ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ενσωμάτωση Linux σε οικιακή πύλη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ιωάννη Χ. Λιβέρεζα

Επιβλέπων: Γ. Στασινόπουλος
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2005

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ενσωμάτωση Linux σε οικιακή πύλη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ιωάννη Χ. Λιβέρεζα

Επιβλέπων: Γ. Στασινόπουλος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Ιουνίου 2005

.....	.....	.....
Γ. Στασινόπουλος Καθηγητής Ε.Μ.Π	Μ. Θεολόγου Καθηγητής Ε.Μ.Π.	Ε. Συκάς Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2005

.....

Ιωάννης Χ. Λιβέρεζας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών
Ε.Μ.Π.

Περίληψη

Η διπλωματική αυτή εργασία πραγματεύεται τη χρήση του λειτουργικού συστήματος GNU/Linux σε σύστημα εντός ολοκληρωμένου κυκλώματος (System on a Chip, SoC), που προορίζεται για οικιακή πύλη. Περιγράφει την παρούσα κατάσταση στο χώρο των οικιακών πυλών και αναπτύσσει μεθοδολογία για τον έλεγχο ενσωματωμένων συστημάτων. Ο έλεγχος αυτός περιλαμβάνει τόσο το στάδιο σχεδιασμού του SoC όσο και την λειτουργία του μετά την κατασκευή του.

Συγκεκριμένα περιγράφει τα στάδια που χρειάστηκαν για τη μεταφορά του Linux στο πρωτότυπο SoC **REGATE** της εταιρίας InAccess NetWorks. Περιλαμβάνει τον έλεγχο των περιφερειακών συσκευών που περιλαμβάνει το SoC , τη μεταφορά του πυρήνα του Linux και την υλοποίηση των οδηγών συσκευών.

Με τις μετρήσεις που έγιναν, κυρίως ως προς τις δικτυακές επιδόσεις του συστήματος, συμπεραίνουμε ότι το REGATE μπορεί να αποτελέσει ένα επαρκές μοντέλο για το σχεδιασμό οικιακών πυλών.

Λέξεις Κλειδιά

Οικιακή Πύλη, Σύστημα Εντός Ολοκληρωμένου Κυκλώματος, Ενσωματωμένο Σύστημα, Linux, Οδηγός Συσκευής, REGATE, LEON, SPARC

Abstract

This thesis deals with the use of the GNU linux operating system in System on a Chip (SoC)platforms, designed to serve the needs of residential gateways. It describes the current development of residential gateways and develops a methodology for testing embedded systems. This testing takes place both in designing stage of the system and the function of the constructed chip.

Specifically, it describes the steps taken to port Linux to the prototype SoC of the company InAcces Networks, called REGATE. It contains the testing of the included in the chip peripherals, the Linux porting and the creation of linux device drivers for those peripherals.

The network performance of the system suggests that REGATE may be the basis on the design model of residential gateways.

Keywords

Residential Gateway, System on a Chip, SoC, Embedded System, Linux, Device Driver, REGATE, LEON, SPARC

Ευχαριστίες

Ευχαριστώ βαθύτατα όλους αυτούς που με τον ένα ή τον άλλον τρόπο συνέβαλαν στην εκπόνηση αυτής της διπλωματικής εργασίας. Ευχαριστώ ιδιαίτερα:
Την οικογένεια μου για τη στήριξη που μου έδωσε όλο αυτόν τον καιρό

Τον Γιώργο Κωνσταντουλάκη για την εμπιστοσύνη που μου έδειξε σ΄ αυτό το δύσκολο έργο και την στήριξη στον επαγγελματικό χώρο.

Τους συναδέλφους στην InAccess Networks για την αδιάκοπη και άμεση υποστήριξη σε επίπεδο μικροηλεκτρονικής και hardware . Ευχαριστώ ιδιαίτερα τους Γιώργο Λυκάκη για όλη την βοήθεια, υποστήριξη και καθοδήγηση που μου παρείχε.

Τον επιβλέποντα καθηγητή κ. Στασινόπουλο και τα μέλη της κριτικής επιτροπής για την εμπιστοσύνη τους και την ευκαιρία που μου έδωσαν να εκπονήσω τη διπλωματική αυτή εργασία

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Οικιακές Πύλες (Residential Gateways)

1.1 Οικιακή δικτύωση

Τα τελευταία χρόνια παρατηρείται εκρηκτική αύξηση της χρήσης ηλεκτρονικών συσκευών στα οικιακά περιβάλλοντα. Στις Ηνωμένες Πολιτείες για παράδειγμα, από το 2001, περίπου το 90% των οικιακών υπολογιστών έχουν πρόσβαση στο διαδίκτυο. Παράλληλα, η χρήση τεχνολογιών ευρείας ζώνης στο κομμάτι πρόσβασης δεκαπλασιάστηκε μέσα σε πέντε χρόνια. Η ύπαρξη περισσότερων του ενός υπολογιστών, οι οποίοι θα έχουν πρόσβαση στο διαδίκτυο και θα μπορούν να μοιράζονται κοινές συσκευές όπως εκτυπωτές scanners κλπ, δημιουργεί την ανάγκη για ανάπτυξη των οικιακών δικτύων. Η ελληνική πραγματικότητα διαφέρει πολύ από την παραπάνω εικόνα, ωστόσο αναμένεται ότι στα επόμενα χρόνια θα αυξηθεί ραγδαία ο αριθμός των χρηστών ευρυζωνικών υπηρεσιών.

Τα δίκτυα προέκυψαν από την ανάγκη εξυπηρέτησης εφαρμογών δεδομένων. Η εμφάνισή τους έγινε αρχικά στον επαγγελματικό χώρο, και στη συνέχεια, με την τεχνολογική εξέλιξη και την εισχώρηση των πολυμέσων στην καθημερινότητα, επεκτάθηκαν και στον οικιακό χώρο, είτε για την εξυπηρέτηση των αναγκών γραφείου στο στο σπίτι (home office) είτε καθαρά για εφαρμογές πολυμέσων.

Έτσι, από δίκτυα δεδομένων επεκτείνονται ήδη σε εφαρμογές φωνής, ήχου και εικόνας για τους εξής λόγους:

- Η χρήση τους δε φτάνει το 100% των δυνατοτήτων τους και έτσι υπάρχει πλεόνασμα στο εύρος ζώνης
- Η ανάπτυξη της ευρυζωνικής δικτυακής πρόσβασης επιτρέπει τη μεταφορά εικόνας και ήχου, τη χρήση εφαρμογών πολυμέσων και τη τηλεφωνία μέσω του διαδικτύου

- Η ανάπτυξη νέων οικιακών τεχνολογιών και συσκευών όπως ψηφιακή τηλεόραση και το Set Top Box

Για την υλοποίηση των οικιακών δικτύων χρησιμοποιούνται τρεις βασικοί τύποι οικιακής δικτύωσης. Ο διαχωρισμός του γίνεται με βάση το φυσικό μέσο που χρησιμοποιούν. Αυτοί οι τύποι είναι οι:

- Οικιακή δικτύωση χωρίς νέα καλώδια
- Οικιακή δικτύωση με νέα καλώδια
- Ασύρματη οικιακή δικτύωση

1.1.1 Οικιακή δικτύωση "χωρίς νέα καλώδια"

Μέχρι πρόσφατα, η ποικιλομορφία των δικτύων που συναντούσε κανείς στον οικιακό χώρο απαιτούσε για τη διασύνδεση τους τη χρήση μεγάλου όγκου εξωτερικής καλωδίωσης. Η λύση αυτή, πέρα από το αυξημένο κόστος, προκαλεί αρκετά προβλήματα λειτουργικότητας και αισθητικής, ιδίως σε περιπτώσεις όπου είναι επιθυμητή η διασύνδεση συσκευών που βρίσκονται σε διαφορετικούς χώρους. Τα τελευταία χρόνια, χάρη στις νέες τεχνολογίες, οι χρήστες μπορούν να αποφύγουν την εγκατάσταση νέων καλωδιώσεων και να χρησιμοποιούν τα υπάρχοντα τηλεφωνικά δίκτυα, τα δίκτυα ηλεκτρικής ισχύος και την ασύρματη μετάδοση για την διασύνδεση υπολογιστών ή άλλων ψηφιακών συσκευών. Τα σημερινά δίκτυα δεδομένων είναι ικανά να μεταφέρουν δεδομένα σε πολύ υψηλές ταχύτητες. Τα δίκτυα αυτά όμως χρησιμοποιούν συνήθως οπτικές ίνες, συνεστραμμένο καλώδιο ή ομοαξονικό καλώδιο έτσι ώστε να μειωθούν οι εξωτερικές παρεμβολές. Επειδή τα περισσότερα σπίτια δεν έχουν την κατάλληλη υποδομή σε καλωδίωση για τις τεχνολογίες αυτές, το κόστος εγκατάστασης τέτοιας υποδομής είναι συνήθως υψηλό. Για το λόγο αυτό, οι εταιρείες που αναπτύσσουν προϊόντα στον χώρο, διαμορφώνουν τα προιόντα και τις υπηρεσίες του λαμβάνοντας υπ όψιν ότι:

- Τα καινούρια προϊόντα πρέπει να μπορούν να εκμεταλλεύονται τις υπάρχουσες υποδομές
- πρέπει να είναι συμβατά με υπάρχοντα πρότυπα και πλατφόρμες λογισμικού
- Η εγκατάσταση και χρήση τους να είναι εφικτή και εύκολη από το μέσο χρήστη
- Να υποστηρίζεται ένα αρκετά υψηλό εύρος ζώνης για την κάλυψη των εφαρμογών πολυμέσων

- Να υποστηρίζουν ασφαλή μεταφορά δεδομένων.

Στο χώρο αυτό δημιουργήθηκαν διάφορες τεχνολογίες, οι οποίες βασίζονται:

- Στο τηλεφωνικό δίκτυο
- Στο δίκτυο ισχύος
- Σε ασύρματα δίκτυα.

Τεχνολογίες βασισμένες στο δίκτυο τηλεφωνίας

Η πιο σημαντική τεχνολογία που αναπτύχθηκε βασισμένη στο οικιακό τηλεφωνικό δίκτυο είναι αυτή που δημιούργησε το consortium HPNA. Ξεκίνησε το 1998 με εύρος ζώνης 1MBps και στην τρίτη του έκδοση (HPNA 3.0, 2003) υποστηρίζει ταχύτητες έως και 240 MBps. Η λειτουργία του HPNA βασίζεται στη πολύπλεξη των σημάτων των επιμέρους δικτύων στο χώρο της συχνότητας. Έτσι μπορεί να υποστηρίζει ταυτόχρονα κλασική τηλεφωνία, xDSL και οικιακό δίκτυο.

Τεχνολογίες βασισμένες στο δίκτυο ισχύος

Ο τρόπος λειτουργίας του οικιακού δικτύου ισχύος είναι τέτοιος που το καθιστά ιδιαίτερα εχθρικό μέσον για την μετάδοση δεδομένων. Το κυριότερο πρόβλημα είναι το επίπεδο του θορύβου που μεταβάλλεται στοχαστικά λόγω της διακοπτόμενης συμπεριφοράς των ηλεκτρικών συσκευών. Παρόλα αυτά, με την εξέλιξη των τεχνολογιών στο χώρο αυτό επί σειρά ετών, το οικιακό δίκτυο ισχύος αποτελεί σήμερα ένα μέσο, ικανό για τη μεταφορά δεδομένων και τον έλεγχο οικιακών συσκευών. Η χρήση του οικιακού δικτύου ισχύος έχει τα παρακάτω πλεονεκτήματα:

- η παρουσία πληθώρας ρευματοληπτών σε ένα σπίτι ή ακόμη και σε ένα δωμάτιο επιτρέπει τη χρήση της υπάρχουσας καλωδίωσης
- ικανότητα μετάδοσης δεδομένων χρησιμοποιώντας το ελεύθερη περιοχή του φάσματος
- το οικιακό δίκτυο ισχύος επιτρέπει τη μετάδοση πληροφοριών εικόνας και ήχου σε σε ολόκληρο το σπίτι
- υποστηρίζει ρυθμούς της τάξης των 10 Mbps.

Τα βασικότερα μειονεκτήματα της χρήσης του οικιακού δικτύου ισχύος είναι:

- η μεγάλη ισχύς θορύβου που υπάρχει στο δίκτυο ισχύος περιορίζει σημαντικά το μέγιστο ρυθμό μετάδοσης δεδομένων.

- ο μεγάλος αριθμός ρευματοληπτών στο οικιακό δίκτυο ισχύος περιορίζει την ασφαλή μετάδοση δεδομένων

- η πληθώρα των συνδεόμενων οικιακών συσκευών συνιστά μια πολύ χαμηλή αντίσταση στην γραμμή. Απαιτείται έτσι μεγάλη ισχύς μετάδοσης στον πομπό καθώς και μεγάλη ευαισθησία στον δέκτη

- τα modems που συνδέονται στο δίκτυο ισχύος έχουν αισθητά υψηλότερο κόστος από τα modems που συνδέονται στο τηλεφωνικό δίκτυο.

Αρχικά, λόγω της μειωμένης ασφάλειας στη μεταφορά δεδομένων, η χρήση του οικιακού δικτύου ισχύος στόχευε στον έλεγχο των οικιακών συσκευών. Η ανάπτυξη όμως των αντίστοιχων τεχνολογιών επιτρέπει σήμερα τη μεταφορά δεδομένων σε επίπεδο IP. Οι πιο σημαντικές τεχνολογίες συστημάτων οικιακού ελέγχου είναι οι Lonworks, HomePlug, CeBus και X-10.

Η τεχνολογία LonWorks είναι η πιο διαδεδομένη και αυτή που προσφέρει τις περισσότερες δυνατότητες στον χρήστη. Αναπτύχθηκε και εξελίχθηκε από την εταιρεία Echelon. Μέχρι σήμερα υπάρχουν παγκοσμίως πάνω από 6 εκατομμύρια συσκευές της τεχνολογίας αυτής. Χαρακτηριστικό των δικτύων αυτών είναι ότι δεν υπάρχει κεντρικός ελεγκτής ή ελεγχόμενες συσκευές, αλλά η επικοινωνία μεταξύ των συσκευών (κόμβοι) επιτυγχάνεται μέσω ενός πρωτοκόλλου.

Τεχνολογίες βασισμένες σε ασύρματα δίκτυα

Τα ασύρματα δίκτυα αποτελούν εκ πρώτης όψεως την καλύτερη λύση για την οικιακή δικτύωση. Τα κυριότερα πλεονεκτήματα της ασύρματης δικτύωσης είναι:

- λόγω της φύσης του δικτύου δεν χρειάζεται υποδομή καλωδίωσης

- απλή εγκατάσταση και χρήση των συσκευών

- ανεξάρτητη λειτουργία του τηλεφωνικού δικτύου

- δυνατότητα ελέγχου των συσκευών εν κινήσει

Οι δύο κύριες τεχνολογίες των ασύρματων δικτύων είναι η επικοινωνία μέσω υπέρυθρης ακτινοβολίας και η επικοινωνία μέσω μικροκυμάτων. Στον τομέα των μικροκυμάτων χρησιμοποιούνται τα πρότυπα 802.11a/b/g και Bluetooth.

1.1.2 Τεχνολογίες με νέα καλωδίωση

Η περίπτωση της χρήσης νέας καλωδίωσης πρέπει να εξεταστεί ξεχωριστά στην περίπτωση ήδη χτισμένων οικιών και στην περίπτωση οικιών προς ανέγερση. Στην πρώτη κατηγορία, το κόστος εγκατάστασης της κατάλληλης υποδομής καθιστά τη λύση αυτή ασύμφορη. Στη δεύτερη περίπτωση όμως, με κατάλληλη μελέτη και εγκατάσταση της υποδομής κατά την ανέγερση της οικίας, το κόστος μειώνεται πάρα πολύ. Η νέα οικία μπορεί να υποστηρίζει ταχύτατα δίκτυα, κατάλληλα για την κοινή δικτύωση οικιακών συσκευών και δικτύων δεδομένων. Έτσι μπορούν να χρησιμοποιηθούν τεχνολογίες όπως Firewire, Fast Ethernet και USB.

1.2 Χαρακτηριστικά οικιακής πύλης

Μια οικιακή πύλη έχει ως στόχο την ενοποίηση υπηρεσιών και την διασύνδεση πολλών επιμέρους δικτύων. Οι βασικότερες υπηρεσίες που καλείται να ενοποιήσει κατατάσσονται στις παρακάτω κατηγορίες:

- έλεγχος και αυτοματισμός οικιακών συσκευών
- τηλεφωνία
- τοπικά δίκτυα δεδομένων και πρόσβαση στο διαδίκτυο
- εφαρμογές πολυμέσων και ψυχαγωγίας

Καθώς οι οικιακές πύλες αποτελούν συστήματα για την διευκόλυνση της καθημερινής ζωής των χρηστών, μεγάλο μέρος στο σχεδιασμού τους βασίζεται στις απαιτήσεις και τις ανάγκες του μέσου χρήστη. Οι βασικότερες από αυτές είναι:

- η εγκατάσταση της οικιακής πύλης πρέπει να αποφεύγει την οποιαδήποτε παρέμβαση και προσθήκη στην υπάρχουσα καλωδίωση του σπιτιού. Στις υπάρχουσες οικίες, όπου η προσθήκη καλωδίωσης συνεπάγεται μεγάλο χρηματικό και πιθανότατα και εργονομικό κόστος, οι προσθήκες πρέπει να είναι ελάχιστες. Είναι προτιμότερο, αν αυτό είναι δυνατό, να χρησιμοποιηθεί ασύρματη δικτύωση.
- Πρόσβαση στο διαδίκτυο με υψηλές ταχύτητες και δυνατότητα ύπαρξης τοπικού δικτύου

- η τηλεφωνία χαμηλού κόστους με δυνατότητα επιλογής του παροχέα υπηρεσιών ανάλογα με τον προορισμό της κλήσης. Ζητείται συχνά ο ενιαίος τρόπος κλήσης σε κάθε περίπτωση. Ανεξάρτητα από την υποκείμενη τεχνολογία κατά την εγκατάσταση ή την διάρκεια μιας κλήσης VoIP , είναι επιθυμητή η διαφανής χρήση κοινών αναλογικών ή και ψηφιακών τηλεφώνων.

- δυνατότητα απομακρυσμένου ελέγχου του σπιτιού από μια πληθώρα μέσων, όπως το διαδίκτυο, το τηλέφωνο, ασύρματα εντός του σπιτιού, πχ με ένα PDA.

- υπηρεσίας ψυχαγωγίας, όπως πολυμέσα και βίντεο κατέπιλογήν (video on demand).

1.3 Το σύστημα εντός ολοκληρωμένου κυκλώματος REGATE

Στο πλαίσιο της διπλωματικής αυτής εξετάζεται η χρήση του Linux στο σύστημα εντός ολοκληρωμένου κυκλώματος REGATE . Το σύστημα αυτό σχεδιάστηκε και υλοποιήθηκε από την εταιρία InAccess Networks. Το REGATE έχει σχεδιαστεί με τέτοιο τρόπο, ώστε να αποτελεί μια έξυπνη λύση στον χώρο των οικιακών πυλών. Στην ενότητα αυτή περιγράφεται η αρχιτεκτονική του συστήματος.

Ο σχεδιασμός του REGATE βασίζεται στην ιδέα ενός αποκεντρωμένου συστήματος. Το σύστημα αυτό περιέχει τις απαραίτητες δομικές μονάδες που αναλαμβάνουν την επιτάχυνση, από το υλικό, διεργασιών απαιτητικών σε υπολογιστική ισχύ. Με τον τρόπο αυτό αυξάνεται η διαθεσιμότητα του κεντρικού επεξεργαστή. Έτσι, με μια έξυπνη αρχιτεκτονική αποφεύγεται η χρήση κάποιου ισχυρότατου επεξεργαστή, διατηρώντας έτσι το κόστος σε χαμηλότερα επίπεδα, τόσο σε επίπεδο κόστους υλοποίησης όσο και κατανάλωσης ισχύος.

Με βάση το παραπάνω σκεπτικό, το υλικό του REGATE περιλαμβάνει μονάδες που αναλαμβάνουν την εκτέλεση λειτουργιών, όπως:

- Δημιουργία μονάδας άμεσης μεταφοράς δεδομένων μεταξύ περιφερειακών και κεντρικής μνήμης, χωρίς τη μεσολάβηση του επεξεργαστή. Επιτυγχάνεται με την ενσωμάτωση ελεγκτή άμεσης πρόσβασης στη μνήμη (Direct Memory Access controller, DMAc).

- Επιτάχυνση του ρυθμού μετάδοσης των δεδομένων, με την αυτόματη διανομή των εισερχόμενων πακέτων δεδομένων στα κατάλληλα περιφερεια-

κά ανάλογα με το πρωτόκολλο. Για σκοπό αυτό υλοποιήθηκαν μονάδες MAC, AAL0/5, TDM/HDLC, Utopia, 802.3 MAC σε υλικό.

- Ενσωμάτωση μονάδα κρυπτογράφησης σε υλικό. Μια οικιακή πύλη πρέπει να προσφέρει ασφαλή μεταφορά δεδομένων. Για την συγκεκριμένη υλοποίηση επιλέχθηκε η κρυπτογράφηση αλγορίθμων DES/TDES, η οποία απαντάται ευρύτατα σε εφαρμογές όπως SSHSFTP και στο IPSec.

- Διεπαφή PCMCIA για την επέκταση του συστήματος. Μέσω της διεπαφής αυτής ενσωματώνονται στο σύστημα κάρτες δικτύου 802.11, Bluetooth, ISDN κλπ.

- Συμπίεση φωνής, κυρίως για τηλεφωνία. Επιτυγχάνεται μέσω διεπαφής με εξωτερικό επεξεργαστή DSP. Υποστηρίζει τα πρωτόκολλα G.711, G.829, G.726 κλπ.

- Ενσωμάτωση επεξεργαστή RISC για τον έλεγχο του συστήματος, σηματοδοσία και εκτέλεση εφαρμογών.

Η αρχιτεκτονική του REGATE φαίνεται στο σχήμα 1.1. Συγκεκριμένα, οι μονάδες και διεπαφές που αποτελούν το REGATE είναι:

- Κεντρικός δίαυλος AMBA. Έχει το πλεονέκτημα ότι είναι συμβατός με πάρα πολλές υπάρχουσες περιφερειακές μονάδες.

- Κεντρικό επεξεργαστή LEON-2. Είναι συμβατός με την αρχιτεκτονική του SPARC V8. Περιγράφεται αναλυτικά σε επόμενο κεφάλαιο.

- Δύο διεπαφές MII 10/100. Συνδέονται είτε με ολοκληρωμένα φυσικού επίπεδου Ethernet για δίκτυο δεδομένων είτε HPNA για πρόσβαση στο οικιακό δίκτυο.

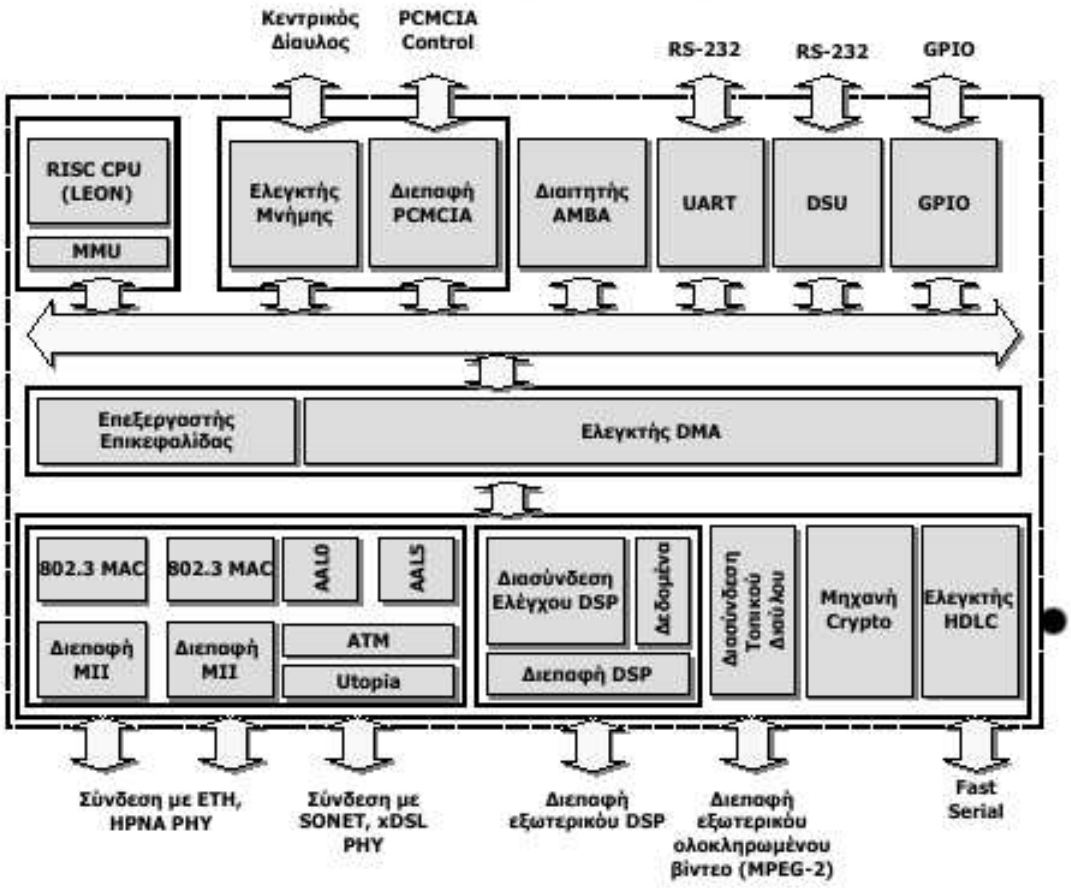
- Διεπαφή TDM/Fast serial 2 Mbps για επικοινωνία φωνής (ISDN-PRI)ή HDLC επικοινωνία δεδομένων.

- Δυο ασύγχρονες σειριακές διεπαφές

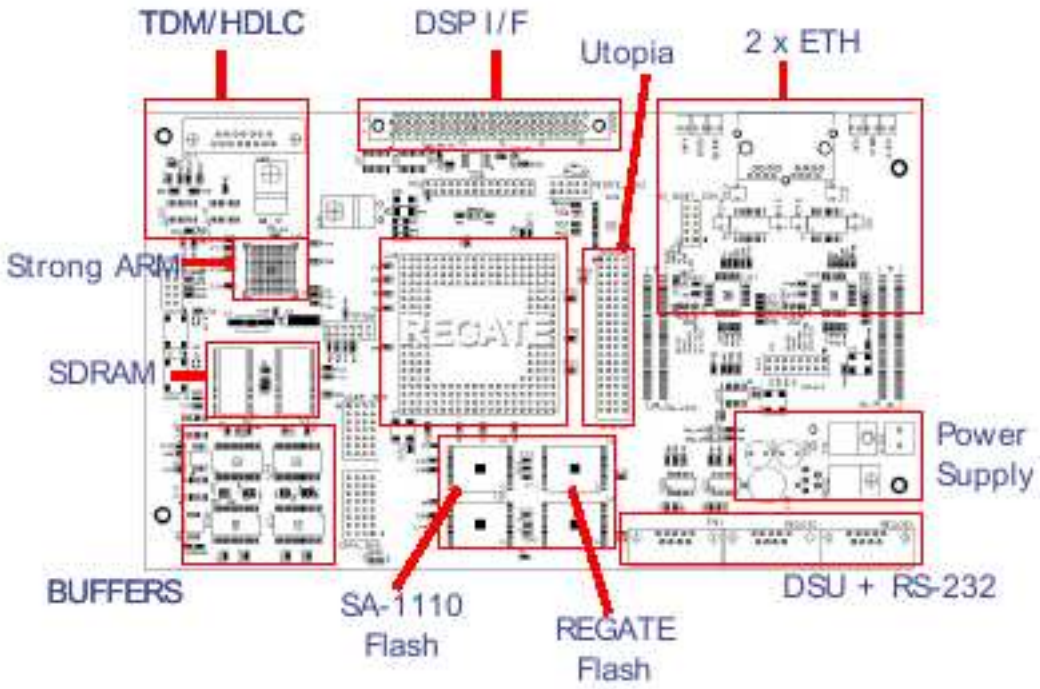
- Διεπαφή τοπικού διαύλου 8/16 βιτς για σύνδεση ολοκληρωμένων για streaming, πχ. αποκωδικοποιητή MPEG-2 ή εξωτερικό DSP.

- 32-bit δίαυλο για διασύνδεση με εξωτερικές μνήμες (SRAM, SDRAM, FLASH)

- 16 σήματα εισόδου/εξόδου γενικού σκοπού (GPIO)



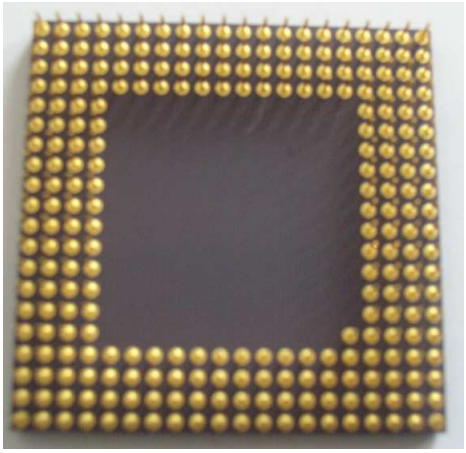
Σχήμα 1.1: Αρχιτεκτονική του REGATE



Σχήμα 1.2: Διάταξη της πλακέτας του REGATE

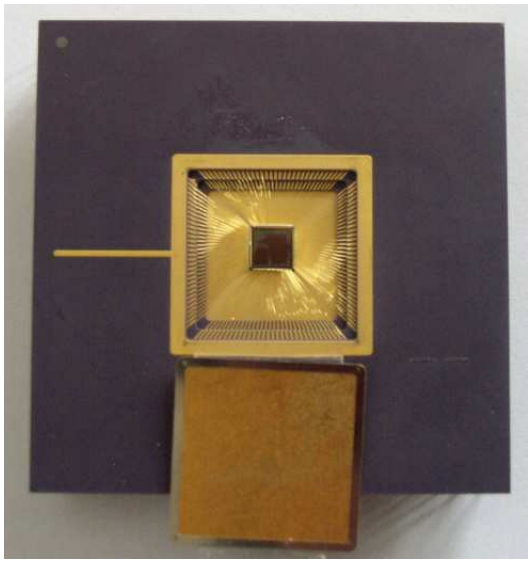
Στο σχήμα 1.2 φαίνεται η διάταξη της πλακέτας του REGATE. Στο σχήμα διακρίνεται η θέση καθενός από τα παραπάνω συστατικά στοιχεία του συστήματος. Στα υπόλοιπα σχήματα εμφανίζεται το REGATE στην πλακέτα του και μόνο του.

Σχήμα 1.3: Η πλακέτα του REGATE



Σχήμα 1.4: Κάτω όψη του REGATE SoC

Σχήμα 1.5: Πάνω όψη του REGATE SoC



Σχήμα 1.6: Πάνω όψη του REGATE SoC με εκτεθειμένο τον πυρήνα

Κεφάλαιο 2

Ο επεξεργαστής LEON-2

Το REGATE χρησιμοποιεί τον επεξεργαστή LEON . Ο LEON είναι ένα VHDL μοντέλο με δυνατότητα σύνθεσης ενός 32-bit επεξεργαστή που είναι συμβατός με την αρχιτεκτονική του SPARC V8 (IEEE-1754). Σχεδιάστηκε από τον Jiri Gaisler (Gaisler Research, www.gaisler.com) με την υποστήριξη της Ευρωπαϊκής Διαστημικής Υπηρεσίας (European Space Agency, ESA)και της Silicon and Software Systems. Το ιδιαίτερο χαρακτηριστικό του είναι ότι είναι ανεκτικός σε διάφορα σφάλματα που μπορεί να προκληθούν από ποικίλες πηγές, όπως για παράδειγμα η έκθεση σε ηλεκτρομαγνητική ακτινοβολία, και να καταλήξει στο σωστό αποτέλεσμα. Στην υλοποίηση του REGATE χρησιμοποιείται η έκδοση που ακολουθεί την άδεια χρήσης LGPL και δε διαθέτει το παραπάνω χαρακτηριστικό. Συγκεκριμένα, ο επεξεργαστής που χρησιμοποιείται στο REGATE είναι ο LEON-2. Ο επεξεργαστής αυτός έχει τα ακόλουθα χαρακτηριστικά:

- Αρχιτεκτονική μειωμένου σετ εντολών (RISC)
- Συμβατός με SPARC V8, Pipeline5 σταδίων
- Ξεχωριστές μνήμες cache εντολών και δεδομένων (αρχιτεκτονική τύπου Harvard)
- Διάδρομο AMBA-2.0 AHB και APB
- Μονάδα διαχείρισης μνήμης (MMU)
- Γρήγορη μνήμη με συσχέτιση 1-4 σετ και 1-64 Kbyte/sec. Ο αλγόριθμος αντικατάστασης μπορεί να είναι τυχαίος, LRUή LRR
- Παρακολούθηση γρήγορης μνήμης δεδομένων

- 8/ 16 /32-bits ελεγκτή μνήμης για εξωτερική μνήμη PROM και SRAM
- 32-bit PC133 SDRAM ελεγκτή μνήμης
- Περιφερειακά on chip, όπως σειριακές θύρες, χρονιστές, ελεγκτή διακοπών και 16-bit θύρες εισόδου/εξόδου
- Υποστηρίζει μονάδα debug και ανίχνευσης (trace buffer)
- κατάσταση power down

Ο LEON είναι κατάλληλος για embedded συστήματα και εφαρμογές system on a chip, καθώς είναι επεκτάσιμος και διαθέτει LGPL άδεια, που τον κάνει μια οικονομική και συμφέρουσα επιλογή.

Στο κεφάλαιο αυτό περιγράφονται σύντομα τα σπουδαιότερα χαρακτηριστικά της αρχιτεκτονικής του LEON και του SPARC V8 γενικότερα, τα οποία είναι αναγκαία για την κατανόηση, τόσο της διαδικασίας ελέγχου καλής λειτουργίας του όλου συστήματος, όσο και της μεταφοράς του Linux στο σύστημα αυτό.

2.1 Καταχωρητές

Ο LEON , καθώς είναι συμβατός με το πρότυπο του SPARC V8, έχει δύο τύπους καταχωρητών:

- τους καταχωρητές γενικής χρήσης ή λειτουργίας(general purpose/working registers)
- τους καταχωρητές ελέγχου/κατάστασης (control/status registers)

Οι καταχωρητές γενικής χρήσης της μονάδας ακεραίων ονομάζονται ‘‘καταχωρητές r’’, ενώ οι αντίστοιχοι της μονάδας κινητής υποδιαστολής ονομάζονται ‘‘καταχωρητές f’’. Επειδή στην υλοποίηση της οικιακής πύλης REGATE δεν χρησιμοποιείται μονάδα κινητής υποδιαστολής, στη συνέχεια, το ενδιαφέρον θα επικεντρωθεί στη μονάδα ακεραίων.

Οι βασικοί καταχωρητές που χρησιμοποιούνται από την IU είναι οι:

- Processor Status Register (PSR), περιγράφει την κατάσταση του συστήματος
- Window Invalid Mask (WIM), , δίνει το ‘‘λάθος’’ παράθυρο καταχωρητών
- Trap Base Register (TBR), περιέχει την βάση του πίνακα με τις συναρτήσεις εξυπηρέτησης διακοπών

- Multiply / Divide Register (Y), για πράξεις πολλαπλασιασμού διαίρεσης
- Program Counters (PC, nPC), μετρητές προγράμματος

Στη συνέχεια αυτού του κειμένου, οι παραπάνω καταχωρητές θα αναφέρονται με τη συντομογραφία τους. Καθώς η πλήρης κατανόηση της λειτουργίας τους είναι απαραίτητη, τόσο για την κατανόηση των δοκιμών που έγιναν για την διαπίστευση της καλής λειτουργίας ολόκληρου του συστήματος, όσο και για την μεταφορά του linux στην πλατφόρμα του REGATE , οι καταχωρητές αυτοί περιγράφονται λεπτομερώς στην επόμενη υποενότητα.

2.1.1 Παράθυρα Καταχωρητών (Register Windows)

Η μονάδα ακεραίων μπορεί να περιέχει από 40 μέχρι 520 καταχωρητές τύπου r. Χωρίζονται σε 8 ***καθολικούς (global)*** καταχωρητές και ένα αριθμό, που εξαρτάται από την υλοποίηση, από ***ομάδες (sets)*** 16 καταχωρητών. Η κάθε ομάδα καταχωρητών υποδιαιρείται σε 8 καταχωρητές ***εισόδου (in)*** και 8 ***τοπικούς (local)*** καταχωρητές.

Ένα ***παράθυρο (window)*** καταχωρητών αποτελείται από τους 8 καταχωρητές εισόδου και τους 8 τοπικούς καταχωρητές μιας ομάδας, μαζί με τους 8 καταχωρητές εισόδου της γειτονικής ομάδας, που όμως λειτουργούν σαν ***καταχωρητές εξόδου (out)*** για το παράθυρο αυτό. Σε μια δεδομένη χρονική στιγμή, μια εντολή μπορεί να "δει" τους 8 καθολικούς καταχωρητές μαζί με ένα παράθυρο καταχωρητών. Οι γενικοί καταχωρητές συνήθως ονομάζονται είτε βάσει του απόλυτου αριθμού τους σε κάθε συνδυασμό παραθύρου και καθολικών μεταβλητών, είτε βάσει της δομής του παραθύρου. Η ονοματολογία που χρησιμοποιείται φαίνεται στον πίνακα 2.1.

Ονομασία κατά παράθυρο	**Γενική ονομασία**
in[0] - in[7]	r[24] - r[31]
local[0] - local[7]	r[16] - r[23]
out[0] - out[7]	r[8] - r[15]
global[0] - global[7]	r[0] - r[7]

Πίνακας 2.1: Ονοματολογία καταχωρητών

Ένα πολύ ενδιαφέρον χαρακτηριστικό της αρχιτεκτονικής του SPARC V8 είναι η ***επικάλυψη των παραθύρων (window overlapping)***. Ο αύξων

αριθμός του τρέχοντος παραθύρου δίνεται από το πεδίο CWP του καταχωρητή PSR. Κάθε παράθυρο μοιράζεται τους in και out καταχωρητές του με τα δύο γειτονικά παράθυρα του. Έστω ότι n είναι το τρέχον παράθυρο. Τότε, οι out καταχωρητές του *n+1* παραθύρου είναι οι ίδιοι με τους in καταχωρητές του n. Αντίστοιχα, οι out καταχωρητές του n ταυτίζονται με τους in του παραθύρου *n-1*.

2.1.2 Καταχωρητές ελέγχου/κατάστασης της IU

Οι καταχωρητές ελέγχου/κατάστασης της IU έχουν μέγεθος 32 bits . Σε κάθε υλοποίηση ενός SPARC V8 συστήματος υπάρχουν οπωσδήποτε οι καταχωρητές PSR, WIM, TBR, Y, PC, nPC. Η δομή και η λειτουργία καθενός τους περιγράφεται παρακάτω:

PSR

Ο καταχωρητής PSR περιέχει στοιχεία για την κατάσταση του επεξεργαστή και για την ταυτότητα του. Τα πεδία του (bitfields) φαίνονται στο σχήμα.

impl αποτελείται από τα bits 28 έως και 31. Η τιμή του είναι σταθερή και προκαθορισμένη από την εκάστοτε υλοποίηση του επεξεργαστή. Δίνει την "ταυτότητα" της υλοποίησης.

ver στα bits 24 έως και 27 περιέχεται η έκδοση της υλοποίησης. Η τιμή αυτή εξαρτάται επίσης από το hardware και είναι σταθερή.

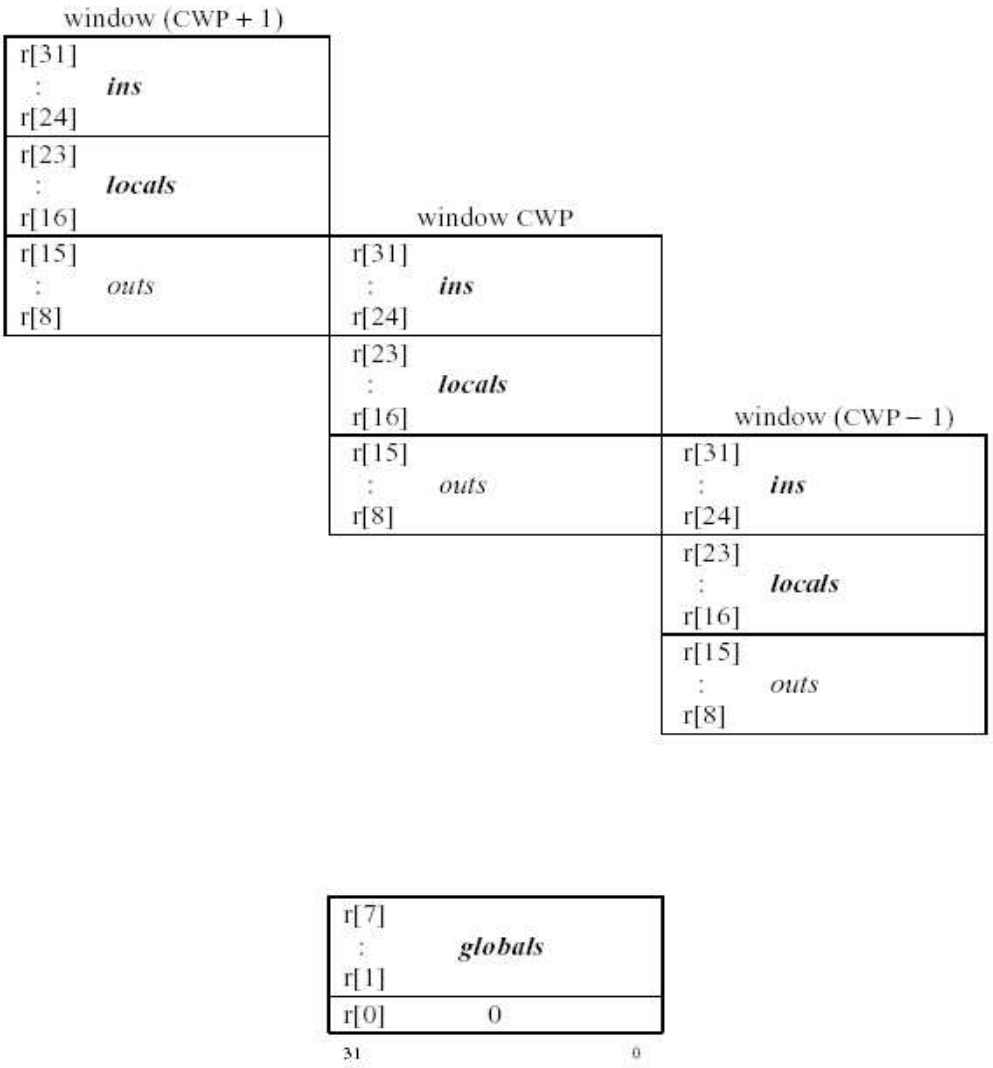
icc το όνομα του αποτελεί συντόμευση των λέξεων *integer condition codes*. Η τιμή του διαμορφώνεται από τα αποτελέσματα των αριθμητικών και λογικών πράξεων. Είναι δομημένο ως εξής *(βλ. σχήμα 2.3)*:

n Όταν το αποτέλεσμα της πράξης στην ALU είναι αρνητικό, παίρνει την τιμή 1

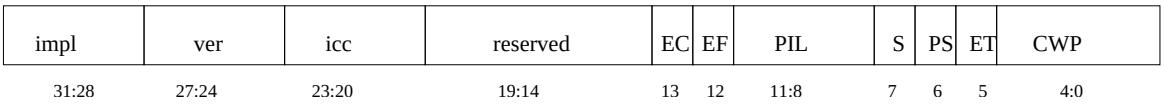
z Παίρνει την τιμή 1 όταν το αποτέλεσμα της τελευταίας πράξης στην ALU που επηρέασε το πεδίο icc είναι 0

v Αυτό είναι το bit υπερχείλισης(overflow). Γίνεται 1 όταν κάποια πράξη ξεπερνά το διάστημα τιμών που ορίζουν τα 32 bits με συμπλήρωμα ως προς δύο

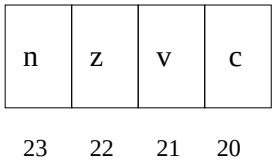
c Αν υπάρχει κρατούμενο (carry) ή δανεικό (borrow) από την τελευταία εντολή που μετέβαλε το πεδίο icc , παίρνει την τιμή 1



Σχήμα 2.1: επικάλυψη παραθύρων



Σχήμα 2.2: Ο καταχωρητής PSR



Σχήμα 2.3: το πεδίο icc

reserved τα bits 19:24 δεν χρησιμοποιούνται ακόμα. Όταν διαβάζονται, δίνουν την τιμή 0

EC το πεδίο αυτό, που σημαίνει enable coprocessor, είναι 1 όταν στην υλοποίηση υπάρχει συνεπεξεργαστής και είναι ενεργοποιημένος

EF αντίστοιχα με το προηγούμενο πεδίο, γίνεται 1, όταν υπάρχει και είναι ενεργοποιημένη μονάδα κινητής υποδιαστολής

PIL καθορίζει το επίπεδο πάνω απ' το οποίο ο επεξεργαστής θα δέχεται διακοπές (interrupts). Περιγράφεται λεπτομερώς σε επόμενη ενότητα

S όταν είναι 1, ο επεξεργαστής λειτουργεί σε κατάσταση "ἐπόπτη" (supervisor) και όταν είναι 0 σε κατάσταση απλού χρήστη (user)

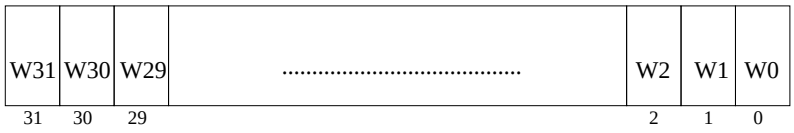
PS περιέχει την τιμή του bit S κατά την πιο πρόσφατη παγίδα (trap) ή διακοπή (interrupt).

ET ενεργοποιεί τα traps. Όταν είναι 1 τα traps είναι ενεργοποιημένα. Γίνεται αυτόματα 0 όταν ο επεξεργαστής μπαίνει σε ένα χειριστή παγίδας (trap handler) και ξαναγίνεται 1 μόλις βγει απ' αυτόν. Περιγράφεται σε επόμενη ενότητα.

CWP σημαίνει current window pointer και δείχνει το ποιο είναι το τρέχων παράθυρο καταχωρητών γενικού σκοπού

WIM

Ο WIM αποτελεί έναν από τους πιο χαρακτηριστικούς καταχωρητές στην αρχιτεκτονική του SPARC V8. Παίρνει το όνομα του από τους όρους Window Invalid Mask. Η τιμή του μειώνεται με την εντολή save και αυξάνεται με την εντολή restore . Η δομή του φαίνεται στο σχήμα 2.4.

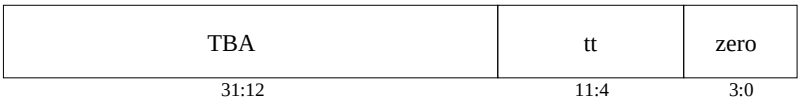


Σχήμα 2.4: ο καταχωρητής wim

Κάθε bit στον καταχωρητή αυτόν αντιστοιχεί στο αντίστοιχο παράθυρο καταχωρητών. Ο αριθμός των ενεργών bits του εξαρτάται από την υλοποίηση. Προφανώς για τον LEON-2 χρησιμοποιούνται 8 bits.

TBR

Σ′ αυτόν τον καταχωρητή αποθηκεύεται η διεύθυνση της βάσης του πίνακα που περιέχει τις διευθύνσεις των συναρτήσεων εξυπηρέτησης των traps και των interrupts. Όπως φαίνεται και στο σχήμα 2.5, αποτελείται από τα ακόλουθα πεδία:



Σχήμα 2.5: ο καταχωρητής tbr

TBA περιέχει τα 20 πιο σημαντικά bits της διεύθυνσης της βάσης του πίνακα. Η τιμή του ανατίθεται από supervisor software

tt περιέχει το εκτόπισμα (offset) από την βάση του πίνακα που αντιστοιχεί στην πιο πρόσφατη εμφάνιση κάποιου trap ή interrupt

zero είναι πάντα μηδέν

Y

Στον 32 bits καταχωρητή Y αποθηκεύεται η πιο σημαντική λέξη του διπλής ακρίβειας (double precision) γινομένου ενός πολλαπλασιασμού ακεραίων. Σ′ αυτόν τον καταχωρητή επίσης αποθηκεύεται η πιο σημαντική λέξη ενός ακέραιου διαιρετέου διπλής ακρίβειας κατά την διαίρεση ακεραίων.

PC, nPC

Οι δυο αυτοί καταχωρητές αποτελούν τους μετρητές προγράμματος. Ο πρώτος απ′ αυτούς περιέχει την διεύθυνση της τρέχουσας εντολής, ενώ ο nPC περιέχει την επόμενη εντολή. Στην περίπτωση που η τρέχουσα

εντολή είναι εντολή καθυστερημένης μεταφοράς, η επόμενη εντολή εκτελείται. Κατά την εκτέλεση της καθυστερημένης εντολής ο PC περιέχει την καθυστερημένη εντολή και ο nPC τον προορισμό της εντολής καθυστερημένης μεταφοράς.

2.2 Μνήμη και μονάδα διαχείρισης μνήμης (MMU)

Η μνήμη ενός SPARC V8 συστήματος ορίζεται ως η συλλογή όλων εκείνων των τοποθεσιών του συστήματος, στις οποίες υπάρχει πρόσβαση μέσω των εντολών load και store. Οι τοποθεσίες αυτές περιλαμβάνουν την πραγματική μνήμη (SRAM ή SDRAM), καταχωρητές εισόδου/ εξόδου και ειδικούς καταχωρητές, οι οποίοι είναι προσβάσιμοι μέσω *δεικτών εναλλακτικού χώρου (alternate space identifiers, ASIs).*

2.2.1 Μνήμη

Η *κύρια ή πραγματική μνήμη (real or main memory)* περιλαμβάνει τις τοποθεσίες στις οποίες η πρόσβαση γίνεται είτε με χρήση κάποιου ASI εκ των 0x8, 0x9, 0xA, 0xB, είτε όταν το πεδίο ASI σε συνδυασμό με κάποιο αντίστοιχο στοιχείο της MMU δηλώνει αναφορά στην κύρια μνήμη.

Αυτοί είναι οι μοναδικοί τρόποι που επιτρέπουν πρόσβαση στην κύρια μνήμη. Η ακριβής απαρίθμηση των ASIs και οι λειτουργικές λεπτομέρειες της MMU εξαρτώνται από την εκάστοτε υλοποίηση.

Οι καταχωρητές εισόδου/εξόδου περιλαμβάνουν τις τοποθεσίες στις οποίες η πρόσβαση γίνεται με:

- ASI διαφορετικό από 0x8, 0x9, 0xA, 0xB και 0x20-0x2F.
- τοποθεσίες που χρησιμοποιούν τα παραπάνω ASIs , με κατάλληλες όμως ρυθμίσεις της MMU.
- χρήση του συνεπεξεργαστή ή άλλους βοηθητικούς καταχωρητές

Οι καταχωρητές αυτοί διαβάζονται και γράφονται αντίστοιχα με τις εντολές load και store, ή τις alternate load και store. Οι τελευταίες μπορούν να ορίσουν το ποιο ASI θα χρησιμοποιήσουν και μπορούν να εκτελεστούν όταν ο επεξεργαστής βρίσκεται σε κατάσταση supervisor.

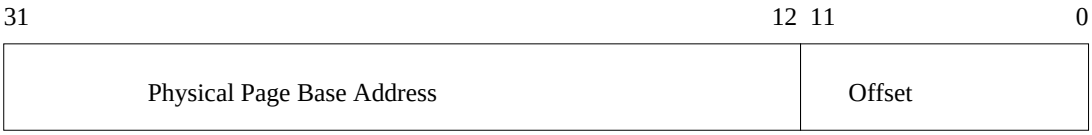
ASI	Χρήση
0x5	Άδειασμα (flush) της cache εντολών
0x6	Άδειασμα cache δεδομένων
0x8 0x9 0xA 0xB	Κανονική πρόσβαση με χρήση cache
0xC	Ετικέτες (tags) cache εντολών
0xD	Δεδομένα της cache εντολών
0xE	Ετικέτες (tags) cache δεδομένων
0xF	Δεδομένα της cache δεδομένων
0X10	Άδειασμα cache
0x13	Άδειασμα cache
0x19	Καταχωρητές της MMU
0x1C	Παράκαμψη (bypass) της MMU

Πίνακας 2.2: Ορισμοί των ASIs

2.2.2 Δομή και λειτουργία της MMU

Η υλοποίηση της MMU στον LEON ακολουθεί τις προδιαγραφές της MMU αναφοράς για SPARC V8. Τα κύρια χαρακτηριστικά μιας τέτοιας υλοποίησης είναι:

- 32-bit εικονική διεύθυνση (virtual address)
- 36-bit φυσική διεύθυνση (physical address)
- Σταθερό μέγεθος σελίδας 4K
- Υποστηρίζει αραιό χώρο διευθύνσεων με αντιστοίχηση τριών σταδίων (3 level map)
- Υποστηρίζει μεγάλες γραμμικές αντιστοιχίσεις (linear mappings), 4K, 256K, 16M, 4G bytes
- Υποστηρίζει πολλαπλά πλαίσια διεργασιών *(contexts)*
- Προστασία σε επίπεδο σελίδας



Σχήμα 2.6: Δομή φυσικής διεύθυνσης

- Επεξεργασία αστοχίας του hardware (hardware miss)

Η MMU παρέχει τρεις βασικές λειτουργίες:

1. Μεταφράζει εικονικές διευθύνσεις σε πραγματικές της πραγματικής μνήμης. Κάθε διεργασία μπορεί να έχει το δικό της πίνακα αντιστοίχησης και να μη βρίσκεται σε συνεχόμενες διευθύνσεις στην πραγματική μνήμη

2. Παρέχει προστασία μνήμης. Αυτό σημαίνει ότι μια διεργασία δεν μπορεί να διαβάζει ή να γράφει τις διευθύνσεις μιας άλλης. Δίνει έτσι τη δυνατότητα σε λειτουργικά συστήματα να υποστηρίζουν πολλαπλές διεργασίες (multitasking)

3. Υλοποιεί την εικονική μνήμη. Οι σελίδες που υπάρχουν στην κύρια μνήμη μπορούν να βρεθούν μέσω των πινάκων σελίδων (page tables). Αν ζητηθεί μια διεύθυνση κάποιας σελίδας που δε βρίσκεται στην κύρια μνήμη, παράγεται ένα σφάλμα σελίδας (page fault).

Μετάφραση διευθύνσεων

Η μετάφραση των διευθύνσεων στον LEON γίνεται είτε απευθείας, με παράκαμψη της MMU,είτε με μετάφραση από ένα ως τρία στάδια. Κατά τη διαδικασία της μετάφρασης, μία 32bits εικονική διεύθυνση μεταφράζεται σε μία 36bits φυσική διεύθυνση. Το μέγεθος των 36 bits παρέχει τη δυνατότητα για την διευθυνσιοδότηση μεγάλου χώρου φυσικών διευθύνσεων, μέχρι 64 GB, μέσω ενός 32bit διαύλου διευθύνσεων.

Στο σχήμα 2.6 απεικονίζεται η δομή μιας φυσικής διεύθυνσης. Όλες οι σελίδες είναι στοιχισμένες (aligned) σε περιοχές μεγέθους 4K. Με τη συνθήκη αυτή, τα 12 χαμηλότερης τάξης bits της εικονικής διεύθυνσης είναι ίδια με τα αντίστοιχα 12 bits της φυσικής διεύθυνσης, χωρίς να χρειαστεί κάποιου είδους μετάφραση. Για τη μετάφραση της εικονικής διεύθυνσης, υπάρχει στην κατάλληλη θέση ενός ειδικού πίνακα ένα στοιχείο που δίνει τον αριθμό της φυσικής σελίδας που αντιστοιχεί στην εικονική σελίδα που περιέχει την διεύθυνση αυτή.

Σχήμα 2.7: Δομή εικονικής διεύθυνσης

Επίπεδο	Μέγεθος αντιστοίχησης
0 (root)	4 Gbytes
1	16 Mbytes
2	256 Kbytes
3	4 Kbytes

Πίνακας 2.3: Μέγεθος σελίδων

Πιο συγκεκριμένα, ανάλογα με τον αριθμό των σταδίων που χρησιμοποιούνται για τη μετάφραση μιας διεύθυνσης, χρησιμοποιούνται οι αντίστοιχοι πίνακες. Οι πίνακες αυτοί, επειδή τα στοιχεία τους είναι σελίδες, ονομάζονται πίνακες σελίδων (page tables). Ανάλογα με το αν τα στοιχεία ενός τέτοιου πίνακα είναι δείκτες σε σελίδες ή σε πίνακες σελίδων, ο πίνακας αυτός ονομάζεται αντίστοιχα πίνακας στοιχείων σελίδας (page table entry, PTE) ή πίνακας καταλόγου σελίδων (page table directory, PTD). Όταν χρησιμοποιούνται τρία στάδια, πάντα ο τελευταίος πίνακας σελίδων είναι τύπου PTE και τα στοιχεία του δείχνουν σε φυσικές σελίδες.

Η δομή μιας εικονικής διεύθυνσης μνήμης απεικονίζεται στο σχήμα 2.7. Μια εικονική διεύθυνση χωρίζεται σε τέσσερα τμήματα. Το πρώτο τμήμα αποτελείται από τα 12 μικρότερης τάξης bits (0 - 11) και όπως προαναφέρθηκε, είναι ίδια με τα αντίστοιχα bits της φυσικής σελίδας. Τα επόμενα τρία μέρη χρησιμοποιούνται για δεικτοδότηση και η ακριβής σημασία τους εξαρτάται από τον αριθμό των σταδίων που χρησιμοποιούνται για τη μετάφραση των σελίδων. Το δεύτερο και το τρίτο τμήμα έχουν μέγεθος 6 bits και καταλαμβάνουν τα bits 12-17 και 18-23 αντίστοιχα, ενώ το τέταρτο μεγέθους 8 bits καταλαμβάνει τα bits 24 - 31.

Ανάλογα με τα επίπεδα αντιστοίχησης που χρησιμοποιούνται, το μέγεθος της σελίδας μπορεί να ποικίλει από 4 Kbytes μέχρι 4 Gigabytes. Στον πίνακα 2.3 φαίνεται το μέγεθος σελίδας που αντιστοιχεί σε κάθε επίπεδο αντιστοίχησης

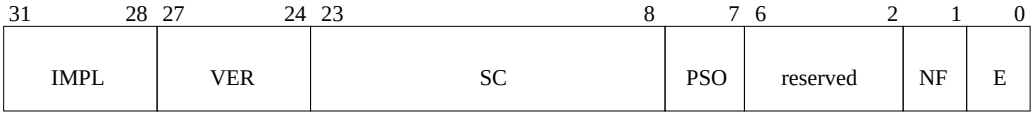
Καταχωρητές της Μονάδας διαχείρισης μνήμης

Στην υποενότητα αυτή περιγράφονται οι καταχωρητές της μονάδας διαχείρι-

σης μνήμης. Η κατανόηση της λειτουργίας τους είναι σημαντική για την κατανόηση τόσο της λειτουργίας της ίδιας της μονάδας διαχείρισης μνήμης όσο και των προγραμμάτων που δημιουργήθηκαν για τον έλεγχο της σωστής λειτουργίας της τελευταίας.

Control Register

Ο καταχωρητής ελέγχου περιέχει πληροφορίες για την ταυτότητα και την αρχιτεκτονική της μονάδας διαχείρισης μνήμης και την ενεργοποιεί. Τα πεδία του φαίνονται στο σχήμα 2.8.



Σχήμα 2.8: Καταχωρητής ελέγχου

impl τα τέσσερα μεγίστης τάξεως bits του καταχωρητή αυτού ορίζουν ένα πεδίο (implementation) που δίνει την ταυτότητα της υλοποίησης της εν λόγω μονάδας διαχείρισης μνήμης.

ver δίνει την έκδοση της υλοποίησης μιας συγκεκριμένης μονάδας διαχείρισης μνήμης.

sc Τα bits ελέγχου συστήματος (system control) εξαρτώνται από την εκάστοτε υλοποίηση και δεν είναι αναγκαίο να είναι όλα υλοποιημένα. Όποιο δεν έχει υλοποιηθεί διαβάζεται και γράφεται με την τιμή μηδέν.

pso όταν η τιμή του είναι ένα, το μοντέλο μνήμης που χρησιμοποιείται είναι το PSO (Partial Store Ordering), ενώ όταν είναι μηδέν χρησιμοποιείται το TSO (Total Store Ordering).

reserved δεσμευμένα bits για μελλοντική χρήση.

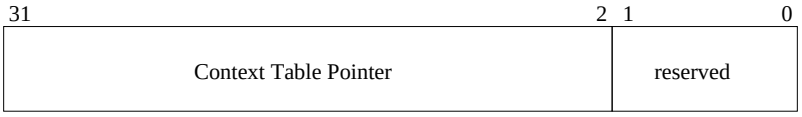
nf όταν το "no fault bit" είναι μηδέν, όταν εντοπιστεί από τη μονάδα διαχείρισης μνήμης σφάλμα, ανανεώνει τις τιμές των καταχωρητών FSR και FAR και δημιουργεί ένα σήμα σφάλματος στον επεξεργαστή. Όταν είναι ένα, αν το ASI που χρησιμοποιείται είναι 9, έχει την ίδια συμπεριφορά με όταν το bit αυτό έχει την τιμή 0. Όταν το ASI είναι διαφορετικό από 9, τότε επίσης ενημερώνει τις τιμές των δύο παραπάνω καταχωρητών, αλλά δε δημιουργεί σήμα σφάλματος στον επεξεργαστή.

e ενεργοποιεί και απενεργοποιεί τη μονάδα διαχείρισης μνήμης.

Context Register

Ο καταχωρητής δείκτη πίνακα πλαισίων (context table pointer register) δείχνει στον πίνακα πλαισίων διεργασιών (context table), που βρίσκεται στη φυσική μνήμη. Ο πίνακας δεικτοδοτείται από το περιεχόμενο του καταχωρητή αυτού. Ο δείκτης πίνακα πλαισίων (Context Table Pointer) βρίσκεται στα bits 2 - 35 του διαύλου φυσικών διευθύνσεων, κατά την πρώτη μεταφορά που συμβαίνει κατά την επεξεργασία αστοχίας εντολής ή δεδομένων.

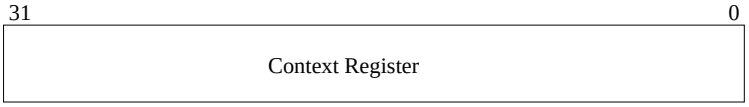
Ο πίνακας περιεχομένων που δείχνει ο δείκτης πίνακα πλαισίων πρέπει να είναι ευθυγραμμισμένος ως προς την αρχική του διεύθυνση με ένα πλαίσιο ίσο με το μέγεθος του.



Σχήμα 2.9: Καταχωρητής δείκτη πίνακα πλαισίων

Context Register

Ο καταχωρητής πλαισίου (context register) ορίζει το ποιος απ' όλους τους υπάρχοντες χώρους εικονικών διευθύνσεων είναι ο τρέχων. Κάθε υλοποίηση μονάδας διαχείρισης μνήμης πρέπει να ορίζει ένα μέγιστο αριθμό πλαισίων, που η τιμή του πρέπει να είναι κάποια δύναμη του δύο μείον ένα.



Σχήμα 2.10: Καταχωρητής πλαισίου

Diagnostic Registers

Οι διαγνωστικοί καταχωρητές παρέχουν πληροφορίες για την κατάσταση της μονάδας διαχείρισης μνήμης. Η ύπαρξη, η δομή και η λειτουργία τους εξαρτώνται από την υλοποίηση της μονάδας διαχείρισης μνήμης. Πρόσβαση στους καταχωρητές αυτούς γίνεται μέσω ενός εναλλακτικού χώρου διευθύνσεων. Οι τιμές αναφοράς των ASI για τους διαγνωστικούς καταχωρητές αναφέρονται στο εγχειρίδιο αρχιτεκτονικής του SPARC V8.

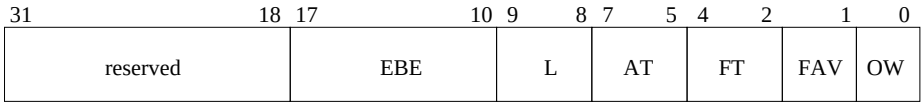
Fault Status Register

Ο καταχωρητής κατάστασης σφάλματος παρέχει πληροφορίες για τα σφάλματα που παράγονται από τη μονάδα διαχείρισης μνήμης. Λόγω της σωληναγωγικής αρχιτεκτονικής (pipelined architecture) της κεντρικής μονάδας επεξεργασίας, είναι δυνατό να συμβούν πολλά σφάλματα πριν τη δημιουργία μιας διακοπής. Τα σφάλματα της μονάδας διαχείρισης μνήμης κατατάσσονται σε τρεις κατηγορίες: στα σφάλματα πρόσβασης εντολής, στα σφάλματα πρόσβασης δεδομένων και τέλος στα σφάλματα μετάφρασης πίνακα. Αν συμβεί ένα σφάλμα εντολής πριν η ΚΜΕ προλάβει να διαβάσει ένα προηγούμενο σφάλμα εντολής, ο καταχωρητής κατάστασης περιέχει το τελευταίο σφάλμα. Στην περίπτωση αυτή ο καταχωρητής διεύθυνσης σφάλματος περιέχει την τελευταία διεύθυνση και το bit OW (overwrite) του καταχωρητή κατάστασης παίρνει την τιμή ένα.

Αν συμβούν πολλαπλά σφάλματα δεδομένων, η μονάδα διαχείρισης μνήμης διασφαλίζει ότι ο καταχωρητής κατάστασης σφάλματος περιέχει την κατάσταση σφάλματος του σφάλματος που έχει διαβάσει η κεντρική μονάδα επεξεργασίας. Στην περίπτωση που σφάλμα δεδομένων διαγράφει σφάλμα εντολής, το bit OW έχει την τιμή μηδέν, γιατί η κατάσταση σφάλματος απεικονίζεται σωστά. Ένα σφάλμα εντολής δεν μπορεί να γραφτεί πάνω από ένα σφάλμα δεδομένων.

Το σφάλμα μετάφρασης πίνακα βρίσκεται στην κορυφή της ιεραρχίας των σφαλμάτων της μονάδας διαχείρισης μνήμης και δε μπορεί να το αντικαταστήσει κανένα από τα παραπάνω σφάλματα.

Στο σχήμα 2.11 απεικονίζεται η δομή του καταχωρητή κατάστασης σφάλματος.



Σχήμα 2.11: Καταχωρητής κατάστασης σφάλματος

EBE Το πεδίο αυτό δηλώνει την ύπαρξη εξωτερικού σφάλματος στον δίαυλο (External Bus Error). Παίρνει τιμή όταν σε περίπτωση κάποιου εξωτερικού προβήματος. Οι τιμές του πεδίου εξαρτώνται από την υλοποίηση.

L το πεδίο αυτό δηλώνει το επίπεδο της μετάφρασης διευθύνσεων στο οποίο σημειώθηκε το σφάλμα.

AT Ο τύπος της πρόσβασης (access type) που προκάλεσε το σφάλμα καταγράφεται στο πεδίο αυτό.

FT Περιέχει τον τύπο του σφάλματος. Η τιμή του εξαρτάται από τον τύπο πρόσβασης σε μια σελίδα, τα δικαιώματα πρόσβασης της σελίδας και από εξωτερικά προβλήματα.

FAV Το bit Fault Address Valid παίρνει την τιμή ένα, αν το περιεχόμενο του καταχωρητή FAR είναι σωστό. Σε περίπτωση σφάλματος εντολής δεν είναι αναγκαίο να έχει την τιμή ένα. Σε περίπτωση όμως σφάλματος δεδομένων και μετάφρασης πρέπει πάντα να είναι ένα.

OW Η λειτουργία του bit επανεγγραφής έχει περιγραφεί παραπάνω.

Fault Address Register

Ο καταχωρητής διεύθυνσης σφάλματος είναι ένας 32 bits καταχωρητής που περιέχει την εικονική διεύθυνση που προκάλεσε το σφάλμα που έχει καταγραφεί στον καταχωρητή κατάστασης σφάλματος. Ο καταχωρητής αυτός είναι μόνο αναγνώσιμος. Οποιαδήποτε προσπάθεια εγγραφής του καταχωρητή αυτού αγνοείται.

Κεφάλαιο 3

Έλεγχος ορθής λειτουργίας υλικού

3.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφεται η διαδικασία με την οποία πραγματοποιήθηκε ο έλεγχος της καλής λειτουργίας του υλικού και η διόρθωση των σφαλμάτων. Για τη διεκπεραίωση αυτής της φάσης χρησιμοποιήθηκαν κατάλληλα εργαλεία τόσο για την εκτέλεση διαφόρων τεστ σε εξομοιωτή καθώς και αλλαγές στον VHDL κώδικα που περιγράφει το REGATE. Έλεγχοι έγιναν και στην πλατφόρμα με το FPGA και στην πλατφόρμα του REGATE. Οι δομικές μονάδες του REGATE που ελέγχθηκαν στα πλαίσια αυτής της διπλωματικής εργασίας είναι:

- ο ελεγκτής της μνήμης SDRAM και η μνήμη SDRAM
- η μονάδα διαχείρισης μνήμης (MMU)
- η μονάδα κρυπτογράφησης αλγορίθμων DES
- οι μονάδες MAC
- ο ελεγκτής διακοπών (Interrupt Controller)
- ο ελεγκτής άμεσης πρόσβασης μνήμης (DMA Controller)

Κάθε μία ενότητα του κεφαλαίου αυτού αντιστοιχεί σε κάθε μια από αυτές τις δομικές μονάδες.

3.2 Έλεγχος ελεγκτή μνήμης και μνήμης SDRAM

Στόχος του ελέγχου αυτού ήταν η διεξοδική διερεύνηση της ορθής λειτουργίας του ελεγκτή της μνήμης SDRAM. Ένας πολύ στοιχειώδης έλεγχος έγινε με τη διαδικασία εκκίνησης του micromonitor. Στον έλεγχο αυτό, ένα μέρος της SDRAM γράφεται σε ένα πρώτο πέρασμα με μία συγκεκριμένη τιμή. Σε ένα δεύτερο πέρασμα γίνεται ανάγνωση της περιοχής αυτής της μνήμης. Αν κάθε μία διεύθυνση περιέχει την τιμή που γράφτηκε στο πρώτο πέρασμα, τότε το τεστ θεωρείται επιτυχές.

Ένα πλήρες τεστ για την SDRAM πρέπει να ελέγχει τις εξής περιπτώσεις και προϋποθέσεις:

- Να καλύπτει όλη την έκταση της SDRAM.
- Να ελέγχει όλες τις πιθανές περιπτώσεις βραχυκυκλωμάτων
- Να ελέγχει το ρυθμό ανανέωσης της μνήμης
- Να ελέγχει όλα τα παραπάνω για όλους τους δυνατούς τρόπους πρόσβασης στη μνήμη

Η κάλυψη του συνόλου της μνήμης SDRAM είναι μια εύκολη διαδικασία. Ο ίδιος ο κώδικας του δοκιμαστικού προγράμματος καταλαμβάνει ένα πολύ μικρό μέρος της μνήμης RAM. Η ορθή λειτουργία αυτού του κώδικα πιστοποιεί τη καλή λειτουργία αυτής της περιοχής της μνήμης. Το υπόλοιπο μέρος της μνήμης, δηλαδή από το τέλος του κώδικα του τεστ μέχρι το τέλος της μνήμης SDRAM υπόκειται στις δοκιμασίες που περιγράφονται παρακάτω. Το τέλος του κώδικα του τεστ είναι πολύ εύκολο να βρεθεί μέσω ενός συμβόλου που καθορίζεται στη σύνδεση (linking) του προγράμματος.

Η πρώτη δοκιμασία που περνάνε η μνήμη και ο ελεγκτής μνήμης είναι η εξακρίβωση πιθανών βραχυκυκλωμάτων. Για το σκοπό αυτό χρησιμοποιείται η μέθοδος των κινούμενων άσσων. Με τη μέθοδο αυτή, κάθε φορά όλες οι διευθύνσεις της μνήμης γράφονται με μια τιμή που όλα τα bits της είναι μηδενικά πλην ενός. Μετά γίνεται ανάγνωση της μνήμης και σύγκριση με την τιμή που γράφτηκε. Αν όλα λειτουργούν σωστά, στην επόμενη επανάληψη ο άσσος της τιμής αναφοράς ολισθαίνει μία θέση και επαναλαμβάνεται η ίδια διαδικασία. Αυτό γίνεται 32 φορές, μέχρι δηλαδή ο άσσος να βρεθεί σε όλα τα bits μιας λέξης. Αν υπάρχει κάποιο βραχυκύκλωμα ή λάθος στη λογική σχεδίαση, με τον τρόπο αυτό θα φανεί αμέσως το πρόβλημα, γιατί σηκώνοντας ένα bit στην τιμή ένα, μπορεί να συμβούν οι εξής περιπτώσεις λάθους:

- Η τιμή στη διεύθυνση που παρουσιάζει πρόβλημα να παραμείνει μηδέν

- Να υπάρχουν παραπάνω από ένας άσσοι.

Για ακόμη μεγαλύτερη πληρότητα σ′ αυτό το τεστ, πέρα από τους κινούμενους άσσους χρησιμοποιούνται για το ίδιο είδος δοκιμασίας και οι τιμές 0x55555555 και 0xaaaaaaaa.

Ο επόμενος έλεγχος στοχεύει στην εξακρίβωση της ορθής λειτουργίας του ρυθμού ανανέωσης της μνήμης. Και αυτός ο έλεγχος γίνεται με δύο περάσματα. Στο πρώτο πέρασμα σε κάθε διεύθυνση μνήμης γράφεται σαν τιμή η ίδια η διεύθυνση. Για παράδειγμα η διεύθυνση 0x40200000 θα έχει την τιμή 0x40200000, η 0x40200004 την 0x40200004 κοκ. Μόλις γραφτεί και η τελευταία διεύθυνση μνήμης, ο επεξεργαστής μπαίνει σε ένα βρόχο καθυστέρησης (delay loop), ο οποίος υλοποιείται με χρήση καταχωρητή για τον μετρητή, αποφεύγοντας έτσι κάποια λειτουργία ανάγνωσης/εγγραφής στην μνήμη. Στη συνέχεια, ξεκινάει το δεύτερο πέρασμα. Αν όλα λειτουργούν σωστά, κάθε διεύθυνση θα έχει σαν τιμή την ίδια τη διεύθυνση. Αν υπάρξει κάποια αλλαγή τιμής σε κάποια διεύθυνση, ενώ ο έλεγχος των κινούμενων άσσων ήταν επιτυχής, υπάρχει πιθανότατα πρόβλημα με το ρυθμό ανανέωσης της μνήμης. Το πρόβλημα αυτό μπορεί να οφείλεται σε:

- κακή επιλογή τιμών χρονισμού της μνήμης στους αντίστοιχους καταχωρητές του επεξεργαστή
- κατασκευαστικό πρόβλημα της μνήμης
- κακή σχεδίαση σε επίπεδο μικροηλεκτρονικής, με αποτέλεσμα το τελικό κύκλωμα να έχει προβλήματα χρονισμού

Μετά την επιτυχή ολοκλήρωση των παραπάνω ελέγχων, επαναλαμβάνεται όλη η παραπάνω διαδικασία με τις ακόλουθες ρυθμίσεις:

- Ενεργοποιημένη μόνο την γρήγορη μνήμη δεδομένων (data cache)
- Ενεργοποιημένη μόνο την γρήγορη μνήμη εντολών (instruction cache)
- Ενεργοποιημένες τις γρήγορες μνήμες δεδομένων και εντολών
- Ενεργοποιημένη burst instruction χωρίς γρήγορες μνήμες
- Ενεργοποιημένη μόνο την γρήγορη μνήμη δεδομένων με burst
- Ενεργοποιημένη μόνο την γρήγορη μνήμη εντολών με burst
- Ενεργοποιημένες τις γρήγορες μνήμες δεδομένων και εντολών με burst

Οι παραπάνω επαναλήψεις των ελέγχων με διάφορους συνδυασμούς λειτουργίας των μνημών cache αποσκοπούν στον εντοπισμό προβλημάτων χρονισμού του συστήματος. Για τον ίδιο λόγο, στη δοκιμή του ελεγκτή και της μνήμης SDRAM , ακολουθεί η επανάληψη του παραπάνω συνόλου δοκιμών με την εξής τροποποίηση: η πρόσβαση και η εγγραφή στη μνήμη γίνεται με εντολές διπλής πρόσβασης. Οι εντολές αυτές είναι οι load double και store double και περιγράφονται αναλυτικά στο εγχειρίδιο αναφοράς του επεξεργαστή SPARC V8.

Οι εκτελέσεις των παραπάνω δοκιμών έδειξαν ότι δεν υπήρχε κάποιο πρόβλημα σχετικά με τη λειτουργία του ελεγκτή μνήμης και της μνήμης.

3.3 Έλεγχος της μονάδας διαχείρισης μνήμης (MMU)

Η μονάδα διαχείρισης μνήμης αναλαμβάνει την μετάφραση φυσικών διευθύνσεων σε εικονικές. Η δομή, οι καταχωρητές και ο τρόπος λειτουργίας της έχουν περιγραφεί στο προηγούμενο κεφάλαιο. Στην ενότητα αυτή περιγράφεται η διαδικασία που ακολουθήθηκε για την πιστοποίηση της σωστής λειτουργίας της. Η σωστή λειτουργία της μονάδας διαχείρισης μνήμης μπορεί να καθοριστεί από την σωστή λειτουργία των εξής επιμέρους τμημάτων:

- των επιμέρους δομικών μονάδων (καταχωρητές)
- μετάφραση σελίδων με αντιστοίχηση ¨ένα προς ένα'
- τη μετάφραση σελίδων με τυχαία αντιστοίχηση
- τη διαχείριση πολλών διεργασιών
- τη διαχείριση διακοπών
- τη συνεργασία με τον ελεγκτή άμεσης πρόσβασης στη μνήμη
- τη συνεργασία με τις γρήγορες μνήμες εντολών και δεδομένων

Για τον έλεγχο της μονάδας διαχείρισης μνήμης χρησιμοποιήθηκαν πολλές μέθοδοι και εργαλεία. Τα κυριότερα από αυτά ήταν η ενσωματωμένη μονάδα αποσφαλμάτωσης του επεξεργαστή, τα GNU binutils (κυρίως το πρόγραμμα objdump), ο προσομοιωτής ενός LEON-2 συστήματος tsim, το πρόγραμμα Modelsim για εξομοίωση, αυτόνομα μικρά προγράμματα που εκκινούν μόνα τους το σύστημα και ελέγχουν κάποιες συγκεκριμένες λειτουργίες του συστήματος.

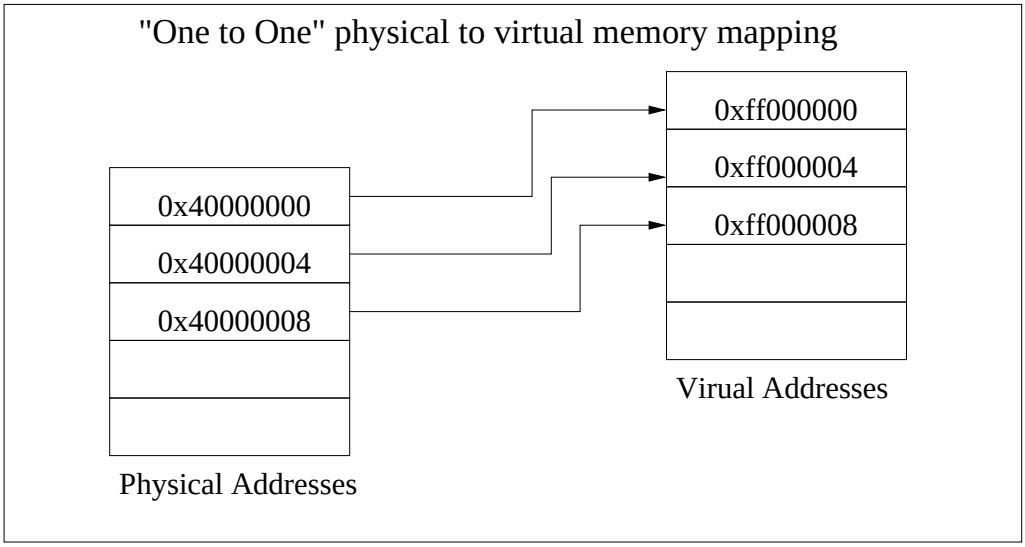
Λόγω της ιδιαιτερότητας της λειτουργίας της μονάδας διαχείρισης μνήμης είναι πολύ δύσκολο να απομονωθούν κάποια στοιχεία της παραπάνω λίστας και

να ελεγχθούν ένα προς ένα. Για το λόγο αυτό, τα διάφορα τεστ σχεδιάστηκαν με τέτοιο τρόπο, έτσι ώστε να καλύπτουν αρχικά όσο το δυνατόν λιγότερες διαφορετικές λειτουργίες το καθένα, και στη συνέχεια να γίνονται όλο και ποιο ολοκληρωτικά. Στο τελευταίο στάδιο χρησιμοποιήθηκε ο πυρήνας 2.6.5 του Linux μαζί με το BusyBox που κάλυπτε τις ανάγκες διεργασιών χώρου χρήστη. Για παράδειγμα, οι επιμέρους καταχωρητές εξετάζονται ουσιαστικά σε όλες τις δοκιμές και είναι ουσιαστικά αδύνατο να απομονωθούν. Αυτό που μπορεί να γίνει όμως είναι να έρθει το σύστημα σε μία τέτοια κατάσταση που να είναι προβλέψιμη, ντετερμινιστική και ανακτήσιμη, έτσι ώστε να μπορούν να εξαχθούν χρήσιμα συμπεράσματα και να μελετηθούν πιθανές προβληματικές καταστάσεις.

Ο πρώτος έλεγχος που έγινε στη μονάδα διαχείρισης μνήμης έγινε με ένα μικρό σχετικά πρόγραμμα, το οποίο ξεκινούσε μόνο του το σύστημα. Το πρόγραμμα αυτό καθορίζει την αντιστοίχηση της περιοχής της φυσικής μνήμης που καταλαμβάνουν τόσο ο κώδικας του ίδιου του προγράμματος αυτού με τα δεδομένα του όσο και η στοίβα του μαζί με τα μη αρχικοποιημένα δεδομένα σε μια εικονική περιοχή μνήμης. Η εικονική περιοχή υπολογίζεται προσθέτοντας μία συγκεκριμένη σταθερή τιμή στην φυσική διεύθυνση. Η διαδικασία αυτή αποτελεί την αντιστοίχηση "ένα προς ένα". Για παράδειγμα, έστω ότι έχουμε τη περιοχή μνήμης στη φυσική διεύθυνση 0x40000000 που εκτείνεται ας πούμε μέχρι την διεύθυνση 0x41000000. Έστω τώρα ότι θέλουμε να απεικονίσουμε την περιοχή αυτή σε μια εικονική που ξεκινάει από την διεύθυνση 0xff000000. Για να υπολογίσουμε την εικονική διεύθυνση που αντιστοιχεί σε κάθε φυσική αρκεί να προσθέσουμε την ποσότητα 0xbf000000. Την πρόσθεση αυτή την κάνει φυσικά η μονάδα διαχείρισης μνήμης. Ο τρόπος αυτός αντιστοίχησης απεικονίζεται στο σχήμα 3.1.

Στο πείραμα αυτό, έτσι όπως είναι οργανωμένο, δεν υπάρχει περίπτωση να σημειωθεί αστοχία δεδομένων ή εντολών (data/instruction miss), και οι αντίστοιχες διακοπές είναι επίσης απενεργοποιημένες. Απενεργοποιημένες είναι και οι γρήγορες μνήμες. Αν και αυτός ο έλεγχος είναι πάρα πολύ απλός, έχει το πλεονέκτημα ότι ορίζει ένα μικρό πείραμα, με εύκολα αναμενόμενη λειτουργία και υπολογίσιμα αποτελέσματα. Επιτυχία στο πείραμα αυτό δηλώνει ότι ένα στοιχειώδες "μονοπάτι" της μετάφρασης φυσικών διευθύνσεων σε εικονικές λειτουργεί σωστά.

Το επόμενο πείραμα αποτελεί επέκταση του πρώτου πειράματος. Καί στην περίπτωση αυτή, όλες οι τιμές και οι αρχικοποιήσεις είναι έτσι επιλεγμένες, ώστε να μην προκληθεί διακοπή από αστοχία ανάγνωσης δεδομένων ή εντολών. Η διαφορά εδώ είναι ότι δε χρησιμοποιείται η αντιστοίχηση ένα προς ένα αλλά μια τυχαία αντιστοίχηση μεταξύ φυσικών και εικονικών διευθύνσεων. Έτσι, η περιοχή της φυσικής μνήμης που περιέχει τον κώδικα και τα δεδομένα του δοκιμαστικού προγράμματος αντιστοιχίζεται σε πολλά μικρά κομμάτια με τη

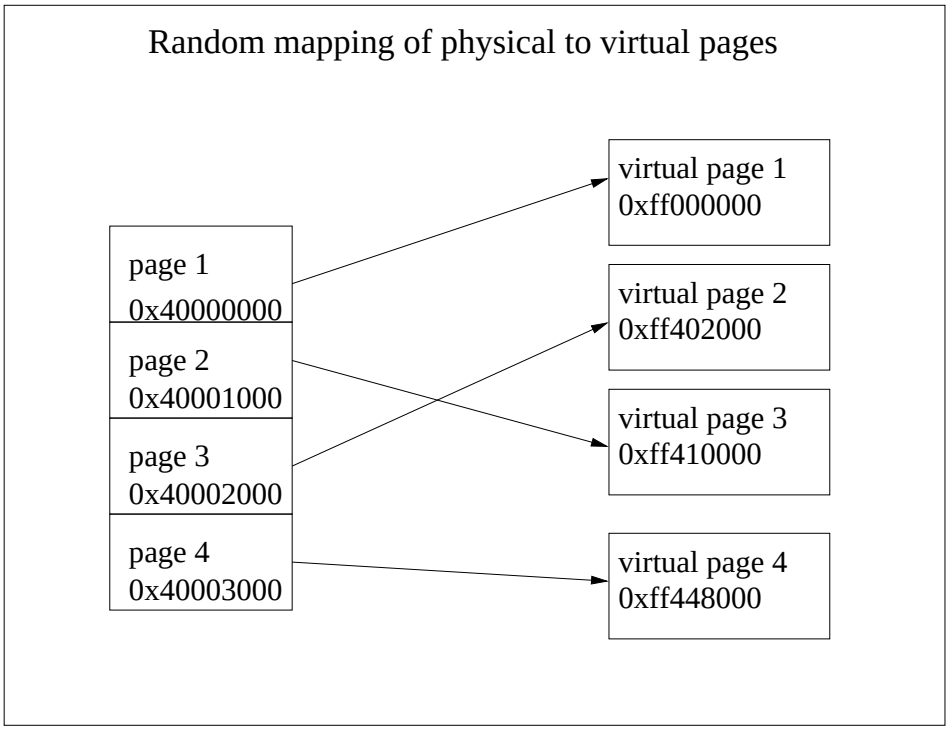


Σχήμα 3.1: Αντιστοίχηση φυσικών με εικονικές διευθύνσεις ¨ένα προς ένα'

μορφή σελίδων στην εικονική μνήμη. Η διαφορά αυτού του πειράματος από το προηγούμενο είναι ότι στην περίπτωση αυτή τίθεται σε λειτουργία μεγαλύτερο μέρος του κυκλώματος της μονάδας διαχείρισης της μνήμης. Στο σχήμα 3.2 φαίνεται η τυχαία αντιστοίχηση φυσικών και εικονικών σελίδων.

Σε αυτό το σημείο έχει εξασφαλιστεί η σωστή λειτουργία κάποιων βασικών διαδικασιών σχετικά με τη μετάφραση διευθύνσεων. Έχουμε εξασφαλίσει δηλαδή ότι ο καταχωρητής Context Table Ptr. λειτουργεί σωστά και ακόμη ότι λειτουργεί σωστά η μετάφραση των ενός έως τριών σταδίων (βλ. κεφάλαιο 2). Το επόμενο στάδιο είναι να διαπιστωθεί αν τα δύο πρώτα πειράματα περνούν επιτυχώς με ενεργοποιημένες τις γρήγορες μνήμες και την επιλογή burst instruction του επεξεργαστή. Στην περίπτωση αυτή εμφανίστηκαν τα πρώτα προβλήματα. Για την αντιμετώπιση τους έπρεπε να ακολουθηθούν οι εξής ενέργειες:

1. να βρεθεί ποια ακριβώς θα έπρεπε να είναι η σωστή εκτέλεση των εντολών
2. σε ποιο σημείο της ροής εκτέλεσης του κώδικα εντοπίζεται το πρώτο πρόβλημα
3. να προσδιοριστεί η φύση του προβλήματος, δηλαδή στη λειτουργία ποιας γρήγορης μνήμης οφείλεται.
4. να απομονωθεί ο κώδικας που εντοπίζει το σφάλμα σε ένα ελάχιστο δυνατό μέγεθος, έτσι ώστε να εξομοιωθεί η εκτέλεσή του με το Modelsim .



Σχήμα 3.2: Τυχαία αντιστοίχηση φυσικών με εικονικές διευθύνσεις

5. να διορθωθεί ο VHDL κώδικας που περιγράφει το κύκλωμα του REGATE

Για την εύρεση της σωστής εκτέλεσης των εντολών χρησιμοποιείται ο συνδυασμός της ενσωματωμένης μονάδας αποσφαλμάτωσης του επεξεργαστή, του προσομοιωτή tsim και του παραγόμενου κώδικα σε γλώσσα μηχανής, όπως αυτός εμφανίζεται με το πρόγραμμα objdump. Με τη μονάδα αποσφαλμάτωσης εντοπίζουμε πρώτα την περιοχή της εκτέλεσης του προγράμματος που δημιουργεί το πρόβλημα. Αυτή είναι γενικά μια επίπονη διαδικασία που απαιτεί πολλές δοκιμές και αρκετό χρόνο, δεδομένου ότι από τη μια το πρόγραμμα ελέγχου της ενσωματωμένης μονάδας αποσφαλμάτωσης του επεξεργαστή κρατάει ιστορία τριάντα εντολών και από την άλλη όταν λειτουργεί η μονάδα διαχείρισης μνήμης το πρόγραμμα αυτό απενεργοποιεί τα σημεία διακοπής εκτέλεσης (breakpoints) και παρατήρησης (watchpoints). Έτσι, για να μπορέσουμε να δούμε κάτι χρήσιμο από τη μονάδα αποσφαλμάτωσης, πρέπει χειροκίνητα να σταματήσουμε την εκτέλεση του προγράμματος την κατάλληλη χρονική στιγμή. Υπάρχει βέβαια και η επιλογή της εκτέλεσης των εντολών του προγράμματος μία μία με διακοπή (step execution), αλλά αυτό όπως θα εξηγηθεί παρακάτω εγκυμονεί κινδύνους για εσφαλμένα συμπεράσματα.

Παρόλα αυτά, στα δύο αυτά πειράματα αυτό δεν ήταν ιδιαίτερα δύσκολο, καθώς το μέγεθος των προγραμμάτων ήταν περιορισμένο. Μετά συγκρίνουμε με το objdump τις εντολές που εκτελεί ο επεξεργαστής σε σχέση με τις εντολές που υπάρχουν στον κώδικα του προγράμματος που τρέχουμε. Αν για κάποιο λόγο υπάρχει αμφιβολία για το ποια είναι η ροή του κώδικα, μπορεί να γίνει ο ίδιος έλεγχος με το ζεύγος objdump και εξομοιωτή tsim. Στην περίπτωση που τελικά διαπιστωθεί ότι υπάρχει ασυμφωνία, ξεκινάει η δημιουργία ενός νέου, όσο το δυνατόν μικρότερου δοκιμαστικού προγράμματος. Το πρόγραμμα αυτό έχει στόχο να φέρει το σύστημα στην ίδια ακριβώς προβληματική κατάσταση μπορώντας ταυτόχρονα να είναι αρκούντως μικρό, έτσι ώστε να μπορεί να εκτελεστεί στο Modelsim σε ένα εύλογο χρονικό διάστημα. Για το λόγο αυτό, στο νέο πρόγραμμα σχεδόν κόβονται όλες οι χρήσεις τον περιφερειακών εξόδου. Αξίζει να σημειωθεί ότι η εξομοίωση ενός προγράμματος λίγων μόνο Kbytes σε ένα Pentium 4 3.0 Ghz μπορεί να διαρκέσει πάνω από μισή ώρα[1]. Με την εξομοίωση με το Modelsim είναι πλέον δυνατό να παρατηρηθεί η ακριβής λειτουργία του κυκλώματος, να εντοπιστούν τα σήματα που παράγονται, να προσδιοριστούν ποια από αυτά εμπλέκονται στο συγκεκριμένο προς επίλυση πρόβλημα και εν τέλει να γίνουν όλες οι απαραίτητες διορθώσεις στο κύκλωμα.

[1] Ο χρόνος αυτός είναι ενδεικτικός για πολύ απλά προγράμματα. Αν το εξεταζόμενο πρόγραμμα περιέχει βρόχους, όπως γίνεται π.χ με την αρχικοποίηση των πινάκων σελίδων στα δοκιμαστικά προγράμματα για τη μονάδα διαχείρισης μνήμης, ο χρόνος εκτέλεσης μεγαλώνει πάρα πολύ

Όσον αφορά την αλληλεπίδραση της μονάδας διαχείρισης μνήμης με τις γρήγορες μνήμες, το πρόβλημα που παρουσιάστηκε ήταν ότι, όταν γέμιζε η γρήγορη μνήμη εντολών, η MMU προκαλούσε την ανάγνωση της επόμενης εντολής από αυτήν που έδειχνε ο μετρητής προγράμματος του επεξεργαστή. Το πρόβλημα αυτό αντιμετωπίστηκε με τη μέθοδο που περιγράφηκε παραπάνω.

Τα επόμενα πειράματα που έχουν σχεδιαστεί για την εξασφάλιση της ορθής λειτουργίας της μονάδας διαχείρισης μνήμης είναι αρκετά πιο πολύπλοκα και προκαλούν την ταυτόχρονη λειτουργία και συνεργασία πολλών δομικών μονάδων του συστήματος. Έτσι έχουμε συνδυασμένη λειτουργία της μονάδας διαχείρισης μνήμης, του συστήματος διακοπών του επεξεργαστή, των γρήγορων μνημών και του ελεγκτή άμεσης πρόσβασης στη μνήμη. Μερικά από τα πειράματα αυτά φέρνουν επίτηδες το σύστημα σε πλέον μη λειτουργική κατάσταση, για να ελεγχθεί και παρατηρηθεί η συμπεριφορά του συστήματος σε προβληματικές καταστάσεις και αν ακολουθεί τις προδιαγραφές. Μια τέτοια κατάσταση μπορεί να προκληθεί παραδείγματος χάριν με μια εσφαλμένη αρχικοποίηση των πινάκων σελίδων και των καταλόγων πινάκων σελίδων.

Κατά τη λειτουργία της μονάδας διαχείρισης μνήμης είναι δυνατό να προκληθούν διακοπές στις εξής γενικές περιπτώσεις:

- σε πρόσβαση δεδομένων (data access)
- σε πρόσβαση εντολών (instruction access)
- σε σφάλμα μετάφρασης (translation error)

Οι δύο πρώτες περιπτώσεις καλύπτουν το μεγαλύτερο μέρος των διακοπών που εμφανίζονται στην πράξη και σε συνθήκες καλής λειτουργίας. Τα πειράματα που χειρίζονται διακοπές που προκαλούνται από τη μονάδα διαχείρισης μνήμης ακολουθούν τη φιλοσοφία των προηγούμενων πειραμάτων. Έχουν δηλαδή σχεδιαστεί με τέτοιο τρόπο ώστε να απομονώνουν τις αιτίες πρόκλησης σφαλμάτων και διακοπών. Έτσι πραγματοποιούν μια σειρά σταδιακών ελέγχων για τις ακόλουθες περιπτώσεις:

- αστοχία δεδομένων (data miss)
- αστοχία εντολών (instruction miss)
- εσφαλμένη αρχικοποίηση πινάκων σελίδων
- προστασία σελίδων βάσει δικαιωμάτων πρόσβασης
- πρόκληση εξωτερικής διακοπής
- αλλαγή διεργασίας (context switch)

- διακοπή με εντολή διπλής ανάγνωσης/εγγραφής
- πρόσβαση με παράκαμψη της μονάδας διαχείρισης μνήμης (MMU bypass)
- όλα τα παραπάνω με ενεργοποιημένες τις γρήγορες μνήμες του επεξεργαστή

Σε περίπτωση αστοχίας δεδομένων ή αστοχίας εντολών, προκαλούνται αντίστοιχα οι διακοπές με αριθμούς 0x09 και 0x01 αντίστοιχα [2]. Στην περίπτωση αυτή, η ρουτίνα εξυπηρέτησης της διακοπής οφείλει να φορτώσει στους αρμόδιους καταχωρητές της μονάδας διαχείρισης μνήμης τις σωστές τιμές, έτσι ώστε επιστρέφοντας από τη διακοπή, να συνεχιστεί κανονικά η εκτέλεση του προγράμματος. Αυτό στην πράξη το κάνει το λειτουργικό σύστημα. Στην περίπτωση μας το έκανε το κάθε δοκιμαστικό πρόγραμμα, ανάλογα με τη συμπεριφορά που ήθελε να ελέγξει. Στην περίπτωση εσφαλμένης αρχικοποίησης σελίδων το πιθανότερο είναι ότι μετά από μια σειρά ανεπιτυχών αναζητήσεων από τη μονάδα διαχείρισης μνήμης ο επεξεργαστής θα έρθει σε κατάσταση λάθους. Αυτό θα γίνει επειδή λόγω τελικά εσφαλμένης μετάφρασης ο επεξεργαστής θα κληθεί να εκτελέσει άγνωστη εντολή.

Κατά την εκτέλεση των πειραμάτων παρατηρήθηκαν προβλήματα στη δημιουργία των διακοπών. Συγκεκριμένα, στην αρχική υλοποίηση της μονάδας διαχείρισης μνήμης, η περίπτωση αστοχίας δεδομένων προκαλούσε αφενός διακοπή με διαφορετική τιμή από την τιμή αναφοράς και αφετέρου με καθυστέρηση δύο εντολών. Επίσης παρατηρήθηκε ασυμφωνία μεταξύ του αναμενόμενου και του πραγματικού περιεχομένου των bitfields του καταχωρητή διεύθυνσης σφάλματος της μονάδας διαχείρισης μνήμης.

Μετά την ολοκλήρωση ενός εκτεταμένου συνόλου δοκιμών και διορθώσεων, το επόμενο στάδιο ήταν η δοκιμή της εκτέλεσης του πυρήνα του Linux στο σύστημα. Όπως ήταν αναμενόμενο και αποδείχτηκε και στην πράξη, αυτό ήταν το δυσκολότερο στάδιο, διότι οποιοδήποτε σφάλμα μπορούσε να οφείλεται σε πρόβλημα του υλικού (όχι απαραίτητα μόνο της μονάδας διαχείρισης μνήμης), σε πρόβλημα της μεταφοράς (port) του linux, ή και στα δύο. Κάτω απ΄ αυτές τις συνθήκες, χωρίς δηλαδή την ύπαρξη μιας επιβεβαιωμένης ως προς την ορθή της λειτουργία βάσης, η διεξαγωγή οποιωνδήποτε πειραμάτων απαιτούσε εκτεταμένη προσοχή. Αξίζει να σημειωθεί ότι ο χρόνος μιας δοκιμής του linux στην πλατφόρμα με το FPGA ανέρχεται στα 25 με 30 λεπτά περίπου. Ο μεγάλος αυτός χρόνος οφείλεται αφενός στον χαμηλό χρονισμό του συστήματος (15 MHz) και στην απουσία κάποιας μόνιμης μνήμης. Ο πυρήνας που χρησιμοποιήθηκε ήταν ο 2.6.5 και ήταν ο νεότερος που κυκλοφορούσε τότε.

[2] Η ακριβής αρίθμηση των διακοπών καθορίζεται από τις προδιαγραφές του εκάστοτε επεξεργαστή. Οι τιμές αναφοράς περιγράφονται στο κείμενο The Sparc Architecture Manual Version 8. Η συγκεκριμένη υλοποίηση χρησιμοποιεί τις τιμές αναφοράς

Η χρήση του linux για τον έλεγχο της καλής λειτουργίας του συστήματος έχει το πλεονέκτημα ότι αποτελεί ένα περιβάλλον ελέγχου πραγματικών συνθηκών και ότι μπορεί να ελέγξει τη συμπεριφορά ολόκληρου του συστήματος κάτω από συνθήκες εκτεταμένου φόρτου του επεξεργαστή. Σε μια τέτοια περίπτωση, κι ενώ το σύστημα λειτουργούσε φαινομενικά καλά, μετά από κάποιο χρόνο λειτουργίας παρατηρήθηκαν κάποια περίεργα συμπτώματα, όπως ήταν η στοχαστική αέναη είσοδος σε κάποια τυχαία κάθε φορά ρουτίνα. Αυτό είχε σαν αποτέλεσμα να διατηρείται κάποια επικοινωνία με την κονσόλα, δίνοντας έτσι ένδειξη ότι ο το λειτουργικό συνεχίζει αν εκτελείται αφού τύπωνε την είσοδο στην έξοδο, δε συνέχιζε όμως η εκτέλεση του φλοιού. Ο εντοπισμός του προβλήματος ήταν πολύ δύσκολος. Ακόμα και με εκτύπωση των αποτελεσμάτων των μεταφράσεων της μονάδας διαχείρισης μνήμης, δε φαινόταν να υπάρχει κάποιο πρόβλημα ή κάποια λάθος μετάφραση, γιατί ο κύκλος επανεισόδου στην ρουτίνα ήταν τεράστιος. Συνεπώς, ο όγκος των δεδομένων που τυπώνονταν στην οθόνη ήταν απαγορευτικός για τη χρήση τους για σύγκριση αποτελεσμάτων και εξαγωγή κάποιου συμπεράσματος. Χρησιμοποιώντας όμως εκτέλεση ανά μία εντολή με παύση για τον εντοπισμό του προβλήματος, παρατηρήθηκε ότι η συμπεριφορά του συστήματος ήταν σωστή και σε καμία περίπτωση δεν παρατηρήθηκε το παραπάνω πρόβλημα. Αυτό, σε συνδυασμό με την στοχαστικότητα της εμφάνισης του παραπάνω προβλήματος, αποτελούσαν μια σοβαρή ένδειξη ότι υπήρχε κάποιο πρόβλημα με το συγχρονισμό των δομικών μονάδων του chip. Μετά από πολύωρες δοκιμές με τυχαίες διακοπές της εκτέλεσης για τον εντοπισμό κάποιου λάθους με τη μονάδα αποσφαλμάτωσης, εντοπίστηκε το πρόβλημα, το οποίο ήταν αλλοίωση της διεύθυνσης επιστροφής μετά από εξυπηρέτηση διακοπής της μονάδας διαχείρισης μνήμης.

Με την διόρθωση του παραπάνω προβλήματος, το σύστημα ήταν λειτουργικό και πλέον στα8σταθερό. Έτσι, το linux μπορούσε να χρησιμοποιηθεί σαν βάση για την διεξαγωγή άλλων πειραμάτων και τον έλεγχο περιφερειακών, όπως για παράδειγμα η μονάδα κρυπτογράφησης.

3.4 Έλεγχος μονάδων δικτύου MAC

Στην ενότητα αυτή περιγράφεται ο έλεγχος που έγινε για τις ενσωματωμένες μονάδες δικτύου. Ο έλεγχος των μονάδων αυτών ξεκίνησε μετά την μεταφορά του πυρήνα του linux στο REGATE. Για τον έλεγχο αυτό χρησιμοποιήθηκαν παραλλαγές των γνωστών από τα προηγούμενα μικρών αυτόνομων προγραμμάτων, το micromonitor, και ένας στοιχειώδης οδηγός για linux. Κάθε ένα από τα τρία αυτά περιβάλλοντα εξυπηρετούσε κι ένα διαφορετικό και πιο πλήρη έλεγχο.

Τα μικρά αυτόνομα προγράμματα ελέγχουν την πολύ βασική λειτουργία των

μονάδων. Αυτή περιλαμβάνει την σωστή εγγραφή και ανάγνωση των καταχωρητών των μονάδων και τη συμπεριφορά των διακοπών. Οι τελευταίες παράγονται μόνο όταν χρησιμοποιείται ο ελεγκτής άμεσης πρόσβασης στη μνήμη. Στις διακοπές ελέγχεται η χρονική στιγμή παραγωγής τους και η επιστροφή από αυτές.

Με την εξασφάλιση της στοιχειώδους λειτουργικότητας των μονάδων ακολουθεί ο έλεγχος με το micromonitor. Το micromonitor είναι ένα περιβάλλον που ξεκινάει το σύστημα και παρέχει μια κονσόλα με κάποιες πολύ βασικές λειτουργίες, όπως για παράδειγμα έλεγχο και εγγραφή διευθύνσεων μνήμης. Για τον έλεγχο των μονάδων στο micromonitor χρησιμοποιήθηκε ένας οδηγός, που αρχικοποιούσε τα MAC, σε συνδυασμό με μια μικρή υλοποίηση του πρωτοκόλλου TCP/IP. Στόχος των ελέγχων σε αυτό το στάδιο είναι η επιβεβαίωση της σωστής μεταφοράς δεδομένων, τόσο με χρήση των διακοπών του ελεγκτή όσο και χωρίς. Στη δεύτερη περίπτωση, ο επεξεργαστής ελέγχει κατά τακτά χρονικά διαστήματα τους κατάλληλους καταχωρητές (polling). Για πληρέστερο έλεγχο, οι γειτονικές περιοχές των buffers που χρησιμοποιούνται για την αποστολή και λήψη δεδομένων συμπληρώνονται με κάποιες συγκεκριμένες τιμές. Μετά από κάθε αποστολή και λήψη δεδομένων, ελέγχονται αυτές οι περιοχές, έτσι ώστε να επαληθευτεί ότι ο ελεγκτής άμεσης πρόσβασης στη μνήμη λειτουργεί σωστά και δε καταστρέφει τα περιεχόμενα των γειτονικών περιοχών μνήμης.

Το τρίτο μέρος περιλαμβάνει τον έλεγχο με το linux. Στην περίπτωση αυτή ελέγχεται η συμπεριφορά των μονάδων δικτύου όταν λειτουργεί παράλληλα και η μονάδα διαχείρισης μνήμης. Η ροή των δεδομένων ακολουθεί σ΄ αυτή την περίπτωση ένα αρκετά μεγάλο μονοπάτι και θέτει σε λειτουργία ένα μεγάλο αριθμό δομικών στοιχείων του συστήματος. Στον έλεγχο με το linux χρησιμοποιήθηκαν τόσο πακέτα TCP όσο και UDP. Σε πρώτο στάδιο χρησιμοποιήθηκε ένας απλός οδηγός που χρησιμοποιούσε ένα μόνο buffer. Αυτό δεν αποτελούσε πρόβλημα, καθώς στόχος του οδηγού στη φάση αυτή ήταν η ανάδειξη της ορθής λειτουργίας του συστήματος και όχι η βέλτιστη απόδοση. Στη συνέχεια, όπως περιγράφεται στο επόμενο κεφάλαιο, σχεδιάστηκε και υλοποιήθηκε ένας νέος οδηγός, ο οποίος εκμεταλλεύεται πλήρως την αρχιτεκτονική του συστήματος και οδηγεί σε βέλτιστα αποτελέσματα.

Αξίζει να σημειωθεί ότι στην περίπτωση των μονάδων MAC, όπως επίσης και στην περίπτωση της μονάδας κρυπτογράφησης, γίνεται παράλληλα έλεγχος του ελεγκτή άμεσης πρόσβασης στη μνήμη. Ο έλεγχος του τελευταίου δε μπορεί να είναι αυτόνομος, καθώς η λειτουργία του είναι άμεσα συνδεδεμένη με κάποιο περιφερειακό Ο ελεγκτής αυτός υποστηρίζει την ταυτόχρονη λειτουργία δεκαέξι ζευγαριών καναλιών εισόδου εξόδου. Κατά την περίοδο δοκιμών με το FPGA, λόγων περιορισμών του τελευταίου, ήταν αδύνατος ο έλεγχος πολλών περιφερειακών ταυτόχρονα.

3.5 Μονάδα κρυπτογράφησης

Η μονάδα κρυπτογράφησης είναι ένα δομικό στοιχείο του συστήματος που αναλαμβάνει την επιτάχυνση εκτέλεσης αλγορίθμων κρυπτογράφησης / αποκρυπτογράφησης τύπου DES. Η λειτουργία του χωρίζεται σε ένα σύστημα ελέγχου και σε ένα σύστημα διαχείρισης δεδομένων. Η μεταφορά των κρυπτογραφημένων /αποκρυπτογραφημένων δεδομένων γίνεται με άμεση πρόσβαση στη μνήμη.

Ο έλεγχος αυτής της μονάδας ξεκινάει από το σύστημα ελέγχου. Ανάλογα με τις τιμές των πεδίων ενός καταχωρητή, η μονάδα κρυπτογράφησης επιλέγει τον αλγόριθμο που θα χρησιμοποιηθεί. Στην περίπτωση αλγορίθμου CBC επιλέγεται αν η αρχικοποίηση του αλγορίθμου θα γίνεται εσωτερικά από τη μονάδα κρυπτογράφησης ή θα δίνεται εξωτερικά. Με μικρά και γρήγορα ως προς το χρόνο εκτέλεσης τους προγράμματα γίνεται ο έλεγχος της ορθής λειτουργίας του καταχωρητή αυτού. Ελέγχεται ακόμα αν οι διακοπές που παράγονται με τη βοήθεια του ελεγκτή άμεσης πρόσβασης στη μνήμη λειτουργούν σωστά.

Στη συνέχεια, ελέγχονται τα αποτελέσματα της εκτέλεσης των αλγορίθμων από το υλικό. Για τη σύγκριση χρησιμοποιούνται τριάδες αρχείων. Το πρώτο αρχείο παράγει το δεύτερο με χρήση κάποιου από τους διαθέσιμους αλγόριθμους κρυπτογράφησης. Το τρίτο παράγεται από το δεύτερο αποκρυπτογράφοντας το δεύτερο. Το πρώτο αρχείο πρέπει να είναι πανομοιότυπο με το τρίτο. Η συνθήκη αυτή όμως δεν είναι ικανή για τη διεξαγωγή ασφαλών συμπερασμάτων. Λόγω της συμμετρίας των αλγορίθμων, είναι δυνατό ενώ το τελικό αρχείο να συμπίπτει με το αρχικό, το κρυπτογραφημένο να είναι λάθος. Για το λόγο αυτό τα αποτελέσματα συγκρίνονται με αυτά που παράγει για το ίδιο αρχείο με τα ίδια κλειδιά κωδικοποίησης η βιβλιοθήκη OpenSSL. Ο έλεγχος γίνεται τόσο σε σύστημα μεγάλης ακρικότητας (big endianess), όπως για παράδειγμα το REGATE, όσο και σε σύστημα μικρής ακρικότητας (little endianess), όπως ένας συνηθισμένος προσωπικός υπολογιστής.

Καθώς η αρχιτεκτονική της μονάδας κρυπτογράφησης υποστηρίζει την αποθήκευση κλειδιών κρυπτογράφησης για δεκαέξι ταυτόχρονες διεργασίες κρυπτογράφησης, οι παραπάνω έλεγχοι έγιναν εξαντλώντας αυτό το όριο. Επιπροσθέτως, χρησιμοποιώντας βοηθητικό αποθηκευτικό χώρο από την SDRAM του συστήματος, χρησιμοποιήθηκαν μέχρι 48 ταυτόχρονες διεργασίες κρυπτογράφησης, απορροφώντας έτσι τους πόρους του συστήματος στο μέγιστο. Με τον τρόπο αυτό διαπιστώθηκε ότι το σύστημα μπορεί να λειτουργεί απρόσκοπτα σε τέτοιες συνθήκες και ότι λογικά δεν πρέπει να υπάρχει κάποιο πρόβλημα στη λογική σχεδίαση του κυκλώματος. Στην αντίθετη περίπτωση, ένα πρόβλημα παραδείγματος χάριν χρονισμού ή κακής διεπαφής μεταξύ δύο δομικών στοιχείων του συστήματος, συνήθως εντοπίζεται αμέσως σε αυτές τις συνθήκες λειτουργίας.

Κεφάλαιο 4

Το Linux στον LEON

Στο κεφάλαιο αυτό περιγράφεται η διαδικασία μεταφοράς (porting) του λειτουργικού συστήματος Linux. Περιγράφονται τα εργαλεία που χρησιμοποιήθηκαν για το σκοπό αυτό, τα σημεία του πυρήνα που χρειάστηκαν αλλαγές, η υλοποίηση οδηγών συσκευών (device drivers) για τις ενσωματωμένες δομικές μονάδες του συστήματος (embedded blocks) και ορισμένες εφαρμογές που μεταφέρθηκαν στην πλατφόρμα REGATE. Επιπλέον καταδεικνύονται οι λόγοι οι οποίοι οδήγησαν στην επιλογή του GNU Linux.

4.1 Επιλογή λειτουργικού συστήματος

Τα τελευταία χρόνια, με την ραγδαία ανάπτυξη των ενσωματωμένων συστημάτων, υπάρχει μια μεγάλη άνθηση στον τομέα του σχεδιασμού και της υλοποίησης κατάλληλων λειτουργικών συστημάτων. Υπάρχουν πάρα πολλοί τύποι λειτουργικών συστημάτων που μπορεί να διαφέρουν ελάχιστα ή ριζικά. Αυτό είναι αποτέλεσμα της εξειδικευμένης συνήθως χρήσης των ενσωματωμένων συστημάτων, που στοχεύουν σε πολλές διαφορετικές εφαρμογές. Υπάρχουν συστήματα πραγματικού χρόνου, μονολιθικά, πολλαπλών διεργασιών, πολλαπλών χρηστών κτλ. Στη σημερινή αγορά τα δυο κυρίαρχα λειτουργικά συστήματα είναι τα Windows CE και το Embedded Linux.

Στην περίπτωση του REGATE επιλέχθηκε το GNU Linux. Η επιλογή αυτή βασίστηκε σε τρεις βασικούς άξονες:

- εφαρμογές που θα εξυπηρετεί το όλο σύστημα (συνδυασμός υλικού και λογισμικού)
- Σταθερότητα, ασφάλεια και αξιοπιστία

- κόστος και χρόνος υλοποίησης

Η πλατφόρμα REGATE έχει ως στόχο να εξυπηρετεί ενοποιημένες δικτυακές υπηρεσίες. Αυτό καθιστά άμεση την ανάγκη της βέλτιστης εξυπηρέτησης πολλών διεργασιών από πολλούς χρήστες με τη μέγιστη δυνατή αξιοποίηση και κατανομή των πόρων του συστήματος. Επιπλέον υπάρχει η ανάγκη της επικοινωνίας μεταξύ διαφορετικών δικτυακών υπηρεσιών μέσω κατάλληλων στρωμάτων. Για παράδειγμα, είναι επιθυμητή η δυνατότητα τηλεφωνίας πάνω από δίκτυο, η παροχή εικόνας μέσω του δικτύου, ο απομακρυσμένος έλεγχος οικιακών συσκευών κτλ. Το linux παρέχει τη δυνατότητα εξυπηρέτησης τέτοιων αναγκών με μεγάλη επάρκεια και ταχύτητα.

Πολύ σημαντικό στοιχείο για την επιλογή του Linux αποτελεί το γεγονός ότι και το λειτουργικό σύστημα και η πλατφόρμα είναι εύκολα μεταβαλλόμενα και παραμετροποιήσιμα. Συνεπώς, από τη στιγμή που η πλατφόρμα μπορεί να αλλάξει και να πάρει ποικίλες μορφές, είναι φανερό ότι το λειτουργικό σύστημα πρέπει να μπορεί να ακολουθεί και να προσαρμόζεται στις μεταβολές αυτές. Ο πυρήνας του Linux , λόγω της δομής του και της υποστήριξης των δομοστοιχείων (modules), αποτελεί το πλέον κατάλληλο λειτουργικό σύστημα για το σκοπό αυτό.

Η σταθερότητα ενός ενσωματωμένου συστήματος είναι μια από τις παραμέτρους που καθορίζουν την επιτυχία ή την αποτυχία του. Το Linux είναι ένα από τα πιο σταθερά συστήματα. Αυτό οφείλεται στον ανοιχτό κώδικα που δίνει τη δυνατότητα στον προγραμματιστή να ξέρει ακριβώς πως λειτουργεί ο πυρήνας του λειτουργικού και να σχεδιάσει και να υλοποιήσει ανάλογα τους οδηγούς των συσκευών του συστήματος. Ακόμα περισσότερο, ο πηγαίος κώδικας είναι διαθέσιμος σε μια πολύ μεγάλη προγραμματιστική κοινότητα, με αποτέλεσμα η ανάπτυξη και η βελτίωση του να είναι ταχύτατες.

Χρησιμοποιώντας το Linux η προστασία του συστήματος από πιθανές επιθέσεις και εισβολές είναι πολύ επαρκής. Υπάρχουν πολλά αξιόπιστα και πολύ καλά δοκιμασμένα εργαλεία για την αποτελεσματική προστασία και ασφάλεια του συστήματος, για την παροχή πρόσβασης στο διαδίκτυο και σε τοπικό δίκτυο και για την ασφαλή επικοινωνία με κωδικοποίηση. Μερικά από τα πιο γνωστά από αυτά είναι ο εξυπηρετητής δικτύου Apache, το πακέτο openssh για ασφαλή επικοινωνία και ο εξυπηρετητής proxy squid.

Το τελικό κόστος παίζει πολύ σημαντικό ρόλο στην υλοποίηση ενός ερευνητικού προγράμματος. Στην προκειμένη περίπτωση το Linux κερδίζει πολύ έδαφος σε σχέση με τα Windows CE, κυρίως λόγω του κόστους αδειοδότησης, τόσο του λειτουργικού συστήματος όσο και του ανάλογου προγραμματιστικού περιβάλλοντος. Στον προσδιορισμό του κόστους πρέπει επίσης να συμπεριληφθούν το κόστος σε ανθρώπινο δυναμικό, το κόστος σε χρόνο διεκπεραίωσης και το κόστος της τεχνικής υποστήριξης. Αν υπάρχει στόχος για μεγάλη παρα-

γωγή του τελικού προϊόντος, τότε το κόστος της άδειας χρήσης των windows γίνεται πολύ μεγάλο, καθώς χρειάζεται μια άδεια για κάθε συσκευή.

4.2 Εργαλεία ανάπτυξης λογισμικού

Για την μεταφορά του Linux στον LEON αλλά και ποικίλου λογισμικού, πχ. OpenSSL, χρησιμοποιήθηκαν τα παρακάτω εργαλεία:

- Οι επεξεργαστές κειμένου vi και emacs.
- Ο μεταγλωττιστής (compiler) sparc-linux-gcc 3.2.2
- GNU binutils
- GNU emacs
- Ο εξομοιωτής tsim
- Το εργαλείο dsumon, που συνδέει τη μονάδα DSU με ένα PC.

Η ανάπτυξη του λογισμικού έγινε σε ένα PC αρχιτεκτονικής x86 με Debian GNU/Linux.

4.2.1 Τα GNU binutils

Τα binutils αποτελούν μια συλλογή από εργαλεία για δυαδικά αρχεία. Τα προγράμματα που συνιστούν αυτή τη συλλογή είναι:

- **ld** - Ο GNU συνδέτης (linker)
- **as** - Ο GNU συμβολομεταφραστής (assembler)
- **addr2line** - μετατρέπει διευθύνσεις σε ονόματα αρχείων και αριθμούς γραμμών
- **ar** - ένα εργαλείο για τη δημιουργία, αλλαγή και εξαγωγή αρχειοθηκών (archives)
- **c++filt** - είναι ένα φίλτρο που μεταφράζει κωδικοποιημένα σύμβολα C++
- **gprof** - προβάλλει περιγραμματικές πληροφορίες

- **nmlconv** - μετατρέπει κώδικα αντικειμένου σε NLM
- **nm** - δίνει λίστα συμβόλων από αρχεία αντικειμένου (object files)
- **objcopy** - αντιγράφει και μετατρέπει αρχεία αντικειμένου
- **objdump** - τυπώνει πληροφορίες από αρχεία αντικειμένου
- **ranlib** - Δημιουργεί ένα ευρετήριο για τα περιεχόμενα μιας αρχειοθήκης
- **readelf** - Προβάλλει πληροφορίες από οποιοδήποτε αρχείο αντικειμένου τύπου ELF
- **size** - Παραθέτει τα μεγέθη των τμημάτων (sections) μιας αρχειοθήκης ή ενός αρχείου αντικειμένου
- **strings** - Παραθέτει τις συμβολοσειρές (strings) που υπάρχουν σε κάποιο αρχείο
- **strip** - Αφαιρεί σύμβολα από ένα αρχείο
- **windres** - Ένας μεταγλωττιστής για πηγαία αρχεία για Windows

Τα πιο πολλά απ' αυτά χρησιμοποιούν την βιβλιοθήκη BFD (Binary File Descriptor) για τη διαχείριση εργασιών χαμηλού επιπέδου και τη βιβλιοθήκη opcodes για τη συμβολομετάφραση και την αντίστροφη συμβολομετάφραση (disassembly) εντολών μηχανής.

Όλα τα προγράμματα που είναι διαθέσιμα για το REGATE είναι διασταυρωτά μεταγλωττισμένα (cross compiled) και συνδεδεμένα (cross linked). Για το σκοπό αυτό χρησιμοποιήθηκε η αλυσίδα εργαλείων (toolchain) sparc-linux. Αυτή η αλυσίδα εργαλείων πάρθηκε από την ιστοσελίδα www.gaisler.com της Gaisler Research. Τα εργαλεία που παρέχει είναι διασταυρωτά binutils για αρχιτεκτονική SPARC από μηχάνημα αρχιτεκτονικής x86, ο διασταυρωτός μεταγλωττιστής sparc-linux-gcc-3.2.2, και οι απαραίτητες βιβλιοθήκες.

4.2.2 Επεξεργαστές κειμένου

Για τη γραφή του απαραίτητου κώδικα για τη μεταφορά του linux και του υπόλοιπου λογισμικού στο REGATE χρησιμοποιήθηκαν οι επεξεργαστές κειμένου emacs και vi. Οι δύο αυτοί κειμενογράφοι αποτελούν δύο ισχυρότατα εργαλεία, κυρίως όσον αφορά την ανάπτυξη πηγαίου κώδικα. Μερικά από ένα πλήθος χρήσιμων χαρακτηριστικών που προσφέρουν είναι:

- Μεγάλος αριθμός ρυθμίσεων και ικανότητα προσαρμογής στην αρέσκεια του χρήστη, προσφέροντας έτσι μεγάλη εργονομία
- Δυνατότητα κατάλληλου χρωματισμού του κώδικα (highlight), καθιστώντας ευκολότερη τη γραφή αλλά και την ανάγνωση του κώδικα.
- Πλήθος χρήσιμων εργαλείων, όπως αριθμομηχανή με δυνατότητα μετατροπής αριθμητικού συστήματος

Περισσότερες πληροφορίες για τους κειμενογράφους αυτούς υπάρχουν στις αντίστοιχες ιστοσελίδες:
http://www.gnu.org/software/emacs/emacs.html
http://www.vim.org.

4.2.3 Τα εργαλεία tsim και dsumon

Τα προγράμματα dsumon και tsim [1] αποτελούν δύο εργαλεία για τον επεξεργαστή LEON-2 και παρέχονται από την Gaisler Research. Το dsumon παρέχει τη δυνατότητα σύνδεσης ενός τερματικού με τη μονάδα αποσφαλμάτωσης DSU (Debug Support Unit) του LEON. Η επικοινωνία επιτυγχάνεται μέσω μιας σειριακής γραμμής, που χρησιμοποιείται μόνο από τη μονάδα αποσφαλμάτωσης. Η ταχύτητα της γραμμής μπορεί να ρυθμιστεί κατάλληλα, προσφέροντας ταχύτητες από 9600 μέχρι 460800 bps. Τα κύρια χαρακτηριστικά του dsumon είναι:

- Πρόσβαση ανάγνωσης/γραφής σε όλους τους καταχωρητές του LEON και στη μνήμη
- Ενσωματωμένος αντίστροφος συμβολομεταφραστής και διαχείριση ενδιάμεσης αποθηκευτικής μνήμης ανίχνευσης (trace buffer)
- Μεταφόρτωση και εκτέλεση εφαρμογών για τον LEON
- Διαχείριση σημείων διακοπής (breakpoints) και παρατήρησης (watchpoint)
- Απομακρυσμένη σύνδεση με τον αποσφαλματωτή gdb
- Εντολές ορισμένες από τον χρήστη

[1] Στα πλαίσια του REGATE χρησιμοποιήθηκαν οι εκδόσεις εκτίμησης των προγραμμάτων αυτών, που η χρήση τους είναι ελεύθερη.

- Δομοστοιχεία (modules) για σημεία ελέγχου (checkpoints), περιγραμματικές πληροφορίες και είσοδο/έξοδο ορισμένες από το χρήστη

- Προγραμματισμός μνήμης Flash

Το πρόγραμμα tsim είναι ένας εξομοιωτής επιπέδου εντολών που μπορεί να εξομοιώσει υπολογιστικά συστήματα βασισμένα στον επεξεργαστή LEON. Το tsim παρέχει:

- Ακριβή και πραγματικού κύκλου εξομοίωση του επεξεργαστή LEON

- Εξαιρετική απόδοση: μέχρι 30 MIPS σε ισχυρό PC (XEON@3.2GHz)

- Επιταχυμένη κατάσταση ηρεμίας του επεξεργαστή που επιτρέπει ταχύτερη εκτέλεση εξομοίωσης από την πραγματική λειτουργία

- Αυτόνομη λειτουργία και απομακρυσμένη σύνδεση στον GNU αποσφαλματωτή

- 64-bit μέτρηση χρόνου για απεριόριστο χρόνο εξομοίωσης

- Φορτώσιμα δομοστοιχεία για την υποστήριξη ορισμένης από το χρήστη συσκευής εισόδου/εξόδου

- Βοηθητική μνήμη ανίχνευσης εντολών

- Πίσω ανίχνευση της στοίβας με πληροφορίες για τα σύμβολα

- Μη εισβάλλουσα εκτέλεση σχεδίασης κατατομής χρόνου

- Απεριόριστος αριθμός σημείων διακοπής και παρατήρησης

- Δυνατότητα χρήσης σημείων ελέγχου για την αποθήκευση και επανάκτηση της πλήρους κατάστασης του εξομοιωτή

- 'Ελεγχο κάλυψης κώδικα

Τα δύο αυτά προγράμματα χρησιμοποιήθηκαν κατά κόρον κατά την διάρκεια της μεταφοράς του πυρήνα του GNU/Linux στον LEON και στην εκτέλεση και παρατήρηση ενός πλήθους δοκιμαστικών προγραμμάτων.

4.3 Μεταφορά του Λ.Σ. GNU/Linux

Το GNU/Linux είναι ένα πολύ σταθερό και ταχύτατο λειτουργικό σύστημα (Λ.Σ.), εύκολα προσαρμόσιμο και επεκτάσιμο, και αποτελεί έτσι μια ιδανική λύση για εφαρμογές ενσωματωμένων συστημάτων. Η μεταφορά για το REGATE βασίστηκε σε μια προϋπάρχουσα μεταφορά για τον επεξεργαστή LEON, από τον Stefan Holst. Οι κυριότερες διαφορές εντοπίζονται στη διαδικασία εκκίνησης του συστήματος (boot process) και στο σύστημα διαχείρισης μνήμης. Επιπλέον, έχουν δημιουργηθεί εξαρχής δύο νέοι οδηγοί, ένας για μια πρωτότυπη ενσωματωμένη κάρτα δικτύου (MAC) και ένας για την ενσωματωμένη μονάδα κρυπτογράφησης (DES unit).

4.3.1 Προϋπάρχουσες μεταφορές του Linux για SPARC και LEON

Στην υποενότητα αυτή παρουσιάζεται μια ιστορική αναδρομή της μεταφοράς του Linux για τον επεξεργαστή SPARC αρχικά και για τον LEON στη συνέχεια.

Η μεταφορά σε SPARC

Η μεταφορά του GNU Linux σε υπολογιστικά συστήματα βασισμένα σε επεξεργαστές SPARC ξεκίνησε το 1994 από μια μικρή ομάδα προγραμματιστών, που επικοινωνούσαν και συννενοούνταν μεταξύ τους μέσω του διαδικτύου. Εκείνη την περίοδο το Linux έτρεχε σε μηχανήματα Intel 80386 και Alpha. Και οι δύο αυτές αρχιτεκτονικές είναι μικροακρικές (little endian), ενώ ο SPARC είναι μεγαλοακρικός (big endian). Ένα από τα πρώτα πράγματα που υλοποιήθηκαν ήταν να γίνει ο πυρήνας διαφανής ως προς τον ακρικό χαρακτήρα (endianess) της εκάστοτε αρχιτεκτονικής.

Στη συνέχεια έγιναν αλλαγές στο σύστημα αρχείων, έτσι ώστε τα ίδια συστήματα αρχείων να μπορούν να χρησιμοποιηθούν από μηχανήματα διαφορετικών αρχιτεκτονικών που τρέχουν Linux.

Λόγω της μεγάλης ποικιλίας από μονάδες διαχείρισης μνήμης στις διάφορες υλοποιήσεις SPARC μηχανημάτων, χρησιμοποιήθηκε τέτοιος σχεδιασμός για την διαχείριση της μνήμης, ώστε να δίνει τη δυνατότητα στον πυρήνα να υποστηρίζει όλες αυτές τις διαφορετικές υλοποιήσεις. Οι διάφορες απαραίτητες αρχικοποιήσεις για κάθε τέτοια υλοποίηση δε γίνονται κατά τη μεταγλώττιση του πυρήνα, αλλά κατά την διαδικασία εκκίνησης του συστήματος. Με αυτή την τακτική επιτεύχθηκε αφενός η δημιουργία ενός μόνο πυρήνα για πολλές

πλατφόρμες και αφετέρου οι λεπτομέρειες των αρχικοποιήσεων μεταξύ των διαφορετικών υλοποιήσεων να είναι διαφανείς στον υπόλοιπο πυρήνα.

Η απευθείας πρόσβαση στη μνήμη (Direct Memory Access) επιτυγχάνεται με ένα μοναδικό τρόπο σε μηχανήματα που βασίζονται σε SPARC επεξεργαστές. Στα περισσότερα συστήματα αυτού του τύπου, οι DMA εγγραφές και αναγνώσεις γίνονται με χρήση φυσικών διευθύνσεων. Για τη διαχείριση της DMA πρόσβασης στο Linux υπάρχει μια καθορισμένη περιοχή διευθύνσεων που χρησιμοποιούνται για το σκοπό αυτό. Όταν πρόκειται να εξυπηρετηθεί μια αίτηση εγγραφής ή ανάγνωσης, οι απαραίτητες διευθύνσεις κλειδώνονται και ελευθερώνονται ξανά όταν διεκπεραιωθεί η ζητούμενη εργασία. Για την καλύτερη εκμετάλλευση και αποτελεσματικότητα του συστήματος, το πραγματικό μέγεθος της περιοχής των διευθύνσεων αυτών είναι μεταβλητό και μπορεί να μεγαλώσει όσο χρειάζεται. Μόλις τελειώσει η αυξημένη ζήτηση η περιοχή αυτή μικραίνει ξανά.

Η δημιουργία οδηγών για τις διάφορες συσκευές που χρησιμοποιούνται σε μια υλοποίηση SPARC ακολούθησε λίγο πολύ τα βήματα της αντίστοιχης διαδικασίας στην x86 αρχιτεκτονική. Στην αρχή, οι διάφοροι κατασκευαστές υλικού ήταν απρόθυμοι να δώσουν επαρκείς πληροφορίες και προδιαγραφές για τις συσκευές που παρήγαγαν, για την δημιουργία οδηγών στο Linux. Στη συνέχεια, με την γενικότερη εξάπλωση και αποδοχή του Linux, η εικόνα αυτή άλλαξε δραματικά. Έτσι σήμερα, κάποιες τεχνικές πολύ απαιτητικές σε χρόνο και γνώση, όπως πχ. η αντίστροφη μηχανική (reverse engineering), χρησιμοποιούνται πολύ λιγότερο απ΄ ότι παλιά και με τη βοήθεια των κατασκευαστριών εταιριών δημιουργούνται αποτελεσματικοί και αξιόπιστοι οδηγοί.

Το πρόγραμμα LEOX

Μια άλλη πολύ σημαντική συνεισφορά στη μεταφορά του Linux στην οικογένεια των SPARC υπολογιστών αποτελεί το πρόγραμμα LEOX. Στην περίπτωση του LEOX έχει γίνει μεταφορά του uClinux στον LEON. Το uClinux είναι ένα παράγωγο του πυρήνα 2.0 του Linux. Το κύριο χαρακτηριστικό αυτού του λειτουργικού είναι ότι προορίζεται για μηχανήματα που δε διαθέτουν μονάδα διαχείρισης μνήμης. Το uClinux, καθώς αναφέρεται σε μηχανήματα χωρίς MMU, έχει ορισμένες βασικές διαφορές από το κανονικό Linux, κυρίως ως προς τον τρόπο που υποστηρίζει τις πολλαπλές διεργασίες.

Αξίζει να σημειωθεί ότι το uClinux μεταφέρθηκε κι εκτελέστηκε επιτυχώς στο REGATE στα πλαίσια δοκιμών πριν την ολοκλήρωση της διαπίστευσης της καλής λειτουργίας της MMU, οπότε και άρχισε η μεταφορά του κανονικού Linux με πυρήνα 2.5.75. Οι κυριότερες αλλαγές που έγιναν στο uClinux όπως δίνεται από την ομάδα LEOX για να τρέξει στο REGATE, αφορούν τη διαδικασία εκκίνησης από την SDRAM απουσία μνήμης flash, τον οδηγό της σειριακής

θύρας και την εξομοίωση μονάδας κινητής υποδιαστολής.

Linux για LEON με MMU

Η μεταφορά η οποία αποτέλεσε και τη βάση για τη μεταφορά στο REGATE, ήταν αυτή του πυρήνα 2.5.75. Το έργο αυτό έγινε αρχικά από τον Jiri Gaisler και τον Konrad Eisele, που τροποποίησαν τη μεταφορά που υπήρχε για τον SPARC. Πάνω στη βάση αυτή κινήθηκε η προσπάθεια του Stefan Holst, ο οποίος μετέφερε αυτές τις τροποποιήσεις μαζί με κάποιες νέες βελτιώσεις στον πυρήνα 2.6.5. Οι αλλαγές αυτές είναι διαθέσιμες στην ιστοσελίδα http://www.ra.informatik.uni-stuttgart.de/ holstsn.

4.3.2 Διαχείριση μνήμης

Το κυριότερο πρόβλημα στη μεταφορά του Linux στο REGATE οφείλεται στη διαφορετική ανάθεση τιμών για τα ASIs. Η μονάδα διαχείρισης μνήμης που χρησιμοποιεί το REGATE δεν ακολουθεί την αρίθμηση που ορίζει το μοντέλο αναφοράς του SPARC V8. Ακολουθεί σε γενικές γραμμές την αρίθμηση που ορίζει για τον LEON η κατασκευάστρια εταιρία του επεξεργαστή, Gaisler Research. Ωστόσο υπάρχουν ορισμένες διαφορές στην αρίθμηση των ASIs από την MMU του REGATE και την MMU της Gaisler Research. Οι τιμές που ισχύουν στην αρίθμηση των ASIs στο REGATE απεικονίζονται στον πίνακα 4.1.

Το μοντέλο αναφοράς του SPARC V8 αφήνει κάποια παράθυρα διαφορετικής υλοποίησης της εκάστοτε μονάδας διαχείρισης μνήμης. Έτσι, η μονάδα διαχείρισης μνήμης του REGATE, χωρίς να αποκλίνει από το μοντέλο αναφοράς, χρησιμοποιεί μόνο καθολικό καθαρισμό για τον καθαρισμό των Translation Lookaside Buffers (TLBs) , ενώ το Linux δέχεται ότι υπάρχει δυνατότητα για καθαρισμό μιας σελίδας, ενός πίνακα σελίδων, ενός πλαισίου και όλης της μνήμης cache. Η διαφορά αυτή δημιουργούσε σοβαρά προβλήματα στην ευστάθεια και την ομαλή λειτουργία του συστήματος. Για τη λύση του προβλήματος έγιναν αλλαγές σε εκείνες τις χαμηλού επιπέδου συναρτήσεις, οι οποίες είναι υπεύθυνες για τον καθαρισμό της μνήμης cache της μονάδας διαχείρισης μνήμης.

Αυτή τη στιγμή, η έκδοση του πυρήνα του Linux που τρέχει στο REGATE είναι η 2.6.5. Η συγκεκριμένη μεταφορά του λειτουργικού συστήματος δεν υποστηρίζει δυναμική φόρτωση και αποφόρτωση δομοστοιχείων. Για να λειτουργήσει σωστά το σύστημα, πρέπει να μεταγλωττιστεί και να συνδεθεί με εξομοίωση μαθηματικού συνεπεξεργαστή. Αυτό ισχύει και για οποιοδήποτε

Περιγραφή	REGATE ASI
MMU registers	0x19
MMU flush/probe	0x13
MMU bypass	0x1c
MMU flush page	0x13
MMU flush region	0x13
MMU flush context	0x13
Total MMU flush	0x13
MMU diagnostics	0x1a
MMU diagnostics	0x1a

Πίνακας 4.1: Αρίθμηση των ASIs

λογισμικό που χρησιμοποιεί εντολές κινητής υποδιαστολής και προορίζεται να τρέξει στο REGATE.

4.3.3 Εκκίνηση του συστήματος

Η εκκίνηση του συστήματος γίνεται με τη βοήθεια της μονάδας αποσφαλμάτωσης και του dsumon. Καθώς δεν υπάρχει κάποιου είδους BIOS, οι βασικές παράμετροι του συστήματος πρέπει να ρυθμιστούν εξωτερικά. Στην προκειμένη περίπτωση, αυτό γίνεται με τη βοήθεια του dsumon. Με την εντολή *wmem* γράφονται στις κατάλληλες θέσεις μνήμης οι σωστές τιμές. Οι θέσεις αυτές αντιστοιχούν σε καταχωρητές του LEON (memory mapped registers). Στον πίνακα 4.2 παρουσιάζονται οι καταχωρητές αυτοί, μαζί με τις διευθύνσεις και τις τιμές τους. Οι καταχωρητές αυτοί ρυθμίζουν σημαντικές παραμέτρους για την μνήμη του συστήματος, όπως τον τύπο της μνήμης, το μέγεθός της και το χρονισμό της.

Στη συνέχεια, με την εντολή *load* του dsumon, μια "εικόνα" που περιλαμβάνει ένα συμπιεσμένο σύστημα αρχείων τύπου ROM και τον πυρήνα του λειτουργικού συστήματος φορτώνεται στη μνήμη RAM. Αυτό το στάδιο απαιτεί αρκετό χρόνο με μια συνήθη σειριακή θύρα που φτάνει μέχρι τα 115200 bps. Μπορεί όμως να επιταχυνθεί σημαντικά, έως και τέσσερις φορές, αν χρησιμοποιηθεί ένας μετατροπέας usb θύρας σε RS232, οπότε το κατέβασμα μπορεί να γίνει με ταχύτητα 460800 bps. Μόλις τελειώσει η διαδικασία αυτή το σύστημα

Καταχωρητής	**Διεύθυνση**	**Τιμή**	**Περιγραφή**
MEMCFG1	0x80000000	0x80000311	Memory Configuration Register 1
MEMCFG2	0x80000000	0x8222723f	Memory Configuration Register 2
MEMCFG3	0x80000000	0x00072000	Memory Configuration Register 3

Πίνακας 4.2: Καταχωρητές που αρχικοποιούνται στην εκκίνηση

είναι έτοιμο να ξεκινήσει το Linux. Αυτό γίνεται με με την εντολή *go*, ακολουθούμενη από την διεύθυνση από την οποία έχει καθοριστεί από τον συνδέτη ότι ξεκινάει η εκτέλεση του πυρήνα.

Στην πραγματικότητα ξεκινάει ένας εσωτερικός φορτωτής εκκίνησης (bootloader), ο οποίος εξομοιώνει μια PROM κατά τα πρότυπα του SPARC. Η PROM αυτή παρέχει πληροφορίες για τα χαρακτηριστικά του συστήματος, όπως τον τύπο και το ποσό της μνήμης RAM, και τις συσκευές που είναι συνδεδεμένες σε αυτό. Μετά από μια σειρά αρχικοποιήσεων ξεκινάει ο πυρήνας του linux. Ακολουθεί μετά ο εντοπισμός και η ταυτοποίηση του συστήματος αρχείων. Το τελευταίο αποσυμπιέζεται στη RAM του συστήματος και ακολουθεί η εκτέλεση της αρχικής διεργασίας με τη φόρτωση του φλοιού.

Υπάρχει κι ένας εναλλακτικός τρόπος εκκίνησης του συστήματος. Στην αρχή, ξεκινάει η εκτέλεση του micromonitor. Τόσο το micromonitor όσο και ο πυρήνας του Linux βρίσκονται αποθηκευμένα στη μνήμη FLASH. Μετά, με τη βοήθεια ενός φορτωτή εκτελέσιμων αρχείων τύπου ELF που κατασκευάστηκε για το σκοπό αυτό, φορτώνεται στη μνήμη RAM ο πυρήνας του Linux και ξεκινάει η εκτέλεσή του. Ο τρόπος αυτός εκκίνησης του Linux δεν ισχύει για το σύστημα με το FPGA.

Ο φλοιός και τα βασικότερα εργαλεία του συστήματος παρέχονται από το BusyBox [2]. Το BusyBox είναι μια εφαρμογή που συνδυάζει πολλά συνήθη Unix εργαλεία σε ένα μόνο μικρό εκτελέσιμο αρχείο. Για τον περιορισμό του μεγέθους του τελικού αρχείου, τα εργαλεία αυτά έχουν περικοπεί με τέτοιο τρόπο, έτσι ώστε να προσφέρουν ένα στοιχειώδες υποσύνολο με τις απολύτως βασικές λειτουργίες τους. Παρόλα αυτά, οι λειτουργίες που προσφέρουν παρέχουν σε ένα σύστημα επαρκή λειτουργικότητα. Το BusyBox αποτελεί έτσι μια ιδανική λύση για ενσωματωμένα συστήματα, αφού πετυχαίνει ένα πλήρως ικανοποιητικό συμβιβασμό μεταξύ του μεγέθους της εφαρμογής και της λειτουργικότητας που προσφέρει.

Ένα σημαντικό χαρακτηριστικό του BusyBox, που το καθιστά ιδανικό για ενσωματωμένα συστήματα, είναι η μεταβλητότητα και η προσαρμοστικότητα του.

[2] βλ. http://www.busybox.net

Ανάλογα με τις εκάστοτε ανάγκες, μπορεί να επιλεγεί ένα κατάλληλο σύνολο από εργαλεία, να προστεθούν ή να αφαιρεθούν κάποια, και να διατηρείται έτσι το μικρότερο δυνατό μέγεθος.

4.3.4 Ανάπτυξη οδηγών συσκευών

Για τις ανάγκες του REGATE δημιουργήθηκαν δύο οδηγοί συσκευών για Linux. Ο πρώτος απ΄ αυτούς αφορά την ενσωματωμένη μονάδα MAC για σύνδεση σε τοπικό δίκτυο. Ο δεύτερος οδηγός αναλαμβάνει την ενσωματωμένη μονάδα κρυπτογράφησης DES [3] αλγορίθμων. Και οι δύο οδηγοί χρησιμοποιούν τον ελεγκτή άμεσης πρόσβασης στη μνήμη (Direct Memory Access Controller, DMAC) για την επικοινωνία των αντίστοιχων μονάδων με τη κεντρικής μονάδας επεξεργασίας και τη φυσική μνήμη του συστήματος.

Οι οδηγοί συσκευών στο Linux χρησιμοποιούν ένα ειδικό σύστημα για την ταυτοποίηση τους. Το σύστημα αυτό περιλαμβάνει δύο αριθμούς, τον κύριο (major) και τον δευτερεύοντα (minor) αριθμό συσκευής. Κάθε κατασκευαστής μετά από σχετική αίτηση αποκτά ένα κύριο αριθμό και ένα σύνολο από δευτερεύοντες. Οι αριθμοί αυτοί είναι συγκεκριμένοι και μοναδικοί για κάθε συσκευή. Στην περίπτωση που οι συσκευές είναι ειδικές ή πειραματικές και προορίζονται για πολύ περιορισμένη χρήση, πχ. συσκευές που χρησιμοποιούνται σε ερευνητικά εργαστήρια και δε βγαίνουν στην αγορά, μπορεί να χρησιμοποιηθεί μια ειδική μικρή περιοχή από κύριους και δευτερεύοντες αριθμούς, που έχουν κρατηθεί γι΄ αυτόν ακριβώς τον σκοπό. Τόσο το DES όσο και το MAC ανήκουν στη δεύτερη κατηγορία. Έτσι το DES για παράδειγμα χρησιμοποιεί για κύριο αριθμό το 242 και για δευτερεύοντα το 0.

Δομή των μονάδων MAC, DES και DMAC

Για την καλύτερη κατανόηση του σχεδιασμού και της λειτουργίας των οδηγών για τις μονάδες MAC και DES, είναι απαραίτητη η περιγραφή [4] της δομής και της λειτουργίας των μονάδων αυτών και του ελεγκτή DMA.

Ο ελεγκτής DMA διαθέτει ένα αριθμό καναλιών, συγκεκριμένα 16, για την επικοινωνία και τη μεταφορά δεδομένων με τις διάφορες συσκευές που τον χρησιμοποιούν. Πέρα από κάποιους καταχωρητές που καθορίζουν κάποια γενικά χαρακτηριστικά του ελεγκτή, όπως π.χ. το χρονισμό του, και επηρεάζουν όλα

[3] Data Encryption Standard (DES), δημοσιεύτηκαν το 1977 από το Εθνικό Ινστιτούτο Τυποποίησης και Τεχνολογίας των Η.Π.Α.(NIST)

[4] Η περιγραφή που λαμβάνει χώρα σ΄ αυτήν την υποενότητα είναι σύντομη και στοιχειώδης. Λεπτομερέστερη περιγραφή υπάρχει στα αντίστοιχα εγχειρίδια των συσκευών και στα σχόλια του πηγαίου κώδικα

τα κανάλια, το κάθε κανάλι έχει μια δική του ομάδα καταχωρητών που επηρεάζουν τη λειτουργία αυτού και μόνο αυτού του καναλιού. Οι καταχωρητές της κάθε ομάδας, που ως προς τη δομή της είναι η ίδια σε όλα τα κανάλια, ορίζουν τη συμπεριφορά του καναλιού για τη δημιουργία διακοπών και την εξυπηρέτηση τους, παρέχουν στατιστικές πληροφορίες και τη δυνατότητα ελέγχου της λειτουργίας του καναλιού. Μέσα στην ομάδα των καταχωρητών συμπεριλαμβάνονται ένας καταχωρητής για την αποστολή και λήψη δεδομένων, που δέχεται δείκτες σε διευθύνσεις μνήμης που περιέχουν πακέτα δεδομένων, και ένας καταχωρητής που στην περίπτωση που το κανάλι λειτουργεί ως κανάλι αποστολής χρησιμοποιείται για να ορίσει το μέγεθος του πακέτου προς αποστολή. Συνήθως η κάθε συσκευή χρησιμοποιεί ένα ζεύγος καναλιών, ένα για είσοδο που συμβολίζεται με τη συντομογραφία rx και ένα για έξοδο tx.

Η μονάδα MAC για τη μεταφορά των δεδομένων χρησιμοποιεί δύο κανάλια του ελεγκτή DMA. Η αποστολή και λήψη δεδομένων γίνεται στέλνοντας δείκτες στις θέσεις μνήμης των πακέτων στα αντίστοιχα κανάλια του ελεγκτή DMA. Για τον εσωτερικό έλεγχο και τη ρύθμιση της συσκευής υπάρχει ένα πλήθος από καταχωρητές. Η μονάδα MAC, ανάλογα με τις τιμές των παραπάνω καταχωρητών υποστηρίζει λειτουργία σε κατάσταση half/full duplex και ακόμα κατάσταση loopback. Επίσης υποστηρίζει CRC έλεγχο και φιλτράρισμα των εισερχόμενων πακέτων.

Η μονάδα DES είναι ένα μέρος του συστήματος που αναλαμβάνει την εκτέλεση των απαιτητικών σε υπολογιστική δύναμη αλγορίθμων DES, δίνοντας έτσι τη δυνατότητα στην κεντρική μονάδα επεξεργασίας να προχωρήσει ταυτόχρονα στην εκτέλεση άλλων διεργασιών. Οι αλγόριθμοι που υποστηρίζονται είναι:

- DES-ECB
- DES-EDE3-ECB
- DES-CBC
- DES-EDE3-CBC

Ως προς τη διαδικασία αποστολής και λήψης δεδομένων η μονάδα DES χρησιμοποιεί την ίδια μέθοδο που χρησιμοποιεί και η μονάδα MAC. Η λειτουργία όμως της μονάδας DES χωρίζεται σε δύο μονοπάτια. Το πρώτο απ΄ αυτά ορίζει τον τύπο του DES αλγορίθμου που θα χρησιμοποιηθεί και το είδος της ενέργειας, κρυπτογράφηση ή αποκρυπτογράφηση και ονομάζεται "μονοπάτι ελέγχου". Στο μονοπάτι ελέγχου ορίζονται τα κλειδιά της (απο)κρυπτογράφησης και στην περίπτωση αλγορίθμου DES-CBC ή DES3-CBC δίνεται επίσης προαιρετικά ένα "διάνυσμα αρχικοποίησης". Η συσκευή μπορεί να αποθηκεύσει τα κλειδιά κρυπτογράφησης για δεκαέξι ροές δεδομένων. Η ροή στην οποία ανήκει

ένα πακέτο, είτε είναι πακέτο δεδομένων είτε είναι πακέτο ελέγχου, καθορίζεται από την τιμή της πρώτης λέξης του πακέτου. Η μονάδα DES, όσον αφορά τους CBC αλγορίθμους, μπορεί να ανανεώνει και να κρατάει σε εσωτερικούς της καταχωρητές τη νέα τιμή του διανύσματος αρχικοποίησης. Εδώ χρειάζεται ιδιαίτερη προσοχή στην περίπτωση πολλών ροών, γιατί οι καταχωρητές αυτοί είναι οι ίδιοι για όλες τις ροές. Έτσι το λογισμικό πρέπει να είναι κατάλληλα σχεδιασμένο ώστε να μπορεί να χειριστεί επιτυχώς αυτήν την ιδιαιτερότητα.

Λειτουργία του οδηγού της μονάδας DES

Ο οδηγός για το DES είναι ένας οδηγός τύπου συσκευής χαρακτήρων (character device driver) [5]. Κατ΄ αυτόν τον τρόπο ακολουθεί τη βασική δομή ενός οδηγού συσκευής χαρακτήρων. Συνεπώς "άνοίγει" και "κλείνει " με τις συνήθεις κλήσεις συστήματος *open()* και *close()*. Η ανάγνωση και γραφή δεδομένων στη συσκευή υλοποιούνται με τις κλήσεις *read()* και *write* αντίστοιχα. Για τον έλεγχο της συσκευής, πχ. αρχικοποιήσεις, κλειδιά κρυπτογράφησης, υλοποιούνται και χρησιμοποιούνται οι εντολές εισόδου/εξόδου/ελέγχου (IOCTL commands).

Ο οδηγός αυτός υποστηρίζει τους τέσσερις αλγορίθμους τύπου DES που υποστηρίζει και η συσκευή. Στόχος του οδηγού είναι η επίτευξη της καλύτερης δυνατής επίδοσης σε συνδυασμό με την καλύτερη δυνατή λειτουργικότητα του συστήματος. Εποπτικά αναφέρεται ότι οι διάφορες μετρήσεις που έγιναν έδειξαν ένα λόγο χρόνων εκτέλεσης των αλγορίθμων μεταξύ υλικού και λογισμικού της τάξης του 1/5, 1/20 και 1/1000. Ο πρώτος λόγος αναφέρεται στην περίπτωση της βιβλιοθήκης OpenSSL, όπου το μεγαλύτερο μέρος του χρόνου που απαιτεί η εκτέλεση της μέτρησης το καταλαμβάνουν η αρχικοποίηση της βιβλιοθήκης και οι κλήσεις συστήματος. Στην περίπτωση όμως που το μέγεθος του πακέτου είναι μεγάλο, της τάξης των 4Kbytes και πάνω, και ο όγκος των δεδομένων είναι επίσης μεγάλος ($>$ 1MB), ο λόγος αυτός εκτοξεύεται στο 1/100. Ο δεύτερος λόγος καταγράφηκε από μετρήσεις που έγιναν με μικρά και ελαφριά δοκιμαστικά προγράμματα που δημιουργήθηκαν γι΄ αυτόν ακριβώς το σκοπό. Ο τελευταίος λόγος μετρήθηκε με μέτρηση του καθαρού χρόνου που απαιτείται για τη διεκπεραίωση της κρυπτογράφησης από το υλικό και από το λογισμικό.

Όπως προαναφέρθηκε, η επικοινωνία της συσκευής με το υπόλοιπο σύστημα επιτυγχάνεται μέσω του DMAC. Ο οδηγός εκμεταλλεύεται τη δημιουργία σημάτων διακοπών από τα κανάλια του DMAC που αντιστοιχούν στη μονάδα του DES. Επειδή όλες οι συσκευές που κάθονται στον ελεγκτή DMA χρησιμοποιούν την ίδια γραμμή διακοπής, ο οδηγός δηλώνει στον πυρήνα ότι η διακοπή

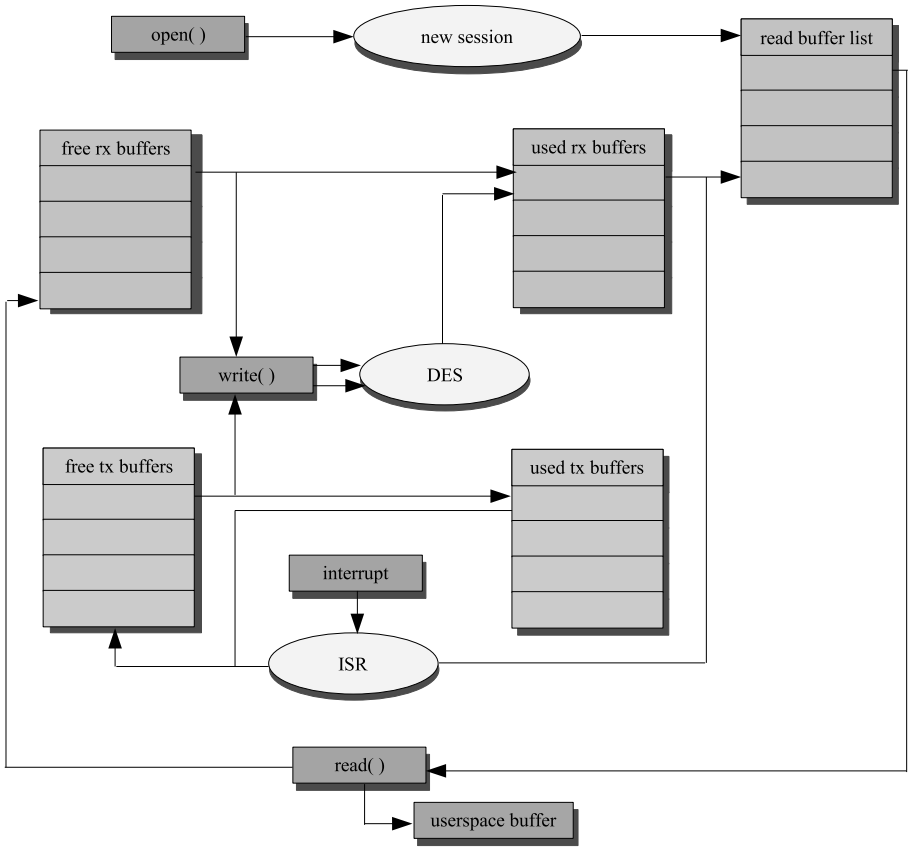
[5] Στο Linux υπάρχουν τρεις βασικές κατηγορίες οδηγών: χαρακτήρων (character), πλοκάδας (block) και διεπαφής δικτύου (network interface)

αυτή είναι μοιραζόμενη. Έτσι, με ένα μικρό έλεγχο στην αρχή της ρουτίνας εξυπηρέτησης της διακοπής, αν η διακοπή έχει προέλθει από τη μονάδα DES εξυπηρετείται από τον οδηγό αυτό, αλλιώς την αναλαμβάνει ο κατάλληλος οδηγός της συσκευής από την οποία προκλήθηκε η διακοπή.

Λόγω της δομής του DMAC, για τη μεγαλύτερη δυνατή χρήση του συστήματος, ο οδηγός του DES προκαταλαμβάνει συνολικά 32 προσωρινούς καταχωρητές (buffers) μεγέθους 32 KBytes. Οι καταχωρητές αυτοί μοιράζονται εξίσου στις δύο κατευθύνσεις ροής, δηλαδή σε 16 καταχωρητές εισόδου (rx) και 16 καταχωρητές εξόδου (tx). Στην πρώτη υλοποίηση του οδηγού χρησιμοποιήθηκε δυναμική κατακράτηση μνήμης για τους προσωρινούς καταχωρητές, με στόχο την καλύτερη διαχείριση της διαθέσιμης μνήμης του συστήματος. Αποδείχτηκε όμως τελικά ότι η δυναμική κατακράτηση και απελευθέρωση μνήμης για τους προσωρινούς καταχωρητές κόστιζε πάρα πολύ στην επίδοση του συστήματος, καθιστώντας πιο συμφέρουσα τη στατική κατακράτηση μνήμης για όλους τους καταχωρητές κατά την αρχικοποίηση του οδηγού και την απελευθέρωση της με την αφαίρεση του οδηγού. Για μεγαλύτερη ακρίβεια, προκαταλαμβάνονται περιγραφητές προσωρινών καταχωρητών. Ο περιγραφέας προσωρινού καταχωρητή (buffer descriptor) είναι μια δομή δεδομένων, που εκτός από τον καταχωρητή περιλαμβάνει κι ένα πλήθος πεδίων με πληροφορίες που χρησιμοποιεί ο οδηγός. Οι πληροφορίες αυτές έχουν να κάνουν με την υποστήριξη πολλαπλών ροών δεδομένων και το συγχρονισμό της συσκευής με το λειτουργικό σύστημα και τις εφαρμογές χρήστη.

Το σχήμα 4.1 περιγράφει τη λειτουργία του οδηγού. Ο οδηγός έχει υλοποιηθεί με βάση μια αρχιτεκτονική ανάγνωσης και εγγραφής χωρίς κλείδωμα (non-block read/write design). Για τη μεταφορά των δεδομένων χρησιμοποιούνται τέσσερις καθολικές FIFO λίστες για buffers και μία επιπλέον λίστα για κάθε σύνοδο του οδηγού. Οι τέσσερις καθολικές λίστες έχουν όλες το ίδιο μέγεθος και χωρητικότητα 16 buffers. Στην πραγματικότητα, τα στοιχεία που περιέχουν οι λίστες αυτές είναι κάποιες δομές που λειτουργούν σαν περιγραφητές των buffers, που έχουν μεταξύ άλλων ένα δείκτη στην θέση μνήμης των πραγματικών buffers. Οι λίστες χωρίζονται σε μια "ελεύθερη" και μια "χρησιμοποιημένη" λίστα για κάθε κατεύθυνση μεταφοράς, δηλαδή λήψης (rx) και αποστολής (tx). Το μέγεθος των FIFOs των συνόδων είναι απεριόριστο.

Μία διεργασία χώρου χρήστη, όπως πχ. openssl, δημιουργεί μία σύνοδο του οδηγού με τη κλήση συστήματος *open()*. Στο σημείο αυτό ο οδηγός ορίζει ένα εσωτερικό αριθμό ταυτοποίησης (id) για τη σύνοδο που μόλις δημιουργήθηκε. Η ανταλλαγή δεδομένων με το υλικό ξεκινάει με τη κλήση συστήματος *write*. Τότε, ο οδηγός παίρνει ένα ζεύγος από περιγραφητές buffers , ένα για λήψη κι ένα για μετάδοση. Οι περιγραφητές αυτοί αφαιρούνται αντίστοιχα από την κορυφή των ελεύθερων λιστών λήψης και μετάδοσης και εισέρχονται στις αντίστοιχες ουρές των χρησιμοποιημένων περιγραφητών. Στο



Σχήμα 4.1: Λειτουργία του οδηγού της μονάδας κρυπτογράφησης

σώμα της *write()*, τα δεδομένα της εφαρμογής χώρου χρήστη αντιγράφονται και διαμορφώνονται σε ένα πακέτο, που αποθηκεύεται στην λίστα μετάδοσης που βρίσκεται στο χώρο διευθύνσεων του πυρήνα. Αμέσως μετά, και οι δύο περιγραφητές στέλνονται μέσω του ελεγκτή DMA στη μονάδα κρυπτογράφησης DES. Και οι δύο περιγραφητές παίρνουν το id της συνόδου που τους χρησιμοποιεί. Μόλις η μονάδα κρυπτογράφησης ολοκληρώσει την επεξεργασία των δεδομένων, το κρυπτογραφημένο/αποκρυπτογραφημένο πακέτο γράφεται στον περιγραφητή λήψης, ο οποίος βρίσκεται ακόμα στη λίστα των χρησιμοποιημένων περιγραφητών λήψης και ταυτόχρονα δημιουργείται μια διακοπή. Η ρουτίνα εξυπηρέτησης της διακοπής παίρνει τον περιγραφητή που βρίσκεται στην κορυφή της λίστας των χρησιμοποιημένων περιγραφητών μετάδοσης και τον τοποθετεί στην ουρά της λίστας των ελεύθερων περιγραφητών μετάδοσης. Μετά μετακινεί τον περιγραφητή λήψης από τη λίστα χρησιμοποιημένων περιγραφητών λήψης στη λίστα περιγραφητών "ἀνάγνωσης" της συνόδου, ανάλογα με το id του. Από τη λίστα αυτή, με την κλήση συστήματος *read()* το κρυπτογραφημένο/αποκρυπτογραφημένο πακέτο αντιγράφεται από τον buffer σε κατάλληλο χώρο αποθήκευσης της εφαρμογής του χώρου χρήστη, αφού πρώτα έχει αφαιρεθεί η παραπανίσια πληροφορία που είχε προστεθεί για την επεξεργασία από το υλικό. Ο περιγραφητής λήψης αυτός, αφαιρείται από τη λίστα ανάγνωσης της συνόδου και τοποθετείται στη λίστα των ελεύθερων περιγραφητών λήψης. Ο ασύγχρονος αυτός μηχανισμός διασφαλίζει την σωστή λειτουργία του οδηγού και την ταυτόχρονη ύπαρξη πολλών συνόδων με μέγιστη εκμετάλλευση του υλικού.

4.3.5 Μεταφορά και δημιουργία λογισμικού στο REGATE

Η μεταφορά και δημιουργία εφαρμογών στο REGATE απαιτεί τη χρήση μιας κατάλληλης αλυσίδας εργαλείων και την επιλογή κατάλληλων παραμέτρων για το μεταγλωττιστή και το συνδέτη. Λόγω της έλλειψης μονάδας κινητής υποδιαστολής καθώς και της έλλειψης εξομοίωσης μαθηματικού επεξεργαστή στον πυρήνα, οι εφαρμογές (όπως και ο πυρήνας) πρέπει να μεταγλωττιστούν και να συνδεθούν με τις ακόλουθες παραμέτρους:

- απενεργοποιημένη τη μονάδα κινητής υποδιαστολής (-mno-fpu)
- ενεργοποιημένη την εξομοίωση πράξεων κινητής υποδιαστολής (-msoft-float)

Έχει παρατηρηθεί ότι η βελτιστοποίηση μεγέθους (-Os) μπορεί να προκαλέσει προβλήματα παράγοντας ελαττωματικό κώδικα. Συνεπώς η βελτιστοποίηση

αυτή δεν πρέπει να χρησιμοποιείται για την παραγωγή λογισμικού για το REGATE.

Λαμβάνοντας υπ΄ όψη το μέγεθος της βιβλιοθήκης glibc και των απαραίτητων εργαλείων για τη χρήση δυναμικών βιβλιοθηκών και τους πόρους ενός ενσωματωμένου συστήματος σε μνήμη RAM και μόνιμο αποθηκευτικό χώρο, πρέπει να επιλεχθεί αν το λογισμικό θα μεταγλωττιστεί και συνδεθεί στατικά ή δυναμικά. Αν τα προγράμματα που θα τρέξουν στο σύστημα είναι λίγα, συμφέρει να "χτιστούν" στατικά και στη συνέχεια με το πρόγραμμα sparc-linux-strip να μειωθεί το μέγεθος τους με αφαίρεση άχρηστων σχολίων και συμβόλων αποσφαλμάτωσης. Αν όμως υπάρχουν πολλά προγράμματα που χρησιμοποιούν ως επί το πλείστον κοινές βιβλιοθήκες, τότε συμφέρει η δυναμική μεταγλώττιση και σύνδεση.

4.4 Μετρήσεις και αξιολόγηση του συστήματος

Στην ενότητα αυτή περιγράφονται αναλυτικά οι μετρήσεις που έχουν γίνει στο chip μέχρι τώρα και το αντίστοιχο πειραματικό περιβάλλον. Οι μετρήσεις αυτές αφορούν τις επιδόσεις των διεπαφών MAC και της μονάδας κρυπτογράφησης, καθώς και την επεξεργαστική ισχύ του συστήματος.

4.4.1 Μονάδα κρυπτογράφησης

Για τη μέτρηση των επιδόσεων της μονάδας κρυπτογράφησης χρησιμοποιήθηκε το REGATE σε συνδυασμό με το Λ.Σ. Linux και την OpenSSL.Με το πείραμα αυτό μετρήθηκε ο λόγος της βελτίωσης που επιφέρει η χρήση του υλικού για την επιτάχυνση της εκτέλεσης των αλγορίθμων κρυπτογράφησης, σε σχέση με την εκτέλεσή τους εξολοκλήρου από το λογισμικό. Οι μετρήσεις αυτές, όπως και όλες οι μετρήσεις που αναφέρονται σ΄ αυτή τη διπλωματική εργασία, έχουν γίνει με το ρολόι του REGATE ρυθμισμένο στα 75MHz.

Το δεύτερο πείραμα μετράει τον καθαρό χρόνο που ξοδεύει το σύστημα για την επεξεργασία αποκρυπτογράφησης/κρυπτογράφησης ενός προκαθορισμένου όγκου δεδομένων. Με τον όρο " καθαρό χρόνο " για κάθε πακέτο, εννοούμε τον χρόνο που μεσολαβεί μεταξύ της αποστολής ενός πακέτου στον DMAC μέχρι τη δημιουργία της διακοπής από τη λήψη του πακέτου, δηλαδή τον καθαρό χρόνο επεξεργασίας ενός πακέτου από τη μονάδα κρυπτογράφησης. Για το σκοπό αυτό έγιναν κάποιες μικρές αλλαγές στον οδηγό της συσκευής.

Τα αποτελέσματα εξαρτώνται πολύ από το μέγεθος του πακέτου και του συνολικού αρχείου. Σε πολύ μικρά αρχεία η βελτίωση είναι περίπου δεκαπλάσια. Καθώς το μέγεθος του αρχείου μεγαλώνει, ο χρόνος εκτέλεσης μέσω του υλικού είναι 100 φορές μικρότερος, και σε αρκετά μεγάλα αρχεία, της τάξης των 10MB και άνω, η βελτίωση φτάνει σε ένα κορεσμό κοντά στον λόγο 1 προς 180. Εντυπωσιακές είναι οι επιδόσεις της μονάδας κρυπτογράφησης στο δεύτερο πείραμα, όπου η βελτίωση στον καθαρό χρόνο εκτέλεσης του αλγορίθμου κρυπτογράφησης είναι περίπου στο 1 προς 1000.

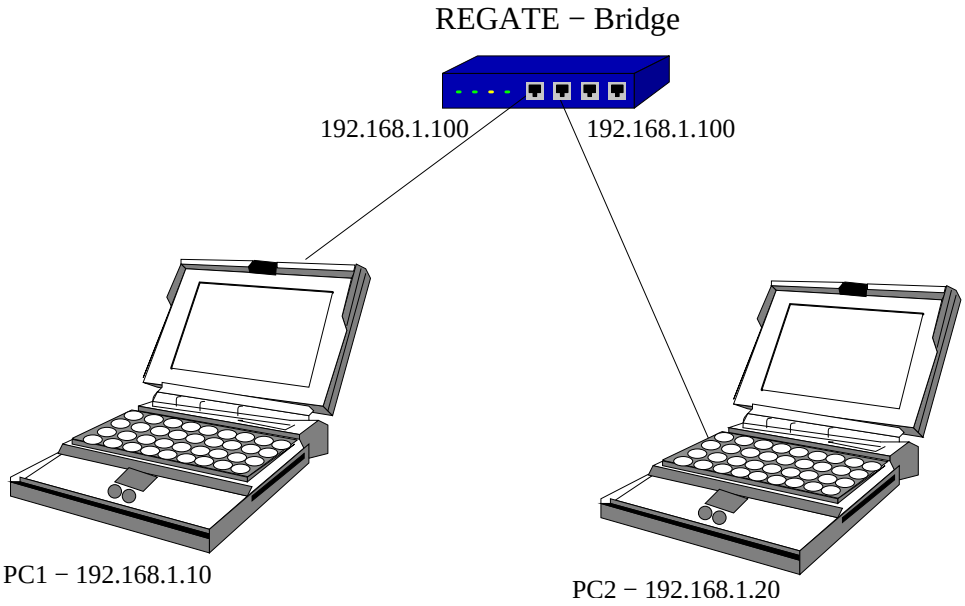
Τα δυο αυτά πειράματα εκτελέστηκαν τόσο στο περιβάλλον της πλατφόρμας με το FPGA στα 14,75 MHz, όσο και στο REGATE. Όπως ήταν αναμενόμενο, η αύξηση της επίδοσης είναι σχεδόν γραμμική. Η μικρή απόκλιση οφείλεται στην επίδραση της κρυφής μνήμης (εντολών και δεδομένων), που είναι αρκετά μεγαλύτερες στο REGATE σε σχέση με το FPGA.

4.4.2 Διεπαφές δικτύου

Στην περίπτωση των διεπαφών δικτύου έγιναν διάφορες τροποποιήσεις στον οδηγό της συσκευής, αφενός για να μπορεί να υποστηρίξει δύο συσκευές, αφετέρου για την επίτευξη δύο βασικών δικτυακών λειτουργιών. Αυτές είναι η χρήση του REGATE ως γέφυρα (bridge) και ως δρομολογητής (router).

Στην περίπτωση που το σύστημα λειτουργεί ως γέφυρα (βλ. σχήμα 4.2), όταν ένα πακέτο φτάσει στη μια από τις δύο πόρτες, γίνεται σύγκριση της MAC διεύθυνσης αποστολής της επικεφαλίδας του πακέτου και της MAC διεύθυνσης της διεπαφής αυτής. Αν συμπίπτουν, τότε το πακέτο προορίζεται για το REGATE, οπότε και προωθείται στη στοίβα δικτύου του Linux. Αν όμως οι διευθύνσεις αυτές δε συμπίπτουν, το πακέτο προωθείται και ξαναστέλνεται από την άλλη διεπαφή. Για την αποφυγή της καθυστέρησης που επιφέρει η μεταφορά του πακέτου στη στοίβα δικτύου του Linux και η εκ νέου αποστολή του από την άλλη πόρτα, το κομμάτι αυτό έχει παρακαμφθεί με μια έξυπνη υλοποίηση που εκμεταλλεύεται την ομοιότητα των δύο διεπαφών και τον ελεγκτή DMA.

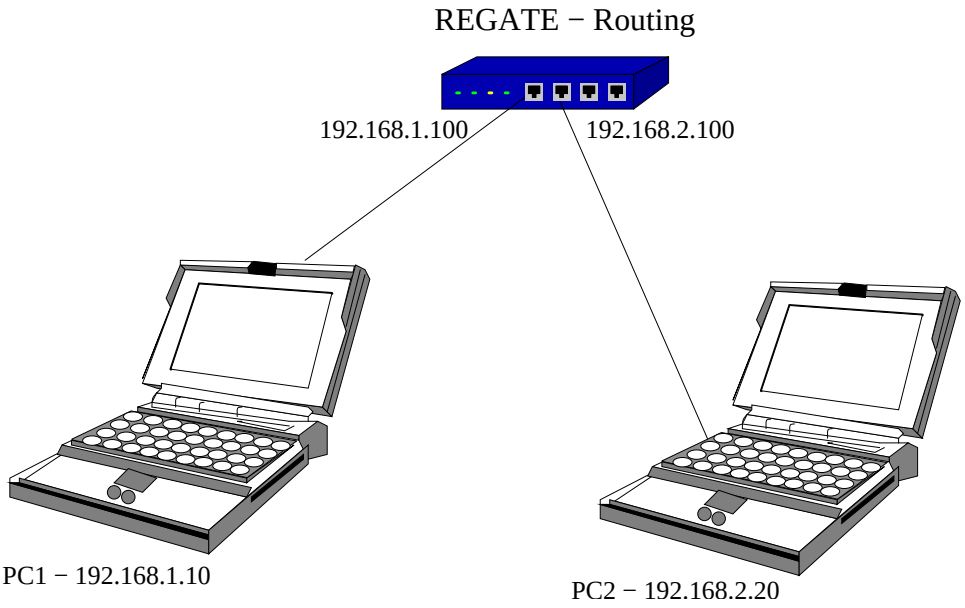
Στην περίπτωση του δρομολογητή έχει υλοποιηθεί ένα υποσύνολο των λειτουργιών δρομολόγησης. Το υποσύνολο αυτό περιλαμβάνει επεξεργασία IP επικεφαλίδων πακέτου και διεπαφής, αναζήτηση σε πίνακες δρομολόγησης βάσει των MAC και IP διευθύνσεων καθώς και επεξεργασία ARP πακέτων. Αν το εκάστοτε πακέτο πληρεί τις προϋποθέσεις για δρομολόγηση, όλη η επεξεργασία του και η δρομολόγηση λαμβάνουν χώρα μέσα στον οδηγό της συσκευής. Σε αντίθετη περίπτωση, το πακέτο προωθείται στην στοίβα δικτύου του Linux. Για τη μέγιστη δυνατή απόδοση και ικανοποιώντας τις απαιτήσεις ενός μικρού τοπικού δικτύου, η υλοποίηση αυτή υποστηρίζει δίκτυα τρίτης τάξης. Με ελάχιστες τροποποιήσεις και χωρίς μείωση της απόδοσης, μπορεί πολύ εύκολα να υπόστηρίξει και δίκτυα δεύτερης τάξης.



Σχήμα 4.2: Πειραματική διάταξη για χρήση γέφυρας

Για τις μετρήσεις που έγιναν, χρησιμοποιήθηκε η ακόλουθη πειραματική διάταξη. Δύο προσωπικοί υπολογιστές είναι συνδεδεμένοι ο καθένας σε μία πόρτα του REGATE. Για την περίπτωση που η διάταξη είναι ρυθμισμένη σε λειτουργία γέφυρας, οι δύο μονάδες MAC του REGATE έχουν την ίδια IP διεύθυνση. Όταν λειτουργεί ως δρομολογητής, η κάθε διεπαφή αναφέρεται σε ένα ξεχωριστό υποδίκτυο τρίτης τάξης. Για παράδειγμα, αμέσως μετά την εκκίνηση του συστήματος, η μία διεπαφή δείχνει στο υποδίκτυο τρίτης τάξης 192.168.1.0 με διεύθυνση 192.168.1.100 και η δεύτερη στο υποδίκτυο 192.168.2.0 με διεύθυνση 192.168.2.100 (σχήμα 4.3). Φυσικά, μέσω των ευρέως χρησιμοποιούμενων εργαλείων του Linux μπορεί να χρησιμοποιηθεί οποιοδήποτε υποδίκτυο 3ης τάξης.

Στον πίνακα 4.3 παρουσιάζονται οι μετρήσεις που έγιναν με τη συνδεσμολογία δρομολογητή με το εργαλείο μέτρησης επιδόσεων ettcp. Συγκεκριμένα, μετρήθηκε η απόδοση του συστήματος για αποστολή UDP πακέτων ποικίλου μεγέθους από άκρο σε άκρο. Με το ίδιο εργαλείο, για την περίπτωση πακέτων TCP, μετρήθηκε κίνηση της τάξης των 87-90 MBits/sec. Στα TCP πακέτα, το τελικό μέγεθος του πακέτου που καταλήγει στο σύρμα είναι 1514 bytes.Αξίζει να σημειωθεί ότι οι επιδόσεις μετρήθηκαν με το σύστημα ρυθμισμένο στα 75MHz.



Σχήμα 4.3: Πειραματική διάταξη για χρήση δρομολογητή

buffer size (bytes)	**throughput (Mbits/sec)**
150	23.19
200	30.63
400	31.87
600	48.86
800	58.36
1000	63.78
1200	69.69
1400	71.29

Table 4.3: Μετρήσεις δρομολόγησης πακέτων UDP με το ettcp.

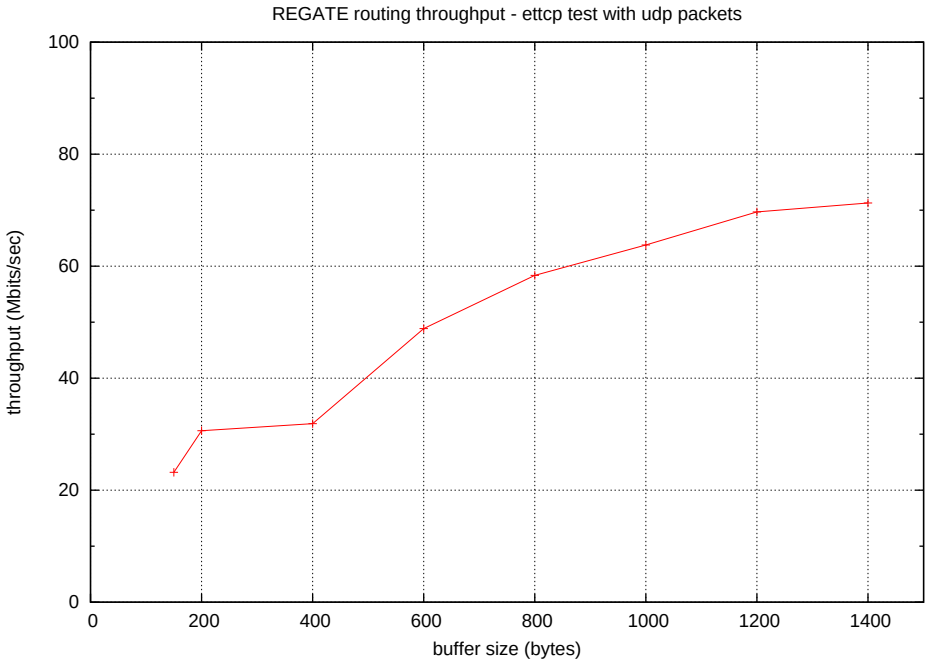


Figure 4.4: Δρομολόγηση πακέτων UDP

receive socket size (bytes)	send socket size (bytes)	send message size (bytes)	throughput (Mbits/sec)
256	2048	64	0.41
256	2048	128	0.01
512	2048	256	11.61
1024	2048	512	13.37
2048	2048	1024	27.61
2048	2048	4096	36.51
2048	2048	8192	36.81
2048	2048	16384	38.97
2800	2800	1400	37.86
8192	8192	4096	59.52
8192	8192	8192	56.52
8192	8192	16384	56.6
16384	16384	8192	90.85
16384	16384	16384	90.89
32768	32768	16384	93.18

Table 4.4: Δρομολόγηση πακέτων TCP με το netperf

Κεφάλαιο 5

Επεκτάσεις και συμπεράσματα

Η διεκπαιρέωση της διπλωματικής αυτής εργασίας οδήγησε σε πολλά και χρήσιμα συμπεράσματα. Τα συμπεράσματα αυτά καλύπτουν ποικίλες πτυχές, όπως το σχεδιασμό λογισμικού και υλικού, τις διαδικασίες ελέγχου, τις επιδόσεις και τις απαιτήσεις υλοποιήσεων οικιακών πυλών και την καταλληλότητα του λογισμικού ανοιχτού κώδικα για εφαρμογές αυτού του τύπου.

Όσον αφορά το σχεδιασμό υλικού, λαμβάνοντας υπόψιν και τις δικτυακές επιδόσεις του REGATE συμπεραίνουμε:

- Η γενική ιδέα της υλοποίησης συστήματος εντός ολοκληρωμένου κυκλώματος είναι κατάλληλη για εφαρμογές οικιακών πυλών. Η επιτάχυνση αναγκαίων διεργασιών με υλικό είναι ταχύτερη και πιο οικονομική από την χρήση ενός ισχυρότατου επεξεργαστή

- Η χρήση ελεγκτή άμεσης πρόσβασης στη μνήμη είναι αναγκαία προϋπόθεση για την επίτευξη υψηλών επιδόσεων. Είναι αξιοσημείωτο ότι το σύστημα με το FPGA, αν και έτρεχε στα 15 MHz , ήταν ικανό να τρέξει πυρήνα Linux τελευταίας γενιάς.

- Το υλικό πρέπει να δίνει την δυνατότητα παρατήρησης κατά τη λειτουργία του. Στο REGATE αυτό επιτυγχάνεται με την υπάρξη μονάδας αποσφαλμάτωσης στον επεξεργαστή και με τους ακροδέκτες εισόδου εξόδου γενικού σκοπού.

Ως προς τον έλεγχο του υλικού μπορούμε να συνοψίσουμε τα εξής:

- Για τον λεπτομερή και επαρκή έλεγχο του υλικού απαιτείται το υλικό να περιγράφεται πλήρως σε κατάλληλα τεχνικα κείμενα

- Είναι αναγκαία η συνεργασία και ο σωστός συντονισμός διαφόρων ομάδων, όπως υλικού, μικροηλεκτρονικής, δημιουργίας και ελέγχου λογισμικού.

- Η εξέταση πιθανών προβλημάτων πρέπει να είναι πολύπλευρη με στόχο τη σαφή διατύπωση και την απομόνωση του προβλήματος. Είναι πιθανό ένα πρόβλημα να κρύβει την ύπαρξη ενός δεύτερου, και η λύση του πρώτου να μη συνεπάγεται την αυτόματη λύση και του άλλου.

- ο έλεγχος του υλικού είναι μια επίπονη και χρονοβόρα διαδικασία. Ο κύκλος ελέγχου μπορεί να μειωθεί σημαντικά με χρήση ισχυρού περιβάλλοντος εξομοίωσης

Καθώς το σύστημα βρίσκεται στη διαδικασία ελέγχου και αξιολόγησης, δεν είναι συνετό να εξαχθούν βιαστικά συμπεράσματα. Ωστόσο, με βάση τις δικτυακές επιδόσεις του REGATE, το λειτουργικό σύστημα Linux αποδεικνύεται παραπάνω από κατάλληλο για την οδήγηση ενσωματωμένων συστημάτων. Η ελευθερία της αλλαγής του πυρήνα και η βελτιστοποίηση του ανάλογα με τις υπηρεσίες που το σύστημα καλείται να προσφέρει, καθιστούν το Linux ιδιαίτερα ευέλικτο και προσαρμόσιμο. Αυτό φάνηκε και στην περίπτωση του οδηγού των διεπαφών δικτύων, που με μικρές προσαρμογές μπορούσε να βελτιστοποιήσει το σύστημα είτε σαν γέφυρα είτε σαν δρομολογητή.

Το σύστημα μπορεί να επεκταθεί σε διάφορους τομείς, κυρίως σε λογισμικό. Οι πιο βασικοί συνοψίζονται στους ακόλουθους:

- Στην τρέχουσα υλοποίηση απουσιάζει η υποστήριξη δυναμικής προσθήκης και αφαίρεσης modules του πυρήνα.

- Πρέπει να κατασκευαστούν οδηγοί για Linux για τα υπόλοιπα περιφερειακά του SoC.

- Αντικατάσταση υπάρχοντος λογισμικού με νέο, που να εκμεταλλεύεται τις ιδιαιτερότητες της αρχιτεκτονικής του REGATE

Το REGATE αποτελεί ένα πρωτότυπο, που μπορεί να αποτελέσει τη βάση για μια νέα, πιο αποδοτική και εξελιγμένη πλατφόρμα στο χώρο των οικιακών πυλών, λόγω της επεκτάσιμης αρχιτεκτονικής του. Καθώς ο χώρος αυτός είναι πολύ καινούριος, είναι πιθανό ο χρόνος να αποδείξει ότι κάποιες σχεδιαστικές επιλογές ήταν σωστές και κάποιες όχι. Αυτό όμως θα μπορεί να γίνει όταν τελειώσει η αξιολόγηση του SoC και όταν το όλο σύστημα συγκριθεί με άλλες υλοποιήσεις.

Παράρτημα Α΄

Πηγαίος κώδικας

Στο παράρτημα αυτό παρατίθεται ένα βασικό μέρος του κώδικα που χρησιμοποιήθηκε στη διπλωματική αυτή. Ολόκληρος ο κώδικας καταλαμβάνει τεράστιο όγκο (᾽ 20000 γραμμές κώδικα). Για το λόγο αυτό στο παράρτημα αυτό υπάρχουν μόνο αρχεία που η λειτουργία τους περιγράφεται στο κείμενο της διπλωματικής εργασίας.

Α΄.1 Δοκιμαστικά προγράμματα

Το πρώτο μέρος περιλαμβάνει τον κορμό ενός στοιχειώδους προγράμματος που χρησιμοποιήθηκε για τον έλεγχο του υλικού. Πάνω στη βάση αυτή γίνονταν ποικίλες αλλαγές, ανάλογα με τις ιδιαιτερότητες του προς εξέταση υλικού.

```
 1: /**************************************************************************
 2:  * vi: set sw=4 ts=4:
 3:  *
 4:  * main.c
 5:  *
 6:  * Simple testing main.c
 7:  *
 8:  * by Ioannis Liverezas (iliverez@inaccessnetworks.com)
 9:  *
10:  * $Id$
11:  */
12:
13: #include "leon.h"
14: #include "sysconfig.h"
15: #include "uart0.h"
16:
17: #include "backtrace.h"
18: #include "ctraps.h"
19:
20: #include "mem.h"
21: #include "cache.h"
22:
23:
24:
25: /**************************************************************************/
26: #define TESTMEMEND   0x43fffff0
27: /**************************************************************************/
28:
29: int do_all_tests(unsigned int saddr);
30: int testaddr(unsigned int saddr, unsigned int test_val);
31: void loaddouble(unsigned int doublevar, unsigned int var1, unsigned int var2);
32: void storedouble(unsigned int doublevar, unsigned int var1, unsigned int var2);
```

```
 33: int do_double_access(unsigned int saddr, unsigned int test_val);
 34:
 35: int
 36: main() {
 37:     /* External symbols from the linker script. With these we keep out of the
 38:         * testing process the memory that is allocated by the test_membinary
 39:      */
 40:
 41:     extern unsigned int sdata, stext, edata, etext, stack_top;
 42:     unsigned int startadd;
 43:     int count = 0;
 44:
 45:     uart0_init();
 46:     uart0_puts("\r\nREGATE Memory test \n\r");
 47:     uart0_puts("test_mem binary text start: 0x");
 48:     uart0_putw((unsigned int) &stext);
 49:     uart0_puts("\n\rtest_mem binary text end: 0x");
 50:     uart0_putw((unsigned int) &etext);
 51:     uart0_puts("\n\rtest_mem binary data start: 0x");
 52:     uart0_putw((unsigned int) &sdata);
 53:     uart0_puts("\n\rtest_mem binary data end: 0x");
 54:     uart0_putw((unsigned int) &edata);
 55:     uart0_puts("\n\rstack top at: 0x");
 56:     uart0_putw((unsigned int) stack_top);
 57:
 58:
 59:     startadd = (stack_top + 0x120);
 60:     // startadd = 0x40002c00;
 61:
 62:     uart0_puts("\n\rstart_add at: 0x");
 63:     uart0_putw((unsigned int) startadd);
 64:     uart0_puts("\n\r");
 65:
 66: #if 1
 67:     uart0_puts("\r\n---------------------------------------\r\n");
 68:     uart0_puts("IC enabled, DC disabled, Bursts disabled\r\n");
 69:     *(volatile unsigned int *) 0x80000014 = 0x00000003;
 70:
 71:     count = do_all_tests(startadd);
 72:     if (count) {
 73:         uart0_puts("Test Errors = ");
 74:         uart0_putw(count);
 75:         uart0_puts("\r\n");
 76:     } else
 77:         uart0_puts("\r\nPASSED OK   :-)\r\n");
 78:
 79:     uart0_puts("\r\n---------------------------------------\r\n");
 80:     uart0_puts("IC enabled, DC enabled, Bursts disabled\r\n");
 81:     *(volatile unsigned int *) 0x80000014 = 0x0000000f;
 82:
 83:     count = do_all_tests(startadd);
 84:     if (count) {
 85:         uart0_puts("Test Errors = ");
 86:         uart0_putw(count);
 87:         uart0_puts("\r\n");
 88:     } else
 89:         uart0_puts("\r\nPASSED OK   :-)\r\n");
 90:
 91:     uart0_puts("\r\n---------------------------------------\r\n");
 92:     uart0_puts("IC enabled, DC enabled, Bursts enabled\r\n");
 93:     *(volatile unsigned int *) 0x80000014 = 0x0001000f;
 94:
 95:     count = do_all_tests(startadd);
 96:     if (count) {
 97:         uart0_puts("Test Errors = ");
 98:         uart0_putw(count);
 99:         uart0_puts("\r\n");
100:     } else
101:         uart0_puts("\r\nPASSED OK   :-)\r\n");
102:
103: #endif
104:
105:     uart0_puts("\r\n---------------------------------------\r\n");
106:     uart0_puts("IC enabled, DC enabled, Bursts enabled, Double access\r\n");
107:     *(volatile unsigned int *) 0x80000014 = 0x0001000f;
108:
109:     count = do_all_tests(startadd);
110:     if (count) {
111:         uart0_puts("Test Errors = ");
112:         uart0_putw(count);
113:         uart0_puts("\r\n");
114:     } else
115:         uart0_puts("\r\nPASSED OK   :-)\r\n");
```

```
116:
117:
118: while(1) {};
119: }
120:
121:
122: int
123: do_all_tests(unsigned int saddr) {
124:     int error_count = 0;
125:     int test_status = 0;
126:     int i ;
127:
128: #if 1
129:     for (i = 0; i < 32;  i++ ) {
130:         unsigned int v;
131:         v = 1 << i;
132:         uart0_puts("\r\nTestval = "); uart0_putw(v); uart0_puts("\n\r");
133:         test_status = testaddr(saddr, v);
134:         error_count += test_status;
135:     }
136:
137:     uart0_puts("\r\nTestval = ");
138:     uart0_putw(0); uart0_puts("\n\r");
139:     test_status = testaddr(saddr, 0);
140:     error_count += test_status;
141:
142:     uart0_puts("\r\nTestval = ");
143:     uart0_putw(0xffffffff); uart0_puts("\n\r");
144:     test_status = testaddr(saddr, 0xffffffff);
145:     error_count += test_status;
146:
147:
148:     for (i = 0; i < 32;  i++ ) {
149:         unsigned int v;
150:         v = 1 << i;
151:         uart0_puts("Double Access\r\nTestval = ");
152:         uart0_putw(v); uart0_puts("\n\r");
153:         test_status = do_double_access(saddr, v);
154:         error_count += test_status;
155:     }
156:
157:
158:     uart0_puts("\r\nDouble Access Testval = ");
159:     uart0_putw(0); uart0_puts("\n\r");
160:     test_status = do_double_access(saddr, 0);
161:     error_count += test_status;
162: #endif
163:
164:     uart0_puts("\r\nDouble Access, Testval = ");
165:     uart0_putw(0xaaaaaaaa); uart0_puts("\n\r");
166:     test_status = do_double_access(saddr, 0xaaaaaaaa);
167:     error_count += test_status;
168:
169:     return error_count;
170: }
171:
172: int
173: testaddr(unsigned int saddr, unsigned int test_val) {
174:     unsigned int addr;
175:     volatile unsigned int *a;
176:     int errors = 0;
177:     int register i;
178:     unsigned int test_value;
179:
180:
181:     /* write pattern */
182:     test_value = test_val;
183:     for (addr = saddr; addr < TESTMEMEND; addr += 4) {
184:         a = (volatile unsigned int *) addr;
185:         if (test_val == 0) {
186:             *a = addr;
187:         } else if (test_val == 0xffffffff) {
188:             *a = test_value;
189:             test_value = ~test_value;
190:         } else {
191:             *a = test_value;
192:         }
193:
194:         if ((addr & 0xfffff) == 0)
195:             uart0_puts("w");
196:     }
197:     uart0_puts("\n\r");
198:
```

```
199:     /* Delay */
200:     for (i = 0; i <= 10000000; i++);
201:     asm("flush"); asm("flush"); asm("flush");
202:     asm("nop"); asm ("nop"); asm ("nop"); asm ("nop");
203:     asm("nop"); asm ("nop"); asm ("nop"); asm ("nop");
204:     asm("nop"); asm ("nop"); asm ("nop"); asm ("nop");
205:     asm("nop"); asm ("nop"); asm ("nop"); asm ("nop");
206:
207:
208:     /* read pattern */
209:     test_value = ~test_val;
210:     for (addr = saddr; addr < TESTMEMEND; addr += 4) {
211:         if (test_val == 0)
212:             test_value = addr;
213:         else if (test_val == 0xffffffff)
214:             test_value = ~test_value;
215:         else
216:             test_value = test_val;
217:
218:         a = (volatile unsigned int *) addr;
219:         if  ( *a != test_value) {
220:             uart0_puts("\r\nError in 0x");
221:             uart0_putw(addr); uart0_puts("\r\n");
222:             errors++;
223:         }
224:
225:         /* type a "." at every 1MB or SDRAM */
226:         if ((addr & 0xfffff) == 0)
227:             uart0_puts("r");
228:     }
229:
230:     uart0_puts("\r\nTest for val = ");
231:     uart0_putw(test_val);
232:     uart0_puts(" ended \n\r");
233:
234:     return errors;
235: }
236:
237:
238:
239:
240: int
241: do_double_access(unsigned int saddr, unsigned int test_val) {
242:
243:     unsigned int addr;
244:     volatile unsigned int *a;
245:     int errors = 0;
246:     int register i;
247:     unsigned int test_value;
248:     volatile unsigned int d1, d2, t1, t2;
249:
250:     /* write pattern */
251:     test_value = test_val;
252:     for (addr = saddr; addr < TESTMEMEND; addr += 8) {
253:         a = (volatile unsigned int *) addr;
254:         if (test_val == 0) {
255:             storedouble((unsigned int)a, addr, addr + 4);
256:         } else if (test_val == 0xaaaaaaaa) {
257:             d1 = test_value;
258:             d2 = ~test_value;
259:             storedouble((unsigned int)a, d1, d2);
260:         } else {
261:             storedouble((unsigned int)a, test_value, test_value);
262:         }
263:
264:         if ((addr & 0xfffff) == 0)
265:             uart0_puts("w");
266:     }
267:     uart0_puts("\n\r");
268:
269:     /* Delay */
270:     for (i = 0; i <= 10000000; i++);
271:     asm("flush"); asm("flush"); asm("flush");
272:     asm("nop"); asm ("nop"); asm ("nop"); asm ("nop");
273:     asm("nop"); asm ("nop"); asm ("nop"); asm ("nop");
274:     asm("nop"); asm ("nop"); asm ("nop"); asm ("nop");
275:     asm("nop"); asm ("nop"); asm ("nop"); asm ("nop");
276:
277:
278:     /* read pattern */
279:     test_value = test_val;
280:     for (addr = saddr; addr < TESTMEMEND; addr += 8) {
281:         if (test_val == 0) {
```

```
282:             d1 = addr;
283:             d2 = addr+4;
284:         } else if (test_val == 0xaaaaaaaa) {
285:             d1 = test_value;
286:             d2 = ~test_value;
287:
288:         } else {
289:             d1 = test_val;
290:             d2 = test_val;
291:         }
292:         a = (volatile unsigned int *) addr;
293:         loaddouble((unsigned int)a,(unsigned int) &t1,(unsigned int)  &t2);
294:         if ((t1 != d1) || (t2 != d2)) {
295:             uart0_puts("\r\nError in 0x");
296:             uart0_putw(addr); uart0_puts("\r\n");
297:             errors++;
298:         }
299:
300:         /* type a "." at every 1MB or SDRAM */
301:         if ((addr & 0xfffff) == 0)
302:             uart0_puts("r");
303:     }
304:
305:     uart0_puts("\r\nTest for double access with test val = ");
306:     uart0_putw(test_val);
307:     uart0_puts(" ended \n\r");
308:
309:     return errors;
310: }
311:
312:
313: void
314: loaddouble(unsigned int doublevar, unsigned int var1, unsigned int var2) {
315:     asm volatile (" ldd [%o0], %i4
316:                     st %i4, [%o1]
317:                     st %i5, [%o2]");
318: }
319:
320: void
321: storedouble(unsigned int doublevar, unsigned int var1, unsigned int var2) {
322:     asm volatile (" mov %o1, %i4
323:                     mov %o2, %i5
324:                     std %i4, [%o0]");
325: }
```

```
  1: /* vi: set sw=4 ts=4:
  2:  *
  3:  * main.c
  4:  *
  5:  * MMU tests
  6:  *
  7:  * by Ioannis Liverezas (iliverez@inaccessnetworks.com)
  8:  *
  9:  * $Id$
 10:  */
 11:
 12: #include "leon.h"
 13: #include "sysconfig.h"
 14: #include "uart0.h"
 15:
 16: #include "backtrace.h"
 17: #include "ctraps.h"
 18:
 19: #include "mem.h"
 20: #include "cache.h"
 21:
 22:
 23:
 24: /*************************************************************************/
 25: #define BURST 0x1000f
 26:
 27: /*************************************************************************/
 28:
 29: int test_instr()   __attribute__ ((section(".test1")));
 30: int test_inst_trap() __attribute__ ((section(".test4")));
 31: int test_contexta() __attribute__ ((section(".test7")));
 32: int test_contextb() __attribute__ ((section(".test8")));
 33: int test_contextc() __attribute__ ((section(".test9")));
 34:
 35:
 36: int testdouble[2]  __attribute__ ((aligned (8))) = {0xdeadbeef, 0x12345678};
 37: int testdouble2[2] __attribute__ ((aligned (8))) = {0xaaaabbbb, 0xccccdddd};
 38: int testdouble3[2] __attribute__ ((aligned (8))) = {0};
```

```
 39: int testdouble4[2] __attribute__ ((aligned (8))) = {0xdeadbeef, 0x12345678};
 40: int tstrd1[20] __attribute__ ((section(".test2")));
 41: volatile int tstrd2[20] __attribute__ ((section(".test3")));
 42: volatile int ftype[10] __attribute__ ((section(".test5")));
 43: volatile int bittest[10] __attribute__ ((section(".test10"))) = {0};
 44: volatile int trapdouble[2] __attribute ((section(".test11"))) = {0xcabebeca, 0xbebababe};
 45:
 46: int tryfin()  __attribute__ ((section(".test6")));
 47: int globalcheck = 0;
 48: extern int mmuexist;
 49:
 50: /* extern int test_flush */
 51: extern int reg_windows_num;
 52: extern int dtest1;
 53: extern int dtest2;
 54: extern int intr_table[128];
 55: int LE_MAIN_STACK=0;
 56: extern int soft_interrupt_num;
 57: extern int intr_table[128];
 58: extern int cause_instruc();
 59: volatile int  dbptr;
 60: int testok = 1;
 61:
 62: /* testing variables, pointers and arrays */
 63: volatile int tarray[100] __attribute__((section(".mydata")))={0};
 64: volatile int *tpointer;
 65: int rpointer[200];
 66: int *npointer;
 67: int *rdpointer;
 68: volatile int *trapptr;
 69: int testfault = 0;
 70: volatile int global_ft = 0;
 71: volatile int global_fti = 0;
 72: int testcounter = 0;
 73:
 74: int tempcntxt = 0;
 75: volatile int tbypass[20] __attribute__((section(".bydata"))) = {0};
 76: volatile int tbypass2[20];
 77: int *tbptr1;
 78: int *tbptr2;
 79: int temp1;
 80: int temp2;
 81: int *tptr1;
 82: int *tptr2;
 83:
 84: /* memory mapping pointers */
 85: int *l1a;
 86: int *l2a;
 87: int *l3a1;
 88: int *l3a2;
 89: int *l3a3;
 90: int *l1b;
 91: int *l2b;
 92: int *l3b;
 93: int *l1c;
 94: int *l2c;
 95: int *l3c;
 96: int *l2d;
 97: int *l3d;
 98: int *l2e;
 99: int *l3e;
100: int *l3f;
101: int *cntxt;
102:
103: /* second context pointers */
104: int *bl1;
105: int *bl2;
106: int *bl3a;
107: int *bl3b;
108: int *bl3c;
109: int *bl2b;
110: int *bl3d;
111:
112: int *cl1;
113: int *cl2;
114: int *cl3a;
115: int *cl3b;
116: int *cl3c;
117: int *cl2b;
118: int *cl3d;
119:
120: int *dl1;
121: int *dl2;
```

```
122: int *dl3a;
123: int *dl3b;
124: int *dl3c;
125: int *dl2b;
126: int *dl3d;
127: int *el2;
128: int *el3;
129:
130: /*****************************************************************************/
131: int
132: main (void)
133: {
134:     /* initialize serial port */
135:     uart0_init();
136:     init_soft_traps();
137:
138:     set_cache(0);
139:     testbench();
140:     uart0_puts("1");
141: #if 1
142:     set_cache(0x3);
143:     asm("flush");
144:     testbench();
145:     asm("flush");
146:     uart0_puts("2");
147:
148:     set_cache(0xc);
149:     asm("flush");
150:     testbench();
151:     uart0_puts("3");
152:
153:     asm("flush");
154:     set_cache(0xf);
155:     asm("flush");
156:     testbench();
157:     asm("flush");
158:     uart0_puts("4");
159:
160:     set_cache(BURST);
161:     asm("flush");
162:     testbench();
163:     asm("flush");
164:     uart0_puts("5");
165:
166: #endif
167:
168:     asm("flush");
169:     if (testok == 1)
170:         uart0_puts("ok\r\n");
171:     while(1) {  }
172:
173: }
174:
175: /*****************************************************************************/
176: int testbench(void)
177: {
178:     /* TESTBENCH SECTION A: Leon cache disabled */
179:     /* initialize test data, using helping pointer *tpointer */
180:     volatile int tval, tval1;
181:     int i;
182:     int address;
183:
184:     /* initialize pointers */
185:     l1a = (int *)   0x43001000;
186:     l2a = (int *)   0x43002000;
187:     l3a1 = (int *)  0x43003000;
188:     l3a2 = (int *)  0x43004000;
189:     l3a3 = (int *)  0x43005c00;
190:     cntxt = (int *) 0x43006000;
191:     l1b = (int *)   0x43007000;
192:     l2b = (int *)   0x43008000;
193:     l3b = (int *)   0x43009000;
194:     l1c = (int *)   0x4300a000;
195:     l2c = (int *)   0x4300b000;
196:     l3c = (int *)   0x4300c000;
197:     l2d = (int *)   0x4300d000;
198:     l3d = (int *)   0x4300e000;
199:     l2e = (int *)   0x43010000;
200:     l3e = (int *)   0x43011000;
201:
202:     /* pointers for second context */
203:     bl1 = (int *)   0x43012000;
204:     bl2 = (int *)   0x43012400;
```

```
205:     bl3a = (int *)  0x43012500;
206:     bl3b = (int *)  0x43012600;
207:     bl3c = (int *)  0x43012700;
208:     bl2b = (int *)  0x43012800;
209:     bl3d = (int *)  0x43012900;
210:
211:
212:     /* pointers for third context */
213:     cl1 = (int *)   0x43012c00;
214:     cl2 = (int *)   0x43013000;
215:     cl3a = (int *)  0x43013100;
216:     cl3b = (int *)  0x43013200;
217:     cl3c = (int *)  0x43013300;
218:     cl2b = (int *)  0x43013400;
219:     cl3d = (int *)  0x43013500;
220:
221:      /* pointers for fourth context */
222:     dl1 = (int *)   0x43013800;
223:     dl2 = (int *)   0x43013c00;
224:     dl3a = (int *)  0x43013d00;
225:     dl3b = (int *)  0x43013e00;
226:     dl3c = (int *)  0x43013f00;
227:     dl2b = (int *)  0x43014000;
228:     dl3d = (int *)  0x43014100;
229:     el2  = (int *)  0x43014200;
230:     el3  = (int *)  0x43014400;
231:     l3f  = (int *)  0x43014600;
232:
233:     /* test A1: check 1-1 memory maps */
234:     l1a[0x00] = 0x8E;
235:     l1a[0x40] = ((((int) (l2a)) >> 4) & 0xfffffffc) | 0x1;
236:     l1a[0x80] = 0x08000006;
237:
238:     l2a[0x08] = ((( (int) (l3a1)) >> 4) & 0xfffffffc) | 0x1;
239:     l2a[0x09] = ((( (int) (l3a2)) >> 4) & 0xfffffffc) | 0x1;
240:
241:     address = 0x40200000;
242:
243:     for (i = 0;  i < 64; i++) {
244:         l3a1[i] = ((address >> 4) & 0xFFFFFF00) | 0x8E;
245:         address = address + 4096;
246:     }
247:
248:     address = 0x40240000;
249:     for (i = 0;  i < 64; i++) {
250:         l3a2[i] = ((address >> 4) & 0xFFFFFF00) | 0x8E;
251:     }
252:
253:     l1a[0x50] =  ((((int) (l2b)) >> 4) & 0xfffffffc) | 0x1;
254:     l2b[0x00] = ((( (int) (l3b)) >> 4) & 0xfffffffc) | 0x1;
255:     l3b[0x00] = 0x041a008e;
256:     l1a[0x41] =  ((((int) (l2d)) >> 4) & 0xfffffffc) | 0x1;
257:     l2d[0] = ((( (int) (l3f)) >> 4) & 0xfffffffc) | 0x1;
258:     l2d[0x28] = ((( (int) (l3d)) >> 4) & 0xfffffffc) | 0x1;
259:     l3f[0x0b] = 0x0410028e;
260:
261:     /* map instruction section */
262:     l2a[0x20] = ((((int) (l3a3)) >> 4) & 0xfffffffc) | 0x1;
263:     l3a3[0x00] = 0x0420008e;
264:     l3a3[0x01] = 0x0420018e;
265:     l3a3[0x02] = 0x0420028e;
266:
267:     /* map .bydata section for bypass test */
268:     l3d[0x0b] = 0x041c008e;
269:
270:     /* initialize VA for .test2 and .test3 sections, so that data read
271:      * trap occurs
272:      */
273:
274:     l3b[0x04] = 0x04090000;
275:     l3b[0x05] = 0x040a0000;
276:     l3b[0x06] = 0x040b0000;
277:     l3b[0x07] = 0x040c0000;
278:     l3b[0x08] = 0x040d0000;
279:
280: #if 1
281:     /* Create mappings for second context */
282:     bl1[0x00] = 0x8E;
283:     bl1[0x40] = ((((int) (bl2)) >> 4) & 0xfffffffc) | 0x1;
284:     bl1[0x80] = 0x08000006;
285:
286:     bl2[0x08] = ((( (int) (bl3a)) >> 4) & 0xfffffffc) | 0x1;
287:     bl2[0x09] = ((( (int) (bl3b)) >> 4) & 0xfffffffc) | 0x1;
```

```
288:
289:
290:      address = 0x40200000;
291:     for (i = 0;  i < 64; i++) {
292:         bl3a[i] = ((address >> 4) & 0xFFFFFF00) | 0x8E;
293:         address = address + 4096;
294:     }
295:     address = 0x40240000;
296:     for (i = 0;  i < 64; i++) {
297:         bl3b[i] = ((address >> 4) & 0xFFFFFF00) | 0x8E;
298:     }
299:
300:     bl1[0x50] = ((( int) (bl2b)) >> 4) & 0xfffffffc) | 0x1;
301:     bl2b[0x0] = ((( int) (bl3d)) >> 4) & 0xfffffffc) | 0x1;
302:     bl3d[0x09] = 0x040e008e;
303: #endif
304:
305: #if 1
306:     /* Create mappings for third context */
307:     cl1[0x00] = 0x8E;
308:     cl1[0x40] = ((((int) (cl2)) >> 4) & 0xfffffffc) | 0x1;
309:     cl1[0x80] = 0x08000006;
310:     cl2[0x08] = ((( int) (cl3a)) >> 4) & 0xfffffffc) | 0x1;
311:     cl2[0x09] = ((( int) (cl3b)) >> 4) & 0xfffffffc) | 0x1;
312:     address = 0x40200000;
313:     for (i = 0;  i < 64; i++) {
314:         cl3a[i] = ((address >> 4) & 0xFFFFFF00) | 0x8E;
315:         address = address + 4096;
316:     }
317:     address = 0x40240000;
318:     for (i = 0;  i < 64; i++) {
319:         cl3b[i] = ((address >> 4) & 0xFFFFFF00) | 0x8E;
320:     }
321:     cl1[0x50] = ((( int) (cl2b)) >> 4) & 0xfffffffc) | 0x1;
322:     cl2b[0x0] = ((( int) (cl3d)) >> 4) & 0xfffffffc) | 0x1;
323:     cl3d[0x0a] = 0x040f008e;
324: #endif
325:
326: #if 1
327:     /* Create mappings for fourth context */
328:     dl1[0x00] = 0x8E;
329:     dl1[0x40] = ((((int) (dl2)) >> 4) & 0xfffffffc) | 0x1;
330:     dl1[0x80] = 0x08000006;
331:
332:     dl2[0x08] = ((( int) (dl3a)) >> 4) & 0xfffffffc) | 0x1;
333:     dl2[0x09] = ((( int) (dl3b)) >> 4) & 0xfffffffc) | 0x1;
334:
335:     address = 0x40200000;
336:     for (i = 0;  i < 64; i++) {
337:         dl3a[i] = ((address >> 4) & 0xFFFFFF00) | 0x8E;
338:         address = address + 4096;
339:     }
340:     address = 0x40240000;
341:     for (i = 0;  i < 64; i++) {
342:         dl3b[i] = ((address >> 4) & 0xFFFFFF00) | 0x8E;
343:     }
344:
345:     dl1[0x50] = ((( int) (dl2b)) >> 4) & 0xfffffffc) | 0x1;
346:     dl2b[0x0] = ((( int) (dl3d)) >> 4) & 0xfffffffc) | 0x1;
347:     dl3d[0x0b] = 0x0410008e;
348: #endif
349:
350:     /* Initialize data elements for data access memory map test */
351:     npointer = (int *) 0x41a00000;
352:     for (i = 0;  i <  10;  i++)
353:        npointer[i] = (int)(0xdeadbeef +i);
354:
355:     /* Initialize data for bypass read/write test */
356:     tbptr1 = (int *) 0x41a0b000;
357:     tbptr2 = (int *) 0x41c00000;
358:     for (i = 0;  i < 10;  i++) {
359:         tbptr1[i] = (int)(0xaabbccdd +i);
360:         tbptr2[i] = i;
361:     }
362:
363:     /* Initialize data for data read exception test */
364:
365:     rdpointer = (int *)0x40900000;
366:     for (i = 0; i  < 10;  i++)
367:         rdpointer[i] = 0xeeffffee + i;
368:
369:     testdouble4[0] = 0xdeadbeef;
370:     testdouble4[1] = 0x12345678;
```

```
371:
372: /****************** enable mmu **********************************/
373:
374:
375:     cntxt[0]  = ((((int)(&l1a[0])) >> 4) & 0xfffffffc)|0x1 ;
376:     tempcntxt = (((int)(&(cntxt[0]))) >> 4) & 0xfffffffc;
377:
378:     /* initialize mmu control registers and flush mmu */
379:     set_context_num(0);
380:     set_context_pointer(tempcntxt);
381:     enable_mmu();
382:     asm volatile ("flush");
383:     flush_mmu();
384:
385:     testcounter =0;
386:     dbptr = 0;
387:     global_ft = 0;
388:     global_fti = 0;
389: #if 1
390:     address=0x40240000;
391:     for (i=0; i<2; i++) {
392:           dbptr += *((int *)address);
393:           testcounter ++;
394:           address += 4096;
395:     }
396:     if (testcounter == 2) {
397:         testok = 1;
398:     }
399:     else {
400:         testok = 0;
401:         uart0_puts("e1");
402:     }
403:     /* various memory mappings --- first data */
404:     tpointer = (int *) 0x50000000;
405:     for (i=0; i < 2; i++)
406:         rpointer[i] = i;
407:     for (i=0; i < 2; i++)
408:         dbptr = tarray[i];
409:
410:     asm("flush");
411:     flush_mmu();
412:     if ((tarray[1] == 0xdeadbef0) && (testok == 1)) {
413:         testok = 1;
414:     }
415:     else {
416:         testok = 0;
417:         uart0_puts("e2");
418:     }
419:     flush_mmu();
420:
421:     for (i=0; i <2; i++)
422:         tpointer[i] = rpointer[i];
423:     flush_mmu();
424:
425:
426:     /* now test instructions */
427:     testcounter = test_instr();
428:     if ((testcounter == 0x78) && (testok == 1)) {
429:         testok = 1;
430:     }
431:     else {
432:         testok = 0;
433:         uart0_puts("e3");
434:     }
435:     flush_mmu();
436:
437:     /************************** Bypass ******************************/
438:     /* uses readbypass, storebypass from mem.c. Remember to use '&' for
439:      * addresses
440:      */
441:     flush_mmu();
442:
443:     tbptr1 = (int *) 0x41a0b000;
444:     for (i = 0; i < 2; i++)
445:         dbptr = tbypass[i];
446:
447:     testcounter = tbypass[1];
448:     if ((testcounter == 1) && (testok == 1)) {
449:         testok = 1;
450:     }
451:     else {
452:         testok = 0;
453:         uart0_puts("e4");
```

```
454:         uart0_putw(testcounter);
455:     }
456:
457:
458:     flush_mmu();
459:     asm("flush");
460:     for (i =0;  i < 2; i++) {
461:         tbypass2[i] = (int)readbypass((int)(&tbptr1[i]));
462:         dbptr = tbypass2[i];
463:     }
464:
465:     testcounter = tbypass2[1];
466:     if ((testcounter == 0xaabbccde) && (testok == 1)) {
467:         testok = 1;
468:     }
469:     else {
470:         testok = 0;
471:         uart0_puts("e5");
472:         uart0_putw(testcounter);
473:     }
474:
475:     flush_mmu();
476:     flush_mmu();
477:     asm("flush");
478:
479:     /* try bypass write test */
480:     for (i=0; i < 2; i++)
481:         rpointer[i] = 100+i;
482:     for (i=0; i < 2; i++) {
483:         storebypass((int)(&tbptr1[i]), rpointer[i]);
484:         asm("flush");
485:         tbptr1[i] = rpointer[i]+200;
486:     }
487:     flush_mmu();
488:
489:     asm("flush");
490:     for (i =0;  i < 2; i++)
491:         tbypass2[i] = readbypass((int)(&tbptr1[i]));
492:
493:     testcounter = tbypass2[1];
494:     if ((testcounter == 101) && (testok == 1)) {
495:         testok = 1;
496:     }
497:     else {
498:         testok = 0;
499:         uart0_puts("e6");
500:     }
501:
502:     flush_mmu();
503:     asm("flush");
504:
505:     for (i = 0; i < 2; i++) {
506:         dbptr = tbypass[i];
507:     }
508:
509:     testcounter = tbypass[1];
510:     if ((testcounter == 301) && (testok == 1)) {
511:         testok = 1;
512:     }
513:     else {
514:         testok = 0;
515:         uart0_puts("e7");
516:     }
517:
518:
519:     flush_mmu();
520:     asm("flush");
521:
522:
523:     /* test load/store double, with or without use of ASI */
524:     temp1 = 0;
525:     temp2 = 0;
526:     loadfromdouble((int)&testdouble4[0],(int) &temp1, (int)&temp2, 0);
527:     testcounter = testdouble4[0];
528:     if ((testcounter == 0xdeadbeef) && (testok == 1)) {
529:         testok = 1;
530:     }
531:     else {
532:         testok = 0;
533:         uart0_puts("e8");
534:     }
535:     testcounter = testdouble4[1];
536:     if ((testcounter == 0x12345678) && (testok == 1)) {
```

```
537:         testok = 1;
538:     }
539:     else {
540:         testok = 0;
541:         uart0_puts("e9");
542:     }
543:
544:     flush_mmu();
545:     temp1 = 0x1a1b1c1d;
546:     temp2 = 0x21222324;
547:     store2double((int) &testdouble4[0], temp1, temp2, 0);
548:     testcounter = testdouble4[0];
549:     if ((testcounter == 0x1a1b1c1d) && (testok == 1)) {
550:         testok = 1;
551:     }
552:     else {
553:         testok = 0;
554:         uart0_puts("ea");
555:     }
556:     testcounter = testdouble4[1];
557:     if ((testcounter == 0x21222324) && (testok == 1)) {
558:         testok = 1;
559:     }
560:     else {
561:         testok = 0;
562:         uart0_puts("eb");
563:     }
564:
565:
566:     /* try store/load double with bypass */
567:     flush_mmu();
568:     tptr1 = (int *) (0x41a0b000);
569:     storebypass((int)&tptr1[0], 0x10101010);
570:     storebypass((int)&tptr1[1], 0x20202020);
571:
572:     flush_mmu();
573:     asm("flush");
574:     tptr1[0] = 0xb0b0b0b0;
575:     tptr1[1] = 0xa0a0a0a0;
576:
577:     /* read with bypass */
578:     loadfromdouble((int)&tptr1[0], (int) &temp1, (int) &temp2, 1);
579:     asm("flush");
580:     testcounter = temp1;
581:     if ((testcounter == 0x10101010) && (testok == 1)) {
582:         testok = 1;
583:     }
584:     else {
585:         testok = 0;
586:         uart0_puts("ec");
587:     }
588:     testcounter = temp2;
589:     if ((testcounter == 0x20202020) && (testok == 1)) {
590:         testok = 1;
591:     }
592:     else {
593:         testok = 0;
594:         uart0_puts("ed");
595:     }
596:
597:     /* read without bypass */
598:     flush_mmu();
599:     loadfromdouble((int)&tptr1[0], (int) &temp1, (int) &temp2, 0);
600:     testcounter = temp1;
601:     if ((testcounter == 0xb0b0b0b0) && (testok == 1)) {
602:         testok = 1;
603:     }
604:     else {
605:         testok = 0;
606:         uart0_puts("ef");
607:     }
608:     testcounter = temp2;
609:     if ((testcounter == 0xa0a0a0a0) && (testok == 1)) {
610:         testok = 1;
611:     }
612:     else {
613:         testok = 0;
614:         uart0_puts("eg");
615:     }
616:
617:     asm("flush");
618:     /* store double bypass */
619:     temp1   = 0xab0000cd;
```

```
620:     temp2   = 0xcd0000ef;
621:     flush_mmu();
622:
623:     store2double((int)&tptr1[0], temp1, temp2, 1);
624:     temp1   = 0xbbbbbbbb;
625:     temp2   = 0xaaffffaa;
626:     flush_mmu();
627:     asm("flush");
628:     store2double((int)&tptr1[0], temp1, temp2, 0);
629:     myprints("Reading with bypass\n\r");
630:     loadfromdouble((int)&tptr1[0], (int) &temp1, (int) &temp2, 1);
631:     asm("flush");
632:     /* reading bypass */
633:     testcounter = temp1;
634:     if ((testcounter == 0xab0000cd) && (testok == 1)) {
635:         testok = 1;
636:     }
637:     else {
638:         testok = 0;
639:         uart0_puts("eh");
640:     }
641:     testcounter = temp2;
642:     if ((testcounter == 0xcd0000ef) && (testok == 1)) {
643:         testok = 1;
644:     }
645:     else {
646:         testok = 0;
647:         uart0_puts("ei");
648:     }
649:     /* reading without bypass */
650:     flush_mmu();
651:     asm("flush");
652:     loadfromdouble((int)&tptr1[0], (int) &temp1, (int) &temp2, 0);
653:     testcounter = temp1;
654:     if ((testcounter == 0xbbbbbbbb) && (testok == 1)) {
655:         testok = 1;
656:     }
657:     else {
658:         testok = 0;
659:         uart0_puts("ej");
660:     }
661:
662:     testcounter = temp2;
663:     if ((testcounter == 0xaaffffaa) && (testok == 1)) {
664:         testok = 1;
665:     }
666:     else {
667:         testok = 0;
668:         uart0_puts("ek");
669:     }
670:
671:
672:     /* Interrupts Testing - Int handlers are in mmutraps.c */
673:     /* First try data read interrupt */
674:     global_ft = 0;
675:     for (i = 0; i < 2; i++) {
676:         flush_mmu();
677:         asm("flush");
678:         dbptr = tstrd1[i];
679:         if (i == 1)
680:             testcounter = dbptr;
681:         flush_mmu();
682:         asm("flush");
683:         storebypass(((int)(&l3b[0x4])), 0x04090000);
684:     }
685:     asm("flush");
686:     if ((testcounter == 0xeeffffef) && (global_ft = 1) && (testok == 1)) {
687:         testok = 1;
688:     }
689:     else {
690:         testok = 0;
691:         uart0_puts("el");
692:     }
693:
694:     flush_mmu();
695:     asm("flush");
696:     storebypass(((int)(&l3b[0x5])), 0x040a0000);
697:     /*-------------------------------------------------------*/
698:
699:     global_ft = 0;
700:     for (i = 0; i < 2; i++)  {
701:         flush_mmu();
702:         tstrd2[i] = 0x200300 + i;
```

```
703:         if (i == 1) {
704:             testcounter = tstrd2[i];
705:             dbptr = global_ft;
706:         }
707:         flush_mmu();
708:         storebypass(((int)(&l3b[0x5])), 0x040a0000);
709:     }
710:     asm("flush");
711:     if ((testcounter == 0x200301) && (dbptr == 1) && (testok == 1)) {
712:         testok = 1;
713:     }
714:     else {
715:         testok = 0;
716:         uart0_puts("em");
717:     }
718:
719:     /*-------------------------------------------------------*/
720:
721:     /* test instruction excecption */
722:      for (i = 0; i < 2; i++)  {
723:          flush_mmu();
724:          global_fti = 0;
725:          test_inst_trap();
726:          dbptr = global_fti;
727:          flush_mmu();
728:          asm("flush");
729:          storebypass(((int)(&l3b[0x6])), 0x040b0000);
730:          if ((testcounter == 0xafa) && (dbptr == 1) && (testok == 1)) {
731:              testok = 1;
732:          }
733:          else {
734:              testok = 0;
735:              uart0_puts("en");
736:          }
737:      }
738:
739:      /*-------------------------------------------------------*/
740:     /* check fault types, first with root access, then user */
741:     /* read only non-cahceable */
742:     storebypass(((int)(&l3b[0x7])), 0x040c0002);
743:     flush_mmu();
744:     global_ft = 0;
745:     ftype[0]=0x423329;
746:     dbptr = ftype[0];
747:     if ((global_ft == 2) && (dbptr == 0x423329) && (testok == 1)) {
748:         testok = 1;
749:     }
750:     else {
751:         testok = 0;
752:         uart0_puts("eo");
753:     }
754:
755:
756:     flush_mmu();
757:     storebypass(((int)(&l3b[0x7])), 0x040c0000);
758:     asm("flush");
759:     flush_mmu();
760:     global_ft = 0;
761:     storebypass(((int)(&l3b[0x8])), 0x040d0006);
762:     flush_mmu();
763:     tryfin();
764:     if ((global_fti == 2) && (testcounter = 0xae) && (testok == 1)) {
765:         testok = 1;
766:     }
767:     else {
768:         testok = 0;
769:         uart0_puts("ep");
770:     }
771:
772:     asm("flush");
773:     global_ft = 0;
774:     flush_mmu();
775:     storebypass(((int)(&l3b[0x7])),0x040c0012);
776:     ftype[0]=0xabf429;
777:     if ((global_fti == 2) && (testok == 1)) {
778:         testok =1;
779:     }
780:     else {
781:         testok = 0;
782:         uart0_puts("eq");
783:     }
784:
785:     flush_mmu();
```

```
786:     asm("flush");
787:     global_ft = 0;
788:
789:     storebypass(((int)(&l3b[0x7])),0x040c0012);
790:     dbptr = ftype[0];
791:     if ((global_fti == 2) && (dbptr == 0xabf429) && (testok == 1)) {
792:         testok = 1;
793:     }
794:     else {
795:         testok = 0;
796:         uart0_puts("er");
797:     }
798:     storebypass(((int)(&l3b[0x7])),0x040c0000);
799:
800:
801:     global_ft = 0;
802:     flush_mmu();
803:     asm("flush");
804:     storebypass(((int)(&l3b[0x7])),0x040c000a);
805:     ftype[3] = 0x4523;
806:     dbptr = ftype[3];
807:     if ((global_ft == 2) && (dbptr == 0x4523) && (testok == 1)) {
808:         testok = 1;
809:     }
810:     else {
811:         testok =0;
812:         uart0_puts("es");
813:     }
814:     flush_mmu();
815:     storebypass(((int)(&l3b[0x7])),0x040c0000);
816:
817:     global_ft = 0;
818:
819:     /* set machine to user mode to check page protection */
820:     storebypass(((int)(&l3b[0x7])),0x040c0016);
821:     flush_mmu();
822:     asm("flush");
823:     set_user();
824:
825:     asm("nop");
826:     ftype[5] = 0xbba25;
827:     dbptr = ftype[5];
828:     if ((global_ft == 2) && (dbptr == 0xbba25) && (testok == 1)) {
829:         testok = 1;
830:     }
831:     else {
832:         testok = 0;
833:         uart0_puts("et");
834:     }
835:     intr(2);
836: #endif
837:     global_ft = 0;
838:     asm("flush");
839:     storebypass(((int)(&l3b[0x7])),0x040c0000);
840:     flush_mmu();
841:     storebypass(((int)(&l3b[0x7])),0x040c001e);
842:     set_user();
843:     ftype[7] = 0x3aaa25;
844:     dbptr = ftype[7];
845:     if ((global_ft == 3) && (dbptr == 0x3aaa25) && (testok == 1)) {
846:         testok = 1;
847:     }
848:     else {
849:         testok = 0;
850:         uart0_puts("eu");
851:     }
852:     intr(2);
853:     storebypass(((int)(&l3b[0x7])),0x040c0000);
854:     asm("flush");
855:     flush_mmu();
856:     /*-----------------------------------------------*/
857:     global_ft = 0;
858:     asm("flush");
859:     storebypass(((int)(&l3b[0x7])),0x040c0000);
860:     flush_mmu();
861:     storebypass(((int)(&l3b[0x7])),0x040c001a);
862:     set_user();
863:     ftype[7] = 0x3a;
864:     dbptr = ftype[7];
865:     if ((global_ft == 3) && (dbptr == 0x3a) && (testok == 1)) {
866:         testok = 1;
867:     }
868:     else {
```

```
869:         testok = 0;
870:         uart0_puts("eu");
871:     }
872:     intr(2);
873:     storebypass(((int)(&l3b[0x7])),0x040c0000);
874:     asm("flush");
875:     flush_mmu();
876:
877:     /*-------------------------------------------------*/
878:     global_ft = 0;
879:     asm("flush");
880:     dbptr = 0;
881:     storebypass(((int)(&l3b[0x7])),0x040c0000);
882:     flush_mmu();
883:     storebypass(((int)(&l3b[0x7])),0x040c001a);
884:     set_user();
885:     dbptr = ftype[7];
886:     if ((global_ft == 3) && (dbptr == 0x3a) && (testok == 1)) {
887:         testok = 1;
888:     }
889:     else {
890:         testok = 0;
891:         uart0_puts("eu");
892:     }
893:     intr(2);
894:     storebypass(((int)(&l3b[0x7])),0x040c0000);
895:     asm("flush");
896:     flush_mmu();
897:
898:     /*-------------------------------------------------------*/
899:
900:         global_ft = 0;
901:     asm("flush");
902:     storebypass(((int)(&l3b[0x7])),0x040c0000);
903:     flush_mmu();
904:     dbptr = 0;
905:     storebypass(((int)(&l3b[0x7])),0x040c001e);;
906:     set_user();
907:     dbptr = ftype[7];
908:     if ((global_ft == 3) && (dbptr == 0x3a) && (testok == 1)) {
909:         testok = 1;
910:     }
911:     else {
912:         testok = 0;
913:         uart0_puts("eu");
914:     }
915:     intr(2);
916:     storebypass(((int)(&l3b[0x7])),0x040c0000);
917:     asm("flush");
918:     flush_mmu();
919:
920:     /*-------------------------------------------------------*/
921:     flush_mmu();
922:     asm("flush");
923:     testcounter = 0;
924:     global_fti == 0;
925:      storebypass(((int)(&l3b[0x6])), 0x040b001a);
926:
927:
928:      for (i = 0; i < 1; i++)  {
929:          set_user();
930:          global_fti = 0;
931:          test_inst_trap(1, 2, 3, 4, 5, 6, 7, 8, 9);
932:          dbptr = global_fti;
933:          intr(2);
934:          flush_mmu();
935:          storebypass(((int)(&l3b[0x6])), 0x040b001a);
936:
937:          if ((testcounter == 0xafa) && (dbptr == 3) && (testok == 1)) {
938:              testok = 1;
939:          }
940:          else {
941:              testok = 0;
942:              uart0_puts("en");
943:          }
944:          flush_mmu();
945:          asm("flush");
946:      }
947:      /*-------------------------------------------------------*/
948: #if 1
949:     /* fill cam */
950:     testcounter = 0;
951:     address=0x40240000;
```

```
952:     for (i=0; i<2; i++) {
953:         dbptr += *((int *)address);
954:         testcounter++;
955:         address += 4096;
956:     }
957:     if ((testcounter == 2) && (testok == 1)) {
958:         testok = 1;
959:     }
960:     else {
961:         testok = 0;
962:         uart0_puts("ev");
963:     }
964:
965:
966:     /* test context switch */
967:     disable_mmu();
968:     cntxt[1] = ((((int)(&bl1[0])) >> 4) & 0xfffffffc)|0x1 ;
969:     set_context_num(1);
970:     enable_mmu();
971:     flush_mmu();
972:     asm("flush");
973:     test_contexta();
974:     disable_mmu();
975:     set_context_num(0);
976:     enable_mmu();
977:     flush_mmu();
978:     asm("flush");
979:     if ((testcounter == 0xba) && (testok == 1)) {
980:         testok = 1;
981:     }
982:     else {
983:         testok = 0;
984:         uart0_puts("ew");
985:     }
986: #endif
987:
988: #if 1
989:     disable_mmu();
990:     cntxt[2] = ((((int)(&cl1[0])) >> 4) & 0xfffffffc)|0x1 ;
991:     set_context_num(2);
992:     enable_mmu();
993:     asm("flush");
994:     flush_mmu();
995:     test_contextb();
996:     disable_mmu();
997:     set_context_num(0);
998:     enable_mmu();
999:     flush_mmu();
1000:     asm("flush");
1001:     if ((testcounter == 0xbd) && (testok == 1)) {
1002:         testok = 1;
1003:     }
1004:     else {
1005:         testok = 0;
1006:         uart0_puts("ex");
1007:     }
1008:     flush_mmu();
1009:     asm("flush");
1010:     disable_mmu();
1011:     cntxt[3] = ((((int)(&dl1[0])) >> 4) & 0xfffffffc)|0x1 ;
1012:     set_context_num(3);
1013:     asm("flush");
1014:     enable_mmu();
1015:     flush_mmu();
1016:     asm("flush");
1017:     test_contextc();
1018:     disable_mmu();
1019:     set_context_num(0);
1020:     enable_mmu();
1021:     flush_mmu();
1022:     asm("flush");
1023:     if ((testcounter == 0xbf) && (testok == 1)) {
1024:         testok = 1;
1025:     }
1026:     else {
1027:         testok = 0;
1028:         uart0_puts("ey");
1029:     }
1030: #endif
1031:
1032:     /* test modified and accessed bits */
1033: #if 1
1034:     asm("flush");
```

```
1035:     disable_mmu();
1036:     l1a[0x55] = ((( (int) (el2) >> 4)) & 0xfffffffc) | 0x1;
1037:     el2[0x00] = ((( (int) (el3) >> 4)) & 0xfffffffc) | 0x1;
1038:     el3[0x00] = 0x0410018e;
1039:     int *testbit = (int *)0x41001000;
1040:     testbit[0]=0x4444;
1041:     testbit[1]=0x5555;
1042:     testbit[2]=0x6666;
1043:     testbit[3]=0x7777;
1044:     testbit[4]=0x8888;
1045:
1046:     enable_mmu();
1047:     asm("flush");
1048:     flush_mmu();
1049:
1050:     dbptr = bittest[0];
1051:     dbptr = (int)readbypass((int)(&el3[0x0]));
1052:     if ((dbptr == 0x041001ae) && (testok == 1))
1053:         testok = 1;
1054:     else {
1055:         testok = 0;
1056:         uart0_puts("e");
1057:     }
1058:
1059:     asm("flush");
1060:     flush_mmu();
1061:
1062:     bittest[1] = 0x333;
1063:     dbptr = (int)readbypass((int)(&el3[0x0]));
1064:     if ((dbptr == 0x041001ee) && (testok == 1))
1065:         testok = 1;
1066:     else {
1067:         testok = 0;
1068:         uart0_puts("e");
1069:     }
1070:
1071:
1072:     asm("flush");
1073:     flush_mmu();
1074:     dbptr = bittest[3];
1075:     dbptr = (int)readbypass((int)(&el3[0x0]));
1076:     if ((dbptr == 0x041001ee) && (testok == 1))
1077:         testok = 1;
1078:     else {
1079:         testok = 0;
1080:         uart0_puts("e");
1081:     }
1082:
1083:     asm("flush");
1084:     flush_mmu();
1085:     bittest[4] = 0x3432d;
1086:     dbptr = (int)readbypass((int)(&el3[0x0]));
1087:     if ((dbptr == 0x041001ee) && (testok == 1)) {
1088:         testok = 1;
1089:     }
1090:     else {
1091:         testok = 0;
1092:         uart0_puts("ez");
1093:     }
1094:
1095:
1096:     /*--------------------------------------------------------------------*/
1097:     /* test trap double */
1098:     asm("flush");
1099:     flush_mmu();
1100:     global_ft = 0;
1101:     tbptr1 = (int *) 0x4100b000;
1102:     storebypass(((int)&l3f[0x0b]), 0x0410028e);
1103:     temp1 = 0x8945;
1104:     temp2 = 0xaccd;
1105:
1106:     store2double((int)&trapdouble[0], temp1, temp2, 0);
1107:     storebypass(((int) &l3f[0x0b]), 0x04100200);
1108:     storebypass((int)&tbptr1[0], 0xabbabaab);
1109:     storebypass((int)&tbptr1[1], 0xcbbcbccb);
1110:
1111:     temp1 = 0;
1112:     temp2 = 0;
1113:     asm("flush");
1114:     flush_mmu();
1115:
1116:     loadfromdouble((int)&trapdouble[0], (int) &temp1, (int) &temp2, 0);
1117:     if ((temp1 == 0x8945) && (temp2 == 0xaccd) &&
```

```
1118:                (global_ft == 1) && (testok == 1))
1119:         testok = 1;
1120:     else {
1121:         testok = 0;
1122:         uart0_puts("probo1\r\n");
1123:          uart0_putw(global_ft); uart0_puts("\r\n");
1124:         uart0_putw(temp1); uart0_puts("\r\n");
1125:         uart0_putw(temp2); uart0_puts("\r\n");
1126:     }
1127:
1128:
1129:     asm("flush");
1130:     flush_mmu();
1131:     loadfromdouble((int)&tbptr1[0], (int) &temp1, (int) &temp2, 1);
1132:     if ((temp1 == 0xabbabaab) && (temp2 == 0xcbbcbccb) &&
1133:                (global_ft == 1) && (testok == 1))
1134:         testok = 1;
1135:     else {
1136:         testok = 0;
1137:         uart0_puts("probo2\r\n");
1138:         uart0_putw(global_ft); uart0_puts("\r\n");
1139:         uart0_putw(temp1); uart0_puts("\r\n");
1140:         uart0_putw(temp2); uart0_puts("\r\n");
1141:
1142:     }
1143:
1144:     asm("flush");
1145:     flush_mmu();
1146:     /*-------------------------------------------------------------------*/
1147:
1148:     asm("flush");
1149:     flush_mmu();
1150:     global_ft = 0;
1151:     tbptr1 = (int *) 0x4100b000;
1152:     storebypass(((int)&l3f[0x0b]), 0x0410028e);
1153:     trapdouble[0]=0xe00e;
1154:     trapdouble[1]=0xd00d;
1155:     storebypass(((int)&l3f[0x0b]), 0x04100200);
1156:     temp1 = 0x2aa2;
1157:     temp2 = 0xb44b;
1158:     store2double((int)&trapdouble[0], temp1, temp2, 1);
1159:     asm("flush");
1160:     flush_mmu();
1161:     //temp1 = readbypass((int)0x41002000);
1162:     //  uart0_puts("temp1=");uart0_putw(temp1); uart0_puts("\r\n");
1163:     temp1 = 0xffaa;
1164:     temp2 = 0xabcd;
1165:     store2double((int)&trapdouble[0], temp1, temp2, 0);
1166:     storebypass(((int) &l3f[0x0b]), 0x04100200);
1167:
1168:     //temp2 = readbypass((int)0x41002000);
1169:     //uart0_puts("temp2=");uart0_putw(temp2); uart0_puts("\r\n");
1170:     temp1 = 0;
1171:     temp2 = 0;
1172:     asm("flush");
1173:     flush_mmu();
1174:     asm("flush");
1175:     flush_mmu();
1176:     loadfromdouble((int)&trapdouble[0], (int) &temp1, (int) &temp2, 0);
1177:     asm("flush");
1178:     if ((temp1 == 0xffaa) && (temp2 == 0xabcd) &&
1179:                (global_ft == 1) && (testok == 1)) {
1180: //      uart0_puts("ye\r\n");
1181:         testok = 1;
1182:     }
1183:     else {
1184:         testok = 0;
1185:         uart0_puts("p\r\n");
1186:         //  uart0_putw(global_ft); uart0_puts("\r\n");
1187:         //uart0_putw(temp1); uart0_puts("\r\n");
1188:         //art0_putw(temp2); uart0_puts("\r\n");
1189:     }
1190:
1191:
1192:     asm("flush");
1193:     flush_mmu();
1194:
1195:     loadfromdouble((int)&tbptr1[0], (int) &temp1, (int) &temp2, 1);
1196:     if ((temp1 == 0x2aa2) && (temp2 == 0xb44b) &&
1197:         (global_ft == 1)) //&& (testok == 1))
1198:         {   testok = 1;
1199:    }
1200:     else {
```

```
1201:         testok = 0;
1202:         uart0_puts("probo2\r\n");
1203:         uart0_putw(global_ft); uart0_puts("\r\n");
1204:         uart0_putw(temp1); uart0_puts("\r\n");
1205:         uart0_putw(temp2); uart0_puts("\r\n");
1206:
1207:     }
1208: #endif
1209:     /*-------------------------------------------------------------*/
1210:
1211:     /* clear mmu and disable it, to continue with next test */
1212:     flush_mmu();
1213:     asm("flush");
1214:     disable_mmu();
1215:     asm("flush");
1216:     global_ft = 0;
1217:     global_fti = 0;
1218:     return(0);
1219: }
1220:
1221:
1222:
1223:
1224:
1225: /*************************************************************************/
1226:
1227: /*
1228:  * Local Variables:
1229:  * mode:c
1230:  * tab-width: 4
1231:  * c-basic-offset: 4
1232:  * End:
1233:  */
1234:
1235: int test_instr()
1236: {
1237:  volatile   int b=0;
1238:  volatile   int a=0;
1239:
1240:       a = 7 +b;
1241:       b=a+2;
1242:
1243:     //uart0_puts("t");
1244: return (0x78);
1245: }
1246:
1247: test_inst_trap(int a1, int a2, int a3, int a4, int a5, int a6, int a7, int a8, int a9)
1248: {
1249:     volatile int b=5;
1250:
1251:         b++;
1252:     myprints("inst trap test ok\r\n");
1253:     testcounter=0xafa;
1254:     b = a1 + a2 + a3 + a4 +a5 +a6 + a7 + a8 + a9;
1255:
1256: }
1257:
1258:
1259: int tryfin()
1260: {
1261:     volatile int k;
1262:     volatile int a;
1263:     k = 0;
1264:     a = 6;
1265:     k++;
1266:     a += k;
1267:     testcounter = 0xae;
1268: }
1269:
1270: int test_contexta()
1271: {
1272:     volatile int a = 0;
1273:     volatile int b = 0;
1274:
1275:     a+=23;
1276:     b = a + b;
1277:     testcounter = 0xba;
1278:     return(0);
1279: }
1280:
1281:
1282: int test_contextb()
1283: {
```

```
1284:     volatile int a = 0;
1285:     volatile int b = 0;
1286:     a+=0x2d3;
1287:     b = a + b;
1288:     testcounter = 0xbd;
1289:     return(0);
1290: }
1291:
1292:
1293: int test_contextc()
1294: {
1295:     volatile int a = 0;
1296:     volatile int b = 0;
1297:     a+=23134;
1298:     b = a + b;
1299:     testcounter = 0xbf;
1300:     return(0);
1301: }
1302:
1303:
```

A′.2 Οδηγοί για Linux

Περιλαμβάνει τους οδηγούς για τις διεπαφές δικτύου και τη μονάδα κρυπτογράφησης

```
  1: /*****************************************************************************
  2:  * ianmac.c
  3:  *
  4:  * Driver for IAN MAC
  5:  *
  6:  *
  7:  * written by Ioannis Liverezas (iliverez@inaccessnetworks.com)
  8:  *
  9:  */
 10:
 11: #include <linux/config.h>
 12: #include <linux/module.h>
 13: #include <asm/irq.h>
 14: #include <linux/sched.h>
 15: #include <linux/kernel.h> /* printk() */
 16: #include <linux/errno.h>  /* error codes */
 17: #include <linux/types.h>  /* size_t */
 18: #include <linux/interrupt.h> /* mark_bh */
 19: #include <linux/inet.h>
 20: #include <linux/timer.h>
 21: #include <linux/in.h>
 22: #include <linux/netdevice.h>   /* struct device, and other headers */
 23: #include <linux/etherdevice.h> /* eth_type_trans */
 24: #include <linux/ip.h>          /* struct iphdr */
 25: #include <linux/tcp.h>         /* struct tcphdr */
 26: #include <linux/skbuff.h>
 27: #include <linux/in6.h>
 28: #include <asm/checksum.h>
 29: #include <linux/module.h>
 30: #include <linux/list.h>
 31: #include <asm/page.h>
 32: #include <asm/leon.h>
 33: #include <asm/iandma.h>
 34: #include <linux/time.h>
 35: #include <asm/checksum.h>
 36: #include <net/ip.h>
 37: #include <linux/icmp.h>
 38: #include <net/icmp.h>
 39: #include "macian.h"
 40: #include "macian_driver.h"
 41:
 42:
 43:
 44: #define MACOADDR0 0x00
 45: #define MACOADDR1 0x1e
 46: #define MACOADDR2 0x28
 47: #define MACOADDR3 0x32
 48: #define MACOADDR4 0x3c
 49: #define MACOADDR5 0x46
 50:
```

```
 51: #define MAC1ADDR0 0x00
 52: #define MAC1ADDR1 0x1c
 53: #define MAC1ADDR2 0x29
 54: #define MAC1ADDR3 0x33
 55: #define MAC1ADDR4 0x3f
 56: #define MAC1ADDR5 0x16
 57:
 58: #undef MACIAN_DEBUG
 59:
 60: /* GPIO register addresses */
 61: #define GPIOR 0x800000a0
 62: #define GPDR  0x800000a4
 63:
 64: /* define the following macro to print out debug messages. This is taken
 65:  * by Linux Device Drivers */
 66:
 67: #undef PDEBUG
 68:
 69: #ifdef MACIAN_DEBUG
 70: #   define PDEBUG(fmt, args...)   printk(KERN_DEBUG "macian: " fmt, ## args)
 71: #else
 72: #   define PDEBUG(fmt, args...)
 73: #endif
 74:
 75: #define BRIDGE_MAC 1
 76:
 77: static struct net_device *macian_dev[2];
 78:
 79: static struct n_arphdr {
 80:     __u16  hw;
 81:     __u16  pt;
 82:     __u8   hwsize;
 83:     __u8   protsize;
 84:     __u16  opcode;
 85:     __u8   smac[6];
 86:     __u32  sip;
 87:     __u8   tmac[6];
 88:     __u32  tip;
 89: } __attribute__((packed));
 90:
 91: /*
 92:  * macian_open: Open the ethernet interface and reset the
 93:  * corresponding DMAC channel
 94:  */
 95: static int
 96: macian_open(struct net_device *dev) {
 97:     unsigned int dma_imr_rx;
 98:     unsigned int dma_imr_tx;
 99:     unsigned int mcr, flags;
100:     unsigned int tmp;
101:     int i,j;
102:     struct macian_priv *priv = (struct macian_priv *) dev->priv;
103:
104:     MOD_INC_USE_COUNT;
105:
106:     PDEBUG("macian_open\n");
107:
108:     if ( dev->tx_timeout == NULL )
109:         dev->tx_timeout = macian_timeout;
110: #if 0
111:
112:     if (dev->watchdog_timeo <= 0)
113:         dev->watchdog_timeo =TX_TIMEOUT;
114: #endif
115:
116:     dma_imr_rx = 0 | bFm(IANDMA_IMR_PC01);
117:     dma_imr_tx = 0 | bFm(IANDMA_IMR_PC08);
118:
119: #if 0
120:     /* trying to do some metrics */
121:     ulong malloc_secs, malloc_usecs, copy_secs, copy_usecs;
122:     struct timeval intime, outtime, btv;
123:     volatile int i, j;
124:     int bufnum, bufsize;
125:
126:     intime.tv_usec = 0;
127:     intime.tv_sec = 0;
128:     outtime.tv_usec = 0;
129:     outtime.tv_sec = 0;
130:     bufnum = 15000;
131:
132:     bufsize = 64;
133:     for (j = 0; bufsize < 1600; j++) {
```

```
134:         intime.tv_usec = 0;
135:         intime.tv_sec = 0;
136:         outtime.tv_usec = 0;
137:         outtime.tv_sec = 0;
138:
139:         do_gettimeofday(&intime);
140:         for (i = 0; i < bufnum ; i++) {
141:             struct sk_buff *skb;
142:             skb = dev_alloc_skb(bufsize);
143:             dev_kfree_skb(skb);
144:         }
145:
146:         do_gettimeofday(&outtime);
147:         malloc_secs = outtime.tv_sec - intime.tv_sec;
148:         if (malloc_secs) {
149:             malloc_secs--;
150:             outtime.tv_usec = outtime.tv_usec + 1000000;
151:         }
152:         malloc_usecs = outtime.tv_usec - intime.tv_usec;
153:         malloc_usecs += malloc_secs * 1000000;
154:         printk(KERN_ALERT "%s: skballoc/release time for %d buffers of %d size is %d usecs \n",
155:                __FUNCTION__, bufnum, bufsize, malloc_usecs);
156:         bufsize +=300;
157:     }
158:
159:     bufnum = 15000;
160:
161:     bufsize = 64;
162:     for (j = 0; bufsize < 1500; j++) {
163:         intime.tv_usec = 0;
164:         intime.tv_sec = 0;
165:         outtime.tv_usec = 0;
166:         outtime.tv_sec = 0;
167:         copy_secs = 0;
168:         copy_usecs = 0;
169:
170:         for (i = 0; i < bufnum ; i++) {
171:             struct sk_buff *skb1;
172:             struct sk_buff *skb2;
173:             skb1 = dev_alloc_skb(bufsize+100);
174:             skb2 = dev_alloc_skb(bufsize+100);
175:             /*
176:             skb_reserve(skb1, 2);
177:             skb_reserve(skb2, 2);
178:             skb_put(skb1, bufsize);
179:             skb_put(skb2, bufsize);
180:             */
181:             do_gettimeofday(&intime);
182:             memcpy(skb2->data, skb1->data, bufsize);
183:             do_gettimeofday(&outtime);
184:             copy_secs = 0;
185:             copy_secs = outtime.tv_sec - intime.tv_sec;
186:             if (copy_secs) {
187:                 copy_secs--;
188:                 outtime.tv_usec = outtime.tv_usec + 1000000;
189:             }
190:             copy_usecs += outtime.tv_usec - intime.tv_usec;
191:             copy_usecs += copy_secs * 1000000;
192:             dev_kfree_skb(skb1);
193:             dev_kfree_skb(skb2);
194:         }
195:         printk(KERN_ALERT
196:                "%s:  memcpy time for %d buffers of %d size is %d usecs \n",
197:                __FUNCTION__, bufnum, bufsize, copy_usecs);
198:         bufsize +=300;
199:     }
200: #endif
201:
202:     /* initialize timers */
203:     priv->isr_secs = 0;
204:     priv->isr_usecs = 0;
205:     priv->txsecs = 0;
206:     priv->txusecs = 0;
207:
208:
209:     priv->total_rx_pks = 0;
210:     priv->total_tx_pks = 0;
211:     /* request irq to set up interrupt handler */
212:     spin_lock_irqsave(&priv->lock, flags);
213:     init_macian(dev, 1);
214:     request_irq(IANDMA_IRQ,  macian_interrupt, SA_SHIRQ, dev->name, dev);
215:
216:     /* initialize routing tables */
```

```
217:     for (i = 0; i < 255; i++)
218:         for (j = 0; j < 6; j++)
219:             priv->routemac[i][j] = 0;
220:     for (i = 0; i < 6; i++)
221:         priv->routemac[255][i] = 0xf;
222:
223:     /* Enable Interrupts for the DMAC channels that correspond to MACIAN */
224:     SET_IANDMA_REG(IANDMA_IMR_A(priv->DRC), dma_imr_rx);
225:     SET_IANDMA_REG(IANDMA_IMR_A(priv->DTC), dma_imr_tx);
226:
227:     /* enable MACIAN receive - transmit */
228:     mcr = 0 | bFm(MACIAN_MCR_REN) | bFm(MACIAN_MCR_TEN) | bFm(MACIAN_MCR_PRM);
229:     SET_MACIAN_REG(MACIAN_MCR_A(priv->ethnum), mcr);
230:     spin_unlock_irqrestore(&priv->lock, flags);
231:     netif_start_queue(dev);
232: #if 0
233:
234:     printk("Leaving %s\n",__FUNCTION__);
235: #endif
236:
237: #if 0
238:     /* set GPIO 0-3 ports to outputs */
239:     tmp = GET_IANDMA_REG(GPDR);
240:     tmp |= 0xf;
241:     SET_IANDMA_REG(GPDR, tmp);
242:     /* initialize each pin's value */
243:     tmp = GET_IANDMA_REG(GPIOR);
244:     tmp &= 0xfffffff0;
245:     SET_IANDMA_REG(GPIOR, tmp);
246: #endif
247:     __u32 ipaddr;
248:     struct  in_device * indev = (struct in_device *) dev->ip_ptr;
249:     ipaddr = indev->ifa_list->ifa_address;
250:     printk(KERN_ALERT "ip address %08x\n", ipaddr);
251:     return 0;
252: }
253:
254: static int
255: macian_close(struct net_device *dev) {
256:     unsigned int mcr;
257:     struct macian_priv *priv = (struct macian_priv *) dev->priv;
258:
259:     mcr = 0;
260:     mcr &= ~bFm(MACIAN_MCR_REN);
261:     mcr &= ~bFm(MACIAN_MCR_TEN);
262:     SET_MACIAN_REG(MACIAN_MCR_A(priv->ethnum), mcr);
263:     SET_IANDMA_REG(IANDMA_IMR_A(priv->DRC), 0);
264:     SET_IANDMA_REG(IANDMA_IMR_A(priv->DTC), 0);
265:     netif_stop_queue(dev);
266:     printk(KERN_ALERT "%s rx pks = %d\n", dev->name, priv->total_rx_pks);
267:     printk(KERN_ALERT "%s tx pks = %d\n", dev->name, priv->total_tx_pks);
268: #if 0
269:
270:     printk(KERN_ALERT "%s: xmit  time = %u usecs \n",
271:            __FUNCTION__, priv->txusecs);
272:     printk(KERN_ALERT "%s: isr time = %u usecs \n", __FUNCTION__, priv->isr_usecs);
273: #endif
274:
275:     free_irq(IANDMA_IRQ, (void *)dev);
276:     MOD_DEC_USE_COUNT;
277:     return 0;
278: }
279:
280:
281:
282: static int
283: macian_start_xmit(struct sk_buff *skb, struct net_device *dev) {
284:     struct macian_priv *priv = (struct macian_priv *)dev->priv;
285:     unsigned int len, slen, idx;
286:     unsigned int *buf;
287:     unsigned int secs = 0;
288:     unsigned int usecs = 0;
289:     struct timeval intime, outtime;
290:     unsigned int tmp;
291:
292:     /* set pin GPIO pin 0 to 1*/
293: #if 0
294:
295:     do_gettimeofday(&intime);
296: #endif
297:
298:     /* set GPIO pin 0 to 1 */
299: #if 0
```

```
300:     tmp = GET_IANDMA_REG(GPIOR);
301:     tmp |= 1;
302:     SET_IANDMA_REG(GPIOR, tmp);
303: #endif
304:     spin_lock_irq(&priv->lock);
305:     PDEBUG("in macian_start_xmit: %s\n", dev->name);
306:     //    cli();
307:     len = skb->len;
308: #if 1
309:
310:     tmp = LEON_BYPASS_LOAD_PA(0x80000090);
311:     tmp &= (unsigned int) ~0x4000;
312:     LEON_BYPASS_STORE_PA(0x80000090, tmp);
313: #endif
314:
315:     if (priv->tx_count == MACIAN_TX_BUFFERS) {
316:         netif_stop_queue(dev);
317:         PDEBUG("%s(): tx_fifo full\n",__FUNCTION__);
318:         priv->stats.tx_errors++;
319:         sti();
320:         spin_unlock_irq(&priv->lock);
321:         return 1;
322:     }
323:     /*
324:     #if 0
325:     #   ifdef MACIAN_DEBUG
326:     dumpbuf(skb->data, len);
327:     #   endif
328:     #endif
329:
330:     */
331:     idx = priv->tx_next;
332:     slen = len < ETH_ZLEN ? ETH_ZLEN : len;
333:     buf = priv->txd[idx];
334:     //flush_cache_all();
335:     //    memset(buf, 0, MACIAN_BUF_SIZE);
336:     memcpy(buf, skb->data, len);
337:     //__flush_page_to_ram((unsigned int)buf);
338:
339:     SET_IANDMA_REG(IANDMA_SF_A(priv->DTC), IANDMA_MKLEN(slen));
340:     SET_IANDMA_REG(IANDMA_AF_A(priv->DTC),
341:                    IANDMA_MKADR(__pa((unsigned int)buf)));
342:     priv->tx_next = ( idx == MACIAN_TX_BUFFERS -1) ? 0 : idx + 1;
343:     priv->tx_count++;
344:     if (priv->tx_count == MACIAN_TX_BUFFERS)
345:         netif_stop_queue(dev);
346:     dev->trans_start = jiffies;
347:     dev_kfree_skb(skb);
348:     priv->stats.tx_bytes += len;
349:     PDEBUG("exiting %s()\n", __FUNCTION__);
350:
351: #if 0
352:
353:     do_gettimeofday(&outtime);
354:     secs = 0;
355:     usecs = 0;
356:     secs = outtime.tv_sec - intime.tv_sec;
357:     while (secs) {
358:         secs--;
359:         outtime.tv_usec = outtime.tv_usec + 1000000;
360:     }
361:
362:     usecs += outtime.tv_usec - intime.tv_usec;
363:     usecs += secs * 1000000;
364:     priv->txusecs += usecs;
365:
366:     /* set pin GPIO pin 0 to 0*/
367:     tmp = GET_IANDMA_REG(GPIOR);
368:     tmp &= (unsigned int) ~1;
369:     SET_IANDMA_REG(GPIOR, tmp);
370: #endif
371: #if 1
372:
373:     tmp = LEON_BYPASS_LOAD_PA(0x80000090);
374:     tmp |= (unsigned int) 0x4000;
375:     LEON_BYPASS_STORE_PA(0x80000090, tmp);
376: #endif
377:
378:     spin_unlock_irq(&priv->lock);
379:
380:
381:     return 0;
382: }
```

```
383:
384:
385:
386: /* The interrupt handler */
387:
388: static irqreturn_t
389: macian_interrupt(int irq, void *dev_id, struct pt_regs *regs) {
390:     struct net_device *dev = (struct net_device *) dev_id;
391:     struct net_device *dev0, *dev1;
392:     struct macian_priv *priv0, *priv1;
393:     unsigned int flags;
394:     int int_gen = 0;
395:     int i;
396:     struct timeval intime, outtime, rxtime, txtime, itertime;
397:     unsigned int secs, usecs;
398:     unsigned int tmp;
399: #if 0
400:
401:     do_gettimeofday(&intime);
402: #endif
403:
404:     unsigned int rx_packets, tx_packets;
405:     unsigned int cont;
406: #if 0
407:     tmp = GET_IANDMA_REG(GPIOR);
408:     tmp |= 2;
409:     SET_IANDMA_REG(GPIOR, tmp);
410: #endif
411:     /* poll the ethernet devices, since we have a shared interrupt */
412:     dev0 = (struct net_device *) macian_dev[0];
413:     dev1 = (struct net_device *) macian_dev[1];
414:
415:     if ((dev == NULL) || ((dev != dev0) && (dev != dev1))) {
416:         printk("interrupt on irq %d for unknown device\n", irq);
417:         return IRQ_HANDLED;
418:     }
419:     priv0 = (struct macian_priv *) dev0->priv;
420:     spin_lock_irq(&priv0->lock);
421:     priv1 = (struct macian_priv *) dev1->priv;
422:     spin_lock_irq(&priv1->lock);
423:
424:     /* Disable DMAC interrupts for MAC */
425:     SET_IANDMA_REG(IANDMA_IMR_A(priv0->DRC), int_gen);
426:     SET_IANDMA_REG(IANDMA_IMR_A(priv0->DTC), int_gen);
427:     SET_IANDMA_REG(IANDMA_IMR_A(priv1->DRC), int_gen);
428:     SET_IANDMA_REG(IANDMA_IMR_A(priv1->DTC), int_gen);
429:
430:     /* Handle receive events */
431:     cont = 1;
432:     while (cont) {
433:         cont = 0;
434:         /* check for received packets */
435:         rx_packets = GET_IANDMA_REG(IANDMA_CNT_A(priv0->DRC)) & 0xffff;
436:         if (rx_packets) {
437: #if 0
438:             /* enable this for time counters */
439:             tmp = GET_IANDMA_REG(GPIOR);
440:             tmp |= 0x4;
441:             SET_IANDMA_REG(GPIOR, tmp);
442: #endif
443:             cont = 1;
444:             priv0->total_rx_pks += rx_packets;
445:             PDEBUG("%s(): %d packets received\n", __FUNCTION__, rx_packets);
446:             macian_rx(dev0, rx_packets);
447: #if 0
448:             /* enable this for time counters */
449:             tmp = GET_IANDMA_REG(GPIOR);
450:             tmp &= (unsigned int) ~4;
451:             SET_IANDMA_REG(GPIOR, tmp);
452: #endif
453:         }
454:
455:         /* Check for transmitted packets */
456:         tx_packets = GET_IANDMA_REG(IANDMA_CNT_A(priv0->DTC)) & 0xffff;
457:         if (tx_packets) {
458:             cont = 1;
459:             priv0->total_tx_pks += tx_packets;
460:             PDEBUG("%s(): %d packets transmitted\n", __FUNCTION__, tx_packets);
461:             macian_tx(dev0, tx_packets);
462:         }
463:     }
464:
465:     cont = 1;
```

```
466:     while (cont) {
467:         cont = 0;
468:         /* check for received packets */
469:         rx_packets = GET_IANDMA_REG(IANDMA_CNT_A(priv1->DRC)) & 0xffff;
470:         if (rx_packets) {
471:             cont = 1;
472:             priv1->total_rx_pks += rx_packets;
473:             PDEBUG("%s(): %d packets received\n", __FUNCTION__, rx_packets);
474:             macian_rx(dev1, rx_packets);
475:         }
476:
477:         /* Check for transmitted packets */
478:         tx_packets = GET_IANDMA_REG(IANDMA_CNT_A(priv1->DTC)) & 0xffff;
479:         if (tx_packets) {
480:             cont = 1;
481:             priv1->total_tx_pks += tx_packets;
482:             PDEBUG("%s(): %d packets transmitted\n", __FUNCTION__, tx_packets);
483:             macian_tx(dev1, tx_packets);
484:         }
485:     }
486:     /* RE-enable DMAC interrupts for MAC */
487:
488:     SET_IANDMA_REG(IANDMA_IMR_A(priv0->DRC), bFm(IANDMA_IMR_PC01));
489:     SET_IANDMA_REG(IANDMA_IMR_A(priv0->DTC), bFm(IANDMA_IMR_PC08));
490:     spin_unlock_irq(&priv0->lock);
491:
492:     SET_IANDMA_REG(IANDMA_IMR_A(priv1->DRC), bFm(IANDMA_IMR_PC01));
493:     SET_IANDMA_REG(IANDMA_IMR_A(priv1->DTC), bFm(IANDMA_IMR_PC08));
494:     spin_unlock_irq(&priv1->lock);
495:
496:     netif_wake_queue(dev0);
497:     netif_wake_queue(dev1);
498:
499: #if 0
500:     /* enable this for time counters */
501:     do_gettimeofday(&outtime);
502:     usecs = 0;
503:     secs = outtime.tv_sec - intime.tv_sec;
504:     while (secs) {
505:         secs--;
506:         outtime.tv_usec = outtime.tv_usec + 1000000;
507:     }
508:     usecs += outtime.tv_usec - intime.tv_usec;
509:     priv0->isr_usecs += usecs;
510:     priv1->isr_usecs += usecs;
511:     //priv->risr_usecs += secs * 1000000;
512:
513:
514:     tmp = GET_IANDMA_REG(GPIOR);
515:     tmp &= (unsigned int) ~2;
516:     SET_IANDMA_REG(GPIOR, tmp);
517: #endif
518:
519:     unsigned int pending;
520:     pending = LEON_BYPASS_LOAD_PA(0x80000094);
521:     if (pending & 0x4000) {
522:         LEON_BYPASS_STORE_PA(0x8000009c, 0x4000);
523:     }
524:     return IRQ_HANDLED;
525: }
526:
527:
528: /* transmit interrupt handler */
529: static void
530: macian_tx(struct net_device *dev, int cnt) {
531:     struct timeval outtime;
532:     struct macian_priv *priv = (struct macian_priv *) dev->priv;
533:     unsigned int tmp1;
534:
535:
536:     PDEBUG("%s: %s\n",__FUNCTION__, dev->name);
537:     priv->tx_count -= cnt;
538:     priv->stats.tx_packets += cnt;
539:
540:     PDEBUG("macian_tx ok \n");
541:
542:
543: }
544:
545:
546: /* The receive function */
547: static inline void
548: macian_rx(struct net_device *dev, int cnt) {
```

```
549:
550:     struct macian_priv *priv = (struct macian_priv *) dev->priv;
551:     unsigned int i, len, idx;
552:     unsigned int *buf;
553:     volatile register int j;
554:     unsigned int tmp, tmp1;
555:     struct iphdr *iph;
556:     unsigned char isarp, isicmp, isip;
557:     __u32 ipaddr, subnet;
558:     struct  in_device * indev;
559:     struct net_device *dev0, *dev1;
560:     int rtidx, empty;
561:     volatile register int dmadelay = 1000;
562:
563:     dev0 = (struct net_device *) macian_dev[0];
564:     dev1 = (struct net_device *) macian_dev[1];
565:
566: #if 0
567:     /* enable this for time counters */
568:     tmp1 = GET_IANDMA_REG(GPIOR);
569:     tmp1 |= 8;
570: #endif
571:     SET_IANDMA_REG(GPIOR, tmp1);
572:
573:     PDEBUG("%s: in %s\n", dev->name, __FUNCTION__);
574:     for (i = 0; i < cnt; i++) {
575:         idx = priv->rx_next;
576:         //flush_cache_all();
577:         //__flush_page_to_ram((unsigned int)priv->rxd[idx]);
578:
579:
580:         len =  ((*(priv->rxd[idx]) & 0xffff0000) >> 16);
581:         if (len < 4) {
582:       //      printk("%s(): Bad packet size: %u\n", __FUNCTION__, len);
583:             priv->stats.rx_errors++;
584:             priv->stats.rx_length_errors++;
585:             dev->last_rx = jiffies;
586:             SET_IANDMA_REG(IANDMA_AF_A(priv->DRC),
587:                            IANDMA_MKADR(__pa(priv->rxd[idx])));
588:             priv->rx_next = (idx == MACIAN_RX_BUFFERS - 1) ? 0 : idx + 1;
589: #if 0
590:             /* enable this for time counters */
591:             tmp1 &= (unsigned int) ~8;
592:             SET_IANDMA_REG(GPIOR, tmp1);
593: #endif
594:     //        for (dmadelay = 0; dmadelay < 100000; dmadelay++);
595:             return;
596:         }
597:         else len -= 4;
598:
599:         if (len < 60 || len > 8192) {
600:   //        printk("%s(): Bad packet size: %u\n", __FUNCTION__, len);
601:             priv->stats.rx_errors++;
602:             priv->stats.rx_length_errors++;
603:             dev->last_rx = jiffies;
604:             SET_IANDMA_REG(IANDMA_AF_A(priv->DRC),
605:                            IANDMA_MKADR(__pa(priv->rxd[idx])));
606:             priv->rx_next = (idx == MACIAN_RX_BUFFERS - 1) ? 0 : idx + 1;
607:        //   for (dmadelay = 0; dmadelay < 100000; dmadelay++);
608:         } else {
609:             struct sk_buff *skb;
610:
611:             struct net_device *odev;
612:             struct macian_priv *opriv;
613:             int resend_len;
614:             isip = 0;
615:             isarp = 0;
616:
617:             /* recognize the mac device and set odev to point to the other */
618:             if (dev == dev0)
619:                 odev = dev1;
620:             else
621:                 odev = dev0;
622:             opriv = (struct macian_priv *) odev->priv;
623:
624:             /* check destination mac address and decide whether the packet
625:              *belongs to regate or not */
626:             unsigned char  *txc = (unsigned char *) &priv->rxd[idx][1];
627:
628:             /* check type of packet.0x0800  = IP , 0x0806 = ARP */
629:             if ( ( ((*(unsigned int *) &txc[0xc]) >> 16) & 0x0000ffff) == 0x0800)
630:                 isip = 1;
631:             else
```

```
632:                 if ( ( ((*(unsigned int *) &txc[0xc]) >> 16) & 0x0000ffff) == 0x0806)
633:                     isarp = 1;
634:
635:             if (!memcmp((unsigned char *) &priv->rxd[idx][1], (unsigned char *)dev->dev_addr, 6)) {
636:                 /* process ip packet */
637:                 if (isip) {
638:                     /* get ip header */
639:                     iph = (struct iphdr *) &txc[0xe];
640:                     /* find out ip address of the iface and subnet address */
641:                     indev = (struct in_device *) dev->ip_ptr;
642:                     ipaddr = indev->ifa_list->ifa_address;
643:                     if (ipaddr != iph->daddr) {
644:                         /* decrease ttl */
645:                         ip_decrease_ttl(iph);
646:                         /* change the destination macaddress. */
647:                         rtidx = iph->daddr & 0xff;
648:                         memcpy((unsigned char *) &txc[0],
649:                                (unsigned char *)&opriv->routemac[rtidx][0], 6);
650:
651:                         /* forward packet to the other mac */
652:                         SET_IANDMA_REG(IANDMA_SF_A(opriv->DTC),
653:                                        IANDMA_MKLEN(len));
654:                         SET_IANDMA_REG(IANDMA_AF_A(opriv->DTC),
655:                                        IANDMA_MKADR(__pa((unsigned int )txc)));
656:                     } else {
657:                         skb = dev_alloc_skb(len+2);
658:                         if (!skb) {
659:                             printk("%s(): Could not allocate skb for \
660: size %d \n", __FUNCTION__, len);
661:                             priv->stats.rx_dropped++;
662:                             break;
663:                         }
664:                         /* align IP on 16B boundary */
665:                         skb_reserve(skb,2);
666:                         skb->dev = dev;
667:                         skb_put(skb, len);
668:                        // flush_cache_all();
669:                         buf = priv->rxd[idx] + 1;
670:                         __flush_page_to_ram((unsigned int)buf);
671:                         memcpy(skb->data, buf, len);
672:                         skb->protocol = eth_type_trans(skb, dev);
673:                         //              skb->ip_summed = CHECKSUM_HW;
674:                         priv->stats.rx_packets += cnt;
675:                         priv->stats.rx_bytes += len;
676:                         netif_rx(skb);
677:                         dev->last_rx = jiffies;
678:                     }
679:                 } else {
680:                     skb = dev_alloc_skb(len+2);
681:                     if (!skb) {
682:                         printk("%s(): Could not allocate skb for size %d \n", __FUNCTION__, len);
683:                         priv->stats.rx_dropped++;
684:                         break;
685:                     }
686:                     /* align IP on 16B boundary */
687:                     skb_reserve(skb,2);
688:                     skb->dev = dev;
689:                     skb_put(skb, len);
690:                     //flush_cache_all();
691:                     buf = priv->rxd[idx] + 1;
692:                     memcpy(skb->data, buf, len);
693:                     __flush_page_to_ram((unsigned int)buf);
694:                     skb->protocol = eth_type_trans(skb, dev);
695:                     //                skb->ip_summed = CHECKSUM_HW;
696:                     priv->stats.rx_packets += cnt;
697:                     priv->stats.rx_bytes += len;
698:                     netif_rx(skb);
699:                     dev->last_rx = jiffies;
700:                 }
701:
702:             } else {
703:                 /* process arp packet */
704:                 if (isarp) {
705:                     unsigned int shi, slo, thi, tlo, hi, lo;
706:                     struct n_arphdr *arph = (struct n_arphdr *) &txc[0xe];
707:                     indev = (struct in_device *) dev->ip_ptr;
708:                     ipaddr = indev->ifa_list->ifa_address;
709:                     if (arph->opcode == 1)  {
710:                         rtidx = arph->sip & 0xff;
711:                         memcpy((unsigned char *) &priv->routemac[rtidx][0],
712:                                (unsigned char *) &arph->smac[0], 6);
713:                     }
714:                 }
```

```
715:                 skb = dev_alloc_skb(len+2);
716:                 if (!skb) {
717:                     printk("%s(): Could not allocate skb for size %d \n",
718:                            __FUNCTION__, len);
719:                     priv->stats.rx_dropped++;
720:                     break;
721:                 }
722:                 /* align IP on 16B boundary */
723:                 skb_reserve(skb,2);
724:                 skb->dev = dev;
725:                 skb_put(skb, len);
726:                 flush_cache_all();
727:                 buf = priv->rxd[idx] + 1;
728:                 memcpy(skb->data, buf, len);
729:                 //__flush_page_to_ram((unsigned int)buf);
730:                 skb->protocol = eth_type_trans(skb, dev);
731:                 //              skb->ip_summed = CHECKSUM_HW;
732:                 priv->stats.rx_packets += cnt;
733:                 priv->stats.rx_bytes += len;
734:                 netif_rx(skb);
735:                 dev->last_rx = jiffies;
736:             }
737:             SET_IANDMA_REG(IANDMA_AF_A(priv->DRC),
738:                            IANDMA_MKADR(__pa(priv->rxd[idx])));
739:             priv->rx_next = (idx == MACIAN_RX_BUFFERS - 1) ? 0 : idx + 1;
740:
741:         }
742:     }
743: #if 0
744:     /* time counter */
745:     tmp1 &= (unsigned int) ~8;
746:     SET_IANDMA_REG(GPIOR, tmp1);
747: #endif
748: }
749:
750:
751:
752: static struct net_device_stats
753:     *macian_get_stats(struct net_device *dev) {
754:     struct macian_priv *priv = (struct macian_priv *) dev->priv;
755:     return &priv->stats;
756: }
757:
758:
759:
760: /* initialize the device. This one is called each time the open() system call
761:  * is used
762:  */
763:
764: static void
765: init_macian(struct net_device *dev, int strt) {
766:     int i, tmp;
767:
768:     /* variables for macian registers' values */
769:     unsigned int mcr;
770:     unsigned int rpr, tpr;
771:     unsigned int flt;
772:     unsigned int rpt;
773:     unsigned int tpt;
774:     unsigned int ftr;
775:     unsigned int scr;
776:     unsigned int did;
777:     unsigned int ipg;
778:     unsigned int ral, rah;
779:     unsigned int tal, tah;
780:
781:     /* variables for iandma registers' values */
782:     unsigned int dma_cycle_budget;
783:     unsigned int dma_eth_rx_weight;
784:     unsigned int dma_eth_tx_weight;
785:     unsigned int dma_ccr;
786:     unsigned int dma_cer;
787:     unsigned int dma_csr;
788:
789:     struct macian_priv *priv = (struct macian_priv *) dev->priv;
790:
791:     /* Set values for macian registers */
792:     /* reset the controller */
793:     mcr = 0 | bFi(1,MACIAN_MCR_RRE) | bFi(1,MACIAN_MCR_TRE);
794:     SET_MACIAN_REG(MACIAN_MCR_A(priv->ethnum), mcr);
795:
796:     /* set rx/tx param and rxfilter registers */
797:     rpr = 0 | bFi(0x00, MACIAN_RPR_CDI);
```

```
798:     tpr = 0 | bFi(0x01, MACIAN_TPR_CEN);
799:     flt = 0 | bFm(MACIAN_FLT_LNG) | bFm(MACIAN_FLT_SHT);
800:     /* set rx and tx pause times */
801:     rpt = 0 | bFi(0x00, MACIAN_RPT_IPS1) | bFi(0x00, MACIAN_RPT_IPS2);
802:     tpt = 0 | bFi(0x12, MACIAN_TPT_IPS1) | bFi(0x34, MACIAN_TPT_IPS2);
803:     /* set fifo threshold register*/
804:     ftr = 0 | bFi(0x01, MACIAN_FTR_TTH) | bFi(0x01, MACIAN_FTR_RTH);
805:     /* set serial configuration register */
806:     scr = 0 | bFi(1,MACIAN_SCR_FDU);
807:     /* check device id */
808:     if ( (did = (GET_MACIAN_REG(MACIAN_DID_A(priv->ethnum)) & 0xff)) != 0x10) {
809:         printk(KERN_ALERT "%s(): MACIAN ERROR: Device ID incorrect!\n ", __FUNCTION__);
810:     }
811:
812:     /* set ipg register */
813:     ipg = 0 | bFi(0x09, MACIAN_IPG_IPG2) | bFi(0x0f, MACIAN_IPG_IPG1);
814:     /* set MAC address */
815:     rah = priv->MACADDR0 << 8 | priv->MACADDR1;
816:     tah = rah;
817:     ral = priv->MACADDR2 << 24 | priv->MACADDR3 << 16 | priv->MACADDR4 << 8 | priv->MACADDR5;
818:     tal = ral;
819:
820:     /* enable interface */
821:     mcr = 0 | bFi(1, MACIAN_MCR_TEN) | bFi(1, MACIAN_MCR_REN) | bFi(1, MACIAN_MCR_PRM);
822:     /* set the registers */
823:     SET_MACIAN_REG(MACIAN_RPR_A(priv->ethnum), rpr);
824:     SET_MACIAN_REG(MACIAN_TPR_A(priv->ethnum), tpr);
825:     SET_MACIAN_REG(MACIAN_FLT_A(priv->ethnum), flt);
826:     SET_MACIAN_REG(MACIAN_RPT_A(priv->ethnum), rpt);
827:     SET_MACIAN_REG(MACIAN_TPT_A(priv->ethnum), tpt);
828:     SET_MACIAN_REG(MACIAN_FTR_A(priv->ethnum), ftr);
829:     SET_MACIAN_REG(MACIAN_SCR_A(priv->ethnum), scr);
830:     SET_MACIAN_REG(MACIAN_IPG_A(priv->ethnum), ipg);
831:     SET_MACIAN_REG(MACIAN_RAL_A(priv->ethnum), ral);
832:     SET_MACIAN_REG(MACIAN_RAH_A(priv->ethnum), rah);
833:     SET_MACIAN_REG(MACIAN_TAL_A(priv->ethnum), tal);
834:     SET_MACIAN_REG(MACIAN_TAH_A(priv->ethnum), tah);
835:     SET_MACIAN_REG(MACIAN_MCR_A(priv->ethnum), mcr);
836:
837:     dev->dev_addr[0] = priv->MACADDR0;
838:     dev->dev_addr[1] = priv->MACADDR1;
839:     dev->dev_addr[2] = priv->MACADDR2;
840:     dev->dev_addr[3] = priv->MACADDR3;
841:     dev->dev_addr[4] = priv->MACADDR4;
842:     dev->dev_addr[5] = priv->MACADDR5;
843:
844: #if 1
845:     dma_cer = GET_IANDMA_REG(IANDMA_CER_A);
846:     dma_csr = GET_IANDMA_REG(IANDMA_CSR_A);
847:
848:     /* disable and corresponding dma controller's chanel */
849:     dma_csr &= ~(bFm(IANDMA_CSR_EN(priv->DRC)) | bFm(IANDMA_CSR_EN(priv->DTC)));
850:     dma_cer &= ~(bFm(IANDMA_CER_EN(priv->DRC)) | bFm(IANDMA_CER_EN(priv->DTC)));
851:     SET_IANDMA_REG(IANDMA_CER_A, dma_cer);
852:     SET_IANDMA_REG(IANDMA_CSR_A, dma_csr);
853:     /* enable and start corresponding dma channels */
854:     dma_cer |= bFm(IANDMA_CER_EN(priv->DRC)) | bFm(IANDMA_CER_EN(priv->DTC));
855:     dma_csr |= bFm(IANDMA_CSR_EN(priv->DRC)) | bFm(IANDMA_CSR_EN(priv->DTC));
856:     SET_IANDMA_REG(IANDMA_CER_A, dma_cer);
857:     SET_IANDMA_REG(IANDMA_CSR_A, dma_csr);
858:
859: #endif
860:
861:     /* Clear all pending interrupts for eth0 */
862:     tmp = GET_IANDMA_REG(IANDMA_CNT_A(priv->DRC));
863:     tmp = GET_IANDMA_REG(IANDMA_CNT_A(priv->DTC));
864:
865:     /* SETUP dma registers */
866: #if 1
867:     /* setup cycle budget */
868:     dma_cycle_budget = bFi(0xf0, IANDMA_CBR_TB) |
869:                        bFi(0xf0, IANDMA_CBR_RB) |
870:                        bFi(0x50, IANDMA_CBR_IC);
871:
872:     /* set etherport weights */
873:     dma_eth_rx_weight = bFi(0x40, IANDMA_CWR_CW);
874:     dma_eth_tx_weight = bFi(0x40, IANDMA_CWR_CW);
875: #endif
876:     /* start the dma controller */
877:     dma_ccr = GET_IANDMA_REG(IANDMA_CCR_A);
878:     dma_ccr |= bFi(1, IANDMA_CCR_SS) | bFi(1, IANDMA_CCR_CC(priv->DRC));
879:
880:     /* store values to dma registers */
```

```
881:     SET_IANDMA_REG(IANDMA_CBR_A, dma_cycle_budget);
882:     SET_IANDMA_REG(IANDMA_CWR_A(priv->DRC), dma_eth_rx_weight);
883:     SET_IANDMA_REG(IANDMA_CWR_A(priv->DTC), dma_eth_tx_weight);
884:     SET_IANDMA_REG(IANDMA_CCR_A, dma_ccr);
885:
886:     if (strt) {
887:         dma_cer = GET_IANDMA_REG(IANDMA_CER_A);
888:         dma_csr = GET_IANDMA_REG(IANDMA_CSR_A);
889:
890:         /* enable and start corresponding dma channels */
891:         dma_cer |= bFm(IANDMA_CER_EN(priv->DRC))
892:             | bFm(IANDMA_CER_EN(priv->DTC));
893:         dma_csr |= bFm(IANDMA_CSR_EN(priv->DRC))
894:             | bFm(IANDMA_CSR_EN(priv->DTC));
895:
896:         SET_IANDMA_REG(IANDMA_CER_A, dma_cer);
897:         SET_IANDMA_REG(IANDMA_CSR_A, dma_csr);
898:
899:         /* allocate memory for rx and rx buffers */
900:         for (i = 0; i < MACIAN_RX_BUFFERS ; i++)
901:             priv->rxd[i] =
902:                 (unsigned int *) kmalloc(MACIAN_BUF_SIZE, GFP_KERNEL);
903:         for (i = 0; i < MACIAN_RX_BUFFERS ; i++)
904:             priv->txd[i] =
905:                 (unsigned int *) kmalloc(MACIAN_BUF_SIZE, GFP_KERNEL);
906:
907:         priv->tx_next  = 0;
908:         priv->rx_next  = 0;
909:         priv->tx_count = 0;
910:
911:         /* Send Rx pointers to DMAC */
912:         for (i = 0; i < MACIAN_RX_BUFFERS ; i++)
913:             SET_IANDMA_REG(IANDMA_AF_A(priv->DRC),
914:                            IANDMA_MKADR(__pa(priv->rxd[i])));
915:     }
916: }
917:
918:
919: /* do_macian_probe: Initialize MACIAN */
920: static int
921: do_macian_probe(struct net_device *dev) {
922:     struct macian_priv *priv;
923:
924:     /* initialize the priv structure */
925:     priv = (struct macian_priv *)dev->priv;
926:     if (dev == NULL) {
927:         dev = alloc_etherdev(0);
928:         if (dev == NULL)
929:             return -ENOMEM;
930:     }
931:
932:     /* Initialize the device structure */
933:     if (dev->priv == NULL) {
934:         priv = kmalloc(sizeof(struct macian_priv), GFP_KERNEL);
935:         dev->priv = priv;
936:         if (dev->priv == NULL)
937:             return -ENOMEM;
938:     }
939:     __clear_user(priv, sizeof(*priv));
940:
941:
942:     /* setup the ethernet device */
943:     ether_setup(dev);
944:     dev->open            = macian_open;
945:     dev->hard_start_xmit = macian_start_xmit;
946:     dev->stop            = macian_close;
947:     dev->get_stats       = macian_get_stats;
948:
949:     if (register_netdev(dev)) {
950:         printk(KERN_ERR "macian: netdevice %s registration failed\n",
951:                dev->name);
952:         return -ENOMEM;
953:     }
954:
955:     //    SET_MODULE_OWNER(dev);
956:
957:     if (!strcmp(dev->name, "eth0")) {
958:         printk(KERN_ALERT "Setting up eth0\n");
959:         priv->ethnum = 0;
960:         priv->DRC = EOR;
961:         priv->DTC = EOT;
962:         priv->MACADDR0 = MAC0ADDR0;
963:         priv->MACADDR1 = MAC0ADDR1;
```

```
964:         priv->MACADDR2 = MAC0ADDR2;
965:         priv->MACADDR3 = MAC0ADDR3;
966:         priv->MACADDR4 = MAC0ADDR4;
967:         priv->MACADDR5 = MAC0ADDR5;
968:         printk(KERN_ALERT "%s: Assigned priv->ethnum = %d \n", dev->name, priv->ethnum );
969:     }
970:
971:     if (!strcmp(dev->name, "eth1")) {
972:         printk(KERN_ALERT "Setting up eth1\n");
973:         priv->ethnum = 1;
974:         priv->DRC = E1R;
975:         priv->DTC = E1T;
976:         priv->MACADDR0 = MAC1ADDR0;
977:         priv->MACADDR1 = MAC1ADDR1;
978:         priv->MACADDR2 = MAC1ADDR2;
979:         priv->MACADDR3 = MAC1ADDR3;
980:         priv->MACADDR4 = MAC1ADDR4;
981:         priv->MACADDR5 = MAC1ADDR5;
982:
983:         printk(KERN_ALERT "%s: Assigned priv->ethnum = %d \n",
984:                dev->name, priv->ethnum );
985:     }
986:
987:     printk("%s: MACIAN Ethernet Core Version 1.0\n", dev->name);
988:     spin_lock_init(& ((struct macian_priv *) dev->priv)->lock);
989:     return 0;
990: }
991:
992:
993:
994: static void
995: macian_timeout(struct net_device *dev) {
996:     struct macian_priv *priv = (struct macian_priv *)dev->priv;
997:     printk("Transmit timout at %lu, latency %lu\n", jiffies,
998:            jiffies - dev->trans_start);
999:     priv->stats.tx_errors++;
1000:     //    netif_wake_queue(dev);
1001: }
1002:
1003:
1004: /* module init function */
1005: static int macian_probe(void) {
1006:     int device_present = 0;
1007:     int device_start = 0;
1008:     unsigned int dma_eth_rx_weight, dma_eth_tx_weight, dma_cycle_budget;
1009:     printk(KERN_ALERT "Init: MACIAN Ethernet probe \n");
1010:
1011:     macian_dev[0] = alloc_etherdev(sizeof(struct macian_priv));
1012:     macian_dev[1] = alloc_etherdev(sizeof(struct macian_priv));
1013:     strcpy(macian_dev[0]->name, "eth%d");
1014:     strcpy(macian_dev[1]->name, "eth%d");
1015: #if 1
1016:     /* setup cycle budget */
1017:     dma_cycle_budget = bFi(0xf0, IANDMA_CBR_TB) |
1018:                        bFi(0xf0, IANDMA_CBR_RB) |
1019:                        bFi(0x50, IANDMA_CBR_IC);
1020:
1021:     /* set etherport weights */
1022:     dma_eth_rx_weight = bFi(0x40, IANDMA_CWR_CW);
1023:     dma_eth_tx_weight = bFi(0x40, IANDMA_CWR_CW);
1024: #endif
1025:
1026:     printk(KERN_ALERT "trying do_macian_probe\n");
1027:     device_start = do_macian_probe(macian_dev[0]);
1028:     if (!device_start)
1029:         device_present++;
1030:     else
1031:         printk("do_macian_probe failed for %s\n", macian_dev[0]->name);
1032:     device_start = do_macian_probe(macian_dev[1]);
1033:     if (!device_start)
1034:         device_present++;
1035:     else
1036:         printk("do_macian_probe failed for %s\n", macian_dev[1]->name);
1037:     return device_present ? 0 : -ENODEV;
1038: }
1039:
1040: /* module cleanup function */
1041: static void macian_cleanup(void) {
1042:     printk(KERN_ALERT "MACIAN Ethernet Core Cleanup\n");
1043:
1044:     if (macian_dev) {
1045:         unregister_netdev(macian_dev);
1046:         kfree(macian_dev);
```

```
1047:     }
1048: }
1049:
1050:
1051:
1052: static void
1053: read_macian_regs(void) {
1054:     macian_regs regs;
1055:     iandma_regs dregs;
1056:
1057:     /* read macian regs */
1058:     regs.min   = GET_MACIAN_REG(MACIAN_INT_A(0));
1059:     regs.isr   = GET_MACIAN_REG(MACIAN_ISR_A(0));
1060:     regs.imr   = GET_MACIAN_REG(MACIAN_IMR_A(0));
1061:     regs.tsr   = GET_MACIAN_REG(MACIAN_TSR_A(0));
1062:     regs.rsr   = GET_MACIAN_REG(MACIAN_RSR_A(0));
1063:     regs.mcr   = GET_MACIAN_REG(MACIAN_MCR_A(0));
1064:     regs.did   = GET_MACIAN_REG(MACIAN_DID_A(0));
1065:     regs.scr   = GET_MACIAN_REG(MACIAN_SCR_A(0));
1066:     regs.scc   = GET_MACIAN_REG(MACIAN_SCC_A(0));
1067:     regs.ftr   = GET_MACIAN_REG(MACIAN_FTR_A(0));
1068:     regs.rpr   = GET_MACIAN_REG(MACIAN_RPR_A(0));
1069:     regs.tpr   = GET_MACIAN_REG(MACIAN_TPR_A(0));
1070:     regs.flt   = GET_MACIAN_REG(MACIAN_FLT_A(0));
1071:     regs.rpt   = GET_MACIAN_REG(MACIAN_RPT_A(0));
1072:     regs.tpt   = GET_MACIAN_REG(MACIAN_TPT_A(0));
1073:     regs.ipg   = GET_MACIAN_REG(MACIAN_IPG_A(0));
1074:     regs.ral   = GET_MACIAN_REG(MACIAN_RAL_A(0));
1075:     regs.rah   = GET_MACIAN_REG(MACIAN_RAH_A(0));
1076:     regs.tal   = GET_MACIAN_REG(MACIAN_TAL_A(0));
1077:     regs.tah   = GET_MACIAN_REG(MACIAN_TAH_A(0));
1078:
1079:     /* read iandma regs */
1080:     dregs.cwr_rx      = GET_IANDMA_REG(IANDMA_CWR_A(EOR));
1081:     dregs.cwr_tx      = GET_IANDMA_REG(IANDMA_CWR_A(EOT));
1082:     dregs.cbr           = GET_IANDMA_REG(IANDMA_CBR_A);
1083:     dregs.ccr           = GET_IANDMA_REG(IANDMA_CCR_A);
1084:     dregs.imr_rx        = GET_IANDMA_REG(IANDMA_IMR_A(E0R));
1085:     dregs.imr_tx        = GET_IANDMA_REG(IANDMA_IMR_A(E0T));
1086:     dregs.cmr           = GET_IANDMA_REG(IANDMA_CMR_A);
1087:     dregs.cer           = GET_IANDMA_REG(IANDMA_CER_A);
1088:     dregs.csr           = GET_IANDMA_REG(IANDMA_CSR_A);
1089:     dregs.isr_rx        = GET_IANDMA_REG(IANDMA_ISR_A(E0R));
1090:     dregs.isr_tx        = GET_IANDMA_REG(IANDMA_ISR_A(E0T));
1091:     dregs.cnt_rx        = GET_IANDMA_REG(IANDMA_CNT_A(E0R));
1092:     dregs.cnt_tx        = GET_IANDMA_REG(IANDMA_CNT_A(E0T));
1093:
1094:     /* print debug information */
1095:     /* first the contents of macian registers */
1096:     printk("MACIAN registers\n");
1097:     printk("int  : 0x%08x\n", regs.min);
1098:     printk("isr  : 0x%08x\n", regs.isr);
1099:     printk("imr  : 0x%08x\n", regs.imr);
1100:     printk("tsr  : 0x%08x\n", regs.tsr);
1101:     printk("rsr  : 0x%08x\n", regs.rsr);
1102:     printk("mcr  : 0x%08x\n", regs.mcr);
1103:     printk("did  : 0x%08x\n", regs.did);
1104:     printk("scr  : 0x%08x\n", regs.scr);
1105:     printk("scc  : 0x%08x\n", regs.scc);
1106:     printk("ftr  : 0x%08x\n", regs.ftr);
1107:     printk("rpr  : 0x%08x\n", regs.rpr);
1108:     printk("tpr  : 0x%08x\n", regs.tpr);
1109:     printk("flt  : 0x%08x\n", regs.flt);
1110:     printk("rpt  : 0x%08x\n", regs.rpt);
1111:     printk("tpt  : 0x%08x\n", regs.tpt);
1112:     printk("ipg  : 0x%08x\n", regs.ipg);
1113:     printk("ral  : 0x%08x\n", regs.ral);
1114:     printk("rah  : 0x%08x\n", regs.rah);
1115:     printk("tal  : 0x%08x\n", regs.tal);
1116:     printk("tah  : 0x%08x\n", regs.tah);
1117:
1118:     /* then the contents of iandma registers */
1119:     printk("IANDMA registers (for eth0)\n");
1120:     printk("cwr_rx          : 0x%08x\n", dregs.cwr_rx);
1121:     printk("cwr_tx          : 0x%08x\n", dregs.cwr_tx);
1122:     printk("cbr             : 0x%08x\n", dregs.cbr);
1123:     printk("ccr             : 0x%08x\n", dregs.ccr);
1124:     printk("imr_rx          : 0x%08x\n", dregs.imr_rx);
1125:     printk("imr_tx          : 0x%08x\n", dregs.imr_tx);
1126:     printk("cmr             : 0x%08x\n", dregs.cmr);
1127:     printk("cer             : 0x%08x\n", dregs.cer);
1128:     printk("csr             : 0x%08x\n", dregs.csr);
1129:     printk("isr_rx          : 0x%08x\n", dregs.isr_rx);
```

```
1130:     printk("isr_tx          : 0x%08x\n", dregs.isr_tx);
1131:     printk("cnt_rx          : 0x%08x\n", dregs.cnt_rx);
1132:     printk("cnt_tx          : 0x%08x\n", dregs.cnt_tx);
1133: }
1134:
1135:
1136: module_init(macian_probe);
1137: module_exit(macian_cleanup);
1138:
1139: MODULE_AUTHOR("Ioannis Liverezas");
1140: MODULE_LICENSE("GPL");
1141:
1142: #ifdef MACIAN_DEBUG
1143: static void
1144: dumpbuf(char *buf, int len) {
1145:     int i = 0;
1146:     int temp = 0;
1147:     int wcount = 0;
1148:
1149:     for ( i = 0; i < len ; i+=4) {
1150:         temp = *(int *)&buf[i];
1151:         printk("%0x ", temp);
1152:         wcount++;
1153:         if (wcount == 8) {
1154:             printk("\n");
1155:             wcount = 0;
1156:         }
1157:     }
1158: }
1159: #endif
```

```
  1: /*****************************************************************
  2:  *  file        : iandes.c
  3:  *
  4:  *  description : Driver for IANDES, DES encryption
  5:  *                and decryption unit
  6:  *
  7:  *  author      : Ioannis Liverezas
  8:  *                      iliverez@inaccessnetworks.com
  9:  *
 10:  */
 11:
 12: #include <linux/config.h>
 13: #include <linux/module.h>
 14: #include <asm/irq.h>
 15: #include <asm/cacheflush.h>
 16: #include <linux/interrupt.h>
 17: #include <linux/sched.h>
 18: #include <linux/kernel.h>
 19: #include <linux/fs.h>
 20: #include <linux/errno.h>
 21: #include <linux/types.h>
 22: #include <linux/kdev_t.h>
 23: #include <linux/proc_fs.h>
 24: #include <asm/system.h>
 25: #include <asm/bitfield.h>
 26: #include <asm/iandma.h>
 27: #include <asm/uaccess.h>
 28: #include <linux/cdev.h>
 29: #include <asm/page.h>
 30: #include <linux/init.h>
 31: #include <linux/wait.h>
 32: #include <linux/fcntl.h>
 33: #include <linux/ioctl.h>
 34: #include <asm/ioctl.h>
 35: #include <asm/leon.h>
 36: #include <linux/sem.h>
 37: #include "iandes.h"
 38: #include "iandes_ioctl.h"
 39: #include <linux/major.h>
 40: #include <linux/kobject.h>
 41: #include <linux/devfs_fs_kernel.h>
 42: #include <linux/slab.h>
 43: #include <linux/module.h>
 44: #include <asm/leon.h>
 45: #include <linux/time.h>
 46:
 47: #define IANDES_MAJOR 242
 48:
 49:
 50: #ifndef __KERNEL__
 51: #  define __KERNEL__
 52: #endif
```

```
 53:
 54:
 55: MODULE_AUTHOR("Ioannis Liverezas");
 56:
 57:
 58:
 59:
 60: /* variables global for the driver */
 61: /* rx and tx pointers */
 62: static unsigned long *rx_pointers[MAX_DES_FIFO_LEN], *tx_pointers[MAX_DES_FIFO_LEN];
 63:
 64: /* lists that hold descriptors for the corresponding to DES
 65:  * DMA channels' fifo pointers
 66:  */
 67:
 68: struct list_head rx_fifo_free, rx_fifo_used;
 69: struct list_head tx_fifo_free, tx_fifo_used;
 70:
 71:
 72: /* this struct keeps track of which devices use the same flow */
 73: struct list_head dvs_for_flow[MAX_DES_FLOWS];
 74: int flow_devs_count[MAX_DES_FLOWS];
 75:
 76: /* Status of each flow, used for keys values updating */
 77: struct iandes_flow_lock flow_lock[MAX_DES_FLOWS];
 78: int flow_packets_count[MAX_DES_FLOWS];
 79:
 80: static struct iandes_device iandes_dev;
 81: int min_devs;
 82:
 83: int first_time = 0;
 84: int whofirst = 0;
 85: static int packets_to_serve;
 86:
 87: int pointer_lock;
 88:
 89:
 90:
 91:
 92:
 93:
 94: /* iandes_write: It is called when a userspace application issues a write command.
 95:  * The data ( = packet) is copied from a  userspace to a kernel buffer, and a
 96:  * pointer to this kernel space buffer is passed to DMA. Along with this transmit
 97:  * buffer, a read kernel space buffer is allocated and its address is stored
 98:  * in ptr_addrs_rx. When the encryptes/decrypted packet arrives from des to dma,
 99:  * the kernel finds the buffer's starting address from the fifo pointer
100:  * descriptor, and puts the buffer in the read buffer queue of the appropriate
101:  * device
102:  */
103:
104: ssize_t
105: iandes_write(struct file *filp, const char *buf, size_t count, loff_t *fpos)
106: {
107:     ssize_t ret = -ENOMEM;
108:     //  int i;
109:     unsigned int control_wd;
110:     int data_offt;
111:
112:     int tx_int_status;
113:     struct iandes_priv *priv;
114:     struct timeval intime, outtime, btv;
115:     long write_secs, write_usecs;
116:     struct fifo_ptr_desc *fptr;
117:     int nonblock;
118:     data_offt = 0;
119:     struct list_head     *rx_ptr, *tx_ptr;
120:     struct fifo_ptr_desc *rx_fifoptr, *tx_fifoptr;
121:     struct iandes_flow_lock *fl;
122:
123:
124:     priv = (struct iandes_priv *) filp->private_data;
125:
126: #if 0
127:     do_gettimeofday(&intime);
128: #endif
129:
130:     nonblock = filp->f_flags & O_NONBLOCK;
131:     if ((count % 8) != 0) {
132:         printk(KERN_ALERT "iandes write: wrong packet size\n");
133:         return -EFAULT;
134:     }
135:
```

```
136:     if (priv->write_lock == 1) {
137:         printk("iandes_write: oops, write_lock\n");
138:         return  -EAGAIN;
139:     }
140:
141:     priv->write_lock = 1;
142:     fl = (struct iandes_flow_lock *) &flow_lock[priv->fid];
143:
144:     /* set flow lock */
145:     if (nonblock) {
146:         if (set_flow_lock(priv) < 0) {
147:             priv->write_lock = 0;
148:             return -EAGAIN;
149:         }
150:
151:         set_flow_keys(priv, 0);
152:     }
153:
154:     /* get a free rx fifo pointer */
155:     rx_ptr = &rx_fifo_free;
156:     tx_ptr = &tx_fifo_free;
157:     if (nonblock) {
158:         if (pointer_lock == 1) {
159:             priv->write_lock = 0;
160:             printk(KERN_ALERT "iandes_write: pointer_lock\n");
161:             release_flow_lock(priv);
162:             return -EAGAIN;
163:         }
164:         pointer_lock = 1;
165:     }
166:
167:     while ( ((rx_ptr->next == &rx_fifo_free) && (rx_ptr->prev == &rx_fifo_free)) ||
168:             ((tx_ptr->next == &tx_fifo_free) && (tx_ptr->prev == &tx_fifo_free)))
169:     {
170:         if (nonblock) {
171:             priv->write_lock = 0;
172:             pointer_lock = 0;
173:             release_flow_lock(priv);
174:             printk(KERN_ALERT "iandes_write: pointer fifo full\n");
175:             return -EAGAIN;
176:         }
177:     }
178:
179:
180:     move_fifo_pointer(rx_fifo_free, rx_fifo_used, 0);
181:     rx_ptr = (struct list_head *) rx_fifo_used.next;
182:     rx_fifoptr = list_entry(rx_ptr, struct fifo_ptr_desc, used_list);
183:     move_fifo_pointer(tx_fifo_free, tx_fifo_used, 0);
184:     tx_ptr = tx_fifo_used.next;
185:     tx_fifoptr = list_entry(tx_ptr, struct fifo_ptr_desc, used_list);
186:
187:
188:     if (nonblock)
189:         pointer_lock = 0;
190:
191: #if 0
192:     printk(KERN_ALERT "devid = %d got rx ptr num %d and tx ptr num %d\n", priv->dev_id,
193:            rx_fifoptr->ptr_num, tx_fifoptr->ptr_num);
194:     /* initialize the rx pointer descriptor */
195: #endif
196:     rx_fifoptr->priv = (struct iandes_priv *) filp->private_data;
197:     rx_fifoptr->dec = priv->dec;
198:     rx_fifoptr->cbc = priv->cbc;
199:     rx_fifoptr->dev_id = priv->dev_id;
200:     rx_fifoptr->flow_id = priv->fid;
201:
202: #if 0
203:     printk(KERN_ALERT "dev %d:write:tx_ptr->data = %08x, rx->ptr_data = %08x\n",priv->dev_id,
204:            (unsigned int) tx_fifoptr->data, (unsigned int)rx_fifoptr->data);
205: #endif
206:
207:     tx_fifoptr->priv = (struct iandes_priv *) priv;
208:     tx_fifoptr->flow_id = priv->fid;
209:     tx_fifoptr->dev_id  = priv->dev_id;
210:     /* now create the packet to be sent to the dma */
211:     control_wd =
212:         bFi(priv->fid, IANDES_DCW_FID) |
213:         bFi(priv->dec, IANDES_DCW_DEC) |
214:         bFi(priv->tri, IANDES_DCW_3DE) |
215:         bFi(priv->hiv, IANDES_DCW_HIV) |
216:         bFi(priv->cbc, IANDES_DCW_CBC);
217:     data_offt = 4;
218:
```

```
219:     /* form the packet header. If cbc is enabled, add two 32bit words for
220:      * the IV (initialization Vector)  block. If hiv is 0, the IV value is
221:      * important only for the first packet that will be encrypted/decrypted.
222:      */
223:
224:     if (priv->cbc == 1) {
225:         if ((priv->hiv == 1) && (priv->dec == 0)) {
226:             *(unsigned int *)(&tx_fifoptr->data[4]) = priv->hiv1;
227:             *(unsigned int *)(&tx_fifoptr->data[8]) = priv->hiv2;
228:         }
229:         data_offt = 12;
230:         if (priv->dec == 1) {
231:             *(unsigned int *)(&tx_fifoptr->data[4]) = priv->hiv1;
232:             *(unsigned int *)(&tx_fifoptr->data[8]) = priv->hiv2;
233:             data_offt = 12;
234:         }
235:     }
236:
237:     tx_fifoptr->data_offt = data_offt;
238:
239:
240:
241:     __flush_page_to_ram((unsigned int) tx_fifoptr->data);
242:     if (copy_from_user((char *) &tx_fifoptr->data[data_offt], buf, count)) {
243:         ret = (ssize_t) -EFAULT;
244:         return ret;
245:     }
246:
247:     *(unsigned int *)(&tx_fifoptr->data[0]) = control_wd;
248:
249:     count += data_offt;
250:     ret = count - 4;
251:
252:
253: #if 0
254:     if (priv->measure == 1)
255:         return -EAGAIN;
256:     else {
257:         do_gettimeofday(&priv->btv);
258:         priv->measure = 1;
259:     }
260: #endif
261:
262:     /* set the size of the packet and send size and packet to the
263:      * appropriate dma channels
264:      */
265:     __flush_page_to_ram((unsigned int) rx_fifoptr->data);
266:     SET_IANDMA_REG(IANDMA_AF_A(D0R), IANDMA_MKADR(__pa((unsigned int)rx_fifoptr->data)));
267:
268:     SET_IANDMA_REG(IANDMA_SF_A(D0T), IANDMA_MKLEN(count));
269:     SET_IANDMA_REG(IANDMA_AF_A(D0T), IANDMA_MKADR(__pa((unsigned int) tx_fifoptr->data)));
270:
271:     /* increase the number of pending packets for the selected flow */
272:     fl->pending_pointers += 2;
273:     priv->pending += 2;
274:
275:
276:
277: #if 0
278:     do_gettimeofday(&outtime);
279:
280:     write_secs = outtime.tv_sec - intime.tv_sec;
281:     if(write_secs) {
282:         write_secs--;
283:         outtime.tv_usec = outtime.tv_usec + 1000000;
284:     }
285:     write_usecs = outtime.tv_usec -  intime.tv_usec;
286:     write_usecs += write_secs * 1000000;
287:     if (write_usecs <= priv->wmin_usecs)
288:         priv->wmin_usecs = write_usecs;
289:     if (write_usecs >= priv->wmax_usecs)
290:         priv->wmax_usecs = write_usecs;
291:     priv->wavg_usecs = priv->wavg_usecs + write_usecs;
292:     priv->wrtimes++;
293: #endif
294:
295:     priv->write_lock = 0;
296:     return ret;
297:
298: }
299:
300:
301: ssize_t
```

```
302: iandes_read(struct file *filp, char *buf, size_t count, loff_t *f_pos)
303: {
304:     ssize_t ret = 0;
305:     struct iandes_priv *priv;
306:     struct list_head *listptr, *ptr2;
307:
308:     struct fifo_ptr_desc *fpd, *item;
309:     int data_offt;
310:     char *ptr;
311:     int len;
312:     unsigned int tmp;
313:
314:     char *temp_ptr;
315:     unsigned int dec, cbc;
316:     struct timeval intime, outtime, ntime;
317:     long read_secs, read_usecs, ir_secs, ir_usecs;
318:     int bufdevid;
319:     int bufnum;
320:     priv = (struct iandes_priv *) filp->private_data;
321:
322:
323:
324:
325: #if 0
326:     /* Count the time that read takes to execute. Two times are measured:
327:      *    - The time between the end of the isr and the entrance of read (includes userspace time)
328:      *    - T
329:      */
330:     do_gettimeofday(&intime);
331:     do_gettimeofday(&ntime);
332:     ir_secs = ntime.tv_sec - priv->irtv.tv_sec;
333:     if (ir_secs) {
334:         ir_secs--;
335:         ntime.tv_usec = ntime.tv_usec + 1000000;
336:     }
337:     ir_usecs =  ntime.tv_usec - priv->irtv.tv_usec;
338:     ir_usecs = ir_usecs + ir_secs * 1000000;
339:     if (ir_usecs <= priv->irmin_usecs)
340:         priv->irmin_usecs = ir_usecs;
341:     if (ir_usecs >= priv->irmax_usecs)
342:         priv->irmax_usecs = ir_usecs;
343:     priv->iravg_usecs = priv->iravg_usecs + ir_usecs;
344:     priv->rdtimes++;
345: #endif
346:
347:
348:     if ((count % 8) != 0) {
349:         printk(KERN_ALERT "iandes_read: wrong packet size\n");
350:         return -EFAULT;
351:     }
352:
353:
354:     /* block until data to be read is ready. If no data is available, put the
355:      * reading process to sleep
356:      */
357:     while (priv->to_read_packet_num == 0) {
358:         return -EAGAIN;
359:     }
360:
361:     listptr = (struct list_head *) &priv->read_buf_list;
362:     listptr = listptr->prev;
363:     item = list_entry(listptr, struct fifo_ptr_desc, rxbuf_list);
364:     ptr = item->data;
365:     dec = item->dec;
366:     cbc = item->cbc;
367:     bufnum = item->ptr_num;
368:     bufdevid = item->dev_id;
369:     tmp = (unsigned int) ptr;
370:
371: #if 0
372:     printk(KERN_ALERT "iandes_read:dev %d will read data from ptr num %d for devid %d at address %08x \n",
373:            priv->dev_id, bufnum, bufdevid, (unsigned int)item->data);
374: #endif
375:
376:     /* adjust offset from which bytes will by copied to userspace buf */
377:     if (cbc == 1) {
378:         if (dec == 1) {
379:             data_offt = 12;
380:             ret = count - 8;
381:         } else if (dec == 0) {
382:             data_offt = 12;
383:             ret = count - 8;
384:         }
```

```
385:     }
386:     else {
387:         data_offt = 4;
388:         ret = count;
389:     }
390:
391:     temp_ptr = (char *) &ptr[data_offt];
392:     __flush_page_to_ram((unsigned int) temp_ptr);
393:     /* No perform the userspace copy */
394:     if (copy_to_user(buf, temp_ptr, ret)) {
395:         printk(KERN_ALERT "iandes_read: copy to user failed\n");
396:         return -EFAULT;
397:     }
398:
399:     /* remove node from read buffer list, free buffer and
400:      *   diminish to_read_packet_num
401:      */
402:
403:
404:     priv->to_read_packet_num--;
405:
406: #if 0
407:     listptr = &rx_fifo_used;
408:     for (listptr = listptr->next ; listptr != &rx_fifo_used; listptr = listptr->next) {
409:         fpd = list_entry(listptr, struct fifo_ptr_desc, used_list);
410:         if (bufnum == fpd->ptr_num) {
411:             list_del(&fpd->used_list);
412:             ptr2 = (struct list_head *) &rx_fifo_free;
413:             ptr2 = ptr2->next;
414:             list_add_tail(&fpd->free_list, ptr2);
415:         }
416:     }
417: #endif
418:
419:     move_fifo_pointer(priv->read_buf_list, rx_fifo_free, 3);
420:
421: #if 0
422:     do_gettimeofday(&outtime);
423:
424:     read_secs = outtime.tv_sec - intime.tv_sec;
425:     if (read_secs) {
426:         read_secs--;
427:         outtime.tv_usec = outtime.tv_usec + 1000000;
428:     }
429:     read_usecs =  outtime.tv_usec - intime.tv_usec;
430:     read_usecs = read_usecs + read_secs * 1000000;
431:     if (read_usecs <= priv->rmin_usecs)
432:         priv->rmin_usecs = read_usecs;
433:     if (read_usecs >= priv->rmax_usecs)
434:         priv->rmax_usecs = read_usecs;
435:     priv->ravg_usecs = priv->ravg_usecs + read_usecs;
436:     priv->rdtimes++;
437: #endif
438:
439:     return ret;
440: }
441:
442: /* DES can't generate interrupts on LEON by itself. Interrupts are generated
443:  * by DMA. The DES channels configuration suggests that in the reception path,
444:  * an interrupt will be generated when at least one packet is sent from DES to
445:  * the corresponding DMA channel. The kernel is not able to know which of the
446:  * open devices has sent the packet to des, and send back the received packet to
447:  * that device.
448:  *
449:  */
450:
451: static irqreturn_t
452: iandma_interrupt(int irq, void *dev_id, struct pt_regs *regs)
453: {
454:     static unsigned int rx_int_status, tx_int_status, fid;
455:     struct iandes_priv *priv;
456:
457:     struct list_head *ptr;
458:     struct list_head *buf_ptr;
459:     struct fifo_ptr_desc *fifo_descriptor;
460:     struct rx_buf *rx_buffer;
461:     struct iandes_device *desdev;
462:     struct iandes_flow_lock *fl;
463:     int tx_num;
464:     int rx_imr;
465:
466:     int ptr_num, len;
467:     int count = 0;
```

```
468:     int data_offt;
469:     int tmp_cnt;
470:
471:     struct timeval intime, outtime, int_time;
472:
473:     long int_secs, int_usecs;
474:     long buf_secs, buf_usecs;
475:
476:     //  do_gettimeofday(&intime);
477:
478:     len = 0;
479:     fid = 0;
480:
481:     spin_lock(&iandes_dev.lock);
482:     //  desdev = (struct iandes_device *) dev_id;
483:
484: #if 0
485:     do_gettimeofday(&int_time);
486:     do_gettimeofday(&intime);
487: #endif
488:     rx_imr = 0;
489:
490:     SET_IANDMA_REG(IANDMA_IMR_A(DOR), rx_imr);
491:
492:     /* Determine whether we have an rx or tx interrupt */
493:     rx_int_status = GET_IANDMA_REG(IANDMA_ISR_A(DOR));
494:     tx_int_status = GET_IANDMA_REG(IANDMA_ISR_A(DOT));
495:
496:
497:     /* Handle receive interrupt */
498:     if (rx_int_status & bFm(IANDMA_ISR_PC01)) {
499:         tmp_cnt = GET_IANDMA_REG(IANDMA_CNT_A(DOT));
500:
501:         //desdev->rx_ints++;
502:
503:         ptr = (struct list_head *)&tx_fifo_used;
504:         ptr = ptr->prev;
505:         fifo_descriptor = list_entry(ptr, struct fifo_ptr_desc, used_list);
506:         tx_num = fifo_descriptor->ptr_num;
507:         //      priv = (struct iandes_priv *) fifo_descriptor->priv;
508:
509: #if 0
510:         buf_secs  = int_time.tv_sec - priv->btv.tv_sec;
511:         if(buf_secs) {
512:             buf_secs--;
513:             int_time.tv_usec = int_time.tv_usec + 1000000;
514:         }
515:         buf_usecs = int_time.tv_usec - priv->btv.tv_usec;
516:         buf_usecs = buf_usecs + (buf_secs * 1000000);
517:         if (buf_usecs <= priv->bmin_usecs)
518:             priv->bmin_usecs = buf_usecs;
519:         if (buf_usecs > priv->bmax_usecs)
520:             priv->bmax_usecs = buf_usecs;
521:         priv->bavg_usecs = priv->bavg_usecs + buf_usecs;
522:         priv->measure = 0;
523: #endif
524:
525:         move_fifo_pointer(tx_fifo_used, tx_fifo_free, 1);
526:         iandes_dev.tx_packets += tmp_cnt;
527:         ptr = (struct list_head *) &rx_fifo_used;
528:         ptr = ptr->prev;
529:         fifo_descriptor = list_entry(ptr, struct fifo_ptr_desc, used_list);
530:         ptr_num = fifo_descriptor->ptr_num;
531:
532:         __flush_page_to_ram((unsigned int)fifo_descriptor->data);
533:         priv = (struct iandes_priv *) fifo_descriptor->priv;
534:
535: #if 0
536:         printk("received tx and rx pointer for devid = %d\n", fifo_descriptor->dev_id);
537:         printk("tx num = %d, rx_num = %d\n", tx_num, ptr_num);
538:
539:         printk("iandma_interrupt:rx_ptrr->data = %08x",
540:                (unsigned int) fifo_descriptor->data);
541: #endif
542:
543:         move_fifo_pointer(rx_fifo_used, priv->read_buf_list, 2);
544:         priv->to_read_packet_num++;
545:         iandes_dev.rx_packets += GET_IANDMA_REG(IANDMA_CNT_A(DOR));
546:         priv->pending -= 2;
547:         fl = (struct iandes_flow_lock *) &flow_lock[priv->fid];
548:         fl->pending_pointers -= 2;
549:         release_flow_lock(priv);
550:
```

```
551: #if 0
552:         do_gettimeofday(&outtime);
553:
554:         int_secs = outtime.tv_sec - intime.tv_sec;
555:         if(int_secs) {
556:             int_secs--;
557:             outtime.tv_usec = outtime.tv_usec + 1000000;
558:         }
559:         int_usecs = outtime.tv_usec - intime.tv_usec;
560:         int_usecs = int_usecs + (int_secs * 1000000);
561:         if (int_usecs <= priv->imin_usecs)
562:             priv->imin_usecs = int_usecs;
563:         if (int_usecs >= priv->imax_usecs)
564:             priv->imax_usecs = int_usecs;
565:         priv->iavg_usecs = priv->iavg_usecs + int_usecs;
566:         priv->isrtimes++;
567:
568:         do_gettimeofday(&priv->irtv);
569: #endif
570:     }
571:     spin_unlock(&iandes_dev.lock);
572:     rx_imr = 0x4;
573:     SET_IANDMA_REG(IANDMA_IMR_A(DOR), rx_imr);
574:     return IRQ_HANDLED;
575: }
576:
577:
578:
579: /*  iandes_ioctl : contains all the ioctl commands that program DES or get info
580:  *  abouts its current state.
581:  */
582: int
583: iandes_ioctl(struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)
584: {
585:     int err = 0;
586:     int ret = 0;
587:     unsigned int fid, dec, tri, hiv, cbc, ctrlwd;
588:     int hiv1, hiv2;
589:     struct iandes_priv *priv;
590:     struct iandes_keys *keys_ptr;
591:     struct iandes_key *key_ptr;
592:     struct iandes_iv  *iv_ptr;
593:     struct iandes_params *params_ptr;
594:     int keysaddr, keyaddr, ivaddr, params_addr;
595:     int *ptr;
596:     struct timeval intime, outtime;
597:     unsigned long ioctl_secs, ioctl_usecs;
598:     int nonblock;
599:     int keylen;
600:     int session_addr;
601:     struct session_op *session_ptr;
602:     int cryp_addr;
603:     struct iandes_crypto *cryp_ptr;
604: #if 0
605:     do_gettimeofday(&intime);
606: #endif
607:
608:     /* make sure that the command is correct */
609:     if (_IOC_TYPE(cmd) != IANDES_IO_MAGIC) {
610:         printk("_IOC_TYPE doesn't match IANDES_IO_MAGIC\n");
611:         return -ENOTTY;
612:     }
613:
614:     if (_IOC_NR(cmd) > MAX_DES_IO_NR) {
615:         printk("_IOC_NR > MAX_DES_IO_NR\n");
616:         return -ENOTTY;
617:     }
618:
619:     if (_IOC_DIR(cmd) & _IOC_READ)
620:         err = !access_ok(VERIFY_WRITE, (void *)arg, _IOC_SIZE(cmd));
621:     else
622:         if (_IOC_DIR(cmd) & _IOC_WRITE)
623:             err = !access_ok(VERIFY_READ, (void *)arg, _IOC_SIZE(cmd));
624:
625:     if (err) {
626:         printk("in iandes_ioctl: could not get accessok, for cmd num %d \n", cmd);
627:         return -EFAULT;
628:     }
629:
630:
631:     /* now, the implementation of the commands */
632:     priv = (struct iandes_priv *) filp->private_data;
633:     nonblock = priv->nonblock;
```

```
634:
635:     switch(cmd) {
636:     case IANDES_IO_RESET:
637:         priv->dec  = 0;
638:         priv->tri  = 0;
639:         priv->hiv  = 0;
640:         priv->cbc  = 0;
641:         priv->hiv1 = 0;
642:         priv->hiv2 = 0;
643:
644:         priv->key1a = 0;
645:         priv->key1b = 0;
646:         priv->key2a = 0;
647:         priv->key2b = 0;
648:         priv->key3a = 0;
649:         priv->key3b = 0;
650:
651:         /*  set_flow_keys(priv);*/
652:         break;
653:
654:     case IANDES_IO_SETKEYS:
655:         /* now set the values to the keys */
656:         keysaddr = arg;
657:         keys_ptr = (struct iandes_keys *) keysaddr;
658:         if (nonblock)
659:             if ( set_flow_lock(priv) < 0) {
660:                 printk(KERN_ALERT "IANDES_IO_SETKEYS: flow locked\n");
661:                 return -EAGAIN;
662:             }
663:         priv->key1a = keys_ptr->key1a;
664:         priv->key1b = keys_ptr->key1b;
665:         priv->key2a = keys_ptr->key2a;
666:         priv->key2b = keys_ptr->key2b;
667:         priv->key3a = keys_ptr->key3a;
668:         priv->key3b = keys_ptr->key3b;
669:         set_flow_keys(priv, 1);
670:
671:         break;
672:
673:     case IANDES_IO_GETKEYS:
674:         keysaddr = arg;
675:         keys_ptr = (struct iandes_keys *) keysaddr;
676:         if (nonblock)
677:             if ( set_flow_lock(priv) < 0) {
678:                 printk(KERN_ALERT "IANDES_GETKEYS: flow locked\n");
679:                 return -EAGAIN;
680:             }
681:         keys_ptr->key1a = priv->key1a;
682:         keys_ptr->key1b = priv->key1b;
683:         keys_ptr->key2a = priv->key2a;
684:         keys_ptr->key2b = priv->key2b;
685:         keys_ptr->key3a = priv->key3a;
686:         keys_ptr->key3b = priv->key3b;
687:         break;
688:
689:     case IANDES_IO_SETKEY1:
690:         keyaddr = arg;
691:         key_ptr = (struct iandes_key *) keyaddr;
692:         if (nonblock)
693:             if ( set_flow_lock(priv) < 0) {
694:                 printk(KERN_ALERT "IANDES_IO_SETKEY1: flow locked\n");
695:                 return -EAGAIN;
696:             }
697:         priv->key1a = key_ptr->keya;
698:         priv->key1b = key_ptr->keyb;
699:         set_flow_keys(priv, 1);
700:
701:         break;
702:
703:     case IANDES_IO_SETKEY2:
704:         keyaddr = arg;
705:         key_ptr = (struct iandes_key *) keyaddr;
706:         if (nonblock)
707:             if ( set_flow_lock(priv) < 0) {
708:                 printk(KERN_ALERT "IANDES_IO_SETKEY2: flow locked\n");
709:                 return -EAGAIN;
710:             }
711:         priv->key2a = key_ptr->keya;
712:         priv->key2b = key_ptr->keyb;
713:         set_flow_keys(priv, 1);
714:         kfree(key_ptr);
715:         break;
716:
```

```
717:     case IANDES_IO_SETKEY3:
718:         if (nonblock)
719:             if ( set_flow_lock(priv) < 0) {
720:                 printk(KERN_ALERT "IANDES_IO_SETKEY3: flow locked\n");
721:                 return -EAGAIN;
722:             }
723:         keyaddr = arg;
724:         key_ptr = (struct iandes_key *) keyaddr;
725:         priv->key3a = key_ptr->keya;
726:         priv->key3b = key_ptr->keyb;
727:         set_flow_keys(priv, 1);
728:         kfree(key_ptr);
729:         break;
730:
731:     case IANDES_IO_GETKEY1:
732:         keyaddr = arg;
733:         key_ptr = (struct iandes_key *) keyaddr;
734:         key_ptr->keya = priv->key1a;
735:         key_ptr->keyb = priv->key1b;
736:
737:         break;
738:
739:     case IANDES_IO_GETKEY2:
740:         keyaddr = arg;
741:         key_ptr = (struct iandes_key *) keyaddr;
742:         key_ptr->keya = priv->key2a;
743:         key_ptr->keyb = priv->key2b;
744:         break;
745:
746:     case IANDES_IO_GETKEY3:
747:         //  printk("in IANDES_IO_GETKEY3\n");
748:         keyaddr = arg;
749:         key_ptr = (struct iandes_key *) keyaddr;
750:         key_ptr->keya = priv->key3a;
751:         key_ptr->keyb = priv->key3b;
752:         break;
753:
754:     case IANDES_IO_SETCTRLWD:
755:         ret = __get_user(ctrlwd, (int *) arg);
756:         if (ret) {
757:             up(&priv->sem);
758:             return ret;
759:         }
760:         fid = DESPCK_CWD_FID(ctrlwd);
761:         if (priv->fid != fid) {
762:             ret = -2;
763:             up(&priv->sem);
764:             return ret;
765:         }
766:         priv->dec = DESPCK_CWD_DEC(ctrlwd);
767:         priv->tri = DESPCK_CWD_3DE(ctrlwd);
768:         priv->hiv = DESPCK_CWD_HIV(ctrlwd);
769:         priv->cbc = DESPCK_CWD_CBC(ctrlwd);
770:         break;
771:
772:     case IANDES_IO_SETDEC:
773:         ret = __get_user(dec, (int *) arg);
774:         if (ret) {
775:             printk("get user failed \n");
776:             up(&priv->sem);
777:             return ret;
778:         }
779:         priv->dec = dec;
780:         break;
781:
782:     case IANDES_IO_SET3DE:
783:         ret = __get_user(tri, (int *) arg);
784:         if (ret) {
785:             up(&priv->sem);
786:             return ret;
787:         }
788:         priv->tri = tri;
789:         break;
790:
791:     case IANDES_IO_SETHIV:
792:         ret = __get_user(hiv, (int *) arg);
793:         if (ret) {
794:             up(&priv->sem);
795:             return ret;
796:         }
797:         priv->hiv = hiv;
798:         break;
799:
```

```
800:     case IANDES_IO_SETCBC:
801:         ret = __get_user(cbc, (int *) arg);
802:         if (ret) {
803:             up(&priv->sem);
804:             return ret;
805:         }
806:         priv->cbc = cbc;
807:         break;
808:
809:     case IANDES_IO_GETCTRLWD:
810:         ctrlwd =
811:             bFi(priv->fid, IANDES_DCW_FID) |
812:             bFi(priv->dec, IANDES_DCW_DEC) |
813:             bFi(priv->tri, IANDES_DCW_3DE) |
814:             bFi(priv->hiv, IANDES_DCW_HIV) |
815:             bFi(priv->cbc, IANDES_DCW_CBC);
816:         ret = __put_user(ctrlwd, (int *) arg);
817:         if (ret) {
818:             up(&priv->sem);
819:             return ret;
820:         }
821:         break;
822:
823:     case IANDES_IO_GETDEC:
824:         dec = priv->dec;
825:         ret = __put_user(dec, (int *) arg);
826:         if (ret) {
827:             up(&priv->sem);
828:             return ret;
829:         }
830:         break;
831:
832:     case IANDES_IO_GET3DE:
833:         tri = priv->tri;
834:         ret = __put_user(tri, (int *) arg);
835:         if (ret) {
836:             up(&priv->sem);
837:             return ret;
838:         }
839:         break;
840:
841:     case IANDES_IO_GETHIV:
842:         hiv = priv->hiv;
843:         ret = __put_user(hiv, (int *) arg);
844:         if (ret) {
845:             up(&priv->sem);
846:             return ret;
847:         }
848:         break;
849:
850:     case IANDES_IO_GETCBC:
851:         cbc = priv->cbc;
852:         ret = __put_user(cbc, (int *) arg);
853:         if (ret) {
854:             up(&priv->sem);
855:             return ret;
856:         }
857:         break;
858:
859:     case IANDES_IO_SETIV1:
860:         ret = __get_user(hiv1, (int *) arg);
861:         if (ret) {
862:             printk("get user failed \n");
863:             up(&priv->sem);
864:             return ret;
865:         }
866:         priv->hiv1 = hiv1;
867:         break;
868:
869:     case IANDES_IO_SETIV2:
870:         ret = __get_user(hiv2, (int *) arg);
871:         if (ret) {
872:             printk("get user failed \n");
873:             up(&priv->sem);
874:             return ret;
875:         }
876:         priv->hiv2 = hiv2;
877:         break;
878:
879:     case IANDES_IO_GETIV1:
880:         hiv1 = priv->hiv1;
881:         ret = __put_user(hiv1, (int *) arg);
882:         if (ret) {
```

```
883:             up(&priv->sem);
884:             return ret;
885:         }
886:         break;
887:
888:     case IANDES_IO_GETIV2:
889:         hiv2 = priv->hiv2;
890:         ret = __put_user(hiv2, (int *) arg);
891:         if (ret) {
892:             up(&priv->sem);
893:             return ret;
894:         }
895:         break;
896:
897:     case IANDES_IO_GETIV:
898:         ivaddr = arg;
899:         iv_ptr = (struct iandes_iv *) ivaddr;
900:         iv_ptr->iv1 = priv->hiv1;
901:         iv_ptr->iv2 = priv->hiv2;
902:         break;
903:
904:     case IANDES_IO_SETIV:
905:         ivaddr = arg;
906:         iv_ptr = (struct iandes_iv *) ivaddr;
907:         set_flow_lock(priv);
908:         priv->hiv1 = iv_ptr->iv1;
909:         priv->hiv2 = iv_ptr->iv2;
910:         set_flow_keys(priv, 1);
911:         break;
912:
913:     case IANDES_IO_SETPARAMS:
914:         params_addr = arg;
915:         params_ptr = (struct iandes_params *) params_addr;
916:         priv->dec = params_ptr->dec;
917:         priv->tri = params_ptr->tri;
918:         priv->hiv = params_ptr->hiv;
919:         priv->cbc = params_ptr->cbc;
920:         break;
921:     case IANDES_IO_GETPARAMS:
922:         params_addr = arg;
923:         params_ptr = (struct iandes_params *) params_addr;
924:         params_ptr->dec = priv->dec;
925:         params_ptr->tri = priv->tri;
926:         params_ptr->hiv = priv->hiv;
927:         params_ptr->cbc = priv->cbc;
928:         break;
929:     case IANDES_IO_SETSESSION:
930:         session_addr = arg;
931:         session_ptr = (struct session_op *) session_addr;
932:         keylen = session_ptr->keylen;
933:         if (keylen == 8) {
934:             priv->key1a = *(unsigned int *) &session_ptr->key[0];
935:             priv->key1b = *(unsigned int *) &session_ptr->key[4];
936:         }
937:         else {
938:             priv->key1a = *(unsigned int *) &session_ptr->key[0];
939:             priv->key1b = *(unsigned int *) &session_ptr->key[4];
940:             priv->key2a = *(unsigned int *) &session_ptr->key[8];
941:             priv->key2b = *(unsigned int *) &session_ptr->key[12];
942:             priv->key3a = *(unsigned int *) &session_ptr->key[16];
943:             priv->key3b = *(unsigned int *) &session_ptr->key[20];
944:         }
945:         set_flow_keys(priv, 1);
946:         break;
947:     case IANDES_IO_SETCRYP:
948:         cryp_addr = arg;
949:         cryp_ptr = (struct iandes_crypto *) cryp_addr;
950:         priv->dec = cryp_ptr->dec;
951:         priv->tri = cryp_ptr->tri;
952:         priv->hiv = cryp_ptr->hiv;
953:         priv->cbc = cryp_ptr->cbc;
954:         if (priv->cbc) {
955:             priv->hiv1 = *(unsigned int *) &cryp_ptr->iv[0];
956:             priv->hiv2 = *(unsigned int *) &cryp_ptr->iv[4];
957:         }
958:
959:         break;
960:     default:
961:         printk("in iandes_ioctl: default. You shouldn't be here. Something is wrong\n");
962:         up(&priv->sem);
963:         return -ENOTTY;
964:     }
965:
```

```
966: #if 0
967:     do_gettimeofday(&outtime);
968:
969:     ioctl_secs = outtime.tv_sec - intime.tv_sec;
970:     if(ioctl_secs) {
971:         ioctl_secs--;
972:         outtime.tv_usec = outtime.tv_usec + 1000000;
973:     }
974:     ioctl_usecs = outtime.tv_usec -  intime.tv_usec;
975:     ioctl_usecs += ioctl_secs * 1000000;
976:     if (ioctl_usecs <= priv->iomin_usecs)
977:         priv->iomin_usecs = ioctl_usecs;
978:     if (ioctl_usecs >= priv->iomax_usecs)
979:         priv->iomax_usecs = ioctl_usecs;
980:     priv->ioavg_usecs = priv->ioavg_usecs + ioctl_usecs;
981: #endif
982:
983: #if 0
984:     up(&priv->sem);
985: #endif
986:     return ret;
987:
988:
989: }
990:
991:
992:
993: /* inades_open: Opens a des device. All devices use the same device file. When open
994:  * is called multiple times, multiple instances of the device are created
995:  */
996:
997:
998: static int
999: iandes_open(struct inode *inode, struct file *filp)
1000: {
1001:
1002:
1003:     /* Allocate and initialize the priv structure for a specific instance of the
1004:      * opened device
1005:      */
1006:
1007:     struct iandes_priv *priv;
1008:
1009:     priv = kmalloc(sizeof(struct iandes_priv), GFP_KERNEL);
1010:     (struct iandes_priv *) filp->private_data  = priv;
1011:
1012:     if (filp->private_data == NULL) {
1013:         printk("iandes: Not enough memory to allocate priv structure\n");
1014:         return -ENOMEM;
1015:     }
1016:
1017:     __clear_user(priv, sizeof(*priv));
1018:
1019:     iandes_dev.dev_ids++;
1020:     priv->dev_id = iandes_dev.dev_ids;
1021:
1022:
1023:     priv->fid = get_flow_id(priv->dev_id);
1024:     priv->nonblock = filp->f_flags & O_NONBLOCK;
1025:     priv->write_lock = 0;
1026:     MOD_INC_USE_COUNT;
1027:
1028:     priv->dec = 0;   /* default direction is encryption */
1029:     priv->tri = 0;   /* simple des                      */
1030:     priv->hiv = 0;
1031:     priv->cbc = 0;   /* use ecb. HIV value is indifferent */
1032:
1033:
1034:     priv->imin_usecs = 0;   priv->imax_usecs = 0;   priv->iavg_usecs = 0;
1035:     priv->rmin_usecs = 0;   priv->rmax_usecs = 0;   priv->ravg_usecs = 0;
1036:     priv->wmin_usecs = 0;   priv->wmax_usecs = 0;   priv->wavg_usecs = 0;
1037:     priv->bmin_usecs = 0;   priv->bmax_usecs = 0;   priv->bavg_usecs = 0;
1038:
1039:     priv->wrtimes = 0;
1040:     priv->rdtimes = 0;
1041:     priv->isrtimes = 0;
1042:     priv->measure = 0;
1043:     priv->pending = 0;
1044:
1045:     /* Initialize the read buffer list for the device. Upon receive interrupt, pointers
1046:      * to the buffers that will be read by the userspace application are inserted in
1047:      * read_buf_list of the device
1048:      */
```

```
1049:
1050:     INIT_LIST_HEAD(&priv->read_buf_list);
1051:
1052:     /* initialize the wait queue for read buffers */
1053:     priv->to_read_packet_num = 0;
1054:     return 0;
1055: }
1056:
1057:
1058: /* iandes_release: closes an instance of DES device. It has the responsibility
1059:  * to remove the elements that the device inserted in the driver's global lists,
1060:  * reduce the module use count, and destroy the file descriptor for this instance
1061:  */
1062: static int iandes_release(struct inode *inode, struct file *filp)
1063: {
1064:     unsigned int fid;
1065:     int dev_id, i;
1066:     struct iandes_priv *priv;
1067:     struct list_head *listptr, *ptr;
1068:     struct rx_buf *item;
1069:     struct devices_per_flow *item_dpf;
1070:
1071:
1072:     priv = (struct iandes_priv *) filp->private_data;
1073:     dev_id = priv->dev_id;
1074:     fid = priv->fid;
1075:
1076: #if 0
1077:     printk(KERN_ALERT "tx ints = %d \n", iandes_dev.tx_ints);
1078:     printk(KERN_ALERT "rx ints = %d \n", iandes_dev.rx_ints);
1079:
1080:
1081:     printk(KERN_ALERT "rdtimes = %d\n", priv->rdtimes);
1082:     printk(KERN_ALERT "wrtimes = %d\n", priv->wrtimes);
1083:     printk(KERN_ALERT "isrtimes = %d\n", priv->isrtimes);
1084:
1085:
1086:     printk(KERN_ALERT "imin = %ud    imax = %ud      iavg = %ud\n",
1087:           priv->imin_usecs, priv->imax_usecs, priv->iavg_usecs);
1088:     printk(KERN_ALERT "wmin = %ud    wmax = %ud      wavg = %ud\n",
1089:           priv->wmin_usecs, priv->wmax_usecs, priv->wavg_usecs);
1090:     printk(KERN_ALERT "rmin = %ud    rmax = %ud      ravg = %ud\n",
1091:           priv->rmin_usecs, priv->rmax_usecs, priv->ravg_usecs);
1092:
1093:     printk(KERN_ALERT "bmin = %ud    bmax = %ud      bavg = %ud\n",
1094:           priv->bmin_usecs, priv->bmax_usecs, priv->bavg_usecs);
1095:     printk(KERN_ALERT "irmin = %ud    irmax = %ud     iravg = %ud\n",
1096:           priv->irmin_usecs, priv->irmax_usecs, priv->iravg_usecs);
1097:     printk(KERN_ALERT "iomin = %ud    iomax = %ud     ioavg = %ud\n",
1098:           priv->iomin_usecs, priv->iomax_usecs, priv->ioavg_usecs);
1099: #endif
1100:
1101:     ptr = (struct list_head *) &dvs_for_flow[fid];
1102:     for (ptr = ptr->next ; ptr != &dvs_for_flow[fid] ; ptr = ptr->next) {
1103:         item_dpf = list_entry(ptr, struct devices_per_flow, list);
1104:         if (item_dpf->dev_id == dev_id) {
1105:             list_del(&item_dpf->list);
1106:             kfree(item_dpf);
1107:             break;
1108:         }
1109:     }
1110:
1111:
1112:     while (priv->to_read_packet_num > 0) {
1113:         listptr = (struct list_head *) &priv->read_buf_list;
1114:         if ((listptr = listptr->prev)!= &priv->read_buf_list) {
1115:             item = list_entry(listptr, struct rx_buf, rxbuf_list);
1116:             list_del(listptr);
1117:             priv->to_read_packet_num--;
1118:             move_fifo_pointer(rx_fifo_used, rx_fifo_free, 1);
1119:         }
1120:     }
1121:
1122:     /* now free the iandes_priv sructure of the device */
1123:     kfree(priv);
1124:
1125:     /* decrease module usage counter */
1126:
1127:     MOD_DEC_USE_COUNT;
1128:     return 0;
1129: }
1130:
1131:
```

```
1132: /* iandes_init_module  : Initializes DMA parameters and the DMA chanells for
1133:  * DES. It also initializes all the necessary stuff that is used throught the
1134:  * driver
1135:  */
1136:
1137: static int
1138: iandes_init_module(void)
1139: {
1140:     int result, i;
1141:     dev_t from;
1142:     int iandes_major;
1143:     static unsigned int dma_cycle_budget, dma_des_rx_weight, dma_des_tx_weight;
1144:     static unsigned int dma_ccr, dma_cer, dma_csr, dma_des_imr_rx, dma_des_imr_tx;
1145:     static volatile int tmp;
1146:
1147:     iandes_major = IANDES_MAJOR;
1148:     from = MKDEV(iandes_major, 0);
1149:     result = register_chrdev(iandes_major , (char *)"des", &iandes_fops);
1150:
1151:     if (result < 0) {
1152:         printk("iandes: can't get major %d\n", iandes_major);
1153:         return result;
1154:     }
1155:
1156:     printk("IANDES module registered.\n");
1157:     /* Initialize DMA, so that access to DES is possible */
1158:
1159:     /* setup cycle budget */
1160:     dma_cycle_budget =
1161:         bFi(0xf0, IANDMA_CBR_TB) |
1162:         bFi(0xf0, IANDMA_CBR_RB) |
1163:         bFi(0x50, IANDMA_CBR_IC);
1164:
1165:     /* setup des weights */
1166:     dma_des_rx_weight = bFi(0x40, IANDMA_CWR_CW);
1167:     dma_des_tx_weight = bFi(0x40, IANDMA_CWR_CW);
1168:
1169:     /* start the dma controller */
1170:     dma_ccr = bFi(1, IANDMA_CCR_SS) | bFi(0, IANDMA_CCR_CC(DOR));
1171:     dma_cer = GET_IANDMA_REG(IANDMA_CER_A);
1172:     dma_csr = GET_IANDMA_REG(IANDMA_CSR_A);
1173:
1174:     dma_cer |= bFm(IANDMA_CER_EN(DOR)) | bFm(IANDMA_CER_EN(DOT));
1175:     dma_csr |= bFm(IANDMA_CSR_EN(DOR)) | bFm(IANDMA_CSR_EN(DOT));
1176:
1177:     /* get interrupt when 1 pakcet is received /transmitted */
1178:     dma_des_imr_rx = 0x4;
1179:     dma_des_imr_tx = 0;
1180:
1181:     /* store values to DMA registers */
1182:     SET_IANDMA_REG(IANDMA_CBR_A, dma_cycle_budget);
1183:     SET_IANDMA_REG(IANDMA_CWR_A(DOR), dma_des_rx_weight);
1184:     SET_IANDMA_REG(IANDMA_CWR_A(DOT), dma_des_tx_weight);
1185:     SET_IANDMA_REG(IANDMA_CCR_A, dma_ccr);
1186:     SET_IANDMA_REG(IANDMA_CER_A, 0);
1187:     SET_IANDMA_REG(IANDMA_CSR_A, 0);
1188:     SET_IANDMA_REG(IANDMA_CER_A, dma_cer);
1189:     SET_IANDMA_REG(IANDMA_CSR_A, dma_csr);
1190:     SET_IANDMA_REG(IANDMA_IMR_A(DOR), dma_des_imr_rx);
1191:     SET_IANDMA_REG(IANDMA_IMR_A(DOT), dma_des_imr_tx);
1192:
1193:     /* clear possible pending interrupts for DES channels */
1194:     tmp = GET_IANDMA_REG(IANDMA_CNT_A(DOR));
1195:     tmp = GET_IANDMA_REG(IANDMA_CNT_A(DOT));
1196:
1197:     /* Initialize local fifos for DMA communication.
1198:      * We use four fifos:
1199:      * One fifo for free pointers and one for used pointers for the rx direction
1200:      * and one identical pair for the tx direction.
1201:      * When transmitting a packet, one pointer is popped out of the tx free list and
1202:      * inserted in the used list. This pointer is passed to DMA . In combination with
1203:      * this pointer, one pointer is pulled out of the rx free list, inserted in the
1204:      * rx used list, and passed to DMA. When the encryption/decryption operation is
1205:      * complete, the packet is sent back from des to dma and an interrupt occurs.
1206:      * The packet is copied and the pair of tx and rx pointers that were used leave
1207:      * the "used" fifos and get into the "free" ones. That way, synchronization is
1208:      * achieved and transfers are expected to be correct. Moreover, DMA and DES fifo
1209:      * overflow is avoided
1210:      */
1211:
1212:     INIT_LIST_HEAD(&rx_fifo_free);
1213:     INIT_LIST_HEAD(&rx_fifo_used);
1214:     INIT_LIST_HEAD(&tx_fifo_free);
```

```
1215:     INIT_LIST_HEAD(&tx_fifo_used);
1216:
1217:     /* fill rx and tx free fifos with local pointers */
1218:     fill_fifo_desc();
1219:
1220:     /* initialize fields of iandes_dev structure */
1221:     iandes_dev.flows = 0;
1222:     iandes_dev.rx_packets = 0;
1223:     iandes_dev.tx_packets = 0;
1224:     iandes_dev.dev_ids = 0;
1225:     iandes_dev.opendevs = 0;
1226:
1227:     iandes_dev.tx_ints = 0;
1228:     iandes_dev.rx_ints = 0;
1229:
1230:     /* initialize variables that are used for flow allocation */
1231:     packets_to_serve = 0;
1232:     min_devs = 0;
1233:
1234:     for (i = 0; i < MAX_DES_FLOWS; i++) {
1235:         INIT_LIST_HEAD(&dvs_for_flow[i]);
1236:         flow_devs_count[i] = 0;
1237:         iandes_dev.len[i] = 0;
1238:         flow_lock[i].lock = 0;
1239:         flow_lock[i].lock_owner = -1;
1240:         flow_lock[i].change_request = 0;
1241:         flow_lock[i].pending_pointers = 0;
1242:         flow_lock[i].dev_counter = 0;
1243:     }
1244:
1245:
1246:     request_irq(IANDMA_INTERRUPT, iandma_interrupt, 0, "des", (void *) &iandes_dev);
1247:     //request_irq(IANDMA_INTERRUPT, iandma_interrupt, 0, "des", NULL);
1248:     up(&iandes_dev.sem);
1249:     spin_lock_init(&iandes_dev.lock);
1250:     return 0;
1251:
1252: }
1253:
1254: static void
1255: iandes_cleanup_module(void)
1256: {
1257:     int iandes_major;
1258:     printk(KERN_ALERT "IANDES_Tx: Sent %d packets\n", iandes_dev.tx_packets);
1259:     printk(KERN_ALERT "IANDES_Rx: Received %d packets\n", iandes_dev.rx_packets);
1260:     iandes_major = IANDES_MAJOR;
1261:     unregister_chrdev(iandes_major, "des");
1262:
1263: }
1264:
1265:
1266:
1267: /*-----------------------------------------------------------------------------*/
1268: /*
1269:  * Below follow some helper functions
1270:  *
1271:  */
1272:
1273:
1274: /* fill a fifo with fifo pointers descriptors. It used at module initialization
1275:  * to fill the rx_fifo_free, tx_fifo_free queues
1276:  */
1277: static void
1278: fill_fifo_desc(void)
1279: {
1280:
1281:     struct fifo_ptr_desc *item;
1282:     int i;
1283:
1284:     /* fill rx_fifo_free first */
1285:     for (i =0 ; i < MAX_DES_FIFO_LEN ; i++) {
1286:         rx_pointers[i] = (unsigned long *) kmalloc(MAX_DES_BUF, GFP_KERNEL);
1287:         item = (struct fifo_ptr_desc *)
1288:             kmalloc(sizeof(struct fifo_ptr_desc), GFP_KERNEL);
1289:         item->data = (unsigned char *) rx_pointers[i];
1290:         item->ptr_num = i;
1291:         item->len = 0;
1292:         item->data_offt = 0;
1293:         add_fifo_pointer(rx_fifo_free, item);
1294:     }
1295:
1296:     /* then tx_fifo_free */
1297:
```

```
1298:     for (i = 0 ; i < MAX_DES_FIFO_LEN; i++) {
1299:         tx_pointers[i] = (unsigned long *) kmalloc(MAX_DES_BUF, GFP_KERNEL);
1300:         item = (struct fifo_ptr_desc *)
1301:             kmalloc(sizeof(struct fifo_ptr_desc), GFP_KERNEL);
1302:         item->data = (unsigned char *) tx_pointers[i];
1303:         item->ptr_num = i;
1304:         item->len = 0;
1305:         item->data_offt = 0;
1306:         add_fifo_pointer(tx_fifo_free, item);
1307:     }
1308:
1309: }
1310:
1311: /* get flowid : Get a flow for the opened device. It uses the total number of
1312:  * opened devices to assign the number
1313:  * it updates the devs_per_flow list of the the corresponding flow
1314:  */
1315: static unsigned int
1316: get_flow_id(int dev_id)
1317: {
1318:     struct list_head *ptr;
1319:     struct devices_per_flow *item;
1320:     int result, i;
1321:
1322:
1323:     ptr = (struct list_head *) &dvs_for_flow[min_devs];
1324:     ptr = ptr->next;
1325:
1326:     item = (struct devices_per_flow *)
1327:         kmalloc(sizeof(struct devices_per_flow), GFP_KERNEL);
1328:     item->dev_id = dev_id;
1329:     item->keys_modified = 1;
1330:     list_add_tail(&item->list, ptr);
1331:     flow_devs_count[min_devs]++;
1332:     result = min_devs;
1333:
1334:     /* find new min_devs value */
1335:     for (i = 1; i < MAX_DES_FLOWS; i++)
1336:         if (flow_devs_count[i] < flow_devs_count[min_devs])
1337:             min_devs = i;
1338:
1339:     return result;                          /* return flow id */
1340: }
1341:
1342:
1343:
1344: /* add item to a pointer fifo. Actually, the fifo doesn't contain simple pointers.
1345:  * Its elements are structs that are composed of a pointer that points to a packet
1346:  * and an id that keeps the file descriptor of the device that uses the pointer
1347:  */
1348:
1349: void
1350: add_fifo_pointer(struct list_head hlist, struct fifo_ptr_desc *new)
1351: {
1352:     struct list_head *ptr;
1353:
1354:     ptr = (struct list_head *) &hlist;
1355:     ptr = ptr->prev;
1356:     list_add_tail(&new->free_list, ptr);
1357:
1358: }
1359:
1360: /* move_fifo_pointer : Get a pointer from the head of src fifo and put it
1361:  * in the tail of dst fifo
1362:  */
1363:
1364: static void
1365: move_fifo_pointer(struct list_head src_list, struct list_head dst_list, int direction)
1366: {
1367:     /* if direction is 0, then move from free to used, else from used to free */
1368:
1369:     struct list_head *ptr1, *ptr2;
1370:     struct fifo_ptr_desc *item;
1371:
1372:     ptr1 = (struct list_head *) &src_list;
1373:     ptr2 = (struct list_head *) &dst_list;
1374:
1375:     ptr1 = ptr1->prev;
1376:     ptr2 = ptr2->next;
1377:     if ( direction == 0) {
1378:         item = list_entry(ptr1, struct fifo_ptr_desc, free_list);
1379:         list_add_tail(&item->used_list, ptr2);
1380:         list_del(&item->free_list);
```

```
1381:     }
1382:
1383:     if (direction == 1) {
1384:         item = list_entry(ptr1, struct fifo_ptr_desc, used_list);
1385:         list_add_tail(&item->free_list, ptr2);
1386:         list_del(&item->used_list);
1387:     }
1388:
1389:     if (direction == 2) {
1390:         item = list_entry(ptr1, struct fifo_ptr_desc, used_list);
1391:         list_add_tail(&item->rxbuf_list, ptr2);
1392:         list_del(&item->used_list);
1393:     }
1394:
1395:     if (direction == 3) {
1396:         item = list_entry(ptr1, struct fifo_ptr_desc, rxbuf_list);
1397:         list_add_tail(&item->free_list, ptr2);
1398:         list_del(&item->rxbuf_list);
1399:     }
1400: }
1401:
1402: /* use this function to empty fifos at module cleanup */
1403: static void
1404: remove_fifo_pointer(struct list_head list)
1405: {
1406:     struct list_head *ptr;
1407:     ptr = (struct list_head *) &list;
1408:     ptr = ptr->prev;
1409:     list_del(ptr);
1410: }
1411:
1412:
1413:
1414: /* request_fifo_ptr: This function requests a pair of fifo pointers, one from
1415:  * the tx fifo and one from the rx fifo. These two pointers are removed from
1416:  * their corresponding "free" fifos and inserterd in the "used" ones.
1417:  * This function is called when a transmission to des begins. The pointers are
1418:  * sent together to the corresponding dma channels and are released together
1419:  * upon reception of the encrypted/decrypted packet.
1420:  */
1421: static int
1422: request_fifo_ptr(int nonblock, struct iandes_priv *priv)
1423: {
1424:     struct list_head     *rx_ptr, *tx_ptr;
1425:     struct fifo_ptr_desc *rx_fifoptr, *tx_fifoptr;
1426:
1427:
1428:     int dev_id;
1429:     unsigned int fid;
1430:
1431:     fid = priv->fid;
1432:     dev_id = priv->dev_id;
1433:
1434:     /* block until one pointer at least is available */
1435:
1436:     rx_ptr = &rx_fifo_free;
1437:     while ((rx_ptr->next == &rx_fifo_free) && (rx_ptr->prev == &rx_fifo_free)){
1438:         if (nonblock)
1439:             return -EAGAIN;
1440:     }
1441:
1442:     /* now get one pointer from rx free fifo and put it in
1443:      * corresponding rx used fifo
1444:      */
1445:     move_fifo_pointer(rx_fifo_free, rx_fifo_used, 0);
1446:     rx_ptr = (struct list_head *) rx_fifo_used.next;
1447:
1448:     /* initialize the rx pointer descriptor */
1449:     rx_fifoptr = list_entry(rx_ptr, struct fifo_ptr_desc, used_list);
1450:     rx_fifoptr->priv = (struct iandes_priv *) priv;
1451:     rx_fifoptr->len = priv->rx_len;
1452:     rx_fifoptr->data = priv->rx_data;
1453:     rx_fifoptr->data_offt = priv->data_offt;
1454:     rx_fifoptr->dec = priv->dec;
1455:     rx_fifoptr->cbc = priv->cbc;
1456:     priv->rx_ptr_num = rx_fifoptr->ptr_num;
1457:
1458:
1459:
1460:     /* Then get the tx pointer. Again, block until one pointer is free for use */
1461:     tx_ptr = &tx_fifo_free;
1462:     move_fifo_pointer(tx_fifo_free, tx_fifo_used, 0);
1463:     tx_ptr = tx_fifo_used.next;
```

```
1464:     tx_fifoptr = list_entry(tx_ptr, struct fifo_ptr_desc, used_list);
1465:     tx_fifoptr->priv = (struct iandes_priv *) priv;
1466:     tx_fifoptr->len = priv->tx_len;
1467:     tx_fifoptr->data = priv->tx_data;
1468:     tx_fifoptr->data_offt = priv->data_offt;
1469:     tx_fifoptr->dec = priv->dec;
1470:     tx_fifoptr->cbc = priv->cbc;
1471:     priv->tx_ptr_num = tx_fifoptr->ptr_num;
1472:     return 0;
1473: }
1474:
1475:
1476: /* set_flow_lock: Check if the keys for this flow are valid for the calling device
1477:  * If not, block until all pointers for this flow are serviced and then get the lock
1478:  * and set new values for the keys.
1479:  */
1480: static int
1481: set_flow_lock(struct iandes_priv *priv)
1482: {
1483:     unsigned int flow_id;
1484:     int dev_id;
1485:     struct iandes_flow_lock *fl;
1486:     int set_keys;
1487:
1488:     dev_id = priv->dev_id;
1489:     flow_id = priv->fid;
1490:
1491:     fl = (struct iandes_flow_lock *) &flow_lock[flow_id];
1492:
1493:     if ( (fl->lock == 0) || ((fl->lock == 1) && (fl->lock_owner = dev_id))) {
1494:         fl->lock = 1;
1495:         fl->lock_owner = dev_id;
1496:         return 0;
1497:     }
1498:
1499:     if ((fl->lock == 1) && (fl->lock_owner != dev_id) && (fl->pending_pointers == 0 )) {
1500:         fl->lock_owner = dev_id;
1501:         return 0;
1502:     }
1503:
1504:
1505:     if ((fl->lock == 1) && (fl->lock_owner != dev_id) && (fl->pending_pointers > 0))
1506:         return -1;
1507:
1508:     return 0;
1509: }
1510:
1511: static void
1512: release_flow_lock(struct iandes_priv *priv)
1513: {
1514:     int dev_id;
1515:     struct iandes_flow_lock *fl;
1516:     int flow_id;
1517:
1518:     dev_id = priv->dev_id;
1519:     flow_id = priv->fid;
1520:
1521:     fl = (struct iandes_flow_lock *) &flow_lock[flow_id];
1522:     if ((!priv->pending) && (fl->lock_owner == priv->dev_id))
1523:         fl->lock = 0;
1524: }
1525:
1526: /* set_flow_keys : Set the new values to the keys for a selected flow */
1527: static void
1528: set_flow_keys(struct iandes_priv *priv, int doit)
1529: {
1530:
1531:     unsigned int fid, modify_keys, tmp;
1532:     struct list_head *ptr, *this_flow_devs;
1533:     struct devices_per_flow *item;
1534:     struct iandes_flow_lock *fl;
1535:
1536:
1537:
1538:     fid = priv->fid;
1539:
1540:     modify_keys = 0;
1541:     fl = (struct iandes_flow_lock *) &flow_lock[fid];
1542:
1543:
1544:     this_flow_devs = (struct list_head *) &dvs_for_flow[fid];
1545:     for (ptr = dvs_for_flow[fid].next; ptr != &dvs_for_flow[fid]; ptr = ptr-> next) {
1546:         item = list_entry(ptr, struct devices_per_flow, list);
```

```
1547:         if (item->dev_id == priv->dev_id) {
1548:             if (item->keys_modified == 1)
1549:                 modify_keys = 1;
1550:             else
1551:                 modify_keys = 0;
1552:         }
1553:     }
1554:
1555:     if ((modify_keys) || (doit == 1)) {
1556:         /* At this point, we can safely change the key values for the selected flow */
1557:         SET_IANDES_REG(IANDES_CNF_A, fid);
1558:         tmp = priv->key1a;
1559:         SET_IANDES_REG(IANDES_CNF_A, tmp);
1560:         tmp = priv->key1b;
1561:         SET_IANDES_REG(IANDES_CNF_A, tmp);
1562:         tmp = priv->key2a;
1563:         SET_IANDES_REG(IANDES_CNF_A, tmp);
1564:         tmp = priv->key2b;
1565:         SET_IANDES_REG(IANDES_CNF_A, tmp);
1566:         tmp = priv->key3a;
1567:         SET_IANDES_REG(IANDES_CNF_A, tmp);
1568:         tmp = priv->key3b;
1569:         SET_IANDES_REG(IANDES_CNF_A, tmp);
1570:
1571:
1572:         this_flow_devs = (struct list_head *) &dvs_for_flow[fid];
1573:         for (ptr = dvs_for_flow[fid].next; ptr != &dvs_for_flow[fid] ; ptr = ptr->next) {
1574:             item = list_entry(ptr, struct devices_per_flow, list);
1575:             if (item->dev_id != priv->dev_id)
1576:                 item->keys_modified = 1;
1577:             else
1578:                 item->keys_modified = 0;    /* this device, that called set_flows_keys, has
1579:                                              * now valid keys
1580:                                              */
1581:         }
1582:
1583:     }
1584:
1585: }
1586:
1587:
1588: /* check dma registers */
1589:
1590: static void
1591: read_iandma_regs(void)
1592: {
1593:     iandma_regs dregs;
1594:
1595:     /* read iandma regs */
1596:     dregs.cwr_rx        = GET_IANDMA_REG(IANDMA_CWR_A(D0R));
1597:     dregs.cwr_tx        = GET_IANDMA_REG(IANDMA_CWR_A(D0T));
1598:     dregs.cbr           = GET_IANDMA_REG(IANDMA_CBR_A);
1599:     dregs.ccr           = GET_IANDMA_REG(IANDMA_CCR_A);
1600:     dregs.imr_rx        = GET_IANDMA_REG(IANDMA_IMR_A(D0R));
1601:     dregs.imr_tx        = GET_IANDMA_REG(IANDMA_IMR_A(D0T));
1602:     dregs.cmr           = GET_IANDMA_REG(IANDMA_CMR_A);
1603:     dregs.cer           = GET_IANDMA_REG(IANDMA_CER_A);
1604:     dregs.csr           = GET_IANDMA_REG(IANDMA_CSR_A);
1605:     dregs.isr_rx        = GET_IANDMA_REG(IANDMA_ISR_A(D0R));
1606:     dregs.isr_tx        = GET_IANDMA_REG(IANDMA_ISR_A(D0T));
1607:     dregs.cnt_rx        = GET_IANDMA_REG(IANDMA_CNT_A(D0R));
1608:     dregs.cnt_tx        = GET_IANDMA_REG(IANDMA_CNT_A(D0T));
1609:
1610:     /* print debug information */
1611:
1612:     printk("IANDMA registers (for eth0)\n");
1613:     printk("cwr_rx          : 0x%08x\n", dregs.cwr_rx);
1614:     printk("cwr_tx          : 0x%08x\n", dregs.cwr_tx);
1615:     printk("cbr             : 0x%08x\n", dregs.cbr);
1616:     printk("ccr             : 0x%08x\n", dregs.ccr);
1617:     printk("imr_rx          : 0x%08x\n", dregs.imr_rx);
1618:     printk("imr_tx          : 0x%08x\n", dregs.imr_tx);
1619:     printk("cmr             : 0x%08x\n", dregs.cmr);
1620:     printk("cer             : 0x%08x\n", dregs.cer);
1621:     printk("csr             : 0x%08x\n", dregs.csr);
1622:     printk("isr_rx          : 0x%08x\n", dregs.isr_rx);
1623:     printk("isr_tx          : 0x%08x\n", dregs.isr_tx);
1624:     printk("cnt_rx          : 0x%08x\n", dregs.cnt_rx);
1625:     printk("cnt_tx          : 0x%08x\n", dregs.cnt_tx);
1626: }
1627:
1628:
1629:
```

```
1630: module_init(iandes_init_module);
1631: module_exit(iandes_cleanup_module);
1632:
1633: MODULE_LICENSE("GPL");
```

Βιβλιογραφία

[1] SPARC International Inc., *The SPARC Architecture Manual Version 8*

[2] Jiri Gaisler, *The LEON-2 Processor User's Manual XST Edition, Version 1.0.19* (Gaisler Research, August 2003)

[3] Αντώνιος Θ. Ταβουλάρης, *Αποδοτική Αρχιτεκτονική Οικιακής Πύλης για την Υποστήριξη Ενοποιημένων Υπηρεσιών,* (Διδακτορική Διατριβή: Αθήνα 2004)

[4] Alessandro Rubini-Jonathan Corbet, Linux Device Drivers, 2nd Edition, (O'Reilly: 2nd edition, June, 2001)

[5] Daniel P. Bovet, Marco Cesati *Understanding the Linux Kernel,* (O'Reilly: 2nd edition, December, 2002)

[6] OpenSSL Project, *OpenSSL Documentation,* (url: http://www.openssl.org)

[7] Tony Bautts, Terry Dawson, Gregor N. Purdy, *Linux Network Administrator's Guide,* (O'Reilly; 3 edition February 8, 2005)

[8] Mark Mitchell, Jeffrey Oldham,and Alex Samuel, *Advanced Linux Programming,* (New Riders Publishing, June 2001)

[9] Andrew S. Tanenbaum, *Computer Networks* (Prentice Hall PTR: 4 edition, August 9, 2002)

[10] Jim Turley, *Embedded systems survey: Operating systems up for grabs,* (url: http://www.embedded.com, April, 2005)

[11] Thomas F. Herbert, The Linux TCP/IP Stack: Networking for Embedded Systems, (Charles River Media: May 2004)

[12] Embedded Linux/Microcontroller Project, *uClinux,* (url://www.uclinux.org)

[13] Free Resources for System on Chip, *LEOX Project,* (url: http://www.leox.org)