ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΠΟΔΟΤΙΚΗ ΔΙΑΧΕΙΡΙΣΗ ΜΝΗΜΗΣ ΓΙΑ ΔΙΚΤΥΑ ATM ΥΨΗΛΗΣ ΤΑΧΥΤΗΤΑΣ

Παναγιώτης Η. Καρακωνσταντής

Μεταπτυχιακή Εργασία

Ηράκλειο, Οκτώβριος 1998

Αποδοτική Διαχείριση Μνήμης για Δίκτυα ATM Υψηλής Ταχύτητας

Μεταπτυχιακή εργασία
που υποβλήθηκε στην επιτροπή μεταπτυχιακών σπουδών
του τμήματος Επιστήμης Υπολογιστών
της σχολής Θετικών Επιστημών
του Πανεπιστημίου Κρήτης
σε μερική εκπλήρωση των απαιτήσεων για την απόκτηση του
ΔΙΠΛΩΜΑΤΟΣ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΙΔΙΚΕΥΣΗΣ
Ηράκλειο, Οκτώβριος 1998

Συγγραφέας:

Παναγιώτης Η. Καρακωνσταντής
Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

Δημήτριος Σερπάνος, Επίκουρος Καθηγητής
(Επόπτης)

Μανόλης Κατεβαίνης, Καθηγητής
(Μέλος)

Γεώργιος Δ. Σταμούλης, Επίκουρος Καθηγητής
(Μέλος)

Δεκτή:

Πάνος Κωνσταντόπουλος, Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

ΑΠΟΔΟΤΙΚΗ ΔΙΑΧΕΙΡΙΣΗ ΜΝΗΜΗΣ ΓΙΑ ΔΙΚΤΥΑ ATM ΥΨΗΛΗΣ ΤΑΧΥΤΗΤΑΣ

Παναγιώτης Η. Καρακωνσταντής
Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

ΠΕΡΙΛΗΨΗ

Το ATM είναι μια τεχνολογία δικτύων βασισμένη σε συνδέσεις που υποστηρίζει μεταφορά δεδομένων, εικόνας και φωνής. Η δυνατότητά της να πολυπλέκει ένα μεγάλο αριθμό από εικονικές συνδέσεις πάνω από τον ίδιο φυσικό σύνδεσμο εισόδου/εξόδου αυξάνει τις απαιτήσεις επεξεργασίας και διαχείρισης της μνήμης αποθήκευσης στα συστήματα ATM. Αυτό το πρόβλημα οξύνεται καθώς αυξάνεται η ταχύτητα και ο αριθμός των συνδέσμων εισόδου/εξόδου. Σε αυτή την εργασία αναλύουμε τις απαιτήσεις για διαχείριση μνήμης αποθήκευσης όπως αυτές καθορίζονται από τις κύριες λειτουργίες του ATM (Κατακερματισμός και Επανασύνδεση, Έλεγχος Ροής, Επιλεκτική Απόρριψη κυττάρων) και προσδιορίζουμε ένα βασικό σύνολο από λειτουργίες των οποίων η γρήγορη εκτέλεση οδηγεί σε αποδοτικά υποσυστήματα μνήμης κατάλληλα για οποιοδήποτε σύστημα ATM (μεταγωγέα ή κάρτα δικτύου). Προτείνουμε αρχιτεκτονικές υλικού που υλοποιούν το σύνολο αυτό των λειτουργιών και μια υλοποίηση αυτών σε λογισμικό κατάλληλη για συστήματα που εμπεριέχουν μικροεπεξεργαστές (embedded systems). Τέλος, παρουσιάζουμε μετρήσεις που ελήφθησαν εκτελώντας το λογισμικό στο μικροεπεξεργαστή της INTEL i960 και αποτιμούμε την απόδοση και το κόστος αυτών των αρχιτεκτονικών έτσι ώστε να μπορεί να επιλέξει κάποιος την καταλληλότερη υλοποίηση για τις ανάγκες του συστήματος που σχεδιάζει.

Efficient Memory Management for high-speed ATM networks

Panagiotis I. Karakonstantis
Master of Science Thesis

Department of Computer Science
University of Crete

ABSTRACT

Asynchronous Transfer Mode (ATM) is a connection-oriented networking technology that supports transfer of data, video and voice. The ability to multiplex a large number of connections over a single physical link places high requirements for processing and memory management in ATM systems. The problem becomes more acute as ATM systems scale to both higher speeds and increased number of links. In this work, we analyze the memory management requirements imposed by the main ATM functions (segmentation and reassembly, flow control and selective discard) and identify a core set of operations whose fast execution leads to efficient memory subsystems for generic ATM systems (switches or adapters). We propose hardware architectures that implement this set of operations as well as a software implementation suitable for embedded systems. Finally, we present performance and cost measurements of the various implementations so that one can choose the implementation most suitable for one's target system.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω πρώτα από όλα τον επόπτη καθηγητή μου κ. Δημήτρη Σερπάνο, για την πολύτιμη καθοδήγησή του στην διάρκεια των μεταπτυχιακών μου σπουδών καθώς και για την βοήθειά του στην ανάπτυξη της επιστημονικής μου γνώσης. Επίσης ευχαριστώ τα μέλη της εισηγητικής επιτροπής κ. Μανόλη Κατεβαίνη και κ. Γεώργιο Δ. Σταμούλη για τις εποικοδομητικές τους παρατηρήσεις. Θα ήταν παράλειψη να μην ευχαριστήσω τον κ. Μανόλη Κατεβαίνη και για τις κρίσιμες επισημάνσεις που έκανε στις τακτικές εβδομαδιαίες συναντήσεις του MuQPro I.

Επίσης ευχαριστώ το Πανεπιστήμιο Κρήτης και πιο συγκεκριμένα το τμήμα Επιστήμης Υπολογιστών για τις γνώσεις και την υλικοτεχνική υποδομή που μου παρείχε καθώς και το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας (τομέα Αρχιτεκτονικής Υπολογιστών και Συστημάτων VLSI) για την οικονομική και τεχνική υποστήριξη.

Θα ήθελα να ευχαριστήσω ιδιαίτερα όλα τα μέλη του τομέα Αρχιτεκτονικής Υπολογιστών και Συστημάτων VLSI για την προθυμία να βοηθήσουν σε κάθε μορφής τεχνικά θέματα καθώς και το σύνολο των μεταπτυχιακών συμφοιτητών μου για το κλίμα συνεργασίας και αλληλεγγύης που ποτέ δεν έλλειψε.

Περισσότερο από όλους θα ήθελα να ευχαριστήσω τους γονείς μου που για τόσα χρόνια σπουδών (και όχι μόνο) με στήριξαν ηθικά και οικονομικά σε εύκολες αλλά κυρίως σε δύσκολες στιγμές. Τέλος θα ήθελα να ευχαριστήσω το κορίτσι μου, την Αλίκη, που ήταν πάντα δίπλα μου όταν την χρειάστηκα.

Περιεχόμενα

ΠΕΡΙΛΗΨΗ **iii**

ABSTRACT **v**

Ευχαριστίες **vii**

1 ΕΙΣΑΓΩΓΗ **1**

2 ΜΟΝΤΕΛΟ ΚΑΙ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ ΣΥΣΤΗΜΑΤΩΝ ATM **5**

2.1 ΛΕΙΤΟΥΡΓΙΑ ΚΑΙ ΕΦΑΡΜΟΓΕΣ ΤΟΥ ΜΟΝΤΕΛΟΥ 8

2.2 ΛΕΙΤΟΥΡΓΙΕΣ ΕΝΟΣ ΣΥΣΤΗΜΑΤΟΣ ATM 10

2.2.1 ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ ΚΑΙ ΕΠΑΝΑΣΥΝΔΕΣΗ (SAR) 11

2.2.2 ΕΛΕΓΧΟΣ ΡΟΗΣ ΒΑΣΙΣΜΕΝΟΣ ΣΤΟΝ ΡΥΘΜΟ ΜΕΤΑΔΟΣΗΣ (Rate-based flow control) 12

2.2.3 ΕΛΕΓΧΟΣ ΡΟΗΣ ΒΑΣΙΣΜΕΝΟΣ ΣΕ ΕΙΣΙΤΗΡΙΑ (Credit-based flow control) 12

2.2.4 ΕΠΙΛΕΚΤΙΚΗ ΑΠΟΡΡΙΨΗ ΚΥΤΤΑΡΩΝ (Selective Discard) 14

3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΔΙΑΧΕΙΡΙΣΗΣ ΟΥΡΩΝ **17**

3.1 ΒΑΣΙΚΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ 20

3.2 ΚΑΘΟΡΙΣΤΙΚΕΣ ΑΠΟΦΑΣΕΙΣ ΤΗΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ 22

4 ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΔΙΑΧΕΙΡΙΣΤΗ ΟΥΡΩΝ ΣΕ ΥΛΙΚΟ **27**

4.1 Η ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΔΙΑΧΕΙΡΙΣΤΗ ΟΥΡΩΝ MuQPro I 28

4.1.1 ΔΙΕΠΑΦΗ (INTERFACE) 30

4.1.2 ΤΟ ΣΥΝΟΛΟ ΕΝΤΟΛΩΝ 31

4.1.3 ΕΣΩΤΕΡΙΚΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ 32
4.1.4 ΑΠΑΙΤΗΣΕΙΣ ΑΠΟ ΤΗΝ ΕΞΩΤΕΡΙΚΗ SRAM ΚΑΙ ΤΗΝ ΕΞΩΤΕΡΙΚΗ ΣΥΣΚΕΥΗ 37
4.1.5 ΣΥΝΕΝΩΝΟΝΤΑΣ ΔΥΟ ΔΙΑΧΕΙΡΙΣΤΕΣ ΟΥΡΩΝ 37
4.1.6 ΚΟΣΤΟΣ ΚΑΙ ΑΠΟΔΟΣΗ 42
4.1.7 ΕΛΕΓΧΟΣ (TESTING) ΤΟΥ ΔΙΑΧΕΙΡΙΣΤΗ ΟΥΡΩΝ 43
4.2 Η ‘‘ΠΛΗΡΩΣ ΕΠΕΚΤΑΣΙΜΗ’’ ΥΛΟΠΟΙΗΣΗ 44
4.3 Η ‘‘ΠΛΗΡΩΣ ΠΑΡΑΛΛΗΛΗ’’ ΥΛΟΠΟΙΗΣΗ 45

5 ΥΛΟΠΟΙΗΣΗ ΔΙΑΧΕΙΡΙΣΗΣ ΟΥΡΩΝ ΣΕ ΛΟΓΙΣΜΙΚΟ 47
5.1 ΤΟ ΠΕΡΙΒΑΛΛΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ 48
5.2 Η ΛΕΙΤΟΥΡΓΙΑ ENQ ΜΕ ΚΑΝΟΝΙΚΗ ΕΙΣΟΔΟ 48
5.3 ΜΙΚΤΕΣ ΛΕΙΤΟΥΡΓΙΕΣ ENQ/DEQ ΜΕ ΚΑΝΟΝΙΚΗ ΕΙΣΟΔΟ 49
5.4 Η ΛΕΙΤΟΥΡΓΙΑ ENQ ΜΕ ΤΥΧΑΙΑ ΕΙΣΟΔΟ 50
5.5 ΤΟ ΜΟΝΤΕΛΟ SAR 51
5.5.1 Η ΡΟΥΤΙΝΑ ΕΠΕΞΕΡΓΑΣΙΑΣ ΤΩΝ ΠΑΚΕΤΩΝ 53
5.5.2 Η ΕΠΑΝΑΛΗΠΤΙΚΗ ΔΙΑΔΙΚΑΣΙΑ 55
5.5.3 ΟΙ ΠΑΡΑΜΕΤΡΟΙ 55
5.5.4 ΜΕΤΡΗΣΕΙΣ 56
5.5.5 ΑΝΑΛΥΣΗ 59

6 ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΚΑΤΕΥΘΥΝΣΕΙΣ 63

Α ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ ΚΑΙ ΔΙΑΓΡΑΜΜΑΤΑ ΧΡΟΝΙΣΜΟΥ ΕΝΤΟΛΩΝ 67
Α.1 Η ΕΝΤΟΛΗ ENQUEUE 70
Α.2 Η ΕΝΤΟΛΗ DEQUEUE 74
Α.3 Η ΕΝΤΟΛΗ GETFREE 79
Α.4 Η ΕΝΤΟΛΗ RETFREE 80
Α.5 Η ΕΝΤΟΛΗ INIT 86
Α.6 Η ΕΝΤΟΛΗ READ 87
Α.7 Η ΕΝΤΟΛΗ WRITE 87
Α.8 Η ΕΝΤΟΛΗ TOP 91
Α.9 ΤΟ ΜΟΝΟΠΑΤΙ ΔΕΔΟΜΕΝΩΝ ΤΟΥ ΔΙΑΧΕΙΡΙΣΤΗ ΟΥΡΩΝ ΤΟΥ MuQPro I . 91

Β Η ‘‘ΠΛΗΡΩΣ ΕΠΕΚΤΑΣΙΜΗ’’ ΥΛΟΠΟΙΗΣΗ **93**

Β.1 ΤΟ ΜΟΝΟΠΑΤΙ ΔΕΔΟΜΕΝΩΝ 93

Β.2 Η ΜΗΧΑΝΗ ΠΕΠΕΡΑΣΜΕΝΩΝ ΚΑΤΑΣΤΑΣΕΩΝ 94

C Η ‘‘ΠΛΗΡΩΣ ΠΑΡΑΛΛΗΛΗ’’ ΥΛΟΠΟΙΗΣΗ **97**

C.1 ΤΟ ΜΟΝΟΠΑΤΙ ΔΕΔΟΜΕΝΩΝ 97

C.2 Η ΜΗΧΑΝΗ ΠΕΠΕΡΑΣΜΕΝΩΝ ΚΑΤΑΣΤΑΣΕΩΝ 98

Βιβλιογραφία **99**

Κατάλογος Πινάκων

3.1 Ποσοστό μνήμης που χρησιμοποιείται για τις δομές δεδομένων 22

4.1 Επεξήγηση της εντολής NOOP 30
4.2 Οι κώδικες εντολών του MuQPro I 31

5.1 Αριθμός εντολών γλώσσας μηχανής για την εντολή EnQ 49
5.2 Χρόνος ‘‘αδειάσματος’’ των (μεταβλητού αριθμού) ουρών κυττάρων 56
5.3 Διαμοιρασμός χρόνου μεταξύ των λειτουργιών για πακέτα 1 κυττάρου . . 59
5.4 Διαμοιρασμός χρόνου μεταξύ των λειτουργιών για πακέτα 100 κυττάρων . 60
5.5 Χρόνος εκτέλεσης της Διαχείρισης Ουρών ανά κύτταρο 60

6.1 Στατιστικά στοιχεία των υλοποιήσεων υλικού 64
6.2 Σύγκριση της παροχής των διαφόρων υλοποιήσεων 65

A.1 Επεξήγηση των σημάτων του Διαχειριστή Ουρών του MuQPro I 68
A.2 Αριθμός κύκλων ρολογιού για όλες τις εντολές του MuQPro I 68

Κατάλογος Σχημάτων

2.1 Κατά OSI στοίβα πρωτοκόλλων και επεξεργασία AAL-5 και ATM 5
2.2 Μοντέλο συστήματος ATM 7
2.3 Χρησιμοποιώντας ουρές για να υλοποιήσουμε έλεγχο ροής με εισιτήρια . . 14
2.4 Χρησιμοποιώντας ουρές για να υλοποιήσουμε επιλεκτική απόρριψη κυττάρων 15

3.1 Γενική Αρχιτεκτονική του Διαχειριστή Ουρών 18
3.2 Βασικές Δομές Δεδομένων του Διαχειριστή Ουρών 21

4.1 Αρχιτεκτονική του συστήματος MuQPro I 29
4.2 Γενικό διάγραμμα του Διαχειριστή Ουρών 30
4.3 Ο Διαχειριστής Ουρών σαν υπολογιστής διευθύνσεων 33
4.4 Αναλυτικό διάγραμμα του Διαχειριστή Ουρών 34
4.5 Επικάλυψη εντολών που ακολουθούν την Enqueue και την Dequeue 35
4.6 Επικάλυψη εντολών που ακολουθούν την Getfree και την Retfree 37
4.7 Συνενώνοντας 2 σύνολα ουρών 39
4.8 Η Μηχανή Πεπερασμένων Καταστάσεων του κυκλώματος διαιτησίας . . . 41

5.1 Σειριακές λειτουργίες Enqueue 49
5.2 Μικτές λειτουργίες Enqueue και Dequeue 50
5.3 Τυχαίες λειτουργίες Enqueue με βελτιστοποίηση 4ου επιπέδου 51
5.4 Το μοντέλο SAR 53
5.5 Μετρήσεις για πακέτα 1-1200 κυττάρων και μετρήσεις για πακέτα 1 κυττάρου σε SUN Ultra 58
5.6 Μετρήσεις του SAR για μεταβλητό αριθμό ουρών 58

A.1 Επικάλυψη εντολών που ακολουθούν την Enqueue και την Dequeue 69
A.2 Επικάλυψη εντολών που ακολουθούν την Getfree και την Retfree 69
A.3 Κώδικας C της εντολής EnQ 70
A.4 Προγραμματισμός μικροεντολών για την εντολή EnQ 72
A.5 Αρχική κατάσταση των ουρών, EnQ στην ουρά 0 το slot 0, EnQ στην ουρά 0 το slot 1 72
A.6 Εντολή EnQ σε μη άδεια ουρά 73
A.7 Εντολή EnQ σε άδεια ουρά 73
A.8 Κώδικας C της εντολής DeQ 74
A.9 Αρχική κατάσταση των ουρών, DeQ από την ουρά 0, DeQ από την ουρά 0 . 75
A.10 Προγραμματισμός μικροεντολών για την εντολή DeQ 76
A.11 Εντολή DeQ σε άδεια ουρά 77
A.12 Εντολή DeQ σε ουρά με ένα μόνο στοιχείο 78
A.13 Εντολή DeQ σε ουρά με περισσότερα από ένα στοιχεία 78
A.14 Κώδικας C της εντολής GetFree 79
A.15 Αρχική κατάσταση των ουρών, GetFree, GetFree 80
A.16 Προγραμματισμός μικροεντολών για την εντολή GetFree 81
A.17 Εντολή GetFree σε άδεια ουρά 82
A.18 Εντολή GetFree σε ουρά με ακριβώς ένα στοιχείο 82
A.19 Εντολή GetFree σε ουρά με περισσότερα από ένα στοιχεία 83
A.20 Κώδικας C της εντολής RetFree 84
A.21 Αρχική κατάσταση των ουρών, RetFree, RetFree 84
A.22 Προγραμματισμός μικροεντολών για την εντολή RetFree 85
A.23 Εντολή RetFree σε μη άδεια ουρά 85
A.24 Εντολή RetFree σε άδεια ουρά 86
A.25 Παράδειγμα αρχικοποίησης για 4 ουρές και 8 θέσεις μνήμης (slots) 87
A.26 Διάγραμμα χρονισμού I της εντολής Init 88
A.27 Διάγραμμα χρονισμού II της εντολής Init 88
A.28 Διάγραμμα χρονισμού III της εντολής Init 89
A.29 Διάγραμμα χρονισμού της εντολής Read 89
A.30 Διάγραμμα χρονισμού της εντολής Write 90
A.31 Διάγραμμα χρονισμού της εντολής Top 91
A.32 Το μονοπάτι δεδομένων του Διαχειριστή Ουρών του MuQPro I 92

B.1 Το μονοπάτι δεδομένων της ‘‘πλήρως επεκτάσιμης’’ υλοποίησης 93
B.2 Προγραμματισμός μικροεντολών της εντολής EnQ 94
B.3 Προγραμματισμός μικροεντολών της εντολής GetFree 94
B.4 Προγραμματισμός μικροεντολών της εντολής DeQ 95
B.5 Προγραμματισμός μικροεντολών της εντολής RetFree 95

C.1 Το μονοπάτι δεδομένων της ‘‘πλήρως παράλληλης’’ υλοποίησης 97
C.2 Η Μηχανή Πεπερασμένων Καταστάσεων της ‘‘πλήρως παράλληλης’’ υλοποίησης 98

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Το ATM είναι μια τεχνολογία βασισμένη σε συνδέσεις που μεταφέρει κύτταρα σταθερού μεγέθους και υποστηρίζει με ενιαίο τρόπο μεταφορά δεδομένων, φωνής και εικόνας ορίζοντας πολλές εικονικές συνδέσεις μέσα από το ίδιο φυσικό μέσο. Συγχρόνως το ATM είναι επεκτάσιμο σε πολύ υψηλές ταχύτητες, όπως αυτές ορίζονται από την ιεραρχία του SONET. Αυτή η επεκτασιμότητα θέτει αυστηρούς χρονικούς περιορισμούς στην επεξεργασία των πρωτοκόλλων και την αποθήκευση των κυττάρων στην προσωρινή μνήμη.

Το πρόβλημα γίνεται εντονότερο καθώς ο αριθμός των συνδέσμων εισόδου/εξόδου τείνει να αυξάνεται λόγω της ανάγκης των χρηστών για μεγαλύτερο αριθμό συνδέσεων. Επιπλέον, η συγκριτικά νέα αυτή τεχνολογία προσφέρει ποιότητα υπηρεσιών (QoS) ανά σύνδεση η οποία πρακτικά συνεπάγεται προκαθορισμένη καθυστέρηση και ρυθμό απώλειας των κυττάρων. Οι παράμετροι αυτές καθορίζονται μετά από διαπραγμάτευση μεταξύ του διαχειριστή του δικτύου και του χρήστη και πρέπει να κυμαίνονται σε κάποια αποδεκτά όρια. Οι παραπάνω περιορισμοί αναγκάζουν τα συστήματα ATM να έχουν αποθηκευτικό χώρο ώστε να ομαλοποιούν την κυκλοφορία στο δίκτυο και να απορρίπτουν όσο δυνατόν μικρότερο αριθμό κυττάρων.

Η προσωρινή αποθήκευση στο ATM έχει μια ειδική δομή όπως αυτή καθορίζεται από τις λειτουργίες του: τα κύτταρα συσχετίζονται μεταξύ τους με διαφόρους τρόπους και σχηματίζουν λογικές ουρές. Ο συσχετισμός αυτός προκύπτει είτε από δομικές σχέσεις (κύτταρα που ανήκουν στο ίδιο πακέτο) είτε από παρόμοιες λειτουργικές απαιτήσεις (κύτταρα που κατευθύνονται προς τον ίδιο προορισμό) και χρειάζεται μεγάλο αριθμό λογικών ουρών.

Όλα τα συστήματα ATM μπορούν να μοντελοποιηθούν με ένα απλό τρόπο χωρίς να

παραληφθεί σημαντική πληροφορία. Ένα τέτοιο μοντέλο μπορεί να χρησιμοποιηθεί για την μοντελοποίηση καρτών (adapters) και μεταγωγέων (switches) και δείχνει τις ομοιότητες μεταξύ των επιφανειακά ανόμοιων αυτών στοιχείων ενός δικτύου. Σε αυτό το μοντέλο η ανταλλαγή σημάτων (signalling) και η επεξεργασία πρωτοκόλλων υψηλότερου επιπέδου εκτελείται από ένα στοιχείο επεξεργασίας (Processing Element) ενώ τα χαμηλότερου επιπέδου πρωτόκολλα εκτελούνται σε υλικό μέσω ενός μιας διεπαφής με το δίκτυο (Network I/F). Τα κύτταρα μεταφέρονται μεταξύ του στοιχείου επεξεργασίας (PE) και ενός ή περισσοτέρων Network I/Fs και αποθηκεύονται ενδιάμεσα σε λογικές ουρές που βρίσκονται στην προσωρινή μνήμη αποθήκευσης (buffers). Σε αυτό το μοντέλο είναι δυνατόν να διαχωριστεί η προσωρινή μνήμη αποθήκευσης από την επεξεργασία των πρωτοκόλλων και τα Network I/Fs και να αποτελέσει μια ξεχωριστή μονάδα. Ο διαχωρισμός αυτός δεν είναι χρήσιμος μόνο για λόγους παρουσίασης αλλά επιπλέον προσφέρει παραλληλισμό μεταξύ της αποθήκευσης και της επεξεργασίας όπως φαίνεται στο [2]. Η αποθήκευση μπορεί με την σειρά της να χωριστεί σε 2 μέρη: μια μνήμη όπου αποθηκεύονται τα κύτταρα (CBM) και μια Μονάδα Διαχείρισης Μνήμης (MMU) υπεύθυνη για την αποθήκευση και διαχείριση δεικτών προς τα κύτταρα που βρίσκονται στην μνήμη έτσι ώστε να δημιουργεί, να τροποποιεί και να καταστρέφει ουρές. Μια τέτοια προσέγγιση είναι επεκτάσιμη επειδή οι λειτουργίες στις ουρές μπορούν να εκτελεστούν χωρίς επέμβαση στα σώματα των κυττάρων. Συνολικά το μοντέλο αυτό καταφέρνει να μοντελοποιεί τα περισσότερα συστήματα ATM που κυκλοφορούν στην αγορά με σημαντική ακρίβεια.

Για να δείξουμε ότι οι ουρές είναι αρκετές, σαν δομή, για να υποστηρίξουν την λειτουργικότητα των συστημάτων ATM, μελετήσαμε ένα βασικό σύνολο από λειτουργίες που υπάρχει σε οποιοδήποτε σύστημα ATM. Αυτές είναι ο κατακερματισμός και επανασύνδεση των κυττάρων (SAR) όπου κύτταρα που ανήκουν στην ίδια σύνδεση σχηματίζουν πακέτα, ο έλεγχος ροής, όπου τα κύτταρα σχηματίζουν ουρές ανάλογα με τον αν μπορούν να μεταδοθούν ή όχι, και η επιλεκτική απόρριψη (Selective Discard) που είναι υπεύθυνη για την απελευθέρωση χώρου αποθήκευσης απορρίπτοντας ολόκληρες ουρές από κύτταρα χαμηλής προτεραιότητας στην περίπτωση που η μνήμη τείνει να γεμίσει.

Λαμβάνοντας υπόψιν όλες τις απαιτήσεις προτείνουμε μια αρχιτεκτονική ενός Διαχειριστή Ουρών (Queue Manager). Σε αυτήν την αρχιτεκτονική ο Διαχειριστής Ουρών (QM)

είναι μια σύγχρονη συσκευή σκλάβος (synchronous slave device) που δέχεται ένα ελάχιστο σύνολο από εντολές (Enqueue, Dequeue, GetFree, RetFree, Read, Write, Top) και διαχειρίζεται ουρές μέσω δεικτών. Οι βασικές δομές που υλοποιούν ουρές είναι ένας πίνακας (Head/Tail table) του οποίου κάθε στοιχείο περιλαμβάνει ένα δείκτη στην αρχή και το τέλος μιας λογικής ουράς, ένας πίνακας από bits (Empty Bits) ο οποίος αντιστοιχεί 1 bit σε κάθε ουρά και καθορίζει αν η συγκεκριμένη ουρά είναι άδεια ή γεμάτη καθώς και ένας πίνακας δεικτών όπου κάθε δείκτης αντιστοιχεί σε μια μονάδα αποθήκευσης (δηλαδή σε ένα κύτταρο) ώστε να μπορουν να συνδεθούν πολλά κύτταρα μεταξύ τους σε ουρές. Υπάρχουν πολλές κρίσιμες αποφάσεις που πρέπει να ληφθούν σχετικά με την τοποθέτηση των δομών αυτών σε εσωτερικές ή εξωτερικές μνήμες του ολοκληρωμένου κυκλώματος καθώς και αποφάσεις σχετικές με την διάσπαση των δομών αυτών σε μία ή περισσότερες μνήμες, τις οποίες και αναλύουμε.

Επίσης παρουσιάζουμε 3 διαφορετικές υλοποιήσεις* του Διαχειριστή Ουρών σε υλικό με διαφορετική τοποθέτηση των δομών. Η πρώτη υλοποίηση έγινε για τις ανάγκες του μεταγωγέα πακέτων MuQPro I ο οποίος έχει 4 εισόδους και 1 έξοδο στα 155Mb/s και χρειάζεται επίσης ουρές για τον μικροεπεξεργαστή και τον προγραμματιστή εξόδου (output scheduler). Μια δεύτερη υλοποίηση είναι η ‘‘πλήρως επεκτάσιμη’’ η οποία ελαχιστοποιεί την υπάρχουσα μνήμη αποθήκευσης δεικτών. Τέλος η τρίτη υλοποίηση - η ‘‘πλήρως παράλληλη’’ - έγινε για την επίτευξη του μέγιστου βαθμού παραλληλισμού που υπάρχει στις λειτουργίες της διαχείρισης ουρών. Όλες οι υλοποιήσεις υλικού χρησιμοποιούν τεχνολογία FPGA και εξωτερική μνήμη SRAM και υλοποιούνται με ρολόϊ 30MHz.

Τέλος παρουσιάζουμε μια υλοποίηση λογισμικού για τον Διαχειριστή Ουρών κατάλληλη για συστήματα που εμπεριέχουν μικροεπεξεργαστές (embedded systems). Ελέγχουμε την απόδοση αυτής της υλοποίησης σε ένα δημοφιλή μικροεπεξεργαστή τύπου RISC (INTEL i960) και την συγκρίνουμε με τις υλοποιήσεις υλικού ως προς την επεκτασιμότητα - σε υψηλότερους ρυθμούς αποστολής κυττάρων και μεγαλύτερους αριθμούς ουρών - και ως προς το σχετικό κόστος.

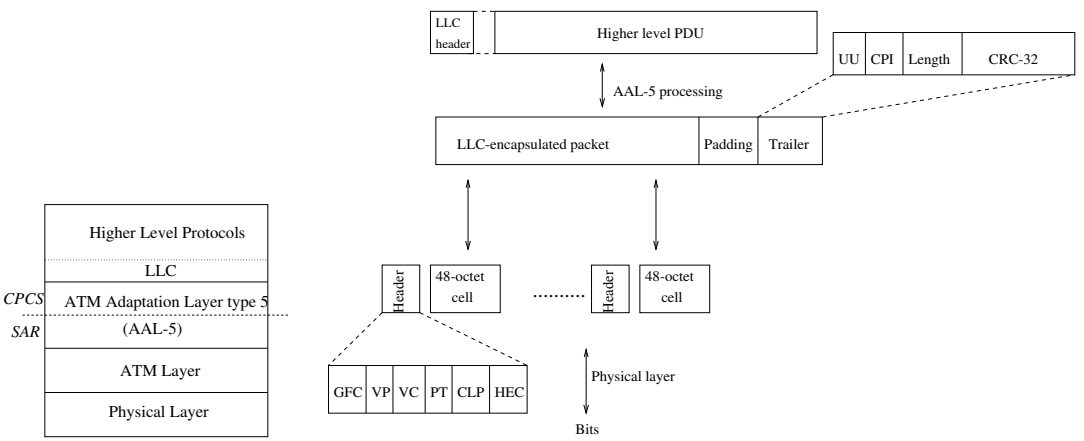
Στο Κεφάλαιο 2 παρουσιάζουμε το μοντέλο των συστημάτων ATM καθώς και τις λειτουργικές απαιτήσεις του ATM. Στο κεφάλαιο 3 καθορίζουμε γενικές αρχιτεκτονικές διαχείρισης ουρών μαζί με τις κρίσιμες παραμέτρους, ενώ τα κεφάλαια 4 και 5 περιγράφουν τις υλοποιήσεις σε υλικό και λογισμικό αντίστοιχα. Τέλος συγκρίνουμε τις διαφορετικές

*Με τον όρο υλοποίηση εννοούμε ορισμό αρχιτεκτονικής και υλοποίηση σε επίπεδο χρονικής προσομοίωσης του υλικού

αρχιτεκτονικές (στο Κεφάλαιο 6) και αναλύουμε τα πλεονεκτήματα και μειονεκτήματα κάθε μιας βοηθώντας τον χρήστη να επιλέξει την καταλληλότερη δεδομένων των αναγκών του. Στο παράρτημα Α περιγράφουμε αναλυτικά το σύνολο εντολών και τους χρονισμούς του Διαχειριστή Ουρών που σχεδιάστηκε για το MuQPro I, ενώ στα παραρτήματα Β και C υπάρχουν διαγράμματα για την ‘‘πλήρως επεκτάσιμη’’ και ‘‘πλήρως παράλληλη’’ υλοποίηση αντίστοιχα.

ΚΕΦΑΛΑΙΟ 2

ΜΟΝΤΕΛΟ ΚΑΙ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ ΣΥΣΤΗΜΑΤΩΝ ATM



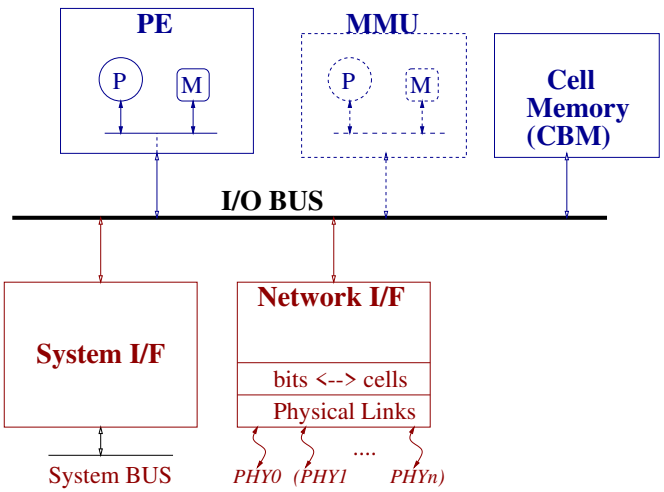
Σχήμα 2.1: Κατά OSI στοίβα πρωτοκόλλων και επεξεργασία AAL-5 και ATM

Η τεχνολογία ATM βασίζεται σε μεταφορά κυττάρων σταθερού μεγέθους που μεταφέρουν δεδομένα, φωνή ή εικόνα. Καταλαμβάνει μέρος των υποστρωμάτων data link και physical στην κατά OSI στοίβα πρωτοκόλλων όπως φαίνεται στο σχήμα 2.1(a). Η στοίβα

πρωτοκόλλων του ATM με την σειρά της χωρίζεται σε δύο υποστρώματα: Το υπόστρωμα προσαρμογής (AAL) και το υπόστρωμα ATM. Το υπόστρωμα προσαρμογής AAL είναι υπεύθυνο για την μετατροπή μεταβλητού μεγέθους PDUs σε κύτταρα σταθερού μεγέθους και το αντίστροφο. Υπάρχουν συνολικά 5 πρωτόκολλα AAL, το καθένα κατάλληλο για ένα διαφορετικό τύπο κυκλοφορίας (πραγματικού χρόνου video, μη πραγματικού χρόνου video, ήχου και δεδομένων). Σε αυτή την εργασία επικεντρώνουμε το ενδιαφέρον μας στο AAL-5 το οποίο είναι και το πιο δημοφιλές και χρησιμοποιείται όχι μόνο για δεδομένα αλλά και για συμπιεσμένο video όπως ορίζεται από τον επίσημο καθορισμό του ATM (ATM Forum standard) στο [24].

Ένα δίκτυο ATM αποτελεί μια διασύνδεση από κόμβους οι οποίοι είναι είτε μεταγωγείς (switches) είτε κάρτες δικτύου (adapters). Οι μεταγωγείς είναι συστήματα που δρομολογούν τα κύτταρα ATM μέσα στο δίκτυο, ενώ οι κάρτες δικτύου βρίσκονται στα ‘‘άκρα’’ του δικτύου, συγκεντρώνουν τα κύτταρα που κατευθύνονται προς αυτές σχηματίζοντας πακέτα και τα κατευθύνουν προς υψηλότερα στρώματα του δικτύου. Παρόλο που φαίνεται ότι οι κάρτες δικτύου και οι μεταγωγείς είναι κόμβοι δύο διαφορετικών τύπων, έχουν πολλές ομοιότητες στην δομή και στις λειτουργίες χαμηλότερου επιπέδου. Ένα τυπικό σύστημα ATM (μεταγωγέας ή κάρτα δικτύου) αποτελείται από ένα στοιχείο επεξεργασίας (PE) το οποίο εκτελεί τα πρωτόκολλα υψηλότερου επιπέδου, μνήμη για αποθήκευση των κυττάρων (και των πακέτων στην περίπτωση που έχουμε κάρτες δικτύου) καθώς και ένα αριθμό από Network I/Fs ανάλογα με την δομή του συστήματος. Το μοντέλο μιας τυπικής κάρτας δικτύου που μπορεί να χρησιμοποιηθεί σαν κάρτα δικτύου και σαν υποσύστημα μεταγωγέα παρουσιάζεται στο σχήμα 2.2.

Η αρχιτεκτονική καρτών δικτύου του σχήματος 2.2 έχει εισαχθεί στο [2] και σχετίζεται με τις απαιτήσεις δικτύων υψηλής ταχύτητας με μεταβλητό μέγεθος πακέτου. Ωστόσο, η ίδια δομή μπορεί να χρησιμοποιηθεί και για συστήματα ATM που μεταφέρουν σταθερού μεγέθους κύτταρα όπως έχει προταθεί στο [1]. Η κάρτα δικτύου διαχειρίζεται λογικές ουρές κυττάρων - και πακέτων αν αυτό είναι απαραίτητο - όπου κάθε λογική ουρά αναπαριστά μια ακολουθία από λογικά συσχετισμένα κύτταρα. Ο συσχετισμός αυτός μπορεί να είναι είτε δομικός (κύτταρα που ανήκουν στο ίδιο πακέτο) είτε λειτουργικός (κύτταρα που κατευθύνονται προς τον ίδιο προορισμό π.χ. το PE) και διαφοροποιεί την αποθήκευση σε τέτοια συστήματα από προσπέλαση δεδομένων σε συνεχόμενες διευθύνσεις σε μια απλή μνήμη τυχαίας προσπέλασης.



Σχήμα 2.2: Μοντέλο συστήματος ATM

Μια από τις πιο σημαντικές λειτουργίες συστημάτων ATM είναι η αποθήκευση κυττάρων και πακέτων. Όσον αφορά τους μεταγωγείς η αποθήκευση χρειάζεται για προσωρινή διατήρηση των κυττάρων όταν πολλά από αυτά χρειάζεται να εξέλθουν από τον ίδιο σύνδεσμο εξόδου. Σε κάρτες δικτύου η αποθήκευση χρειάζεται για επανασύνδεση των κυττάρων ώστε να κατασκευαστούν τα αρχικά πακέτα. Επιπλέον η αποθήκευση είναι απαραίτητη λόγω της χρήσης μηχανισμών που προσφέρουν έλεγχο ροής, ποιότητα υπηρεσιών (QoS) κλπ.

Η δομή και η υλοποίηση των μνημών και του ελέγχου αυτών για αποθήκευση σε συστήματα δικτύων είναι κρίσιμες στην απόδοση ολόκληρου του συστήματος όπως έχει αποδειχθεί στα [1], [16] και [18]. Το πρόβλημα γίνεται εντονότερο καθώς η ιεραρχία του SONET είναι επεκτάσιμη σε υψηλότερους ρυθμούς μετάδοσης για κάθε σύνδεσμο του Network I/F και καθώς η βιομηχανία προσπαθεί να τοποθετήσει όσο το δυνατόν περισσότερους συνδέσμους εισόδου/εξόδου στο ίδιο σύστημα. Για παράδειγμα ο χρόνος μετάδοσης/λήψης ενός κυττάρου πάνω από ένα σύνδεσμο εισόδου/εξόδου μειώνεται από 2.7 μs για SONET OC-3 (155 Mb/s) σε 0.68 μs για SONET OC-12 (622 Mb/s), 0.34 μs για SONET OC-24 (1.244 Gb/s) και φθάνει στα 0.17 μs για SONET OC-48 (2.48 Gb/s). Από την άλλη πλευρά ο συνολικός αριθμός συνδέσμων σε τυπικούς μεταγωγείς είναι 16 (π.χ. [11]) και αυξάνεται ταχύτατα καθώς το ATM μετακινείται προς δίκτυα μεγάλων αποστάσεων (WANs). Αυτό πρακτικά σημαίνει είτε ότι ο χρόνος μετάδοσης/λήψης ενός κυττάρου σε ένα τέτοιο σύστημα για

16 συνδέσμους είναι $\frac{0.17}{16} = 0.1\mu s$ -που δεν είναι εύκολα υλοποιήσιμο σε εμπορικά συστήματα σήμερα- είτε ότι υπάρχει υλικό που να εξυπηρετεί και τους 16 συνδέσμους παράλληλα. Σε κάθε περίπτωση η παροχή πρέπει να είναι 16 φορές μεγαλύτερη.

Έτσι η επεκτασιμότητα γίνεται ο πιο σημαντικός παράγοντας σε σημερινές και μελλοντικές αρχιτεκτονικές αποθήκευσης και εκεί επικεντρώνουμε το ενδιαφέρον μας στην εργασία αυτή η οποία ορίζει την εσωτερική δομή υποσυστημάτων μνήμης τα οποία μπορούν να ενταχθούν σε υψηλής απόδοσης συστήματα ATM.

2.1 ΛΕΙΤΟΥΡΓΙΑ ΚΑΙ ΕΦΑΡΜΟΓΕΣ ΤΟΥ ΜΟΝΤΕΛΟΥ

Το μοντέλο του συστήματος που παρουσιάζεται στο σχήμα 2.2 αποτυπώνει με λεπτομέρεια την λειτουργικότητα σημερινών συστημάτων ATM και θα περιγράψουμε τον τρόπο που αυτό συμβαίνει πρώτα για μεταγωγείς και αργότερα για κάρτες δικτύου. Όπως αναφέρθηκε προηγουμένως, η σύνδεση κυττάρων που συσχετίζονται με κάποιο τρόπο πραγματοποιείται μέσω λογικών ουρών (ή απλά ουρών) που υλοποιούνται από την MMU. Το κύριο χαρακτηριστικό συσχετισμού κυττάρων είναι ο αριθμός σύνδεσης (VPI/VCI) ο οποίος περιγράφει ένα μονοπάτι μεταξύ της πηγής και του προορισμού. Αυτός ο αριθμός καθορίζει τον πόρο προς τον οποίο κατευθύνονται τα κύτταρα ενώ συγκεκριμένοι αριθμοί σύνδεσης χρησιμοποιούνται αποκλειστικά για μεταφορά πληροφορίας διαχείρισης υψηλής προτεραιότητας (π.χ. αρχικοποίηση μιας σύνδεσης) από και προς το PE. Για κάθε εισερχόμενο κύτταρο (μέσω του Network I/F) μια απόφαση βασισμένη στον αριθμό σύνδεσής του πρέπει να ληφθεί: Αν είναι κύτταρο διαχείρισης (OAM cell) που κατευθύνεται προς το PE με υψηλή προτεραιότητα τότε πρέπει να αποθηκευτεί σε μια αφιερωμένη -σε αυτού του τύπου κύτταρα- ουρά. Αλλιώς πρέπει να αποθηκευτεί σε μια κατάλληλη ουρά μαζί με άλλα κύτταρα που ανήκουν στην ίδια σύνδεση περιμένοντας για κάποιο συγκεκριμένο σύνδεσμο ενός Network I/F να γίνει διαθέσιμος. Απευθείας μεταφορά μεταξύ συνδέσμων (cut-through) μπορεί επίσης να υλοποιηθεί με αυτό το σχήμα.

Τέλος κύτταρα μπορούν να τοποθετηθούν σε ουρές με διαφορετικές προτεραιότητες ελεγχόμενα από το PE και ορίζοντας ποιότητα υπηρεσιών. Για παράδειγμα κάποιες ουρές μπορεί να είναι αφιερωμένες σε υψηλής προτεραιότητας, σταθερού ρυθμού μετάδοσης κυκλοφορία (CBR traffic) με συγκεκριμένη καθυστέρηση και πιθανότητα απόρριψης κυττάρων ενώ άλλες ουρές μπορεί να είναι αφιερωμένες σε χαμηλής προτεραιότητας κυκλοφορία διαθέσιμου ρυθμού μετάδοσης (ABR traffic) η οποία δεν δίνει καμμιά εγγύηση

και απλώς χρησιμοποιεί ότι απομένει από την κυκλοφορία CBR. Επιπρόσθετα, το PE δημιουργεί κύτταρα (κοινά και OAM) τα οποία πρέπει να αναχωρήσουν μέσω συγκεκριμένων συνδέσμων των Network I/Fs, ενώ είναι και υπεύθυνο για την μετάφραση των αριθμών σύνδεσης (VPI/VCI translation) των εισερχόμενων κυττάρων όπως αυτή ορίζεται από τα πρωτόκολλα δρομολόγησης που πιθανόν να εκτελούνται στο PE αν πρόκειται για μεταγωγέα.

Σε αυτό το μοντέλο τα σώματα των κυττάρων βρίσκονται στην μνήμη CBM (μνήμη αποθήκευσης των σωμάτων των κυττάρων), ενώ η κατασκευή των ουρών εκτελείται ξεχωριστά από την MMU η οποία έχει την δική της μνήμη και λογική. Αυτό επιτυγχάνεται αφήνοντας την MMU να διαχειρίζεται δείκτες σε κύτταρα αντί για τα ίδια τα κύτταρα. Έτσι αποφεύγονται οι πολλαπλές αντιγραφές για μεταφορές μεταξύ των διαφόρων στοιχείων του συστήματος και οι ουρές μπορούν εύκολα να δημιουργηθούν και να καταστραφούν χωρίς να χρειαστεί να προσπελαστούν τα αντίστοιχα κύτταρα ή πακέτα. Όσο περισσότερα Network I/Fs είναι συνδεδεμένα σε ένα σύστημα -δηλαδή όσοι περισσότεροι σύνδεσμοι εισόδου/εξόδου υπάρχουν στο σύστημα- τόσο πιο κρίσιμη γίνεται η ταχύτητα της MMU για το υπόλοιπο σύστημα. Υπάρχουν και άλλες προσεγγίσεις στην τοποθέτηση της MMU, που μπορεί να βρίσκεται είτε σε κάποιο Network I/F, είτε στο PE άλλα σε κάθε τέτοια περίπτωση δεν επιτρέπεται παραλληλισμός μεταξύ των λειτουργιών, τουλάχιστον στο επίπεδο του μοντέλου.

Η παραπάνω διαδικασία εφαρμόζεται και στις κάρτες δικτύου. Αν θεωρήσουμε ένα Network I/F με ένα μόνο σύνδεσμο εισόδου/εξόδου τότε το μοντέλο μας μετατρέπεται σε μοντέλο για κάρτες δικτύων. Οι αποφάσεις για δρομολόγηση που θα έπρεπε να λάβουμε στην περίπτωση του μεταγωγέα αντικαθίστανται από τον κατακερματισμό και επανασύνδεση (SAR) στην περίπτωση των καρτών δικτύου. Τα κύτταρα ATM που συλλέγονταν ανάλογα με τον σύνδεσμο εξόδου στους μεταγωγείς, στις κάρτες δικτύου συλλέγονται ανάλογα με την σύνδεση. Αντίστοιχα, όσον αφορά τον έλεγχο ροής, κύτταρα τα οποία (ανά σύνδεση) έχουν εισιτήριο εξόδου διαχωρίζονται και εισέρχονται σε διαφορετικές ουρές από κύτταρα χωρίς εισιτήριο. Μπορούμε να δούμε τις ομοιότητες μεταξύ των δύο ειδών κόμβων ενός δικτύου ATM: στους μεταγωγείς, ο αριθμός σύνδεσης καθορίζει την ουρά που πραγματοποιείται η λειτουργία SAR, ενώ στις κάρτες δικτύου ο αριθμός σύνδεσης καθορίζει την ουρά που αντιστοιχεί σε κάποιο σύνδεσμο εξόδου.

Το μοντέλο που παρουσιάσαμε μέχρι τώρα είναι απλό, αλλά αναπαριστά με μεγάλη

ακρίβεια τα περισσότερα εμπορικά συστήματα ATM και έχει το πλεονέκτημα να αποκαλύπτει τυχόν ανισσοροπίες μεταξύ των πρωτοκόλλων που έχουν υλοποιηθεί σε υλικό και αυτών που έχουν υλοποιηθεί σε λογισμικό. Εμπορικά συστήματα όπως το ολοκληρωμένο για SAR της SUN ([10]) και ο μεταγωγέας ATM της FORE ([11]) μπορούν εύκολα να περιγραφούν με βάση αυτό το μοντέλο. Το ολοκληρωμένο SUN ATM622-s SAR που χρησιμοποιείται ευρύτατα σε κάρτες δικτύου υλοποιεί τα στρώματα ATM και AAL σε υλικό. Περιέχει 3 interfaces: Με το φυσικό στρώμα (Utopia), τον επεξεργαστή (SBus) και την μνήμη αποστολής και λήψης κυττάρων. Η επικοινωνία μεταξύ του επεξεργαστή και του ολοκληρωμένου μπορεί να γίνει σε επίπεδο πακέτων, δηλαδή ανταλλαγή PDUs (μονάδων ανταλλαγής πληροφορίας μεταξύ πρωτοκόλλων) επιπέδου SAR. Έτσι υψηλότερα από το AAL-5 στρώματα του δικτύου εκτελούνται από τον επεξεργαστή ενώ τα χαμηλότερου επιπέδου εκτελούνται σε υλικό από το ολοκληρωμένο. Η διαχείριση της μνήμης γίνεται εξ' ολοκλήρου σε υλικό από το ολοκληρωμένο έτσι ώστε να μπορεί να ικανοποιηθεί ο ρυθμός εισόδου και εξόδου κυττάρων των 622 Mb/s. Ο μεταγωγέας ATM ASX-200 της εταιρίας FORE, είναι από τους πιο δημοφιλείς στην αγορά. Έχει 22 συνολικά συνδέσμους εισόδου/εξόδου, 18 των 25 Mb/s και 4 των 155 Mb/s, ενώ υλοποιεί διαχείριση μνήμης σε υλικό και αποθήκευση κυττάρων ανά σύνδεση. Άλλα ολοκληρωμένα για κάρτες δικτύου όπως τα ολοκληρωμένα για SAR από την NEC που χρησιμοποιούνται σε κάρτες δικτύου και συνδέονται με επεξεργαστή μέσω του διαύλου PCI, υλοποιούν το SAR με ένα συνδυασμό υλικού και λογισμικού: ο επεξεργαστής παρέχει στο ολοκληρωμένο ένα μέρος από την μνήμη του και το ολοκληρωμένο την χρησιμοποιεί για να αποθηκεύει κύτταρα ενώ η μνήμη που βρίσκονται οι δείκτες είναι μέσα στο ολοκληρωμένο. Πιο απλά και εφαρμόζοντας το μοντέλο μας, η μνήμη που αποθηκεύονται τα κύτταρα βρίσκεται εκτός του ολοκληρωμένου και είναι μέρος της μνήμης του εξωτερικού επεξεργαστή, ενώ η μνήμη που διατηρεί τις λογικές ουρές βρίσκεται μέσα στο ολοκληρωμένο.

2.2 ΛΕΙΤΟΥΡΓΙΕΣ ΕΝΟΣ ΣΥΣΤΗΜΑΤΟΣ ATM

Μέσω του απλοποιημένου μοντέλου που περιγράφηκε προηγουμένως παρουσιάσαμε την ανάγκη για αρχιτεκτονικές διαχείρισης μνήμης που γενικά διαχειρίζονται ουρές. Ωστόσο, χρειάζεται μια πιο προσεκτική μελέτη των λειτουργιών που απαιτούνται στα συστήματα ATM ώστε να διασαφηνιστεί ο τρόπος με τον οποίο οι ουρές τις υποστηρίζουν. Οι βασικές λειτουργίες που εκτελούνται από το ATM είναι ο *κατακερματισμός και επανασύνδεση*

(SAR) όπου κύτταρα που ανήκουν στην ίδια σύνδεση εισέρχονται στην ίδια ουρά για να σχηματίσουν πακέτα, ο *έλεγχος ροής* (με εισιτήρια ή βασισμένος στον ρυθμό) που προφυλλάσει το δίκτυο από το πλημμύρισμα λόγω κάποιων συνδέσεων που εισάγουν μεγάλη κυκλοφορία στο δίκτυο αδικώντας τις υπόλοιπες και η *επιλεκτική απόρριψη* κυττάρων όταν η μνήμη έχει ξεπεράσει κάποια όρια χρησιμοποίησης. Όλες αυτές οι λειτουργίες υλοποιούνται στο AAL-5 και αναλύονται στα επόμενα υποκεφάλαια.

2.2.1 ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ ΚΑΙ ΕΠΑΝΑΣΥΝΔΕΣΗ (SAR)

Το στρώμα AAL-5 χωρίζεται σε δύο μέρη: Το Convergence Sublayer (CS) και το Segmentation and Reassembly Sublayer (SAR). Το πρώτο μέρος εκτελεί αναγνώριση μηνυμάτων (message identification) και επανάκτηση του ρολογιού (time/clock recovery) και δεν αναλύεται σε αυτήν την εργασία ενώ το δεύτερο μέρος είναι υπεύθυνο για τον κατακερματισμό των LLC PDUs (PDUs επιπέδου LLC, όπως φαίνεται στο σχήμα 2.1(a)) σε κύτταρα μεγέθους 48 bytes κατά την μετάδοση και την ανακατασκευή των εισερχόμενων κυττάρων σε μια LLC PDU κατά την λήψη. Πιο λεπτομερώς το AAL-5 εκτελεί τα ακόλουθα κατά την μετάδοση (και τα αντίστροφα κατά την λήψη) όπως φαίνεται στο σχήμα 2.1(b):

1. Προσθέτει (ή αφαιρεί) όσα επιπλέον bytes χρειάζεται (padding) ώστε η εισερχόμενη LLC PDU να έχει συνολικά μέγεθος πολλαπλάσιο των 48 bytes.

2. Προσθέτει (ή αφαιρεί) μια επικεφαλίδα των 8 bytes (8-byte trailer) στην LLC PDU.

3. Κατακερματίζει (ή επανασυνδέει) το προκύπτον πακέτο σε κύτταρα των 48 bytes και τα προωθεί στο στρώμα ATM.

4. Προσθέτει (ή αφαιρεί) σε κάθε κύτταρο των 48 bytes μια επικεφαλίδα (header) των 5 bytes έτσι ώστε να σχηματιστεί ένα κύτταρο των 53 bytes.

Η διαδικασία αυτή είναι γνωστή σαν SAR και χρειάζεται μνήμη για να υλοποιηθεί ανά σύνδεση. Για να έχουμε τέτοιου τύπου λογική αποθήκευση (ανά σύνδεση) και δεδομένου ότι το ATM είναι πρωτόκολλο βασισμένο σε συνδέσεις, απαιτείται μια διαφορετική ουρά ανά ενεργή σύνδεση. Το γεγονός ότι χρειαζόμαστε ουρές μόνο για τις ενεργές συνδέσεις μειώνει τον αριθμό των ουρών που πρέπει να διαθέσουμε αλλά ο αριθμός αυτός παραμένει μια κρίσιμη παράμετρος. Μια έμμεση απαίτηση του SAR είναι η υψηλή χρησιμοποίηση της μνήμης των λογικών ουρών. Αυτό μπορεί να επιτευχθεί υλοποιώντας όλες

τις ουρές σε ένα κοινό διαμοιραζόμενο χώρο. Η υλοποίηση σε κοινόχρηστο χώρο έχει ένα επιπλέον πλεονέκτημα: Κάθε ουρά μπορεί να καταλαμβάνει ένα ποσοστό μνήμης που ορίζεται από τον χρήστη. Η λειτουργία SAR επανακατασκευάζει πακέτα AAL-5 και τα δίνει στα υψηλότερα στρώματα. Ωστόσο, ο διαχειριστής μνήμης μπορεί να προσφέρει μηχανισμούς που να προσφέρουν τα πακέτα σε FIFO σειρά χρησιμοποιώντας ουρές από ουρές όπως θα δούμε πιο αναλυτικά στο Κεφάλαιο 3.

2.2.2 ΕΛΕΓΧΟΣ ΡΟΗΣ ΒΑΣΙΣΜΕΝΟΣ ΣΤΟΝ ΡΥΘΜΟ ΜΕΤΑΔΟΣΗΣ (Rate-based flow control)

Ο έλεγχος ροής που βασίζεται στο ρυθμό (Rate-based flow control) είναι ο standard μηχανισμός ελέγχου της συμφόρησης που χρησιμοποιείται από το ATM Forum. Όταν η μνήμη που αντιστοιχεί σε μια σύνδεση ξεπερνά ένα συγκεκριμένο όριο, ένα ειδικό κύτταρο διαχείρισης πόρων (RM cell) δημιουργείται και στέλνεται στην πηγή για να την ειδοποιήσει να μειώσει τον ρυθμό αποστολής κυττάρων (σύμφωνα με μια συνάρτηση) για αυτή την σύνδεση. Ένας άλλος τρόπος για να υλοποιήσει κανείς αυτό τον μηχανισμό είναι να μειώνει η πηγή τον ρυθμό μετάδοσης έως ότου λάβει κύτταρα RM οπότε και αυξάνει τον ρυθμό. Συνολικά ο μηχανισμός για Rate-based flow control δουλεύει καλά υπό προϋποθέσεις αλλά όχι κάτω από όλες τις συνθήκες. Επίσης ο μηχανισμός αυτός δεν θέτει επιπλέον περιορισμούς σε σχέση με τον έλεγχο ροής με εισιτήρια και για αυτό τον λόγο θα αναφερόμαστε μόνο στον έλεγχο ροής με εισιτήρια από εδώ και πέρα. Περισσότερες πληροφορίες για έλεγχο ροής που βασίζεται στον ρυθμό μπορεί να βρεθούν στα [5] και [6].

2.2.3 ΕΛΕΓΧΟΣ ΡΟΗΣ ΒΑΣΙΣΜΕΝΟΣ ΣΕ ΕΙΣΙΤΗΡΙΑ (Credit-based flow control)

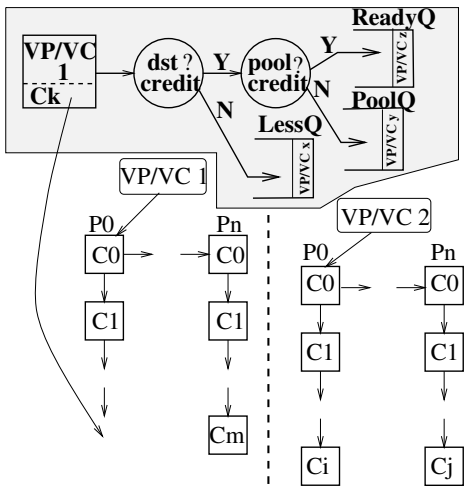
Ο έλεγχος ροής με εισιτήρια είναι ένας μηχανισμός που εγγυάται δικαιοσύνη μεταξύ διαφορετικών κυκλοφοριακών ροών και γενικά εφαρμόζει έλεγχο συμφόρησης μεταξύ γειτονικών κόμβων. Ο κόμβος που στέλνει (πομπός) ξεκινάει να στέλνει μόνο όταν ξέρει ότι υπάρχει ελεύθερος χώρος αποθήκευσης στον δέκτη. Για να ξέρει ο πομπός αν υπάρχει ελεύθερος χώρος στον δέκτη πρέπει να υπάρχει ένας μετρητής εισιτηρίων και κάποιος τρόπος κωδικοποίησης των εισιτηρίων. Ο δέκτης στέλνει εισιτήρια στον πομπό ενημερώνοντας τον για την ελεύθερη μνήμη του, ενώ ο πομπός όποτε λαμβάνει ένα εισιτήριο αυξάνει τον μετρητή εισιτηρίων. Κάθε φορά που ο μετρητής είναι μη μηδενικός ο πομπός στέλνει ένα κύτταρο και μειώνει τον μετρητή. Αυτό το απλό σχήμα είναι ο έλεγχος ροής με

εισιτήρια μιας λωρίδας (single-lane credit-based flow control). Το κύριο πρόβλημα με αυτό το σχήμα είναι στην περίπτωση που μια σύνδεση είναι πολύ συμφορημένη και καταναλώνει όλη την διαθέσιμη μνήμη. Έτσι άλλες συνδέσεις που εισάγουν μικρή κυκλοφορία στο δίκτυο δεν μπορούν να χρησιμοποιηθούν ακόμη κι αν το συγκεκριμένο κομμάτι του δικτύου στο οποίο θέλουν να φθάσουν παραμένει αχρησιμοποίητο. Η κατάσταση αυτή οδηγεί το δίκτυο σε αχρηστία (starvation). Αυτή η κατάσταση είναι παρόμοια με το head-of-line blocking.

Πιο περίπλοκα σχήματα έχουν προταθεί για να αντιμετωπιστεί αυτό το πρόβλημα όπως το [8]. Ένα από τα πιο ενδιαφέροντα είναι είναι ο έλεγχος ροής με εισιτήρια πολλών λωρίδων όπως περιγράφεται στο [7], όπου η μνήμη χωρίζεται σε μικρά κομμάτια ένα για κάθε σύνδεση. Έτσι κάθε σύνδεση έχει ένα ελάχιστο κομμάτι μνήμης ανεξαρτήτως των υπολοίπων και δεν εμφανίζεται το παραπάνω φαινόμενο. Μια βελτιστοποίηση ως προς το μέγεθος της χρησιμοποιούμενης μνήμης μπορεί να γίνει αν η μνήμη διασπαστεί σε λιγότερα κομμάτια όσα οι διαφορετικές ροές προς διαφορετικούς προορισμούς. Μια πιο περιληπτική περιγραφή του έλεγχου ροής μπορεί κανείς να βρει στα [7] και [8]. Ο έλεγχος ροής εφαρμόζεται κυρίως σε μεταγωγείς αλλά και σε κάρτες δικτύου για λόγους τερματισμού. Είναι ένας αποδοτικός μηχανισμός με μεγάλη πολυπλοκότητα υλοποίησης. Σε κάθε περίπτωση η διαχείριση της μνήμης παίζει ένα πολύ σημαντικό ρόλο στο να ξεχωρίζει κανείς μεταξύ χώρων αποθήκευσης που ανήκουν σε διαφορετικές συνδέσεις. Παρέχει την βασική λειτουργικότητα για να αποδοθεί σε κάθε σύνδεση ένα δίκαιο ποσοστό μνήμης και αντίστοιχα ένας δίκαιος ρυθμός μετάδοσης όπως αυτός έχει προκαθοριστεί για την συγκεκριμένη σύνδεση.

Για παράδειγμα ας σκεφτούμε ένα σχήμα με εισιτήρια 2 επιπέδων όπου ο πρώτος τύπος εισιτηρίου δίνεται όταν υπάρχει ελεύθερη μνήμη για αυτή την σύνδεση στον κόμβο προορισμού (destination credit) και ο δεύτερος τύπος εισιτηρίου (pool credit) δίνεται όταν υπάρχει ελεύθερος χώρος στον ενδιάμεσο γειτονικό κόμβο. Τα κύτταρα που ανήκουν στην ίδια σύνδεση σε αυτό το σχήμα ανήκουν επίσης σε μια από τις παρακάτω 4 κατηγορίες:

1. Κύτταρα χωρίς καθόλου εισιτήρια
2. Κύτταρα με εισιτήριο προορισμού μόνο (destination credit)
3. Κύτταρα που έχουν εισιτήριο για το γειτονικό κόμβο μόνο (pool credit)
4. Κύτταρα που έχουν και των δύο ειδών εισιτήρια



Σχήμα 2.3: Χρησιμοποιώντας ουρές για να υλοποιήσουμε έλεγχο ροής με εισιτήρια

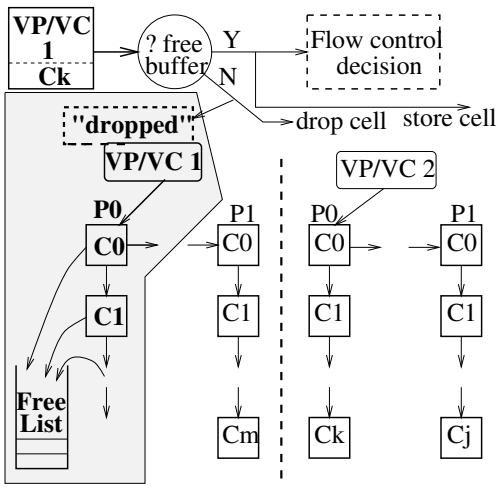
Αυτή η ανάλυση οδηγεί σε χρήση 4 ουρών ανά σύνδεση. Ένας πιο έξυπνος τρόπος για να επιτύχουμε την ίδια λειτουργικότητα με 3 μόνο ουρές ανά σύνδεση (σχήμα 2.3) είναι να χρησιμοποιήσουμε ιεραρχικά τα εισιτήρια. Αρχικά κάθε εισερχόμενο κύτταρο προσπαθεί να πάρει και τα 2 ειδών εισιτήρια και αν τα καταφέρει τότε εισέρχεται στην ουρά των έτοιμων προς αναχώρηση (Ready Queue) ενώ αν καταφέρει να λάβει μόνο εισιτήριο προορισμού τότε εισέρχεται στην Pool Queue. Σε κάθε άλλη περίπτωση το κύτταρο θα εισέλθει στην Less Queue όπου βρίσκονται κύτταρα χωρίς εισιτήρια. Τα εισιτήρια pool δίνονται μόνο μαζί με εισιτήρια προορισμού αλλιώς δεν δίνονται καθόλου. Στο σχήμα αυτό τα εισιτήρια προορισμού είναι ‘‘ακριβότερα’’ από τα εισιτήρια Pool.

2.2.4 ΕΠΙΛΕΚΤΙΚΗ ΑΠΟΡΡΙΨΗ ΚΥΤΤΑΡΩΝ (Selective Discard)

Η Επιλεκτική Απόρριψη κυττάρων (όπως έχει εισαχθεί στο [1]) είναι ένας μηχανισμός που δεν υπάρχει υποχρεωτικά σε όλα τα συστήματα ATM αλλά βοηθά σημαντικά στην αποσυμφόρηση των δικτύων σε περιόδους έντονης κυκλοφορίας (bursty traffic). Όσο μεγάλο χώρο μνήμης κι αν έχουμε, θα υπάρξουν φορές όπου η κυκλοφορία στο δίκτυο θα γεμίσει αυτή την μνήμη, αναγκάζοντας το σύστημα να απορρίψει τα εισερχόμενα κύτταρα ακόμη κι αν είναι υψηλής προτεραιότητας. Στην επιλεκτική απόρριψη απορρίπτουμε μια ολόκληρη λογική ουρά πού ανήκει σε μια σύνδεση χαμηλής προτεραιότητας, ώστε να

απελευθερωθεί χώρος για τα νεοεισερχόμενα κύτταρα. Στο [1] η λογική ουρά που απορρίπτεται ανήκει σε ένα μη ολοκληρωμένο πακέτο, δηλαδή σε μια μη ολοκληρωμένη AAL-5 PDU.

Σε άλλη περίπτωση, είναι δυνατό να απορρίπτουμε κυκλοφορία ABR δίνοντας προτεραιότητα σε κυκλοφορία CBR. Σε όλες αυτές τις περιπτώσεις, ο διαχειριστής μνήμης χρειάζεται να παρέχει μηχανισμούς απόρριψης μιας ολόκληρης λογικής ουράς. Ο μηχανισμός αυτός φαίνεται και στο σχήμα 2.4 όπου ολόκληρη η λογική ουρά του VP/VC 1 απορρίπτεται (‘‘αδειάζει’’).



Σχήμα 2.4: Χρησιμοποιώντας ουρές για να υλοποιήσουμε επιλεκτική απόρριψη κυττάρων

Μια άλλη προσέγγιση είναι το EPD (Early Packet Discard), όπου, ολόκληρες AAL-5 PDUs απορρίπτονται κατευθείαν καθώς εισέρχονται στο σύστημα: όταν αποφασιστεί ότι μια σύνδεση έχει φθάσει κάποιο προκαθορισμένο όριο ή έχει χαμηλή προτεραιότητα, τότε όλα τα εισερχόμενα κύτταρα αυτής της σύνδεσης απορρίπτονται μέχρι να πεταχτεί μια ολόκληρη AAL-5 PDU. Το EPD δεν χρειάζεται καμμία ειδική υποστήριξη από την μονάδα διαχείρισης μνήμης. Περισσότερες πληροφορίες πάνω στο EPD μπορούν να βρεθούν στα [20] και [22].

ΚΕΦΑΛΑΙΟ 3

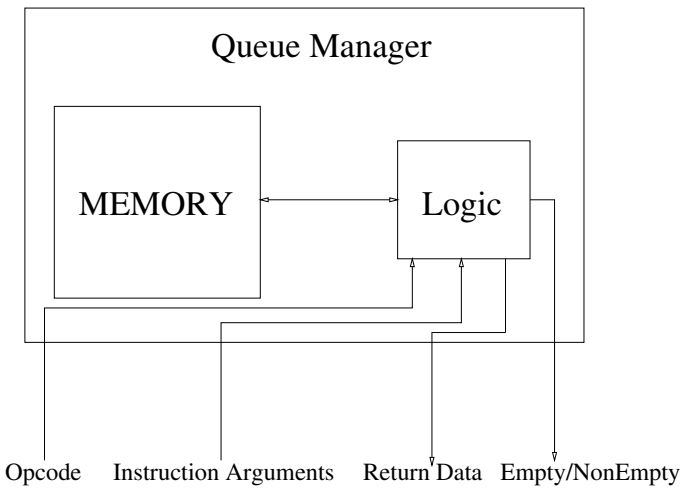
ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΔΙΑΧΕΙΡΙΣΗΣ ΟΥΡΩΝ

Μέχρι τώρα θεωρούσαμε το υποσύστημα μνήμης σαν ένα ‘‘μαύρο κουτί’’ που χειρίζεται διευθύνσεις στη μνήμη κατασκευάζοντας ουρές κυττάρων που αποθηκεύονται στη μνήμη CBM. Από εδώ και στο εξής θα αναφερόμαστε σε αυτό το σύστημα σαν Διαχειριστή Πολλαπλών Ουρών ή απλώς Διαχειριστή Ουρών (QM). Ο Διαχειριστής Ουρών, του οποίου η λειτουργικότητα έχει οριστεί στο κεφάλαιο 2, μπορεί να θεωρηθεί ως ένας Ειδικός Επεξεργαστής που εκτελεί εντολές σχετικές με ουρές. Αυτή η προσέγγιση κυρίου-σκλάβου (QM είναι ο σκλάβος), όπως παρουσιάζεται στο σχήμα 3.1, έχει θεωρηθεί ως η πιο κατάλληλη επειδή απλοποιεί την αλληλεπίδραση μεταξύ της εξωτερικής κύριας συσκευής (που ελέγχει τον QM) και του QM και προβάλλει ένα interface γενικού σκοπού που μπορεί να ενσωματωθεί εύκολα σε οποιοδήποτε σύστημα ATM.

Ένα βασικό σύνολο εντολών που καλύπτει όλη την απαιτούμενη λειτουργικότητα είναι το παρακάτω:

Init(arg1,arg2,arg3,arg4): Θέτει το χώρο διευθύνσεων που κάθε μια από τις απαιτούμενες δομές δεδομένων (Head /Tail table,Pointer Memory) καταλαμβάνει. Η δομή Head/Tail table ξεκινάει στη διεύθυνση arg1 και χρησιμοποιεί 2∗arg2 λέξεις, και η δομή Pointer Memory ξεκινάει στη διεύθυνση arg3 και χρησιμοποιεί arg4 λέξεις. Επιπλέον η διαδικασία αρχικοποίησης ‘‘γεμίζει’’ τη Free List και μαρκάρει όλες τις ουρές σαν άδειες.

EnQ(arg1,arg2): Τοποθετεί στην ουρά arg1 τη διεύθυνση που ορίζεται από την arg2 και απαντά εάν η ουρά ήταν άδεια πριν δοθεί αυτή η εντολή ή όχι.



Σχήμα 3.1: Γενική Αρχιτεκτονική του Διαχειριστή Ουρών

DeQ(arg1): Εξάγει ένα στοιχείο από την ουρά arg1 και απαντά με την διεύθυνση του εξαγόμενου στοιχείου. Απαντά επίσης αν η ουρά arg1 έχει αδειάσει μετά την ολοκλήρωση αυτής της εντολής.

Top(arg1): Επιστρέφει το κορυφαίο στοιχείο της ουράς arg1. Η ουρά arg1 δεν αλλάζει. Επιπλέον απαντά αν η ουρά είναι άδεια ή όχι. Αυτή η εντολή παίζει το διπλό ρόλο της εντολής ‘‘Top’’ και της εντολής ‘‘IsEmptyQ’’.

GetFree(): Εξάγει μια διεύθυνση από την Λίστα Ελευθέρων Διευθύνσεων (Free List). Επιστρέφει επίσης αν η Free List έχει αδειάσει μετά την εκτέλεση αυτής της εντολής ή όχι.

RetFree(arg1): Επιστρέφει την διεύθυνση arg1 στη Free List. Με άλλα λόγια η διεύθυνση arg1 δεν χρησιμοποιείται πια, και με αυτή την εντολή απελευθερώνεται για χρήση. Επίσης επιστρέφεται μια απάντηση για το αν η Free List ήταν άδεια πριν την εκτέλεση αυτής της εντολής ή όχι.

Read(arg1): Η εντολή αυτή απλά διαβάζει μια λέξη από την διεύθυνση arg1 της εσωτερικής μνήμης του QM. Χρησιμοποιείται μόνο για την εύρεση πιθανών λαθών.

Write(arg1,arg2): Η εντολή αυτή γράφει την λέξη arg2 στην διεύθυνση arg1. Χρησιμοποιείται μόνο για την εύρεση πιθανών λαθών.

Το σύνολο εντολών ορίζει όχι μόνον τον λειτουργικό πυρήνα αλλά και εντολές που βοηθούν στη διαδικασία διόρθωσης λαθών (debugging), που είναι μια από τις πιο επίπονες εργασίες στη διαδικασία σχεδίασης ενός ψηφιακού συστήματος. Αυτές είναι οι εντολές Read/Write που μπορούν να διαβάσουν και να γράψουν αυθαίρετες διευθύνσεις στη μνήμη του QM καθώς και η εντολή Top που επιστρέφει το κορυφαίο στοιχείο κάθε ουράς χωρίς να το εξάγει από την ουρά. Όλες οι εντολές μπορούν να κωδικοποιηθούν σε 3 bits καταλήγοντας σε μια μικρή και αποδοτική κωδικοποίηση του κώδικα εντολής (opcode) που μπορεί να επεκταθεί εύκολα σε συστήματα υψηλής ταχύτητας.

Σε πολλές εφαρμογές απαιτείται μια ουρά για κάθε VPI/VCI, αλλά αυτό δεν είναι πρακτικά εφικτό γιατί ο αριθμός όλων των πιθανών συνδέσεων είναι υπερβολικά μεγάλος: το πεδίο VPI/VCI είναι 24 bits (όπως ορίζεται από το UNI στο [23]), καταλήγοντας σε 2^{24} πιθανές συνδέσεις. Επιπλέον, η παροχή 2^{24} ουρών ακόμα κι αν ήταν τεχνολογικά εφικτή δεν θα ήταν αποδοτική, θεωρώντας ότι ο αριθμός των ενεργών συνδέσεων σε ένα τυπικό σύστημα ATM είναι σημαντικά μικρότερος. Συμπερασματικά, πρέπει να υπάρχει κάποια υποστήριξη για αντιστοίχιση όλων των πιθανών συνδέσεων σε μια μικρότερη ομάδα παρεχόμενων ουρών. Υπάρχουν πολλές διαφορετικές προσεγγίσεις για αντιστοίχιση (ή μετάφραση) από έναν αριθμό σύνδεσης σε ένα αριθμό ουράς:

1. Χρήση μιας Προσεταιριστικής Μνήμης (CAM) για την πλήρη μετάφραση των VPI/VCI σε αριθμούς ουρών. Η λύση αυτή απαιτεί πλήρη αντιστοίχιση του VPI/VCI σε ένα αριθμό ουρών καθοριζόμενο από το μέγεθος της CAM.

2. Χρήση μιας Δυναμικής Μνήμης (DRAM) διευθυνσιοδοτούμενης από τα bits του VPI/VCI. Η λύση αυτή παρέχει επίσης πλήρη αντιστοίχιση του VPI/VCI σε έναν αριθμό ουρών καθοριζόμενο από το μέγεθος της DRAM.

3. Απευθείας χρήση κάποιων bits του VPI/VCI σαν αριθμό ουράς. Η τεχνική αυτή καταλήγει σε πολλές συνδέσεις αντιστοιχούμενες στον ίδιο αριθμό ουράς. Το πρόβλημα μπορεί να ελαττωθεί (όχι όμως να εξαληφθεί) μέσω σωστής απόδοσης των VPI/VCIs από τον διαχειριστή του δικτύου. Ο αριθμός των ουρών είναι παράμετρος του συστήματος .

4. Καθορισμός ενός συγκεκριμένου αριθμού bits από το VPI και το VCI αφήνοντας τον χρήστη να καθορίσει ποιά από αυτά. Αυτή η λύση είναι η πιο κοινή σε εμπορικά συστήματα και επιτρέπει στο διαχειριστή του δικτύου να χρησιμοποιήσει τη δική

του πολιτική στη μετάφραση του VPI/VCI κατά την έναρξη της σύνδεσης. Ο αριθμός των ουρών είναι μια παράμετρος του συστήματος που οδηγεί στην επιλογή των κατάλληλων ομάδων bits από το VPI και το VCI.

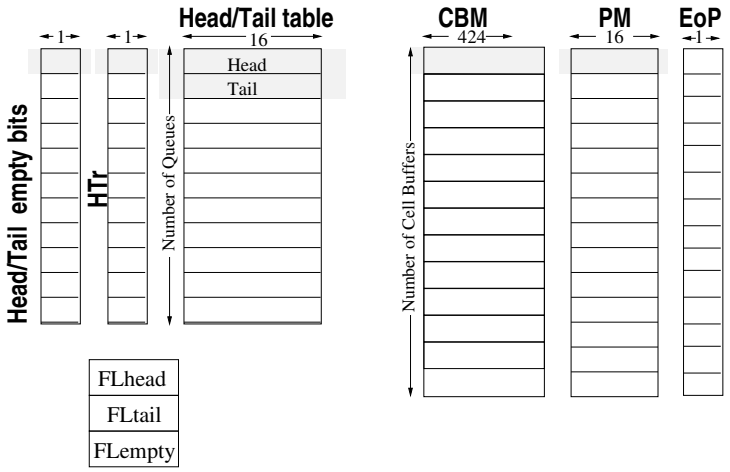
5. Αντιστοίχιση των VPI/VCIs σε αριθμούς ουρών μέσω μιας καλά ορισμένης συνάρτησης κατακερματισμού (π.χ. CRC-10). Προφανώς ο αριθμός των ουρών καθορίζεται από τις ιδιότητες της συνάρτησης κατακερματισμού έτσι ώστε να έχουμε λίγες συγκρούσεις κατά τη διάρκεια της αντιστοίχισης των VPI/VCI σε αριθμούς ουρών.

Κάθε λύση έχει τα πλεονεκτήματα και τα μειονεκτήματά της, με την πρώτη λύση να είναι η πιο ακριβή, γρήγορη και ελαστική, τη δεύτερη να είναι δίκαιη στη μετάφραση όλων των VPI/VCI bits με μια λογική καθυστέρηση, τη τρίτη να είναι η πιο απλή υποφέροντας από συγκρούσεις , την τέταρτη να χρησιμοποιείται στα περισσότερα εμπορικά συστήματα με δίκαιη αναλογία κόστους/απόδοσης και την πέμπτη να έχει χαμηλό κόστος, αλλά να εξαρτάται πολύ από τον τρόπο απόδοσης των VPI/VCI από τον διαχειριστή του δικτύου και τις ιδιότητες της συνάρτησης κατακερματισμού. Οι μηχανισμοί που περιγράφηκαν για την αντιστοίχιση του αριθμού σύνδεσης σε αριθμό λογικής ουράς δεν ενδιαφέρουν την αρχιτεκτονική μας και από εδώ και στο εξής θα θεωρούμε ότι ο αριθμός σύνδεσης με τον αριθμό λογικής ουράς είτε συμπίπτουν είτε κάποια εξωτερική λογική αναλαμβάνει να τους αντιστοιχίζει. Οι δομές δεδομένων που χρειάζονται σε αυτή την αρχιτεκτονική ακολουθούν στο Κεφάλαιο 3.1.

3.1 ΒΑΣΙΚΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Για την καλύτερη κατανόηση της αρχιτεκτονικής θα ορίσουμε μια υλοποίηση των ουρών FIFO (ουρών που τα στοιχεία εξέρχονται με την σειρά εισόδου) σε κοινόχρηστο χώρο αποθήκευσης, όπως έχει παρουσιαστεί στα [2] και [9]. Εξειδικεύσαμε την υλοποίηση αυτή σε συστήματα ATM για τα οποία και είναι βέλτιστη ως προς το μέγεθος της χρησιμοποιούμενης μνήμης για τους δείκτες καθότι δεν μπορεί να χρησιμοποιήσει κανείς λιγότερους δείκτες για να περιγράψει μια συνδεδεμένη λίστα με ιδιότητες FIFO. Ορίζουμε ως *Cell Buffer* το χώρο μνήμης που απαιτείται για την αποθήκευση ενός κυττάρου ATM, π.χ. 53 bytes. Για να έχουμε πρόσβαση σε ένα *Cell Buffer (CB)* και να συνδέουμε πολλούς από αυτούς σε ουρές, απαιτείται ένας δείκτης για κάθε έναν, που ονομάζεται *Cell Pointer (CP)*. Η μνήμη που κρατάει τους CBs είναι η *Cell Buffer Memory (CBM)*, ενώ αυτή που κρατάει όλους

τους CPs ονομάζεται *Pointer Memory (PM)*. Επιπλέον η ελάχιστη απαίτηση για υλοποίηση μιας ουράς FIFO είναι δύο δείκτες: ο ένας δείχνει στην κεφαλή (Head) και ο άλλος στο τέλος (Tail) της κάθε ουράς και ο πίνακας που διατηρεί αυτούς τους δείκτες για όλες τις ουρές ονομάζεται *Head/Tail table*. Ένα επιπλέον bit (για κάθε ουρά) που αναγνωρίζει την κατάσταση της ουράς -δηλαδή αν η ουρά είναι άδεια ή όχι- απαιτείται και το σύνολο αυτών ονομάζεται *Head/Tail Empty Bits*. Οι δομές δεδομένων που παρουσιάστηκαν μέχρι τώρα είναι ικανές να συνδέσουν και να δημιουργήσουν ουρές FIFO από CBs. Ωστόσο δεν μπορούμε να ξεχωρίσουμε ανάμεσα σε άδειους και χρησιμοποιούμενους CBs. Για αυτό το λόγο απαιτείται μια *Λίστα Ελευθέρων Διευθύνσεων (Free List)* που να καταλαμβάνει σταθερό χώρο μνήμης. Για μια τέτοια υλοποίηση χρειάζεται ένας καταχωρητής *Free List Head*, ένας *Free List Tail* και ένα flip-flop *Free List Empty Bit*. Αν συνδέσουμε όλους τους CBs διαδοχικά (ο CB 0 να δείχνει στον CB 1, ο οποίος να δείχνει στον CB 2 κτλ) και αρχικοποιήσουμε τον καταχωρητή Free List Head να δείχνει στον CB 0 και τον καταχωρητή Free List Tail να δείχνει στον CB $N_c - 1$, όπου N_c είναι ο ολικός αριθμός των CBs, τότε έχουμε μια υλοποίηση της Free List που καταλαμβάνει ελάχιστο χώρο μνήμης. Επίσης πρέπει να είμαστε προσεκτικοί στο να θέσουμε το Free List empty bit στο 0. Τέλος χρειαζόμαστε ένα EoP bit για κάθε CB, που σηματοδοτεί ότι το συγκεκριμένο κύτταρο είναι το τελευταίο μιας AAL-5 PDU και ένα HTr bit ανά ουρά που να καθορίζει αν η συγκεκριμένη ουρά χρησιμοποιείται από τον Διαχειριστή ή όχι. Όλες οι δομές δεδομένων που αναφέρθηκαν παρουσιάζονται στο σχήμα 3.2.



Σχήμα 3.2: Βασικές Δομές Δεδομένων του Διαχειριστή Ουρών

Η εξίσωση που μας δίνει την συνολική μνήμη που χρησιμοποιήθηκε από όλες τις δομές σε bits ως συνάρτηση του αριθμού των παρεχόμενων ουρών (N_q) και του αριθμού των κυττάρων (N_c) έχει ως εξής:

$$\begin{aligned} S_{all} &= S_{CBM} + S_{PM} + S_{HTm} + S_{HTe} + S_{FLm} \\ &= 424N_c + N_c(\log_2 N_c + 1) + 2(\log_2 N_c + 1)N_q + N_q + 2(\log_2 N_c + 1) + 1 \end{aligned}$$

Η συνολική μνήμη που χρησιμοποιείται από τους χώρους αποθήκευσης των κυττάρων είναι μεγέθους 424 N_c σε bits και συμβολίζεται με S_{CBM}. Η Pointer Memory πρέπει να έχει πλάτος ίσο με τα bits που χρειάζονται για να διευθυνσιοδοτήσουν όλους τους CBs ($log_2 N_c + 1$) και ύψος ίσο με τον αριθμό των χώρων αποθήκευσης (N_c). Ο Head/Tail table έχει ύψος ίσο με τον αριθμό των υποστηριζόμενων ουρών (N_q) επί 2(κεφαλή και τέλος), και πλάτος ίσο με τα bits που απαιτούνται για να διευθυνσιοδοτήσουν όλους τους CBs. Τα Empty Bits είναι N_q και οι καταχωρητές Free List Head/Tail απαιτούν $log_2 N_c + 1$ bits ο καθένας. Επιπλέον απαιτείται το Free List Empty Bit. Χρησιμοποιώντας αυτή τη εξίσωση μπορούμε εύκολα να υπολογίσουμε την ποσότητα της μνήμης που χρησιμοποιείται για τους δείκτες και να την συγκρίνουμε με αυτήν που χρησιμοποιείται για αποθήκευση των δεδομένων (CBM). Μερικοί ενδεικτικοί αριθμοί για λογικές διαμορφώσεις μνήμης φαίνονται στον Πίνακα 3.1.

N_c	1024	4096	8192	32768	131072
N_q	128	512	1024	4096	16384
$\frac{S_{all}-S_{CBM}}{S_{all}}$	3%	4%	4%	5%	5.1%

Πίνακας 3.1: Ποσοστό μνήμης που χρησιμοποιείται για τις δομές δεδομένων

3.2 ΚΑΘΟΡΙΣΤΙΚΕΣ ΑΠΟΦΑΣΕΙΣ ΤΗΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ

Υπάρχουν αρκετές καθοριστικές αποφάσεις που αφορούν την αρχιτεκτονική σχετικά με τη θέση των δομών δεδομένων στην μνήμη (ή σε πολλές μνήμες):

1. Πολλές χωριστές μνήμες για τις δομές δεδομένων επιτρέπουν παραλληλισμό αλλά έχουν υψηλό κόστος. Για παράδειγμα, αν όλες οι απαιτούμενες δομές δεδομένων

ήταν τοποθετημένες σε μνήμη διπλού πλάτους όπου τα Head, Tail και Empty Bit χωράνε σε μία λέξη, τότε ο χρόνος εκτέλεσης μιας λειτουργίας Enqueue είναι δύο κύκλοι σε σύγκριση με τους τέσσερεις κύκλους σε μια υλοποίηση μνήμης μονού πλάτους όπως θα δούμε στο κεφάλαιο 4. Έτσι η τοποθέτηση των Head, Tail και Empty Bits παίζει σημαντικό ρόλο στο κόστος και στην ταχύτητα ακόμα κι αν διαλέξουμε να έχουμε μία μόνο μνήμη. Υπάρχουν δύο επιλογές όσον αφορά την τοποθέτηση των Head/Tail στον πίνακα Head/Tail table,

(a) Τα Head και Tail να βρίσκονται σε μια λέξη μνήμης

(b) Τα Head και Tail να βρίσκονται σε συνεχείς λέξεις μνήμης όπως φαίνεται στο σχήμα 3.2.

και δύο επιλογές που αφορούν την τοποθέτηση των Empty Bits:

(a) Τα Empty Bits είναι αποθηκευμένα σε έναν ξεχωριστό χώρο μνήμης (χρησιμοποιώντας 100% την υπάρχουσα μνήμη).

(b) Τα Empty Bits κωδικοποιούνται μέσα στο Head, το Tail ή μέσα και στα δυο με αποτέλεσμα τον παραλληλισμό και την επιτάχυνση ορισμένων λειτουργιών.

2. Για να κατακερματίσουμε και να επανασυνδέσουμε πακέτα π.χ. ουρές από κύτταρα, χρειαζόμαστε επιπλέον πληροφορία για ουρές ουρών. Υπάρχουν δύο τρόποι προσθήκης της επιπλέον πληροφορίας: Είτε υλοποιούμε ουρές ουρών χρησιμοποιώντας ένα επιπλέον σύνολο από τις ίδιες δομές δεδομένων (συνδέουμε τις ουρές των κυττάρων κεφαλή με κεφαλή) ή απλά προσθέτουμε ένα επιπλέον bit στην Pointer Memory του προηγούμενου σχήματος για να σηματοδοτήσουμε το τέλος του ενός πακέτου και την αρχή ενός άλλου (συνδέουμε τις ουρές των κυττάρων Τέλος με Κεφαλή). Η δεύτερη υλοποίηση είναι προτιμότερη γιατί στο ATM τα πακέτα λαμβάνονται και αναχωρούν με τρόπο FIFO. Ωστόσο η πρώτη προσφέρει την επιπλέον δυνατότητα να εξάγουμε από την ουρά (i+1) κύτταρα, χωρίς να χρειαστεί να προσπελάσουμε ολόκληρη την ουρά i.

3. Τοποθέτηση των δομών δεδομένων μέσα ή έξω από το ολοκληρωμένο. Η πρώτη λύση προσφέρει ταχύτητα αλλά δεν παρέχει ευελιξία, ενώ έχει και μικρή επεκτασιμότητα ως προς το μέγεθος: Είναι πολύ ευκολότερο και στοιχίζει λιγότερο να αφαιρέσουμε ένα ολοκληρωμένο μνήμης και τοποθετήσουμε ένα μεγαλύτερο σε μέγεθος από το

να χρειαστεί να παράγουμε ένα νέο ASIC προηγμένης τεχνολογίας με περισσότερη μνήμη μέσα στο ολοκληρωμένο.

Μια πρώτη καθοριστική αρχιτεκτονική απόφαση σχετίζεται με το αν πρέπει να τοποθετήσουμε τις δομές μέσα ή έξω από το ολοκληρωμένο. Μια τέτοια αρχιτεκτονική επιλογή είναι πολύ σημαντική γιατί καθορίζει την τεχνολογία που πρέπει να χρησιμοποιηθεί. Οι μνήμες μέσα στο ολοκληρωμένο είναι γρήγορες, μικρές και απαιτούν υλοποίηση σε ASIC του χώρου αποθήκευσης. Παρόλα αυτά υπάρχουν FPGAs με μνήμη μέσα στο ολοκληρωμένο που είναι όμως ακόμη στενές και αργές και δεν προσφέρονται σε μέγεθος κατάλληλο για τις ανάγκες μας. Μια εκτεταμένη συζήτηση μιας αρχιτεκτονικής υλοποιημένης σε ASIC με μνήμη μέσα στο ολοκληρωμένο, μπορεί να βρεθεί στην [4]. Από την άλλη πλευρά μνήμες έξω από το ολοκληρωμένο είναι ευρέως διαδεδομένες και προσφέρονται σε μια ποικιλία ταχυτήτων και μεγεθών. Έχουν επίσης το πλεονέκτημα της επεκτασιμότητας ως προς μέγεθος και ταχύτητα: Είναι πάντα δυνατό να εξάγουμε ένα ολοκληρωμένο με μικρή σε ποσότητα και αργή μνήμη και να τοποθετήσουμε ένα μεγαλύτερο και γρηγορότερο. Η δική μας επιλογή είναι να χρησιμοποιήσουμε ολοκληρωμένα μνήμης του εμπορίου τα οποία είναι χαμηλού κόστους και εύκολα επεκτάσιμα ως προς το μέγεθος και την ταχύτητα. Κατά δεύτερο λόγο, αλλά εξίσου σημαντική, είναι η επιλογή μεταξύ δυναμικής και στατικής μνήμης (DRAM ή SRAM). Είναι αλήθεια ότι οι DRAMs επιτυγχάνουν υψηλό ρυθμό μετάδοσης (π.χ. SDRAM), αλλά η καθυστέρηση είναι ακόμα μεγάλη και το interface πιο πολύπλοκο σε σύγκριση με τις SRAMs. Έτσι καταλήξαμε ότι οι στατικές μνήμες του εμπορίου είναι μια λύση με πολλά πλεονεκτήματα, κατάλληλες για την αρχιτεκτονική μας.

Όλες οι απαιτούμενες δομές δεδομένων μπορούν να τοποθετηθούν σε ξεχωριστές μνήμες, σε μια μνήμη ή μερικές από αυτές σε μια μνήμη και μερικές σε ξεχωριστές μνήμες. Πολλές διαφορετικές μνήμες προσθέτουν στο κόστος της αρχιτεκτονικής γιατί πιο πολλά chips και pins (πιο ακριβή συσκευασία) απαιτούνται. Έτσι μια εξωτερική μνήμη για όλες τις δομές φαίνεται να είναι μια καλή επιλογή λαμβάνοντας υπόψιν το ποσοστό χρησιμοποίησης αυτής και την επεκτασιμότητά της. Ακόμα ένα πλεονέκτημα του να χρησιμοποιούμε μια μόνο μνήμη εκτός του ολοκληρωμένου είναι η ευελιξία, δηλαδή κάθε δομή δεδομένων μπορεί να προγραμματιστεί να χρησιμοποιήσει ένα ποσοστό αυτής της μνήμης. Σίγουρα το μεγάλο μειονέκτημα είναι η σειριοποίηση κάθε λειτουργίας πάνω στις ουρές. Μια βελτιστοποίηση είναι η να τοποθέτηση των Head/Tail Empty Bits (που είναι σχετικά λίγα) σε ξεχωριστή μνήμη μέσα στο ολοκληρωμένο, κάτι που επιτρέπει κάποιον

παραλληλισμό. Παρόλα αυτά η λύση αυτή θα μείωνε την επεκτασιμότητα της αρχιτεκτονικής καθώς ο μέγιστος αριθμός των ουρών θα έπρεπε να μην υπερβαίνει τον παρεχόμενο αριθμό από bits που έχει η μνήμη μέσα στο ολοκληρωμένο. Για υποστήριξη αρχιτεκτονικών με προκαθορισμένο αριθμό ουρών ωστόσο, είναι μια δίκαιη λύση.

Αφού αποφασίσαμε ότι η καλύτερη λύση για τους σκοπούς μας είναι να έχουμε μόνο μια μνήμη για όλες τις δομές, το επόμενο πρόβλημα είναι το πώς να τοποθετήσουμε τους δείκτες Head και Tail του Head/Tail table για κάθε ουρά μέσα στην μνήμη. Εάν τους βάλουμε τον έναν δίπλα στον άλλον έχουμε το πλεονέκτημα της προσπέλασης των Head και Tail μιας ουράς σε έναν κύκλο, και το μειονέκτημα διαπλάτυνσης της μνήμης. Το μέγεθος του δείκτη Κεφαλής πρέπει να είναι τουλάχιστον 13 bits για να διευθυνσιοδοτήσουμε 8192 CBs που είναι λογικός αριθμός μόνο για ένα πρωτότυπο. Ένα εμπορικό chip πρέπει να έχει σαφώς περισσότερα bits (μεταξύ 15-20). Μνήμες με πλάτος μεγαλύτερο από 16 bits τείνουν να είναι ακριβές και δεν υπάρχουν πολλές επιλογές. Τοποθετώντας το Tail κάτω από το Head δηλαδή σε ακολουθιακές διευθύνσεις μπορεί να επιτευχθεί οικονομία στο πλάτος της μνήμης. Επιπλέον παρέχουμε μια πιο επεκτάσιμη αρχιτεκτονική διότι μπορούμε να μετακινηθούμε πιο εύκολα από μνήμες πλάτους 16-bit σε μνήμες πλάτους 32-bit σε σύγκριση με το πέρασμα από μνήμες πλάτους 32-bit σε μνήμες πλάτους 64-bit. Φυσικά θα πρέπει να πληρώσουμε το τίμημα για τον διπλασιασμό του χρόνου προσπέλασης των Head και Tail για την εντολή Dequeue.

Μια άλλη καθοριστική αρχιτεκτονική απόφαση αφορά το πώς θα τοποθετήσουμε τα Head/Tail Empty Bits: Ακόμα κι αν τα τοποθετήσουμε στην ίδια μνήμη μαζί με τις άλλες δομές, υπάρχει ένας ικανός αριθμός επιλογών. Μπορούμε να τοποθετήσουμε τα Empty Bits μαζί με το Head ή το Tail, ή δίπλα και στα δύο (στον Head/Tail table). Με αυτό το τρόπο μπορούμε να διαβάσουμε το Empty Bit κάθε ουράς μαζί με το Head ή Tail. Τοποθετώντας το Empty Bit δίπλα στους δείκτες Head και Tail καθυστερούμε όμως την εντολή Enqueue γιατί πρέπει να διαβάσουμε και να γράψουμε τον δείκτη Κεφαλής (αυτό δεν θα το κάναμε φυσιολογικά) στην περίπτωση που θα εκτελέσουμε μια Enqueue σε άδεια ουρά. Αλλά αν το τοποθετήσουμε δίπλα στον δείκτη Tail μόνο, κερδίζουμε την επιτάχυνση της Enqueue κατά έναν κύκλο (διαβάζουμε τα Tail και Empty Bit σε έναν κύκλο), και το ίδιο ισχύει στην εντολή Dequeue ,όπου έχουμε να διαβάσουμε τα Head και Tail μαζί. Σε κάθε μια από αυτές τις υλοποιήσεις κερδίζουμε σε ταχύτητα αλλά χάνουμε σε ποσοστό χρησιμοποίησης μνήμης. Μια άλλη εναλλακτική λύση είναι να τοποθετήσουμε τα Head/Tail Empty

Bits σε ένα ξεχωριστό χώρο μνήμης μέσα στην ίδια μνήμη όπου βρίσκονται και οι υπόλοιπες δομές δεδομένων. Με αυτόν τον τρόπο χρησιμοποιούμε πλήρως την υπάρχουσα μνήμη με το επιπλέον κόστος ενός ακόμα κύκλου κατά την εκτέλεση της εντολής Enqueue. Για παράδειγμα εάν η εξωτερική μνήμη έχει πλάτος 16 bits και θέλουμε να υλοποιήσουμε 1024 ουρές, τότε 2/32 της συνολικής μνήμης που χρησιμοποιήθηκε για την δομή Head/Tail table έχει σπαταληθεί ή 2048 bits βρίσκονται σε αχρηστία εάν διαλέξουμε να υλοποιήσουμε τα Empty Bits στο Head/Tail table. Αυτά τα bits μπορούν να σχηματίσουν 2048/16=128 δείκτες για χρησιμοποίηση στην Pointer Memory, που σημαίνει 128 περισσότερους δείκτες κυττάρων και δυνατότητα να έχουμε 128 περισσότερες θέσεις αποθήκευσης κυττάρων εάν υπάρχει διαθέσιμη μνήμη στη CBM.

Κάτι που επίσης πρέπει να λάβουμε υπόψιν μας είναι το πώς θα υλοποιήσουμε τις ιεραρχικές ουρές (ουρές ουρών). Υπάρχουν δύο δυνατές υλοποιήσεις:

1. Είτε συνεχίζοντας να προσθέτουμε κύτταρα στο τέλος της ουράς, ακόμα κι αν η ουρά περιέχει μια ολόκληρη AAL-5 PDU (σύνδεση ουρών κεφαλής με τέλος) και μαρκάροντας το τελευταίο κύτταρο σαν το τέλος της PDU,

2. Είτε χρησιμοποιώντας άλλο ένα σύνολο ουρών που έχουν σαν στοιχεία τους ουρές κυττάρων (σύνδεση ουρών κεφαλή με κεφαλή).

Στην πρώτη περίπτωση κάθε ουρά ουρών θεωρείται σαν μια σειρά κυττάρων που σχηματίζουν σειρές από AAL-5 PDUs. Στην δεύτερη περίπτωση κάθε AAL-5 PDU θεωρείται σαν ένα στοιχείο μιας άλλης ομάδας ουρών. Το πλεονέκτημα στην πρώτη περίπτωση είναι ότι δεν απαιτείται άλλη Free List και το μειονέκτημα ότι δεν μας παρέχει τη λειτουργικότητα εξαγωγής μιας ολόκληρης AAL-5 PDU από την ουρά, πριν εξάγουμε όλα τα κύτταρά της ένα προς ένα. Το αντίστροφο ισχύει στη δεύτερη περίπτωση. Ωστόσο, μια προσεκτικότερη μελέτη του ATM μας υποδυκνείει την καλύτερη επιλογή για την περίπτωσή μας: Στο AAL-5 για μια συγκεκριμένη σύνδεση, κανένα κύτταρο δεν προσπερνά κάποιο αρχαιότερο σε χρόνο άφιξης, λόγω της βασισμένης σε συνδέσεις φύσης του πρωτοκόλλου. Έτσι η καλύτερη επιλογή είναι η απλούστερη κατά την οποία χρησιμοποιούμε ένα επιπλέον bit σε κάθε CP για να σημειώσουμε το τέλος της AAL-5 PDU και να σηματοδοτήσουμε την έναρξη της επόμενης. Αυτό το bit ονομάζεται End-of-PDU bit ή EoP bit (δες και κεφάλαιο 3.1).

ΚΕΦΑΛΑΙΟ 4

ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΔΙΑΧΕΙΡΙΣΤΗ ΟΥΡΩΝ ΣΕ ΥΛΙΚΟ

Σε αυτό το κεφάλαιο περιγράφουμε 3 διαφορετικών ειδών υλοποιήσεις διαχείρισης ουρών:

- Η υλοποίηση που χρησιμοποιήθηκε στο πρωτότυπο MuQPro I
- Μία υλοποίηση παρόμοια με αυτή του MuQPro I, που επιτυγχάνει 100% χρησιμοποίηση της μνήμης προσφέροντας χαμηλότερη ταχύτητα,
- Και μία υλοποίηση που επιτυγχάνει τον μέγιστο δυνατό παραλληλισμό χωρίς να χρησιμοποιεί τεχνικές σχεδίασης όπως VLIW ή pipelining.

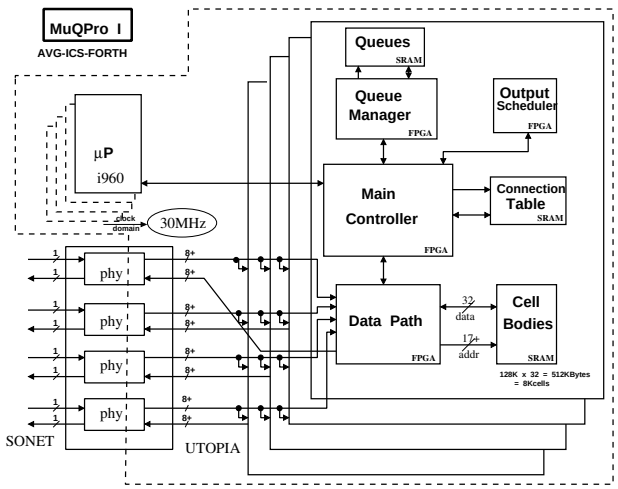
Η πρώτη υλοποίηση, που έχει χρησιμοποιηθεί στο MuQPro I, είναι μία 16-bit αρχιτεκτονική που τοποθετεί όλες τις δομές δεδομένων σε μία μνήμη με τα Empty Bits ενσωματωμένα στο πεδίο Tail του Head/Tail table. Χρησιμοποιεί ρολόϊ 30 MHz και διατηρεί δύο σύνολα ουρών, ένα μόνο για κύτταρα και ένα άλλο για να προγραμματίζει τις συνδέσεις σε μία από τις 8 διαθέσιμες προτεραιότητες. Ο διαχειριστής ουρών MuQPro I μπορεί να εξυπηρετεί πέντε συνδέσμους των 155 Mb/s SONET OC-3 (τέσσερις εισερχόμενους και έναν εξερχόμενο), τον μικροεπεξεργαστή που διαχειρίζεται τα κύτταρα OAM και τον προγραμματιστή που ταξινομεί τα 1024 VPI/VCIs σε οκτώ προτεραιότητες σύμφωνα με τον αλγόριθμο Highest Priority First όπως περιγράφεται στο [3]. Ο QM υλοποιήθηκε σε συντηρητική τεχνολογία FPGA.

Η δεύτερη υλοποίση -επίσης 16-bit- έχει μία βασική διαφορά από αυτή του MuQPro I: απαιτεί μόνο έναν επιπλέον κύκλο για την εκτέλεση της εντολής Enqueue, αλλά χρησιμοποιεί πλήρως την υπάρχουσα μνήμη. Χρησιμοποιεί ρολόϊ των 30 MHMHz και χωράει μέσα σε ένα EPF10K40 FPGA. Τέλος, η επονομαζόμενη ‘‘πλήρως παράλληλη’’ υλοποίηση είναι μία αρχιτεκτονική που επιδεικνύει τον διαθέσιμο παραλληλισμό της διαχείρισης ουρών, χρησιμοποιώντας τον περιορισμένο ρυθμό μετάδοσης της εσωτερικής μνήμης της FPGA και όσα pins είναι αναγκαία για να υπάρξει πλήρης παραλληλισμός. Στο Κεφάλαιο 6 γίνεται μία εκτεταμένη παρουσίαση με τα μειονεκτήματα και τα πλεονεκτήματα όλων των αρχιτεκτονικών.

4.1 Η ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΔΙΑΧΕΙΡΙΣΤΗ ΟΥΡΩΝ MuQPro I

Το MuQPro I είναι το πρωτότυπο ενός υποσυστήματος μεταγωγέα τεσσάρων εισόδων και μίας εξόδου που υλοποιείται σε FPGA. Επιδεικνύει την χρήση ενός μοντέλου διαχείρισης ουρών το οποίο υποστηρίζει πλήρως όλες τις απαραίτητες λειτουρίες που εκτελούνται από τον μεταγωγέα. Η αρχιτεκτονική του συστήματος το οποίο λειτουργεί στα 30 MHz παρουσιάζεται στο σχήμα 4.1 όπου τέσσερις πλακέτες απαιτούνται για να μετατρέψουν το σύστημα σε ένα μεταγωγέα 4x4. Σε κάθε μία από τις πλακέτες υπάρχουν 5 στοιχεία SONET OC3 τα οποία μεταδίδουν κυκλοφορία 155 Mb/s προς και από το δίκτυο καταλήγοντας σε μία συνολική παροχή κοντά στο 1 Gb/s. Το μονοπάτι δεδομένων (datapath) είναι ο φυσικός δρόμος των κυττάρων από τα στοιχεία SONET προς και από την μνήμη SRAM (Cell Bodies) όπου όλα τα κύτταρα ATM αποθηκεύονται μαζί με τις επικεφαλίδες τους. Άλλη μια στατική μνήμη (connection table) χρησιμοποιείται για την φύλαξη πληροφοριών ανά σύνδεση VPI/VCI τις οποίες ανανεώνει σε τακτά χρονικά διαστήματα ο μικροεπεξεργαστής (μP) i960. Ο μικροεπεξεργαστής υλοποιεί Quantum Flow Control (ένας υλοποιημένος σε λογισμικό μηχανισμός ελέγχου ροής βασισμένος σε εισιτήρια), αποσπά/εισάγει κύτταρα διαχείρισης του ATM (κύτταρα OAM) και χρησιμοποιείται για την αρχικοποίηση και έλεγχο όλων των στοιχείων του συστήματος. Προκειμένου η αρχιτεκτονική του συστήματος να γίνει πιο γενική χρησιμοποιείται ένας ελεγκτής ο οποίος παρέχει το δίκτυο διασύνδεσης και λειτουργίες διεπαφής ανάμεσα στα διάφορα στοιχεία. Ο προγραμματιστής εξόδου (scheduler) υλοποιεί τον αλγόριθμο Highest Priority First οργανώνοντας τα κύτταρα για μετάδοση. Τέλος ο Διαχειριστής Ουρών (QM module) υλοποιεί όλες τις απαιτούμενες λειτουργίες διαχείρισης μνήμης και είναι αυτός ο οποίος θα παρουσιαστεί σε

αυτό το κεφάλαιο.



Σχήμα 4.1: Αρχιτεκτονική του συστήματος MuQPro I

Γενικά η ροή δεδομένων που αφορά διαχείριση ουρών είναι :

- Από το μονοπάτι δεδομένων στον διαχειριστή ουρών για αίτηση και αποδέσμευση διευθύνσεων της μνήμης Cell Bodies προκειμένου το μονοπάτι δεδομένων να μεταφέρει σώματα κυττάρων προς/από τα SONET I/Fs και προς/από το μP διαμέσου του κύριου ελεγκτή.

- Από τον μP προς τον Διαχειριστή Ουρών για εισαγωγή και εξαγωγή διευθύνσεων κυττάρων OAM (στην μνήμη Cell Bodies) και για βοήθεια στον έλεγχο λαθών του Διαχειριστή Ουρών.

- Από τον προγραμματιστή εξόδου (Output Scheduler) έτσι ώστε να εισάγει/εξάγει συνδέσεις (VPI/VCIs) σε ουρές προτεραιοτήτων.

Ο Διαχειριστής Ουρών έχει την δυνατότητα να εξυπηρετεί όλες τις ροές δεδομένων με την απαιτούμενη ταχύτητα χωρίς να θυσιάζει τα χαρακτηριστικά που τον κάνουν αρκετά γενικό ώστε να μπορεί να χρησιμοποιηθεί σε πολλών ειδών συστήματα ATM. Παρατηρείστε ότι ο Διαχειριστής Ουρών δεν περιορίζεται σε 8 προτεραιότητες (8 ουρές προτεραιοτήτων πιο σωστά), 1024 ουρές κυττάρων και 8192 στοιχεία (δηλαδή διευθύνσεις) ανά ουρά. Ο μόνος περιορισμός που έχει είναι το 16-bit μονοπάτι δεδομένων (datapath) που του

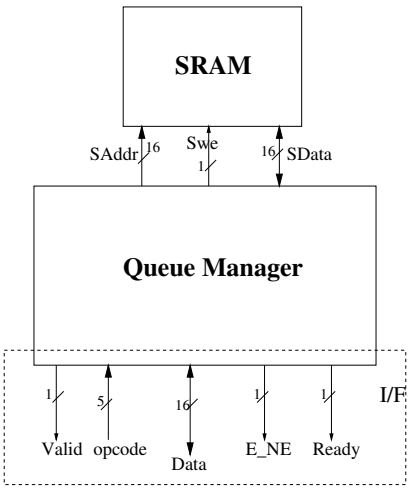
Opcode	Instruction
X1XXX	NOOP
Y0$I_2I_1I_0$	instruction $I_2I_1I_0$ on set of queues Y

Πίνακας 4.1: Επεξήγηση της εντολής NOOP

ορίζει ότι μπορεί να προσπελάσει μέχρι 64K στοιχεία καθώς και το μέγεθος της εξωτερικής στατικής μνήμης (SRAM).

4.1.1 ΔΙΕΠΑΦΗ (INTERFACE)

Η διεπαφή του QM με τις εξωτερικές συσκευές παρουσιάζεται στο σχήμα 4.2. Γενικά ο QM είναι μια συσκευή σκλάβος (slave) σύγχρονη ως προς το ρολόϊ που πολυπλέκει τα ορίσματα εντολών και την απάντηση πάνω από τα ίδια σύρματα (Data). Πιο λεπτομερώς τα σήματα εισόδου/εξόδου είναι:



Σχήμα 4.2: Γενικό διάγραμμα του Διαχειριστή Ουρών

Opcode (κώδικας εντολής): Ο κώδικας εντολής της εντολής είναι μεγέθους 5 bits με το σημαντικότερο ψηφίο (bit 5) να επιλέγει το σύνολο το σύνολο από ουρές που θα ενεργοποιηθεί ενώ το bit 4 επιλέγει τον κώδικας εντολής NOOP. Πιο περιληπτικά περιγράφονται παρακάτω οι κώδικες εντολής όλων των εντολών. Μια λεπτομερής περιγραφή όλων των εντολών μπορεί να βρεθεί στο Παράρτημα Α.

Opcode ($I_2I_1I_0$)	0	1	2	3	4	5	6	7
Instruction	Init	Enqueue	Write	Dequeue	Retfree	Read	Getfree	Top
Arguments	4	2	2	1	1	1	0	1

Πίνακας 4.2: Οι κώδικες εντολών του MuQPro I

Data: Είναι ένας διπλής κατεύθυνσης δίαυλος που χρησιμοποιείται για να περνούν τα ορίσματα των εντολών που δίνονται και οι απαντήσεις που λαμβάνονται. Ένας κύκλος (turnaround cycle) απαιτείται μεταξύ διαδοχικών εγγραφών στον δίαυλο από τον QM και την εξωτερική συσκευή. Η πολύπλεξη των ορισμάτων εντολών δεν δημιουργεί καθυστέρηση γιατί όσο ο QM ετοιμάζει την διεύθυνση για την προσπέλαση της εξωτερικής μνήμης, μπορεί να δέχεται το δεύτερο όρισμα μιας εντολής. Ο πλάτους 16-bit δίαυλος είναι ικανοποιητικός για τις ανάγκες του MuQPro I που χρησιμοποιεί 1024 ουρές (10 bits) και ένα μέγιστο αριθμό από 8192 στοιχεία (13 bits) και άρα ουσιαστικά δεν χρειάζεται να είναι πλάτους μεγαλύτερου από 13 bits.

E/NE: Το σήμα αυτό χρησιμεύει για να δείχνει αν μια εκτελούμενη εντολή άλλαξε την κατάσταση μιας εσωτερικής ουρά (π.χ. από άδεια σε μη άδεια).

Valid: Το σήμα αυτό χρησιμοποιείται για να δείχνει πότε μια εντολή που εκτελείται επιστρέφει απάντηση (είτε μέσω των Data είτε μέσω του E/NE) και χρειάζεται ώστε να μην αναγκαστεί η εξωτερική συσκευή να έχει κάποιο μετρητή για να καταλαβαίνει πότε έρχεται η απάντηση.

Ready: Το σήμα αυτό τίθεται όποτε ο QM είναι έτοιμος να λάβει μια νέα εντολή στην αμέσως επόμενη θετική ακμή του ρολογιού. Η χρησιμότητά του έγκειται στο να μπορεί η εξωτερική συσκευή να παρέχει μια νέα εντολή πριν τερματίσει την εκτέλεσή της η τρέχουσα (instruction overlapping).

4.1.2 ΤΟ ΣΥΝΟΛΟ ΕΝΤΟΛΩΝ

Init(arg1,arg2,arg3,arg4): Θέτει τον χώρο διευθύνσεων που καταλαμβάνει κάθε μια από τις απαιτούμενες δομές. Τα arg1, arg2 ορίζουν ότι ο Head/Tail table ξεκινά από την διεύθυνση arg1 και χρησιμοποιεί arg2 ουρές. Τα arg3, arg4 καθορίζουν ότι η Pointer

memory ξεκινά από την διεύθυνση arg3 και θα έχει arg4 στοιχεία. Επίσης αρχικοποιεί τις δομές αυτές καθώς και την Free List (για περισσότερες πληροφορίες στο Παράρτημα Α).

EnQ(arg1,arg2): Εισάγει στην ουρά arg1 την διεύθυνση arg2. Αν το E/NE έχει τεθεί τότε η ουρά ήταν άδεια πριν από αυτή την εντολή.

DeQ(arg1): Εξάγει από την ουρά arg1 το κορυφαίο στοιχείο. Αν το E/NE έχει τεθεί τότε η ουρά άδειασε μετά από αυτή την εντολή.

Top(arg1): Επιστρέφει το κορυφαίο στοιχείο (διεύθυνση) της ουράς arg1 χωρίς όμως να το εξάγεις από την ουρά. Αν η ουρά είναι άδεια τότε το σήμα E/NE τίθεται για ένα κύκλο.

GetFree(): Εξάγει το κορυφαίο στοιχείο (διεύθυνση) της Free List. Αν η Free List άδειασε μετά από την εκτέλεση αυτής της εντολής τότε το σήμα E/NE τίθεται για ένα κύκλο.

RetFree(arg1): Εισάγει στην Free List τη διεύθυνση arg1. Αν το E/NE έχει τεθεί τότε η ουρά ήταν άδεια πριν από αυτή την εντολή.

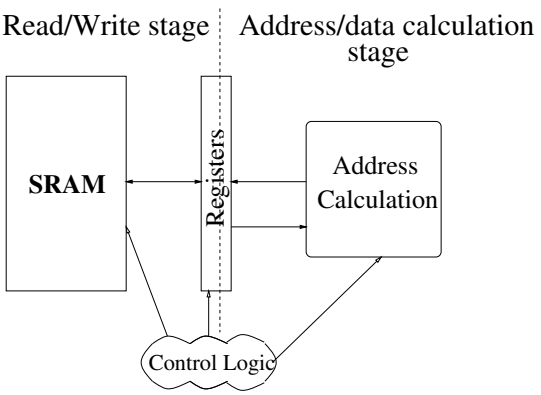
Read(arg1): Διαβάζει και επιστρέφει τα περιεχόμενα της διεύθυνσης arg1 της SRAM. Χρησιμοποιείται μόνο για ανίχνευση σφαλμάτων.

Write(arg1,arg2): Γράφει τα δεδομένα arg2 στην διεύθυνση arg1. Χρησιμοποιείται μόνο για ανίχνευση σφαλμάτων.

4.1.3 ΕΣΩΤΕΡΙΚΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ

Ο διαχειριστής ουρών διατηρεί δύο σύνολα ουρών, ένα για να συνδέει κύτταρα και ένα για συνδέσεις (VPI/VCIs). Έχουμε ακολουθήσει μια ιεραρχική προσέγγιση σχεδίασης: σχεδιάσαμε πρώτα ένα στοιχείο που επιτελεί πράξεις σε ένα σύνολο ουρών και μετά συνδιάσαμε δύο αντίγραφα μαζί για να δημιουργήσουμε το ζητούμενο (2 σύνολα ουρών). Με αυτό τον τρόπο ο χρόνος σχεδίασης και εύρεσης λαθών μειώθηκε σημαντικά. Σε κάθε ουρά μπορούμε να εκτελέσουμε το πλήρες σύνολο εντολών που περιγράψαμε προηγουμένως.

Γενικότερα ο Διαχειριστής Ουρών (ας θεωρήσουμε ένα σύνολο ουρών) είναι ένας υπολογιστής διευθύνσεων και δεδομένων (Σχήμα 4.3) που προσπαθεί να απασχολεί την εξωτερική στατική μνήμη όσο το δυνατόν περισσότερο δεδομένου ότι υπάρχει συνεχής ροή



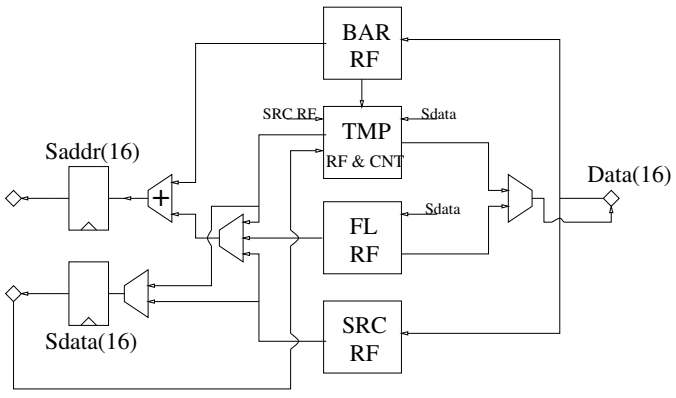
Σχήμα 4.3: Ο Διαχειριστής Ουρών σαν υπολογιστής διευθύνσεων

εντολών από την εξωτερική συσκευή προς τον QM. Από την άλλη πλευρά το pipelining είναι μια αποδοτική λύση για υψηλή χρησιμοποίηση του υπάρχοντος υλικού και προσπαθήσαμε να το χρησιμοποιήσουμε στην αρχιτεκτονική μας παρόλη την σειριοποίηση των λειτουργιών λόγω του κοινού πόρου (στατική μνήμη). Ωστόσο, διαιρώντας τις λειτουργίες σε τρία μέρη, μια για τον υπολογισμό των κατάλληλων δεδομένων και διευθύνσεων, μια για προσπέλαση στην στατική μνήμη και μια για την υποδοχή (fetching) των εντολών καταφέρνουμε να έχουμε αλληλοκαλυπτόμενες εκτελούμενες εντολές. Στο Σχήμα 4.4 βλέπουμε ένα εσωτερικό διάγραμμα του QM. Υπάρχουν 4 πίνακες καταχωρητών:

- BAR RF : Διατηρεί τις διευθύνσεις βάσης όλων των δομών (Head/Tail table, Pointer Memory) καθώς και τα μεγέθη τους.
- SRC RF : Διατηρεί τα ορίσματα εντολών.
- FL RF : Διατηρεί την αρχή το τέλος καθώς και την πληροφορία αν είναι άδεια ή όχι η Λίστα Ελευθέρων Διευθύνσεων (Free List).
- TMP RF & CNT : προσωρινοί καταχωρητές που αποθηκεύουν τις προηγούμενες τιμές των Head, Tail μιας ουράς καθώς και τον μετρητή για την αρχικοποίηση της Free List.

Τρεις λειτουργίες μπορούν να συμβαίνουν παράλληλα σε αυτό το μονοπάτι δεδομένων κατά την διάρκεια ενός κύκλου: Κατ' αρχήν μια προσπέλαση στην εξωτερική SRAM όπου η διεύθυνση και τα δεδομένα δίνονται από τους καταχωρητές Saddr και Sdata αντίστοιχα.

Αν πρόκειται για ανάγνωση τότε τα εισερχόμενα δεδομένα πηγαίνουν στον πίνακα καταχωρητών TMP RF. Δεύτερον, μια νέα διεύθυνση και δεδομένα υπολογίζονται για να χρησιμοποιηθούν στον επόμενο κύκλο και αποθηκεύονται στους καταχωρητές Saddr και Sdata. Τέλος, μια νέα εντολή μπορεί να έρθει και τα ορίσματά της αποθηκεύονται στον SRC RF. Η αποδοτική χρησιμοποίηση του παραλληλισμού των λειτουργιών αυτών μπορεί να επιτευχθεί μόνο με κατάλληλη σχεδίαση του μονοπατιού ελέγχου. Πιο συγκεκριμένα, κάθε εντολή μπορεί να διασπαστεί σε μια σειρά από προσπελάσεις στην μνήμη που τις καλούμε μικρολειτουργίες. Ένας βέλτιστος προγραμματισμός των λειτουργιών αυτών είναι απαραίτητος για να έχουμε τον μέγιστο δυνατό παραλληλισμό για κάθε εντολή, δεδομένου ότι όλες οι δομές βρίσκονται σε μια μνήμη. Όσο έξυπνο προγραμματισμό των λειτουργιών αυτών κι αν κάνουμε για κάθε εντολή ξεχωριστά δεν είναι δυνατό να χρησιμοποιούμε και τις τρεις αυτές διαθέσιμες λειτουργίες συνεχώς σε κάθε κύκλο. Για παράδειγμα η εντολή EnQ δεν μπορεί να πραγματοποιήσει την πρώτη προσπέλαση στην μνήμη πριν να λάβει το πρώτο της όρισμα και να υπολογίσει την διεύθυνση. Γι' αυτό τον λόγο, περισσότερες από μία εντολές πρέπει να μπορούν να εκτελεστούν σε κάθε κύκλο. Η τεχνολογία που χρησιμοποιούμε (FPGA) δεν επιτρέπει μεγάλο αριθμό από αθροιστές (για τον υπολογισμό των διευθύνσεων) και καταχωρητών που να λειτουργούν σε υψηλή ταχύτητα. Γι' αυτό τον λόγο, προτιμήσαμε την φθηνότερη σε πόρους τεχνική της επικάλυψης και με λίγο πολυπλοκότερο έλεγχο επιτύχαμε την επικάλυψη 2 εντολών.



Σχήμα 4.4: Αναλυτικό διάγραμμα του Διαχειριστή Ουρών

Ο θεωρητικά ελάχιστος αριθμός κύκλων, δεδομένου ότι χρησιμοποιούμε μια κοινή μνήμη για όλες τις δομές, διαφέρει από τον πραγματικό αριθμό κύκλων που χρειάζεται κάθε

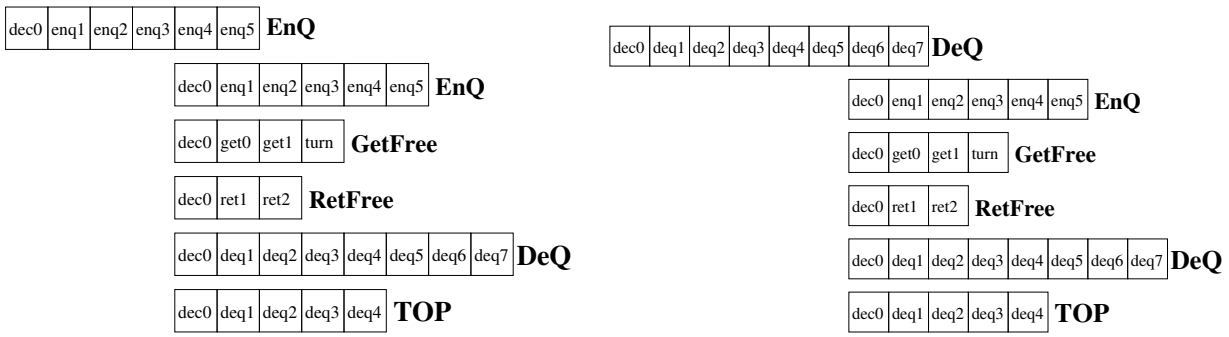
εντολή για να εκτελεστεί. Η διαφορά αυτή προκύπτει από τα παρακάτω:

- Δεν υπάρχει χρόνος υπολογισμού μιας διεύθυνσης που βασίζεται σε κάποιο όρισμα της εντολής μέσα στον ίδιο κύκλο που εισέρχεται το όρισμα στο σύστημα. Ο υπολογισμός μιας διεύθυνσης χρειάζεται έναν ολόκληρο κύκλο.

- Χρειάζεται περίπου μισός κύκλος για την αποκωδικοποίηση της εντολής.

Συνεπώς χρειάζεται τουλάχιστον 1.5 κύκλος παραπάνω από τον θεωρητικά βέλτιστο για την εκτέλεση μιας εντολής.

Η εντολή EnQ απαιτεί 6 κύκλους για να τερματίσει την λειτουργία της, ενώ το βέλτιστο είναι 4. Ωστόσο η τεχνική της επικάλυψης (Σχήμα 4.5) μας επιτρέπει να ξεκινάμε την εκτέλεση μιας νέας εντολής EnQ κάθε 4 κύκλους. Όσο περισσότερες εντολές είναι διαθέσιμες για να δοθούν στον QM τόσο πλησιάζει ο αριθμός των κύκλων εκτέλεσης της εντολής EnQ τον βέλτιστο αριθμό των τεσσάρων. Το ίδιο ισχύει, όσον αφορά την επικάλυψη, για τα περισσότερα ζεύγη εντολών με λίγες εξαιρέσεις που θα αναφερθούν πιο κάτω. Αναλυτικά διαγράμματα για κάθε εντολή δίνονται στο Παράρτημα Α.



Σχήμα 4.5: Επικάλυψη εντολών που ακολουθούν την Enqueue και την Dequeue

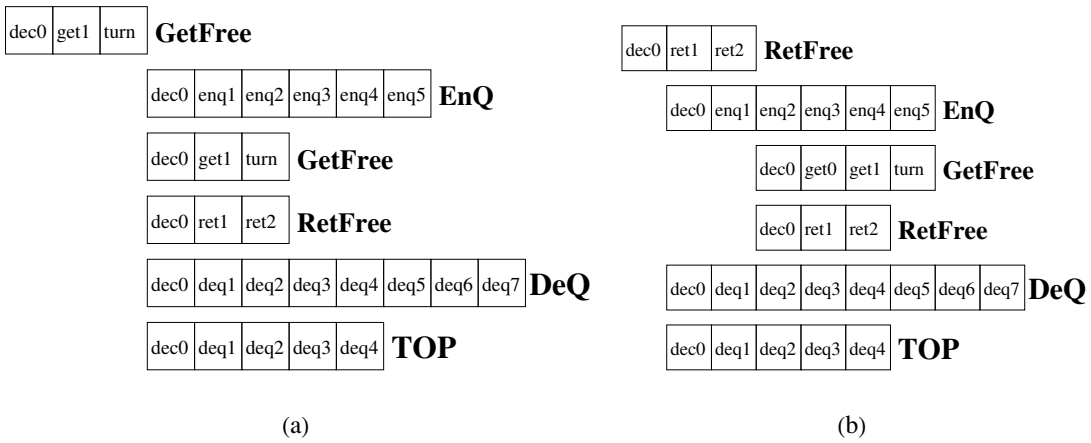
Η εντολή DeQ απαιτεί 4, 7 ή 8 κύκλους για να ολοκληρωθεί ανάλογα με το αν η ουρά είναι άδεια, περιέχει ένα μόνο στοιχείο, ή περιέχει περισσότερα από ένα στοιχεία. Στο σχήμα 4.5(b) βλέπουμε την επικάλυψη που έχει αυτή η εντολή σε σχέση με τις υπόλοιπες. Επιστρέφει το κορυφαίο στοιχείο της ουράς που ζητήθηκε στον κύκλο 5 και απαντά αν η ουρά άλλαξε κατάσταση (δηλαδή αν έγινε άδεια) στον κύκλο 6. Αν η ουρά είναι άδεια δεν

επιτρέπεται επικάλυψη (ούτως ή άλλως η εντολή τερματίζει νωρίς). Στις άλλες περιπτώσεις είναι άχρηστο να δοθεί μια νέα εντολή στον κύκλο 6, γιατί ο κύκλος αυτός χρησιμοποιείται σαν κύκλος turnaround (ορισμός του κύκλου turnaround δίνεται στο Παράρτημα Α.3).

Η εντολή GetFree ολοκληρώνεται σε 3 κύκλους και δεν επιτρέπει επικάλυψη άλλων εντολών κατά την διάρκεια εκτέλεσής της, όπως φαίνεται και από το σχήμα 4.6(a). Στον κύκλο 2 απαντάει με μια διεύθυνση σε ένα ελεύθερο χώρο μνήμης (buffer) ενώ ο κύκλος 3 χρησιμοποιείται σαν κύκλος turnaround. Για να επιταχύνουμε την εκτέλεση αυτής, όποτε ο QM δεν εκτελεί κάποια εντολή υπολογίζει την διεύθυνση για την πρώτη προσπέλαση στην μνήμη, ώστε να μπορεί να την επιτελέσει αμέσως μόλις έρθει η εντολή GetFree. Αυτό είναι δυνατό αφού η εντολή αυτή δεν χρειάζεται ορίσματα και άρα μπορεί πάντοτε να υπολογίζεται η πρώτη διεύθυνση προσπέλασης. Παρόλο που δεν μπορεί καμμιά νέα εντολή να έρθει σε επικάλυψη με την GetFree, το αντίστροφο ισχύει και η GetFree μπορεί να επικαλύπτεται με άλλες εντολές κατά 1 κύκλο. Η επικάλυψη της GetFree με τις υπόλοιπες εντολές (EnQ, DeQ) είναι ένας κύκλος αντί των αναμενόμενων δύο, γιατί η GetFree ουσιαστικά χρησιμοιεί και τους κύκλους που ο QM δεν επιτελεί κάποια εργασία, για να υπολογίζει την πρώτη διεύθυνση προσπέλασης της μνήμης.

Η εντολή RetFree είναι η γρηγορότερη όλων των εντολών καθώς μπορεί να εκτελεστεί σε 3 κύκλους ενώ μια καινούρια εντολή (EnQ, DeQ) μπορεί να ξεκινήσει τον αμέσως επόμενο κύκλο από αυτόν που δόθηκε η RetFree, όπως φαίνεται στο σχήμα 4.6(b). Ωστόσο έχει και έναν περιορισμό: αν υποθέσουμε ότι έχει ξεκινήσει την εκτέλεσή της μια εντολή EnQ και στον κύκλο 4 ξεκινήσει μια RetFree, τότε στον κύκλο 5 δεν μπορεί να ξεκινήσει άλλη μία RetFree. Αυτό συμβαίνει γιατί δεν επιτρέπουμε την σύγχρονη εκτέλεση περισσοτέρων από 2 εντολές λόγω της υψηλής πολυπλοκότητας που θα προσετίθετο στο μονοπάτι ελέγχου.

Η εντολή Top έχει την ίδια λειτουργικότητα με την DeQ μέχρι τον κύκλο 5 (ολοκληρώνει την εκτέλεσή της σε 5 κύκλους). Όσον αφορά τις εντολές Read και Write κάνουν απλή πρόσβαση στην μνήμη και δεν έχει γίνει καμμία βελτιστοποίηση για αυτές. Η εντολή Init εκτελείται μόνο κατά την αρχικοποίηση του συστήματος και δέχεται σαν ορίσματα την διεύθυνση στην κοινή μνήμη όπου ξεκινά η κάθε μια από τις δομές (Pointer memory,



Σχήμα 4.6: Επικάλυψη εντολών που ακολουθούν την Getfree και την Retfree

Head/Tail table) καθώς και το μέγεθος αυτών. Αρχικοποιεί την λίστα ελευθέρων διευθύνσεων (Free List) διασχίζοντας την Pointer memory και γράφοντας συνεχόμενες διευθύνσεις ώστε να γεμίσει η Free list. Επίσης αρχικοποιεί όλες τις ουρές σε ‘‘άδειες’’ διασχίζοντας όλα τα Empty Bits και θέτοντας τα σε 1.

4.1.4 ΑΠΑΙΤΗΣΕΙΣ ΑΠΟ ΤΗΝ ΕΞΩΤΕΡΙΚΗ SRAM ΚΑΙ ΤΗΝ ΕΞΩΤΕΡΙΚΗ ΣΥΣΚΕΥΗ

Η εξωτερική στατική μνήμη πρέπει να επιστρέφει τα ζητούμενα δεδομένα 10-15 ns πριν την επόμενη θετική ακμή του ρολογιού στην περίπτωση της ανάγνωσης. Αυτό σημαίνει πρακτικά ότι για ρολόϊ των 33 ns όπως αυτό του MuQPro I, χρειαζόμαστε μια ασύγχρονη στατική μνήμη με χρόνο προσπέλασης τα 20-25ns. Επιπρόσθετα ο κώδικας εντολής και τα δεδομένα πρέπει να είναι σωστά πριν την μέση του κύκλου, ώστε ο αποκωδικοποιητής να προλάβει να αποκωδικοποιήσει την τρέχουσα εντολή και να αποθηκευτούν τα ορίσματα εντολών στους εσωτερικούς καταχωρητές πριν το τέλος του κύκλου.

4.1.5 ΣΥΝΕΝΩΝΟΝΤΑΣ ΔΥΟ ΔΙΑΧΕΙΡΙΣΤΕΣ ΟΥΡΩΝ

Για να σχεδιάσει κανείς δύο σύνολα ουρών υπάρχουν δύο προσεγγίσεις:

- Διπλασιασμός των πολυπλεκτών και των μονοπατιών δεδομένων καθώς και προσθήκη ενός επιπλέον πολυπλέκτη 2 σε 1 σε κάθε ζεύγος πολυπλεκτών (ενός πολυπλέκτη που υπήρχε προηγούμενα και του προστιθέμενου). Οι νέοι πολυπλέκτες σχετίζονται με το δεύτερο σύνολο ουρών.

- Διπλασιασμός του δρόμου δεδομένων και του ελέγχου (δηλαδή χρησιμοποίηση 2 QMs).

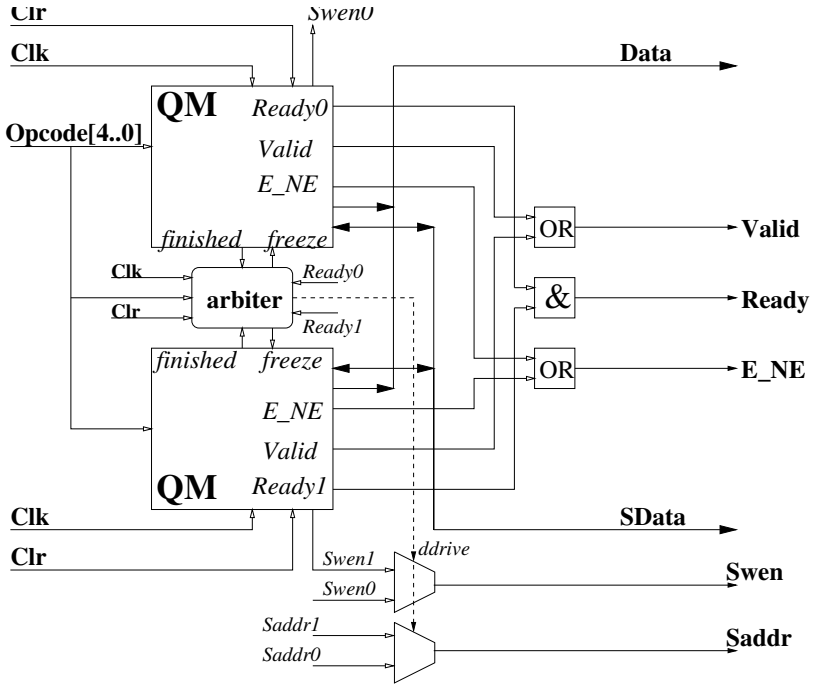
Η πρώτη λύση χρησιμοποιεί μικρότερη λογική αλλά στην τεχνολογία FPGA μεγάλοι πολυπλέκτες έχουν μεγάλη καθυστέρηση. Για αυτό τον λόγο διαλέξαμε την δεύτερη λύση όπου όμως χρειάζεται επιπλέον λογική που να υλοποιεί την διαιτησία μεταξύ των 2 QMs.

Στο Σχήμα 4.7 βλέπουμε την διασύνδεση 2 QMs. Η ανάγκη για το κύκλωμα διαιτησίας πηγάζει από το γεγονός ότι δύο εντολές διαφορετικών QM μπορούν να επικαλυφθούν ενώ και οι δύο QMs χρησιμοποιούν σαν κοινούς πόρους τα σήματα εισόδου/εξόδου και την εξωτερική στατική μνήμη. Αν δύο εντολές εκτελούνται κάποια χρονική στιγμή συγχρόνως -μια που αφορά τον QM_0 και μια που αφορά τον QM_1- τότε κάποια στιγμή θα χρειαστεί να αποφασιστεί ποιός θα οδηγεί τα σήματα εισόδου/εξόδου. Το σωστό είναι η αρχαιότερη εντολή να έχει τον έλεγχο των σημάτων μέχρι που να μην τα χρειάζεται πλέον. Από τον τρόπο σχεδίασης των εντολών δεν υπάρχει περίπτωση 2 εντολές που επικαλύπτονται με τους κανόνες που ορίστηκαν στο προηγούμενο κεφάλαιο να χρειαστούν κοινούς πόρους. Ωστόσο στην περίπτωση που ο χρήστης δώσει κάποια εντολή σε μη έγκυρη χρονική στιγμή και κατά την διάρκεια εκτέλεσης μιας προηγούμενης είναι δυνατό να καταστραφούν κάποιες ουρές και να οδηγηθούν κάποια κρίσιμα σήματα (π.χ. το Swen) από περισσότερους από έναν QM. Για λόγους ασφάλειας λοιπόν, θεωρήσαμε χρήσιμο, η αρχαιότερη εντολή να κλειδώνει τους πόρους που χρειάζεται και να τους απελευθερώνει καθώς παύει να τους χρησιμοποιεί. Κάποια από τα σήματα δεν χρειάζονται λογική διαιτησίας και αυτά είναι:

- Το σήμα Ready είναι το λογικό ΚΑΙ (AND) μεταξύ των σημάτων Ready των δυο QM.

- Τα σήματα E_NE και Valid είναι το λογικό Ή (OR) μεταξύ των αντίστοιχων σημάτων των δυο QM.

Όσον αφορά τα διπλής κατεύθυνσης pin Data και Sdata δεν χρειάζονται λογική διαιτησίας καθώς -έμμεσα μέσω του ‘‘κλειδώματος’’- κάθε εντολή που εκτελείται σε κάποιον από του δύο QMs τα οδηγεί μόνο στους κύκλους που είναι απαραίτητο. Ωστόσο τα σήματα Saddr



Σχήμα 4.7: Συνενώνοντας 2 σύνολα ουρών

και Swen χρειάζονται διαιτησία γιατί πρέπει πάντα κάποιος να τα οδηγεί (ιδιαίτερα το σήμα που καθορίζει την εγγραφή στην στατική μνήμη Swen).

Το κύκλωμα διαιτησίας δέχεται σαν είσοδο τον κώδικα εντολής και τα σήματα finished και Ready από τους δύο QMs και επιστρέφει 3 σήματα: ένα σήμα ddrive που επιλέγει ποιός οδηγεί το σήμα Swen καθώς και 2 σήματα freeze που απενεργοποιούν τον αντίστοιχο QM έτσι ώστε να μην δέχεται ούτε να εκτελεί εντολές. Τα σήματα finished έρχονται από το μονοπάτι ελέγχου και όταν έχουν τεθεί ειδοποιούν ότι η εκτελούμενη εντολή του αντίστοιχου QM θα ολοκληρωθεί στον επόμενο κύκλο. Οι καταστάσεις της μηχανής πεπερασμένων καταστάσεων (FSM) φαίνονται στο Σχήμα 4.8. Σε αυτό το σχήμα μπορούμε να διακρίνουμε τί συμβαίνει κατά την διάρκεια εκτέλεσης μιας εντολής που αφορά τον QM_0 (η περίπτωση του QM_1 είναι συμμετρική). Όταν η FSM βρίσκεται σε κατάσταση απενεργοποίησης (idle state) και αν ο opcode[3] είναι σβηστός (αν έχουμε εντολή NOOP) τότε ανάλογα με την τιμή του bit opcode[4] που επιλέγεται ένας εκ των 2 QMs και γίνεται κύριος του σήματος Swen (κατάσταση m0 για τον QM_0). Αν συμβεί αυτό η εντολή που δίνεται από τον opcode[3..0] αρχίζει να εκτελείται. Ας υποθέσουμε ότι μια εντολή για τον

QM_0 έχει δοθεί (η άλλη περίπτωση είναι συμμετρική). Υπάρχουν διάφορες περιπτώσεις ανάλογα με τον τύπο της εντολής:

Α. Αν η εντολή είναι Enqueue, Dequeue, Top ή Init μετακινούμαστε στην κατάσταση edt_m0 όπου ο QM_0 γίνεται απόλυτος κύριος του σήματος Swen και καμμία νέα εντολή για οποιοδήποτε QM δεν μπορεί να δοθεί. Αυτή την κατάσταση μπορεί να διαδεχθεί μια άλλη κατάσταση όπου μια νέα εντολή να μπορεί να δοθεί προς οποιονδήποτε QM.

Β. Αν η εντολή είναι Retfree μετακινούμαστε στην κατάσταση r_r0 όπου ο QM_0 γίνεται κύριος του σήματος Swen ενώ μπορεί να δοθεί μια νέα εντολή προς οποιονδήποτε από τους 2 QMs. Η νέα αυτή εντολή μπορεί να είναι Enqueue, Dequeue ή Top.

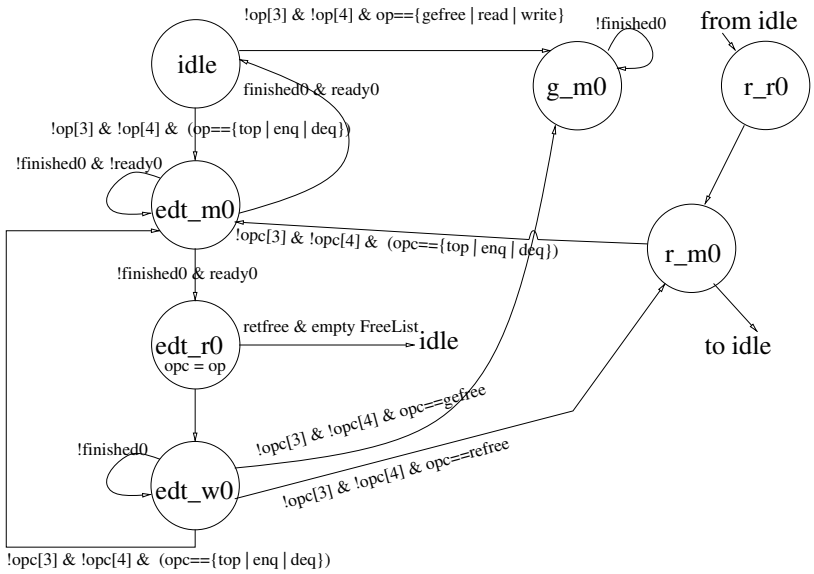
Γ. Αν η εντολή είναι Getfree μετακινούμαστε στην κατάσταση r_r0 όπου ο QM_0 γίνεται κύριος του σήματος Swen ενώ δεν μπορεί να δοθεί νέα εντολή.

Η ανάγκη για ομαδοποίηση των εντολών αυτού του είδους, πηγάζει από το γεγονός ότι 2 εκτελούμενες εντολές μπορεί να τερματίσουν στον ίδιο κύκλο και ο διαιτητής πρέπει να ξέρει ότι και οι δύο ολοκλήρωσαν την λειτουργία τους. Για παράδειγμα υπάρχει το σενάριο μια εντολή Retfree σε άδεια Free List που να ακολουθεί μια εντολή Enqueue. Και οι δύο ολοκληρώνουν την λειτουργία τους στον ίδιο κύκλο. Αν δεν υπήρχε αυτή η ομαδοποίηση δεν θα ήταν δυνατό να καταλάβει το κύκλωμα διαιτησίας ποιά από τις 2 εντολές τερμάτισε την λειτουργία της. Δηλαδή, το κύκλωμα διαιτησίας πρέπει να έχει κάποιο τρόπο να θυμάται ποιά εντολή ξεκίνησε πρώτη και ποιά δεύτερη για να πάρει την σωστή απόφαση. Έτσι μια εκτελούμενη εντολή (ουσιαστικά ο QM που την εκτελεί) μπορεί να είναι σε μια από τις εξής 3 καταστάσεις:

1. Κύριος των σημάτων εισόδου/εξόδου και καμμία νέα εντολή δεν μπορεί να ξεκινήσει.

2. Κύριος των pins εισόδου/εξόδου, δεν εκτελείται δεύτερη εντολή και μπορεί να ξεκινήσει να εκτελείται μια νέα εντολή.

3. Κύριος των pins εισόδου/εξόδου μια δεύτερη εντολή εκτελείται και δεν μπορεί να ξεκινήσει να εκτελείται μια νέα εντολή.



Σχήμα 4.8: Η Μηχανή Πεπερασμένων Καταστάσεων του κυκλώματος διαιτησίας

Η ΕΝΤΟΛΗ ΕΙΝΑΙ ENQUEUE, DEQUEUE, TOP ή INIT (ΠΕΡΙΠΤΩΣΗ Α)

Όταν η FSM βρίσκεται στον κύκλο edt_m0 (περίπτωση Α):

- Είτε finished=1 και Ready=1 που σημαίνει ότι ο QM_0 θα ολοκληρώσει την εκτέλεση της τρέχουσας εντολής στον επόμενο κύκλο και άρα θα μπορεί να δεχτεί μια καινούρια στον ίδιο κύκλο οδηγούμενος στην κατάσταση idle,
- Είτε finished=0 και Ready=1 που σημαίνει ότι ο QM_0 θα συνεχίσει την εκτέλεση της τρέχουσας εντολής στον επόμενο κύκλο και θα μπορεί να δεχτεί μια καινούρια σε εκείνο τον κύκλο οδηγούμενος στην κατάσταση edt_r0.

Στην κατάσταση edt_r0 μια νέα εντολή μπορεί να εισαχθεί. Αν η νέα αυτή εντολή είναι Retfree και τύχει η Free List να είναι άδεια, τότε μετακινούμαστε σε κατάσταση idle αλλά ο QM_0 εξακολουθεί να είναι κύριος των σημάτων για ένα ακόμη κύκλο. Σε κάθε άλλη περίπτωση η FSM μετακινείται στην κατάσταση edt_w0 από όπου (η FSM) ανάλογα με την εντολή που είχε δοθεί στην κατάσταση edt_r0 μετακινείται στην σωστή επόμενη κατάσταση:

Enqueue,Dequeue,Top: Η επόμενη κατάσταση θα είναι η edt_m0.

Retfree and non-empty free list: η επόμενη κατάσταση θα είναι η r_m0. Κατά την διάρκεια αυτής της κατάστασης ο QM_0 είναι κύριος των pins και δεν μπορεί να δοθεί καμμία νέα εντολή.

Getfree and non-empty free list: Η επόμενη κατάσταση θα είναι η g_m0 και δεν μπορεί να δοθεί προς εκτέλεση καμμία νέα εντολή.

Η ΕΝΤΟΛΗ ΕΙΝΑΙ RETFREE (ΠΕΡΙΠΤΩΣΗ Β)

Όταν η FSM βρίσκεται στον κύκλο r_r0 (περίπτωση Β), μια νέα εντολή μπορεί να δοθεί προς εκτέλεση καθώς ο QM είναι κύριος των σημάτων. Η νέα αυτή εντολή δεν μπορεί να είναι Retfree γιατί οι καταχωρητές Free List Head και Free List Tail δεν έχουν ανανεωθεί από την πρώτη Retfree ακόμη. Για τον ίδιο λόγο δεν μπορεί να είναι εντολή Getfree. Η επόμενη κατάσταση είναι η r_m0, όπου ο QM_0 είναι κύριος των σημάτων και καμμία νέα εντολή δεν μπορεί να δοθεί στην διάρκεια αυτού του κύκλου. Λαμβάνοντας υπόψιν εάν και ποιά εντολή δόθηκε στον προηγούμενο κύκλο η FSM μετακινείται σε μια από τις καταστάσεις edt_m0, edt_m1 ή idle.

Η ΕΝΤΟΛΗ ΕΙΝΑΙ GETFREE, READ ή WRITE (ΠΕΡΙΠΤΩΣΗ Γ)

Όταν η FSM βρίσκεται στον κύκλο g_m0 (περίπτωση Γ), ο QM_0 είναι ο κύριος των σημάτων και δεν μπορεί να δοθεί μια νέα εντολή. Αυτό συμβαίνει λόγω του κύκλου turnaround. Η επόμενη κατάσταση που μετακινείται η FSM είναι η idle όπου μετακινούμαστε όταν το σήμα finished (του QM_0) γίνει 1.

4.1.6 ΚΟΣΤΟΣ ΚΑΙ ΑΠΟΔΟΣΗ

Ο QM του MuQPro I υλοποιήθηκε σε μια συσκευή FPGA FLEX10K50 της ALTERA σε Γλώσσα Ορισμού Υλικού AHDL (Altera Hardware Definition Language) και απαιτεί 2198 λογικά κύτταρα αυτής της συσκευής. Ο βαθμός χρησιμοποίησης της λογικής (logic utilization) που υπάρχει μέσα στην συσκευή είναι 76% ενώ ο βαθμός χρησιμοποίησης των pins είναι 31%. Η συχνότητα του ρολογιού που επιτύχαμε είναι 30MHz σε ανάλυση χειρίστης περίπτωσης όπως μετρήθηκε από τον Αναλυτή Χρονισμού της ALTERA. Ο μέσος βαθμός φόρτωσης της εισόδου (fan-in) είναι 3.11/4 που σημαίνει ότι κάθε λογικό κύτταρο της συγκεκριμένης συσκευής έχει κατά μέσο όρο 4 εισόδους και κατά μέσο όρο 3.11 από αυτές

χρησιμοποιούνται. Για να επιτύχουμε τις απαιτήσεις χρόνου (30 MHz) το πρόγραμμα λογικής σύνθεσης (logic synthesizer) πρόσθεσε 30% επιπλέον λογική (874 λογικά κύτταρα σε απόλυτους αριθμούς). Ο μικρός βαθμός χρησιμοποίησης των pins προέρχεται από το γεγονός ότι δεν υπήρχε συσκευασία (packaging) με λιγότερα pins για τον συγκεκριμένο τύπο συσκευής. Ο QM μπορεί να υλοποιηθεί σε μια συσκευή FLEX10K40 με κύκλο ρολογιού 15 MHz και βαθμό χρησιμοποίησης της λογικής 95%.

4.1.7 ΕΛΕΓΧΟΣ (TESTING) ΤΟΥ ΔΙΑΧΕΙΡΙΣΤΗ ΟΥΡΩΝ

Το περιβάλλον σχεδίασης ψηφιακών συσκευών της ALTERA, MAX+PLUSII μπορεί να παράγει κώδικα Verilog επιπέδου πυλών. Για να προσομοιώσουμε την εξωτερική στατική μνήμη γράψαμε ένα απλό μοντέλο στην γλώσσα Verilog. Όποτε δίνεται μια αίτηση προς το μοντέλο και μετά από σταθερό χρόνο (παράμετρος του μοντέλου) αυτό απαντά. Το επόμενο βήμα για την σωστή προσομοίωση ήταν να μπορέσουμε να παράγουμε σωστά διανύσματα (vectors) και να εξακριβώσουμε ότι η έξοδος από την προσομοίωση (δηλαδή τα περιεχόμενα της στατικής μνήμης) είναι τα σωστά. Για την παραγωγή αυτών των διανυσμάτων γράψαμε κώδικα στην γλώσσα προγραμματισμού C που κάνει τα ακόλουθα:

- Διαχείριση Ουρών σε λογισμικό
- Τυχαία επιλογή μιας εντολής για εκτέλεση μέσω της συνάρτησης random()
- Δημιουργεί μια ενδιάμεση δομή ουρών για να κρατά στατιστικά στοιχεία για το πόσες και ποιές διευθύνσεις σε ελεύθερους χώρους μνήμης είναι διαθέσιμες. Κάθε φορά που επιλέγεται (τυχαία) μια εντολή Getfree ή Dequeue για εκτέλεση, η επιστρεφόμενη διεύθυνση αποθηκεύεται σε αυτή την δομή και όποτε επιλέγεται μια εντολή Retfree ή Enqueue για εκτέλεση, εξάγεται μια διεύθυνση από αυτή την δομή για να χρησιμοποιηθεί σαν όρισμα.

Σε κάθε επανάληψη επιλέγουμε τυχαία μια εντολή και ελέγχουμε αν είναι σωστή. Μια εντολή Enqueue είναι σωστή αν η ενδιάμεση δομή ουρών είναι μη άδεια (οπότε και κάνουμε Enqueue ένα στοιχείο της ενδιάμεσης ουράς). Μια εντολή Dequeue μπορεί να εκτελεστεί μόνο αν η τυχαία επιλεγόμενη ουρά είναι μη άδεια και αυτό μπορούμε να το εξακριβώσουμε μέσω μετρητών που διατηρούμε για κάθε ουρά. Παρόμοια, μια εντολή Getfree ελέγχεται αν μπορεί να εκτελεστεί μέσω ελέγχου της Free List να είναι μη άδεια ενώ η Retfree μπορεί να εκτελεστεί μόνο αν η Free List είναι μη γεμάτη. Αφού επιλέξουμε μια

έγκυρη εντολή με βάση τους παραπάνω κανόνες, χρησιμοποιούμε τις ρουτίνες σε λογισμικό για να την εκτελέσουμε και γράφουμε κι ένα διάνυσμα με συντακτικό Verilog σε ένα εξωτερικό αρχείο. Αφού επαναλάβουμε αυτή την διαδικασία όσες φορές χρειαστεί για να δημιουργήσουμε τον προκαθορισμένο αριθμό εντολών καταλήγουμε να έχουμε 3 αρχεία:

- Ένα αρχείο που περιέχει τα περιεχόμενα της μνήμης (memory dump) μετά την εκτέλεση της Διαχείρισης Μνήμης σε λογισμικό. Από αυτό το αρχείο και μέσω ενός προγράμματος σε C παράγεται ένα νέο αρχείο, σε μια πιο αναγνώσιμη μορφή, που περιέχει τα στοιχεία κάθε ουράς.

- Ένα αρχείο που περιέχει τα διανύσματα Verilog

Το επόμενο βήμα είναι να εκτελέσουμε τον κώδικα Verilog επιπέδου πυλών του Διαχειριστή Ουρών μαζί με το μοντέλο στατικής μνήμης, στον προσομοιωτή της CADENCE με είσοδο το αρχείο με τα διανύσματα που προέκυψε από την προηγούμενη διαδικασία. Μετά το τέλος της εκτέλεσης γράφονται σε ένα αρχείο τα περιεχόμενα της στατικής μνήμης και χρησιμοποιώντας ένα μικρό πρόγραμμα που έχουμε αναπτύξει μετατρέπουμε τα περιεχόμενα της μνήμης σε στοιχεία που έχει η κάθε ουρά και τα εξάγουμε σε ένα άλλο αρχείο. Συγκρίνοντας τα περιεχόμενα των αρχείων με τα στοιχεία που έχει η κάθε ουρά στην περίπτωση του λογισμικού και στην περίπτωση της προσομοίωσης βρίσκουμε διαφορές που μας βοηθούν σημαντικά στον έλεγχο του κώδικα Verilog.

Επαναλάβαμε αυτή διαδικασία για αρκετές εκατοντάδες χιλιάδες εντολών και διορθώσαμε πολλά λογικά λάθη που υπήρχαν. Το αρχείο διανυσμάτων είναι επίσης ένα καλό παράδειγμα του τρόπου με τον οποίο δίνει κανείς εντολές στο Διαχειριστή Ουρών του MuqPRo I.

4.2 Η ‘‘ΠΛΗΡΩΣ ΕΠΕΚΤΑΣΙΜΗ’’ ΥΛΟΠΟΙΗΣΗ

Η υλοποίηση του QM του MuQPro I έγινε με βάση τις συγκεκριμένες ανάγκες του συστήματος. Η ‘‘πλήρως επεκτάσιμη’’ υλοποίηση σκοπεύει στο να χρησιμοποιεί στον μέγιστο δυνατό βαθμό την εξωτερική στατική μνήμη. Αυτή η σχεδίαση έχει μικρές διαφορές στο μονοπάτι δεδομένων και περισσότερες στο μονοπάτι ελέγχου σε σύγκριση με την υλοποίηση του MuQPro I. Πιο συγκεκριμένα οι διαφορές στο μονοπάτι δεδομένων είναι οι εξής:

- Ένας επιπλέον καταχωρητής Βάσης και ένας Μεγέθους που διατηρεί την διεύθυνση που αρχίζει η μνήμη των Empty Bits και το μέγεθός της.
- Ένας επιπλέον πολυπλέκτης για να επιλέγει από τα 16 bits που είναι το μέγεθος λέξης της εξωτερικής στατικής μνήμης το Empty Bit της ζητούμενης ουράς.

Η επιπλέον καθυστέρηση ενός κύκλου της εντολής Enqueue οφείλεται στο γεγονός ότι δεν προλαβαίνουμε να διαβάσουμε μια λέξη 16-bits και να διαλέξουμε ένα από αυτά τα bits σε ένα κύκλο σε τεχνολογία FPGA και δεδομένης της σχετικά υψηλής ταχύτητας (30 MHz). Όλες οι υπόλοιπες εντολές χρειάζονται τον ίδιο αριθμό κύκλων με την υλοποίηση του QM για το MuQPro I. Διάγραμμα του μονοπατιού δεδομένων αυτής της αρχιτεκτονικής βρίσκεται στο Παράρτημα Β.1 ενώ διαγράμματα του μονοπατιού ελέγχου στο Παράρτημα Β.2.

4.3 Η ‘‘ΠΛΗΡΩΣ ΠΑΡΑΛΛΗΛΗ’’ ΥΛΟΠΟΙΗΣΗ

Αυτή η υλοποίηση έγινε για να δούμε τις εναλλακτικές λύσεις στην σχεδίαση και να εξερευνήσουμε τον παραλληλισμό που υπάρχει στην διαχείριση ουρών. Κάθε εντολή ξεκινά με την ενεργοποίηση ενός διαφορετικού pin και υπάρχουν 2 μνήμες μικρού πλάτους (στενές) μέσα στο ολοκληρωμένο (on-chip memories):

1. Μία για τον Head/Tail table, και
2. Μία για την Pointer memory.

Τα Empty Bits είναι ενσωματωμένα στον Head/Tail table.

Μια λέξη της πρώτης μνήμης περιέχει ένα δείκτη στην κεφαλή και το τέλος μιας ουράς καθώς και το bit που δείχνει την κατάσταση αυτής της ουράς (Empty Bit). Το πλάτος αυτής της μνήμης είναι 11 bits (5 για την κεφαλή, 5 για το τέλος και 1 για το Empty Bit). Προφανώς η αρχιτεκτονική αυτή έχει υψηλό κόστος και χαμηλή επεκτασιμότητα γιατί χρειάζεται μνήμες μέσα στο ολοκληρωμένο με μεγάλο πλάτος που είναι μη εφικτό με σημερινή τεχνολογία FPGA. Το σύνολο εντολών αυτής της υλοποίησης είναι Enqueue, Dequeue, GetFree, RetFree and Init. Οι εντολές Enqueue, GetFree, και RetFree εκτελούνται σε 2 κύκλους χωρίς επικάλυψη ή pipelining. Το μονοπάτι δεδομένων και ελέγχου παρουσιάζεται στο Παράρτημα C.

ΚΕΦΑΛΑΙΟ 5

ΥΛΟΠΟΙΗΣΗ ΔΙΑΧΕΙΡΙΣΗΣ ΟΥΡΩΝ ΣΕ ΛΟΓΙΣΜΙΚΟ

Ο ταχύς ρυθμός ανάπτυξης γρήγορων μP χαμηλού κόστους έχει οδηγήσει σε υποσυστήματα που συμπεριλαμβάνουν μικροεπεξεργαστές (embedded systems). Αυτά τα συστήματα είναι ελκυστικά λόγω της επεκτασιμότητάς τους ως προς την λειτουργικότητα και την ταχύτητα η οποία αυξάνεται σημαντικά. Για αυτό τον λόγο αναπτύσσουμε λογισμικό για διαχείριση ουρών και ελέγχουμε την απόδοσή του σε ένα δημοφιλή μικροεπεξεργαστή μειωμένου συνόλου εντολών όπως ο i960 της INTEL. Πιο συγκεκριμένα, υλοποιήσαμε τις εντολές EnQ, DeQ, GetFree και RetFree και μετρήσαμε τον χρόνο που χρειάζεται ο i960 για να εκτελέσει εντολές EnQ/DeQ μετακινώντας μόνο τους δείκτες προς τα δεδομένα με δύο ειδών εισόδους: (α) ακολουθιακή, δηλαδή εντολές σε συνεχόμενους αριθμούς ουρών και θέσεις μνήμης και (β) τυχαία είσοδο, δηλαδή εντολές σε τυχαίους αριθμούς ουρών και θέσεις μνήμης. Δεν μετρήσαμε τις εντολές GetFree και GetFree γιατί απαιτούν μόνο μία πρόσβαση στην δομή Pointer Memory και μία πρόσβαση σε συγκεκριμένους καταχωρητές (Free List Head, Tail, Empty Bit) και συνεπώς μπορούν να εκτελεστούν γρήγορα συγκριτικά με τις EnQ, DeQ. Σαν είσοδο στην εντολή EnQ ορίζουμε τον αριθμό ουράς και ένα δείκτη προς τα δεδομένα ενώ η εντολή DeQ χρειάζεται σαν όρισμα μόνο ένα αριθμό ουράς. Το επόμενο βήμα είναι να ενσωματώσουμε αυτό τον κώδικα -που ουσιαστικά εκτελεί την διαχείριση μνήμης- σε ένα μεγαλύτερο κομμάτι κώδικα που εκτελεί τη λειτουργία SAR και να εξακριβώσουμε την διασυνδεσιμότητά του καθώς και τον χρόνο που παίρνει σαν ποσοστό επί του χρόνου εκτέλεσης ολόκληρης της λειτουργίας SAR. Επιπρόσθετα, συγκρίνουμε τις διάφορες υλοποιήσεις υλικού και λογισμικού καταλήγοντας σε λύσεις

με υψηλό δείκτη απόδοσης ως προς το κόστος, δεδομένων των απαιτήσεων.

5.1 ΤΟ ΠΕΡΙΒΑΛΛΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

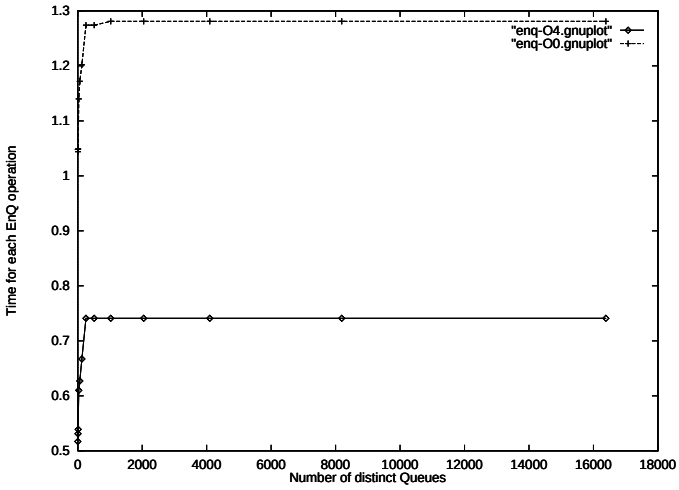
Χρησιμοποιήσαμε μια πλακέτα εκτίμησης (evaluation board) κατασκευασμένη από την CYCLONE με ενσωματωμένο επεξεργαστή INTEL i960CA στα 40 MHz, 2 MB μνήμης και PCI interface για σύνδεση με PC ([25]). Ο κώδικας αναπτύχθηκε εξ' ολοκλήρου στη γλώσσα προγραμματισμού C και μεταγλωττίστηκε με τον μεταγλωττιστή της INTEL gcc960 που τρέχει σε PC. Η μεταγλώττιση έγινε με 2 τρόπους: χωρίς καθόλου βελτιστοποίηση και με πλήρη βελτιστοποίηση του κώδικα. Για την μέτρηση των χρόνων χρησιμοποιήσαμε την οικογένεια ρουτινών bentime() της CYCLONE που επιτρέπουν ακρίβεια ενός μικροδευτερόλεπτου. Η δομή Pointer Memory, όπου κρατούνται οι δείκτες κάθε μονάδας αποθήκευσης (buffer) είναι ένας πίνακας πλάτους 32 bits (όσο και το μέγεθος της λέξης του μικροεπεξεργαστή) και ύψους 384K. Συνολικά η μνήμη που καταλαμβάνει αυτή η δομή είναι 1.5 MB από τα 2 MB υπάρχουσας μνήμης.

5.2 Η ΛΕΙΤΟΥΡΓΙΑ ENQ ΜΕ ΚΑΝΟΝΙΚΗ ΕΙΣΟΔΟ

Στο πρώτο μας πείραμα χρησιμοποιούμε τον ελάχιστο κώδικα που χρειάζεται για να υλοποιηθεί η λειτουργία EnQ. Για 10^8 επαναλήψεις εισάγαμε ακολουθιακές διευθύνσεις μνήμης σε ακολουθιακούς αριθμούς ουρών. Έτσι στην πρώτη επανάληψη εισάγαμε την διεύθυνση 0 στην ουρά 0, στην δεύτερη την διεύθυνση 1 στην ουρά 1, κ.ο.κ. Όταν εξαντλούνται οι ουρές ή οι διευθύνσεις ξαναξεκινάμε από την αρχή. Την είσοδο αυτού του είδους αποκαλούμε *κανονική είσοδο.* Πρέπει να επισημάνουμε ότι ο κώδικας είναι αρκετά μικρός σε μέγεθος για να χωράει στην κρυφή μνήμη του μικροεπεξεργαστή. Έτσι οι μετρήσεις που παρουσιάζουμε αποτελούν ένα κάτω φράγμα ως προς τον χρόνο εκτέλεσης αυτής της λειτουργίας σε μια πραγματική εφαρμογή στο συγκεκριμένο μP. Ο αριθμός των εντολών γλώσσας μηχανής ανάλογα με την επιλογή βελτιστοποίησης του μεταγλωττιστή παρουσιάζεται στον Πίνακα 5.1. Στο σχήμα 5.1 παρουσιάζεται η επίδραση της αύξησης του αριθμού ουρών N_{queues} στο χρόνο εκτέλεσης. Ξεκάθαρα υπάρχει ένα όριο (για $N_{queues} = 512$) πάνω από το οποίο ο χρόνος εκτέλεσης παραμένει σταθερός. Αυτό συμβαίνει γιατί η δομή Head/Tail table γίνεται επαρκώς μεγάλη, δεν χωράει ολόκληρη μέσα στην κρυφή μνήμη και συνεπώς ο μP αναγκάζεται να προσπελαύνει συχνά την κύρια μνήμη του.

Optimization level	Number of instructions	Loads	Stores
0	47-50	7-8	4-5
4	20-22	2-3	2-3

Πίνακας 5.1: Αριθμός εντολών γλώσσας μηχανής για την εντολή EnQ

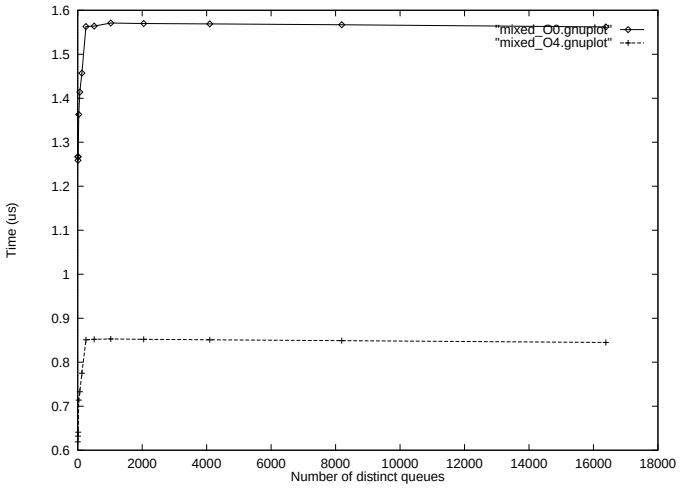


Σχήμα 5.1: Σειριακές λειτουργίες Enqueue

5.3 ΜΙΚΤΕΣ ΛΕΙΤΟΥΡΓΙΕΣ ENQ/DEQ ΜΕ ΚΑΝΟΝΙΚΗ ΕΙΣΟΔΟ

Η λειτουργία DeQ είναι δύσκολο να μετρηθεί, γιατί οι ουρές μετά από μερικές επαναλήψεις αδειάζουν με αποτέλεσμα να εκτελούμε DeQ σε άδειες ουρές. Η εκτέλεση της εντολής DeQ σε άδεια ουρά έχει σαν αποτέλεσμα την καθυστέρηση μίας μόνο ανάγνωσης στην μνήμη. Από την άλλη πλευρά αν αφήσουμε το πρόγραμμα να εκτελεστεί για μικρό αριθμό επαναλήψεων (μέχρι να αδειάσουν οι ουρές) τότε η ακρίβεια της μέτρησης είναι μικρή. Σκεφτήκαμε την πιθανότητα να χρησιμοποιήσουμε DeQ σε ουρές που να μην τις αφήνουμε να αδειάζουν (με το να μην γράφουμε το Empty Bit όταν αδειάζει μια ουρά) αλλά τότε έχουμε το πρόβλημα ότι προσπελαύνουμε πάντα το ίδιο στοιχείο. Ένας έμμεσος τρόπος μέτρησης προκύπτει με το να χρησιμοποιήσουμε μικτές EnQ/DeQ λειτουργίες και να ξαναγεμίζουμε τις ουρές όποτε αυτές αδειάζουν. Έτσι μετράμε τον μέσο χρόνο εκτέλεσης

μιας μικτής εντολής (EnQ ή DeQ) και γνωρίζοντας τον μέσο χρόνο εκτέλεσης της EnQ μπορούμε να προσεγγίσουμε με αρκετή ακρίβεια τον χρόνο εκτέλεσης της DeQ (Σχήμα 5.2). Χρησιμοποιήσαμε κανονική είσοδο και για αυτό το πείραμα γιατί αν χρησιμοποιούσαμε τυχαία είσοδο δεν θα άδειαζαν όλες οι ουρές με την σειρά γεμίσματος.



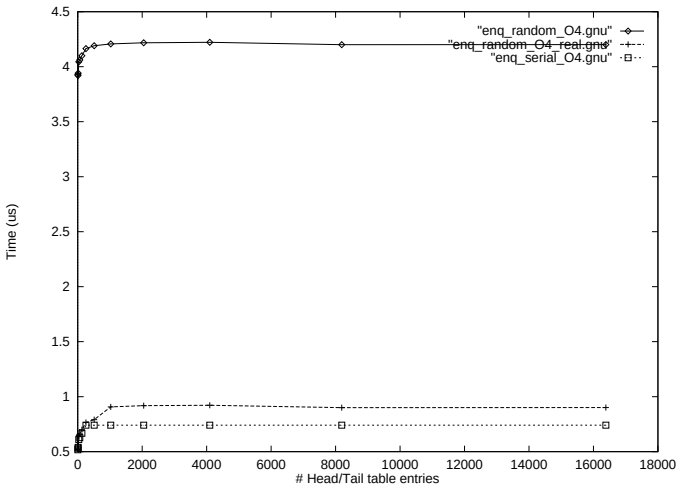
Σχήμα 5.2: Μικτές λειτουργίες Enqueue και Dequeue

Μετά το σημείο κορεσμού βλέπουμε μια μικτή εντολή να έχει χρόνο εκτέλεσης 0.85μs και επειδή ο χρόνος εκτέλεσης μιας τυπικής λειτουργίας EnQ είναι 0.75μs και εκτελούμε τον ίδιο αριθμό από λειτουργίες EnQ/DeQ μια τυπική λειτουργία DeQ στο σημείο κορεσμού έχει χρόνο εκτέλεσης 0.95μs.

5.4 Η ΛΕΙΤΟΥΡΓΙΑ ENQ ΜΕ ΤΥΧΑΙΑ ΕΙΣΟΔΟ

Σε πραγματικές εφαρμογές η είσοδος στην λειτουργία EnQ δεν είναι κανονική αλλά τυχαία. Για αυτό τον σκοπό μετρήσαμε τον χρόνο εκτέλεσης της EnQ με τυχαία είσοδο όπως φαίνεται στην πάνω καμπύλη του Σχήματος 5.3. Σε αυτά τα πειράματα, γίνεται κλήση της ρουτίνας rand() δύο φορές για να επιλεχτεί μια τυχαία ουρά και μια τυχαία διεύθυνση. Σύμφωνα με ανεξάρτητες μετρήσεις που κάναμε, η κλήση της ρουτίνας rand() παίρνει χρόνο 1.7μs και έτσι ο καθαρός χρόνος εκτέλεσης της εντολής EnQ φαίνεται στη μεσαία καμπύλη του Σχήματος 5.3. Αυτό που περιμένουμε είναι η λειτουργία EnQ με τυχαία είσοδο

να έχει λίγο υψηλότερο χρόνο εκτέλεσης σχετικά με την λειτουργία EnQ με κανονική είσοδο (κάτω καμπύλη Σχήματος 5.3).



Σχήμα 5.3: Τυχαίες λειτουργίες Enqueue με βελτιστοποίηση 4ου επιπέδου

5.5 ΤΟ ΜΟΝΤΕΛΟ SAR

Η βασική λειτουργία στα συστήματα ATM όπως είδαμε στο Κεφάλαιο 2, είναι να μετακινούμε κύτταρα από ένα σύνολο ουρών (τις ουρές εισόδου) προς ένα άλλο σύνολο ουρών (τις ουρές εξόδου). Η λειτουργία SAR ακολουθεί αυτό το μοντέλο προσθέτοντας/ αφαιρώντας τις κατάλληλες επικεφαλίδες (headers και trailer) από/σε μεταβλητού μεγέθους πακέτα καθώς αυτά μετακινούνται από τις ουρές πακέτων προς τις ουρές κυττάρων και το αντίστροφο.

Υλοποιήσαμε την μια κατεύθυνση (Segmentation) γιατί η άλλη κατεύθυνση είναι περίπου ισοδύναμη: στον Κατακερματισμό, σε κάθε επανάληψη, ένα πακέτο εξάγεται από μια ουρά πακέτων προστίθεται το padding και ο trailer. Η προκύπτουσα PDU διασπάται σε κύτταρα μεγέθους 48 bytes τα οποία εισάγονται στην κατάλληλη ουρά κυττάρων μαζί με την επικεφαλίδα τους. Στην Επανασύνδεση, σε κάθε επανάληψη, ένα κύτταρο εξέρχεται από την κατάλληλη ουρά κυττάρων και εισάγεται στην κατάλληλη ουρά πακέτων αφαιρώντας συγχρόνως την επικεφαλίδα (header). Αν το κύτταρο αυτό είναι το τελευταίο του πακέτου AAL-5 τότε πρέπει βρεθεί το μέγεθος του πακέτου από το κατάλληλο πεδίο του

trailer, να αφαιρεθεί το padding και ο trailer αφήνοντας το πακέτο έτοιμο για μεταφορά στα ανώτερα στρώματα πρωτοκόλλων του δικτύου. Οι διαφορές μεταξύ των δύο συμμετρικών λειτουργιών είναι οι εξής:

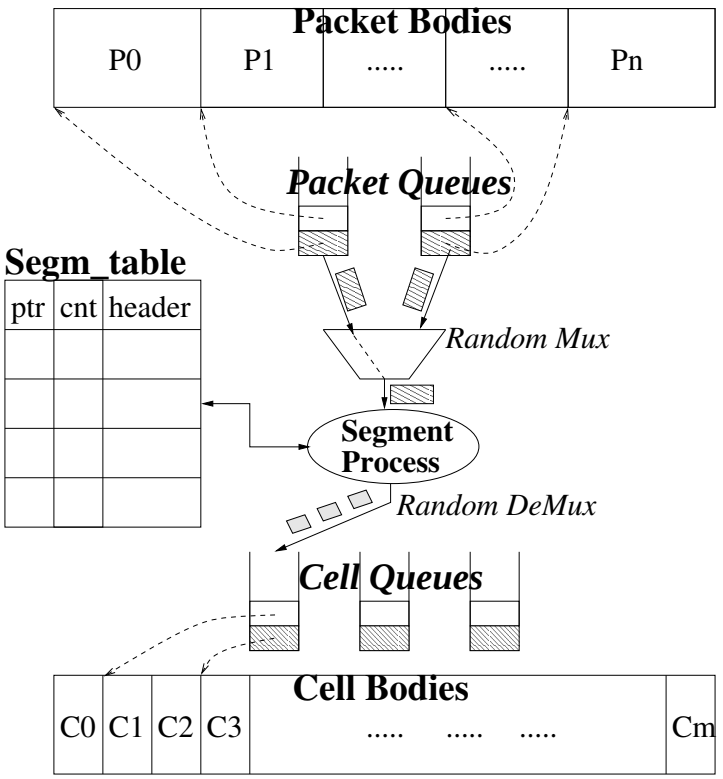
- Σε κάθε επανάληψη στον Κατακερματισμό τα κύτταρα του πακέτου μετακινούνται στην ίδια ουρά κυττάρων, ενώ στην Επανασύνδεση κάθε κύτταρο που εξέρχεται από την ουρά κυττάρων μπορεί να χρειάζεται να μετακινηθεί σε διαφορετική ουρά πακέτων. Αυτό συμβαίνει μόνο όταν το μέγεθος του πακέτου είναι μεγαλύτερο από ένα κύτταρο.

- Στον Κατακερματισμό πρέπει να υπολογισθεί το κατάλληλο padding ανάλογα με το μέγεθος του πακέτου, ενώ στην Επανασύνδεση το μέγεθος του επανασυνδεδεμένου πακέτου μπορεί να βρεθεί στον trailer του πακέτου και η αφαίρεση του padding μπορεί να γίνει απλώς με το να μην προσπελασθεί κατά την ανάγνωση του πακέτου.

- Στον Κατακερματισμό πρέπει να προστεθεί ένας trailer που είναι πιο χρονοβόρα διαδικασία σε σχέση με το να αφαιρεθεί ένας trailer.

- Όμοια η προσθήκη της επικεφαλίδας κάθε κυττάρου (Κατακερματισμός) είναι πιο χρονοβόρα διαδικασία από την αποκόλληση αυτής (Επανασύνδεση).

Γράψαμε κώδικα C που εκτελεί Κατακερματισμό ενός πακέτου μεταβλητού μεγέθους (1-64000 bytes) σύμφωνα με το AAL-5. Αρχικά καταλαμβάνουμε δύο συνεχή και ισομεγέθη κομμάτια μνήμης: ένα για την αποθήκευση των πακέτων (Packet Bodies) και ένα για την αποθήκευση των κυττάρων (Cell Bodies). Το κομμάτι μνήμης Cell Bodies χωρίζεται σε κομμάτια των 53 bytes, ενώ το κομμάτι μνήμης Packet Bodies χωρίζεται σε κομμάτια ίσα με το μέγιστο μέγεθος του πακέτου που μπορεί να έχουμε κατά την εκτέλεση του κώδικα. Έτσι κάθε κύτταρο (όπως και κάθε πακέτο) μπορεί να βρεθεί μέσα στην μνήμη Cell Bodies γνωρίζοντας την διεύθυνση βάσης που ξεκινά αυτή η μνήμη και ένα δείκτη που να αναφέρεται σε αυτό. Για παράδειγμα, για να βρούμε την διεύθυνση του κυττάρου i στην μνήμη Cell Bodies που ξεκινά από την διεύθυνση X χρειάζεται να κάνουμε τον υπολογισμό X+53∗i. Εκτός αυτών των δύο κομματιών μνημών χρησιμοποιούμε και άλλα για τις δομές του Διαχειριστή Ουρών και χρησιμοποιούμε ξεχωριστές τέτοιες δομές για πακέτα και κύτταρα. Όλες οι μνήμες φαίνονται στο Σχήμα 5.4. Επίσης χρησιμοποιούμε μια ρουτίνα παραγωγής τυχαίων αριθμών άλλη από αυτή που δίνεται από την CYCLONE, την random() η οποία είναι γρηγορότερη καθώς και την Ρουτίνα Επεξεργασίας των πακέτων.



Σχήμα 5.4: Το μοντέλο SAR

5.5.1 Η ΡΟΥΤΙΝΑ ΕΠΕΞΕΡΓΑΣΙΑΣ ΤΩΝ ΠΑΚΕΤΩΝ

Η ρουτίνα Επεξεργασίας των πακέτων εκτελεί όλες τις απαραίτητες λειτουργίες πάνω στο πακέτο έτσι ώστε να μετατραπεί σε μια ακολουθία από έγκυρα κύτταρα ATM. Σαν είσοδο δέχεται τον συνολικό αριθμό από κύτταρα που θα επεξεργαστεί. Πιο λεπτομερειακά η ακόλουθη επαναληπτική διαδικασία εκτελείται όταν καλείται αυτή η ρουτίνα:

1. Μια ουρά πακέτων επιλέγεται τυχαία

2. Μια διεύθυνση σε κάποιο πακέτο στην μνήμη Packet Bodies εξέρχεται από αυτή την ουρά.

3. Το μέγεθος του πακέτου, σε bytes, υπολογίζεται τυχαία με πάνω και κάτω όρια τις παραμέτρους MIN_PACKET_SIZE και PACKET_SIZE (μέγεθος πακέτου 0 δεν είναι αποδεκτό).

4. Υπολογίζεται το απαραίτητο padding και το συνολικό μέγεθος του πακέτου (μαζί με τον trailer), σε αριθμό κυττάρων, με βάση το αρχικό μέγεθος αυτού (χωρίς τον trailer).

5. Ένας trailer παράγεται τυχαία και αντιγράφεται στο τέλος του πακέτου.

6. Μια επικεφαλίδα επίσης παράγεται τυχαία για χρήση όταν το πακέτο μετατραπεί σε κύτταρα των 48 bytes.

7. Μια ουρά κυττάρων επιλέγεται τυχαία.

8. Διαβάζουμε το πακέτο σε κομμάτια των 48 bytes και εκτελούμε τα ακόλουθα:

   (a) Καλείται η ρουτίνα GetFree() για να επιστρέψει μια διεύθυνση σε ένα ελεύθερο χώρο αποθήκευσης ενός κυττάρου. εξέρχεται με την κλήση της ρουτίνας GetFree().

   (b) Η διεύθυνση αυτή εισέρχεται στην ουρά κυττάρων που επιλέχτηκε τυχαία προηγουμένως.

   (c) Το κύτταρο που διαβάζουμε αντιγράφεται στην διεύθυνση που εισάγαμε προηγουμένως στην ουρά κυττάρων.

   (d) Στο τέλος του κυττάρου προστίθεται η επικεφαλίδα του κυττάρου η οποία είχε προϋπολογιστεί.

9. Η διεύθυνση στο τρέχον πακέτο επιστρέφεται μέσω μιας λειτουργίας EnQ στην ουρά πακέτων από την οποία εξήλθε.

10. Αυξάνεται ένας τοπικός μετρητής για να υποδυκνείει τον συνολικό αριθμό κυττάρων του πακέτου που έχουν επεξεργαστεί μέχρι τώρα. Έτσι σηματοδοτείται η λήξη της επεξεργασίας του συγκεκριμένου πακέτου.

Αυτή ήταν η περιγραφή μιας μόνο επανάληψης της επαναληπτικής διαδικασίας, η οποία τερματίζει όταν ο συνολικός αριθμός από κύτταρα που επεξεργάστηκαν γίνει ίσος με τον ζητούμενο αριθμό από κύτταρα που έπρεπε να επεξεργαστούν. Όπως εξακριβώνει κανείς δεν έχουμε συμπεριλάβει καθόλου υπολογισμούς CRC γιατί απαιτούνται επιπρόσθετες προσπελάσεις στην μνήμη και επιπλέον υπολογισμοί. Η προσθήκη αυτής της λειτουργίας θα αύξανε περαιτέρω τον χρόνο επεξεργασίας ενός πακέτου. Είναι εύκολο να ενσωματωθεί αυτή η λειτουργία στον κώδικα.

5.5.2 Η ΕΠΑΝΑΛΗΠΤΙΚΗ ΔΙΑΔΙΚΑΣΙΑ

Καλούμε την Ρουτίνα Επεξεργασίας των πακέτων τόσες φορές όσες υποδεικνύονται από την παράμετρο ITERATIONS, ώστε να αυξήσουμε την ακρίβεια. Σε κάθε επανάληψη γίνεται επεξεργασία όλων των πακέτων που υπάρχουν στην μνήμη και όλα τα κύτταρα εισάγονται σε ουρές. Συνεπώς, κάθε φορά πρέπει να αδειάζουμε τις ουρές κυττάρων, πριν ξανακαλέσουμε την Ρουτίνα Επεξεργασίας των πακέτων. Να σημειωθεί ότι κάθε φορά που εξάγουμε ένα πακέτο από μια ουρά και αφού το επεξεργαστούμε, το επανατοποθετούμε πίσω στην ίδια ουρά έτσι ώστε να έχουμε συνεχώς πακέτα προς επεξεργασία. Φυσικά, πριν την πρώτη κλήση της Ρουτίνας Επεξεργασίας των πακέτων, πρέπει να αρχικοποιήσουμε τις ουρές κυττάρων, ώστε να είναι άδειες, και τις ουρές πακέτων, ώστε να είναι γεμάτες.

Το πλήθος των συνολικών κυττάρων που επεξεργάζονται είναι τελικά ITERATIONS∗PACKET_SIZE_IN_CELLS. Διαιρώντας τον συνολικό χρόνο εκτέλεσης με τον συνολικό αριθμό κυττάρων που πέρασαν από επεξεργασία βρίσκουμε τον μέσο χρόνο εκτέλεσης της διαδικασίας ανά κύτταρο. Αυτή η μονάδα μέτρησης φαίνεται να είναι η καταλληλότερη, γιατί η βασική μονάδα μεταφοράς στο ATM είναι το κύτταρο και μας ενδιαφέρει ιδιαίτερα ο μέσος χρόνος καθυστέρησης ενός κυττάρου λόγω της διαχείρισης μνήμης. Περιμένουμε ο μετρούμενος χρόνος να είναι υψηλότερος, όσο πιο μικρό είναι το μέγεθος του πακέτου.

5.5.3 ΟΙ ΠΑΡΑΜΕΤΡΟΙ

Παράμετροι του συστήματος είναι το ελάχιστο και μέγιστο μέγεθος πακέτου, ο συνολικός αριθμός των πακέτων που μπορούν να αποθηκευτούν στο σύστημα και ο αριθμός των ουρών πακέτων και κυττάρων. Το μέγεθος του πακέτου είναι μια από τις πιο κρίσιμες παραμέτρους όπως θα δούμε στις μετρήσεις και είναι μεταξύ 1 και 1200 κυττάρων. Οι ουρές πακέτων είναι σταθερά δύο (παρόλο που είναι παράμετρος), ενώ ο αριθμός των ουρών κυττάρων είναι από 2 έως και 16000. Ορίζουμε τον αριθμό των πακέτων με τέτοιο τρόπο ώστε τουλάχιστον 50000 κύτταρα να επεξεργάζονται σε κάθε επανάληψη. Για 400 επαναλήψεις, ο μέσος χρόνος του κάθε πειράματος είναι περίπου 5 λεπτά. Όλες οι μετρήσεις χρειάστηκε να καταγραφούν με το χέρι, γιατί δεν υπήρχε τρόπος αποθήκευσης των αποτελεσμάτων σε επεξεργαστή χωρίς λειτουργικό όπως ο i960. Η συνολική μνήμη που χρησιμοποιήθηκε για τις μνήμες Cell και Packet Bodies ήταν 4.8MB και σε αυτό το νούμερο

πρέπει να προσθέσουμε την απαιτούμενη μνήμη για την διαχείριση των ουρών, καθώς και τις υπόλοιπες μεταβλητές του προγράμματος. Για να σιγουρευτούμε ότι ο αριθμός των επαναλήψεων μας δίνει αρκετή ακρίβεια τετραπλασιάσαμε τις επαναλήψεις για επιλεγμένα πειράματα και δεν είδαμε καμμία διαφορά στους χρόνους εκτέλεσης ανά κύτταρο στα 3 πρώτα δεκαδικά ψηφία.

5.5.4 ΜΕΤΡΗΣΕΙΣ

Όλες οι μετρήσεις παρουσιάζουν μια ανάλυση της καλύτερης περίπτωσης (ένα κάτω φράγμα) της καθυστέρησης της λειτουργίας SAR σε ένα σύστημα με μικροεπεξεργαστή. Στα ακόλουθα σχήματα βλέπουμε διάφορες καμπύλες με μεταβλητές τον αριθμό των ουρών κυττάρων και το μέγεθος του πακέτου. Η διαχείριση ουρών, όπως είδαμε σε προηγούμενα υποκεφάλαια, αλλάζει λίγο με το πλήθος των ουρών από ένα σημείο κορεσμού και πάνω. Το γεγονός αυτό δείχνει την επεκτασιμότητα της διαχείρισης ουρών, αν υλοποιηθούν σε λογισμικό. Για να αντιγραφεί το σώμα ενός κυττάρου από ένα μέρος της μνήμης σε κάποιο άλλο χρειάζονται 5.6μs κατά μέσο όρο. Επίσης, μια κλήση στην ρουτίνα random() παίρνει 0.4μs, όπως βρήκαμε από ανεξάρτητες μετρήσεις. Τέλος διεξάγαμε μετρήσεις ως προς τον χρόνο αδειάσματος των ουρών κυττάρων για 50000 κύτταρα και μεταβλητό αριθ-μό ουρών, οι οποίες παρουσιάζονται στον Πίνακα 5.2.

Cell Queues	2	16	64	256	1024	2048	4096	8192	16384
Time (us)	49,895	49,895	49,903	49,927	50,024	50,153	50,412	50,928	51,961

Πίνακας 5.2: Χρόνος ‘‘αδειάσματος’’ των (μεταβλητού αριθμού) ουρών κυττάρων

Όπως παρατηρούμε καταναλώνεται 1μs ανά κύτταρο κατά μέσο όρο για το άδειασμα των ουρών. Επίσης, ας θυμηθούμε ότι οι εντολές EnQ και DeQ παίρνουν μεταξύ 0.6 και 0.9μs. Για την επεξεργασία ενός πακέτου που αποτελείται από N κύτταρα πρέπει να γίνουν οι εξής κλήσεις στις διάφορες ρουτίνες:

- 16 κλήσεις στην ρουτίνα random() : 2 για την επιλογή ουράς πακέτων και κυττάρων, 5 για την επικεφαλίδα, 8 για τον trailer και 1 για το μέγεθος του πακέτου
- N κλήσεις στην GetFree() (για να εξάγουμε N διευθύνσεις σε χώρους αποθήκευσης

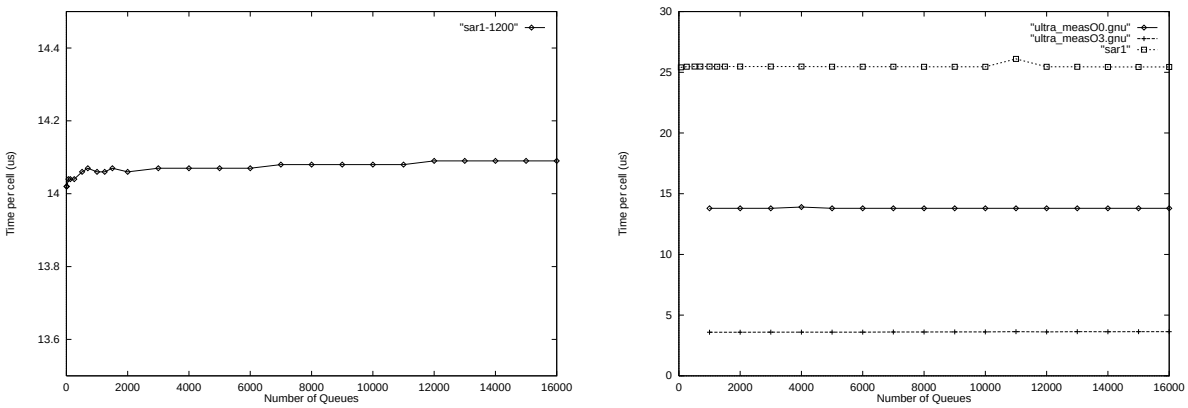
κυττάρων)

- 1 κλήση στην dequeue() (για την εξαγωγή του πακέτου προς επεξεργασία)
- N+1 κλήσεις στην enqueue() (για να εισάγουμε στις ουρές N κύτταρα του πακέτου και το ίδιο το πακέτο)
- 1 αντιγραφή του trailer των 8 bytes στο τέλος του πακέτου
- N αντιγραφές των κυττάρων (48 bytes) από την Packet στην Cell Bodies μνήμη
- N αντιγραφές της επικεφαλίδας κάθε κυττάρου στην μνήμη Cell Bodies

Τα αποτελέσματα των μετρήσεων για διάφορα μεγέθη πακέτων παρουσιάζονται στο Σχήμα 5.6. Όπως παρατηρούμε το κόστος (χρόνος εκτέλεσης) ανά κύτταρο πέφτει σημαντικά καθώς το μέγεθος των πακέτων αυξάνει. Αυτό οφείλεται στο ότι η διαδικασία SAR είναι χρονοβόρα και συμβαίνει ανά πακέτο με αποτέλεσμα όταν το πακέτο έχει το μέγεθος του κυττάρου να επιβαρύνει σημαντικά το χρόνο αποστολής του κυττάρου. Σε όλες τις περιπτώσεις, πακέτα N κυττάρων σημαίνει ότι το μέγεθος του πακέτου κυμαίνεται από N∗48 έως 40+N∗48 bytes, έτσι ώστε πάντα μαζί με τον trailer και το padding να είναι πολλαπλάσιο των 48 bytes.

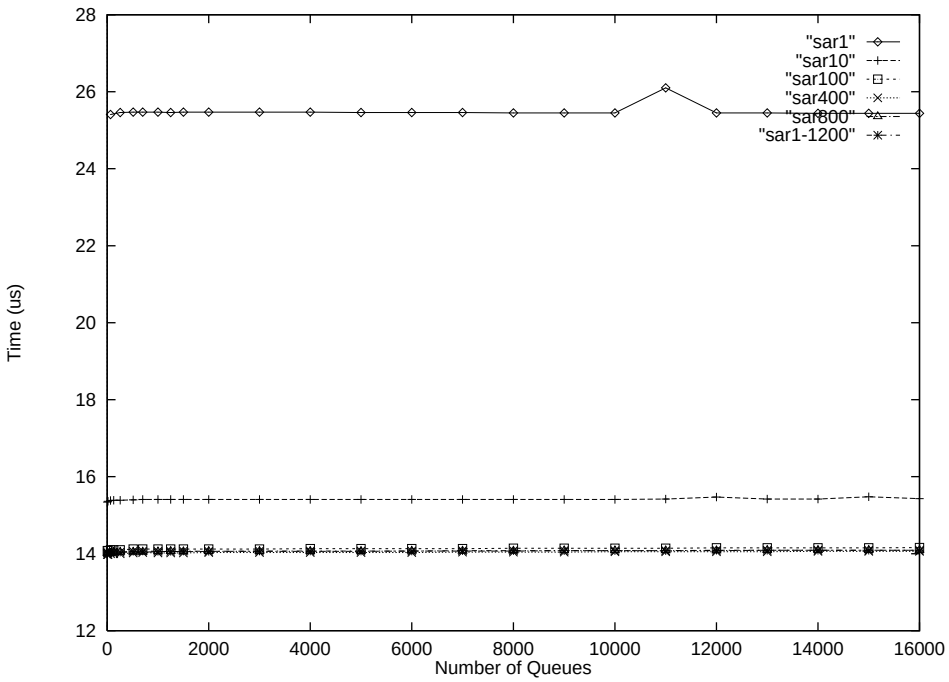
Η διαχείριση ουρών δεν επηρεάζει σημαντικά τον χρόνο εκτέλεσης της λειτουργίας όσο αυξάνει ο αριθμός των ουρών. Ουσιαστικά, για την διαχείριση μνήμης πληρώνουμε ένα σχεδόν σταθερό κόστος ανεξαρτήτως του αριθμού των ουρών. Για μεγέθη πακέτων άνω των 100 κυττάρων, υπάρχει μια μικρή μεν αλλά αυξητική τάση στον χρόνο εκτέλεσης ανά κύτταρο καθώς αυξάνεται ο αριθμός των ουρών. Στις καμπύλες (π.χ. Σχήμα 5.5(α)), παρουσιάζονται ορισμένες διακυμάνσεις που οφείλονται στο πρότυπο (pattern) των αποτυχιών (misses) στην κρυφή μνήμη του μP. Μετρήσεις που έγιναν σε SUN ULTRA SparcStation 60 (σχήμα 5.5(β), μεσαία και κάτω καμπύλη) υποδεικνύουν την επίδραση του μεγέθους της κρυφής μνήμης στον χρόνο εκτέλεσης, καθώς και οι διακυμάνσεις εξαφανίζονται*.

*Να αναφέρουμε ότι το SUN ULTRA SparcStation 60 έχει 2 MB κρυφής μνήμης και η μεταγλώττιση του κώδικα έγινε με τον gcc 2.8.1



(a) (b)

Σχήμα 5.5: Μετρήσεις για πακέτα 1-1200 κυττάρων και μετρήσεις για πακέτα 1 κυττάρου σε SUN Ultra



Σχήμα 5.6: Μετρήσεις του SAR για μεταβλητό αριθμό ουρών

5.5.5 ΑΝΑΛΥΣΗ

Στους πίνακες που ακολουθούν αναλύεται ο χρόνος εκτέλεσης της λειτουργίας κατακερματισμού ανά κύτταρο στις επιμέρους συνιστώσες του. Ο πρώτος πίνακας δείχνει την ανάλυση για πακέτα μεγέθους 1 κυττάρου και ο δεύτερος παρουσιάζει την ίδια ανάλυση για πακέτα των 100 κυττάρων (υπάρχουν μικρές διαφορές για πακέτα μεγαλύτερα των 100 κυττάρων). Για κάθε σύνολο επιμέρους συναρτήσεων (Op. Type) μετράμε τον αριθμό των κλήσεων (Number of Op.) για την επεξεργασία ενός κυττάρου και θέτοντας μια μέση καθυστέρηση για κάθε μια από αυτές (Time/Op.), όπως την έχουμε μετρήσει από ανεξάρτητες μετρήσεις, καταλήγουμε σε ένα συνολικό χρόνο εκτέλεσης της κάθε συνιστώσας. Αυτές οι επιμέρους συνιστώσες είναι:

- Οι λειτουργίες EnQ/DeQ (Queue Management)
- Η αντιγραφή της επικεφαλίδας, του trailer και του σώματος του κυττάρου
- Η διαδικασία αδειάσματος των ουρών κυττάρων
- Κλήσεις στην ρουτίνα random()

Η εγγραφή Total των Πινάκων 5.3 και 5.4 είναι το άθροισμα όλων αυτών των χρόνων εκτέλεσης, ενώ η εγγραφή Remaining δίνει την διαφορά ανάμεσα στις μετρήσεις και στον υπολογισμό με βάση τις μέσες καθυστερήσεις των συνιστωσών. Η διαφορά αυτή προκύπτει από το γεγονός ότι η ενσωμάτωση ενός συνόλου ρουτινών σε ένα μεγαλύτερο πρόγραμμα συνεπάγεται αλλαγή στην σειρά και τον αριθμό των αποτυχιών (misses) κατά την προσπέλαση της κρυφής μνήμης (cache) δεδομένων και εντολών του μP.

Op. Type	Number of Op.	Time/Op.	Total time(us)
DeQ	1	0.85	0.85 (5.1%)
EnQ	2	0.75	1.5 (9.1%)
Copy	61 bytes	0.11us/byte	6.7 (40.7%)
Empty Queues	1.0	1.0	1.0 (6.1%)
Random()	16	0.4	6.4 (39.0%)
Total	-	-	16.45 (100%)
Remaining	-	-	+8.55 (+52%)

Πίνακας 5.3: Διαμοιρασμός χρόνου μεταξύ των λειτουργιών για πακέτα 1 κυττάρου

Op. Type	Number of Op.	Time/Op.	Total time(us)
DeQ	$\frac{1}{100}$	0.85	0.0 (0.0%)
EnQ	$1+\frac{1}{100}$	0.75	0.75 (9.9%)
Copy	53 bytes	0.11us/byte	5.83 (76.9%)
Empty Queues	1.0	1.0	1.0 (13.2%)
Random()	$\frac{16}{100}$	0.4	0.0 (0.0%)
Total	-	-	7.58 (100%)
Remaining	-	-	+6.42 (+84%)

Πίνακας 5.4: Διαμοιρασμός χρόνου μεταξύ των λειτουργιών για πακέτα 100 κυττάρων

Η διαχείριση μνήμης απαιτεί τον χρόνο για να εκτελεστούν 3 λειτουργίες στην περίπτωση πακέτων μεγέθους 1 κυττάρου:

1. εξαγωγή ενός πακέτου από την ουρά πακέτων,

2. εισαγωγή ενός κυττάρου στην ουρά κυττάρων και

3. επαναεισαγωγή του πακέτου πίσω στην ουρά από όπου προήλθε.

Στην περίπτωση πακέτων μεγάλου αριθμού κυττάρων (άνω των 10) η διαχείριση μνήμης απαιτεί τον χρόνο για να εκτελεστεί η λειτουργία της εισαγωγής ενός κυττάρου στην ουρά κυττάρων, καθώς οι υπόλοιπες δύο λειτουργίες που αναφέρονται στην προηγούμενη περίπτωση γίνονται σπάνια (κάθε N κύτταρα, όπου N ο αριθμός των κυττάρων ενός πακέτου).

	Time per cell	Queue Management
1 cell packet	25.0	14.2%
100 cell packet	14.0	9.9%

Πίνακας 5.5: Χρόνος εκτέλεσης της Διαχείρισης Ουρών ανά κύτταρο

Το γενικό συμπέρασμα είναι ότι η Διαχείριση Ουρών καταλαμβάνει το 10-14% της ισχύος της CPU σε ένα βελτιστοποιημένο σύστημα ATM που εκτελεί SAR και αυτός ο αριθμός μπορεί να παρεκλίνει 52-84% κατά την δυναμική εκτέλεση σε ένα μεγαλύτερο κομμάτι κώδικα. Παρόλο που η απόκλιση αυτή φαίνεται μεγάλη δεν μας εκπλήσσει, καθώς

η ενσωμάτωση των ρουτινών που εκτελούν Διαχείριση Ουρών σε ένα μεγαλύτερο πρόγραμμα (κώδικας SAR) προκαλεί αλλαγή στις αποτυχίες προσπέλασης της κρυφής μνήμης (cache misses). Άρα, η Διαχείριση Ουρών χρησιμοποιούμενη σαν υπορουτίνα σε βελτιστοποιημένα συστήματα ATM μπορεί να χρησιμοποιεί μέχρι και το 25% της διαθέσιμης ισχύος της CPU που είναι ένα σημαντικό ποσοστό. Η ρουτίνα που επιτελεί τον Κατακερματισμό (Segmentation) έχει χαμηλή καθαρή παροχή στον i960 που κυμαίνεται από 15.4Mb/s έως 27.2Mb/s και μπορεί να εξυπηρετήσει μόνο συστήματα χαμηλής ταχύτητας με σχετικά μεγάλα πακέτα. Από την άλλη πλευρά, αν χρησιμοποιήσουμε τον μP μόνο για να εκτελούμε την λειτουργία Διαχείρισης Ουρών και υποθέτοντας 2 λειτουργίες ανά κύτταρο μπορούμε να έχουμε καθαρή παροχή $\frac{8*53}{0.75+0.85} = 265 Mb/s$ μαζί με τις επικεφαλίδες. Περισσότερες συγκρίσεις μεταξύ των διαφορετικών υλοποιήσεων υλικού μπορούν να βρεθούν στο κεφάλαιο 6.

ΚΕΦΑΛΑΙΟ 6

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΚΑΤΕΥΘΥΝΣΕΙΣ

Έχουμε διερευνήσει δυο διαφορετικές προσεγγίσεις για τη διαχείριση ουρών σε υλικό και σε λογισμικό. Παρουσιάσαμε 3 υλοποιήσεις σε υλικό και μία σε λογισμικό για τον μικροεπεξεργαστή i960. Παρακάτω συγκρίνουμε τις διαφορετικές προσεγγίσεις, αξιολογούμε τα πλεονεκτήματα και τα μειονεκτήματα και καταλήγουμε σε συμπεράσματα για την ταχύτητα και την πολυπλοκότητα των υλοποιήσεων.

Για τη υλοποίηση σε υλικό αναλύουμε το βασικό πυρήνα της μονάδας Διαχείρισης Ουρών για κάθε αρχιτεκτονική (MuQPro I, ''Πλήρως επεκτάσιμη''), δηλαδή αυτή που υλοποιεί ένα σύνολο ουρών. Δεν αναφέρουμε συγκριτικά στοιχεία για την ''πλήρως παράλληλη'' υλοποίηση γιατί, σε αντίθεση με τις άλλες δύο υλοποιήσεις, περιέχει μνήμες που βρίσκονται μέσα στο ολοκληρωμένο (on-chip memories) και το μονοπάτι δεδομένων της είναι μικρού πλάτους. Άρα δεν έχουμε κάποιο άμεσο μέτρο σύγκρισης. Όλες οι αρχιτεκτονικές στοχεύουν στις συχνότητες των 20 και 30 MHz ώστε να διατηρηθεί μια ισορροπία μεγέθους - ταχύτητας και να ελαττωθεί το κόστος χρησιμοποιώντας μικρές συσκευές (devices) FPGA. Τα στατιστικά χαρακτηριστικά των υλοποιήσεων παρουσιάζονται στον Πίνακα 6.1, όπου φαίνεται ότι η απαιτούμενη λογική είναι συγκρίσιμη σε κάθε περίπτωση. Η ''Πλήρως επεκτάσιμη'' υλοποίηση απαιτεί κάπως περισσότερη λογική σε σύγκριση με αυτή του MuQPro I, ενώ ο αριθμός των λογικών κυττάρων που προσθέτει ο μεταγλωττιστής σχεδίου (design compiler) αυξάνει όσο μικρότερο κύκλο ρολογιού προσπαθούμε να επιτύχουμε.

Implementation	Pin util.	Logic util.	Avg Fan-in	Extra cells
Fully scalable @ 20 MHz	57%	87%	3.15/4	27%
MuQPro I @ 20 MHz	57%	82%	3.21/4	28%
Fully scalable @ 30 MHz	57%	89%	3.18/4	31%
MuQPro I @ 30 MHz	57%	85%	3.14/4	32%

Πίνακας 6.1: Στατιστικά στοιχεία των υλοποιήσεων υλικού

Επειδή οι υλοποιήσεις υλικού έχουν μικρές διαφορές ως προς την ποσότητα της λογικής που απαιτούν, εστιάζουμε την προσοχή μας στη ταχύτητα που μπορούν να επιτύχουν ορίζοντας ένα μέτρο σύγκρισης. Ακολουθώντας το μοντέλο που παρουσιάσαμε στο Κεφάλαιο 2, κύτταρα που εισέρχονται στο σύστημα ATM αποθηκεύονται σε λογικές ουρές εισόδου και κύτταρα που αναχωρούν εξάγονται από λογικές ουρές εξόδου. Σε ένα τέτοιο σύστημα διπλής κατεύθυνσης απαιτείται μια εντολή Enqueue για κάθε άφιξη κυττάρου και μια Dequeue για κάθε αναχώρηση. Παραλείποντας τις εντολές Getfree και Retfree, καθώς είναι οι λιγότερο χρονοβόρες -με τη μισή ή και κατά περίπτωση μικρότερη καθυστέρηση συγκριτικά με τις EnQ και DeQ (Πίνακας 6.2)- υπολογίζουμε τις επιδόσεις των υλοποιήσεων με βάση τις καθυστερήσεις των ‘‘αργών’’ εντολών EnQ και DeQ. Η παροχή των συστημάτων μετράται βάσει του τύπου $\frac{P}{C*N}$, όπου P είναι το μέγεθος του πακέτου σε bits, C είναι ο κύκλος ρολογιού του συστήματος σε ns και N είναι ο αριθμός των κύκλων εκτέλεσης της εντολής. Το γινόμενο $C * N$ στην περίπτωση της υλοποίησης σε λογισμικό είναι 0.75 μs και 0.95 μs για τις εντολές EnQ και DeQ αντίστοιχα, όπως προκύπτει από τις μετρήσεις του Κεφαλαίου 5. Ο Πίνακας 6.2 συνοψίζει την απόδοση του Διαχειριστή Ουρών για τις υλοποιήσεις σε υλικό και λογισμικό. Όλα τα νούμερα δείχνουν την καλύτερη και την χειρότερη περίπτωση λειτουργίας εισόδου και εξόδου από την ουρά χωρίς καθυστερήσεις των εντολών Getfree και Retfree. Η παροχή μετράται χωρίς τις επικεφαλίδες, δηλαδή μετράμε καθαρή παροχή στο επίπεδο ATM (δηλαδή $P = 384\,bits$ στον παραπάνω τύπο).

Επιπρόσθετα, πρέπει να υπάρχει κάποιο υλικό (hardware) που να μεταφέρει τα σώματα των κυττάρων και τις επικεφαλίδες σε μια μνήμη με παροχή που να ικανοποιεί τέτοιες ταχύτητες, ώστε να είναι αποδοτικός στο σύστημα ο Διαχειριστής Ουρών. Όλες οι υλοποιήσεις σε υλικό προσφέρουν επεκτασιμότητα, αφού πάντοτε μπορούμε να μετακινούμαστε

Implementation	EnQ Throughput	DeQ Throughput	Throughput/link
i960 @ 40MHz	512 Mb/s	404 Mb/s	226 Mb/s
fully scalable @ 20MHz	1.09 Gb/s - 1.53 Gb/s	960 Mb/s - 1.28 Gb/s	512 Mb/s - 590 Mb/s
MuQPro I @ 20MHz	1.28 Gb/s - 1.92 Gb/s	960 Mb/s - 1.28 Gb/s	549 Mb/s - 640 Mb/s
fully scalable @ 30MHz	1.66 Gb/s - 2.33 Gb/s	1.45 Gb/s - 1.94 Gb/s	776 Mb/s - 895 Mb/s
MuQPro I @ 30MHz	1.94 Gb/s - 2.9 Gb/s	1.45 Gb/s - 1.94 Gb/s	831 Mb/s - 970 Mb/s

Πίνακας 6.2: Σύγκριση της παροχής των διαφόρων υλοποιήσεων

σε μια μεγαλύτερη και πιο ακριβή συσκευή και να φτάνουμε υψηλότερες ταχύτητες. Η υψηλότερη που έχουμε φτάσει για μικρό αριθμό από pin και χαμηλού κόστους συσκευή FPGA είναι 32MHz σε συσκευή EPF10K40 αλλά, όσο η τεχνολογία FPGA αναπτύσσεται, υψηλότερες ταχύτητες φαίνονται εφικτές. Εκτός της επεκτασιμότητας ως προς την ταχύτητα, σημαντική παράμετρος είναι και η επεκτασιμότητα ως προς το αριθμό των ουρών. Όλο το υλικό που συζητήθηκε μέχρι τώρα έχει μονοπάτι δεδομένων 16-bit και μπορεί να διευθυνσιοδοτήσει μέχρι 64K διαφορετικές θέσεις μοιρασμένες σε διάφορες δομές δεδομένων. Και οι δυο υλοποιήσεις σε υλικό (MuQPro I, ''πλήρως επεκτάσιμη'') είναι παραμετροποιημένες ώστε με την αλλαγή της παραμέτρου του πλάτους να προκύπτουν πλατύτερα μονοπάτια δεδομένων και να μπορούν να χωρέσουν σε νεώτερη και γρηγορότερη συσκευή.

Αν υλοποιήσουμε τον Διαχειριστή Ουρών και την αποθήκευση των επικεφαλίδων και του σώματος σε υλικό και τα υπόλοιπα μέρη του συστήματος ATM σε λογισμικό, τότε η παροχή του συστήματος, (υποθέτοντας μηδενική επικάλυψη και μηδενική καθυστέρηση του interface μεταξύ του επεξεργαστή και του υπόλοιπου υλικού) είναι μεταξύ $\frac{48*8\ bits}{25-5.6-0.75\mu s} = 20.59Mb/s$ και $\frac{48*8\ bits}{25-(5.6+0.75)*1.84\mu s} = 28.84b/s$ σε σύγκριση με μια απλή υλοποίηση σε λογισμικό ($\frac{48*8\ bits}{25\mu s} = 15.36Mb/s$) που πρακτικά σημαίνει μια επιτάχυνση (speedup) μεταξύ 1.34-1.88. Ο παρονομαστής των κλασμάτων προκύπτει από τους χρόνους που αναλύθηκαν στον Κεφάλαιο 5 για την συνολική επεξεργασία SAR ενός κυττάρου, αν αφαιρέσουμε τους χρόνους αποθήκευσης των σωμάτων και των επικεφαλίδων των κυττάρων. Οι χρόνοι αυτοί μπορεί να είναι αυξημένοι μέχρι και κατά 84% (Κεφάλαιο 5). Συνεπώς η μεταφορά δεδομένων και ο διαχειριστής ουρών σε υλικό αυξάνουν σημαντικά την απόδοση του συστήματος, η οποία μπορεί επιπλέον να αυξηθεί αν η δημιουργία των επικεφαλίδων υλοποιηθεί επίσης σε υλικό.

Συμπερασματικά ένας μικροεπεξεργαστής που εκτελεί τις βασικές λειτουργίες ενός

συστήματος ATM, έχει χαμηλή απόδοση με χαμηλή παροχή 15-27 Mb/s, με τυπική σημερινή τεχνολογία. Πρόσθετο υλικό ειδικού σκοπού μπορεί να οδηγήσει στην επίτευξη υψηλότερων ταχυτήτων. Οι διαφορετικές υλοποιήσεις υλικού που παρουσιάστηκαν, επιτυγχάνουν σημαντική αύξηση ταχύτητας. Ανάμεσα σε αυτές τις υλοποιήσεις, είναι προτιμητέα αυτή του MuQPro I στα 20MHz και 30MHz, που χωράει στην FPGA FLEX10K20 της ALTERA με λογικό κόστος. Από την άλλη πλευρά η ''πλήρως επεκτάσιμη'' υλοποίηση χρησιμοποιεί πλήρως την εξωτερική μνήμη (SRAM), αλλά με μεγαλύτερη πολυπλοκότητα και χαμηλότερη επεκτασιμότητα ως προς την ταχύτητα.

Υπάρχουν διάφορες κατευθύνσεις για επέκταση της εργασίας που παρουσιάστηκε. Μια σημαντική επέκταση αυτής της εργασίας είναι η χρήση ιχνών (traces) πραγματικής κυκλοφορίας δικτύων ATM σαν είσοδο στο μοντέλο SAR καθώς και την Διαχείριση Ουρών (σε λογισμικό και υλικό), που θα μας έδινε πιο ρεαλιστικά αποτελέσματα ως προς την απόδοση αυτών κάτω από πραγματικές συνθήκες. Επίσης σημαντικό θα ήταν να συνδεθεί το FPGA και η στατική μνήμη με τον μP i960 και να γίνει αποτίμηση της επιτάχυνσης που πραγματικά επιτυγχάνεται με την χρήση Διαχείρισης Ουρών σε υλικό (στην δική μας αποτίμηση θεωρούμε μηδενική καθυστέρηση διεπαφής μεταξύ μP και QM). Σύμφωνα με τις μετρήσεις μας αναμένουμε μια συνολική βελτίωση της απόδοσης του συστήματος από 10-20% ανάλογα με το πραγματικό ποσοστό της επεξεργαστικής ισχύος που χρησιμοποιεί ο Διαχειριστής Ουρών, καθώς η Διαχείριση Ουρών μπορεί να εκτελεστεί παράλληλα με την αποθήκευση των σωμάτων των κυττάρων στην μνήμη. Τέλος μια μελλοντική επέκταση της εργασίας αυτής θα ήταν η χρήση τεχνικών VLIW για παράλληλο υπολογισμό διευθύνσεων περισσοτέρων από μιας εντολών. Επίσης είναι προς μελέτη η ομαδοποίηση ζευγών ή τριάδων εντολών (π.χ. GetFree - EnQ) σε ένα κώδικα εντολής και συνολικός προγραμματισμός των μικροεντολών αυτών, έτσι ώστε να επιτύχουμε ακόμη μεγαλύτερη ταχύτητα (λιγότερους κύκλους).

ΠΑΡΑΡΤΗΜΑ Α

ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ ΚΑΙ ΔΙΑΓΡΑΜΜΑΤΑ ΧΡΟΝΙΣΜΟΥ ΕΝΤΟΛΩΝ

Σε αυτό το παράρτημα αναλύουμε το σύνολο εντολών του Διαχειριστή Ουρών (QM).

Στον Πίνακα Α.1 παρουσιάζονται όλα τα σήματα μαζί με την σημασιολογία τους (συμπεριλαμβάνονται και σήματα που χρησιμοποιήθηκαν για έλεγχο σφαλμάτων και φαίνονται στα διαγράμματα χρονισμού που θα ακολουθήσουν).

Στον Πίνακα Α.2, φαίνεται ο αριθμός των κύκλων εκτέλεσης κάθε εντολής όπου το πρώτο νούμερο δείχνει την απόλυτη καθυστέρηση και το δεύτερο τον ελάχιστο αριθμό των κύκλων εκτέλεσης στην περίπτωση χρήσης της επικάλυψης.

Αναφορικά με την εντολή Init αυτή χρειάζεται $2N_q + N_c + 3$ κύκλους ανά σύνολο ουρών, όπου N_q είναι ο αριθμός των ουρών και N_c ο αριθμός των θέσεων μνήμης (buffers). Για κάθε σύνολο ουρών χρειάζεται μια ξεχωριστή εντολή Init.

Στα σχήματα Α.1 και Α.2 παρουσιάζεται ο βαθμός επικάλυψης των εντολών EnQ, DeQ, GetFree, RetFree τις οποίες και βελτιστοποιήσαμε καθώς είναι οι σημαντικότερες. Κάθε ‘‘τετράγωνο’’ παριστάνει ένα κύκλο ρολογιού. Παρατηρείστε ότι μετά από μια εντολή RetFree (που δόθηκε στον κύκλο 0) δεν μπορούμε να δώσουμε μια εντολή GetFree ή RetFree στον κύκλο 1, γιατί οι καταχωρητές Free List Head και Free List Tail δεν έχουν την σωστή τιμή ακόμη. Μη δίνοντας στον κύκλο 1 την νέα εντολή ‘‘χάνουμε’’ και τον επόμενο αυτού (δηλαδή τον κύκλο 2) καθώς η αρχιτεκτονική δεν μας επιτρέπει να δώσουμε

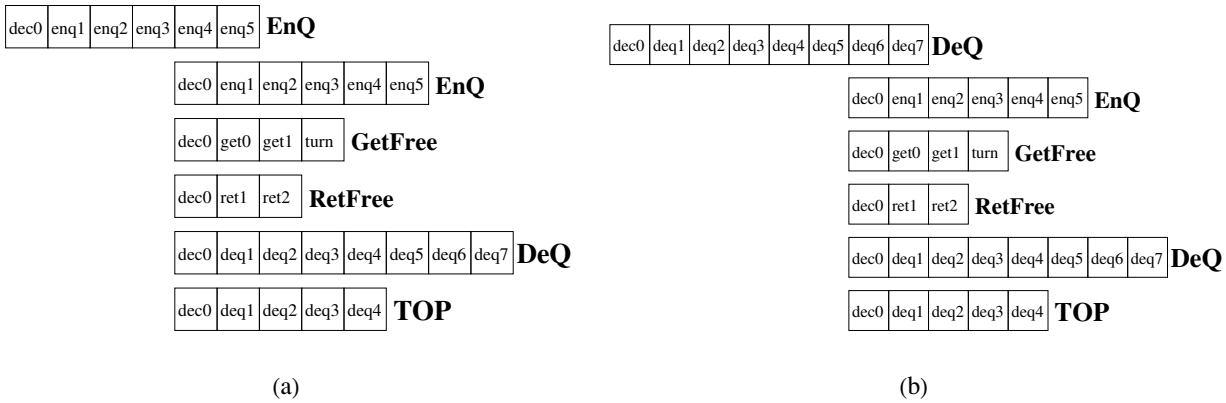
Pin	Semantics
clk	clock
clr	reset
op[4..0]	opcode
valid	valid
ready	ready
e_ne	e_ne
dio	Data to/from Master
swen	SRAM Write Enable
saddr	SRAM Address
sdio	SRAM Data I/O
qmdpfs	QM FSM state
ddrive	Pins driven by QM_{ddrive}
flebit	Free List Empty Bit

Πίνακας Α.1: Επεξήγηση των σημάτων του Διαχειριστή Ουρών του MuQPro I

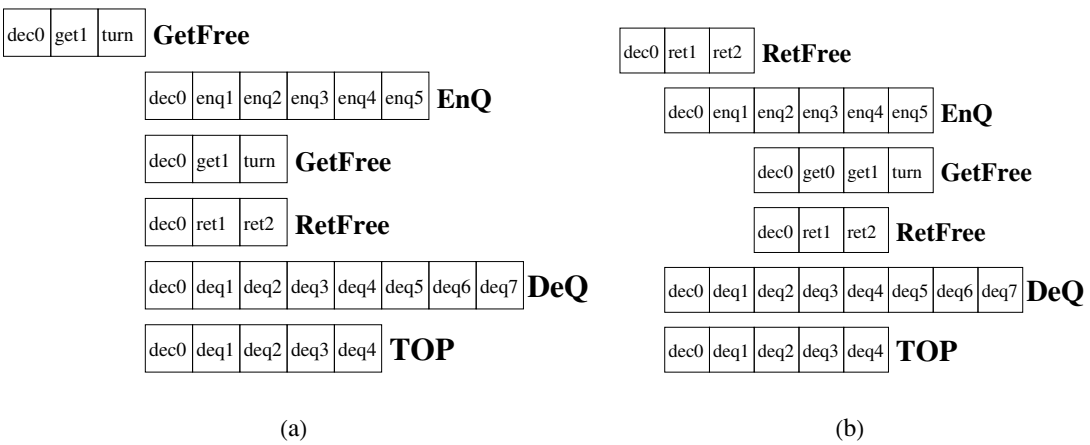
εντολή στη διάρκεια του κύκλου 2 για λόγους αυξημένης πολυπλοκότητας του μονοπατιού ελέγχου. Άλλο ένα χαρακτηριστικό της αρχιτεκτονικής μας είναι ότι *δεν επιτρέπεται να εκτελούνται ταυτόχρονα περισσότερες από δύο εντολές.* Συνεπώς, δεν είναι έγκυρο το σενάριο μιας εντολής EnQ ακολουθούμενης από μια εντολή RetFree σε μια άδεια Free List, και αυτή ακολουθούμενη με την σειρά της από μια νέα εντολή RetFree τον αμέσως επόμενο κύκλο (από την έναρξη της πρώτης RetFree) *δεν* είναι έγκυρο.

Instruction	Maximum number of cycles	Minimum number of cycles
EnQ	6	4
DeQ	8	6
GetFree	3	2
RetFree	3	1
Top	5	5
Init	$2N_q + N_c + 3$	$2N_q + N_c + 3$
Read	3	3
Write	3	3

Πίνακας Α.2: Αριθμός κύκλων ρολογιού για όλες τις εντολές του MuQPro I



Σχήμα A.1: Επικάλυψη εντολών που ακολουθούν την Enqueue και την Dequeue



Σχήμα A.2: Επικάλυψη εντολών που ακολουθούν την Getfree και την Retfree

Α.1 Η ΕΝΤΟΛΗ ENQUEUE

Η εντολή αυτή δέχεται 2 ορίσματα: τον αριθμό ουράς και την διεύθυνση σε ένα ελεύθερο χώρο μνήμης. Η υλοποίηση σε C φαίνεται στο σχήμα Α.3.

```
unsigned int
EnQ(unsigned int *HTm, unsigned char *HTe,
    unsigned int *PMm, unsigned int Slotn, unsigned int Qn)
{
  unsigned int pTail;

  if (HTe[Qn]==1)
    {
      HTe[Qn]=0;
      HTm[Qn<<1]=Slotn;
      HTm[(Qn<<1)+1]=Slotn;
    }
  else
    {
      pTail=HTm[(Qn<<1)+1];
      PMm[prevTail]=Slotn;
      HTm[(Qn<<1)+1]=Slotn;
    }
}
```

Σχήμα Α.3: Κώδικας C της εντολής EnQ

Ένα παράδειγμα λειτουργίας της εντολής, για 8 θέσεις μνήμης (slots) και 4 ουρές, φαίνεται στο Σχήμα Α.5. Αυτή η λειτουργία περιέχει 4 μικροεντολές (προσπελάσεις στην μνήμη): Προσπέλαση του Empty Bit και έλεγχος αν έχει τεθεί, και 3 ακόμη προσπελάσεις σε κάθε μια από τις περιπτώσεις που παρουσιάζονται στο Σχήμα Α.4. Η ακριβής εκτέλεση των μικροεντολών σε κύκλους ρολογιού είναι η εξής:

Κύκλος 0: Αποθήκευση (latch) του πρώτου ορίσματος που είναι ο αριθμός ουράς (Qn).

Κύκλος 1: Αποθήκευση του δεύτερου ορίσματος Slotn (διεύθυνση σε χώρο μνήμης) και υπολογισμός της διεύθυνσης για την προσπέλαση του Empty Bit του αριθμού ουράς που έχουμε από τον προηγούμενο κύκλο. Η διεύθυνση αυτή είναι (2∗Qn + 1 + BAR_HT), όπου BAR_HT είναι είναι η διεύθυνση βάσης της δομής Head/Tail table

στην μνήμη.

Κύκλος 2: Ανάγνωση του Empty Bit και του δείκτη τέλους (Tail pointer) από την στατική μνήμη (και εγγραφή τους στον προσωρινό καταχωρητή pTail) με βάση την διεύθυνση που υπολογίσαμε στον προηγούμενο κύκλο. Δεν ξέρουμε αν θα διαβάσουμε ή θα γράψουμε τον δείκτη τέλους στον επόμενο κύκλο γιατί η απόφαση αυτή βασίζεται στον Empty Bit που θα έχουμε διαβάσει στο τέλος του κύκλου.

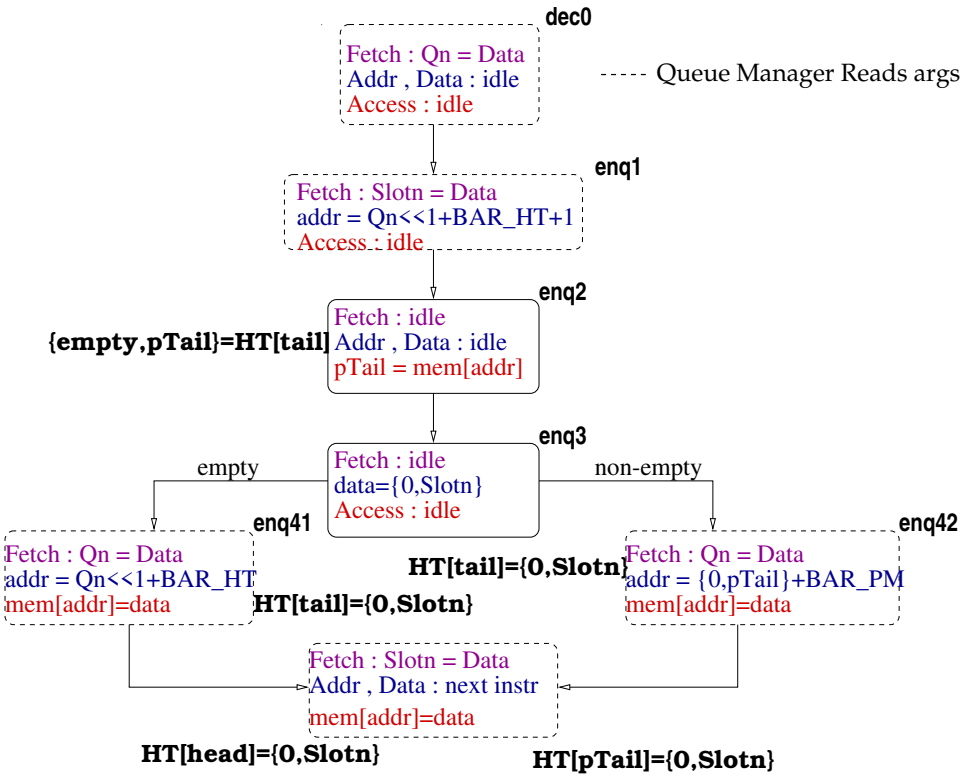
Κύκλος 3: Έχοντας διαβάσει το Empty Bit μπορούμε να αποφασίσουμε σε αυτό τον κύκλο αν η ουρά που προσπελαύνουμε είναι άδεια. Επίσης σε αυτό τον κύκλο μπορούμε να υπολογίσουμε τα δεδομένα που θα γραφούν στον επόμενο ανεξάρτητα από την κατάσταση του Empty Bit. Τα δεδομένα προς εγγραφή είναι {0,Slotn} (το Empty Bit είναι αποθηκευμένο μαζί με το Tail) και η διεύθυνση είναι (2∗Qn + 1 + BAR_HT). Επίσης Ready=1, στον επόμενο κύκλο μπορούμε να δεχθούμε μια νέα εντολή.

Κύκλος 4(α): Αν Empty Bit=0 γράφουμε στην μνήμη την νέα τιμή του Tail pointer (υπολογίστηκε στον κύκλο 3). Επίσης υπολογίζουμε την διεύθυνση του τελευταίου στοιχείου της ουράς που είναι BAR_PM+pTail, όπου BAR_PM η διεύθυνση βάσης της Pointer memory. Σε αυτό τον κύκλο Valid=1 και E_NE=Empty Bit. Μια νέα εντολή (EnQ, DeQ, GetFree, RetFree) μαζί με το πρώτο της όρισμα μπορεί να δοθεί σε αυτό τον κύκλο.

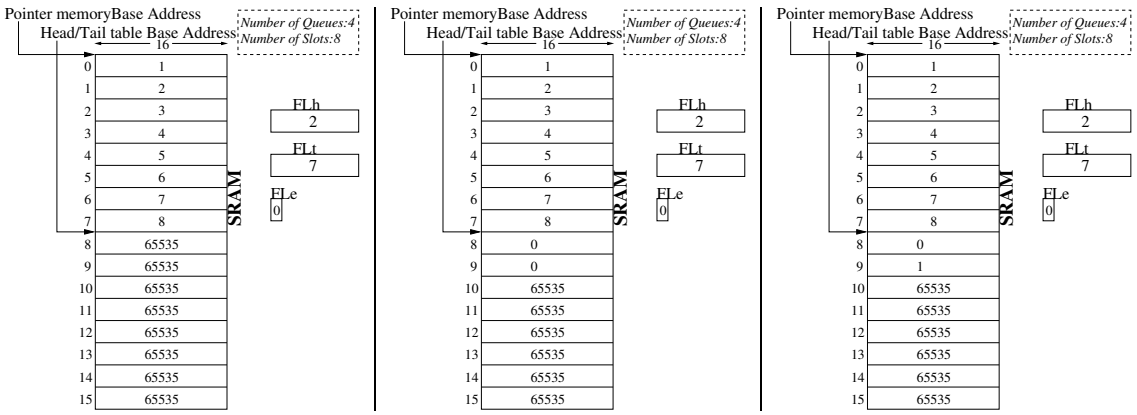
Κύκλος 4(β): Αν Empty Bit=1 γράφουμε στην μνήμη όπως και στην περίπτωση 4(α). Σαν διεύθυνση επόμενης προσπέλασης υπολογίζουμε την (2∗Qn + BAR_HT). Σε αυτό τον κύκλο Valid=1 και E_NE=Empty Bit. Μια νέα εντολή (EnQ, DeQ, GetFree, RetFree) μαζί με το πρώτο της όρισμα

Κύκλος 5: Σε αυτό τον κύκλο γίνεται μια εγγραφή στην στατική μνήμη στην διεύθυνση που υπολογίσαμε στον κύκλο 4. Επίσης φέρνουμε και το δεύτερο όρισμα (αν υπάρχει) της εντολής που δόθηκε στον κύκλο 4.

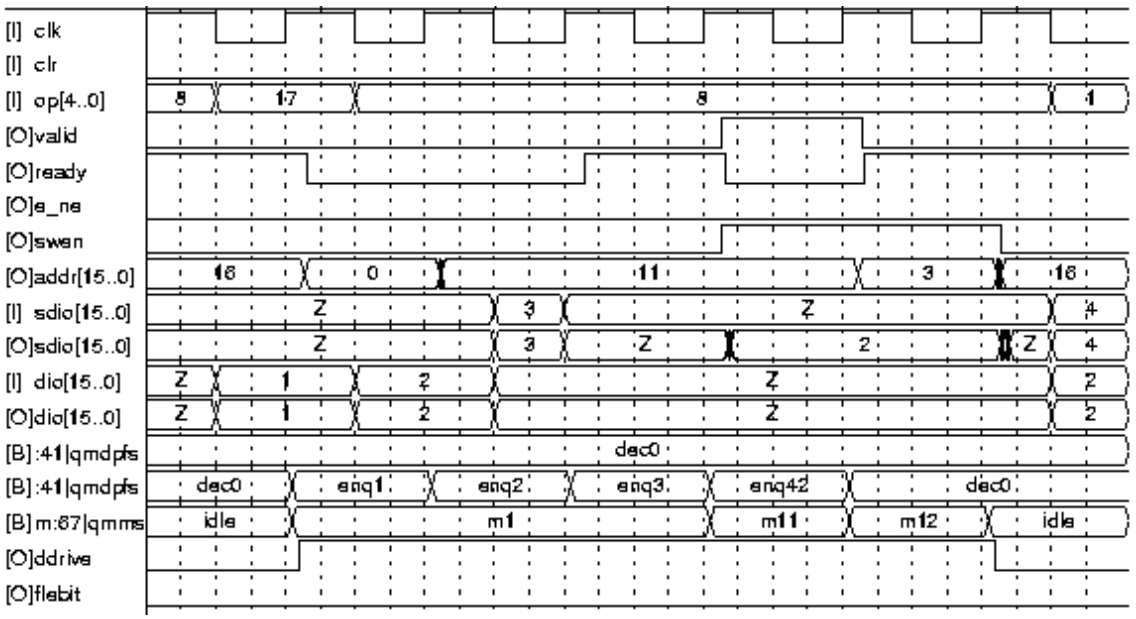
Τα διαγράμματα χρονισμού για τις δύο περιπτώσεις της εντολής EnQ παρουσιάζονται στα Σχήματα Α.6 και Α.7.



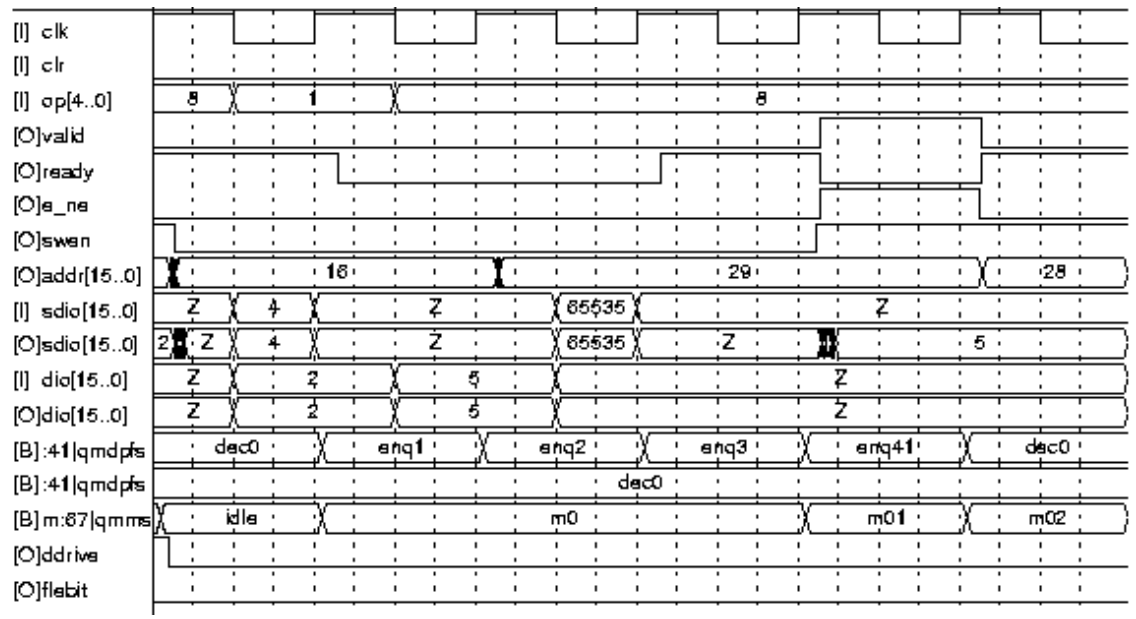
Σχήμα Α.4: Προγραμματισμός μικροεντολών για την εντολή EnQ



Σχήμα Α.5: Αρχική κατάσταση των ουρών, EnQ στην ουρά 0 το slot 0, EnQ στην ουρά 0 το slot 1



Σχήμα A.6: Εντολή EnQ σε μη άδεια ουρά



Σχήμα A.7: Εντολή EnQ σε άδεια ουρά

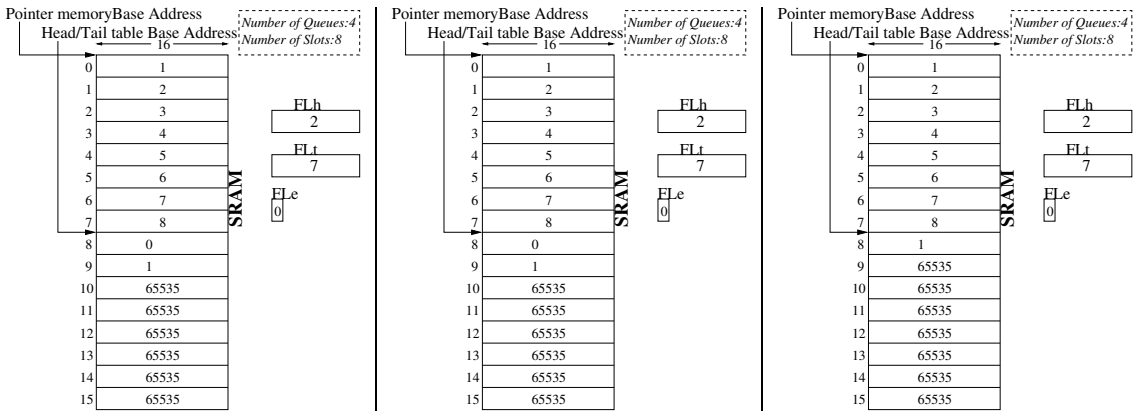
Α.2 Η ΕΝΤΟΛΗ DEQUEUE

Η εντολή αυτή δέχεται ένα όρισμα, τον αριθμό ουράς Qn, και επιστρέφει μια διεύθυνση σε χώρο μνήμης, Slotn. Η υλοποίησή της σε κώδικα C παρουσιάζεται στο σχήμα Α.8.

```
unsigned int
DeQ(unsigned int *HTm, unsigned char *HTe,
    unsigned int *PMm, unsigned int Qn, unsigned int *valid)
{
  unsigned int pHead, pTail, pPM, ret_val=0;

  *valid=0;
  if (HTe[Qn]==1)
    {
      *valid=0;
    }
  else
    {
      pHead=HTm[Qn<<1];
      pTail=HTm[(Qn<<1)+1];
      ret_val = pHead;
      *valid = 1;
      if (pHead==pTail)
        HTe[Qn]=1;
      else
        {
          pPM=PMm[pHead];
          HTm[Qn<<1]=pPM;
        }
    }
  return ret_val;
}
```

Σχήμα Α.8: Κώδικας C της εντολής DeQ

Η εντολή αυτή αναλύεται σε: (α) 6 μικροεντολές αν η ουρά είναι μη άδεια και έχει περισσότερα από 1 στοιχεία, (β) 5 μικροεντολές αν η ουρά είναι μη άδεια και έχει μόνο ένα στοιχείο και (γ) 2 μικροεντολές αν η ουρά είναι άδεια. Ένα παράδειγμα των 2 περιπτώσεων που η ουρά είναι μη άδεια φαίνεται στο Σχήμα Α.9. Ο προγραμματισμός των μικροεντολών παρουσιάζεται στο σχήμα Α.10.



Σχήμα A.9: Αρχική κατάσταση των ουρών, DeQ από την ουρά 0, DeQ από την ουρά 0

Η ακριβής εκτέλεση των μικροεντολών σε κύκλους ρολογιού είναι η εξής:

Κύκλος 0: Αποθήκευση (latch) του πρώτου ορίσματος που είναι ο αριθμός ουράς (Qn).

Κύκλος 1: Υπολογισμός της διεύθυνσης για την προσπέλαση του Empty Bit του αριθμού ουράς που έχουμε από τον προηγούμενο κύκλο. Η διεύθυνση αυτή είναι (2∗Qn + 1 + BAR_HT).

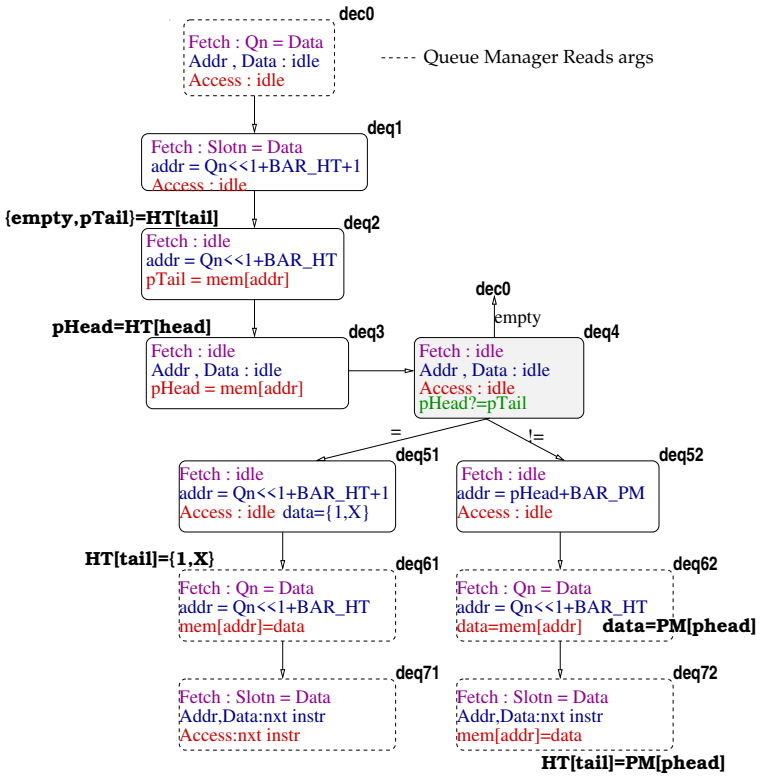
Κύκλος 2: Ανάγνωση του Empty Bit και του δείκτη τέλους (Tail pointer) από την στατική μνήμη (και εγγραφή τους στον προσωρινό καταχωρητή pTail) με βάση την διεύθυνση που υπολογίσαμε στον προηγούμενο κύκλο. Υπολογισμός της διεύθυνσης του στοιχείου κεφαλής της ουράς που είναι (2∗Qn + BAR_HT).

Κύκλος 3: Προσπέλαση του στοιχείου κεφαλής της ουράς και αποθήκευση στον καταχωρητή pHead.

Κύκλος 4: Επιστροφή του κορυφαίου στοιχείου της ουράς οδηγώντας τα Data, Valid pins. Υπολογισμός αν pHead == pTail. Υπολογισμός των δεδομένων {1,Slotn}.

Κύκλος 5(α): *Αν pHead == pTail* τότε E_NE = Valid = 1. Υπολογίζουμε την διεύθυνση του Tail pointer που είναι (2∗Qn + BAR_HT + 1). Επίσης Ready=1.

Κύκλος 6(α): Γράφουμε στην διεύθυνση που υπολογίσαμε στον κύκλο 5(α) το {1,Slotn}. Φέρνουμε μια νέα εντολή μαζί με το πρώτο της όρισμα.



Σχήμα Α.10: Προγραμματισμός μικροεντολών για την εντολή DeQ

Κύκλος 7(α): Φέρνουμε το δεύτερο όρισμα της εντολής που δόθηκε στον κύκλο 6(α) (αν υπάρχει).

Κύκλος 5(β): *Αν pHead != pTail* τότε E_NE = 0 και Valid = 1. Υπολογίζουμε την διεύθυνση του στοιχείου της κεφαλής της ουράς (Head element) που είναι pHead + BAR_PM. Επίσης Ready=1.

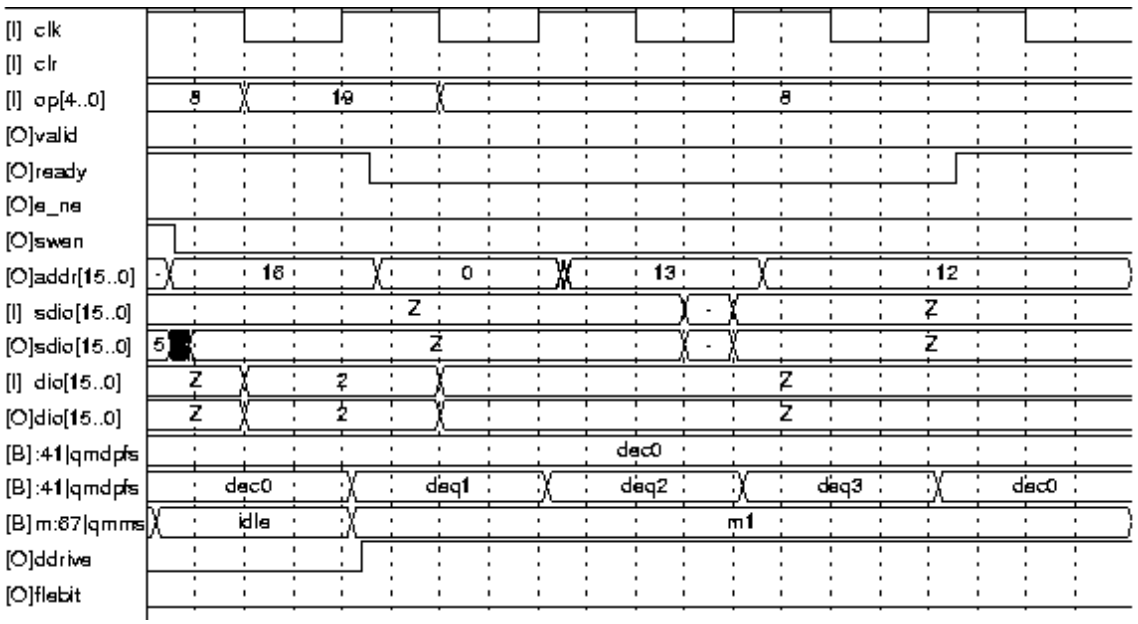
Κύκλος 6(β): Υπολογίζουμε την διεύθυνση του δείκτη κεφαλής, δηλαδή (2*Qn + BAR_HT) και διαβάζουμε το κορυφαίο στοιχείο της ουράς.

Κύκλος 7(β): Φέρνουμε το δεύτερο όρισμα της εντολής που δόθηκε στον κύκλο 6(β) (αν υπάρχει) και γράφουμε στην διεύθυνση που υπολογίσαμε στον κύκλο 6(β) τα δεδομένα που διαβάσαμε στον ίδιο κύκλο.

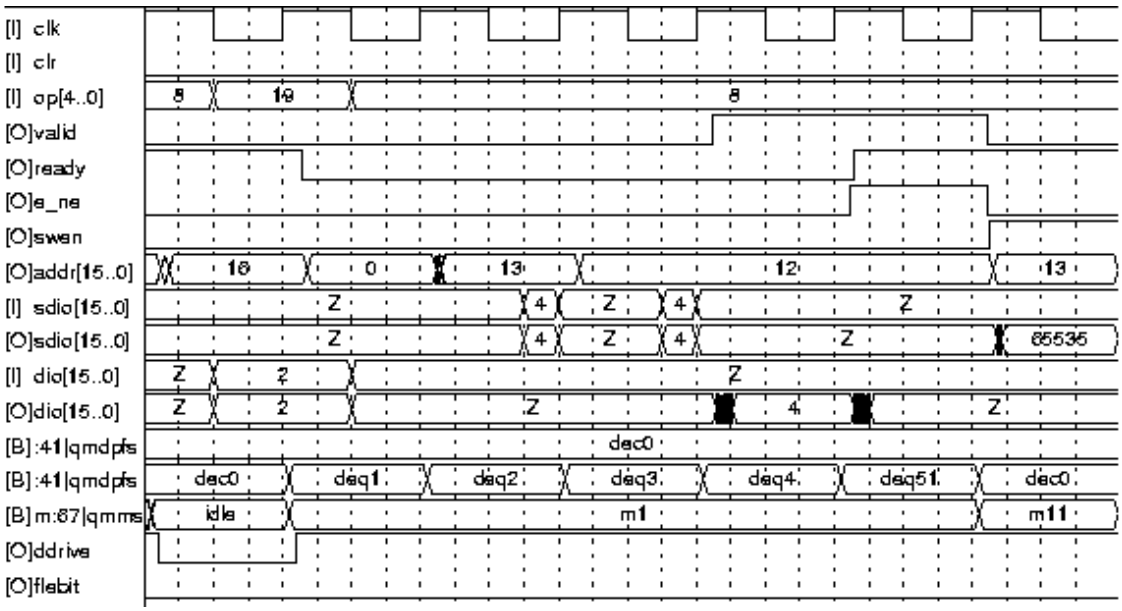
Παρατήρηση: Παρατηρείστε ότι η εντολή DeQ σε μια περίπτωση διαβάζει κάτι από την μνήμη (κύκλος 6(β)) το οποίο γράφει στον αμέσως επόμενο κύκλο (κύκλος 7(β)). Αυτή

η κατάσταση δεν δημιουργεί πρόβλημα, γιατί ό,τι διαβάζεται στον κύκλο 6(β) αποθηκεύται σε ένα ακμοπυρόδοτο καταχωρητή που συνδέεται απευθείας με τα pin Sdata. Το μονοπάτι από τα pin Sdata (από τα οποία διαβάζουμε) προς τον καταχωρητή που τα αποθηκεύει (SData στο Σχήμα A.32) είναι κρίσιμο (critical path). Η κατάσταση που δημιουργεί πραγματικά πρόβλημα είναι αν διαβάσουμε μια λέξη Α από την μνήμη και την χρησιμοποιήσουμε σαν διεύθυνση στον αμέσως επόμενο κύκλο. Το πρόβλημα έγκειται στο ότι στην περίπτωσή μας πρέπει να υπολογίσουμε την τελική διεύθυνση βάσει του Α και αυτό μπορεί να συμβεί σε ένα κύκλο ταυτόχρονα με την ανάγνωση. Και οι 2 αυτές συνθήκες ελήφθησαν υπόψιν κατά τον προγραμματισμό των μικροεντολών της εντολής DeQ όπου και συμβαίνουν αυτά τα φαινόμενα εξασφαλίζοντας την δυνατότητα χρήσης πιο αργών εξωτερικών μνημών (σε σχέση με το αν δεν λαμβάναμε υπόψιν αυτούς τους παράγοντες).

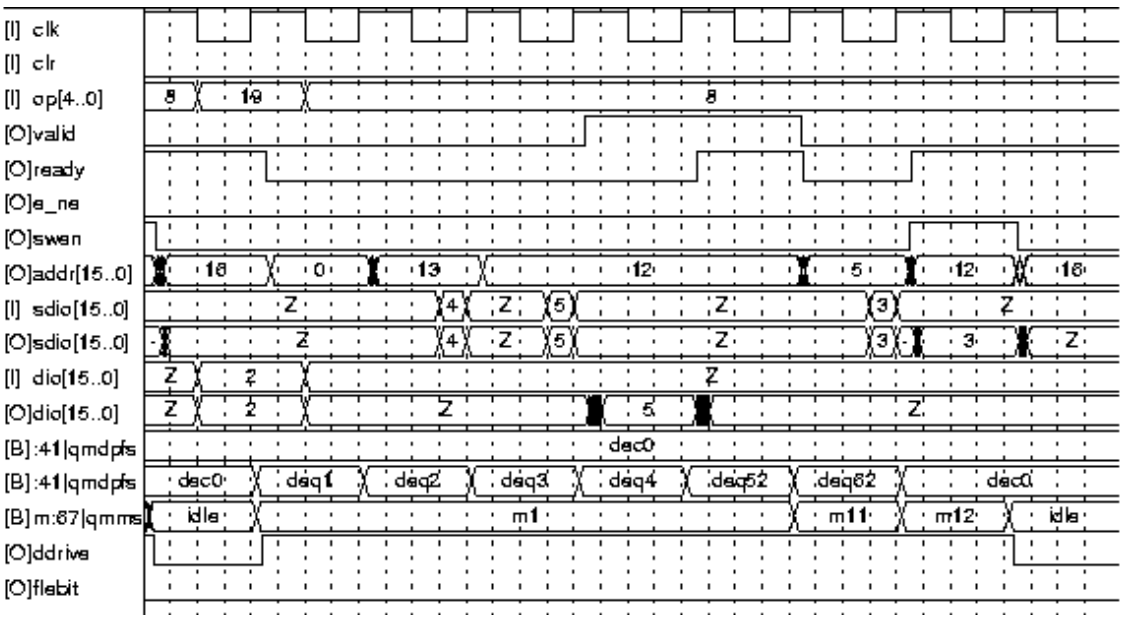
Ο χρονισμός των 3 διαφορετικών περιπτώσεων της εντολής DeQ φαίνεται στα Σχήματα A.11, A.12 και A.13.



Σχήμα A.11: Εντολή DeQ σε άδεια ουρά



Σχήμα Α.12: Εντολή DeQ σε ουρά με ένα μόνο στοιχείο



Σχήμα Α.13: Εντολή DeQ σε ουρά με περισσότερα από ένα στοιχεία

A.3 Η ΕΝΤΟΛΗ GETFREE

Η εντολή GetFree δεν χρειάζεται ορίσματα και επιστρέφει μια διεύθυνση σε ένα ελεύθερο χώρο μνήμης. Ο κώδικας C που την υλοποιεί παρουσιάζεται στο Σχήμα A.14.

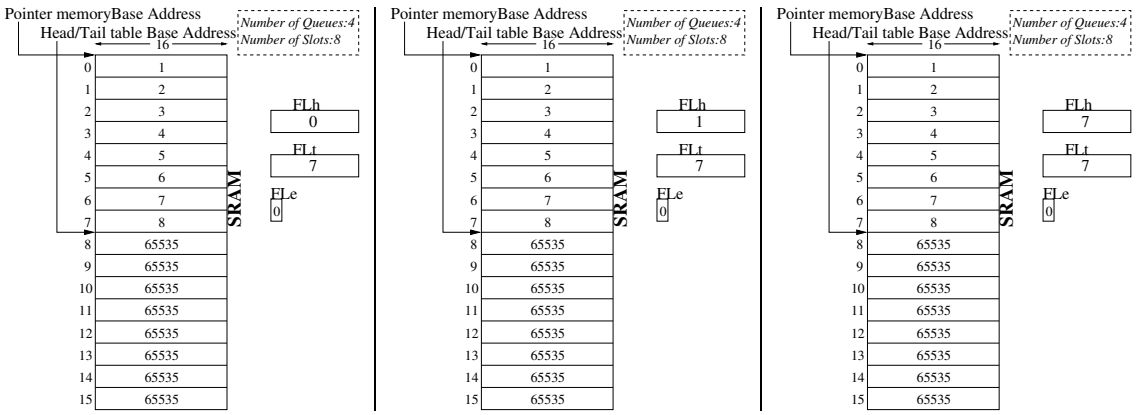
```
unsigned int
GetFree(unsigned int *FLh, unsigned int *FLt, unsigned char *FLe,
    unsigned int *PMm, unsigned int *valid)
{
  unsigned int pPM, ret_val;

  *valid = 0;
   if (*FLe==1)
   {
    ret_val=0;
   }
   else
   {
    ret_val=*FLh;
    *valid = 1;
    if (*FLh==*FLt)
     *FLe=1;
    else
    {
     pPM=PMm[*FLhead];
     *FLh=pPM;
    }
   }
   return ret_val;
}
```

Σχήμα A.14: Κώδικας C της εντολής GetFree

Ένα παράδειγμα των 2 περιπτώσεων που η Free List είναι μη άδεια φαίνεται στο Σχήμα A.15. Ο προγραμματισμός των μικροεντολών παρουσιάζεται στο σχήμα A.16.

Η ακριβής εκτέλεση των μικροεντολών σε κύκλους ρολογιού είναι η εξής:

Κύκλος 0: Σύγκριση των καταχωρητών Free List Head και Free List Tail. Αν το Free List Empty Bit είναι 1 τότε η FSM δεν αλλάζει κατάσταση. Επιπλέον, υπολογίζουμε την διεύθυνση του στοιχείου κορυφής της ουράς (Free List Head element).



Σχήμα Α.15: Αρχική κατάσταση των ουρών, GetFree, GetFree

Κύκλος 1(α): *Αν Free List Head == Free List Tail* τότε Data = Free List Head, Valid = 1 και Free List Empty Bit = 1.

Κύκλος 1(β): *Αν Free List Head != Free List Tail* τότε Data = Free List Head και Valid = 1. Επίσης διαβάζουμε από την διεύθυνση που υπολογίσαμε στον κύκλο 0 το στοιχείο κορυφής και το γράφουμε στον Free List Head.

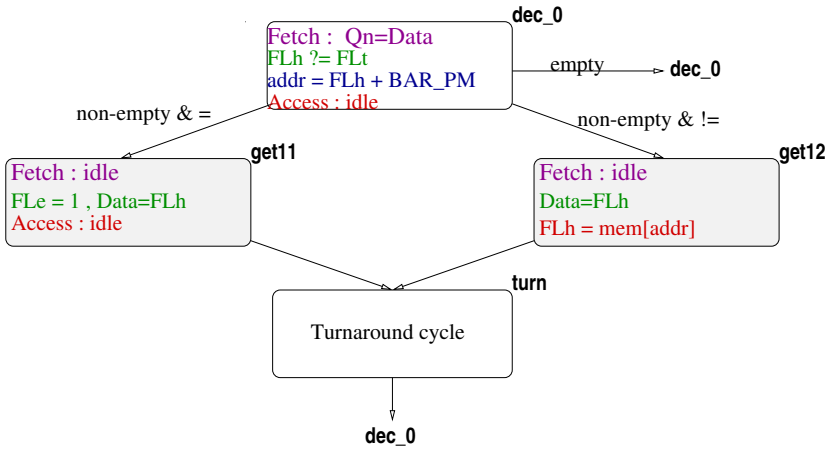
Κύκλος 2: Κύκλος turnaround (ο χρόνος μεταξύ δύο διαδοχικών προσπελάσεων των σημάτων Data από διαφορετικούς οδηγητές είναι ένας κύκλος, για ομαλή λειτουργία του κυκλώματος).

Οι χρονισμοί της εντολής στις 3 διαφορετικές περιπτώσεις παρουσιάζονται στα Σχήματα Α.17, Α.18, Α.19.

Α.4 Η ΕΝΤΟΛΗ RETFREE

Η εντολή RetFree δέχεται σαν όρισμα μια διεύθυνση σε ένα ελεύθερο χώρο μνήμης και την επιστρέφει εσωτερικά στην Free List. Ο κώδικας C που την υλοποιεί παρουσιάζεται στο Σχήμα Α.20.

Ένα παράδειγμα των 2 περιπτώσεων που η Free List είναι άδεια και μη άδεια παρουσιάζεται στο Σχήμα Α.21. Ο προγραμματισμός των μικροεντολών φαίνεται στο σχήμα Α.22 και ακολουθεί η ανάλυσή του:



Σχήμα A.16: Προγραμματισμός μικροεντολών για την εντολή GetFree

Κύκλος 0: Αποθήκευση (latch) του πρώτου ορίσματος που είναι μια διεύθυνση σε ελεύθερο χώρο μνήμης (Slotn).

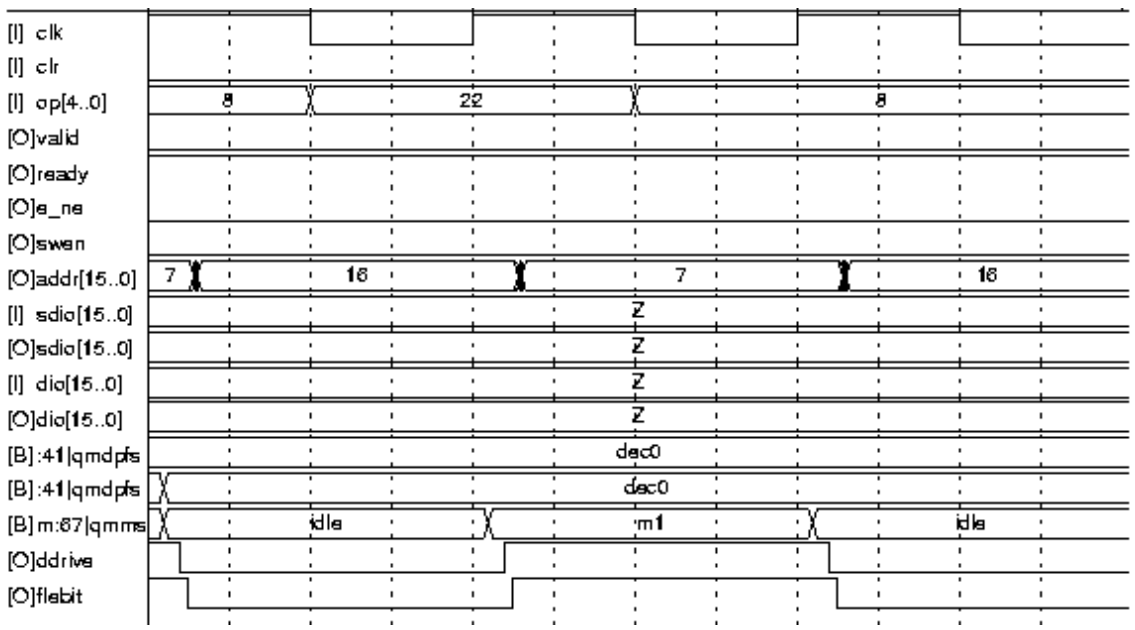
Κύκλος 1(α): *Αν Free List Empty Bit = 1* τότε γίνεται 0 και Free List Head = Free List Tail = Slotn. Μια νέα εντολή μαζί με το πρώτο της όρισμα μπορεί να δοθεί σε αυτό τον κύκλο.

Κύκλος 2(α): Υπολογίζεται η διεύθυνση προσπέλασης της μνήμης της εντολής που δόθηκε στον κύκλο 1(α).

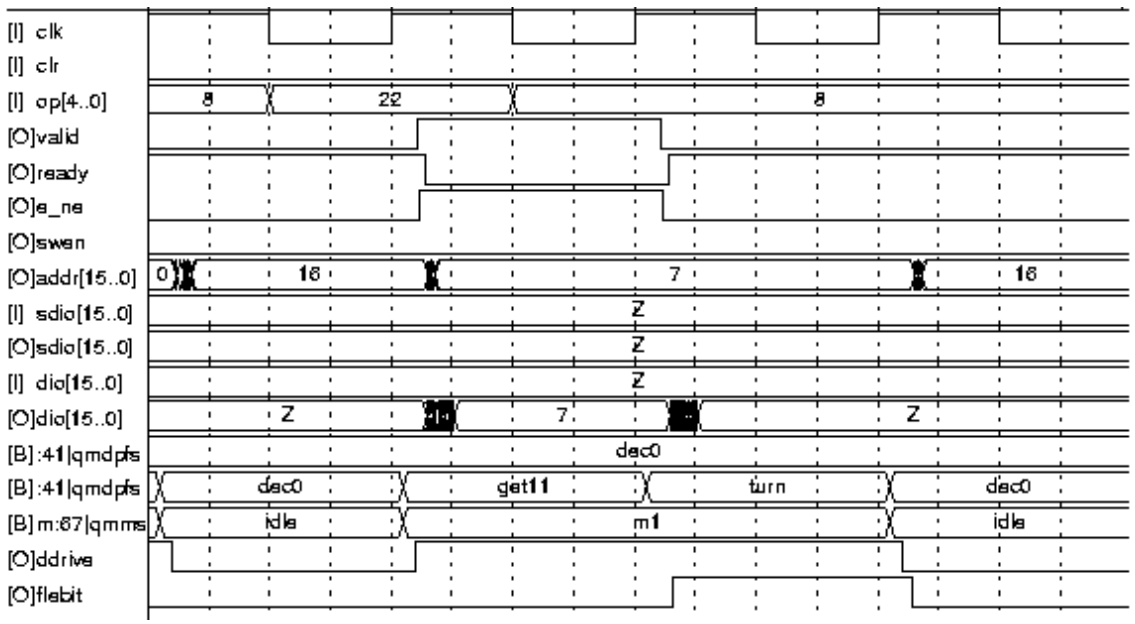
Κύκλος 1(β): *Αν Free List Empty Bit = 0* τότε υπολογίζουμε την διεύθυνση του στοιχείου τέλους (tail element) της ουράς BAR_PM + Slotn και σαν δεδομένα της επόμενης εγγραφής το Slotn.

Κύκλος 2(β): Υπολογίζεται η διεύθυνση προσπέλασης της μνήμης της εντολής που δόθηκε στον κύκλο 1(α) και γράφουμε στην μνήμη στην διεύθυνση που υπολογίσαμε στον κύκλο 1(β) το Slotn.

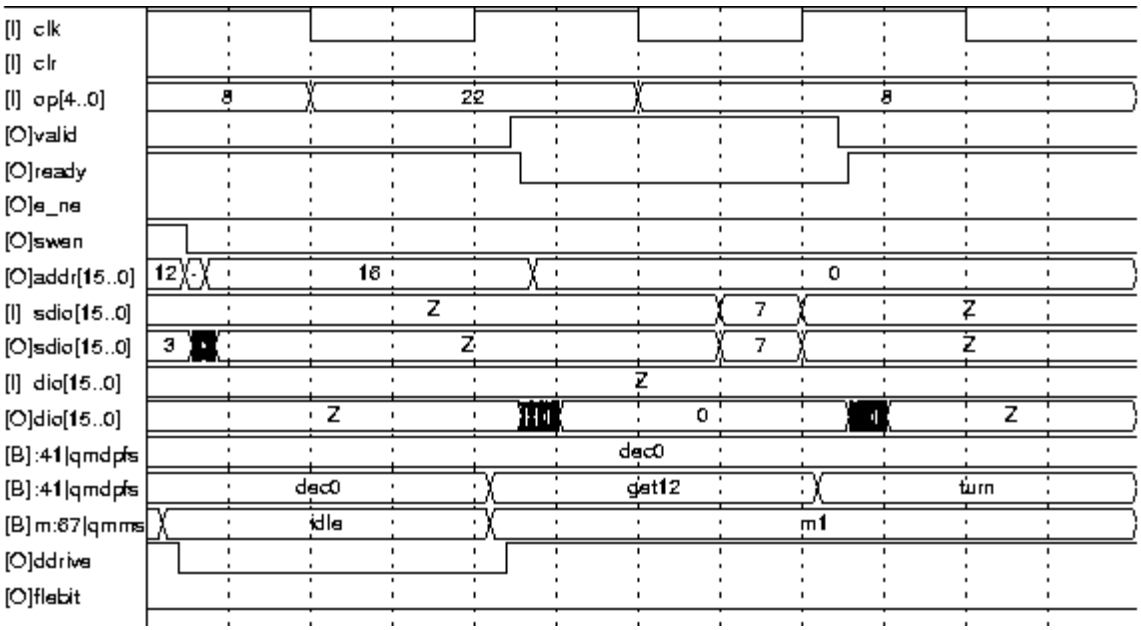
Οι χρονισμοί των σημάτων της Retfree στις δύο διαφορετικές περιπτώσεις παρουσιάζονται στα Σχήματα A.23 και A.24.



Σχήμα Α.17: Εντολή GetFree σε άδεια ουρά



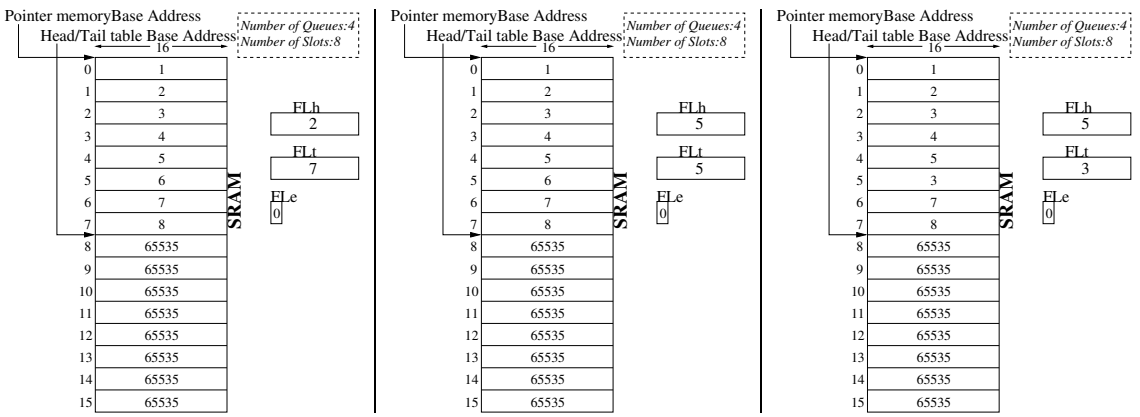
Σχήμα Α.18: Εντολή GetFree σε ουρά με ακριβώς ένα στοιχείο



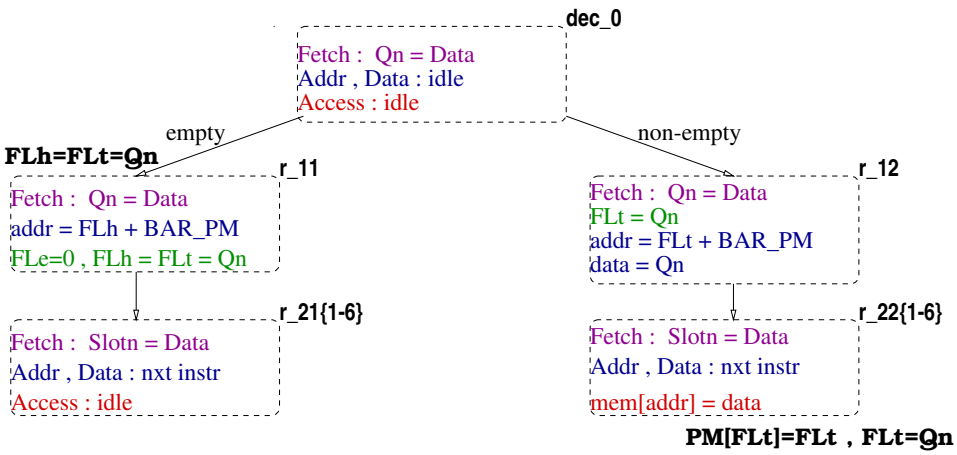
Σχήμα A.19: Εντολή GetFree σε ουρά με περισσότερα από ένα στοιχεία

```
unsigned int
RetFree(unsigned int *FLh, unsigned int *FLt, unsigned char *FLe,
    unsigned int *PMm, unsigned int Slotn)
{
 unsigned int ret_val;

 if (*FLe)
  {
   *FLh=*FLt=Slotn;
   *FLe=0;
   ret_val = 0;
  }
 else
  {
   ret_val = 1;
   PMm[*FLtail]=Slotn;
   *FLt=Slotn;
  }
 return ret_val;
}
```

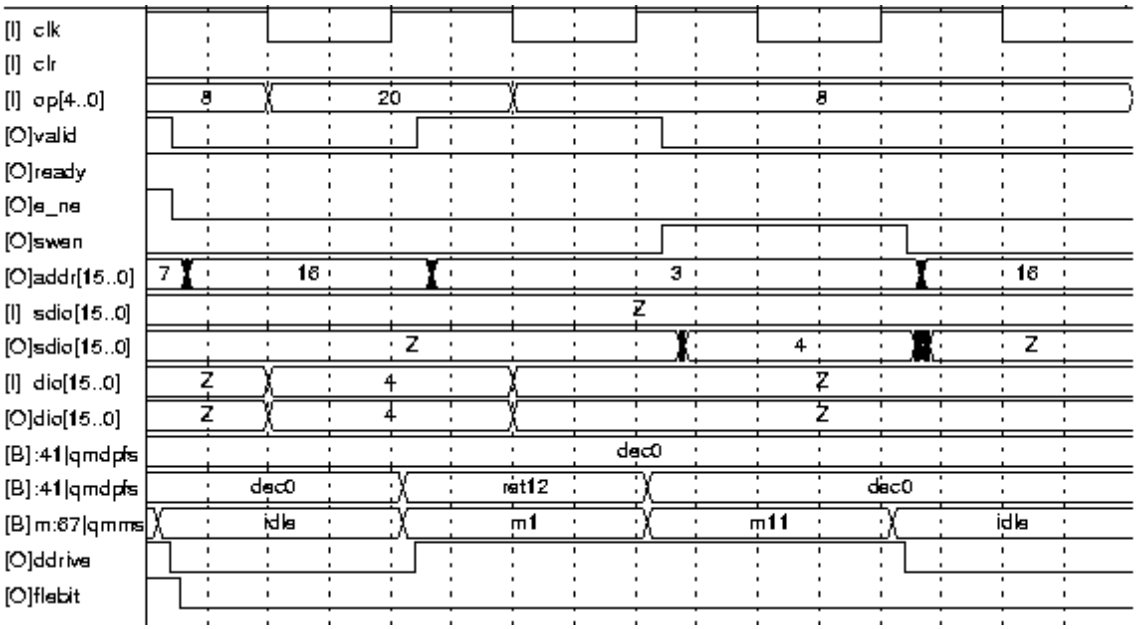
Σχήμα Α.20: Κώδικας C της εντολής RetFree



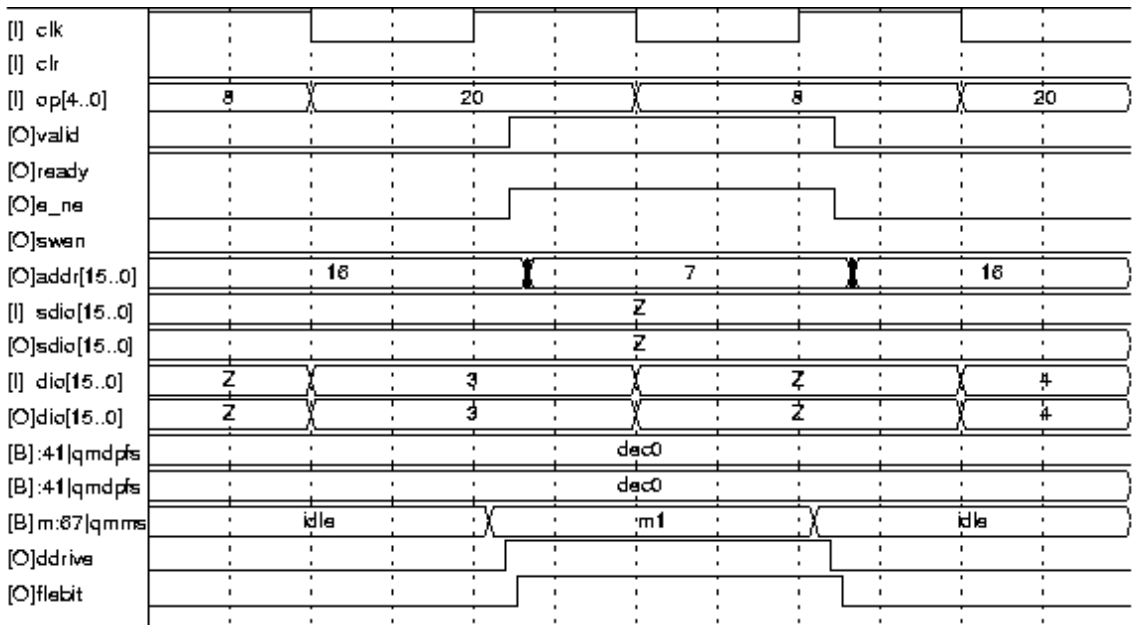
Σχήμα Α.21: Αρχική κατάσταση των ουρών, RetFree, RetFree



Σχήμα A.22: Προγραμματισμός μικροεντολών για την εντολή RetFree



Σχήμα A.23: Εντολή RetFree σε μη άδεια ουρά

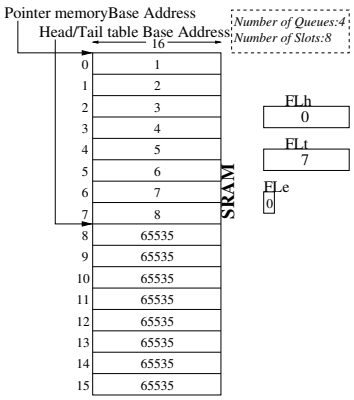


Σχήμα Α.24: Εντολή RetFree σε άδεια ουρά

Α.5 Η ΕΝΤΟΛΗ INIT

Η διαδικασία αρχικοποίησης θέτει όλες τις δομές στις κατάλληλες αρχικές τιμές. Η εντολή Init, δέχεται 4 ορίσματα ξεκινώντας από τον 2ο κύκλο: την διεύθυνση βάσης της Pointer memory καθώς και το μέγεθος αυτής και την διεύθυνση βάσης του Head/Tail table και το μέγεθός του. Δεν διατίθεται υλικό που να ελέγχει αν οι δοθείσες τιμές είναι σωστές (δηλαδή να μην υπάρχει επικάλυψη). Η εντολή Init χρειάζεται πολλούς κύκλους για να εκτελεστεί και πιο συγκεκριμένα $2N_q + N_c + 3$, όπου N_q είναι ο αριθμός των ουρών και N_c ο αριθμός των διαθέσιμων χώρων μνήμης (buffers). Ουσιαστικά γράφει ολόκληρη την Pointer Memory και στην θέση i γράφει την τιμή (i+1). Επίσης διασχίζει και την μνήμη Pointer memory γράφοντας στο πιο σημαντικό ψηφίο (δηλαδή το Empty Bit) την τιμή 1, ώστε όλες οι ουρές να είναι αρχικά άδειες. Τέλος ο καταχωρητής Free List Tail δείχνει στο τελευταίο στοιχείο της Pointer memory και ο Free List Head στο πρώτο. Το Free List Empty Bit τίθεται στο 0. Στο Σχήμα Α.25 παρουσιάζεται ένα παράδειγμα αρχικοποίησης

για 4 ουρές και 8 χώρους μνήμης (slots). Τα διαγράμματα χρονισμού της εντολής φαίνονται στα Σχήματα A.26, A.27 και A.28.



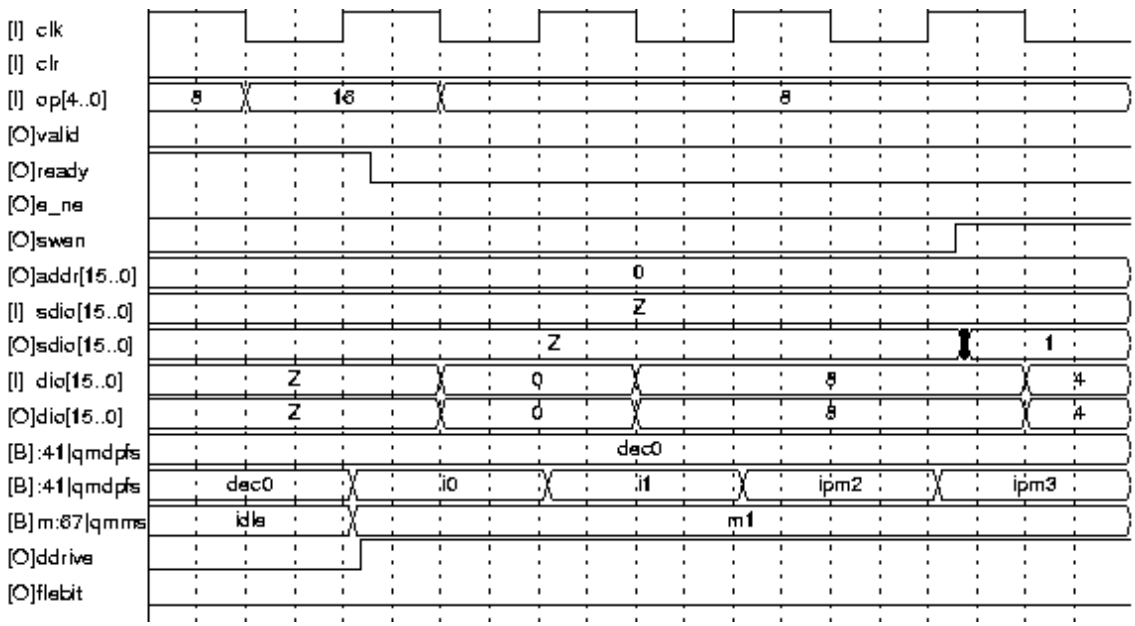
Σχήμα A.25: Παράδειγμα αρχικοποίησης για 4 ουρές και 8 θέσεις μνήμης (slots)

A.6 Η ΕΝΤΟΛΗ READ

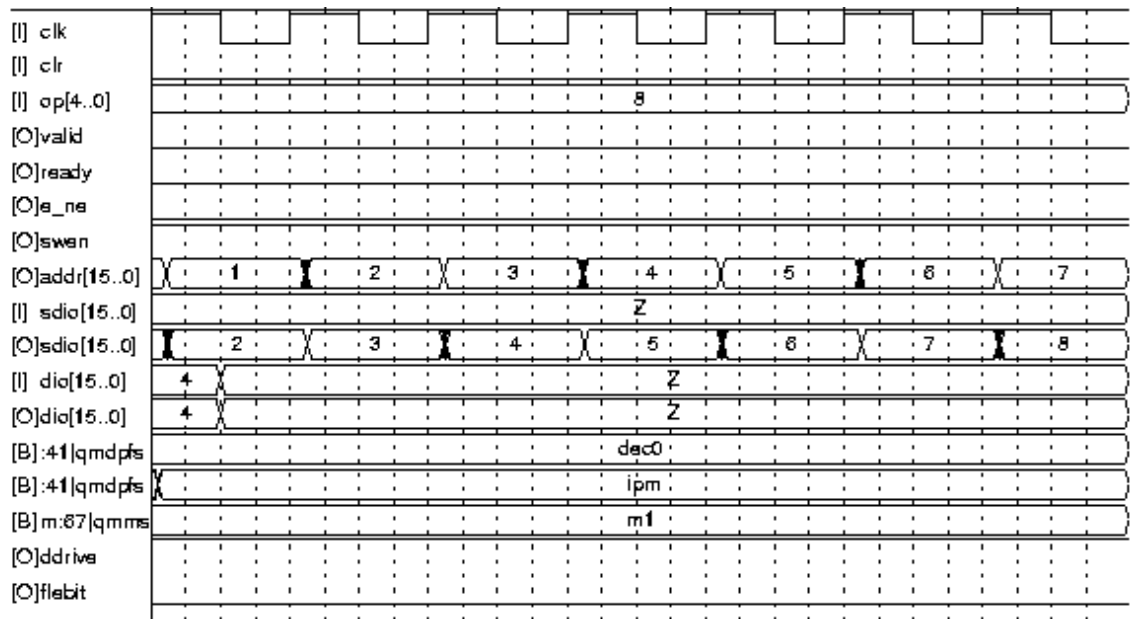
Η εντολή Read δέχεται ένα όρισμα, την διεύθυνση ανάγνωσης στην στατική μνήμη. Δεν έχει γίνει κάποια βελτιστοποίηση για αυτή την εντολή. Το διάγραμμα χρονισμού της φαίνεται στο Σχήμα A.29.

A.7 Η ΕΝΤΟΛΗ WRITE

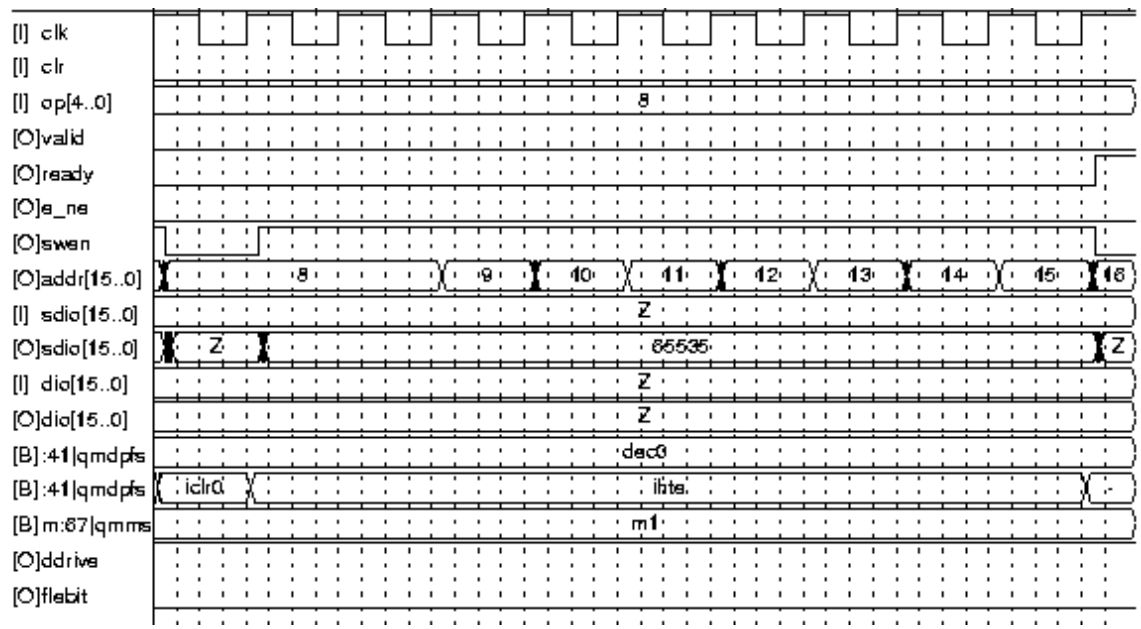
Η εντολή Write, δέχεται δύο ορίσματα: την διεύθυνση και τα δεδομένα εγγραφής στην στατική μνήμη. Χρησιμοποιείται μόνο για διόρθωση σφαλμάτων και δεν έχει βελτιστοποιήσεις. Το διάγραμμα χρονισμού της φαίνεται στο Σχήμα A.30.



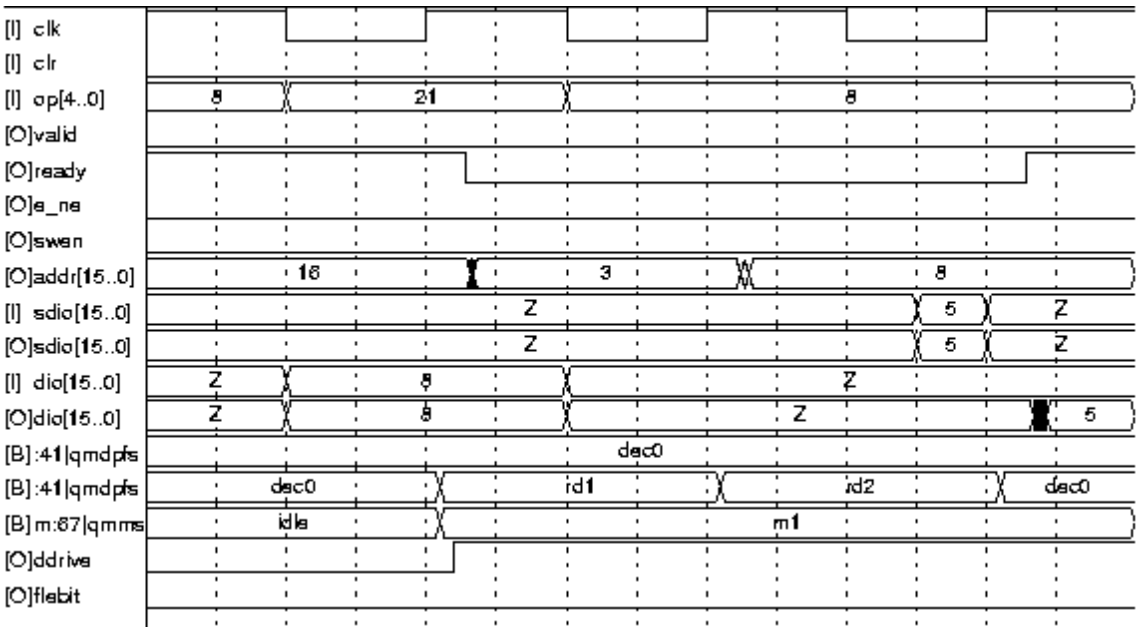
Σχήμα Α.26: Διάγραμμα χρονισμού Ι της εντολής Init



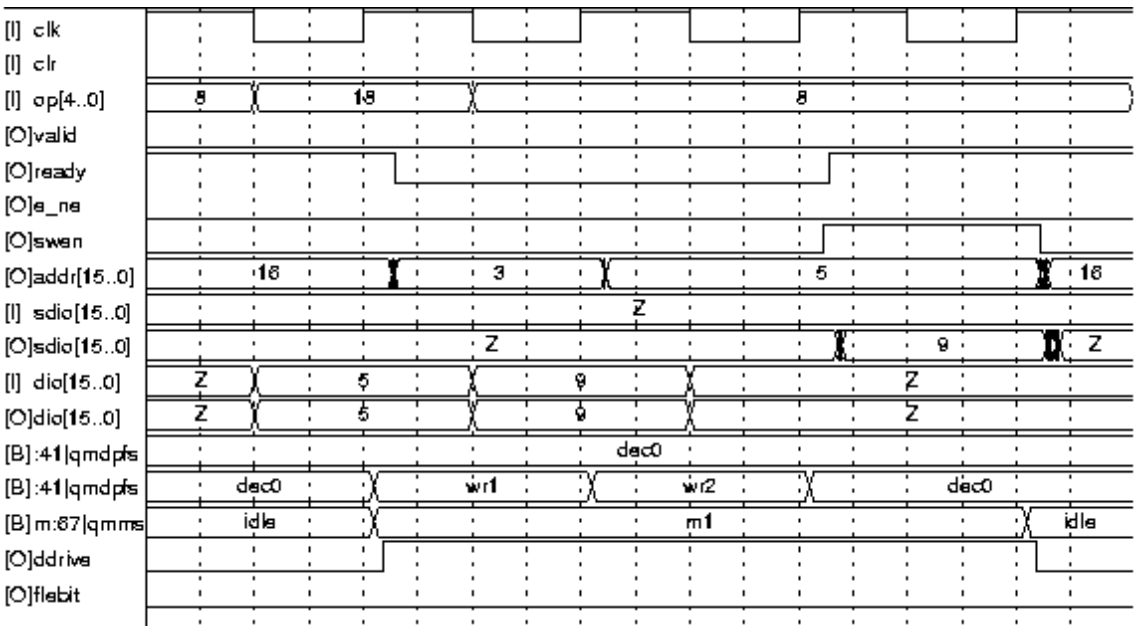
Σχήμα Α.27: Διάγραμμα χρονισμού ΙΙ της εντολής Init



Σχήμα A.28: Διάγραμμα χρονισμού ΙΙΙ της εντολής Init



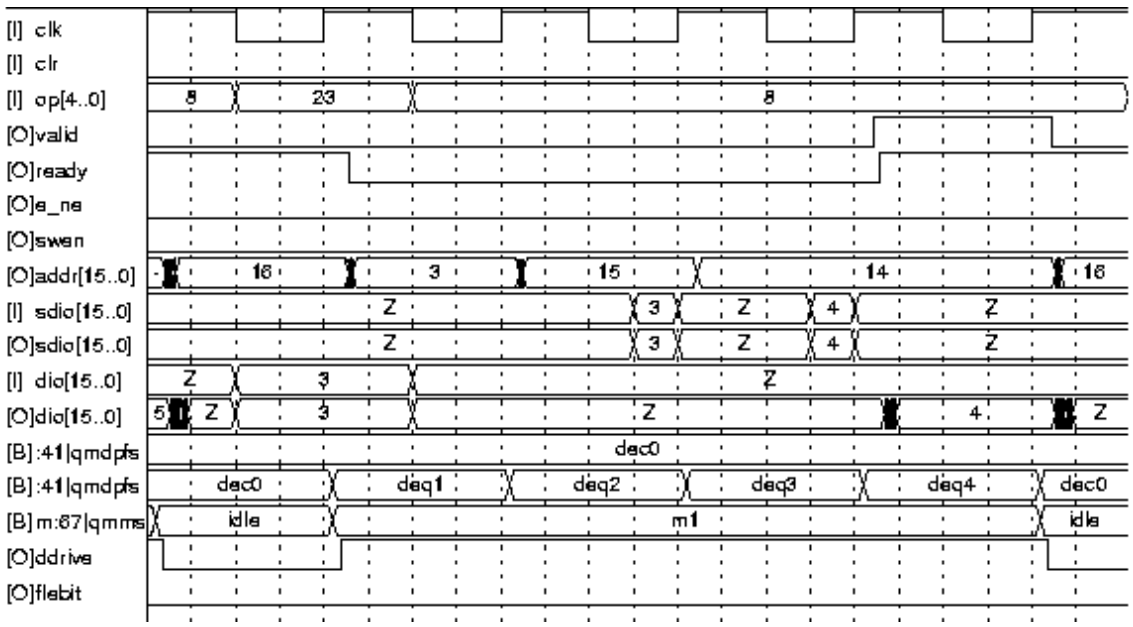
Σχήμα A.29: Διάγραμμα χρονισμού της εντολής Read



Σχήμα Α.30: Διάγραμμα χρονισμού της εντολής Write

A.8 Η ΕΝΤΟΛΗ TOP

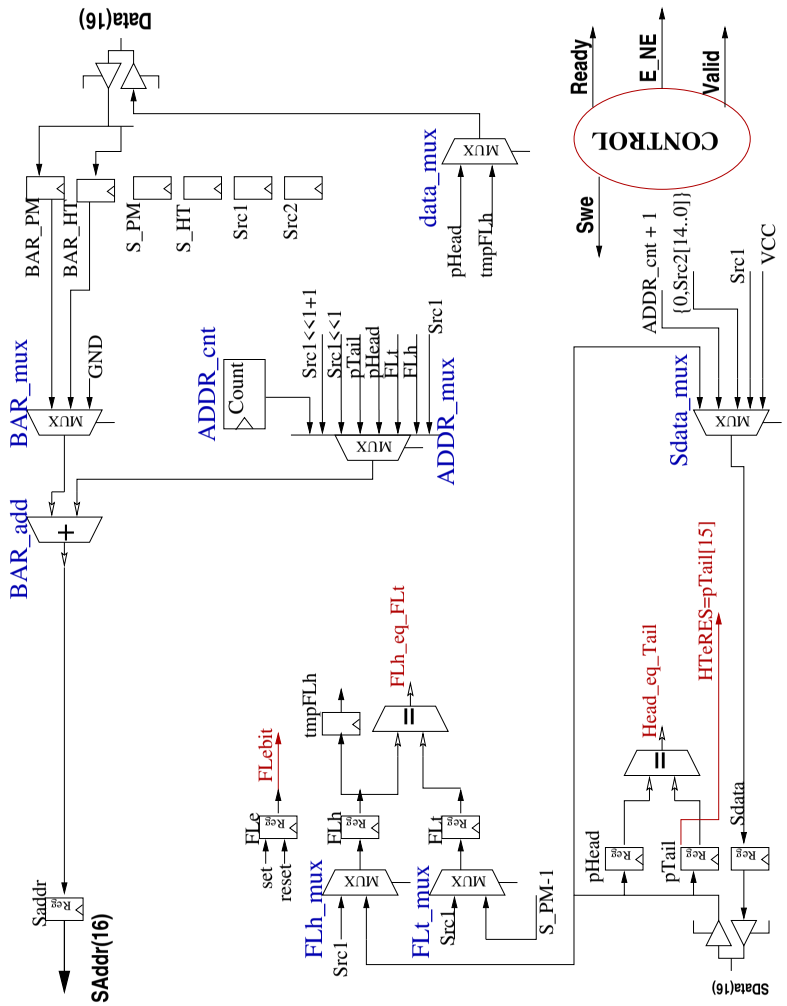
Η εντολή Top, είναι ίδια με την εντολή EnQ για τους 5 πρώτους κύκλους και δεν έχει δυνατότητα επικάλυψης με άλλες εντολές. Το διάγραμμα χρονισμού της φαίνεται στο Σχήμα A.31.



Σχήμα A.31: Διάγραμμα χρονισμού της εντολής Top

A.9 ΤΟ ΜΟΝΟΠΑΤΙ ΔΕΔΟΜΕΝΩΝ ΤΟΥ ΔΙΑΧΕΙΡΙΣΤΗ ΟΥΡΩΝ ΤΟΥ MuQPro I

Το μονοπάτι δεδομένων του QM παρουσιάζεται στο Σχήμα A.32. Το κρίσιμο μονοπάτι ξεκινά από τους καταχωρητές Src1, Src2 και μέσω του μεγάλου πολυπλέκτη ADDR_mux και του αθροιστή BAR_add καταλήγει στον καταχωρητή Saddr που παρέχει την διεύθυνση της στατικής μνήμης. Επίσης κρίσιμο είναι το μονοπάτι από την μονάδα ελέγχου προς το μονοπάτι δεδομένων, δηλαδή το μονοπάτι που ξεκινά από την μονάδα ελέγχου και καταλήγει στους πολυπλέκτες και τους καταχωρητές του μονοπατιού δεδομένων.

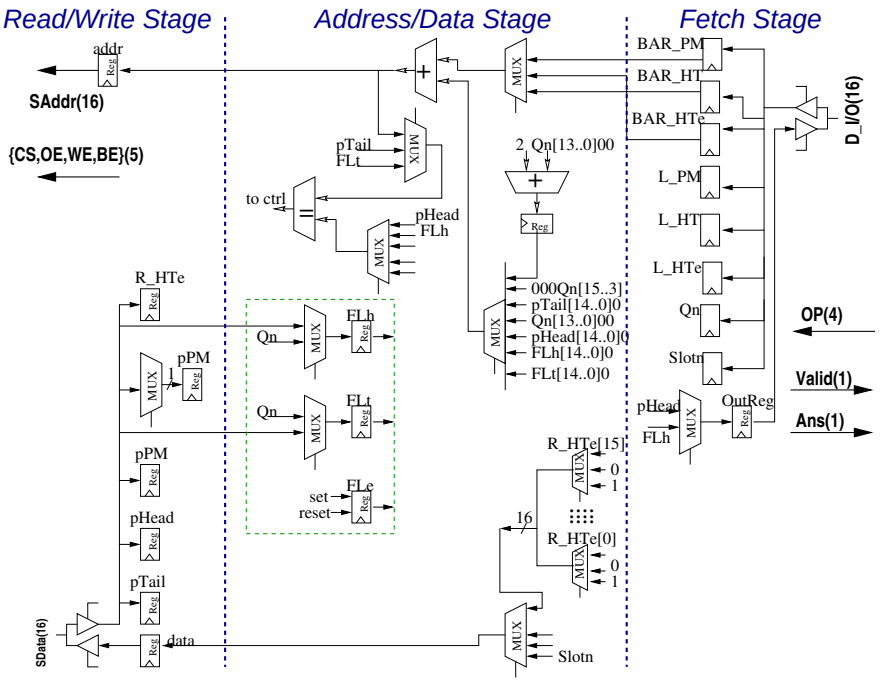


Σχήμα Α.32: Το μονοπάτι δεδομένων του Διαχειριστή Ουρών του MuQPro I

ΠΑΡΑΡΤΗΜΑ Β

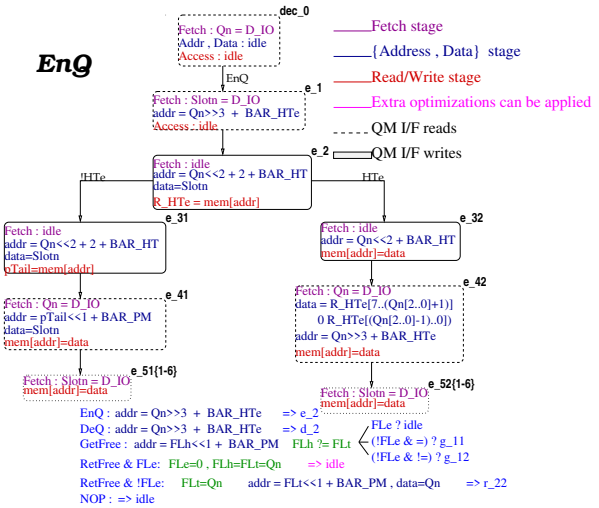
Η ‘‘ΠΛΗΡΩΣ ΕΠΕΚΤΑΣΙΜΗ’’ ΥΛΟΠΟΙΗΣΗ

Β.1 ΤΟ ΜΟΝΟΠΑΤΙ ΔΕΔΟΜΕΝΩΝ

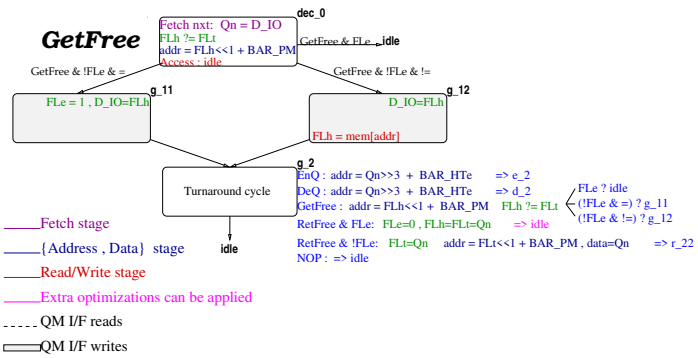


Σχήμα Β.1: Το μονοπάτι δεδομένων της ‘‘πλήρως επεκτάσιμης’’ υλοποίησης

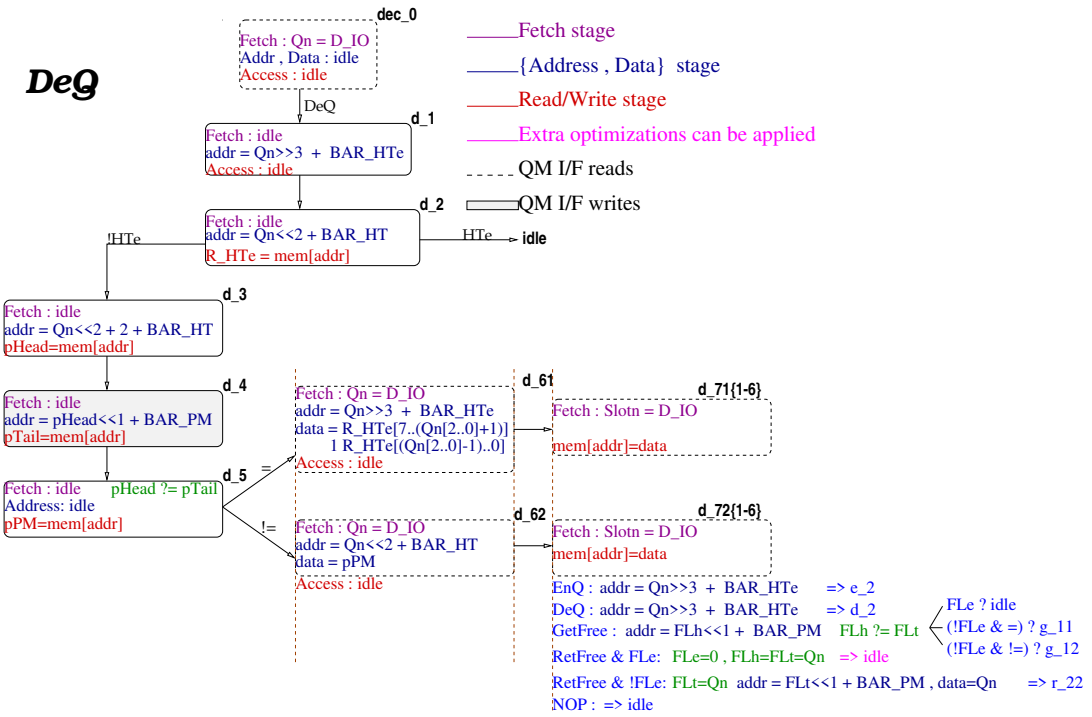
Β.2 Η ΜΗΧΑΝΗ ΠΕΠΕΡΑΣΜΕΝΩΝ ΚΑΤΑΣΤΑΣΕΩΝ



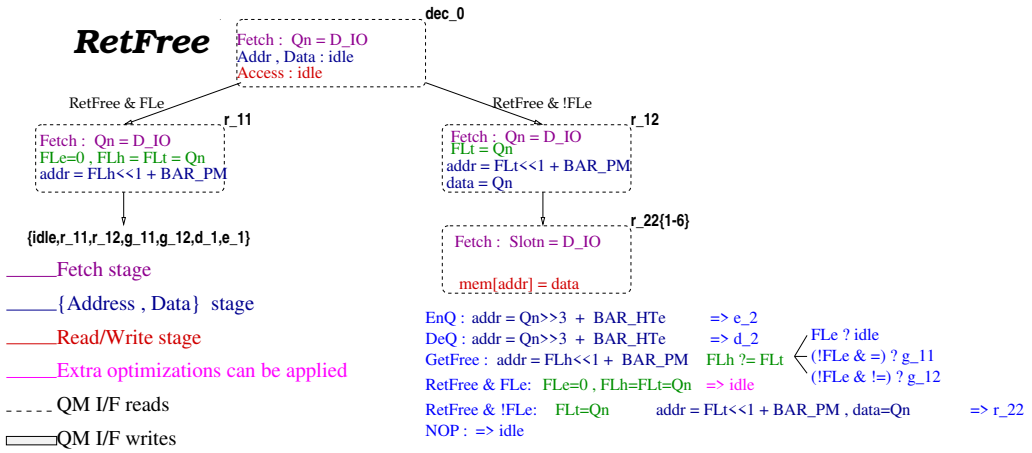
Σχήμα Β.2: Προγραμματισμός μικροεντολών της εντολής EnQ



Σχήμα Β.3: Προγραμματισμός μικροεντολών της εντολής GetFree



Σχήμα B.4: Προγραμματισμός μικροεντολών της εντολής DeQ

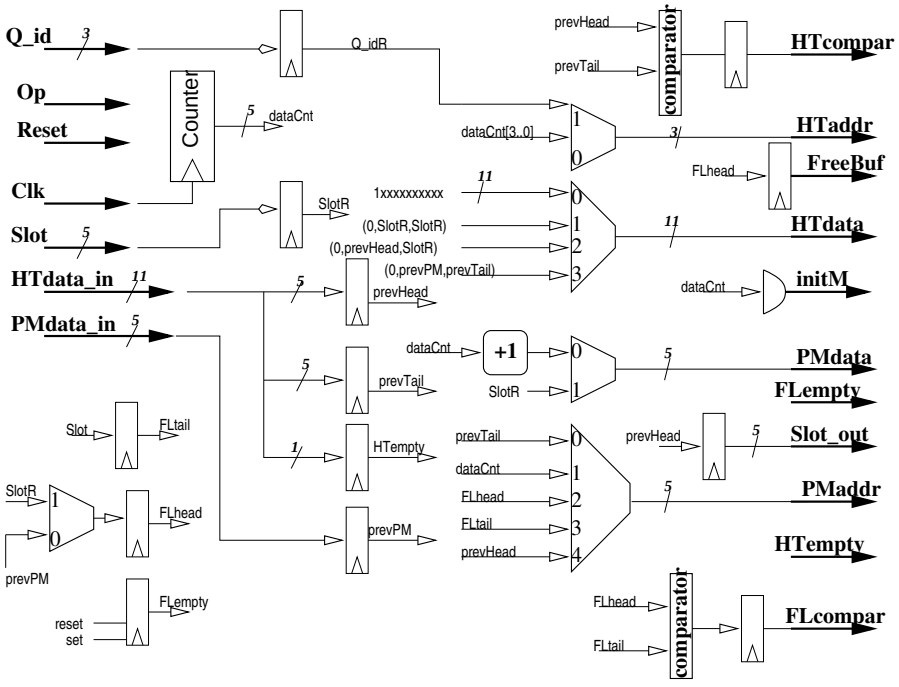


Σχήμα B.5: Προγραμματισμός μικροεντολών της εντολής RetFree

ΠΑΡΑΡΤΗΜΑ C

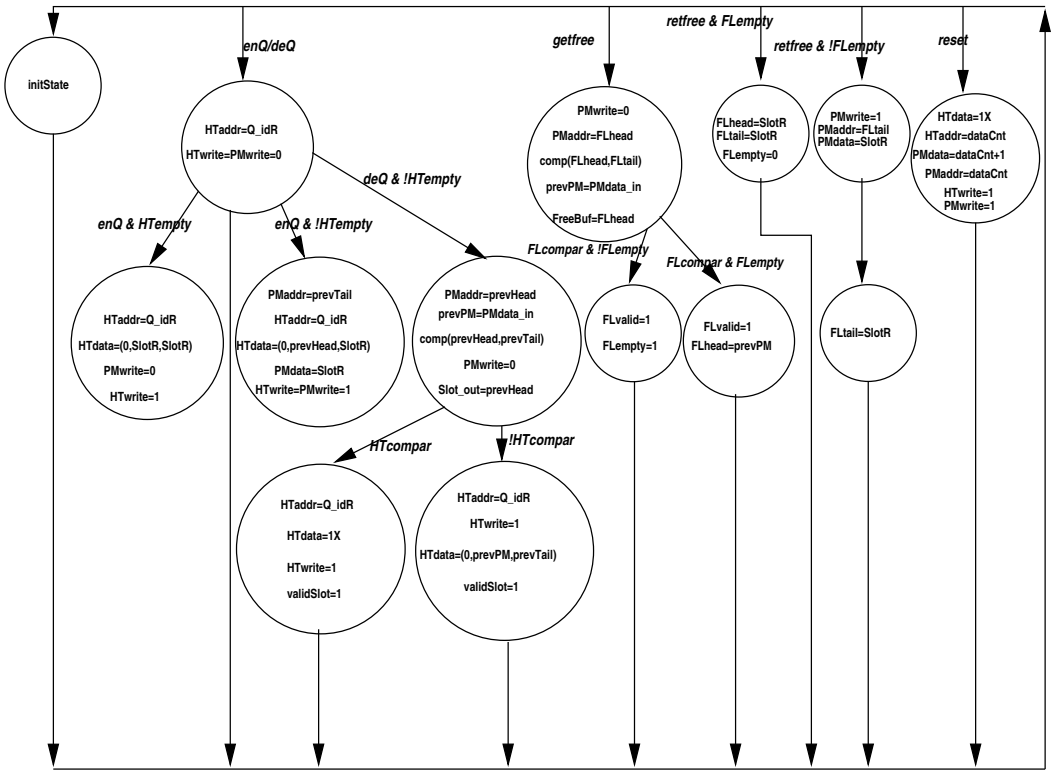
Η ‘‘ΠΛΗΡΩΣ ΠΑΡΑΛΛΗΛΗ’’ ΥΛΟΠΟΙΗΣΗ

C.1 ΤΟ ΜΟΝΟΠΑΤΙ ΔΕΔΟΜΕΝΩΝ



Σχήμα C.1: Το μονοπάτι δεδομένων της ‘‘πλήρως παράλληλης’’ υλοποίησης

C.2 Η ΜΗΧΑΝΗ ΠΕΠΕΡΑΣΜΕΝΩΝ ΚΑΤΑΣΤΑΣΕΩΝ



Σχήμα C.2: Η Μηχανή Πεπερασμένων Καταστάσεων της ''πλήρως παράλληλης'' υλοποίησης

Βιβλιογραφία

[1] D. Serpanos, ”Communication Subsystems for High-speed networks: ATM requirements”, In Asynchronous Transfer Mode, Proceedings of TRICOMM’93, Raleigh, North Carolina, pp. 31-38, April 26-27, 1993.

[2] H.E. Meleis and D.N. Serpanos, ”Designing Communication Subsystems for High-Speed Networks”, IEEE Network Magazine, pp. 40-46, July 1992.

[3] M. Katevenis, D. Serpanos and E. Markatos, ”Multi-Queue Management and Scheduling for improved QoS in Communication Networks”, Proceedings of the European Multimedia Microprocessor Systems and Electronic Commerce (EMMSEC’97), Florence, Italy, November 1997.

[4] G. Kornaros, C. Kozyrakis, P. Vatsolaki and M. Katevenis, ”Pipelined Multi-Queue Management in a VLSI ATM Switch Chip with Credit-Based Flow Control”, Proceedings of the 17th Conference on Advanced Research in VLSI (ARVLSI’97), pp. 127-144, Univ. of Michigan, Ann Arbor, USA, Sept. 1997.

[5] Flavio Bonomi and Kerry W. Fendick, ” The Rate-Based Flow Control Framework for the Available Bit Rate ATM Service”, IEEE Network Magazine, Vol. 9, No. 2, March/April 1995, pp. 25-39.

[6] Raj Jain, ”Congestion Control and Traffic Management in ATM Networks: Recent Advances and A Survey”, Proceedings of the 4th Int. Symposium on High-Performance Computer Architecture (HPCA-4),

[7] M. Katevenis, D. Serpanos and E. Spyridakis, ”Credit-Flow-Controlled ATM for MP Interconnection:the ATLAS I Single-Chip ATM Switch”, Proceedings of the 4th Int. Symposium on High-Performance Computer Architecture (HPCA-4), pp. 47-56, Las Vegas, Nevada USA, February 1998.

[8] Kung and R.Morris, ”Credit-Based Flow Control for ATM Networks”, IEEE Network Magazine, Vol. 9, No. 2, March/April 1995, pp. 40-48.

[9] IBM Corporation, ”Algorithm for Managing multiple First-in, First-out Queues from a single shared random-access memory”, IBM Technical Disclosure Bulletin: Vol.32, No 3B, August 1989.

[10] SUN Microelectronics, ”ATM622-s single-chip ATM SAR”, Data Sheet, July 1997.

[11] FORE Systems, ”ForeRunner ASX-200BX and ASX-1000”, 1998.

[12] ALTERA, ”MAX+PLUS II Getting Started”.

[13] ALTERA, ”AHDL Manual”.

[14] John David Carnaugh and Timothy J. Salo, ”Internetworking with ATM WANs”, Minnesota Supercomputer Center Inc, December 1992.

[15] M. Batubara and A. J.Mc Gregor, ”An Introduction to B-ISDN and ATM”, MONASH-TR 93/14, September 1993.

[16] C.Kosak, D.Eckhardt, T. Mummert, P.Steenkiste, A. Fisher, ”Buffer Management and Flow Control in the Credit Net ATM Host Interface”, School of Computer Science - Carnegie Mellon University.

[17] C. Brendan and S. Traw, ”Hardware/Software Organization of a High-Performance ATM Host Interface”, IEEE Journal on Selected Areas in Communications, Vol 11, No 2, February 1993.

[18] K.K. Ramakrishnan , ”Performance Considerations in Designing Network Interfaces”, IEEE Journal on Selected Areas in Communications, Vol 11, No 2, February 1993.

[19] Tim Moors and Antonio Cantoni, ”ATM Receiver Implementation Issues”, IEEE Journal on Selected Areas in Communications, Vol 11, No 2, February 1993.

[20] Romanow, A., and Floyd, S., ”Dynamics of TCP Traffic over ATM Networks”, IEEE Journal on Selected Areas in Communications, Vol 13, No 4, May 1995, pp. 633-641

[21] Kenji Kawahara, Kouichirou Kitajima, Tetsuya Takine, and Yuji Oie, ”Packet loss performance of selective cell discard schemes in ATM networks”, IEEE Journal on Selected Areas in Communications, Vol 15, No 5, June 1997, pp. 903-913

[22] Casoni, M., and Turner, J. S., ”On the Performance of Early Packet Discard”, IEEE Journal on Selected Areas in Communications, Vol 15, No 5, June 1997.

[23] ATM Forum, ”ATM User-Network Interface Specification V3.1”, 1994

[24] ATM Forum, ”Audio/Visual Multimedia Services: Video on Demand v1.1”, March 1997

[25] INTEL Corporation, ”i960(R) Microprocessor User Guide for Cyclone and PCI-SDK Evaluation Platforms”,
http://www.zettweb.com/CDROMs/cdrom004/DESIGN/I960/manuals/272577.htm.