Πανεπιστήμιο Κρήτης
Σχολή Θετικών Επιστημών
Τμήμα Επιστήμης Υπολογιστών

Σχεδίαση και Υλοποίηση σε FPGA ενός χρονοπρογραμματιστή κίνησης ABR για έναν μεταγωγέα ATM

Γεώργιος Παπαδάκης

Μεταπτυχιακή Εργασία

Ηράκλειο
Νοέμβρης 2001

Περίληψη

Η κίνηση τύπου ABR είναι μία χαμηλής προτεραιότητας κλάση υπηρεσίας των δικτύων ATM. Παρέχει εγγύηση ελάχιστης διαμεταγωγής κελιών (cell bandwidth), αλλά όχι εγγυήσεις για καθυστέρηση ή διακύμανση καθυστέρησης. Στοχεύει στην αξιοποίηση του εύρους ζώνης, το οποίο δεν χρησιμοποιείται από την κίνηση τύπου CBR και VBR. Αυτό αφορά τόσο την κίνηση που δεν δεσμεύεται για CBR και VBR, όσο και αυτή που δεσμεύεται αλλά δεν χρησιμοποιείται. Το δεύτερο είδος δεν είναι πάντα εφικτό να αξιοποιηθεί και η προσπάθεια για αξιοποίηση του μπορεί να οδηγήσει σε Head of Line Blocking (HLB). Αυτό συμβαίνει όταν πολλαπλοί σύνδεσμοι εξόδου χρησιμοποιούν κοινή FIFO για να λαμβάνουν κελιά, οπότε ένας σύνδεσμος με συμφόρηση μπορεί να προκαλέσει την υποχρησιμοποίηση των υπολοίπων. Αυτό η τοπολογία υπάρχει στο δεδομένο μεταγωγέα, ο οποίος χρησιμοποιεί αρχιτεκτονική διαμοιραζόμενου διαύλου και τοποθετεί πολλαπλούς συνδέσμους εξόδου πίσω από μία συσκευή του διαύλου. Σχεδιάσαμε τον χρονοπρογραμματιστή της κίνησης ABR ώστε να είναι κατάλληλος για υλοποίηση σε FPGA. Η υλοποίηση σε FPGA θέτει περιορισμούς στην εφικτή ταχύτητα και στους διαθέσιμους πόρους όταν την συγκρίνουμε με αυτήν σε ASIC. Παρόλα αυτά , ο χρονοπρογραμματιστής μπορεί να αξιοποιήσει και τα δύο είδη bandwidth και συγχρόνως να αποφύγει το φαινόμενο HLB. Αυτά τα επιτυγχάνει ομαδοποιώντας τις ροές ABR σε ομάδες ροών, ακριβέστερα χρησιμοποιεί μία ομάδα ανά σύνδεσμο εξόδου. Η συγκεκριμένη υλοποίηση χρησιμοποιεί μια διμερής σχεδίαση, η οποία δεν υποστηρίζει εγγυήσεις ελαχίστου bandwidth (MCR). Χρησιμοποιώντας ρολόι των 50 MHz μπορεί να εξυπηρετήσει μέχρι 128 συνδέσμους των 155 Mbps. Το σχέδιο μπορεί να τροποποιηθεί ώστε να υποστηρίξει και εγγυήσεις τύπου MCR. Αυτό μπορεί να γίνει με την προσθήκη ομάδων ροών οι οποίες έχουν bandwidth όσο το άθροισμα των MCR των ροών που συμμετέχουν στην ομάδα. Η υποστήριξη MCR θα μειώσει τον μέγιστο αριθμό από συνδέσμους που το σύστημα μπορεί να υποστηρίξει, αυτό εξαρτάται από το αριθμό των ομάδων που είναι αθροισμένες κατά MCR. Επιπλέον μειώνοντας τον αριθμό των συνδέσμων εξόδου μπορούν να υποστηριχθούν υψηλότερες ταχύτητες για τους συνδέσμους.

Abstract

ABR traffic is a low priority service class for ATM networks. It has a minimum cell rate guarantee per contract, but no guarantee for cell delay and cell delay variance. It aims at exploiting the bandwidth that is not utilized by CBR and VBR traffic. This applies for two types of bandwidth, that not reserved by CBR and VBR contracts and that reserved but not actually used. The second type of bandwidth is not always possible to exploit and attempting to do so, can lead to Head of Line Blocking. This occurs when multiple output links are using a single output FIFO, so a congested link can lead to underutilization of the others. This is the case for our ATM network switch, which utilizes shared bus architecture and features multiple physical links behind a single bus device. We designed a scheduler for ABR traffic, which is suitable for FPGA implementation. Although the FPGA implementation poses limits on the available resources and achievable speed, compared to an ASIC one, our scheduler can potentially exploit both types of bandwidth while avoiding Head of Line Blocking. We achieve the above through aggregation of the ABR connections into flow groups, one per physical output link. Our scheduler's FPGA implementation employs a decoupled design which does not include support for minimum cell rate guarantees (MCR). When using a 50 MHz clock it is able to serve up to 128 physicals operating up to 155 Mbps each. The design can be modified to support minimum cell rate guarantees by adding MCR aggregated flow groups. MCR support will reduce the maximum number of installable physicals; this depends on the number of MCR aggregated flow groups inserted into the system. Also, by reducing the maximum number of physicals, higher speed rates can be attained by each of the physical links.

Πίνακας Περιεχομένων

1. Εισαγωγή

1.1 Η κίνηση τύπου ABR και η χρησιμότητα αυτής

Κατά την τελευταία δεκαετία γίναμε μάρτυρες μίας εκρηκτικής αύξησης στην χρήση του διαδικτύου (Internet) , η οποία οδήγησε σε ανάγκες bandwidth οι οποίες διπλασιάζονται περίπου κάθε 6 μήνες. Το διαδίκτυο είναι βασισμένο στην στοίβα πρωτοκόλλων TCP/IP. Το πρωτόκολλο IP παρέχει υπηρεσία τύπου best-effort, το οποίο σημαίνει ότι το δίκτυο θα προσπαθήσει να παρέχει ένα «δίκαιο» κομμάτι από το bandwidth που είναι διαθέσιμο στην (μεταβαλλόμενη) διαδρομή την οποία ακολουθούν τα πακέτα IP. Δεν παρέχονται άλλες εγγυήσεις. Αρκετοί όμως χρήστες χρειάζονται εγγυήσεις για bandwidth και καθυστέρηση και δεν μπορούν να τις έχουν παρόλο που διατίθενται να πληρώσουν για να αυτές.

Τα δίκτυα ATM διαθέτουν κλάσεις υπηρεσίας οι οποίες είναι κατάλληλες σχεδόν για κάθε χρήση. Μάλιστα η κλάση υπηρεσίας ABR ομοιάζει με την best-effort φύση του διαδικτύου όσα αφορά την «δίκαιη» διανομή του διαθέσιμου bandwidth, αλλά συγχρόνως παρέχει και μία εγγυήσεις ελαχίστου bandwidth ανά ροή. Άρα μεταφέροντας IP κίνηση πάνω από ABR μπορούμε να διατηρήσουμε την best-effort φύση του διαδικτύου και συγχρόνως να παρέχουμε και να χρεώσουμε εγγυήσεις ελαχίστου bandwidth.

Ακολουθεί μία σύντομη εισαγωγή στην αρχιτεκτονική υπηρεσιών του ATM και μία σύντομη περιγραφή της κλάσης υπηρεσίας ABR.

1.2 Δίκτυα ATM

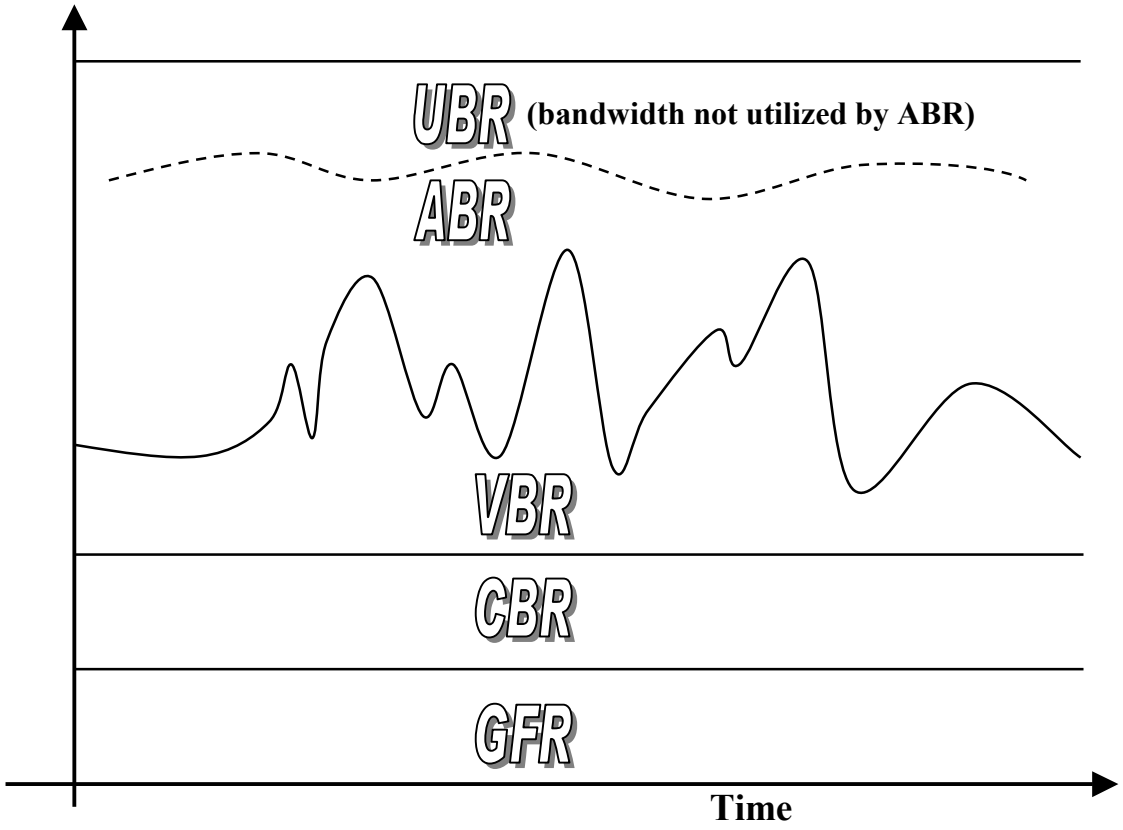
a. Αρχιτεκτονική υπηρεσιών

Η αρχιτεκτονική υπηρεσιών που παρέχεται από το επίπεδο του ATM αποτελείται από τις εξής 6 κατηγορίες υπηρεσίας:

- **CBR** Constant Bit Rate
- **rt-VBR** Real-Time Variable Bit Rate
- **nrt-VBR** Non-Real-Time Variable Bit Rate
- **ABR** Available Bit Rate
- **UBR** Unspecified Bit Rate
- **GFR** Guaranteed frame rate

Αυτές οι κλάσεις υπηρεσίας σχετίζουν τα ζητούμενα χαρακτηριστικά κίνησης με την συμπεριφορά του δικτύου. Λειτουργίες όπως δρομολόγηση, Έλεγχος Αποδοχής Κλήσεων (Call Admission Control) και δέσμευση πόρων γενικά γίνονται διαφορετικά για κάθε κλάση. Οι κλάσεις υπηρεσίας διαχωρίζονται σε πραγματικού και μη πραγματικού χρόνου.

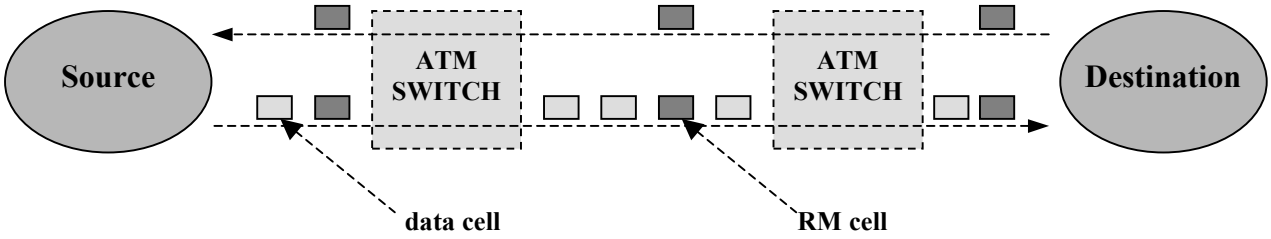
Στην κατηγορία του πραγματικού χρόνου ανήκουν οι CBR και rt-VBR και διαχωρίζονται με μεταξύ τους από το αν περιγραφέας δικτύου περιέχει μόνο το Peak Cell Rate ή το PCR και το Sustainable Cell Rate (SCR) αντίστοιχα. Τόσο το CBR όσο και το rt-VBR παρέχουν εγγυήσεις για την μέγιστη καθυστέρηση αλλά και για την μέγιστη διακύμανση αυτής της καθυστέρησης. Στις μη πραγματικού χρόνου υπηρεσίες υπάγονται οι nrt-VBR, ABR και UBR, αυτές δεν προσφέρουν εγγυήσεις σχετικές με την καθυστέρηση ή την διακύμανση αυτής. Διαφέρουν μεταξύ τους στις εγγυήσεις που το δίκτυο δίνει για το bandwidth και το ποσοστό απώλειας κελιών καθώς και στους μηχανισμούς που υλοποιούνται στους μεταγωγείς και στα άκρα για να υποστηριχθούν. Η εικόνα 1 δείχνει την διανομή του bandwidth ανά κλάση υπηρεσίας του ATM.



Εικόνα 1. Διανομή του bandwidth σε δίκτυα ATM

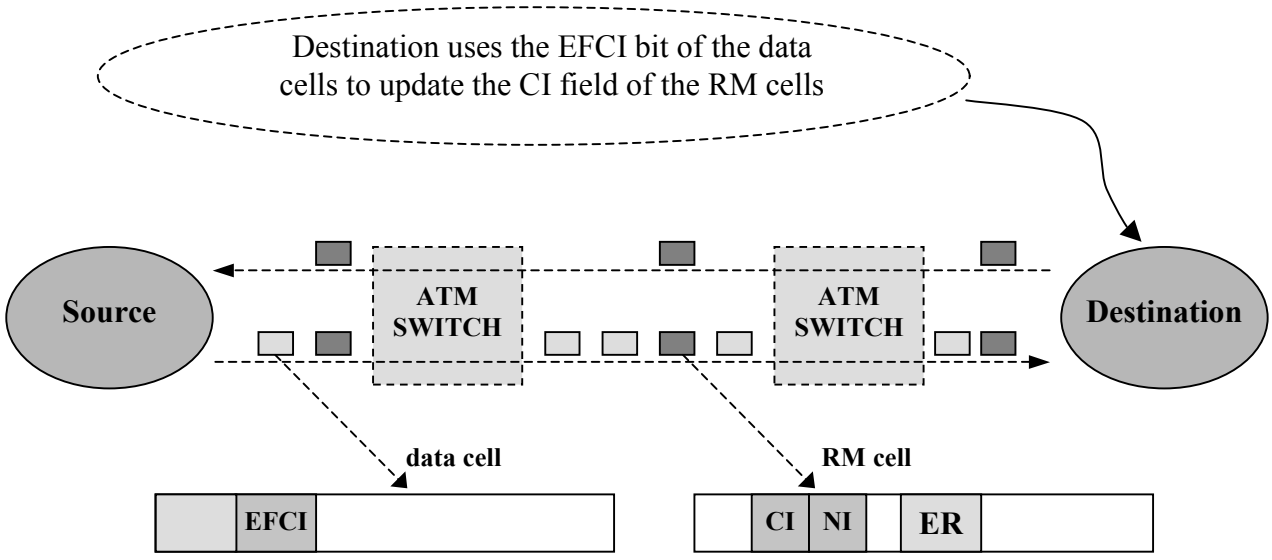
b. Η κλάση Υπηρεσίας ABR

Η ABR είναι η μόνη υπηρεσία των δικτύων ATM για την οποία η συμπεριφορά του δικτύου μπορεί να αλλάξει μετά την εγκαθίδρυση της σύνδεσης. Για αυτό το λόγο είναι και η μόνη υπηρεσία που υπόκεινται σε έλεγχο ροής, ο οποίος μπορεί να γίνει τρεις διαφορετικούς μηχανισμούς. Αυτοί διαφέρουν στην ταχύτητα προσαρμογής του ρυθμού μετάδοσης, στην παρεχόμενη ακρίβεια και στο υπολογιστικό κόστος που απαιτεί η υλοποίηση τους.



Εικόνα 2. Κελιά δεδομένων και κελιά ελέγχου (RM)

Η πληροφορία του ελέγχου ροής μεταφέρεται από ειδικού τύπου κελιά, τα κελιά RM (Resource and Management). Αυτά δημιουργούνται από την πηγή, παίρνουν από όλους τους μεταγωγείς της διαδρομής και καταλήγουν στον προορισμό, ο οποίος τα προωθεί πίσω στην πηγή [Εικόνα 2].Οι ενδιάμεσοι κόμβοι ενημερώνουν αυτά τα κελιά είτε με θέτοντας ακριβώς τον ρυθμό ροής, είτε θέτοντας 2 bits, τα CI (Congestion Indication) και NI (No Increase). Το πρώτο λέγεται Explicit Rate (ER) και προσφέρει γρήγορη και ακριβή προσαρμογή αλλά είναι υπολογιστικά ακριβό στην υλοποίηση του. Το δεύτερο ονομάζεται Relative Rate (RR) και προσφέρει μέτρια ταχύτητα προσαρμογής και ικανοποιητική ακρίβεια, ενώ παράλληλα η υλοποίηση του δεν είναι υπολογιστικά ακριβή. Τέλος υπάρχει και ο διάδικος έλεγχος ροής, σύμφωνα με τον οποίο οι μεταγωγείς θέτουν το EFCI bit (Explicit Forward Congestion Indication) στα κελιά δεδομένων για να δείξουν στον προορισμό ότι πρέπει να θέσει το CI bit στα αντίστοιχα κελιά RM [Εικόνα 3]. Ο μηχανισμός αυτός είναι αργός και δεν προσφέρει ακρίβεια αλλά είναι εύκολος και φθηνός για να υλοποιηθεί.



Εικόνα 3. Μηχανισμοί ελέγχου ροής σε κίνηση ABR

1.3 Υποστήριξη κίνησης ABR σε μεταγωγείς ATM

Η υποστήριξη κίνησης ABR από τους μεταγωγείς και τα άκρα (end systems) είναι αρκετά πιο δύσκολη συγκρινόμενη με την υποστήριξη για κίνηση CBR και VBR. Αυτό είναι λόγω του ότι οι ροές τύπου ABR είναι το μόνο είδος ροών που υπόκεινται σε έλεγχο ροής, δηλαδή απαιτείται από αυτές να προσαρμόσουν την ταχύτητα τους με βάση την επιστρεφόμενη πληροφορία από το δίκτυο (κελιά RM). Αυτό δεν ισχύει για τις ροές CBR και VBR οι οποίες απλώς αστυνομεύονται στο σημείο εισαγωγής τους στο δίκτυο (ingress point) και στη συνέχεια τα κελιά τους προωθούνταί από τους μεταγωγείς χρησιμοποιώντας μικρή ποσότητα προσωρινής μνήμης (buffering). Επιπλέον το δίκτυο πρέπει να παρέχει στις ροές ABR ένα «δίκαιο» κομμάτι του bandwidth και συγχρόνως να εγγυάται χαμηλή απώλεια κελιών σε ροές που συμμορφώνονται προς τον έλεγχο ροής.

Οι τελευταίες δύο απαιτήσεις σε συνδυασμό με την απαίτηση για εγγυημένο ελάχιστο bandwidth, καθιστούν την υποστήριξη κίνησης ABR ένα μη τετριμμένο θέμα για τους μεταγωγείς δικτύων ATM. Για να μπορέσουν οι ροές ABR να εκμεταλλευτούν δυναμικά το διαθέσιμο bandwidth πρέπει να χρησιμοποιηθεί αρκετή μνήμη στους μεταγωγείς. Αυτό είναι απαραίτητο για να κρατιέται ένας επαρκής αριθμός από κελιά ABR μέχρι ο έλεγχος ροής να ειδοποιήσει για αύξηση της ταχύτητας πριν οι ουρές των ροών αδειάσουν. Ο χώρος αυτός είναι απαραίτητος και για το αντίστροφο δηλαδή να μπορούμε να αποθηκεύουμε κελιά χωρίς να υπερχειλίσει η μνήμη μέχρι ο έλεγχος ροής να ειδοποιήσει την πηγή να μειώσει ταχύτητα. Βέβαια θα πρέπει να χρησιμοποιούμε τόση μνήμη όσο πραγματικά χρειάζεται γιατί εκτεταμένο συσσώρευση κελιών οδηγεί σε αυξημένες καθυστερήσεις χωρίς συγχρόνως να επιτυγχάνουμε καλύτερη αξιοποίηση του bandwidth. Δηλαδή θα πρέπει το μέγεθος της ουράς που προσπαθούμε να πετύχουμε να είναι ανάλογο του χρόνο Round Trip (πηγή – προορισμός - πηγή).

1.4 Αντικείμενο της εργασίας

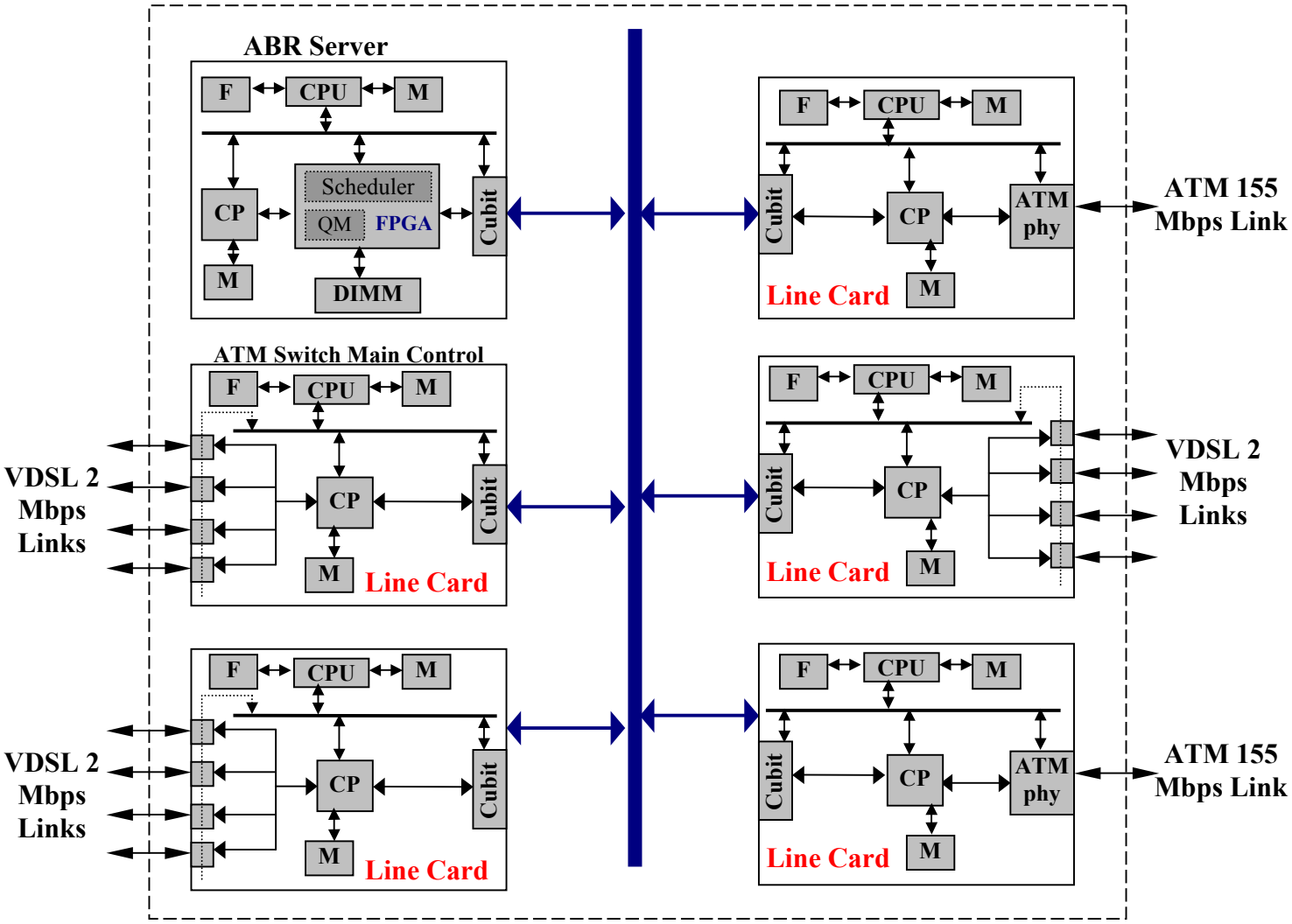
Η εργασία αυτή αποτελείται από την σχεδίαση και υλοποίηση σε FPGA των μονάδων “scheduler” και “sender”, οι οποίες υλοποιούν τον χρονοπρογραμματισμό της κίνησης ABR για τον μεταγωγέα που κατασκευάστηκε στα πλαίσια της του προγράμματος **ΔΙΠΟΛΟ**. Ο εξυπηρετητής κίνήσης ABR κατασκευάστηκε από το Ι.Π. του FORTH και το Τμήμα Επιστήμης Υπολογιστών του πανεπιστήμιου Κρήτης.

Η αναφοράς αυτή είναι οργανωμένη σε 4 κεφάλαια: «1. Εισαγωγή», «2. Επισκόπηση Μεταγωγέα», «3. Πολιτική και σχεδίαση χρονοπρογραμματισμού», «4. Μονάδα Scheduler», «5. Μονάδα Sender», «6. Αρχικοποίηση Μονάδων», «7 Διεπαφές Utopia», «8. Αποτελέσματα Υλοποίησης», «9. Μελλοντική εργασία και βελτιώσεις», «10. Σχόλια και συμπεράσματα».Το κεφάλαιο 2 εισάγει την αρχιτεκτονική του μεταγωγέα και παρέχει μια αρκετά λεπτομερή περιγραφή του Εξυπηρετητή κίνησης ABR. Το κεφάλαιο 3 πραγματεύεται την πολιτική και τον μηχανισμό χρονοπρογραμματισμού. Τα κεφάλαια 4,5 και 6 περιγράφουν την παρούσα υλοποίηση για το χρονοπρογραμματισμό της κίνησης ABR. Το κεφάλαιο 7 παρουσιάζει τις 4 διεπαφές utopia της FPGA. Ακολουθεί το κεφάλαιο 8 το οποίο

ασχολείται με τις δυνατότητες της υλοποίησης , ενώ το κεφάλαιο 9 περιγράφει επιθυμητές αλλαγές και επεκτάσεις όπως η υποστήριξη εγγυήσεων MCR. Η αναφορά αυτή ολοκληρώνεται με μερικά σχόλια και συμπεράσματα στο κεφάλαιο 10.

2. Επισκόπηση Μεταγωγέα

Πριν περιγραφεί η πολιτική χρονοπρογραμματισμού, είναι σημαντικό να δοθεί μία σύντομη αλλά περιεκτική εισαγωγή στην αρχιτεκτονική του μεταγωγέα [Εικόνα 4]. Αυτό είναι απαραίτητο για να μπορέσουμε να δικαιολογήσουμε υποθέσεις και συμβιβάσιμους στην σχεδίαση του χρονοπρογραμματιστή.



Εικόνα 4. Αρχιτεκτονική μεταγωγέα ATM

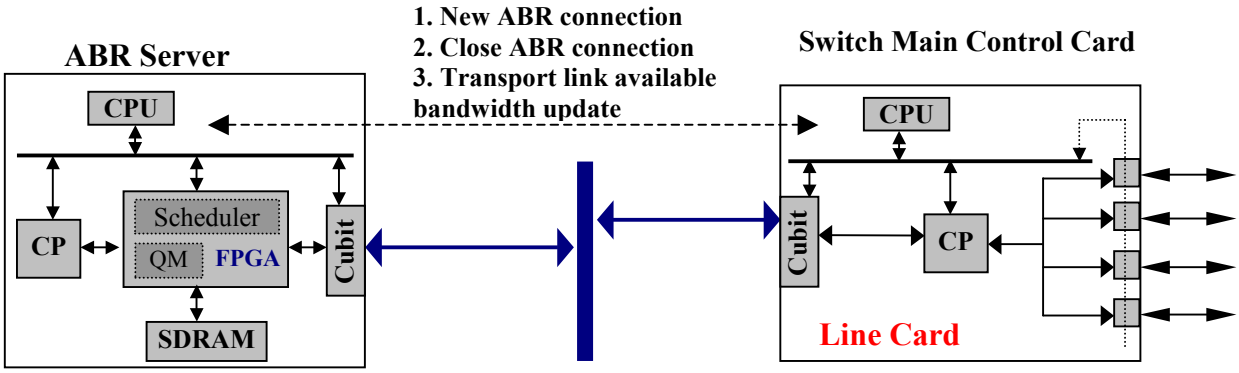
Όπως δείχνει η εικόνα 4, ο μεταγωγέας ATM χρησιμοποιεί αρχιτεκτονική διαμοιραζόμενου διαύλου για την ανταλλαγή κελιών ανάμεσα στις κάρτες. Η κάθε κάρτα ζητάει και αποκτάει το δίαυλο για μετάδοση του κάθε εισερχόμενου κελιού του οποίο ο σύνδεσμος εξόδου βρίσκεται σε διαφορετική κάρτα. Μία από τις κάρτες

ορίζεται να είναι ο διαιτητής ο οποίος διανέμει το bandwidth του διαύλου. Από κάθε σύνδεσμο εισέρχεται κίνηση τύπου CBR, VBR και ABR. Κάθε κάρτα δρομολογεί τα κελιά των συνδέσεων CBR και VBR στην αντίστοιχη κάρτα με τον σύνδεσμο εξόδου. Όλα τα κελιά τύπου ABR δρομολογούνται στον εξυπηρετητή ABR (ABR server), όπου αποθηκεύονται και χρονοπρογραμματίζονται για αναχώρηση.

Η κάθε κάρτα περιέχει είτε 4 συνδέσμους VDSL είτε ένα σύνδεσμο ATM στα 155 Mbps. Η τωρινή υλοποίηση υποστηρίζει μέχρι 16 τέτοιες κάρτες. Η αρχιτεκτονική διαμοιραζόμενου διαύλου που χρησιμοποιείται είναι το CellBus, σχεδιασμένο από την εταιρεία Transwitch. Η συσκευές πρόσβασης στον δίαυλο είναι τα Cubit Pro chips τα οποία επίσης σχεδιάστηκαν από την Transwitch. Ο κεντρικός έλεγχος κάθε κάρτα γίνεται από τον επεξεργαστή Motorola 860, ενώ το σχήμα CP που φαίνεται παραπάνω είναι ο Cell Processor mc92501 επίσης από την Motorola. Αυτό το chip στέλνει και δέχεται κελιά από τους συνδέσμους E/E, και παράλληλα κάνει την μετάφραση των VP/VC. Προαιρετικά ο Cell Processor που βρίσκεται στην κάρτα του εξυπηρετητή ABR μπορεί να κάνει την επεξεργασία των κελιών RM. Το CellBus και τα κελιά είναι αναβαθμίσιμα με συμβατά σε επίπεδο pins chips για να υποστηριχθεί CellBus αρκετών και links των 622 Mbps.

2.2 Κεντρικός έλεγχος του μεταγωγέα

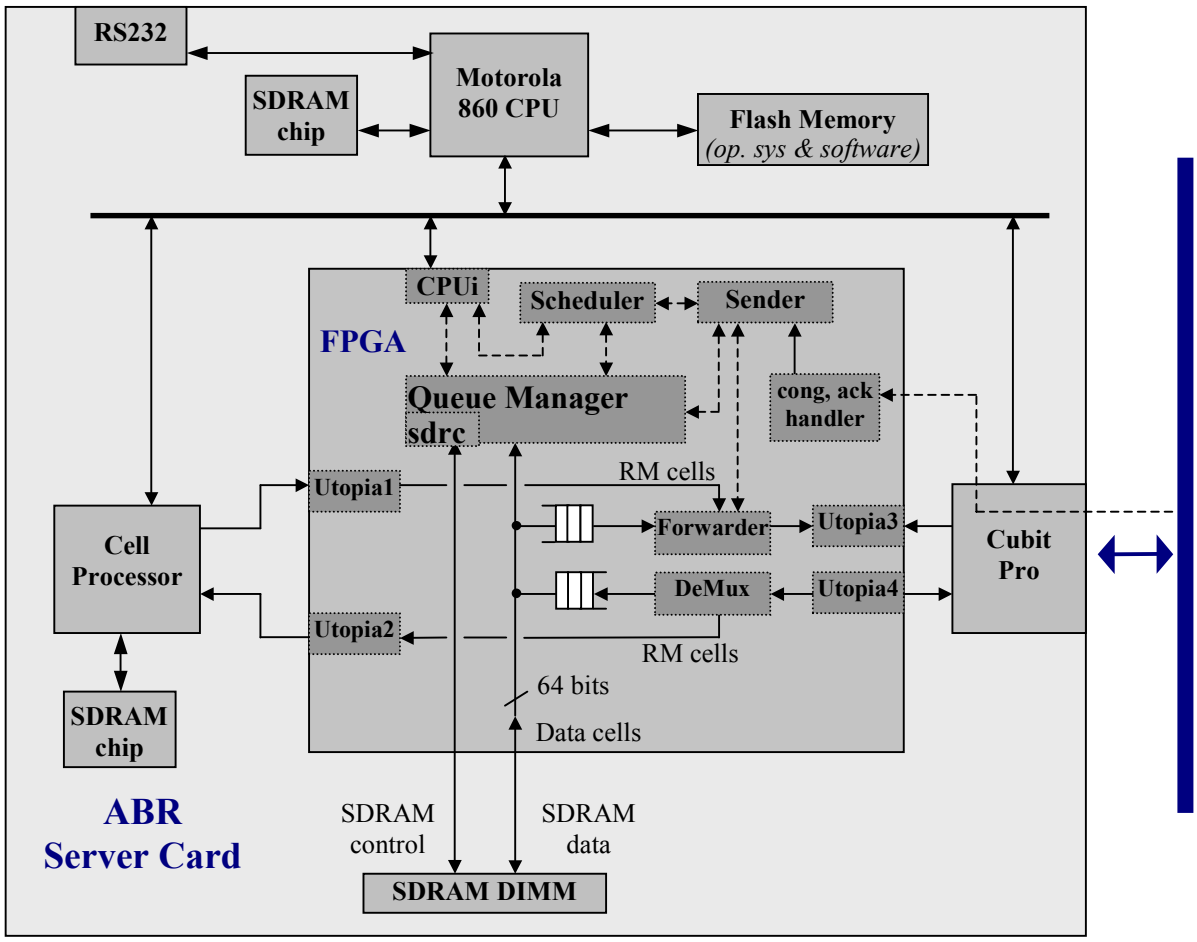
Οποιαδήποτε κάρτα εκτός από τον Εξυπηρετητή ABR μπορεί να κάνει τον κεντρικό έλεγχο του μεταγωγέα. Η κύρια ευθύνη αυτής της κάρτας είναι να διενεργεί τον «Έλεγχο Αποδοχής Κλήσης» (Call Admission Control) για κάθε εισερχόμενη αίτηση σύνδεσης, δηλαδή να αποδέχεται, να τροποποιεί ή και να απορρίπτει συνδέσεις σύμφωνα με το διαθέσιμο bandwidth και άλλους πόρους του μεταγωγέα. Επιπλέον, αυτή η κάρτα είναι υπεύθυνη για να ειδοποιεί τον Εξυπηρετητή ABR για κάθε σύνδεση ABR που ανοίγει ή κλείνει καθώς και για τις αλλαγές στο διαθέσιμο bandwidth του κάθε συνδέσμου εξόδου [Εικόνα 5]. Για αυτούς τους συνδέσμους οι οποίοι δεν μοιράζονται την συσκευή πρόσβασης στο CellBus είναι μόνο απαραίτητο να ειδοποιηθεί ο εξυπηρετητής κίνησης ABR μία φορά μόνο για την ταχύτητα του συγκεκριμένου συνδέσμου εξόδου. Αυτό γιατί ο χρονοπρογραμματιστής μπορεί δυναμικά να προσαρμοστεί στο εκάστοτε διαθέσιμο bandwidth μέσω του πληροφορίας για συμφόρηση που έρχεται από το αντίστοιχο Cubit Pro της κάρτας που φέρει τον σύνδεσμο εξόδου.



Εικόνα 5. Πληροφορίες που παρέχονται στον Εξυπηρετητή ABR

2.3 Ο Εξυπηρετητής κίνησης ABR

Το διάγραμμα του εξυπηρετητή ABR φαίνεται στη εικόνα 6. Αυτή η κάρτα συγκεντρώνει και χρονοπρογραμματίζει όλη την κίνηση ABR του μεταγωγέα και επομένως έχει αρκετή μνήμη για να αποθηκεύσει δεκάδες χιλιοστά του δευτερολέπτου για κάθε σύνδεση ABR. Για το σκοπό αυτό χρησιμοποιεί μια μεγάλο αποθηκευτικό χώρο τύπου SDR SDRAM (ένα 64-bit DIMM των 256 MB).



Εικόνα 6. Ο Εξυπηρετητής κίνησης ABR

Οι λειτουργίες του χρονοπρογραμματισμού και των διαχείρισης των ουρών υλοποιούνται σε μία FPGA της εταιρείας Altera και πιο συγκεκριμένα το μοντέλο EPF10K200EBC600-2 της οικογένειας FLEX10KE. Επιπλέον στην FPGA αυτήν υπάρχουν 4 διεπαφές utopia οι οποίες χρησιμεύουν για την επικοινωνία με τα Cubit Pro και Cell Processor. Ακριβέστερα, οι διεπαφές που συνομιλούν με τον Cell Processor χρησιμοποιούν 8-bit πλάτος και προσομοιώνουν το επίπεδο ATM του utopia, ενώ οι διεπαφές που συνομιλούν με το Cubit Pro χρησιμοποιούν 16-bit πλάτος και προσομοιώνουν το φυσικό επίπεδο του utopia. Υπάρχει και μία πέμπτη διεπαφή η οποία χρησιμεύει για τον μικροεπεξεργαστή. Αυτή υλοποιείται στη μονάδα CPUi και επιτρέπει στις μονάδες scheduler και QM να επικοινωνούν με τον επεξεργαστή, ο scheduler λαμβάνει τις ειδοποιήσεις για την μεταβαλλόμενες ταχύτητες των

συνδέσμων εξόδου, ενώ ο QM λαμβάνει εντολές όπως το άνοιγμα και κλείσιμο συνδέσεων ABR. Επιπλέον μέσω αυτής της διεπαφής μπορεί να παρακολουθείται η λειτουργία των μανάδων της FPGA.

Τα αφιχθούμενα κελιά από το utopia3 διαχειρίζονται από την μονάδα DeMux η οποία προαιρετικά περνά τα κελιά RM στο utopia1 για να δοθούν στον Cell Processor, ενώ τα κελιά δεδομένων εισάγονται στην 64-bit Cell Enqueue FIFO για να γραφούν στην SDRAM. Ο διαχειριστής ουρών (QM) αναγνωρίζει τα κελιά που βρίσκονται στην Cell Enqueue FIFO και τα προσθέτει στην κατάλληλη ουρά στην SDRAM. Η εξερχόμενη κίνηση είναι είτε ενημερωμένα κελιά RM από το utopia2 είτε κελιά δεδομένων από την 64-bit Cell Dequeue FIFO, η οποία αποθηκεύει τα κελιά που έρχονται από την SDRAM. Η μονάδα forwarder είναι υπεύθυνη για να προωθήσει την εξερχόμενη κίνηση στο utopia4, το οποίο με την σειρά του θα το δώσει στο Cubit Pro για μετάδοση πάνω στο CellBus.

Η λειτουργία του χρονοπρογραμματισμού γίνεται από τη μονάδα scheduler, η οποία δίνει αναγνωριστικά ομάδων ροών που δικαιούνται εξυπηρέτηση στη μονάδα sender. Η μονάδα sender, αφού πραγματοποιήσει ορισμένους ελέγχους, ζητάει από τον QM ένα κελί από κατάλληλη σύνδεση ABR. Ο QM διαβάζει, όταν γίνει διαθέσιμος ένα κελί από την SDRAM και το εισάγει στην Cell Dequeue FIFO. Η τελευταία μαζί με την Cell Enqueue FIFO επιτρέπουν στον QM να γράφει και να διαβάζει κατά ριπές τα κελιά από το SDRAM DIMM. Παρόλο που αυτές οι FIFO είναι ακριβές να υλοποιηθούν στην FPGA, η χρήση τους ήταν απαραίτητη για να μην περιορίσουμε σημαντικά τις επιδόσεις της SDRAM.

3. Πολιτική και σχεδίαση χρονοπρογραμματισμού

3.1 Είδη διαθέσιμου bandwidth για κίνηση ABR

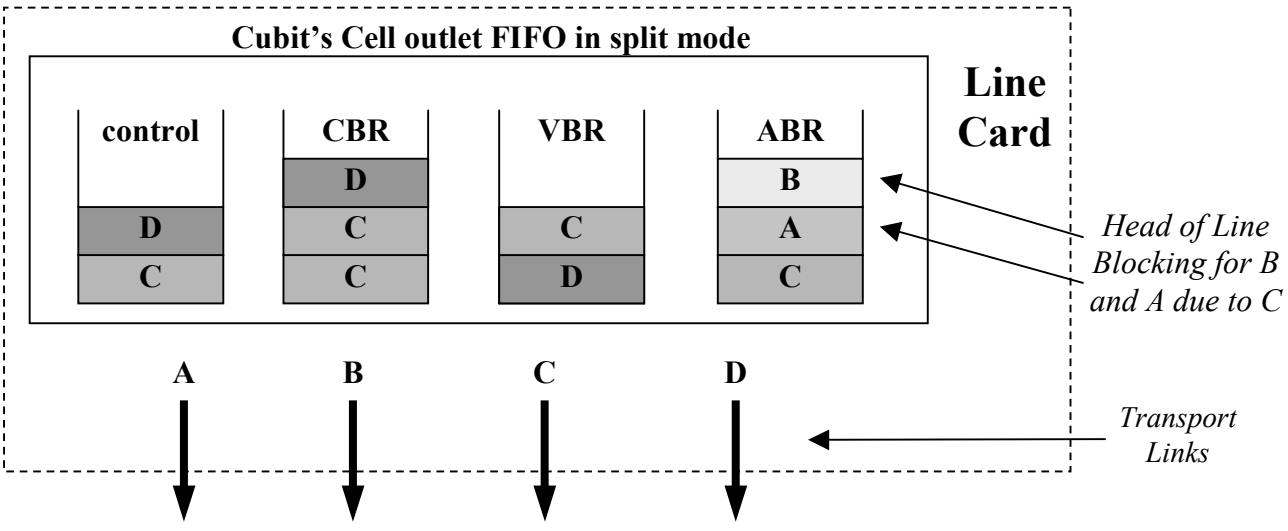
Για να γίνει ο χρονοπρογραμματισμός της κίνησης ABR, είναι απαραίτητο να υπολογίζεται το μη χρησιμοποιούμενο bandwidth του κάθε συνδέσμου. Αυτή η υποχρέωση βαραίνει την μονάδα που διεξάγει το Έλεγχο Αποδοχής Κλήσεων (CAC). Η μονάδα αυτή είναι πάντα ενήμερη για το bandwidth το οποίο δεν έχει αποδοθεί στις διαφόρων τύπων συνδέσεις, αυτό της είναι αναγκαίο προκειμένου να μπορεί να αποφασίσει την αποδοχή ή τροποποίηση των εισερχόμενων αιτήσεων για άνοιγμα σύνδεση. Παρόλα αυτά δεν είναι όμως απαραίτητο να υπολογίζει το bandwidth το οποίο έχει δεσμευτεί αλλά δεν αξιοποιείται, γιατί αυτό έχει μικρή χρησιμότητα για την διενέργεια του CAC μιας και αφορά μόνο τον τωρινό χρόνο και δεν δίνει κάποια εγγύηση για το κοντινό μέλλον. Επιπλέον για να υπολογιστεί αυτό το είδος bandwidth είναι αναγκαίο να παρακολουθούνται δυναμικά οι έξοδοι, η να γίνεται κάποια «έξυπνη» εκτίμηση, και τα δύο είναι δύσκολα με την παρούσα αρχιτεκτονική. Με την προϋπόθεση ότι η μονάδα scheduler ενημερώνεται με το άθροισμα των δύο ειδών bandwidth, -δηλαδή με το άθροισμα του μη δεσμευμένου bandwidth με αυτό του δεσμευμένου αλλά μη χρησιμοποιούμενου – είναι εφικτό να αξιοποιηθούν πλήρως όλοι οι σύνδεσμοι εξόδου ακόμα και αυτοί που μοιράζονται την ίδια συσκευή Cubit pro.

Αν το δεύτερο μέρος του αθροίσματος είναι άγνωστο, τότε η μονάδα scheduler υποχρεούται να βασιστεί μόνο στη γνώση του πρώτου μέρος, και έτσι να αποφύγει την εμφάνιση του φαινομένου HLB (Head of Line Blocking). Αυτό το φαινόμενο εμφανίζεται όταν υπάρχουν πολλαπλοί σύνδεσμοι εξόδου πίσω από μία FIFO. Η έλλειψη της γνώσης του δεύτερου bandwidth δεν είναι πρόβλημα για τους συνδέσμους που δεν τροφοδοτούνται από κοινόχρηστη FIFO. Σε αυτούς τους συνδέσμους μπορεί να επιτευχθεί πλήρης αξιοποίηση χρησιμοποιώντας τον έλεγχο ροής που παρέχουν οι συσκευές Cubit Pro. Η μονάδα scheduler χρειάζεται να ξέρει για αυτούς μόνο την μέγιστη χωρητικότητα τους.

3.2 Μηχανισμός ομαδοποίησης

Για να είναι σχεδιαστεί ένας εφικτός χρονοπρογραμματιστής είναι απαραίτητο να αποφθεχθεί ο χρονοπρογραμματισμός σε επίπεδο ροής. Ένα κατάλληλο σχήμα συναθροισμού χρησιμοποιείται για τον σκοπό αυτόν. Ομαδοποιούμε τις ροές σε ομάδες ροών και έτσι πετυχαίνουμε να μειώσουμε τον αριθμό των αντικειμένων που χρονοπρογραμματίζονται. Η συνάθροιση των ροών γίνεται με βάση τον σύνδεσμό εξόδου και χρησιμοποιείται Weighted Round Robin για τον χρονοπρογραμματισμό των προκυπτόντων ομάδων ροών.

Κάθε ομάδα ροών ενός συνδέσμου εξόδου που δεν μοιράζεται τον FIFO (δηλαδή την συσκευή Cubit Pro) της με άλλη ομάδα, χρονοπρογραμματίζεται σε ταχύτητα τόση όση είναι η χωρητικότητα του συνδέσμου, και όταν παρουσιάζεται συμφόρηση στο αντίστοιχο Cubit Pro, τότε πραγματοποιείται «οπισθοχώρηση» (Back Off). Χρησιμοποιούμε οπισθοχώρηση για να αποφευχθούν μη απαραίτητες μεταδόσεις και έτσι σπατάλη bandwidth στο CellBus καθώς και επιβάρυνση τους διαχειριστή ουρών. Όταν μία ομάδα ροών χρησιμοποιεί αντιστοιχεί σε σύνδεσμο ο οποίος μοιράζεται την συσκευή διαύλου με άλλους (πολλαπλοί σύνδεσμοι εξόδου στην κάρτα), τότε η μονάδα scheduler ενημερώνεται δυναμικά για το διαθέσιμο bandwidth τέτοιων συνδέσμων. Αυτό στοχεύει στην αποφυγή του φαινομένου Head of Line Blocking [Εικόνα 7], με την προϋπόθεση βέβαια ότι οι ενημερώσεις είναι σωστές και γρήγορες.



Εικόνα 7. Head of Line Blocking

Είναι κρίσιμο να επιλυθούν τα συμβάντα HLB όσο το δυνατό ταχύτερα, για το σκοπό αυτό χρησιμοποιείται εκθετική πολιτική οπισθοχώρησης και αποκατάστασης για τις ομάδες ροών που εμφανίζουν συμφόρηση. Αυτό σημαίνει ότι μία ομάδα ροών η οποία βλέπει συμφόρηση θα οπισθοχωρήσει γρήγορα ώστε να διευκολύνει την επίλυση του HLB. Φυσικά όλες οι ομάδες ροών που διέρχονται από το συμφορημένο Cubit Pro θα δουν την συμφόρηση και θα οπισθοχωρήσουν παρόλο που μπορεί να μην ήταν αυτές που την προκάλεσαν. Εδώ γίνεται η υπόθεση ότι η ομάδα (ή ομάδες) ροών που προκάλεσαν το φαινόμενο HLB θα ενημερωθούν σύντομα στην σωστή ταχύτητα, και το φαινόμενο θα αναιρεθεί. Η εκθετική πολιτική αποκατάστασης της ταχύτητας των ροών βοηθάει να μην τιμωρηθούν περαιτέρω οι λοιπές ομάδες, οι οποίες δεν ήταν υπεύθυνες για την εμφάνιση του φαινομένου.

3.3 Διμερής Σχεδίαση χρονοπρογραμματισμού

Για τον χρονοπρογραμματισμό της κίνησης ABR χρησιμοποιείται ένα διμερές σχέδιο, το οποίο αποτελείται από τις μονάδες «scheduler» και «sender». Η μονάδα scheduler υλοποιεί την πολιτικές χρονοπρογραμματισμό (WRR) και οπισθοδρόμισης (εκθετική) για τις ομάδες ροών. Η μονάδα sender είναι υπεύθυνη για να δώσει τις κατάλληλες αιτήσεις στον διαχειριστή ουρών (QM), να χειριστεί τις επαναμεταδώσεις, να αναγνωρίσει τις ελεύθερες θέσεις μνήμης για κελιά και να στείλει τα κελιά για μετάδοση στο Cubit Pro.

Για να λειτουργήσει η συνάθροιση των ροών ανά σύνδεσμο, ο διαχειριστής ουρών διατηρεί μια κυκλική λίστα από τις μη άδειες ροές του κάθε συνδέσμου (ομάδα ροών). Κάθε φορά που μία ομάδα δικαιούται εξυπηρέτηση η μονάδα scheduler εισάγει το αναγνωριστικό της σε μία FIFO την ποία ονομάζουμε Eligible FIFO. Η μονάδα sender εξάγει αναγνωριστικά από αυτή την FIFO και αφού πραγματοποιήσει ορισμένους ελέγχους δίνει αίτηση στον διαχειριστή ουρών για ένα κελί που ανήκει στην δεδομένη ομάδα. Ο διαχειριστής ουρών πραγματοποιεί απλό Round Robin (RR) ανάμεσα στις μη άδειες ροές τις ομάδες και διαλέγει ένα κελί από κάθε ροή πριν προχωρήσει στην επόμενη.

Για κάθε κελί που επιχειρείται να μεταδοθεί πάνω από το CellBus, η μονάδα sender διατηρεί ένα δείκτη στη θέση μνήμη όπου είναι αποθηκευμένο. Αυτό γίνεται γιατί ο διαχειριστής ουρών το εξάγει από την ουρά της αντίστοιχη ροής. Έτσι, αν μία μετάδοση πάνω από το δίαυλο αποτύχει, τότε ο sender μπορεί να παρέχει στον διαχειριστή ουρών αυτόν τον δείκτη την επόμενη φορά που η ίδια ομάδα ροών θα δικαιούται εξυπηρέτηση. Είναι λογικό να υποθέσουμε ότι μία μετάδοση στο CellBus θα αποτύχει πολύ σπάνια λόγω σφάλμα τύπου CRC, άρα σχεδόν όλες οι αποτυχημένες μεταδόσεις θα οφείλονται στην γεμάτη FIFO στο Cubit Pro προορισμού. Θα δούμε στο κεφάλαιο 9 πως μπορούμε να εκμεταλλευτούμε αυτό για να βελτιώσουμε την μονάδα sender.

Δηλαδή η μονάδα sender προωθεί στον διαχειριστή ουρών δύο ειδών αιτήσεις, είτε ζητάει εξαγωγή κελιού και παρέχει ένα αναγνωριστικό ομάδας, είτε ζητάει διάβασμα κελιού και παρέχει τον αντίστοιχο δείκτη μνήμης. Αν μία μετάδοση είναι επιτυχής ο διαχειριστής ουρών ειδοποιείται ότι η συγκεκριμένη θέση μπορεί να προστεθεί στην λίστα ελευθέρων θέσεων. Αυτό γίνεται μέσω μίας μικρής FIFO ελευθέρων δεικτών η οποία γράφεται από τον sender και διαβάζεται από τον διαχειριστή ουρών.

Σε αυτή τη σχεδίαση κάναμε την υπόθεση ότι κάθε σύνδεση ABR ζητάει (ή ο CAC το επιβάλει) μηδενική εγγύηση ελαχίστου bandwidth (MCR=0). Αν μία εφαρμογή απαιτεί εγγύηση MCR τότε θα πρέπει να την πάρει μέσω ενός συμβολαίου CBR ή VBR. Η σχεδίαση μπορεί να αλλαχθεί ώστε να υποστηριχθούν εγγυήσεις τύπου MCR μέσω ομάδων που συναθροίζουν το bandwidth των εγγυήσεων MCR. Για γίνει αυτό η μονάδα sender πρέπει να υποστεί σημαντικές αλλαγές, ενώ η μονάδα scheduler δεν χρειάζεται ελάχιστες αν όχι καθόλου, αυτό οφείλεται στην διμερή αρχιτεκτονική μας.

Παράρτημα (Αγγλική Μετάφραση)

1. Introduction

1.1 ABR traffic and its importance

During the last decade we have witnessed an explosive growth in Internet usage, which has resulted in bandwidth needs that double every six-months. Internet is based on the TCP / IP protocol stack. The IP protocol provides best effort service, meaning the user gets its "fair" share [15] of whatever bandwidth is available along the route that the IP packets follow, no other guarantees are provided. This has the undesired effect that users who need bandwidth and latency guarantees and are willing to pay for them, are unable to get them. ATM Networks feature service categories that are suitable for almost any usage. In particular, the ABR Service resembles the best-effort nature of the Internet in that the user gets a fair share of whatever bandwidth is available, but it can also provide a minimum bandwidth guarantee, if one is required. So, by transferring IP traffic over ABR connections we can retain the best-effort behavior of Internet, while at the same time charge and provide minimum bandwidth guarantees. Following is short introduction into ATM Service architecture and a short description of the ABR service category.

1.2 ATM Networks

a. ATM service Architecture

The service architecture provided at the ATM layer consists of the following six service categories [12]:

- **CBR** Constant Bit Rate
- **rt-VBR** Real-Time Variable Bit Rate
- **nrt-VBR** Non-Real-Time Variable Bit Rate
- **ABR** Available Bit Rate
- **UBR** Unspecified Bit Rate
- **GFR** Guaranteed frame rate

These service categories relate traffic characteristics and QoS requirements to network behaviour. Functions such as routing, Call Admission Control (CAC) and resource allocation are in general structured differently for each service category. Service categories are distinguished as being either real-time or non-real-time. For real-time traffic there are two categories, CBR and rt-VBR, distinguished by whether the traffic descriptor contains only the Peak Cell Rate (PCR) or both PCR and the Sustainable Cell Rate (SCR) parameters. Both CBR and rt-VBR have guarantees for cell delay and cell delay variance. The non-real-time categories include nrt-VBR, ABR, and UBR, these services categories do not offer cell delay guarantees. They differ as to the nature of the bandwidth and cell loss ratio guarantees provided by the network and the mechanisms that are implemented in the end-systems and network to support them.

Selection of an appropriate service category is application specific. Real time service categories may be used by applications such teleconferencing. Audio teleconferencing would use CBR, while video teleconferencing would use rt-VBR since video has a more bursty nature than audio and its higher bandwidth demands prevent us from using an expensive and unnecessary high bandwidth CBR contract. The nrt-VBR service class is suitable for use by applications that do video or audio streaming, since they require bandwidth guarantee and produce bursty traffic but don't need delay guarantees since with an appropriate initial amount of buffering at the destination, the stream playback will have no hic-ups. ABR service category provides an optional minimal cell rate guarantee per contract and potentially low cell loss ratio, so it is suitable for web browsing and file transfer since such usage is not sensitive to bandwidth and cell delay variations. UBR offers no guarantees whatsoever, it just allows the bandwidth that the above service categories do not use to be utilized by applications that are not very sensitive to cell loss, such as new groups broadcasting and email. GFR (Guaranteed Frame Rate) resembles CBR in that it has a bandwidth guarantee, but this applies to frames rather than cells. GFR has no flow control and if the bandwidth guarantee is exceeded complete frames will be discarded if network can not transmit any cell of the frame. This service is useful for relaying higher level packets in GFR frames, e.g. IP packets, since if just one cell of the frame is discarded it's of no use to transmit the remaining cells of the frame. Bandwidth partitioning per service category is depicted in figure 1.

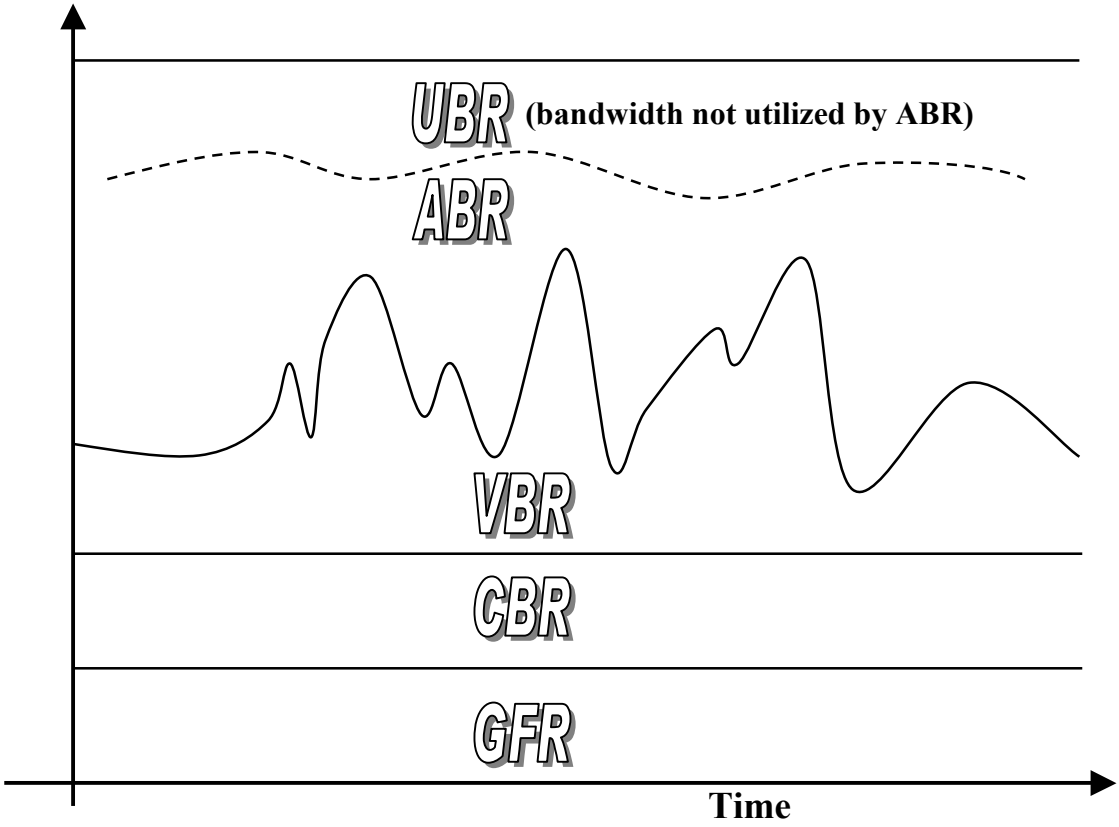


Figure 1. Bandwidth partitioning in ATM Networks

b. ABR service category

ABR is an ATM layer service category for which the limiting ATM layer transfer characteristics provided by the network may change subsequent to connection establishment. ABR traffic is the only ATM traffic that is subject to flow control [14]. Several mechanisms of feedback are available to control the source rate in response to changing bandwidth and network resources:

- **Explicit Rate (ER)**

  Feedback is conveyed to the source through specific control cells called Resource Management Cells, or RM cells. The source interleaves a small amount of RM cells in its data stream containing the desired transmission rate. These cells are updated along the network route and eventually they return back to the source [Figure 2]. In particular, the intermediate switches and the destination update a field in the RM cells called the Explicit Rate field (ER) to match the current bandwidth that they can sustain. Each of them updates this field only if it requires a lower rate. They are also allowed to insert new RM cells in either direction, so that dropped RM cells due to congestion can be replaced. Updating backward RM cells achieves faster feedback. So it is usually preferable to update only backward RM cells in order to avoid wasting computational resources. This mechanism is fast and exact in adjusting the transmission rate according to the changing network conditions, yet it is usually too demanding to be implemented for high speed transport links.

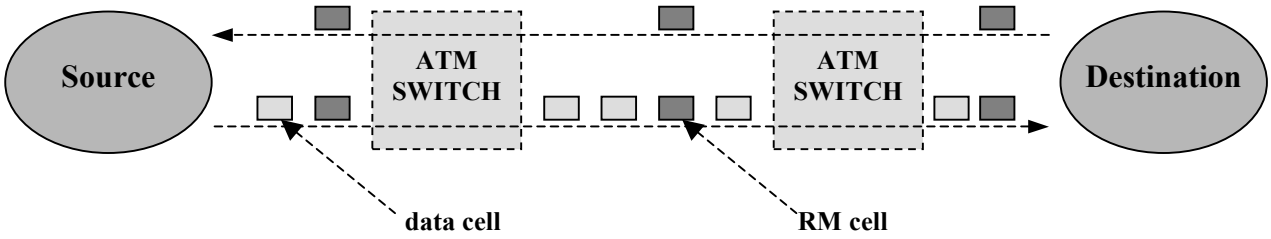


Figure 2. Data and control (RM) cells in ABR traffic

- **Relative Rate (RR)**

  The destination and intermediate switches use two bits in RM cells to send feedback to the source. The explicit rate field in the RM cells is left untouched, instead the bits CI (Congestion Indication) and NI (No Increase) of the RM cells are used to notify the source about network condition. The source adjusts its rate by additive increase and decrease according to these bits. The NI bit indicates that the rate should not be increased, while the CI dictates that rate should be decreased. If both bits are cleared the source is allowed to increase its rate. This mechanism offers medium speed and satisfactory accuracy for adapting the transmission rate to network conditions. Furthermore it is rather easy to implement since it only requires two thresholds to be examined per ABR flow queue, one to set the No Increase bit and one to set the Congestion Indication bit. As we said before it is preferable to set the fields in backward rather than forward RM cells to achieve faster feedback.

- **Binary Mode (EFCI)**
  The intermediate switches use the EFCI bit in data cells to send forward congestion indications (i.e. to the destination). The destination uses this bit to update the CI field of the backward RM cells that it sends to the source [Figure 3]. The EFCI mechanism offers slow and coarse-grain adaptation to network congestion, but it is easy to be implemented since the intermediate switch can set EFCI in the data cells by just using one or two thresholds per ABR flow queue, no need to identify and process backward RM cells.

The ABR service has no guarantees for the delay or the delay variation experienced by a given connection, thus is not intended to support real-time applications. During connection establishment, the end-system specifies to the network both the maximum and minimum required bandwidth. These are called peak cell rate (PCR) and minimum cell rate (MCR) respectively. The MCR may be specified as zero. The bandwidth available from the network may vary, but is guaranteed not to become less than MCR. PCR is usually specified as the maximum capacity of the route.

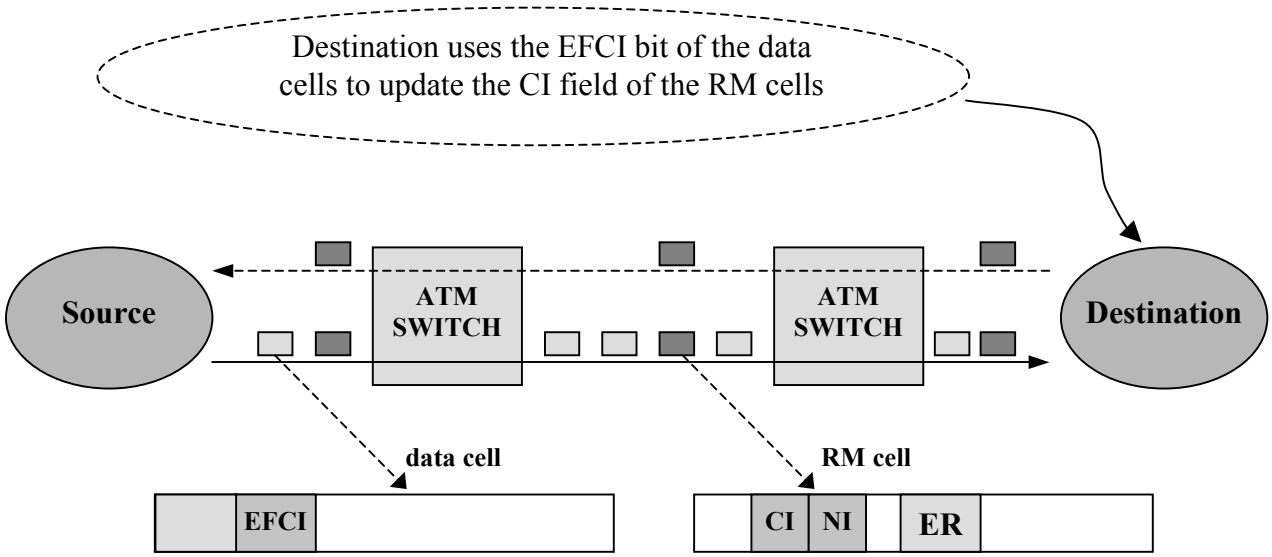


Figure 3. Combining the flow control mechanisms

An end-system that adapts its traffic according to the feedback is expected to experience low cell loss ratio and obtain a fair share of the available bandwidth. Fair share depends on the allocation policy of each network. Here we must note a problem related to RR and EFCI mechanisms. The fact that a specific rate is not provided makes policing at the ingress point difficult yet feasible. Without such policing at the ingress point some users, either unintentionally or maliciously, can ignore the network's congestion indications and continue to transmit at the same rate. This will effectively punish the conforming users while at the same time will give additional network resources to the non conforming users. Of course, the problem involves only ABR traffic, the VBR and CBR traffic use reserved resources per connection, thus non-conforming ABR users won't effect them.

1.3 Supporting ABR traffic in ATM switches

Supporting ABR traffic in a network switch is much harder than supporting CBR and VBR. This is due to the fact that ABR connections are the only kind of ATM connections that are subject to flow control and are required to adapt their rate according to the available bandwidth. This is not the case for CBR and VBR connections that are simply policed once at their ingress point (the point that they enter the ATM network), subsequently the traffic is just forwarded through the network switches using minimal buffering. Furthermore the network is required to provide all ABR connections with a fair share of the available bandwidth, while at the same time avoid dropping any ABR cells of flows that conform to flow control.

The above two requirements and the guarantee of a minimum bandwidth per connection makes support for ABR traffic a challenging issue in ATM network switches. In order to exploit the available bandwidth dynamically sufficient buffering capabilities need to be added to the switches. However, extensive buffering should not be employed, since it won't increase the throughput of the ABR connections and may lead to unnecessary increased average delays for the ABR cells. Buffering should be analogous to the roundtrip time of the network to avoid such delays, while providing enough time to the source to adjust its transmission rate according to the feedback.

ATM network switches have been commercially available for more than six years. The initial ATM switches immediately supported CBR and VBR service categories. The support for ABR has grown steadily the last couple of years, now most switches support ABR traffic with ER flow control.

1.4 This Thesis

This thesis consists of the design and FPGA implementation of the scheduler, sender sub-systems and the utopia interfaces of an ABR server card developed by I.C.S F.O.R.T.H. and C.S.D. of the University of Crete. The ATM network switch [1][2] that hosts this card was developed for the project DIPOLO, other members of the are the N.T.U.A. university and INTRACOM corporation.

The thesis is organized in 10 chapters: "1. Introduction", "2. Switch Overview", "3. Scheduling Policy and design", "4. Scheduler module", "5. Sender module", "6. Module Initialization", "7. Utopia Interfaces", "8. Implementation Results", "9. Future work and enhancements", "10. Discussion and Conclusions".

Chapter 2 introduces the architecture of our ATM network switch and gives a detailed description of our ABR server card. Chapter 3 discusses and justifies our scheduling policy and design choices. Chapters 4,5 and 6 describe our current implementation of ABR traffic scheduling. Chapter 7 presents the 4 utopia interfaces of the FPGA. Chapter 8 discusses the capabilities of the current design, while chapter 9 describes desirable modifications and enhancements such as the support for MCR guarantees. We conclude with a short discussion of this work in chapter 10.

2. Switch Overview

Before we introduce our scheduling policy, it is important to provide a short but concrete introduction into the switch architecture. This is necessary so that we can justify our assumptions and explain the trade-offs in our scheduler design.

2.1 Switch architecture

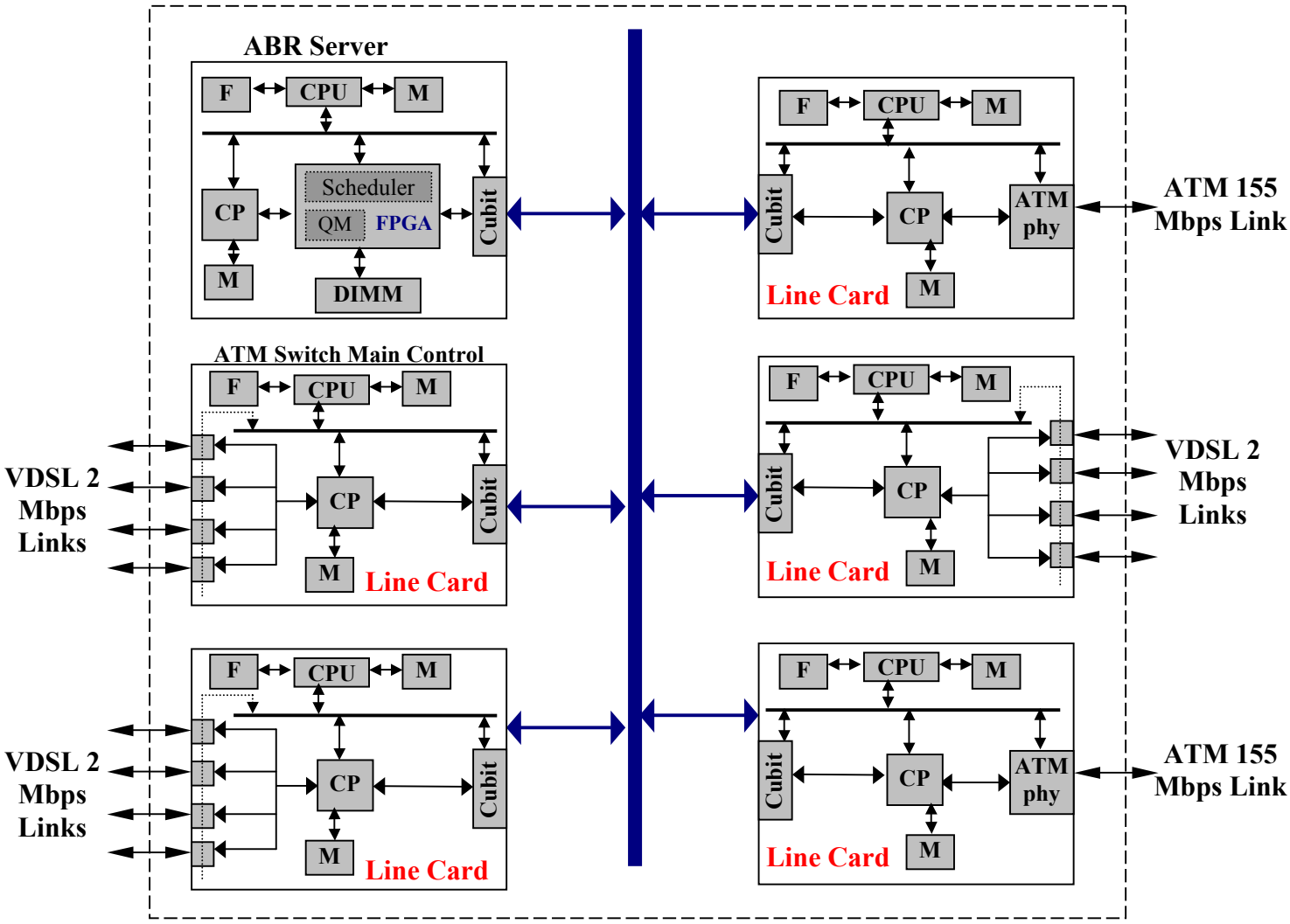


Figure 4. ATM switch architecture

As the figure 4 depicts, our ATM switch utilizes shared bus architecture for switching cells between line cards. Each card requests and obtains the bus for the transmission of every arriving cell whose output link is located in a different card. One of the line cards is designated to act as the arbiter, who is responsible for distributing bus bandwidth. From each transport link CBR, VBR, and ABR traffic arrives. Each line card routes the arriving cells that belong to CBR and VBR

connections to the appropriate output line card over the shared bus. All ABR cells are forwarded to the ABR Server Card [7], where they are stored and scheduled for departure.

Each line card may contain up to four VDSL modems or one ATM Link operating at 155 Mbps. Our current implementation supports up to 16 line cards. The shared bus architecture used is the CellBus by Transwitch with a maximum throughput of 1 Gbps. The bus devices are the Cubit Pro chips also manufactured by Transwitch. The main control of each line card is done by Motorola 860 microprocessor, while the CP block shown above is the Cell Processor mc92501 also by Motorola. This chip forwards and accepts traffic to and from the physical links and does the VP/VC translation, optionally the cell processor on the ABR server card can process the ABR RM cells. The CellBus and the bus devices are upgradeable. The cubit pro can be replaced with pin compatible chips to support a multi-gigabit CellBus and 622 Mbps ATM links.

2.2 Switch main control

Any of the line cards, other than the ABR server, can be assigned as the **switch's main control card**. Its main responsibility is to perform the Call Admission Control (CAC) for each connection request, accepting, modifying and rejecting connections according to available bandwidth and other switch resources. Also, this card is responsible for notifying the ABR Server Card for every new or deprecated ABR connection as well as for every change of unused bandwidth in the output transport links [Figure 5]. For those output links that do not share their bus device, it is only necessary to notify the ABR server card once about their transmit capacity, since the ABR scheduler can dynamically adapt to their available bandwidth through the backpressure information provided by the corresponding bus device of the line card that carries the output link.

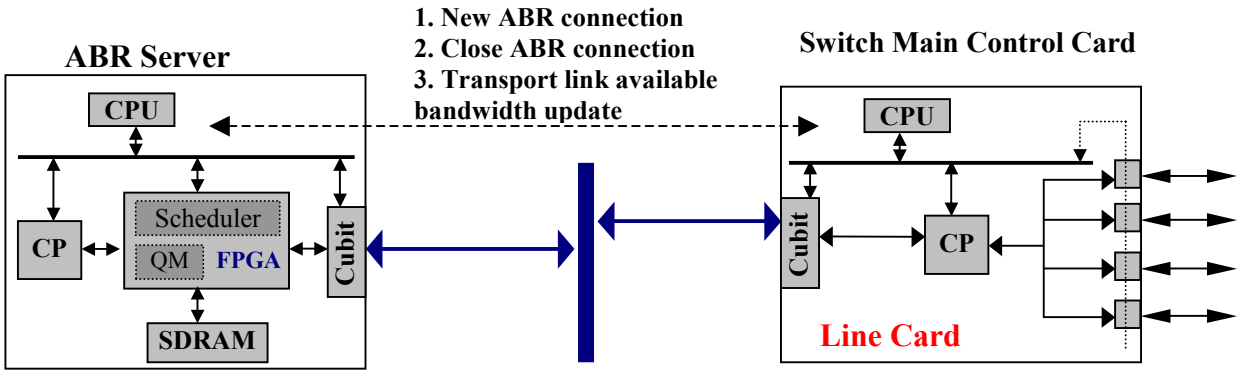


Figure 5. Status information conveyed to ABR server

2.3 ABR server card

The block diagram of the ABR server card is depicted by figure 6. This card concentrates and schedules all ABR traffic of the switch, thus it has sufficient storage capacity to store several tens of milliseconds of traffic for every ABR connection. For this reason it includes a large SDRAM storage (a 64-bit DIMM of 256 MB).

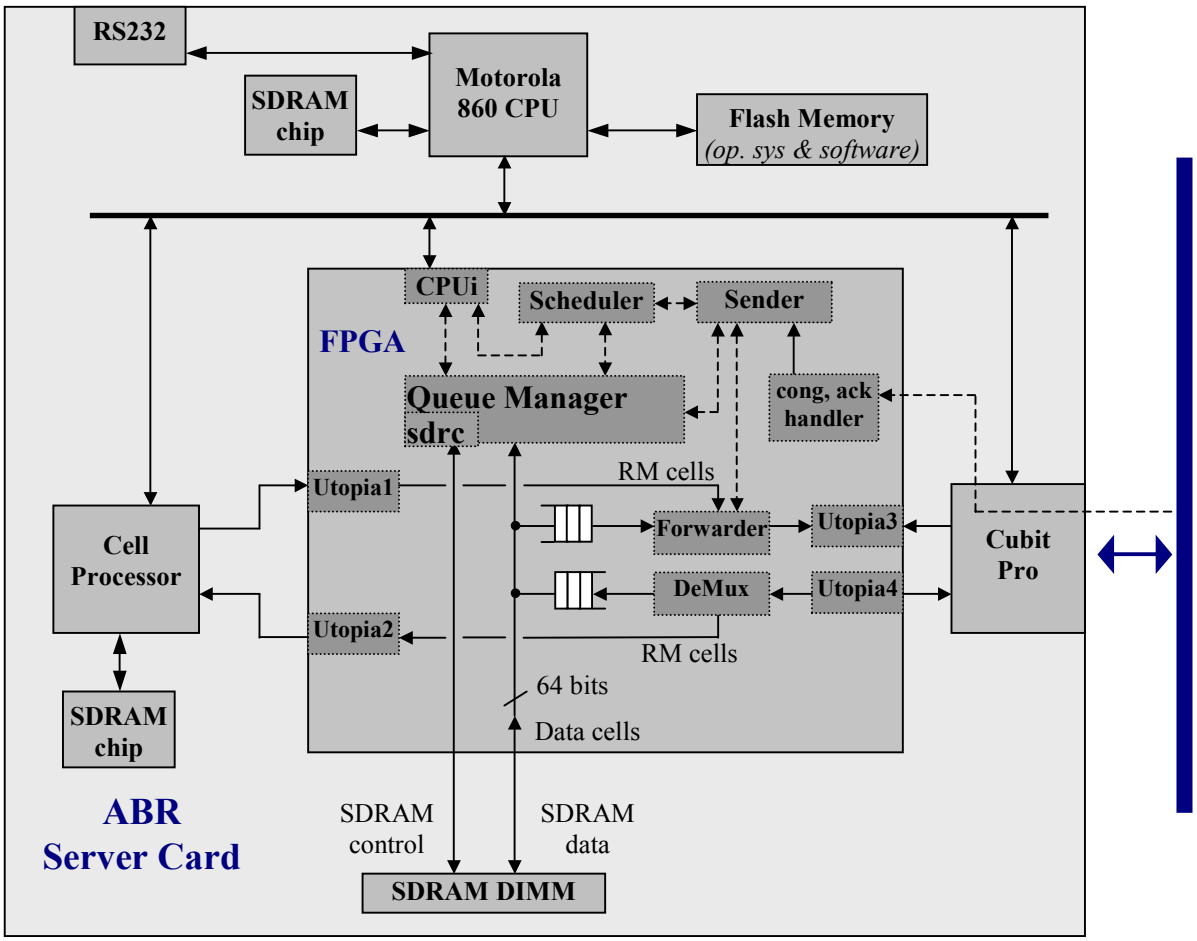


Figure 6. ABR Server Architecture

The scheduling and queuing functions of the ABR Server are implemented in an Altera FPGA, specifically the model EPF10K200EBC600-2. Also four utopia interfaces are implemented in this FPGA for talking to the Cubit Pro and Cell Processor chips. In particular, the interfaces talking to the Cell Processor feature an 8-bit width and emulate the ATM layer side of a utopia transceiver, while the interfaces talking to the cubit feature a 16-bit width and emulate the Physical Layer side of a utopia transceiver. There exists a fifth interface for talking to the CPU. This interface is implemented by module CPUi and allows the processor to control the scheduling and queuing functions of the ABR Server. In particular, it allows the scheduler and queue manager modules to communicate with the CPU, the scheduler receives its link speed updates while the queue manager receives commands such as setting up or discarding queues of ABR connections. Also through this interface the CPU can monitor and profile the queuing and scheduling mechanisms.

Incoming cells arrive from utopia3 are handled by the DeMux module, which optionally passes RM cells to the utopia1 for transmission to the Cell Processor, while the data cells to the 64-bit Cell Enqueue FIFO for writing them to the external SDRAM device. The queue manager recognizes the cell stored in the Cell Enqueue FIFO and adds it to the appropriate queue in the SDRAM. Outgoing traffic is either updated RM cells or data cells coming from memory. The RM cells come from utopia2 while data cells come from the 64-bit Cell Dequeue FIFO, which stores the cells that are read from memory. The forwarder module is responsible for passing the outgoing traffic to the utopia4, which in turn will hand it over to the Cubit Pro for transmission over the CellBus.

The scheduling of outgoing ABR traffic is handled by the scheduler module, which gives eligible for service flow group ids to the sender module. The sender, after performing some checks, requests the queue manager for a cell of an appropriate ABR connection. The queue manager will read, at its convenience, a cell from memory and store in Cell Dequeue FIFO. This FIFO along with the Cell Enqueue FIFO allows the queue manager to use bursts for reading and writing cells. Despite the fact that these FIFOs are expensive to implement in the FPGA, their usage was necessary in order to avoid crippling the SDRAM performance.

3. Scheduling policy and design

3.1 Types of available bandwidth for ABR traffic

In order to schedule ABR traffic, we need to calculate the unutilized bandwidth of each output link, but this is not really the job of the scheduler. This burden belongs to the entity performing Call Admission Control (CAC). CAC entity is always aware of the output link bandwidth not allocated to CBR, VBR and ABR connections, because it needs it in order to accept or reject an incoming connection. However, it is not necessary to explicitly calculate the part of output link bandwidth allocated but not actually used, because this has little usefulness for performing CAC, since it involves current time and thus no certainty that it will apply during the near future. Furthermore for this kind of bandwidth to be calculated it is required dynamic monitoring of output links, or some other smart estimation mechanism, both are quite difficult with our current architecture. Provided that the scheduler is dynamically notified of the sum of both types, that is the sum of the unallocated bandwidth and that allocated but not used, it is feasible to fully exploit all output links even when multiple ones that use the same Cubit pro device.

If the second part of the sum is unknown, the scheduler is forced to rely only on the knowledge of the first part for scheduling transmissions, and thus to avoid head of line blocking that can occur when multiple output links are using a single FIFO. Lack of knowledge of the second part is not a problem for output links that do not share their FIFO. For these links full utilization can be achieved by using the backpressure from the corresponding bus device and employing a suitable back off policy. The scheduler only needs to know the maximum transmission bandwidth of each such link.

3.2 Aggregation mechanism

In order to construct a feasible design, scheduling per connection is avoided. An appropriate aggregation scheme is adapted to achieve this goal. We group the ABR connections into flow groups and thus reduce the number of scheduled objects. The aggregation of the connections is done per output link and Weighted Round Robin (WRR) algorithm is deployed for scheduling the resulting flow groups.

For every flow group of an output link that does not share its output FIFO (i.e. its bus device), we schedule it at the bandwidth capacity of the output link and back off as necessary when congestion occurs at the Cubit-Pro. We employ back off for these flow groups to avoid unnecessary retransmissions, thus wasting shared bus bandwidth and unnecessarily burdening the queue manager. When a flow group corresponds to a link that does share its bus device (i.e. there are multiple links on its line card), the scheduler is dynamically notified of the available bandwidth of the link. This aims to avoid Head of Line Blocking [Figure 7], assuming these updates are correct and fast.

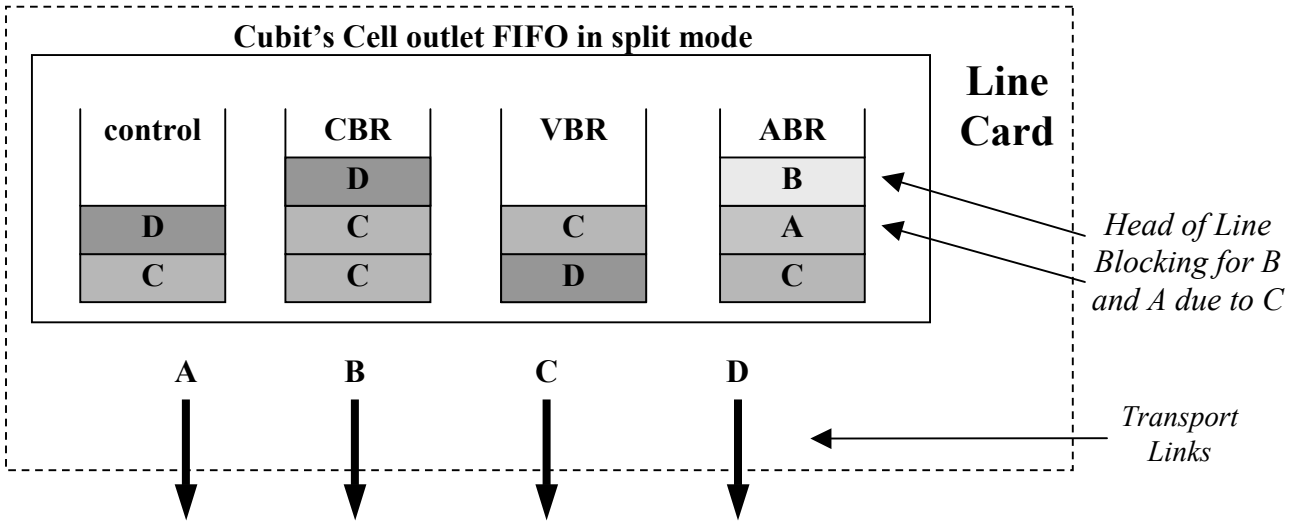


Figure 7. Head of Line Blocking

Because it is critical to resolve HLB occurrences quickly, we decided to use exponential back off / restore policies for modifying the service speed of flow groups that experience congestion. This means that a flow group that sees congestion at its output link FIFO will quickly back off, so as to facilitate the removal of the HLB. Of course, all flow groups that go through the congested Cubit Pro will see the congestion and will back off, although they may not be the ones that caused it. We make the assumption that the flow group(s) that caused the HLB to the output FIFO resulting in congestion (FIFO almost full or full) will be soon updated to their correct speeds, thus the HLD occurrences that led to FIFO congestion will be resolved. The exponential restore policy helps to avoid further unnecessary punishment of the non offending flow groups.

3.3 Decoupled Scheduler Design

Our scheduler features a decoupled design, which consists of the "scheduler" and "sender" modules. The scheduler module performs the actual scheduling and employs back off when congestion occurs. The sender module is responsible for issuing commands to the queue manager, handling retransmissions, identifying free cells locations and sending data and control cells to the Cubit Pro (bus device).

For the link-aggregation to work, the queue manager maintains a round linked list of the non-empty connections of every output link. Each time a flow group is entitled service, the scheduler enqueues it to a FIFO called the Eligible FIFO. The sender module dequeues from this FIFO and after performing some checks, it issues a service request to the queue manager for a cell of a connection that belongs to the particular flow group. The queue manager [4] performs RR (round robin) between non-empty connections inside the flow group, choosing one cell from every flow before advancing to the next one.

For every cell that we attempt to transmit over the CellBus, the sender maintains a pointer to its memory location. This is done because the queue manager dequeues the cell from its corresponding queue. Thus, if the transmission over the shared bus fails, then the sender can provide the queue manager with this pointer when the flow group becomes eligible for service again. It is reasonable to assume that transmission failures over the CellBus will rarely occur due to CRC error, and that almost all transmission failures will be due to full FIFO at the destination Cubit Pro. We will see at chapter 9 how we can take advantage of this to improve the sender module.

So, the sender issues 2 types of requests to the QM, either a dequeue request for a particular flow group and provides a flow group id, or a cell read request and provides the appropriate memory pointer. If a transmission is successful, the queue manager is notified that the memory location can be added to the free list. This is done through a small Free Cell pointer FIFO inside the sender module, which is written by the sender and read by the QM.

In this design we made the assumption that each ABR connection requested (or the CAC forced it) a zero Minimum Cell Rate (MCR=0). If an application requires a Cell Rate guarantee, it must get through a CBR or VBR connection. This design can be modified to support MCR through the use of MCR-aggregated flow groups. For this purpose the sender module must go under major changes while the scheduler module needs only minimal changes, this is thanks to our decoupled architecture.

4. Scheduler module

4.1 Implementation alternatives

The scheduler module implementation features 5 small dual-ported memories with one dedicated input port and one dedicated output port. These memories hold all the necessary information for the execution of the scheduling algorithm, and are implemented using Embedded Array Blocks (EABs), found in the FPGA (Altera Flex10KE FPGA family). Also, the scheduler module utilizes a four-stage fixed length pipeline, which is traversed for every flow group that we want to examine for service eligibility. One such examination is performed per clock cycle and each examination requires three cycles to complete. So, the required time to check N flow groups for service eligibility is N cycles. The number N is defined as our virtual time unit. This polling-like implementation was preferred over other alternatives due to speed and cost concerns. The two other alternatives [3][6][10][11][13] that we considered were a systolic-buffer [5][8] and a heap-like scheduler [9].

a. Systolic-buffer scheduler

The systolic buffer [5][8] is very demanding of logic cells. A experimental implementation [Figure 8] of systolic-buffer scheduler module showed that for 64 flow groups it requires more than twice the logic cells and achieves half the clock frequency when compared with our current one. The increased demand for logic cells is due to the fact that our scheduler is non-work conserving, thus the time field must be large to achieve the required accuracy. This means increased demand for logic cells to hold the time values and for logic cells to implement the comparator in each systolic buffer. The reduced clock frequency comes as a result of the shared bus, and although there are suggestions how to remove this limitation they do not deal with the increased demand for logic cells.

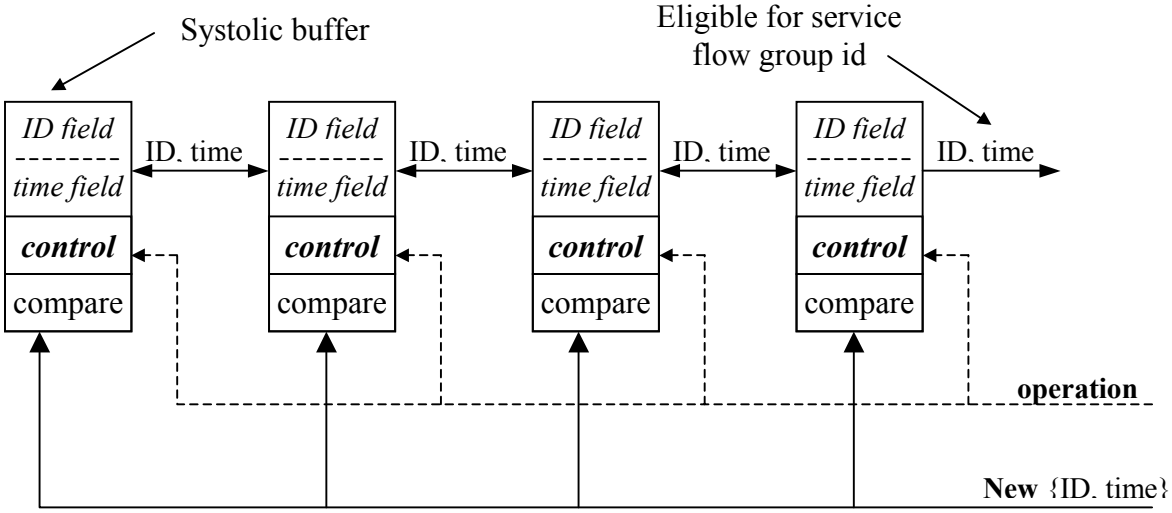


Figure 8. Systolic-buffer scheduler

b. Heap-like scheduler

Our other option was a heap-like implementation [Figure 9]. Examination has shown that such an implementation is feasible using about the same number of EABs, nevertheless it would only yield better results for 128 or more flow groups [9], because this is the point that our polling-like implementation starts to feel the burden of continuously checking every flow group for service eligibility. But also at this point the maximum capacity of the EABs is reached, so we would have resort in external srams. In such a case the heap-like implementation would be indeed necessary.

For a heap-like implementation we can exploit the fact that we have a predefined number of flow groups, to drop a main characteristic of the heap its dynamic memory allocation. We can allocate each flow group entry statically according to its id number, e.g. to use the id number as the address of the sram memory location containing the flow group entry. Two flow group ids stored in each such entry can be used to depict the left and right children. Furthermore, the fact that the flow groups are usually all active, shows us that we can choose to support only two operations on the heap, the GET_MINIMUM and UPDATE_HEAD operations. No DELETE or INSERT_NEW would be necessary, instead the flow groups would always be positioned inside the heap. A second SRAM is anyway needed to keep the Back-off information, this SRAM could also be used to hold empty, congestion and nack flags. The empty and acknowledge flags, along with the back off information, can be used to indicate whether the GET_MINIMUM has returned an eligible for service flow group.

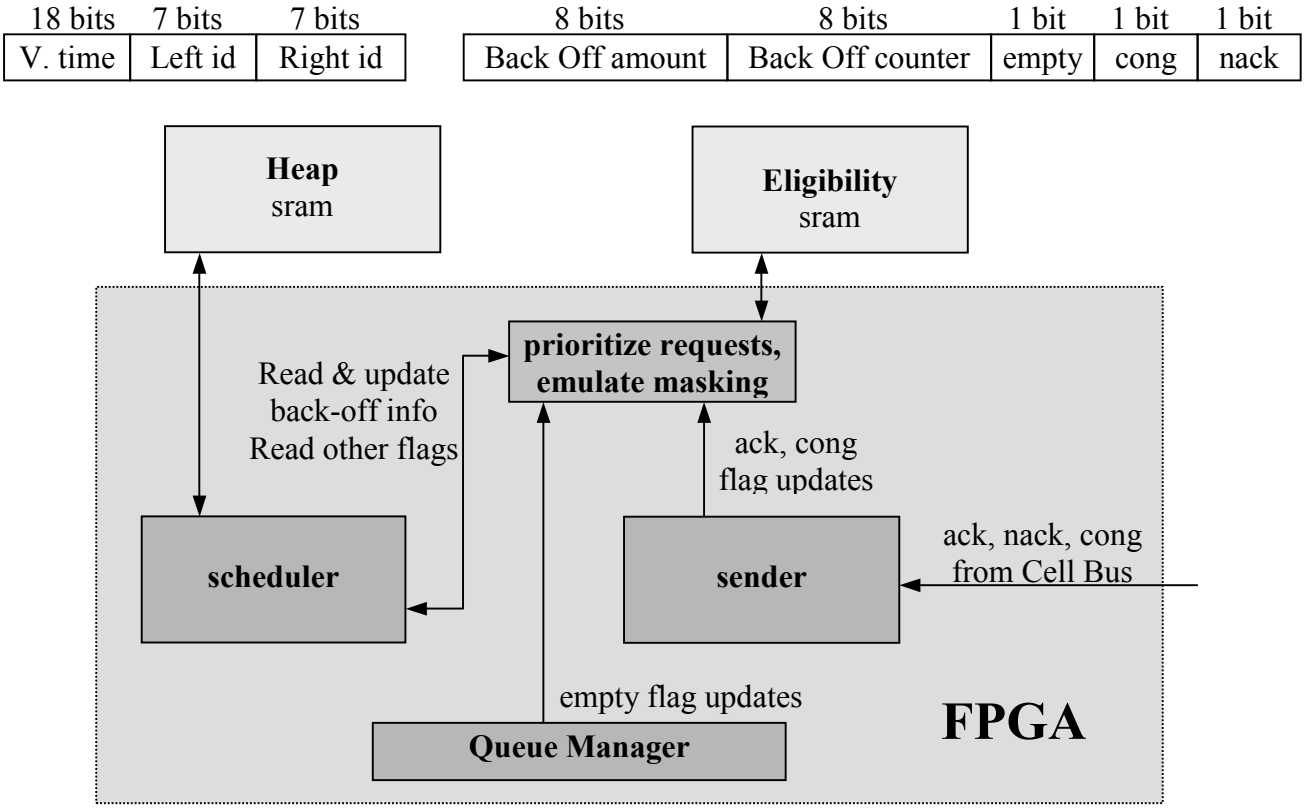


Figure 9. Heap-like scheduler

The heap-like implementation would offer good results but there is little to gain in implementing it using EABs, since it would only shows it potential for large number of flow groups, that is 128 or more. In fact for 32 flow groups or fewer it performance is lacking compared to the polling-like scheduler.

4.2 The Polling-like scheduler

a. Memory components

We called this scheduler polling-like because it examines all flow groups one by one for service eligibility. In order to understand how this scheduler works, we need to describe the utility of each of the five dual-ported memories it uses:

- *Service Interval memory*
  Holds the time interval between two successive services of the flow group. This effectively defines the speed at which each flow group is served. The interval is maintained in virtual time rather than actual time. A single virtual time unit is N clock cycles, where N is the maximum number of flow groups the scheduler can serve. The interval values stored in this memory may be decimal and in general they are in order to achieve a fine grain scheduling accuracy.

- *Service Time memory*
  Holds the time at which each flow group becomes eligible for service, while it is in non congested condition. In order to decide whether a flow group is entitled service, the integer part of its service time is compared to the current virtual time. The virtual time progresses in integral time units. For every flow group that its service time is reached, the new service time (NewServiceTime) is computed as follows:
  (OldServiceTime + ServiceInterval) mod MaxServiceTime.

- *Back Off memory*
  For every flow group this memory holds the current "back-off amount", which shows the number of times we have backed off the normal speed and the "back-off counter". Every time the service time is reached, the back off counter, if not already zero, is decremented by one till it reaches zero, then the particular flow group becomes eligible for service. The way we update the back off amount defines our back off policy. We have decided to use exponential back off and restore, since it overall gives the best results.

- *Empty Memory*
  Holds information about which flow groups are "empty", meaning that none of the connections that belong to the particular flow group have currently any cells for transmission. In case that this memory is not quickly enough updated with empty flow groups, the scheduler may issue service requests for empty flow groups to the Queue Manager. In this case the Queue Manager should just ignore them.

- *Congestion and Negative Acknowledge memory*
  This memory holds information about which flow groups are experiencing congestion and / or failed to transmit a cell over the shared bus. Every time a flow group becomes eligible for service while having congestion, the back-off amount is doubled, else it is halved. We note that a particular flow group can become eligible for service when it is either non-empty or have a failed transmission attempt.

b. Eligible FIFO

Our design also features a FIFO which holds the ids of the flow groups that are entitled service, this is called the "eligible FIFO" and depending on its size, it might be necessary to implement it in an EAB. We don't expect this FIFO to get full due to service time synchronization, since we have further reduced this probability by randomly selecting the low order bits of the service intervals. Nevertheless the FIFO can get full if the aggregate speed of all flow groups surpasses the maximum capacity of the Queue Manager. Should such a situation occur, we think that the flow groups should be punished proportionally to their speed. So instead of dropping service requests and just punishing some flow groups, we choose to stall the scheduler and thus achieve a "fair" punishment to all flow groups.

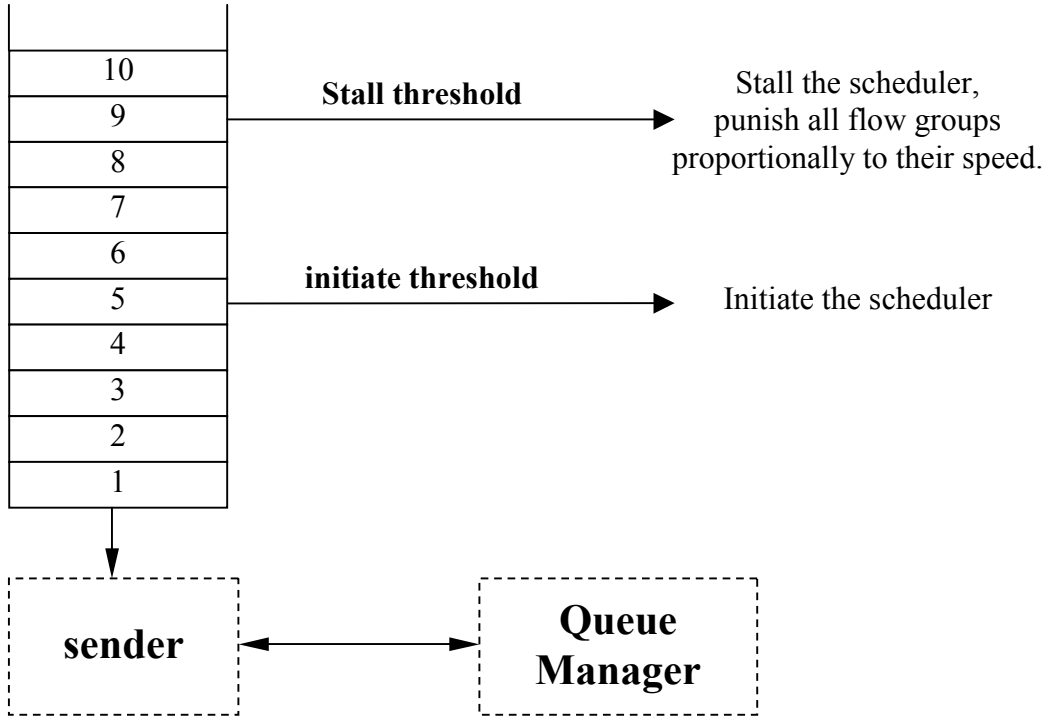


Figure 10. Stall and Initiate Thresholds

For stalling and reinitiating the pipeline we define 2 thresholds, the **stall threshold** and the **initiate threshold** [Figure 10]. Let k be the size of the eligible FIFO. We defined k-1 as the stall threshold and k/2 as the initiate threshold. Although we stall at k-1 the next flow group check is not discarded so the FIFO may get full. For example if the flow groups I and I + 1 are service eligible at a particular virtual time and the enqueue of I in the eligible FIFO triggers the stall threshold, the flow group I + 1 will be enqueued before pipeline is stalled and flushed.

c. The 4-stage Pipeline logic

As we have said previously the scheduler module utilizes a four-staged, fixed-length pipeline. At stage one all memories are read using the same address. This address is a flow group id which at every positive clock edge is incremented by the flow group counter. By saying read we mean that the read address is registered in by the memory components. The results will be latched out at the next clock edge. So at the third stage the results latched out by the memories are used to evaluate the service eligibility of the current flow group. Also part of the new values of the service interval and back off memories are prepared.

At final stage (e.g. stage 3), the service time and back off memories are updated, and the flow group id is enqueued in the eligible FIFO, provided of course that it is eligible. These operations could not be performed at stage three or they will increase the clock cycle period. Stage three already contains the output delay of the memories and delay of some combinational logic, so it is preferable to locate the multiplexers and the setup delay of the memories and the eligible FIFO at a fourth stage.

As we have said above the pipeline is stalled when the Stall threshold is reached by the eligible FIFO. While being in this state both the virtual-time and the flow group counters cease to advance. As real time progresses all flow groups receive a punishment proportional to their speed. As soon as the Initiate threshold is passed the these counters start to operate again and scheduling algorithm continues as if the pipeline has never been stalled.

d. Putting it all together

The block diagram of the scheduler is shown in figure 11. This block diagram consists of the 4 stages of the pipeline, the memory components and the eligible fifo. Also, this figure shows the four modules that interface with the scheduler. The CPUi module provides the service interval updates which are calculated by the CPU software. The sender and queue manager modules update the congestion/nack and empty flag memories respectively. Of course the sender module also controls the read port of the eligible FIFO.

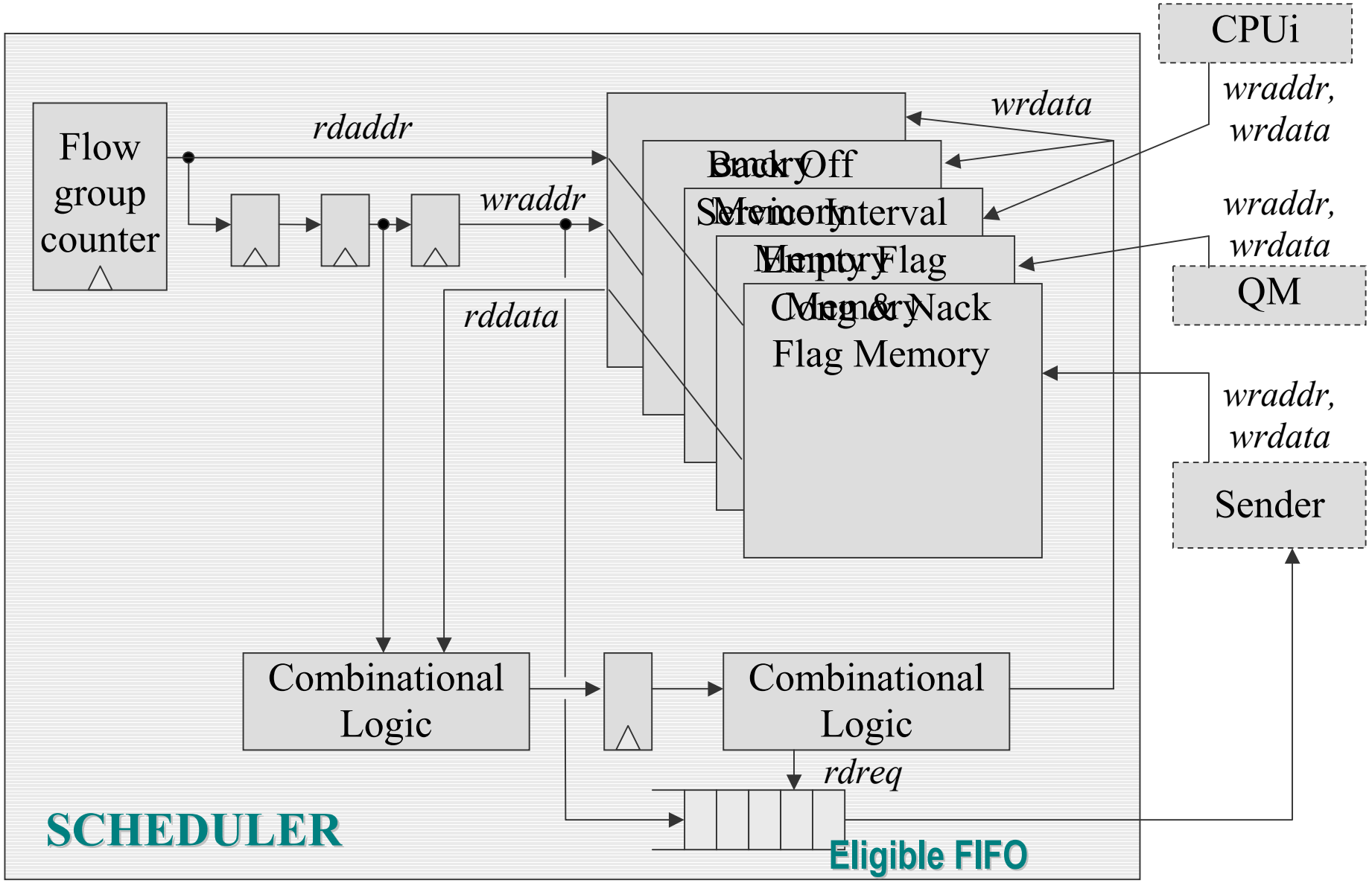


Figure 11. Polling-like scheduler

4.3 Service Intervals

The time intervals were chosen to be non-integer for solving two problems. The first one concerns the accuracy of the scheduler in high speeds. Should we have chosen to use integer time intervals, the accuracy of the above scheduler in high speeds would be disappointing. This is because such a scheme would only be able to distinguish multiples of the N clock cycles. So, the fastest service interval that a particular flow group could attain would be 1, effectively having 1*N clock cycles as service time interval. The immediately slower attainable service interval would be 2, since the flow group would have to be scheduled at 2*N clock cycles as service time interval.

The second issue that we are able to solve by employing non-integer time units is that of service time synchronization. Although this would rarely occur, it is a problem once it occurs, since it leads in many flow groups becoming eligible for service at the same virtual time. This results in a full Eligible FIFO, which effectively stalls the scheduler and punishes all flow groups. It also introduces further delays for the departing cells since they do not require the queue manager when it is idle but when it is pretty busy. So we need to avoid this synchronization to prevent the scheduler from stalling.

By using decimal time intervals we are able to schedule the flow groups at an accurate enough speed. Although we only use the integer part of a service time to decide that a flow group is eligible for service, the decimal part is not discarded but instead used to calculate the next service time. Thus a flow group can now attain any speed not just multiples of N clock cycles. Furthermore, the low significant bits of the decimal part could be chosen randomly to make the probability of service time synchronization even smaller. The fact that we effectively alternate between two integral service intervals for a flow group doesn't pose any problem since our flows are ABR flows and the flow group could alter speed many times before a complete round of all connections is completed. Figure 12 shows an example of a flow group scheduled at 2.3 time units, having 4 non-empty flows A, B, C and D. The arrows show the virtual time at which the flow group is serviced, while the numbers above the arrows are the scheduled service times. The top row depicts the flows being serviced.

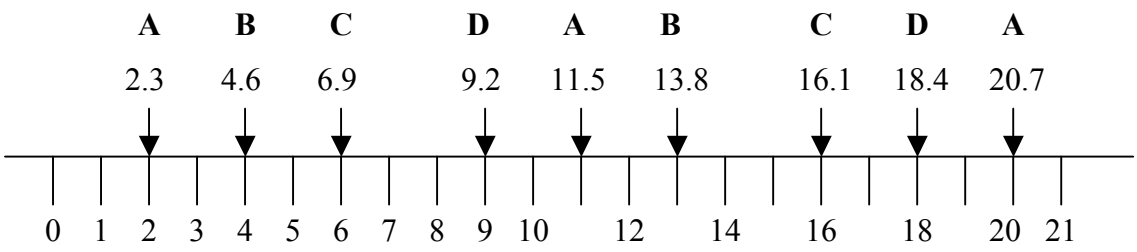


Figure 12. Flow group schedule example

4.4 Service eligibility

In order for a flow group to become service eligible and thus enqueue its id in the eligible FIFO, some conditions, other than just a matching service time must hold. Figure 13 shows the service eligibility scheme that decides if "current" flow group should be enqueued in the eligible FIFO.

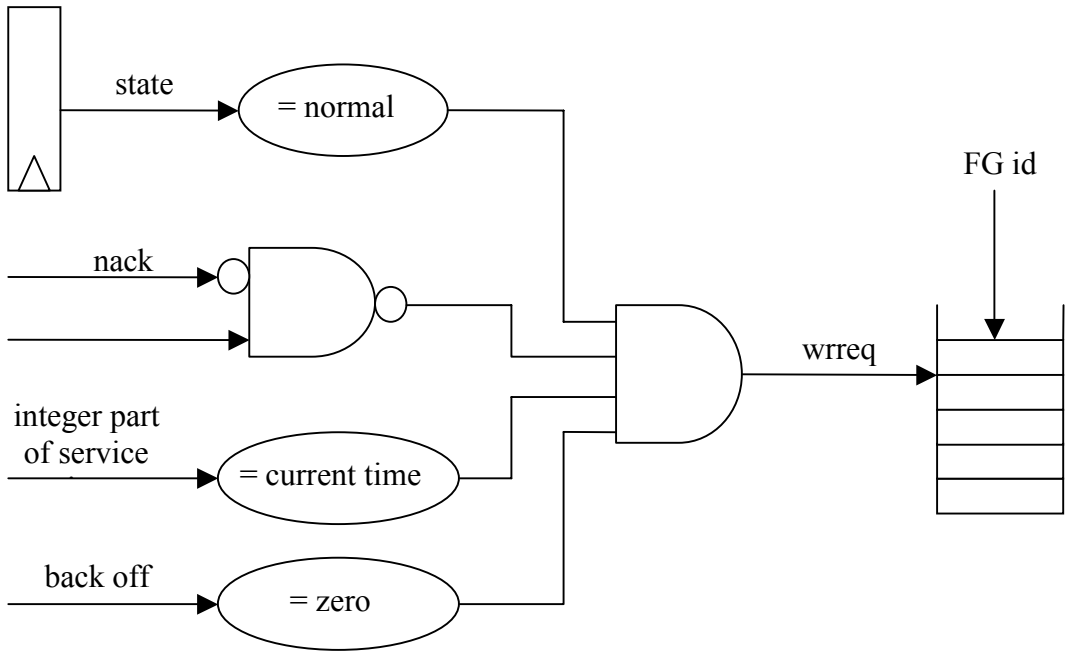


Figure 13. Service eligibility scheme

As the above figure shows a flow group is considered eligible for service if its service time has been reached and any pending back off has finished. Furthermore, the flow group must be non-empty or have a failed cell transmission. A non-empty flow group is one having at least one non-empty flow. The back off of a congested flow group is considered finished when its back off counter has reached zero.

4.5 Congestion and Negative Acknowledge handling

A congestion indication is returned to the source bus interface when a cell transmitted over the shared bus (CellBus) causes the FIFO of the destination bus interface (Cubit-Pro) to become almost full. This congestion indication is matched with a flow group id by the sender module and the scheduler is notified that the particular flow group is experiencing congestion. The same process takes place when we have a transmission failure over the shared bus. In fact, for every transmission the sender matches both signals with a flow group id and the scheduler is notified of the status of the transmission. This is so that the previous congestion and negative acknowledge bits can be cleared.

5. Sender module

5.1 Overview

Prior to issuing dequeue requests to the Queue Manager, an additional check needs to take place to decide whether it is a new transmission or a retransmission of a failed one. This is the responsibility of the sender module. It is this module that issues dequeue or cell read requests to the queue manager, sends cells to the Cubit-Pro for transmission over the CellBus, and interprets the responses of the Cell Bus. Should we have included these tasks to the scheduler module it would have been unnecessary more complex and difficult to design. The sender module communicates with four modules, the scheduler, the queue manager, the forwarder and the cong-ack handler modules. The block diagram of the sender module along with its interfaces are depicted in figure 14. The components and interfaces of this figure are explain in the subsequent paragraphs of this chapter.

The FSM transitions are shown in figure 15. What makes this FSM more even complex is the fact we need to perform read-modify-write in two occasions. This is because each cell pointer is stored in two consecutive words, the low 16 bits take one memory word while the high order are stored along with the pointer status bits. This despite the fact that it makes the FSM of the sender more complex it does reserve one EAB to be used by other modules. This choice was made because at the time of the sender design it was unknown if the number of EABs would suffice for all modules. Summarizing the usage of EABs we note that the scheduler uses 7 EABs for its memories possibly one more is needed for the eligible FIFO. This because if the eligible FIFO is implemented with logic cells it cannot be larger than 7 entries or it would limit clock speed. This also applies to the Cell Free FIFO and Cell History FIFO used by the sender. Also each of the four utopia interfaces utilizes one EAB. Furthermore the queue manager module uses three, while the each of the 2 SDRAM fifos takes four EABs. These make a total of 24 EABs, which is the number of EABs in our FPGA.

Following we will describe the interfaces of the sender, provide a short description of the forwarder module, and explain in detail the tasks that the sender module performs, along with justification of for our implementation choices.

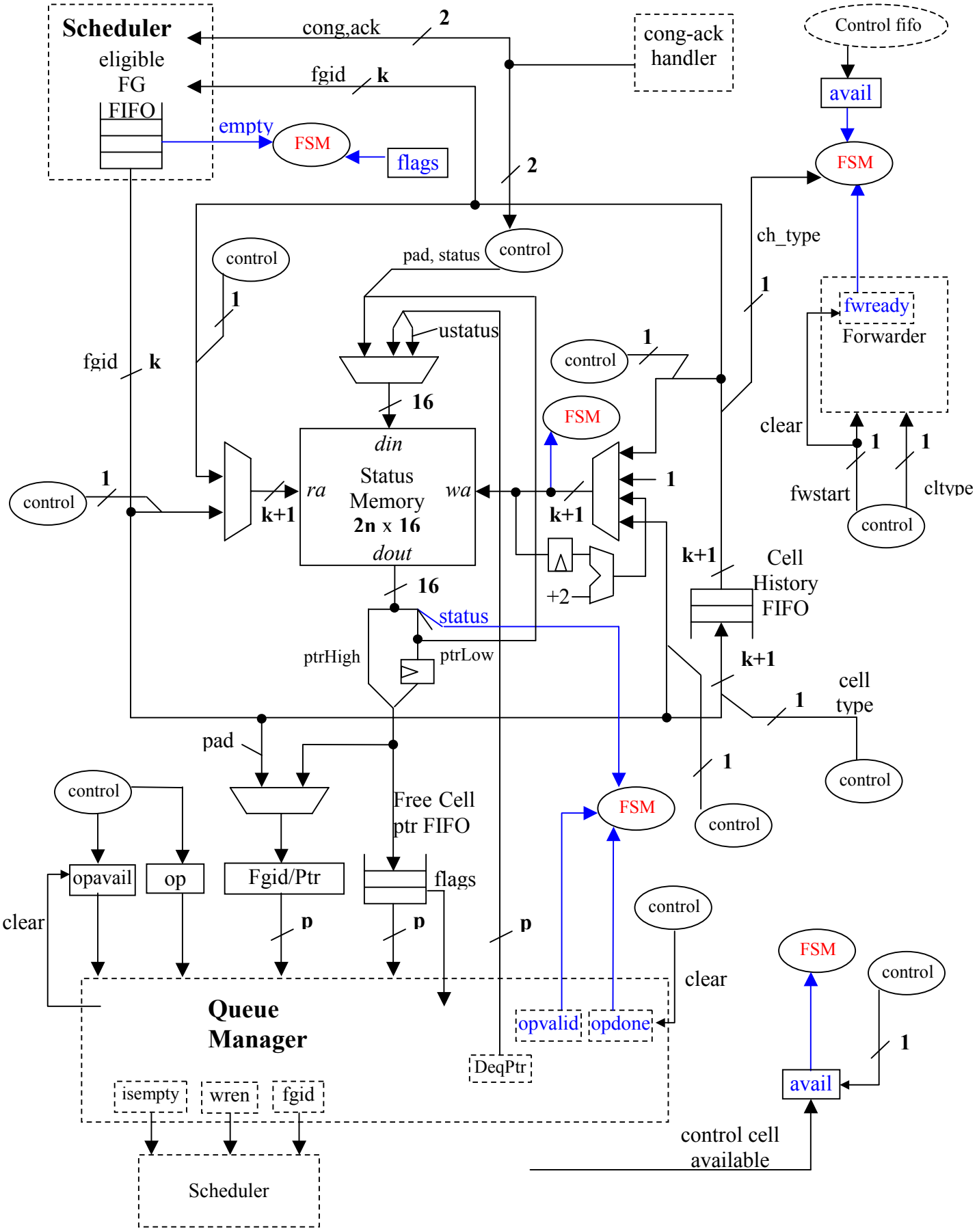


Figure 14. Sender block diagram

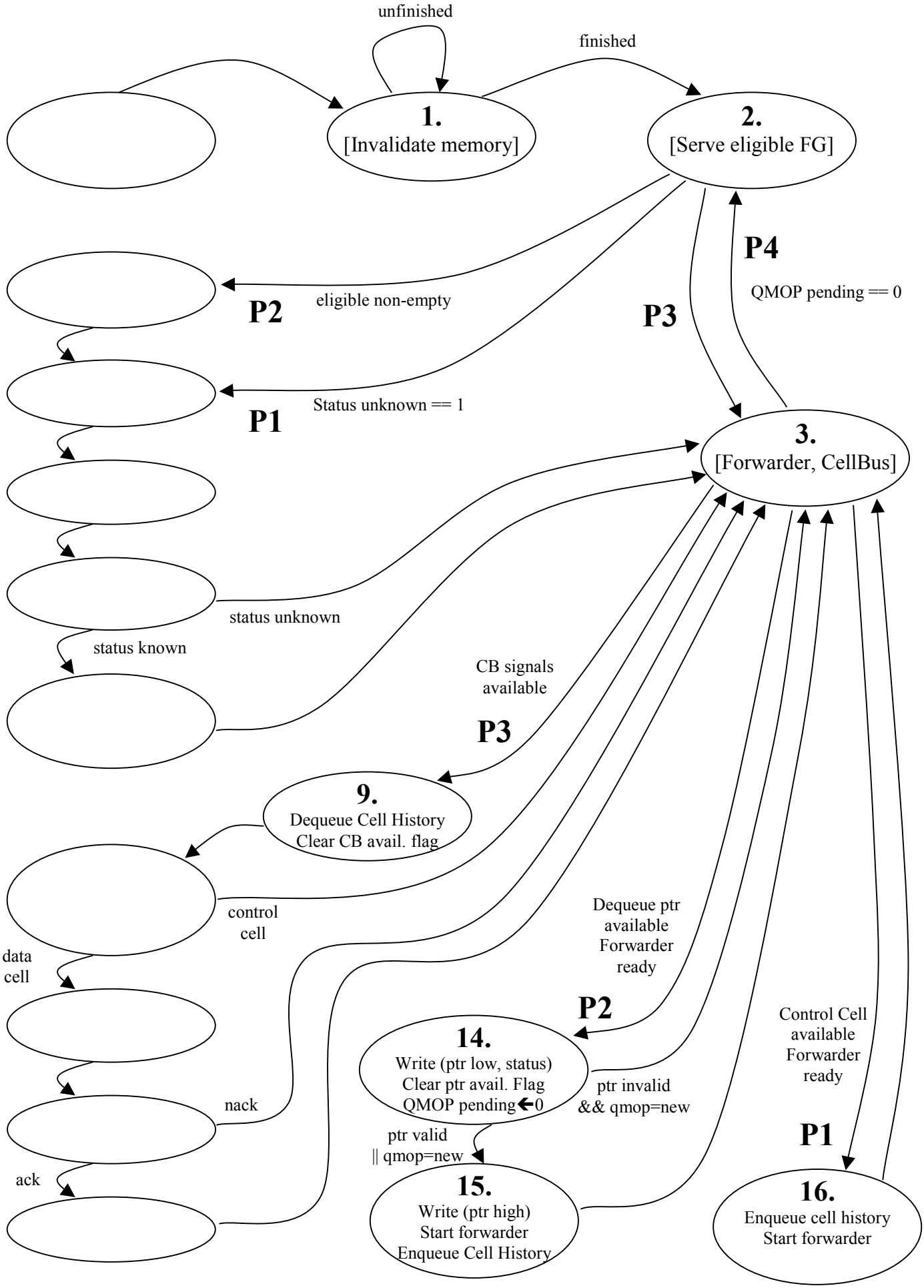


Figure 15. FSM of the Sender module

5.2 Sender Module Interfaces

Interface with Scheduler module

Sender dequeues flow group ids from the **eligible FIFO** output port, and after checking the status of the particular flow group issues a command to the queue manager. The interface with the scheduler also includes the updates for the status of a flow group. There is one such update per cell transmission of the flow group. The information conveyed to the scheduler module is the **congestion and negative acknowledgement** signals of the cell bus. Of course these signals are passed to the scheduler along with the corresponding flow group id. These signals may be provided to the scheduler at any time as they become available.

Interface with Queue Manager module

The interface with queue manager is somewhat more complex. The sender may issue two types of different commands to the queue manager. The first type is the *dequeue request***,** which is used for new cell transmissions. While the second type is the *cell read request*, which is used for cell retransmissions.

The dequeue request is made by setting the *opavail* flag, setting the *op* register to "dequeue" and providing a flow group id in the *Fgid / Ptr* register. When the queue manager has executed the operation, it sets the *DeqPtr* register to the pointer of the newly dequeued cell. It also sets the *opdone* flag to indicate that the operation has completed and the *opvalid* flag to indicate that the operation was successful and the cell pointer is valid. The queue manager may clear the *opavail* flag at any time but prior or concurrently with setting *opdone* flag. Also the sender module clears the *opdone* flag at its convenience but prior to setting *opavail* flag.

The cell read request is similar to the dequeue request. The *opavail* flag is set, the *op* register is set to "read" and a cell pointer is provided in the *Fgid/Ptr* register. When the queue manager has served the request it sets the *opdone* flag. The *opvalid* flag and the *DeqPtr* register are ignored as a cell read request may not fail and the cell pointer is already known.

Interface with forwarder

The forwarder is a simple module that forwards either data or control and RM cells to utopia module talking to the cubit pro device. The forwarder provides the *fwready* flag to notify the sender that it is available. The sender starts the forwarder through an *fwstart* pulse, which also clears the *fwready* flag. Along with the *fwstart* pulse the *cltype* signal indicates whether the forwarder is to handle a data or control/RM cell. The forwarder will set the *fwready* flag as soon as it finishes the operation.

5.3 The Forwarder Module

The data cells come from the 64-bit Cell Dequeue FIFO, which is necessary for reading a cell from memory using burst transfer, while the control/RM cells come from a 8-bit FIFO of a utopia interface talking to the cell processor. The outgoing utopia interface talking to the cubit pro device uses a 16-bit FIFO. The forwarder module does the necessary buffering to adjust between these different widths and controls the timing of the above FIFOs appropriately.

5.4 Sender Module Main Components

The status memory

After dequeueing a flow group id from the eligible FIFO, we need to decide what type of command we should issue to the queue manager, this is done by indexing the status memory using the flow group id. The status memory holds a 2-bit status and a 22-bit cell pointer value for every flow group. These make a total of 24-bits. Since the number of EABs in our FPGA is limited and the maximum width of an EAB is 16 bits, we choose to use only one EAB to implement this memory, thus we store the 24 bits in 2 consecutive addresses. This made our design more complex. In order to read the pointer we must read 2 memory locations, also modifying the status may require to read the pointer bits first, effectively doing a read-modify-write sequence, when we would prefer to do just a write operation.

The status bits may indicate 3 different states of the pointer bits; these are *valid, invalid* and *unknown*. An invalid status indicates that last cell transmission of the flow group succeeded, meaning that the pointer is invalid and a new cell should be dequeued. A valid status indicates that last cell transmission of the flow group failed so the cell should read again and retransmitted. The state could also be unknown, meaning that the cell has not yet been transmitted over the CellBus. In this case we wait for the cell to be transmitted and get the result of the transmission, the alternative is not to serve the flow group and continue with the next one.

The situation where a flow group becomes eligible for service and its id is dequeued from the eligible FIFO before its last cell gets transmitted over the CellBus is quite problematic since we do not know what type of command to issue to the queue manager. Waiting for the CellBus outcome means that sender will stall, effectively throttling the speed of the flow group. Moreover the eligible FIFO may eventually get full and thus stall the scheduler and punish all the flow groups. This situation may occur only for the faster flow groups, how fast depends on the latency to get a cell from memory to the CellBus. There is no easy way to deal with this problem, but there is a solution that we will suggest later. Anyway its preferable that when this problem occurs we do not serve the flow group and instead dequeue the next flow group id, this prevents the eligible fifo from getting full but doesn’t remove the throttling effect.

Free cell pointer FIFO

For every successful transmission of a cell over the CellBus the corresponding cell pointer must be added to the free cell pointer list maintained by the queue manager. Because we may have more than one pointer freed in a relatively short period of time we use a FIFO to hold them. The queue manager dequeues from this FIFO to add the pointer to the free list or perform a bypass enqueue. The sender reads the pointer from the status memory updates the status bits to *invalid* and enqueues the pointer in the free cell pointer FIFO. The FIFO must be large enough so as not to get full. A full FIFO means that the next free cell pointer will be lost and thus the corresponding memory words will be rendered unusable until the system is reset. On the other hand we wouldn't want the queue manager to be too aggressive when adding cell pointers to the free list because such a policy would do a small number of bypass enqueues.

Cell History FIFO

We do not know the exact time that a cell will be actually transmitted on the CellBus. This is because a particular cell is transmitted on the CellBus after reaching the front of the cubit pro FIFO and when the particular cubit pro gets the CellBus by the arbiter. At that time we receive through the cubit pro two signals, a congestion indication and a negative or positive acknowledge. We need to hold appropriate information so as to match these signals with a particular flow group. For this purpose we maintain the Cell History FIFO, which stores the ids of the latest flow groups that have transmitted a cell. For every departing cell we enqueue the corresponding flow group id in this FIFO, and for every pair of CellBus signals that we receive, we dequeue an id and match it with the CellBus signals. Following we update the status memory and issue a notification to the scheduler module.

Controls cells are not subject to retransmissions so we need to ignore them. For this reason the Cell History FIFO is one bit wider, and the extra bit is used as a flag to indicate whether this is a data or a control/RM cell entry. CellBus signals matched with a control/RM cell are ignored. This will be corrected since the correct behavior is to only ignore the negative acknowledge because these cells are not retransmitted. However the congestion information is desirable to reach the scheduler.

5.5 Policy for data and control cells

Control cells are given higher priority than data cells. So while control cells are available the sender module will always direct the forwarder to handle them instead of data cells. We choose this policy because the number of control cells is relatively small and most likely contains important information. We do not support retransmissions for the control cells since it is unlikely that their transmission will fail due to full FIFO, however the retransmission of RM cells is desirable although not implemented. We leave the responsibility of handling a lost control cell to the software.

6. Module Initialization

The scheduler module does self-initialization. The processor only needs to raise the aclr (asynchronous clear) signal for one clock cycle, and wait for a specified amount of time before issuing speed updates to the scheduler. The time needed by the scheduler to self-initialize depends on the number N of groups the scheduler is configured to serve. This time is N+3 complete cycles after the clock cycle that the aclr signal was high. Speed, congestion, and non-empty updates issued to the scheduler before the above time are ignored. The 3 additional cycles are due to the pipelined architecture.

During initialization all flow groups are marked as empty, not congested, with positive acknowledge. The back-off counters are set to zero, while the service intervals are initialized to maximum speed, and the service times to zero. We choose to do this so that the first service interval updates after every reset will take immediate effect., since for a new service interval to take effect the service time for the particular flow group must be reached. The time required by sender module to initialize is not greater that of the scheduler and only requires the same aclr signal. During initialization the sender module invalidates the pointer status field of the status memory. Figure 16 shows the initial values of all memory components.

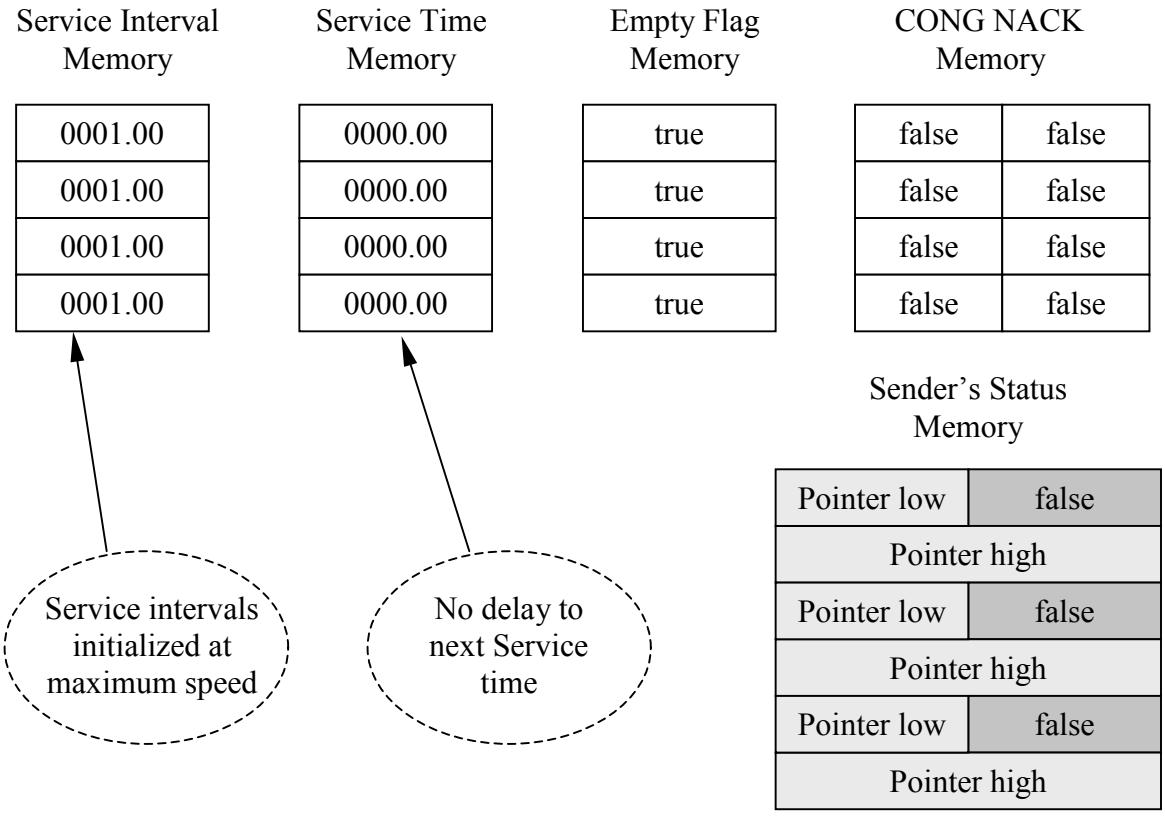


Figure 16. Initial values of all memory components

7. Utopia interfaces

The FPGA of the ABR server uses four utopia interfaces. More specifically the FPGA utilizes:

- One 16-bit Physical Layer Utopia Transmitter.
  This is used to forward outgoing ABR traffic to the Cubit-Pro for transmission over the CellBus.
- One 16-bit Physical Layer Utopia Receiver.
  This used to receive incoming ABR traffic from the Cubit-Pro coming from the CellBus.
- One 8-bit ATM Layer Utopia Transmitter.
  Used to forward RM cells to the Cell Processor for updating the flow control fields.
- One 8-bit ATM Layer Utopia Receiver
  Used to receive updated RM cells from the Cell Processor.

The utopia interfaces were implemented to conform with the utopia-2 standard. Figure 17 and 18 show two timing diagrams for transmitting cells according to the utopia standard. Figures 19 and 20 show the block diagram of the 16-bit Physical Layer Utopia Transmitter και 16-bit Physical Layer Utopia Receiver respectively.

Each block includes two counters, the *word counter* and the *cell counter*. The first one counts the words (either 8-bit or 16-bit) of each cell being transmitted or received, this depends on type of the utopia interface. The second counter counts the cells that are present in the interface FIFO, and is used to perform flow control with the internal module of the FPGA, e.g. the Forwarder and DeMux modules. For example the Physical Layer Utopia Transmitter, produces the signals cellspc (cell space) and ciclav. The signal cellspc, is used to inform the Forwarder module that there is available space in the FIFO for at least one cell, while the ciclav signal allows or prevents the transmission of cell to the opposite utopia interface which is located in the cell processor. The *cell counter* changes its value when the *word counter* indicates a complete cell receipt or transmission, or when a *celldec* or *cellinc* pulse appears from the Forwarder, DeMux modules.

The interfaces support different clock for transmitting / receiving cells over the utopia than that used for internal operation. The *synchro* and *synchro pulse* sub-modules are used to perform the necessary synchronization. The first one just synchronizes its input to the clock of its output, while the second one produces a pulse when it detects a change from 0 to 1. For the utopia interfaces that transmit we use Cut-Through policy while so as to minimize the time require to output the cells, while for receiving we employ store-and-forward. Cut-through means that we start transmitting the cell as soon as its first bytes are available in the FIFO, while store-and-forward requires that the receipt of the cell is complete before it is forwarded.

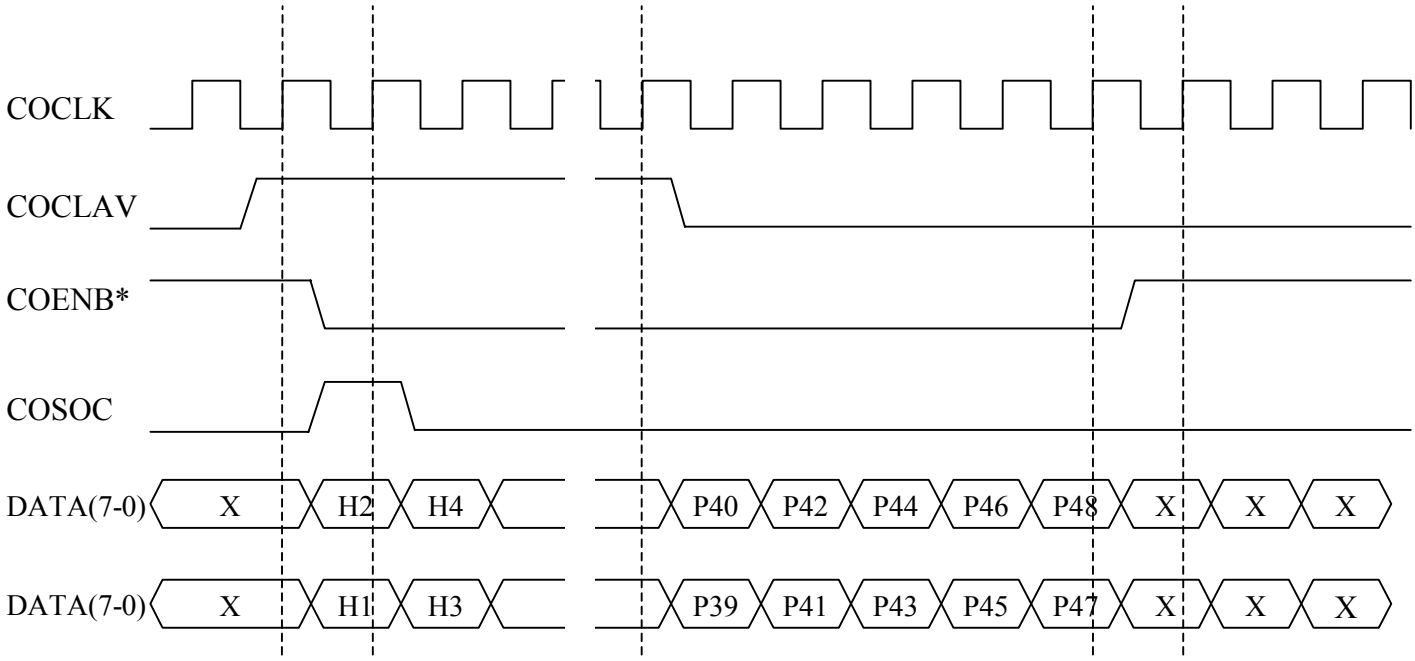


Figure 17. Utopia timing diagram, coclav prevents the transmission of a second cell

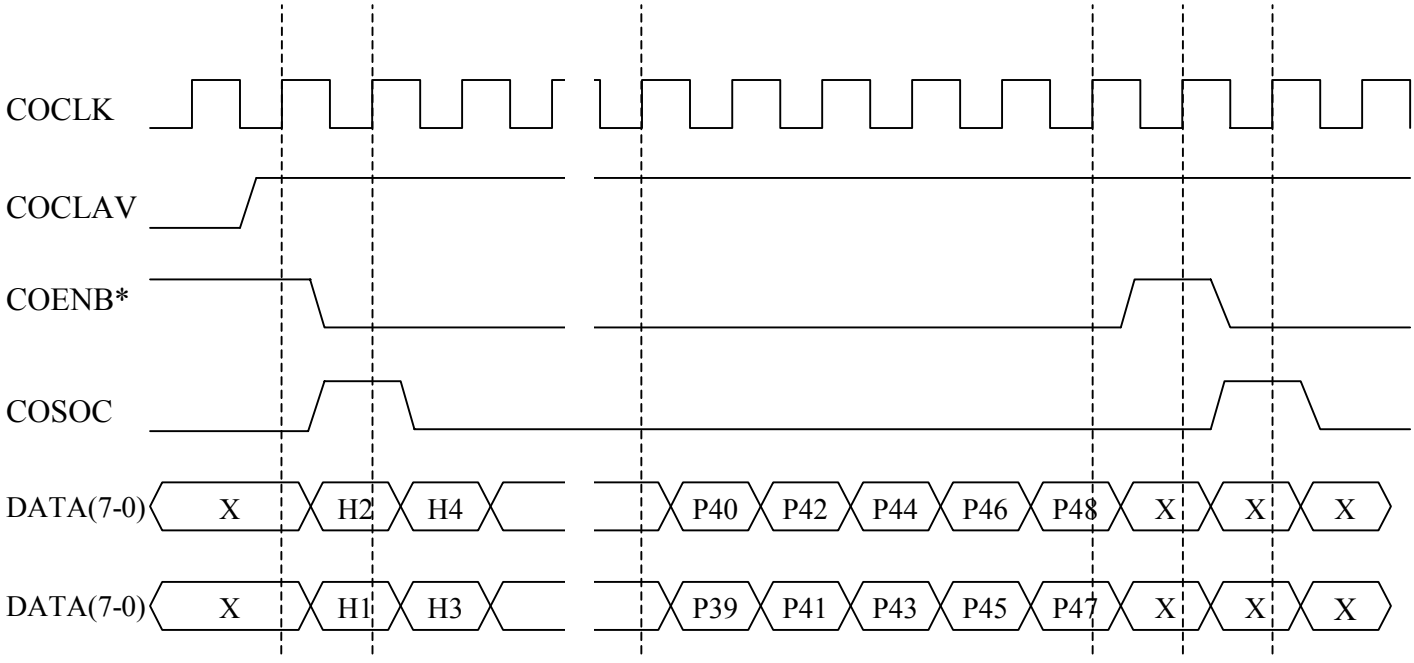


Figure 18. Utopia timing diagram, coclav allows two consecutive cell transmissions

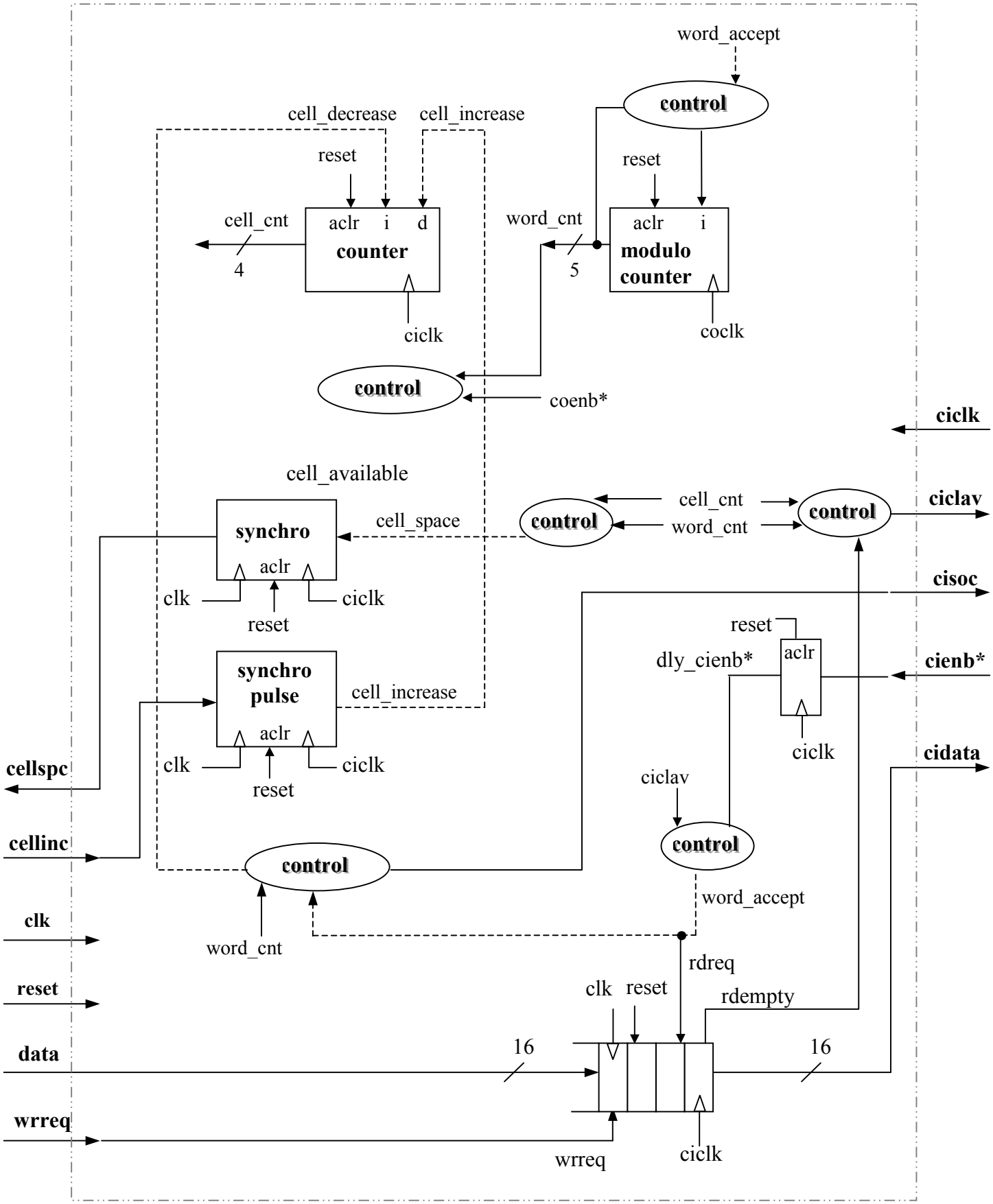


Figure 19. 16-bit Physical Layer Utopia Transmitter

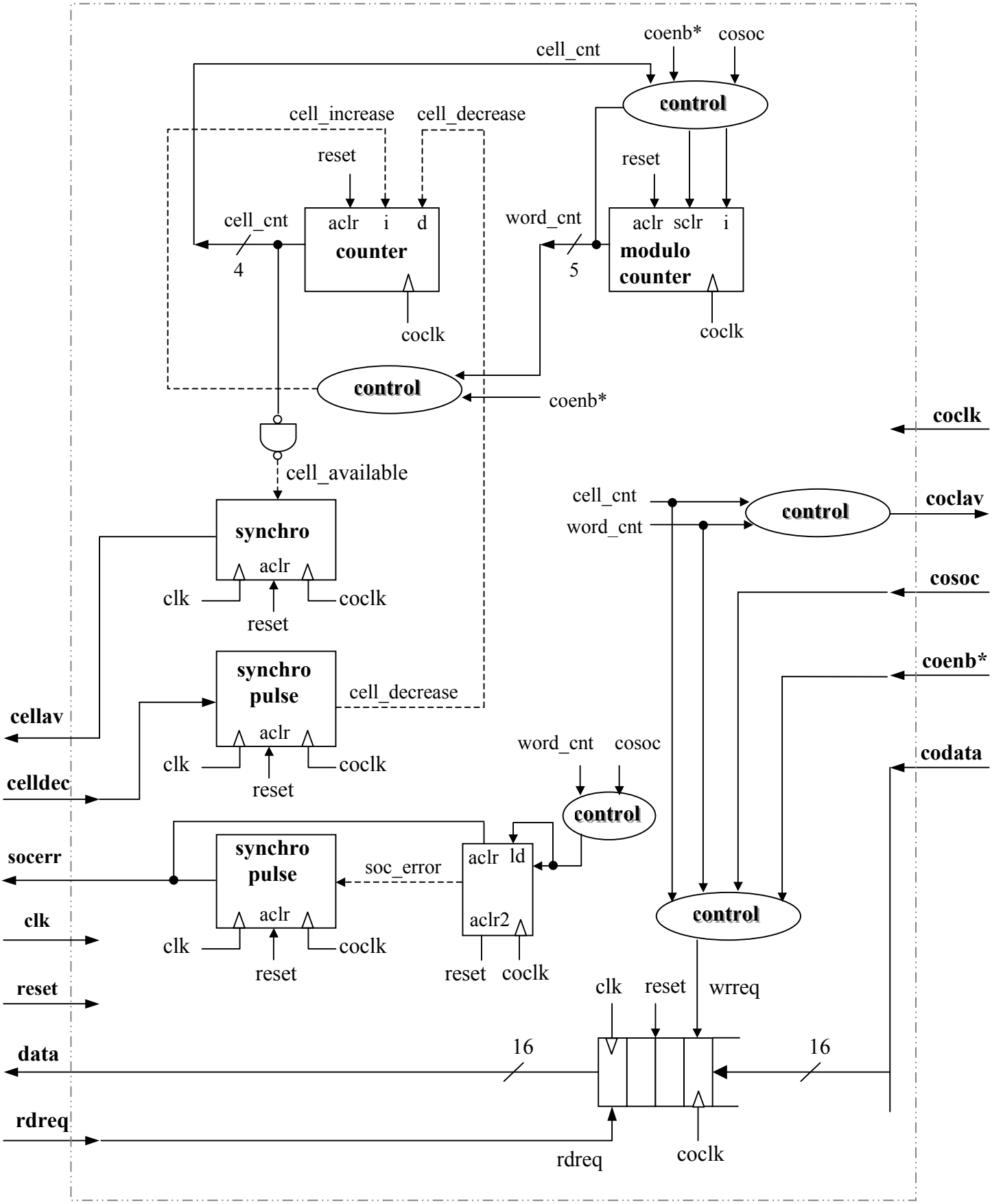


Figure 20. 16-bit Physical Layer Utopia Receiver

8. Implementation Results

The maximum transmission speed that our scheduler module achieves depends on the number (**N)** of flow groups that it is required to serve, and whether MCR support is included. The maximum attainable clock for the FPGA implementation of the scheduler module was 66 to 75 MHz depending on the number of flow groups. Nevertheless a 50 MHz clock is used because this is the speed that other modules of the FPGA can operate, thus this is the clock available in our system.

Without support for minimum rate guarantees the scheduler can reach the following speed per output link:

$$\frac{\text{bits per cell}}{\text{clock cycle time} * \text{N}}$$

N	Clock Frequency	**Maximum Transmission Speed**
16	50 MHz	1325 Mbps
32	50 MHz	662 Mbps
64	50 MHz	331 Mbps
128	50 MHz	165 Mbps

While the maximum speed of the scheduler itself is:

$$\frac{\text{bits per cell}}{\text{clock cycle}} * \frac{\sum_{i=1}^{N} \text{Max Speed of } i}{\text{N}}$$

Which in all cases is 21.200 Gbps. But what is really important is that the above table shows that our scheduler module is able to sustain 128 OC-3 links (155 Mbps each) or 32 OC-12 links (620 Mbps each). This means that the scheduler is not likely to become the bottleneck in our system. Instead the sender and even more the queue manager are potential bottlenecks. This problem could be solved by adding a second instance of the queue manager and also using a second SDRAM device. This however would definitely require a larger FPGA.

Adding support for MCR by using K different MCR-aggregated flow groups reduces our effective scheduling capacity. Following are the new maximum speeds per output link attainable by the scheduler.

N	Clock Frequency	K	**Maximum Transmission Speed**
16	50 MHz	16	662 Mbps
16	50 MHz	112	165 Mbps
32	50 MHz	96	165 Mbps
64	50 MHz	64	165 Mbps

From the above results we can see that when MCR support is added the scheduler module is able to sustain 32 OC-3 links (155 Mbps each) when having 96 MCR-aggregated flow groups or 16 OC-12 links (620 Mbps each) when having 16 MCR-aggregated flow groups.

We have said previously that above that the aggregate speed of the scheduler is 20 Gbps and noted that this is not really important. The reason is that the sender and queue manager modules can not match it. In particular the sender requires 15 clock cycles for every departing cell, this includes 6 cycles to issue an appropriate command to the queue manager, 6 cycles to interpret the response from the CellBus and 3 cycles to initiate the forwarder module. Every of the above jobs needs to be done once for every cell. So the maximum speed of the sender module allows 1.33 Gbps of outgoing ABR traffic. The queue manager restricts our capabilities even further, since it requires on average a little more than 40 cycles for every cell, this includes about 20 cycles to enqueue it and about 20 cycles to dequeue it. So the maximum ABR traffic is reduced to about 0.5 Gbps. Nevertheless there is potential for improvement for both modules the sender module could avoid the read-modify-write to save a few cycles and the queue manager could use a faster and more advanced memory controller.

9. Future work and enhancements

As the previous chapter showed the scheduler module is no bottleneck for our system, since the specifications of the project require less that 16 OC-3 links, so the heap-like implementation suggested in paragraph 4.1, may be desirable but not really necessary, moreover it hardly needs any change to support MCR-aggregated flow groups. However the sender module has several points that can be improved. One is support for MCR flow groups, an other is the removal of the throttling effect described in paragraph 5.4.

9.1 Removing the throttling effect

The FIFO used by each cubit pro device to accept cells from the bus is operated in split mode. This means that each type of traffic is allocated in a different sub-fifo. In particular the ABR traffic FIFO has a size of 32 cells with the congestion threshold fixed at the 28 cells. We can safely assume that failed transmissions over the CellBus will rarely occur due to CRC-failure, thus the only failed transmissions will be the ones that failed due to a full destination FIFO. We can exploit this fact to allow the sender to have five cell transmissions on-the-fly per cubit pro device, thus effectively removing the throttling effect. For this to work the output link aggregated flow groups must have an id that shows the corresponding cubit pro device, e.g. a part of each flow group id is a cubit pro id. In this way the output links corresponding to a particular cubit pro can share a pool of five allowable unacknowledged cell transmissions. Of course here we note that one wouldn't put four OC-3 links behind a single cubit pro device. Actually our system uses either one OC-3 link or four VDSL modems on every line card.

When the pool of five is emptied the pointer of the sixth cell is kept in the status memory of the sender, so that it can be retransmitted if necessary. Should we ever need to have seven unconfirmed cells for a cubit pro the corresponding service request is ignored. When we receive a congestion indication for a cell transmission the pool of the cubit pro does not grow by one, only non-congested confirmations are allowed to grow the pool. Of course once the FIFO of the output cubit pro is full there is no point in worrying about flow groups being throttled because of the CellBus delayed responses.

This scheme has the added advantage that once find out that a cubit pro is congested, the other flow groups of the particular cubit pro will be notified as soon as a service request for them reaches the sender, instead of attempting the transmission and waiting the result from the CellBus to indicate the congestion. Moreover once we discover that the ABR FIFO of a cubit pro is almost full we could decide to drop all service requests that want to send it a cell, unless the a carry a flag that indicates that the particular request was produced by the back off mechanism.

9.2 Adding MCR support to the Sender

In order to add MCR support to the above scheme, we need to device a way so that the MCR-aggregated flow groups can use the unacknowledged cell pool. We can achieve this if for every cell belonging to an MCR-aggregated flow group we are notified which is the next cubit pro that this MCR-aggregated flow group will transmit to. So when the queue manager serves an MCR-aggregated flow group it checks the next ABR flow that will be served to identify the cubit pro it uses and gives this information to the sender. We use the second part of the status memory to hold this information. The MCR-aggregated flow groups are not subject to back off, and in the case that the pool is completely empty the service request is dropped.

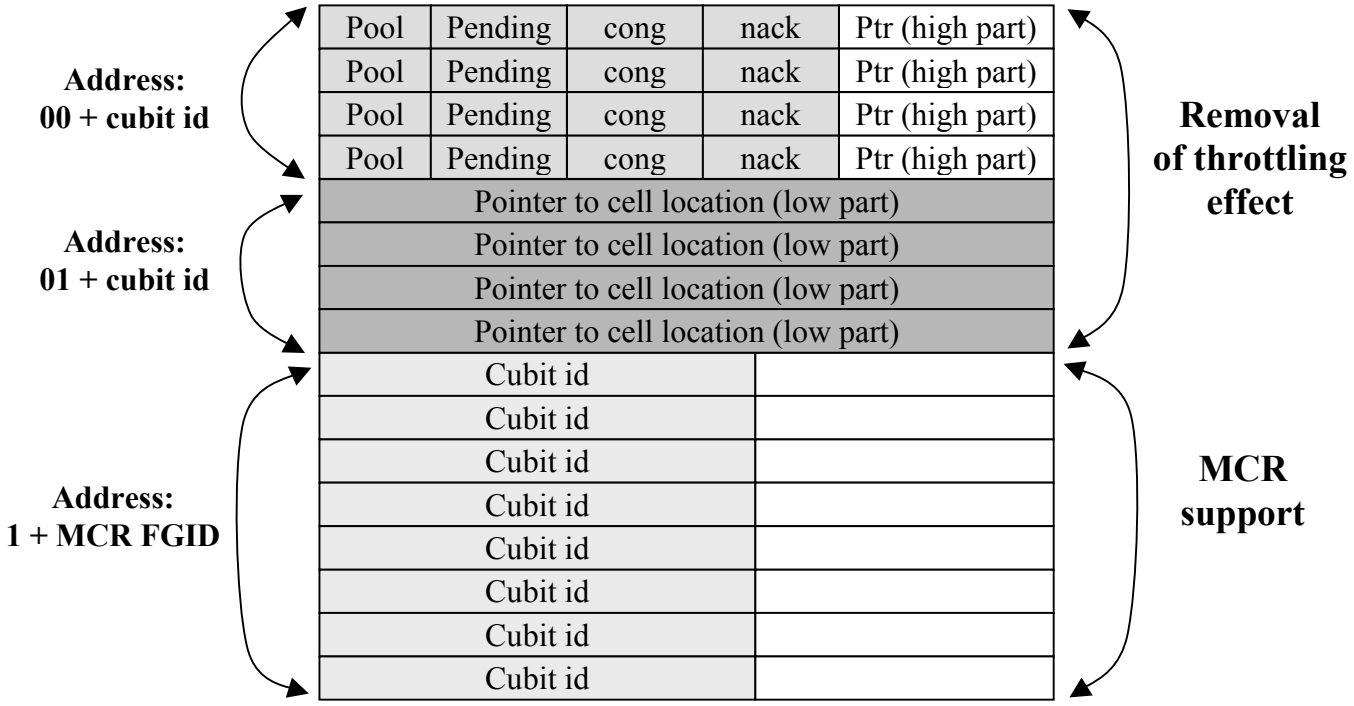


Figure 21. Sender's status memory with MCR support and no throttling effect

10. Discussion and Conclusions

We have presented a design of an ABR traffic scheduler design that is suitable for FPGA implementation. Although limited by the FPGA implementation the scheduler has met the performance requirements of the system in which it operates. It is able to saturate the output links as well as the shared bus. Also we are able add support MCR guarantees without seriously degrading the performance of the overall scheduler design.

Nevertheless the results regarding support of MCR guarantees are valid when two assumptions hold. The first assumption is that the number of different MCR speed rates is not high. This is rather safe assumption to make since the intermediate MCR can be rounded by the CAC to the predefined MCR speeds. The second assumption is that the number of ABR connections requiring a particular MCR speed is relatively small. If this assumption does not hold, then it is possible that the some MCR-aggregated flow groups will be oversubscribed above their scheduling capacity. This means that we may have to use more than one flow group for some MCR speeds. This would have a considerable but tolerable impact not on the scheduling performance but rather at the complexity of the queue manager. Here we must note again that the FPGA implementation introduces limits in both speed and available resources for the ABR server design. So, for higher speed links and better MCR support an ASIC implementation is preferable to satisfy such needs.

References

[1] D. Towsley, Performance Evaluation of Computer and Communication Systems, Springer, Berlin, 1993, pp. 560–586.
[2] C.M. Aras, J.F. Kurose, D.S. Reeves, H. Schulzrinne, Real-time communication in packet-switched networks, Proc. IEEE 82 (1994) 122–139.
[3] R. Brown, Calendar queues: a fast O (1) priority queue implementation for the simulation event set problem, Commun. ACM 31 (10) (1998) 1220–1227.
[4] J. Chao, A novel architecture for queue management in the ATM network, IEEE J. Selec. Areas Commun. 9 (7) (1991) 1110–1118.
[5] C.E. Leiserson, Systolic priority queues, CalTech. Conference on VLSI, 1979, pp. 200–214.
[6] D. Picker, R. Fellman, A VLSI priority packet queue with inheritance and overwrite, IEEE Trans. VLSI 3 (2) (1995) 245–252.
[7] H.J. Chao, N. Uzun, Sequencer chip for ATM traffic shaper and queue management, IEEE J. Solid-State Circuits 27 (11) (1992) 1634–1643.
[8] P. Lavoie, D. Haccoun, Y. Savaria, Systolic architecture for fast stack sequential decoders, IEEE Trans. Commun. (1994) 324–334.
[9] S.W. Moon, K.G. Shin, J. Rexford, Scalable hardware priority queue architectures for high-speed packet switches, IEEE Real-time Technology and Applications Symposium, 1997, pp. 203–212.
[10] M.R. Hashemi, A. Leon-Garcia, The single queue switch, INFOCOM'97, 1997, pp. 533–540.
[11] M.R. Hashemi, A. Leon-Garcia, A general cell sequencer/scheduler for ATM switches, INFOCOM'97, 1997, pp. 29–37.
[12] J.M. Tsai, C.Y. Lee, Novel architecture for ATM QoS management, IEEE Proc. Commun. 144 (6) (1997) 412–418.
-[13] H.J. Chao, H. Cheng, Y.R. Jenq, D. Jeong, Design of a generalized priority queue manager for ATM switches, IEEE J. Selec. Areas Commun. 15 (5) (1997) 867–879.
[14] A.K. Parekh, R.G. Gallager, A generalized processor sharing approach to flow control in integrated services networks, IEEE INFOCOM'93, 1993, pp. 521–530.
[15] S.J. Golestani, A self-clocked fair queuing scheme for broadband applications, IEEE INFOCOM'94, 1994, pp. 636–646.