ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Ανάπτυξη Ενσωματωμένων Δικτυακών Συστημάτων στο Περιβάλλον Εξομοίωσης QEMU

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Φοιτητής: Γυφτάκης Ιωάννης, Ε 07039

Επιβλέπων: Μηλιώνης Απόστολος, Λέκτορας

Περίληψη

Αυτό το κείμενο αναλύει τη δομή και τους μηχανισμούς του ανοιχτού λογισμικού εξομοιωτή QEMU ο οποίος μπορεί να χρησιμοποιηθεί στην ανάπτυξη Ενσωματωμένων Συστημάτων. Θα αναφερθούν τα πλεονεκτήματα και τα μειονεκτήματα σε σχέση με την ανάπτυξη σε πραγματικό υλικό(hardware). Επίσης θα περιγραφεί ο τρόπος με τον οποίο εξομοιώνει την αρχιτεκτονική x86 σε άλλες αρχιτεκτονικές οι οποίες αποτελούν τις βασικές επιλογές στο χώρο των Δικτυακών Ενσωματωμένων Συστημάτων. Εστιάζονται περισσότερο οι δικτυακές συσκευές που υποστηρίζει και θα αναλύονται δείγματα αυτών με παραδείγματα από τον πηγαίο κώδικα. Θα ακολουθηθεί η τακτική αυτή της έκθεσης αποσπασμάτων από τον πηγαίο κώδικα , της πιο πρόσφατης έκδοσης του λογισμικού , ώστε να γίνει κατανοητή η λειτουργία του αλλά και πιο εύκολη η ανάπτυξη στο μέλλον. Τέλος θα γίνει προσπάθεια να σχεδιαστεί και να αναπτυχθεί κώδικας για αρχιτεκτονικές οι οποίες δεν υποστηρίζονται ακόμη και οι οποίες είναι δημοφιλείς επιλογές στον τομέα, όπως επίσης θα γίνει προσπάθεια να παρεχθεί μια ανάλυση των επεξεργαστών επικοινωνιών PowerQUICC-II με στόχο την μελλοντική προσομοίωση δικτυακών συσκευών fast Ethernet για ελεγκτές FCC (Fast Communication Controllers).

Ευχαριστίες

Η παρούσα πτυχιακή εργασία εκπονήθηκε από τον φοιτητή Γυφτάκη Ιωάννη του Τμήματος Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς κατά το ακαδημαϊκό έτος 2010-2011 υπό την επίβλεψη του καθηγητή του τμήματος Απόστολου Μηλιώνη.

Οφείλω στον καθηγητή μου Α. Μηλιώνη τις θερμές μου ευχαριστίες για την καθοδήγηση και την υποστήριξή του καθ' όλη τη διάρκεια διεκπεραίωσης της παρούσας πτυχιακής.

Επίσης θερμές ευχαριστίες οφείλω στην κοινότητα του QEMU και ιδιαίτερα στα μέλη του IRC καναλιού του QEMU που ανταποκρίθηκαν στην έκκληση μου για βοήθεια ουκ ολίγες φορές και ξεδιάλυναν πολλές από τις απορίες μου.

Περιεχόμενα

Εισαγωγή

Αντικείμενο της Εργασίας

Ως στόχο η εργασία έχει να αναλύσει και να περιγράψει τους εσωτερικούς μηχανισμούς και τη δομή του QEMU, ενός σύγχρονου εξομοιωτή ανοιχτού λογισμικού. Η ανάλυση αυτή έχει ως γνώμωνα την ανάπτυξη στον τομέα των Ενσωματωμένων Συστημάτων και ιδιαίτερα δίνεται έμφαση στην ανάπτυξη των Δικτυο-στρεφών Ενσωματωμένων Συστημάτων. Επίσης θα αναδείξει τα προτερήματα της εξομοίωσης έναντι της ανάπτυξης σε πραγματικό υλικό όπου πολλές φορές ίσως να μην είναι καν διαθέσιμο στον μηχανικό ή τον προγραμματιστή. Η μεθοδολογία που θα ακολουθήσουμε είναι αυτή της αποσπασματικής επεξήγησης σημαντικών σημείων του πηγαίου κώδικα όπου η κατανόησή τούς μπορεί να δώσει μια πιο ξεκάθαρη εικόνα της γενικής λειτουργίας. Επίσης θα παραθέτονται σχεδιαγράμματα όπου κρίνονται απαραίτητα για τη διευκόλυνση και την οπτικοποίηση των λειτουργιών και των γενικών ιδεών. Τέλος ένα σημαντικό κομμάτι αυτής της εργασίας είναι να προτείνει κάποιες λύσεις για την υποστήριξη και υλοποίηση κάποιων δημοφιλών αρχιτεκτονικών για το σχεδιασμό δικτυοστρεφών Ενσωματωμένων Συστημάτων με κύρια να είναι η υποστήριξη CPM.

Στο **πρώτο κεφάλαιο** θα πούμε λίγα λόγια για την ιστορική αναδρομή του και για την άδειά με την οποία προσφέρεται στο κοινό όπως επίσης και σε ποια γλώσσα έχει αναπτυχθεί. Ύστερα θα δούμε τους ορισμούς της εξομοίωσης και της προσομοίωσης και σε ποια κατηγορία εντάσσεται ο QEMU. Θα αναδείξουμε τα πλεονεκτήματα μιας προσέγγισης από πλευράς λογισμικού στην ανάπτυξη Ενσωματωμένων Συστημάτων και θα δούμε παραδείγματα όπου ίσως αυτή να είναι και η μοναδική λύση που έχει ο ενδιαφερόμενος μηχανικός ή προγραμματιστής. Προς το τέλος του κεφαλαίου θα αφιερώσουμε ένα μέρος για την σύγκριση του QEMU με άλλους emulators και την ανάδειξη των πλεονεκτημάτων και μειονεκτημάτων του έναντι αυτών.

Στο **δεύτερο κεφάλαιο** θα ασχοληθούμε με τις τεχνικές που χρησιμοποιεί ο QEMU όπως είναι η JIT(Just In Time) μεταγλώττιση για να μεταφράσει τις εντολές άλλων αρχιτεκτονικών σε αυτή της x86 όπως είναι ένα desktop περιβάλλον. Θα μελετήσουμε μέρη από τον πηγαίο κώδικα όπου υλοποιεί την παραπάνω διαδικασία.

Στο **τρίτο κεφάλαιο** θα εστιάσουμε στη μελέτη της υποστήριξης των δικτυακών συσκευών που προσφέρονται από τον QEMU. Για αυτό το σκοπό θα χρησιμοποιήσουμε και πάλι δείγματα από τον πηγαίο κώδικα των επιλεγμένων δικτυακών υποστηριζόμενων συσκευών . Συγκεκριμένα θα επικεντρωθούμε περισσότερο σε δημοφιλείς συσκευές όπου αναλύοντας αυτές μπορούμε να χρησιμοποιηθούν ως πρότυπο για τη δημιουργία νέων.

Στο **τέταρτο κεφάλαιο** θα προτείνουμε κάποιες λύσεις για την υποστήριξη κάποιων συγκεκριμένων δημοφιλών αρχιτεκτονικών που λείπουν από την “εργαλειοθήκη” του QEMU και οι οποίες είναι πολύ βασικές για την ανάπτυξη σύγχρονων δικτυακών Ενσωματωμένων Συστημάτων.

Στο **πέμπτο κεφάλαιο** αναφερόμαστε στα εξαγόμενα συμπεράσματα τα οποία αντιπροσωπεύουν

συνολικά τα αποτελέσματα της συνθετικής και αναλυτικής μεθόδου από όλα τα κεφάλαια της εργασίας.

Ύστερα παραθέτουμε τη **βιβλιογραφία** που χρησιμοποιήθηκε. Όπου αυτή αποτελείται από βιβλία, επιστημονικά άρθρα, άρθρα περιοδικού τύπου, ηλεκτρονικές πηγές κλπ.

Κεφάλαιο 1 – Ο QEMU στην υπηρεσία των Ενσωματωμένων Συστημάτων

1.1 Εξομοίωση και Προσομοίωση

Η εξομοίωση εισήχθη , στον κόσμο της Πληροφορικής , ως έννοια το 1957 από την γνωστή εταιρία IBM όταν πρόσθεσε στον υπολογιστή IBM 709 τη δυνατότητα να εκτελεί προγράμματα τα οποία είχαν αναπτυχθεί για το παλιότερο μοντέλο IBM 704. Οι καταχωρητές και οι assembly εντολές του IBM 704 είχαν εξομοιωθεί στο υλικό του IBM 709. Περίπλοκες εντολές και ρουτίνες εισόδου-εξόδου του IBM 704 είχαν εξομοιωθεί σε λογισμικό του IBM 709.

Έχει επικρατήσει η τάση να χρησιμοποιούμε τη λέξη εξομοίωση στο ευρύ περιβάλλον του λογισμικού. Ωστόσο, πρίν το 1980 , η εξομοίωση αναφερόταν μόνο στην εξίσωση της λειτουργικότητας που προσέφερε ένα υλικό ή μικροκώδικας με αυτή ενός άλλου υλικού ή μικροκώδικα, ενώ προσωμοίωση ονόμαζαν την εξομοίωση βασισμένη εξ' ολοκλήρου σε λογισμικό.

Για παράδειγμα, ένας υπολογιστής ειδικά σχεδιασμένος να εκτελεί προγράμματα διαφορετικής αρχιτεκτονικής είναι ένας εξομοιωτής. Σε αντίθεση, ένας προσομοιωτής θα μπορούσε να είναι ένα πρόγραμμα που εκτελείται σε έναν υπολογιστή έτσι ώστε παλιά παιχνίδια Atari να μπορούν να τρέχουν σε αυτόν.

1.2 Ο QEMU και η χρήση του

Ο QEMU είναι ένας ανοιχτού λογισμικού(open source), γρήγορος εξομοιωτής επεξεργαστών χρησιμοποιώντας έναν δυναμικό μεταφραστή ο οποίος έχει σχεδιαστεί να είναι όσο το δυνατόν περισσότερο μεταφέρσιμος γίνεται.

Ο QEMU έχει δύο κύριες λειτουργίες τις οποίες περιγράφουμε , συντόμως , παρακάτω:

- **Πλήρης Εξομοίωση του Συστήματος**(Full System Emulation)

  Σε αυτή την λειτουργία(full platform virtualization) ο QEMU εξομοιώνει ένα πλήρες σύστημα(συνήθως έναν Η/Υ), συμπεριλαμβανομένου του επεξεργαστή και διαφόρων περιφερειακών. Μπορεί έτσι να χρησιμοποιηθεί για να εκτελέσει και λειτουργήσει διαφορετικά Λειτουργικά Συστήματα χωρίς να επανεκκινήσει το σύστημα οικοδεσπότη(host machine) ή να απασφαλματώσει τον κώδικα από την αρχή.

- **Εξομοίωση σε Επίπεδο Χρήστη**(User Mode Emulation)

  Όταν ο QEMU χρησιμοποιηθεί με αυτή την λειτουργία μπορεί να εκκινήσει διαδικασίες οι οποίες μεταγλωττίστηκαν για μία CPU_1 σε μία CPU_2, ωστόσο τα λειτουργικά συστήματα

είναι απαραίτητο να ταιριάζουν. Αυτή η λειτουργία συνήθως χρησιμοποιείται για να διευκολύνει την κάθετη μεταγλώττιση(cross-compilation) και την κάθετη αποσφαλμάτωση(cross-debugging).

Σχήμα 1.1: Γενική αναπαράσταση της τοποθεσίας του QEMU και της ανάδρασης του με το Λειτουργικό Σύστημα.

1.3 Σημαντικά Χαρακτηριστικά του QEMU

Θέλοντας να εμβαθύνουμε στην ανάλυση των χαρακτηριστικών της **Πλήρους Εξομοιώσεως του Συστήματος** αναφέρουμε ότι ο QEMU προσομοιώνει εξ' ολοκλήρου την MMU για μέγιστη μεταφερσιμότητα (portability). Δηλαδή η MMU είναι λογισμικό και όχι υλικό.

Επίσης ο QEMU μπορεί προαιρετικά να χρησιμοποιήσει την βοήθεια ενός ενδο-Kernel επιταχυντή όπως είναι ο kvm. Οι επιταχυντές αυτοί δίνουν το πλεονέκτημα εκτελώντας ένα μέρος του φιλοξενούμενου κώδικα(guest code) σαν να εκτελούσαν κανονικά τη διαδικασία στον οικοδεσπότη ενώ το υπόλοιπο μέρος του φιλοξενούμενου συστήματος συνεχίζει να προσομοιώνεται. Δίνοντας έτσι περισσότερη ταχύτητα στη διαδικασία της εξομοίωσης γενικά. Όπως εξηγήσαμε και θα βλέπουμε συχνά στην πορεία της παρούσας εργασίας ο QEMU εμπεριέχει την έννοια της προσομοίωσης και της

εξομοίωσης παράλληλα κατά τη διάρκεια της λειτουργίας του και πολλές φορές γίνονται και ταυτόσημες.

Επιπροσθέτως, ο QEMU καταφέρνει να εξομοιώσει πολλές συσκευές όπως USB, σκληρούς δίσκους κλπ. Επίσης μπορεί να επικοινωνήσει με εξωτερικές περιφερειακές συσκευές μέσω της διασύνδεσης του οικοδεσπότη(πχ modem, webcam κλπ).

Όσο αναφορά την **Εξομοίωση σε Επίπεδο Χρήστη** , η οποία μας ενδιαφέρει περισσότερο στην ανάπτυξη Ενσωματωμένων Συστημάτων , ο QEMU μας δίνει τη δυνατότητα μετατροπής Κλήσεων Συστήματος Linux συμπεριλαμβανομένων των περισσοτέρων **ioctls**.

Επίσης μας δίνει τη δυνατότητα για ακριβή αντιστοίχηση στη μνήμη(remapping) και μετέπειτα διαχείριση των σημάτων του οικοδεσπότη συστήματος σε αυτά του φιλοξενούμενου συστήματος.

Έχοντας αναφερθεί στα παραπάνω καλό θα ήταν να δούμε και κάποια χαρακτηριστικά που είναι στοχευμένα για συγκεκριμένες αρχιτεκτονικές.

Αρχιτεκτονική x86

Ο QEMU για την x86 αρχιτεκτονική

- υποστηρίζει 16-bit και 32-bit διευθυνσιοδότηση(addressing) με τμηματοποίηση (segmentation).
- υποστηρίζει μέγεθος σελίδας(page size) για τον οικοδεσπότη μεγαλύτερο από 4KB σε λειτουργία **Εξομοίωσης σε Επίπεδο Χρήστη**.
- μπορεί να εξομοιώσει την ίδια την x86 πάνω από x86.

Αρχιτεκτονική ARM

Ο QEMU για την ARM αρχιτεκτονική

- υποστηρίζει πλήρως την λειτουργία **Πλήρης Εξομοίωση Συστήματος**
- μπορεί να τρέξει τα περισσότερα ARM Linux binaries

Αρχιτεκτονική PowerPC

Ο QEMU για την PowerPC αρχιτεκτονική

- υποστηρίζει πλήρως την λειτουργία Πλήρης Εξομοίωση Συστήματος για το σύστημα των 32 bits συμπεριλαμβανομένων των εντολών όπως FPU και MMU.
- μπορεί να τρέξει τα περισσότερα PowerPC Linux binaries.

Κάποιες άλλες από τις υποστηριζόμενες Αρχιτεκτονικές είναι η Sparc32, MIPS, Alpha, CRIS, M68k και SH4.

1.4 Πλεονεκτήματα και Μειονεκτήματα του QEMU στην ανάπτυξη Ενσωματωμένων Συστημάτων

Πλεονεκτήματα:

Έχοντας αναφέρει τα βασικά χαρακτηριστικά του QEMU είμαστε σε θέση να κατανοήσουμε γιατί γίνεται δημοφιλής στη διαδικασία της ανάπτυξης Ενσωματωμένων Συστημάτων.

- ✔ **Μείωση Testing Χρόνου**(hardware testing time):

Τις περισσότερες φορές οι προγραμματιστές Ενσωματωμένων Συστημάτων χρειάζεται να περιμένουν μέχρι την ολοκλήρωση του επικείμενου υλικού , από τους μηχανικούς , για το οποίο θα αναπτύξουν το λογισμικό. Όμως αυτή η διαδικασία συνήθως είναι επαναληπτική και επίπονη και σχεδόν πάντα διαρκεί πολύ χρόνο, χρόνος τον οποίο θα μπορούσαν να είχαν εκμεταλλευτεί οι προγραμματιστές και παράλληλα να ανέπτυσσαν το λογισμικό.

Ο QEMU δίνει τη λύση σε αυτό το πρόβλημα εξομοιώνοντας το υλικό οπότε οι προγραμματιστές μπορούν να ξεκινήσουν την ανάπτυξη του λογισμικού καθώς οι μηχανικοί εργάζονται στην ολοκλήρωση του υλικού. Όταν το υλικό ολοκληρωθεί τότε τα τελικά tests μπορούν γίνουν εκεί.

- ✔ **Ασφαλές Περιβάλλον Ελέγχου**(safe sandbox for testing):

Στο υλικό ή λογισμικό ο έλεγχος είναι μία επίπονη διαδικασία και η πιθανότητα να προκληθεί

σοβαρό πρόβλημα στον εξοπλισμό είναι μεγάλη. Η διαδικασία της εξομοίωσης μας λύνει τα χέρια δημιουργώντας όχι μόνο ένα αλλά πολλά ασφαλή εικονικά περιβάλλοντα για έλεγχο. Αν ο έλεγχος αποτύχει ή κάτι προκαλέσει ένα πρόβλημα τότε απλά επανεκκινείς τον QEMU και όλα επανέρχονται στην αρχική κατάσταση. Τρανταχτό παράδειγμα είναι το testing κατά τη διάρκεια ανάπτυξης Οδηγών Συσκευών(Device Drivers) ή τροποποίηση στον Kernel ενός Λειτουργικού Συστήματος όπου ένα λάθος και μόνο κατά το testing σταματά την καθολική λειτουργία του συστήματος.

- ✔ **Διαθεσιμότητα Υλικού**(hardware availability):

Πολλές φορές οι μηχανικοί και προγραμματιστές Ενσωματωμένων Συστημάτων δεν έχουν άλλη επιλογή εκτός από την εξομοίωση. Το υλικό δεν υπάρχει για διάφορους λόγους αλλά αυτό δε θα έπρεπε να σταματήσει την ανάπτυξη των παραπάνω επιπέδων της ανάπτυξης Ενσωματωμένων Συστημάτων.

Μειονεκτήματα:

- ✗ **Μειωμένη Απόκριση:**

Αν και η εξομοίωση και η προσομοίωση είναι ώριμα εργαλεία πια και σημαντικά για την ανάπτυξη των Ενσωματωμένων Συστημάτων, η εκτέλεση στο πραγματικό υλικό είναι ακόμα η καλύτερη απάντηση που μπορείς να πάρεις για τη συμπεριφορά του Συστήματος.

1.5 Σύγκριση με άλλους εξομοιωτές

Ο QEMU δεν είναι ο μόνος εξομοιωτής που υπάρχει για να μας βοηθήσει στην ανάπτυξη Ενσωματωμένων Συστημάτων. Αυτό είναι επιθυμητό διότι κάθε ένας μαθαίνει από τα πλεονεκτήματα και μειονεκτήματα της υλοποίησης και του σχεδιασμού του άλλου και έτσι με τον καιρό έχουμε πιο ώριμους και στιβαρούς εξομοιωτές.

- **Bochs**

Όπως ο bochs, έτσι και ο QEMU εξομοιώνει την αρχιτεκτονική x86. Όμως ο QEMU είναι πολύ γρηγορότερος χάριν στη δυναμική μετάφραση που χρησιμοποιεί(θα μιλήσουμε στο Κεφάλαιο 2 αναλυτικότερα). Επίσης ο bochs είναι στενά συνδεδεμένος με την εξομοίωσης της x86 ενώ ο QEMU μπορεί να εξομοιώσει πολύ περισσότερους επεξεργαστές.

- **Valgrind**

Όπως ο Valgrind, έτσι και ο QEMU χρησιμοποιεί δυναμική μετάφραση και υποστηρίζει την Εξομοίωση σε Επίπεδο Χρήστη όπως αναφερθήκαμε πιο πριν στο ίδιο Κεφάλαιο. Ο Valgrind είναι , κυρίως , ένας απασφαλματωτής μνήμης ενώ ο QEMU δεν παρέχει υποστήριξη για τέτοιου είδους λειτουργικότητα(ο QEMU θα μπορούσε να εντοπίσει περιπτώσεις όπου έχουμε πρόσβαση στη μνήμη εκτός ορίων όπως ο Valgrind , αλλά δε μπορεί να ιχνηλατήσει μη αρχικοποιημένα δεδομένα όπως δύναται ο Valgrind). Επίσης ο Valgrind παρέχει και αυτός δυναμική μετάφραση με καλύτερο παραγώμενο κώδικα από τον QEMU αλλά είναι πολύ στενά συνδεδεμένος με την αρχιτεκτονική x86 όπου αυτό περιορίζει την χρήση του σε σχέση με τον QEMU.

- **EM86**

Ο EM86 είναι το κοντινότερο σε υλοποίηση αλλά και σχεδιασμό σε σχέση με την Εξομοίωση σε Επίπεδο Χρήστη προς τον QEMU(ο QEMU χρησιμοποιεί ακόμα κάποια μέρη από την κώδικα του EM86, ιδιαίτερα τον ELF file loader). Ο EM86 περιορίστηκε με ένα πρώιμο στάδιο οικοδεσπότη και χρησιμοποιούσε κλειστό λογισμικό και έναν αργό μεταφραστή.

- **TWIN**

Ο TWIN είναι ένας εξομοιωτής με σκοπό να τρέχει στο λειτουργικό σύστημα Windows. Έχει πολλά κοινά χαρακτηριστικά με τον Wine αλλά είναι λιγότερο ακριβής από αυτόν. Όμως περιέχει μία λειτουργία προσανατολισμένη στην x86 αρχιτεκτονική για να εκτελεί x86 Windows εκτελέσιμα αρχεία.

- **User mode Linux**

Πριν τον QEMU η μόνη επιλογή για να τρέξεις τον Linux Kernel ως διεργασία χωρίς να χρειάζεται να προσθέσεις patches ήταν η χρήση του User mode Linux εξομοιωτή. Το μειονέκτημα του User mode Linux είναι ότι για να τρέξει σωστά χρειάζεται μεγάλα Kernel patches ενώ ο QEMU δε χρειάζεται καθόλου patches. Το τίμημα που πληρώνει ο QEMU είναι ότι αργεί παραπάνω η διαδικασία.

- **Εμπορικοί PC Virtualizers**(VMware, VirtualPC, TowOStwo)

Οι εμπορικοί PC Virtualizers είναι γρηγορότεροι από τον QEMU, όμως όλοι τους χρειάζονται ειδικό, κλειστό λογισμικό και πιθανώς μη ασφαλείς οδηγούς συσκευών. Επιπλέον, είναι ανίκανοι να παρέχουν ακριβή προσομοιούμενους κύκλους όπως μπορεί ένας εξομοιωτής.

- **VirtualBox, Xen, KVM**

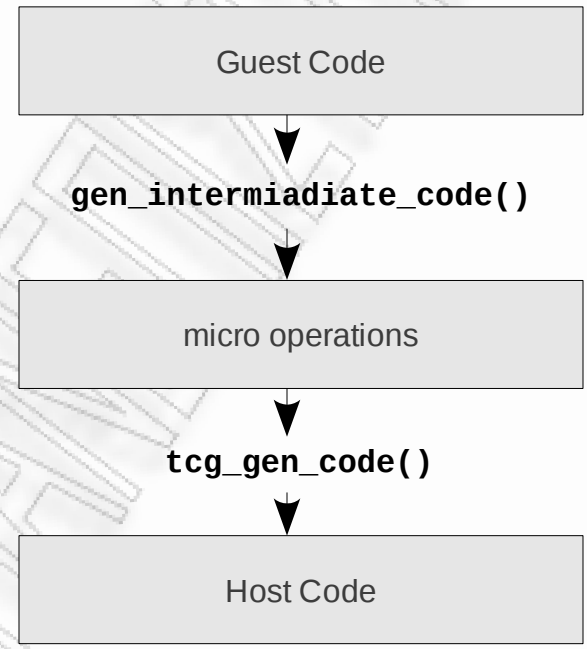
Οι παραπάνω είναι βασισμένοι στον QEMU. Ειδικότερα ο QEMU-SystemC χρησιμοποιεί τον QEMU για να εξομοιώσει ένα σύστημα στο οποίο οι οδηγοί συσκευών υλικού έχουν αναπτυχθεί σε SystemC.

Κεφάλαιο 2 – Εσωτερικοί Μηχανισμοί του QEMU

2.1 Δυναμική Μετάφραση Ομάδων Εντολών(Dynamic Translation)

Η ιδέα πίσω από την Δυναμική Μετάφραση είναι να χωριστούν οι Εντολές συγκεκριμένων αρχιτεκτονικών σε λιγότερες ομάδες εντολών εμφωλευμένες στις λεγόμενες **micro operations**. Κάθε micro operation είναι ένα μικρό κομμάτι κώδικα γραμμένο στη γλώσσα προγραμματισμού C. Επίσης κάθε mirco operation μεταγλωττίζεται από τον GCC και έτσι παράγεται ο αντικείμενος κώδικας. Οι micro operations είναι επιλεγμένες έτσι ώστε να είναι πολύ λιγότερες από ότι οι πιθανοί συνδυασμοί των Εντολών Assemby της εκάστωτε αρχιτεκτονικής. Η μετάφραση των Εντολών Assembly σε micro operations γίνεται από τον QEMU.

Τη διαδικασία της μετάφρασης από Εντολές Assembly σε micro operations την αναλαμβάνει ο **TCG**(μέρος του QEMU). Ο TCG παίρνει ως είσοδο τον αντικείμενο κώδικα που παράχθηκε και περιέχει τις micro operations και με τη σειρά του παράγει έναν **Παραγωγό Δυναμικού Κώδικα**(dynamic code generator). Αυτός ο Παραγωγός Δυναμικού Κώδικα καλείται κατά τη διάρκεια της εκτέλεσης για να παράγει μία ολοκληρωμένη συνάρτηση οικοδεσπότη η οποία διασυνδέει πολλές από τις micro operations που εισήχθησαν με την μορφή του αντικείμενου κώδικα όπως αναφέραμε στην αρχή της παραγράφου.



Σχήμα 2.1: Περιγραφή βασικών βημάτων Δυναμικής Μετάφρασης του QEMU

Θα εξηγήσουμε παρακάτω τι δέχονται και τι εξάγουν οι ενδιάμεσες συναρτήσεις του QEMU. Για τώρα αρκεί να καταλάβουμε τη γενική εικόνα των παραδοτέων από βήμα σε βήμα.

Ακολουθεί ένα παράδειγμα για να δούμε στην πράξη τις **micro operations** και τον **TCG**.

Παράδειγμα 2.1 – Δυναμική Μετάφραση στην Πράξη

Έστω ότι θέλουμε να μεταφράσουμε μια Assembly PowerPC εντολή σε x86:

```
addi      r1, r2, -16        #επεξήγηση σε ψευδογλώσσα, r1 = r1 – 16
```

Από την παραπάνω εντολή PowerPC θα παραχθούν από τον PowerPC μεταφραστή οι παρακάτω εντολές οι οποίες θα μετατραπούν σε **micro operations** από τον QEMU:

```
movl_T0_r1                   #T0 = r1
addl_T0_im  -16              #T0 = T0 – 16
mov_r1_T0                    #r1 = T0
```

Ο QEMU αντί να παράγει κάθε πιθανή μετακίνηση των 32 καταχωρητών PowerPC, απλά μετακινείται από και προς μερικούς προσωρινούς καταχωρητές. Οι καταχωρητές `T0, T1, T2` συνήθως αποθηκεύονται σε αντίστοιχους καταχωρητές αρχιτεκτονικής του οικοδεσπότη χρησιμοποιώντας τον προσδιορισμό μεταβλητής `static register` που προσφέρει η γλώσσα C.

Η εντολή `movl_T0_r1` υλοποιείται σε micro operation από τον QEMU σε γλώσσα C ως εξής:

```
void op_movl_T0_r1(void)
{
    T0 = env->regs[1];       #env είναι μία structure που-->
}                            #-->δείχνει την κατάσταση της CPU.
                             #regs είναι πίνακας και μέλος της env
```

Η `env` είναι μία δομή(structure της γλώσσας C) η οποία περιέχει την κατάσταση της φιλοξενούμενης αρχιτεκτονικής δηλαδή στην περίπτωσή μας της PowerPC.

Οπότε οι 32 καταχωρητές (registers) της PowerPC είναι αποθηκευμένοι στον πίνακα `env->regs[32]`.

Μπορεί κανείς να επιβεβαιώσει ότι η `env` δείχνει την κατάσταση της φιλοξενούμενης CPU αν ψάξει στον πηγαίο κώδικα του QEMU.

Όπως κάναμε εμείς παρακάτω από δύο κομμάτια πηγαίου κώδικα από διαφορετικά αρχεία όμως το καθένα:

```
//path: qemu-0.14.0/target-ppc/translate.c:9127

void gen_intermediate_code(CPUState *env,struct TranslationBlock *tb)
{
    gen_intermediate_code_internal(env, tb, 0);
}
```

```
//path: qemu-0.14.0/target-ppc/cpu.h:72

#define CPUState struct CPUPPCState
```

όπως φαίνεται από τα χρωματισμένα με κόκκινο σημεία του κώδικα αποδεικνύεται του λόγου το αληθές. Δώσαμε έμφαση στην `env` γιατί χρησιμεύει στην κατανόηση και άλλων δομικών στοιχείων στον QEMU όπως θα δούμε παρακάτω στο ίδιο κεφάλαιο.

Συνεχίζουμε το **Παράδειγμα 2.1** με την δεύτερη εντολή `addl_T0_im -16` η οποία είναι διαφορετική από την πρώτη εντολή με την έννοια ότι περιέχει και παράμετρο. Η παράμετρος `-16` καθορίζεται κατά τη διάρκεια της εκτέλεσης.

Ο κώδικας για αυτή την micro operation είναι:

```
extern int __op_param1; #η παράμετρος ,σε αυτή την περίπτωση η -16
void op_addl_T0_im(void)
{
    T0 = T0 + ((long))(&__op_param1));
}
```

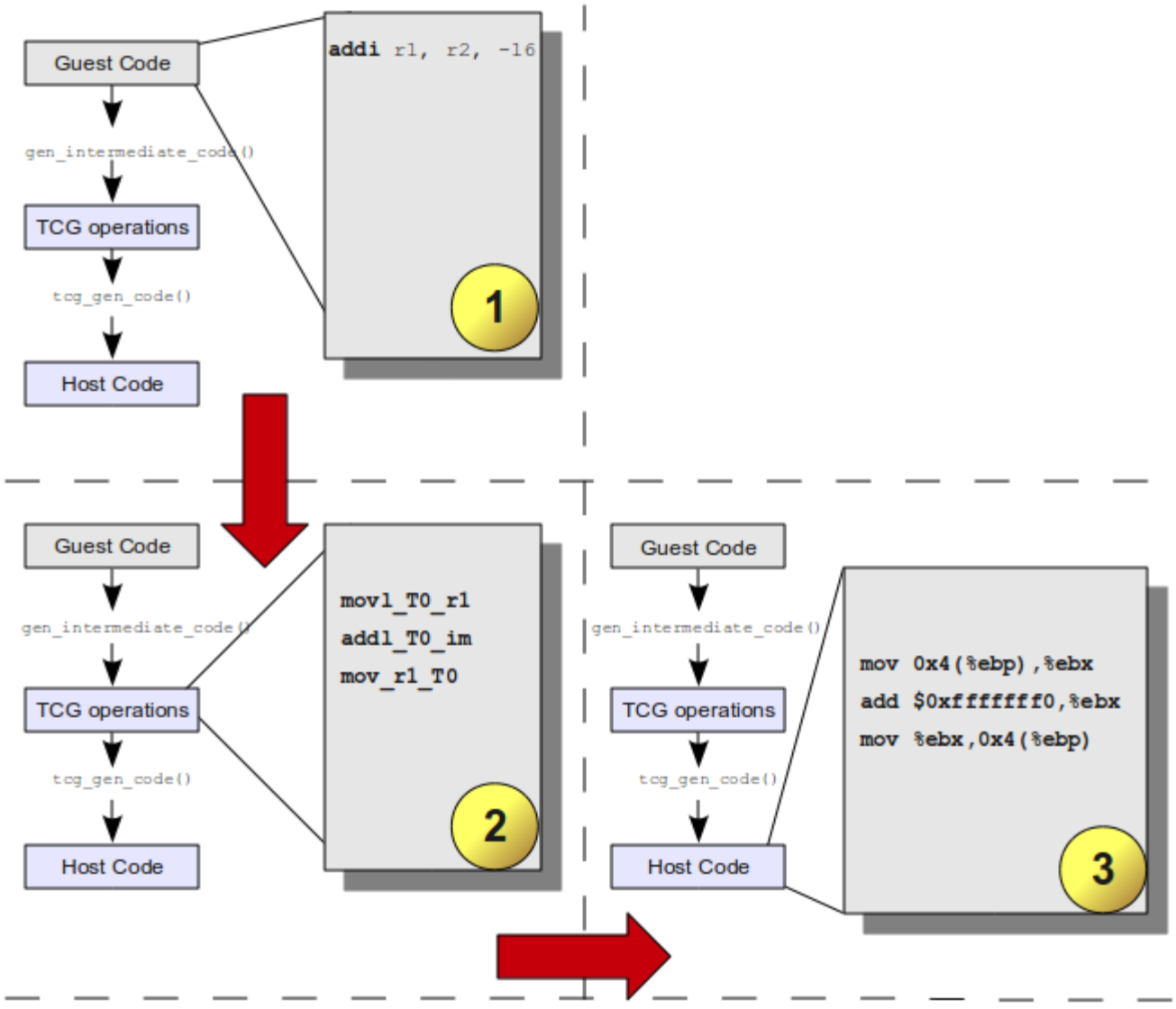
Ο Παραγωγός Δυναμικού Κώδικα δέχεται μία micro operation η οποία δεικτοδοτείται από τον `gen_opc_ptr` και εξάγει τον κατάλληλο κώδικα αρχιτεκτονικής οικοδεσπότη(x86) στην τοποθεσία που ξεκινά εκεί που δείχνει ο δείκτης `gen_code_ptr` . Ενώ οι micro operations δεικτοδοτούνται από τον δείκτη `gen_opparam_ptr`.

Η τελευταία εντολή `mov_r1_T0` ακολουθεί ακριβώς την ίδια διαδικασία με την πρώτη οπότε δημιουργείται μία αντίστοιχη συνάρτηση σε γλώσσα C.

Οπότε ως τελικό αποτέλεσμα θα έχουμε τον εξής κώδικα σε αρχιτεκτονική οικοδεσπότη:

```
#T0 αντιστοιχεί στον ebx για την x86
mov  0x4(%ebp), %ebx    #ebp αντίστοιχο του env->regs[1] δείχνει->
add  $0xfffffff0, %ebx  #-->την κατάσταση της CPU
mov  %ebx, 0x4(%ebp)
```

Πιο σχηματικά τα βήματα του **Παραδείγματος 2.1** θα μπορούσαν να αναπαρασταθούν ως εξής:



Σχήμα 2.2: Υλοποίηση του **Παραδείγματος 2.1** και σχηματική ανάλυση των βημάτων.

2.2 Ο μηχανισμός των Τμημάτων Εντολών Μετάφρασης

Η εικονική CPU της φιλοξενούμενης αρχιτεκτονικής μπορεί να βρεθεί σε πολλές καταστάσεις στις οποίες μπορεί να εναλλάσσεται κατά τη διάρκεια της λειτουργίας της. Η κατάσταση στην οποία βρίσκεται καθορίζει και τον τρόπο με τον οποίο χρησιμοποιεί και αξιολογεί τις εκάστοτε εντολές. Οπότε ο QEMU αντιμετωπίζει αυτό το πρόβλημα με το να καταγράφει την κάθε κατάσταση στην οποία βρέθηκε η φιλοξενούμενη CPU στα λεγόμενα **Τμήματα Εντολών Μετάφρασης**(Translation Blocks ή TB εν συντομία). Εάν η κατάσταση της CPU αλλάξει τότε ένα νέο Translation Block δημιουργείται και το προηγούμενο θα παραμείνει αδρανές έως ότου η κατάσταση να ταυτιστεί με την κατάσταση που είχε καταγραφεί στο προηγούμενο TB.

Η επιλογή του πρώτου τμήματος του πηγαίου κώδικα της συνάρτησης `gen_intermediate_code()` στο **Παράδειγμα 2.1** δεν ήταν τυχαία επιλογή. Αν παρατηρήσει κανείς προσεκτικά είναι η συνάρτηση που δέχεται τις εντολές της φιλοξενούμενης αρχιτεκτονικής και τις μετατρέπει σε micro operations στο **Σχήμα 2.1**. Όπως φαίνεται από τον κώδικα δέχεται ως παραμέτρους την `env` και το αντίστοιχο `Translation Block` όπως ακριβώς αναφέραμε στην αρχή της προηγούμενης παραγράφου. Έτσι μπορούμε να κατανοήσουμε καλύτερα την διασύνδεση μεταξύ Translation Block και της κατάστασης της CPU.

Διασύνδεση Τμημάτων Εντολών Μετάφρασης(block chaining)

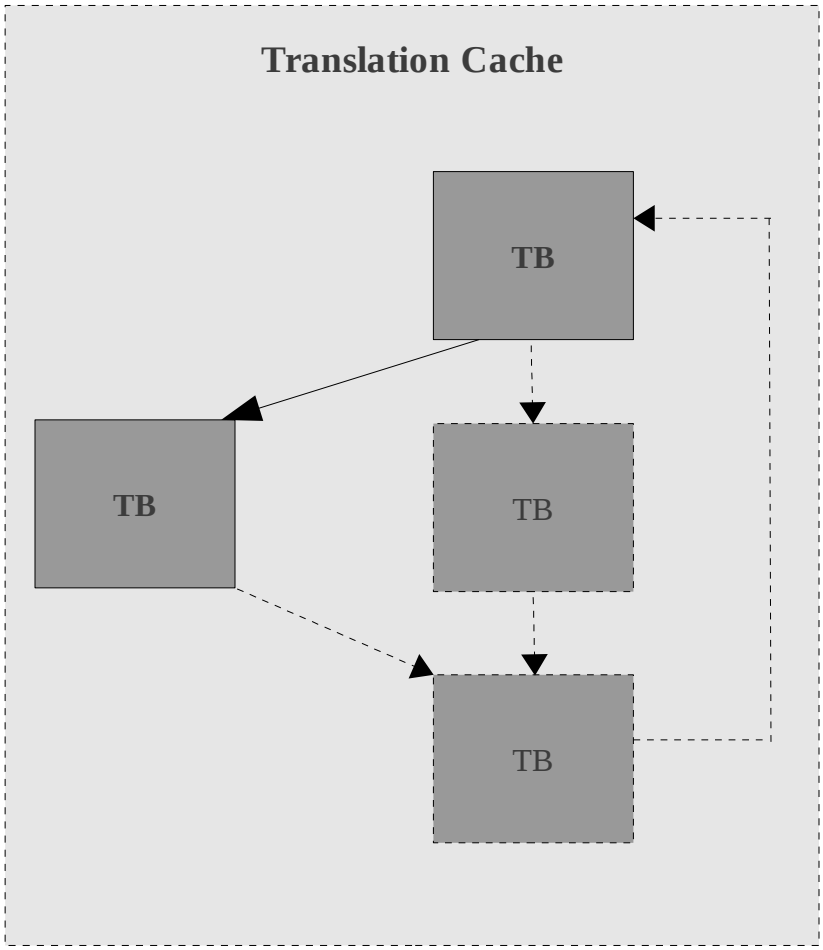
Η εκτέλεση κάθε Translation Block περικλείεται από την εκτέλεση δύο τμημάτων κώδικα ειδικά σχεδιασμένων επονομαζώμενων ως **πρόλογος**(prologue) και **επίλογος**(epilogue). Ο επίλογος αρχικοποιεί τον επεξεργαστή για τον παραγώμενο κώδικα εκτέλεσης και πηδά στο κατάλληλο Translation Block. Ο επίλογος επαναφέρει την CPU στην κανονική κατάσταση και επιστρέφει στην αρχική δομή επανάληψης(mail loop). Κάθε φορά που ένα Translation Block επιστρέφει στην αρχική δομή επανάληψης και γνωρίζει ότι το επόμενο από αυτό είναι ήδη μεταφρασμένο τότε πηγαίνει απευθείας στο επόμενο και όχι στον επίλογο που θα πήγαινε κανονικά. Αυτό είναι ένα από τα τρικς που κάνει τον QEMU γρήγορο σε σχέση με τους άλλους εξομοιωτές.

Πώς όμως γνωρίζει ένα TB ποιό είναι το επόμενό του και πόσο μάλλον αν είναι και ήδη μεταφρασμένο για να μπορέσει να κάνει το άλμα που θα του δώσει την ταχύτητα την οποία εκθειάσαμε προηγουμένως;

Για να μπορέσουμε να απαντήσουμε χρειαζόμαστε να μιλήσουμε για ένα χαρακτηριστικό της λειτουργίας του QEMU σε σχέση με τα Translation Blocks και αυτή είναι η Αποθήκευση πολλών Translation Blocks σε μία προσωρινή μνήμη(cache).

2.3 Προσωρινή Αποθήκευση Τμημάτων Εντολών Μετάφρασης (Translation Cache)

Μία προσωρινή μνήμη των 32 Mbytes κρατά τις πιο πρόσφατες μεταφράσεις , δηλάδη τα πιο πρόσφατα Translation Blocks όπως αναφέραμε , η οποία αδειάζει εντελώς όταν γεμίσει. Το μέγεθος της μνήμης μπορεί να τροποποιηθεί αλλά η αρχικά δεδομένη τιμή(default) είναι όπως αναφέραμε τα 32 Mbytes(σε παλαιότερες εκδόσεις 16 Mbytes). Σημαντική πληροφορία για την σωστή χρήση της Προσωρινής Αποθήκευσης είναι η γνώση του αν η CPU είναι σε λειτουργία Πυρήνα(Kernel mode) ή Χρήστη(user mode).



Σχήμα 2.3: Περιγραφή αρχιτεκτονικής της μνήμης προσωρινής αποθήκευσης τμημάτων εντολών μεταφράσεων και πιθανών αλμάτων που μπορεί να προκύψουν. Βέλος με έντονη γραμμή το πραγματικό άλμα.

2.4 Αντιστοίχιση Καταχωρητών Φιλοξενούμενης Αρχιτεκτονικής και Οικοδεσπότη

Ο QEMU χρησιμοποιεί έναν στατικό τρόπο στην αντιστοίχηση των καταχωρητών. Αυτό σημαίνει ότι κάθε καταχωρητής της φιλοξενούμενης αρχιτεκτονικής αντιστοιχεί σε έναν έναν γνωστό εκ των προτέρων καταχωρητή ή σε μία συγκεκριμένη θέση της μνήμης. Στις περισσότερες αρχιτεκτονικές οικοδεσπότη ο QEMU αντιστοιχεί τους καταχωρητές σε θέσεις μνήμης και μόνο λίγους τους αντιστοιχεί με πραγματικούς καταχωρητές του οικοδεσπότη. Παλαιότερα η αντιστοίχηση γινόταν με το χέρι για κάθε φιλοξενούμενη CPU αλλά , όπως είχε τεθεί μέσα στους στόχους από τον δημιουργό και την κοινότητα , τώρα πια χρησιμοποιεί συναρτήσεις για να πραγματοποιήσει μέρος της αντιστοίχησης η οποία στην ουσία ήταν κοινότυπη διαδικασία και μπορούσε εύκολα να αυτοματοποιηθεί.

Μία από αυτές τις βοηθητικές συναρτήσεις παρουσιάζεται παρακάτω:

```
//path: qemu-0.14.0/target-ppc/helper_regs.h:52

static inline void hreg_compute_hflags(CPUPPCState *env)
{
   target_ulong hflags_mask;

   /* We 'forget' FE0 & FE1: we'll never generate imprecise
    exceptions*/
   hflags_mask = (1 << MSR_VR) | (1 << MSR_AP) | (1 << MSR_SA) |
       (1 << MSR_PR) | (1 << MSR_FP) | (1 << MSR_SE) | (1 << MSR_BE)
|
       (1 << MSR_LE);
   hflags_mask |= (1ULL << MSR_CM) | (1ULL << MSR_SF) | MSR_HVB;
   hreg_compute_mem_idx(env);
   env->hflags = env->msr & hflags_mask;
   /* Merge with hflags coming from other registers */
   env->hflags |= env->hflags_nmsr;
}
```

Η παραπάνω συνάρτηση κάνει τον υπολογισμό παίρνοντας τους καταχωρητές από την φιλοξενούμενη αρχιτεκτονική PowerPc, αυτό φαίνεται διότι η συνάρτηση δέχεται ως όρισμα την `CPUPPCState` η οποία είναι μια δομή της C η οποία περιέχει τους καταχωρητές , και κάνει τις μετατροπές ώστε να αντιστοιχηθεί με καταχωρητές της x86.

Κεφάλαιο 3 – Αναπαράσταση Δικτυακών Συσκευών στον QEMU

Τα συστήματα που εξομοιώνονται από τον QEMU κατηγοριοποιούνται σε πλακέτες(boards). Κατά τη φάση της αρχικοποίησης κάθε πλακέτα εκκινεί έναν αριθμό από CPUs, συσκευές, RAM και ROM. Κάθε συσκευή με τη σειρά της μπορεί να εκχωρήσει θύρες Εισόδου/Εξόδου(I/O) ή περιοχές μνήμης σε κομμάτια κώδικα που διαχειρίζονται τη χρήση τους, τους λεγόμενους handlers. Όταν ο εξομοιωτής ξεκινά, μία πρόσβαση στις θύρες Εισόδου/Εξόδου ή στις περιοχές μνήμης οι οποίες δόθηκαν στη συσκευή τότε καλείται και ο κατάλληλος handler.

Ο QEMU υποστηρίζει διάφορες κατηγορίες συσκευών όπως σειριακές και παράλληλες πόρτες, USB, οδηγούς συσκευών και δικτυακές συσκευές οι οποίες είναι και ο πυρήνας αυτού του Κεφαλαίου και γενικότερα της εργασίας. Η υποστήριξη αυτή επιτυγχάνεται μέσω διάφορων βιβλιοθηκών που προσφέρει ο QEMU για ευκολότερη διασύνδεση με τα παραπάνω επίπεδα υλοποίησης. Αυτές οι βιβλιοθήκες αποκρύπτουν τις λεπτομέριες από τις συσκευές, όπως είναι παραδείγματος χάρην ο τρόπος που χρησιμοποιείται η συσκευή στην εκάστοτε αρχιτεκτονική.

Συνήθως οι συσκευές υλοποιούν μία μέθοδο επαναφοράς και μία μέθοδο υποστήριξης καταχωρητών με σκοπό την αποθήκευση και εκκίνηση της κατάστασης της οποίας βρίσκεται μία συσκευή. Επίσης, οι συσκευές χρησιμοποιούν χρονομετρητές, συνήθως η χρήση τους έρχεται σε στενή επαφή με τη χρήση των bottom halves (BHs).

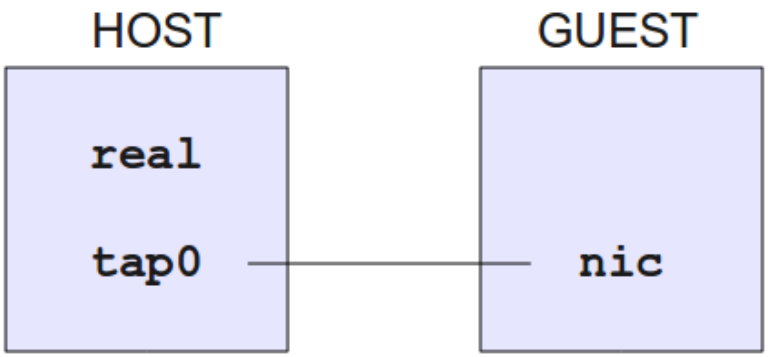
3.1 Η ιδέα πίσω από την αναπαράσταση των δικτυακών συσκευών

Ο QEMU για να μπορέσει να καταστήσει δυνατή τη λειτουργία μιας δικτυακής συσκευής , ας υποθέσουμε μία κάρτα δικτύου(nic-**n**etwork **i**nterface **c**ard) , δημιουργεί δύο(2) εικονικές συσκευές:

- μία προσομοιούμενη συσκευή όπου ακολουθεί τις αρχές της φιλοξενούμενης αρχιτεκτονικής
- μία προσομοιούμενη συσκευή όπου λειτουργεί ως διαμεσολαβητής μεταξύ της προσομομοιούμενης και της πραγματικής συσκευής. Η λεγόμενη `tap` συσκευή.

Οπότε η φιλοξενούμενη προσομοιούμενη συσκευή ας την ονομάσουμε `nic` επικοινωνεί με την `tap` και η `tap` με τη σειρά της προωθεί όλα τα πακέτα στην πραγματική συσκευή ας την ονομάσουμε `real`. Ίσως κάποιος αναρωτηθεί γιατί δεν επικοινωνεί η `nic` απευθείας με την `real`, αλλά είναι εύκολο να απαντηθεί αυτό το ερώτημα αν σκεφτεί κανείς ότι τότε θα μπορούσε να προσωμιωθεί μόνο μια συσκευή άρα μόνο ένα στιγμιότυπο του QEMU θα μπορούσαμε να ενεργοποιήσουμε κάθε φορά. Με την `tap` ο QEMU μπορεί να τρέξει πάνω από ένα στιγμιότυπο και όλα να έχουν ενεργοποιημένη δικτύωση.

Σχηματικά τα παραπάνω παίρνουν τη μορφή:



Εικόνα 3.1: Βασικό σενάριο χρήσης της tap συσκευής

3.2 TUN/TAP Συσκευές και Οδηγοί Συσκευών

Για να καταλάβουμε σε μεγαλύτερο βάθος τη διαδικασία που ακολουθεί ο QEMU για να αναπαραστήσει μία δικτυακή συσκευή θα αναλύσουμε κάθε εμπλεκόμενη οντότητα και τις αναδράσεις μεταξύ τους. Θα ξεκινήσουμε με την `tap` συσκευή.

Η `tap` συσκευή που παρουσιάσαμε στην προηγούμενη ενότητα είναι στην ουσία η υλοποίηση μίας διεπαφής(interface) που προσφέρεται από λειτουργικά συστήματα UNIX-like τα οποία μπορούν να δώσουν το δικαίωμα σε απλά προγράμματα επιπέδου χρήστη να μπορούν να “βλέπουν” την δικτυακή κίνηση μέχρι και σε επίπεδο Ethernet μέσα στον πυρήνα και να επενεργούν σε αυτή.

Ο QEMU εκμεταλλεύεται αυτή τη δυνατότητα που του δίνεται από τον Kernel και δημιουργεί τις `tap` συσκευές όπως είπαμε. Η δημιουργία αυτών των συσκευών έγκειται όμως στην χρησιμοποιήση του TUN/TAP οδηγού συσκευής(driver) που έχει υλοποιηθεί από την κοινότητα του λειτουργικού συστήματος Linux. Για να μπορέσει κάποιο πρόγραμμα(στην περίπτωση μας ο QEMU) να τον χρησιμοποιήσει χρειάζεται να ανοίξει το αρχείο `/dev/net/tun` και να εκτελέσει την κατάλληλη `ioclt()` για να εγγράψει τη δικτυακή συσκευή στον Kernel. Με το που ολοκληρωθεί η εγγραφή θα εμφανιστεί η `tap` του παραδείγματός μας. Όταν το πρόγραμμα διακόψει την προσωμοίωση τότε αυτομάτως σταματά και η λειτουργία της δικτυακής συσκευής `tap` και όλα τα συσχετιζόμενα αρχεία με αυτή διαγράφονται.

Παρακάτω παραθέτουμε και ύστερα επεξηγούμε τον κώδικα που χρησιμοποιεί ο QEMU για κάνει την εγγραφή για την οποία μιλήσαμε:

```
//path: qemu-0.14.0/net/tap-linux.c:38

38 int tap_open(char *ifname, int ifname_size, int *vnet_hdr, int
                                                    vnet_hdr_required)
39 {
40     struct ifreq ifr;
41     int fd, ret;
42
43     TFR(fd = open(PATH_NET_TUN, O_RDWR));
44     if (fd < 0) {
45         error_report("could not open %s: %m", PATH_NET_TUN);
46         return -1;
47     }
48     memset(&ifr, 0, sizeof(ifr));
49     ifr.ifr_flags = IFF_TAP | IFF_NO_PI;
50
51     if (*vnet_hdr) {
52         unsigned int features;
53
54         if (ioctl(fd, TUNGETFEATURES, &features) == 0 &&
55             features & IFF_VNET_HDR) {
56             *vnet_hdr = 1;
57             ifr.ifr_flags |= IFF_VNET_HDR;
58         } else {
59             *vnet_hdr = 0;
60         }
61
62         if (vnet_hdr_required && !*vnet_hdr) {
63             error_report("vnet_hdr=1 requested, but no kernel "
64                          "support for IFF_VNET_HDR available");
65             close(fd);
```

```
66              return -1;
67          }
68      }
69
70      if (ifname[0] != '\0')
71          pstrcpy(ifr.ifr_name, IFNAMSIZ, ifname);
72      else
73          pstrcpy(ifr.ifr_name, IFNAMSIZ, "tap%d");
74      ret = ioctl(fd, TUNSETIFF, (void *) &ifr);
75      if (ret != 0) {
76          error_report("could not configure %s (%s): %m", PATH_NET_TUN,
                        ifr.ifr_name);
77          close(fd);
78          return -1;
79      }
80      pstrcpy(ifname, ifname_size, ifr.ifr_name);
81      fcntl(fd, F_SETFL, O_NONBLOCK);
82      return fd;
83  }
```

Παρακάτω αναλύουμε τις σημαντικές γραμμές κώδικα όπου θα μας βοηθήσουν στην κατανόηση του αποσπάσματος κώδικα παραπάνω.

Γραμμή 38:

Εδώ βλέπουμε τα ορίσματα της συνάρτησης δημιουργίας και εγγραφής μιας νέας εικονικής συσκευής `tap` για την οποία μιλάμε.

Η παράμετρος `char *ifname` (για παράδειγμα `tap0`, `tap2` κλπ) περιέχει την ονομασία της συσκευής στη δική μας περίπτωση είναι `tap0`. Κάθε όνομα θα μπορούσε να χρησιμοποιηθεί αν και είναι καλύτερα η διαλογή ενός αντιπροσωπευτικού ονόματος όπου να σηματοδοτεί ποιο interface είναι. Συνήθως από την κοινότητα χρησιμοποιούνται ονόματα τύπου `tapX`. Αν το `*ifname` είναι `'\0'` τότε ο Kernel θα δημιουργήσει ένα δικό του με βάση τον τύπο που είπαμε πιο πρίν.

Η παράμετρος `int ifname_size` δείχνει απλά το μήκος του `string`.

Η παράμετρος `int *vnet_hdr` είναι σημαίες(flags) όπου λένε στον Kernel ποιες από τις εικονικές συσκευές είναι `tap` και ποιες είναι άλλου τύπου όπου ο QEMU υποστηρίζει αλλά εμάς μας απασχολούν μόνο οι `tap`.

Η παράμετρος `int vnet_hdr_required` είναι και αυτή μία σημαία όπου μας προειδοποιεί αν όντως χρειάζεται να ερευνήσουμε ή όχι τις παραπάνω σημαίες `int *vnet_hdr` .

Γραμμή 74:

Σε αυτή τη γραμμή μπορούμε να δούμε τη χρήση της `ioctl()` και όπως είχαμε αναφέρει και στην αρχή της ενότητας εδώ λαμβάνει μέρος η δημιουργία της εικονικής συσκευής.

Γραμμή 80:

Αν η δημιουργία ήταν επιτυχής τότε το όνομα της εκχωρείται μέσα στο πρώτο πεδίο της `*ifname`.

Η διεπαφή TAP που έχει υλοποιηθεί από τον QEMU έχει προσθέσει και μία συνάρτηση για την εξομάλυνση των αποδεχόμενων bit μέσω buffers. Παρακάτω παρουσιάζεται και πάλι κομμάτι από τον πηγαίο κώδικα του QEMU και επεξηγείται καταλλήλως ανά γραμμή. Έτσι με βάση τα δύο κομμάτια του κώδικα μπορεί κάποιος να μελετήσει και να καταλάβει όλη τη διαδικασία που ο QEMU πραγματοποιεί ώστε να δημιουργήσει αλλά και να κρατήσει στην εύρυθμη λειτουργία του τη νέα εικονική συσκευή. Σε παρακάτω ενότητα συνεχίζουμε την επεξήγηση του πηγαίου κώδικα της διεπαφής TAP αλλά σε ένα περιβάλλον όπου η επεξήγηση του θα είναι ευκολότερα κατανοητή λόγω μιας λεπτομερούς εφαρμογής που θα παρουσιαστεί.

```
//path: qemu-0.14.0/net/tap-linux.c:95

95 #define TAP_DEFAULT_SNDBUF 0
96
97 int tap_set_sndbuf(int fd, QemuOpts *opts)
98 {
99     int sndbuf;
100
101     sndbuf = qemu_opt_get_size(opts, "sndbuf", TAP_DEFAULT_SNDBUF);
102     if (!sndbuf) {
103         sndbuf = INT_MAX;
```

```
104     }
105
106     if (ioctl(fd, TUNSETSNDBUF, &sndbuf) == -1 && qemu_opt_get(opts,
                                                            "sndbuf")) {
107         error_report("TUNSETSNDBUF ioctl failed: %s", strerror(errno));
108         return -1;
109     }
110     return 0;
111 }
```

Η συνάρτηση `tap_set_sndbuf()` ύλοποιεί ένα είδος ελέγχου ροής για μία `tap`X εικονική συσκευή. Οδηγίες των δημιουργών του TAP interface του QEMU αναφέρουν ότι για να αποφευχθεί η απώλεια πακέτων, ο buffer θα χρειαστεί να πάρει μικρότερες τιμές από το μέγεθος της ουράς `Tx` ενός τυχαίου δικτυακού interface. Οι Ethernet NICs γενικά έχουν `txqueuelen = 1000`, οπότε `1MB` είναι μία καλή τιμή, δεδομένου ότι η `MTU=1500 byte`.

3.3 Τα βήματα μιας ρεαλιστικής βασικής εφαρμογής

Βήμα 1ο – Χαρτογράφηση της Συσκευής

Καθώς εξηγήσαμε πώς ο QEMU χρησιμοποιεί τη διεπαφή TAP για να προσομοιώσει τις εικονικές συσκευές παρακάτω θα παρουσιάσουμε μία εφαρμογή της θεωρίας χρησιμοποιώντας τη συσκευή RTL8139c.

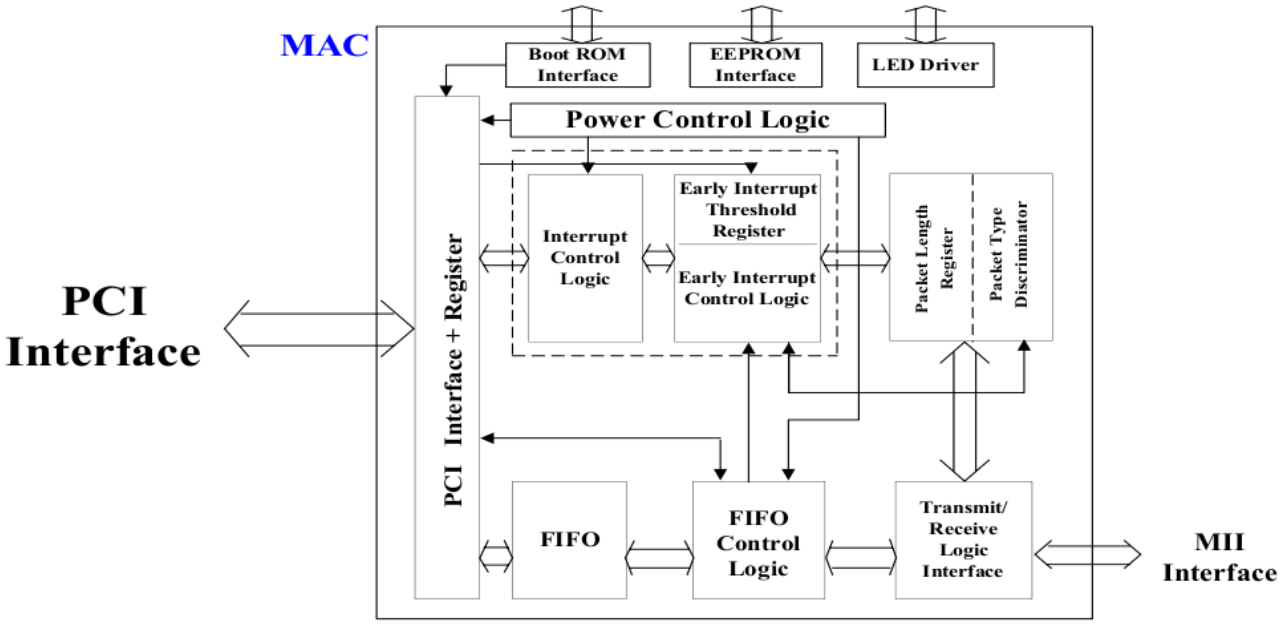
Η συσκευή RTL8139c είναι ένας 32bit Fast Ethernet Controller πλήρους συμβατότητας ως προς τις επιταγές του IEEE 802.3u 100Base-T αλλά και του IEEE 802.3x Full Duplex Flow Control. Η λειτουργία του σε full-duplex καθιστά δυνατή την αύξηση του εύρους ζώνης έως και 200 Mbps χωρίς πρόσθετο κόστος [22].

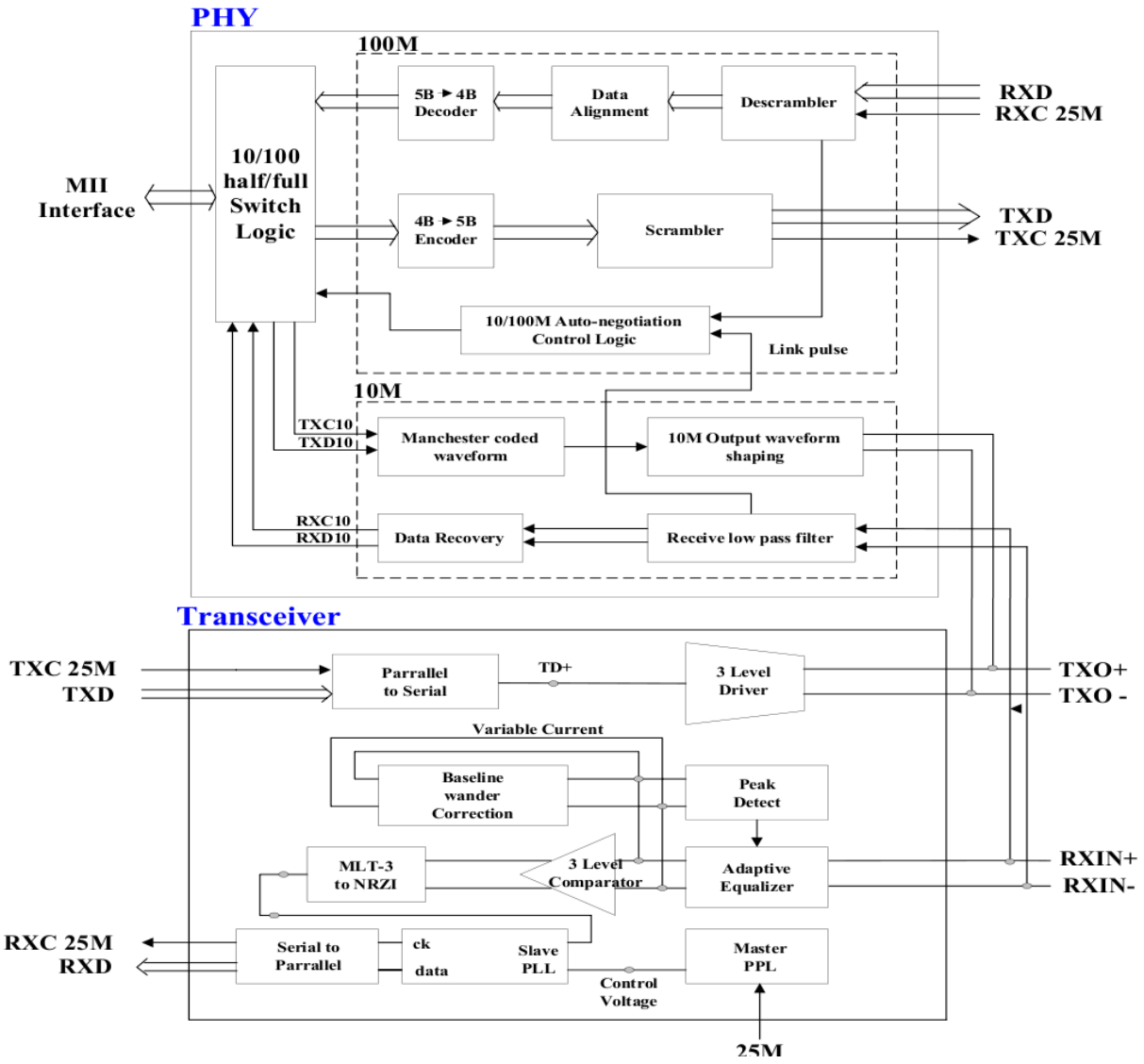
Η λογική που θα ακολουθήσουμε θα είναι αυτή της επεξήγησης των λειτουργιών του RTL8139c και παράλληλα η παράθεση πηγαίου κώδικα των αντίστοιχων αποσπασμάτων. Έτσι κάποιος μπορεί να δεί πως πραγματικά ο QEMU υλοποιεί(προσομοιώνει) μία δικτυακή συσκευή.

Οι λειτουργίες του RTL8139c μπορούν να βρεθούν από το Εγχειρίδιο(datasheet & manual) της κατασκευάστριας εταιρίας του όπου εμείς θα το χρησιμοποιήσουμε ως οδηγό. Ο τρόπος για να το

αποκτήσει οποιοσδήποτε αναγνώστης το επιθυμεί αναφέρεται στο [17 - Μηλιώνης2006].

Για λόγους διευκόλυνσης παρουσιάζουμε εδώ το Block Diagram.





Εικόνα 3.2: Block Diagram της συσκευής RTL8139 [22]

Αρχίζουμε δίνοντας έναν πίνακα όπου δείχνει τις αντιστοιχίες στην ονοματοδοσία των καταχωρητών του RTL8139c που δίνεται στο επίσημο Εγχειρίδιο και στην ονοματοδοσία των δημιουργών & επιτηρητών του QEMU στο κατάλληλο αρχείο του πηγαίου κώδικα .

```
//path: qemu-0.14.0/hw/rtl8139.c:86 - 133


86 /* Symbolic offsets to registers. */
87 enum RTL8139_registers {
88     MAC0 = 0,        /* Ethernet hardware address. */
89     MAR0 = 8,        /* Multicast filter. */
90     TxStatus0 = 0x10,/*Transmit status (Four 32bit registers).C mode
only*/
91     /* Dump Tally Conter control register(64bit). C+ mode only */
92     TxAddr0 = 0x20,  /* Tx descriptors (also four 32bit). */
93     RxBuf = 0x30,
94     ChipCmd = 0x37,
95     RxBufPtr = 0x38,
96     RxBufAddr = 0x3A,
97     IntrMask = 0x3C,
98     IntrStatus = 0x3E,
99     TxConfig = 0x40,
100    RxConfig = 0x44,
101    Timer = 0x48,        /* A general-purpose counter. */
102    RxMissed = 0x4C,    /* 24 bits valid, write clears. */
103    Cfg9346 = 0x50,
104    Config0 = 0x51,
105    Config1 = 0x52,
106    FlashReg = 0x54,
107    MediaStatus = 0x58,
108    Config3 = 0x59,
109    Config4 = 0x5A,        /* absent on RTL-8139A */
```

```
110     HltClk = 0x5B,
111     MultiIntr = 0x5C,
112     PCIRevisionID = 0x5E,
113     TxSummary = 0x60,/*TSAD register.Transmit Status of All Descriptor*/
114     BasicModeCtrl = 0x62,
115     BasicModeStatus = 0x64,
116     NWayAdvert = 0x66,
117     NWayLPAR = 0x68,
118     NWayExpansion = 0x6A,
119     /* Undocumented registers, but required for proper operation. */
120     FIFOTMS = 0x70,     /* FIFO Control and test. */
121     CSCR = 0x74,        /* Chip Status and Configuration Register. */
122     PARA78 = 0x78,
123     PARA7c = 0x7c,        /* Magic transceiver parameter register. */
124     Config5 = 0xD8,        /* absent on RTL-8139A */
125     /* C+ mode */
126     TxPoll = 0xD9,     /* Tell chip to check Tx descriptors for work */
127     RxMaxSize = 0xDA,  /* Max size of an Rx packet (8169 only) */
128     CpCmd = 0xE0,   /* C+ Command register (C+ mode only) */
129     IntrMitigate = 0xE2,     /* rx/tx interrupt mitigation control */
130     RxRingAddrLO = 0xE4,     /* 64-bit start addr of Rx ring */
131     RxRingAddrHI = 0xE8,     /* 64-bit start addr of Rx ring */
132     TxThresh = 0xEC,         /* Early Tx threshold */
133 };
```

Ο εξαγώμενος πίνακας για τις αντιστοιχίες μεταξύ του πηγαίου κώδικα του QEMU και των Καταχωρητών(Registers) στον RTL8139c είναι ο εξής:

Πίνακας 3.1: Περιγραφή Καταχωρητών και αντιστοιχίες με τον πηγαίο κώδικα του QEMU [22]

Offset	Καταχωρητής	Ονομασία Καταχωρητή στον κώδικα του QEMU	Πλήρης Ονομασία
0001h	-	MAC0 = 0	-
0008h	MAR0	MAR0 = 8	Multicast Register 0
0010h	TSD0	TxStatus0 = 0x10	Trasmit Status of Descriptor 0
0020h	TSAD1	TxAddr0 = 0x20	Trasmit Status of Descriptor 0
0030h	RBSTART	RxBuf = 0x30	Recieve (Rx) Buffer Start Address
0037h	CR	ChipCmd = 0x37	Command Register
0038h	CAPR	RxBufPtr = 0x38	Current Address of Packet Read
003Ah	CBR	RxBufAddr = 0x3A	Current Buffer Address
003Ch	IMR	IntrMask = 0x3C	Interrupt Mask Register
003Eh	ISR	IntrStatus = 0x3E	Interrupt Status Register
0040h	TCR	TxConfig = 0x40	Transmit (Tx) Configuration Table
0044h	RCR	RxConfig = 0x44	Recieve (Tx) Configuration Table
0048h	TCTR	Timer = 0x48	Timer Count Register
004Ch	MPC	RxMissed = 0x4C	Missed Packet Counter
0050h	9346CR	Cfg9346 = 0x50	93C46 (93C56) Commad Register
0051h	CONFIG0	Config0 = 0x51	Configuration Register 0
0052h	CONFIG1	Config1 = 0x52	Configuration Register 1
0058h	MSR	MediaStatus = 0x58	Media Status Register
0059h	CONFIG3	Config3 = 0x59	Configuration Register 3
005Ah	CONFIG4	Config4 = 0x5A	Configuration Register 4
005Ch	MULINT	MultiIntr = 0x5C	Multiple Interrupt Select
005Eh	RERID	PCIRevisionID = 0x5E	PCI Revision ID = 10h
0060h	TSAD	TxSummary = 0x60	Transmit Status of All Descriptors

0062h	BMCR	BasicModeCtrl = 0x62	Basic Mode Control Register
0064h	BMSR	BasicModeStatus = x64	Basic Mode Status Register
0066h	ANAR	NWayAdvert = 0x66	Auto-Negotiation Advertisement Register
0068h	ANLPAR	NWayLPAR = 0x68	Auto-Negotiation Link Partner Register
006Ah	ANER	NWayExpansion = 0x6A	Auto-Negotiation Expansion Register
0074h	CSCR	CSCR = 0x74	CS Configuration Register
0078h	PHY1_PARM	PARA78 = 0x78	PHY parameter 1
007Ch	TW_PARM	PARA7c = 0x7c	Twister parameter
00D4h	FLASH	FlashReg = 0x54	Flash Memory Read/Write Register
00D8h	CONFIG5	Config5 = OxD8	Configuration Register 5

Συνεχίζουμε δείχνοντας τον κώδικα και τον Πίνακα αντιστοιχιών για τον Recieve Status Registers, ύστερα για τον Transmit Status Register και τέλος για τους Interrupt Mask Register και Interrupt Status Register οι οποίοι είναι σημαντικοί για τη λειτουργία του RTL8139c. Τους παρουσιάσαμε πρώτον για να αποτελέσει μία σοβαρή ένδειξη του πως ο QEMU διευθυνσιοδοτεί τους Καταχωρητές της συσκευής αλλά και δεύτερον για να μπορέσει να πλοηγηθεί εύκολα μέσα στον κώδικα του QEMU για τον RTL8139 κάποιος που έχει στα χέρια του την παρούσα εργασία αλλά και το Datasheet.

```
//path: qemu-0.14.0/hw/rtl8139.c:179 - 189

179 enum RxStatusBits {
180     RxMulticast = 0x8000,
181     RxPhysical = 0x4000,
182     RxBroadcast = 0x2000,
183     RxBadSymbol = 0x0020,
184     RxRunt = 0x0010,
185     RxTooLong = 0x0008,
186     RxCRCErr = 0x0004,
187     RxBadAlign = 0x0002,
188     RxStatusOK = 0x0001,
189 };
```

Πίνακας 3.2: Επεξήγηση της δομής του Recieve Status Register και η αντιστοίχηση στον πηγαίο κώδικα [22]

Bit	Καταχωρητής	Ονομασία Καταχωρητή στον κώδικα του QEMU	Πλήρης Ονομασία
15	MAR	RxMulticast = 0x8000	Multicast Address Recieved
14	PAM	RxPhysical = 0x4000	Physical Address Recieved
13	BAR	RxBoardcast = 0x2000	Broadcast Address Recieved
12-6	-	-	Reserved
5	ISE	RxBadSymbol = 0x0020	Invalid Symbol Error
4	RUNT	RxRunt = 0x0010	Runt Packet Recieved
3	LONG	RxTooLong = 0x0008	Long Packet
2	CRC	RxCRCErr = 0x0004	CRC Error
1	FAE	RxBadAlign = 0x0002	Frame Alignment Error
0	ROK	RxStatusOK = 0x0001	Receive OK

Ακολουθεί ο κώδικας για τον Transmit Status Register και ύστερα ο Πίνακας αντιστοιχίσεων.

```
//path: qemu-0.14.0/hw/rtl8139.c:171 - 178

171 enum TxStatusBits {
172     TxHostOwns = 0x2000,
173     TxUnderrun = 0x4000,
174     TxStatOK = 0x8000,
175     TxOutOfWindow = 0x20000000,
176     TxAborted = 0x40000000,
177     TxCarrierLost = 0x80000000,
178 };
```

Πίνακας 3.3: Επεξήγηση της δομής του Transmit Status Register και η αντιστοίχηση στον πηγαίο κώδικα [22]

Bit	Καταχωρητής	Ονομασία Καταχωρητή στον κώδικα του QEMU	Πλήρης Ονομασία
31	CRS	TxCarrierLost = 0x80000000	Carrier Sense Lost
30	TABT	TxAborted = 0x40000000	Transmit Abort
29	OWC	TxOutOfWindow = 0x20000000	Out of Window Collision
28	CDH	-	CD Heart Beat
27-24	NCC3-0	-	Number of Collision Count
23-22	-	-	Reserved
21-16	ERTXTH5-0	-	Early Tx Threshold
15	TOK	TxStatOK = 0x8000	Transmit OK
14	TUN	TxUnderrun = 0x4000	Transmit FIFO Underrun
13	OWN	TxHostOwns = 0x2000	OWN
12-0	SIZE	-	Descriptor Size

Ακολουθεί ο κώδικας για τον Interrupt Status Register και ύστερα ο Πίνακας αντιστοιχίσεων.

```
//path: qemu-0.14.0/hw/rtl8139.c:157 - 169

156 /* Interrupt register bits, using my own meaningful names. */
157 enum IntrStatusBits {
158     PCIErr = 0x8000,
159     PCSTimeout = 0x4000,
160     RxFIFOOver = 0x40,
161     RxUnderrun = 0x20,
162     RxOverflow = 0x10,
163     TxErr = 0x08,
164     TxOK = 0x04,
```

```
165     RxErr = 0x02,
166     RxOK = 0x01,
167
168     RxAckBits = RxFIFOOver | RxOverflow | RxOK,
169 };
```

Πίνακας 3.4: Επεξήγηση της δομής του Interrupt Status Register και η αντιστοίχηση στον πηγαίο κώδικα [22]

Bit	Καταχωρητής	Ονομασία Καταχωρητή στον κώδικα του QEMU	Πλήρης Ονομασία
15	SERR	PCIErr = 0x8000	System Error Interrupt
14	TimeOut	PCSTimeout = 0x4000	Time Out Interrupt
13	LenChg	-	Cable Length Change Interrupt
12-7	-	-	Reserved
6	FOVW	0x40	Rx FIFO Overflow Interrupt
5	PUN/LinkChg	RxUnderrun = 0x20	Packet Underrun/ Link Change Interrupt
4	RXOVW	RxOverflow = 0x10	Rx Buffer Overflow Interrupt
3	TER	TxErr = 0x08	Transmit Error Interrupt
2	TOK	TxOK = 0x04	Transmit OK Interrupt
1	RER	RxErr = 0x02	Recieve Error Interrupt
0	ROK	RxOK = 0x01	Recieve OK Interrupt

Βήμα 2ο – Εγγραφή και Δημιουργία της Δικτυακής Συσκευής

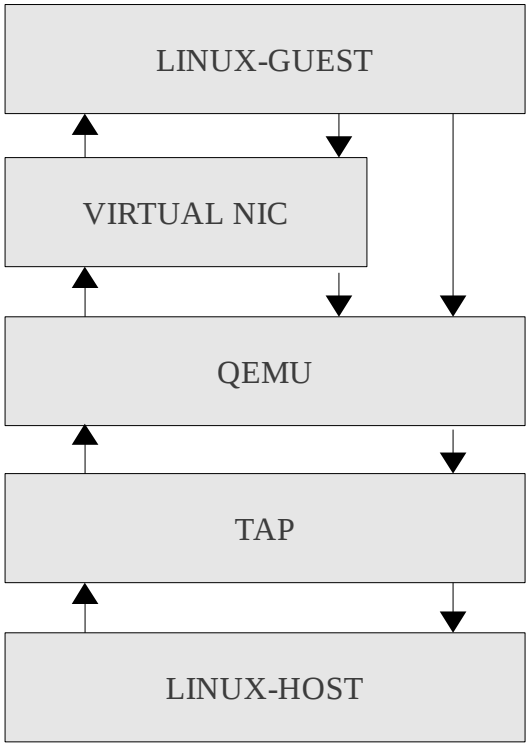
Αυτό το στάδιο το περιγράψαμε και στην **Ενότητα 3.2** λεπτομερώς. Συνοπτικά θα αναφέρουμε ότι ο QEMU εκμεταλλεύεται το δικαίωμα που του δίνεται από τον Kernel μέσω του TAP interface(ή virtual

driver) για να εγγράψει & δημιουργήσει μια νέα συσκευή. Σε αυτό το σημείο επίσης πραγματοποιείται και η διασύνδεση με την `nic` στην περίπτωση μας η δικτυακή συσκευή RTL8139c.

Βήμα 3ο – Γέννηση και Εξυπηρέτηση Αιτήσεων Διακοπών

Η εικονική συσκευή(`virtual nic`) έχει ήδη προσομοιωθεί σε αυτό το Βήμα και έχει υλοποιημένες διεργασίες που επιτρέπουν τη γέννηση και την αποστολή αιτήσεων στο Φιλοξενούμενο Λειτουργικό Σύστημα όπου με τη σειρά του ο Kernel αποφασίζει ανάλογα με το χρονοπρογραμματισμό του πότε θα την εξυπηρετήσει.

Για να καταλάβουμε καλύτερα ας δούμε σχηματικά τις διασυνδέσεις όλων των συστατικών στην περίπτωση που η `nic` κάνει μια αίτηση για να μεταδώσει δεδομένα:



Εικόνα 3.3: Σχηματική απεικόνιση των συστατικών και των αναδράσεων τους στη διάρκεια προσομοίωσης

Για να έχουμε ολική εικόνα χρειάζεται να δούμε τον πηγαίο κώδικα με τον οποίο η εικονική συσκευή επικοινωνεί με το Φιλοξενούμενο Λειτουργικό Σύστημα. Ακολουθούν και επεξηγούνται τα επιλεγμένα

αποσπάσματα κώδικα παρακάτω:

```
//path: qemu-0.14.0/hw/rtl8139.c:2546 - 2577

2546 static void rtl8139_IntrStatus_write(RTL8139State *s, uint32_t val)
2547 {
2548     DEBUG_PRINT(("RTL8139: IntrStatus write(w) val=0x%04x\n", val));
2549
2550 #if 0
2551
2552     /* writing to ISR has no effect */
2553
2554     return;
2555
2556 #else
2557     uint16_t newStatus = s->IntrStatus & ~val;
2558
2559     /* mask unwriteable bits */
2560     newStatus = SET_MASKED(newStatus, 0x1e00, s->IntrStatus);
2561
2562     /* writing 1 to interrupt status register bit clears it */
2563     s->IntrStatus = 0;
2564     rtl8139_update_irq(s);
2565
2566     s->IntrStatus = newStatus;
2567     /*
2568      * Computing if we miss an interrupt here is not that correct but
2569      * considered that we should have had already an interrupt
2570      * and probably emulated is slower is better to assume this 2571resetting was
*/done before testing on previous rtl8139_update_irq lead to IRQ loosing */
2573     rtl8139_set_next_tctr_time(s, qemu_get_clock(vm_clock));
2574     rtl8139_update_irq(s);
```

```
2575
2576 #endif
2577 };
```

Η συνάρτηση `rtl8139_IntrStatus_write( )` θέτει ένα αίτημα διακοπής της εικονικής συσκευής – `nic` μέσω της του καταχωρητή Interrupt Status Register.

Γραμμή 2557:

Θέτει την τιμή στον καταχωρητή με την οποία θα διεκδικήσει την εξυπηρέτησή του από τον Kernel αφού όπως είπαμε ο QEMU προσομοιώνει την εικονική συσκευή και ύστερα της ενεργοποίησης της συσκευής αυτή στέλνει τα αιτήματα διακοπής στον Kernel του φιλοξενούμενου Λειτουργικού Συστήματος όπου λαμβάνοντας αυτή την τιμή από το Αίτημα Διακοπής μπορεί να θέσει σε σειρά την εξυπηρέτησή του.

Γραμμή 2564:

Συγκεκριμένα η εικονική συσκευή παρακολουθεί συνεχώς αν το Αίτημά της ολοκληρώθηκε και αν το επιβεβαιώσει τότε μέσω της συνάρτησης `rtl8139_update_irq(s)` ενημερωώνει την κατάσταση των Αιτημάτων Διακοπής της.

Γραμμή 2573:

Μέσω της συνάρτησης `rtl8139_set_next_tctr_time(s, qemu_get_clock(vm_clock))` γίνεται η ενημέρωση για την επόμενη time-slot όπου η nic θα θέσει το Αίτημα Διακοπής της από την μεριά του QEMU εξ' ού και η χρήση του `qemu_get_clock(vm_clock)` δηλαδή του ρολογιού του QEMU για την διευθέτηση και την παράταξη των Αιτημάτων Διακοπής που θα παραπεμπφθούν στον Kernel.

Ύστερα από την παραπομπή στον Kernel, ο ίδιος ξεκινά την διαδικασία των διακοπών όπου τις αναφέρουμε συνοπτικά , σε βήματα χάριν , πληρότητας:

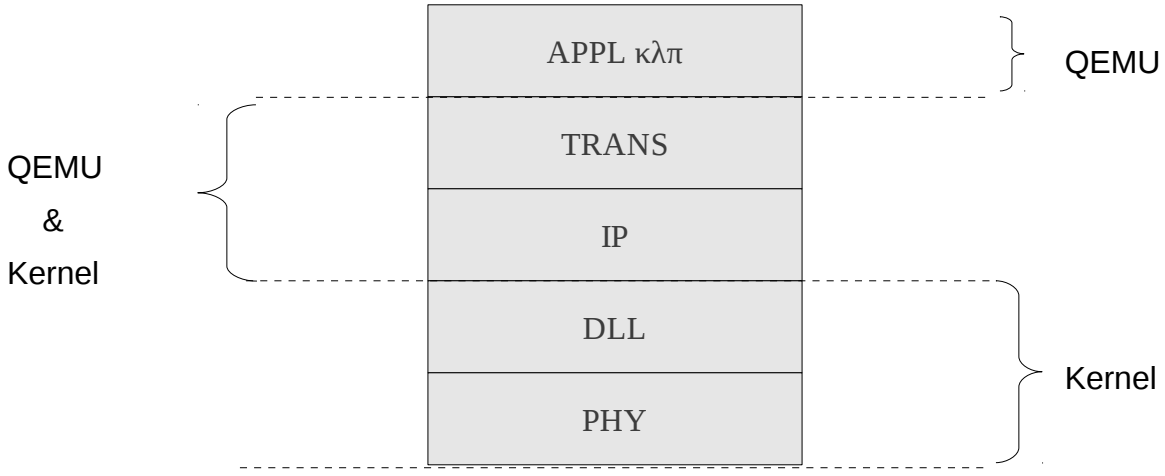
1. Το υλικό αποθηκεύει στη Στοίβα τον Program Counter και άλλες πληροφορίες
2. Το υλικό φορτώνει έναν νέο Program Counter από το διάνυσμα των Αιτήσεων Διακοπής
3. Μέσω μιας διαδικασίας σε γλώσσα προγραμματισμού Assembly οι καταχωρητές αποθηκεύονται

4. Η ίδια διαδικασία δημιουργεί μία νέα Στοίβα
5. Η εξυπηρέτηση της Αίτησης Διακοπής σε προγραμματιστική γλώσσα C λαμβάνει χώρα
6. Ο Scheduler αποφασίζει ποιά εφαρμογή να τρέξει ως επόμενη
7. Η διαδικασία από την προγραμματιστική γλώσσα C επιστρέφει σε Assembly
8. Η διαδικασία σε Assembly ξεκινά μια νέα εφαρμογή όπου θα εξυπηρετηθεί άμεσα

Αυτά τα βήματα επαναλαμβάνονται κάθε φορά που ο QEMU παραδίδει ένα Αίτημα Διακοπής στον Kernel.

Βήμα 4ο – Παράδοση των πακέτων από τις προσομοιούμενες συσκευές στην πραγματική

Όταν εισέλθει ένα πακέτο η `nic` το δίνει στην `tap` και η `tap` με τη σειρά της στην `real` όπου εκεί λαμβάνει δράση πραγματική διαδικασία για την προώθηση ή επεξεργασία από τον Kernel. Αν τυχόν επρόκειτο για μια συσκευή όπου είναι διαφορετικής αριχτεκτονικής από την x86 τότε ότι εξηγήσαμε στο Κεφάλαιο 2 , για τις Δυναμικές Μεταφράσεις και πώς ο QEMU γενικά λειτουργεί , εφαρμόζεται πιστά ώσπου τελικά το Λειτουργικό Σύστημα να δώσει τη λύση εκμεταλλευόμενοι έτσι την ήδη έτοιμη αποτελεσματική λύση που έχει προκύψει από τη σκληρή εργασία των Kernel Developers.



Εικόνα 3.4: Τα στρώματα του μοντέλου OSI στα οποία επιδρά ο QEMU ή ο Kernel

Κεφάλαιο 4 -

Ανάλυση Προσομοίωσης Δικτυακών Συσκευών fast Ethernet για Επεξεργαστές Επικοινωνιών PowerQUICC-II

Σε αυτό το κεφάλαιο θα προσπαθήσουμε να συμβάλλουμε στην ανάπτυξη του QEMU προτείνοντας ένα σχέδιο υλοποίησης με σκοπό τη μερική υποστήριξη επεξεργαστών επικοινωνιών PowerQUICC-II την οποία η κοινότητα του QEMU δεν έχει υλοποιήσει ακόμα.

Φυσικά όπως είναι αντιληπτό η πλήρης υποστήριξη επεξεργαστών PowerQUICC-II απαιτεί και την υλοποίηση του Επεξεργαστικού Πυρήνα αριχτεκτονικής PowerPC , η οποία ευτυχώς έχει υλοποιηθεί από την κοινότητα του QEMU αλλά εμείς θα χρειαστεί να σχεδιάσουμε την διασύνδεσή του με τις υπόλοιπες λειτουργικές μονάδες που απαρτίζουν έναν επεργαστή επικοινωνιών PowerQUICC-II. Ένα από αυτά τα συστατικά είναι ο Ενσωματωμένος Συνεπεξεργαστής Επικοινωνιών(Communication Processor Module-CPM).

4.1 Η Γενική Εικόνα

Μιλήσαμε στην **Ενότητα 3.2** για το πως διασυνδέονται και πως προσομοιώνονται οι συσκευές στον QEMU. Οπότε για να υποστηριχθούν τελικά οι επεξεργαστές PowerQUICC-II θα ληφθεί το εξής σενάριο υπόψιν:

- **Σενάριο :**

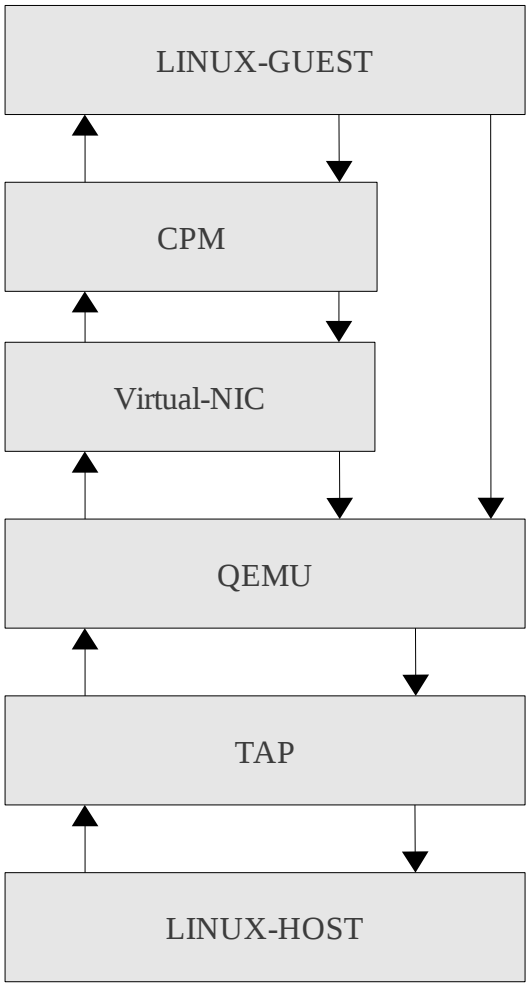
  Ο CPM θα προσομοιωθεί από τον QEMU και θα μπει ως ένα στρώμα παραπάνω από αυτό που τοποθετήσαμε τον RTL8139 στην **Εικόνα 3.3**. Με λίγα λόγια θα γίνει ο μεσάζοντας μεταξύ του Linux-Guest και της προσομοιούμενης κάρτας δικτύου(δηλαδή του επεξεργαστή επικοινωνιών).

  ✔ **Πλεονεκτήματα:**

  1. Δε θα χρειαστούν πολλές TAP συνδέσεις αφού η επικοινωνία του CPM με την Virtual-Nic θα γίνεται εγγενώς από τον QEMU.
  2. Θα γλιτώσουμε την πολυπλοκότητα της υλοποίησης όσο αναφορά τη συσσώρευση των Αιτημάτων Διακοπής και της αποστολής τούς στον Kernel. Δηλαδή, ο CPM θα παίρνει τα Αιτήματα Διακοπής από την Virtual-Nic και αναλόγως την κρίση του θα στέλνει ή θα περιμένει για να εξυπηρετηθούν. Ο υπόλοιπος σχεδιασμός θα παραμείνει ο ίδιος, έτσι θα κάνουμε επαναχρησιμοποίηση των μεθόδων επικοινωνίας που έχει κάνει η κοινότητα του QEMU και θα αποφθευχθεί το customazation.

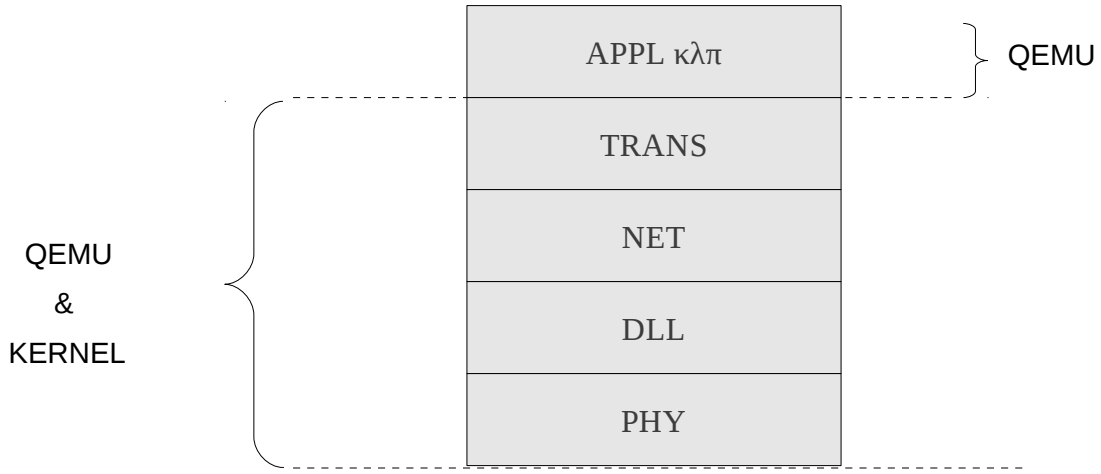
✗ **Μειονεκτήματα:**

1. Το προτέρημα της εγγενούς επικοινωνίας του CPM με την Virtual-Nic είναι ότι δεν υπάρχει υλοποιημένη στον κώδικα του QEMU οπότε θα χρειαστεί να σχεδιαστεί. Είναι ένας από τους στόχους της παρούσας εργασίας.



Εικόνα 4.1: Σχηματική απεικόνιση των συστατικών και των αναδράσεων τους στη διάρκεια προσομοίωσης μαζί με το στρώμα του CPM.

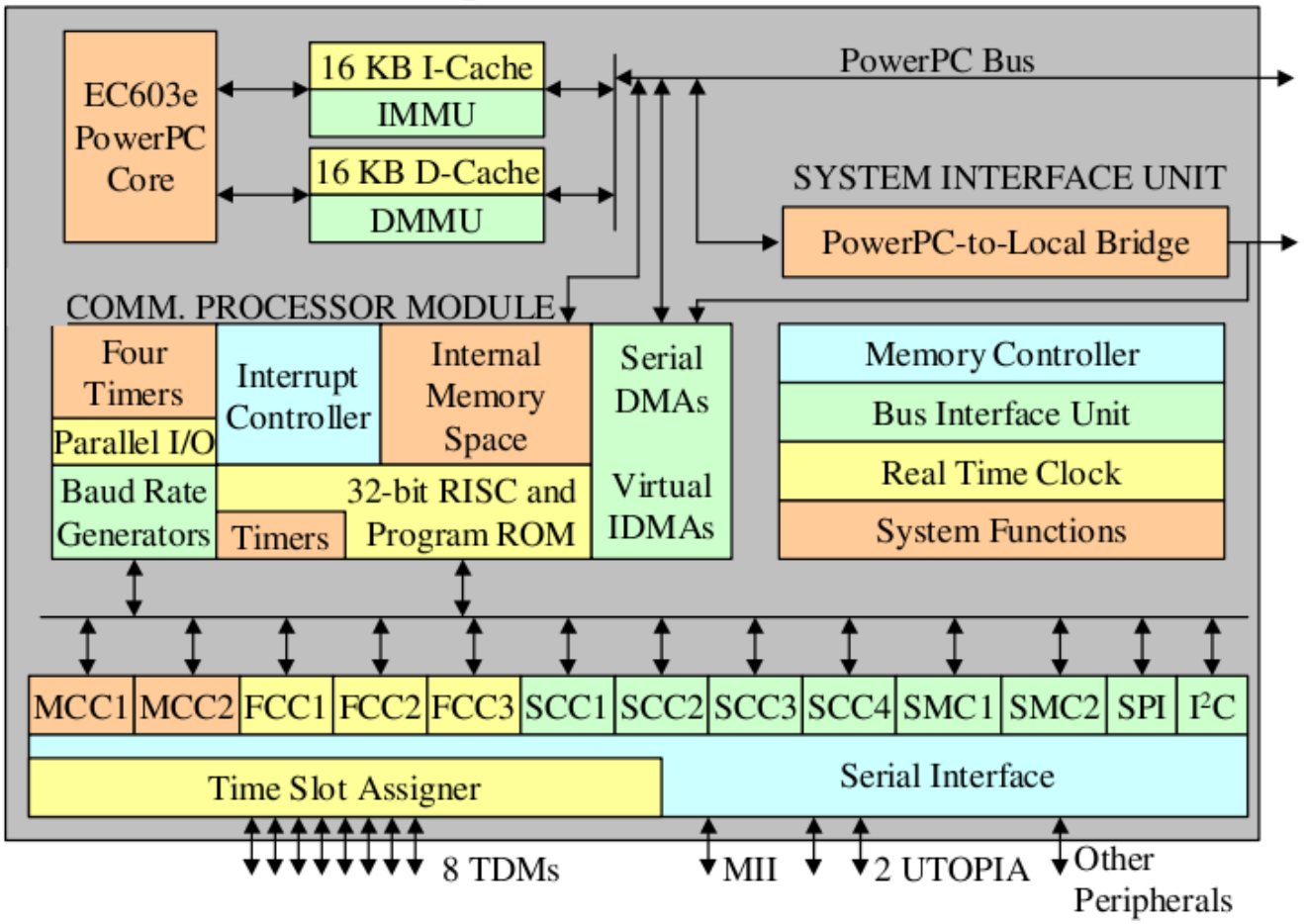
Δηλαδή θα χρειαστεί να προσθέσουμε τον CPM για να αναλάβει τα δύο πρώτα επίπεδα του OSI. Το Φυσικό Επίπεδο (physical layer) και το Επίπεδο Διασύνδεσης Δεδομένων (data link layer). Οπότε ο QEMU θα ανακατευτεί σε ακόμα δύο επίπεδα του OSI από ότι δείξαμε στην **Εικόνα 3.4** .



Εικόνα 4.2: Τα στρώματα του μοντέλου OSI στα οποία επιδρά ο QEMU ή ο Kernel (ύστερα από υλοποίηση του CPM)

4.2 Απαραίτητες προϋποθέσεις για την υποστήριξη του Ενσωματωμένου Συνεπεξεργαστή Επικοινωνιών (CPM)

Στο παρακάτω σχεδιάγραμμα βλέπουμε τα συστατικά του CPM και τις διασυνδέσεις του τον κεντρικό επεξεργαστή δια μέσω της μνήμης του, επίσης παρατηρούμε τη διασύνδεση με τη Μονάδα Διασύνδεσης Συστήματος(SIU), τους σειριακούς ελεγκτές, τους πολυκαναλικούς ελεγκτές(MCC) κλπ αλλά εμάς μας ενδιαφέρει η διασύνδεση με τους ελεγκτές επικοινωνιών. Ότι είναι μέσα στο επονομαζόμενο πλαίσιο Ενσωματωμένος Συνεπεξεργαστής Επικοινωνιών(CPM) είναι τα απαραίτητα συστατικά και οι διασυνδέσεις τους είναι ότι χρειάζεται για να μπορέσουμε να καταφέρουμε μια πλήρη προσομοίωση του CPM.



Εικόνα 4.3: Block Diagram MPC8260. Δείχνει τη δομή του CPM. [17 – Μηλιώνης2006]

Έχοντας τη λίστα με τις απαραίτητες λειτουργικές μονάδες:

- Χρονιστές (Timers)
- Σήματα Εισόδου/Εξόδου (Parallel I/O)
- Γεννήτριες Ρυθμού Μετάδοσης (Baud Rate Generators)
- Χώρος Εσωτερικής Μνήμης (Internal Memory Space)
- Ελεγκτής Αιτήσεων Διακοπής (Interrupt Controller)
- RISC & ROM

- Ελεγκτές Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης (FCC)
- Πολυκαναλικοί Ελεγκτές (MCC)
- Serial Interface
- Time Slot Assigner

τώρα μένει να βρούμε ποιές και πως θα σχεδιάσουμε και υλοποιήσουμε για να μπορέσουμε να καταφέρουμε την προσομοίωση επαναχρησιμοποιώντας όλο το έργο που προσέφεραν οι προγραμματιστές της κοινότητας του QEMU. Επίσης θα ονοματίσουμε τα ήδη υπάρχοντα αρχεία του QEMU που θα χρειαστούν τροποποίηση.

4.2.1 Λειτουργική Μονάδα CPM #1: RISC & ROM

Η εργασία της λειτουργικής μονάδας RISC & ROM είναι η διαχείριση των λειτουργιών του CPM με την μέθοδο της εκτέλεσης μίας εντολής σε κάθε κύκλο ρολογιού. Δυστυχώς αυτή η λειτουργία δεν έχει υλοποιηθεί στον QEMU. Υπήρχε φυσικά η εκτέλεση εντολών ανά κύκλο ρολογιού αλλά τα αιτήματα πήγαιναν απευθείας στον Επεξεργαστικό Πυρήνα χωρίς να αναμένουν ή να περνούν από κάποιου είδους διαδικασία. Δηλαδή εμείς θα χρειαστεί να τροποποιήσουμε τον QEMU ώστε ο αποδέκτης πια να είναι ο CPM και αυτός πλέον να είναι η οντότητα που θα αποφασίζει για τις εντολές που θα εκτελεστούν και αφορούν το δικτυακό κομμάτι.

Η εντολή που εκτελείται σε κάθε κύκλο ρολογιού έχει υλοποιηθεί στον πηγαίο κώδικα του QEMU στην συνάρτηση `cpu_exec_all()`.

```
//path: qemu-0.14.0/cpus.c:911 - 933

911  bool cpu_exec_all(void)
912  {
913    if (next_cpu == NULL)
914        next_cpu = first_cpu;
915    for (; next_cpu != NULL && !exit_request; next_cpu = next_cpu->next_cpu) {
916        CPUState *env = next_cpu;
917
918        qemu_clock_enable(vm_clock,
```

```
919                         (env->singlestep_enabled & SSTEP_NOTIMER) == 0);
920
921         if (qemu_alarm_pending())
922             break;
923         if (cpu_can_run(env)) {
924             if (qemu_cpu_exec(env) == EXCP_DEBUG) {
925                 break;
926             }
927         } else if (env->stop) {
928             break;
929         }
930     }
931     exit_request = 0;
932     return any_cpu_has_work();
933 }
```

Γραμμή 915:

Αυτή η `for` δείχνει την εναλλαγή ανάμεσα στις CPUs και την αναμονή τους για έναν κύκλο ρολογιού για να μπορέσουν να εκτελέσουν την επόμενη εντολή.

Γραμμές 921-930:

Εδώ παίρνεται η απόφαση για το αν θα υπάρχει εντολή/αίτημα για να εκτελεστεί από την CPU.

Ο σκοπός μας είναι να εμφυσήσουμε αυτή την λειτουργικότητα στο νέο αρχείο που θα χρειαστεί να υλοποιηθεί να για να μπορούμε να πούμε ότι ο CPM όντως προσομοιώνεται από τον QEMU.

4.2.2 Λειτουργική Μονάδα CPM #2: Χρονιστές (Timers)

Ευτυχώς η κοινότητα των προγραμματιστών του QEMU μας έχει μια ευχάριστη έκπληξη. Έχει υλοποιήσει την οντότητα όπου θα μας λύσει τα χέρια και θα την χρησιμοποιήσουμε για να

αναπαραστήσουμε τους Χρονιστές του CPM ειδικά.

Η οντότητα αυτή είναι ο `QEMUTimer`. Με την αυτή θα μπορούμε να καταχωρήσουμε την μέγιστη συχνότητα λειτουργίας του Ενσωματωμένου Επεξεργαστή Επικοινωνιών

Για να δούμε λίγη από την ανατομία της οντότητας που αναφέραμε, θα εστιάσουμε σε κομμάτια του πηγαίου κώδικα του QEMU.

```
//path: qemu-0.14.0/qemu-timer.c:155 - 167

155 struct QEMUClock {
156     int type;
157     int enabled;
158     /* XXX: add frequency */
159 };
160
161 struct QEMUTimer {
162     QEMUClock *clock;
163     int64_t expire_time;
164     QEMUTimerCB *cb;
165     void *opaque;
166     struct QEMUTimer *next;
167 };
```

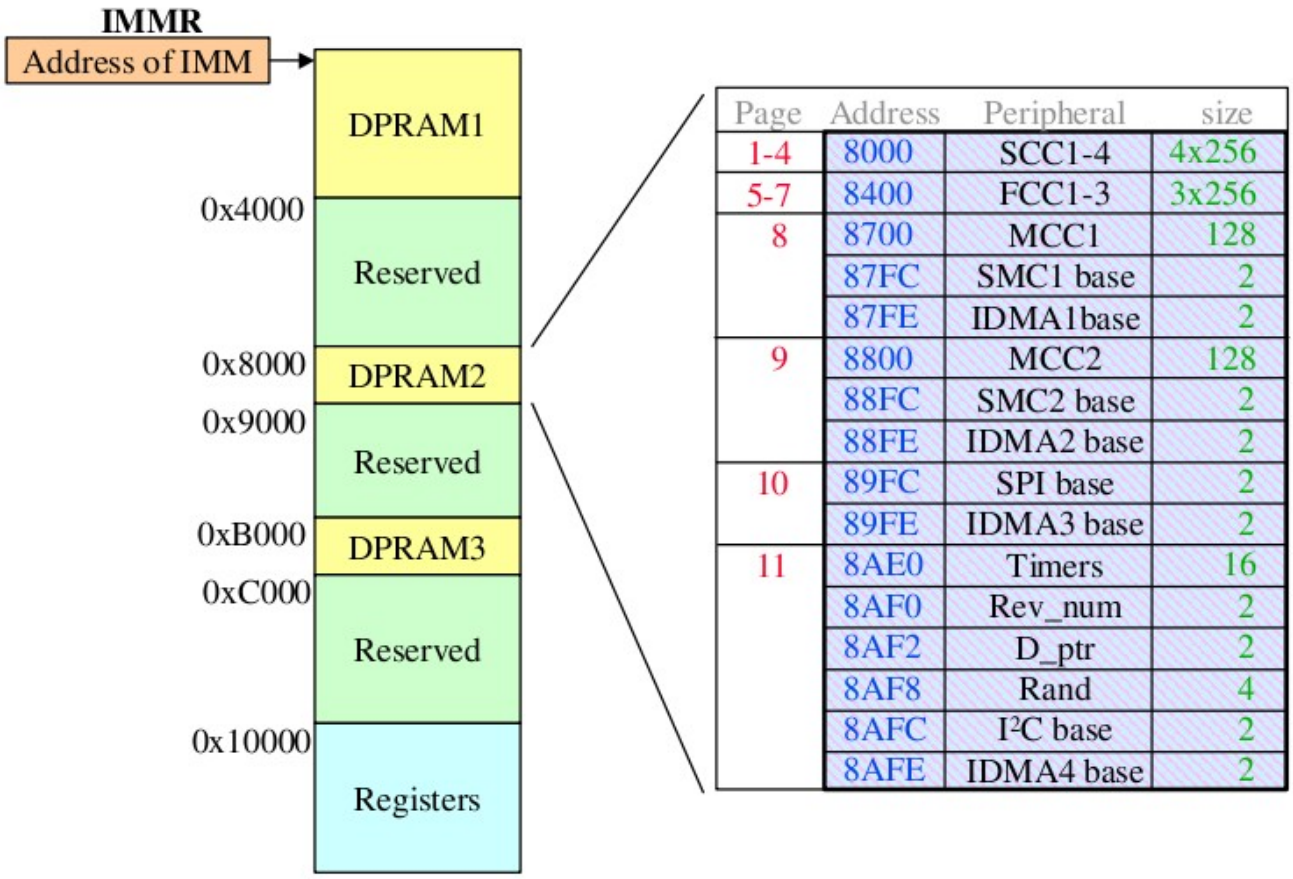
Από τον ορισμό του `QEMUTimer` μπορούμε να καταλάβουμε ότι πρόκειται για μία μονά συνδεδεμένη λίστα όπου ο κάθε κόμβος δείχνει στο επόμενο συγχρονιστικό γεγονός. Επίσης περιέχει και ένα ρολόι όπου μπορούμε να εισάγουμε τη συχνότητα που θέλουμε.

4.2.3 Λειτουργική Μονάδα CPM #3: Εσωτερική Μνήμη (Internal Memory Space)

Η λειτουργική μονάδα της Εσωτερικής Μνήμης του Ενσωματωμένο Συνεπεξεργαστή Επικοινωνιών δεν υπάρχει υλοποιημένη στον πηγαίο κώδικα του QEMU. Θα χρειαστεί να σχεδιαστεί εκ νέου.

Αυτό που χρειαζόμαστε είναι να έχουμε μία πλήρη εικόνα της δομής της μνήμης και φυσικά μία διεύθυνση μνήμης που θα σηματοδοτεί την αφετηρία της. “Την ευθύνη αυτή θα την επωμιστεί ένας καταχωρητής που θα δημιουργήσουμε με το όνομα IMMR. Με βάση αυτή τη διεύθυνση θα γίνεται η διευθυνσιοδότηση όλων των καταχωρητών και παραμέτρων του Ενσωματωμένου Επεξεργαστή Επικοινωνιών. Επίσης θα χρειαστεί να δημιουργήσουμε τρεις(3) περιοχές ταχύτατης στατικής μνήμης οι οποίες θα περιλαμβάνονται μέσα στην Εσωτερική Μνήμη. Είναι οι επονομαζόμενες Dual Port RAM (DPRAM) οι οποίες περιλαμβάνουν παραμέτρους και δομές δεδομένων που χρειάζονται για τη λειτουργία των ελεγκτών του Ενσωματωμένου Επεξεργαστή Επικοινωνιών” [17].

Μία πιθανή δομή θα μπορούσε να είναι η παρακάτω, θέτοντας τρεις(3) FCC.



Page	Address	Peripheral	size
1-4	8000	SCC1-4	4x256
5-7	8400	FCC1-3	3x256
8	8700	MCC1	128
	87FC	SMC1 base	2
	87FE	IDMA1base	2
9	8800	MCC2	128
	88FC	SMC2 base	2
	88FE	IDMA2 base	2
10	89FC	SPI base	2
	89FE	IDMA3 base	2
11	8AE0	Timers	16
	8AF0	Rev_num	2
	8AF2	D_ptr	2
	8AF8	Rand	4
	8AFC	I^2C base	2
	8AFE	IDMA4 base	2

Εικόνα 4.4: Δεικτοδότηση του Χάρτη Εσωτερικής Μνήμης [17 – Μηλιώνης2006]

Οπότε θα χρειαστεί να φτιάξουμε ένα αρχείο `cpm.c` όπου θα περιέχει τον κώδικα που θα προσομοιώνει τον Ενσωματωμένο Συνεπεξεργαστή Επικοινωνιών.

Ένα μέρος αυτού θα είναι η δήλωση του καταχωρητή IMMR που αναφέραμε πριν όπως και η δήλωση της DPRAM ως δομή(`struct`).

```
//path: qemu-0.14.0/cpm.c


enum InternalMemory
{
     IMMR = 0x8000;
};


typedef struct DPRAM2
{
     uint32_t scc[NUMBER_OF_SCC];
     uint32_t fcc[NUMBER_OF_FCC];
     uint16_t mcc1;
     int smc1_base;
     int idma1_base;
     uint16_t mcc2;
     int smc2_base;
     int idma2_base;
     int spi_base;
     int idma3_base;
     uint2_t timers;
     int rev_num;
     int d_ptr;
     int rand;
     int i2c_base;
     int idma4_base;
}DPRAM2;
```

Στη συνέχεια θα χρειαστεί να δηλώσουμε και τις δομές που περιέχει κάθε οντότητα της DPRAM. Πιο κάτω , ειδικά , θα παραμείνουμε στους ελεγκτές επικοινωνιών υψηλού ρυθμού μετάδοσης.

4.2.4 Λειτουργική Μονάδα CPM #4: Γεννήτριες Ρυθμού Μετάδοσης (Baud Rate Generators)

Η γεννήτρια ρυθμού μετάδοσης χρειάζεται για να παρέχει ένα σήμα σε κάποια συχνότητα όπου χρησιμοποιείται για να ελέγχει τον χρονισμό των σειριακών διεπαφών. Εφόσον υπάρχουν διαφορετικές ταχύτητες γραμμών τότε απαιτείται και διαφορετικός χρονισμός. Άρα ο ρυθμός που παράγεται χρειάζεται ευελιξία.

Στην περίπτωσή μας είμαστε τυχεροί διότι είμαστε σε επίπεδο λογισμικού και έτσι μπορούμε να διαχειριστούμε το πόσο και αν θα είναι ευέλικτος ο ρυθμός μετάδοσης.

Παρακάτω επιδεικνύουμε ένα κομμάτι κώδικα από τον πηγαίο κώδικα του QEMU όπου θα μπορούσαμε να δανειστούμε ιδέες για την υλοποίηση στον CPM.

```
//path: qemu-0.14.0/hw/serial.c:748 - 753


748 /* Change the main reference oscillator frequency. */
749 void serial_set_frequency(SerialState *s, uint32_t frequency)
750 {
751     s->baudbase = frequency;
752     serial_update_parameters(s);
753 }
```

Βλέπουμε ότι μπορούμε να προσαρμόσουμε την συχνότητα ανάλογα με τις απαιτήσεις αν το επιθυμούμε όπως ακριβώς ορίζεται και από κάποιες μεθόδους χρήσης της Γεννήτριας Ρυθμού Μετάδοσης.

4.2.5 Λειτουργική Μονάδα CPM #5: Ελεγκτής Αιτήσεων Διακοπής (Interrupt Controller)

Πριν την υλοποίηση του CPM όταν συνέβαινε ένα I/O γεγονός (I/O event) ο Επεξεργαστικός Πυρήνας χρησιμοποιούσε έναν αλγόριθμο χρονοπρογραμματισμού (scheduling algorithm) ο οποίος λειτουργούσε με τη μέθοδο round-robin. Ο στόχος είναι να μεταφέρουμε αυτή τη λειτουργικότητα και στον CPM ώστε να μπορεί να επωμιστεί αυτός το φορτίο μερικών γεγονότων που αφορούν τις δικτυακές ανάγκες του συστήματός μας και έτσι ο Επεξεργαστικός Πυρήνας να μη διακόπτεται ή να περιμένει χωρίς λόγο.

Μέρος του αλγόριθμου παρουσιάζεται και επεξηγείται παρακάτω έχοντας υλοποιηθεί στον κώδικα μιας συνάρτησης του πηγαίου κώδικα του QEMU, την `main_loop_wait()` :

```
//path: qemu-0.14.0/vl.c:1314 - 1389

1314 void main_loop_wait(int nonblocking)
1315 {
1316     IOHandlerRecord *ioh;
1317     fd_set rfds, wfds, xfds;
1318     int ret, nfds;
1319     struct timeval tv;
1320     int timeout;
1321
1322     if (nonblocking)
1323         timeout = 0;
1324     else {
1325         timeout = qemu_calculate_timeout();
1326         qemu_bh_update_timeout(&timeout);
1327     }
1328
1329     os_host_main_loop_wait(&timeout);
1330
1331     /* poll any events */
```

```
1332    /* XXX: separate device handlers from system ones */
1333    nfds = -1;
1334    FD_ZERO(&rfds);
1335    FD_ZERO(&wfds);
1336    FD_ZERO(&xfds);
1337    QLIST_FOREACH(ioh, &io_handlers, next) {
1338        if (ioh->deleted)
1339            continue;
1340        if (ioh->fd_read &&
1341            (!ioh->fd_read_poll ||
1342             ioh->fd_read_poll(ioh->opaque) != 0)) {
1343            FD_SET(ioh->fd, &rfds);
1344            if (ioh->fd > nfds)
1345                nfds = ioh->fd;
1346        }
1347        if (ioh->fd_write) {
1348            FD_SET(ioh->fd, &wfds);
1349            if (ioh->fd > nfds)
1350                nfds = ioh->fd;
1351        }
1352    }
1353
1354    tv.tv_sec = timeout / 1000;
1355    tv.tv_usec = (timeout % 1000) * 1000;
1356
1357    slirp_select_fill(&nfds, &rfds, &wfds, &xfds);
1358
1359    qemu_mutex_unlock_iothread();
1360    ret = select(nfds + 1, &rfds, &wfds, &xfds, &tv);
1361    qemu_mutex_lock_iothread();
1362    if (ret > 0) {
```

```
1363        IOHandlerRecord *pioh;
1364
1365        QLIST_FOREACH_SAFE(ioh, &io_handlers, next, pioh) {
1366            if (!ioh->deleted && ioh->fd_read && FD_ISSET(ioh->fd, &rfds)) {
1367                ioh->fd_read(ioh->opaque);
1368            }
1369            if (!ioh->deleted && ioh->fd_write && FD_ISSET(ioh->fd, &wfds)) {
1370                ioh->fd_write(ioh->opaque);
1371            }
1372
1373            /*Do this last in case read/write handlers marked it for
deletion*/
1374            if (ioh->deleted) {
1375                QLIST_REMOVE(ioh, next);
1376                qemu_free(ioh);
1377            }
1378        }
1379    }
1380
1381    slirp_select_poll(&rfds, &wfds, &xfds, (ret < 0));
1382
1383    qemu_run_all_timers();
1384
1385    /* Check bottom-halves last in case any of the earlier events triggered
         them.  */
1386    qemu_bh_poll();
1387
1388 }
```

Γραμμή 1325, 1326:

Θέτει ένα λογικό χρονικό περιθώριο για να καλεστεί ο handler.

Γραμμή 1337-1352:

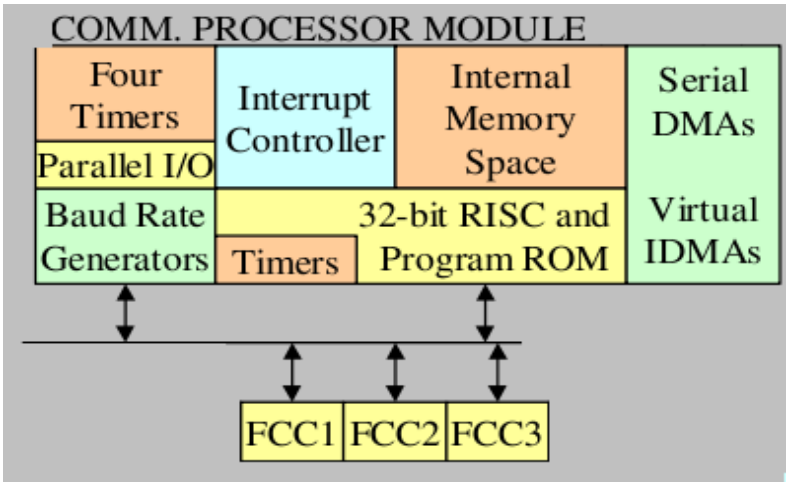
Χρησιμοποιείται μία συνάρτηση που υλοποιεί τον αλγόριθμο round robin όπως προείπαμε. Δέχεται τον event και καθορίζει από μιά πλειάδα αιτημάτων ποιο θα είναι το επόμενο.

Αυτή τη φορά δεν είμαστε τόσο τυχεροί ώστε να έχει υλοποιηθεί από την κοινότητα του QEMU ένας αλγόριθμος για την διευθέτηση των προτεραιοτήτων των Αιτημάτων Διακοπής. Πριν τα Αιτήματα τα αναλάμβανε το λειτουργικό σύστημα (στην περίπτωσή μας το Linux). Η ευτυχής συγκυρία είναι ότι μπορούμε να δανειστούμε ιδέες από τον τρόπο λειτουργίας του λειτουργικού συστήματος Linux και να το εμφωλεύσουμε μέσα στην λειτουργικότητα του CPM.

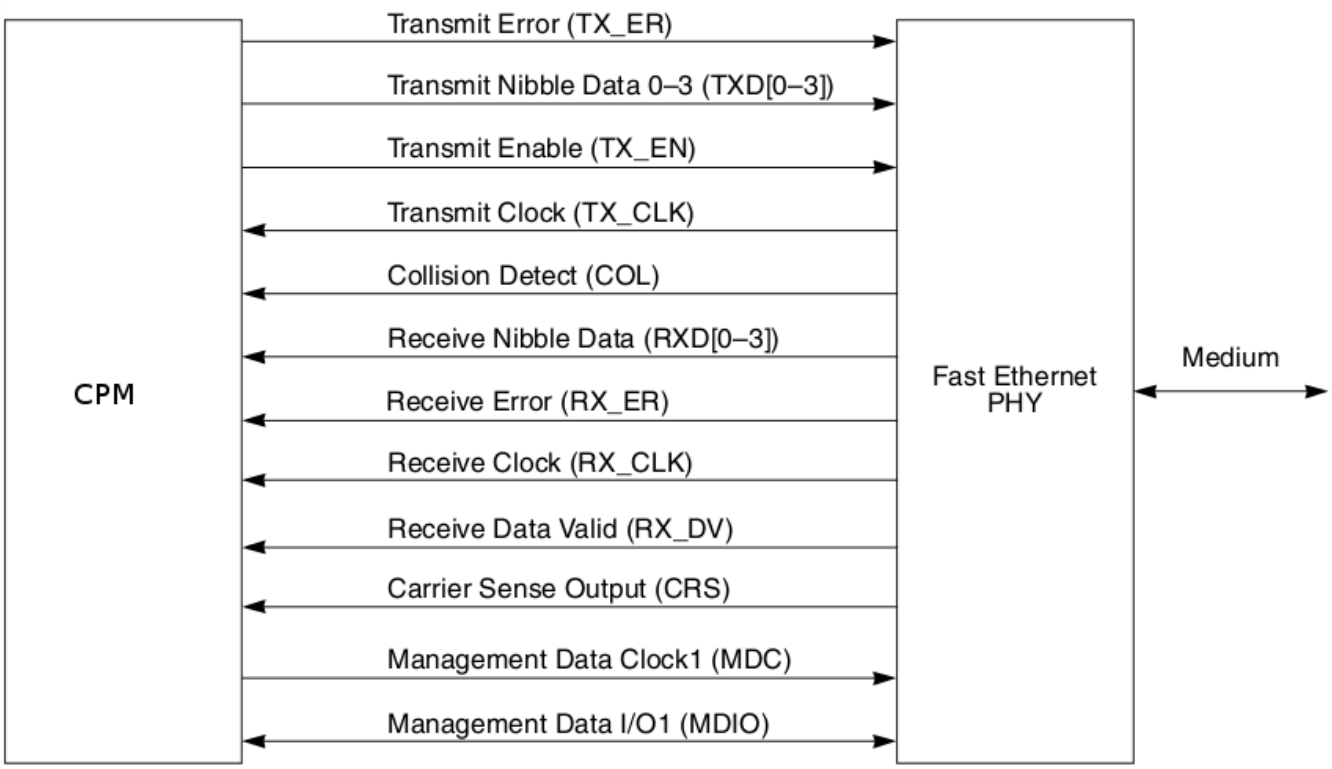
4.2.6 Λειτουργική Μονάδα CPM #6: Ελεγκτές Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης (Fast Communication Controller)

Στο τελευταίο σκέλος της ανάλυσης για μία μελλοντική προσομοίωση του QEMU θα θέλαμε να αναπτύξουμε έναν σχεδιασμό για να μπορέσουν κάποιοι προγραμματιστές , που θέλουν να συνεισφέρουν στην κοινότητα το QEMU , να το εκμεταλλευτούν.

Οι Ελεγκτές Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης (FCC) υποστηρίζουν συγχρονισμένα πρωτόκολλα υψηλού ρυθμού μετάδοσης, όπως HDLC, Ethernet και ATM. Οπότε πρόκειται για ένα από τα πιο σημαντικά στοιχεία για την προσπάθεια υποστήριξης Δικτυακών Ενσωματωμένων Συστημάτων. Συνδέονται και εξυπηρετούνται με τον Ενσωματωμένο Συνεπεξεργαστή Επικοινωνιών όπως φαίνεται στις παρακάτω εικόνες:



Εικόνα 4.5: Σύνδεση CPM και FCC σε Block Diagram επίπεδο [21]



Εικόνα 4.6: Σύνδεση CPM και FCC [21]

Οι Ελεγκτές Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης(FCC) είναι ανεξάρτητοι από τη διεπαφή φυσικού επιπέδου, αλλά η λογική των FCC πλαισιώνει και χειραγωγεί τα δεδομένα που δέχεται από τη διεπαφή φυσικού επιπέδου. Οι FCC περιγράφονται υπό όρους του πρωτοκόλλου που έχει επιλεχθεί να εκτελεστεί. Όταν ένας FCC είναι προγραμματισμένος και συσχετισμένος με ένα συγκεκριμένο πρωτόκολλο, τότε υλοποιεί ταυτόχρονα μία λειτουργικότητα αυτού του πρωτοκόλλου. Για τα περισσότερα πρωτόκολλα αυτό αφορά το δεύτερο επίπεδο του OSI μοντέλου(Link Layer). Εμείς θα εστιάσουμε στο Ethernet πρωτόκολλο.

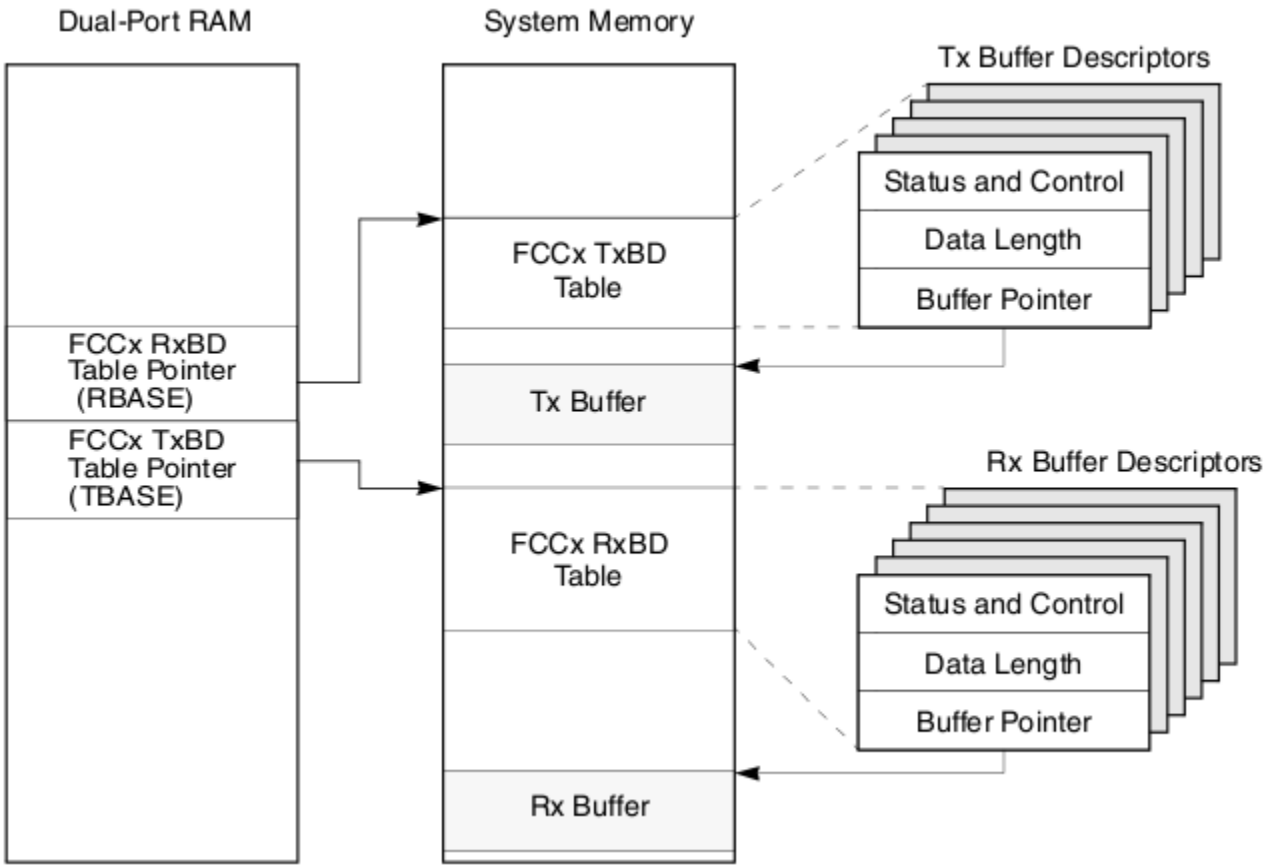
4.2.6.1 Περιγραφή και μεταφορά του Fast Communication Controller στον QEMU

Για να μπορέσει να λειτουργήσει το πρόγραμμα προσομοίωσης θα χρειαστεί να αναπαρασταθεί η

δομή του FCC υπό προγραμματιστικούς όρους. Δηλαδή, χρειάζεται να υλοποιηθούν συστατικά όπως:

- Γενικού Τύπου Καταχωρητές Ελεγκτών Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης - General FCC Mode Registers(GFMRx)
- Πρωτοκολλο-στρεφείς Καταχωρητές Ελεγκτών Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης - FCC Protocol-Specific Mode Registers(FPSMRx)
- Καταχωρητές Συγχρονισμού Δεδομένων Ελεγκτών Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης - FCC Data Synchronization Registers(FDSRx)
- Μεταφοράς υπό Απαίτηση Καταχωρητές Ελεγκτών Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης - FCC Transmit-on-Demand Registers(FTODRx)
- Ενταμιευτές Ελεγκτών Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης - FCC Buffer Descriptors
- Μνήμη RAM Ελεγκτών Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης - FCC Parameter RAM
- Καταχωρητές Συναρτησιακού Κώδικα Ελεγκτών Επικοινωνιών Υψηλού Ρυθμού Μετάδοσης - FCC Function Code Regiters(FCRx)
- Interrupts from the FCCs
- FCC Event Registers(FCCEx)
- FCC Mask Registers(FCCMx)
- FCC Status Registers(FCCSx)
- FCC Intialization
- FCC Interrupt Handling
- FCC Transmit Errors
- FCC Timing Control

Για τα συστατικά(components) εως και το FCC Status Registers(FCCSx) μπορούν να υλοποιηθούν σχετικά εύκολα. Αυτό που χρειάζεται να κάνουμε είναι να πάμε στο Manual του PowerQUICC-II και να περάσουμε τους πίνακες που βλέπουμε σε ένα νέο αρχείο `fcc_powerquicc2.c` . Για να έχουμε μια πιο γραφική αντίληψη πως δομούνται τα συστατικά του FCC ας δούμε το παρακάτω σχήμα δανειζόμενο από το [21].



Εικόνα 4.7: Δομή της Μνήμης των FCC [21]

Καταχωρητής	Μέγεθος
GFMRx	32-bit
FPSMRx	32-bit
FDSRx	16-bit
FTODR	16-bit
Buffer Discriptors	16-bit, 16-bit, 32-bit, 32-bit
RIPTR	16-bit
TIPTR	16-bit

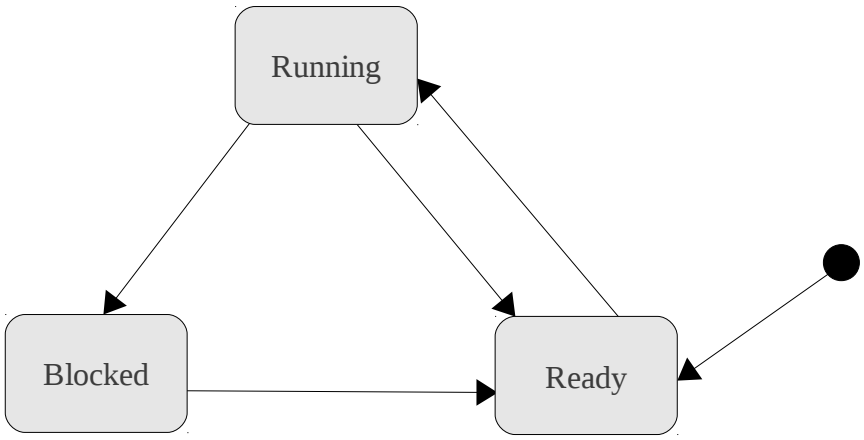
-	16-bit
MRBLR	16-bit
RSTATE	32-bit
RBASE	32-bit
RBDSTAT	16-bit
RBDLLEN	16-bit
RDPTR	32-bit
TSTATE	32-bit
TBASE	32-bit
TBDSTAT	16-bit
TBDLEN	16-bit
TDPTR	32-bit
RBPTR	32-bit
TBPTR	32-bit
RCRC	32-bit
-	32-bit
TCRC	32-bit
-	32-bit
FCRx	8-bit
FCCEx	
FCCMx	8-bit
FCCSx	8-bit

Χρειάζεται να δοθεί προσοχή στην στοίχιση-αρίθμηση των καταχωρητών για να μπορέσει να γίνει σωστά η χαρτογράφηση των διευθύνσεων.

4.2.6.2 Περιγραφή και προτάσεις υλοποίησης του Ελεγκτή Ethernet Υψηλού Ρυθμού Μετάδοσης (FEC - Fast Ethernet Controller) στον QEMU

Για να μπορέσουμε να αναλύσουμε και να μεταφέρουμε τον Fast Ethernet Controller στον QEMU θα χρησιμοποιήσουμε ένα Αυτόματο Πεπερασμένων Καταστάσεων (FSM – Finite State Machine). Εφόσον ο CPM είναι συνδεδεμένος με τον Fast Ethernet Controller οι εναλλαγές των καταστάσεων θα αναφέρονται σε αυτές τις οντότητες.

Παρακάτω φαίνεται το σχεδιάγραμμα του Αυτόματου Πεπερασμένων Καταστάσεων (FSM) όπου αναπαριστάται μέσω του UML Διάγραμμα Ακολουθίας (UML Sequence Diagram).



Εικόνα 4.8: Αυτόματο Πεπερασμένων Καταστάσεων του Ελεγκτή Ethernet Υψηλού Ρυθμού Μετάδοσης

Ορισμοί καταστάσεων:

Running – Εκτελείται αυτή τη στιγμή

Ready – Δεν εκτελείται αλλά θέλει να εκτελεστεί να μεταβεί στην κατάσταση **Running**

Blocked – Αποκοπή από τα δικαιώματα εκτέλεσης, για να εκτελεστεί θα χρειαστεί πρώτα να μεταβεί στην **Ready** κατάσταση και αν επιλεχθεί τότε θα μεταβεί στην **Running**.

Επεξήγηση Μεταβάσεων Κατάστασης:

Ready -> Running:

- Όταν ο χρήστης έχει έτοιμο προς μετάδοση ένα frame, τότε ο FEC αρχίζει να παίρνει τα δεδομένα από τον ενταμιευτή δεδομένων και τροποποιεί με TX_EN . Οπότε βλέπουμε ότι ο FEC ήταν έτοιμος αλλά δεν εκτελεστεί αλλά δεν εκτελούνταν, οπότε η μετάβαση αυτή των καταστάσεων ήταν αναγκαία για να ξεκινήσει να λειτουργεί.

Blocked -> Ready:

- Όταν αναφερόμαστε σε half-duplex λειτουργία, ο FEC σταματά την μετάδοση εάν η σύνδεση είναι απασχολημένη. Πριν την μετάδοση, ο FEC περιμένει έως ότου αντιληφθεί ότι μπορεί να μεταδώσει (Carrier Sense). Όταν αναφέρουμε ότι σταματά την μετάδοση εννοούμε ότι βρίσκεται στην κατάσταση **Blocked** όπου πια δεν λειτουργεί και περιμένει να βρεθεί στην κατάσταση **Ready** για να μπορέσει να μπει στην αναμονή και κάποια στιγμή να επιλεχθεί να λειτουργήσει. Αυτό το καταφέρνει αφού “ακούσει” ότι κανένας άλλος δεν μεταδίδει.

Running -> Ready:

- [**1η περίπτωση**] Εάν συμβεί μία σύγκρουση(collision) κατά τη διάρκεια της μετάδοσης του frame, ο FEC ακολουθεί μια συγκεκριμένη διαδικασία οπισθοχώρησης και προσπαθεί να επαναμεταδόσει το frame μέχρι να φτάσει το όριο των επαναμεταδόσεων. Έχουμε δηλαδή μία αλλαγή κατάστασης από την εκτέλεση όπου βρισκόταν καθώς μετέδιδε στην Ready κατάσταση που σηματοδοτείται από το οπισθοχώρηση που τον καθιστά υποψήφιο για μετάδοση αλλά όχι και τρέχον.
- [**2η περίπτωση**] Όταν ένα frame ληφθεί έχοντας ένα CRC λάθος, τότε ο αποδέκτης μπαίνει σε λειτουργία Hunt. Ο έλεγχος για CRC λάθη δε μπορεί να αποσυνδεθεί, αλλά το CRC λάθος μπορεί να αγνοηθεί εάν ο έλεγχος είναι προαιρετικός. Το ότι μπορεί να αγνοηθεί το CRC λάθος είναι ο λόγος που δεν πηγαίνουμε στην **Blocked** κατάσταση αλλά στην **Ready**.

Running -> Blocked:

- [**1η περίπτωση**] Εάν μια διεύθυνση χρειαστεί να διαγραφεί από τον Hash Table, τότε ο FEC χρειάζεται να αποσυνδεθεί, οι καταχωρητές του Hash Table χρειάζεται να εκκαθαριστούν και η εντολή SET GROUP ADDRESS θα εκτελεστεί για τις υπόλοιπες επιλεγμένες διευθύνσεις. Η αποσύνδεση είναι υπεύθυνη για τη μετάβαση στην **Blocked** κατάσταση.

- [**2η περίπτωση**] Όταν μια σύγκρουση συμβεί κατά τη διάρκεια της ακολουθίας του αρχικού μέρους του frame. Η σύγκρουση αναγκάζει την άμεση μετάβαση στην **Blocked** κατάσταση.
- [**3η περίπτωση**] Όταν μία σύγκρουση συμβεί μέσα σε 64 bytes φορές, η διαδικασία επαναλαμβάνεται και επανεκπέμπεται. Ο αποστολέας περιμένει για ένα τυχαίο αριθμό χρονικών διαστημάτων(κάθε χρονικό διάστημα είναι 512 bit φορές). Το ίδιο αποτέλεσμα με την [1η περίπτωση].
- [**4η περίπτωση**] Όταν μία σύγκρουση συμβεί κατά τη διάρκεια της λήψης του frame, τότε η λήψη σταματά.

Blocked -> Ready -> Running:

- [**1η περίπτωση**] Όταν ο FEC στέλνει 32 bits ώστε να επαληθεύσει ένα CRC λάθος, σταματά τον ενταμιευτή αποστολών, κλείνει τον ενταμιευτή στη συνέχεια και στο τέλος τροποποιεί τον TxBD[UN] και FCCE[TXE]. Ο FCE επανέρχεται στη διαδικασία της αποστολής ύστερα από την λήψη της εντολής RESTART TRANSMIT.
- [**2η περίπτωση**] Ο FCE τερματίζει τον ενταμιευτή μεταδόσεων, κλείνει τον ενταμιευτή και τέλος θέτει τους TxBD[RL] και FCCE[TXE]. Η μετάδοση επανέρχεται ύστερα από την εκτέλεση της εντολής RESTART TRANSMIT.
- [**3η περίπτωση**] Ο FCE τερματίζει τον ενταμιευτή μεταδόσεων, κλείνει τον ενταμιευτή και τέλος θέτει τους TxBD[LC] και FCCE[TXE]. Η μετάδοση επανέρχεται ύστερα από την εκτέλεση της εντολής RESTART TRANSMIT.

Running -> Blocked -> Ready -> Running:

- [**1η περίπτωση**] Όταν εκτελεστεί η εντολή GRACEFUL STOP TRANSMIT, ο FCE σταματά αμέσως. Αν δεν υπάρχει καμία μετάδοση σε λειτουργία τότε, είτε συνεχίζει την μετάδοση έως ότου να τελιώσει η μετάδοση του τρέχοντος frame είτε η τερματίζει με σύγκρουση(collision). Όταν στον FEC δοθεί η εντολή RESTART TRANSMIT, τότε η μετάδοση επαναλαμβάνεται.
- [**2η περίπτωση**] Όταν η flow-control λειτουργία είναι ενεργή και ο παραλήπτης ανιχνεύει μία pause-flow control frame η οποία εστάλη σε ανεξάρτητες ή προς όλες τις διευθύνσεις. Οι μεταδόσεις σταματούν για συγκεκριμένο χρονικό διάστημα για το ελεγχόμενο frame. Κατά τη διάρκεια αυτής της παύσης, μόνο το ένα frame εκτός ακολουθίας στέλνεται. Υπό κανονικές συνθήκες η μετάδοση επανέρχεται ύστερα από την παύση και ο μετρητής σταματά να μετράει.

Συμπεράσματα

Έχοντας δει τις ικανότητες του QEMU και πως βοηθά στην ανάπτυξη των Ενσωματωμένων Συστημάτων η παρούσα εργασία μπορεί να αποτελέσει την έναρξη της μελέτης των εσωτερικών μηχανισμών από μελετητές οι οποίοι θα ήθελαν να αναμειχθούν στην ανάπτυξη του QEMU ή έστω και να τον κατανοήσουν βλέποντας τη διασύνδεση ανάμεσα στον πηγαίο κώδικα, τα εγχειρίδια δικτυακών συσκευών και τη θεωρία των Δικτυακών Ενσωματωμένων Συστημάτων. Επιπρόσθετα η εργασία είναι μία πυξίδα για την μελλοντική υλοποίηση της υποστήριξης επεξεργαστών επικοινωνιών PowerQUICC-II όπου θα είναι ένα σημαντικό βήμα στην ανάπτυξη του QEMU και κατά συνέπεια μία βοήθεια στην ανάπτυξη των Ενσωματωμένων Συστημάτων καθολικά.

Βιβλιογραφία

Α – Ξενόγλωσση

[1]. Michael Barr (1999), “Programming Embedded Systems in C and C++”, O'Reilly Media

[2]. Michael Barr, Anthony Massa (2006), “Programming Embedded Systems: With C and GNU Development Tools”, O'Reilly Media, 2nd Edition

[3]. Mel Gorman (2004), “Understading the Linux Virtual Memory Manager”, Prentice-Hall

[4]. C. Hallinan (2010), "Embedded Linux Primer: A Practical Real-World Approach", Prentice-Hall, 2nd Edition

[5]. Graig Hollabaugh (2002), “Embedded Linux: Hardware, Software and Interfacing”, Addison Wesley

[6]. Daniel W. Lewis (2001), “Fundamentals of Embedded Software Where C and Assembly Meet”, Prentice-Hall

[7]. Tammy Noergaard (2005), “Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers”, Elsevier

[8]. Gene Sally (2009), “Pro Linux Embedded Systems”, Apress

[9]. David E. Simon (1999), “An Embedded Software Primer”, Addison-Wesley Professional

[10]. Andrew S. Tanenbaum (2006), “Operating Systems Design and Implementation”, Prentice-Hall, 3rd Edition

[11]. Andrew S. Tanenbaum (2007), “Modern Operating Systems”, Prentice-Hall, 2nd Edition

[12]. Tim Wilmshrst (2009), “Designing Embedded Systems with PIC Microcontrollers: Principles and Applications”, Newnes, 2nd Edition

[13]. Wayne Wolf (2008), “Computers as Components: Principles of Embedded Computing System Design”, Morgan Kaufmann Publishers, 2nd Edition

[14]. Karim Yaghmour, Jon Masters, Gilad Ben-Yossef, Philippe Gerum (2008), “Building Embedded Linux Systems: Concepts, Techniques, Tricks and Traps”, O'Reilly Media, 2nd Edition

Β – Ελληνόγλωσση

[15]. Andrew S. Tanenbaum (2007), “Μοντέρνα Λειτουργικά Συστήματα”, Κλειδάριθμος, 2η Έκδοση

[16]. Wayne Wolf (2008), “Οι Υπολογιστές ως Συστατικά Στοιχεία: Αρχές Σχεδίασης Ενσωματωμένων Υπολογιστικών Συστημάτων”, ΕΚΔΟΣΕΙΣ ΝΕΩΝ ΤΕΧΝΟΛΟΓΙΩΝ ΜΟΝ. ΕΠΕ, μεταφρασμένο

[17]. Απόστολος Μηλιώνης (2006), “Ενσωματωμένα Συστήματα: Πανεπιστημιακές Σημειώσεις”

Γ – Δημοσιεύσεις

[18]. Fabrice Bellard (2005), “QEMU, a Fast and Portable Dynamic Translator”

[19]. Nathaniel Wesley Filardo (2007), “Porting QEMU to Plan 9: QEMU Internals and Port Strategy”, September 11

[20]. Pradyumma Sampath, Rachana Rao (2010), “Efficient embedded software development using QEMU”

Δ – Εγχειρίδια

[21]. Freescale™ Semiconductor (2005), “MPC8260 PowerQUICC™ II Family Reference Manual”, Rev. 2, December 12

[22]. Realtek Semiconductor Corp. (2005), “RTL8139C 3.3V Single-chip Fast Ethernet Controller with Power Managment Datasheet”, Rev. 1.6, December 29

Ε - Ηλεκτρονικές Πηγές

[23]. Documentatin/Networking - http://wiki.qemu.org/Documentation/Networking (14-03-2011)

[24]. QEMU SystemC - http://www.greensocs.com/projects/QEMUSystemC (15-03-2011)

[25]. REALTEK 8139c Datasheet - http://download2.dvd-driver.cz/realtek/datasheets/pdf/rtl8139c_datasheet_1.6.pdf (10-03-2011)

[26]. tantap interface documentation - http://www.kernel.org/doc/Documentation/networking/tuntap.txt (2-04-2011)

[27]. IRC > Server: OFTC > Channel: #qemu

[28]. The official forum for QEMU questions and learning - http://qemu-forum.ipi.fi/ (29-03-2011)