρθρο τους με τίτλο "Microprocessors Circa 2000"[3], προέβλεπαν την προώθηση στην αγορά των διπύρηνων επεξεργαστών. Αυτό έγινε 15 χρόνια αργότερα τόσο από την Intel όσο και από την AMD. Η αρχιτεκτονική των επεξεργαστών αυτών εμπεριέχει λεπτομερή σχεδίαση του πυριτίου έτσι ώστε να τοποθετούνται δύο ή περισσότεροι μικροεπεξεργαστές

υπολογισμών σε ένα επεξεργαστή. Αυτός ο διπύρηνος επεξεργαστής αν και εφαρμόζεται άμεσα σε μια υποδοχή (socket), καθένας από τους δύο πυρήνες του ενεργεί σαν μια ξεχωριστή λογική μονάδα επεξεργασίας. Αυτό διαφέρει από την Hyper-Threading τεχνολογία, η οποία κάνει τους υπολογισμούς με ένα πυρήνα χρησιμοποιώντας πιο αποδοτικά τις υπάρχουσες πηγές για τη εκτέλεση (καλύτερο threading). Με τον τρόπο αυτό εκτελούνται πιο πολλές εργασίες σε κάθε κύκλο ρολογιού. Ο συνδυασμός της τεχνολογίας Hyper-Threading και της τεχνολογίας 2 πυρήνων, έχει σαν αποτέλεσμα να εκτελούνται 4 εργασίες παράλληλα. Αυτό φαίνεται σχηματικά στο σχήμα 2.



Σχήμα 2

Συνδυασμός τεχνολογιών Hyper Threading και διπλού πυρήνα
για εξυπηρέτηση εργασιών.

1.2 Χρήση ομοχειρίας στους επεξεργαστές

Ο όρος ομοχειρία (Pipeline) αναφέρεται στη τεχνική υλοποίησης κατά την οποία η εκτέλεση εντολών επικαλύπτεται[1]. Με την τεχνική αυτή γίνεται εκμετάλλευση του παραλληλισμού που υπάρχει μεταξύ των διαφόρων ενεργειών που χρειάζονται να γίνουν για να εκτελεστεί μια εντολή, χωρίς να είναι ορατή στο προγραμματιστή. Αποτέλεσμα του πιο πάνω είναι η επιτάχυνση της εκτέλεσης εντολών ανά κύκλο στους επεξεργαστές. Σήμερα η ομοχειρία είναι η βασικότερη τεχνική υλοποίησης στους γρήγορους επεξεργαστές.

Ο πρώτος ομόχειρος επεξεργαστής γενικού σκοπού θεωρείται ο Stretch[1], δηλαδή ο IBM 7030. Στόχος του συγκεκριμένου επεξεργαστή ήταν να αυξηθούν οι επιδόσεις των επεξεργαστών της τότε εποχής. Ακολούθησαν το 1959 οι υλοποιήσεις επεξεργαστών που περιλάμβαναν και τη προώθηση δεδομένων από την έξοδο της ALU. Αργότερα, τις αρχές του 1980, παρουσιάστηκαν και επεξεργαστές που περιλάμβαναν τις περισσότερες από τις απλές τεχνικές που χρησιμοποιούνται σήμερα.

Με την ομοχειρία, κάθε βήμα (βαθμίδα ομοχειρίας) ολοκληρώνει ένα κομμάτι μιας εντολής. Η διασύνδεση των επιμέρους βαθμίδων δημιουργεί μια αλυσίδα ομοχειρίας, στην οποία εντολές εισάγονται από τη μια άκρη επεξεργάζονται από τις βαθμίδες και βγαίνουν από την άλλη άκρη. Ο ρυθμός παραγωγής μιας αλυσίδας ομοχειρίας εξαρτάται από το πόσο συχνά μια εντολή βγαίνει από την αλυσίδα. Ο απαιτούμενος χρόνος μετακίνησης μιας εντολής ανάμεσα σε διαδοχικές βαθμίδες ονομάζεται κύκλος του επεξεργαστή.

Στόχος της ομόχειρης σχεδίασης είναι η αξιοποίηση του μήκους κάθε βαθμίδας ομοχειρίας. Για τέλεια εξισορροπημένες βαθμίδες ο χρόνος που απαιτεί ο ομόχειρος επεξεργαστής για κάθε εντολή είναι:

$$\frac{\text{Χ}\textit{ρόνος} \text{ ανά εντολή στο μη-ομόχειρο επεξεργαστή}}{\textit{αριθμός βαθμίδων} \text{ ομοχειρίας}}$$

Κάτω από αυτές τις συνθήκες, η επιτάχυνση(speed up) λόγω της ομοχειρίας είναι ίση με τον αριθμό των βαθμίδων της. Με τη τεχνική αυτή μειώνεται ο μέσος χρόνος εκτέλεσης εντολών. Αυτό μπορεί να σημαίνει είτε μείωση στον αριθμό των κύκλων ανά εντολή (CPI), όταν κύριο σημείο αναφοράς είναι ο επεξεργαστής που χρειάζεται πολλαπλούς κύκλους ανά εντολή, είτε στη διάρκεια του κύκλου ρολογιού, είτε και στα δύο.

1.3 Σκοπός της Διπλωματικής εργασίας

Σκοπός της παρούσας εργασίας είναι η υλοποίηση του μονοπατιού δεδομένων ενός υποσυνόλου εντολών του ομόχειρου επεξεργαστή DLX, της οικογένειας RISC. Ο επεξεργαστής είναι απλοποιημένος ώστε να εκτελεί μόνο:

- Απλές εντολές πρόσβασης μνήμης (lw, sw)
- Αριθμητικές και Λογικές Εντολές (add, sub, and, or, xor, addi, ori, andi,xori, subi)
- Εντολές ελέγχου εκτέλεσης προγράμματος (beq, bne, j, jr)

Η υλοποίηση της σχεδίασης θα γίνει με τη γλώσσα systemC, που είναι μια βιβλιοθήκη της C++ με τις κατάλληλες δομές έτσι ώστε να είναι εφικτή η περιγραφή υλοποίηση και σύνθεση

ψηφιακών κυκλωμάτων. Αφού γίνει η υλοποίηση και προσομοίωση της σχεδίασης με τα στάνταρ εργαλεία ανάπτυξης της C++ όπως το Visual Studio 6.0 και το πρόγραμμα G.T.K. wave 1.3.19 για εμφάνιση των κυματομορφών προσομοίωσης, χρησιμοποιείται το εργαλείο SystemCrafter, για τη μεταγλώττιση της systemC σε RTL[1] VHDL, έτσι ώστε η σχεδίαση να επιδεχθεί περεταίρω σύνθεσης από εργαλεία downstream και να φορτωθεί σε FPGA. Για Xilinx FPGAs ένα τέτοιο εργαλείο σύνθεσης είναι το Xilinx XST και τα εργαλεία τοποθέτησης και δρομολόγησης (place and route tools). Αφού γίνει η μετατροπή σε RTL VHDL, χρησιμοποιείται το εργαλείο Xilinx XST για τη σύνθεση και προσομοίωση της σχεδίασης σε μια SPARTAN 3 xc3s400 FPGA (Βλέπε παράρτημα Β). Τέλος μετά την επιτυχή σύνθεση, η σχεδίαση φορτώνεται στο αναπτυξιακό της MEMEC που περιέχει την συγκεκριμένη FPGA έτσι ώστε να ελεγχθεί η λειτουργία της σχεδίασης όταν μεταφερθεί και στην FPGA.

Μετά την επιβεβαίωση της λειτουργίας της σχεδίασης στην FPGA, χρησιμοποιούνται οι διακόπτες (4 deep switches) και το 7-segment display που παρέχει το συγκεκριμένο αναπτυξιακό για τον έλεγχο του αναπτυξιακού από το χρήστη έτσι ώστε να ελέγχεται η ροή εκτέλεσης εντολών στο ομόχειρο μονοπάτι δεδομένων του μικροεπεξεργαστή μέσω των διακοπτών, ενώ στο 7-segment display εμφανίζεται σε κάθε περίπτωση κάποιος αριθμός που υποδηλώνει την ενέργεια που ασκείται κάθε στιγμή στο κύκλωμα. Επίσης τα 4 λετάκια που υπάρχουν στο αναπττυξιακό συσχετίζονται με τέσσερις διευθύνσεις του μετρητή προγράμματος, έτσι ώστε καθένα από αυτά να ανάβει όταν ο έλεγχος του προγράμματος περνά από τη συγκεκριμένη διεύθυνση. Με τον τρόπο αυτό δίνεται μια ένδειξη του σημείου στο οποίο βρίσκεται ο έλεγχος κάθε χρονική στιγμή.

Τέλος γίνεται μια αποτίμηση της εργασίας και δίνεται έμφαση στα πλεονεκτήματα – μειονεκτήματα που παρέχει η σχεδίαση και υλοποίηση ενός ψηφιακού κυκλώματος με τη γλώσσα systemC. Γίνεται παραλληλισμός και σύγκριση από προηγούμενη μικρή εμπειρία σχεδίασης ψηφιακών κυκλωμάτων με τη γλώσσα περιγραφής υλικού VHDL.

[1] Είδος περιγραφής γλώσσας υλικού (HDL), που χρησιμοποιείται για την περιγραφή των καταχωρητών ενός υπολογιστικού ή ψηφιακού κυκλώματος, καθώς και για τη περιγραφή του τρόπου με τον οποίο τα δεδομένα μεταφέρονται από τον ένα καταχωρητή στον άλλο.

1.4 Παρουσίαση κεφαλαίων που ακολουθούν

Στο 2ο κεφάλαιο αφού γίνεται μια περιγραφή της γλώσσας systemC. Τι είναι, που αποσκοπεί και πώς χρησιμοποιείται για τη σχεδίαση αρχιτεκτονικών ψηφιακών κυκλωμάτων. Παρατίθενται τα πλεονεκτήματα της σε σχέση με τις γνωστές γλώσσες υψηλού επιπέδου, καθώς και η αντιστοίχηση της με τη γλώσσα περιγραφής υλικού VHDL. Ακολούθως παρουσιάζεται το εργαλείο systemCrafter για τη μετατροπή της systemC σχεδίασης σε RTL VHDL έτσι ώστε να καταστεί δυνατή η περεταίρω σύνθεση της σχεδίασης χρησιμοποιώντας το εργαλείο Xilinx XST και τα εργαλεία τοποθέτησης και δρομολόγησης για Xilinx FPGAs.

Στο 3ο κεφάλαιο περιγράφονται τα βασικά χαρακτηριστικά της αρχιτεκτονικής του DLX. Γίνεται αναφορά στους καταχωρητές που χρησιμοποιούνται, στους τύπους δεδομένων που υποστηρίζει, τον τρόπο που γίνεται η πρόσβαση στις μονάδες μνήμης, την αρίθμηση των bytes και τη αρχιτεκτονική του συνόλου εντολών. Ποιοι τύποι εντολών υπάρχουν, πώς υλοποιούνται οι διακλαδώσεις, τα άλματα, οι συγκρίσεις και το κάλεσμα διαδικασιών.

Στο 4ο κεφάλαιο περιγράφεται η μεθοδολογία υλοποίησης σε systemC ενός υποσυνόλου του συνόλου εντολών του DLX, έτσι ώστε να εκτελούνται μόνο απλές εντολές πρόσβασης μνήμης (lw, sw), αριθμητικές και λογικές εντολές (add, sub, and, or, xor, addi, subi, ori, andi, xori) καθώς και εντολές ελέγχου εκτέλεσης προγράμματος (beq, bne, j, jr). Περιγράφονται ακόμα η μεθοδολογία υλοποίησης του αρχείου καταχωρητών, καθώς και οι τεχνικές της προώθησης δεδομένων όταν υπάρχουν περιορισμοί δεδομένων, καθώς και η μέθοδος της ανάσχεσης που προκαλείται από περιορισμούς δεδομένων.

Στο 5ο κεφάλαιο περιγράφεται το αναπτυξιακό της MEMEC που περιέχει την Xilinx FPGA SPARTAN 3 xc3s400. Περιγράφονται οι διεπαφές του με τον έξω κόσμο, οι θύρες επικοινωνίας με τον υπολογιστή για το προγραμματισμό του, καθώς και η διαδικασία φόρτωσης και ελέγχου της ορθότητας της σχεδίασης στην συγκεκριμένη FPGA. Ο έλεγχος του κυκλώματος γίνεται με τη διασύνδεση του αναπτυξιακού με λογικό αναλυτή[2]. Επίσης περιγράφεται ο τρόπος χρησιμοποίησης των διεπαφών του αναπτυξιακού με το χρήστη όπως

[2] ένας υπολογιστής που λαμβάνει ψηφιακά δεδομένα από ένα ψηφιακό σύστημα (που είναι πολύ γρήγορο για να παρατηρηθεί από ανθρώπινο μάτι) και τα αναπαριστά στο χρήστη έτσι ώστε να μπορεί ο χρήστης να εντοπίσει τυχόν λάθη του ψηφιακού συστήματος.

οι διακόπτες, το 7-segment display και τα λετάκια, έτσι ώστε να ελέγχεται το κύκλωμα μέσω των διακοπτών.

Τέλος στο 6^{ο} κεφάλαιο γίνεται αποτίμηση της εργασίας, εξάγονται συμπεράσματα από την χρησιμοποίηση της systemC ως γλώσσας για τη σχεδίαση υλικού, καθώς και του εργαλείου SC για την μετατροπή σε RTL VHDL και χρησιμοποίηση του παραγόμενου κώδικα σχεδίασης για σύνθεση με τα εργαλεία της Xilinx. Επίσης γίνεται σύγκριση με άλλες γλώσσες περιγραφής υλικού όπως VHDL, όσο αφορά το τρόπο σχεδίασης καθώς και τη μεθοδολογία ελέγχου σφαλμάτων στη σχεδίαση.

ΚΕΦΑΛΑΙΟ 2
Η ΓΛΩΣΣΑ SYSTEMC ΚΑΙ ΤΟ ΕΡΓΑΛΕΙΟ SYSTEMCRAFTER

2.1 Εισαγωγή

Η systemC είναι μια γλώσσα σχεδίασης βασισμένη στη C++. Αποτελεί ουσιαστικά μια βιβλιοθήκη της C++[4, 5]. Με τη βιβλιοθήκη αυτή παρέχεται η δυνατότητα ανάπτυξης μιας αποτελεσματικής μεθοδολογίας για τη σχεδίαση χρονισμένων (cycle accurate) λογισμικών αλγοριθμικών μοντέλων, περιγραφής της αρχιτεκτονικής ενός ψηφιακού κυκλώματος(hardware architecture) και μέσων διασύνδεσης των SoC[3] (system on a Chip) και system-level σχεδιάσεων. Η systemC χρησιμοποιείται σε συνδυασμό με τα στάνταρ εργαλεία ανάπτυξης της γλώσσας C++, για τη δημιουργία system-level μοντέλων, για την γρήγορη προσομοίωση της σχεδίασης έτσι ώστε να επιβεβαιωθεί η λειτουργικότητα της σχεδίασης και να βελτιστοποιηθεί η σχεδίαση και για τη προβολή στους σχεδιαστές υλικού και λογισμικού ενός εκτελέσιμου αρχείου προδιαγραφών (specification of the system), που είναι κατά βάση ένα πρόγραμμα C++, το οποίο έχει την ίδια συμπεριφορά με το προς εκτέλεση σύστημα.

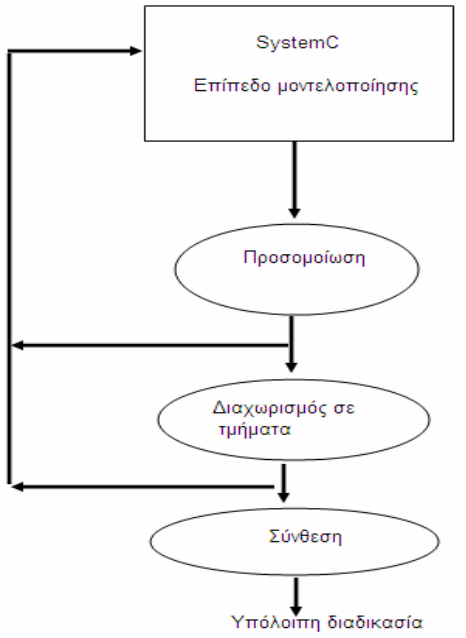
Η γλώσσες C, C++ είναι γλώσσες υψηλού επιπέδου για σχεδίαση προγραμμάτων λογισμικού, και μέσων αλληλεπίδρασης του υλικού με το λογισμικό. Οι γλώσσες αυτές είναι ιδιαίτερα δημοφιλείς στους σχεδιαστές. Η systemC βιβλιοθήκη, παρέχει τις απαραίτητες δομές ενός αρχιτεκτονικού μοντέλου συστήματος, συμπεριλαμβανομένου του χρονισμού του υλικού και τη συνέπεια, δομές που λείπουν από τη στάνταρ C++. Επίσης η systemC προβάλλει και τις απαραίτητες κλάσεις που παρέχει η γλώσσα οντοκεντρικού προγραμματισμού C++, καθιστώντας έτσι εφικτή τη χρησιμοποίηση των δημοφιλών εργαλείων ανάπτυξής της C++.

[3] Ολοκληρωμένο που εμπεριέχει όλο το απαραίτητο υλικό και ηλεκτρονική διασύνδεση ενός ολοκληρωμένου συστήματος. Αποτελείται από μνήμη (RAM και ROM), μικροεπεξεργαστή, περιφεριακές διεπαφές, I/O έλεγχους λογικής, μετατροπείς δεδομένων και άλλων συστατικών που αποτελούν ένα ολοκληρωμένο υπολογιστικό σύστημα.

2.2 Διαφορές systemC με C/C++

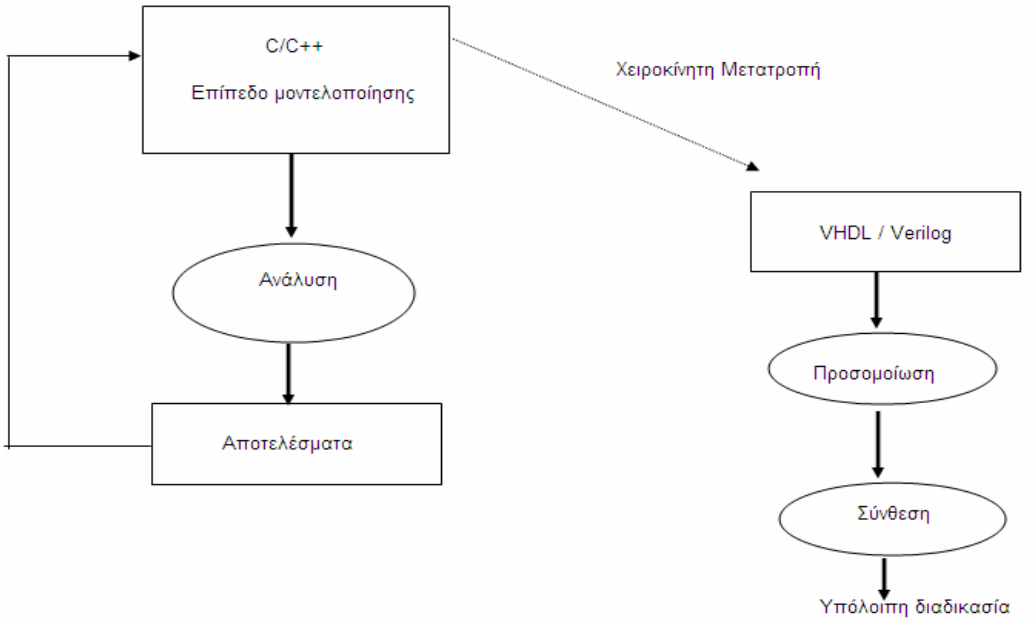
Η systemC όπως προαναφέρθηκε, αν και έχει πολλά κοινά χαρακτηριστικά γνωρίσματα μιας ψηλού επιπέδου γλώσσας προγραμματισμού, μπορεί να αποτελέσει και μια γλώσσα περιγραφής υλικού (HDL). Η μεθοδολογία σχεδίασης ενός ψηφιακού κυκλώματος χρησιμοποιώντας τη γλώσσα systemC φαίνεται στο σχήμα 2-1A, ενώ στο σχήμα 2-1B φαίνεται η μεθοδολογία σχεδίασης χρησιμοποιώντας μια γλώσσα ψηλού επιπέδου (C/C++). Σε σχέση με γνωστές υψηλού επιπέδου γλώσσες προγραμματισμού (C/C++) υπάρχουν μερικές βασικές διαφορές :

- Με τη μεθοδολογία σχεδίασης βασισμένη στη systemC, η σχεδίαση μετατρέπεται από το C επίπεδο στο HDL επίπεδο ως εξής: Η σχεδίαση χωρίζεται σε μικρά τμήματα έτσι ώστε να προστεθούν οι κατάλληλες δομές περιγραφής υλικού και χρονισμού. Με τον τρόπο αυτό ο σχεδιαστής έχει τη δυνατότητα να υλοποιήσει με ευκολία αλλαγές στη σχεδίαση και να ανιχνεύσει bugs κατά το σπάσιμο της σχεδίασης. Αντίθετα στις γλώσσες ψηλού επιπέδου η μετατροπή από το C επίπεδο στο HDL επίπεδο γίνεται σε μια μεγάλη προσπάθεια.
- Δεν χρειάζεται η γνώση πολλαπλών γλωσσών με την χρησιμοποίηση της systemC, αφού επιτρέπει τη μοντελοποίηση από το επίπεδο ελέγχου στο RTL επίπεδο εάν χρειάζεται.
- Επιπλέον πλεονέκτημα είναι το γεγονός ότι η σχεδίαση του μοντέλου γίνεται σε ψηλό επίπεδο. Αυτό έχει ως αποτέλεσμα να απαιτείται μικρότερος κώδικας, που υλοποιείται ευκολότερα και προσομοιώνεται γρηγορότερα από τους παραδοσιακούς τρόπους.
- Άλλο σημαντικό πλεονέκτημα είναι το ότι τα αρχεία ελέγχου (test benches) μπορούν να επαναχρησιμοποιηθούν από το C μοντέλο στο RTL μοντέλο, αφενός μεν εξοικονομώντας χρόνο, και αφετέρου εξασφαλίζουν στο σχεδιαστή τη πεποίθηση ότι η μετατροπή από το C επίπεδο στο RTL επίπεδο εξασφαλίζει την ίδια λειτουργικότητα. Αντίθετα κατά τη σχεδίαση με γλώσσες υψηλού επιπέδου, απαιτούνται διαφορετικά αρχεία ελέγχου για τα δύο στάδια.



Σχήμα 2-1 Α

Μεθοδολογία σχεδίασης ενός ψηφιακού κυκλώματος χρησιμοποιώντας τη γλώσσα systemC.



Σχήμα 2-1 Β

Μεθοδολογία σχεδίασης ενός ψηφιακού κυκλώματος χρησιμοποιώντας γλώσσα υψηλού επιπέδου C/C++.

2.3 Βασικά Χαρακτηριστικά

Η systemC υποστηρίζει τη περιγραφή υλικού, λογισμικού και διεπαφών αλληλεπίδρασης στο περιβάλλον της C++. Παρακάτω παρατίθενται τα βασικά γνωρίσματα της systemC v.2.0[4] που την καθιστούν μια γλώσσα σχεδίασης (co-design):

- Μονάδα (module): Είναι μια κλάση που περιγράφει μια ιεραρχική οντότητα, η οποία μπορεί να εμπεριέχει άλλες υπομονάδες(sub modules) ή άλλες διαδικασίες (processes).
- Διαδικασία (process): χρησιμοποιείται για να περιγράψει λειτουργικότητα. Εμπεριέχεται στο εσωτερικό μιας μονάδας. Οι διαδικασίες καλούνται όποτε τα σήματα στα οποία είναι «ευαίσθητες» αλλάζουν τιμή. Η systemC παρέχει τρεις διαφορετικούς τύπους διαδικασιών που μπορούν να χρησιμοποιηθούν από σχεδιαστές υλικού και λογισμικού. Αυτές είναι οι εξής:
    1. Methods: Στις διαδικασίες αυτές, όταν συμβαίνουν κάποια γεγονότα (αλλαγή τιμών) σε σήματα στα οποία οι διαδικασίες είναι ευαίσθητες, εκτελείται η συγκεκριμένη διαδικασία. Μετά το πέρας της εκτέλεσης ο έλεγχος μεταφέρεται πίσω στο πυρήνα της systemC.
    2. Threads: Οι διαδικασίες αυτές μπορούν να παρακολουθούνται και να ξαναενεργοποιούνται. Η παρακολούθηση γίνεται με την εισαγωγή των wait() συναρτήσεων, οι οποίες παρακολουθούν την διαδικασία εκτέλεσης έως ότου συμβεί κάποιο γεγονός σε κάποιο από τα σήματα στα οποία είναι «ευαίσθητη». Η ξαναενεργοποίηση μιας thread διαδικασίας, συνεχίζει από το σημείο στο οποίο σταμάτησε προηγούμενο κάλεσμά της. Η διαδικασία εκτελείται μέχρι το επόμενο wait().
    3. Clocked Threads: Αποτελούν ειδικές περιπτώσεις των thread διαδικασιών. Σκοπός τους είναι να βοηθούν τους σχεδιαστές να περιγράψουν τις υλοποιήσεις τους για καλύτερα αποτελέσματα σύνθεσης. Ελέγχονται μόνο από μια ακμή (θετική ή αρνητική) του

ρολογιού, πράγμα που αντιστοιχίζει τον τρόπο που υλοποιείται το hardware με εργαλεία σύνθεσης.

- Πύλη (Port): είναι τμήματα των μονάδων που επιτρέπουν τη επικοινωνία διαφορετικών μονάδων μεταξύ τους. Υποστηρίζονται μονής-κατεύθυνσης πύλες(input, output) και διπλής-κατεύθυνσης πύλες(inout).
- Σήματα (signals): Χρησιμοποιούνται για τη συνένωση πυλών μονάδων χαμηλότερου επιπέδου. Αναπαριστούν ουσιαστικά τα φυσικά καλώδια για τη διασύνδεση συσκευών που αποτελούν μια φυσική υλοποίησης σχεδίασης. Μεταφέρουν δεδομένα, ενώ οι πύλες καθορίζουν τη διεύθυνση που ακολουθούν τα δεδομένα από τη μια μονάδα στην άλλη. Υπάρχουν δύο ειδών: Τα resolved σήματα τα οποία μπορούν να έχουν περισσότερους από ένα οδηγούς (buses) και στα unresolved σήματα τα οποία μπορούν να έχουν μόνο ένα οδηγό.
- Αποδοτικό σύνολο από τύπους πυλών και σημάτων: με τον τρόπο αυτό πετυχαίνεται η μοντελοποίηση σε διαφορετικά επίπεδα αφαίρεσης από λειτουργική systemC σε RTL (Register Transfer Language). Η systemC σε αντίθεση με γλώσσες όπως η Verilog, υποστηρίζει και two-valued τύπους και four-valued.
- Αποδοτικό σύνολο από τύπους δεδομένων: Η γκάμα των τύπων δεδομένων της systemC, υποστηρίζει πολλαπλά σχεδιαστικά επίπεδα πεδία και επίπεδα αφαίρεσης. Οι αμετάβλητης ακρίβειας (fixed precision) τύποι δεδομένων, επιτρέπουν γρήγορη προσομοίωση, οι μεταβλητής ακρίβειας τύποι δεδομένων, χρησιμοποιούνται για υπολογισμούς με μεγάλους αριθμούς, οι fixed-point τύποι δεδομένων χρησιμοποιούνται για εφαρμογές DSP. Υποστηρίζονται και two-valued τύπους και four-valued τύποι δεδομένων.
- Ρολόι: υπάρχουν δύο έννοιες του ρολογιού. Ως ειδικά σήματα που κρατούν το χρόνο του συστήματος κατά την προσομοίωση και ως σήματα με μεταβλητή φάση.
- Πολλαπλά επίπεδα αφαίρεσης: Υποστηρίζονται άκαιρα μοντέλα σε διαφορετικά επίπεδα αφαίρεσης, διαβαθμιζόμενα από υψηλού επιπέδου λειτουργικά μοντέλα, έως και συνεπή ως προς το ρολόι RTL μοντέλα.

- Διόρθωση λαθών προγράμματος (debugging): όλες οι κλάσεις της systemC παρέχουν έλεγχο λαθών σε πραγματικό χρόνο.
- Παρακολούθηση μέσω κυματομορφών: Υποστηρίζεται η παρακολούθηση κυματομορφών του τύπου VCD, WIF και ISDB.

2.4 Αντιστοίχηση κώδικα SystemC - VHDL

Όπως προαναφέρθηκε η systemC είναι μια γλώσσα που έχει πολλά κοινά χαρακτηριστικά με τη C++, αφού αποτελεί και μια από τις βιβλιοθήκες της. Επίσης σαν γλώσσα περιγραφής υλικού έχει πάρα πολλά κοινά χαρακτηριστικά με τις γλώσσες περιγραφής υλικού VHDL και Verilog. Έτσι εάν κάποιος είναι εξοικειωμένος με τη γλώσσα C++ και μια εκ των VHDL ή Verilog είναι πάρα πολύ εύκολο να εξοικειωθεί στο προγραμματισμό με τη systemC. Στο παρακάτω πίνακα παρουσιάζεται η αντιστοίχηση του κώδικα υλοποίησης ενός ψηφιακού κυκλώματος σε VHDL, με τον αντίστοιχο κώδικα υλοποίησης σε systemC.

Με έντονα γράμματα είναι οι λέξεις κλειδιά της κάθε γλώσσας.
Με italics είναι ονοματολογία που χρησιμοποιεί ο χρήστης κατά την υλοποίηση.

Task	VHDL	SystemC
Basic Model Structure	**library** IEEE; **use** IEEE.std_logic_1164.all; **entity** *my_model* **is** **port**(*input1*: **in STD_LOGIC;** *input2*: **in STD_LOGIC;** *output1*: **out STD_LOGIC;** *output2*: **out STD_LOGIC;** **);** **end** *my_model*; **architecture** *my_arch* **of** *my_model* **is** **begin** **process(** *input1, input2***)** **variable** *my_var1,my_var2*: **STD_LOGIC;** **begin** *my_var1* := **not** *input1*; *my_var2* := **not** *input2*;	**#include** “systemc.h” **SC_MODULE** (*my_model)* { **sc_in<sc_logic>** *input1***;** **sc_in<sc_logic>** *input2***;** **sc_out<sc_logic>** *output1***;** **sc_out<sc_logic>** *output2***;** **SC_CTOR** (*my_model)* { **SC_METHOD** (process); **sensitive** << *input1* << *input2*; } **void** process( ) { **sc_logic** *my_var1*, *my_var2*; *my_var1* = ~*input1*;

	output1 <= *input1* **and** *my_var2*; *output2* <= *input2* **and** *my_var1*; **end process;** **end** *my_arch*;	*my_var2* = ~*input2*; *output1* = *input1* & *my_var2*; *output2* = *input2* & *my_var1*; } };
Logic Values	'X', '0', '1', 'Z'	"SC_LOGIC_X" , "SC_LOGIC_0" "SC_LOGIC_1", "SC_LOGIC_Z"
Variable:Type Logic Bit	**variable** *my_bit*: **STD_LOGIC;**	**sc_logic** *my_bit*;
variable: 3-bit Logic Vector	**variable** *my_vector*: **STD_LOGIC_VECTOR**(2 **downto** 0) ;	**sc_lv**<3> *my_vector*;
Signal of Type Logic	**signal** *my_signal*: **STD_LOGIC;**	**sc_signal<sc_logic>** *my_signal*;
Signal for a 3-bit Logic Vector	**signal** *my_signal*: **STD_LOGIC_VECTOR** (2 **downto** 0);	**sc_signal<sc_lv<3> >** *my_signal*;
Input Ports – Logic Type	*input1*: **in STD_LOGIC;** *input2,input3,input4*: **in STD_LOGIC;**	**sc_in<sc_logic>** *input1*; **sc_in<sc_logic>** *input1,input2,input3*;
Output Ports – Logic Type	*output1*: **out STD_LOGIC;** *output2,output3,output4*: **out STD_LOGIC;**	**sc_out<sc_logic>** *output1*; **sc_out<sc_logic>** *output1,output2,output3*;
Process & Sensitivity List	**process**(*input1, input2*)	**SC_METHOD** (process); **sensitive** << *input1* << *input2*;
Process: Positive Edge-Triggered	**[no predefined function-must use this code]** **process**(clk) . . . **if** (clk'event **and** clk = '1') **then** . . .	**SC_METHOD** (process); **sensitive_pos** << *clk*;
Process: Negative Edge-Triggered	**[no predefined function-must use following code]** **process**(clk) . . . **if** (clk'event **and** clk = '0') **then** . . .	**SC_METHOD** (process); **sensitive_neg** << *clk*;
"Not" Input1	**not** input1	~input1
Input1 "AND" Input2	*...input1* **and** *input2;*	*...input1* **&** *input2;*
Input1 "OR" Input2	*...input1* **or** *input2;*	*...input1* | *input2;*
Input1 "XOR" Input2	*...input1* **xor** *input2;*	*...input1* ^ *input2;*
Input1 "NAND" Input2	*...input1* **nand** *input2;*	**[no operator-must use 2 statements of code]** *... input1 & input2;* *... ~input;*
Input1 "NOR" Input2	*...input1* **nor** *input2;*	**[no operator-must use 2 statements of code]** *... input1* | *input2;* *... ~input;*
Input1 "XNOR" Input2	*...input1* **xnor** *input2;*	**[no operator-must use 2 statements of code]** *... input1 ^ input2;* *... ~input;*
Variable Assignment	*var1 := input1 and input2;*	*var1 = input1 & input2;*
Signal	*sig1 <= input1 and input2;*	*sig1 = input1 & input2;*

Assignment		
Accessing a Value From a Port	**variable** *temp*; *temp = input1;*	**sc_logic** *temp**;*** *temp = input1.**read()** ;*
Equality Operator	= *example:* **if** (*ENABLE* = '1') **then**	== *example*: **if** (ENABLE == SC_LOGIC_1);
Arithmetic Operations	use IEEE.std_logic_arith.all; ... *input1 = input1 + 1;* *input1 = input1 - 1;*	**[no header file for arithmetic on logic types, must convert to unsigned integer type first]** **sc_uint** *temp**;*** *temp = input1.**read()** ;* *temp = temp + 1;* *temp = temp – 1;* *input1 = temp;*
If ... Else... Structure	**if**(*E*='1') **then** *Q = D*; *Q_P* = **not** *D*; **elsif**(*E*='0')**then** *Q_P = D*; *Q* = **not** *D*; **else** *Q = D*; *Q_P* = **not** *D*; **end if**;	**if**(*E*='1') { *Q = D*; *Q_P = ~D*; } **elsif**(*E*='0') { *Q_P = D*; *Q = ~D*; } **else** { *Q = D*; *Q_P = ~D*;}
Switch/Case Statement	**case** *ctrl* **is** **when** "0" => *f* <= *a*; **when** "1" => *f* <= *b*; **end case**;	**[illegal to use logic type in switch statement – must use temp variable]** **sc_uint**<2> temp = ctrl.read(); **switch**(temp) { **case** 0: f = a; **break**; **case** 1: f = b; **break**; }

2.5 Το εργαλείο SystemCrafter (SC)

2.5.1 Γενικά στοιχεία του SC

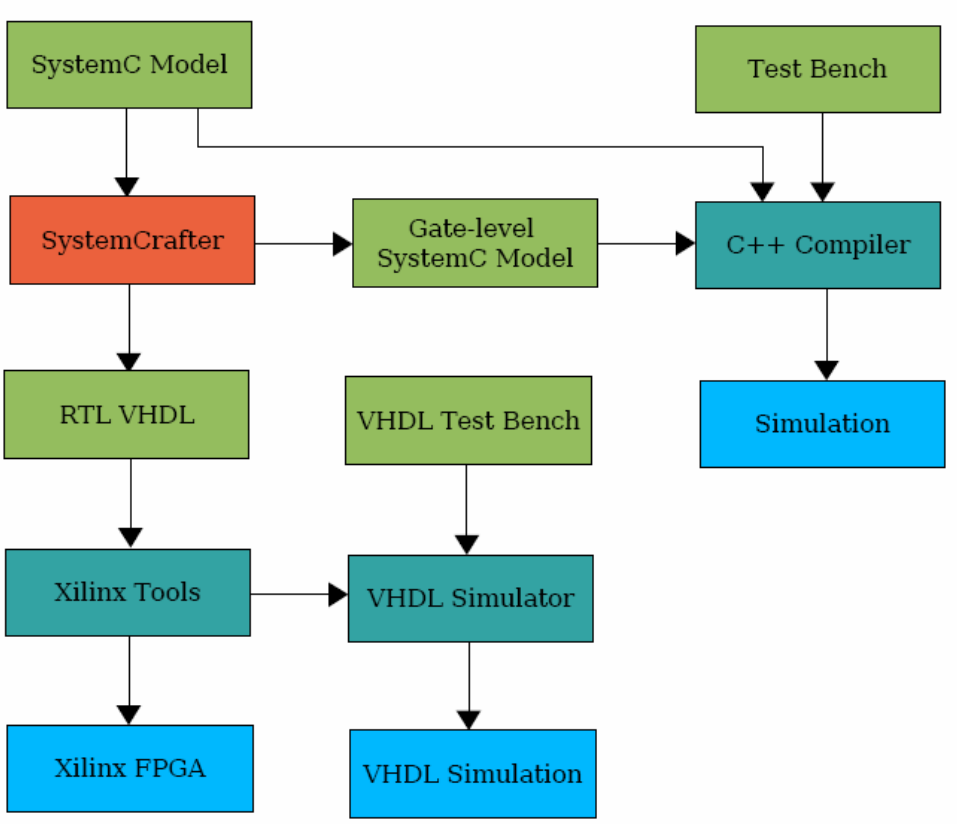
Για να έχει αξία η σχεδίαση ενός αρχιτεκτονικού μοντέλου σε systemC, πρέπει αυτό με κάποιο τρόπο να μετατραπεί σε RTL VHDL ή Verilog, έτσι ώστε να επιδεχθεί περαιτέρω σύνθεσης από εργαλεία κατώτερου επιπέδου. Για να γίνει αυτό πρέπει η systemC σχεδίαση να είναι σε μορφή που να μετατρέπεται σε υλικό. Αυτό διότι η systemC είναι ένα υπερσύνολο της C++ που αρχικά σχεδιάστηκε για προσομοίωση μόνο. Έτσι κάποιες από τις δομές της είναι αδύνατο να μετατραπούν σε υλικό.

Ένα εργαλείο σύνθεσης της systemC που παρέχει τη δυνατότητα μετατροπής της systemC σε RTL VHDL κάτω από κάποιες προϋποθέσεις, είναι ο systemCrafter (SC)[6]. Η μετατροπή μπορεί να καταστεί εφικτή μόνο εάν χρησιμοποιηθούν οι δομές που υποστηρίζει ο SC. Μπορεί να χρησιμοποιηθεί σε ένα διάγραμμα σχεδίασης υλικού για να αποτελέσει ένα εργαλείο εισόδου που θα χρησιμοποιηθεί σαν προπομπός σε εργαλεία σύνθεσης της Xilinx. Επίσης μπορεί να χρησιμοποιηθεί σε διαγράμματα σχεδίασης υλικού/λογισμικού (hardware/software co-design flow) έτσι ώστε να παραχθεί αυτόματα γλώσσα περιγραφής υλικού από επιλεγμένα τμήματα του system-level μοντέλου γραμμένου σε systemC. Το διάγραμμα σχεδίασης της μετατροπής της systemC σε υλικό φαίνεται στο σχήμα 2-2.

Με το εργαλείο αυτό εκτελούνται οι εξής ενέργειες:

- αρχικά ο σχεδιαστής περιγράφει ένα ψηφιακό κύκλωμα σε systemC, χρησιμοποιώντας ένα αρχείο ελέγχου(testbench) και το μεταγλωτιστή της C++ για προσομοίωση της σχεδίασης.
- Εάν τα αποτελέσματα της σχεδίασης είναι ικανοποιητικά, μέσω του systemCrafter γίνεται μετατροπή του systemC κώδικα σε RTL VHDL. Η παραγόμενη σχεδίαση μπορεί να επιδεχθεί περεταίρω σύνθεσης χρησιμοποιώντας στάνταρ εργαλεία σύνθεσης.
- Για Xilinx FPGAs ένα τέτοιο εργαλείο σύνθεσης είναι το Xilinx XST και τα εργαλεία τοποθέτησης και δρομολόγησης (place and route tools).

- Ο systemCrafer παρέχει επίσης και μια περιγραφή του επιπέδου-πυλών (gate-level) του παραγόμενου VHDL αρχείου. Αυτό μπορεί να χρησιμοποιηθεί για επιβεβαίωση στο αυθεντικό αρχείο ελέγχου της systemC.



Σχήμα 2-2

Το διάγραμμα σχεδίασης για μετατροπή της systemC σε υλικό.

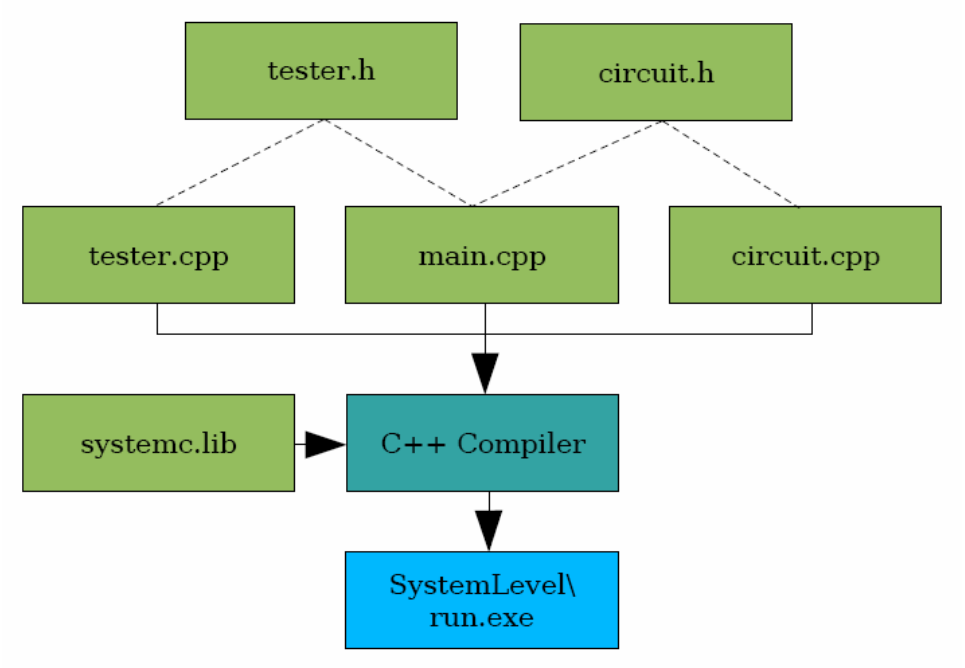
2.5.2 Χρησιμοποίηση SC στη σχεδίαση ψηφιακών κυκλωμάτων

Η περιγραφή ενός ψηφιακού κυκλώματος, μαζί με το αρχείο ελέγχου, χωρίζεται στα εξής αρχεία: circuitname.h, circuitname.cpp, tester.h, tester.cpp και main.cpp. Στο circuitname.h αρχείο δηλώνεται το όνομα του κυκλώματος, οι είσοδοι και έξοδοί του, καθώς και οι διαδικασίες στις οποίες είναι «ευαίσθητο». Στο circuitname.cpp αρχείο περιγράφονται οι διαδικασίες που συμβαίνουν όταν ο έλεγχος μεταφερθεί στην εκτέλεση των διαδικασιών. Στο αρχείο ελέγχου tester.h δηλώνονται οι έξοδοι που θα τροφοδοτούν τις εισόδους του

κυκλώματος καθώς και οι είσοδοι οι οποίες διαβάζουν τις εξόδους του κυκλώματος. Στο αρχείο ελέγχου tester.cpp γίνεται τροφοδοσία των εισόδων του κυκλώματος και εξαγωγή των αποτελεσμάτων στις εξόδους. Τέλος στο αρχείο main.cpp, γίνεται συνένωση του αρχείου ελέγχου με το κύκλωμα.

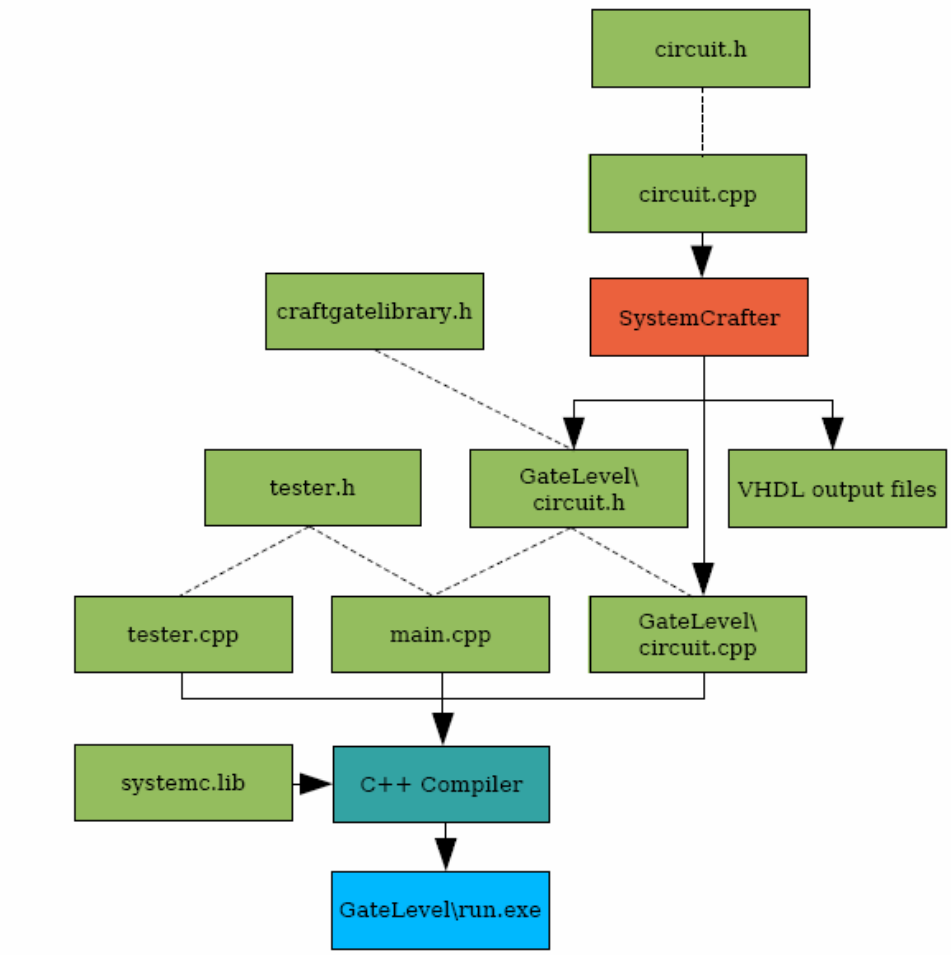
Κατά τη φάση της μεταγλώττισης του systemC αρχείου περιγραφής του κυκλώματος σε υλικό, η μακροεντολή SC_SYNTHESIS, δηλώνεται από το προεπεξεργαστή. Αυτό γίνεται για να συμπεριληφθεί (#include) το αρχείο systemc.h της βιβλιοθήκης systemC για προσομοίωση και όχι σύνθεση.

Χρησιμοποιούνται δύο ρεύματα για τη σύνθεση και επιβεβαίωση της systemC σχεδίασης. Το πρώτο "system-level"(σχήμα 2-3), προσομοιώνει άμεσα τα αρχεία της σχεδίασης. Το δεύτερο, "gate-level"(σχήμα 2-4), τρέχει το systemCtrafter SC για τη σύνθεση της σχεδίασης και μετά προσομοιώνει το συνδεθέν κύκλωμα. Το κύκλωμα αυτό μπορεί να προσομοιωθεί για επιβεβαίωση της λειτουργίας του. Τα αρχεία GateLevel\circuit.cpp, main.cpp και tester.cpp μεταγλωττίζονται και διασυνδέονται με τη systemC βιβλιοθήκη έτσι ώστε να παραχθεί το εκτελέσιμο αρχείο για προσομοίωση. Στη περίπτωση αυτή το αρχείο main.cpp συμπεριλαμβάνει το συνδεθέν header αρχείο GateLevel\circuit.h, ενώ το GateLevel\circuit.h εμπεριέχει μια βιβλιοθήκη της systemC για περιγραφή πυλών, το craft_gatelibrary.vhd. Μετά τη gate-level προσομοίωση, δημιουργούνται τα VHDL αρχεία που περιγράφουν τη σχεδίαση. Αυτά, μαζί με το αρχείο craft_gatelibrary.vhd, δίνουν τη δυνατότητα της περεταίρω σύνθεσης, τοποθέτησης και δρομολόγησης του κυκλώματος.



Σχήμα 2-3

Διάγραμμα ροής system-level σχεδίασης. Τα αρχεία circuit.cpp, main.cpp, tester.cpp, μεταγλωττίζονται άμεσα χρησιμοποιώντας το C++ μεταγλωττιστή και μετά συντίθενται με τη βιβλιοθήκη systemC έτσι ώστε να παραχθεί ένα εκτελέσιμο αρχείο προσομοίωσης.



Σχήμα 2-4

Διάγραμμα ροής gate-level σχεδίασης. Εδώ χρησιμοποιείται και ο SC ως εξής: Πρώτα ο SC χρησιμοποιείται για τη σύνθεση του circuit.cpp αρχείου (και του εμπεριεχομένου circuit.h αρχείου). Από τη σύνθεση αυτή προκύπτει μια systemC περιγραφή του συντιθέμενου κυκλώματος, το οποίο τοποθετείται στην υποδιεύθυνση "GateLevel", παράγοντας τα αρχεία GateLevel/circuit.cpp και GateLevel/circuit.h. Πέρα από αυτό δημιουργούνται και τα αντοίστιχα VHDL αρχεία που θα χρησιμοποιηθούν με τα στάνταρ εργαλεία σύνθεσης. Τα αρχεία αυτά μαζί με το αρχείο craft_gatelibrary.vhd μπορούν να συντεθούν σε FPGAs, χρησιμοποιώντας τα εργαλεία σύνθεσης Xilinx XST.

2.5.3 Υποσύνολο systemC δομών που υποστηρίζεται από SC

Η παρούσα έκδοση του SC υποστηρίζει το εξής υποσύνολο της sytemC γλώσσας (δομές και περιορισμοί που διέπουν το SC):

2.5.3.1 Τύποι δεδομένων

Υποστηρίζονται οι ακόλουθοι τύποι δεδομένων:

- sc_int, sc_uint, int, unsigned, bool, sc_bit, sc_logic, sc_lv, sc_bv, sc_bigint, sc_biguint, short, long, char, unsigned int, unsigned long, unsigned char.
- Σταθεροί (const) qualifiers
- Μονής διεύθυνσης πίνακες (Single dimensional arrays).
- Η διαδικασία enum

Δεν υποστηρίζονται οι ακόλουθοι τύποι δεδομένων:

- User-defined classes, multi-dimensional arrays, pointers, references, fixed point, floating point, functional-style constructors (π.χ. sc_uint<3>(5)), typedef.

2.5.3.2 Εκφράσεις

Υποστηρίζονται οι εξής εκφράσεις:

- Μοναδιαίες εκφράσεις: + - ! ~
- Δυαδικές εκφράσεις: * / % + - << >> < > <= >= & ^ | && || == !=
- Εκφράσεις ανάθεσης: = *= /= %= += -= <<= >>= &= ^= |=
- Εκφράσεις συνθήκης: x?y:z
- Postincrement, postdecrement, preincrement και predecrement.
- .or_reduce() .xor_reduce() .and_reduce() .range(x,1) (x,y)(σύνδεση).
- a[x] όπου a είναι πίνακας (μονής διεύθυνσης άδεια εισόδου πίνακα)
- a[x] όπου a είναι μεταβλητή του κατάλληλου τύπου δεομένων (bit selection)

- x.read(), x.write(), όπου x είναι πόρτα.
- Δεν υποστηρίζονται πολλαπλής διεύθυνσης πίνακες.
- Δεν υποστηρίζεται επίσης το casting, αλλά οι τύποι δεδομένων μετατρέπονται αυτόματα όταν χρειάζεται.

2.5.3.3 Δηλώσεις

Υποστηρίζονται οι εξής δηλώσεις:

- Δηλώσεις μεταφοράς ελέγχου προγράμματος (labeled statememts): case, default
- Δηλώσεις επιλογής : if, if else, switch
- Δηλώσεις αλληλεπίδρασης : while, do/while, for. Η τελευταία δήλωση σε περιπτώσεις ανακύκλωσης πρέπει να είναι μια wait() δήλωση.
- Δηλώσεις άλματος: break (υποστηρίζεται μόνο από switch δήλωση), return.
- wait() χρησιμοποιείται για να υποδηλώσει μετάβαση ρολογιού.

Δεν υποστηρίζονται οι εξής δηλώσεις:

- continue
- break (όταν βρίσκονται εκτός της δήλωσης switch).

2.5.3.3 Κλάσεις, δομές, συναρτήσεις

- Οι κλάσεις και οι δομές πρέπει να είναι υποκλάσεις μιας μονάδας. Μπορούν να δηλωθούν και να συνενωθούν αλλά όχι να χρησιμοποιηθούν ως τύποι δεδομένων. Μπορούν να χρησιμοποιηθούν οι ακόλουθοι τύποι θυρών και σημάτων: sc_in, sc_out, sc_signal.
- Η διαδικασία SC_THREAD χρησιμοποιείται για να επισυνάπτει συναρτήσεις μέλους στις κλάσεις.

- Οι λέξεις κλειδιά sensitive_pos και sensitive_neg χρησιμοποιούνται για να σχετίζουν το ρολόι με μια διαδικασία.
- Όλες οι διαδικασίες σε μια κλάση πρέπει να είναι χρονισμένες κάτω από το ίδιο ρολόι της ίδιας πόλωσης.
- Δήλωση μεταβλητών και μεθόδων μπορεί να χρησιμοποιηθεί.
- Δεν υποστηρίζονται οι εξής τύποι θυρών, σημάτων: sc_inout, overloading, templates, inheritance, unions, SC_CTHREAD, SC_METHOD.
- Κλάσεις και δομές δεν μπορούν να χρησιμοποιηθούν σαν τύποι δεδομένων.
- Namespaces, and ένθετα ονόματα, δεν υποστηρίζονται.

Σε ότι αφορά την υλοποίηση της παρούσας εργασίας, λαμβάνοντας υπόψη τις δομές και τους περιορισμούς του SC, αφού υλοποιήθηκε σε systemC ο κώδικας που μοντελοποιεί το μονοπάτι δεδομένων ενός υποσυνόλου εντολών του μικροεπεξεργαστή DLX, με τρόπο που να είναι μετατρέψιμος σε υλικό, χρησιμοποιήθηκε ο SC για τη μετατροπή της σχεδίασης σε RTL VHDL, έτσι που να καταστεί εφικτή η περεταίρω σύνθεση του χρησιμοποιώντας το εργαλείο σύνθεσης Xilinx XST.

ΚΕΦΑΛΑΙΟ 3
Η ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ DLX

3.1 Εισαγωγή

Ο DLX[7] είναι ένας μικροεπεξεργαστής της οικογένειας RISC (Βλέπε παράρτημα Β), που σχεδιάστηκε από τους John L. Hennessy και David A. Patterson, προϊστάμενους σχεδίασης του MIPS και του Berkeley RISC (δύο παραδείγματα σχεδίασης επιδόσεων του RISC). Αποτελεί ουσιαστικά ένα απλοποιημένο επεξεργαστή MIPS, με απλή 32-bit load/store αρχιτεκτονική.

Η επιλογή της αρχιτεκτονικής του DLX βασίστηκε σε παρατηρήσεις σχετικά με τη συχνότητα χρησιμοποίησης σε προγράμματα στοιχειωδών εντολών. Το μοντέλο του DLX είναι ένα καλό αρχιτεκτονικό μοντέλο για μελέτη για τους εξής λόγους:

- Αυτού του τύπου η μηχανή είναι δημοφιλής
- Είναι εύκολα κατανοητή η σχεδίαση
- Δίνεται έμφαση στην απλότητα του load/store συνόλου εντολών.
- Παρέχει την τεχνική της ομοχειρίας.
- Έχει ένα εύκολο από πλευράς αποκωδικοποίησης, σύνολο εντολών.
- Χρησιμοποιείται για εκπαιδευτικούς σκοπούς σε πανεπιστημιακού επιπέδου μαθήματα αρχιτεκτονικής υπολογιστών.

Ο DLX, όπως και ο MIPS, βασίζεται στη τεχνική της ομοχειρίας. Για το λόγο αυτό αποτελεί ένα τυπικό RISC επεξεργαστή. Τα πέντε βασικά στάδια ομοχειρίας του DLX, είναι τα εξής:

- IF – Μονάδα ανάκλησης εντολής. Συχνά αναφέρεται και ως «the load unit».
- ID – Μονάδα αποκωδικοποίησης η οποία παίρνει την εντολή από το προηγούμενο στάδιο και εξάγει το opcode της καθώς και τους τελεσταίους της.
- EX – Μονάδα εκτέλεσης όπου εκτελεί τις ενέργειες κάθε εντολής. Συχνά αναφέρεται και ως ALU.

- MEM – Μονάδα πρόσβασης μνήμης όπου ανακτά δεδομένα από τη κύρια μνήμη κάτω από τον έλεγχο των εντολών από προηγούμενα στάδια (ID, EX).

- WB – Μονάδα εγγραγής αποτελέσματος. Συχνά αναφέρεται και ως μονάδα αποθήκευσης (the store unit).

3.2 Καταχωρητές στον DLX

Ο DLX[8, 9,10] έχει

- τριάντα δύο 32-bit καταχωρητές γενικού σκοπού (general purpose registers), που ονομάζονται R0, R1, …, R31. Ο καταχωρητής R0 έχει τιμή πάντοτε μηδέν. Ο καταχωρητής R31 είναι ο καταχωρητής επιστροφής διεύθυνσης.
- Τριάντα δύο 32-bit καταχωρητές κινητής υποδιαστολής (floating point registers) που ονομάζονται F0, F1, …, F31. Οι καταχωρητές αυτοί μπορούν να αποθηκεύσουν 32 τιμές απλής ακρίβειας (32 bit) ή 32 τιμές διπλής ακρίβειας (64 bit). Όταν αποθηκεύεται ένας αριθμός απλής ακρίβειας το άλλο μισό του καταχωρητή κινητής υποδιαστολής μένει αχρησιμοποίητο.
- Κάποιους επιπλέον ειδικούς καταχωρητές, καθώς και τον μετρητή προγράμματος PC (program counter).
- Η διευθυνσιοδότηση για τις εντολές πρόσβασης μνήμης (load/store) γίνεται χωρίς αλλαγή κατεύθυνσης στο μονοπάτι δεδομένων (indirection). Μόνο με βάση + displacement.
- Οι συνθήκες για περιπτώσεις διακλάδωσης είναι απλές

3.3 Τύποι δεδομένων στον DLX

Για δεδομένα ακεραίων υπάρχουν:

- Bytes των 8-bit
- Μισές λέξεις των 16-bit

- Λέξεις των 32-bit

Για δεδομένα κινητής υποδιαστολής υπάρχουν

- 32-bit απλής ακρίβειας (single precision)
- 64-bit διπλής ακρίβειας (double precision)

Όλες οι λειτουργίες όπως πρόσθεση, αφαίρεση κ.τ.λ. είτε δουλεύουν με βάση 32-bit ακέραιους, είτε με 32 ή 64 -bit αριθμούς κινητής υποδιαστολής. Οι μισές λέξεις (half words) ή τα bytes φορτώνονται στους καταχωρητές γενικού σκοπού είτε με συμπλήρωση μηδενικών (zero filling) όπου συμπληρώνονται τα bits που απαιτούνται για να γίνει η 32-bit ποσότητα με μηδενικά, είτε με προέκταση πρόσημου (sign extension) όπου τα bits που απαιτούνται για να συμπληρωθεί η 32-bit ποσότητα γεμίζουν με το bit πρόσημου ($16^{ο}$ bit της εντολής).

3.4 Πρόσβαση μνήμης στον DLX

3.4.1 Διευθύνσεις Bytes και Περιορισμοί Ευθυγράμμισης

Οι διευθύνσεις μνήμης στον DLX αναφέρονται σε bytes στη μνήμη (Byte addressable). Έτσι, μια 32-bit λέξη καταλαμβάνει 4 "θέσεις μνήμης" (4 bytes). Κατά συνέπεια, ένας πίνακας (array) μεγέθους 100 ακεραίων "πιάνει" 400 (συνεχόμενες) διευθύνσεις (θέσεις) στη μνήμη. Σ' ένα τέτοιο πίνακα, η διεύθυνση του κάθε ακέραιου διαφέρει από αυτήν του διπλανού του κατά 4. Εαν A_0 είναι η διεύθυνση του "πρώτου" (μηδενικού) στοιχείου, a[0], ενός πίνακα ακεραίων της C, τότε το στοιχείο a[i] του πίνακα αυτού θα βρίσκεται στη διεύθυνση (A_0 + 4*i). Αν ο πίνακας αυτός ήταν πίνακας χαρακτήρων (char), τότε το στοιχείο a[i] θα ήταν στη διεύθυνση (A_0 + i).

Τα 4 bytes που αποτελούν έναν ακέραιο έχουν διευθύνσεις που είναι συνεχόμενοι αριθμοί. Διεύθυνση του ακεραίου είναι η διεύθυνση εκείνου από τα 4 bytes του που έχει τη μικρότερη ("πρώτη") από τις 4 διευθύνσεις. Ο DLX επιβάλει περιορισμούς ευθυγράμμισης (alignment restrictions) στις ποσότητες που προσπελαύνουν οι εντολές load και store στη μνήμη: μία ποσότητα μεγέθους *N* bytes επιβάλλεται να έχει διεύθυνση που να είναι ακέραιο πολλαπλάσιο του *N*. Ετσι, όταν το *N* είναι δύναμη του 2, η διεύθυνση κάθε τέτοιας ποσότητας τελειώνει σ' ένα αντίστοιχο πλήθος μηδενικών, και η διεύθυνση των υπολοίπων bytes της ποσότητας διαφέρει μόνο σε αυτά τα λιγότερο σημαντικά (least significant) bits. Λόγω αυτού του περιορισμού, όταν η φυσική μνήμη έχει πλάτος *N* bytes, αρκεί μία μόνο προσπέλαση σε αυτήν για κάθε πρόσβαση σε ποσότητα μεγέθους *N* bytes.

3.4.2 Αρίθμηση των Bytes - Μηχανή Big-Endian

Όταν αποθηκεύεται στη μνήμη ενός υπολογιστή μια ποσότητα αποτελούμενη από πολλαπλά bytes (π.χ. ένας ακέραιος), πρέπει να καθοριστεί με ποια σειρά αριθμούνται (διευθυνσιοδοτούνται) τα επιμέρους bytes μέσα στην ποσότητα αυτή. Ο DLX είναι ένας big-endian (σχήμα 3-1) επεξεργαστής, πράγμα που σημαίνει ότι αρίθμιση των bytes ξεκινά από το "big end", δηλαδή το MS byte. Το MS byte του κάθε ακεραίου έχει τη μικρότερη διεύθυνση, και οι διευθύνσεις των bytes του αυξάνουν (προχωρούν) καθώς προχωράμε "δεξιά", προς το LS byte του.

Little Endian

Διεύθυνση Λέξης				
0	3	2	1	0
4	7	6	5	4

Big Endian

Διεύθυνση Λέξης				
0	0	1	2	3
4	4	5	6	7

Σχήμα 3-1

3.4.3 Προσπελάσεις Μνήμης: Εντολές load και store

Ο DLX, όπως και οι άλλοι επεξεργαστές τύπου RISC, δεν έχει εντολές που να κάνουν αριθμητικές πράξεις πάνω σε τελεστέους που βρίσκονται στη μνήμη. Όλες οι αριθμητικές πράξεις του γίνονται πάνω σε καταχωρητές ή σταθερές ποσότητες (immediate constants). Για να γίνει επεξεργασία μιας λέξης (32 bits), ή μιας μισής λέξης (16 bits), ή ενός byte (8 bits) από τη μνήμη σ' ένα καταχωρητή της CPU, πρέπει να μεταφερθεί από τη μνήμη σε καταχωρητή, να γίνουν οι οποιεσδήποτε ενέργειες και τέλος να αντιγραφεί το αποτέλεσμα από τον καταχωρητή στη μνήμη. Αυτό γίνεται για τους εξής λόγους:

i. για απλότητα του υλικού
ii. Γιατί δεν θα μπορούσε να επιτευχθεί ψηλότερη ταχύτητα αν μία μόνη εντολή έκανε και την αντιγραφή και την επεξεργασία.

Αντιγραφή μιας 32-bit λέξης από τη μνήμη σ' ένα καταχωρητή ("*φόρτωμα* στον καταχωρητή") γίνεται με την εντολή "**`lw $rd, imm($rx)`**" (**load** word), όπου `$rd` είναι ο καταχωρητής προορισμού (destination register), και `$rx` είναι ένας καταχωρητής (index register) που περιέχει μια διεύθυνση μνήμης στην οποία προστίθεται ο σταθερός αριθμός `imm` και το αποτέλεσμα της πρόσθεσης είναι η τελική διεύθυνση μνήμης απ' όπου γίνεται η αντιγραφή στον `$rd`. Η αντιγραφή μιας 32-bit λέξης από ένα καταχωρητή στη μνήμη ("*αποθήκευση* του καταχωρητή") γίνεται με την εντολή "**`sw $rs, imm($rx)`**" (**store** word), η οποία γράφει στη θέση μνήμης με διεύθυνση *(imm + $rx)*, δηλαδή προκαλεί την αντιγραφή *M[imm + $rx]* ← *$rs*. Εδώ, ο `$rs` είναι καταχωρητής πηγής (source register).

3.5 Αρχιτεκτονική συνόλου εντολών του DLX

3.5.1 Εισαγωγή

Όλες οι εντολές του DLX(βλέπε παράρτημα Γ), όπως και των άλλων υπολογιστών της οικογένειας RISC, έχουν σταθερό μέγεθος 32 bit. Τα 32-bit αυτά χωρίζονται σε διάφορα πεδία (fields) με εύρος από 2 έως 6 πεδία, ανάλογα με τη μορφή(format) της εντολής. Υπάρχουν τρεις διαφορετικές μορφές εντολών που είναι οι εξής:

- R-type εντολών
- I-type εντολών
- J-type εντολών

Υπάρχουν επίσης κατηγορίες εντολών που ακολουθούν ένα από τους τρεις πιο πάνω τύπους και οι οποίες είναι οι εξής:

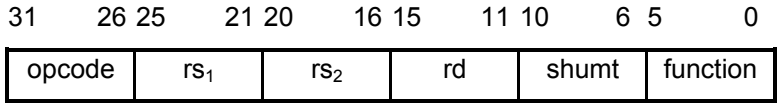
- Load/store – πράξεις μετακίνησης δεδομένων.
- Αριθμητικές εντολές, υπολογιστικές εντολές, λογικές εντολές και εντολές αλλαγής θέσης (shift operations). Τέτοιες εντολές είναι πρόσθεση, αφαίρεση, λογικό ¨ΚΑΙ¨, λογικό ¨Η¨, >=, ==, κ.τ.λ.
- Εντολές διακλάδωσης (jump, branch)
- Co processors – FP εντολές

3.5.2 R-τύπος εντολών καταχωρητή – καταχωρητή (register-register Alu operation)

Οι εντολές αυτού του τύπου, γνωστές ως R-type εντολές, αποτελούνται (σχήμα 3-2 από αριστερά προς τα δεξιά), από το πεδίο **op**, από τρία πεδία των 5 bits καθένα που επιλέγουν

έναν από τους 32 καταχωρητές καθένα, ένα πεδίο των 5 bits που ονομάζεται shumt, και ένα πεδίο μεγέθους 6 bits που αποτελεί επέκταση του **op** και λέγεται **funct** (function code). Κατά την εκτέλεση των εντολών αυτών το πεδίο function καθορίζει τη πράξη ή ενέργεια της εκάστοτε εντολής στο μονοπάτι δεδομένων (datapath). Δηλαδή αν η υπό εκτέλεση ενέργεια θα είναι πρόσθεση, αφαίρεση κ.τ.λ. Ακολούθως η επιλεγόμενη ενέργεια εφαρμόζεται μεταξύ των περιεχομένων των καταχωρητών rs_1 και rs_2 και το αποτέλεσμα εγγράφεται στα περιεχόμενα του καταχωρητή η διεύθυνση του οποίου ορίζεται από το πεδίο rd. Δηλαδή,

$$Rd \leftarrow rs_1 \ \text{func} \ rs_2.$$

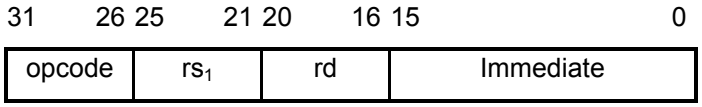


Σχήμα 3-2. Εντολές R-τύπου

3.5.3 I-τύπος εντολών

Οι εντολές αυτού του τύπου αποτελούνται (σχήμα 3-3 από αριστερά προς τα δεξιά), από το πεδίο **op**, από δύο πεδία των 5 bits καθένα που επιλέγουν έναν από τους 32 καταχωρητές καθένα, και ένα πεδίο μεγέθους 16 bits που λέγεται **imm** (immediate, ή offset, ή constant, ή address) και που περιέχει μία "άμεση" σταθερή ποσότητα. Κατά την εκτέλεση των εντολών το πεδίο opcode καθορίζει τη πράξη ή ενέργεια της εκάστοτε εντολής στο μονοπάτι δεδομένων (datapath). Η άμεση σταθερή ποσότητα υπόκειται σε επέκταση πρόσημου έτσι ώστε να γίνει 32-bit ποσότητα και η επιλεγόμενη ενέργεια εφαρμόζεται μεταξύ του περιεχόμενου του καταχωρητή rs_1 και της 32-bit ποσότητας. Το αποτέλεσμα της πράξης εκχωρείται στα περιεχόμενα του καταχωρητή η διεύθυνση του οποίου ορίζεται από το πεδίο rd. Δηλαδή

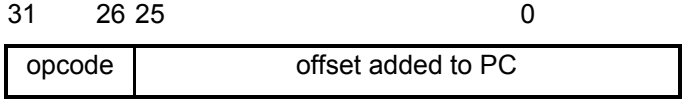
Rd ← rs_1 op signExt Immediate.



Σχήμα 3-3. Εντολές I-τύπου

3.5.4 J-τύπος εντολών

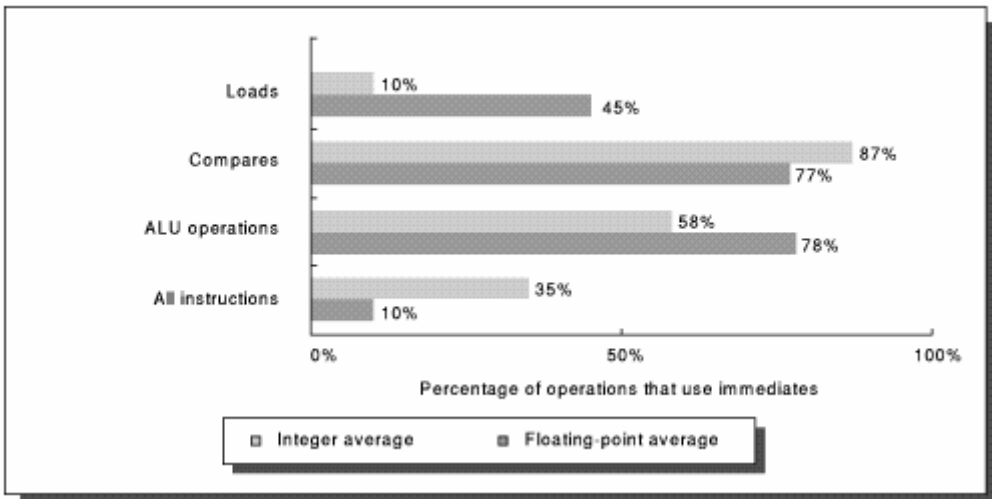
Οι εντολές αυτού του τύπου αποτελούνται (σχήμα 3-4 από αριστερά προς τα δεξιά), από το πεδίο **op**, και ένα πεδίο "**target**" μεγέθους 26 bits που προσδιορίζει μεγάλο μέρος μιας πλήρους διεύθυνσης μνήμης. Κατά την εκτέλεση των εντολών αυτών γίνεται πλήρωση με μηδενικά του πεδίου target έτσι ώστε να γίνει 32-bit ποσότητα, η οποία προστίθεται με τη διεύθυνση της επόμενης προς εκτέλεση λέξης. Το αποτέλεσμα της ενέργειας αυτής προστιθέμενο με τη διεύθυνση της επόμενης προς εκτέλεση εντολής είναι η διεύθυνση στην οποία μεταφέρεται ο έλεγχος του προγράμματος (label).



Σχήμα 3-4. Εντολές J-τύπου

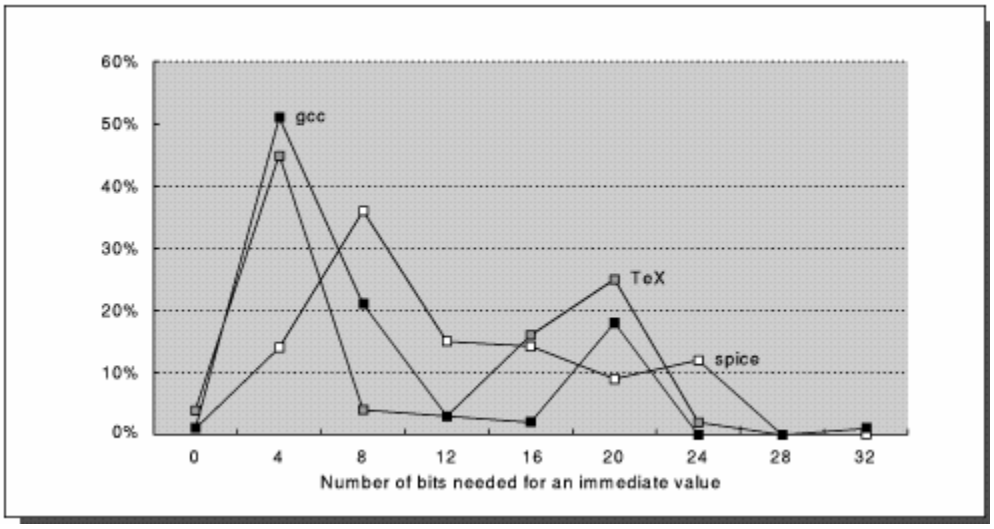
3.5.5 DLX: Immediate Εντολές

Ένας μεγάλος αριθμός ενεργειών σε προγράμματα χρησιμοποιεί άμεσες ποσότητες (immediate values)(σχήμα 3-5). Επίσης μεγάλος αριθμός από αυτές τις άμεσες ποσότητες έχουν μικρή τιμή, πράγμα που σημαίνει ότι απαιτούν λίγα bits για να αναπαρασταθούν(σχήμα 3-6). Ο DLX κάνει πράξη τη λογική που λέει ότι οτιδήποτε είναι πιο συχνά εμφανιζόμενο είναι και γρήγορο. Με βάση τις πιο πάνω διαπιστώσεις, η τιμή της άμεσης ποσότητας στον DLX είναι 16-bit, έτσι ώστε οι πράξεις με τις άμεσες ποσότητες να εκτελούνται σχετικά γρήγορα. Υπάρχουν όμως και εντολές που υποστηρίζουν την δημιουργία μεγάλων τιμών των άμεσων ποσοτήτων όπως η load upper immediate.



Σχήμα 3-5

Φαίνεται ότι για τις αριθμητικές/λογικές πράξεις το 50% -75% των ενεργειών χρησιμοποιεί άμεσες ποσότητες, ενώ για τις συγκρίσεις ακεραίων το 75% -85% των εμφανίσεων χρησιμοποιεί άμεσες ποσότητες.



Σχήμα 3-6

Κατανομή μεγλεθους των τιμών των άμεσων ποσοτήτων που χρησιμοποιούνται κατά την εκτέλεση κάποιου προγράμματος.

3.5.6 Συγκρίσεις, Διακλαδώσεις, κάλεσμα διαδικασιών και άλματα στον DLX

Οι εντολές μεταφοράς ελέγχου (CTI, control transfer instructions) καθορίζουν να εκτελεστεί σαν επόμενη εντολή (πάντοτε ή υπό ορισμένες συνθήκες μόνο) μια εντολή άλλη από την "επόμενη από κάτω" τους εντολή. Όταν η μεταφορά ελέγχου γίνεται υπό συνθήκη, οι εντολές συνήθως ονομάζονται διακλαδώσεις (branch). Όταν η μεταφορά γίνεται πάντοτε, οι εντολές συνήθως λέγονται άλματα (jump). Επίσης υπάρχουν καλέσματα διαδικασιών, λειτουργικού συστήματος, και επιστροφές από αυτά.

Στον DLX οι υπό συνθήκη διακλαδώσεις αφορούν συγκρίσεις ενός καταχωρητή με το μηδέν. Οι εντολές αυτές είναι:

- **`beq $rs, label`** διακλάδωση εάν: $\$r_s = 0$
- **`bne $rs, label`** διακλάδωση εάν: $\$r_s \neq 0$

Στη γλώσσα συμβόλων (Assembly), η διεύθυνση προορισμού της διακλάδωσης δηλώνεται απλά με μία ετικέτα (label), και αναλαμβάνει ο συμβολομεταφραστής (Assembler) να υπολογίσει και να βάλει τη σωστή δυαδική τιμή. Στη γλώσσα μηχανής, οι εντολές διακλάδωσης ακολουθούν το I-format, και η διεύθυνση προορισμού προκύπτει ως εξής:

$$\text{new PC} \leftarrow (\text{PC} + 4) + 4 \times \text{ImmOffset}$$

Όπου PC είναι η διεύθυνση της ίδιας της εντολής διακλάδωσης, ImmOffset είναι η σταθερή ποσότητα των 16 bits του I-format θεωρούμενη ως προσημασμένος αριθμός σε συμπλήρωμα ως προς 2 (δηλαδή sign-extended), και new PC είναι η διεύθυνση της εντολής προορισμού σε περίπτωση επιτυχίας της διακλάδωσης. Η αύξηση (PC + 4) γίνεται για λόγους ευκολίας του hardware (όλες οι εντολές αυξάνουν τον PC κατά 4). Ο πολλαπλασιασμός του ImmOffset επί 4 γίνεται διότι ο αριθμός ImmOffset μετράει πλήθος εντολών μπροστά (θετικός) ή πίσω (αρνητικός), αντί να μετρά πλήθος bytes μπροστά ή πίσω.

Οι υπόλοιπες μορφές συγκρίσεων, που δεν γίνονται μέσα στις εντολές διακλάδωσης, υλοποιούνται με ειδικές, ξεχωριστές, αριθμητικές εντολές σύγκρισης. Πρόκειται για εντολές ανάλογες προς την πρόσθεση ή την αφαίρεση, μόνο που το αποτέλεσμά τους είναι τύπου Boolean αντί τύπου ακέραιος. Τέτοια αποτελέσματα τύπου Boolean έχουν δύο μόνο δυνατές τιμές: 0 γιά ψευδές, και 1 γιά αληθές. Τα αποτελέσματα αυτά γράφονται στους 32 bit καταχωρητές, σαν οι ακέραιοι 0 ή 1, δηλαδή στο LS bit του καταχωρητή, με όλα τα υπόλοιπα bits του καταχωρητή μηδενικά.

Οι εντολές διακλάδωσης που μεταφέρουν τον έλεγχο σε άλλη εντολή πάντοτε (άλματα) χωρίζονται σε:

- Αλμα (jump) χωρίς συνθήκη: Επόμενη προς εκτέλεση εντολή είναι η εντολή στη διεύθυνση (PC+4) + target. Χρησιμοποιεί το J-format. Η τελική διεύθυνση προορισμού (32 bits) προκύπτει από τα 4 παλαιά MS bits του PC, τα 26 bits του πεδίου προορισμού της εντολής, και από 2 μηδενικά LS bits.
- Άλμα σε προορισμό που καθορίζεται από καταχωρητή (jump register): επόμενη προς εκτέλεση εντολή είναι η εντολή στη διεύθυνση που περιέχεται στον καταχωρητή rs. Δηλαδή ο καταχωρητής rs περιέχει τη διεύθυνση προορισμού, δηλαδή έναν δείκτη στην επόμενη προς εκτέλεση εντολή. Η εντολή επιτρέπει να μεταφερθεί ο έλεγχος σε αυθαίρετη θέση μνήμης, η οποία μπορεί και να ποικίλει κατά την εκτέλεση του προγράμματος (run-time variable) και πιθανόν να εξαρτάται και από τα δεδομένα (data

dependent). Χρησιμοποιείται μεταξύ άλλων και για επιστροφή από διαδικασία όπως αναφέρεται αμέσως παρακάνω.

- Αλμα και Σύνδεση (jump and link). Έχει το ίδιο τύπο με την εντολή άλματος (jump), και κάνει τα ίδια με εκείνην (ίδια διεύθυνση προορισμού), συν, επιπλέον, αποθηκεύει τη διεύθυνση της επόμενης της εντολής (παλαιό PC + 4) στον καταχωρητή 31 ($ra, ή $31). Χρησιμοποιείται για κάλεσμα διαδικασίας. Η διεύθυνση που αποθηκεύεται μπορεί στη συνέχεια να χρησιμοποιηθεί για επιστροφή από τη διαδικασία, μέσω της εντολής **jr $ra**. Η διεύθυνση επιστροφής αποθηκεύεται πάντα στον καταχωρητή ($31). Το "31" δεν φαίνεται πουθενά μέσα στην εντολή **jal**, απλώς το βάζει το hardware αυτόματα.

Το κάλεσμα διαδικασίας (procedure call) γίνεται μέσω της εντολής jal (jump and link - άλμα και σύνδεση), η δε επιστροφή από διαδικασία (procedure return) γίνεται μέσω της εντολής jr**.**

ΚΕΦΑΛΑΙΟ 4
ΠΕΡΙΓΡΑΦΗ ΥΛΟΠΟΙΗΣΗΣ ΥΠΟΣΥΝΟΛΟΥ ΕΝΤΟΛΩΝ ΤΟΥ DLX

4.1 Εισαγωγή

Βασιζόμενοι στην αρχιτεκτονική συνόλου εντολών του DLX, προχωρήσαμε στην υλοποίηση ενός υποσυνόλου του συνόλου εντολών του DLX(Βλέπε παράρτημα Γ), έτσι ώστε να εκτελούνται μόνο οι εξής εντολές:

- Απλές εντολές πρόσβασης μνήμης (lw, sw)
- Αριθμητικές και Λογικές Εντολές (add, sub, and, or, xor, addi, subi, ori, andi, xori)
- Εντολές ελέγχου εκτέλεσης προγράμματος (beq, bne, j, jr)

Η παρούσα σχεδίαση έχει κάποιες τροποποιήσεις σε σχέση με τη κλασσική σχεδίαση του μονοπατιού δεδομένων ενός ομόχειρου επεξεργαστή όπως αυτή περιγράφεται στα εγχειρίδια. Αυτό συμβαίνει διότι το εργαλείο σύνθεσης της systemC σε RTL VHDL, έχει κάποιους περιορισμούς που καθιστούσαν την σχεδίαση εφικτή μεν, αλλά αύξαναν τους κύκλους εκτέλεσης των εντολών. Εκτενέστερη αναφορά στην υλοποίηση γίνεται στις πιο κάτω ενότητες. Οι βασικές αλλαγές στο μονοπάτι δεδομένων συνοψίζονται στα εξής:

- Προσθήκη δύο επιπλέον μνημών ROM στο επίπεδο ανάκλησης εντολής που κρατούν τα πεδία Rs, Rt των εντολών.
- Χρησιμοποίηση μνήμης δεδομένων σε περιπτώσεις εκτέλεσης αριθμητικών/λογικών πράξεων ως καταχωρητή. Στη περίπτωση αυτή γίνεται εγγραφή στη μνήμη του αποτελέσματος στη διεύθυνση μηδέν.

Η υλοποίηση βασίστηκε στη γλώσσα sytstemC, λαμβάνοντας υπόψη τις δομές και τους περιορισμούς του SC. Για την μετατροπή του κώδικα υλοποίησης από τη γλώσσα περιγραφής systemC σε RTL VHDL, έτσι ώστε να επιτευχθεί περεταίρω σύνθεση χρησιμοποιώντας δεδομένα εργαλεία σύνθεσης, χρησιμοποιήθηκε ο systemCrafter. Για FPGAs της Xilinx, ένα τέτοιο εργαλείο σύνθεσης είναι το Xilinx XST και τα εργαλεία τοποθέτησης και δρομολόγησης (place and route tools). Ο systemCrafter δεν υποστηρίζει στην παρούσα έκδοσή

του μετατροπή ασύγχρονων κυκλωμάτων όπως πολυπλέκτες. Για το λόγο αυτό η οποιαδήποτε αναγκαία ασύγχρονη λογική απαιτούνταν για τη υλοποίηση του μονοπατιού δεδομένων, έγινε inlined στο μπλοκ της σύγχρονης λογικής η οποία το εμπεριέχει. Δηλαδή έστω ότι στο μπλοκ «αποκωδικοποίησης εντολών» έχω (σε ψευτοκώδικα C) ένα ασύγχρονο αθροιστή add το αποτέλεσμα του οποίου αποθηκεύεται σε ένα καταχωρητή με όνομα reg

```
int add ( a, b)

{

        Ασύγχρονος αθροιστής

}
```

Ο κώδικας που υλοποιεί αυτό τον αθροιστή, συμπεριλαμβάνεται στο σύγχρονο κύκλωμα που υλοποιεί τον καταχωρητή reg.. Παρακάτω παρατίθενται τα επιμέρους κομμάτια της υλοποίησης του ομόχειρου μονοπατιού δεδομένων του DLX έτσι ώστε να εκτελούνται οι πιο πάνω βασικές εντολές. Το πλήρες μονοπάτι δεδομένων φαίνεται στο σχήμα 4-13.

4.2 Σχεδιασμός και υλοποίηση βαθμίδας ανάκλησης εντολών (IF)

Η βαθμίδα ανάκλησης εντολών αποτελείται από τα εξής κυκλώματα:

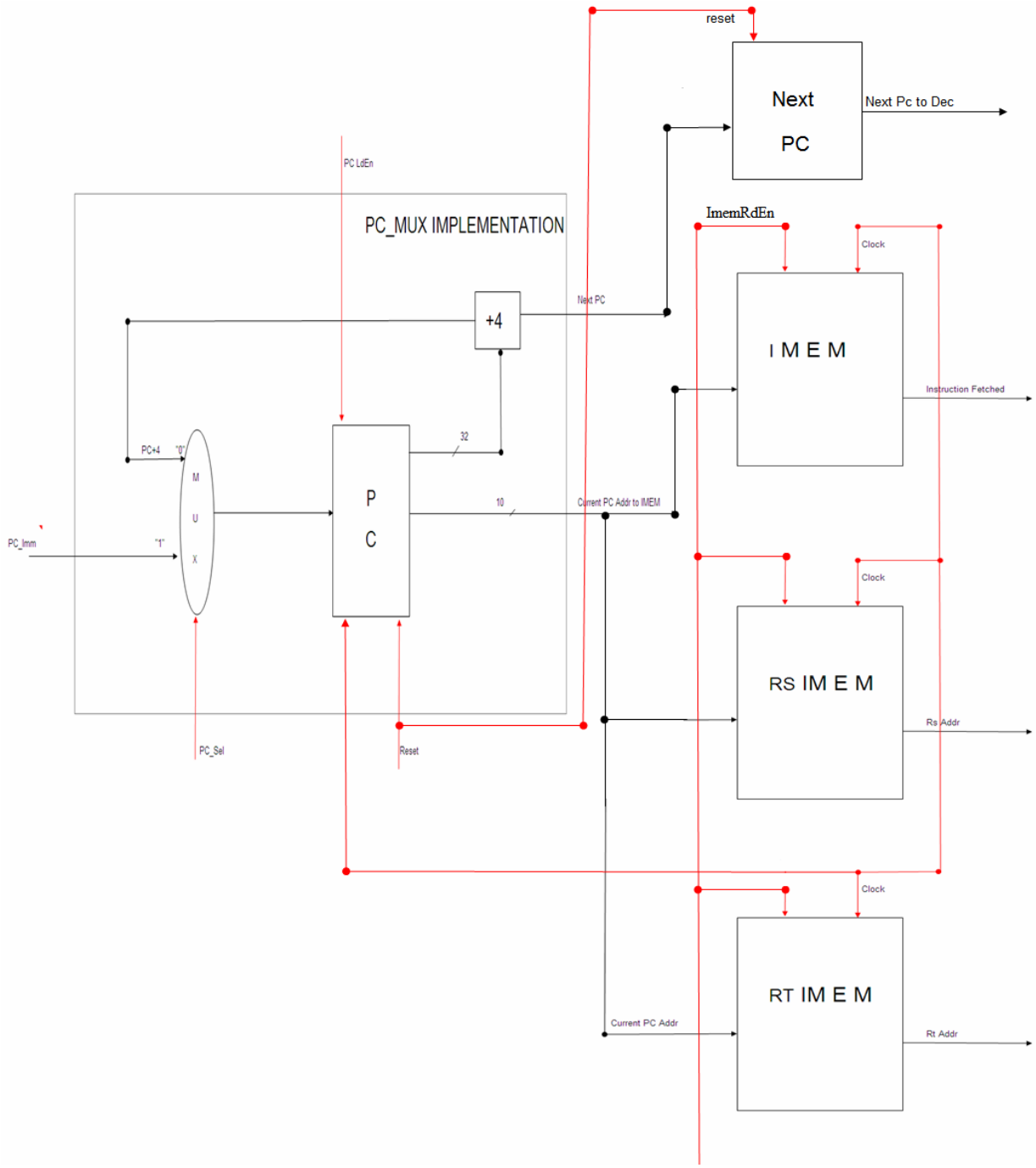
- τον μετρητή προγράμματος PC, ο οποίος αποθηκεύει την διεύθυνση της εντολής που πρόκειται να ανακληθεί.
- ένα πολυπλέκτη 2 σε 1 ο οποίος επιλέγει την επόμενη διεύθυνση που θα εισαχθεί στην μνήμη ανάκτησης εντολών. Οι είσοδοι του πολυπλέκτη είναι: η διεύθυνση του επόμενου μετρητή προγράμματος (PC+4), η διεύθυνση διακλάδωσης είτε όταν υπάρχει μια υπό συνθήκη εντολή διακλάδωσης, είτε όταν η επόμενη εντολή είναι μια εντολή άλματος, είτε όταν η επόμενη προς εκτέλεση εντολή είναι η jr, ένα αθροιστή

που αυξάνει την τιμή του μετρητή προγράμματος κατά 4 (δηλαδή να πηγαίνει στην επόμενη προς εκτέλεση λέξη),

- Η μνήμη ανάκλησης πεδίου Rs, είναι μια ROM μονής πόρτας (single port) 1024 θέσεων. Η μνήμη αυτή δημιουργείται από το core generator του εργαλείου ISE 7.1 της Xilinx.
- Η μνήμη ανάκλησης πεδίου Rt, είναι μια ROM μονής πόρτας (single port) 1024 θέσεων. Η μνήμη αυτή δημιουργείται από το core generator του εργαλείου ISE 7.1 της Xilinx(βλέπε παράρτημα Β-1).
- τον καταχωρητή Next_PC που αποθηκεύει την τιμή PC+4.
- Η μνήμη ανάκλησης εντολών είναι μια ROM μονής πόρτας (single port) 1024 θέσεων. Η μνήμη αυτή δημιουργείται από το core generator του εργαλείου ISE 7.1 της Xilinx(βλέπε παράρτημα Β-1).

Επειδή ο πολυπλέκτης και ο αθροιστής είναι ασύγχρονα κυκλώματα, η υλοποίησή τους γίνεται στο μπλοκ με ονομασία PC_MUX, που υλοποιεί τον μετρητή προγράμματος, όπως περιγράφηκε και στο τμήμα 4.1. Το πλήρες διάγραμμα της βαθμίδας φαίνεται στο σχήμα 4-1.

Οι μνήμες που έχουν τα πεδία Rs, Rt των εντολών υλοποιήθηκαν διότι ο SC δεν παρέχει τη δυνατότητα επιλογής εύρους bit από ένα σήμα (bit-array)εξόδου. Έτσι δεν υπήρχε δυνατότητα αποκωδικοποίησης της εντολής στην έξοδο της μνήμης ανάκλησης εντολών, αφού ούτε η δημιουργία ασύγχρονων κυκλωμάτων είναι δυνατή. Τυχόν εισαγωγή καταχωρητή στην έξοδο της μνήμης θα προσέδιδε ακόμα ένα επίπεδο στο ομόχειρο μονοπάτι δεδομένων, γεγονός που θα αύξανε το CPI του μικροεπεξεργαστή. Αυτό όμως είναι έξω από τις προδιαγραφές σχεδίασης. Η χρησιμοποίηση των δύο επιπλέον μνημών είναι δυνατή καθώς η Spartan FPGA, έχει αρκετά μπλοκ από RAM. Με τον τρόπο αυτό οι μνήμες ουσιαστικά χρησιμοποιούνται σαν δύο ομόχειροι καταχωρητές. Η βαθμίδα ανάκτησης εντολής φαίνεται στο σχήμα 4-1.



Σχήμα 4-1

Βαθμίδα Ανάκλησης εντολής

4.3 Σχεδιασμός και υλοποίηση βαθμίδας αποκωδικοποίησης εντολών (DECODE)

Η βαθμίδα αποκωδικοποίησης εντολών αποτελείται από τα εξής κυκλώματα:

- Το αρχείο καταχωρητών, η υλοποίηση του οποίου παρουσιάζεται στο τμήμα 4.3.1.
- Ένα καταχωρητή που δέχεται σαν είσοδο την υπό εκτέλεση εντολή και αφού επιλέξει ένα από τα 2 πεδία μιας εντολής που καθορίζουν ποιος καταχωρητής θα εγγραφεί (rt ή rd), ανάλογα με τον τύπο της εγγραφής, αποθηκεύει αυτή τη διεύθυνση.
- Δύο καταχωρητές που αποθηκεύουν τις διευθύνσεις των καταχωρητών πηγής (rs_1, rs_2). Οι διευθύνσεις αυτές χρησιμοποιούνται σαν είσοδοι στην μονάδα ελέγχου Fcontrol, σε περιπτώσεις όπου απαιτείται προώθηση δεδομένων.
- Τον καταχωρητή που αποθηκεύει την τιμή του πεδίου Immediate στο οποίο έχει γίνει επέκταση πρόσημου.
- Ένα καταχωρητή που αποθηκεύει την υπό εκτέλεση εντολή για να προωθηθεί στο επόμενο στάδιο.
- Το καταχωρητή που κρατά τη διεύθυνση εγγραφής rd στο αρχείο καταχωρητών.
- Ένα καταχωρητή που κρατά τη τιμή του PC+4 από το προηγούμενο ομόχειρο στάδιο.

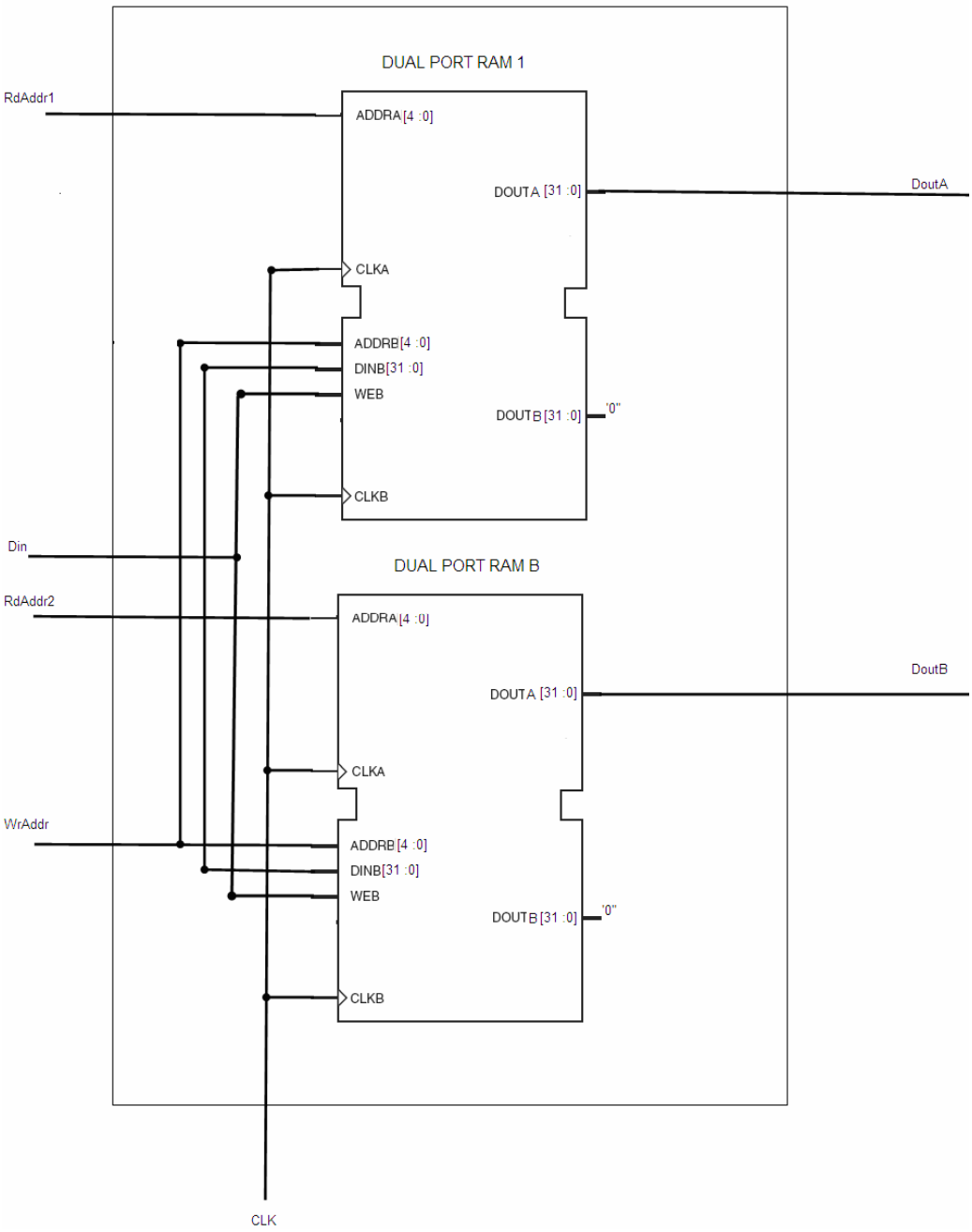
4.3.1 Υλοποίηση αρχείου καταχωρητών

Το αρχείο καταχωρητών αποτελείται από 32 καταχωρητές, δύο θύρες ανάγνωσης και μία θύρα εγγραφής. Για την υλοποίηση του χρησιμοποιήθηκαν δύο δίπορτες (dual port) μνήμες 32 θέσεων των 32 bit. Οι μνήμες αυτές παράχθηκαν από το core jenerator του ISE 7.1.Τα βασικά χαρακτηριστικά των δίπορτων μνημών που χρησιμοποιούνται[11] είναι τα εξής: Η κάθε μια δίπορτη μνήμη έχει δύο ανεξάρτητες θύρες που έχουν πρόσβαση σε κοινόχρηστο χώρο μνήμης. Έχουν την ίδια λειτουργικότητα και κάθε μια από τις θύρες έχει πρόσβαση εγγραφής και ανάγνωσης στη μνήμη. Ταυτόχρονες αναγνώσεις από την ίδια θέση μνήμης είναι εφικτές, αλλά ταυτόχρονη ανάγνωση-από και εγγραφή-σε ίδια θέση μνήμης, έχει ως αποτέλεσμα την ορθή εγγραφή δεδομένων στη μνήμη, αλλά την ανάγνωση μη-έγκυρων δεδομένων από τη μνήμη. Όλες οι ενέργειες στη μνήμη λαμβάνουν χώρα κατά τη θετική ακμή του ρολογιού.

Κατά την εγγραφή των δεδομένων (όταν οι σημαίες WEA, WEB είναι ενεργοποιημένα) τα δεδομένα προς εγγραφή αποθηκεύονται στη θέση μνήμης που επιλέγεται από την εκάστοτε θύρα εισόδου.

Κατά την ανάγνωση δεδομένων από τη μνήμη, διαβάζονται τα περιεχόμενα της μνήμης που ορίζονται από τις διευθύνσεις της εκάστοτε θύρας εισόδου. Η ανάγνωση γίνεται όταν η σημαία εγγραφής δεδομένων είναι απενεργοποιημένη.

Το αρχείο καταχωρητών όπως προαναφέρθηκε, είναι μια τρίπορτη μνήμη, με δύο θύρες ανάγνωσης και μια θύρα εγγραφής. Για να επιτευχθεί αυτό, χρησιμοποιούνται δύο αντίγραφα μιας δίπορτης μνήμης που παράγεται από το core generator του εργαλείου ISE 7.1 της Xilinx. Η διασύνδεση των επιμέρους μνημών φαίνεται στο σχήμα 4-2. Η πρώτη μνήμη της κάθε μίας από τις δίπορτες μνήμες χρησιμοποιείται για ανάγνωση μόνο(οι σημαίες εγγραφής δεδομένων και τα δεδομένα εισόδου στις μνήμες τίθενται στο μηδέν). Έτσι δημιουργούνται οι δύο θύρες ανάγνωσης του αρχείου καταχωρητών. Για την επίτευξη μιας διεύθυνσης εγγραφής, βραχυκυκλώνονται οι διευθύνσεις εισόδου της κάθε μιας από τις δεύτερες μνήμες της κάθε δίπορτης μνήμης στη διεύθυνση εγγραφής δεδομένων. Επίσης βραχυκυκλώνονται τα δεδομένα εισόδου και τα ρολόγια των τεσσάρων μνημών.



Σχήμα 4-2.

Υλοποίηση αρχείου καταχωρητών χρησιμοποιώντας δύο 32-bit δίπορτες μνήμες από το core generator του ISE 7.1.

4.4 Σχεδιασμός και υλοποίηση βαθμίδας Εκτέλεσης Εντολών (ALU)

Η βαθμίδα εκτέλεσης εντολών αποτελείται από τα εξής κυκλώματα:

- Ένα κύκλωμα που κρατά το αποτέλεσμα της μονάδας αριθμητικών και λογικών πράξεων(alu). Βγάζει επίσης στην έξοδο προς το στάδιο πρόσβασης μνήμης τη 10-bit διεύθυνση προς την μνήμη δεδομένων, καθώς και τα 32-bit δεδομένα που πρόκειται να εγγραφούν στην μνήμη. Το σχηματικό διάγραμμα της μονάδας φαίνεται στο σχήμα 4-3. Στην είσοδο A της alu υπάρχει ένας πολυπλέκτης 5 σε 1 με τα εξής σήματα εισόδου:
    1. Εισαγωγή μηδενικών στην alu, στη περίπτωση που απαιτείται να γίνει flush το επίπεδο εκτέλεσης (alu stage) του μονοπατιού δεδομένων. Το επίπεδο αυτό γίνεται flush σε περιπτώσεις επιτυχημένων διακλαδώσεων όπως αναφέρεται παρακάτω.
    2. εισαγωγή των δεδομένων της εξόδου A του αρχείου καταχωρητών.
    3. το αποτέλεσμα της alu από το επίπεδο πρόσβασης μνήμης (memory access stage), για περιπτώσεις που απαιτείται προώθηση δεδομένων.
    4. το αποτέλεσμα της alu στην έξοδο του επιπέδου εκτέλεσης (alu stage), για περιπτώσεις που απαιτείται προώθηση δεδομένων.
    5. Η διεύθυνση PC+4

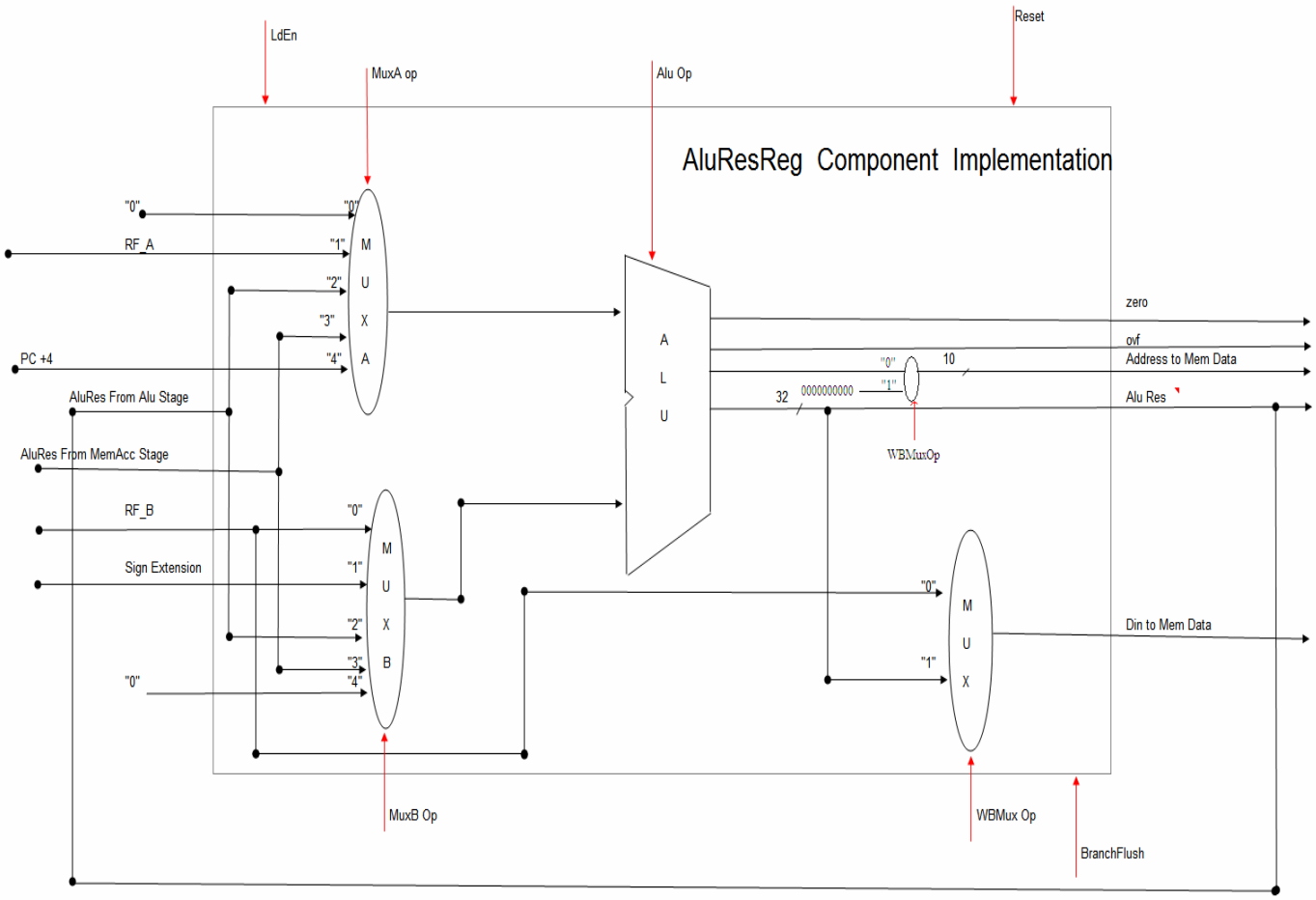
    Στην είσοδο B της alu υπάρχει ένας πολυπλέκτης 5 σε 1 με τα εξής σήματα εισόδου:
    1. εισαγωγή των δεδομένων της εξόδου B του αρχείου καταχωρητών.
    2. εισαγωγή του πεδίου Immediate στο οποίο ανάλογα με το τύπο της εντολής γίνονται οι ακόλουθες ενέργειες: i. επέκταση πρόσημου για περιπτώσεις που η υπό εκτέλεση εντολή θα είναι I-τύπου, ii) επέκταση προσήμου του displacement και αριστερή ολίσθηση κατά 2 έτσι ώστε να παραχθεί το κατάλληλο Label για εντολές άλματος, iii) επέκταση προσήμου της σταθερής ποσότητσας (Immediate 16-bits) και αριστερή ολίσθηση για να παραχθεί το κατάλληλο Label για εντολές υπό συνθήκης διακλάδωσης.

3. το αποτέλεσμα της alu από το επίπεδο πρόσβασης μνήμης (memory access stage), για περιπτώσεις που απαιτείται προώθηση δεδομένων.
4. το αποτέλεσμα της alu στην έξοδο του επιπέδου εκτέλεσης (alu stage), για περιπτώσεις που απαιτείται προώθηση δεδομένων.
5. Εισαγωγή μηδενικών στην alu, στη περίπτωση που απαιτείται να γίνει flush το επίπεδο εκτέλεσης (alu stage) του μονοπατιού δεδομένων. Το επίπεδο αυτό γίνεται flush σε περιπτώσεις επιτυχημένων διακλαδώσεων όπως αναφέρεται παρακάτω. Επίσης όταν πρέπει να αποβληθεί μια εντολή μετά από ανάσχεση ομοχειρίας.

Επίσης υπάρχει ένας πολυπλέκτης που δέχεται σαν εισόδους τα εξής σήματα: Τα περιεχόμενα από την έξοδο B του αρχείου καταχωρητών (σήμα ελέγχου '0') και το αποτέλεσμα της alu (σήμα ελέγχου '1'). Έτσι στην έξοδο επιλέγονται τα προς εγγραφή δεδομένα στη μνήμη δεδομένων. Αυτή η σχεδίαση γίνεται διότι η σχεδίαση απαιτεί εκτέλεση των εντολών σε 5 κύκλους. Εάν χρησιμοποιηθεί καταχωρητής στην έξοδο της μνήμης, αυξάνεται ο κύκλος εκτέλεσης της εντολής lw, σε 6 κύκλους. Με τη τεχνική αυτή αποφεύγεται το γεγονός αυτό. Όταν θα υπάρχει αριθμητική ή λογική εντολή που θα γράφει αποτέλεσμα στο αρχείο καταχωρητών, η μνήμη δεδομένων θα λειτουργεί σαν ομόχειρος καταχωρητής όπως εξηγείται παρακάτω στο τμήμα 4.5. Με την υλοποίηση αυτή όταν θα εκτελείται εντολή πρόσβασης μνήμης lw, να γίνεται η πρόσβαση στη σωστή διεύθυνση, ενώ όταν εκτελείται εντολή εγγραφής στη μνήμη, να επιλέγονται τα σωστά δεδομένα (έξοδος από RF_B) για να εγγραφούν στη σωστή διεύθυνση. Το σχηματικό διάγραμμα της μονάδας αυτής φαίνεται στο σχήμα 4-4.

- Το καταχωρητή που κρατά τη διεύθυνση εγγραφής rd στο αρχείο καταχωρητών.
- Μια σύγχρονη μονάδα με όνομα Branch_Decision που δέχεται σαν είσοδο την υπό εκτέλεση εντολή και τα περιεχόμενα της εξόδου A του αρχείου καταχωρητών και ανάλογα με το ποια εντολή είναι, επιλέγεται το σήμα ελέγχου του πολυπλέκτη στην είσοδο του μετρητή προγράμματος για να επιλεγεί η επόμενη προς εκτέλεση εντολή.

- Δύο καταχωρητές ελέγχου, που αποθηκεύουν τα σήματα ελέγχου των επιπέδων πρόσβασης μνήμης(Memory Access) και εγγραφής δεδομένων(Write Back).



Σχήμα 4-4

Κύκλωμα που υλοποιεί την alu και την επιλογή της διεύθυνσης και των δεδομένων εγγραφής στη μνήμη δεδομένων.

4.5 Σχεδιασμός και υλοποίηση βαθμίδας Πρόσβασης Μνήμης (MEM)

Η βαθμίδα πρόσβασης μνήμης αποτελείται από τα εξής κυκλώματα:

- Τη μνήμη δεδομένων, που είναι μια μονόπορτη RAM μνήμη 1024 θέσεων. Η μνήμη αυτή δημιουργείται με αντίστοιχο τρόπο από το core generator του εργαλείου ISE 7.1 της Xilinx.
- Το καταχωρητή που κρατά τη διεύθυνση εγγραφής rd στο αρχείο καταχωρητών.
- Το καταχωρητή ελέγχου, που αποθηκεύει το σήμα ελέγχου του επιπέδου εγγραφής δεδομένων(Write Back).

Η διαφοροποίηση στη σχεδίαση αυτή σε σχέση με τη κλασσική σχεδίαση του μονοπατιού δεδομένων ενός ομόχειρου μικροεπεξεργαστή η χρησιμοποίηση της μνήμης και σαν καταχωρητή. Αυτό διότι απαιτείται οι εντολές να εκτελούνται σε 5 κύκλους. Οι περιορισμοί όμως του SC, που δεν επιτρέπουν τη σχεδίαση ακολουθιακών κυκλωμάτων, προσθέτουν ακόμα ένα κύκλο στην εκτέλεση της εντολής lw (λόγω πρόσβασης και στη μνήμη). Η χρησιμοποίηση της μνήμης ως καταχωρητή σε ορισμένες περιπτώσεις λύνει αυτό το πρόβλημα. Αυτό γίνεται ως εξής:

Όταν εκτελούνται αριθμητικές ή λογικές εντολές που γράφουν αποτέλεσμα στο αρχείο καταχωρητών, η μνήμη δεδομένων λειτουργεί σαν καταχωρητής. Στη περίπτωση αυτή σαν δεδομένα εισόδου στη μνήμη επιλέγεται το αποτέλεσμα της πράξης από την alu. Σα διεύθυνση εγγραφής επιλέγεται η μηδενική διεύθυνση έτσι ώστε να εγγραφεί το αποτέλεσμα στην διεύθυνση 0. Επειδή η μνήμη αυτή είναι RAW (βλέπε παράρτημα Β) όταν εγγράφεται το αποτέλεσμα, ταυτόχρονα εμφανίζεται και στην έξοδο της μνήμης. Με τον τρόπο αυτό πετυχαίνεται η χρησιμοποίηση της μνήμης ως καταχωρητή.

Όταν εκτελείται η εντολή sw, σαν δεδομένα εγγραφής στη μνήμη επιλέγονται τα δεδομένα από την έξοδο B του αρχείου καταχωρητών και σα διεύθυνση ανάγνωσης/εγγραφής, η διεύθυνση που υπολογίζεται από την alu. Έτσι γίνεται η εγγραφή στη σωστή θέση μνήμης.

Όταν εκτελείται η εντολή lw, απενεργοποιείται η σημαία εγγραφής στη μνήμη (άρα γίνεται ανάγνωση), και σα διεύθυνση ανάγνωσης, η διεύθυνση που υπολογίζεται από την alu. Έτσι διαβάζεται η σωστή λέξη για να αποθηκευτεί στο αρχείο καταχωρητών.

4.6 Υλοποίηση μονάδας ελέγχου

Η μονάδα ελέγχου στη σχεδίαση αυτή, είναι υλοποιημένη σε πολλαπλά στάδια. Υπάρχει ένα σύγχρονο κύκλωμα contol, το οποίο βρίσκεται στο επίπεδο αποκωδικοποίησης εντολής (decode stage). Το κύκλωμα αυτό έχει σαν είσοδο την υπό εκτέλεση εντολή και παράγει τις σημαίες ενεργοποίησης (Load Enable) των καταχωρητών στα επίπεδα ανάκλησης εντολής, απολωδικοποίησης, εκτέλεσης εντολής και πρόσβασης μνήμης. Επίσης παράγει το σήμα εγγραφής στο αρχείο καταχωρητών, το σήμα εγγραφής στη μνήμη δεδομένων και το σήμα ελέγχου του πολυπλέκτη στην έξοδο της μνήμης δεδομένων.

Υπάρχει επίσης το σύγχρονο κύκλωμα που βρίσκεται στο επίπεδο εκτέλεσης εντολής, το Branch_Decision, το οποίο επιλέγει το σήμα ελέγχου του πολυπλέκτη στην είσοδο του μετρητή προγράμματος για την επιλογή της επόμενης προς εκτέλεση εντολή.

Επίσης η μονάδα Fcontrol η οποία υπολογίζει τα κατάλληλα σήματα ελέγχου των πολυπλεκτών στην είσοδο της alu, λαμβάνοντας υπόψη το τύπο της υπό εκτέλεσης εντολής και τους τυχόν περιορισμούς δεδομένων που μπορεί να υπάρχουν.

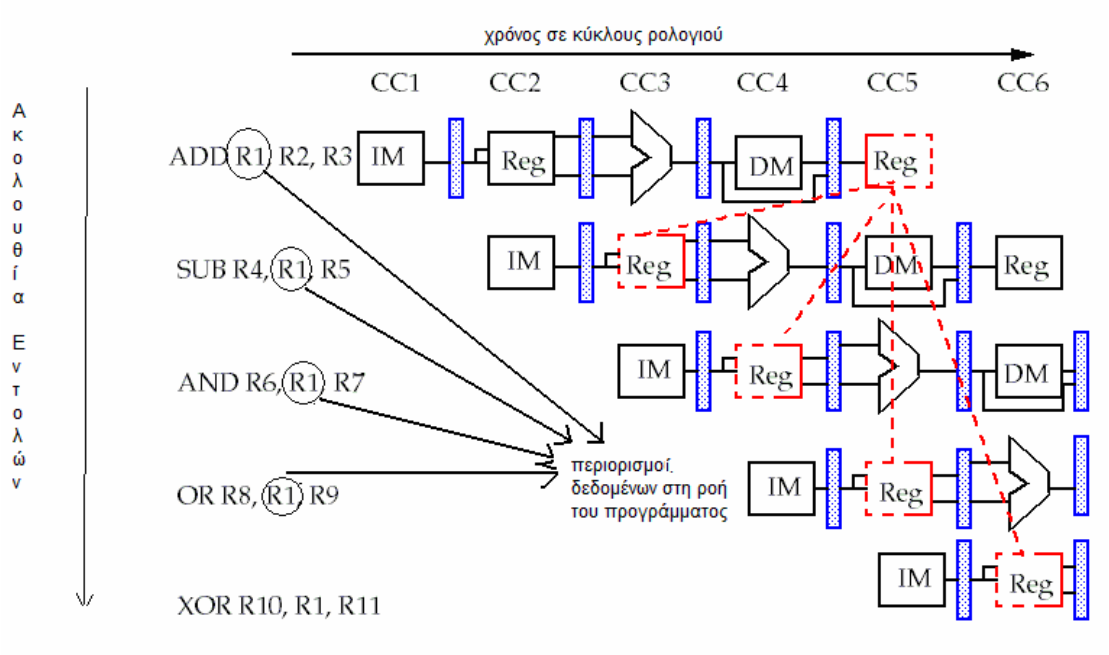
Τέλος το κύκλωμα ανίχνευσης περιορισμών δεδομένων (hazard detection unit) που δεν επιλύνονται με τη τεχνική της προώθησης δεδομένων, παράγει το σήμα ελέγχου LdEn του μετρητή προγράμματος, καθώς και τη σημαία ανάγνωσης των μνημών ανάκτησης εντολής.

4.7 Εξαρτήσεις Δεδομένων και Προώθηση Δεδομένων

Κατά την εκτέλεση μιας ροής εντολών σε ένα πρόγραμμα, σε αρκετές περιπτώσεις οι υπό εκτέλεση εντολές δεν είναι ανεξάρτητες, αλλά διέπονται από κάποιους περιορισμούς (σχήμα 4-5). Υπάρχουν τρεις τύποι περιορισμών:

- Δομικές εξαρτήσεις όταν υπάρχει πολλαπλή απαίτηση για κοινόχρηστο πόρο.
- Εξαρτήσεις ελέγχου όταν η εκτέλεση μιας εντολής εξαρτάται από την έκβαση προηγούμενης εντολής διακλάδωσης.

- Εξαρτήσεις δεδομένων όταν η ανάγνωση του καταχωρητή πρέπει να ακολουθεί την εγγραφή του ίδιου καταχωρητή από προηγούμενη εντολή (RAW), όταν η εγγραφή στον καταχωρητή πρέπει να έπεται της ανάγνωσης του από τις προηγούμενες εντολές (WAR) ή όταν η εγγραφή στον καταχωρητή πρέπει να έπεται όλων των εγγραφών στον ίδιο καταχωρητή από προηγούμενες εντολές.

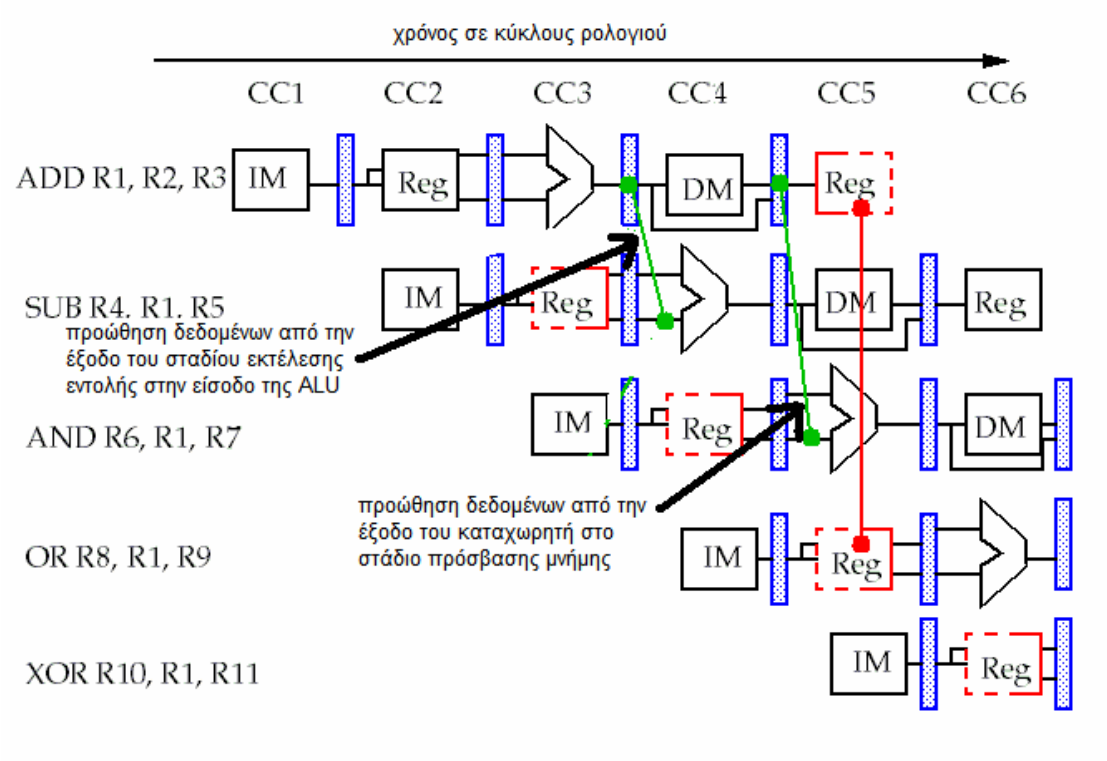


Σχήμα 4-5.

Παράδειγμα ακολουθίας εντολών όπου υπάρχουν περιορισμοί δεδομένων. Αυτό συμβαίνει διότι ο καταχωρητής προορισμού της εντολής ADD, έχει την ίδια διεύθυνση με τους καταχωρητές πηγής των επόμενων εντολών στη ροή της ομοχειρίας. Φαίνεται ότι οι εντολές SUB, AND και OR, διαβάζουν το καταχωρητή πριν αυτός ενημερωθεί από την εκτέλεση της εντολής ADD.

Όταν υπάρχουν περιορισμοί δεδομένων στην ομοχειρία, μια λύση είναι το πάγωμα (stall) της ομοχειρίας μέχρι την ολοκλήρωση της εντολής που γράφει τον καταχωρητή που θα χρησιμοποιήσει η επόμενη εντολή. Αυτή η λύση όμως είναι αρκετά αργή. Μια άλλη λύση είναι η προώθηση (forward) των δεδομένων από επόμενα στάδια στο στάδιο εκτέλεσης εντολής, πριν ακόμα η υπό εκτέλεση εντολή γράψει τα δεδομένα στη μνήμη καταχωρητών.. Η

τεχνική αυτή παρουσιάζεται σχηματικά στο σχήμα 4-6. Η προώθηση δεδομένων εφαρμόζεται μόνο σε περιπτώσεις όπου εκτελούνται εντολές καταχωρητή-καταχωρητή και έχει ως αποτέλεσμα να αποφεύγεται το πάγωμα της ομοχειρίας. Αυτή η τεχνική ονομάζεται και προσπέρασμα/παράκαμψη. Με αυτό τον τρόπο δεν υπάρχει ανάγκη παγώματος του μονοπατιού δεδομένω.



Σχήμα 4-6

Αντιμετώπιση περιορισμών δεδομένων, με την τεχνική προώθησης δεδομένων.

Κατά την υλοποίηση, η μονάδα που υλοποιεί την πιο πάνω τεχνική (Fcontrol), υλοποιείται στο επίπεδο του σταδίου αποκωδικοποίησης εντολής (σχήμα 4-7). Αυτό διότι οι πολυπλέκτες επιλογής εισόδων στην ALU βρίσκονται στο στάδιο εκτέλεσης εντολής. Επειδή η μονάδα εξαγωγής των σημάτων ελέγχου των πολυπλεκτών είναι σύγχρονη, αφού ασύγχρονα κυκλώματα δεν είναι δυνατό να υλοποιηθούν όπως αναλύθηκε σε προηγούμενη ενότητα,

πρέπει να υλοποιηθεί στο αμέσως προηγούμενο στάδιο από αυτό της εκτέλεσης εντολής. Η επιλογή των σημάτων ελέγχου των πολυπλεκτών γίνεται με βάση το τύπο της εκάστοτε υπό εκτέλεσης εντολής. Αυτό γίνεται ελέγχοντας το opcode της εντολής.

Κατά την εκτέλεση εντολών R-τύπου, περιορισμοί δεδομένων υπάρχουν όταν ισχύουν οι εξής συνθήκες:

$$\left.\begin{array}{l}\text{Dec/Ex RdRegister} = \text{IF/Dec RsRegister}\\ \text{Dec/Ex RdRegister} = \text{IF/Dec RtRegister}\end{array}\right\} \text{ Execution Hazards}$$

Κατά την ανίχνευση των πιο πάνω περιορισμών, γίνεται προώθηση των δεδομένων από την έξοδο του σταδίου εκτέλεσης εντολής στην είσοδο των πολυπλεκτών. Το σήμα ελέγχου σε αυτή τη περίπτωση σύμφωνα με σχήμα 4-4 είναι :

ForwardA / ForwardB = “011”;

$$\left.\begin{array}{l}\text{Ex/Mem RdRegister} = \text{IF/Dec RsRegister}\\ \text{Ex/Mem RdRegister} = \text{IF/Dec RtRegister}\end{array}\right\} \text{ Mem Hazards}$$

Κατά την ανίχνευση των πιο πάνω περιορισμών, γίνεται προώθηση των δεδομένων από την έξοδο του πρόσβασης μνήμης στην είσοδο των πολυπλεκτών. Το σήμα ελέγχου σε αυτή τη περίπτωση σύμφωνα με σχήμα 4-4 είναι :

ForwardA / ForwardB = “010”;

Μια άλλη περίπτωση ανίχνευσης ενδεχόμενων περιορισμών δεδομένων που προκύπτουν από το αποτέλεσμα εντολής που βρίσκεται στο στάδιο εγγραφής αποτελέσματος(WB στάδιο), το αποτέλεσμα εντολής που βρίσκεται στο στάδιο πρόσβασης μνήμης (MEM στάδιο) και το καταχωρητή πηγής της εντολής που βρίσκεται στο στάδιο εκτέλεσης εντολής. Ένα τέτοιο παράδειγμα της περίπτωσης αυτής είναι :

```
add $1  $1, $2
add $1  $1, $3
add $1  $1, $4
```

Σε αυτή τη περίπτωση το αποτέλεσμα προωθείται από το στάδιο πρόσβασης μνήμης που είναι το πιο πρόσφατο αποτέλεσμα. Ο έλεγχος για αυτή τη περίπτωση είναι ο εξής:

$$\left.\begin{array}{l}\left(\text{Ex/Mem RdRegister} = \text{IF/Dec RsRegister}\right) \text{ and} \\ \left(\text{Dec/Ex RdRegister} \neq \text{IF/Dec RsRegister}\right) \\ \\ \left(\text{Dec/Ex RdRegister} \neq \text{IF/Dec RtRegister}\right) \text{ and} \\ \left(\text{Ex/Mem RdRegister} = \text{IF/Dec RtRegister}\right)\end{array}\right\} \text{Mem Hazards}$$

Το σήμα ελέγχου σε αυτή τη περίπτωση σύμφωνα με σχήμα 4-4 είναι :

ForwardA / ForwardB = “010”;

Κατά την εκτέλεση εντολών I-τύπου, περιορισμοί δεδομένων υπάρχουν όταν ισχύουν οι εξής συνθήκες:

$$\text{Dec/Ex RdRegister} = \text{IF/Dec RsRegister}\} \text{ Execution Hazards}$$

Κατά την ανίχνευση των πιο πάνω περιορισμών, γίνεται προώθηση των δεδομένων από την έξοδο του σταδίου εκτέλεσης εντολής στην είσοδο των πολυπλεκτών. Το σήμα ελέγχου σε αυτή τη περίπτωση σύμφωνα με σχήμα 4-4 είναι :

ForwardA = “011”;

$$\text{Ex/Mem RdRegister} = \text{IF/Dec RsRegister}\} \text{ Mem Hazards}$$

Κατά την ανίχνευση των πιο πάνω περιορισμών, γίνεται προώθηση των δεδομένων από την έξοδο του πρόσβασης μνήμης στην είσοδο των πολυπλεκτών. Το σήμα ελέγχου σε αυτή τη περίπτωση σύμφωνα με σχήμα 4-4 είναι :

ForwardA = “010”;

Μια άλλη περίπτωση ανίχνευσης ενδεχόμενων περιορισμών δεδομένων που προκύπτουν από το αποτέλεσμα εντολής που βρίσκεται στο στάδιο εγγραφής αποτελέσματος(WB στάδιο), το αποτέλεσμα εντολής που βρίσκεται στο στάδιο πρόσβασης μνήμης (MEM στάδιο) και το καταχωρητή πηγής της εντολής που βρίσκεται στο στάδιο εκτέλεσης εντολής. Σε αυτή τη περίπτωση το αποτέλεσμα προωθείται από το στάδιο πρόσβασης μνήμης που είναι το πιο πρόσφατο αποτέλεσμα. Ο έλεγχος για αυτή τη περίπτωση είναι ο εξής:

$$\left.\begin{array}{l}\left(\text{Ex/Mem RdRegister} = \text{IF/Dec RsRegister}\right) \text{ and} \\ \left(\text{Dec/Ex RdRegister} \neq \text{IF/Dec RsRegister}\right)\end{array}\right\} \text{ Mem Hazards}$$

Το σήμα ελέγχου σε αυτή τη περίπτωση σύμφωνα με σχήμα 4-4 είναι :

ForwardA = “010”;

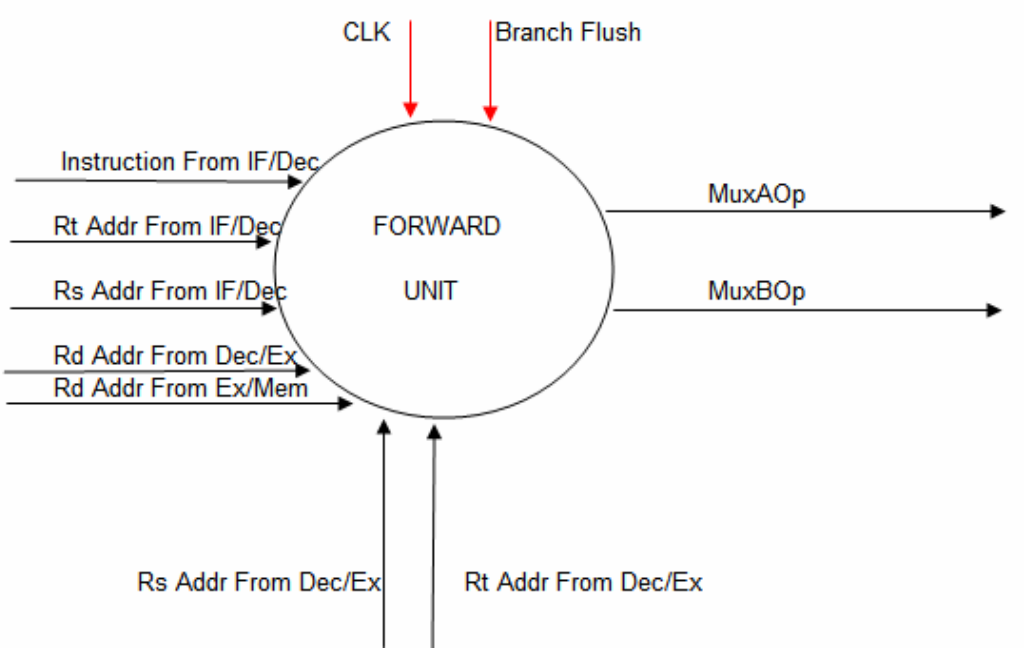
Το σήμα ελέγχου ForwardB σε όλες τις περιπτώσεις επιλέγει τη σταθερή ποσότητα immediate στην οποία έγινε επέκταση πρόσημου. Δηλαδή

ForwardB = “001”;

Σε περιπτώσεις όπου ανιχνευθεί επιτυχημένη διακλάδωση, ή εντολή σφάλματος, ενεργοποιείται η σημαία BranchFlush. Σε αυτή τη περίπτωση επιλέγονται να σταλούν στην μονάδα ALU μηδενικά για δύο κύκλους. Με αυτό τον τρόπο αποβάλλονται οι εντολές που ακολουθούν την εντολή επιτυχημένης διακλάδωσης ή την εντολή άλματος, που κατά την ανίχνευση άλματος ή διακλάδωσης βρίσκονται στο στάδιο ανάκλησης εντολής και στο μετρητή προγράμματος. Το σήμα ελέγχου σε αυτή τη περίπτωση σύμφωνα με σχήμα 4-4 είναι :

ForwardA = “000”;
ForwardB = “100”;

Σε όλες τις υπόλοιπες περιπτώσεις επιλέγονται τα κατάλληλα σήματα ελέγχου των πολυπλεκτών σύμφωνα με το τύπο της υπό εκτέλεσης εντολής που υποδηλώνεται από το opcode της εκάστοτε εντολής.



Σχήμα 4-7

Σχηματικό διάγραμμα Μονάδας ελέγχου για ανίχνευση περιορισμών δεδομένων και προώθηση.

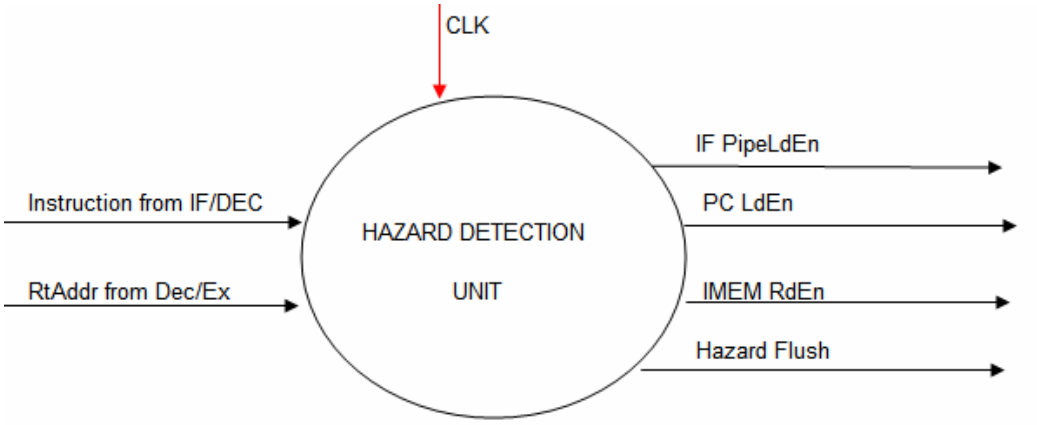
4.8 Εξαρτήσεις Δεδομένων και ανάσχεση ομοχειρίας

Όταν υπάρχουν περιορισμοί δεδομένων που οφείλονται σε εντολή πρόσβασης μνήμης (lw), όταν δηλαδή μια εντολή πρόσβασης μνήμης ακολουθείται από αριθμητική εντολή που διαβάζει το καταχωρητή προορισμού της lw, η τεχνική της προώθησης δεδομένων δεν εξαλείφει το πρόβλημα. Αυτό διότι η εντολή πρόσβασης μνήμης παράγει αποτέλεσμα μετά την πρόσβαση στη μνήμη, έτσι η μόνο η εφαρμογή της τεχνικής προώθησης δεδομένων δε λύνει το πρόβλημα. Χρειάζεται η προσθήκη λογικής ανίχνευσης κινδύνων (pipeline interlock logic). Η τεχνική αυτή εισάγει ανασχέσεις ομοχειρίας μέχρις ότου εξαλειφθεί ο κίνδυνος. Σε υλοποιήσεις όπου υποστηρίζεται η τεχνική προώθησης δεδομένων, η ανάσχεση αυτή διαρκεί 1

κύκλο ρολογιού. Με τον τρόπο αυτό εξαλείφεται ο περιορισμός, ενώ στη συνέχεια εάν χρειαστεί η τεχνική προώθησης δεδομένων εξαλείφει επιπλέον περιορισμούς δεδομένων που τυχόν να υπάρχουν. Σε αντίθετη περίπτωση οι ανασχέσεις διαρκούν πέραν του ενός κύκλου. Το αποτέλεσμα της εφαρμογής της λογικής αυτής είναι η αύξηση του κύκλου εκτέλεσης της εντολής που ακολουθεί τη lw, αυξάνοντας έτσι και το CPI.

Κατά την υλοποίηση, η μονάδα που υλοποιεί την πιο πάνω τεχνική (Hazard Detection Unit), υλοποιείται στο επίπεδο του σταδίου ανάκλησης εντολής. Αφού λάβει σαν εισόδους το καταχωρητή προορισμού της εντολής πρόσβασης μνήμης που βρίσκεται στο στάδιο αποκωδικοποίησης εντολής, ελέγχει εάν ο καταχωρητής προορισμού της είναι ίδιος με κάποιο από τους καταχωρητές πηγής της εντολής που ακολουθεί στην ομοχειρία. Εάν ισχύει αυτό, τότε εισάγεται στο μονοπάτι δεδομένων ένα interlock. Αλλιώς ακολουθείται κανονικά η ροή εκτέλεσης εντολών. Το Interlock εισάγεται απενεργοποιώντας τις σημαίες enable της μνήμης ανάκλησης εντολής καθώς και του PC. Με τον τρόπο αυτό η εντολή που ακολουθεί τη lw, θα καθυστερήσει για 1 κύκλο. Ουσιαστικά ανακτάται δύο φορές. Η συνθήκη που ελέγχεται είναι η εξής:

$$\left.\begin{array}{l}\text{if } (\text{Dec/Ex RtRegister} = \text{IF/Dec\_RsRegister}) \text{ or} \\ (\text{Dec/Ex RtARegister} = \text{IF/Dec RtRegister})\end{array}\right\} \textit{πάγωμα} \text{ ομοχειρίας}$$



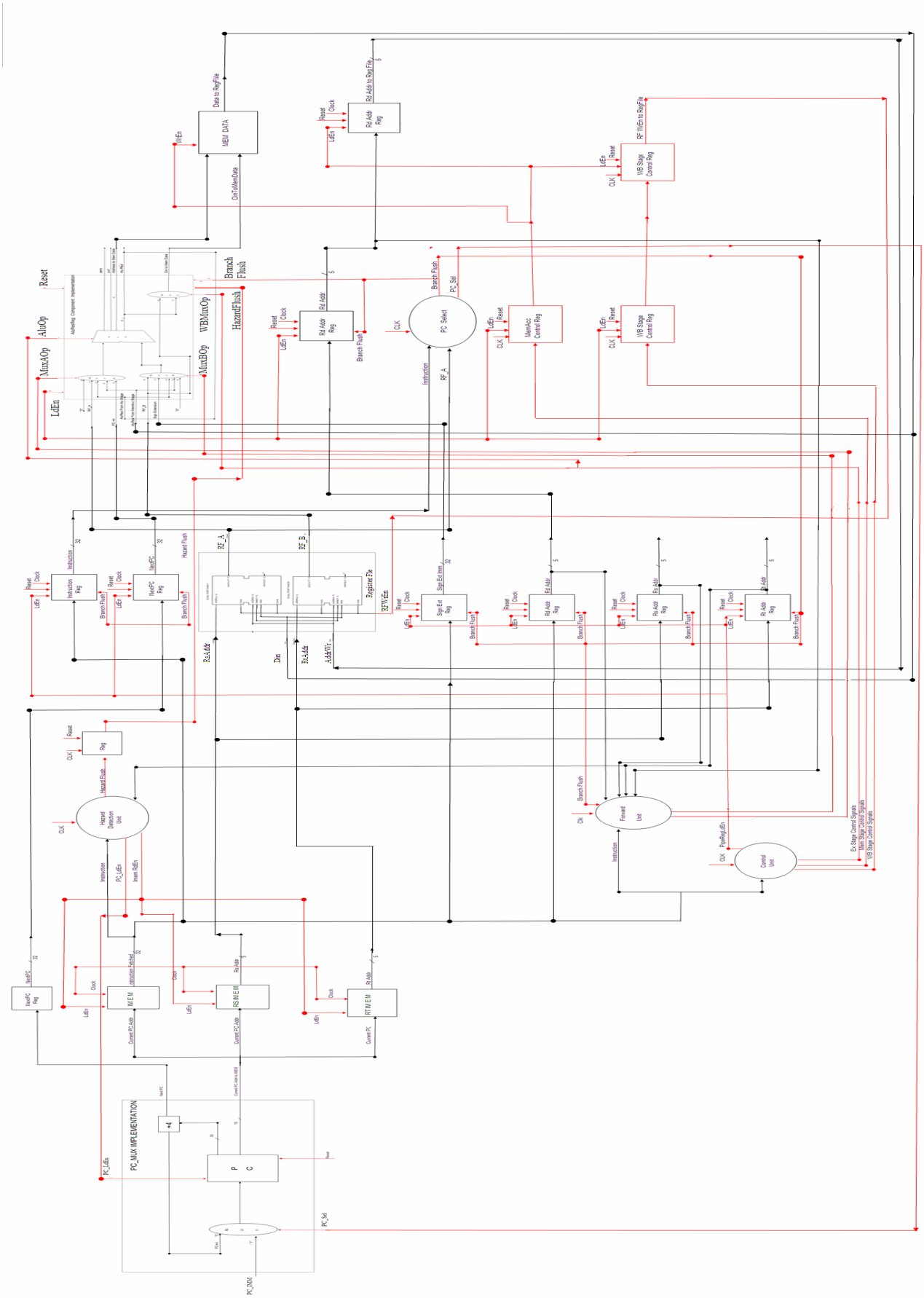
Σχήμα 4.8.

Μονάδα Ανίχνευσης σφαλμάτων που προκαλούν ανασχέσεις ομοχειρίας.

4.9 Εξαρτήσεις κατά τις διακλαδώσεις

Σε περιπτώσεις όπου εκτελούνται εντολές υπό συνθήκης διακλάδωσης, αρχικά θεωρείται ότι η διακλάδωση δεν θα είναι επιτυχής. Ως εκ τούτου, συνεχίζεται κανονικά η ανάκτηση εντολών από τη μνήμη. Εάν στο στάδιο εκτέλεσης εντολής διαπιστωθεί ότι η διακλάδωση είναι επιτυχής, τότε αποβάλλονται όλες οι εντολές που ανακτήθηκαν μέχρι το σημείο αυτό. Η αποβολή των εντολών γίνεται με την εισαγωγή μηδενικών στις εισόδους των σταδίων ανάκλησης εντολής, αποκωδικοποίησης εντολής και εκτέλεσης εντολής. Εισάγεται δηλαδή ένα επιπλέον σήμα ελέγχου για κάθε στάδιο, που ονομάζεται BranchFlush. Όταν ενεργοποιείται αυτό το σήμα, οι έξοδοι των καταχωρητών ομοχειρίας σε κάθε στάδιο στέλνουν μηδενικά στο επόμενο στάδιο. Στη παρούσα υλοποίηση όμως επειδή οι έξοδοι των μνημών (έξοδος μνήμης ανάκτησης εντολής και έξοδος αρχείου καταχωρητών) δε μπορούν να γίνουν flushed, αυτό γίνεται στο στάδιο εκτέλεσης εντολής ως εξής: Όταν ανιχνευθεί επιτυχής εντολή διακλάδωσης, γίνεται flush το στάδιο εκτέλεσης εντολής (γίνεται flush ο καταχωρητής στην έξοδο της alu) ενώ στη μονάδα Fcontrol παράγονται τα σήματα ελέγχου που θα εισαγάγουν μηδενικά στην είσοδο της alu για τους επόμενους 2 κύκλους αφού δε γίνονται flush τα στάδια ανάκτησης εντολής και αποκωδικοποίησης. Έτσι όταν οι δύο εντολές που δεν γίνονται flush φτάσουν στο στάδιο εκτέλεσης εντολής αποβάλλονται εισάγοντας μηδενικά στο στάδιο εκτέλεσης εντολής.

Με την εφαρμογή αυτής της τεχνικής αυτής όταν η υπό συνθήκη διακλάδωση είναι ανεπιτυχής, δεν αυξάνεται ο κύκλος εκτέλεσης της εντολής που ακολουθεί την εντολή διακλάδωσης. Όταν όμως πρόκειται για επιτυχή διακλάδωση, αυξάνεται κατά 3 κύκλους ο κύκλος εκτέλεσης της εντολής που ακολουθεί την εντολή διακλάδωσης, αυξάνοντας έτσι και το CPI.



Σχήμα 4-13
Το πλήρες μονοπάτι δεδομένων του μοκροεπεξεργαστή DLX

4.10 Μετατροπή του SystemC μοντέλου σε RTL VHDL και σύνθεσή του χρησιμοποιώντας το εργαλείο σύνθεσης Xilinx XST.

Αφού ολοκληρώθηκε η υλοποίηση του μοντέλου σε systemC, χρησιμοποιώντας το SC[6] μετατρέπεται το μοντέλο σε RTL VHDL. Στη συνέχεια με τη βοήθεια του εργαλείου σύνθεσης Xilinx XST, και των εργαλείων τοποθέτησης και δρομολόγησης, γίνεται η σύνθεση και προσομοίωση της σχεδίασης. Οι μνήμες της σχεδίασης αρχικοποιήθηκαν έτσι ώστε η μνήμη δεδομένων 1024x32 να περιέχει τις τιμές 0 έως 32 στις διευθύνσεις μνήμης 1 έως 32 και το αρχείο καταχωρητών 32x32 να περιέχει τις τιμές 0 έως 32, δηλαδή η διεύθυνση 0 περιέχει τη τιμή 0, η διεύθυνση 1 περιέχει τη τιμή 1, …, η διεύθυνση 32 περιέχει τη τιμή 32. Στη μνήμη ανάκτησης εντολών αποθηκεύτηκε ένα πρόγραμμα (πίνακας 4-1) το οποίο ελέγχει την υλοποίηση όλων των υποστηριζόμενων εντολών, φαίνεται η τεχνική προώθησης δεδομένων όταν υπάρχουν εξαρτήσεις δεδομένων, καθώς και χρησιμοποίηση καταχωρητών στους οποίους έγινε αλλαγή αποτελέσματος λόγω εκτέλεσης προηγούμενης εντολής.

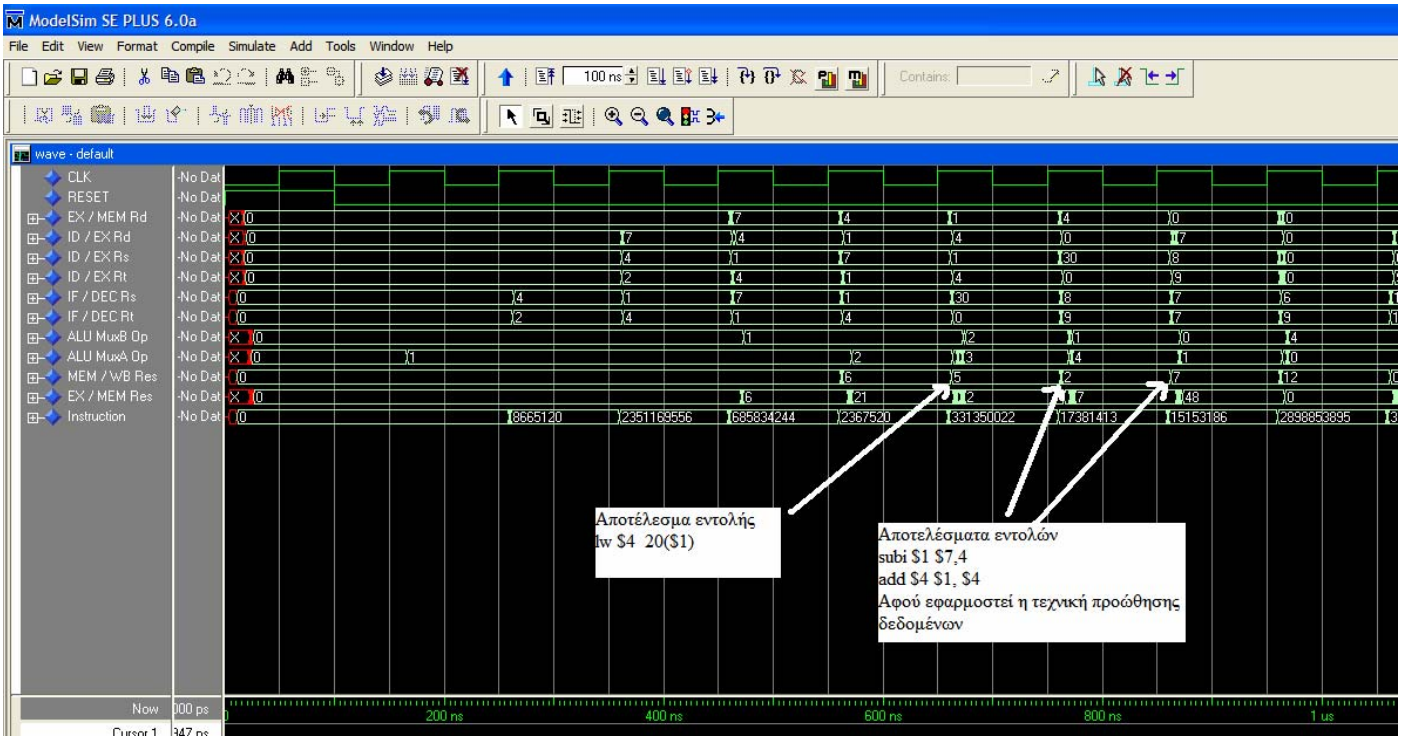
Η προσομοίωση γίνεται στο περιβάλλον του ModelSim 6.0a. Στα σχήματα 4-8 έως 4-11 φαίνονται οι κυματομορφές προσομοίωσης κατά την εκτέλεση του προγράμματος του πίνακα 4-1. Διακρίνονται οι περιπτώσεις επιτυχούς υπό συνθήκης διακλάδωσης, περιπτώσεις όπου χρησιμοποιείται η τεχνική προώθησης δεδομένων για αντιμετώπιση περιορισμών δεδομένων καθώς και περιπτώσεις όπου υπάρχει εκτέλεση εντολής άλματος.

Παρακάτω παρατίθενται οι κυματομορφές για επαλήθευση του net list του κυκλώματος του υπό εκτέλεση προγράμματος, με χρήση του ModelSim 6.0a.

Θέση Μνήμης	Εντολή
4	add $7 $4, $2
5	add $1 $1, $2
8	lw $4 20($1)
9	add $1 $1, $1
12	subi $1 $7, 4
13	add $1 $1, $5
16	add $4 $1, $4
17	j 7
20	beqz $30, 6
21	sub $20 $20, $20
24	or $7 $8, $9
25	add $21 $21, $21
28	sub $7 $7, $7
29	add $22 $22, $22
32	sw $9 7($6)
36	xori $11 $11, $11
40	bne $30, 5
44	and $12 $13, $12
48	sub $15 $16, $1
49	add $5 $5, $5
52	jr 5
53	j -1

Πίνακας 4-1.

Υλοποίηση απλού προγράμματος για έλεγχο σχεδίασης.



Σχήμα 4-8

Net list του κυκλώματος με τις με τις προδιαγραφές που περιγράφονται στο τμήμα 4-10. Στο σχήμα διακρίνονται τα αποτελέσματα των εξής εντολών:

add $7 $4, $2

lw $4 20($1)

subi $1 $7,4

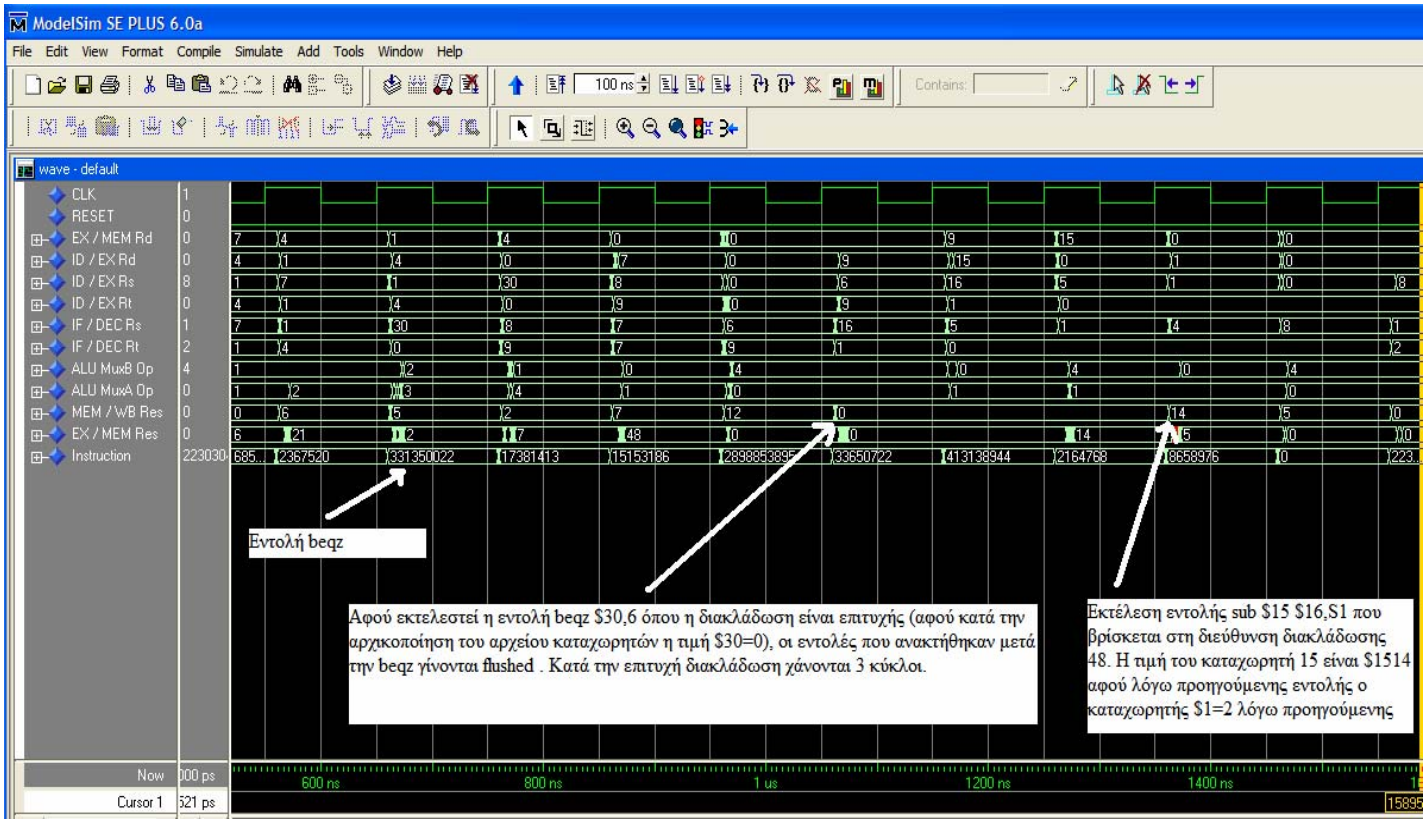
add $4 $1, $4

Αρχικές τιμές καταχωρητών αρχείου καταχωρητών: $1=1, $2=2, $3=3, …, $7=7.

Αρχικές τιμές μνήμης δεδομένων: $1=1, $2=2, $3=3, …, $7=7.

Στα αποτελέσματα λόγω με ασπρο βελάκι, φαίνεται η λειτουργία της τεχνικής προώθησης δεδομένων αφού λόγω της add ο καταχωρητής $7 παίρνει τιμή $7=6 αντί $7=7 που ήταν αρχικοποιημένο το αρχείο καταχωρητών. Λόγω της lw ο καταχωρητής $4 παίρνει τιμή $4=5 αντί της τιμής $4=4 που ήταν αρχικοποιημένο το αρχείο καταχωρητών. Επίσης λόγω της subi ο

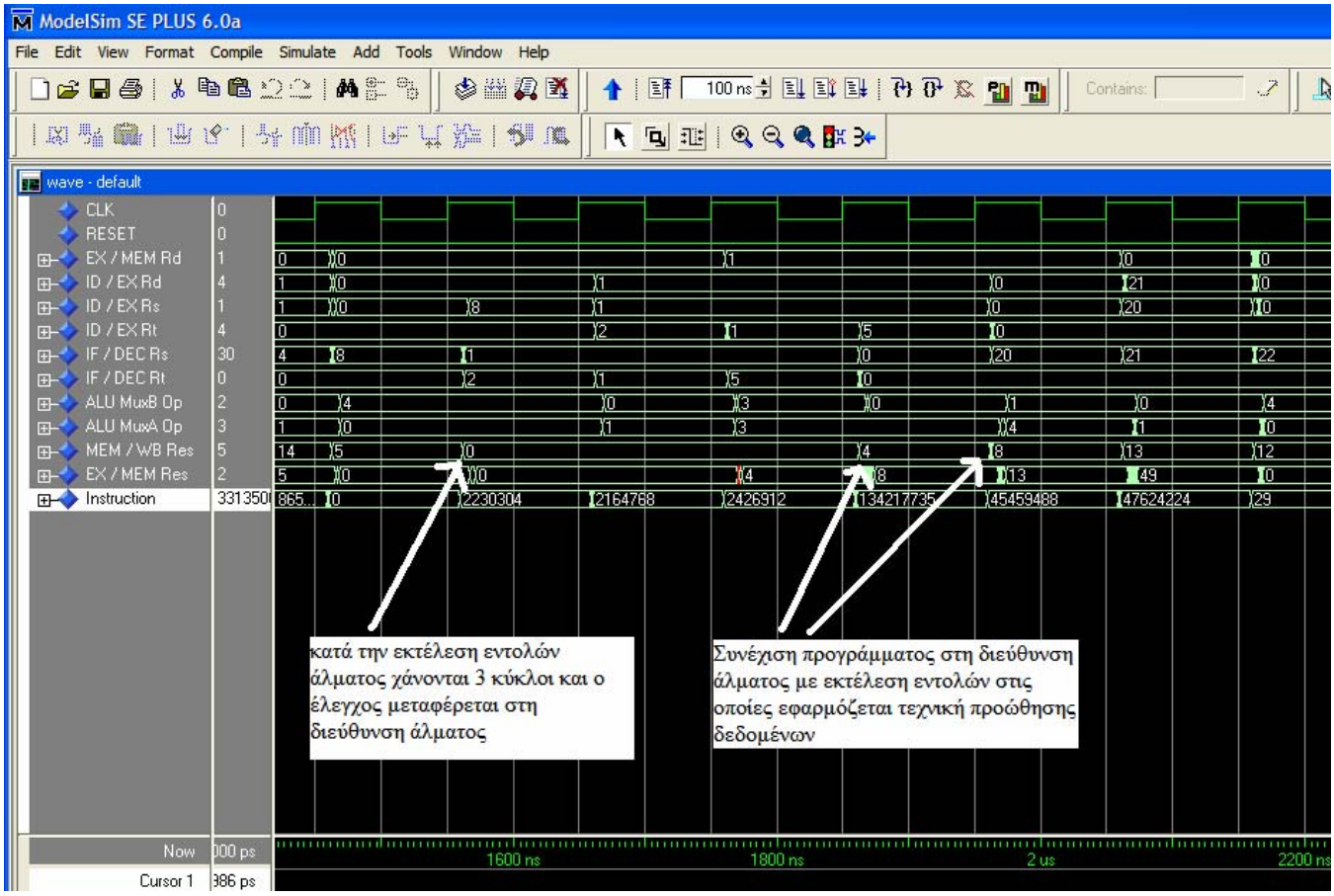
καταχωρητής $1 παίρνει τιμή $1=2 αντί της τιμής $1=1 που ήταν αρχικοποιημένο το αρχείο καταχωρητών. Χωρίς τη προώθηση δεδομένων η τιμή του καταχωρητή $1 της subi θα ήταν $1=3. Με τη προώθηση δεδομένων λόγω των πιο πάνω αλλαγών, η τιμή του $1 γίνεται $1=2 (δείχνεται με βελάκι στο σχήμα). Επίσης η τιμή του καταχωρητή $4 της add χωρίς προώθηση δεδομένων θα ήταν $4=5. Με την εφαρμογή όμως της προώθησης δεδομένων το αποτέλεσμα είναι $4=7(δείχνεται με βελάκι στο σχήμα).



Σχήμα 4-9.

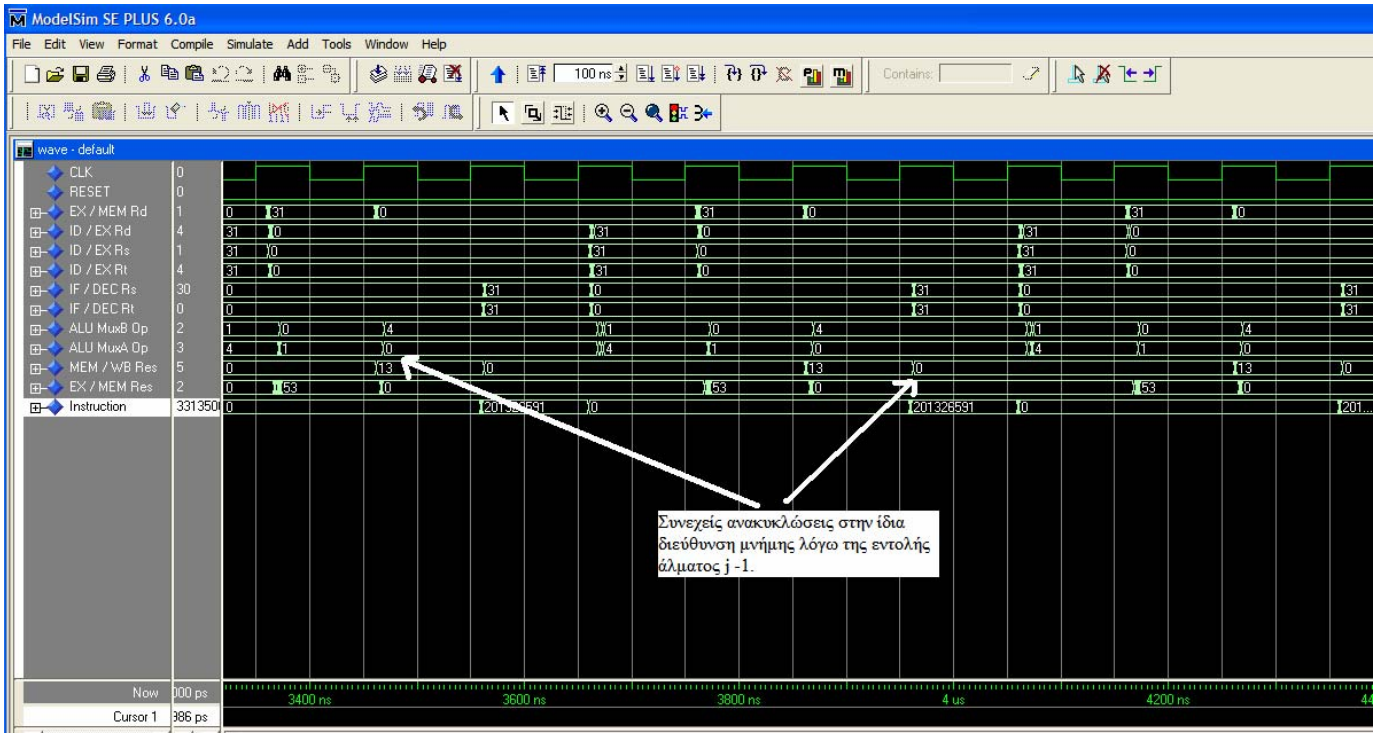
Συνέχεια εκτέλεσης προγράμματος του πίνακα 4-1. Φαίνεται η συμπεριφορά αντιμετώπισης των υπο συνθήκη διακλαδώσεων. Αρχικά θεωρούνται οι διακλαδώσεις ανεπιτυχείς, οπόταν η ροή εντολών στο ομόχειρο μονοπάτι συνεχίζεται. Εάν όμως η διακλάδωση είναι επιτυχής, γίνονται flushed οι εντολές που ακολουθούν τη εντολή διακλάδωσης και ο έλεγχος μεταφέρεται

στη διεύθυνση διακλάδωσης. Σε περίπτωση επιτυχούς διακλάδωσης αυξάνεται κατά 3 κύκλους ο κύκλος εκτέλεσης της εντολής που ακολουθεί την εντολή διακλάδωσης, αυξάνοντας έτσι και το CPI, ενώ σε περίπτωση μη-επιτυχούς διακλάδωσης δεν υπάρχει κανένα κόστος.



Σχήμα 4-10.

Συνέχιση εκτέλεσης προγράμματος όπου κατά την εκτέλεση εντολής άλματος αυξάνεται κατά 3 κύκλους ο κύκλος εκτέλεσης της εντολής που ακολουθεί την εντολή άλματος, αυξάνοντας έτσι και το CPI.



Σχήμα 4-11.

Συνέχιση προγράμματος όπου φαίνεται η συνεχής ανακύκλωση του προγράμματος στο ίδιο σημείο λόγω της εντολής j-1 (ανακύκλωση στον εαυτό της).

4.11 Έκθεση σχεδίασης μετά τη τοποθέτηση-δρομολόγησή της με το εργαλείο Xilinx ISE 7.1

Η σχεδίαση όπως προαναφέρθηκε συντέθηκε και δρομολογήθηκε σε μια Xilinx SPARTAN-3 xc3s400 FPGA. Μετά την ολοκλήρωση της σύνθεσης λαμβάνεται μια έκθεση σχεδίασης που δείχνει πόσα LUTs της FPGA χρησιμοποιήθηκαν, πόσες I/O ακίδες χρησιμοποιήθηκαν, πόσα μπλοκ μνήμης RAM χρησιμοποιήθηκαν καθώς και διάφορες άλλες πληροφορίες. Η παραγόμενη έκθεση φαίνεται στο σχήμα 4-11. Αξίζει να σημειωθεί ότι στην έκθεση αυτή παρουσιάζεται ότι χρησιμοποιήθηκαν 1980 4 input LUTs. Εάν κάποιος δοκιμάσει

να χρησιμοποιήσει μια πιο μικρή FPGA για τη τοποθέτηση-δρομολόγηση της σχεδίασης, παρατηρεί ότι ενδεχόμενα ο αριθμός αυτός μειώνεται.. Αυτό συμβαίνει διότι κατά τη διαδικασία της τοποθέτησης – δρομολόγησης η σχεδίαση απλώνεται στα LUTs χωρίς να δίνεται έμφαση στον αριθμό που θα καταληφθούν. Εάν η σχεδίαση τποποθετηθεί σε μια μικρότερη SPARTAN-3 xc3s200 FPGA (σχήμα 4.12), φαίνεται ότι ο αριθμός αυτός των 4 input LUTs παραμένει ο ίδιος.

Design Overview for dlx

Property	Value
Project Name:	d:\christodoulos\spartan datapath
Target Device:	xc3s400
Report Generated:	Thursday 03/09/06 at 13:09
Printable Summary (View as HTML)	dlx_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	1,010	7,168	14%	
Number of 4 input LUTs:	1,980	7,168	27%	
Logic Distribution:				
Number of occupied Slices:	1,280	3,584	35%	
Number of Slices containing only related logic:	1,280	1,280	100%	
Number of Slices containing unrelated logic:	0	1,280	0%	
Total Number of 4 input LUTs:	1,980	7,168	27%	
Number of bonded IOBs:	49	141	34%	
Number of Block RAMs:	8	16	50%	
Number of GCLKs:	1	8	12%	

Performance Summary

Property	Value
Final Timing Score:	54024
Number of Unrouted Signals:	All signals are completely routed.
Number of Failing Constraints:	0

Failing Constraints (total failing = 1)

Constraint(s)	Requested	Actual	Logic Levels
* OFFSET = OUT 9 ns AFTER COMP "clock"	9.000ns	11.426ns	1

Σχήμα 4-11.

Έκθεση σχεδίασης μετά τη τοποθέτηση- δρομολόγηση σε SPARTAN-3 xc3s400 FPGA.

Design Overview for dlx

Property	Value
Project Name:	c:\documents and settings\christodoulos\desktop\dokimes\spartan s200
Target Device:	xc3s200
Report Generated:	Thursday 03/09/06 at 19:21
Printable Summary (View as HTML)	dlx_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	1,010	3,840	26%	
Number of 4 input LUTs:	1,980	3,840	51%	
Logic Distribution:				
Number of occupied Slices:	1,280	1,920	66%	
Number of Slices containing only related logic:	1,280	1,280	100%	
Number of Slices containing unrelated logic:	0	1,280	0%	
Total Number of 4 input LUTs:	1,980	3,840	51%	
Number of bonded IOBs:	49	141	34%	
Number of Block RAMs:	8	12	66%	
Number of GCLKs:	1	8	12%	

Performance Summary

Property	Value
Number of Unrouted Signals:	All signals are completely routed.
Number of Failing Constraints:	0

Σχήμα 4-12.

Έκθεση σχεδίασης μετά τη τοποθέτηση- δρομολόγηση σε SPARTAN-3 xc3s200 FPGA. Ο απαιτούμενος αριθμός 4 input LUTs σε αυτή τη περίπτωση είναι ίδιος (1980) με αυτό που απαιτείται για τοποθέτηση της σχεδίασης σε μια SPARTAN-3 xc3s400 FPGA.

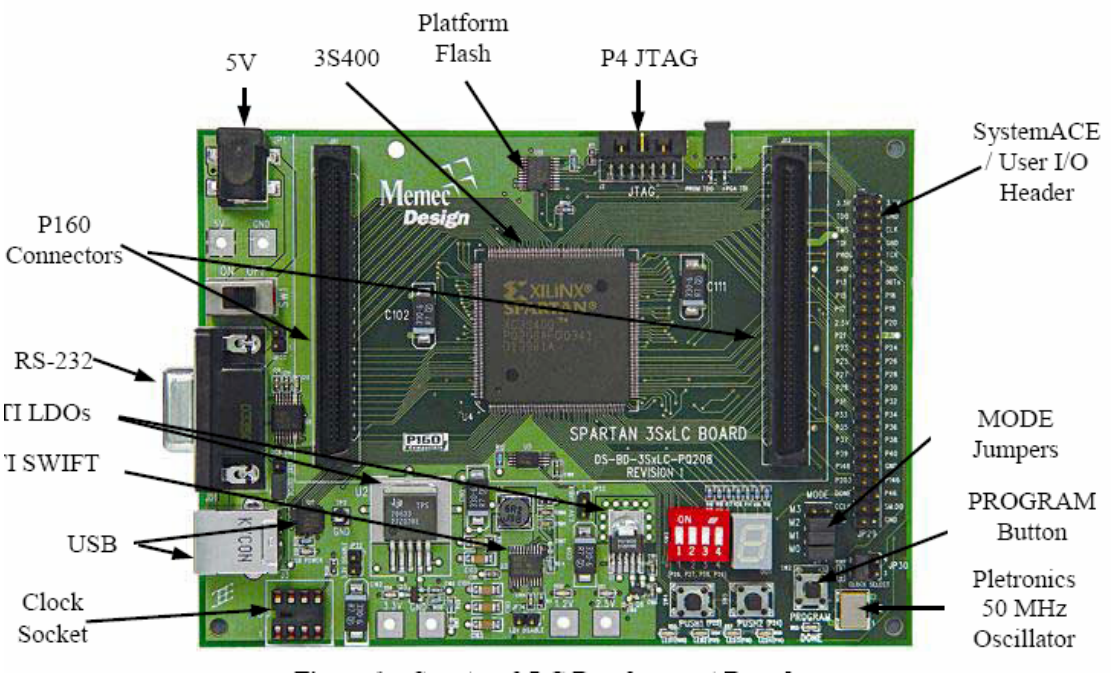
ΚΕΦΑΛΑΙΟ 5
ΥΛΟΠΟΙΗΣΗ ΣΕ SPARTAN-3 xc3S400 FPGA ΚΑΙ ΕΠΙΒΕΒΑΙΩΣΗ ΛΕΙΤΟΥΡΓΙΑΣ

5.1 Εισαγωγή

Αφού ολοκληρώθηκε η σχεδίαση, η σύνθεση και τοποθέτηση των πυλών με τα εργαλεία σύνθεσης XST της Xilinx, η σχεδίαση υλοποιήθηκε σε μια Spartan-3 xc3s400 FPGA. Η FPGA αυτή είναι τοποθετημένη σε ένα αναπτυξιακό της Memec, το οποίο περιγράφεται παρακάτω. Η επιβεβαίωση της λειτουργίας του κυκλώματος γίνεται με τη βοήθεια ενός λογικού αναλυτή στην είσοδο του οποίου ενώνεται το σήμα εξόδου από την FPGA. Επίσης τροποποιείται ο κώδικας έτσι ώστε να ελέγχεται ο μικροεπεξεργαστής από τα μέσα αλληλεπίδρασης χρήστη/αναπτυξιακού όπως οι 4 διακόπτες, το 7-segment display και τα 4 led. Εκτενέστερη περιγραφή της διαδικασίας γίνεται παρακάτω.

5.2 Περιγραφή αναπτυξιακού της MEMEC

Το αναπτυξιακό Memec Spartan-3 LC[12] που χρησιμοποιήθηκε για την υλοποίηση της σχεδίασης στην FPGA είναι μια πλατφόρμα σχετικά χαμηλού κόστους, η οποία χρησιμοποιείται για ανάπτυξη σχεδιασμών και εφαρμογών που βασίζονται στην οικογένεια Xilinx Spartan-3 FPGA. Προσφέρει 400k πυλών σε ένα ευέλικτο σχεδιαστικό χώρο. Το αναπτυξιακό που φαίνεται στο σχήμα 5-1, αποτελείται από την συγκεκριμένη FPGA, μια πηγή τροφοδοσίας, ένα P4-to-P3 καλώδιο εφαρμογής και εγχειρίδιο χρήστη. Το ρολόι υλοποιείται από ένα ταλαντωτή της οικογένειας Pletronics, στα 50MHz. Στο πορτάκι υπάρχουν ακόμα 4 ενσωματωμένα ποδαράκια που παρέχουν 3.3V ταλαντωτές.



Σχήμα 5-1

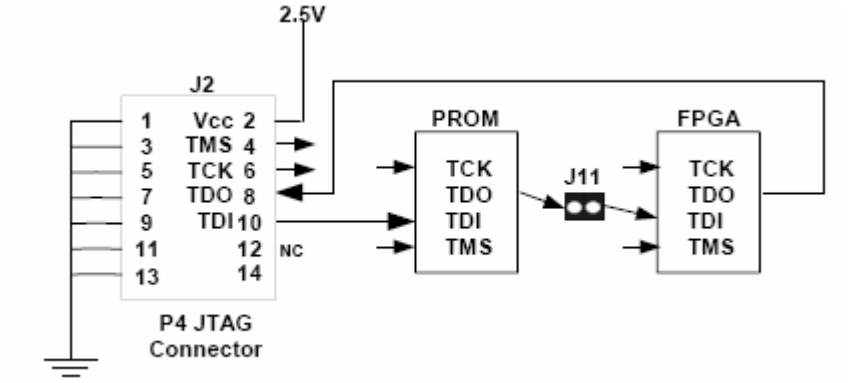
Spartan-3 LC αναπτυξιακό

Για τη διεπαφή του χρήστη με το πορτάκι, υπάρχουν αρκετά μέσα αλληλεπίδρασης με το χρήστη, τα οποία είναι τα εξής:

- Ένα κοινής ανόδου 7-segment LED display το οποίο μπορεί να χρησιμοποιηθεί κατά το έλεγχο και την έρευνα για λάθη (debugging) στη σχεδίαση.
- Τέσσερα LEDs
- Δύο κομβία (push buttons) για την εναλλαγή των εισόδων στην Spartan-3 FPGA.
- Τέσσερις DIP διακόπτες.
- Μια θύρα RS232
- Μια USB θύρα

Σε ότι αφορά το προγραμματισμό της FPGA μέσω του αναπτυξιακού, παρέχονται τρεις διαφορετικές μέθοδοι οι οποίες είναι οι εξής:

- JTAG θύρα, η οποία είναι μια παράλληλη θύρα διασύνδεσης του αναπτυξιακού με το Η/Υ που δίνει τη δυνατότητα προγραμματισμού της FPGA μέσω μιας αλυσίδας (σχήμα 5-2), η οποία περιλαμβάνει αρχικά το προγραμματισμό της ενσωματωμένης στο ISP PROM, και μέσω αυτής το προγραμματισμό της FPGA.



Σχήμα 5-2

Περιγραφή JTAG αλυσίδας

- SystemACE συνδετήρας, ο οποίος δίνει τη δυνατότητα στους σχεδιαστές λογισμικού να δημιουργήσουν ένα αρχείο συστήματος στη μετακινήσιμη CompactFlash κάρτα.
- Δύο P160 σχισμές προέκτασης, οι οποίες υποστηρίζουν διασύνδεση του αναπτυξιακού με άλλες υπομονάδες, για διάφορες εφαρμογές.

5.3 Προγραμματισμός του αναπτυξιακού μέσω της JTAG θύρας

Η υλοποίηση της σχεδίασης στο αναπτυξιακό[13], έγινε μέσω της θύρας JTAG, ως εξής:

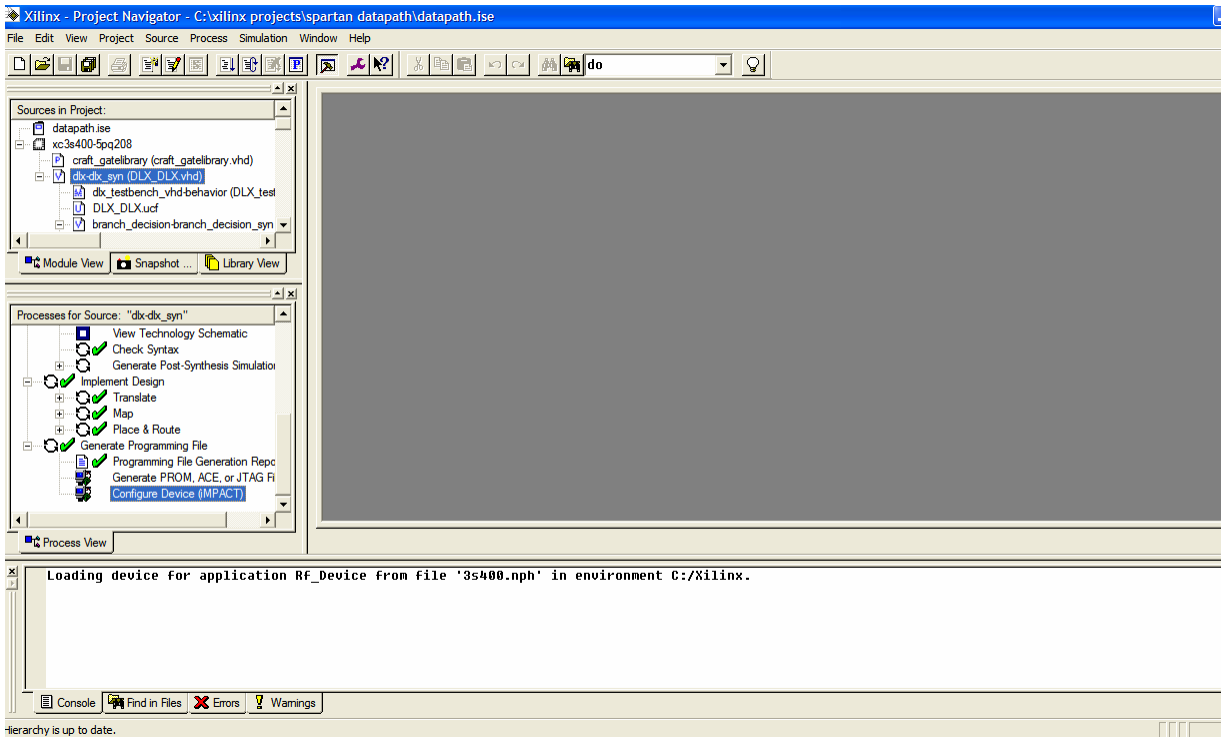
- Αφού έγινε με επιτυχία η τοποθέτηση και δρομολόγηση των πυλών της σχεδίασης μέσω του εργαλείου Xilinx ISE 7.1, δημιουργείται ένα αρχείο περιορισμών (.ucf αρχείο), στο οποίο δηλώνονται σε ποιες ακίδες (pins) της FPGA θα ενωθούν οι είσοδοι και έξοδοι της σχεδίασης. Σαν είσοδοι είναι το ρολόι που συνδέεται στο ταλαντωτή και το reset που συνδέεται στο push button 1. Σαν έξοδος ορίστηκε η ανακτώμενη εντολή σε κάθε κύκλο ρολογιού (32-bit ποσότητα). Κάθε bit της εντολής ενώνεται σε κάποια τυχαία I/O ακίδα της FPGA. Για σκοπούς ευκολίας στην εύρεση των συγκεκριμένων ακίδων εξόδου της σχεδίασης για τη μετέπειτα διασύνδεση τους με το λογικό αναλυτή, τα bits εξόδου που αντιστοιχούν στο πεδίο opcode (6 bits) και στο πεδίο function (6 bits) της εντολής, συσχετίστηκαν με τις αντίστοιχες I/O ακίδες του systmACE connector του αναπτυξιακού, ενώ τα υπόλοιπα bit της εντολής συσχετίσηκαν με τις I/O ακίδες της P160 σχισμής προέκτασης. Ο τύπος κάθε ακίδας της FPGA περιγράφεται στις προδιαγραφές της συγκεκριμένης FPGA στο site www.xilinx.com[14]. Η συσχέτιση των I/O ακίδων του αναπτυξιακού με την έξοδο της σχεδίασης φαίνεται στο πίνακα 5-1.

			Board's Pins	
Instruction bit	FPGA pin #	Pin Name	SystemACE connector pin#	JX1 pin#
0	P90	IO_L27P_4/D1	P90	
1	P93	IO	P93	
2	P120	IO_L22N_3	P120	
3	P115	IO_L20N_3	P115	
4	P109		P109	
5	P71	IO	P71	
6	P141	IO_L23N_2/VREF_2		P141
7	P143	IO_L22P_2		P143
8	P144	IO_L22N_2		P144
9	P146	IO_L21P_2		P146
10	P147	IO_L21N_2		P147
11	P148	IO_L20P_2		P148
12	P149	IO_L20N_2		P149
13	P150	IO_L19P_2		P150
14	P152	IO_L19N_2		P152
15	P187	IO_L31N_0		P187
16	P189	IO		P189
17	P190	IO_L30P_0		P190
18	P191	IO_L30N_0		P191
19	P194	IO_L27P_0		P194
20	P196	IO_L27N_0		P196
21	P197	IO		P197
22	P198	IO_L25P_0		P198
23	P199	IO_L25N_0		P199
24	P165	IO_L10P_1		P165
25	P166	IO_L10N_1/VREF_1		P166
26	P119	IO_L22P_3	P119	
27	P111	IO_L19P_3	P111	
28	P117	IO_L21N_3	P117	
29	P114	IO_L20P_3	P114	
30	P94	IO_L25P_4	P94	
31	P113	IO_L19N_3	P113	

Πίνακας 5-1

Αντιστοίχηση I/O ακίδων της FPGA xc3S400 με τις I/O ακίδες του αναπτυξιακού

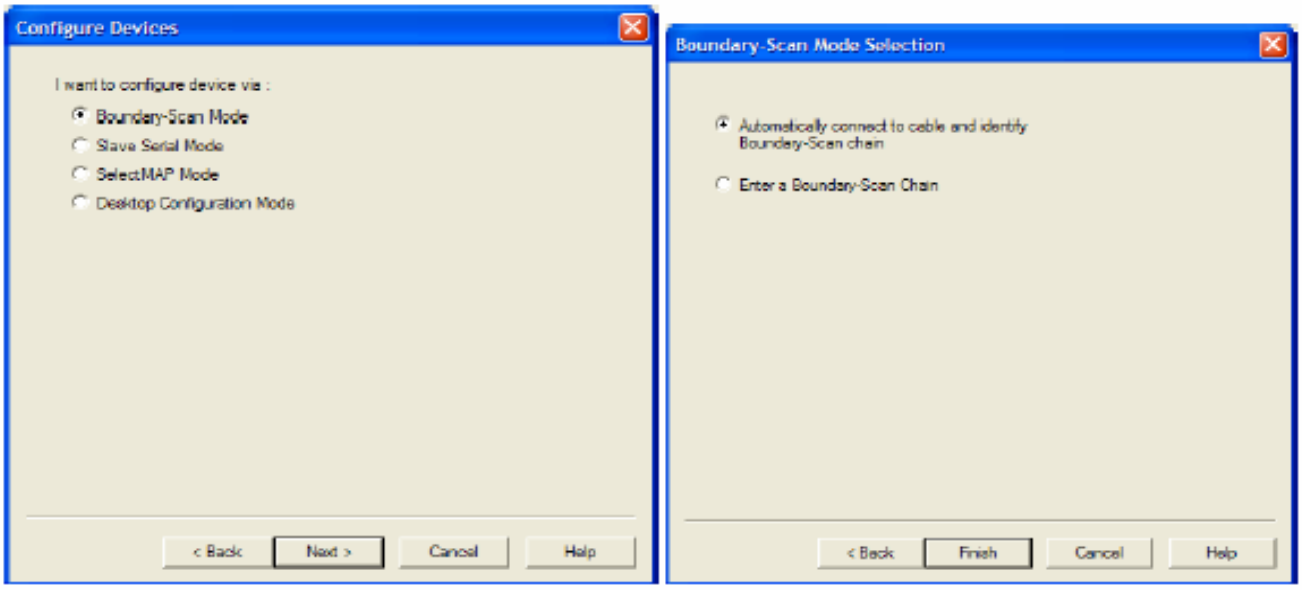
- δημιουργείται το ρεύμα από bits (bitstream) χρησιμοποιώντας την επιλογή **Generate Programming File** μέσα από το project.
- Ακολούθως φορτώνεται το ρεύμα αυτό των bits, στην FPGA, μέσω της JTAG θύρας, ακολουθώντας τις εξής ενέργειες:
    1. ενώνεται το JTAG καλώδιο από τον Η/Υ στο αναπτυξιακό. Ακολούθως παρέχεται τροφοδοσία και αφαιρούνται τα MODE jumpers (J1) έτσι ώστε να μην φορτωθεί μέσω της PROM μνήμης η FPGA. Τέλος στο πεδίο **Generate Programming File** επιλέγεται (διπλό κλικ) η επιλογή **configure Device (iMPACT) (σχήμα 5-1).**



Σχήμα 5-1

Επιλογή Configure Device (iMPACT)

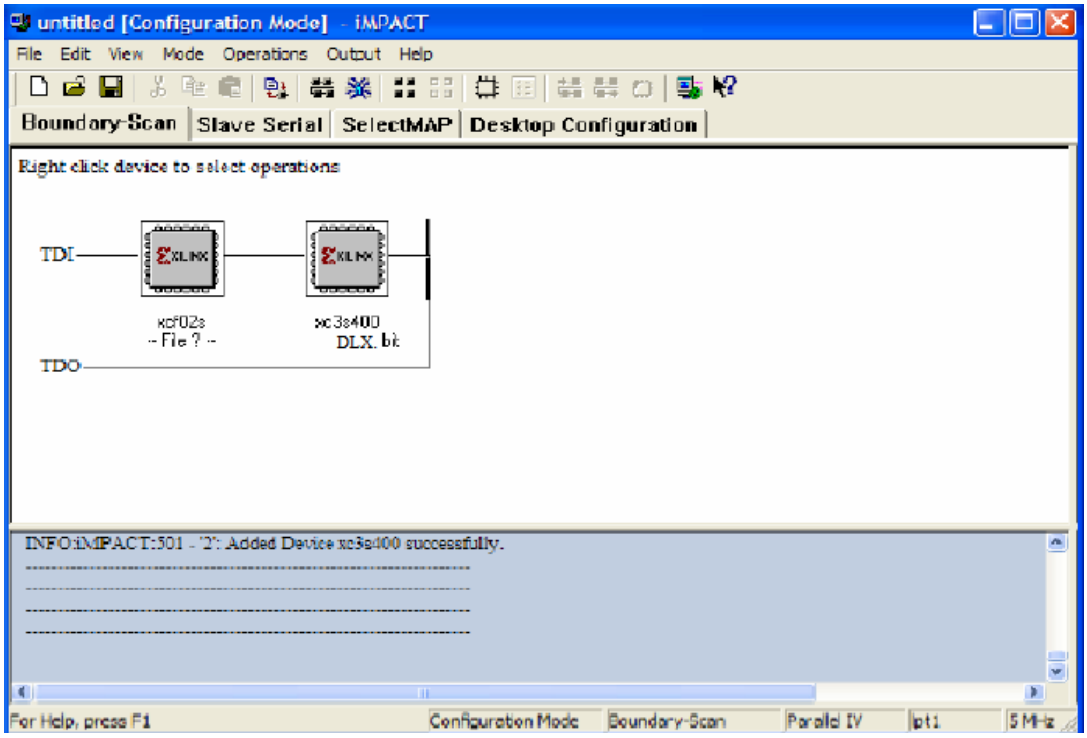
2. Οι επιλογές του προγράμματος οδήγησης του iMPACT μένουν ως έχουν (σχήμα 5-2).



Σχήμα 5-2

Επιλογές οδήγησης προγράμματος iMPACT

3. Στην επιλογή **Assign New Configuration File** επιλέγεται η FPGA και εισάγεται σε αυτή το αρχείο DLX.bit (σχήμα 5-3).



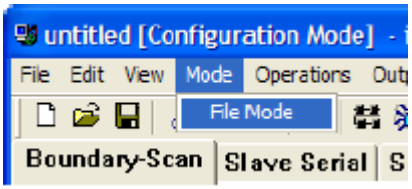
Σχήμα 5-3

Εισαγωγή bitstream DLX.bit στην FPGA

4. Με αριστερό κλικ στην FPGA επιλέγεται η επιλογή **Program**
5. Στις επιλογές προγραμματισμού **δεν** επιλέγεται το πεδίο **verify** και πατιέται το **OK**.
6. Η ροή των bit κατεβαίνει μέσω του iMPACT στην FPGA. Αν όλα πάνε καλά στην οθόνη του Η/Υ αναβοσβήνει η λέξη Programming Succeeded. Αλλιώς επαναλαμβάνεται η ίδια διαδικασία.

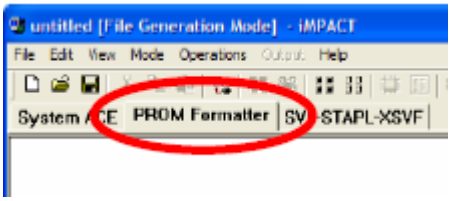
Στη συνέχεια πρέπει να προγραμματιστεί και η PROM έτσι ώστε να φορτώνει το bitstream στην FPGA όταν θα ενεργοποιείται το κύκλωμα, χωρίς να είναι απαραίτητο το ξαναφόρτωμα του μέσω του iMPACT. Για να γίνει αυτό ακολουθούνται τα εξής βήματα:

1. Αρχικά δημιουργείται το .mcs PROM είδωλο ως εξής: Στο iMPACT επιλέγεται **Mode→File Mode (σχήμα 5-4).**



Σχήμα 5-4

2. επιλέγεται το πεδίο **PROM Formatter (σχήμα 5.5).**

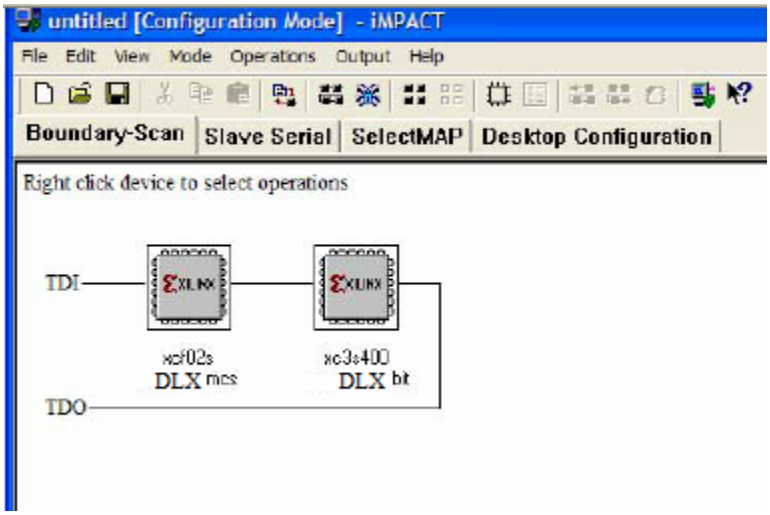


Σχήμα 5-5.

3. Στη συνέχεια με αριστερό κλικ επιλέγεται **Launch Wizard**. Επιλέγεται η οικογένεια xcf02s της PROM και εισάγεται στο wizard. Ακολούθως εισάγεται η ροή των bits DLX.bit του κυκλώματος, και απορρίπτεται η ερώτηση για περισσότερα αρχεία σχεδίασης στη PROM. Με την επιλογή **FINISH** το .mcs αρχείο υπάρχει στο φάκελο του project.

Για την εισαγωγή του .mcs αρχείου στην FPGA ακολουθούνται οι εξής ενέργειες:

4. Στο iMPACT επιλέγεται **Mode→Configuration Mode,** οπόταν το πρόγραμμα επιστρέφει στην JTAC αλυσίδα του αναπτυξιακού. Επιλέγεται η PROM (διπλό κλικ στην xcf2s PROM εικόνα) και εισάγεται σε αυτή το αρχείο .mcs που δημιουργήθηκε πιο πριν(σχήμα 5-6).



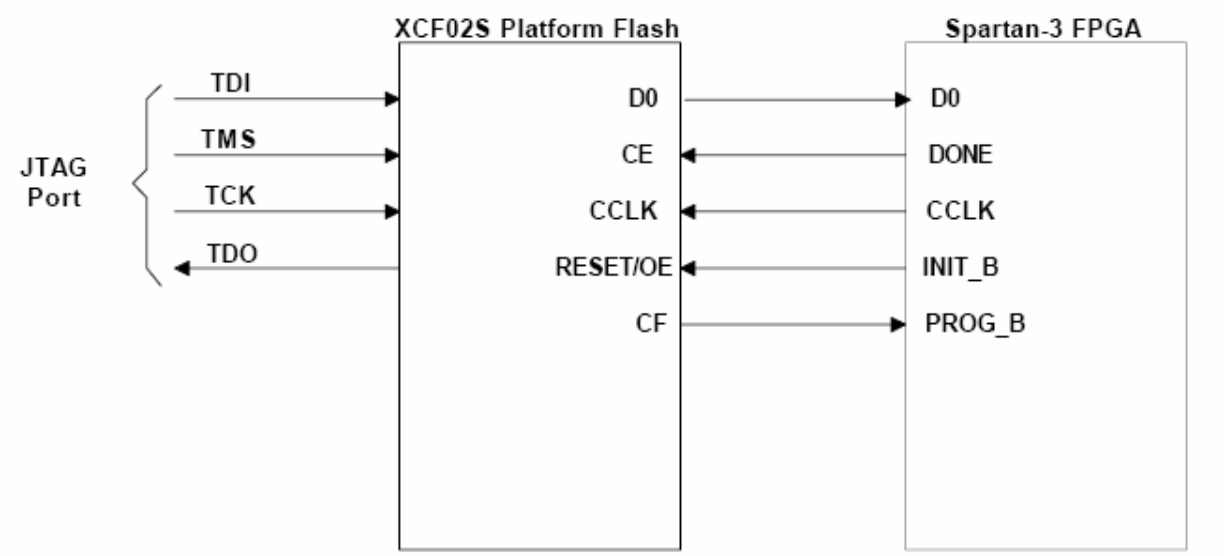
Σχήμα 5-6

5. Με διπλό κλικ στην εικόνα της xcf02s PROM επιλέγεται το **Program**. Πριν την ολοκλήρωση του προγραμματισμού επιλέγονται τα πεδία **Erase Before Programming** και **Verify.**
6. Μετά από κάποια καθυστέρηση μέχρις ότου σβηστεί η PROM, το iMPACT προγραμματίζει και επιβεβαιώνει τη PROM. Εάν είναι επιτυχής, ένα μήνυμα Programming Succeeded εμφανίζεται στην οθόνη του Η/Υ.
7. Μετά τον επιτυχή προγραμματισμό, σβήνει το κύκλωμα θέτοντας το διακόπτη SW1 στο OFF. Επειδή επιδίωξη είναι η αρχικοποίηση της FPGA να γίνεται μέσω της PROM κάθε φορά που παρέχεται τροφοδοσία στο κύκλωμα, πρέπει τα δεδομένα εξόδου (DO), το ρολόι (CCLK), το RESET/OE, και το CF σήματα της PROM να αρχικοποιούν την FPGA (σχήμα 5-7). Αυτό επιτυγχάνεται θέτοντας το Configuration Mode του αναπτυξιακού σε Master Serial Mode (πίνακας 5-2). Για να γίνει αυτό τα M0, M1, M2, βραχυκυκλώνονται με Jumpers. Αφού γίνουν αυτά, παρέχεται ξανά τροφοδοσία στο αναπτυξιακό. Πλέον η PROM αυτόματα αρχικοποιεί την FPGA.

Mode	J1		
	5-6 (M2)	3-4 (M1)	1-2 (M0)
Master Serial	Closed	Closed	Closed
Slave Serial	Open	Open	Open
Master Parallel	Closed	Open	Open
Slave Parallel	Open	Open	Closed
JTAG	Open	Closed	Open

Πίνακας 5-2

Επιλογή configuration mode στο αναπτυξιακό

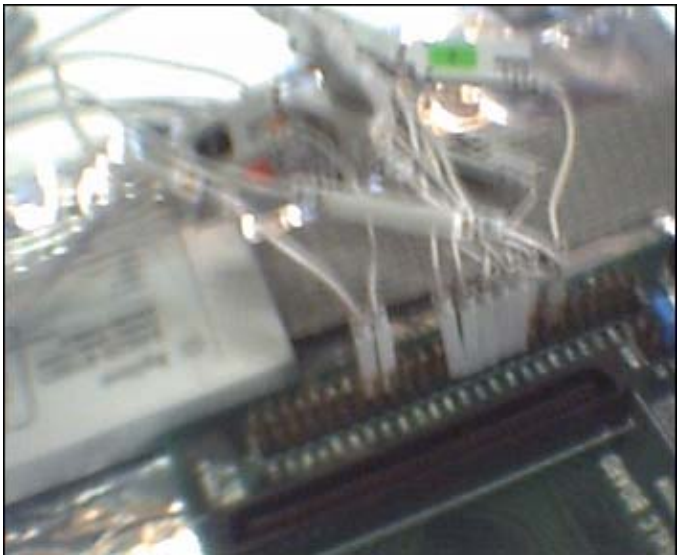


Σχήμα 5-7

Διασύνδεση ISP PROM στη JTAG πύλη και η πύλη διαμόρφωσης της FPGA στην αλυσίδα.

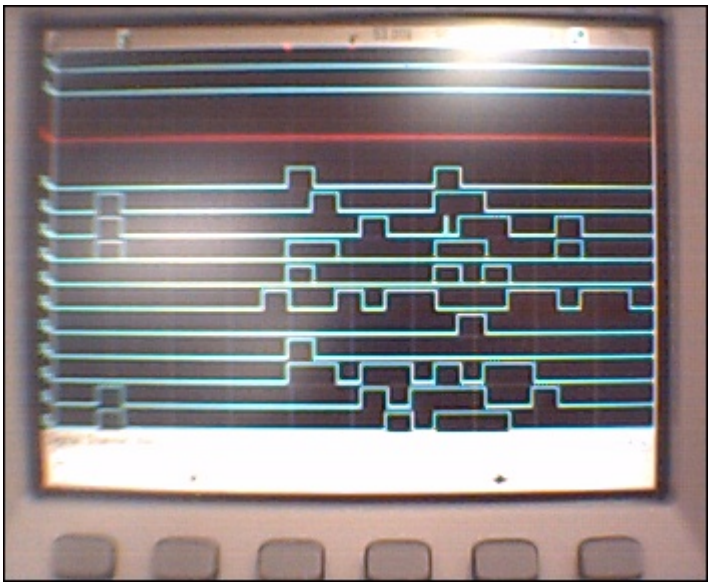
5.4 Επιβεβαίωση λειτουργίας κυκλώματος μέσω του λογικού αναλυτή.

Για την επιβεβαίωση της ορθής λειτουργίας του κυκλώματος σύμφωνα με τις προδιαγραφές υλοποίησης, μέρος από τα δεδομένα εξόδου, το opcode 6-bit και το function 6-bit, οδηγήθηκαν στην είσοδο του λογικού αναλυτή έτσι ώστε να αναπαρασταθούν σε κυματομορφές. Συγκεκριμένα χρησιμοποιούνται δύο σετ από κανάλια (pods) (σχήμα 5-8) καθένα από τα οποία έχει 8 εξόδους (8-bit). Αφού πρόκειται να ελεγχθούν μόνο το opcode και το function του διανύσματος εξόδου (σύνολο 12-bit), επιλέγονται οι έξοδοι 0 έως 5 από το πρώτο σετ καναλιών για να συσχετιστούν με το function της εντολής και οι έξοδοι 8 έως 13 από το δεύτερο σετ για να συσχετιστούν με το Opcode της εντολής. Επίσης η γείωση του κάθε σετ ενώνεται στη γείωση του κυκλώματος έτσι ώστε να βελτιωθεί η ακρίβεια του σήματος στο ταλαντωτή, εξασφαλίζοντας πιο έγκυρες μετρήσεις. Αφού γίνουν σωστά οι αντιστοιχήσεις των καναλιών με τις I/O ακίδες της FPGA (σήμα 5-8), και τεθεί σε λειτουργία το κύκλωμα, στην οθόνη του λογικού αναλυτή φαίνονται οι κυματομορφές που υποδηλώνουν τις τιμές του κάθε bit των πεδίων function και opcode της εκάστοτε ανακτώμενης εντολής. Πράγματι παρατηρείται ότι όντως εκτελείται το πρόγραμμα που περιγράφεται στο πίνακα 4-1, που αποθηκεύτηκε στη μνήμη. Οι κυματομορφές που λαμβάνονται από το λογικό αναλυτή φαίνονται στο σχήμα 5-9.



Σχήμα 5-8

Διασύνδεση των δύο σετ καναλιών του λογικού αναλυτή με τις κατάλληλες ακίδες (έξοδους της FPGA) του αναπτυξιακού.



Σχήμα 5-9

Στην οθόνη του λογικού αναλυτή φαίνεται η διακύμανση των 6-bit του πεδίου opcode και των 6-bit του πεδίου function της εκάστοτε ανακτώμενης εντολής, όταν το κύκλωμα τεθεί σε λειτουργία.

5.5 Έλεγχος λειτουργίας του DLX με χρήση των DIP διακοπτών, του 7-segment display και των led του αναπτυξιακού.

Αφού επιβεβαιώθηκε η λειτουργία του μικροεπεξεργαστή στην FPGA, χρησιμοποιήθηκαν οι τέσσερις DIP διακόπτες, τα τέσσερα Led και το 7-segment display για τον έλεγχο του μικροεπεξεργαστή εφαρμόζοντας μια τεχνική βασικών memory mapped I/O ως εξής:

- Η μνήμη δεδομένων τροποποιείται έτσι ώστε να είναι πλέον μια δίπορτη μνήμη. Αυτό δεν επηρεάζει την εκμετάλλευση (utilization) ούτε την λειτουργία του μικροεπεξεργαστή, αφού ουσιαστικά δεν υπάρχουν μονόπορτες μνήμες στην FPGA. Όταν δηλώνονται μονόπορτες μνήμες χρησιμοποιείται μόνο η μια πόρτα από τις δίπορτες. Η δεύτερη πόρτα στη δίπορτη αυτή μνήμη, απλά θα χρησιμοποιείται για την επιβεβαίωση λειτουργίας ως εξής: Γίνεται ανάγνωση της δεύτερης πόρτας σε 3 διαφορετικές διευθύνσεις έτσι ώστε :
    1. Όταν η διεύθυνση της μνήμης δεδομένων είναι στη $1^{η}$ διεύθυνση, ανάβει το Led 1.
    2. Όταν η διεύθυνση της μνήμης δεδομένων είναι στη $2^{η}$ διεύθυνση και αν τα δεδομένα προς εγγραφή είναι συγκεκριμένος αριθμός, εμφανίζεται κάποιος προκαθορισμένος αριθμός του 7-segment display.
    3. Όταν ενεργοποιείται κάποιο το deep switch SW_1, ο έλεγχος (PC) μεταφέρεται σε κάποια προκαθορισμένη διεύθυνση μνήμης, όπου διαβάζεται η $3^{η}$ διεύθυνση και φορτώνεται κάποιο δεδομένο στη μνήμη.
    4. Με μια εντολή άλματος ο έλεγχος μεταφέρεται στη $1^{η}$ διεύθυνση.

Δηλαδή εκτελείται η ακολουθία εντολών

```
Label1 : sw SA label($0)
        …
        beqz $A label2
        …
        j label1
```

Με τον τρόπο αυτό γίνεται έλεγχος λειτουργίας της εντολής sw.

ΚΕΦΑΛΑΙΟ 6
ΣΥΜΠΕΡΑΣΜΑΤΑ - ΠΑΡΑΤΗΡΗΣΕΙΣ

6.1 Αποτίμηση Εργασίας

Συνοψίζοντας, σκοπός της παρούσας εργασίας είναι η τριβή με τη γλώσσα systemC και η χρησιμοποίηση της ως μιας γλώσσας περιγραφής υλικού που δίνει τη δυνατότητα σχεδίασης και σύνθεσης ψηφιακών κυκλωμάτων. Στα πλαίσια αυτά υλοποιήθηκε ο κλασσικός ομόχειρος μικροεπεξεργαστής DLX και ο έλεγχος ενός υποσυνόλου της αρχιτεκτονικής συνόλου εντολών του έτσι ώστε να εκτελούνται μόνο απλές εντολές πρόσβασης μνήμης (lw, sw), αριθμητικές και λογικές εντολές (add, sub, and, or, xor, addi, subi, ori, andi, xori), και εντολές ελέγχου εκτέλεσης προγράμματος (beq, bne, j, jr). Υλοποιήθηκαν επίσης μονάδες αντιμετώπισης περιορισμών δεδομένων με τη τεχνική προώθησης δεδομένων, καθώς και με ανασχέσεις ομοχειρίας.

Ακολούθως η systemC σχεδίαση μετατράπηκε σε RTL VHDL με το εργαλείο SC και χρησιμοποιήθηκαν τα εργαλεία σύνθεσης και τοποθέτησης της Xilinx, έτσι ώστε η σχεδίαση να φορτωθεί σε μια Xilinx Spartan-3 xc3s400 FPGA. Η συγκεκριμένη FPGA είναι τοποθετημένη σε αναπτυξιακό της MEMEC, το οποίο παρέχει και θύρες για διεπικοινωνία με άλλα περιφερειακά, καθώς και με Η/Υ. Αρχικά η επιβεβαίωση της σχεδίασης γίνεται μέσα από το εργαλείο ModelSim στο οποίο φαίνονται οι κυματομορφές προσομοίωσης για τον έλεγχο της ροής των εντολών στο ομόχειρο μονοπάτι δεδομένων καθώς και η εξαγωγή των αποτελεσμάτων από την εκτέλεση της κάθε εντολής. Αφού επιβεβαιώθηκε η ορθή λειτουργία, ακολούθησε η υλοποίηση του κυκλώματος στην FPGA.

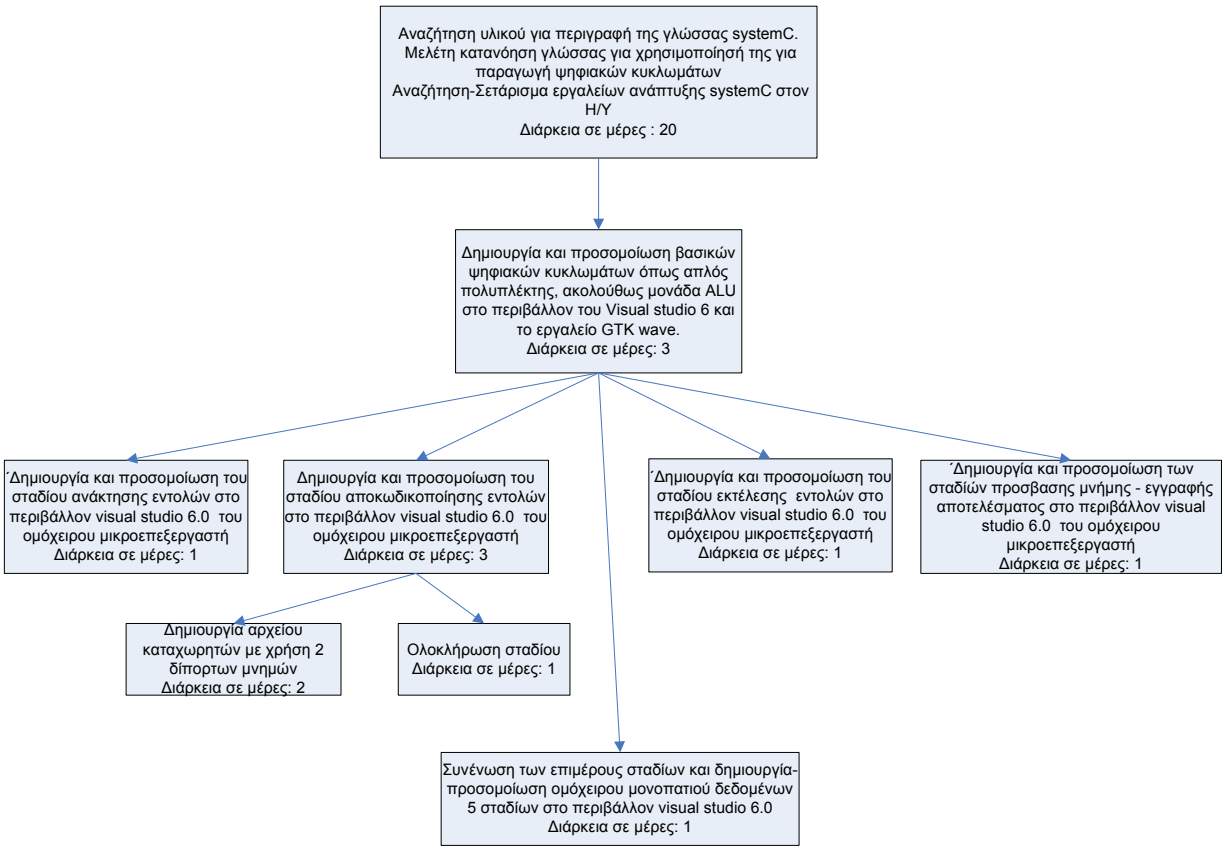
Για τον έλεγχο της ορθότητας της σχεδίασης στην FPGA, χρησιμοποιείται ο λογικός αναλυτής για την αναπαράσταση των ψηφιακών σημάτων στις εξόδους της FPGA. Συγκεκριμένα ελέγχονται το opcode και το function της εκάστοτε ανακαλούμενης εντολής. Από τα αποτελέσματα στην οθόνη του λογικού αναλυτή, επιβεβαιώνεται η λειτουργία της

σχεδίασης και πρακτικά. Τέλος χρησιμοποιούνται οι τέσσερις DIP διακόπτες και το 7-segment display που παρέχει το αναπτυξιακό για σκοπούς επιβεβαίωσης λειτουργίας, έτσι ώστε να ελέγχεται ο DLX μέσω των διακοπτών αυτών, ενώ παράλληλα στην οθόνη του 7-segment display αποτυπώνεται ένας αριθμός ανάλογα με τη κατάσταση στην οποία βρίσκεται το κύκλωμα.

6.2 Παρουσίαση χρονικού διαγράμματος περάτωσης της εργασίας.

Ο προγραμματισμός υλοποίησης της παρούσας εργασίας, ξεκινώντας από μηδενική βάση αφού η γλώσσα systemC ήταν παντελώς άγνωστη στο σχεδιαστή έγινε ως ακολούθως:

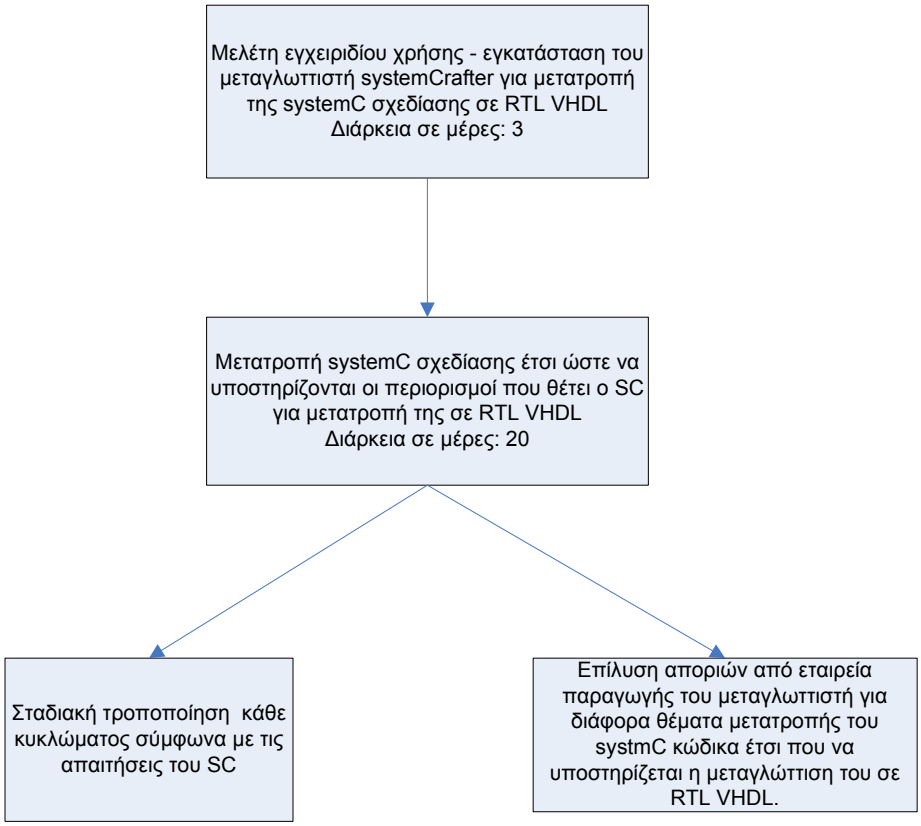
- Φάση $1^{η}$: Μελέτη-κατανόηση γλώσσας, εύρεση – εγκατάσταση εργαλείων στον Η/Υ (σχήμα 6-1).
- Φάση $2^{η}$: Δημιουργία σε systemC και προσομοίωση των ψηφιακών κυκλωμάτων που υλοποιούν το ομόχειρο μονοπάτι δεδομένων του μικροεπεξεργαστή (σχήμα 6-2).
- Φάση $3^{η}$: Μελέτη-κατανόηση του μεταγλωττιστή SC και τροποποίηση σχεδίασης έτσι που να μετατρέπεται σε RTL VHDL μέσω του συγκεκριμένου μεταγλωττιστή. Πλήρης υλοποίηση του μονοπατιού δεδομένων και της μονάδας ελέγχου. Επιβεβαίωση λειτουργίας μέσω προσομοίωσης (σχήμα 6-3).
- Φάση $4^{η}$: Μελέτη κατανόηση αναπτυξιακού MEMEC Spartan-3 LC. Υλοποίηση σχεδίασης στο αναπυξιακό, έλεγχος λειτουργίας με τη χρήση λογικού αναλυτή. Έλεγχος του μικροεπεξεργαστή με τις εισόδους – εξόδους που παρέχει το αναπτυξιακό για αλληλεπίδραση με το χρήστη (σχήμα 6-4).



Συνολική διάρκεια Α φάσης: 30 ημέρες

Σχήμα 6-1

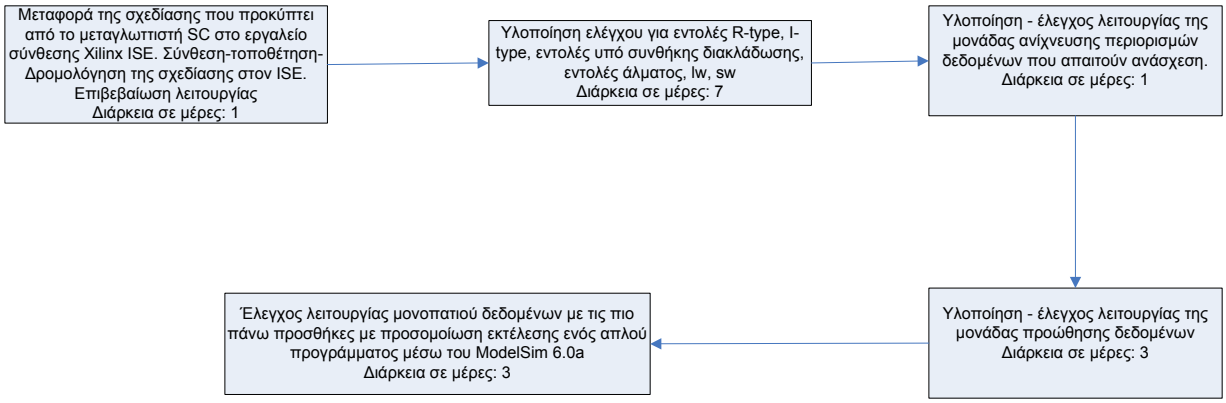
Χρονοδιάγραμμα Α’ φάσης υλοποίησης



Συνολική διάρκεια Β φάσης: 23 ημέρες

Σχήμα 6-2

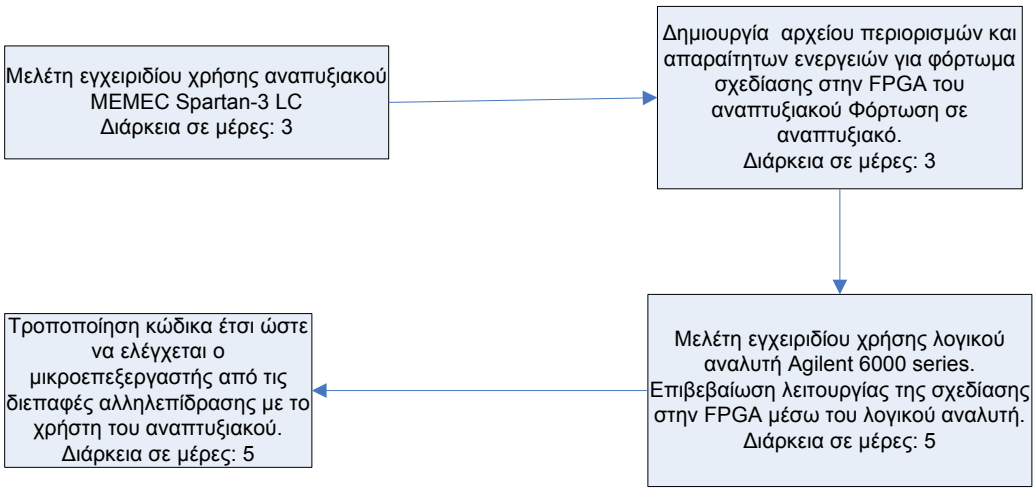
Χρονοδιάγραμμα Β' φάσης υλοποίησης



Συνολική διάρκεια Γ φάσης: 15 μέρες

Σχήμα 6-3

Χρονοδιάγραμμα Γ' φάσης υλοποίησης



Συνολική διάρκεια Δ φάσης: 16 μέρες

Σχήμα 6-4

Χρονοδιάγραμμα Δ' φάσης υλοποίησης

Εάν ο σχεδιαστής είναι ήδη γνώστης τόσο της γλώσσας σχεδίασης systemC, όσο και των περιορισμών που διέπουν τον μεταγλωττιστή μετατροπής της σχεδίασης σε υλικό, ο χρόνος περάτωσης της όλης εργασίας μειώνεται δραματικά. Συγκεκριμένα η $1^{η}$ και $2^{η}$ φάση συμπτύσσονται σε μια με εκτιμώμενο χρόνο περάτωσης και προσομοίωσης της systemC σχεδίασης σε 7 μέρες. Επίσης η $3^{η}$ φάση λόγω πλέον εμπειρίας και ανάπτυξης καλύτερης μεθοδικότητας εκτιμάται ότι παίρνει γύρω στις 7 μέρες. Τέλος η $4^{η}$ φάση μειώνεται δραματικά σε σύνολο 3 ημερών, αφού πλέον η εξοικείωση με το λογικό αναλυτή καθώς και με το αναπτυξιακό μειώνουν κατά πολύ το χρόνο περάτωσης. Άρα σε μια τέτοια περίπτωση ο συνολικός χρόνος περάτωσης και ελέγχου της λειτουργίας της σχεδίασης εκτιμάται σε:

- Φάση $1^{η}$: Υλοποίηση βασικού μονοπατιού δεδομένων του ομόχειρου επεξεργαστή: 7 μέρες.
- Φάση $2^{η}$: Ολοκλήρωση μονοπατιού δεδομένων του ομόχειρου επεξεργαστή με μονάδα ελέγχου, λογική ανίχνευσης περιορισμών δεδομένων που απαιτούν ανάσχεση και λογική προώθησης δεδομένων: 7 μέρες

- Φάση $3^{η}$: Υλοποίηση σχεδίασης στο αναπτυξιακό, έλεγχος λειτουργίας χρησιμοποιώντας λογικό αναλυτή, και έλεγχος μικροεπεξεργαστή από τα παρεχόμενα μέσα αλληλεπίδρασης με χρήστη: 3 μέρες.

6.3 Συμπεράσματα

Η χρησιμοποίηση της systemC για τη σχεδίαση και υλοποίηση του συγκεκριμένου μικροεπεξεργαστή, ήταν μια πολύ καλή εμπειρία εξερεύνησης και εμβάθυνσης των δυνατοτήτων της συγκεκριμένης γλώσσας. Τα συμπεράσματα από την όλη διαδικασία συνοψίζονται στις εξής παρατηρήσεις:

- Με τη systemC η πρόκυψε μια συνεπής και χρονισμένη (cycle accurate) σχεδίαση, που οδήγησε στην επιτυχή και χωρίς προβλήματα (κυρίως χρονισμού) σύνθεση και τοποθέτηση στην FPGA.
- Η ανίχνευση σφαλμάτων κατά τη σχεδίαση ήταν πάρα πολύ εύκολη αφού τις πλείστες φορές τα σφάλματα κατά τη σύνθεση, ήταν ξεκάθαρα και προέτρεπαν στη συγκεκριμένη γραμμή στο κώδικα όπου υπήρχε το σφάλμα. Αντίθετα, προηγούμενη εμπειρία σχεδίασης με χρήση της γλώσσας περιγραφής υλικού VHDL σε πάρα πολλές περιπτώσεις ένα σφάλμα στο κώδικα, προκαλούσε την εμφάνιση μιας σειράς σφαλμάτων συσχετιζόμενων με αυτό, πράγμα που μπερδεύει το σχεδιαστή. Για την εύρεση απλών σφαλμάτων σε αρκετές περιπτώσεις απαιτούνταν πάρα πολύς χρόνος ψαξίματος στη γκάμα των σφαλμάτων μεταγλώττισης, πράγμα που δημιουργούσε εκνευρισμό και άσκοπο χάσιμο χρόνου.
- Ένα άλλο σημαντικό πλεονέκτημα αποτελεί και το γεγονός ότι η σχεδίαση γίνεται σε ψηλότερο επίπεδο από τις κλασσικές γλώσσες περιγραφής υλικού. Το περιβάλλον σχεδίασης και η υποστήριξη των δομών και λειτουργιών της C++ κάνει τη διαδικασία σχεδίασης πιο ανώδυνη, αφού το περιβάλλον σχεδίασης είναι φιλικότερο προς το χρήστη. Επίσης σε αρκετές περιπτώσεις λόγω ευελιξίας, απαιτήθηκε μικρότερος κώδικας για υλοποίηση κάποιων λειτουργιών. Σε γενικές γραμμές η χρησιμοποίηση της systemC για σχεδίαση και υλοποίηση ψηφιακών κυκλωμάτων αντί μιας γλώσσας περιγραφής υλικού όπως η VHDL, είναι πολύ πιο εύκολη και πιο δομημένη

διαδικασία, μιας και η systemC σαν μια βιβλιοθήκη της C++, είναι πιο καλά σχεδιασμένη και δομημένη γλώσσα, με πάρα πολλές δυνατότητες.

- Εξαιτίας των πιο πάνω παρατηρήσεων, ο χρόνος περάτωσης υλοποίησης των επιμέρους τμημάτων της σχεδίασης, είναι σημαντικά πιο μικρός από ότι σε μια σχεδίαση με τη κλασσική γλώσσα VHDL. Αυτό γίνεται σημαντικά αντιληπτό σε περιπτώσεις ακόμα πιο πολύπλοκων σχεδιάσεων όπου περιπτώσεις μικρών σφαλμάτων σε ασύμαντα τμήματα κώδικα, ανιχνεύονται και διορθώνονται πολύ πιο εύκολα.
- Το μειονέκτημα της σχεδίασης σε μια τέτοια γλώσσα, μπορεί να θεωρηθεί το γεγονός ότι πρέπει να ληφθούν υπόψη αρκετοί περιορισμοί κατά τη σχεδίαση. Αυτό διότι αν και οι δυνατότητες που παρέχει η systemC είναι απεριόριστες, εντούτοις αρκετές από τις δομές της είναι αδύνατο να μετατραπούν σε RTL επίπεδο για περεταίρω σύνθεση της σχεδίασης.
- Σημαντικό ρόλο για το πιο πάνω παίζουν τα εργαλεία μεταγλώττισης της systemC σχεδίασης σε hardware.
- Ο μεταγλωττιστής που χρησιμοποιήθηκε systemCrafter (SC) έκδοση 2.0, σε γενικές γραμμές είναι ένας ευέλικτος και εύκολος στη χρήση μεταγλωττιστής. Σημαντικότερο του μειονέκτημα αποτελεί το γεγονός της μη υποστήριξης μετατροπής ασύγχρονων κυκλωμάτων σε RTL επίπεδο. Αυτό σε αρκετές περιπτώσεις όπου η χρήση κυρίως πολυπλεκτών είναι επιβεβλημένη, περιπέκει τη διαδικασία σχεδίασης και δένει τα χέρια στο σχεδιαστή.
- Η υποστήριξη μετατροπής και ασύγχρονων κυκλωμάτων σε RTL VHDL σε επόμενη έκδοση του συγκεκριμένου εργαλείου θα λύσει τα χέρια των σχεδιαστών σε αρκετές περιπτώσεις όπου η χρησιμοποίηση ασύγχρονων κυκλωμάτων επιβάλλεται.

6.3 Επίλογος

Συνοψίζοντας, παρουσιάστηκαν τα βασικά χαρακτηριστικά της γλώσσας systemC και πως η τελευταία μπορεί να χρησιμοποιηθεί και για σχεδίαση και υλοποίηση ψηφιακών κυκλωμάτων που να είναι συνεπή και χρονισμένα. Επίσης παρουσιάστηκε η διαδικασία μετατροπής της systemC σχεδίασης σε RTL VHDL, καθώς και η περεταίρω σύνθεση και τοποθέτηση της RTL πλέον σχεδίασης, χρησιμοποιώντας τα εργαλεία σύνθεσης και τοποθέτησης Xilinx ISE 7.1. Τέλος παρουσιάστηκε η διαδικασία φορτώματος της σχεδίασης σε μια SPARTAN-3 xc3S400 FPGA, καθώς και η επιβεβαίωση της λειτουργίας στην FPGA μέσω του λογικού αναλυτή και των διεπαφών χρήστη που παρείχε το αναπτυξιακό της MEMEC στο οποίο ήταν τοποθετημένη η συγκεκριμένη FPGA.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] John L Hennessy, David A. Patterson, Αρχιτεκτονική Υπολογιστών, Τρίτη εκδοση.

[2]http://www.intel.com/design/intarch/pentium4/pentium4.htm?ppc_cid=ggl|ipd_pentium_4|k142C|c

[3] Gelsinger, Gargini, Parker, Yu, "Microprocessors Circa 2000", IEEE spectrum, October 1989, Pages 43-47

[4] SystemC Version2.0 User's quide.

[5] David C.Black, Jack Donovan, " SystemC: From the ground up".

[6] SystemCrafter SC User Manual, version 2.0.0.

[7] John L Hennessy, David A. Patterson, "Computer Organization & Design. The Hardware-Software interface, second edition 1997.

[8] Ηλεκτρονική διεύθυνση http://www.cs.iastate.edu/~prabhu/Tutorial/title.html

[9] Ηλεκτρονική διεύθυνση http://www.cs.cinvestav.mx/SC/prof_personal/adiaz/vhdl/DLX/

[10] Πολυτεχνείο Κρήτης, Σημειώσεις μαθήματος " Οργάνωση Υπολογιστών"

[11] datasheet δίπορτων μνημών από www.xilinx.com.

[12] Memec Spartan-3 LC, User's guide, version 2.0, 2004.

[13] Memec Spartan-3 LC simple LED Reference Design.

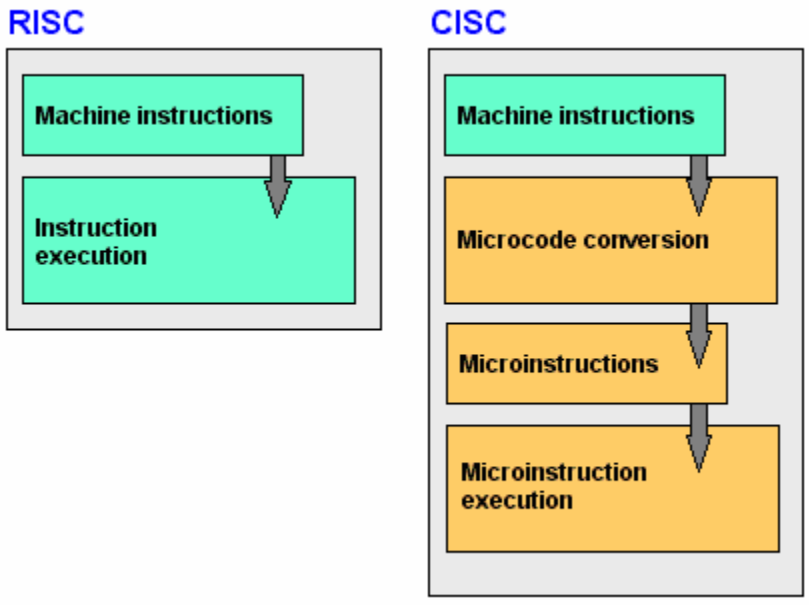
[14] Spartan-3 FPGA Family: Pinout Descriptions

[15] Ηλεκτρονική διεύθυνση http://www.pctechguide.com

ΠΑΡΑΡΤΗΜΑ Α

Ο όρος RISC(**R**educed **I**nstruction **S**et **C**omputer) αναφέρεται στην αρχιτεκτονική υπολογιστών που ελαχιστοποιεί τη πολυπλοκότητα του ολοκληρωμένου (chip) χρησιμοποιώντας πιο απλές εντολές. Οι μεταγλωττιστές RISC πρέπει να παράγουν ρουτίνες λογισμικού έτσι ώστε να εκτελούν πολύπλοκες εντολές που υποστήριζαν αρχιτεκτονικές CISC. Στη τεχνολογία RISC, το επίπεδο μικροκώδικα και τα σχετιζόμενα με αυτό επίπεδα, που ακολουθούν (τμήματα της αρχιτεκτονικής CISC) παραλείπονται. Αυτό φαίνεται σχηματικά στο σχήμα Α-1.

Στην αρχιτεκτονική RISC, το μέγεθος της εντολής διατηρείται σταθερό, απαγορεύεται η έμμεση διευθυσνιοδότηση και διασφαλίζονται μόνο οι εντολές που μπορούν να επικαλυφθούνκαι να εκτελεστούν σε ένα ή λιγότερους κύκλους ρολογιού. Το κύκλωμα της αρχιτεκτονικής RISC είναι γρηγορότερο από το αντίστοιχο της CISC και η σχεδίαση και υλοποίηση του είναι πιο φθηνή.



Σχήμα Α-1

Η μηχανή RISC εκτελεί τις εντολές γρηγορότερα διότι παραλείπει τα επίπεδα μετατροπής του μικροκώδικα. Ο μεταγλωττιστής RISC παράγει περισσότερες εντολές παρά ο μεταγλωττιστής CISC για την ίδια ενέργεια.

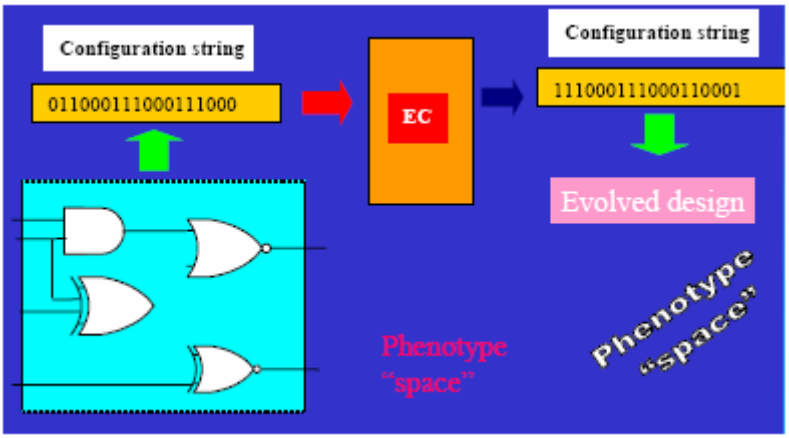
ΠΑΡΑΡΤΗΜΑ Β

Η FPGA (Field Programmable Gate Array), είναι το πιο πρόσφατο είδος προγραμματιζόμενης λογικής συσκευής. Είναι ψηφιακά ολοκληρωμένα κυκλώματα, (ICs) τα οποία περιέχουν προγραμματιζόμενα μπλοκ λογικής, με προγραμματιζόμενες αλληλοσυνδέσεις ανάμεσα σε αυτά τα μπλοκ. Επιτρέπει την υλοποίηση ολοκληρωμένων ψηφιακών ηλεκτρονικών κυκλωμάτων χωρίς να απαιτούνται πολύπλοκες οπτικές, χημικές και μηχανικές διεργασίες που λαμβάνουν χώρα σε βιομηχανίες παραγωγής ολοκληρωμένων. Για το λόγο αυτό υιοθετήθηκε ο όρος «Field Programmable».

Ανάλογα με τον τρόπο υλοποίησης τους, η σύνθεση των μπλοκ σε μια FPGA επιτυγχάνεται με μια αλληλουχία σύνθεσης (string configuration). Υπάρχουν δύο τρόποι σύνθεσης:

- Στατική όπου η σύνθεση γίνεται μια μόνο φορά για να εκτελείται μια συγκεκριμένη εφαρμογή, χωρίς να μπορεί να ξαναπρογραμματιστεί. Τέτοιες FPGAs ονομάζονται one-time programmable (OTP).
- Δυναμική όπου η σύνθεση μπορεί να αλλάξει χειροκίνητα περισσότερες από μια φορά.

Η διαδικασία σύνθεσης σε μια FPGA, περιγράφεται σχηματικά στο σχήμα Β-1.



Σχήμα Β-1
Διαδικασία σύνθεσης σχεδίασης και εφαρμογής της σε FPGA

ΠΑΡΑΡΤΗΜΑ Γ

Η κωδικοποίηση του συνόλου εντολών του DLX φαίνεται στους παρακάτω πίνακες:

IR[31:26]		IR[5:0]		Mnemonic	Effect
Εντολές ολίσθησης					
0	0x00	0	0x00	slli	RD = RS1 << SA
0	0x00	1	0x01	slai	RD = RS1 << SA (arith.)
0	0x00	10	0x02	srli	RD = RS1 >> SA
0	0x00	11	0x03	srai	RD = RS1 >> SA (arith.)
0	0x00	100	0x04	sll	RD = RS1 << RS2[4:0]
0	0x00	101	0x05	sla	RD = RS1 << RS2[4:0] (arith.)
0	0x00	110	0x06	srl	RD = RS1 >> RS2[4:0]
0	0x00	111	0x07	sra	RD = RS1 >> RS2[4:0] (arith.)
Εντολές μεταφοράς δεδομένων					
0	0x00	10000	0x10	movs2i	RD = SA
0	0x00	10001	0x11	movi2s	SA = RS1
Αριθμητικές, λογικές πράξεις					
0	0x00	100000	0x20	add	RD = RS1 + RS2
0	0x00	100001	0x21	addu	RD = RS1 + RS2 (no overflow)
0	0x00	100010	0x22	sub	RD = RS1 - RS2
0	0x00	100011	0x23	subu	RD = RS1 - RS2 (no overflow)
0	0x00	100100	0x24	and	RD = RS1 $\wedge$ RS2
0	0x00	100101	0x25	or	RD = RS1 $\vee$ RS2
0	0x00	100110	0x26	xor	RD = RS1 $\oplus$ RS2
0	0x00	100111	0x27	lhg	RD = RS2[15:0] 0^{16}
Test Set Operation					
0	0x00	101000	0x28	clr	RD = (false ? 1 : 0)
0	0x00	101001	0x29	sgr	RD = (RS1 > RS2 ? 1 : 0)
0	0x00	101010	0x2a	seq	RD = (RS1 = RS2 ? 1 : 0)
0	0x00	101011	0x2b	sge	RD = (RS1 $\geq$ RS2 ? 1 : 0)
0	0x00	101100	0x2c	sls	RD = (RS1 < RS2 ? 1 : 0)
0	0x00	101101	0x2d	sne	RD = (RS1 $\neq$ RS2 ? 1 : 0)
0	0x00	101110	0x2e	sle	RD = (RS1 $\leq$ RS2 ? 1 : 0)
0	0x00	101111	0x2f	set	RD = (true ? 1 : 0)

Πίνακας Γ-1: R-τύπος εντολών

IR[31:26]		Μνημονικό	d	Αποτέλεσμα
Μεταφορά δεδομένων, mem = M[RS1 + Sext(imm)]				
100000	0x20	lb	1	RD = Sext(mem)
100001	0x21	lh	2	RD = Sext(mem)
100011	0x23	lw	4	RD = mem
100100	0x24	lbu	1	RD = 0^{24}mem
100101	0x25	lhu	2	RD = 0^{16}mem
101000	0x28	sb	1	mem = RD[7:0]
101001	0x29	sh	2	mem = RD[15:0]
101011	0x2b	sw	4	mem = RD
Αριθμητικές, λογικές πράξεις				
1000	0x08	addi		RD = RS1 + Sext(imm)
1001	0x09	addiu		RD = RS1 + Sext(imm) (no overflow)
1010	0x10	subi		RD = RS1 - Sext(imm)
1011	0x11	subiu		RD = RS1 - Sext(imm) (no overflow)
1100	0x12	andi		RD = RS1 $\wedge$ Sext(imm)
1101	0x13	ori		RD = RS1 $\vee$ Sext(imm)
1110	0x14	xori		RD = RS1 $\oplus$ Sext(imm)
1111	0x15	lhgi		RD = imm0^{16}
Test Set Operation				
11000	0x18	clri		RD = (false ? 1 : 0)
11001	0x19	sgri		RD = (RS1 > Sext(imm) ? 1 : 0)
11010	0x1a	seqi		RD = (RS1 = Sext(imm) ? 1 : 0)
11011	0x1b	sgei		RD = (RS1 $\geq$ Sext(imm) ? 1 : 0)
11100	0x1c	slsi		RD = (RS1 < Sext(imm) ? 1 : 0)
11101	0x1d	snei		RD = (RS1 $\neq$ Sext(imm) ? 1 : 0)
11110	0x1e	slei		RD = (RS1 $\leq$ Sext(imm) ? 1 : 0)
11111	0x1f	seti		RD = (true ? 1 : 0)
Εντολές μεταφοράς ελέγχου				
100	0x04	beqz		PC = PC + 4 + (RS1 = 0 ? Sext(imm) : 0)
101	0x05	bnez		PC = PC + 4 + (RS1 $\neq$ 0 ? Sext(imm) : 0)
110	0x16	jr		PC = RS1
111	0x17	jalr		R31 = PC + 4; PC = RS1

Πίνακας Γ-2: I-τύπος εντολών

IR[31:26]		Μνημονικό	Αποτέλεσμα
Εντολές μεταφοράς ελέγχου			
10	0x02	j	PC = PC + 4 + Sext(imm)
11	0x03	jal	R31 = PC + 4; PC = PC + 4 + Sext(imm)
111110	0x3e	trap	trap = 1; EPC = PC; PC = SISR;
			ESR = SR; ECA = masked CA; SR = 0;
			EDATA = Sext(imm);
			clear CA but catch new interrupt events
111111	0x3f	rfe	SR = ESR; PC = EPC

Πίνακας Γ-3: J-τύπος εντολών

IR[31:26]		Μνημονικό	d	Αποτέλεσμα
Load, Store				
110001	0x31	load.s	4	FD[31:0] = mem
110101	0x35	load.d	8	FD[63:0] = mem
111001	0x39	store.s	4	m = FD[31:0]
111101	0x3d	store.d	8	m = FD[63:0]
Εντολές ελέγχου				
110	0x06	fbeqz		PC = PC + 4 + (FCC = 0 ? Sext(imm) : 0)
111	0x07	fbnez		PC = PC + 4 + (FCC $\neq$ 0 ? Sext(imm) : 0)

Πίνακας Γ-3: FI-τύπος εντολών

RM	Σύμβολο	Rounding Mode	Bit	Σύμβολο	Σκοπός
0	RZ	toward zero	0	OVF	overflow
1	RNE	to next even	1	UNF	underflow
10	RPI	toward +∞	2	INX	inexact result
11	RMI	toward -∞	3	DBZ	divide by zero
			4	INV	invalid operation

Πίνακας Γ-4: κωδικοποίηση του τρόπου περιστροφής RM και οι μάσκες διακοπών IEEEf

IR[31:26]		IR[5:0]		Fmt	Mnemonic	Effect
Arithmetic and Compare Operations						
10001	0x11	0	0x00		fadd [.s, .d]	FD = FS1 + FS2
10001	0x11	1	0x01		fsub [.s, .d]	FD = FS1 - FS2
10001	0x11	10	0x02		fmul [.s, .d]	FD = FS1 * FS2
10001	0x11	11	0x03		fdiv [.s, .d]	FD = FS1 / FS2
10001	0x11	100	0x04		fneg [.s, .d]	FD = - FS1
10001	0x11	101	0x05		fabs [.s, .d]	FD = abs(FS1)
10001	0x11	110	0x06		fsqt [.s, .d]	FD = sqrt(FS1)
10001	0x11	111	0x07		frem [.s, .d]	FD = rem(FS1, FS2)
10001	0x11	$11c_3\ c_2\ c_1\ c_0$			fc.cond [.s, .d]	FCC = (FS1 *cond* FS2)
Data Transfer						
10001	0x11	1000	0x08	0	fmov.s	FD[31:0] = FS1[31:0]
10001	0x11	1000	0x08	1	fmov.d	FD[63:0] = FS1[63:0]
10001	0x11	1001	0x09		mf2i	RS = FS1[31:0]
10001	0x11	1010	0x0a		mi2f	FD[31:0] = RS
Conversion						
10001	0x11	100000	0x20	1	cvt.s.d	FD = cvt(FS1, s, d)
10001	0x11	100000	0x20	100	cvt.s.i	FD = cvt(FS1, s, i)
10001	0x11	100001	0x21	0	cvt.d.s	FD = cvt(FS1, d, s)
10001	0x11	100001	0x21	100	cvt.d.i	FD = cvt(FS1, d, i)
10001	0x11	100100	0x24	0	cvt.i.s	FD = cvt(FS1, i, s)
10001	0x11	100100	0x24	1	cvt.i.d	FD = cvt(FS1, i, d)

Πίνακας Γ-5: FR-type instruction layout. Fmt=IR[8:6]