ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τίτλος

Υλοποίηση Χρονοδρομολογητή με Υλικό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΝΤΩΝΗ-ΠΑΝΑΓΙΩΤΗ Ι. ΜΑΝΔΗΛΑ

Επιβλέπων : Γεώργιος Παπακωνσταντίνου

Αθήνα, Ιούλιος 2007

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τίτλος
Υλοποίηση Χρονοδρομολογητή με Υλικό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΝΤΩΝΗ-ΠΑΝΑΓΙΩΤΗ Ι. ΜΑΝΔΗΛΑ

Επιβλέπων : Γεώργιος Παπακωνσταντίνου

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 24/7/2007

.....	.....	.....
Γεώργιος Παπακωνσταντίνου	Παναγιώτης Τσανάκας	Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2007

.....

ΑΝΤΩΝΗΣ-ΠΑΝΑΓΙΩΤΗΣ Ι. ΜΑΝΔΗΛΑΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Αντώνης – Παναγιώτης Ι. Μάνδηλας, 2007
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν
τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του

Εθνικού Μετσόβιου Πολυτεχνείου.

© 2007 – All rights reserved

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Επιβλέπων καθηγητής είναι ο κ. Γιώργος Παπακωνσταντίνου τον οποίο ευχαριστώ για την εμπιστοσύνη που μου έδειξε κατά τη διάρκεια της εκπόνησης της παρούσας εργασίας. Αισθάνομαι τυχερός που γνώρισα και συνεργάστηκα με τους διδάκτορες Στέργιο Στεργίου και Δώρα Σουλίου, οι οποίοι με βοήθησαν για την ολοκλήρωση της παρούσας διπλωματικής και τους ευχαριστώ για τη βοήθειά τους.

Πως θα μπορούσα να μην ευχαριστήσω τους γονείς μου για τη μέχρι τώρα πορεία μου; Το λιγότερο που θα μπορούσα να πω, για την ώρα, είναι απλά ότι προσπαθώ να συνεχίσω το έργο τους, διατηρώντας συγχρόνως μία απλή νοοτροπία ζωής, η οποία χαρακτηρίζεται από εργατικότητα και αναζήτηση και δίψα για μάθηση.

Ιούλιος 2007

Μάνδηλας Αντώνης - Παναγιώτης

Περίληψη στα ελληνικά

Σκοπός της παρούσας διπλωματικής εργασίας είναι η μετατροπή της αρχιτεκτονικής ενός επεξεργαστή τεχνολογίας RISC, έτσι ώστε να είναι εφικτή η εναλλαγή των ενεργών διεργασιών με βάση προτεραιότητες που δίνει ο χρήστης σε κάθε διεργασία. Ο επεξεργαστής μετά τις αλλαγές θα μπορεί να εκτελεί τις διεργασίες σε επίπεδο πραγματικού χρόνου (Real Time). Ο επεξεργαστής του οποίου αλλάζεται η αρχιτεκτονική ανήκει στην κατηγορία των “Soft-Core” επεξεργαστών. Οι επεξεργαστές αυτής της κατηγορίας είναι γραμμένοι σε κάποια γλώσσα περιγραφής υλικού, στην περίπτωσή μας Verilog.

Η ενεργή διεργασία εξορισμού δίνει τη θέση της σε μία με υψηλότερη προτεραιότητα, όταν αυτή ζητήσει την επεξεργαστική μονάδα. Μια σημαντική αλλαγή που έχει γίνει στον επεξεργαστή είναι ότι η κάθε διεργασία που εκτελείται έχει το δικό της σύνολο καταχωρητών σε επίπεδο υλικού. Αυτό έχει ως αποτέλεσμα να επιτυγχάνεται πολύ μικρός χρόνος αλλαγής μεταξύ των ενεργών διεργασιών.

Επίσης έχει γίνει προσπάθεια να αυτοματοποιηθεί η λειτουργία του συνόλου επεξεργαστή – εφαρμογής. Έτσι με μία εφαρμογή γραμμένη σε C, γίνεται επεξεργασία του κώδικα που έχει γράψει ο χρήστης και θέλει να εκτελεστεί στον επεξεργαστή, αναγνωρίζονται οι διευθύνσεις έναρξης και επιστροφής των διεργασιών και αλλάζονται τα αντίστοιχα πεδία στα αρχεία Verilog που απαρτίζουν τον επεξεργαστή. Στη συνέχεια ο χρήστης πρέπει απλά να συνθέσει εκ νέου το μοντέλο του επεξεργαστή και να το φορτώσει στην FPGA.

Λέξεις Κλειδιά: << FPGA, RISC, Soft-Core, Verilog, Real Time, Επεξεργαστής, Καταχωρητής, Προτεραιότητα, Σύνθεση, Χρονοδρομολόγηση>>

Summary in English

The main aim of this diploma thesis is the upgrade of the Central Processor Unit of a RISC processor. This upgrade is performed so that it will be able to change between the running processes according to user specified priorities for each one of them. The processor after the upgrade will be able to run the processes in a near real time environment. The processor whose architecture is to be changed belongs to Soft-Core family of processors. These processors are usually coded in a Hardware Description language, in our case this language is Verilog.

According to the specifications the running process will always allow a higher priority process to gain the access to the CPU unit. Another major change that has been done to the architecture is that each process has its own register set. By this way the time needed to change between running processes is much less than in ordinary CPUs.

Also effort has been done so that user can automatically make the upgrade procedure according to the specifications for the processes to be executed, for this purpose a small utility has been developed using C language. With this utility the user application source code is being parsed and the start – return addresses of each process are located. Afterwards the Verilog source code files are then upgraded so that the user needs only to resynthesize the processor model and load it to the FPGA.

Keywords: <<FPGA, RISC, Soft-Core, Verilog, Real Time, Processor, Register, Priority, Synthesis, Time scheduling >>

Πίνακας περιεχομένων

1 Εισαγωγή 19

1.1 Αντικείμενο της διπλωματικής 19

1.2 Οργάνωση του τόμου 20

2 Verilog 23

2.1 Εισαγωγή στη Verilog 23

2.2 Ανταγωνισμός και ιστορία 24

2.3 Δυνατότητες 24

2.4 Αναγκαιότητα χρήσης ΓΠΥ 25

2.5 Δομή γλώσσας και λεξικοί κανόνες 25

2.6 Τρόπος περιγραφής ενός ψηφιακού συστήματος 27

2.6.1 Οντότητες 27

2.6.2 Δομή οντότητας 28

2.6.3 Αποθήκευση και μεταφορά δεδομένων 28

2.6.4 Διαφορές οντοτήτων με συναρτήσεις C 29

2.6.5 Δημιουργία «εικόνας» μίας οντότητας 29

2.7 Τελεστές πράξεων 30

2.7.1 Αριθμητικοί τελεστές 31

2.7.2 Συγκρισιακοί τελεστές 31

2.7.3 Λογικοί τελεστές 31

2.7.4 Τελεστές bitwise 32

2.7.5 Τελεστές μοναδιαίας μείωσης (Unary reduction) 32

2.7.6 Προτεραιότητα τελεστών 33

2.8 Δομές ελέγχου 33

2.9 Δομές επανάληψης 34

2.10 Μεταφορές δεδομένων Blocking και non-Blocking 35

2.11 Παραδείγματα προγραμμάτων Verilog 35

2.11.1 Πολυπλέκτης 2 σε 1 35

2.11.2 Αποκωδικοποιητής προτεραιοτήτων 38

2.11.3 Έλεγχος διαύλου δεδομένων ... *39*
2.11.4 Συγκριτής σημάτων ... *40*
2.11.5 Flip – Flops ... *41*
2.11.6 Flip – Flops με δυνατότητα αρχικοποίησης (reset) ... *42*
2.11.7 Οδηγοί τριών καταστάσεων (Tri-State Drivers) ... *44*
2.11.8 Μετρητές ... *47*
2.11.9 Καταχωρητής ολίσθησης με ενεργοποίηση ... *48*
3 XR16 ... 51
3.1 Επεξεργαστές soft-core ... 51
3.2 XR16 ... 52
3.3 Compiler ... 52
3.4 Καταχωρητές ... 52
3.5 Σύνολο εντολών ... 53
3.6 Αρχιτεκτονική ... 55
3.6.1 Αρχείο καταχωρητών ... *56*
3.6.2 Δεσμευμένος χώρος στην FPGA ... *57*
3.6.3 Ανάγνωση και διάβασμα στους καταχωρητές ... *57*
3.6.4 Επιλογή τελεστών ... *58*
3.6.5 Μονάδα αριθμητικής και λογικής (ALU) ... *58*
3.6.6 Καταχωρητές ολίσθησης ... *59*
3.6.7 Πολυπλέκτης επιλογής αποτελέσματος ... *59*
3.6.8 Μονάδα διεύθυνσης και μετρητή προγράμματος ... *60*
3.7 Λειτουργία επεξεργαστή ... 60
3.7.1 Pipeline του XR16 ... *61*
3.7.2 Προβλήματα λειτουργίας pipeline ... *62*
3.7.3 Πρόβλημα διάδοσης αποτελέσματος ... *62*
3.8 Μονάδα ελέγχου ... 65
3.8.1 Εκτέλεση εντολής ... *66*
4 Αλγόριθμοι χρονοδρομολόγησης ... 67
4.1 Περιγραφή ... 67
4.2 Ιδιότητες εφαρμογών - Θεωρία ... 67

4.3 Πραγματικότητα ... 68
4.4 Εφαρμογές ελέγχου και παρακολούθησης ... 68
4.5 Ορισμός ... 69
4.6 Λειτουργικά συστήματα πραγματικού χρόνου ... 69
4.7 Αλγόριθμος με σταθερή απόδοση προτεραιοτήτων ... 70
4.7.1 Περιβάλλον ανάπτυξης ... 70
4.7.2 Περιγραφή διεργασίας ... 71
4.7.3 Ανάλυση αλγορίθμου ... 72
4.7.4 Μέγιστη χρήση επεξεργαστικής μονάδας ... 75
5 Αρχιτεκτονική συστήματος ... 83
5.1 Εισαγωγή στην αρχιτεκτονική του συστήματος ... 83
5.2 Πολλές διεργασίες στον XR16 ... 84
5.2.1 Εισαγωγή στην εκτέλεση πολλών διεργασιών ... 84
5.2.2 Μια διαφορετική προσέγγιση ... 85
5.2.3 Η προσέγγιση που έγινε στο XR16 ... 86
5.2.4 Εισαγωγή προτεραιοτήτων στο σύστημα ... 97
5.2.5 Νέες λειτουργίες του συστήματος ... 97
5.2.6 Νέα αρχιτεκτονική επεξεργαστή ... 104
5.2.7 Ανάλυση νέου τρόπου λειτουργίας επεξεργαστή ... 104
5.2.8 Αλλαγές στη λειτουργία των διεργασιών ... 105
5.2.9 Αλλαγές στο θεωρητικό μοντέλο ... 106
6 Εγκατάσταση απαραίτητων προγραμμάτων ... 107
6.1 Εισαγωγή ... 107
6.2 Πηγαίος κώδικας επεξεργαστή ... 107
6.3 LCC Compiler ... 108
6.3.1 Διαδικασία δημιουργίας προγράμματος ... 109
6.4 Xilinx ISE ... 110
6.5 ModelSim XE ... 111
7 Αυτοματοποίηση διαδικασίας ... 113
7.1 Εισαγωγή ... 113
7.2 Διαδικασία που ακολουθεί ο χρήστης ... 114

7.3 Κατά βήμα χρήση του προγράμματος 114
7.3.1 Πρώτο μενού επιλογών 115
7.3.2 Επεξεργασία προγράμματος προς εκτέλεση 115
7.3.3 Ενημέρωση αρχείων Verilog 117
8 Έλεγχος 119
8.1 Εγκατάσταση απαραίτητων προγραμμάτων 119
8.2 Μεθοδολογία Ελέγχου 119
8.3 Αναλυτική παρουσίαση του ελέγχου 120
8.3.1 Πρώτο πρόγραμμα – Μία διεργασία 120
8.3.2 Δεύτερο πρόγραμμα – Δύο διεργασίες 122
8.3.3 Τρίτο πρόγραμμα – Τέσσερις διεργασίες 125
9 Επίλογος 131
9.1 Σύνοψη και συμπεράσματα 131
10 Βιβλιογραφία 133

Πίνακας πινάκων

Πίνακας 1 Σχόλια στη Verilog 26
Πίνακας 2 Παραδείγματα αριθμών 26
Πίνακας 3 Παραδείγματα δήλωσης μεταβλητών 27
Πίνακας 4 Πιθανές τιμές καταχωρητών 29
Πίνακας 5 Αριθμητικοί τελεστές 31
Πίνακας 6 Συγκρισιακοί τελεστές 31
Πίνακας 7 Λογικοί τελεστές 31
Πίνακας 8 Τελεστές bitwise 32
Πίνακας 9 Τελεστές μοναδιαίας μείωσης 32
Πίνακας 10 Προτεραιότητα τελεστών 33
Πίνακας 11 Καταχωρητές XR16 53
Πίνακας 12 Σχήματα εντολών XR16 54
Πίνακας 13 Δομή υλοποιημένων εντολών 55

Πίνακας εικόνων

Εικόνα 1 Πολυπλέκτης 2 σε 1 35
Εικόνα 2 Αποκωδικοποιητής προτεραιοτήτων 38
Εικόνα 3 Ελεγκτής διαύλου 4bit 39
Εικόνα 4 Συγκριτής σημάτων 40
Εικόνα 5 Θετικά ακμοπυροδοτούμενο D - Flip Flop 41
Εικόνα 6 Αρνητικά ακμοπυροδοτούμενο D Flip Flop 42
Εικόνα 7 D Flip-Flop με συγχρονισμένη αρχικοποίηση 42
Εικόνα 8 D Flip Flop με ασύγχρονη αρχικοποίηση 43
Εικόνα 9 D Flip Flop με ασύγχρονη αρχικοποίηση (Active Low reset) 44
Εικόνα 10 Οδηγός τριών καταστάσεων 45
Εικόνα 11 Λίστα πρωταρχικών στοιχείων Verilog 46
Εικόνα 12 Μετρητής τριών bit, με ασύγχρονη αρχικοποίηση 47
Εικόνα 13 Καταχωρητής ολίσθησης με ενεργοποίηση 48
Εικόνα 14 Διαδικασία εκτέλεσης πολλών διεργασιών 84
Εικόνα 15 Αρχιτεκτονική XR16 87
Εικόνα 18 Μετρητής επιλογής σετ καταχωρητών 91
Εικόνα 19 Κύκλωμα διαιρέτη συχνότητας 91
Εικόνα 20 Νέα αρχιτεκτονική XR16 92
Εικόνα 22 Τμήμα του κυκλώματος επιλογής διεργασίας 99
Εικόνα 23 Καταχωρητής τριών θέσεων για το παράδειγμα με τις τρεις διεργασίες 100
Εικόνα 24 Εκτέλεση διεργασιών σε περιβάλλον πραγματικού χρόνου 101
Εικόνα 26 Νέα αρχιτεκτονική επεξεργαστή 104
Εικόνα 27 Πρώτο μενού επιλογών 115
Εικόνα 28 Μενού επεξεργασίας προγράμματος C 115
Εικόνα 29 Εισαγωγή διεύθυνσης αρχείου περιγραφής συστήματος 117
Εικόνα 30 Αποδοχή ανοιγμένου αρχείου περιγραφής ή επιλογή νέου 118
Εικόνα 31 Εισαγωγή διεύθυνσης αρχείων επεξεργαστή 118

1

Εισαγωγή

Στο πρώτο κεφάλαιο της διπλωματικής εργασίας, γίνεται μια επιγραμματική αναφορά στο σκοπό της εργασίας. Επίσης δίνονται οι επεξηγήσεις ορισμένων όρων που χρησιμοποιούνται στη συνέχεια. Τέλος δίνεται μία σύντομη περιγραφή κάθε κεφαλαίου της παρούσας εργασίας.

1.1 Αντικείμενο της διπλωματικής

Το αντικείμενο της διπλωματικής εργασίας είναι η αλλαγή της αρχιτεκτονικής ενός επεξεργαστή RISC. Ο επεξεργαστής αυτός δεν είναι συνηθισμένος, αλλά ανήκει στην κατηγορία των soft-core επεξεργαστών. Με την έννοια soft-core περιγράφονται οι επεξεργαστές που είναι γραμμένοι σε κάποια γλώσσα περιγραφής υλικού.

Οι αλλαγές που θα γίνουν στον επεξεργαστή θα δώσουν τη δυνατότητα να εκτελούνται ταυτόχρονα πολλές διεργασίες σε αυτόν χωρίς τη μεσολάβηση του

λειτουργικού συστήματος για την αλλαγή μεταξύ τους, η δεύτερη τροποποίηση στην αρχιτεκτονική του θα δώσει τη δυνατότητα να εκτελούνται διεργασίες σε ένα περιβάλλον πραγματικού χρόνου.

Η έννοια «πραγματικός χρόνος» οδηγεί σε ένα περιβάλλον όπου κάθε διεργασία προς εκτέλεση έχει μία τιμή προτεραιότητας και κάθε διεργασία με μεγαλύτερη προτεραιότητα χρησιμοποιεί την επεξεργαστική μονάδα έναντι οποιασδήποτε άλλης.

Η ιδιότητα προτεραιότητα για μία διεργασία δείχνει πόσο σημαντική είναι έναντι των υπολοίπων, έτσι μία διεργασία με υψηλότερη προτεραιότητα θεωρείται πιο σημαντικό να εκτελεστεί έναντι όλων των υπολοίπων με χαμηλότερη τιμή προτεραιότητας.

1.2 Οργάνωση του τόμου

Στα κεφάλαια τρία και τέσσερα του τόμου γίνεται αναφορά στη γλώσσα προγραμματισμού που έχει χρησιμοποιηθεί για την ανάπτυξη του επεξεργαστή και στη θεωρία πάνω στην οποία βασίστηκε το βασικό κομμάτι της εργασίας. Στο πέμπτο κεφάλαιο αναφέρονται αναλυτικά όλες οι αλλαγές που έχουν γίνει στον επεξεργαστή. Στο έκτο κεφάλαιο γίνεται αναφορά στα προγράμματα που χρειάστηκε να εγκατασταθούν για να γίνουν οι τροποποιήσεις στον επεξεργαστή και να ελεγχθεί η ορθή λειτουργία του. Στο έβδομο κεφάλαιο περιγράφεται μία εφαρμογή που αναπτύχθηκε για την αυτοματοποίηση των αλλαγών που πρέπει να γίνονται στον πυρήνα όταν αλλάζει το προς εκτέλεση πρόγραμμα. Στο όγδοο κεφάλαιο δίνονται παραδείγματα ελέγχου ορθής λειτουργίας του επεξεργαστή καθώς επίσης και τα

αποτελέσματα από την προσομοίωση με τις αλλαγές που έχουν γίνει. Αναλυτικά σε κάθε κεφάλαιο πραγματεύονται τα παρακάτω.

Στο δεύτερο κεφάλαιο του τόμου γίνεται αναφορά στη Γλώσσα Περιγραφής Υλικού Verilog. Μια γλώσσα προγραμματισμού αυτού του είδους μπορεί να περιγράψει ένα ψηφιακό σύστημα. Στο κεφάλαιο αυτό δίνονται τα βασικά χαρακτηριστικά της γλώσσας, τις εντολές που υποστηρίζει καθώς και κάποια ιδιαίτερα χαρακτηριστικά της που είναι καλό κάποιος να γνωρίζει.

Στο τρίτο κεφάλαιο αναλύεται ο επεξεργαστή XR16. Δίνεται αναλυτική περιγραφή της αρχιτεκτονικής του. Επίσης παρουσιάζονται οι εντολές που υποστηρίζει και τον τρόπο με τον οποίο εκτελεί κάθε μία από αυτές. Ακόμη περιγράφονται τα περισσότερα από τα στοιχεία που τον απαρτίζουν και κάνουν εφικτή τη λειτουργία του.

Στο τέταρτο κεφάλαιο γίνεται αναφορά στη θεωρία που διέπει όλα τα συστήματα πραγματικού χρόνου. Αρχικά περιγράφεται ένα περιβάλλον μέσα στο οποίο μπορεί να περιγραφτεί ένα σύστημα πραγματικού χρόνου και να μελετηθούν τα χαρακτηριστικά του. Εισάγεται ο όρος «προτεραιότητα» διεργασίας και ιδιαίτερα περιγράφονται τα συστήματα με σταθερή ανάθεση προτεραιοτήτων στις διεργασίες που εκτελούνται. Με αυτό τον τρόπο καταλήγουμε σε ορισμένα θεωρήματα, πάνω στα οποία μπορεί να στηριχθεί η λειτουργία ενός πραγματικού συστήματος. Επιπλέον φτάνουμε σε ορισμένες τιμές για την μέγιστη χρησιμοποίηση ενός επεξεργαστή, στον οποίο εκτελείται ένα λειτουργικό σύστημα πραγματικού χρόνου.

Στο πέμπτο κεφάλαιο περιγράφονται αναλυτικά όλες οι αλλαγές που έχουν γίνει στον επεξεργαστή. Οι αλλαγές θα γίνουν σε δύο στάδια, στο πρώτο εισάγονται πολλαπλά σύνολα καταχωρητών, ένα για κάθε διεργασία προς εκτέλεση. Στο δεύτερο στάδιο εισάγονται προτεραιότητες σε όλες τις διεργασίες έτσι ώστε να εκτελούνται σε ένα αυστηρό περιβάλλον πραγματικού χρόνου.

Στο έκτο κεφάλαιο γίνεται αναφορά σε όλες τις εφαρμογές που χρειάζονται να εγκατασταθούν, έτσι ώστε να γίνεται εφικτή η ανάπτυξη και τροποποίηση του επεξεργαστή καθώς και η προσομοίωση του. Επίσης γίνεται αναφορά στις εφαρμογές που χρειάζονται για να μπορέσει κάποιος να αναπτύξει μία εφαρμογή για τον επεξεργαστή.

Στο έβδομο κεφάλαιο περιγράφεται μία εφαρμογή, η οποία μπορεί να αυτοματοποιήσει τις αλλαγές που χρειάζεται να γίνονται στο επεξεργαστή όταν αλλάζει η προς εκτέλεση εφαρμογή.

Στο όγδοο κεφάλαιο του τόμου δίνονται παραδείγματα χρησιμοποίησης του επεξεργαστή. Χρησιμοποιούνται διαφορετικές εφαρμογές με διαφορετικό αριθμό διεργασιών. Συγκρίνονται οι τρόποι λειτουργίας τους αλλά και ο χώρος που καταλαμβάνουν στην FPGA, όταν αλλάζει ο αριθμός των διεργασιών.

2

Verilog

2.1 Εισαγωγή στη Verilog

Η Verilog είναι μία Γλώσσα Περιγραφής Υλικού (Hardware Description Language), εφεξής ΓΠΥ. Μία γλώσσα προγραμματισμού αυτού του είδους χρησιμοποιείται για την περιγραφή ενός ψηφιακού συστήματος. Ο σχεδιαστής μπορεί να περιγράψει το σύστημα σε διαφορετικά επίπεδα σχεδίασης, από το πιο χαμηλό έως το πιο αφηρημένο. Το πιο χαμηλό είναι το επίπεδο των αντιστάσεων και των τρανζίστορ, το επονομαζόμενο “switch level”. Το αμέσως ανώτερο επίπεδο ονομάζεται “gate level”, σε αυτό μπορούν να περιγράφονται οι πύλες και τα flip-flop, τέλος στο πιο αφηρημένο επίπεδο, ο σχεδιαστής μπορεί να περιγράψει τους τρόπους διασύνδεσης των εξαρτημάτων και πως μεταφέρονται τα δεδομένα, το επίπεδο αυτό ονομάζεται “Register Transfer Level”.

2.2 *Ανταγωνισμός και ιστορία*

Η Verilog είναι μία από τις ΓΠΥ που χρησιμοποιούνται σήμερα τόσο σε βιομηχανικό όσο και σε ακαδημαϊκό επίπεδο, μία άλλη επίσης πολύ διαδεδομένη είναι η VHDL. Η VHDL είναι αναγνωρισμένη από τον οργανισμό IEEE από το 1987, ενώ η Verilog επισημοποιήθηκε το 1995. Η Verilog συντακτικά μοιάζει πολύ με τη γλώσσα προγραμματισμού υψηλού επιπέδου C, γι' αυτό το λόγο προτιμάται από τους Ηλεκτρολόγους Μηχανικούς και τους Μηχανικούς Υπολογιστών. Αντίθετα η VHDL μοιάζει πολύ με την ADA.

Η ΓΠΥ Verilog εμφανίστηκε πρώτη φορά το 1985 από της εταιρεία Gateway Design System Corporation, που τώρα αποτελεί τμήμα της Cadence Design System.

2.3 *Δυνατότητες*

Η Verilog δίνει στο σχεδιαστή υλικού τη δυνατότητα να περιγράψει το σύστημα του σε υψηλό επίπεδο αφαίρεσης, όπως το αρχιτεκτονικό επίπεδο ή εκείνο της επικοινωνίας μεταξύ των εξαρτημάτων, αλλά παράλληλα του δίνει τη δυνατότητα να «κατέβει» στο επίπεδο των πυλών. Ένας από τους σημαντικότερους λόγους για να σχεδιάσει κανείς ένα σύστημα με μία ΓΠΥ, όπως η Verilog, είναι η δυνατότητα προσομοίωσης του και βελτίωσης των επιμέρους χαρακτηριστικών του, πριν προχωρήσει στην υλοποίηση του συστήματος σε μορφή ολοκληρωμένου κυκλώματος (IC[1]).

[1] IC : Integrated Circuit

2.4 *Αναγκαιότητα χρήσης ΓΠΥ*

Η πολυπλοκότητα των ψηφιακών συστημάτων σήμερα έχει αυξηθεί σε μεγάλο βαθμό. Στο χαμηλότερο τους επίπεδο αποτελούνται από εκατομμύρια στοιχεία, όπως αντιστάσεις, πυκνωτές και τρανζίστορ. Αποτέλεσμα του παραπάνω είναι ο σχεδιασμός σε επίπεδο πυλών να μη χρησιμοποιείται πλέον. Για πολλές δεκαετίες τα σχέδια σε επίπεδο πυλών χρησιμοποιούνταν ευρέως, ωστόσο σήμερα όχι. Αν κάποιος προσπαθήσει να δει ένα ολοκληρωμένο κύκλωμα σε επίπεδο πυλών θα παρατηρήσει ένα αχανές και ακαταλαβίστικο δίκτυο από πύλες, αντιστάσεις και διασυνδέσεις. Ήδη από το 1970, για τον προαναφερθέντα λόγο, οι σχεδιαστές μεταπήδησαν προς τις ΓΠΥ για το σχεδιασμό των συστημάτων τους. Σήμερα πάνω από 10000 σχεδιαστές σε επιχειρήσεις, όπου παράγονται ολοκληρωμένα κυκλώματα, όπως η Motorola, η Apple και η Sun, χρησιμοποιούν τη Verilog.

Ο σχεδιαστής με τη χρήση της Verilog έχει τη δυνατότητα να περιγράψει το κύκλωμα, σε επίπεδο αρχιτεκτονικής με λιγότερο κόπο, αφήνοντας τις λεπτομέρειες της υλοποίησης σε επόμενο στάδιο της παραγωγικής διαδικασίας. Μέσω προσομοιώσεων μπορεί να βρει τα αδύναμα σημεία της κατασκευής, αυτά που προκαλούν καθυστερήσεις ή δημιουργούν επιπλέον άχρηστο υλικό, και να τα διορθώσει πριν φτάσει στο σημείο να συνθέσει το ολοκληρωμένο κύκλωμα.

2.5 *Δομή γλώσσας και λεξικοί κανόνες*

Η Verilog μοιάζει με τη γλώσσα υψηλού επιπέδου C. Για την περιγραφή ενός συστήματος δημιουργούνται οντότητες, που ονομάζονται “modules”, αυτές αντιστοιχούν στις συναρτήσεις που δημιουργούνται για την επίλυση ενός

προβλήματος στη C. Αυτά τα “modules” συνδέονται μεταξύ τους σε ένα ανώτερο επίπεδο του προγράμματος, όπως θα «συνδέονταν» στη C τα αποτελέσματα των συναρτήσεων για να λυθεί ένα πρόβλημα.

Οι λεξικοί κανόνες στη Verilog μοιάζουν πολύ με αυτούς της C, για τα σχόλια χρησιμοποιείται το // ή για μία ομάδα σχολίων τα /* και */.

// Αυτό είναι ένα απλό σχόλιο
/* Αυτή είναι μία ομάδα σχολίων */

Πίνακας 1 Σχόλια στη Verilog

Η γλώσσα είναι ευαίσθητη στο αν τα γράμματα είναι κεφαλαία ή μικρά (case sensitive). Οι αριθμοί γράφονται με τη συνηθισμένη μορφή της αλληλουχίας ψηφίων ή με την παρακάτω μορφή

<πρόσημο><μέγεθος><βάση αριθμού><αριθμός>

όπου το «μέγεθος» δίνει τον αριθμό των ψηφίων που αποτελούν τον αριθμό, η «βάση αριθμού» είναι ένας χαρακτήρας “’” ακολουθούμενος από κάποιο γράμμα από τα b,d,o,h. Τα γράμματα αυτά αντιστοιχούν σε **b**inary ή δυαδικός αριθμός, **d**ecimal ή δεκαδικός αριθμός, **o**ctal ή οκταδικός αριθμός και, τέλος, (**h)**ex ή δεκαεξαδικός αριθμός.

Τύπος αριθμού	**Τιμή**
Δεκαδικός	9 ή ’d9
Δεκαεξαδικός	3’h189
Δυαδικός	2’b1Z
Οκταδικός	2’o17

Πίνακας 2 Παραδείγματα αριθμών

Επίσης για τις ανάγκες προγραμματισμού στη Verilog είναι δυνατόν να οριστούν μεταβλητές δεδομένων ή και πίνακες, όπως σε μία γλώσσα υψηλού επιπέδου. Οι μεταβλητές που ορίζονται εξ' ορισμού στη Verilog είναι για την περιγραφή τόσο ακέραιων όσο και πραγματικών αριθμών, έτσι ορίζονται μεταβλητές τύπου *integer* και μεταβλητές τύπου *real*.

real a, b, c	// οι a, b, c είναι μεταβλητές πραγματικών αριθμών
integer j, k	// απλή μεταβλητή ακεραίου
integer i[1:32]	// πίνακας μεταβλητών ακέραιων αριθμών

Πίνακας 3 Παραδείγματα δήλωσης μεταβλητών

2.6 Τρόπος περιγραφής ενός ψηφιακού συστήματος

Η Verilog περιγράφει ένα ψηφιακό σύστημα ως ένα σύνολο από οντότητες, "modules", κάθε ένα από τα οποία έχει μία διεπαφή (interface) για να μπορεί να συνδεθεί με τις υπόλοιπες οντότητες του συστήματος.

2.6.1 Οντότητες

Οι οντότητες, "modules", περιγράφουν τμήματα του υλικού, η πολυπλοκότητα των οποίων ξεκινά από μερικές πύλες και μπορεί να φτάσει έως και ένα ολόκληρο σύστημα, όπως ένας επεξεργαστής. Μια τέτοια οντότητα μπορεί να περιγραφθεί με βάση τη συμπεριφορά της, τη θέση της στην ιεραρχία του συστήματος σε σχέση με τις υπόλοιπες ή και με τους δύο τρόπους. Μια περιγραφή με βάση τη συμπεριφορά γίνεται χρησιμοποιώντας τη γλώσσα προγραμματισμού και δομές *if-else*, *select-case* και άλλες αντίστοιχες. Μία περιγραφή με βάση την ιεραρχική της θέση, μπορεί να ελεγχθεί η συμπεριφορά ενός ολόκληρου συστήματος και πως αυτό δουλεύει όταν συνδεθούν όλα τα επιμέρους τμήματα του.

2.6.2 Δομή οντότητας

Η βασική μορφή μίας οντότητας στη Verilog είναι η παρακάτω

```
module <όνομα> (<λίστα θυρών>)
        <declares>
        <module items>
end module
```

Όπου το **<όνομα>** είναι ένα μοναδικό αναγνωριστικό για την οντότητα που περιγράφεται. Η **<λίστα θυρών>,** που μπορεί και να μην υπάρχει, είναι μια λίστα με τις εισόδους και τις εξόδους της οντότητας για να μπορεί να συνδεθεί με τις υπόλοιπες. Στο τμήμα **<declares>** δηλώνονται αντικείμενα δεδομένων, όπως καταχωρητές, μνήμες ή απλά καλώδια. Επίσης εδώ μπορούν να δημιουργηθούν και δομές διαδικασιών, που ονομάζονται **"functions"** και **"tasks"**, Η διαφορά τους είναι ότι τα πρώτα επιστρέφουν κάποια τιμή, ενώ τα δεύτερα δεν επιστρέφουν. Στο τμήμα **<module items>** τοποθετούνται οι δομές αρχικοποίησης και υλοποίησης του κυκλώματος. Αυτές μπορεί να είναι **"initial"** δομές ή συνεχείς αναθέσεις τιμών ή ακόμη και **"instances"** από άλλες οντότητες.

2.6.3 Αποθήκευση και μεταφορά δεδομένων

Οι βασικές λειτουργίες των οντοτήτων είναι η αποθήκευση και η μεταφορά δεδομένων από και προς τις άλλες οντότητες που υπάρχουν σε ένα σύστημα. Η Verilog έχει δύο βασικές δομές, μία για αποθήκευση και άλλη για μεταφορά δεδομένων. Αυτές είναι οι καταχωρητές **<reg>** και τα καλώδια **<wire>**.

Οι καταχωρητές χρησιμοποιούνται για την αποθήκευση των δεδομένων, ενώ τα καλώδια για τη διασύνδεση των οντοτήτων του συστήματος. Οι μεταβλητές τύπου

καλωδίων <wire> ΔΕΝ αποθηκεύουν τιμές. Οι παραπάνω τύποι δεδομένων μπορεί να έχουν μια από τις παρακάτω τιμές

0	Λογικό μηδέν
1	Λογικό ένα
x	μη έγκυρη τιμή – άγνωστη τιμή
z	τιμή υψηλής αντίστασης σε πύλες με τρία επίπεδα κατάστασης

Πίνακας 4 Πιθανές τιμές καταχωρητών

Έχοντας πλέον τα δύο βασικά χαρακτηριστικά ενός συστήματος, μνήμη και γραμμές δεδομένων, μπορεί να μοντελοποιηθεί από το πιο απλό έως το πιο σύνθετο σύστημα.

2.6.4 Διαφορές οντοτήτων με συναρτήσεις C

Μια σημαντική διαφορά των οντοτήτων της Verilog σε σχέση με τις συναρτήσεις των άλλων γλωσσών προγραμματισμού, είναι ότι μια οντότητα ποτέ δεν καλείται, δημιουργείται στην αρχή του προγράμματος και παραμένει εν ζωή μέχρι το τέλος του.

2.6.5 Δημιουργία «εικόνας» μίας οντότητας

Η γενικευμένη μορφή δημιουργίας ενός καινούργιου “instance” μίας οντότητας είναι η παρακάτω:

<module name><λίστα παραμέτρων><όνομα “instance”>(<λίστα θυρών>)

όπου η <λίστα παραμέτρων> είναι κάποιες τιμές που περνιούνται στο “instance” για την αρχικοποίηση του, όπως η καθυστέρηση μία πύλης.

Κατά τη διάρκεια λειτουργίας του συστήματος πολύ συχνά χρειάζεται να ανατεθεί κάποια τιμή είτε σε κάποιο καταχωρητή, είτε σε κάποιο καλώδιο. Στη Verilog

υπάρχουν δύο τρόποι ανάθεσης τιμής η διαδικαστική και η συνεχής. Η πρώτη γράφεται ως:

<μεταβλητή μνήμης> = <τιμή>

ενώ η δεύτερη γράφεται ως:

assign <όνομα καλωδίου> = <τιμή>

Οι δύο παραπάνω τρόποι ανάθεσης τιμής έχουν μία σημαντική διαφορά. Στην πρώτη περίπτωση η ανάθεση της νέας τιμής γίνεται όπως στις συνηθισμένες γλώσσες προγραμματισμού υψηλού επιπέδου. Στη δεύτερη περίπτωση η ανάθεση της τιμής περιγράφει τη δομή ενός συνδυαστικού κυκλώματος. Οπότε μέσω της ανάθεση τιμών στα καλώδια περιγράφεται η επικοινωνία της οντότητας με τις υπόλοιπες οντότητες στο σύστημα, ενώ με αναθέσεις τιμών σε μεταβλητές ελέγχεται η μνήμη του συστήματος μας.

2.7 Τελεστές πράξεων

Στη Verilog υπάρχουν αρκετοί τελεστές πράξεων που ενεργούν είτε μεταξύ μεταβλητών, είτε πάνω σε μία μοναδική. Οι τελεστές που υποστηρίζονται περιγράφονται στη συνέχεια.

2.7.1 Αριθμητικοί τελεστές

Οι αριθμητικοί τελεστές στη Verilog επενεργούν συνήθως σε δύο τελεστέους. Οι τελεστέοι είναι μη προσημασμένοι αριθμοί όταν είναι καταχωρητές ή καλώδια, ενώ είναι προσημασμένοι, όταν είναι πραγματικοί ή ακέραιοι. Οι τελεστές που υπάρχουν είναι οι παρακάτω:

Τελεστής	**Όνομα**
+	πρόσθεση
-	αφαίρεση
*	πολλαπλασιασμός
/	διαίρεση
%	modulus

Πίνακας 5 Αριθμητικοί τελεστές

2.7.2 Συγκρισιακοί τελεστές

Οι συγκρισιακοί τελεστές συγκρίνουν δύο τελεστέους και επιστρέφουν μία τιμή λογικού 1 (αληθής) ή λογικού 0 (ψευδής)

Τελεστής	**Όνομα**
>	μεγαλύτερο
>=	μεγαλύτερο ή ίσο
<	μικρότερο
<=	μικρότερο ή ίσο
==	ίσο
!=	διάφορο

Πίνακας 6 Συγκρισιακοί τελεστές

2.7.3 Λογικοί τελεστές

Οι λογικοί τελεστές ενεργούν σε λογικούς τελεστέους και επιστρέφουν λογικό 1 ή λογικό 0.

Τελεστής	**Όνομα**
>	μεγαλύτερο
>=	μεγαλύτερο ή ίσο
<	μικρότερο

Πίνακας 7 Λογικοί τελεστές

2.7.4 Τελεστές bitwise

Οι τελεστές bitwise ενεργούν στα bit ενός ή δύο τελεστέων. Η τιμή που επιστρέφουν είναι ίση με το μέγεθος των τελεστέων

Τελεστής	**Όνομα**
~	λογική άρνηση
&	λογικό ΚΑΙ
|	λογικό Η'
^	αποκλειστικό Η'
~&	λογικό ΚΑΙ με άρνηση
~|	λογικό Η' με άρνηση
~^ *ή* ^~	λογικό αποκλειστικό Η' με άρνηση

Πίνακας 8 Τελεστές bitwise

2.7.5 Τελεστές μοναδιαίας μείωσης (Unary reduction)

Οι τελεστές μοναδιαίας μείωσης ενεργούν στα bit ενός τελεστέου και επιστρέφουν μία τιμής ενός bit. Κάνουν τι ζητούμενη πράξη σε ένα προς ένα στα bit του τελεστέου μεταξύ τους και επιστρέφουν μία τιμή λογικού 1 ή λογικού 0.

Για παράδειγμα: ***reg[7..0] val; assign is.zero = ~|val***

Τελεστής	**Όνομα**
~	λογική άρνηση
&	λογικό ΚΑΙ
|	λογικό Η'
^	αποκλειστικό Η'
~&	λογικό ΚΑΙ με άρνηση
~|	λογικό Η' με άρνηση
~^	λογικό αποκλειστικό Η' με άρνηση

Πίνακας 9 Τελεστές μοναδιαίας μείωσης

2.7.6 Προτεραιότητα τελεστών

Η προτεραιότητα των τελεστών που ακολουθείται στα προγράμματα της Verilog είναι η παρακάτω:

Τελεστές μοναδιαίας μείωσης	: ! & ~& | ~| ^ ~^ + -	υψηλότερη προτεραιότητα
	* / %	
	<< >>	
	< <= > >=	
	== != === ~==	
	& ~& ^ ~^	
	| ~|	
	&&	
	||	
	?:	χαμηλότερη προτεραιότητα

Πίνακας 10 Προτεραιότητα τελεστών

2.8 Δομές ελέγχου

Οι δομές ελέγχου στη Verilog είναι παρόμοιες με τις δομές ελέγχου στη C. Η διαφορά που έχουν στη σύνταξη είναι ότι αντί για τις συνηθισμένες αγκύλες {}, χρησιμοποιούνται τα begin – end. Οι δομές που υπάρχουν είναι οι παρακάτω.

- **if – else**

```
if (A = = 4)
   begin
      B = 2;
   end
else
   begin
      B = 4;
   end
```

- **case select**

```
case (<μεταβλητή>)
   <τιμή 1>:     εντολές
   <τιμή 2>:     εντολές
   <default>:    εντολές
end case
```

Η διαφορά της δομής «case select» από την αντίστοιχη της C είναι ότι εκτελούνται μόνο οι εντολές που αντιστοιχούν στη πρώτη τιμή που είναι ίση με τη μεταβλητή μας και μετά το πρόγραμμα βγαίνει από τη δομή ελέγχου, ούτως ή άλλως χωρίς να χρειάζεται η αντίστοιχη εντολή “break” της C.

2.9 Δομές επανάληψης

Στη Verilog υπάρχουν δομές επανάληψης αντίστοιχες με της C, αυτές είναι οι for,while,repeat. Η δομή «for» συντάσσεται παρόμοια με την αντίστοιχη δομή της C, αλλά εδώ δεν υφίσταται το «i++» και πρέπει να συντάσσεται σε μορφή «i=i+1». Αντίστοιχα συντάσσονται και οι δομές while, repeat. Όλες ωστόσο γράφονται με προκαθορισμένο αριθμό κύκλων εκτέλεσης. Αν ο αριθμός αυτός χρειάζεται να αλλάζει, τότε πρέπει η εκτέλεση να συγχρονιστεί με το ρολόι του συστήματος.

Το παραπάνω είναι πολύ σημαντικό όσον αφορά ένα ψηφιακό σύστημα, το οποίο χρειάζεται να περιγραφθεί. Μία λύση ενός προβλήματος με σταθερούς κύκλους εκτέλεσης μπορεί να απλοποιηθεί με τους γνωστούς τρόπους και να βελτιστοποιηθεί, αυτό γίνεται κατά τη διάρκεια της σύνθεσης. Επίσης εάν χρειαστεί τότε δίναται κατά τη σύνθεση κάποια τμήματα να γίνουν παράλληλα, ανάλογα με τη διάταξη των πυλών του FPGA. Ωστόσο όταν η εκτέλεση απαιτεί μεταβλητό αριθμό κύκλων τότε αυτό είναι αδύνατο, έτσι πρέπει ο σχεδιαστής να φτιάξει μια πρωταρχική οντότητα την οποία να συγχρονίσει με το ρολόι και να εκτελεστεί όσες φορές τελικά χρειαστεί. Το πρόγραμμα που υλοποιεί τη σύνθεση θα μπορέσει να βελτιώσει μονάχα την πρωταρχική οντότητα σε αυτή την περίπτωση.

2.10 Μεταφορές δεδομένων Blocking και non-Blocking

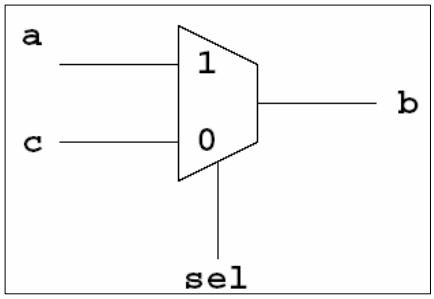
Υπάρχουν δύο τρόποι ανάθεσης τιμής στη Verilog. Αυτοί είναι οι “Blocking” και “non-Blocking”. Ξεχωρίζουν στον τρόπο σύνταξης, γιατί η πρώτη αναφέρεται με το σύμβολο «=» και η δεύτερη με το σύμβολο «<=». Κατά την εκτέλεση τους η διαφορά που έχουν έγκειται στο ότι στη μεν πρώτη η τιμή μεταφέρεται εκείνη τη στιγμή στις μεταβλητές, όπως γίνεται στις παραδοσιακές γλώσσες προγραμματισμού. Στη δεύτερη περίπτωση υπολογίζεται η νέα τιμή που πρέπει να πάρει η μεταβλητή, αλλά ανατίθεται σε αυτή στο τέλος της μονάδας του χρόνου.

2.11 Παραδείγματα προγραμμάτων Verilog

Στις παραπάνω παραγράφους περιγράφθηκαν σε μεγάλο βαθμό τα χαρακτηριστικά της γλώσσας Verilog, στη συνέχεια θα δοθούν ορισμένα παραδείγματα έτσι ώστε να γίνουν πιο κατανοητά τα προηγούμενα.

2.11.1 Πολυπλέκτης 2 σε 1

Ο πολυπλέκτης που φαίνεται στο διπλανό σχήμα μπορεί να υλοποιηθεί με διάφορους τρόπους, παρατίθενται τρεις τρόποι.



Εικόνα 1 Πολυπλέκτης 2 σε 1

2.11.1.1 Υλοποίηση με διαδικασία (χρησιμοποιείται always)

```
always@(a or b or sel)
     if (sel == 1'b1)
          c = a;
     else
          c = b;
```

Στην πρώτη υλοποίηση του πολυπλέκτη χρησιμοποιείται η ντιρεκτίβα “always”, αυτή χρησιμοποιείται όταν πρέπει ένα σύνολο εντολών να εκτελούνται πάντα, όταν αλλάζει η τιμή ενός σήματος. Η σύνταξη της εντολής γίνεται με τον παρακάτω τρόπο:

always@(*trigger1 or trigger2 and trigger3*) **begin**
<σύνολο εντολών προς εκτέλεση>
end

Τα triggerX είναι τα σήματα που ελέγχουν το κατά πόσο θα τρέξουν οι εντολές. Στην περίπτωση του πολυπλέκτη πρέπει όταν αλλάζει η είσοδος «a» ή η είσοδος «b» ή η είσοδος «c», να υπολογίζεται ξανά η τιμή της εξόδου. Για τον έλεγχο της σωστής εξόδου γίνεται χρήση μία απλής δομή “if-else”.

Αυτό που μπορεί να παρατηρηθεί είναι ότι το κύκλωμα θα δουλέψει εάν αλλάξει τουλάχιστον μία από τις τρεις εισόδους του.

2.11.1.2 Χρησιμοποιώντας τον τελεστή «?»

```
wire c = sel ? a : b;
```

Ο τελεστής «?» έχει ακριβώς ίδια σύνταξη με τον αντίστοιχο τελεστή της C. Το αποτέλεσμα της σύνθεσης είναι ένα πολυπλέκτης 2 σε 1, ενώ συντακτικά έχει χρησιμοποιηθεί συνεχής ανάθεση τιμών, έτσι όποτε αλλάξει η τιμή του “sel” τότε

επιλέγεται αντίστοιχα η “a” ή “b”, ενώ σε κάθε μία από της προηγούμενες περιπτώσεις είτε το a είτε το b, είναι απευθείας συνδεδεμένο με την έξοδο μας.

2.11.1.3 Χρησιμοποιώντας “select-case”

```
always @ (a or b or sel)
case (sel)
1’b1: c = a;
1’b0: c = b;
endcase
```

Στην παραπάνω υλοποίηση χρησιμοποιείται πάλι η ντιρεκτίβα “always” για τον έλεγχο της αλλαγής της τιμής των σημάτων ελέγχου. Ωστόσο για την επιλογή της σωστής τιμής εξόδου χρησιμοποιείται η δομή “case”.

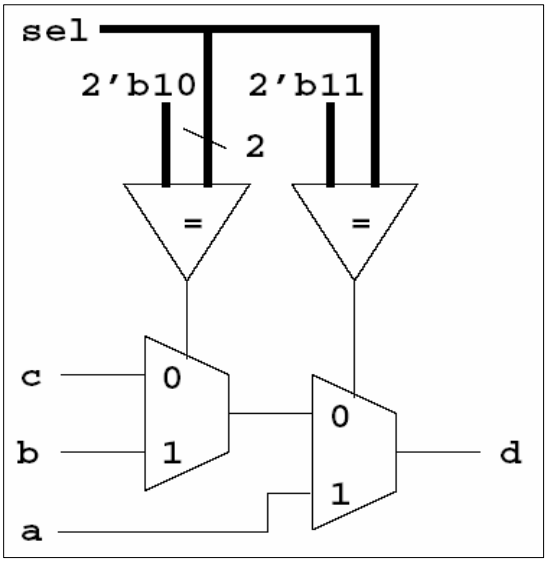
Η σύνταξη της είναι σχεδόν ίδια με την αντίστοιχη εντολή της C, ωστόσο εδώ δε χρειάζεται “break” μετά από κάθε ομάδα εντολών.

Παραπάνω δόθηκαν τρεις διαφορετικοί τρόποι σύνταξης του ίδιου κυκλώματος, το αποτέλεσμα και των τριών είναι ένας πολυπλέκτης 2-1.

2.11.2 Αποκωδικοποιητής προτεραιοτήτων

Στη συνέχεια θα δοθούν διαφορετικοί τρόποι υλοποίησης ενός αποκωδικοποιητή προτεραιοτήτων.

Το διάγραμμα του αποκωδικοποιητή φαίνεται στο διπλανό σχήμα, στη συνέχεια θα δοθούν οι τρόποι υλοποίησης του.



Εικόνα 2 Αποκωδικοποιητής προτεραιοτήτων

2.11.2.1 Χρησιμοποιώντας δομή Case

```
always @ (sl or a or b or c)
   case (sel)
   2'b11: d = a;
   2'b10: d = b;
   default: d = c;
endcase
```

Ο αποκωδικοποιητής προτεραιοτήτων, μπορεί να συνταχθεί είτε χρησιμοποιώντας δομή case είτε if. Το ποια είσοδος θα έχει τη μεγαλύτερη προτεραιότητα εξαρτάται από τη σειρά που έχουν γραφτεί οι διαφορετικές επιλογές μέσα στη δομή ελέγχου.

2.11.2.2 Χρησιμοποιώντας δομή "if"

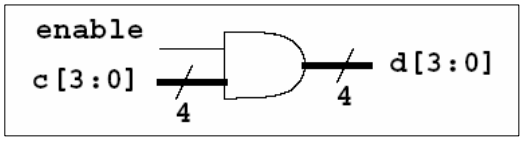
```
always @ (sl or a or b or c)
if (sel == 2'b11)
   d = a;
else if (sel ==2'b10)
   d = b;
else
   d = c;
```

Το κύκλωμα του αποκωδικοποιητή χρησιμοποιώντας δομή “if-then-else”, φαίνεται και εδώ ότι η σύνταξη οδηγεί σε κύκλωμα ελέγχου προτεραιοτήτων, η σειρά με την οποία γράφονται τα “if”, ελέγχουν την προτεραιότητα κάθε πύλης εισόδου.

Και στα δύο παραπάνω κυκλώματα χρησιμοποιείται η εντολή always@(..) για να ελεγχθεί το πότε μεταβάλλεται ένα σήμα εισόδου, έτσι ώστε να αλλάζει αντίστοιχα η έξοδος του κυκλώματος.

2.11.3 Έλεγχος διαύλου δεδομένων

Σκοπός του κυκλώματος είναι να ελέγξει ένα δίαυλο δεδομένων πλάτους 4 bit, από ένα σήμα ελέγχου (enable), το πλάτος του διαύλου μπορεί να είναι όσο μεγάλο χρειάζεται.



Εικόνα 3 Ελεγκτής διαύλου 4bit

2.11.3.1 Υλοποίηση με «καλώδιο» (wire)

```
wire [3:0] d = ({4{enable}} & c);
```

Στην παραπάνω υλοποίηση χρησιμοποιείται συνεχής ανάθεση τιμής σε μία πύλη εξόδου, πλάτους τεσσάρων bit. Επαναλαμβάνεται ότι σε αυτή την περίπτωση δεν υπάρχει αποθήκευση της τιμής, αλλά απλή εμφάνιση ή όχι του σήματος εισόδου στην έξοδο.

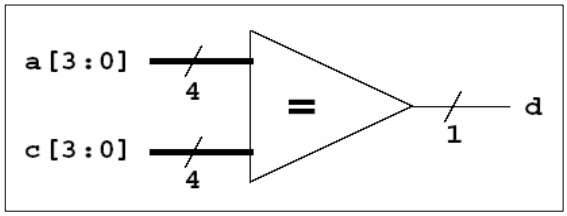
2.11.3.2 Υλοποίηση με καταχωρητή

```
reg [3:0] d;
always @ (c or enable)
    d = c & {4{enable}};
```

Σε αυτή την υλοποίηση χρησιμοποιείται ένας καταχωρητής για αποθήκευση των δεδομένων, το αποτέλεσμα εμφανίζεται στην έξοδο (καταχωρητή), μόλις αλλάξει κάποιο από τα σήματα εισόδου. Σε αυτή την περίπτωση η τιμή εξόδου αποθηκεύεται. Αυτό το κύκλωμα μπορεί με μία μικρή μετατροπή να χρησιμοποιηθεί σε ένα συγχρονισμένο κύκλωμα με ένα ρολόι, αρκεί οι είσοδοι να γίνονται AND με το σήμα από το ρολόι, έτσι θα είναι βέβαιο ότι το αποτέλεσμα εμφανίζεται στην έξοδο σε ένα κτύπο του ρολογιού.

2.11.4 Συγκριτής σημάτων

Στόχος του αυτού του κυκλώματος είναι ο έλεγχος δύο σημάτων πλάτους 4bit και εμφάνιση ενός σήματος 1bit στην έξοδο, το οποίο θα γίνεται 1 εάν τα σήματα είναι ίδια, διαφορετικά θα γίνεται 0.



Εικόνα 4 Συγκριτής σημάτων

2.11.4.1 Χρησιμοποιώντας καλώδιο

```
wire d;
assign d = (a == c);
```

2.11.4.2 Χρησιμοποιώντας καταχωρητή στην έξοδο

```
reg d;
always @ (a or c)
d = (a == c);
```

Οι διαφορές στις δύο υλοποιήσεις είναι αντίστοιχες με τις διαφορές ελεγκτή δίαυλου όταν χρησιμοποιήθηκε καταχωρητής εξόδου και όταν δε χρησιμοποιήθηκε.

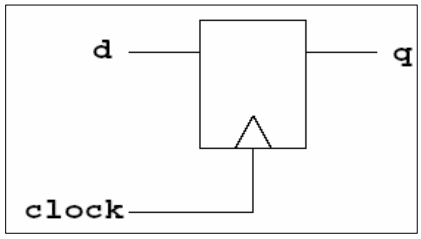
Το κύκλωμα συντακτικά δεν παρουσιάζει κάποιο δύσκολο σημείο, γίνεται η σύγκριση “a==b” και είτε το αποτέλεσμα πάει στην έξοδο κατευθείαν (πρώτη περίπτωση), είτε αποθηκεύεται στον καταχωρητή εξόδου.

2.11.5 Flip – Flops

Πολύ σημαντικό ρόλο στα συνδυαστικά κυκλώματα παίζουν τα Flip – Flops, αφού μπορούν να χρησιμοποιηθούν ως μονάδες μνήμης. Στη συνέχεια παρουσιάζονται ορισμένα κυκλώματα “D Flip Flop”.

2.11.5.1 Θετικά ακμοπυροδοτούμενο D - Flip Flop

```
always @ (posedge clock)
q <= d;
```

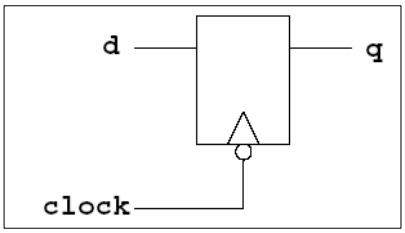


Εικόνα 5 Θετικά ακμοπυροδοτούμενο D - Flip Flop

Και εδώ χρησιμοποιείται η εντολή “always”, ενώ για να ενεργοποιείται το κύκλωμά μόνο στη θετική ακμή του ρολογιού, χρησιμοποιείται η εντολή “posedge”. Επίσης επειδή η διαδικασία χρησιμοποιεί το ρολόι του συστήματος, χρησιμοποιείται «μη-διακοπτόμενη» (non-blocking), ανάθεση δεδομένων στην έξοδο, κάνοντας χρήση του συμβόλου «<=».

2.11.5.2 Αρνητικά ακμοπυροδοτούμενο D Flip Flop

```
always @ (negedge clock)
   q <= d;
```



Εικόνα 6 Αρνητικά ακμοπυροδοτούμενο D Flip Flop

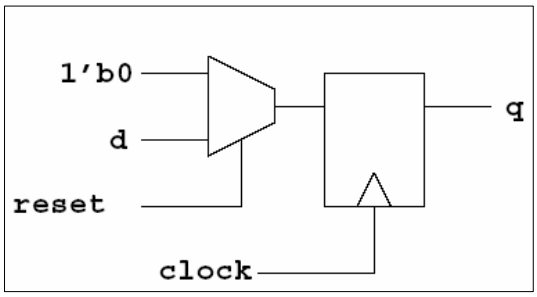
Η μόνη διαφορά ανάμεσα στα δύο Flip Flop, είναι ότι το ένα ενεργοποιείται στη θετική ακμή, ενώ το άλλο στην αρνητική η διαφορά τους βρίσκεται στο ότι το δεύτερο χρησιμοποιεί την εντολή "negedge". Έτσι δηλώνεται ότι το κύκλωμα ενεργοποιείται στην αρνητική ακμή του σήματος ελέγχου.

2.11.6 Flip – Flops με δυνατότητα αρχικοποίησης (reset)

Σκοπός εδώ είναι η δημιουργία Flip Flop, τα οποία όμως να μπορούν να αρχικοποιούνται σε μία συγκεκριμένη τιμή.

2.11.6.1 D Flip-Flop με συγχρονισμένη αρχικοποίηση

```
always @ (posedge clock)
if (reset)
q <= 1'b0;
else
q <= d;
```



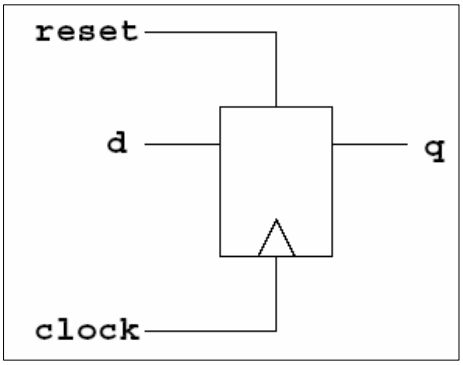
Εικόνα 7 D Flip-Flop με συγχρονισμένη αρχικοποίηση

Το παραπάνω κύκλωμα δημιουργεί ένα flip-flop, το οποίο όμως όταν ενεργοποιείται η είσοδος "reset", η έξοδος του γίνεται 1. Αυτή η διαδικασία συμβαίνει συγχρονισμένα με το ρολόι του συστήματος. Αυτό συμβαίνει, γιατί στη λίστα των

σημάτων ελέγχου (***always@(posedge clock***)) του κυκλώματος βρίσκεται μονάχα το σήμα από το ρολόι του συστήματος.

2.11.6.2 D Flip-Flop με ασύγχρονη αρχικοποίηση

Στο παράδειγμα που ακολουθεί παρουσιάζεται ένα D Flip Flop με ασύγχρονη αρχικοποίηση, δηλαδή η αρχικοποίηση του μπορεί να γίνει ανεξάρτητα από το ρολόι του συστήματος.



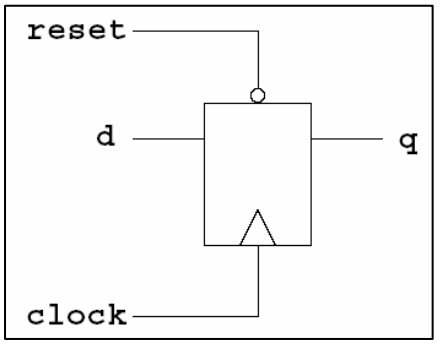
Εικόνα 8 D Flip Flop με ασύγχρονη αρχικοποίηση

Η αρχικοποίηση του Flip Flop γίνεται με σήμα σε κατάσταση 1 (Active High), ο κώδικας που το υλοποιεί φαίνεται στη συνέχεια.

```
always @ (posedge clock or posedge reset)
if (reset)
    q <= 1'b0;
else
    q <= d;
```

Για την υλοποίηση χρησιμοποιείται διαδικασία με “while”. Στη λίστα με τα σήματα ενεργοποίησης του κυκλώματος, έχουν προστεθεί το ρολόι του συστήματος και το σήμα αρχικοποίησης. Τα δύο σήματα ενεργοποιούν το κύκλωμα στη θετική ακμή τους και αυτό φαίνεται από το “posedge” που έχει προστεθεί πριν από τη δήλωση τους.

Στη συνέχεια παρουσιάζεται το ίδιο παράδειγμα με αρχικοποίηση παλμού σε κατάσταση 0 (Active Low reset). Το κύκλωμά μετά τη σύνθεση φαίνεται στη διπλανή εικόνα.



Εικόνα 9 D Flip Flop με ασύγχρονη αρχικοποίηση (Active Low reset)

Ο κώδικας που το υλοποιεί είναι ο παρακάτω

```
always @ (posedge clock or negedge reset)
if (~reset)
q <= 1'b0;
else
q <= d;
```

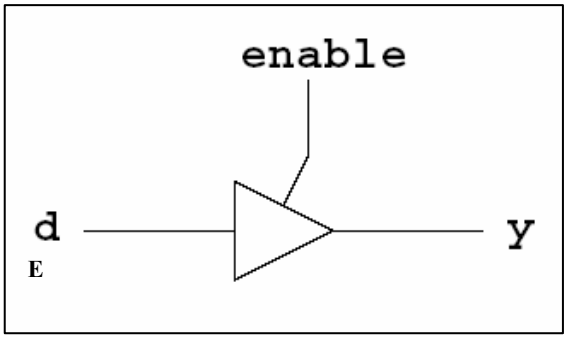
Οι διαφορές από την προηγούμενη υλοποίηση είναι το "negedge" που προσδιορίζει ότι το σήμα ελέγχου θα ενεργοποιεί το κύκλωμα στην αρνητική ακμή του σήματος.

2.11.7 Οδηγοί τριών καταστάσεων (Tri-State Drivers)

Οι οδηγοί τριών καταστάσεων είναι πολύ σημαντικοί στην υλοποίηση κυκλωμάτων. Συνήθως στα σύνθετα συστήματα υπάρχουν επιμέρους μικρότερα κυκλώματα που μοιράζονται κοινόχρηστα τμήματα του συστήματος. Το πλέον συνηθισμένο είναι η ύπαρξη ενός κοινού διαύλου δεδομένων, όπου πάνω του συνδέονται όλα τα κυκλώματα που επενεργούν πάνω στα δεδομένα του συστήματος. Όταν ένα από αυτά δε χρειάζεται το δίαυλο θέτει τις εξόδους του σε κατάσταση υψηλής αντίστασης (Hi-Z), έτσι ώστε να μην τον επηρεάζει και κάποιο άλλο κύκλωμα να μπορεί να τον

χρησιμοποιήσει. Το διάγραμμα του οδηγού τριών καταστάσεων φαίνεται στην εικόνα 10.

Στη συνέχεια παρουσιάζονται τρεις τρόποι υλοποίησης του οδηγού.



Εικόνα 10 Οδηγός τριών καταστάσεων

2.11.7.1 Χρησιμοποιώντας καταχωρητή

Η υλοποίηση του οδηγού χρησιμοποιώντας καταχωρητή είναι η παρακάτω

```
reg y;
always @ (d or enable)
if (enable)
    y = d;
else
    y = 1'bz;
```

Για την υλοποίηση χρησιμοποιήθηκε η διαδικασία με “always”, “d” είναι το σήμα εισόδου, ενώ το “enable” είναι το σήμα ελέγχου της εξόδου. Όταν το σήμα ελέγχου γίνεται 0, τότε η έξοδος οδηγείται σε κατάσταση υψηλής αντίστασης. Το σήμα εξόδου πρακτικά είναι το περιεχόμενο του καταχωρητή “y”.

2.11.7.2 Χρησιμοποιώντας «καλώδιο»

```
wire y;
assign y = enable ? d : 1'bz;
```

Η υλοποίηση με καλώδιο δε παρουσιάζει κάποια δυσκολία, σε σχέση με την προηγούμενη δεν υπάρχει ανάγκη χρήσης καταχωρητή, ο οποίος σε πολλές περιπτώσεις δεν είναι απαραίτητος για την ορθή λειτουργία του κυκλώματος.

2.11.7.3 Χρησιμοποιώντας πρωταρχικό κύκλωμα (primitive)

Η Verilog έχει ορισμένα έτοιμα κυκλώματα, πρωταρχικού επιπέδου έτσι ώστε να μη χρειάζεται ο προγραμματιστής να τα υλοποιεί. Ένα από αυτά είναι και ο οδηγός τριών καταστάσεων.

```
bufif1 (y,d,enable);
```

Στη συνέχεια παρουσιάζεται μία λίστα με τα πρωταρχικά, έτοιμα κυκλώματα, που υπάρχουν ήδη υλοποιημένα και μπορούν να χρησιμοποιούνται κατευθείαν.

Gate Types		Component
Gates	Allows strengths	and, nand, or, nor,xor, xnor buf, not
Three State Drivers	Allows strengths	buif0,bufif1 notif0,notif1
MOS Switches	No strengths	nmos,pmos,cmos, rnmos,rpmos,rcmos
Bi-directional switches	No strengths, non resistive	tran, tranif0, tranif1
	No strengths, resistive	rtran,rtranif0, rtranif1
	Allows strengths	pullup pulldown

Εικόνα 11 Λίστα πρωταρχικών στοιχείων Verilog

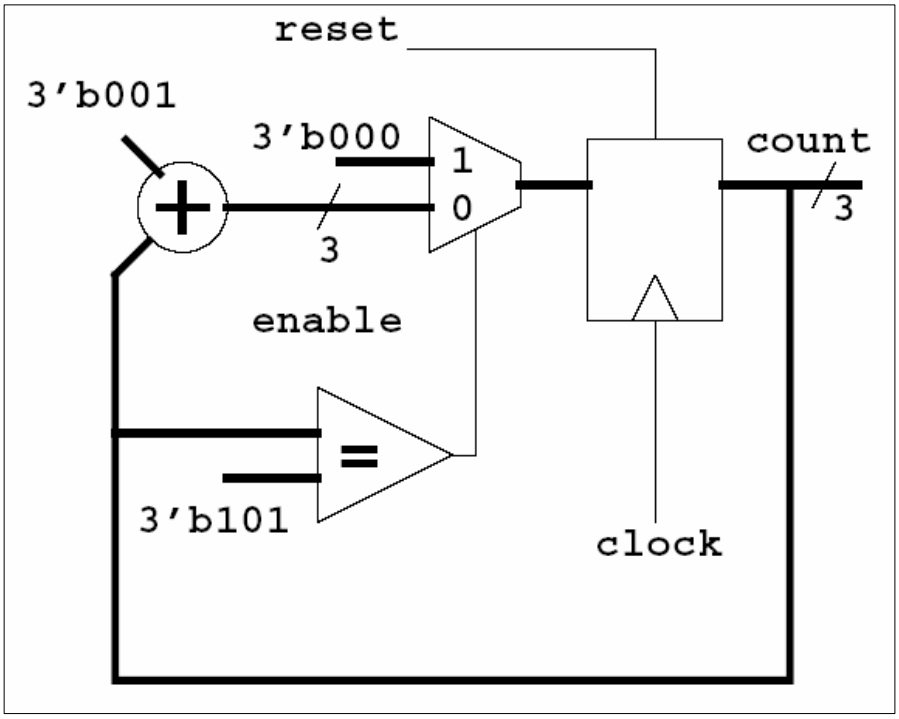
2.11.8 Μετρητές

Πολύ σημαντικά στοιχεία για τα ψηφιακά συστήματος είναι οι μετρητές. Χρησιμοποιούνται για διάφορους σκοπούς, όπως να προκαλούν καθυστερήσεις συγκεκριμένου διαστήματος.

Στη συνέχεια δίνεται το κύκλωμα και ο κώδικα υλοποίησης ενός μετρητή τριών bit, με ασύγχρονη είσοδο αρχικοποίησης και μέγιστη τιμή μέτρησης. Στη συγκεκριμένη υλοποίησης ο μετρητής μετράει σειριακά από το μηδέν (0) έως το πέντε (5).

2.11.8.1 Μετρητής τριών bit

Το κύκλωμα του μετρητή φαίνεται στην παρακάτω εικόνα.



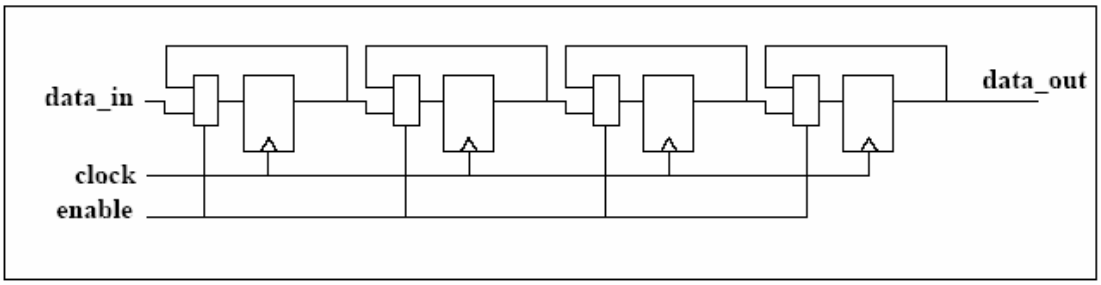
Εικόνα 12 Μετρητής τριών bit, με ασύγχρονη αρχικοποίηση

Ο κώδικας υλοποίησης του παραπάνω κυκλώματος είναι ο ακόλουθος

```
always @ (posedge clock or posedge reset)
if (reset)
    count <= 3'b0;
else
    if (count == 3'b101)
        count <= 3'b0;
    else
        count <= count + 3'b001;
```

Έχει γίνει χρήση μίας διαδικασία "while" για τον έλεγχο του κυκλώματος, η είσοδος αρχικοποίησης είναι ασύγχρονη και μηδενίζει την έξοδο του μετρητή. Η μέγιστη τιμή που λαμβάνει ο μετρητής είναι το πέντε (5) ή σε δυαδική μορφή *3'b101*, μόλις φτάσει σε αυτή την τιμή το κύκλωμα αυτόματα μηδενίζει την τιμή της εξόδου του.

2.11.9 Καταχωρητής ολίσθησης με ενεργοποίηση



Εικόνα 13 Καταχωρητής ολίσθησης με ενεργοποίηση

Τελευταίο παράδειγμα είναι ένας καταχωρητής ολίσθησης με είσοδο ενεργοποίησης. Ο καταχωρητής φαίνεται στο σχ. 13, είναι μήκους τεσσάρων bit. Ο καταχωρητής έχει μία είσοδο δεδομένων, μία είσοδο για το ρολόι του συστήματος, μία είσοδο ενεργοποίησης (enable) και μία σειριακή έξοδο δεδομένων.

Ο κώδικας υλοποίησης του καταχωρητή φαίνεται στη συνέχεια.

```
module enabled_shift_reg (clock,enable,data_in,data_out);
input clock;
input enable;
input [3:0] data_in;
output [3:0] data_out;
reg [3:0] data_out;
reg [3:0] shift_reg_1;
reg [3:0] shift_reg_2;
reg [3:0] shift_reg_2;

always @ (posedge clock)
    if (enable)
    begin
        shift_reg_1 <= data_in;
        shift_reg_2 <= shift_reg_1;
        shift_reg_3 <= shift_reg_2;
        data_out <= shift_reg_3;
    end
endmodule
```

Η είσοδος είναι πλάτους τεσσάρων bit, για τη δημιουργία του καταχωρητή χρησιμοποιούνται τέσσερις παράλληλοι καταχωρητές ίδιου πλάτους. Ο καταχωρητής διαβάζει την είσοδό του στη θετική ακμή κάθε παλμού του ρολογιού ***(always @ (posedge clock)).*** Για να μεταφερθούν ωστόσο τα δεδομένα θα πρέπει η είσοδος ενεργοποίησης να είναι ένα.

Στο παράδειγμα αυτό επίσης γίνεται χρήση “non-Blocking” μεταφοράς δεδομένων, δηλαδή στις τέσσερις εντολές μεταφοράς των δεδομένων υπολογίζονται οι νέες τιμές που πρέπει να έχουν οι τέσσερις παράλληλοι καταχωρητές σε κάθε βήμα, αλλά αυτές παίρνονται μετά το πέρας του παλμού.

Επίσης στο παραπάνω παράδειγμα φαίνεται η τυπική δομή μίας οντότητας, έτσι όπως αυτή υλοποιείται στη Verilog.

3

XR16

3.1 Επεξεργαστές soft-core

Οι επεξεργαστές του παραπάνω τύπου είναι γραμμένοι σε κάποια γλώσσα περιγραφής υλικού και υλοποιούνται σε “Field Programmable Gate Arrays” (FPGA’s). Μπορούν να τρέξουν σε μεγάλη γκάμα ταχυτήτων, έτσι τέτοιου είδους επεξεργαστές είναι η βάση πολλών embedded εφαρμογών, αλλά και συστημάτων που χρειάζονται μία δυνατή, αλλά προσαρμόσιμη επεξεργαστική μηχανή.

Οι επεξεργαστές τύπου soft-core έχουν τη δυνατότητα να αλλάζουν πολύ εύκολα και να προσαρμόζονται στις ανάγκες κάποιας εφαρμογής. Επίσης μπορεί εύκολα να γίνει έλεγχος λαθών και βελτιστοποίηση τους. Έτσι επιτυγχάνονται πολύ μικροί χρόνοι για την παραγωγή νέων συστημάτων.

3.2 XR16

Η ανάπτυξη της παρούσας διπλωματικής εργασίας έχει στηριχθεί πάνω σε έναν επεξεργαστή τύπου soft-core που ονομάζεται XR16. Ο επεξεργαστής αυτός είναι αρχιτεκτονικής RISC 16bit. Μαζί με τον επεξεργαστή παρέχεται και compiler για τη γλώσσα προγραμματισμού C. Έτσι μπορεί κάποιος να γράψει ένα πρόγραμμα σε μία γλώσσα κοινά αποδεκτή και γνωστή, αντί να πρέπει να μάθει μια καινούργια γλώσσα, σχεδιασμένη ειδικά για αυτόν τον επεξεργαστή.

3.3 Compiler

Ο compiler που επιλέχθηκε είναι ο LCC. Αυτός επιλέχθηκε, γιατί ο κώδικας του είναι σχεδόν ανεξάρτητος από το μηχάνημα που θα χρησιμοποιηθεί. Για την επιλογή του επεξεργαστή υπάρχει ένα αρχείο, όπου απλά περιγράφονται οι καταχωρητές και η χρήση τους, το μέγεθος των τύπων των μεταβλητών που χρησιμοποιούνται, καθώς και η αντιστοίχηση των εντολών γλώσσας μηχανής (assembly) σε κωδικούς.

3.4 Καταχωρητές

Ο επεξεργαστής που χρησιμοποιείται υλοποιεί αρχιτεκτονική με 16 καταχωρητές, r0 – r15. Κάποιοι είναι γενικής χρήσης ενώ άλλοι είναι δεσμευμένοι από τον επεξεργαστή. Η χρήση τους φαίνεται στη συνέχεια.

Καταχωρητής	**Χρήση**
r0	Πάντα μηδέν
r1	Δεσμευμένος από τον assembler

r2	Τιμή που επιστρέφουν οι συναρτήσεις
r3-r5	Πεδία συναρτήσεων
r6-r9	(temporaries)
r10-r12	Καταχωρητές μεταβλητές
r13	Δείκτης θέσης μνήμης στοίβας
r14	Διεύθυνση επιστροφής από ρουτίνα διακοπής
r15	Διεύθυνση επιστροφής συνάρτησης

Πίνακας 11 Καταχωρητές XR16

3.5 Σύνολο εντολών

Το σύνολο των εντολών επιπέδου assembly που υλοποιούνται στον επεξεργαστή είναι ικανοποιητικό και μπορεί να καλύψει και σύνθετα προγράμματα. Για τις εντολές έχουν παρθεί οι επόμενες αποφάσεις:

- Οι add, sub, addi είναι εντολές με τρία ορίσματα
- Οι εντολές που δεν εμφανίζονται συχνά, όπως οι λογικές εντολές, πρόσθεση - αφαίρεση με κρατούμενο και εντολές ολίσθησης παίρνουν τρία ορίσματα
- Ο καταχωρητής r0 διαβάζεται πάντα 0
- Τα πεδία άμεσης ανάθεσης τιμής είναι μήκους 4 bit
- Για σταθερές μήκους 16bit χρησιμοποιείται ένα επιπλέον πρόθεμα με όνομα imm. Με αυτό δηλώνεται ότι χρησιμοποιούνται 12 bit της επόμενης εντολής για να συμπληρωθούν τα 16bit.
- Δεν υπάρχουν κωδικοί κατάστασης, αλλά χρησιμοποιείται μία αλληλοσυνδεόμενη εντολή σύγκρισης και μία υπό όρους ακολουθία διακλάδωσης.
- Jal (jump-and-link) μεταφέρεται ο δείκτης προγράμματος σε μία ενεργή διεύθυνση και η διεύθυνση επιστροφής σώζεται στον δεσμευμένο, για αυτή τη δουλειά, καταχωρητή.
- Η εντολή call μεταφράζεται ως “jal r15,func” σε μία εντολή μήκους 16bit
- Οι εντολές πολλαπλασιασμού, διαίρεσης, μεταβλητού ή πολλών βημάτων ολίσθησης καταχωρητών υλοποιούνται σε λογισμικό.

Για τις εντολές που υλοποιούνται στον επεξεργαστή υπάρχουν έξι διαφορετικά σχήματα (format), δηλαδή πώς γίνεται τελικά η αποκωδικοποίηση στον επεξεργαστή των 16bit κάθε εντολής.

Σχήμα	15 12	11 8	7 4	3 0
rrr	op	rd	ra	rb
rri	op	rd	ra	imm
rr	op	rd	fn	rb
ri	op	rd	fn	imm
i12	op	imm12		
br	op	cond	disp8	

Πίνακας 12 Σχήματα εντολών XR16

Στον παραπάνω πίνακα, οι συντομεύσεις αντιστοιχούν στα παρακάτω

- op: Operation Code, κωδικός εντολής
- rd: Destination Register, καταχωρητής αποτελέσματος
- ra: Καταχωρητής A
- rb: Καταχωρητής B
- imm: σταθερά μήκους 4bit
- fn: καινό πεδίο σε περιπτώσεις εντολών με δύο ορίσματα
- imm12: Όταν χρειάζονται σταθερές μήκους 16bit, αυτά είναι τα 12bit απομένουν από τα 4 που αναφέραμε ποιο πάνω.
- cond: Σε εντολές άλματος υπό συνθήκη, εδώ υπάρχει η συνθήκη προς έλεγχο
- disp8: offset από την τρέχουσα διεύθυνση σε περίπτωση που ανάλογα με συνθήκη άλματος το άλμα πρέπει να πραγματοποιηθεί.

Συγκεκριμένα οι εντολές που υλοποιούνται και ο τρόπος με τον οποίο συντάσσονται παρουσιάζονται στο παρακάτω πίνακα

Hex	**Fmt**	**Assembler**	**Semantics**
0dab	rrr	add rd, ra, rb	rd = ra + rb;
1dab	rrr	Sub rd, ra, rb	rd = ra – rb;
2dai	rri	Addi rd, ra, imm	rd = ra + imm;
3d*b	rr	{and or xor andn adc sbc} rd, rb	rd = rd *op* rb;
4d*i	ri	{andi ori xori andni adci sbci slli slxi srai srli srxi} rd, imm	rd = rd *op* imm;
5dai	rri	lw rd,imm(ra)	rd = *(int*)(ra+imm);
6dai	rri	lb rd,imm(ra)	rd = *(byte*)(ra+imm);
8dai	rri	sw rd,imm(ra)	*(int*)(ra+imm) = rd;
9dai	rri	sb rd,imm(ra)	*(byte*)(ra+imm) = rd;
Adai	rri	jal rd,imm(ra)	rd = pc,pc = ra + imm;
B*dd	br	{br brn beq bne bc bnc bv bnv blt bge ble bgt bltu bgeu bleu bgtu} label	if (cond) pc += 2*disp8;
Ciii	i12	call func	r15 = pc, pc = imm12<<4;
Diii	i12	imm imm12	imm'next$_{15:4}$ = imm12;
7xxx Exxx Fxxx	-	reserved	

Πίνακας 13 Δομή υλοποιημένων εντολών

3.6 Αρχιτεκτονική

Η αρχιτεκτονική του επεξεργαστή είναι απλή: υπάρχουν δύο κύριες μονάδες, μια μονάδα ροής δεδομένων (datapath) και μια μονάδα ελέγχου της προηγούμενης. Για κάθε μία εντολή που εκτελείται χρειάζονται στη μονάδα δεδομένων τα παρακάτω στοιχεία:

- Ένα σετ 16 καταχωρητών μήκους 16bit ο κάθε ένας, δύο εξόδους ταυτόχρονης ανάγνωσης, για να διαβάζονται οι τιμές των καταχωρητών που παίρνουν μέρος σε μία εντολή στον ίδιο κύκλο, και μία είσοδος για να γράφεται το αποτέλεσμα της εκτελούμενης εντολής.
- Ένας πολυπλέκτης 2-1 για να μπορεί να επιλέγεται μία σταθερή τιμή για την εντολή, αντί τιμή καταχωρητή.
- Μία αριθμητική λογική μονάδα για την εκτέλεση της εντολής
- Ένας καταχωρητής ολίσθησης
- Ένας επιπλέον αθροιστής για να υπολογίζεται η διεύθυνση της επόμενης εντολής
- Ένα πολυπλέκτης 7-1 για την επιλογή της τιμής εξόδου, από τον αθροιστή, τη μονάδα λογικής, τους καταχωρητές ολίσθησης, τη διεύθυνση επιστροφής ή τα δεδομένα ανάγνωσης
- Ένας ελεγκτής της τιμής εξόδου, με μία λογική έξοδο για το αν η τιμή εισόδου είναι μηδέν, αρνητική, υπάρχει ψηφίο κρατουμένου ή έχει γίνει υπερχείλιση
- Ένας μετρητής προγράμματος, με έναν πλήρη αθροιστή για να υπολογίζεται η τιμή του μετρητή σε περίπτωση άλματος και ένα πολυπλέκτη 2-1 για να φορτώνεται η τιμή από τον αθροιστή
- Ένας πολυπλέκτης για να μοιράζεται η θύρα της μνήμης για να διαβάζονται νέες εντολές και να γίνεται ανάγνωση και εγγραφή των δεδομένων

Στη συνέχεια θα αναλυθούν τα επιμέρους τμήματα που ουσιαστικά αποτελούν την αρχιτεκτονική του επεξεργαστή.

3.6.1 Αρχείο καταχωρητών

Το αρχείο των καταχωρητών αποτελείται από 16 διαθέσιμους καταχωρητές. Ο κάθε ένας από αυτούς είναι μήκους 16bit. Από αυτό το αρχείο υπάρχει η δυνατότητα ανάγνωσης της τιμής δύο διαφορετικών καταχωρητών ταυτόχρονα.

Η επιλογή που έγινε για το αρχείο τον καταχωρητών είναι να υπάρχουν για κάθε καταχωρητή από τους δεκαέξι δύο αντίγραφα του. Με αυτό τον τρόπο προσομοιώνεται μία

μνήμη dual-port που σε ορισμένα FPGA δεν υπάρχει από κατασκευής τους. Έτσι όταν γίνεται εγγραφή μία τιμής σε κάποιον καταχωρητή, αυτή γράφεται παράλληλα και στους δύο καταχωρητές που τον αντιπροσωπεύουν. Με το προηγούμενο τρόπο επιτυγχάνεται η ύπαρξη των ίδιων τιμών και στις δύο ομάδες παράλληλων καταχωρητών, όταν χρειάζεται να διαβαστούν οι τιμές δύο διαφορετικών καταχωρητών για την εκτέλεση μίας εντολής μπορεί να γίνει σε έναν κύκλο διαβάζοντας τον έναν καταχωρητή από το ένα σύνολο και τον άλλον από το δεύτερο.

3.6.2 Δεσμευμένος χώρος στην FPGA

Κάθε αρχείο καταχωρητών είναι ένα τμήμα μνήμης μεγέθους 16x16. Στην FPGA που χρησιμοποιήθηκε για την υλοποίηση του επεξεργαστή, κάθε “Configurable Logic Block” (CLB) έχει δύο Lookup Tables 4 εισόδων (4-LUT), το κάθε ένα από αυτά μπορεί να είναι μία μνήμη 16x1bit, έτσι συνολικά το αρχείο των καταχωρητών δεσμεύει χώρο μεγέθους 8 CLB.

3.6.3 Ανάγνωση και διάβασμα στους καταχωρητές

Για να είναι δυνατή η ανάγνωση και το γράψιμο των καταχωρητών στο ίδιο παλμό ρολογιού, γίνεται αναφορά στον καταχωρητή και στα δύο μισά του παλμού. Στο πρώτο μισό του παλμού η μονάδα ελέγχου εμφανίζει στη θύρα ανάγνωσης τον αριθμό του καταχωρητή που πρέπει να διαβαστεί. Στο πέσιμο του παλμού τα δεδομένα βρίσκονται στη θύρα εξόδου. Στο δεύτερο μισό η μονάδα ελέγχου εμφανίζει στη θύρα ανάγνωσης τον αριθμό του καταχωρητή που πρέπει να αναγνωστεί, στο επόμενο μέτωπο ανόδου του παλμού τα δεδομένα που υπάρχουν στη θύρα εισόδου γράφονται στο αντίστοιχο καταχωρητή.

3.6.4 Επιλογή τελεστών

Επόμενο στάδιο για την εκτέλεση της εντολής είναι η επιλογή των σωστών τελεστών. Συνήθως οι εντολές έχουν τρία ορίσματα, δύο για τα δεδομένα εισόδου και ένα για το αποτέλεσμα (εφεξής τα ορίσματα εισόδου θα τα λέμε A και B). Συνήθως τα A και B είναι η τιμή κάποιου καταχωρητή, ενώ αυτές οι τιμές μπορούν να διαβάζονται ταυτόχρονα.

Ο τελεστής A συνήθως είναι η τιμή ενός καταχωρητή, ωστόσο αυτό μπορεί να αλλάζει ανάλογα με την προηγούμενη εντολή που εκτελέστηκε. Υπάρχει ένας πολυπλέκτης μήκους 16bit, που ονομάζεται FWD. Αυτός επιλέγει για τον τελεστή A, αν η τιμή του θα είναι η τιμή ενός καταχωρητή ή το αποτέλεσμα της προηγούμενης εντολής.

Ο τελεστής B, όπως και ο A, μπορεί να πάρει και άλλες τιμές εκτός από καταχωρητή. Εδώ ωστόσο η επιλογή γίνεται μεταξύ μίας σταθεράς και της τιμής του καταχωρητή. Η τιμής της σταθεράς είναι μήκους 4bit εκτός και αν ακολουθεί εντολή imm. Σε αυτή την περίπτωση διαβάζονται τα 12bit από την εντολή imm, και αποθηκεύονται στις θέσεις $B_{15:4}$, και κατά την αποκωδικοποίηση της κανονικής εντολής διαβάζονται τα υπόλοιπα 4bit και αποθηκεύονται στις θέσεις $B_{3:0}$.

3.6.5 Μονάδα αριθμητικής και λογικής (ALU)

Η αριθμητική και λογική μονάδα του επεξεργαστή αποτελείται από έναν αθροιστή - αφαιρέτη μήκους 16bit, και μία λογική μονάδα μήκους 16bit που μπορεί να ενεργεί ταυτόχρονα πάνω στις A και B εισόδους της μονάδας.

Το τμήμα της λογικής της μονάδας μπορεί και υπολογίζει το αποτέλεσμα των πράξεων ‘Η, ΚΑΙ, Αποκλειστικό ‘Η, Όχι Και μεταξύ των τελεστών Α και Β. Το αποτέλεσμα είναι μήκους 16bit.

Το τμήμα της αριθμητικής μπορεί και προσθέτει τον τελεστή Β στον Α, ή αφαιρεί τον τελεστή Β από τον Α, έχει μία θύρα εισόδου για το κρατουμένου ψηφίο, ενώ έχει θύρα εξόδου υπερχείλισης του αποτελέσματος και για το κρατουμένου ψηφίο.

3.6.6 Καταχωρητές ολίσθησης

Ο καταχωρητής ολίσθησης δημιουργεί το αποτέλεσμα της πράξης A<<1 ή A>>1. Αυτή η πράξη δε χρειάζεται κύκλωμα λογικής, χρειάζεται μόνο ένας πολυπλέκτης, που ανάλογα με το αν η ολίσθηση πρέπει να γίνει αριστερά ή δεξιά, να επιλέγει το SRI || $A_{15:1}$ ή το $A_{14:0}$ || 0. Το σήμα SRI δείχνει εάν η ολίσθηση θα είναι αριθμητική ή λογική.

3.6.7 Πολυπλέκτης επιλογής αποτελέσματος

Ο πολυπλέκτης επιλογής αποτελέσματος πρέπει να μπορεί να επιλέγει το αποτέλεσμα της εκτέλεσης της εντολής ανάμεσα στις μονάδες του αθροιστή, του καταχωρητή ολίσθησης, της λογικής μονάδας, της θύρας ανάγνωσης δεδομένων από τη μνήμη ή της διεύθυνσης επιστροφής. Τα παραπάνω δημιουργούν 7 διαφορετικές εισόδους δεδομένων, έτσι ο πολυπλέκτης αυτός είναι μήκους 16bit και μπορεί να επιλέξει ανάμεσα σε αυτές.

Σε κάθε κύκλο ρολογιού η μονάδα ελέγχου επιλέγει κάποια από της πηγές των δεδομένων του πολυπλέκτη και η τιμή της εμφανίζεται στη γραμμή $RESULT_{15:0}$, που διατρέχει όλη την FPGA.

3.6.8 Μονάδα διεύθυνσης και μετρητή προγράμματος

Η μονάδα αυτή είναι υπεύθυνη για τη δημιουργία όλων των διευθύνσεων του επεξεργαστή, δημιουργεί τις διευθύνσεις ανάγνωσης εντολών, αποθήκευσης και ανάκτησης δεδομένων και για τους κύκλους άμεσης ανάγνωσης από τη μνήμη (DMA).

Σε κάθε κύκλο αυξάνεται κατά δύο η τιμή του μετρητή προγράμματος για να διαβαστεί η επόμενη εντολή από τη μνήμη. Εάν μία εντολή άλματος πρέπει να εκτελεστεί τότε προστίθεται στον μετρητή προγράμματος η τιμή 2xdisp8. Για τις εντολές lw, lb, sw, sb, jal, call φορτώνεται στον μετρητή προγράμματος άμεσα η τιμή που προέρχεται από τη μονάδα του αθροιστή.

3.7 Λειτουργία επεξεργαστή

Πολύ σημαντικό ρόλο στη βελτίωση της ταχύτητας ενός επεξεργαστή παίζουν δύο παράγοντες. Πρώτον μια πολύ γρήγορη μνήμη «κοντά» στον επεξεργαστή, όπου αποθηκεύονται δεδομένα για εύκολη πρόσβαση, και δεύτερον συνεχή ροή δεδομένων προς την επεξεργαστική μονάδα, έτσι ώστε να μη μένει αχρησιμοποίητη η κεντρική μονάδα επεξεργασίας.

Η τεχνική που χρησιμοποιείται για να μη μένει αχρησιμοποίητη η κεντρική μονάδα επεξεργασίας είναι η τεχνική του pipeline. Σύμφωνα με αυτή την τεχνική η μεταφορά των δεδομένων από τη μνήμη προς τον επεξεργαστή και η ενδιάμεση επεξεργασία που χρειάζεται γίνεται σε περισσότερο από ένα επίπεδα. Με αυτό τον τρόπο είναι δυνατόν να υπάρχουν πολλές εντολές προς εκτέλεση, καθεμιά από τις οποίες βρίσκεται σε διαφορετικό

στάδιο. Έτσι είναι σίγουρο ότι σε κάθε παλμό θα υπάρχει πάντα μία εντολή προς εκτέλεση.

3.7.1 Pipeline του XR16

Στον επεξεργαστή που εξετάζουμε υλοποιείται μία τεχνική pipeline με τρία επίπεδα. Το πρώτο επίπεδο είναι αυτό της ανάκτησης της εκτελούμενης εντολής (Instruction Fetch: IF), στο δεύτερο επίπεδο έχουμε την αποκωδικοποίηση της εντολής και την ανάγνωση των τιμών των καταχωρητών που χρειαζόμαστε για την εκτέλεση της (Decode: DC). Τέλος στο τελευταίο επίπεδο εκτελείται πλέον η εντολή (Execute: EX).

Το κάθε επίπεδο του pipeline κάνει μία συγκεκριμένη εργασία. Αναλυτικά στο πρώτο επίπεδο διαβάζεται η θέση μνήμης που δείχνει ο μετρητής προγράμματος. Μεταφέρεται όλη η εντολή που διαβάζεται σε έναν ενδιάμεσο καταχωρητή, που ονομάζεται Καταχωρητής Εντολών (Instruction Register: IR), και αυξάνεται ο μετρητής προγράμματος κατά δύο για να διαβαστεί η σωστή εντολή στον επόμενο κύκλο.

Στο δεύτερο στάδιο η εντολή που έχει διαβαστεί αποκωδικοποιείται. Οι τελεστές που χρειάζονται για την εκτέλεση της διαβάζονται από τον κατάλογο των καταχωρητών, ενώ αν χρειάζεται κάποια σταθερή τιμή αυτή διαβάζεται από τον Καταχωρητή Προγράμματος.

Στο τρίτο και τελευταίο επίπεδο οι μονάδες επεξεργασίας του επεξεργαστή ενεργούν πάνω στους τελεστές. Το αποτέλεσμα της εντολής οδηγείται μέσω buffer τριών καταστάσεων στο δίαυλο αποτελέσματος, και γράφεται στον αντίστοιχο καταχωρητή στο τέλος του κύκλου.

3.7.2 Προβλήματα λειτουργίας pipeline

Ωστόσο η λειτουργία του pipeline δεν είναι χωρίς προβλήματα. Εφόσον η λειτουργία του pipeline στηρίζεται στο ότι τα δεδομένα έρχονται συνεχόμενα, θεωρείται ότι αυτά διαβάζονται και συνεχόμενα. Άρα όταν πρέπει ο επεξεργαστής να εκτελέσει μία εντολή διαφορετική από την επόμενη (περίπτωση ενός άλματος), τότε η λειτουργία του pipeline «διακόπτεται». Αυτό οφείλεται στο γεγονός ότι εκτελώντας την εντολή άλματος ο επεξεργαστής θα φορτώσει ακόμη δύο εντολές, μέχρι να την εκτελεστεί. Προφανώς αυτό είναι λάθος! Θα φορτωθούν οι δύο επόμενες εντολές της εντολής άλματος προς εκτέλεση, ενώ είναι πιθανόν να πρέπει να εκτελεστούν κάποιες άλλες.

Ένα ακόμη πιθανό πρόβλημα είναι αυτό της διάδοσης του αποτελέσματος. Το πρόβλημα προκύπτει, όταν μία εντολή χρειάζεται σαν όρισμα το αποτέλεσμα της προηγούμενης, το οποίο δεν έχει προλάβει να γραφτεί ακόμη στους καταχωρητές, αφού εκτελούνται με διαφορά ενός κύκλου.

Τα δύο παραπάνω προβλήματα είναι πολύ σημαντικά και έχουν επιλυθεί στο σχεδιασμό του XR16, ενώ επίσης κάποια ακόμη προβλήματα αναλύονται στη συνέχεια.

3.7.3 Πρόβλημα διάδοσης αποτελέσματος

Για την επίλυση αυτού του προβλήματος υπάρχουν διάφορες τεχνικές. Μπορεί να λυθεί σε επίπεδο μεταφραστή με αντιμετάθεση των εντολών ή σε επίπεδο επεξεργαστή καθυστερώντας για ένα κύκλο ρολογιού στη λειτουργία του pipeline. Αυτές ωστόσο οι λύσεις προκαλούν καθυστερήσεις στον επεξεργαστή. Στον XR16 υλοποιείται μία διαφορετική λύση, υπάρχει ένας πολυπλέκτης μεγέθους 16bit 2-1, με όνομα FWD, ο οποίος επιλέγει μεταξύ του καταχωρητή που χρειάζεται ο τελεστής A και του διαύλου

αποτελέσματος. Η τεχνική αυτή ονομάζεται παράκαμψη αρχείου καταχωρητών. Έτσι όταν μία εντολή χρειάζεται ως όρισμα το αποτέλεσμα της προηγούμενης, το διαβάζει κατευθείαν από το δίαυλο αποτελεσμάτων και όχι από τον καταχωρητή. Με αυτό τον τρόπο διορθώνεται το πρόβλημα, χωρίς να προκαλούνται επιπλέον καθυστερήσεις.

Εάν δει κανείς τον τρόπο λύσης τους προβλήματος σε επίπεδο υλικού παρατηρεί ότι αυξάνεται το μήκος του datapath, δηλαδή της διαδρομής που κάνουν τα δεδομένα. Παρόλα αυτά φαίνεται ότι δεν προκαλεί καθυστερήσεις στην λειτουργία του pipeline, οπότε θεωρείται αποδοτική αυτή η λύση.

3.7.3.1 Πρόβλημα ανάγνωσης και αποθήκευσης δεδομένων

Η ανάγνωση και η αποθήκευση των δεδομένων χρειάζεται έναν επιπλέον κύκλο πρόσβασης στη μνήμη. Αυτό δημιουργεί πρόβλημα, αφού θεωρείται κατά τη λειτουργία του pipeline, ότι κάθε εντολή διαρκεί ένα κύκλο ρολογιού. Το πρόβλημα προκύπτει όταν σε ένα κύκλο ζητηθεί να εκτελεστεί μία εντολή αποθήκευσης ή ανάγνωσης από τη μνήμη. Στο ίδιο κύκλο που εκτελείται η εντολή, λόγω του pipeline, φορτώνεται και η επόμενη εντολή ωστόσο αυτή δεν πρέπει να οδηγηθεί στο επόμενο στάδιο του pipeline στον επόμενο κύκλο. Αυτό θα είναι σφάλμα αφού ακόμη δε θα έχει τελειώσει η εντολή πρόσβασης στη μνήμη. Μία προφανής λύση είναι να σταματάει η λειτουργία του pipeline για ένα κύκλο, ωστόσο αυτό θα έδινε ένα χρόνο τριών κύκλων ρολογιού για οποιαδήποτε πρόσβαση που θα έπρεπε να γίνει στη μνήμη, πράγμα αδιανόητο.

Για την επίλυση του παραπάνω προβλήματος χωρίς δημιουργία καθυστερήσεων, υπάρχει ένα επιπλέον καταχωρητής μήκους 16bit, με όνομα NEXTIR, και ένας πολυπλέκτης για τον καταχωρητή IR, με όνομα IRMUX. Όταν παρατηρείται ένα τέτοιο πρόβλημα τότε η

εντολή που διαβάζεται αποθηκεύεται προσωρινά στον NEXTIR, εκτελείται κανονικά η εντολή πρόσβασης στη μνήμη και όταν πρέπει να εκτελεστεί η επόμενη εντολή αυτή επιλέγεται μέσω του IRMUX, από τον NEXTIR και όχι από τον IR, που προφανώς έχει λανθασμένη τιμή.

3.7.3.2 Πρόβλημα λανθασμένης ανάγνωσης εντολών

Σε αντίθεση με το πρόβλημα της διάδοσης δεδομένων, σε αυτό το πρόβλημα δεν έχει υιοθετηθεί μια λύση που να αποτρέπει τις καθυστερήσεις. Για την εκτέλεση του κώδικα θεωρείται πάντα ότι το άλμα δε θα γίνει, έτσι πάντα διαβάζονται οι επόμενες εντολές. Για την επίλυση του προβλήματος, όταν πρέπει να γίνει ένα άλμα τότε απλά διαγράφονται οι επόμενες δύο εντολές από το pipeline, αυτό γίνεται ακυρώνοντας την εκτέλεση των δύο εντολών. Αυτός είναι ο λόγος πού μία εντολή άλματος διαρκεί τρεις κύκλους ρολογιού, όταν «παίρνεται» το άλμα.

3.7.3.3 Πρόβλημα εκτέλεσης διακοπών

Όταν εκτελείται κάποια ρουτίνα διακοπής, τότε πρέπει ο επεξεργαστής να εκτελέσει τον κώδικα που βρίσκεται στη θέση του χειριστή της διακοπής, να σωθεί η διεύθυνση επιστροφής, να διαγραφούν τα δεδομένα μέσα από το pipeline, να εκτελεστεί ο χειριστής της διακοπής και τελικά να επιστρέψει ο επεξεργαστής στην εντολή όπου πυροδοτήθηκε η διακοπή.

Αυτό υλοποιείται με ένα απλό τρόπο, όταν ενεργοποιείται το σήμα INTREQ, αντικαθίσταται η επόμενη εντολή με την εντολή int (jal r14,10(r0)), κάνοντας χρήση του IRMUX. Έτσι ο μετρητής προγράμματος πηγαίνει στη θέση 0x0010, όπου βρίσκεται η ρουτίνα διακοπής, ενώ η διεύθυνση επιστροφής μένει στον καταχωρητή r14. Όταν

τελειώσει η ρουτίνα διακοπής τότε εκτελείται η εντολή iret (jal r0,0(r14)) και ο επεξεργαστής συνεχίζει από τη διεύθυνση που έχει αποθηκευτεί στον r14.

Από την εκτέλεση διακοπών προκαλούνται δύο προβλήματα στο pipeline. Πρώτον δεν πρέπει να διακοπεί η εκτέλεση μίας αλληλουχίας εντολών όπου η μία εξαρτάται από την άλλη. Αυτές οι εντολές είναι οι add, sub, shift ή imm, ακολουθούμενες από κάποια άλλη εντολή, εάν αυτό συμβεί τότε απλά προστίθεται ένας κενός κύκλος στη λειτουργία του pipeline.

Επίσης δεν πρέπει να πυροδοτηθεί μία διακοπή όταν εκτελείται κάποια εντολή άλματος. Όταν συμβεί αυτό τότε η εντολή της διακοπής μεταφέρεται σε επόμενο κύκλο.

Η απλότητα ωστόσο στο σχεδιασμό του τρόπου εκτέλεσης των διακοπών του επεξεργαστή έχει και τα αρνητικά της. Αυτό είναι ο χρόνος εκτέλεσης μιας διακοπής, να διαρκεί τουλάχιστον έξι κύκλους ρολογιού.

3.8 Μονάδα ελέγχου

Η μονάδα ελέγχου του επεξεργαστή αποτελείται από τα παρακάτω τμήματα

- Τη μηχανή καταστάσεων για τον έλεγχο
- Τους καταχωρητές για τις εντολές (next, decode state, execute stage)
- Τον αποκωδικοποιητή εντολών
- Τον επιλογέα των τελεστών της εκτελούμενης εντολής (ανάμεσα στους καταχωρητές, της προώθησης του κρατουμένου και σταθερές τιμές)
- Τη βαθμίδα εκτέλεσης (Μονάδα αριθμητικής και λογικής, πολυπλέκτης για την έξοδο του αποτελέσματος)

3.8.1 Εκτέλεση εντολής

Για να εκτελεστεί μία εντολή περνάει από διάφορα στάδια μέσα από τη βαθμίδα ελέγχου. Αυτά τα στάδια είναι τρία, όσα και τα επίπεδα του pipeline, και περιγράφονται στη συνέχεια.

Πρώτο στάδιο είναι αυτό της ανάκτησης της εκτελούμενης εντολής, ονομάζεται IF (instruction fetch). Όταν έχει διαβαστεί η νέα εντολή μεταφέρεται στον καταχωρητή IR.

Δεύτερο στάδιο είναι η αποκωδικοποίηση της εντολής, ονομάζεται DC (decode). Σε αυτό το στάδιο η εντολή αποκωδικοποιείται σε εσωτερικά σήματα ελέγχου. Στο πρώτο μισό του κύκλου ρολογιού, η μονάδα επιλέγει τους καταχωρητές που θα πρέπει να χρησιμοποιηθούν και ελέγχει τους καταχωρητές προώθησης κρατουμένου και σταθερής τιμής, για να χρησιμοποιηθούν στην εντολή. Επίσης σε αυτό το στάδιο, εάν υπάρχει εντολή άλματος, ελέγχεται εάν το άλμα θα πραγματοποιηθεί ή όχι.

Τελευταίο στάδιο είναι αυτό της εκτέλεσης της εντολής. Η μονάδα ελέγχου οδηγεί τη μονάδα αριθμητικής και λογικής για να πάρουμε το ζητούμενο αποτέλεσμα. Επίσης ελέγχεται και ο πολυπλέκτης εξόδου για να μεταφερθεί το αποτέλεσμα στη σωστή θέση. Εάν η εντολή είναι αποθήκευσης ή ανάκτησης δεδομένων από τη μνήμη, τότε σε αυτό το στάδιο προστίθεται το επιπλέον στάδιο της πρόσβασης στη μνήμη. Τέλος στο τελευταίο μισό του κύκλου του ρολογιού, επιλέγονται και στους δύο καταλόγους καταχωρητών, εκείνοι οι καταχωρητές που πρέπει να γραφτεί το αποτέλεσμα.

4

Αλγόριθμοι χρονοδρομολόγησης

4.1 Περιγραφή

Στα σημερινά υπολογιστικά συστήματα υπάρχει η ανάγκη να εκτελούνται πολλές εφαρμογές «παράλληλα», ενώ υπάρχει ένας και μοναδικός επεξεργαστής. Το πρόβλημα αυτό έρχονται να το λύσουν κάποιες τεχνικές, με τις οποίες πολλές εφαρμογές μπορούν να μοιράζονται έναν επεξεργαστή.

4.2 Ιδιότητες εφαρμογών - Θεωρία

Οι παραπάνω τεχνικές στηρίζονται στο γεγονός, ότι μία εφαρμογή σπάνια χρησιμοποιεί συνέχεια τον επεξεργαστή για μεγάλο χρονικό διάστημα, έτσι υπάρχουν κενά παύσης της λειτουργίας του επεξεργαστή. Σε αυτά τα κενά μπορεί να φορτωθεί μία άλλη εφαρμογή.

4.3 Πραγματικότητα

Η πραγματικότητα από τα παραπάνω, είναι λίγο διαφορετική ωστόσο. Οι εφαρμογές, αν και δεν χρειάζεται να τρέχουν συνέχεια αν έπρεπε να περιμένει κανείς κάποια να τελειώσει για να εκτελεστεί κάποια άλλη, θα φαινόταν ότι το σύστημα «κρεμάει» ανά τακτά χρονικά διαστήματα. Το λειτουργικό σύστημα, γι' αυτό το λόγο, ανά κάποιο χρονικό διάστημα αναλαμβάνει να διακόψει τη λειτουργία μίας εφαρμογής και να φορτώσει κάποια άλλη. Η διαδικασία γίνεται πολύ γρήγορα και σε πολύ τακτικά χρονικά διαστήματα έτσι ώστε να υπάρχει η ψευδαίσθηση ότι οι εφαρμογές τρέχουν παράλληλα.

4.4 Εφαρμογές ελέγχου και παρακολούθησης

Ο παραπάνω τρόπος εναλλαγής των εφαρμογών σε έναν επεξεργαστή, παρόλο που δίνει τη δυνατότητα να τρέχουν πολλαπλές εφαρμογές ταυτόχρονα, οι εφαρμογές δεν ελέγχονται για το πότε θα τρέχουν και πότε θα σταματάνε. Αυτό το μειονέκτημα αυτόματα δεν αφήνει περιθώρια χρήσης ενός τέτοιου τρόπου σε εφαρμογές ελέγχου και παρακολούθησης.

Στις εφαρμογές όπου ένας ηλεκτρονικός υπολογιστής χρησιμοποιείται για να ελέγχεται ένα μηχανικό σύστημα ή να παρακολουθεί τις ιδιότητες κάποιας συσκευής, είναι αναγκαίο συγκεκριμένες εφαρμογές να τρέχουν σε συγκεκριμένα χρονικά σημεία. Για παράδειγμα ας θεωρηθεί ότι υπάρχει ένας ηλεκτρονικός υπολογιστής που ελέγχει έναν πυρηνικό αντιδραστήρα, ο υπολογιστής μετράει τη θερμοκρασία και σε περίπτωση υπερθέρμανσης κλείνει ορισμένες βαλβίδες. Οι δύο εφαρμογές που περιγράφθηκαν, οφείλουν να τρέχουν σε τακτά χρονικά διαστήματα η πρώτη, ενώ η δεύτερη σε περίπτωση προβλήματος να εκτελείται ακαριαία. Γίνεται άμεσα αντιληπτό ότι αυτή την εφαρμογή δεν μπορεί να την

ελέγξει ένα λειτουργικό σύστημα που χρησιμοποιεί τον πρώτο αλγόριθμο. Αυτό το σύστημα μπορεί να ελεγχθεί μόνο από ένα *λειτουργικό σύστημα πραγματικού χρόνου*.

4.5 Ορισμός

Ένας αλγόριθμος χρονοδρομολόγησης είναι ένα σύνολο από κανόνες, βάση των οποίων καθορίζεται ποια διεργασία πρέπει να εκτελείται σε μία συγκεκριμένη στιγμή.

4.6 Λειτουργικά συστήματα πραγματικού χρόνου

Ένα λειτουργικό σύστημα πραγματικού χρόνου έχει την ιδιότητα να αλλάζει την ενεργή εφαρμογή ανάλογα με το πότε και αν πρέπει να τρέξει αυτή ή κάποια άλλη και όχι όταν αποφασίζει το ίδιο. Κάθε εφαρμογή που τρέχει σε ένα τέτοιο σύστημα έχει μία ιδιότητα που ονομάζεται προτεραιότητα, βάση της οποίας αλλάζουν οι διεργασίες. Υπάρχουν δύο αλγόριθμοι βάση των οποίων δουλεύει ένα λειτουργικό σύστημα πραγματικού χρόνου και οι δύο έχουν την ιδιότητα να αλλάζουν την ενεργή εφαρμογή εάν μία με υψηλότερη προτεραιότητα ζητήσει να κάνει χρήση του επεξεργαστή. Επίσης και οι δύο αλγόριθμοι εκτελούν κάποια διεργασία ανάλογα με το αν ζητηθεί να εκτελεστεί (preemptive). Το σημείο που διαφέρουν είναι ότι ο μεν πρώτος για κάθε μία εφαρμογή έχει μία σταθερή τιμή προτεραιότητας, ενώ ο δεύτερος μπορεί και μεταβάλει αυτές τις τιμές, καθώς εκτελείται το πρόγραμμα. Έτσι ο δεύτερος μπορεί και επιτυγχάνει πλήρη χρησιμοποίηση της επεξεργαστικής ισχύος, ενώ ο πρώτος μπορεί να φτάσει σε ένα ποσοστό της τάξης του 70%. Σε αυτή την εργασία θα αναλυθεί ο πρώτος αλγόριθμος, που είναι και αυτός που έχει υλοποιηθεί. Ένας αλγόριθμος με σταθερή ανάθεση προτεραιοτήτων ονομάζεται Rate Monotonic Scheduling Algorithm ή RMS.

Επίσης μία ιδιότητα που έχουν ορισμένα λειτουργικά συστήματα αυτού του τύπου, είναι η ολοκλήρωση μίας διεργασίας μέσα σε ένα συγκεκριμένο χρονικό διάστημα, από τη στιγμή που ξεκινάει. Με αυτό τον τρόπο είναι σίγουρο ότι θα έχει ολοκληρώσει τη λειτουργία της πριν χρειαστεί να καλεστεί ξανά. Όταν πληρείται αυτή η προϋπόθεση το λειτουργικό σύστημα ονομάζεται «αυστηρά πραγματικού χρόνου». Στην εργασία αυτή υλοποιείται ένας αλγόριθμος, έτσι ώστε να πληρείται αυτή η προϋπόθεση.

4.7 Αλγόριθμος με σταθερή απόδοση προτεραιοτήτων

Στον αλγόριθμο αυτό κάθε διεργασία έχει μία σταθερή προτεραιότητα η οποία έχει αποδοθεί πριν ξεκινήσει τη λειτουργία του το σύστημα. Κάθε διεργασία έχει δύο βασικές ιδιότητες, η πρώτη είναι η τιμή της προτεραιότητας της, ενώ η δεύτερη είναι το διάστημα που μεσολαβεί από τη στιγμή που θα γίνει μία κλήση της διεργασίας μέχρι την επόμενη.

4.7.1 Περιβάλλον ανάπτυξης

Για να μελετηθεί ο παραπάνω αλγόριθμος χρειάζεται ένα θεωρητικό περιβάλλον. Για το συγκεκριμένο θα γίνουν ορισμένες παραδοχές έτσι ώστε να είναι εφικτή η μελέτη των ιδιοτήτων του αλγορίθμου. Οι παραδοχές που πρέπει να γίνουν είναι τέσσερεις

Π1 Τα request που κάνουν όλες οι διεργασίες, για τις οποίες ισχύουν αυστηρά χρονικά περιθώρια ολοκλήρωσης είναι περιοδικά και σταθερά ανάμεσα σε κάθε ζήτηση.

Π2 Η έννοια χρονικό περιθώριο ολοκλήρωσης αναφέρεται μόνο σε χαρακτηριστικά εκτέλεσης της διεργασίας, π.χ. κάθε διεργασία πρέπει να έχει τελειώσει πριν χρειαστεί να εκτελεστεί ξανά.

Π3 Οι διεργασίες μεταξύ τους είναι ανεξάρτητες στον τρόπο με τον οποίο ζητάνε να εκτελεστούν, έτσι μία διεργασία για να εκτελεστεί δεν εξαρτάται από το πότε θα τελειώσει κάποια άλλη.

Π4 Ο χρόνος εκτέλεσης της κάθε διεργασίας είναι σταθερός για κάθε διεργασία και δεν αλλάζει κατά την εκτέλεση του προγράμματος. Με την έννοια χρόνο εκτέλεσης εννοείται ο χρόνος που χρειάζεται ο επεξεργαστής για να ολοκληρώσει τη διεργασία.

Π5 Διεργασίες που δεν είναι περιοδικές, θεωρούνται ειδικές διεργασίες, όπως αυτές της ανάκαμψης του συστήματος από σφάλματα. Όταν χρειαστεί να τρέξουν εκτελούνται πάντα, παίρνοντας τη θέση της ενεργής διεργασίας και δεν έχουν συγκεκριμένο σταθερό χρόνο εκτέλεσης.

4.7.2 Περιγραφή διεργασίας

Με τις παραπάνω παραδοχές που έχουν γίνει για το περιβάλλον ανάπτυξης, υπάρχει η δυνατότητα πλήρους χαρακτηρισμού μία διεργασίας με δύο αριθμούς μόνο, τη χρονική περίοδο κλήσης της διεργασίας και του χρόνου που χρειάζεται για να εκτελεστεί. Στη συνέχεια θα χρησιμοποιηθούν οι παρακάτω συμβολισμοί για τις διεργασίες και τους δύο χρόνους, που αναφέρθηκαν παραπάνω, για να αναλυθεί ο τρόπος λειτουργίας του αλγορίθμου. Έστω ότι υπάρχουν *n* διεργασίες $\tau_1,\tau_2,..,\tau_n$, οι χρονικές περιόδοι κλήσης τους θα συμβολιστούν με $T_1,T_2,\ldots,T_n$, τέλος οι χρόνοι εκτέλεσης συμβολίζονται με $C_1,C_2,\ldots,C_n$. Θεωρείται ότι ο ρυθμός κλήσης μίας διεργασίας είναι ο ανάστροφος της χρονικής περιόδου κλήσης.

4.7.3 Ανάλυση αλγορίθμου

Η ανάλυση του αλγορίθμου προϋποθέτει των ορισμό συγκεκριμένων ιδιοτήτων, βάσει αυτών θα γίνει η περιγραφή των χαρακτηριστικών του. Ορίζεται ως το τέλος της αίτησης για την ανάκτηση του επεξεργαστή από μία διεργασία ο χρόνος μέχρι την επόμενη αίτηση για την ίδια διεργασία. Για ένα σύνολο από διεργασίες θεωρείται ότι υπάρχει υπερχείλιση του αλγορίθμου σε κάποια χρονική στιγμή ***t***, εάν ***t*** είναι ο χρόνος τέλους για μία διεργασία που δεν έχει εκτελεστεί. Ένας αλγόριθμος χρονοδρομολόγησης είναι εφικτός εάν για ένα σύνολο διεργασιών δεν υπάρχουν ποτέ υπερχειλίσεις. Επίσης ορίζεται ως χρόνος απόκρισης, το χρονικό διάστημα από τη στιγμή που γίνεται αίτηση από μία διεργασία μέχρι τη στιγμή που αυτή εκτελείται. Μία ακόμη βασική ιδιότητα μιας διεργασίας είναι αυτή της *κρίσιμης στιγμής*, ως κρίσιμη στιγμή ορίζεται η στιγμή κατά την οποία η αίτηση μίας διεργασίας έχει το μεγαλύτερο χρόνο απόκρισης. Τέλος ένα κρίσιμο χρονικό διάστημα ορίζεται ως το χρονικό διάστημα από μία κρίσιμη στιγμή, έως την εκτέλεση της αίτησης μίας διεργασίας. Για την κρίσιμη χρονική στιγμή μίας διεργασίας υπάρχει το παρακάτω θεώρημα

4.7.3.1 Θεώρημα 1

Σε μία διεργασία συμβαίνει μία κρίσιμη χρονική στιγμή, όταν η αίτηση για εκτέλεση της συμβαίνει ταυτόχρονα με αιτήσεις διεργασιών με υψηλότερες προτεραιότητες.

<u>Απόδειξη θεωρήματος</u>

Έστω τα $\tau_1,\tau_2,..,\tau_m$ χαρακτηρίζουν ένα σύνολο από διεργασίες ταξινομημένες βάση των προτεραιοτήτων τους, με την τ_m να έχει τη μικρότερη. Έστω ότι τη χρονική στιγμή t_1 η τ_m κάνει μία αίτηση. Ανάμεσα στη χρονική στιγμή t_1 και t_1+T_m, χρονική στιγμή που η

επόμενη αίτηση για τη διεργασία m συμβαίνει, όλες οι διεργασίες με μεγαλύτερη διεργασία και μικρότερο χρόνο T κάνουν τις δικές τους αιτήσεις. Είναι άμεσα αντιληπτό ότι ο χρόνος εκτέλεσης της διεργασίας θα αυξηθεί, αφού κάποια διεργασία με υψηλότερη προτεραιότητα, έστω η τ_1, κάνει αίτηση, άρα θα χρησιμοποιήσει εκείνη τον επεξεργαστή. Αυτό δημιουργεί μία καθυστέρηση ακόμη μεγαλύτερη εάν η διεργασία με αμέσως μικρότερη προτεραιότητα από την τ_1 αλλά μεγαλύτερη από την τ_m, έστω η τ_2, κάνει αίτηση αμέσως μετά το τέλος της t_1. Παρατηρείται ωστόσο ότι ακόμη και εάν μεταφερθεί η χρονική στιγμή που κάνει αίτηση η τ_2, δεν θα αλλάξει ο συνολικός χρόνος εκτέλεσης της τ_m. Εάν η παραπάνω διαδικασία επαναληφθεί και για τ_i, $i<m$, τότε το θεώρημα αποδεικνύεται.

Το αποτέλεσμα του παραπάνω θεωρήματος είναι ότι ένας απλός υπολογισμός των δοσμένων τιμών για τις προτεραιότητες των διεργασιών οδηγούν στο ένα ο προτεινόμενος αλγόριθμος είναι εφικτός. Συγκεκριμένα εάν τα κρίσιμα σημεία όλων των διεργασιών ολοκληρωθούν πριν τελειώσει ο διαθέσιμος χρόνος αναμονής, τότε ο αλγόριθμος θεωρείται εφικτός.

Έστω ότι υπάρχουν δύο διεργασίες, τ_1 και τ_2, που πρέπει να τοποθετηθούν σε μία εφικτή σειρά εκτέλεσης. Έστω ότι οι δύο διεργασίες έχουν ρυθμούς αίτησης T_1 και T_2 με την T_1 μικρότερη της T_2. Εάν τεθεί στην τ_1 υψηλότερη προτεραιότητα, τότε σύμφωνα με το πρώτο θεώρημα θα πρέπει να ισχύει η παρακάτω ανισότητα:

$$\lfloor T_2/T_1 \rfloor C_1 + C_2 \le T_2$$

Εάν τώρα τεθεί στην τ_2 υψηλότερη προτεραιότητα τότε η παρακάτω ανισότητα πρέπει να ισχύει

$$C_1 + C_2 \le T_1$$

Αφού

$$\lfloor T_2/T_1 \rfloor C_1 + \lfloor T_2/T_1 \rfloor C_2 \le \lfloor T_2/T_1 \rfloor T_1 \le T_2$$

Από τα παραπάνω παρατηρείται ότι όταν $T_1<T_2$ και τα C_1,C_2 είναι τέτοια ώστε ο αλγόριθμος να είναι εφικτός με την τ_2 να έχει υψηλότερη προτεραιότητα από την τ_1, είναι επίσης εφικτός με την τ_1 να έχει υψηλότερη προτεραιότητα από την τ_2, το αντίθετο ωστόσο δεν ισχύει, άρα πρέπει να αποδοθεί υψηλότερη προτεραιότητα στην τ_1 και χαμηλότερη στην τ_2. Ένα συμπέρασμα που προκύπτει από τα προηγούμενα είναι ότι ένας λογικός τρόπος απόδοσης προτεραιοτήτων είναι ανάλογα με το ρυθμό αιτήσεων των διεργασιών ανεξάρτητα από το χρόνο εκτέλεσης τους, έτσι ώστε οι διεργασίες με του υψηλότερους ρυθμούς να έχουν και τις υψηλότερες προτεραιότητες. Ένας τρόπος απόδοσης προτεραιοτήτων όπως ο προηγούμενος ονομάζεται “Rate Monotonic Priority Assignment ή RMS”. Επομένως όπως αντιλαμβάνεται κανείς από τα προηγούμενα ένας τέτοιος τρόπος απόδοσης είναι ο καλύτερος, με την έννοια ότι κανείς άλλος αλγόριθμος με σταθερές προτεραιότητες δεν μπορεί να δρομολογήσει ένα σύνολο διεργασιών που να μην μπορεί να δρομολογηθεί από με τον “rate-monotonic”.

4.7.3.2 Θεώρημα 2

Εάν για ένα σύνολο διεργασιών υπάρχει ένας εφικτός τρόπος ανάθεσης προτεραιοτήτων, τότε η ανάθεση κατά “rate-monotonic” είναι εφικτή.

Απόδειξη θεωρήματος

Έστω $\tau_1,\tau_2,\ldots,\tau_m$ ένα σύνολο από διεργασίες με μία εφικτή ανάθεση προτεραιοτήτων. Έστω τ_i, τ_j δύο διεργασίες με την παραπάνω ανάθεση προτεραιοτήτων και η τ_i έχει μεγαλύτερη προτεραιότητα. Έστω ότι $T_i > T_j$. Έστω ότι άλλαζε η μεταξύ τους η

προτεραιότητα για τις δύο διεργασίες. Γίνεται αντιληπτό ότι η νέα ανάθεση προτεραιοτήτων δίνει επίσης έναν εφικτό αλγόριθμο. Εφόσον η ανάθεση προτεραιοτήτων κατά “rate-monotonic” μπορεί να γίνει με οποιαδήποτε ανάθεση προτεραιοτήτων με αμοιβαία αλλαγή σε ζεύγη διεργασιών, όπως παραπάνω, το θεώρημα αποδεικνύεται.

4.7.4 Μέγιστη χρήση επεξεργαστικής μονάδας

Με αυτά που έχουν ειπωθεί παραπάνω είναι δυνατόν να υπολογιστεί ένα μέγιστο επίπεδο χρησιμοποίησης της επεξεργαστικής μονάδας. Ορίζεται ως χρησιμοποίηση της επεξεργαστικής μονάδας το ποσοστό εκείνο του χρόνου που χρειάστηκε για να ολοκληρωθεί η εκτέλεση του συνόλου των διεργασιών, με ένα άλλο τρόπο η χρησιμοποίηση ορίζεται ως ένα μείον το χρόνο που η επεξεργαστική μονάδα μένει αχρησιμοποίητη. Εφόσον για μία διεργασία το ποσοστό της επεξεργαστικής ισχύος που χρειάζεται είναι C_i/T_i για την εκτέλεση της διεργασίας τ_i, για το σύνολο των m διεργασιών η χρησιμοποίηση του επεξεργαστή θα είναι

$$U = \sum_{i=1}^{m} (C_i / T_i)$$

Αν και ο παραπάνω αριθμός μπορεί να βελτιωθεί εάν αυξηθούν τα C_i ή μειωθούν τα T_i, υπάρχει ένα άνω όριο από την ανάγκη όλες οι διεργασίες να έχουν εκτελεστεί μέχρι το τέλος των κρίσιμων στιγμών τους.

Εφόσον δεν είναι προϋπόθεση το πόσο μικρό μπορεί να είναι το ποσοστό χρησιμοποίησης, αλλά μόνο το πόσο μεγάλο μπορεί αυτό να γίνει, πρέπει να οριστεί τι είναι αυτό που ζητείται. Για ένα σύνολο από δοσμένες διεργασίες με κάποια ανάθεση προτεραιοτήτων, θεωρείται ότι υπάρχει μέγιστη χρησιμοποίηση της επεξεργαστικής μονάδας, εάν η ανάθεση είναι εφικτή για το σύνολο των διεργασιών και μία αύξηση στο χρόνο εκτέλεσης

κάποιας από τις διεργασίες θα καθιστούσε την παρούσα ανάθεση προτεραιοτήτων μη-εφικτή.

Σε προηγούμενη παράγραφο ειπώθηκε πως η ανάθεση προτεραιοτήτων κατά “rate-monotonic” είναι η αποδοτικότερη, άρα μπορεί κάνει τη μεγαλύτερη ή ίση χρήση της επεξεργαστικής μονάδας, από οποιαδήποτε άλλο τρόπο ανάθεσης προτεραιοτήτων.

Στη συνέχεια θα βρεθεί η μέγιστη χρησιμοποίηση για δύο διεργασίες και ο τύπος στη συνέχεια θα αναπτυχθεί για το ζητούμενο σύνολο των m διεργασιών.

4.7.4.1 Θεώρημα 3

Για ένα σύνολο δύο διεργασιών με σταθερή ανάθεση προτεραιοτήτων, το ελάχιστο ανώτερο επίπεδο χρησιμοποίησης του επεξεργαστή είναι

$$U = 2(2^{\frac{1}{2}} - 1)$$

Απόδειξη θεωρήματος

Έστω τ_1 και τ_2 είναι δύο διεργασίες με χρόνους αίτησης T_1 και T_2 και χρόνους εκτέλεσης C_1 και C_2. Έστω ότι $T_2 > T_1$. Σύμφωνα με τον “rate monotonic” η τ_1 θα έχει μεγαλύτερη προτεραιότητα από την τ_2. Σε ένα κρίσιμο διάστημα της τ_2, θα υπάρχουν $\lceil T_2/T_1 \rceil$ αιτήσεις της τ_1. Έστω τώρα ότι με το C_2 υπάρχει πλήρη χρησιμοποίηση της επεξεργαστικής ισχύος μέσα στο κρίσιμο διάστημα. Υπάρχουν δύο δυνατές περιπτώσεις

Περίπτωση 1: Ο χρόνος εκτέλεσης C_1 είναι αρκετά μικρός έτσι ώστε όλες οι αιτήσεις της τ_1 μέσα στο κρίσιμο διάστημα της τ_2 να ολοκληρώνονται πριν από μία δεύτερη αίτηση της τ_2, δηλαδή

$$C_1 \le T_2 - T_1 \lfloor T_2/T_1 \rfloor$$

Η μέγιστη τιμή που μπορεί να πάρει η C2 είναι

$$C_2 = T_2 - C_1 \lceil T_2 / T_1 \rceil$$

Η χρησιμοποίηση του επεξεργαστή που επιτυγχάνεται από τα παραπάνω θα είναι

$$U = 1 + C_1 \left[(1/T_1) - (1/T_2) \lceil T_2 / T_1 \rceil \right]$$

Σε αυτή την περίπτωση η χρησιμοποίηση του επεξεργαστή μειώνεται μονοτονικά μαζί με τη C_1.

Περίπτωση 2: Η εκτέλεση της $\lceil T_2 / T_1 \rceil$ αίτησης της τ_1, πέφτει πάνω στην αίτηση της τ_2. Σε αυτή την περίπτωση ισχύει:

$$C_1 \geq T_2 - T_1 \lfloor T_2 / T_1 \rfloor$$

Με άμεσο επακόλουθο η μέγιστη επιτρεπόμενη τιμή για την C_2 να είναι

$$C_2 = -C_1 \lfloor T_2 / T_1 \rfloor + T_1 \lfloor T_2 / T_1 \rfloor$$

και η χρησιμοποίηση του επεξεργαστή που προκύπτει είναι

$$U = (T_1 / T_2) \lfloor T_2 / T_1 \rfloor + C_1 \left[(1/T_1) - (1/T_2) \lfloor T_2 / T_1 \rfloor \right]$$

Σε αυτή την περίπτωση η χρησιμοποίηση αυξάνεται μονοτονικά μαζί με την C_1.

Η ελάχιστη χρησιμοποίηση του επεξεργαστή υφίσταται στο όριο ανάμεσα στις παραπάνω δύο περιπτώσεις. Αυτό συμβαίνει για

$$C_1 = T_2 - T1 \lfloor T_2 / T_1 \rfloor$$

και υπάρχει χρησιμοποίηση

$$U = 1 - (T_1 / T_2) \left[\lceil T_2 / T_1 \rceil - (T_2 / T_1) \right] \left[(T_2 / T_1) - \lfloor T_2 / T_1 \rfloor \right]$$

αν τεθεί $I = \lfloor T_2 / T_1 \rfloor$ και $f = \{T_3 / T_1\}$ η συνάρτηση της χρησιμοποίησης μπορεί να γραφτεί

$$U = 1 - f(1 - f)/(I + f)$$

Εφόσον η U αυξάνει μονοτονικά με την I, το ελάχιστο για την U προκύπτει στην ελάχιστη τιμή της I, δηλαδή όταν I=1. Μειώνοντας την U περισσότερο από την f, φαίνεται ότι $f = 2^{\frac{1}{2}} - 1$, άρα η U παίρνει μία ελάχιστη τιμή ίση με την f που είναι περίπου ίση με *0.83*.

Επίσης γίνεται άμεσα αντιληπτό ότι η χρησιμοποίηση γίνεται 1 εάν η f γίνει 1, δηλαδή το χρονικό διάστημα αίτησης της διεργασίας με την χαμηλή προτεραιότητα είναι πολλαπλάσιο αυτού της άλλης διεργασίας.

4.7.4.2 Θεώρημα 4

Για ένα σύνολο m διεργασιών με σταθερή ανάθεση προτεραιοτήτων και με τον περιορισμό ότι ο λόγος μεταξύ των χρόνων αίτησης δύο διεργασιών είναι μικρότερος από δύο, το ελάχιστο άνω όριο του λόγου χρησιμοποίησης του επεξεργαστή θα είναι $U = m\left(2^{\frac{1}{m}} - 1\right)$

<u>Απόδειξη θεωρήματος</u>

Έστω $\tau_1,\tau_2,\ldots,\tau_m$ δηλώνουν τις m διεργασίες που αποτελούν το σύνολό μας. Έστω $C_1,C_2,\ldots,C_m$ είναι οι χρόνοι εκτέλεσης των διεργασιών με τους οποίους πετυχαίνουν πλήρη χρησιμοποίηση του επεξεργαστή και ελαχιστοποιούν το λόγο χρησιμοποίησης. Έστω $T_m>T_{m-1}>\ldots>T_2,T_1$. Και το U δηλώνει το λόγο χρησιμοποίησης. Θέλουμε να αποδείξουμε ότι

$$C_1=T_2-T_1$$

Θεωρούμε ότι

$$C1=T2-T1+\Delta,\ \Delta>0$$

Έστω

$$C_1' = T_2 - T_1$$
$$C_2' = C_2 + \Delta$$
$$C_3' = C_3$$
$$\vdots$$
$$C'_{m-1} = C_{m-1}$$
$$C_m' = C_m.$$

Εύκολα παρατηρεί κανείς ότι και τα $C'_1, C'_2, \ldots, C'_{m-1}, C'_m$ επίσης χρησιμοποιούν πλήρως τον επεξεργαστή. Έστω U' είναι ο αντίστοιχος λόγος χρησιμοποίησης του επεξεργαστή, έχουμε

$$U - U' = (\Delta/T_1) - (\Delta/T_2) > 0.$$

Εναλλακτικά υποθέτουμε ότι

$$C_1 = T_2 - T_1 - \Delta, \qquad \Delta > 0.$$

Έστω

$$C_1'' = T_2 - T_1$$
$$C_2'' = C_2 - 2\Delta$$
$$C_3'' = C_3$$
$$\vdots$$
$$C''_{m-1} = C_{m-1}$$
$$C_m'' = C_m$$

Και σε αυτή την περίπτωση τα $C_1'', C_2'', \ldots, C''_{m-1}, C_m''$ κάνουν πλήρη χρησιμοποίηση του επεξεργαστή. Έστω U'' είναι ο αντίστοιχος λόγος χρησιμοποίησης του επεξεργαστή, θα είναι

$$U - U'' = -(\Delta/T_1) + (2\Delta/T_2) > 0.$$

Επομένως εάν *U* είναι η ελάχιστη χρησιμοποίηση τότε

$$C_1 = T_2 - T_1$$

Με παρόμοιο τρόπο μπορεί να δει κανείς ότι

$$C_2 = T_3 - T_2$$
$$C_3 = T_4 - T_3$$
$$\vdots$$
$$C_{m-1} = T_m - T_{m-1}$$

Συνεπώς

$$C_m = T_m - 2(C_1 + C_2 + \cdots + C_{m-1}).$$

Για περαιτέρω απλοποίηση της παράστασης, έστω

$$g_i = (T_m - T_i) / T_i, \qquad i = 1, 2, \cdots, m-1$$

Άρα

$$C_i = T_{i+1} - T_i = g_i T_i - g_{i+1} T_{i+1}, \; i = 1, 2, \cdots, m-1$$

και

$$C_m = T_m - 2g_1 T_1$$

τελικά καταλήγουμε στον τύπο

$$\begin{aligned} U &= \sum_{i=1}^{m} (C_i / T_i) = \sum_{i=1}^{m-1} \left[g_i - g_{i+1} (T_{i+1} / T_i) \right] + 1 - 2g_1 (T_1 / T_m) \\ &= \sum_{i=1}^{m-1} \left[g_i - g_{i+1} (g_i + 1) / (g_{i+1} + 1) \right] + 1 - 2\left[g_1 (g_1 + 1) \right] \qquad (1) \\ &= 1 + g_1 \left[(g_1 - 1) / (g_1 + 1) \right] + \sum_{i=2}^{m-1} g_i \left[(g_i - g_{i-1}) / (g_i + 1) \right] \end{aligned}$$

Παρατηρείται ότι όπως και στην περίπτωση των δύο διεργασιών, η χρησιμοποίηση γίνεται 1 εάν $g_i = 0$, για όλα τα i.

Για να βρεθεί το ελάχιστο άνω όριο του δείκτη χρησιμοποίησης, πρέπει η εξίσωση (1) να ελαχιστοποιηθεί σε σχέση με τα g_j. Αυτό μπορούμε να το πετύχουμε εάν θέσουμε την

πρώτη παράγωγο της U ίση με το 0 με άγνωστο τα g_i και επίλυση της εξίσωσης που δημιουργείται:

$$\partial U/\partial g_j = \left(g_j^2 + 2g_j - g_{j-1}\right)/\left(g_1+1\right)^2 - \left(g_{j+1}\right)/\left(g_{j+1}+1\right) = 0, \quad j = 1,2,\cdots,m-1 \qquad (2)$$

Η γενική λύση για την εξίσωση (2) αποδεικνύεται ότι είναι

$$g_j = 2^{(m-j)/m} - 1, \qquad j = 0,1,\cdots,m-1 \qquad (3)$$

Συνέπεια των παραπάνω είναι

$$U = m\left(2^{1/m} - 1\right), \qquad (4)$$

η οποία είναι η σχέση που έπρεπε να αποδειχτεί από την αρχή.

Για m = 2, η εξίσωση (3) δίνει το ίδιο όριο που βρέθηκε για ένα σύνολο δύο διεργασιών χωρίς περιορισμούς στη περιοδικότητα των αιτήσεων. Για m =3 η εξίσωση (4) δίνει αποτέλεσμα

$$U = 3(2^{1/3} - 1) \approx 0.78$$

καθώς το m μεγαλώνει η τιμή του U συγκλίνει στην τιμή $U \approx \ln 2$.

Ο περιορισμός που έχει τεθεί, σε αυτό το θεώρημα, ότι ο μέγιστος λόγος μεταξύ της περιόδου των αιτήσεων να είναι ίσος με 2 μπορεί ουσιαστικά να αναιρεθεί. Αυτό θα αποτελέσει το 5^{ο} θεώρημα

4.7.4.3 Θεώρημα 5

Για ένα σύνολο m διεργασιών με σταθερή ανάθεση προτεραιοτήτων, το ελάχιστο άνω όριο χρησιμοποίησης του επεξεργαστή είναι $U = m(2^{1/m} - 1)$.

Απόδειξη θεωρήματος

Έστω τ_1, τ_2, ··· , τ_i, ··· , τ_m ένα σύνολο από m διεργασίες που χρησιμοποιούν πλήρως τον επεξεργαστή. Έστω U ο αντίστοιχος δείκτης χρησιμοποίησης του επεξεργαστή. Θεωρούμε τώρα ότι ισχύει για κάποια i, $\lfloor T_m / T_i \rfloor > 1$. Για την ακρίβεια του λόγου, έστω $T_m = qT_i + r$, $q > 1$ και $r \geq 0$. Έστω ότι αντικαθίσταται η διεργασία τ_i με τη διεργασία τ_i' έτσι ώστε $T_i^{'} = qT_i$ και $C_i^{'} = C_i$, και αυξηθεί το C_m έτσι ώστε να επιτευχθεί πλήρη χρησιμοποίηση του επεξεργαστή. Η αύξηση να φτάσει σε μία μέγιστη τιμή ίση με $C_i(q-1)$, που ο χρόνος μέσα στο κρίσιμο διάστημα της τ_m που καταλαμβάνεται από την τ, αλλά όχι από την τ_i'. Έστω U' η χρησιμοποίηση του επεξεργαστή ενός τέτοιου συνόλου διεργασιών, τότε

$$U^{'} < U + [(q-1)C_i / T_m] + (C_i / T_i^{'}) - (C_i / T_i)$$

ή

$$U^{'} \leq U + C_i(q-1)[1/(qT_i + r) - (1/qT_i)].$$

Εφόσον q – 1 > 0 και $[1/(qT_i + r)] - (1/qT_i) \leq 0$, $U^{'} \leq U$. Επομένως μπορεί να καταλήξει κανείς στο συμπέρασμα ότι για να υπολογιστεί ο ελάχιστος μέγιστος λόγος χρησιμοποίησης πρέπει απλά να θεωρηθεί ένα σύνολο διεργασιών που να ακολουθεί τη λογική ότι ο λόγος μεταξύ της περιόδου αιτήσεων οποιονδήποτε δύο διεργασιών είναι μικρότερος από δύο. Επομένως το Θεώρημα 5 ακολουθεί ακριβώς αυτά που αποδείχτηκε στο Θεώρημα 4.

5

Αρχιτεκτονική συστήματος

5.1 Εισαγωγή στην αρχιτεκτονική του συστήματος

Στην παρούσα εργασία έχουν γίνει δύο σημαντικές αλλαγές στην αρχιτεκτονική του επεξεργαστή XR16. Η πρώτη αλλαγή συνίσταται στην προσθήκη της δυνατότητας στον επεξεργαστή να μπορεί να τρέχει περισσότερες από μία διεργασίες σε επίπεδο υλικού. Έτσι επιτυγχάνεται η μείωση του χρονικού διαστήματος που μένει ο επεξεργαστής αδρανής για την αλλαγή των διεργασιών, όταν αυτή γίνεται σε επίπεδο λογισμικού.

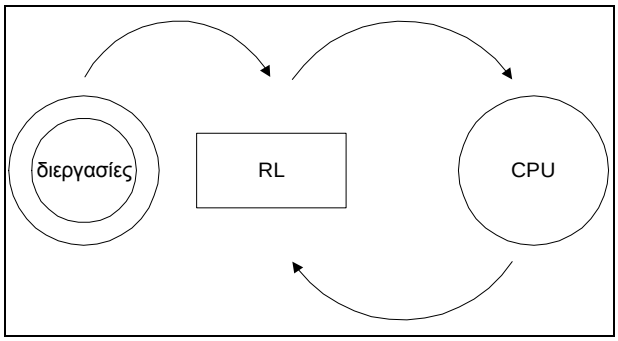
Με τη δεύτερη αλλαγή, που είναι το κυρίως τμήμα της διπλωματικής εργασίας, προστίθεται σε κάθε διεργασία ένα πεδίο προτεραιότητας. Αποτέλεσμα αυτού είναι να γίνεται η αλλαγή των διεργασιών βάση αυτών των προτεραιοτήτων. Επίσης η προτεραιότητες διατηρούνται αυστηρά, έτσι όταν μία διεργασία με υψηλότερη προτεραιότητα ζητάει να κάνει χρήση του επεξεργαστή γίνεται αμέσως η ενεργή διεργασία.

5.2 Πολλές διεργασίες στον XR16

5.2.1 Εισαγωγή στην εκτέλεση πολλών διεργασιών

Μία από τις βασικές απαιτήσεις σε όλα τα σημερινά συστήματα είναι η δυνατότητα να εκτελούνται πολλές διεργασίες «παράλληλα». Ο όρος παράλληλα είναι μέσα σε εισαγωγικά, αφού είναι αληθινός κατά το ήμισυ. Οι επεξεργαστές στην πραγματικότητα δεν έχουν αυτή τη δυνατότητα, αλλά αλλάζουν με ένα υψηλό ρυθμό τις διεργασίες που πρέπει να εκτελεστούν, έτσι ώστε ο χρήστης να έχει την ψευδαίσθηση ότι αυτές εκτελούνται παράλληλα, η διαδικασία αυτή επαναλαμβάνεται έως να μην υπάρχει κάποια διεργασία προς εκτέλεση.

Συνήθως στα προηγμένα λειτουργικά συστήματα υπάρχουν παραπάνω από μία λίστες διεργασιών προς εκτέλεση, όπου κατατάσσονται οι υποψήφιες διεργασίες ανάλογα με την προτεραιότητα που τους δίνει το λειτουργικό σύστημα. Έστω ότι RL (Ready List) είναι μία γενικευμένη λίστα με όλες τις διεργασίες προς εκτέλεση, ένα τυπικό σχήμα που εκτελείται από το λειτουργικό σύστημα είναι το παρακάτω.



Εικόνα 14 Διαδικασία εκτέλεσης πολλών διεργασιών

Στα συνηθισμένα συστήματα τη διαδικασία αλλαγής μεταξύ των ενεργών διεργασιών την αναλαμβάνει το λειτουργικό σύστημα. Εάν υποτεθεί ότι η κάθε διεργασία κάνει πλήρη

χρήση των διαθέσιμων καταχωρητών του συστήματος τότε όταν αλλάζει η ενεργή διεργασία το λειτουργικό σύστημα αναλαμβάνει την αντιγραφή των περιεχομένων των καταχωρητών στην κύρια μνήμη του συστήματος, είτε να επαναφέρει τα δεδομένα μιας διεργασίας που είχε σταματήσει σε πρότερο χρόνο, είτε να τα δώσει στη νέα διεργασία προς χρήση.

Αυτή η διαδικασία έχει κάποιο κόστος σε χρόνο επεξεργασίας ο οποίος καταναλώνεται στην αλλαγή των δεδομένων. Στον επεξεργαστή που χρησιμοποιήθηκε, για παράδειγμα, μπορεί να γίνει ένας πρόχειρος υπολογισμός για το χρόνο που χρειάζεται η εναλλαγή των διεργασιών. Ο επεξεργαστής έχει 16 καταχωρητές εκ των οποίων οι 15 χρησιμοποιούνται, αφού ο καταχωρητής «0» έχει πάντα την τιμή 0. Με ένα τυπικό χρόνο 6 κύκλων ρολογιού για ανάγνωση ή εγγραφή μίας θέσης μνήμης χρειάζονται τουλάχιστον 6x15 κύκλοι για την εγγραφή του ενός συνόλου και 6x15 για την ανάγνωση του επόμενου, σύνολο τουλάχιστον 180 κύκλοι ρολογιού, οι οποίοι τελικά υπερβαίνουν τους 200. Ο χρόνος αυτός είναι σημαντικός και αν υποτεθεί ότι η ενεργή διεργασία σε ένα τυπικό λειτουργικό σύστημα αλλάζει κάθε 20msec (ο χρόνος δεν είναι ακριβής, αλλά είναι μία τιμή που χρησιμοποιείται), τότε γίνονται 50 αλλαγές κάθε δευτερόλεπτο, δηλαδή χάνονται τουλάχιστον **180x50 = 9000** κύκλοι ρολογιού κάθε δευτερόλεπτο.

5.2.2 Μια διαφορετική προσέγγιση

Μια διαφορετική προσέγγιση θα ήταν να υπάρχουν περισσότερα από ένα σύνολα καταχωρητών, κάθε διεργασία έτσι θα μπορούσε να καταλαμβάνει ένα δικό της σύνολο και να το χρησιμοποιεί. Με αυτό τον τρόπο δε θα χρειάζεται να διαβάζεται και να γράφεται συνέχεια ένα και μοναδικό σύνολο καταχωρητών. Ο επεξεργαστής σε αυτή τη μορφή θα χρειαζόταν ένα πολυπλέκτη ο οποίος να συνδέει το σύνολο των καταχωρητών

με την επεξεργαστική μονάδα και σε κάθε αλλαγή απλά να «συνδέεται» η αντίστοιχη ομάδα καταχωρητών με τον επεξεργαστή.

Το παραπάνω προϋποθέτει πως ο επεξεργαστής θα πρέπει να μπορεί να αντιστοιχεί ένα σετ καταχωρητών σε κάθε μία διαδικασία που εκτελείται στο σύστημα. Στα πραγματικά συστήματα όμως δεν είναι απόλυτα εφικτό αυτό, γιατί δεν είναι εκ των προτέρων γνωστός ο αριθμός των διεργασιών που πρόκειται να εκτελεστούν. Έτσι υπάρχουν εταιρίες που αντί για ένα συγκεκριμένο αριθμό καταχωρητών διαθέτουν ένα τμήμα μνήμης του επεξεργαστή, που το λειτουργικό μπορεί να το χρησιμοποιεί ως ένα ευρύτερο σύνολο καταχωρητών και το καταμερίζει εκείνο ανάλογα με τις ανάγκες του επεξεργαστή ανά πάσα στιγμή. Σε άλλες εταιρίες, η αρχιτεκτονική που έχουν επιλέξει δίνει στις διεργασίες πέρα από τους σταθερούς καταχωρητές και ένα επιπλέον σύνολο που μπορούν να το δεσμεύσουν εάν το χρειάζονται (Intel Itanium & Itanium 2) ή τέλος υπάρχει και η σχεδίαση με ένα αριθμό από σετ καταχωρητών έτοιμων προς χρήση (Intel 8XC196KC/KD). Όλες οι παραπάνω λύνουν εν μέρει το πρόβλημα, αφού ο αριθμός των διαθέσιμων τελικά καταχωρητών είναι περιορισμένος, οπότε δε γίνεται να μηδενιστεί ο χαμένος χρόνος.

5.2.3 Η προσέγγιση που έγινε στο XR16

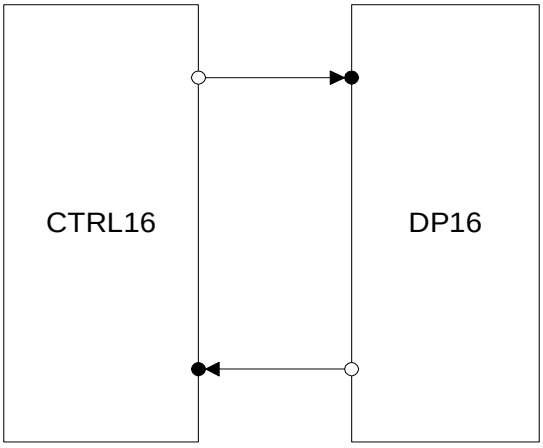
Ο XR16 είναι ένας επεξεργαστής soft-core, δηλαδή είναι γραμμένος σε κάποια γλώσσα ΓΠΥ και έπειτα από τη διαδικασία σύνθεσης μεταφέρεται σε ένα FPGA. Αυτό δίνει τη δυνατότητα να μπορεί να αλλάζει δυναμικά η αρχιτεκτονική του εφόσον είναι γνωστό εκ των προτέρων το πρόγραμμα που πρόκειται να εκτελεστεί. Έτσι μπορεί να μετατραπεί ο αριθμός των διαθέσιμων ομάδων καταχωρητών, έτσι ώστε να υπάρχουν διαθέσιμα τόσα σύνολα όσα και οι διεργασίες του προγράμματος που πρόκειται να εκτελεστούν. Αυτό

δημιουργεί το πρόβλημα του πως θα γίνεται αυτή η αλλαγή στους καταχωρητές. Οι αλλαγές στην αρχιτεκτονική καθώς και ο τρόπος αλλαγής αναλύονται στη συνέχεια.

Ο στόχος που έχει τεθεί έπειτα από τις αλλαγές που θα γίνουν είναι η κάθε διεργασία να τρέχει με ένα ανεξάρτητο σύνολο καταχωρητών. Εδώ πρέπει να αναφερθεί ότι λέγοντας διεργασία εννοείται κάποια συνάρτηση του εκτελέσιμου προγράμματος. Στον επεξεργαστή δε θα τρέχει κάποιο λειτουργικό σύστημα, αλλά το πρόγραμμα προς εκτέλεση κατευθείαν. Η λογική των πολλαπλών διεργασιών θα προστεθεί σε επίπεδο υλικού και τελικά θα είναι ο επεξεργαστής υπεύθυνος για την αλλαγή της τρέχουσας διεργασίας.

5.2.3.1 Αλλαγές στην αρχιτεκτονική

Η αρχική αρχιτεκτονική του επεξεργαστή πριν τις αλλαγές ήταν η ακόλουθη



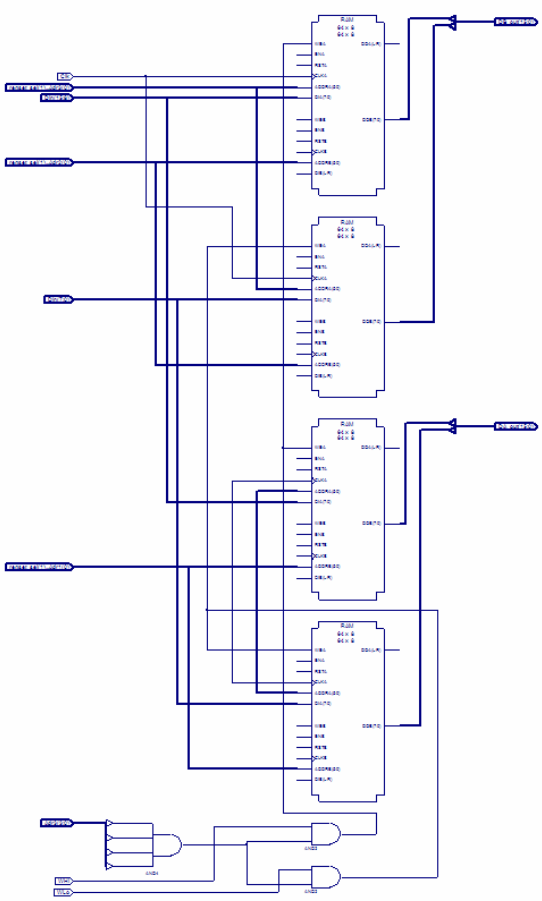
Εικόνα 15 Αρχιτεκτονική XR16

Το παραπάνω σχήμα δείχνει ότι η αρχιτεκτονική είναι σχετικά απλή, υπάρχει μία μονάδα ελέγχου (CTRL 16) και μία μονάδα για τα δεδομένα (DP16). Οι αλλαγές που θα γίνουν

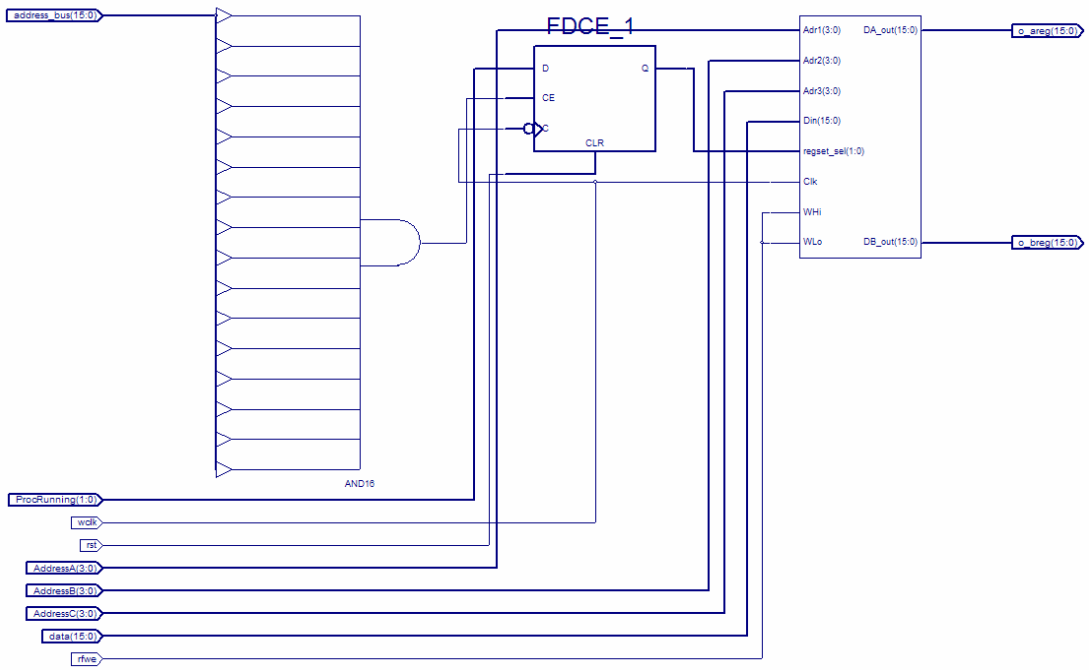
στην αρχιτεκτονική του επεξεργαστή περιορίζονται στη δεύτερη μονάδα. Ο XR16 στο εσωτερικό της μονάδας δεδομένων έχει ένα σύνολο 16 καταχωρητών που μπορούν να διαβάζονται ανά δύο ταυτόχρονα, αυτό το τμήμα θα αλλαχθεί.

Ξέροντας των αριθμό των διεργασιών που πρόκειται να εκτελεστούν, δημιουργούνται τόσα σύνολα καταχωρητών στον επεξεργαστή όσα και οι διεργασίες. Η δημιουργία των σετ γίνεται αλλάζοντας συγκεκριμένες παραμέτρους στα αρχεία του επεξεργαστή. Έτσι με τη σύνθεση του μοντέλου δημιουργούνται οι αναγκαίες θέσεις μνήμης για τους καταχωρητές.

Η επιλογή του κατάλληλου συνόλου γίνεται μέσω ενός καταχωρητή, για ένα σύνολο n διεργασιών χρειάζεται ένας καταχωρητής με μήκος $\boldsymbol{log_2 n}$ bit, το μήκος αυτό είναι το ελάχιστο για τη επιλογή της τρέχουσας διεργασίας. Δημιουργείται ένα κύκλωμα που έχοντας ως είσοδο την τρέχουσα τιμή του παραπάνω καταχωρητή επιλέγει εκείνο το σετ καταχωρητών που πρέπει να είναι ενεργό.



Εικόνα 16 - Θεωρητικό σχήμα καταχωρητών συστηματος για τέσσερις διεργασίες



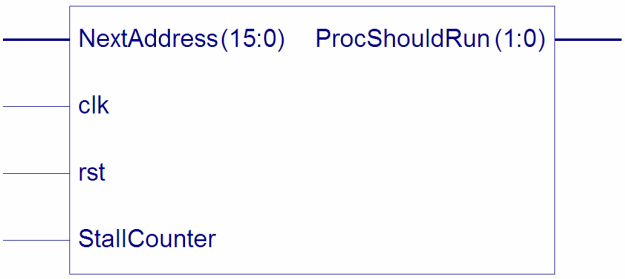
Εικόνα 17 - Θεωρητικό σχήμα καταχωρητών και αποκωδικοποιητή ελέγχου

Ο καταχωρητής που επιλέγει την τρέχουσα διεργασία είναι ένα δυαδικός μετρητής. Η κάθε τιμή του μετρητή αντιστοιχεί σε ένα από τα ***n*** σύνολα. Στον XR16 έχει επιλεχθεί αυτός να διεγείρεται από μία θέση μνήμης, η οποία και δεσμεύεται.

Η τιμή που έχει δεσμευτεί είναι η 0xFFFF, όταν αυτή περνάει από τη δίαυλο δεδομένων τότε το αντίστοιχο Flip Flop, FDCE_1 στο παραπάνω σχήμα, ενεργοποιείται και η τιμή για την επιλογή του σωστού σετ καταχωρητών μεταφέρεται.

Από τα παραπάνω γίνεται κατανοητό ότι οι αλλαγές που πρέπει να γίνουν έτσι ώστε να λειτουργήσει το σύστημα είναι δύο, πρώτον η δημιουργία των πολλαπλών σετ καταχωρητών και δεύτερον η δημιουργία ενός κυκλώματος που θα «ακούει» το διάδρομο διευθύνσεων του επεξεργαστή και όταν «ακούσει» τη δεσμευμένη τιμή να αυξάνει το μετρητή. Ο μετρητής αυτός παίρνει τιμές από ***μηδέν*** έως ***n***, ενώ όταν φτάσει στη μέγιστη τιμή του μηδενίζεται αυτόματα.

Στο εσωτερικό του μετρητή έχει υλοποιηθεί ένας διαιρέτης για το σήμα του ρολογιού. Ως είσοδο παίρνει το σήμα του ρολογιού ενώ στην έξοδο του βγαίνει ένας παλμός ανά 1msec. Αυτός βοηθάει, έτσι ώστε να μπορούν να τίθενται μικρότερες τιμές για τους χρόνους που πρέπει να τρέξει κάθε διεργασία και ο χρήστης να μετράει σε msec αντί για κύκλους ρολογιού τους χρόνους εκτέλεσης κάθε διεργασίας. Ο διαιρέτης αυτός ρυθμίζεται πριν τη σύνθεση του συστήματος έτσι ώστε στην ταχύτητα που τελικά θα τρέξει ο επεξεργαστής να παράγει του παλμούς με τη συχνότητα που χρειάζεται.

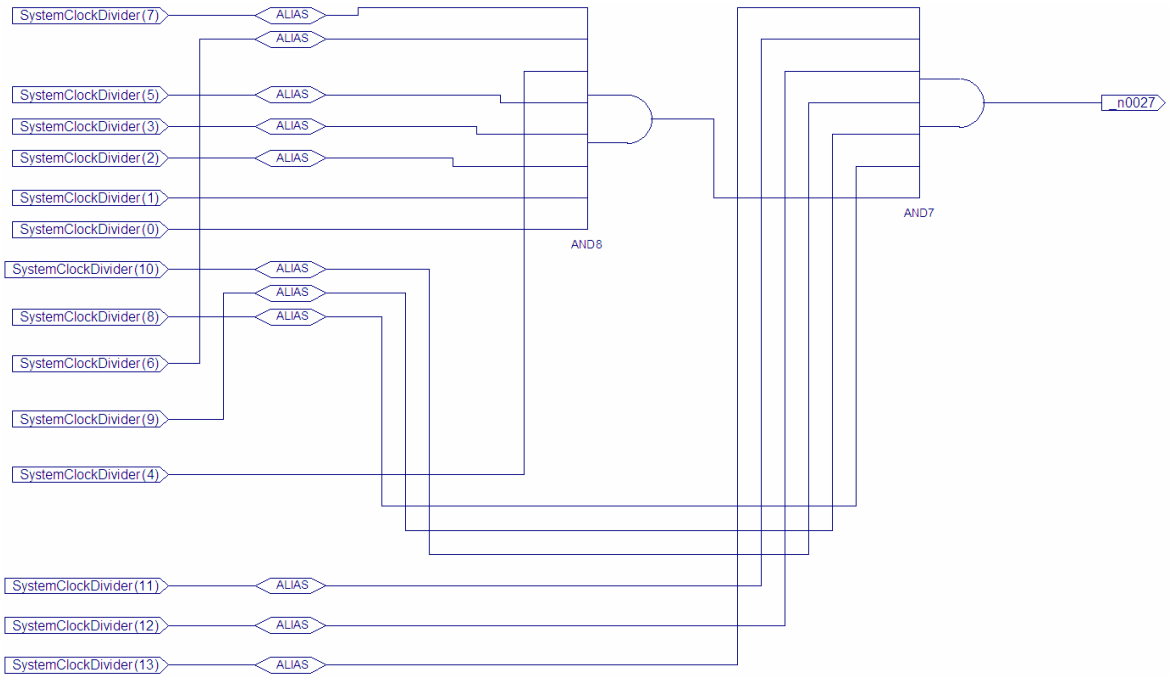


Εικόνα 18 Μετρητής επιλογής σετ καταχωρητών

Στο περιγραφικό μοντέλου του μετρητή παραπάνω φαίνονται οι είσοδοι του και η μία έξοδος που έχει. Οι είσοδοι είναι :

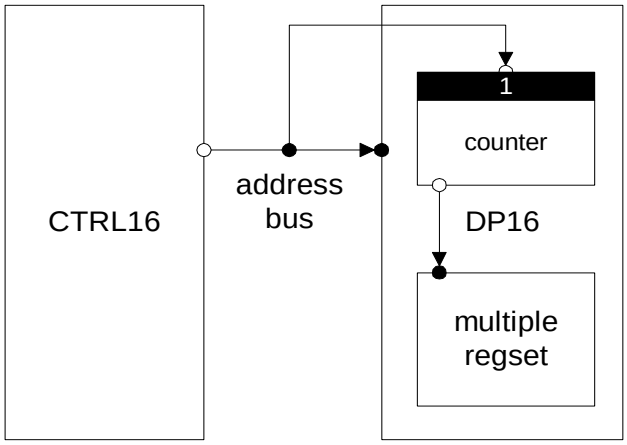
- NextAddress, η είσοδος του διαύλου διευθύνσεων όπου ο μετρητής ακούει τις τιμές
- rst σήμα μηδενισμού του μετρητή
- StallCounter, σήμα ελέγχου για πάγωμα μετρητή.

Και το κύκλωμα του διαιρέτη για τη συχνότητα του ρολογιού και τη δημιουργία του εσωτερικού σήματος



Εικόνα 19 Κύκλωμα διαιρέτη συχνότητας

Ένα γενικό σχήμα της νέας αρχιτεκτονικής είναι το παρακάτω



Εικόνα 20 Νέα αρχιτεκτονική XR16

5.2.3.2 Λειτουργία του συστήματος

Η λειτουργία του όλου συνόλου στηρίζεται στην αύξηση του μετρητή που ελέγχει το ποιο θα είναι το ενεργό σύνολο καταχωρητών. Όπως αναφέρθηκε παραπάνω αυτός ο μετρητής ελέγχεται από μία διεύθυνση μνήμης, εδώ έχει επιλεχθεί η τελευταία, η πιο χαμηλή διεύθυνση από τις διαθέσιμες, δηλαδή η 0xFFFF.

Το κύκλωμα ελέγχου ακούει το διάδρομο διευθύνσεων του επεξεργαστή, μόλις «δει» αυτή την τιμή τότε αυξάνει τον μετρητή. Ωστόσο τίθεται ένα πρόβλημα η αλλαγή του ενεργού συνόλου καταχωρητών πρέπει να γίνει σε ένα σημείο που δε θα δημιουργήσει σφάλματα στην εκτέλεση του προγράμματος, λέγοντας σφάλματα εννοούνται λανθασμένες τιμές ή λανθασμένα αποτελέσματα πράξεων. Σφάλματα μπορεί να προξενηθούν εφόσον ο επεξεργαστής χρησιμοποιεί το pipeline για τη μεταφορά των δεδομένων προς τη μονάδα εκτέλεσης των εντολών στον πυρήνα. Επομένως όταν γίνει η αλλαγή της τρέχουσας διεργασίας θα πρέπει να είναι ελεγμένο ότι το pipeline είναι άδειο και ότι όλες οι εντολές

που εκτελούνται εκείνη τη χρονική στιγμή έχουν τελειώσει. Σε αντίθετη περίπτωση είναι πολύ πιθανό τα αποτελέσματα εντολών από την προηγούμενη διεργασία θα αποθηκευτούν ή οι εντολές θα επιδράσουν στους καταχωρητές της επόμενης.

Η λύση που δόθηκε στο παραπάνω πρόβλημα είναι απλή, πριν την αλλαγή της ενεργής διεργασίας ζητείται από τον επεξεργαστή να εκτελέσει μία διακοπή (interrupt). Αυτή η επιλογή έχει θετικά και αρνητικά στοιχεία. Πρώτον αδειάζει εντελώς το pipeline, από τις τρέχουσες εντολές, με αυτό τον τρόπο δε δημιουργούνται σφάλματα. Αυτό αποτελεί εξορισμού μέρος της λειτουργίας του επεξεργαστή για τη διαχείριση των διακοπών. Επίσης ένα σημαντικό θετικό στοιχείο είναι ότι σώνεται, εξορισμού από τη σχεδίαση του επεξεργαστή, η τελευταία διεύθυνση μνήμης, όπου ήταν ο μετρητής προγράμματος. Επομένως όταν τελειώσει η διεργασία που θα τρέξει ο επεξεργαστής θα γυρίσει στην τελευταία εντολή που εκτελούσε πριν συμβεί η διακοπή. Ωστόσο η λύση με τη χρησιμοποίηση της διακοπής δημιουργεί μία μικρή εξάρτηση για το πώς πρέπει να είναι ο εκτελέσιμος κώδικας.

Στον κώδικα του προγράμματος ο χρήστης πρέπει να προσθέσει τις παρακάτω εντολές στο κώδικα assembly

```
        align 16
isr:
        sb r0, 0xFFFF(r0)
        reti
```

Για ευκολία στη δημιουργία νέων προγραμμάτων αυτές οι εντολές έχουν προστεθεί στο αρχείο *reset.s* το οποίο χρειάζεται για την δημιουργία όλων των προγραμμάτων προς εκτέλεση.

```
always @(negedge wclk or posedge rst) begin
        if (rst)
                active_process=`InitProc; //Αρχικοποίηση συστήματος.
        else begin
//περιμένουμε έως να έρθει ως διεύθυνση η δεσμευμένη τιμή 0xFFFF, αυτή είναι η δεσμευμένη τιμή
//που λειτουργεί ως «σημαία» για την αλλαγή της τρέχουσας διεργασίας.
                if (address_bus == `MemoryMappedCounter) begin
                        active_process = ProcRunning; //- 1'b1;
                end
        end
end
```

Εικόνα 21 - Τμήμα κώδικα ελέγχου για τη δεσμευμένη διευθυνση στο δίαυλο δεδομένων

Στη ρουτίνα εξυπηρέτησης της διακοπής εκτελείται η εντολή :

sb r0, 0xFFFF(r0)

κατά την εκτέλεση της σώνεται στη διεύθυνση "0xFFFF + r0 = 0xFFFF" η τιμή του r0, δηλαδή 0. Παρόλο που η πράξη δεν έχει κάποιο νόημα σε επίπεδο αποτελέσματος, χρειάζεται για να περάσει η δεσμευμένη τιμή από το δίαυλο διευθύνσεων, με αποτέλεσμα την αύξηση του μετρητή ελέγχου των καταχωρητών και να γίνει η αλλαγή της ενεργής διεργασίας.

Στη συνέχεια της ρουτίνας διακοπής, ο επεξεργαστής εκτελώντας την εντολή *reti* θα επιστρέψει στη διεύθυνση που ορίζεται από των καταχωρητή *r14*. Ωστόσο πριν εκτελεστεί η εντολή θα έχει ήδη αλλάξει το ενεργό σετ καταχωρητών, οπότε θα διαβάσει τη διεύθυνση επιστροφής από το σετ καταχωρητών της διεργασίας που θα πρέπει να τρέξει. Στην αρχικοποίηση του συστήματος στους καταχωρητές r14 σε κάθε σετ καταχωρείται η αρχική διεύθυνση κάθε διεργασίας του συστήματος. Με αυτό τον τρόπο την πρώτη φορά που θα διαβαστεί η *r14*, εάν δεν έχει εκτελεστεί ξανά η διεργασία, ο επεξεργαστής απλά να μεταβεί στην αρχική διεύθυνση της διεργασίας προς εκτέλεση.

Η αλλαγή γίνεται πριν εκτελεστεί η εντολή “reti”, όταν πλέον εκτελείται αυτή η εντολή η ενεργή διεργασία έχει αλλάξει και επενεργεί στην επόμενη διεργασία. Οπότε η επιστροφή από τη ρουτίνα εξυπηρέτησης διακοπής θα γίνει στην επόμενη διεργασία και ο επεξεργαστής συνεχίζει να εργάζεται απρόσκοπτα.

Για να ολοκληρωθεί η διαδικασία, να γίνεται διάφανα προς τον χρήστη και το λειτουργικό σύστημα να μη χρειάζεται να κάνει κάτι, έχει προστεθεί ένας μετρητής, ο οποίος μετράει το χρόνο μέχρι να γίνει η αλλαγή της ενεργής διεργασίας. Αυτός ο μετρητής δουλεύει συνέχεια σε επίπεδο υλικού και όταν ολοκληρώνεται το ζητούμενο χρονικό διάστημα, διεγείρεται το σήμα της διακοπής και εκτελείται η ρουτίνα εξυπηρέτησης διακοπής. Είναι άμεσα αντιληπτό ότι εάν ο χρήστης θέλει μέσα από το πρόγραμμα να αλλάξει την ενεργή διεργασία, αρκεί να εκτελέσει μία εντολή αίτησης διακοπής προς τον επεξεργαστή.

Ο χρήστης κατά τη σύνθεση του επεξεργαστή περνάει τις ρυθμίσεις για τους χρόνους που θα πρέπει να τρέχει κάθε διεργασία. Έτσι ο μετρητής ξέρει πόσο χρόνο πρέπει να τρέξει κάθε διεργασία. Αυτές οι σταθερές βρίσκονται μέσα στο αρχείο “process_quantum_timer.v”

Η αλλαγή των διεργασιών που πρέπει να εκτελεστούν γίνεται κυκλικά με ένα απλό αλγόριθμο τύπου round-robin, στον οποίο δεν υπάρχουν προτεραιότητες για τις διεργασίες απλά εκτελούνται όλες όσες θέλει ο χρήστης με τη σειρά που τις έχει δώσει.

Με την παραπάνω διαδικασία και τις αλλαγές που έγιναν, επιτεύχθηκε η αλλαγή της ενεργής διεργασίες να γίνεται σε διάστημα μικρότερο από δέκα κύκλους ρολογιού. Η

βελτίωση από τους διακόσιους είναι πολύ σημαντική και το διάστημα που ο επεξεργαστής μένει αδρανής, στη διάρκεια εναλλαγής των διεργασιών μειώνεται σημαντικά.

Καταλήγοντας για τη λειτουργία του επεξεργαστή σε αυτή τη μορφή, θα πρέπει πριν τη σύνθεση του ο προγραμματιστής να κάνει τα παρακάτω βήματα.

- Να κάνει compile την εφαρμογή που θα τρέξει
- Να βρει από τα listings του προγράμματος της διευθύνσεις αρχής κάθε διεργασίας και να τις αντιγράψει στις σταθερές R14_HI_x και R14_LO_x , που βρίσκονται στο αρχείο «register_files.v».
- Επίσης πρέπει να αρχικοποιήσει τις τιμές των καταχωρητών R13_HI_x και R13_LO_x που βρίσκονται στο ίδιο αρχείο, «register_files.v», αυτοί είναι οι καταχωρητές που δείχνουν τη θέση της μνήμης στοίβας (stack pointers) για κάθε μία από τις διεργασίες που εκτελούνται.
- Να θέσει τον αριθμό των διεργασιών που θα τρέξουν στις σταθερές «instances» και «log_2_instances».

Με αυτές τις τιμές το σύστημα αρχικοποιεί του μετρητές προγράμματος για το ζητούμενο αριθμό διεργασιών. Επίσης αρχικοποιεί του καταχωρητές που χρειάζονται για τη σωστή επιστροφή του επεξεργαστή στη διεύθυνση του προγράμματος που πρέπει να εκτελεστεί. Είτε κατά την είσοδο της διεργασίας την πρώτη φορά, είτε από την είσοδο της εφαρμογής μετά την εκτέλεση της ρουτίνας διακοπής.

5.2.4 Εισαγωγή προτεραιοτήτων στο σύστημα

Με τις αλλαγές που έγιναν παραπάνω δημιουργήθηκε ένας επεξεργαστής ικανός να εκτελεί πολλές διεργασίες ταυτόχρονα, αλλά χωρίς περιορισμούς ως προς τη σειρά εκτέλεσης τους. Στη συνέχεια εισάγεται η έννοια της προτεραιότητας για κάθε μία από τις ενεργές διεργασίες, έτσι ώστε να εκτελείται πάντα η διεργασία με την υψηλότερη προτεραιότητα.

Παραπάνω αναφέρθηκε ότι ο επεξεργαστής, με τις αλλαγές που έγιναν, μπορεί να εκτελεί ένα συγκεκριμένο πρόγραμμα με ένα συγκεκριμένο αριθμό διεργασιών, έτσι αντί να υπάρχει πεδίο προτεραιότητας για κάθε μία από τις πιθανές διεργασίες θεωρείται ότι η σειρά που δίνονται από το χρήστη είναι και η σειρά με τη οποία πρέπει να εκτελούνται. Αυτό αν και φαίνεται περιοριστικό δεν είναι, ενώ περιορίζεται και το αναγκαίο υλικό τμήμα που θα χρειαζόταν για την ανάγνωση και ανάθεση των προτεραιοτήτων σε αντίθετη περίπτωση.

5.2.5 Νέες λειτουργίες του συστήματος

Το νέο σύστημα με τις προτεραιότητες διέπεται από τις ίδιες αρχές λειτουργίας όπως και το προηγούμενο, έτσι υπάρχουν και εδώ τα πολλαπλά σύνολα καταχωρητών, επίσης υπάρχει μετρητής χρόνου για το διαθέσιμο διάστημα επεξεργαστικής ισχύος που δίνεται σε κάθε διεργασία. Ωστόσο έχουν προστεθεί ορισμένα επιπλέον στοιχεία και κάποια υπάρχοντα έχουν υποστεί αλλαγές. Οι αλλαγές και οι προσθήκες προήλθαν από τις απαιτήσεις του ζητούμενου νέου περιβάλλοντος λειτουργίας του επεξεργαστή. Στη συνέχεια αναζητούνται και περιγράφονται οι απαιτήσεις του νέου περιβάλλοντος.

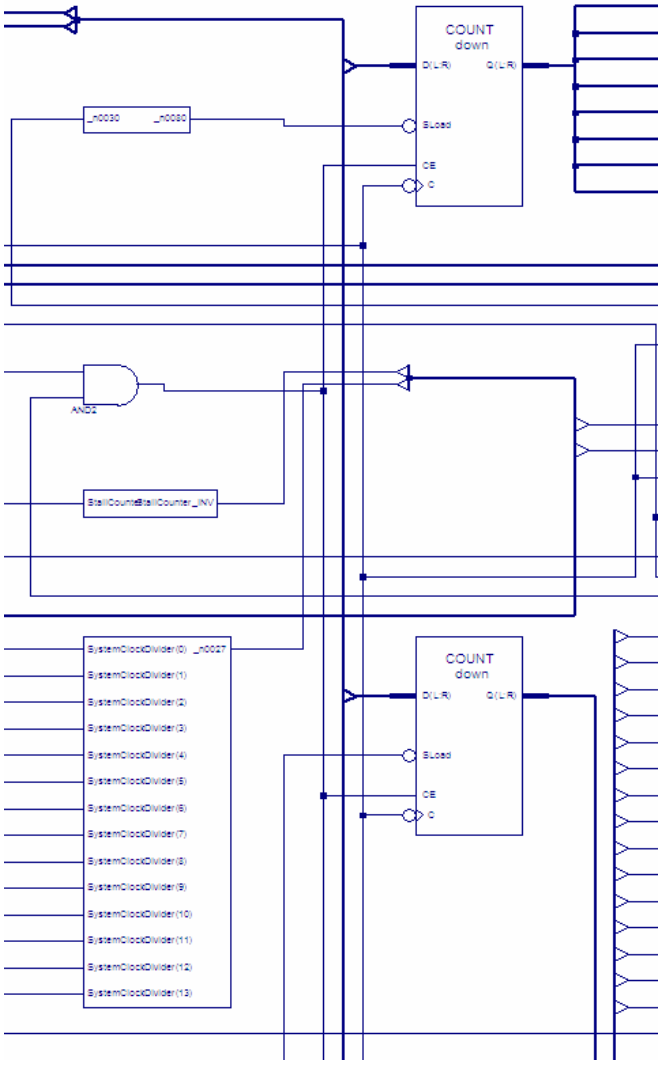
5.2.5.1 Απαιτήσεις περιβάλλοντος

Όπως έχει αναφερθεί σκοπός της εργασίας είναι η λειτουργία του επεξεργαστή σε ένα αυστηρό περιβάλλον πραγματικού χρόνου. Σε μία τέτοια εφαρμογή κάθε διεργασία πρέπει να εκτελείται ανά τακτά χρονικά διαστήματα, έστω T_i είναι το ζητούμενο χρονικό διάστημα για την διεργασία i. Μόλις παρέλθει το χρονικό διάστημα και η διεργασία ζητήσει τον επεξεργαστή τότε πρέπει η αλλαγή να γίνει άμεσα, ή μέσα σε ένα επιτρεπτό όριο (πολύ μικρότερο από το χρόνο αντίδρασης των Windows…). Επομένως πρώτη απαίτηση για το νέο σύστημα είναι ένα μετρητής μήκους $\log_2 T_i$ για κάθε μία από τις διεργασίες.

Οι παραπάνω μετρητές δουλεύουν παράλληλα, οπότε είναι απόλυτα κατανοητό να αναρωτηθεί κανείς εάν θα γινόταν να συμπτυχθούν σε έναν και μοναδικό μετρητή. Αυτό όντως μπορεί να συμβεί αλλά με περιορισμούς ως προς της σχέση μεταξύ των μετρούμενων τιμών, διαφορετικά θα έπρεπε να εκτελείται η πράξη modulo σε κάθε κύκλο ρολογιού, πράγμα αδύνατο, ενώ και αν γινόταν θα υπήρχε μεγάλη απώλεια επεξεργαστικής ισχύος. Έτσι αποφασίστηκε να χρησιμοποιηθούν παράλληλοι μετρητές.

Στην αρχή του κεφαλαίου παρουσιάστηκε το μοντέλο του μετρητή (Εικόνα 18), υπάρχει μία είσοδος που λέγεται “StallCounter”. Αυτή η είσοδος έχει τη λειτουργία ότι κατά την εναλλαγή μεταξύ των διεργασιών, οι μετρητές ελέγχου να μην μετράνε έτσι ώστε οι ζητούμενοι χρόνοι εκτέλεσης κάθε λειτουργίας να είναι οι πραγματικοί χρόνοι χωρίς να υπεισέρχονται οι χρόνοι των καθυστερήσεων μέσα σε αυτούς.

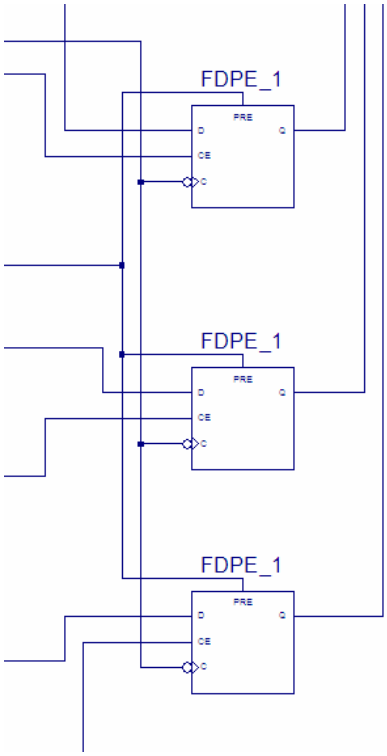
Στη επόμενη εικόνα φαίνεται ένα μέρος του κυκλώματος που ελέγχει ποια θα είναι η ενεργή διεργασία, όπου φαίνονται δύο από τους μετρητές που αναφέρθηκαν παραπάνω, αυτοί είναι οι δύο οντότητες “Count down” στο κύκλωμα. Ενώ φαίνεται και ο διαιρέτης συχνότητας SystemClockDivider κάτω αριστερά.



Εικόνα 22 Τμήμα του κυκλώματος επιλογής διεργασίας

Ο κάθε μετρητής από τους παραπάνω όταν λήξει πρέπει να αναφέρει τη λήξη του στον επεξεργαστή. Άρα πρέπει να υπάρχει ένα σήμα για κάθε ένα μετρητή, έτσι χρησιμοποιείται ένας καταχωρητής μήκους n bit. Κάθε ένα bit από αυτόν τον καταχωρητή αντιστοιχεί σε

μία διεργασία, ξεκινώντας από το bit 0 και φτάνοντας στο bit n-1. Όταν η τιμή του αντίστοιχου bit είναι «ένα» τότε η αντίστοιχη διεργασία πρέπει να εκτελεστεί άμεσα, διαφορετικά εκτελούνται οι διεργασίες με υψηλότερη προτεραιότητα. Στο παράδειγμα ελέγχου με τις τέσσερις διεργασίες, δημιουργείται ένας καταχωρητής τριών θέσεων για να αποθηκεύεται το πότε τρέχει η κάθε διεργασία του, το κύκλωμα του καταχωρητή φαίνεται στη συνέχεια.



Εικόνα 23 Καταχωρητής τριών θέσεων για το παράδειγμα με τις τρεις διεργασίες

5.2.5.2 Παράδειγμα λειτουργίας

Στο σύστημα που περιγράφεται εκτελείται πάντα η διεργασία με την υψηλότερη προτεραιότητα, ας υποτεθεί ένα παράδειγμα με τρεις διεργασίες Α, Β, Γ με αντίστοιχες προτεραιότητες. Επίσης θεωρείται ότι ο επεξεργαστής δίνει 20msec σε κάθε διεργασία, και ότι όλες οι διεργασίες έχουν το αντίστοιχό τους bit, στον καταχωρητή σήμανσης εκτέλεσης διεργασίας, ενεργοποιημένο (τιμή ίση με «ένα»). Επίσης υποτίθεται ότι η διεργασία Α έχει χρόνο εκτέλεσης 20msec, η Β 50msec και η Γ 30msec. Τέλος θεωρείται πως η διεργασία Α καλείται κάθε 40 msec, η Β κάθε 200msec κα η Γ κάθε 240msec. Τα παράδειγμα θα έχει το ακόλουθο σχήμα εκτέλεσης

0	20	40	60	80	100		120	140	160
A	B	A	B	A	B	Γ	A	Γ	A

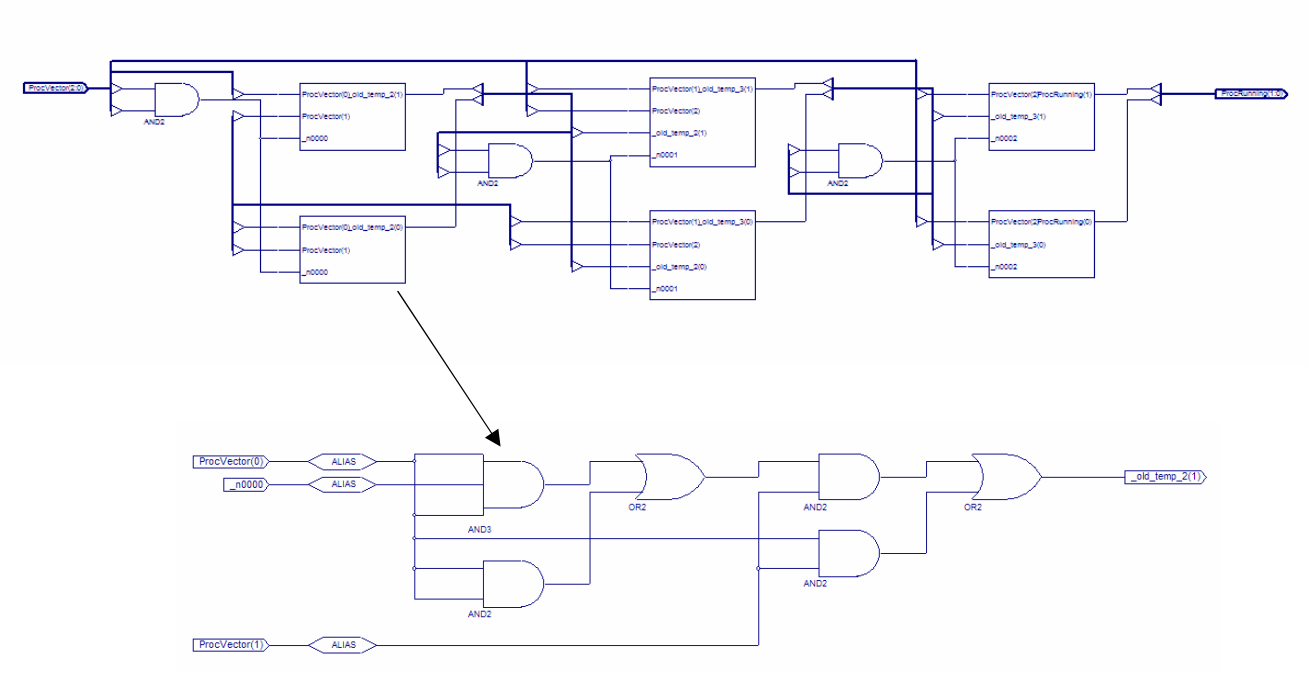
Εικόνα 24 Εκτέλεση διεργασιών σε περιβάλλον πραγματικού χρόνου

Στον παραπάνω πίνακα φαίνεται ότι εκτελούνται συνέχεια οι διεργασίες Α, Β μέχρι να τελειώσει η Β, έτσι ώστε να ξεκινήσει να εκτελείται η Γ, που έχει τη χαμηλότερη προτεραιότητα, ενώ μόλις ζητήσει την επεξεργαστική μονάδα η διεργασία Α η Γ σταματάει και δίνει τη θέση της στη Α.

5.2.5.3 Απαιτήσεις περιβάλλοντος Β

Το προηγούμενο παράδειγμα οδηγεί στο συμπέρασμα ότι χρειάζεται ακόμη ένας αποκωδικοποιητής προτεραιοτήτων, ο οποίος θα επενεργεί στον καταχωρητή με τις διεργασίες προς εκτέλεση και θα επιστρέφει το αριθμό εκείνης που πρέπει να τρέξει στο άμεσο χρονικό παράθυρο.

Αυτός ο αποκωδικοποιητής είναι ένα από τα σημαντικότερα τμήματα του τελικού συστήματος, αφού στην απρόσκοπτη λειτουργία του στηρίζεται η σωστή λειτουργία του επεξεργαστή. Το θεωρητικό σχήμα του αποκωδικοποιητή για μία εφαρμογή με τέσσερις διεργασίες φαίνεται στο επόμενο σχήμα. Το σχήμα έχει προκύψει έπειτα από τη σύνθεση των αντίστοιχων τμημάτων κώδικα



Εικόνα 25 Θεωρητικό σχήμα αποκωδικοποιητή προτεραιοτήτων

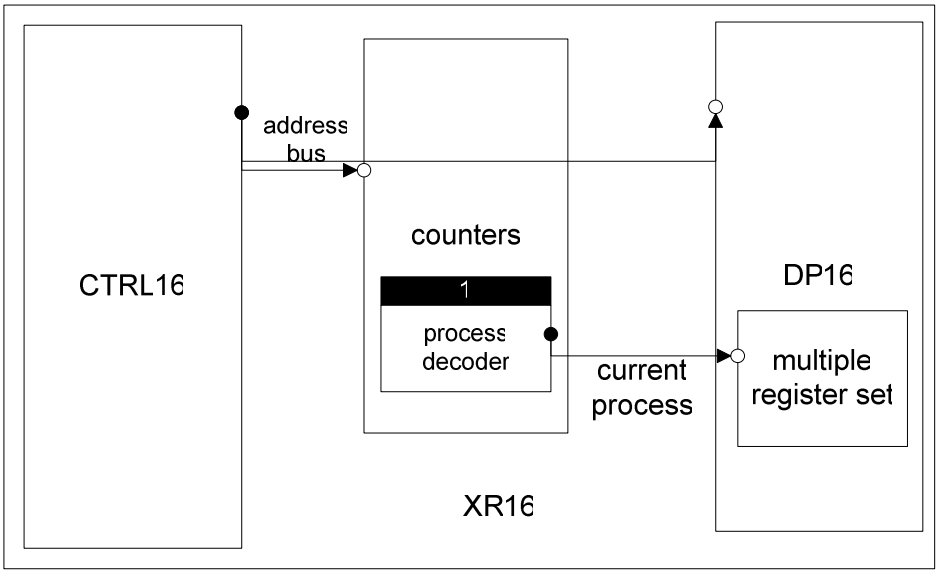
Το κυρίως τμήμα του κώδικα του κώδικας που αποκωδικοποιητή φαίνεται στη συνέχεια

```
always @(ProcVector) begin
        temp = `InitProcVector;

        for (i=0;i<`total_processes;i=i+1) begin
                if (ProcVector[i] == 1) begin
                        if (i==0)
                                temp = 0;
                        else begin
                                if (ProcVector[i-1] == 0) begin
                                        if (temp == `InitProcVector)
                                                temp = i;
                                end
                        end
                end
        end
end

assign ProcRunning = temp;
```

Όπως φαίνεται και από τον κώδικα, όποτε αλλάζει ο καταχωρητής με τις διεργασίες προς εκτέλεση επιλέγεται η επόμενη που θα πρέπει να τρέξει, αν πρέπει να αλλάξει. Αναφέρθηκε στην αρχή ότι η σειρά των διεργασιών, έτσι όπως δίνεται από το χρήστη, είναι και αυτή που δίνει την αντίστοιχη προτεραιότητα σε κάθε διεργασία. Έτσι για να τρέξει μία διεργασία θα πρέπει η προηγούμενη της να έχει τελειώσει την εκτέλεση της, έλεγχος *ProcVector[i-1] == 0* στον κώδικα.

5.2.6 *Νέα αρχιτεκτονική επεξεργαστή*

Οι παραπάνω απαιτήσεις για το περιβάλλον ενός συστήματος πραγματικού χρόνου οδηγούν στην εισαγωγή επιπλέον στοιχειών στον επεξεργαστή. Η τελική αρχιτεκτονική του προσαρμοσμένου στις νέες απαιτήσεις επεξεργαστή είναι η παρακάτω. Οι αλλαγές που χρειάζονται γίνονται στο τμήμα δεδομένων του επεξεργαστή και στο ανώτερο επίπεδο της δομής του επεξεργαστή.



Εικόνα 26 Νέα αρχιτεκτονική επεξεργαστή

5.2.7 *Ανάλυση νέου τρόπου λειτουργίας επεξεργαστή*

Παραπάνω αναφέρθηκε ότι στο νέο σύστημα έχουν διατηρηθεί χαρακτηριστικά από την αρχική του μορφή. Η κύρια διαφορά που υπάρχει είναι ότι η διεργασία που θα εκτελεστεί με το πέρας του χρονικού διαστήματος εκτέλεσης της διεργασίας, που τρέχει κάποια δεδομένη στιγμή ή την άφιξη της χρονικής στιγμής εκτέλεσης την νέας διεργασίας, δεν είναι απαραίτητα η επόμενη στο κώδικα.

Σε κάθε κτύπο του ρολογιού μειώνονται παράλληλα όλοι οι μετρητές (οντότητα “counters” στην Εικόνα 26). Μόλις κάποιος απ’ όλους μηδενιστεί γίνεται το αντίστοιχο bit στον καταχωρητή ελέγχου ίσο με μηδέν, έστω ότι η διεργασία λέγεται ***A***. Ο καταχωρητής αυτός ταυτόχρονα αποκωδικοποιείται και δίνει τον αριθμό της διεργασίας που πρέπει να τρέξει.

Εάν ο αριθμός αυτός συμπίπτει με κάποια διεργασία με υψηλότερη προτεραιότητα τότε το τμήμα ελέγχου ενεργοποιεί το σήμα διακοπής έτσι ώστε να υπάρξει η διαδικασία που περιγράφτηκε παραπάνω για την εναλλαγή των διεργασιών. Μόλις γίνει η διακοπή η επόμενη διεργασία που θα εκτελεστεί είναι η ζητούμενη. Διαφορετικά εάν η νέα διεργασία έχει μικρότερη προτεραιότητα τότε αναγκαστικά περιμένει να τελειώσει την εκτέλεση της η διεργασία που εκτελείται, αφού αυτή έχει μεγαλύτερη προτεραιότητα. Εάν στο χρονικό διάστημα μέχρι να τελειώσει την εκτέλεση της η τρέχουσα διεργασία δεν έχει ζητήσει τον επεξεργαστή κάποια διεργασία με μεγαλύτερη προτεραιότητα από την ***A*** τότε εκτελείται η ***A***.

Παρατηρείται εδώ ότι η αλλαγή δε γίνεται βάση του χρονικού διαστήματος που εκτελείται η πρώτη διεργασία, πριν έρθει στο προσκήνιο η καινούργια, αλλά βάση των προτεραιοτήτων τους. Αυτό είναι και ένα από τα σημαντικότερα χαρακτηριστικά ενός συστήματος πραγματικού χρόνου.

5.2.8 Αλλαγές στη λειτουργία των διεργασιών

Οι διεργασίες που τρέχουν σε ένα σύστημα πραγματικού χρόνου κανονικά δεν πρέπει να σταματούν να δουλεύουν ποτέ, ας υποτεθεί το σύστημα ελέγχου ενός πυρηνικού αντιδραστήρα. Σε ένα τέτοιο σύστημα γίνονται διάφορες μετρήσεις και σε περίπτωση

σφάλματος πρέπει κάποιες ρουτίνες να εκτελεστούν, οι διαδικασίες ελέγχου πρέπει τρέχουν συνέχεια. Αν υποτεθεί ότι η κάθε διαδικασία ελέγχου αποτελεί μία ρουτίνα, του υπό σχεδίαση νέου επεξεργαστή, τότε μόλις φτάσει στο τέλος της, πρέπει να εκτελεστεί από την αρχή ξανά. Γι' αυτό το λόγο ο καταχωρητής R15 σε κάθε μία ομάδα καταχωρητών αρχικοποιείται με την διεύθυνση αρχής της αντίστοιχης διεργασίας, έτσι όταν τελειώσει η διεργασία ξεκινάει ξανά από την αρχή.

5.2.9 Αλλαγές στο θεωρητικό μοντέλο

Το θεωρητικό μοντέλο του περιβάλλοντος πραγματικού χρόνου μιλάει μονάχα για διεργασίες που πρέπει να εκτελεστούν σε κάποιο χρονικό σημείο, με την ελάχιστη δυνατή καθυστέρηση. Σε αυτό το μοντέλο έχει προστεθεί η έννοια της διεργασίας "idle".

Στο σύνολο των διεργασιών που δίνει ο χρήστης θεωρείται ότι η τελευταία, αυτή με την ελάχιστη τιμή προτεραιότητας, είναι αυτή που θα γεμίζει όλα τα κενά διαστήματα λειτουργίας του επεξεργαστή. Στις περιόδους εκείνες όπου δεν εκτελείται κάποια άλλη διεργασία με υψηλότερη προτεραιότητα. Ο χρήστης έτσι μπορεί να πετύχει χρησιμοποίηση του επεξεργαστή κοντά στο απόλυτο 100% στο διαθέσιμο χρόνο για εκτέλεση διεργασιών, χωρίς να υπάρχουν τμήματα χρόνου όπου ο επεξεργαστής δε χρησιμοποιείται.

6

Εγκατάσταση απαραίτητων προγραμμάτων

6.1 Εισαγωγή

Ο έλεγχος και η χρήση του συστήματος προαπαιτούν την εγκατάσταση ορισμένων εφαρμογών. Αυτές χρειάζονται είτε για τη δημιουργία των εφαρμογών που θα τρέξουν στο σύστημα, είτε για την προσομοίωση του στον υπολογιστή και την εξαγωγή των αποτελεσμάτων.

6.2 Πηγαίος κώδικας επεξεργαστή

Ο πηγαίος κώδικας του επεξεργαστή μπορεί να βρεθεί στην παρακάτω διεύθυνση:

http://www.fpgacpu.org/xsoc

Από την παραπάνω τοποθεσία γίνεται download η τελευταία έκδοση του επεξεργαστή, η έκδοση 0.93 έχει εγκατασταθεί για τον έλεγχο και την τροποποίηση του συστήματος. Ο κώδικας έρχεται σε μορφή αρχείου zip. Έστω ότι αυτό αποσυμπιέζεται στον κατάλογο

c:\xsoc

Στον κώδικα αυτό προστίθενται και αλλάζουν τα αρχεία εκείνα που χρειάζονται και έχουν αναφερθεί στα προηγούμενα κεφάλαια.

6.3 LCC Compiler

Ο LCC compiler χρησιμοποιείται για να μετατρέπει τον κώδικα C σε κώδικα assembly ικανό να τρέξει στον επεξεργαστή μας. Έχει επιλεχθεί ο συγκεκριμένος, γιατί είναι open-source και μπορεί να τροποποιηθεί έτσι ώστε να υποστηρίζει διαφορετικά set εντολών assembly.

Η έκδοση που θα εγκατασταθεί είναι η 4.1, έχει επιλεχθεί η συγκεκριμένη γιατί υπάρχει αντίστοιχη έκδοση του compiler για τον xr16. Η εγκατάσταση του γίνεται άμεσα από το πακέτο **lcc41.exe,** και έστω ότι επιλέγεται ως τοποθεσία εγκατάστασης η default τοποθεσία: **C:\Program Files\lcc\4.1\ .** Στον κατάλογο που δημιουργείται, φτιάχνονται διάφοροι υποκατάλογοι, όπως ο ***bin*** που περιέχει τα εκτελέσιμα αρχεία του προγράμματος.

Στη συνέχεια από τον κατάλογο ***c:\xsoc\lcc-xr16\bin*** πρέπει να αντιγραφούν όλα τα αρχεία που περιέχονται στον κατάλογο ***bin*** του LCC, δηλαδή τον **C:\Program Files\lcc\4.1\bin.** Αυτά τα αρχεία αποτελούν τον compiler για τον επεξεργαστή.

Τελευταίο βήμα είναι η προσθήκη στο περιβάλλον των windows των απαραίτητων entries για τη λειτουργία του compiler. Τα παρακάτω entries πρέπει να εισαχθούν στο σύστημα:

- LCCDIR με τιμή **c:\progra~1\lcc\4.1\bin**, αυτή μας δείχνει που είναι εγκατεστημένα τα αρχεία του compiler

  Η τιμή αυτή μπορεί να μπει είτε μέσα από τα windows είτε στο παράθυρο του dos εκτελώντας την εντολή

  C:\> set LCCDIR=c\progra~1\lcc\4.1\bin

- Στο path του συστήματος πρέπει να μπει η παραπάνω εγγραφή, αυτό γίνεται προσθέτοντας στο τέλους του path την τιμή **%LCCDIR%**. Αντίστοιχα με το παραπάνω η τιμή αυτή μπορεί να προστεθεί μέσα από τα windows είτε στο παράθυρο του DOS εκτελώντας την εντολή

  C:\> set path=%path%;%LCCDIR%;

Προσοχή πρέπει να δοθεί σε περίπτωση που έχουν εγκατασταθεί προγράμματα όπως το Visual Studio .NET στον υπολογιστή που θα τρέξει ο compiler, συνήθως οι παραπάνω εφαρμογές προσθέτουν εγγραφές στο περιβάλλον του υπολογιστή όπως οι **include** και **lib,** αυτές πρέπει να σβηστούν κατά τη διάρκεια που χρησιμοποιείται ο compiler. Διαφορετικά μπερδεύεται και δεν μπορεί να τρέξει σωστά. Το σβήσιμο των εγγραφών μπορεί να γίνει σε επίπεδο DOS εκτελώντας τις παρακάτω εντολές

set include= και ***set lib=***

6.3.1 Διαδικασία δημιουργίας προγράμματος

Ο εκτελέσιμος στον XR16 κώδικας δημιουργείται σε δύο βήματα, πρώτα μετατρέπεται ο κώδικας υψηλού επιπέδου σε assembly, αυτό γίνεται με την εντολή:

C:\source-path\> lcc-xr16 -S source.c

όπου **"source-path"** είναι ο κατάλογος που έχει αποθηκευτεί το πρόγραμμα και **source.c** είναι ο πηγαίος κώδικας του προγράμματος που πρέπει να τρέξει στον επεξεργαστή. Η

παραπάνω εντολή παράγει ένα αρχείο της μορφής **“source.s”.** Το αρχείο αυτό περιέχει τον μεταφρασμένο σε assembly κώδικα.

Το επόμενο βήμα της δημιουργίας του προγράμματος είναι να γίνει σύνδεση του κώδικα assembly, που παρήχθη στο προηγούμενο βήμα, με την assembly που χρειάζεται για την αρχικοποίηση του επεξεργαστή και να δημιουργηθεί το τελικό αρχείο. Τα αρχεία assembly για την αρχικοποίηση του επεξεργαστή παρέχονται με τη διανομή του επεξεργαστή. Αυτό γίνεται εκτελώντας την παρακάτω εντολή:

C:\source-path\> xr16 sim.s source.s end.s -hex=source.hex –lst=source.lst

Η παραπάνω εντολή δημιουργεί δύο αρχεία το **source.hex** και το **source.lst.** Στο πρώτο περιέχεται ο κώδικας του προγράμματος σε γλώσσα μηχανής (σε δεκαεξαδικό σύστημα), ενώ στο δεύτερο περιέχονται τα listings του προγράμματος. Τα listings χρειάζονται για να βρεθούν οι διευθύνσεις επιστροφής των διεργασιών του προγράμματος που θα τρέξει στον επεξεργαστή.

6.4 Xilinx ISE

Το ISE της Xilinx χρησιμοποιείται για να γίνεται compile ο κώδικας της Verilog. Μέσω αυτού του προγράμματος έγινε η ανάπτυξη του κώδικα και η βελτιστοποίηση που χρειάστηκε για να προσαρμοστεί στα νέα δεδομένα λειτουργίας του επεξεργαστή. Η έκδοση που χρησιμοποιήθηκε είναι η 7.1i.

6.5 ModelSim XE

Το ModelSim έχει χρησιμοποιηθεί για την προσομοίωση του συστήματος. Μέσω του προγράμματος έχουν εξαχθεί τα αποτελέσματα των πειραμάτων που έγιναν στον επεξεργαστή. Η έκδοση που χρησιμοποιήθηκε στα πειράματα και για τον έλεγχο του συστήματος είναι η ModelSim XE II 5.7g και η XE III 6.0d.

7

Αυτοματοποίηση διαδικασίας

7.1 Εισαγωγή

Σημαντικό βάρος έχει δοθεί στο να γίνει η διαδικασία όσο το δυνατόν πιο εύκολη για τον τελικό χρήστη. Ο τρόπος που έχει επιλεχθεί είναι η δημιουργία ενός μικρού προγράμματος που να μπορεί να κάνει όσες αλλαγές χρειάζονται στα αρχεία της Verilog, έτσι ώστε ο χρήστης να πρέπει απλά να συνθέσει το μοντέλο και να το φορτώσει στο FPGA.

Για να γίνει η διαδικασία ευκολότερη, κυρίως προγραμματιστικά σε επίπεδο εφαρμογής χρήστη, έχουμε κάνει μία παραδοχή. Αυτή είναι ότι η πρώτη συνάρτηση του προγράμματος αποτελεί απαραίτητα μία διεργασία του νέου συστήματος, διαφορετικά θα έπρεπε ο χρήστης να προσθέσει εντολές σε γλώσσα μηχανής με το χέρι.

7.2 Διαδικασία που ακολουθεί ο χρήστης

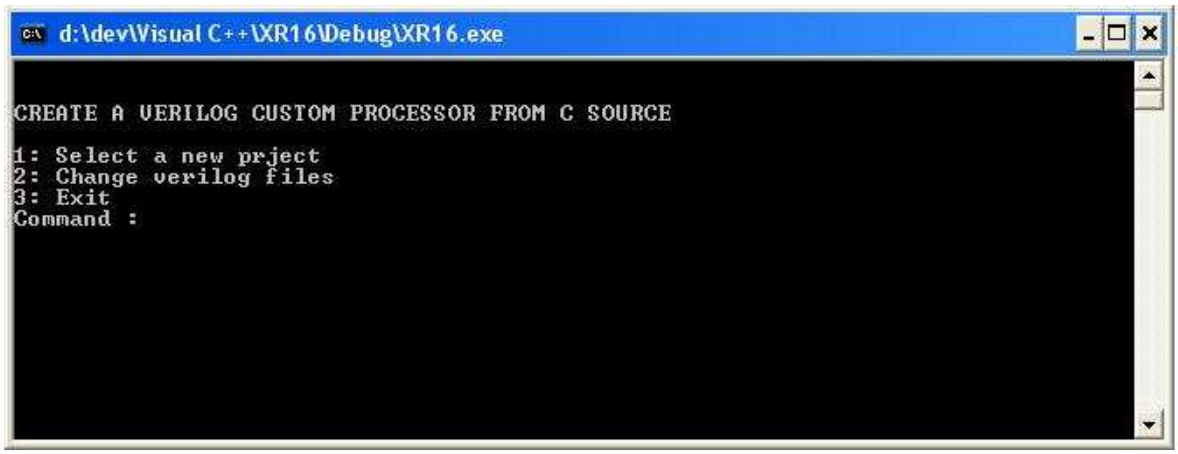
Η διαδικασία που πρέπει να ακολουθήσει ο χρήστης είναι αρκετά απλή. Αρκεί να τρέξει την εφαρμογή και να ακολουθήσει τα βήματα που εμφανίζονται στα μενού. Ο χρήστης σε πρώτη φάση θα πρέπει να δηλώσει στο πρόγραμμα που υπάρχουν τα αρχεία C, LST, HEX του προγράμματος, που θα θέλει τελικά να εκτελεστεί από τον επεξεργαστή. Το πρόγραμμα θα επεξεργαστεί τα αρχεία, θα κάνει ορισμένες ερωτήσεις στο χρήστη και θα δημιουργήσει μία μικρή βάση με τις αλλαγές που πρέπει να γίνουν.

Στο δεύτερο κομμάτι του προγράμματος ο χρήστης μπορεί να ενημερώσει τα αρχεία της Verilog, έτσι ώστε να είναι έτοιμα προς χρήση. Αυτή η διαδικασία χρησιμοποιεί τα δεδομένα που είτε έχουν υπολογισθεί κατά το πρώτο βήμα, είτε τα έχει εισαγάγει ο χρήστης στο ίδιο βήμα. Κατά τη διάρκεια του πρώτου βήματος ο χρήστης μπορεί να δημιουργήσει ένα αρχείο το οποίο θα περιέχει οδηγίες για το ίδιο το πρόγραμμα. Αυτό το αρχείο μπορεί να είναι η είσοδος των δεδομένων στο δεύτερο τμήμα του προγράμματος, αντί ο χρήστης να χρειαστεί να κάνει όλη τη διαδικασία από την αρχή.

7.3 Κατά βήμα χρήση του προγράμματος

Στη συνέχεια αυτής της παραγράφου θα γίνει μία επίδειξη στα βασικά βήματα εκτέλεσης του προγράμματος. Με μερικές εικόνες θα παρουσιαστεί η πορεία που ακολουθείται και θα εξηγηθούν τα βασικά βήματα του προγράμματος.

7.3.1 *Πρώτο μενού επιλογών*



Εικόνα 27 Πρώτο μενού επιλογών

Στην παραπάνω εικόνα φαίνονται οι δύο βασικές επιλογές του προγράμματος, στη μεν πρώτη επιλογή επιλέγεται ένα νέο πρόγραμμα προς εκτέλεση από τον επεξεργαστή, ενώ στη δεύτερη, εφόσον υπάρχει κάποιο αρχείο περιγραφής των αλλαγών ή έχει τελειώσει το πρώτο βήμα, μπορούν να ενημερωθούν τα αρχεία της Verilog.

7.3.2 *Επεξεργασία προγράμματος προς εκτέλεση*

Εικόνα 28 Μενού επεξεργασίας προγράμματος C

Το παραπάνω μενού εμφανίζεται όταν ο χρήστης επιλέξει την πρώτη επιλογή από το πρώτο μενού. Εδώ υπάρχουν τέσσερεις βασικές εντολές. Πρέπει να επισημανθεί ότι τα

βήματα «κλειδώνονται», έτσι ώστε να μπορούν να εκτελεστούν μόνο το ένα μετά το άλλο, με αυτό τον τρόπο ο χρήστης δεν μπερδεύεται και το πρόγραμμα έχει πάντα σωστά δεδομένα προς επεξεργασία. Οι επιλογές αυτού του μενού εξηγούνται στη συνέχεια.

7.3.2.1 Εισαγωγή νέου προγράμματος

Με την πρώτη επιλογή ζητείται από τον χρήστη να δώσει στο πρόγραμμα τη διεύθυνση που μπορεί να βρει το πρόγραμμα που θα εκτελέσει ο επεξεργαστής, δηλαδή εκεί που θα βρει τα αρχεία C, LST, HEX. Εφόσον το πρόγραμμα επιβεβαιώσει ότι η διεύθυνση είναι υπαρκτή ρωτάει το χρήστη το όνομα του αρχείου.

7.3.2.2 Επεξεργασία αρχείων

Η δεύτερη επιλογή δίνει την εντολή στο πρόγραμμα να ξεκινήσει την επεξεργασία. Αναγνωρίζει όλες τις πιθανές διεργασίες που υπάρχουν υπό τη μορφή συναρτήσεων και ρωτάει το χρήστη να επιβεβαιώσει εάν πρόκειται για διεργασίες ή υπορουτίνες που χρησιμοποιούνται από τις κύριες διεργασίες. Όπως αναφέρθηκε και στην αρχή γίνεται η παραδοχή ότι η πρώτη ρουτίνα που ανιχνεύεται αποτελεί μία από τις διεργασίες του προγράμματος. Έτσι ρωτάτε ο χρήστης να απαντήσει με ένα «ναι» ή «όχι» (y/n) εάν η κάθε ρουτίνα αποτελεί και διεργασία του συστήματος.

7.3.2.3 Εισαγωγή στοιχειών από τον χρήστη

Εδώ ζητείται από τον χρήστη να δώσει ορισμένες τιμές για της διεργασίες που έχει επιλέξει. Αυτές είναι το μέγεθος της στοίβας που χρειάζεται να δεσμευτεί για την απρόσκοπτη λειτουργία της διεργασίας. Επίσης ζητείται από τον χρήστη να δώσει το

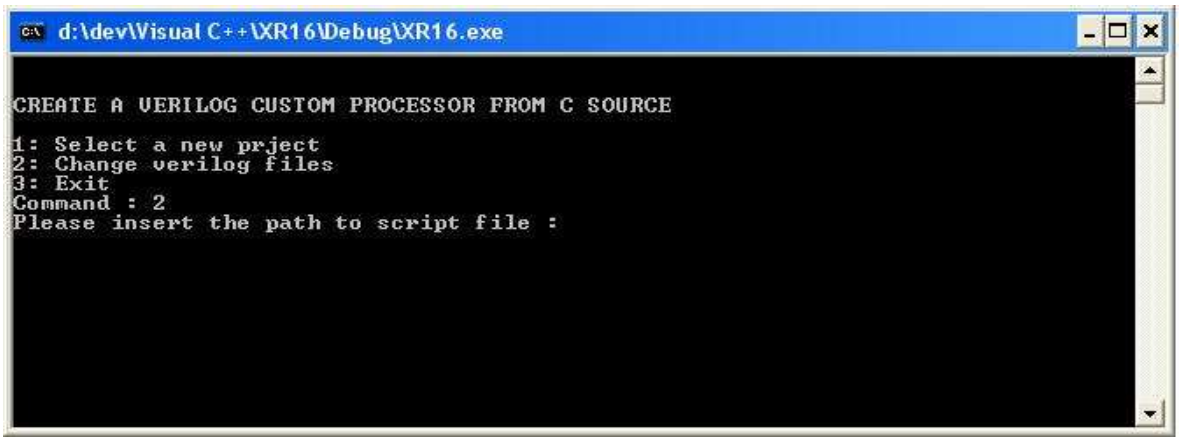
ρυθμό με τον οποίο πρέπει να καλούνται οι διεργασίες, υπό τη μορφή περιόδου. Βάση αυτών των τιμών θα υπολογιστούν οι προτεραιότητες και θα δημιουργηθούν τα αρχεία.

7.3.2.4 Δημιουργία αρχείου περιγραφής συστήματος

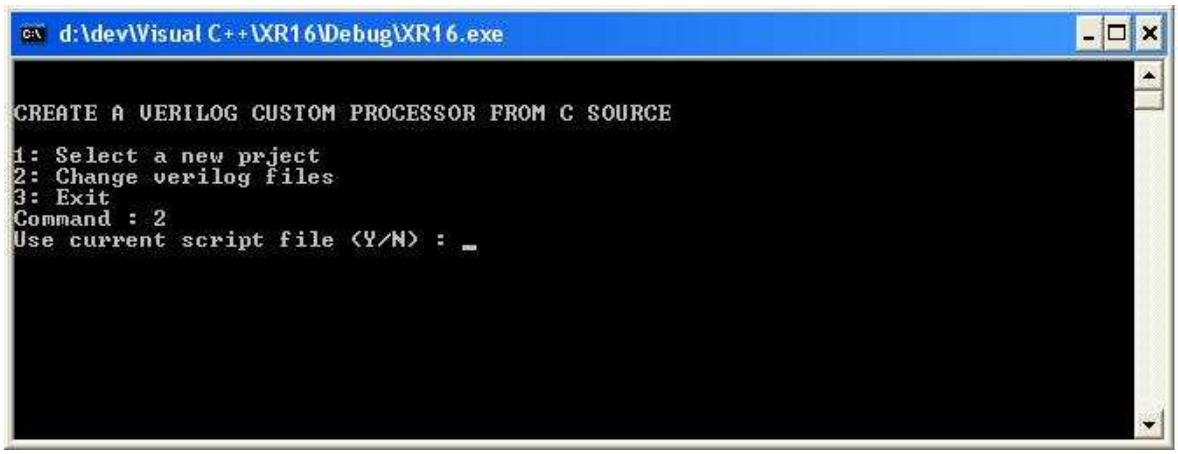
Με την τελευταία επιλογή του μενού ο χρήστης δημιουργεί το αρχείο περιγραφής του συστήματος, αυτό χρειάζεται στο δεύτερο βήμα του προγράμματος, ενώ χρειάζεται και εάν ο χρήστης θέλει να επαναλάβει τη διαδικασία. Το πρόγραμμα δημιουργεί το αρχείο περιγραφής εκεί που υπάρχουν τα αρχεία του προγράμματος προς εκτέλεση.

7.3.3 Ενημέρωση αρχείων Verilog

Με αυτή την επιλογή ο χρήστης ενημερώνει τα αρχεία Verilog, του πηγαίου κώδικα του επεξεργαστή έτσι ώστε να δημιουργηθεί το συνθέσιμο μοντέλο. Αυτό το τμήμα του προγράμματος συνεργάζεται με το πρώτο τμήμα, με την έννοια ότι εάν το αρχείο περιγραφής έχει δημιουργηθεί στο πρώτο βήμα, τότε θα ρωτήσει το χρήστη εάν πρέπει να χρησιμοποιήσει αυτό ή κάποιο άλλο. Εάν δεν έχει δημιουργηθεί τότε θα ρωτήσει το χρήστη για τη διεύθυνση του έτσι ώστε να το ανοίξει.



Εικόνα 29 Εισαγωγή διεύθυνσης αρχείου περιγραφής συστήματος



Εικόνα 30 Αποδοχή ανοιγμένου αρχείου περιγραφής ή επιλογή νέου

Όταν τελικά ο χρήστης επιλέξει το αρχείο περιγραφής του συστήματος που θέλει τότε το πρόγραμμα ρωτάει τον χρήστη τη διεύθυνση των αρχείων περιγραφής του επεξεργαστή.

Εικόνα 31 Εισαγωγή διεύθυνσης αρχείων επεξεργαστή

Κατά τη διάρκεια ενημέρωση των αρχείων του επεξεργαστή επηρεάζονται τα ακόλουθα αρχεία:

1. xr16.v
2. RegSet.v
3. register_files.v
4. timer.v
5. ProcessRunning.v

8 Έλεγχος

Σε αυτό το κεφάλαιο θα αναλυθεί ο τρόπος με τον οποίο ελέγχθηκε το τελικό σύστημα και θα παρουσιαστούν τα αποτελέσματα που προέκυψαν.

Για τον έλεγχο του συστήματος φτιάχτηκαν εφαρμογές με διαφορετικό αριθμό διεργασιών η κάθε μία, με τις εφαρμογές αυτές έγινε η σύνθεση του επεξεργαστή από την αρχή και συγκρίθηκαν τεχνικά χαρακτηριστικά του αποτελέσματος όπως ο χώρος που καταλαμβάνει το τελικό σύστημα στην FPGA όσο αυξάνεται ο αριθμός των διεργασιών και η μέγιστη ταχύτητα λειτουργίας του επεξεργαστή

8.1 Εγκατάσταση απαραίτητων προγραμμάτων

Πρώτο στάδιο για τον έλεγχο του συστήματος είναι η εγκατάσταση των απαραίτητων προγραμμάτων.

8.2 Μεθοδολογία Ελέγχου

Η μεθοδολογία ελέγχου της εφαρμογής είναι κοινή και για τα τέσσερα διαφορετικά προγράμματα που θα δοκιμαστούν. Αρχικά γράφεται ο κώδικας της εφαρμογής σε C. Οι διεργασίες των προγραμμάτων αποτελούνται είτε από ατέρμων Loop που εσωτερικά

γίνονται διάφορες πράξεις, είτε από loop που τελειώνουν. Στη δεύτερη περίπτωση οι διαδικασίες ξεκινάνε ξανά όταν τελειώνει η περίοδος που θέτουμε κατά την υλοποίηση του συστήματος. Έπειτα για κάθε ένα πείραμα ελέγχου γίνεται compile το πρόγραμμα χρησιμοποιώντας το compiler που παρέχεται με τον επεξεργαστή (lcc-xr16 και xr16). Στη συνέχεια χρησιμοποιώντας τα αποτελέσματα του compiling ενημερώνουμε τα αρχεία του συστήματος και γίνεται η σύνθεση του επεξεργαστή.

8.3 Αναλυτική παρουσίαση του ελέγχου

Στη συνέχεια θα παρουσιαστούν τα προγράμματα με τα οποία ελέγχθηκε το σύστημα καθώς και τα πειραματικά αποτελέσματα που εξήχθησαν από κάθε ένα από αυτά.

8.3.1 Πρώτο πρόγραμμα – Μία διεργασία

Το πρώτο πρόγραμμα αποτελείται από μία διεργασία. Με αυτό το πρόγραμμα ο επεξεργαστής θα έχει ένα set από register και όλη την επεξεργαστική ισχύ θα την έχει η μία και μοναδική διεργασία.

```
void process1() {
        int i=0;
        int k=0;
        int key;
        do {
                i=i+1 ;
                k=k-1 + i + 3;

                key = i + k;
        } while(1);
 }
```

Το πρώτο πρόγραμμα ελέγχου είναι αρκετά απλό, στη συνέχεια παρατίθενται τα στοιχεία από τη σύνθεση του κυκλώματος.

HDL Synthesis Report
Macro Statistics
LUT RAMs : 20
16x1-bit single-port distributed RAM : 16
16x8-bit dual-port distributed RAM : 4
Adders/Subtractors : 4
14-bit adder : 1
16-bit adder : 1
17-bit adder carry out : 1
18-bit subtractor : 1
Counters : 1
2-bit down counter : 1
Registers : 78
1-bit register : 67
16-bit register : 8
17-bit register : 1
5-bit register : 1
8-bit register : 1
Comparators : 1
4-bit comparator equal : 1
Multiplexers : 20
1-bit 16-to-1 multiplexer : 1
1-bit 4-to-1 multiplexer : 14
16-bit 4-to-1 multiplexer : 3
4-bit 4-to-1 multiplexer : 2
Tristates : 129
1-bit tristate buffer : 128
8-bit tristate buffer : 1
Xors : 5
1-bit xor2 : 3
1-bit xor4 : 1
16-bit xor2 : 1

Για την επιλεγμένη FPGA το κύκλωμα για μία διεργασία καταλαμβάνει το παρακάτω ποσοστά από τα διαθέσιμα στοιχεία της.

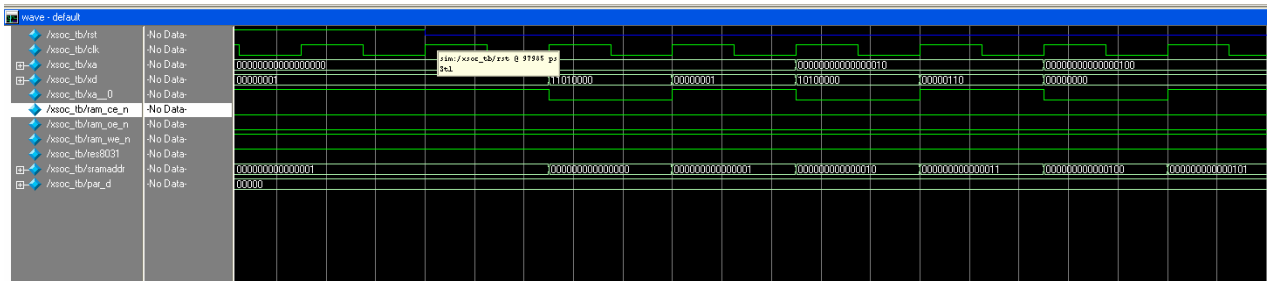
Selected Device: 2s100tq144-6
Number of Slices: 250 out of 1200 20%
Number of Slice Flip Flops: 217 out of 2400 9%
Number of 4 input LUTs: 453 out of 2400 18%
Number used as logic: 373
Number used as RAMs: 80
Number of bonded IOBs: 32 out of 96 33%
Number of GCLKs: 2 out of 4 50%

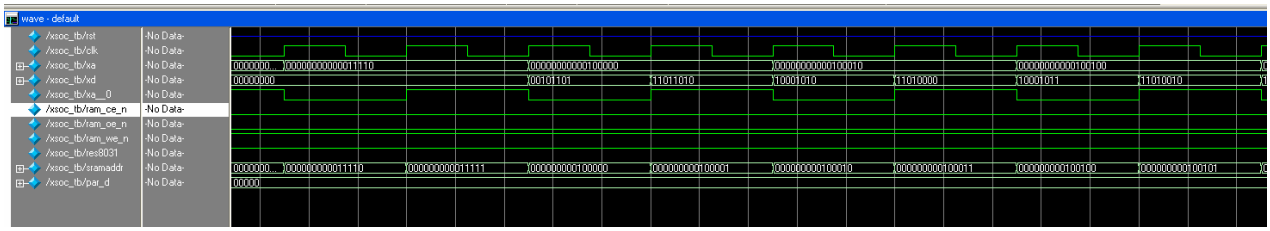
Τέλος το κύκλωμα έχει τις παρακάτω δυνατότητες όσον αφορά την ταχύτητα λειτουργίας.

Timing Summary
Speed Grade: -6
Minimum period: 22.018ns (Maximum Frequency: 45.417MHz)
Minimum input arrival time before clock: 12.274ns
Maximum output required time after clock: 15.718ns
Maximum combinational path delay: 12.690ns

Τα τελευταία στοιχεία της σύνθεσης θα είναι μέτρο σύγκρισης καθώς αυξάνονται οι διεργασίες που τρέχουν στο σύστημα, έτσι ώστε να δούμε την επίδραση των επιπλέον κυκλωμάτων που έχουν προστεθεί.

Επίσης παραθέτουμε και μερικές εικόνες από τον προσομοιωτή του κυκλώματος:





8.3.2 Δεύτερο πρόγραμμα – Δύο διεργασίες

Τα δεύτερο πρόγραμμα ελέγχου αποτελείται από 2 διεργασίες. Η πρώτη διεργασία είναι αυτή με την υψηλότερη προτεραιότητα. Σε αυτό το πρόγραμμα η πρώτη διεργασία θα τρέξει για ορισμένο χρονικό διάστημα, μόλις τελειώσει το σύστημα θα εκτελέσει τη 2η διεργασία, που είναι και η idle. Επιπλέον έχουν θέσει μία περίοδο 512 κύκλων ρολογιού, έπειτα από την οποία θα ξεκινήσει εκ' νέου η εκτέλεση της.

```
void process1() {
        int i=0;
        int k=0;
        int key;

        do {
                i=i+1 ;
                k=k-1 + i + 3;
```

```
                    key = i + k;
            } while(i<50);
}

void process2() {
            int i=0;
            int k=0;
            int key = 0;
            int a = 32000;
            int j;

            do {
                    i=i+1 ;
                    k=k-1 + i + 3;

                    key = i + k;


                    for (j=0;j<10;j++) {
                            a = a-1;
                            i = i + a ;
                    }
            } while(1);
}
```

Τα στοιχεία της σύνθεσης του νέου κυκλώματος:

HDL Synthesis Report
Macro Statistics
LUT RAMs : 20
16x1-bit single-port distributed RAM : 16
32x8-bit dual-port distributed RAM : 4
Adders/Subtractors : 4
14-bit adder : 1
16-bit adder : 1
17-bit adder carry out : 1
18-bit subtractor : 1
Counters : 2
2-bit down counter : 1
8-bit down counter : 1
Registers : 80
1-bit register : 69
16-bit register : 8
17-bit register : 1
5-bit register : 1
8-bit register : 1
Comparators : 1

4-bit comparator equal	: 1
# Multiplexers	: 20
1-bit 16-to-1 multiplexer	: 1
1-bit 4-to-1 multiplexer	: 14
16-bit 4-to-1 multiplexer	: 3
4-bit 4-to-1 multiplexer	: 2
# Tristates	: 129
1-bit tristate buffer	: 128
8-bit tristate buffer	: 1
# Xors	: 5
1-bit xor2	: 3
1-bit xor4	: 1
16-bit xor2	: 1

Αντίστοιχα με το πρώτο κύκλωμα τα ποσοστά χρησιμοποίησης των διαθέσιμων στοιχείων της χρησιμοποιούμενης FPGA, είναι τα παρακάτω:

Selected Device : 2s100tq144-6				
Number of Slices:	320	out of	1200	26%
Number of Slice Flip Flops:	230	out of	2400	9%
Number of 4 input LUTs:	629	out of	2400	26%
Number used as logic: 485				
Number used as RAMs: 144				
Number of bonded IOBs:	32	out of	96	33%
Number of GCLKs:	2	out of	4	50%

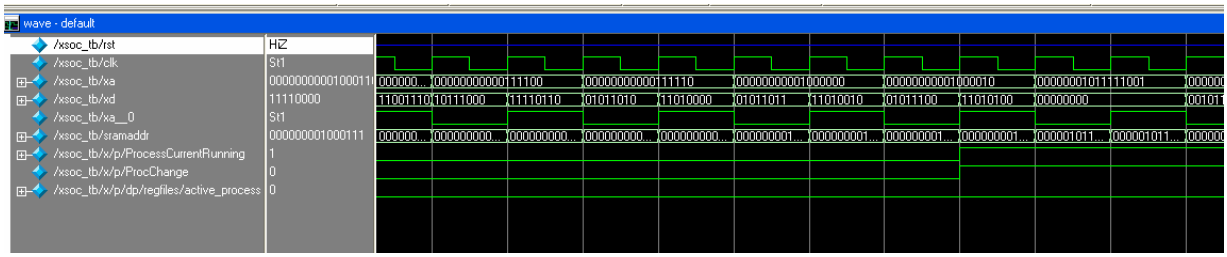
Τέλος όσον αφορά την ταχύτητα λειτουργίας του νέου κυκλώματος τα στοιχεία μετά τη σύνθεση είναι τα παρακάτω:

Timing Summary
Speed Grade: -6
Minimum period: 33.432ns (Maximum Frequency: 29.911MHz)
Minimum input arrival time before clock: 13.444ns
Maximum output required time after clock: 25.661ns
Maximum combinational path delay: 12.915ns

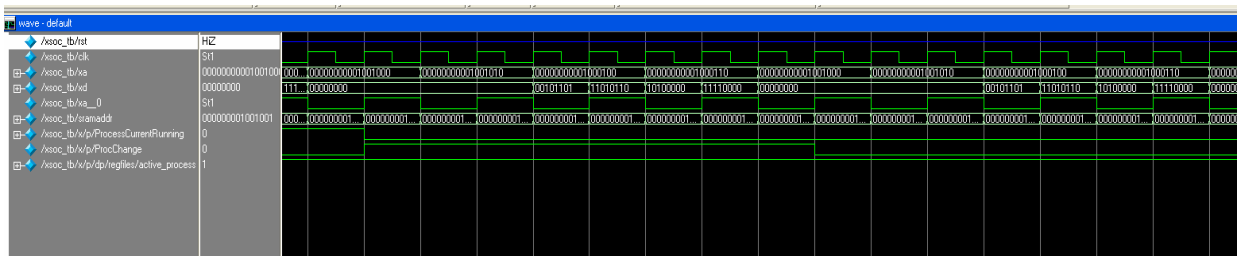
Παρατηρούμε εδώ ότι σε σχέση με το αρχικό κύκλωμα με τη μία διεργασία υπάρχει σημαντική πτώση στη μέγιστη ταχύτητα λειτουργίας του τελικού κυκλώματος. Αυτό οφείλεται κυρίως στην

αυξημένη πολυπλοκότητα των κυκλωμάτων ελέγχου που προστίθενται για τον έλεγχο των διεργασιών.

Στην επόμενη φωτογραφία από τον προσομοιωτή φαίνεται η αλλαγή της τρέχουσας διεργασίας, με το τέλος της πρώτης. Το σύστημα εκτελεί αυτόματα τη δεύτερη διαδικασία, που είναι από default η idle.



Έπειτα βλέπουμε ότι το σύστημα ξεκινάει αυτόματα την πρώτη διεργασία, αφού έχει τελειώσει η προκαθορισμένη περίοδος και το σύστημα αυστηρά ξεκινάει την πρώτη διεργασία ξανά.



8.3.3 Τρίτο πρόγραμμα – Τέσσερις διεργασίες

Το τελευταίο πρόγραμμα ελέγχου αποτελείται από τέσσερις διεργασίας. Η πηγαίος κώδικας της εφαρμογής ελέγχου είναι ο παρακάτω. Σε αυτό το πείραμα ελέγχου έχουμε θέσει αντίστοιχα περιόδους εκτέλεσης : 512, 768 και 592 κύκλους ρολογιού.

```
void process1() {
        int i=0;
        int k=0;
        int key;

        do {
                i=i+1 ;
                k=k-1 + i + 3;

                key = i + k;
```

```
        } while(i < 50);
}

void process2() {
        int i=0;
        int k=0;
        int key = 0;
        int a = 32000;
        int j;

        do {
                i=i+1 ;
                k=k-1 + i + 3;
                key = i + k;
                for (j=0;j<10;j++) {
                        a = a-1;
                        i = i + a ;
                }
        } while(i < 200);
}

void process3() {
        int i=0;
        int k=0;
        int l=0;
        int key = 0;

        do {
                i=i+1 ;
                k=k-1+ i;
                key = i + k + l;
                        l=l+1;
        } while(i < 150);
}

void process4() {
        int i=0;
        int k=0;
        int l=0;
        int key = 0;

        do {
                i=i+1;
                key = i + k;
                k=k-1 + i + 3;
                l=l+4;
                k =k+1;
                key = key + l - 2;
        } while(1);
}
```

Τα στοιχεία της σύνθεσης του νέου κυκλώματος:

HDL Synthesis Report
Macro Statistics
LUT RAMs : 20
16x1-bit single-port distributed RAM : 16
64x8-bit dual-port distributed RAM : 4
Adders/Subtractors : 4
14-bit adder : 1
16-bit adder : 1
17-bit adder carry out : 1
18-bit subtractor : 1
Counters : 4
2-bit down counter : 1
8-bit down counter : 3
Registers : 82
1-bit register : 69
16-bit register : 8
17-bit register : 1
2-bit register : 2
5-bit register : 1
8-bit register : 1
Comparators : 3
2-bit comparator equal : 1
2-bit comparator not equal : 1
4-bit comparator equal : 1
Multiplexers : 20
1-bit 16-to-1 multiplexer : 1
1-bit 4-to-1 multiplexer : 14
16-bit 4-to-1 multiplexer : 3
4-bit 4-to-1 multiplexer : 2
Tristates : 129
1-bit tristate buffer : 128
8-bit tristate buffer : 1
Xors : 4
1-bit xor2 : 2
1-bit xor4 : 1
16-bit xor2 : 1

Αντίστοιχα με τα προηγούμενα κυκλώματα τα ποσοστά χρησιμοποίησης των διαθέσιμων στοιχείων της χρησιμοποιούμενης FPGA, είναι τα παρακάτω:

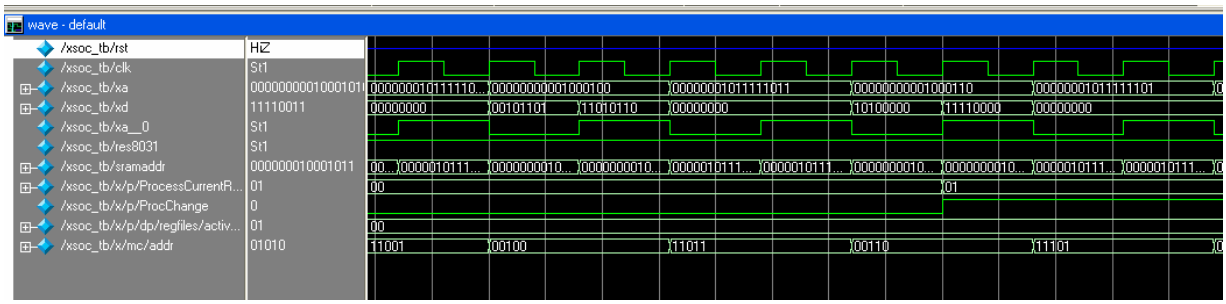
Selected Device : 2s100tq144-6				
Number of Slices:	316	out of	1200	26%
Number of Slice Flip Flops:	233	out of	2400	9%
Number of 4 input LUTs:	811	out of	2400	33%
Number used as logic: 539				
Number used as RAMs: 272				
Number of bonded IOBs:	32	out of	96	33%
Number of GCLKs:	2	out of	4	50%

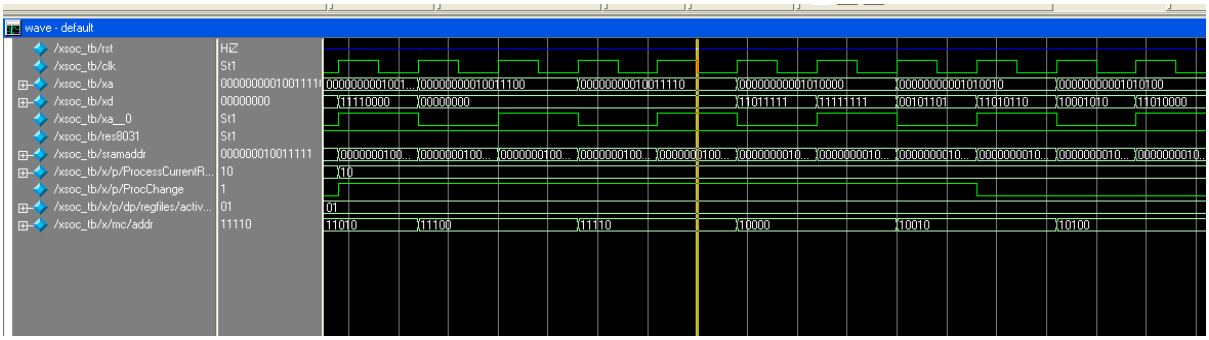
Τέλος όσον αφορά την ταχύτητα λειτουργίας του νέου κυκλώματος τα στοιχεία μετά τη σύνθεση είναι τα παρακάτω:

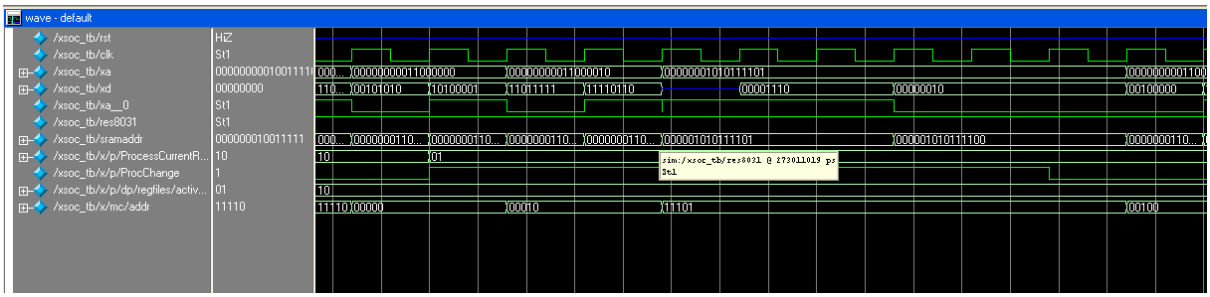
Timing Summary
Speed Grade: -6
Minimum period: 31.812ns (Maximum Frequency: 31.435MHz)
Minimum input arrival time before clock: 13.624ns
Maximum output required time after clock: 28.676ns
Maximum combinational path delay: 13.230ns

Παρατηρείται εδώ ότι η ταχύτητα του επεξεργαστή είναι σε παραπλήσια επίπεδα με την ταχύτητα όταν είχαμε δύο διεργασίες, επομένως τα επιπλέον στοιχεία που προστίθενται για τον έλεγχο των επιπλέων δύο διεργασιών δεν προσθέσανε επιπλέον καθυστερήσεις στο σύστημα.

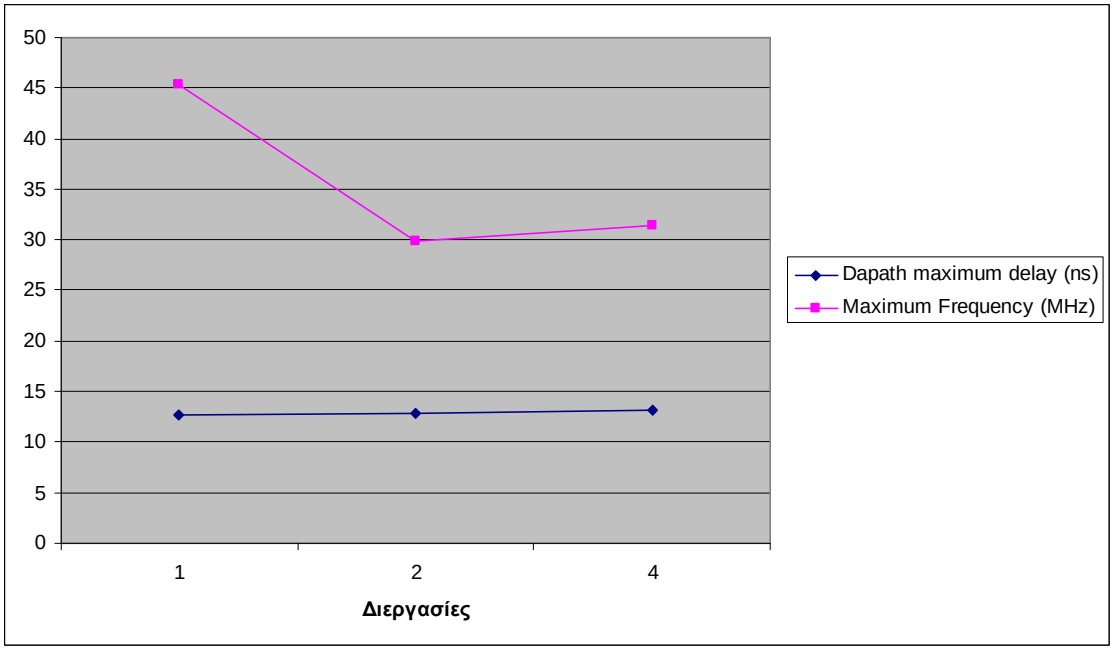
Στη συνέχεια παρατίθενται ορισμένες εικόνες από τον προσομοιωτή κατά την αλλαγή των ενεργών διεργασιών στο σύστημα.







Τέλος παρατίθεται μία γραφική παράσταση στη οποία φαίνεται η συσχέτιση της μέγιστης ταχύτητας λειτουργίας του επεξεργαστή καθώς και της μέγιστης καθυστέρησης στο data path, καθώς αυξάνονται οι ενεργές διεργασίες του συστήματος από μία έως τέσσερεις.



9

Επίλογος

9.1 Σύνοψη και συμπεράσματα

Στην εργασία αυτή περιγράφτηκε πως μπορεί να αλλαχτεί η αρχιτεκτονική ενός επεξεργαστή τεχνολογίας RISC, έτσι ώστε να μειωθεί ο χρόνος που χρειάζεται για την αλλαγή μεταξύ των ενεργών διεργασιών. Αυτό επιτεύχθηκε με τη μεταφορά της διαδικασίας του Context Switch από το λειτουργικό σύστημα, στο υλικό του επεξεργαστή.

Στο δεύτερο στάδιο της εργασίας περιγράφτηκε πως με κάποιες αλλαγές ο ίδιο επεξεργαστής μπορεί να ανταπεξέλθει σε ένα απαιτητικό σύστημα πραγματικού χρόνου, με την προσθήκη μίας τιμής προτεραιότητας για κάθε μία από τις διεργασίες. Έτσι το σύστημα πλησιάζει ποιο κοντά σε ανάγκες πραγματικών συστημάτων εξειδικευμένων εφαρμογών, αφού πλέον κανένας δε χρησιμοποιεί ένα σύστημα με «τυφλή» αλλαγή μεταξύ των διεργασιών για γενική χρήση.

Επιπλέον πρέπει να αναφερθεί ότι το σύστημα θα μπορούσε να εμπλουτιστεί με τη μεταφορά περισσότερων στοιχείων του λειτουργικού συστήματος σε επίπεδο υλικού. Επίσης θα μπορούσε να βελτιωθεί με τον εμπλουτισμό της γλώσσας προγραμματισμού,

για τη δημιουργία των εφαρμογών προς εκτέλεση, με μακροεντολές για τη δυναμική δημιουργία νέων διεργασιών κατά τη διάρκεια της εκτέλεσης του προγράμματος.

Στοιχεία του λειτουργικού που θα μπορούσαν να μεταφερθούν σε επίπεδο υλικού είναι οι σηματοφορείς για την εκτέλεση των εφαρμογών ή ο μετρητής του προγράμματος. Μεταφέροντας τα διάφορα στοιχεία σε επίπεδο υλικού προφανώς θα υπάρχει βελτίωση στην ταχύτητα, ωστόσο δημιουργούνται διάφορα προβλήματα.

Ένα σημαντικό πρόβλημα για την πιθανή μεταφορά όλου του λειτουργικού σε επίπεδο υλικού, είναι ότι στα λειτουργικά συστήματα οι διεργασίες που εκτελούνται είναι δυναμικές, με αποτέλεσμα να μην μπορεί να γίνει ασφαλή πρόβλεψη εκ των προτέρων για τις ανάγκες σε μνήμη ή προσωρινό χώρος αποθήκευσης. Έτσι είναι σχεδόν αδύνατον να γίνει μία πλήρης μεταφορά ενός λειτουργικού συστήματος στο υλικό, όπως τα λειτουργικά δουλεύουν σήμερα εκτός και αν τεθούν περιορισμοί στις δυνατότητες του, π.χ. στον αριθμό των διεργασιών που μπορούν να τρέξουν, στη μνήμη που μπορεί να διαθέσει σε αυτές κτλ.

10

Βιβλιογραφία

[RM95] Rajeev Madhavan, Quick Reference for Verilog HDL, AMBIT Design Systems, Inc.

[FR01] Finbarr O'Regan (finbarr@ee.ucd.ie), Verilog Synthesis Methodology

[DH97] Dr. Daniel, C. Hycle, Computer Science Department, Bucknell University Lewinsburg, CSCI 320 Computer Architecture Handbook on Verilog HDL.

[LL73] C. L. Liu , Massachusetts Institute of Technology, James W. Layland, Jet Propulsion Laboratory, California Institute of Technology, Scheduling Algorithms for Multiprogramming in Hard-Real-Time Environment, Journal of the Association for Computing Machinery, Vol. 20, No. 1, January 1973

[SMA+] S. Stergiou, G. Manis, G. Asimenos, A. Sotiropoulos and G. Papakonstantinou, National Technical University of Athens Dept. of Electrical and Computer Engineering Computing Systems Laboratory, A Novel Reconfigurable Processor Architecture with Hardware Assisted OS Kernel

[G2000] J. Gray. Building a Risc System in an FPGA:

Part 1: Tools, instruction set and datapath;

Part 2: Pipeline and control unit design;

Part 3: System-on-a-chip design. Circuit Cellar Magazine, 116-118, Mar., Apr., May 2000

[X10000] Xilinx, Inc. DS001-2, Spartan-II 2.5V FPGA Family: Functional Description,

[XAPP173] Using Block SelectRAM+ Memory in Spartan-II FPGAs

[X20001] Stanford University, EE183 Advanced Logic Design Laboratory, Generating a VGA monitor signal using the XSA-100board