ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Ενσωματωμένα Συστήματα και σήματα διακοπής. Η περίπτωση του ARM.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μάριος Μπασούκος

Βόλος, Δεκέμβριος 2006

Αφιερωμένη στους
γονείς μου...

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή του Τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων, του Πανεπιστημίου Θεσσαλίας και κύριο επιβλέποντα της διπλωματικής μου εργασίας κ. Γεώργιο Σταμούλη που μου έδωσε την ευκαιρία να πραγματοποιήσω αυτή την μελέτη. Η υποστήριξή του, η αμέριστη συμπαράστασή του, αλλά και οι διαρκείς και εύστοχες υποδείξεις και παρατηρήσεις του, με βοήθησαν στην έγκαιρη ολοκλήρωση αυτής της μελέτης.

Επίσης, θα ήθελα να ευχαριστήσω τον έτερο επιβλέποντα καθηγητή του ιδίου Τμήματος, κ. Νέστορα Ευμορφόπουλο. Οι γνώσεις του και η βοήθειά του αποτέλεσαν τα θεμέλια της προσπάθειας για μια όσο το δυνατόν καλύτερη και πιο εμπεριστατωμένη παρουσίαση του θέματος. Δεν θα μπορούσα να ξεχάσω τον προπτυχιακό φοιτητή του Τμήματος και προσωπικό μου φίλο, Αθανάσιο Γιαννέτσο για την αμέριστη βοήθεια που μου προσέφερε.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου και τον αδερφό μου Γιάννη για την υποστήριξη που μου έδωσαν και που μου δίνουν σε όλα τα βήματα της ζωής μου. Επίσης ευχαριστώ όλους τους φίλους μου οι οποίοι μου συμπαραστάθηκαν αμέριστα καθ' όλη τη διάρκεια εκπόνησης της διπλωματικής αυτής εργασίας και καθ' όλη την διάρκεια της φοιτητικής μου ζωής στο Βόλο και στο Πανεπιστήμιο Θεσσαλίας.

Περιεχόμενα

Εισαγωγή

Η εφαρμογή που αναπτύχθηκε, στα πλαίσια αυτής της διπλωματικής εργασίας, υλοποιήθηκε πάνω σε ένα εκπαιδευτικό ενσωματωμένο σύστημα, της εταιρίας ADI, X-board.

Το ενσωματωμένο αυτό σύστημα, περιελάμβανε το επεξεργαστή της εταιρίας Intel, XScale, ο οποίος είναι ειδικά σχεδιασμένος για ενσωματωμένα συστήματα, και του οποίου η σχεδίαση έχει βασιστεί πάνω στον επεξεργαστή ARM, αρχιτεκτονικής RISC. Ο συνδυασμός αυτός, επιλέχθηκε διότι ο επεξεργαστής XScale, είναι ένας από τους πλέον σύγχρονους και εξελιγμένους επεξεργαστές για ενσωματωμένα συστήματα αυτήν την στιγμή και χρησιμοποιείται σε πληθώρα τέτοιων συσκευών. Ο δε επεξεργαστής ARM, στου οποίου την σχεδίαση έχει βασιστεί ο επεξεργαστής XScale, χρησιμοποιείται ευρύτατα, σήμερα σε εφαρμογές όπως το ipod nano της Apple, το PSP της Sony, το Game Boy Advance.

Το γεγονός αυτό, και η επιλογή αυτή των συστημάτων για την ανάπτυξη της εφαρμογής, έδωσε την δυνατότητα στο να ασχοληθούμε με περισσότερο ζήλο πάνω σε κάτι το οποίο πραγματικά δουλεύει και κυριαρχεί την αγορά των ενσωματωμένων συστημάτων σήμερα.

Κεφάλαιο 1

Περίληψη των βασικών αρχών των ενσωματωμένων συστημάτων

Αλήθεια... τι ακριβώς είναι ένα ενσωματωμένο σύστημα; Ενσωματωμένο σύστημα, είναι ένα υπολογιστικό σύστημα το οποίο έχει ως κύρια λειτουργική και υπολογιστική μονάδα έναν μικρό-επεξεργαστή. Ο μικροεπεξεργαστής αυτός χρησιμοποιείται στο να ελέγχει μιας πληθώρα εφαρμογών και λειτουργιών του συστήματος αυτού.

Δεν είναι απλά ένας υπολογιστής με την ευρύτερη έννοια του προσωπικού υπολογιστή που όλοι γνωρίζουμε! Οι προσωπικοί υπολογιστές, για παράδειγμα, είναι σχεδιασμένοι ώστε να είναι «γενικού τύπου» υπολογιστές (general-purpose computers). Με την έννοια αυτή εννοούμε, ότι οι προσωπικοί υπολογιστές, και γενικότερα οι υπολογιστές, «γενικού τύπου» μπορούν να χρησιμοποιηθούν σε μια πληθώρα διαφορετικών εφαρμογών και λειτουργιών. Το ενσωματωμένα σύστημα, εν αντιθέσει, είναι σχεδιασμένα έτσι ώστε να υλοποιούν μια μόνο λειτουργία, ή ένα

πολύ συγκεκριμένο και ορισμένο σύνολο από λειτουργίες. Το γεγονός αυτό, από μόνο του, αποτελεί την διαχωριστική νησίδα μεταξύ των «κοινών», «γενικού σκοπού» υπολογιστών και των ενσωματωμένων συστημάτων.

Επίσης, τα ενσωματωμένα συστήματα, είναι πολύ παραπάνω από ένα απλό προσωπικό υπολογιστή. Για παράδειγμα, πολύ συχνά τα ενσωματωμένα συστήματα αλληλεπιδρούν με το φυσικό περιβάλλον, που τους περιβάλει, είτε αυτό το περιβάλλον ονομάζεται, δωμάτιο σπιτιού, είτε αυτοκίνητο εν κινήσει, είτε αεροσκάφος εν πτήσει, είτε ακόμα και ένας πυρηνικός αντιδραστήρας σε πλήρη λειτουργία.

Από την μεριά του χρήστη τώρα. Πώς ο χρήστης αντιλαμβάνεται την ύπαρξη ενός ενσωματωμένου συστήματος; Ο τελικός χρήστης, ο χρηστής που αλληλεπιδρά με τον ενσωματωμένο σύστημα, «βλέπει» και «θεωρεί» ότι έχει απέναντι του, ότι αλληλεπιδρά και ότι συναλλάσσεται με ένα «έξυπνο, ευφυές, γενικού τύπου» σύστημα, εν αντιθέσει με το γεγονός ότι το υπολογιστικό τμήμα του συστήματος, δεν αποτελείται από τέτοιου «γενικού» τύπου επεξεργαστή.

Η παραπάνω εντύπωση δημιουργείται πολύ συχνά στον τελικό χρήστη του ενσωματωμένου συστήματος, καθότι το ενσωματωμένο συστημα, τα δεδομένα που δέχεται σαν είσοδο αυτό, δεν μπορούν να τροποποιηθούν ή να αλλάξουν ή να ενημερωθούν απευθείας από αυτόν.

Τελικά και με απτά παραδείγματα, τι είναι ένα ενσωματωμένο σύστημα; Αφού προχωρήσαμε στην επεξήγηση της έννοιας «ενσωματωμένο σύστημα» και στην μερική επεξήγηση της διαφοροποίησης τέτοιων υπολογιστικών συστημάτων από τα «γενικού τύπου», μπορούμε να προχωρήσουμε σε απτά παραδείγματα τα οποία αποτελούν τέτοια συστήματα. Παραδείγματα λοιπόν τέτοιων συστημάτων είναι: τα κινητά τηλέφωνα, οι αποκωδικοποιητές ψηφιακής δορυφορικής και επίγειάς τηλεόρασης, οι συσκευές τηλεόρασης, οι παιχνιδομηχανές, οι ψυγειοκαταψύκτες, τα αυτοκίνητα, τα αεροσκάφη, οι ανελκυστήρες, τα συστήματα συναγερμού, τα συστήματα ελέγχου εργοστασίων, ιατρικές εφαρμογές όπως ο βηματοδότης, οι διάφορες διαστημοσυσκευές, και αλλά...

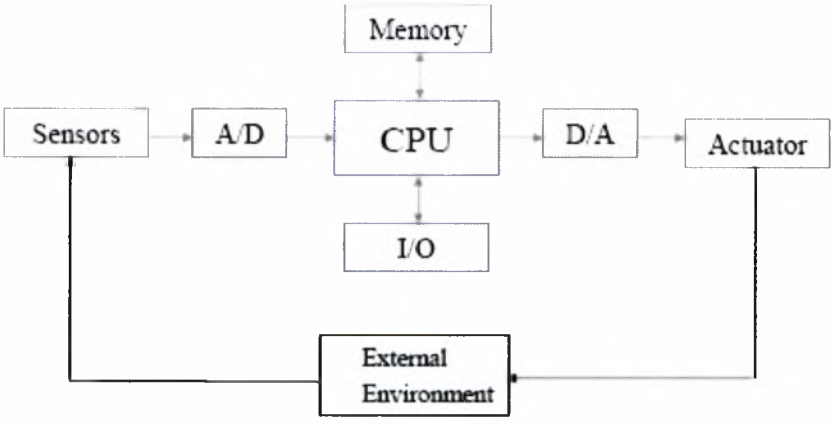
Η αγορά των ενσωματωμένων συστημάτων είναι απλά τεράστια. Το έτος 2000 για παράδειγμα, η αγορά των επεξεργαστών ήταν της τάξης των 8 δισεκατομμυρίων δολλαρίων. Την ίδια χρονιά, η αγορά των επεξεργαστών για

ενσωματωμένα συστήματα αποτελούσε τα 7.8 από τα 8 δισεκατομμύρια δολλάρια της συνολικής αγοράς των επεξεργαστών παγκοσμίως.

Διαφοροποίηση ενσωματωμένων συστημάτων με τα υπόλοιπα υπολογιστικά συστήματα

Τι διαφοροποιεί τα ενσωματωμένα συστήματα, σε σχέση με τα υπόλοιπα συστήματα; Η διαφοροποίηση, πέρα από τη διαφορά, μεταξύ των «γενικού σκοπού» υπολογιστικών συστημάτων, όπως αυτή αναφέρθηκε παραπάνω, είναι το γεγονός, ότι τα ενσωματωμένα συστήματα επιτελούν μια συγκεκριμένη λειτουργικότητα, δηλαδή, μια και μόνο λειτουργικότητα ή μερικές λειτουργικότητες με επαναληπτικότητα. Στην βάση αυτή των παραπάνω, αξίζει να αναφερθεί, ότι σε ένα ενσωματωμένο σύστημα, το υλικό (hardware) που ενσωματώνεται σε αυτό, είναι αυτό και μόνο αυτό ακριβώς και τίποτα παραπάνω από αυτό, που παρέχει την συγκεκριμένη λειτουργικότητα που απαιτεί η εφαρμογή. Το τελευταίο, συνεπάγεται, πώς το πρόγραμμα της εφαρμογής το οποίο θα τρέχει «πάνω» σε ένα τέτοιο σύστημα, σε ένα ενσωματωμένο σύστημα, θα πρέπει να είναι προσεκτικά σχεδιασμένο, υλοποιημένο και προσαρμοσμένο για το σύστημα αυτό και για μόνον αυτό.

Τα ενσωματωμένα συστήματα πολλές φορές αντιδρούν και αλληλεπιδρούν με το περιβάλλον τους και την λειτουργικότητα αυτή πρέπει να την διατηρούν στο ακέραιο. Για παράδειγμα, ένα ενσωματωμένο σύστημα το οποίο βρίσκεται και αλληλεπιδρά με το περιβάλλον του, οφείλει και πρέπει να αντιδρά και να αποκρίνεται στις αλλαγές και στις επιδράσεις του περιβάλλοντος και να υπολογίζει κάθε φορά, συγκεκριμένα αποτελέσματα σε εναπαληπτική βάση. Τέτοια ενσωματωμένα συστήματα, τα οποία αλληλεπιδρούν με το περιβάλλον τους, ενσωματώνουν πολλές φορές διαφόρου τύπου αισθητήρες και ανιχνευτές στην αρχιτεκτονική του συστήματος.



Επίσης, συχνά αναμένουμε, τα ενσωματωμένα συστήματα, να έχουν επιπλέον ιδιότητες, πέρα των παραπάνω βασικών, όπως η λειτουργία τους σε πραγματικό χρόνο (real time).

Περιορισμοί των ενσωματωμένων συστημάτων

Πολύ συχνά, τα ενσωματωμένα συστήματα είναι σχεδιασμένα με βάση πολύ αυστηρές και συγκεκριμένες προδιαγραφές οι οποίες, δύναται, η μια με την άλλη να αλληλεπιδρούν ανταγωνιστικά μεταξύ τους. Παραδείγματα τέτοιων προδιαγραφών:

- Μικρή ποσότητα μνήμης

Η ύπαρξη μνήμης και η ποσότητα αυτής, δύναται πολλές φορές, να είναι ένα σημαντικό και αναπόσπαστο κομμάτι του κόστους τους συστήματος.

- Μέγεθος, βάρος.

Πολλές φορές, τα ενσωματωμένα συστήματα επιθυμούμε να είναι μικρά σε βάρος, και σε όγκο. Παράδειγμα τέτοιων εφαρμογών με τέτοιου είδους προδιαγραφές είναι τα κινητά τηλεφωνά και οι φορητές ηλεκτρονικές συσκευές.

- Μικρή κατανάλωση ισχύος.

Οι περιορισμένες δυνατότητες ψύξης των ενσωματωμένων συστημάτων, επιβάλλούν και την όσο το δυνατόν μικρότερη δυνατή κατανάλωση ενέργειας από αυτά, ακόμα και αν λειτουργούν με σταθερή παροχή ηλεκτρικού ρεύματος.

- Υψηλή επίδοση

Παρά πολύ συχνά τα ενσωματωμένα συστήματα πρέπει να υπολογίσουν και να επιστρέψουν αποτελέσματα σε ιδιαίτερα μικρό χρονικό διάστημα.

- Μικρό κόστος

Τα ενσωματωμένα συστήματα τα οποία έχουν ως αγορά στόχο το ευρύ καταναλωτικό κοινό πρέπει να έχουν αυτή την προϋπόθεση, καθότι, σε ενσωματωμένα συστήματα, με μεγάλο αριθμό επιμέρους συστημάτων, ακόμα και 0.05euro αύξηση στην τιμή του κάθε επιμέρους τμήματος, μπορεί να επιφέρει μεγάλη αύξηση στην τελική τιμή πώλησης του προϊόντος στην αγορά.

- Επικίνδυνο περιβάλλον

Το περιβάλλον μέσα στο οποίο θα λειτουργήσει το ενσωματωμένο σύστημα δύναται να είναι επικίνδυνο, να επηρεάζει και να δυσκολεύει την ίδια την λειτουργικότητα του συστήματος, όπως για παράδειγμα, το διάστημα και οι συνθήκες που επικρατούν εκεί, ή περιβάλλοντα με υπερβολική ζέστη ή με μεγάλες δονήσεις.

- Λειτουργικότητα κρίσιμη για την ασφάλεια.

Τα ενσωματωμένα συστήματα δύναται να λειτουργούν σε περιβάλλοντα ή να υλοποιούν εφαρμογές κρίσιμες, όπως το μπλοκάρισμα των τροχών στ ABS στο αυτοκίνητο, ή ο έγκαιρος εντοπισμός και ενημέρωση για μια βλάβη σε ένα αεροσκάφος εν πτύσει. Γενικά, ένα ενσωματωμένο σύστημα, το οποίο καλείται να υλοποιήσει ειδικά τέτοιες προυποθέσεις ασφάλειας, οφείλει να λειτουργεί σωστά και ταυτόχρονα οφείλει να μην λειτουργεί λανθασμένα.

Παράδειγμα αρχιτεκτονικής ενσωματωμένου συστήματος είναι το σύστημα Αντιμπλοκαρίσματος Τροχών στα αυτοκίνητα, το γνωστό και ως ABS(Antilock

Braking System). Το όλο σύστημα, στηρίζεται στο γεγονός, ότι καθώς ενεργοποιούνται τα φρένα του αυτοκινήτου, οι τροχοί του, παύουν να περιστρέφονται, με αποτέλεσμα το αυτοκίνητο, να αρχίζει να ολισθαίνει και να είναι δύσκολο, έως αδύνατον να ελέγχει η πορεία του. Το ABS, επιτρέπει την παροδική άφεση του φρένου στους τροχούς, ώστε αυτοί να κυλούν και να μην χάνεται ο έλεγχος του αυτοκινήτου. Η παροδική άφεση του φρένου λαμβάνει χώρα, μόνο εάν η ταχύτητα περιστροφής του τροχού, κατά την διαρκεί του φρεναρίσματος, είναι πολύ μικρή.

Από το παραπάνω παράδειγμα, καθίσταται σαφές πώς ένα σύστημα σαν και αυτό, οφείλει να ενσωματώνει προϋποθέσεις αξιοπιστίας, ασφαλείας, καθώς και χρόνου.

Λίγο παραπάνω στα προηγούμενα, αναφέρθηκε, η έννοια ότι τα ενσωματωμένα συστήματα δύναται, να έχουν ιδιότητες λειτουργίας «πραγματικού χρόνου». Με την έννοια αυτή αναφερόμαστε στο γεγονός ότι στα ενσωματωμένα συστήματα, η σωστή, χρονικά, λειτουργικότητα τους, είναι μέρος της συνολικής ορθής και σωστής λειτουργίας του όλού συστήματος. Διακρίνουμε δύο περιπτώσεις:

- Hard Real-Time

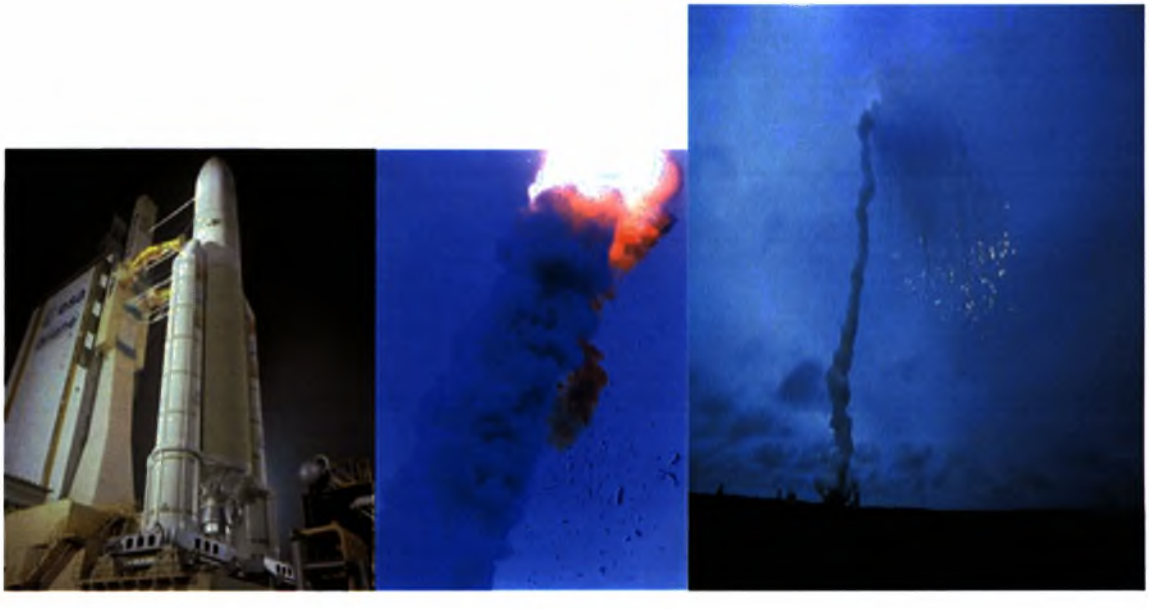
Τα συστήματα αυτά είναι αυστηρά, με την έννοια, ότι λειτουργούν, παράγουν και στέλνουν αποτελέσματα, μέσα σε ένα απολύτως αυστηρά ορισμένο χρονικό πλαίσιο. Σε κάθε διαφορετική περίπτωση που τα αποτελέσματα παραχθούν σε χρονικό διάστημα, πέρα του ορισμένου, τότε είναι απλά άχρηστα.

- Soft Real-Time

Τα συστήματα αυτά διαφέρουν από τα προηγούμενα υπό την έννοια, όχι ότι δεν υπακούν σε χρονικούς περιορισμούς και πλαίσια, αλλά το ότι έχουν την δυνατότητα να παράξουν το αποτέλεσμα, να επιτελέσουν την λειτουργικότητα τους, και πέρα από το όριο του χρονικού περιορισμού που έχει οριστεί. Με άλλα λόγια, εαν παραχθει απότελεσμα εκτος του χρονικού πλαισιου, το γεγονός αυτό δεν αποτελεί κάτι το κακό ή το καταστροφικό για το όλο συστημα.

Ας πάρουμε ένα παράδειγμα για να διαφανεί η διαφοροποιηση των δύο παραπάνω κατηγοριών. Ένας ψηφιακός αποκωδικοποιητής, ένα DVD Player για παράδειγμα, ο χρόνος που έχει στην διαθεσή του, για την αποκωδικοποίηση ενός frame, είναι αυστηρα περιορισμένος και καθορισμένος στα 30 χιλιοστά του δευτερολέπτου, προτού το επόμενο frame έρθει στον αποκωδικοποιητή, για αποκωδικοποιηση.

Κλεινοντας το κεφαλαιο αυτό και την επεξήγηση της έννοιας «ενσωματωμένο σύστημα πραγματικού χρόνου», απαραίτητο, είναι να προχωρήσουμε και σε μια ακομη επεξηγηση. Το γεγονός ότι ένα ενσωματωμενο σύστημα έιναι σύστημα πραγματικού χρόνου, δεν σημαίνει ότι είναι και γρήγορο! Το να είναι ένα σύστημα «γρήγορό» σημάινει, ότι η λειτουργικότητα του, επιτελείται με «υψηλή ταχύτητα», «γρήγορα» δηλαδή. Αντίθετα, το να είναι ένα σύστημα «πραγματικού χρόνου» σημαίνει ότι υλοποιεί κάποιους, αυστηρούς ή μη, χρονικούς περιορισμούς στην επεξεργασία των δεδομένων και στην επιτέλεση της λειτουργικότητας που υλοποιεί. Ένα «πραγματικού χρόνου» συστημα, μπορει κάλλιστα να είναι αργο! Η αλήθεια είναι ότι όμως ότι είναι επιδιωξή μας, να σχεδιάζονται «συστήματα πραγματικού χρόνου» τα οποία να είναι γρήγορα και να λειτουργούν σε υψηλές ταχύτητες επεξεργασίας δεδομένων και παραγωγής αποτελεσμάτων.



Κάτι το οπόιο θα πρεπει να τονιστει με μεγάλη μφάση στο σημειο αυτό ειναι το γεγονός ότι οι εφαρμογές που υλοποιούνται πάνω σε ενσωματωμένα συστήματα, θα πρέπει να είναι πραγματικά «εὐρωστες» (robust). Για να γίνει πιο κατανόητη η αυτή καθαυτή έννοια robust, θα παρουσιαστει ένα πραγματικό πρόβλημα που έλαβε χώρα πρίν από περίπου μία δεκαετία, ίσως και λιγότερο. Ο πυραυλοφορέας της Ευρωπαϊκής Διαστημικής Υπηρεσίας (ESA), του οποίου η αναπτυξη κόστισε

περί τα 7 δισεκατομμυρια δολάρια, μεταφέροντας φορτίο, της τάξης των 500 εκατομμυρίων δολλαρίων, έξεργη κατά την πρώτη φάση της εκτόξευσης του. Ο λόγος που προκάλεσε την καταστροφή αυτή, ήταν ένα προγραμματιστικό λάθος, στο λογισμικό του αδρανειακού συστήματος του πυραύλου. Πιο συγκεκριμένα, ένας 64-bit αριθμός κινητής υποδιαστολής, μετατρεπόταν σε προσημασμένο 16-bit ακέραιο αριθμό. Το γεγονός αυτό καθώς και το γεγονός ότι δεν γινόταν σωστή διαχείριση της εξαίρεσης αυτής (improperly handled exception), που προκαλούνταν από υπερχείλιση της μιας μεταβλητής, οδήγησε στην καταστροφή του πυράυλου.

Κεφάλαιο 2

Γνωριμία με τον επεξεργαστή ARM

Μέτα από την απαραίτητη αυτή αρχική επεξήγηση των θεμελιωδων αρχών και ορισμών, καθώς και εφαρμογών, πάνω στα οποία αναπτύχθηκε η διπλωματική εργασία αυτή, ακολουθεί μια ανάλυση της αρχιτεκτονικής του επεργαστή ARM, στην οποία πάνω είναι βασισμένη η αρχιτεκτονική του επεξεργαστή XScale της Intel, στον οποίο πάνω έγινε η ανάπτυξη της embedded εφαρμογής (εφαρμογής ειδικά σχεδιασμένης και υλοποιημένης για λειτουργία στο συγκεκριμένο ενσωματωμένο σύστημα).

Ο επεξεργαστής ARM, αποτελεί μια υλοποίηση της εταιρείας Advanced RISC Machines, με έδρα την Μεγάλη Βρετανία. Ο XScale αποτελεί την έκδοση v5TE της αρχιτεκτονικής ARM. Είναι ένας επεξεργαστής 32-bit, και είναι υλοποιημένος στο να μπορεί να υποστηρίζει 2 σύνολα εντολών (instruction sets). Ένα σύνολο εντολών 32-bit που ονομάζεται ARM και ένα άλλο σύνολο εντολών 16-bit που ονομάζεται Thumb. Η χρησιμότητα του σετ εντολών Thumb είναι σημαντικότατη καθώς το σετ εντολών αυτό, μας δίνει την δυνατότητα, να εξοικονομούμε μνήμη από τους καταχωρητές του επεξεργαστή, καθώς για μικρού μεγέθους και τύπου δεδομένα, όπως short integers και char, αντί να κρατάμε για τόσο μικρό όγκο δεδομένων μνήμη 32-bit, κρατάμε μνήμη 16-bit, και έτσι εξοικονομείται μνήμη.

Βασικοί τρόποι λειτουργίας του ARM

Στην συνέχεια, γίνεται παρουσιαση επτά βασικών τύπων λειτουργίας του επεξεργαστή ARM (basic operating modes).

Ο επεξεργαστής ARM, διαθέτει επτά βασικούς τρόπους – τύπους λειτουργίας (operating modes):

- User mode: Είναι το βασικό mode λειτουργίας του επεξεργαστή κατά την διάρκεια της οποίας έχουμε την εκτέλεση του προγράμματος του χρήστη, από τον επεξεργαστή. Σε αυτό το mode, δεν διαθέτει περισσότερα δικαιώματα ο χρήστης, πέρα από τα βασικά δικαιώματα του, όπως πρόσβαση σε συγκεκριμένους καταχωρητές και θέσεις μνήμης (unprivileged or user mode)

- FIQ mode: Χρησιμοποιείται για τον χειρισμό των διακοπών και πιο συγκεκριμένα, για τον χειρισμό των γρήγορών διακοπών, υψηλης προτεραιότητας (fast interrupt handling). Σε αυτό το mode, έχουμε πλήρη

έλεγχο της διαδικασίας εκτέλεσης εντολών από τον επεξεργαστή, ή αλλίως δικαιώματα «υπερχρήστη» (privileged or FIQ mode)

- IRQ mode: Χρησιμοποιείται για τον χειρισμό των χαμηλής προτεραιότητας διακοπών (low priority interrupt handling). Σε αυτό το mode έχουμε πλήρη έλεγχο της διαδικασίας εκτέλεσης εντολών από τον επεξεργαστή ή αλλίως δικαιώματα «υπερχρήστη» (privileged or IRQ mode), είναι όμως χαμηλότερο σε προτεραιότητα αυ' το το mode, από το mode του FIQ.

- Supervisor mode: Σε αυτό το mode εισέρχεται ο επεξεργαστής, όταν γίνεται reset και όταν εκτελείται διακοπή προερχόμενη από πρόγραμμα το οποίο το έχει γράψει και το εκτελεί ο χρήστης (Software Interrupt instruction execution). Σε αυτό το mode έχουμε πλήρη έλεγχο της διαδικασίας εκτέλεσης εντολών από τον επεξεργαστή ή αλλίως δικαιώματα «υπερχρήστη» (privileged or Supervisor mode), είναι όμως χαμηλότερο σε προτεραιότητα αυτό το mode, από το mode του FIQ.

- Abort mode: Το mode αυτό χρησιμοποιείται για να γίνεται χειρισμός των παραβιάσεων πρόσβασης στην μνήμη (Memory access violation handling). Σε αυτό το mode, έχουμε πλήρη έλεγχο της διαδικασίας εκτέλεσης εντολών από τον επεξεργαστή ή αλλίως δικαιώματα «υπερχρήστη» (privileged or Abort mode)

- Undefined mode: Το mode αυτό χρησιμοποιείται για να γίνεται χειρισμός των μη ορισμένων εντολών, των λανθασμένων εντολών από το πρόγραμμα του χρήστη προς τον επεξεργαστή (Undefined Instructions handling). Σε αυτό το mode, έχουμε πλήρη έλεγχο της διαδικασίας εκτέλεσης εντολών από τον επεξεργαστή ή αλλίως δικαιώματα «υπερχρήστη» (privileged or Undefined mode)

- System mode: Το mode αυτό λειτουργίας του επεξεργαστή, είναι ένα privileged mode, δηλαδή μπαίνοντας σε αυτό το mode, έχουμε πρόσβαση σε

όλους τους πόρους, μνήμη, καταχωρητές και διεργασίες, χειριστες διακοπών (interrupt handlers), του επεξεργαστή. Το ουσιαστικό στο mode αυτό όμως είναι το γεγονός πώς έχουμε στην διάθεσή μας τους ίδιους ακριβώς καταχωρητές, τους οποίους έχει και το User mode. Δεν έχουμε δηλαδή, ξεχωριστούς καταχωρητές για το mode αυτό, όπως ακριβώς συμβαίνει στα υπόλοιπα, User, FIQ, IRQ mode, όπου σε κάθε mode έχουμε πρόσβαση σε διαφορετικό υποσύνολο του συνόλου των καταχωρητών του επεξεργαστή. (Περισσότερα για αυτό στην συνέχεια)

Μετά την παρουσίαση των επτά διαφορετικών τρόπων – mode λειτουργίας του επεξεργαστή ARM, ακολουθεί η παρουσίαση του συνόλου των καταχωρητών του επεξεργαστή.
Ο ARM, διαθέτει 37 καταχωρητές, μεγέθους 32-bit. Από αυτούς τους 37 καταχωρητές, οι 30 χρησιμοποιούνται ως καταχωρητές γενικού σκοπού (general purpose registers). Από τους υπολοίπους 7 καταχωρητές που απομένουν, ο 1 χρησιμοποιείται αποκλειστικά και μόνο από τον Μετρητή Προγράμματος του επεξεργαστή, τον Program Counter δηλαδή ή αλλιώς PC. Ένας ακόμα καταχωρητής χρησιμοποιείται αποκλειστικά για την αποθήκευση της τρέχουσας κατάστασης εκτέλεσης του προγράμματος (Current Program Status Register - cpsr). Ο τελευταίος αυτός καταχωρητής, αποθηκεύει και την τελευταία εντολή που έχει εκτελεστεί από τον επεξεργαστή. Τέλος, υπάρχουν άλλοι 5 καταχωρητές, οι οποίοι χρησιμοποιούνται για την αποθήκευση των δεδομένων που βρίσκονται στον καταχωρητή cpsr, ακριβώς πριν λάβει χώρα μια εξαίρεση (exception) στο πρόγραμμα που τρέχει εκείνη την ώρα στον επεξεργαστή (Saved Program Status Register - spsr).

Όπως είχε αναφερθεί και παραπάνω, στην ανάλυση των διαφορετικών τύπων λειτουργίας του επεξεργαστή ARM (operating modes), σε κάθε διαφορετικό mode λειτουργίας του επεξεργαστή, αντιστοιχεί και διαφορετικό υποσύνολο του των καταχωρητών. Πιο συγκεκριμένα, κάθε διαφορετικό mode λειτουργίας, «βλέπει», αντιλαμβάνεται και διαφορετικό αριθμό καταχωρητών, από τους 37 συνολικά, που

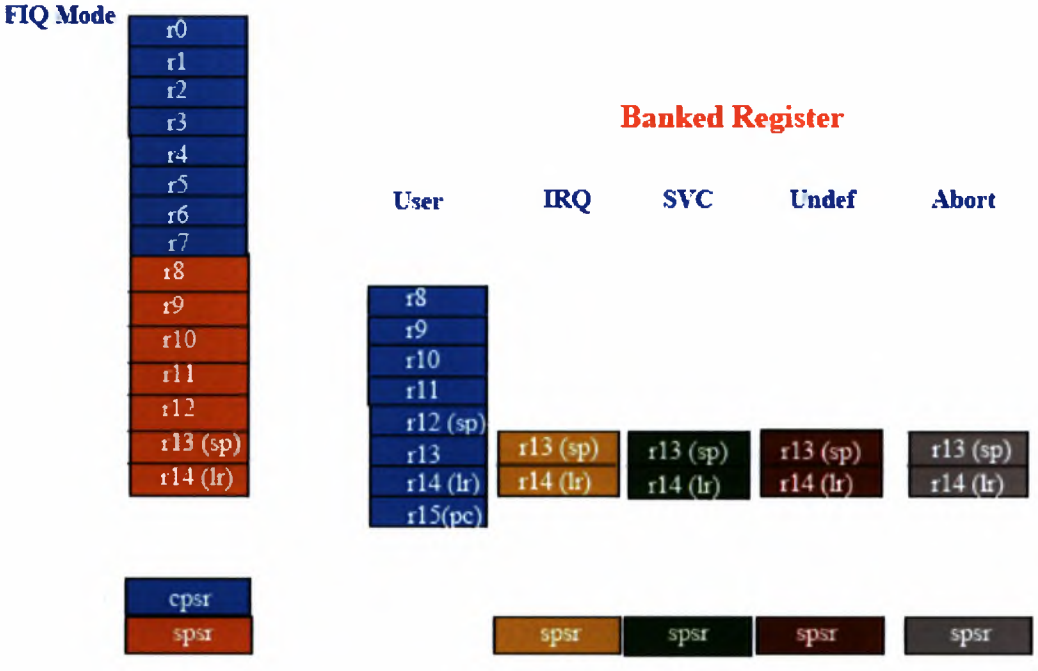
βρίσκονται στον επεξεργαστή. Με τον τρόπο αυτό, το κάθε mode λειτουργίας, έχει στην διάθεσή του, διαφορετικό αριθμό καταχωρητών. Το υπόλοιπο σύνολο των καταχωρητών που δεν είναι «ορατοί» από το κάθε mode, αποκαλείται με τον αγγλικό όρο "banked registers". Το γεγονός αυτό σημαίνει, ότι από το σύνολο τον καταχωρητών, κάποιοι δεν είναι προσβάσιμοι από το κάθε mode λειτουργίας. Οι καταχωρητές αυτοί που δεν είναι προσβάσιμοι, ονομάζονται «κρυφοί καταχωρητές» ή αλλιώς, "banked registers. Κάθε χρονική στιγμή της λειτουργίας του επεξεργαστή, το τρέχον mode λειτουργίας του, καθορίζει και τον αριθμό και το ποιοι συγκεκριμένα, θα είναι οι προσβάσιμοι, σε αυτό το mode λειτουργίας, καταχωρητές. Η ονοματολογία και των περισσότερων καταχωρητών είναι της μορφής rxx, όπου το r αντιστοιχεί στην λέξη register, καταχωρητής δηλαδή, ενω τα επομενα δύο ψηφία, κυμαίνονται από το 0 μέχρι και το 16. Επίσης, υπάρχουν καταχωρητές, οι οποίοι ενώ ακολουθούν την ονοματολογία, έτσι όπως περιγράφηκε παραπάνω, περιγράφονται και με τα ακρωνύμια τους, όπως οι καταχωρητές PC, cpsr και spsr. Σε κάθε περίπτωση, και σε όλη την διάρκεια της παρούσας εργασίας, με τον όρο sp register (ή καταχωρητής sp), θα αναφερόμαστε στον καταχωρητή r13, που είναι ο καταχωρητής που αναφέρεται στο δείκτη της stack. Με τον όρο lr register (ή καταχωρητής lr), θα αναφερόμαστε στον καταχωρητή r14, που είναι ο link καταχωρητής, που δείχνει στην εντολή, της οποίας η εκτέλεση θα ακολουθήσει μετά το πέρας της διαχείρισης μιας εξαίρεσης (exception handling). Με τον όρο pc θα αναφερόμαστε στον καταχωρητή r15, ο οποίος είναι ο καταχωρητής που «δείχνει», που αναφέρεται στην επόμενη προς εκτέλεση εντολή. Οι καταχωρητές cpsr και spsr, δεν ακολουθούν την ονοματοδοσία, έτσι όπως περιγράφηκε παραπάνω, αλλά αναφερόμαστε σε αυτούς με τα ονόματα τους και μόνο. Όπως προαναφέρθηκε, σε κάθε διαφορετικό mode, μπορούν να προσπελάσουν ένα συγκεκριμένο σετ καταχωρητών. Πιο συγκεκριμένα κάθε διαφορετικό mode λειτουργίας, έχει πρόσβαση σε ένα σετ καταχωρητών από τον καταχωρητή r0 έως και τον καταχωρητή r12. Επίσης κάθε διαφορετικό mode λειτουργίας έχει ένα ξεχωριστό stack pointer register καθώς και ένα ξεχωριστό link register. Όμως όλα τα mode λειτουργίας, έχουν πρόσβαση στον ίδιο καταχωρητή pc, και στον ίδιο cpsr. Επιπρόσθετα, όλα τα modes λειτουργίας,

που διαθέτουν δικαιώματα «υπερχρήστη» (privileged modes), πλην του System mode, έχουν πρόσβαση σε έναν ξεχωριστό καταχωρητή, τον spsr.

Για να γίνει ακόμα πιο κατανοητή η λειτουργικότητα των "banked registers" κατά την διάρκεια της εκτέλεσης ενός προγράμματος θα παραθέσουμε το ακόλουθο παράδειγμα, πριν όμως, θα πρέπει να προχωρήσουμε και σε μερικές ακόμα διευκρινιστικές παρατηρήσεις, σχετικα με την έννοια και τον τρόπο λειτουργίας των banked registers. Όπως αναφέραμε και παραπάνω, τα δεδομένα που αντιστοιχούν σε έναν καταχωρητή, δεν αντιστοιχούν μόνο με βάση τον αριθμό του καταχωρητή, αλλα και με βάση το mode λειτουργίας στο οπόιο βρίσκεται εκείνη την στιγμή ο επεξεργαστής. Επιπροσθετα, οι τιμές των μεταβλητών που βρίσκονται στους «κρυφούς», ή αλλίως στους "banked registers", δεν αλλάζουν και παραμένουν σταθερές όσες μεταλλαγές στο mode λειτουργίας του επεξεργαστή και αν έχουμε. Μέτα από αυτές τις απαραίτητες επιπρόσθετες διευκρινιστικες πληροφορίες, μπορούμε να δώσουμε ένα παράδειγμα, ώστε να γίνει πιο κατανοητό και «στην πράξη», η λειτουργία των bakned registers. Θεωρούμε λοιπόν, πως έχουμε τον επεξεργαστή μας, να λειτουργεί στο user mode.

Στο mode αυτό λοιπόν, ο επεξεργαστής απόθηκεύει την τιμή μηδέν (0) στον καταχωρητή r0 και την τιμή οκτώ (8) στον καταχωρητή r8. Μέτα από αυτή την εκτέλεση, ο επξεργαστής αλλάζει mode λειτουργίας και από το user mode, μεταπηδάει και συνεχίζει να λειτουργέι σε FIQ mode.

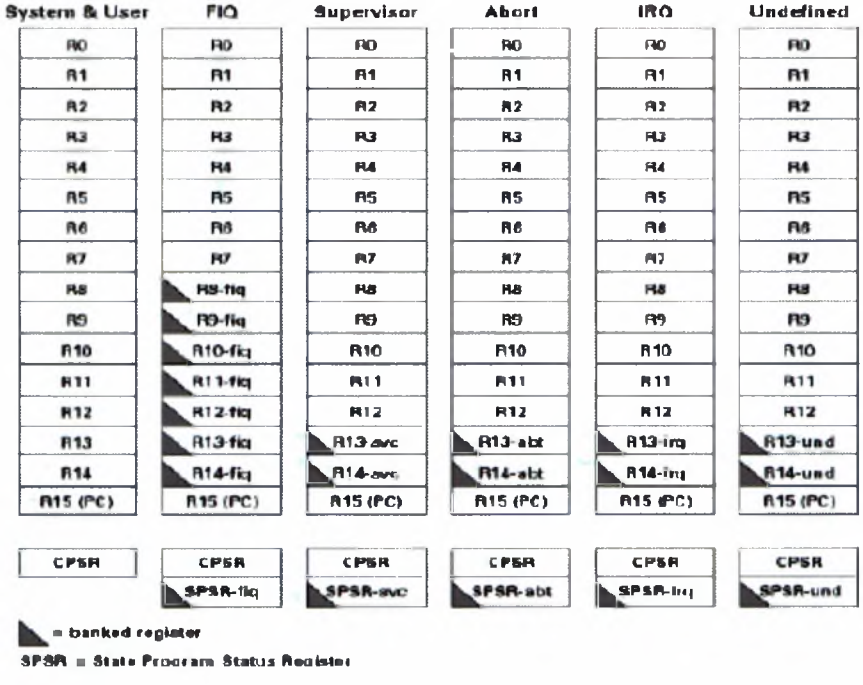
"Ορατοί" - Προσβάσιμοι Καταχωρητές



Στο FIQ mode λοιπόν η τιμή του καταχωρητή r0, είναι η ίδια με την τιμή που είχε και ο καταχωρητής r0 στο user mode, καθώς έχουμε αναφέρει παραπάνω πως στο FIQ mode, έχουμε πρόσβαση στους καταχωρητές από r8 μέχρι και r12. Επόμενως, υποθέτοντας πως στο FIQ mode, γίνεται αποθήκευση εκ νεου και στους δύο καταχωρητές r0 και r8, την τιμή 1, εάν αλλάξει και πάλι το mode λειτουργίας του επεξεργαστή και από το FIQ mode μεταπηδήσουμε και πάλι στο user mode, τότε οι αποθηκευμένες τιμές στους καταχωρητές r0 και r8 θα είναι 1 και 8 αντιστοιχα. Αυτό συμβαίνει ότι στο FIQ mode, ο επεξεργαστής έχει πρόσβαση σε καταχωρητές από r8 μέχρι και r12, και όχι στους r0 μέχρι και r7, επομένως μια αλλαγή στα δεδομένα αυτών των καταχωρητών, θα διατηρήσει και θα προκαλέσει μόνιμη αλλάγη των δεδομένων και στα υπόλοιπα modes λειτουργίας. Αντίθετα,

στους καταχωρητές r8 –r12, οι οποίοι είναι «οράτοι», στο mode λειτουργίας FIQ, μια αλλάγη στα δεδομένα τους, δεν θα προκαλέσει αλλάγη στους αντίστοιχους καταχωρητές του user mode, καθ'οτι επί της ουσίας αναφερόμαστε σε διαφορετικούς καταχωρητές, διότι όπως είχε αναφερθει και παραπάνς ένας καταχωρητής ,δεν διακρίνεται μόνο από την ονοματολογία του, αλλά και από το mode του επεξεργαστή, στο οποίο λειτουργεί.

Παρακάτω παρατίθεται το σχεδιάγραμμα κατανομής και οργάνωσης των καταχωρητών του επεξεργαστή ARM, ανάλογα με το όνομά τους και με το mode λειτουργίας στο οποίο ανήκουν.



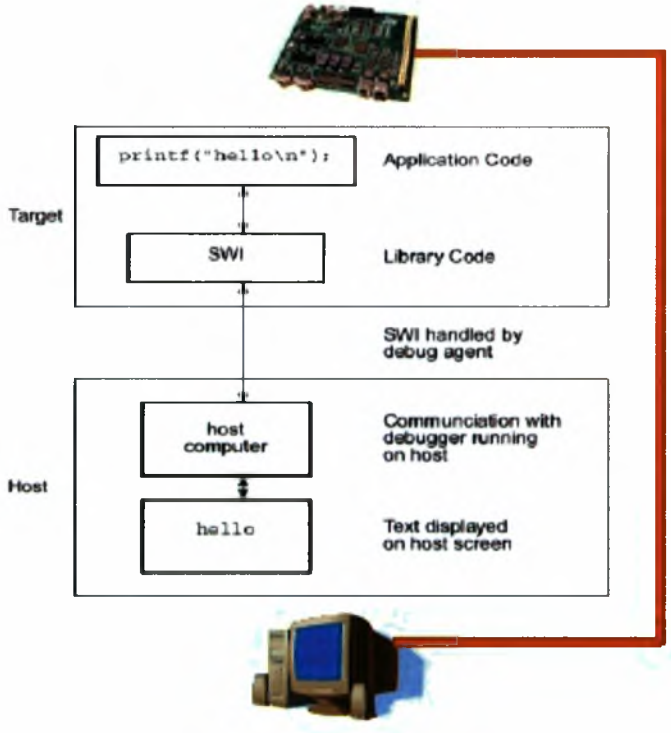
Γνωριμία με την πλατφόρμα X-board

Πρότου προχωρήσουμε στην ανάλυση των εφαρμογών που αναπτύξαμε συνετό θα ήταν να προχωρήσουμε στην εκ βαθέων γνωριμία με την πλατφόρμα, πανω στην οποία αναπτυχθηκε η εφαρμογή.

Όπως αναφέρθηκε και παραπάνω, η πλατφόρμα που χρησιμοποιήθηκε προέρχεται από την εταιρεία ADI, και περιέχει έναν επεξεργαστή Intel XScale, αρχιτεκτονικής ARM. Στην πλατφόρμα αυτή, δεν αναπ ' τυχθηκε η εφαρμογή, αλλα με βάση αυτή, η εφαρμογή μας σχεδιάστηκε και εκτελέστηκε. Η εφαρμογή μας υλοποιηθήκε πάνω σε έναν απλό - κοινό προσωπικό ηλεκτρονικό υπολογιστή (Personal Computer, PC). Παρακάτω επεξηγούμε, πώς γίνεται η διαδικασία επικοινωνίας και εμφάνισης του αποτελέσματος μιας πολύ απλης εντολής, όπως η *printf("hello world \n");* Αυτό που συμβαίνει είναι το εξής:

Καταρχήν ο προσωπικός ηλεκτρονικός υπολογιστής μας, πάνω στον οποίο γίνεται η ανάπτυξη της εφαρμογής μας λέγεται host computer, ενώ η πλατφόρμα επίδειξης και εκπαίδευσης του XScale, πάνω στην οποία θα τρέξει η εφαρμογή μας, λέγεται target computer. Αυτό που λαμβάνει χώρα λοιπόν, είναι ότι κατά την διάρκεια της εκτέλεσης μιας εντολής, για παράδειγμα της: *printf("hello world \n");*, ο target computer, η πλατφόρμα δηλαδή προσπαθεί να «αναγνωρίσει» την λειτουργικότητα της συγκεκριμενης εντολης. Για να το επιτύχει αυτό, απαραίτητη προϋπόθεση είναι να έχει πρόσβαση ο target computer, στον προσωπικό υπολογιστή, και ειδικότερα στο συνολο των βιβλιοθηκών που χρησιμοποιούνται από τον host computer για την αναπτυξη της εφαρμογής. Δεδομένου του γεγονότος αυτού, ο target computer, δημιουργεί μια διακοπή λογισμικού, ένα

software interrupt (SWI), το οπόιο και στέλνει στον host computer. Το software interrupt, το διαχειρίζεται ένας πράκτορας (agent) που β'ρισκεται στον host computer και επιστρέφει το αποτέλεσμα της εντολής στην οθόνη του χ'ρηστη, στον host computer.



Άγγελοι και Δαίμονες

Η πλατφόρμα περιλαμβάνει και δύο στοιχεία τα οποία συνεργάζονται μετάξυ τους και καθορίζουν την ανταλλάγη των δεδομένων μεταξύ των host και του target computer. Για να γίνει η παραπάνεω διαδικασία και να υπάρξει πραγματική συνεργασία μεταξύ της πλατφόρμας και του υπολογιστή στον οποίο λαμβάνει χώρα η αναπτυξη της εφαρμογής. Την διαδικασία αυτή την αναλαμβάνουν αυτά τα δύο στοιχεία, ο Δαιμονας (Debug Demon) και ο Άγγελος (Angel).

Ο Δαίμονας, βρίσκεται στον target computer, στην πλατφόρμα δηλαδή επάνω και είναι σχεδιασμενος εξολοκληρου από την ίδια την ARM, την κατασκευάστρια

εταιρεία του επεξεργαστή ARM. Είναι υλοποιημένος σε assembly, σε γλώσσα μηχανής δηλαδή, και για την επικοινωνία του με το με τον host computer χρησιμοποιεί ενα πρωτόκολλο byte-streaming (ροής δεδομένων, ανα byte). Αυτό που διασφαλίζει ο Δαίμονας ε΄ναι το γεγονός ότι ο κώδικας που αναπτύχθηκε από τον εξομοιωτή του επεξεργαστή ARM, που υπάρχει στον host computer, τον ARMulator, θα εκτελεστεί και στην κανονική πλατφόρμα του ARM, και σε κανονικό επεξεργαστή ARM δηλαδή και όχι μόνο σε εξομοιωτή.

Ο Άγγελος, Angel, χρησιμοποιείται για να κάνει την μετατροπή των δεδομένων σε γλώσσα μηχανής, σε ρουτίνες της γλώσσας προγραμματισμού C. Η επικοινωνία του με τον host computer, επιτυγχάνεται μέσω μεταβλητου μεγέθους πακέτα δεδομένων (variable-sized packet-based communication). Επιπρόσθετα, ο Angel, υποστηρίζει και οδηγούς εξωτερικών συσκευών (external devce drivers), όπως παραδείγματος χάριν, οδηγούς για μια συσκευή τοπικού δικτύου (Ethernet).

Ο Angel, αποτελείται από 2 μέρη:

- Τον Απασφαλματωτής (Debugger), ο οποίος εκτελείται πάνω στον host computer και ο οποίος δίνει τις εντολές στον Angel και εμφανίζει τα αποτελέσματα και τον
- Angel Debug Monitor, το οποίο είναι ένας έλεγχος αποσφαλμάτωσης των προγραμμάτων και ο οποίος εκτελείται στον target computer.
- Τέλος, υπαρχει και το Angel Debug Protocol (ADP) , το οποιο χρησιμοποιείται για την επικοινωνία, μεταξύ του host και του target computer, μεταξύ του προσωπικου υπολογιστή μας δηλαδη και του X-board.

Μέτα από όλα αυτά, ίσως κάποιος να αναρωτηθει, τι πραγματικα κάνει ο Angel; Η απάντηση είναι ότι ο Angel πραγματοποιεί μια σειρά απο ενεργεις κατα την διαρκεια της αρχικοποιησης και της εκκίνησης του συστήματος. Πιο συγκεκριμένα ο Angel, κατα την διαρκεια της φασης εκκινησης και αρχικοποίησης του επεξεργαστή, προβαίνει στις ακόλουθες ενέργειες:

- Προβαίνει στην βασική αρχικοποίηση του επεξεργαστή και του συστήματος (X-board).
- Δημιουργεί μια εντολή διακλάδωσης (branch) στην αρχή της μνήνης ROM,
- Ελέγχει το μέγεθος της μνήμης RAM,

- Φορτώνει δεδομένα από την ROM, στην RAM,
- Αρχικοποιεί τους διαχειριστές των διακοπών IRQ και FIQ
- Αρχικοποιεί τα όποια LED (φωτεινοί κρύσταλλοι) υπάρχουν στην πλατφόρμα επάνω.
- Αρχικοποιεί και καθαρίζει από τα πιθανόν άχρηστα δεδομένα το κανάλι της αποσφαλμάτωσης (debug channel) μεταξύ του host και του target computer, και τέλος προχωρεί στην ενεργοποίηση των διακοπών (interrupts)

Γενικότερα, ο Angel, προσφέρει μια πληθώρα απο δυνατότητες, οι οποιες θα αναλυθούν παρακάτω, όπως:

- Αρχικοποίηση του συστήματος
- Παροχή λειτουργικότητας σαν να επρόκειτο για λειτουργικό συστημα
- Υποστήριξη βιβλιοθηκών της γλώσσας προγραμματισμού C.
- Αποσφαλμάτωση (Debug) στον κώδικα του προγράμματος. Η τελευταία αυτη δυνατότητα παρέχεται μέσω της δυνατότητας να φορτώνει το πρόγραμμα, το εκτελέσιμο πρόγραμμα, από τον host computer στον target computer, και με το να υποστηρίζει σημεία διακοπής των προγραμμάτων (breakpoints).

Πιο συγκεκριμένα, και όσον αφορά την δυνατότητα αποσφαλμάτωσης του Angel, ο Angel μπορεί να υποστηρίξει βασικές λειτουργίες αποσφαλμάτωσης, όπως:
Η μεταφορά του προγράμματος (program downloading). Με αυτή τη λειτουργία ο απασφαλματωτής (debugger) που βρίσκεται στον host computer, στέλνει μια αλληλουχία από δεδομένα και μηνύματα με βάση το πρωτόκολλο ADP στον Angel που βρίσκεται πάνω στο target computer.
Η οριοθέτηση σημείων διακοπής εκτέλεσης προγράμματος (breakpoints setting). Με την λειτουργία αυτή, ο Angel, χρησιμοποιεί μη ορίσμενες εντολές γλώσσας μηχανής (undefined instructions), για να υποστηρίξει τα breakpoints. Από την στιγμή που θα οροθετηθεί το breakpoint, ο Angel, θα προβεί σε δύο ενέργειες:
Θα αποθηκεύσει την εντολή, στην οπόια πάνω γίνεται το breakpoint, έτσι ώστε να διασφαλιστεί ότι η κανονική εκτέλεση του προγράμματος θα επιστρέψει σε εκείνο ακριβώς το σημείο και όχι πουθενά αλλού μετα το πέρας του breakpoint και

Αντικαθιστά την εντολή με μια κατάλληλη μη ορισμένη εντολή
Τέλος, ο Angel, η κανονική εντολή επαναφέρεται και εκτελείται κανονικά, όταν όμως το breakpoint αφαιρείται, ή όταν η εκτέλεση του προγράμματος συνεχίζεται από το σημείο του breakpoint και μετά.

Όσων αφορά τώρα την δυνατότητα, που έχει ο Angel, στον χειρισμο και στην διαχείριση των σημείων διακοπής της κανονικής εκτέλεσης των προγραμμάτων (breakpoints handling), ο Angel μας παρέχει τα εξής:
Όταν ο Angel, εντοπίσει μια εντολή, η οποία να είναι ένα από πριν, προκαθορισμένο σημειοδιακοπής εκτελεσης του προγράμματος (breakpoint), τότε ο Angel:

1) Διακόπτει την εκτέλεση της εφαρμογής στον επεξεργαστή ARM,

2) Αποστέλει ένα μήνυμα στον host computer, ώστε να τον ενημερώει και να του υποδείξει πού ακριβώς βρίσκεται το breakpoint,

3) εκτελει μια «αυστηρή» και «σφιχτή» επαναληπτική διαδικασία (tight poll loop), στο σημειο ακριβώς εκείνο και αναμένει απάντηση από τον host computer.

Εάν όμως η εντολή δεν είναι ενα προκαθορισμένο breakpoint τοτε ο Angel:
Αναφέρει το γεγονός αυτό στον Debugger και το αναφέρει ως μια μη ορισμένη εντολη και
εκτελεί μια «αυστηρή» και «σφιχτή» επαναληπτική διαδικασία (tight poll loop), στο σημειο ακριβώς εκείνο και αναμένει απαντηση από τον host computer.

Ολοκληρώνοντας την ενότητα αυτή για τον Angel και πρίν προχωρησουμε παρακάτω στη λεπτομερή παρουσίαση του X-Board, μπορούμε να αναφέρουμε εν ολίγοις τα ακολουθα για τον Angel:

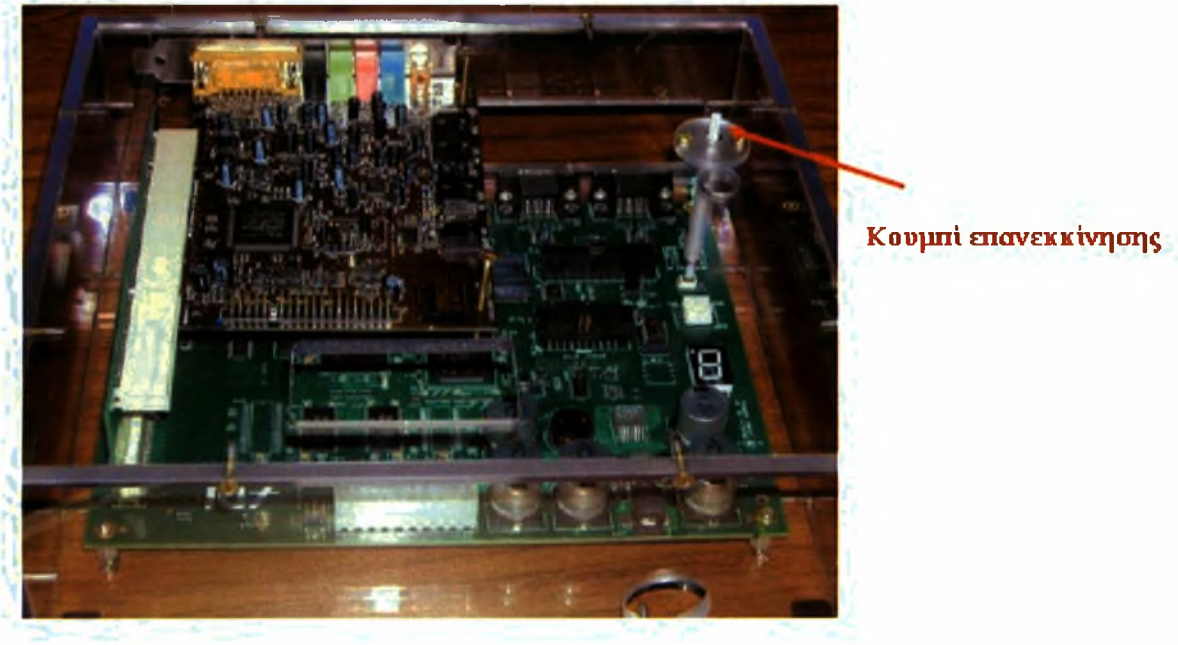
Σαν αρχή, αναφερόμαστε στις διαφοροποιήσεις του Angel οι οποίες είναι δύο: ο Minimal Angel και ο Full Angel. Ο Minimal Angel παρέχει μονο αρχικοποιηση του συστήματος μας και τίποτα άλλο, εν αντιθέσει με τον Full Angel, ο οποίος πέρα απο την αρχικοποίηση του συστήματος που πρσφέρει μας προσφερει και υποστηριξη των βιβλιοθηκών της γλώσσας C, όπως επίσης και δυνατότητες αποσφαλμάτωσης.

Σαν συνέχεια να αναφέρουμε το γεγονός πως η δυνατότητα αποσφαλμάτωσης του Angel χρησιμοποιείται μονο κατα την διαρκεια της ανάπτυξης του συστήματος.

Και μετά από την ανακεφαλαίωση των δυνατοτήτων του Angel, μπορούμε να προχωρήσουμε στην παρουσίαση του target computer, της πλατφόρμα ανάπτυξης της εφαρμογής μας, το X-board.

Αναλυτική περιγραφή του X-board

Όπως αναφέρθηκε και παραπάνω, το X-board, αποτελεί μια εκπαιδευτική πλατφόρμα ανάπτυξης εφαρμογών για ενσωματωμένα συστήματα. Η κατασκευάστρια εταιρεία είναι η ADI, ενώ ενσωματώνει επάνω του τον embedded επεξεργαστή XScale της Intel. Η μορφή του X-board είναι η ακόλουθη:

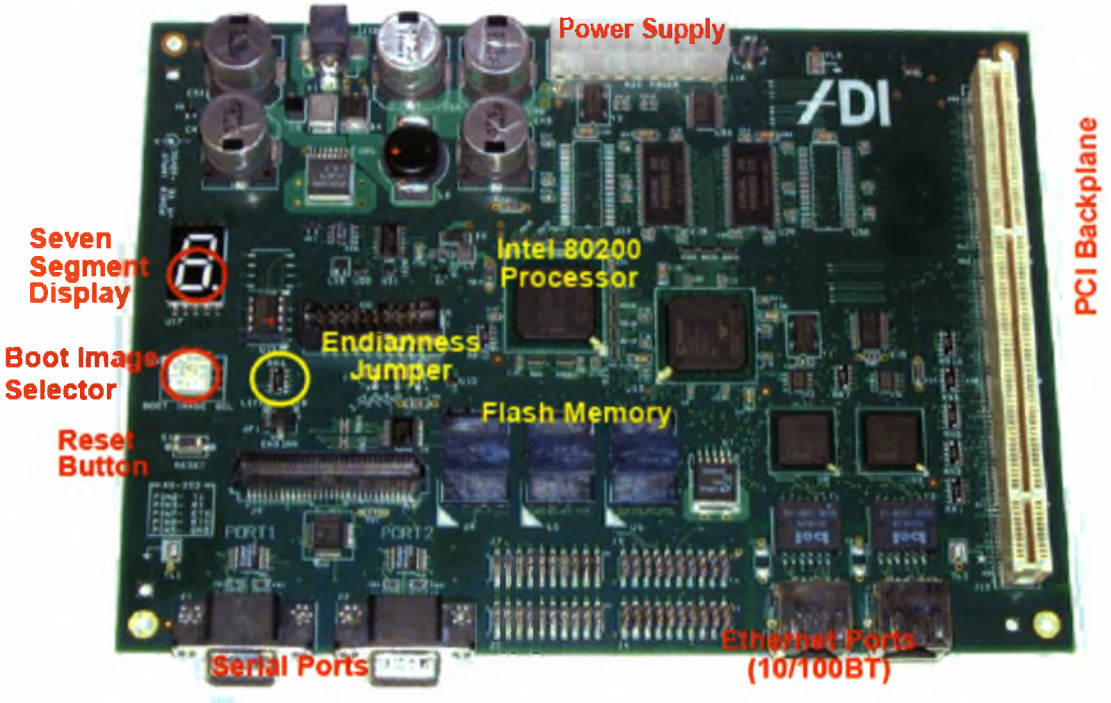


Φυσικά, το X-board, σαν πλήρης embedded υπολογιστική πλατφόρμα, δύναται να φέρει επάνω του, και επιπλέον συσκευές που θα την κάνουν να υποστηρίξει παραπάνω και επιπλέον δυνατότητες, όπως για παράδειγμα μια κάρτα ήχου για την ψηφιακή επεξεργασία δεδομένων φωνής και ήχου ή ένα αριθμητικό πληκτρολογιο, για την υποστήριξη των βασικών εντολών εισοδου στο ενσωματωμενο συστημα, από τον χρήστη του συστήματος, απευθείας σε αυτό και

χωρις να μεσολαβεί το host computer. Στα πλαίσια της εργασίας αυτής, και της εφαρμογής η οποίας αναπτύχθηκε, χρησιμοποιήθηκε η συσκευή του αριθμητικού πληκτολογίου, φωτογραφία της οποίας παρατίθεται παρακάτω:

Πλεον, η πλατφόρμα μας, με συνδεδεμένο το αριθμητικό πληκτρολόγιο έχει την παρακάτω μορφή.

Τα βασικα μερη της πλατφορμας αυτής, μερη τα οποία μπορεί να τα διακρίνει κανείς δια γυμνού οφθαλμού είναι τα ακόλουθα:



Η πλατφόρμα πάνω στην οποία πραγματοποιήθηκε η διπλωματική εργασία και στην οποία επάνω εκτελέστηκε η εφαρμογή διαθέτει τα ακόλουθα τεχνικά χαρακτηριστικά:

Επεξεργαστής	XScale – based Intel 80200 @ 733 MHz
Μνήμη SDRAM	128MB @ 100MHz
Μνήμη Flash	4MB, 100 – 150ns
Ethernet – 2 Ports	10 / 100 BaseT
Serial – 2 Ports	RS232, 115200K

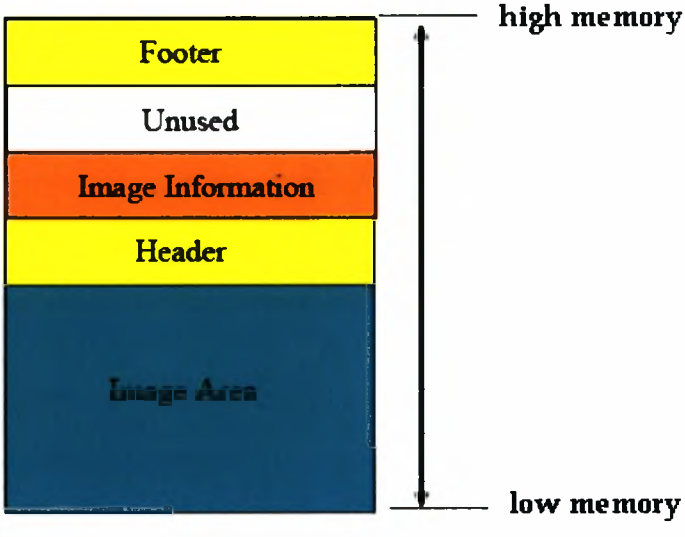
Όπως αναφέρθηκε και στα τεχνικά χαρακτηριστικά της πλατφόρμας, το X-board διαθέτει μνήμη τύπου flash. Προχωρούμε αμέσως παρακάτω στην περιγραφή

αυτού του ειδους της μνημης και στο πως αυτή ειναι οργανωμενη στο X-board, καθ'ότι η λειτουργικότητα της, χρησιμοποιήθηκε στην ανάπτυξη της εφαρμογής.

Η flash memory είναι ένας τύπος μνήμης, μη ευμετάβλητη (non-volatile), της οποίας τα δεδομενα ειναι δυνατόν να σβηστούν και η ιδια η μνημη είναι δυνατόν να επαναπρογραμματιστεί ηλεκτρικά. Οι μνήμες τύπου flash αποθηκεύουν τα δεδομένα σε μπλοκ. Έτσι λοιπον έαν θέλουμε να γράψουμε ή και να διαβάσουμε απο μιά μνήμη τέτοιου τύπου δεδομένα, αυτό θα γίνει ανα ένα ή ανα αριθμό μπλοκ την κάθε φορα. Το X-board διαθέτει 4MB μνήμης τύπου flash, η οποία είναι διαχωρισμένη σε δύο λογικές συσκευές:

Η μια από τις δύο λογικές συσκευές αποτελεί το τμήμα «φορτώματος» των προγραμμάτων στην μνήμη (boot portion), και περιλαμβάνει το boot monitor, το οποίο αποτελεί τον «ελεγχο» και της διαδικασίας «φορτώματος των αρχικών προγραμμα'των στην μνήμη, τον πελάτη (client) του αποσφαλματωτή Angel καθώε και κάποια αρχεία τα οποία χρησιμοποιούνται για να κ'νει τεστ τα δεδομένα της μνήμης καθως και τα δεδομένα του boot monitor, το ίδιο το X-board.

Η δευτερη λογική συσκευή είναι το μέρος της μνήμης που χρησιμοποιείται για την αποθήκευση των εφαρμογών σε αυτήν. Σε αυτό το μέρος της μνήμης, αποθηκεύεται η προς εκτελεση εφαρμογή μας.



Στο σημειο αυτό θα πρεπει να τονιστεί το γεγονός ότι δεν επιτρέπεται σε καμιά περίπτωση να ορίσει ο χρήστης του X-board, ως «εξ'ορισμού» τομέα της μνήμης,

τον τομέα του boot portion, καθ' ότι το γεγονός αυτό θα κατέστρεφε ολοκληρωτικά την μνήμη και θα ήταν αδύνατο από εκεί και στο εξής να ξεκινήσει να λειτουργεί η πλατφόρμα X-board, καθώς θα έσβηναν τα δεδομένα του boot portion και δεν θα ήταν να προσπελάσουν τα δεδομένα αυτά από το X-board, με αποτέλεσμα να μην μπορεί να εκκινηθεί και να αρχικοποιηθεί το όλο ενσωματωμένο συστημα.

Όπως φαίνεται και στην παραπάνω φωτογραφία που αναφέρεται στην οργάνωση της μνήμης flash, που βρίσκεται στο X-board, διακρίνουμε τις εξής περιοχές – τομεις:

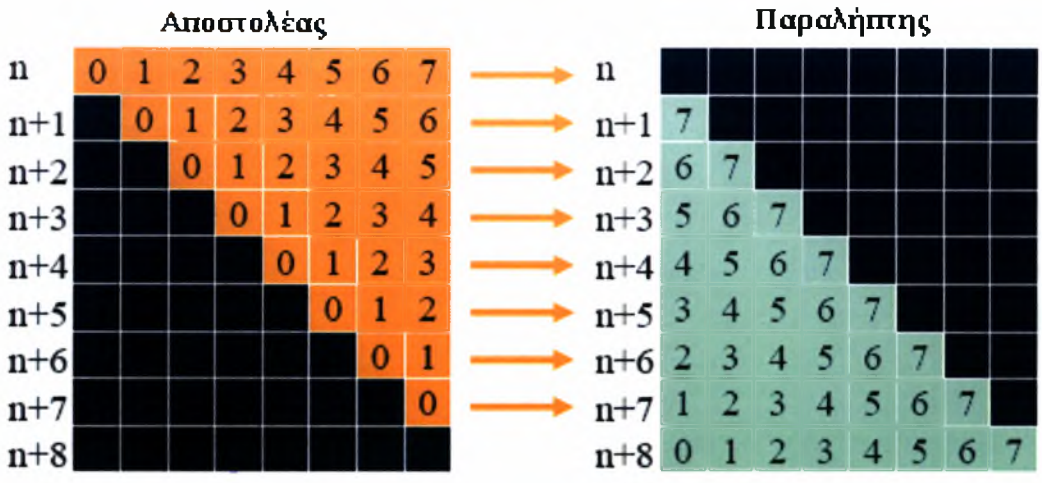
- Image area: Αποτελεί την περιοχή της μνήμης στην οποια αποθηκεύονται όλος ο κωδικας και τα τμήματα δεδομενα που μπορούν να χρησιμοποιηθούν μόνο για αναγνωση(read-only data segments)
- Πληροφορίες Επικεφαλίδας (Header Information): Στο τμήμα αυτό αποθηκεύονται όλοι οι headers(επικεφαλίδες) των αρχείων που έχουν φορτωθεί από τον host computer, στο X-board.
- Image Information: Στο τμήμα αυτό αποθηκεύονται το όνομα του image, καθώς και τα χαρακτηριστικά του(image identification – image name) καθώς και οι λειτουργίες του κώδικα, οι οποίες προστίθενται εκεί από την βιβλιοθήκη της flash μνήμης.
- Μη χρησιμοποιημένη μνήμη (Unused flash): Τομέας της μνήμης που δεν χρησιμοποιείται.
- Πληροφορίες Footer (Footer Information): Με την εννοια footer, εννοούμε το αποτύπωμα που αφήνει η μνήμη. Οι πληροφορίες αυτές του footer περιέχουν:
- Την διευθυνση του μπλοκ της πληροφοριας για το συγκεκριμένο image της μνήμης.
- Την διεύθυνση βάσης (Base Address) για τα δεδομένα
- Μια μοναδικά 32-bit τιμή, η οποία μας χρησιμεύει στην γρήγορη αναζήτηση των δεδομένων, μέσα στην μνήμη.
- Ο τύπος του Image. Δηλαδή μπορεί να είναι τύπου μπλοκ, τύπου image, τύπου SIB, τύπου δεδομένων.

- Άθροισμα ελέγχου(checksum), για την εγκυρότητα και την ακεραιότητα τον δεδομένων που βρίσκονται στο footer.

Όπως παρουσιάσαμε λίγες παραγράφους πριν, το X-board, διαθέτει δύο σειριακές θύρες, από τις οποίες η μία χρησιμοποιείται για την σύνδεση και την επικοινωνία του X-board με τον υπολογιστή, τον host computer, και η άλλη χρησιμοποιείται για την σύνδεση πάνω στο X-board, εξωτερικών, περιφερειακών συσκευών. Καθίσταται σαφές ότι η χρήση της σειριακής θύρας και η γνώση της λειτουργίας της είναι κάτι παραπάνω από απαραίτητα για την ανάπτυξη της εφαρμογής μας. Προχωρούμε λοιπόν στην αναλύση της λειτουργίας της σειριακής θύρας.

Η σειριακή θύρα, χρησιμοποιείται για την σειριακή επικοινωνία μεταξύ δυο υπολογιστικών συστημάτων και την ανταλλαγή δεδομένων μεταξύ τους. Η χρησιμότητα της και ο τρόπος υλοποίησης και λειτουργικότητας της, την καθιστούν σημαντικότατο τρόπο επικοινωνίας. Αυτό που λαμβάνει χώρα στην σειριακή θύρα είναι ότι στέλνονται και λαμβάνονται δεδομενα του ενος bit, την κάθε φορα, πανω απο κάθε καλώδιο. Παρά το γεγονός πως χρειάζονται οκτώ χρονικές στιγμές για να σταλεί ενα byte, καθ' όσων γνωρίζουμε 1 byte = 8 bits, χρειάζονται μόλις λίγα καλώδια για να επιτευχθεί αυτό. Σε μια τυπική σειριακή επικοινωνία, απαιτούνται τρία, μόλις, καλώδια. Ένα για την αποστολή των δεδομένων, ένα για την λήψη των δεδομένων καθώς και ένα κοινό καλώδιο, το οποίο χρησιμοποιείται σαν γείωση.

Παρακάτω, παρατίθεται μια απλοϊκή άποψη του τρόπου λειτουργίας της σειριακής θύρας.



Αυτό που λαμβάνει χώρα, είναι το γεγονός ότι διαθέτουμε δύο καταχωρητές, τον καταχωρητή εκπομπής δεδομένων και τον καταχωρητή λήψης δεδομένων (Transmit register – Tx, Receive register - Rx). Αυτό που συμβαίνει είναι το γεγονός πώς κάθε φορά που ο Tx είναι άδειος, δημιουργεί μια διακοπή (interrupt), σημαίνοντας πως το byte που ήταν αποθηκευμένο στο buffer του, απεστάλη και τώρα είναι έτοιμος να προχωρήσει στην επόμενη αποστολή byte προς το receiver. Όπως αναφέραμε και πριν, κάθε χρονική στιγμή, μεταφέρεται ένα bit, και αυτό φαίνεται καθαρά στην παραπάνω εικόνα, όπου σε 8 χρονικές στιγμές αποστέλεται ένα byte.

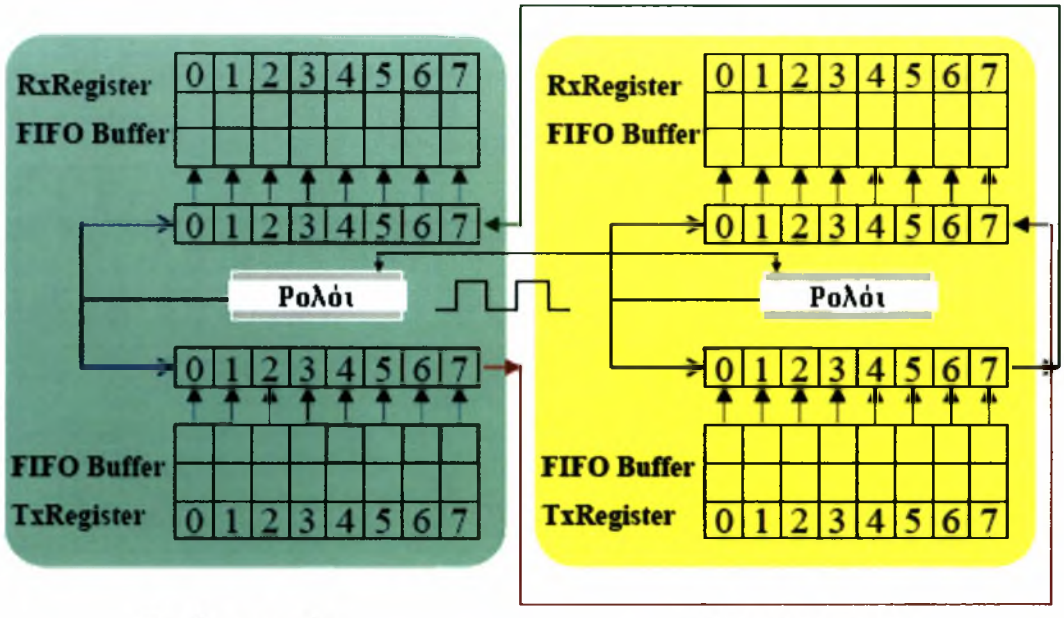
Υπάρχουν περιπτώσεις κατά την διάρκεια της σειριακής μετάδοσης που μπορεί τα δεδομένα να χαθούν. Αυτό μπορεί να γίνει σε δύο περιπτώσεις:

Α) Εάν ο transmitter ξεκινήσει την επόμενη μετάδοση του επόμενου byte, προτού ο receiver να προλαβει να επεξεργαστεί ή και να διαβάσει το byte που μόλις έλαβε από τον transmitter.

Β) Εάν το επόμενο, προς μετάδοση, byte, φορτωθεί στον transmitter προτού το τρέχον byte που μεταδίδεται να προλάβει να αποσταλεί.

Για τους δύο παραπάνω λόγους, οι περισοτερες σειριακές θύρες χρησιμοποιούν FIFO buffers, έτσι ώστε να διασφαλίσουν ότι δεν θα χαθούν τα δεδομένα. Έτσι λοιπόν χρησιμοποιούνται buffers και στον receiver αλλα και στον transmitter. Στον

receiver, χρησιμοποιούνται για να αποθηκεύονται, ώστε να μπορούμε να τα επεξεργαστούμε αργότερα, ενώ στον transmmiter για να μεταδοθούν αργότερα.
Το παρακάτω σχήμα δείχνει πολύ επεξηγηματικά την λειτουργία μιας σειρικαής επικοινωνίας, με την χρήση FIFO buffers και στον transmitter, αλλα και στον receiver.



Υπάρχουν δύο διαφορετικοί τύποι σειριακής επικοινωνίας. Ο ένας τύπος είναι η σύγχρονη επικοινωνία και ο άλλος είναι η ασύγχρονη επικοινωνία.
Με τον όρο συγχρονη επικοινωνία εννοούμε ότι ολα τα bits που μεταδίδονται, μεταδίδονται συγχρονισμενα, με βάση ε'να κοινο σήμα ρολογιού. Επίσης, για να επιτευχθεί αυτού του είδους η επικοινωνία απαιτείται οι δυο συσκευές να έχουν αρχικα συγχρονιστει μεταξύ τους, και να συνεχίσουν να στέλνουν δεδομένα μεταξύ τους, για όσο χρόνο ειναι συνδεδεμενοι μεταξύ τους, ώστε να παραμείνουν συγχρονισμένοι. Το προηγούμενο, υποδηλώνει πως ακόμα και όταν δεν υπάρχουν δεδομένα για ματεφορά από την μια συσκεύη στην άλλη, οι δύο συσκευές θα συνεχίσουν να στέλνουν δεδομένα, κενους χαρακτηρες για την ακριβεια (idle characters), ώστε να παραμεινουν συγχρονισμένες. Η σύγχρονη μετάδοση προσφέρει μεγαλύτερες ταχύτητες μετάδοσης δεδομένων, καθ'ότι δεν απαιτει την ύπαρξη επιπλέον bit, για να υποδηλωθεί έτσι, στην άλλη συσκευή η έναρκη και η λήξη της μετάδοσης του κάθε byte.

Με τον όρο ασύγχρονη μετάδοση, εννοούμε ότι κάθε συσκευή χρησιμοποιεί το δικό της ρολόι, και με βάση αυτό τα δεδομένα αποστέλλονται. Αυτό έχει ως αποτέλεσμα η μετάδοση των δεδομένων, από την μια συσκευή στην άλλη, να γίνεται σε αυθαίρετους χρόνους. Σε αυτή την περίπτωση λοιπόν, αντί να χρησιμοποιούμε τα ρολόγια των συσκευών, για τον συγχρονισμό τους, χρησιμοποιούμε μια συγκεκριμενη αλληλουχία δεδομένων, η οποία εαν ληφθεί από την συσκευή παραλήπτης, υποδηλώνει την έναρξη ή την λήξη της μετάδοσης των δεδομένων. Κλεινοντας, όσον αφορά τους τύπους της σειριακής επικοινωνίας, παρά το γεγονός ότι η ασύγχρονος επικοινωνία είναι ελάχιστα πιο αργή από της συγχρονη επικοινωνία, έχει το πλεονέκτημα, πως ο πεξεργαστής υποχρεώνεται να απασχολείται με την αποστολή και λήψη των κενων χαρακτήρων (idle characters), όπως ακριβώς γίνεται στην συγχρονη επικοινωνία.
Οι σειριακές θύρες που συναντά κανείς σήμερα στους προσωπικούς υπολογιστές, υποστηρίζουν μόνο ασύγχρονη επικοινωνία.

Προχωρώντας την ανάλυση της σειριακής επικοινωνίας, ήρθε η στιγμή να παρουσιαστεί και η απαραιτητη τεχνητη ορολογια, η οποια μας είναι απαραίτητη για την μετέπειτα ανάλυση μας.
Σε μια σειριακή επικοινωνία, παίρνουν μέρος τέσσερα, βασικά, μέρη. Το DTE, το DCE, το RS232, καθώς και το Baud Rate.
Το DTE είναι το ακρονύμιο του Data terminal equipment, το οποίο είναι για παράδειγμα, ένας προσωπικός υπολογιστής.
Το DCE είναι το ακρονύμιο του Data communication equipment, το οποίο ειναι η απομακρυσμένη συσκευή, με την οποία επικοινωνούμε. Στην προκειμένη περίπτωση, η απομακρυσμένη συσκευή είναι το X-board.
Το RS232, είναι το ακρονύμιο του Recommended Standard number 232, και είναι ένα προτοκολλο, σειριακής επικοινωνίας.
Το Baud Rate, αποτελεί τον αριθμό των φόρων, μέσα σε ένα δευτερόλεπτο, κατά την διάρκεια του οποίου η γραμμή μετάδοσης, αλλάζει κατάσταση. Αξίζει να σημειωθεί, και με βάση την επεξήγηση του όρου Baud Rate, ότι το Baud Rate, δεν σημαίνει πάντα τον αριθμό των bits που μεταδίδονται κάθε στιγμή, ανα δευτερόλεπτο.

Τέλος, θα αναφερθούμε στο βύσμα της σειριακής θύρας. Το οποίο βύσμα, αποτελείται είτε από 9 pins έιτε από 25 pins. Στο X-board, επάνω, η σειριακή θύρα που βρίσκεται, έχει 9 pins, επομένως θα αναλύσουμε αμέσως παρακάτω αυτού του είδους το βύσμα. Μέσα σε ένα βύσμα σειριακής με 9 pins, το κάθε ένα από τα 9 pins είναι:

Carrier Detect: Ελέγχει εάν το DCE είναι συνδεδεμένο με μια εν λειτουργία τηλεφωνική γραμμή

Receive Data: Ο υπολογιστής δέχεται τα δεδομένα που στέλνονται από το DCE

Transmit Data: Ο υπολογιστής αποστέλλει τα δεδομένα στο DCE

Data Terminal Ready: Ο υπολογιστής αναφέρει στο DCE ότι είναι έτοιμο να επικοινωνήσει μαζί του.

Signal Ground: Το Pin αυτό είναι στην γείωση

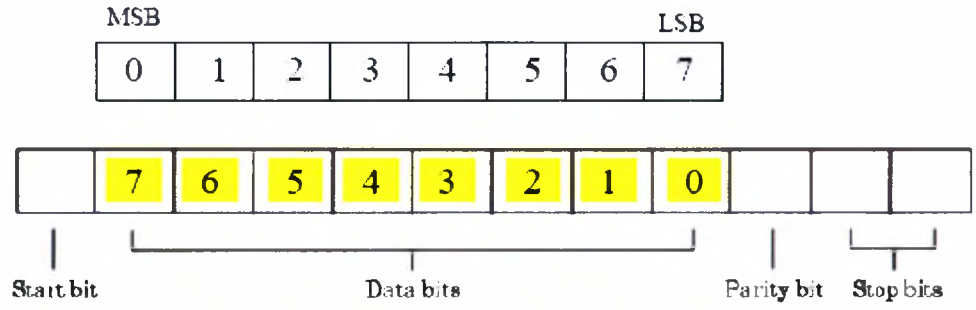
Data Set Ready: Το DCE αναφέρει στον υπολογιστή ότι είναι έτοιμο να επικοινωνήσει μαζί του.

Request To Send: Ο υπολογιστής ερωτά το DCE εάν μπορεί να του στέιλει δεδομένα

Clear To Send: Το DCE ερωτά τον υπολογιστή εάν μπορεί να του στέιλει δεδομένα

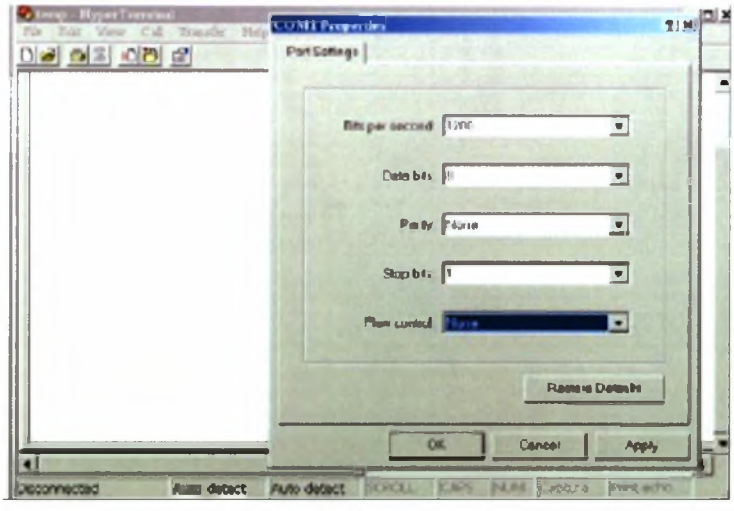
Ring Indicator: Μόλις έχει γίνει η κλήση, ο υπολογιστής επιβεβαιώνει το γεγονος αυτό.

Ένα byte που μεταδίδεται μέσω σειριακής θύρας, αποτελείται συνήθως από τους χαρακτήρες και από τα framing bits. Οι χαρακτήρες κυμαίνονται από 5-8 bits, ενώ τα framing bits, αποτελούνται συνήθως από 1 bit έναρξης μετάδοσης (start bit), 1 parity bit και 1-2 bit λήξης μετάδοσης (stop bits).



Το parity bit αποτελεί μια πολύ απλοϊκή διαδικασία ελέγχου των δεδομένων που μεταδίδονται. Διακρίνουμε δύο περιπτώσεις ελέγχου: την περιττή ισότητα (even parity), και την άρτια ισότητα (odd parity). Στην περιττή ισότητα, ο αριθμός των bit των δεδομένων συν το parity bit, θα πρέπει να παράγουν έναν περιττό αριθμό από άσους (1), ενώ στην άρτια ισότητα το ίδιο άθροισμα θα πρέπει να παράγει ένα άρτιο αριθμό άσων (1). Τον τύπο ελέγχου, εάν θα έιναι τύπου άρτιας ή περιττης ισότητας, τον καθορίζει ο αποστολέας των δεδομένων, ενώ ο παραλήπτης ελέγχει εάν πραγματικά τα δεδομένα που έλαβε είανι έγκυρα και δεν περιέχουν λάθη. Τα μειονεκτήματα του ελέγχου μέσω parity bit, είναι ότι μπορούν να εντοπιστουν λάθη άρτιου αριθμού μεταβολών στα δεδομένα, ενώ πολλαπλά λάθη, είναι δυνατόν να παράγουν αποτέλεσμα, πως τα δεδομένα αυτά είναι έγκυρα, ενώ αυτά να μην είναι καθόλου.

Στο σημείο αυτό θα αναφερθούμε στο πρόγραμμα επικοινωνίας του ηλεκτρονικού υπολογιστή, το hyperterminal, το οποίο καθιστά εφικτή την επικοινωνία μεταξύ των υπολογιστών και της απομακρυσμένης συσκευής, μέσω σειριακής επικοινωνίας. Το πρόγραμμα, μας επιτρέπει να παραμετροποιήσουμε της σειριακή επικοινωνία, δίνοντας μας την δυνατότητα, να αλλάζουμε και να τροποποιούμε ανάλογα με την συσκευή και τις ανάγκες επικοινωνίας μας την σειριακή θύρα, στοιχεία όπως: το Baud Rate, τον αριθμό των data bits, τον αριθμό των parity bits, τον αριθμό των stop bits, καθώς και το flow control.



Παράρτημα Α

Το Εργαλείο που χρησιμοποιήσαμε

Η συγγραφή κώδικα έγινε σε γλώσσα C, ενώ χρησιμοποιήθηκε και το εργαλείο CodeWarrior IDE/ AXD Debugger για την εκτέλεση των προγραμμάτων στον ARM επεξεργαστή.

Παρακάτω, παραθέτουμε κάποιες πληροφορίες για τον τρόπο χρήσης του εργαλείου αυτού. Καταρχήν, ξεκινάμε το πρόγραμμα CodeWarrior IDE (Start Programs->ARM Developer Suite->CodeWarrior). Στην συνέχεια δημιουργούμε ένα νέο project ακολουθώντας τα παρακάτω βήματα :

- File->New
- Επιλέξτε το πεδίο Project
- Location-> Επιλέξτε το location του καινούργιου project
- Όνομα-> Επιλέξτε ένα όνομα για το project
- Add Files to project: Επιλέξτε project-> Add Files

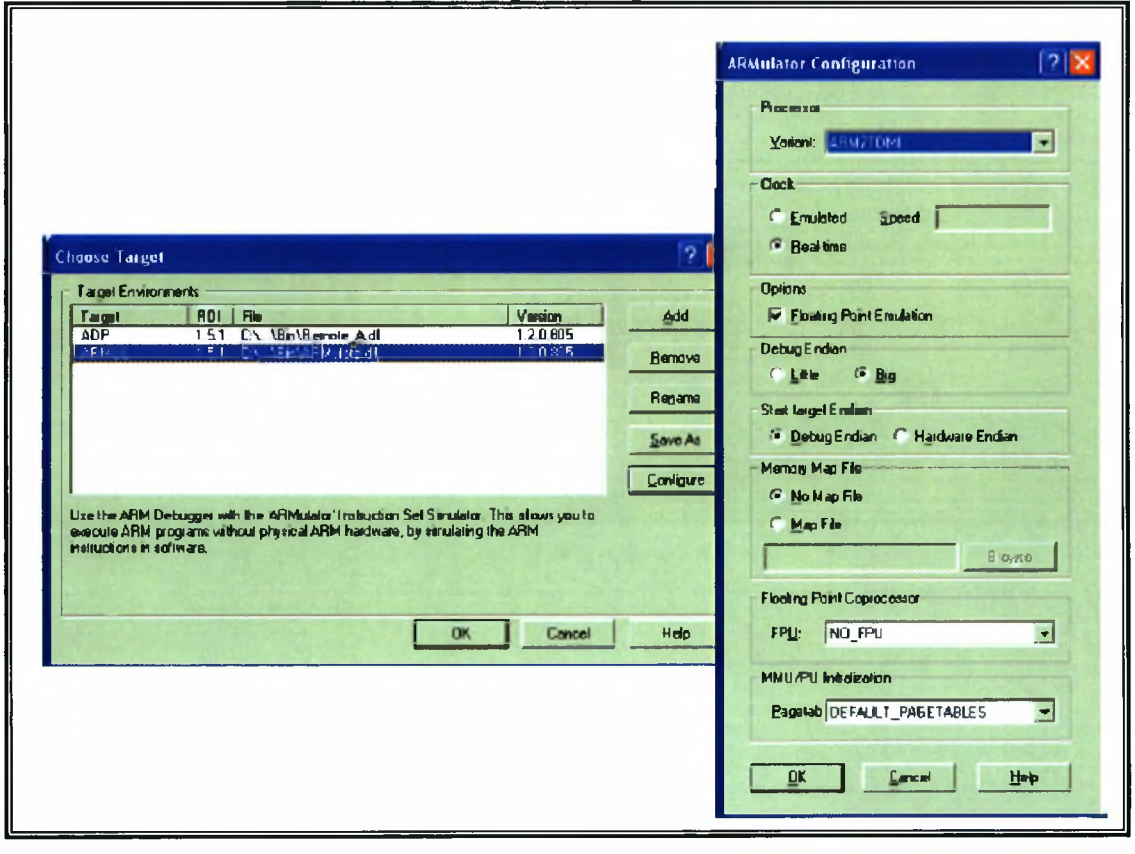
Συνεχίζουμε με το να εισάγουμε τις απαραίτητες ρυθμίσεις. Επομένως, έχουμε ότι :

- Κατάλληλες ρυθμίσεις για την χρήση του Big Endian τρόπου αποθήκευσης: Επιλέξτε Edit-> DebugRel Settings-> Language Settings

- Για τους ARM Assembler, C Compiler, C++ Compiler, Thumb C Compiler και Thumb C++ Compiler επιλέξτε:
  - Architecture or Processor: ARM7TDMI
  - Floating Point: Pure-endian softfp
  - Byte Order: Big Endian

Όταν είμαστε έτοιμοι να προχωρήσουμε στην μεταγλώττιση του κώδικα, επιλέγουμε:

- Project-> Debug (or press F5)
- Εμφανίζεται το AXD παράθυρο. Επιλέξτε Options->Configure Target
  - Επιλέξτε ARMUL Target (ARM emulator)
    - Click configure- σιγουρευτείτε ότι τα ισχύουν τα παρακάτω(βλέπε και την φωτογραφία):
      - Variant: ARM7TDMI
      - Clock: Real – Time
      - Floating Point Emulation is checked
      - Debug Endian is on Big
      - Also No Map File
      - NO_FPU Floating point Coprocessor



Εάν θέλουμε να παράγουμε ένα αρχείο με την συμπεριφορά του συστήματος, ακολουθούμε τα παρακάτω βήματα:

- Στον AXD Debugger, πηγαίνετε στην επιλογή File-> Load Image – στην λίστα που εμφανίζεται επιλέγουμε το δικό μας AXF αρχείο, τσεκάρουμε την επιλογή Profile και επιλέγουμε το Call graph profiling
- Επιλέξτε Options-> Profiling-> Toggle Profiling
- Πηγαίνετε στην επιλογή Execute-> Run (or F5), το εκτελούμε αυτό δυο φορές για να τρέξουμε το πρόγραμμά μας

Στην συνέχεια, έχουμε:

- Επιλέξτε Options->Profiling->Toggle Profiling
- Στο AXD Debugger παράθυρο, επιλέγουμε Options-> Profiling-> Write to File
- Δίνουμε ένα όνομα και αποθηκεύουμε το .prf αρχείο σε μια τοποθεσία
- Ανοίγουμε μια γραμμή εντολών και πληκτρολογούμε "armprof <filename.prf><profilingReport.txt>"

Το αρχείο που θα δημιουργηθεί από την εκτέλεση του προγράμματος, μοιάζει κάπως έτσι:

```
Name                    cum%    self%   desc%   calls
main                    96.4%   0.16%   95.88%  0
   qsort                        0.44%   0.75%   1
   _printf                      0.00%   0.00%   3
   clock                        0.00%   0.00%   6
   _sprintf                     0.34%   3.56%   1000
   randomise                    0.12%   0.69%   1
   hell_sort                    1.59%   3.43%   1
   insert_sort                  19.91%  59.44%  1
-------------------------------------------------------
   main                         19.91%  59.44%  1
insert_sort             79.35%  19.91%  59.44%  1
   strcmp                       59.44%  0.00%   243432
-------------------------------------------------------
   qs_string_compare            3.17%   0.00%   13021
   shell_sort                   3.43%   0.00%   14059
   insert_sort                  59.44%  0.00%   243432
strcmp                  66.05%  66.05%  0.00%   270512
```

Τέλος, εάν θέλουμε να βρούμε τον συνολικό χρόνο εκτέλεσης του προγράμματος, ακολουθούμε τα παρακάτω βήματα:

- Στο AXD παράθυρο, επιλέξτε System Views-> Debugger Internals

- Επιλέξτε Statistics
- Ο αριθμός που χαρακτηρίζεται ως "Total" σε αυτό το παράθυρο, είναι ο συνολικός χρόνος εκτέλεσης του προγράμματός μας

Παράρτημα Β

Στο παρόν παράρτημα, θα δούμε τις βασικότερες εντολές σε assembly, μερικές από τις οποίες χρησιμοποιήσαμε και στην πρώτη εφαρμογή κώδικα που γράψαμε (βλέπε Παράρτημα Γ).

Assembly Εντολές

Παρακάτω παραθέτουμε τις εντολές assembly του επεξεργαστή XScale της Intel.

Intel® XScale™ Microarchitecture Assembly Language Quick Reference Card
ARM Instruction Set

Operation		Assembler	Updates	Action	Notes
Move	Move	MOV{cond}{S} Rd, <Oprnd2>	N Z C	Rd := Oprnd2	
	NOT	MVN{cond}{S} Rd, <Oprnd2>	N Z C	Rd := 0xFFFFFFFF EOR Oprnd2	
	SPSR to register	MRS{cond} Rd, SPSR		Rd := SPSR	
	CPSR to register	MRS{cond} Rd, CPSR		Rd := CPSR	
	register to SPSR	MSR{cond} SPSR_<fields>, Rm		SPSR := Rm (selected bytes only)	
	register to CPSR	MSR{cond} CPSR_<fields>, Rm		CPSR := Rm (selected bytes only)	
	Immediate to SPSR	MSR{cond} SPSR_<fields>, #<Immed_8r>		SPSR := Immed_8r (selected bytes only)	
	Immediate to CPSR	MSR{cond} CPSR_<fields>, #<Immed_8r>		CPSR := Immed_8r (selected bytes only)	
Arithmetic	Add	ADD{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn + Oprnd2	
	with carry	ADC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn + Oprnd2 + Carry	
	saturating	QADD{cond} Rd, Rm, Rn		Rd := SAT(Rm + Rn)	Sticky, No shift/rotate
	double saturating	QDADD{cond} Rd, Rm, Rn		Rd := SAT(Rm + SAT(Rn * 2))	Sticky, No shift/rotate
	Subtract	SUB{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn - Oprnd2	
	with carry	SBC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn - Oprnd2 - NOT(Carry)	
	reverse subtract	RSB{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Oprnd2 - Rn	
	reverse subtract with carry	RSC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Oprnd2 - Rn - NOT(Carry)	
	saturating	QDSUB{cond} Rd, Rm, Rn		Rd := SAT(Rm - Rn)	Sticky, No shift/rotate
	double saturating	QDSUB{cond} Rd, Rm, Rn		Rd := SAT(Rm - SAT(Rn * 2))	Sticky, No shift/rotate
	Multiply	MUL{cond}{S} Rd, Rm, Rs	N Z C	Rd := (Rm * Rs)[31:0]	
	accumulate	MLA{cond}{S} Rd, Rm, Rs, Rn	N Z C	Rd := ((Rm * Rs) + Rn)[31:0]	
	unsigned long	U{MUL,MLA}L{cond}{S} RdLo, RdHi, Rm, Rs	N Z C V	RdHi,RdLo := unsigned(Rm * Rs)	
	unsigned accumulate long	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C V	RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)	
	signed long	SMULL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C V	RdHi,RdLo := signed(Rm * Rs)	
	signed accumulate long	SMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C V	RdHi,RdLo := signed(RdHi,RdLo + Rm * Rn)	
	signed 16 * 16 bit	SMULW{y}{cond} Rd, Rm, Rs		Rd := Rm[x] * Rs[y]	No shift/rotate
	signed 32 * 16 bit	SMULW{y}{cond} Rd, Rm, Rs		Rd := (Rm * Rs[y])[47:16]	No shift/rotate
	signed accumulate 16 * 16	SMLA{x}{cond} Rd, Rm, Rs, Rn		Rd := Rn + Rm[x] * Rs[y] 16bit	No shift/rotate
	signed accumulate 32 * 16	SMLAW{y}{cond} Rd, Rm, Rs, Rn		Rd := Rn + (Rm * Rs[y])[47:16]	Sticky, No shift/rotate
	signed accumulate long 16 * 16	SMLAL{x}{cond} RdLo, RdHi, Rm, Rs		RdHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]	No shift/rotate
	Count leading zeroes	CLZ{cond} Rd, Rm		Rd := number of leading zeroes in Rm	
DSP CP0	Multiply with internal accumulate	MIA{cond} acc0, Rm, Rs		acc0 := Rm[31:0] * Rs[31:0]+(Rm[39:0] + acc0[39:0])	
		MIAPH{cond} acc0, Rm, Rs		acc0 := (sign_extend(Rm[31:16] * Rs[31:16]) + sign_extend(Rm[15:0] * Rs[15:0]) + acc0[39:0]	
		MIAxy{cond} acc0, Rm, Rs		acc0[39:0] = sign_extend(Rm[x] * Rs[y]) + acc0[39:0]	
	Accumulator move to	MAR{cond} acc0, RdLo, RdHi		acc0[39:32] = RdHi[7:0] acc0[31:0] = RdLo[31:0]	
	Accumulator move from	MRA{cond} RdLo, RdHi, acc0		RdHi[31:0] = sign_extend(acc0[39:32]) RdLo[31:0] = acc0[31:0]	
Logical	Test	TST{cond} Rn, <Oprnd2>	N Z C	Update CPSR flags on Rn AND Oprnd2	
	Test equivalence	TEQ{cond} Rn, <Oprnd2>	N Z C	Update CPSR flags on Rn EOR Oprnd2	
	AND	AND{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn AND Oprnd2	
	EOR	EOR{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn EOR Oprnd2	
	ORR	ORR{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn OR Oprnd2	
	Bit Clear	BIC{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn AND NOT Oprnd2	
	No operation	NOP		Rd := Rd	Flags not affected.
	Shift/Rotate				See Table Operand 2.
Compare	Compare	CMP{cond} Rn, <Oprnd2>	N Z C V	Update CPSR flags on Rn - Oprnd2	
	negative	CMN{cond} Rn, <Oprnd2>	N Z C V	Update CPSR flags on Rn + Oprnd2	
Branch	Branch	B{cond} label		R15 := label	label within ±32Mb
	with link	BL{cond} label		R14 := R15-4, R15 := label	label within ±32Mb
	and exchange	BX{cond} Rm		R15 := Rm, Change to Thumb if Rm[0] is 1	
	with link and exchange (1)	BLX label		R14 := R15 - 4, R15 := label	Cannot be conditional
	with link and exchange (2)	BLX{cond}		Rm R14 := R15 - 4, R15 := Rm[31:1]	
Load	Word	LDR{cond} Rd, <a_mode2>		Rd := [address]	
	User mode privilege	LDR{cond}T Rd, <a_mode2P>			
	Branch and exchange	LDR{cond} R15, <a_mode2>		R15 := [address][31:1], Change to Thumb if [address][0] is 1	
	Byte	LDR{cond}B Rd, <a_mode2>		Rd := ZeroExtend(byte from address)	
	User mode privilege	LDR{cond}BT Rd, <a_mode2P>			
	signed	LDR{cond}SB Rd, <a_mode3>		Rd := SignExtend(byte from address)	
	Halfword	LDR{cond}H Rd, <a_mode3>		Rd := ZeroExtend(halfword from address)	
	signed	LDR{cond}SH Rd, <a_mode3>		Rd := SignExtend(halfword from address)	
Load Multiple	Pop, or Block data load	LDM{cond}<a_mode4L> Rd{!}, <reglist-pc>		Load list of registers from [Rd]	
	return (and exchange)	LDM{cond}<a_mode4L> Rd{!}, <reglist-pc>		Load registers, R15 := [address][31:1]	
	and restore CPSR	LDM{cond}<a_mode4L> Rd{!}, <reglist-pc>^		Load registers, branch and exchange	Exception modes only.
	User mode registers	LDM{cond}<a_mode4L> Rd, <reglist-pc>^		Load list of User mode registers from [Rd]	Privileged modes only.
Load Double		LDRD			
Store	Word	STR{cond} Rd, <a_mode2>		[address] := R	
	User mode privilege	STR{cond}T Rd, <a_mode2P>		[address] := Rd	
	Byte	STR{cond}B Rd, <a_mode2>		[address][7:0] := Rd[7:0]	
	User mode privilege	STR{cond}BT Rd, <a_mode2P>		[address][7:0] := Rd[7:0]	
	Halfword	STR{cond}H Rd, <a_mode3>		[address][15:0] := Rd[15:0]	
Store Multiple	Push, or Block data store	STM{cond}<a_mode4S> Rd{!}, <reglist>		Store list of registers to [Rd]	
	User mode registers	STM{cond}<a_mode4S> Rd{!}, <reglist>^		Store list of User mode registers to [Rd]	Privileged modes only.
Store Double		STRD			
Swap	Word	SWP{cond} Rd, Rm, [Rn]		temp := [Rn], [Rn] := Rm, Rd := temp	
	Byte	SWP{cond}B Rd, Rm, [Rn]		temp := ZeroExtend([Rn][7:0]), [Rn][7:0] := Rm[7:0], Rd := temp	
Coprocessor	Move to ARM reg from coproc	MRC{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>			
		MRRC{cond} p<cpnum>, <op1>, <Rd>, <Rn>, <CRm>			
	Move to coproc from ARM reg	MCR{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>			
		MCRR{cond} p<cpnum>, <op1>, <Rd>, <Rn>, <CRm>			
	Load coprocessor	LDC{cond} p<cpnum>, CRd, <a_mode5>		CRd := [address]	
	Store coprocessor	STC{cond} p<cpnum>, CRd, <a_mode5>		[address] := CRd	
Software Interrupt		SWI{cond} <Immed_24>		Software interrupt processor exception	24-bit value.
Breakpoint		BKPT <Immed_16>		Prefetch abort or enter debug state	Cannot be conditional
Pre-load		PLD <a_mode2>		Pre-load cache line	

Move	Immediate	MOV Rd, #<Immed_8>	Rd := Immed_8	8-bit immediate value.
	Lo to Lo	MOV Rd, Rm	Rd := Rm	
	Hi to Lo, Lo to Hi, Hi to Hi	MOV Rd, Rm	Rd := Rm	Not Lo to Lo
Arithmetic	Add	ADD Rd, Rn, #<Immed_3>	Rd := Rn + Immed_3	3-bit immediate value.
	Lo and Lo	ADD Rd, Rn, Rm	Rd := Rn + Rm	
	Hi to Lo, Lo to Hi, Hi to Hi	ADD Rd, Rm	Rd := Rd + Rm	Not Lo to Lo
	Immediate	ADD Rd, #<Immed_8>	Rd := Rd + Immed_8	8-bit immediate value.
	with carry	ADC Rd, Rm	Rd := Rd + Rm + C-bit	
	value to SP	ADD SP, #<Immed_7*4>	SP := SP + Immed_7 * 4	9-bit immediate value (word-aligned).
	form address from SP	ADD Rd, SP, #<Immed_8*4>	Rd := SP + Immed_8 * 4	10-bit immediate value (word-aligned).
	form address from PC	ADD Rd, PC, #<Immed_8*4>	Rd := (PC AND 0xFFFFFFFC) + Immed_8 * 4	10 bit immediate value (word-aligned).
	Subtract	SUB Rd, Rn, Rm	Rd := Rn - Rm	
	Immediate 3	SUB Rd, Rn, #<Immed_3>	Rd := Rn - Immed_3	3-bit immediate value.
	Immediate 8	SUB Rd, #<Immed_8>	Rd := Rd - Immed_8	8-bit immediate value.
	with carry	SBC Rd, Rm	Rd := Rd - Rm - NOT C-bit	
	value from SP	SUB SP, #<Immed_7*4>	SP := SP - Immed_7 * 4	9-bit immediate value (word-aligned).
	Negate	NEG Rd, Rm	Rd := - Rm	
	Multiply	MUL Rd, Rm	Rd := Rm * Rd	
	Compare	CMP Rn, Rm	update CPSR flags on Rn - Rm	Can be Lo to Lo, Lo to Hi, Hi to Lo, or Hi to Hi.
	negative	CMN Rn, Rm	update CPSR flags on Rn + Rm	
	Immediate	CMP Rn, #<Immed_8>	update CPSR flags on Rn - Immed_8	8-bit immediate value.
	No operation	NOP	Rd := Rd	flags not affected.
Logical	AND	AND Rd, Rm	Rd := Rd AND Rm	
	Exclusive OR	EOR Rd, Rm	Rd := Rd EOR Rm	
	OR	ORR Rd, Rm	Rd := Rd OR Rm	
	Bit clear	BIC Rd, Rm	Rd := Rd AND NOT Rm	
	Move NOT	MVN Rd, Rm	Rd := NOT Rm	
	Test bits	TST Rn, Rm	update CPSR flags on Rn AND Rm	
Shift / Rotate	Logical shift left	LSL Rd, Rm, #<Immed_5>	Rd := Rm << Immed_5	5-bit immediate shift. Allowed shifts 0-31.
	LSL Rd, Rs	Rd := Rd << Rs		
	Logical shift right	LSR Rd, Rm, #<Immed_5>	Rd := Rm >> Immed_5	5-bit immediate shift. Allowed shifts 1-32.
	LSR Rd, Rs	Rd := Rd >> Rs		
	Arithmetic shift right	ASR Rd, Rm, #<Immed_5>	Rd := Rm ASR Immed_5	5-bit immediate shift. Allowed shifts 1-32.
	ASR Rd, Rs	Rd := Rd ASR Rs		
	Rotate right	ROR Rd, Rs	Rd := Rd ROR Rs	
Branch	Conditional branch	$[cond] label See Table Condition Field (ARM side). AL not allowed.	R15 := label	label must be within -252 to +258 bytes
	Unconditional branch	B label	R15 := label	label must be within ±2Kb of current instruction.
	Long branch with link	BL label label must be within ±4Mb of current instruction.	R14 := R15 - 2, R15 := label	Encoded as two Thumb instructions.
	Branch and exchange	BX Rm	R15 := Rm AND 0xFFFFFFFE	Change to ARM state if Rm[0] = 0.
	Branch with link and exchange	BLX label	R14 := R15 - 2, R15 := label	Encoded as two Thumb instructions.
	Change to ARM	label must be within ±4Mb of current instruction.		
	Branch with link and exchange Change to ARM if Rm[0] = 0	BLX Rm	R14 := R15 - 2, R15 := Rm AND 0xFFFFFFFE	
Software Interrupt		<Immed_8>	Software interrupt processor exception	8-bit immediate value encoded in instruction.
Breakpoint		BKPT <Immed_8>	Prefetch abort or enter debug state	
Load	with immediate offset, word	LDR Rd, [Rn, #<Immed_5*4>]	Rd := [Rn + Immed_5 * 4]	
	halfword	LDRH Rd, [Rn, #<Immed_5*2>]	Rd := ZeroExtend([Rn + Immed_5 * 2][15:0])	Clears bits 31:16
	byte	LDRB Rd, [Rn, #<Immed_5>]	Rd := ZeroExtend([Rn + Immed_5][7:0])	Clears bits 31:8
	with register offset, word	LDR Rd, [Rn, Rm]	Rd := [Rn + Rm]	
	halfword	LDRH Rd, [Rn, Rm]	Rd := ZeroExtend([Rn + Rm][15:0])	Clears bits 31:16
	signed halfword	LDRSH Rd, [Rn, Rm]	Rd := SignExtend([Rn + Rm][15:0])	Sets bits 31:16 to bit 15
	byte	LDRB Rd, [Rn, Rm]	Rd := ZeroExtend([Rn + Rm][7:0])	Clears bits 31:8
	signed byte	LDRSB Rd, [Rn, Rm]	Rd := SignExtend([Rn + Rm][7:0])	Sets bits 31:8 to bit 7
	PC-relative	LDR Rd, [PC, #<Immed_8*4>]	Rd := [(PC AND 0xFFFFFFFC) + Immed_8 * 4]	
	SP-relative	LDR Rd, [SP, #<Immed_8*4>]	Rd := [SP + Immed_8 * 4]	
	Multiple	LDMIA Rn!, <reglist>	Loads list of registers	Always updates base register.
Store	with immediate offset, word	STR Rd, [Rn, #<Immed_5*4>]	[Rn + Immed_5 * 4] := Rd	
	halfword	STRH Rd, [Rn, #<Immed_5*2>]	[Rn + Immed_5 * 2][15:0] := Rd[15:0]	Ignores Rd[31:16]
	byte	STRB Rd, [Rn, #<Immed_5>]	[Rn + Immed_5][7:0] := Rd[7:0]	Ignores Rd[31:8]
	with register offset, word	STR Rd, [Rn, Rm]	[Rn + Rm] := Rd	
	halfword	STRH Rd, [Rn, Rm]	[Rn + Rm][15:0] := Rd[15:0]	Ignores Rd[31:16]
	byte 0	STRB Rd, [Rn, Rm]	[Rn + Rm][7:0] := Rd[7:0]	Ignores Rd[31:8]
	SP-relative, word	STR Rd, [SP, #<Immed_8*4>]	[SP + Immed_8 * 4] := Rd	
	Multiple	STMIA Rn!, <reglist>	Stores list of registers	Always updates base register.
Push	Push	PUSH <reglist>	Push registers onto stack	Full descending stack.
	Push with link	PUSH <reglist, LR>	Push LR and registers on to stack	
Pop	Pop	POP <reglist>	Pop registers from stack	
	Pop and return	POP <reglist, PC>	Pop registers, branch to address loaded to PC	
	Pop and return with exchange	POP <reglist, PC>	Pop, branch, and change to ARM state if address[0] = 0	

Intel® XScale™ Microarchitecture Assembly Language Quick Reference Card
Key to Tables

CODES

{cond}	Refer to Table Condition Field {cond}
<Operand2>	Refer to Table Operand 2
<fields>	Refer to Table PSR fields
{S}	Updates condition flags if S present
Sticky	Sticky flag, updates on overflow (no S option), read and reset using MRS and MSR
x,y	B meaning half-register [15:0], or T meaning [31:16]
<#immed_8>	A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits
<#immed_8*4>	A 10-bit constant, formed by left-shifting an 8-bit value by two bits
<a_mode2>	Refer to Table Addressing Mode 2
<a_mode2P>	Refer to Table Addressing Mode 2 (Post-Indexed only)
<a_mode3>	Refer to Table Addressing Mode 3
<a_mode4L>	Refer to Table Addressing Mode 4 (Block load or Stack pop)
<a_mode5>	Refer to Table Addressing Mode 5
<reglist>	A list of registers, enclosed in braces ({ and })
{!}	Updates base register after data transfer if ! present

CONDITION FIELD {COND}

Mnemonic	Description
EQ	Equal
NE	Not equal
CS / HS	Carry Set / Unsigned higher or same
CC / LO	Carry Clear / Unsigned lower
MI	Negative
PL	Positive or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater than or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL	Always (normally omitted)

OPERAND 2

Immediate value	#<immed_8r>	
Logical shift left immediate	Rm, LSL #<immed_5>	Allowed shifts 0-31
Logical shift right immediate	Rm, LSR #<immed_5>	Allowed shifts 1-32
Arithmetic shift right immediate	Rm, ASR #<immed_5>	Allowed shifts 1-32
Rotate right immediate	Rm, ROR #<immed_5>	Allowed shifts 1-31
Register	Rm	
Rotate right extended	Rm, RRX	
Logical shift left register	Rm, LSL Rs	
Logical shift right register	Rm, LSR Rs	
Arithmetic shift right register	Rm, ASR Rs	
Rotate right register	Rm, ROR Rs	

PSR FIELDS (USE AT LEAST ONE SUFFIX)

Suffix	Meaning	
c	Control field mask byte	PSR[7:0]
f	Flags field mask byte	PSR[31:24]
s	Status field mask byte	PSR[23:16]
x	Extension field mask byte	PSR[15:8]

ARM ADDRESSING MODES

ADDRESSING MODE 2 - WORD AND UNSIGNED BYTE DATA TRANSFER

Pre-Indexed	Immediate offset	[Rn, #+/-<immed_12>]{!}	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
	Register offset	[Rn, +/-Rm]{!}	
	Scaled register offset	[Rn, +/-Rm, LSL #<immed_5>]{!}	Allowed shifts 0-31
		[Rn, +/-Rm, LSR #<immed_5>]{!}	Allowed shifts 1-32
		[Rn, +/-Rm, ASR #<immed_5>]{!}	Allowed shifts 1-32
		[Rn, +/-Rm, ROR #<immed_5>]{!}	Allowed shifts 1-31
		[Rn, +/-Rm, RRX]{!}	
Post-Indexed	Immediate offset	[Rn], #+/-<immed_12>	
	Register offset	[Rn], +/-Rm {!}	
	Scaled register offset	[Rn], +/-Rm, LSL #<immed_5>	Allowed shifts 0-31
		[Rn], +/-Rm, LSR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ASR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ROR #<immed_5>	Allowed shifts 1-31
		[Rn], +/-Rm, RRX	

ADDRESSING MODE 2 (POST-INDEXED ONLY)

Post-Indexed	Immediate offset	[Rn], #+/-<immed_12>	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
	Register offset	[Rn], +/-Rm	
	Scaled register offset	[Rn], +/-Rm, LSL #<immed_5>	Allowed shifts 0-31
		[Rn], +/-Rm, LSR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ASR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ROR #<immed_5>	Allowed shifts 1-31
		[Rn], +/-Rm, RRX	

ADDRESSING MODE 3 - HALFWORD AND SIGNED BYTE DATA TRANSFER

Pre-Indexed	Immediate offset	[Rn, #+/-<immed_8>]{!}	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
	Register	[Rn, +/-Rm]{!}	
Post-Indexed	Immediate offset	[Rn], #+/-<immed_8>	
	Register	[Rn], +/-Rm	

ADDRESSING MODE 5 - COPROCESSOR DATA TRANSFER

Pre-Indexed	Immediate offset	[Rn, #+/-<immed_8*4>]{!}	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
Post-Indexed	Immediate offset	[Rn], #+/-<immed_8*4>	
Unindexed	No offset	[Rn], {8-bit coproc option}	

ADDRESSING MODE 4 - MULTIPLE DATA TRANSFER

Block load	
IA	Increment After
IB	Increment Before
DA	Decrement After
DB	Decrement Before
Block store	
IA	Increment After
IB	Increment Before
DA	Decrement After
DB	Decrement Before
Stack pop	
FD	Full Descending
ED	Empty Descending
FA	Full Ascending
EA	Empty Ascending
Stack push	
EA	Empty Ascending
FA	Full Ascending
ED	Empty Descending
FD	Full Descending

Παράρτημα Γ

Μετά από τις επεξηγηματικές περιγραφές που αφορούσαν τον τρόπο λειτουργίας και τις βασικές αρχές λειτουργίας των ενσωματωμένων συστημάτων και πιο συγκεκριμένα της πλατφόρμας X-board, είμαστε σε θέση να παραθέσουμε το κώδικα των δύο εφαρμογών που αναπτύχθηκαν και εκτελέστηκαν στην συγκεκριμένη πλατφόρμα.

Εφαρμογή 1η

Η πρώτη εφαρμογή αφορούσε την δημιουργία των «δικών μας» SWIs (Software Handlers). Με τον όρο «δικών μας» αναφερόμαστε στο γεγονός, πως τέτοιου είδους διαχειριστές διακοπών, έχουν υλοποιηθεί από της βιβλιοθήκες του επεξεργαστή ARM, και από την ομάδα ανάπτυξης της πλατφόρμας X-board. Έχοντας μελετήσει και κατανοήσει όλα τα παραπάνω που αναφέρονται στην εργασία, θελήσαμε να δημιουργήσουμε μόνοι μας τέτοια SWIs. Ο κώδικας της εφαρμογής που αναπτύξαμε ακολουθεί. Πριν όμως από αυτό, θα πρέπει να αναφερθεί το γεγονός πως η υλοποίηση και η εκτέλεση της εφαρμογής των SWIs, έγινε στον ARMulator, και όχι πάνω στην πλατφόρμα ανάπτυξης του X-board. Η λειτουργικότητα που προσφέρει η εφαρμογή είναι η ίδια, με την διαφορά ότι εκτελείται στον υπολογιστή, host computer, και όχι στον ίδιο τον επεξεργαστή ARM, αλλά στο πρόγραμμα προσομοίωσης της λειτουργίας του. Από την εφαρμογή η οποία εκτελείται σε user mode, παράγονται SWIs, τα οποία δημιουργούν εξαίρεση στην κανονική ροή εκτέλεσης του προγράμματος στον επεξεργαστή, τότε αναλαμβάνει την διαχείριση της εξαίρεσης ο handler, την διαχειρίζεται και μετά το πέρας της επιτυχούς διαχείρισης της, επιστρέφει στην κανονική εκτέλεση του προγράμματος και στις εντολες που έπονται της εντολης SWI.

Κώδικας Εφαρμογής 1ης (μέρος α)

Dispatcher.c

```
#include <stdlib.h>
#include <stdio.h>
#include "userSWIs.h"

//The below correspondence between processor register numbers and simple numbers is from the ATPCS-
// - ARM-THUMB Procedure Call Standard.
//We use those numbers in SWI_Dispatcher in order to show register r0 with the pointer ptr[0].

/* Processor Register Numbers
   {"r0", 0},   {"r1", 1},    {"r2", 2},    {"r3", 3},
   {"r4", 4},   {"r5", 5},    {"r6", 6},    {"r7", 7},
   {"r8", 8},   {"r9", 9},    {"r10", 10},   {"r11", 11},
   {"r12", 12},  {"r13", REG_SP},{"r14", REG_LR},{"r15", REG_PC} */

extern unsigned int Ang_Handler;
extern unsigned int installSWI_Handler(unsigned int);

//Below we define the functions that are called by the SWI_Dispatcher to handle each SWI

void Handle_printC(char c) //Handle SWI 0x100 requests
{
        SWI_WriteC(c);
}

char Handle_getC(void) //Handle SWI 0x101 requests
{
        return SWI_ReadC();
}

void Handle_printString(char * str) //Handle SWI 0x102 requests
{
        SWI_Write0(str);
}


  void Handle_restore(void) //Handle SWI 0x103 requests
  {
          Ang_Handler=installSWI_Handler(Ang_Handler);
  }

  unsigned int Handle_getClock(void) //Handle SWI 0x104 requests
  {
          return SWI_Clock();
  }
```

```
unsigned int SWI_Dispatcher(unsigned int SWI_num, unsigned int * ptr)
{
        int status = 0; //Flag in order to know whether the SWI request was handled by our SWI_Handler
                                                //or by the Angel_Handler

        switch(SWI_num) //The SWI_num shows the code of the SWI_instruction
        {
                case (0x100):
                status = 1;
                Handle_printC(ptr[0]);
                break;

                case (0x101):
                status = 1;
                ptr[0] = Handle_getC();
                break;

                case (0x102):
                status = 1;
                Handle_printString((char *)ptr[0]);
                break;

                case (0x103):
                status = 1;
                Handle_restore();
                break;

                case (0x104):
                status = 1;
                ptr[0] = Handle_getClock();
                break;

                default:
                status = 0;
                break;
        }

        return status;
}
```

Installer.c

```
#include <stdio.h>
#include <stdlib.h>  //Declares functions for number alterations, memory
dirtibution, etc.
#include "userSWIs.h"

unsigned int Ang_Handler; //The declaration of the original(old)
SWI_handler which is the Angel SWI_Handler

extern SWIhandlerEntry(void); //A reference to the entry routine

unsigned int installSWI_Handler(unsigned int new_handler) //Our installer
which is changing the handleradress which is
                                                  //stored at softvec to the
address of our new handler
{

      unsigned int offset,oldvec;
      unsigned int *SWIvec; //A pointer to the address 0x08 of the Vector
Table
      unsigned int *softvec; //The current handleraddress

      SWIvec = (unsigned *) 0x08;
      offset = (*SWIvec) & 0xfff; //We find the used offset whisch is
stored in the last 12 bits of the SWIvec

      if( offset & 0xFFFFF000 ) //We check if the offset can be
represented using 12 bits
      {                          //else we send a request to print that
the installation has failed
            SWI_Write0("Installation of handler failed");
            exit(0);
      }

      softvec = (unsigned *) (offset + 0x10); //We are finding the
address of the current SWI handler(Angel) by adding
                                                                  //the
offset we found,the 0x08 address of the SWI vector and the 8 bits that
                                                                  //the
pc is ahead due to the pipelining during the execution time
      oldvec = *softvec;                    //We are storing the
address of the old SWI handler in order to use it if
                                                                  //our
SWI handler can't handle the SWI request
      *softvec = new_handler;                        //We are setting
the handleraddress to the address of our SWI handler

      return(oldvec);

}
```

```
void install_user_SWIs(void){      //A wrapper function around
installSWI_Handler to install the handler
      Ang_Handler=installSWI_Handler((int)SWIhandlerEntry);
}
```

userSWIS.h

```
#ifndef USER_SWIS_H_INCLUDED
#define USER_SWIS_H_INCLUDED

//The following SWI_numbers will be handled by our SWI_handler

void          __swi(0x100) SWI_printC(char);
char          __swi(0x101) SWI_getC(void);
void          __swi(0x102) SWI_printString(char *);
void          __swi(0x103) SWI_restore(void);
unsigned int  __swi(0x104) SWI_getClock(void);
int           __swi(0x105) SWI_findInt(void);
void          __swi(0x106) SWI_writeInt(int);
void          __swi(0x107) SWI_CopyFile(char*);
unsigned int  __swi(0x108) SWI_loadAXF(char *);
void          __swi(0x109) SWI_run(unsigned int);
void          __swi(0x200) SWI_CopyMem(unsigned int, unsigned int);

//The following SWI_numbers will be handled by Angel.

void          __swi(0x000) SWI_WriteC(char);
void          __swi(0x002) SWI_Write0(char *);
char          __swi(0x004) SWI_ReadC(void);
unsigned int  __swi(0x61) SWI_Clock(void);

void install_user_SWIs(void);

#endif /* USER_SWIS_H_INCLUDED */
```

```
swi_handler_entry.s
AREA HandlerEntry, CODE, READONLY
        EXPORT SWIhandlerEntry ;Make it visible outside of this file
        IMPORT SWI_Dispatcher  ;Make it visible inside this file
        IMPORT Ang_Handler

SWIhandlerEntry

        SUB   sp,sp,#4          ;We are subtracking 4 from sp in order to leave room
for storing SPSR for supporting nested
                                                ;SWI calls
        STMFD       sp!, {r0-r12,lr} ;We are storing gp registers and the link
register lr in the stack
        MRS         r2,spsr       ;We are moving the SPSR into the gp r2
        STR   r2, [sp, #14*4]  ;We are storing the SPSR above the gp registers
through the r2 register
        MOV         r1,sp                   ;We are setting a pointer on the
parameters of the stack through register r1
        LDR   r0, [lr, #-4]   ;We are extracting the SWI number through the link
register and store it in the register r0
        BIC         r0,r0,#0xff000000;We are getting the code of the SWI by bit-
masking
        BL          SWI_Dispatcher   ;We are going to the body of our SWI
handler
        CMP   r0,#0             ;We are checking whether for the handling of the
SWI request we are using our SWI handler or
                                                ;the old SWI handler(Angel)
        LDR         r2,[sp,#14*4]       ;We are restoring the SPSR from the sp
        MSR         spsr_cf,r2
;http://www.arm.com/support/faqdev/1332.html
                                                ;Strictly speaking, the '_cxsf' form
should be instead of '_cf' in these cases to ensure
                                                ;compatibility with future ARM
cores which may make use of the x and s fields (unused by all current ARM
processors).

        MSREQ       cpsr_cf,r2          ;If the SWI request is to be handled from
the old SWI handler we are transferring the value of the
```

```
                                        ;register r2(SPSR) to the cpsr
        LDMFD       sp!, {r0-r12,lr} ;We are restoring all the gp registers and the lr
into their original values
        ADD         sp, sp, #4      ;We are removing the space that was used for
storing SPSR
        LDREQ       pc,Ang_Handler  ;If the SWI request is to be handled by the
old SWI handler,we are setting the pc to the
                                        ;old SWI handler(Angel)
        MOVS pc,lr          ;We are returning from the handler


        END
```

Κώδικας Εφαρμογής 1ης (μέρος β)

Dispatcher.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h> //Library for the functions memcpy,memset etc
#include <ctype.h>
#include "userSWIs.h"

                    //Below we define the max/min boundaries for an
integer.A signed integer is a 31 bit(1 bit for the signing)
#define max 2147483647   //Max integer
#define min -2147483647  //Min integer
#define EI_NIDENT 16

//The below correspondence between processor register numbers and simple
numbers is from the ATPCS-
// - ARM-THUMB Procedure Call Standard.
//We use those numbers in SWI_Dispatcher in order to show register r0
with the pointer ptr[0].

/* Processor Register Numbers
    {"r0", 0},    {"r1", 1},      {"r2", 2},      {"r3", 3},
    {"r4", 4},    {"r5", 5},      {"r6", 6},      {"r7", 7},
    {"r8", 8},    {"r9", 9},      {"r10", 10},    {"r11", 11},
    {"r12", 12},  {"r13", REG_SP},{"r14", REG_LR},{"r15", REG_PC} */


typedef struct //Format of the ELF header structure
{
      unsigned char e_ident[EI_NIDENT]; //File info(object file or not)
      unsigned short e_type;            //Type of
file(relocatable,executable,etc)
      unsigned short e_machine;         //Target processor(Intel
x86,ARM,SPARC, etc)
      unsigned int e_version;           //Version # (to allow for future
versions of ELF)
      unsigned int e_entry;             //Program entry point(0 if no
entry point)
      unsigned int e_phoff;             //offset of program header(in
bytes)
      unsigned int e_shoff;             //offset of section header table
```

```
	unsigned int e_flags;			//Processor-specific flags
	unsigned short e_ehsize;		//ELF header's size
	unsigned short e_phentsize;		//Entry size in pgm header tbl
	unsigned short e_phnum;			//# of entries in pgm header
	unsigned short e_shentsize;		//Entry size in sec header tbl
	unsigned short e_shnum;			//# of entries in sec header tbl
	unsigned short e_shstrndx;		//sec header tbl index of str tbl
} Elf32_Ehdr;

typedef struct //Program header entry for each segment
{
	unsigned int p_type;			//Type of segment-loadable,
dll,...
	unsigned int p_offset;			//offset in bytes from the start
of file
	unsigned int p_vaddr;			//Virtual address in memory of
segment
	unsigned int p_paddr;			//Physical address in memory of
segment
	unsigned int p_filesz;			//Number of bytes in the file of
the segment
	unsigned int p_memsz;			//Number of bytes in memory of the
process
	unsigned int p_flags;			//Indicates whether segment is
executable
	unsigned int p_align;			//Alignment information
} Elf32_Phdr;



extern unsigned int Ang_Handler;				//Address of the
Angel handler(old handler)
extern unsigned int installSWI_Handler(unsigned int); //We create a
connection to the installSWI_Handler
extern SWIhandlerEntry;					//Assembly entry
routine

//Below we define the functions that are called by the SWI_Dispatcher to
handle each SWI

void Handle_printC(char c) //Handle SWI 0x100 requests
{
	SWI_WriteC(c);
}

char Handle_getC(void) //Handle SWI 0x101 requests
{
	return SWI_ReadC();
}

void Handle_printString(char * str) //Handle SWI 0x102 requests
{
	SWI_Write0(str);
}

void Handle_restore(void) //Handle SWI 0x103 requests
{
```

```
	Ang_Handler=installSWI_Handler(Ang_Handler);
}

unsigned int Handle_getClock(void) //Handle SWI 0x104 requests
{
	return SWI_Clock();
}

int Handle_findInt(void) //Handle SWI 0x105 requests
{
	char tmp[20],ar[20];
	int counter=0,ct=0,i = 0,result=0;
	tmp[counter] = SWI_ReadC();
//Read the first character from the user

	while( tmp[counter] != (char) 10)  //While the incoming character
is different from ENTER(the ASCII encoding for it is 10)
	{
		counter = counter + 1;
		if(isspace(tmp[counter-1])) { //We don't take into account
the spaces(if the user gives any)
			tmp[counter-1] = NULL;
			counter = counter - 1;
			tmp[counter] = SWI_ReadC();
			continue;
		}
		if(counter == 20) {		//We accept maximum of 20
characters from the user
			SWI_printString("You overpassed string boundaries");
			break;
		}
		tmp[counter] = SWI_ReadC();  //We read the next character
from the user
	}
	if(tmp[counter] == (char) 10) { //If the incoming counter is ENTER,
stop
		tmp[counter] = NULL;
		counter = counter - 1;
	}
	while(ct<=counter)
	{
		if(tmp[ct]>(char)47 && tmp[ct]<(char)58) { //We are locating
the first integer('0' is 48 in ASCII encoding and '9' is 57)
			if(ct>0 && tmp[ct-1] == (char) 45) {   //We are
checking if the previous character is '-'(signed integer)
				ct = ct - 1;
			}
			for(i=0;i<=(counter - ct);i++) {	//We are copying
the next section(from the integer or the '-') of the input string
				ar[i] = tmp[i+ct];		//at a table
			}
			break;
		}
		ct = ct + 1;
	}
	result = atoi(ar);				//We are passing this
table to atoi
```

```
	if(result == max || result == (min-1)) {	//We are checking if
the integer exceeds the boundaries
		SWI_printString("You exceeded the max/min int bounds.\n");
	}

	return atoi(ar);
}

void Handle_writeInt(int integ1) //handle SWI 0x106 requests
{
	int tmp_integ =0,counter=0,i=0,j=0;
	int number[10];
	for(i=0;i<10;i++) {
	number[i]=0;
	}

	if(abs(integ1)!=integ1) {			//We are checking if the
integer is positive or negative
		SWI_WriteC('-');
		integ1 = abs(integ1);			//If it is negative we
are taking the absolute value
	}

	while(integ1!=0) {
		tmp_integ=integ1%10;			//We are finding the
least significant digit of our integer
		if(tmp_integ!=0) {
			integ1=(integ1-tmp_integ)/10;	//We are shifting left
in order to find the next digit
			number[counter]=tmp_integ;
		}
		else {
			integ1=(integ1)/10;
			number[counter]=0;
		}
		counter=counter+1;
	}

	if(counter==0) {				//If the user gave the
number 0
		SWI_WriteC('0');
	}
	for(j=counter-1;j>=0;j--) {		//We are printing the number
the user gave(we are adding the 48-ASCII encoding for '0'-
		SWI_WriteC((char)(number[j]+48)); //in order to take the
right character
	}
	SWI_WriteC('\n');
}

void Handle_CopyFile(char * file_name) //Handle SWI 0x107 requests
{
	int handler_open;
	int handler_close;
	int file_length = 0;
	int bytes_not_read=0;
	char * char_buffer;
```

```
	if((handler_open = SWI_Open(file_name, 1)) == 0)		//We are
openning the reading file(input)
	{
		SWI_printString("Reading-File open, FAILED! \n");
		return;
	}
	else
	{
		if((handler_close = SWI_Open("out.txt", 5)) == 0)	//We are
openning the write file(output)
		{
			SWI_printString("Write-File open, FAILED! \n");
			return;
		}

		file_length = SWI_Flen(handler_open);			//We are
finding the length of the input file

		if(file_length == -1)
		{
			SWI_printString("Reading File-Length, FAILED! \n");
		}
		else
		{
			char_buffer = (char *)(malloc (file_length));
//Allocate temporary memory to copy the contents of the input file
			bytes_not_read=SWI_Read(handler_open, char_buffer,
file_length);  //The bytes that were not read and copied to the

//memory from the input file
			SWI_Write(handler_close, char_buffer, file_length-
bytes_not_read);//We are writting the contents into the output file
		}


		if(SWI_Close(handler_close) != 0)			//We are
closing the output file
		{
			SWI_printString("Closing Write-File, FAILED! \n");
		}

		if(SWI_Close(handler_open) != 0)			//We are
closing the input file
		{
			SWI_printString("Closing Read-File, FAILED! \n");
		}
	}

}

unsigned int Handle_loadAXF(char * AXF_Name) //Handle 0x108 requests
{
	int AXF_handler;
	int AXF_length;
	int i = 0;
	char * AXF_buffer;
```

```
	char * memory_buffer;
	unsigned int entry_address;

	Elf32_Ehdr ELF_header;
	Elf32_Phdr ELF_PrHeadTable;

	if((AXF_handler = SWI_Open(AXF_Name, 1)) == 0)            //We are
openning the AXF file
	{
		SWI_printString("AXF File Open, FAILED! \n");
		return 0;
	}
	else
	{
			if((AXF_length = SWI_Flen(AXF_handler)) == -1)   //We
are finding the length of the AXF file
			{
				SWI_printString("AXF File-Length, FAILED!\n");
				return 0;
			}

			AXF_buffer = (char *)malloc(AXF_length);
//Allocate temporary memory for holding the contents of the AXF file
			SWI_Read(AXF_handler, AXF_buffer, AXF_length);  //We
are loading the contents of the AXF file

			memcpy(&ELF_header,AXF_buffer,52);              //We
are copying the first 52 bytes(header size) int the
                                                            //ELF
header structure
			//Below we are checking if the file is an ELF object
file
			if((int)ELF_header.e_ident[0]!=0x7f ||
ELF_header.e_ident[1] != 'E' || ELF_header.e_ident[2] != 'L' ||
ELF_header.e_ident[3] != 'F')
			{
				SWI_printString("This is NOT an ELF Object File!
\n");
				return 0;
			}

			if((int)ELF_header.e_ident[5] != 2)             //Here
we are checking the code encoding of all data(whether it is
			{                                               //Big
Endian or Little Endian)
				SWI_printString("Wrong Format! (It is NOT a Big
Endian!) \n");
				return 0;
			}

			if((int)ELF_header.e_type != 2)                 //We are
checking the identification of the object file type
			{
//(Relocatable,Executable or Shared object file)
				SWI_printString("NOT an executable file! \n");
				return 0;
			}
```

```
                if((int)ELF_header.e_machine != 40)          //We are
checking the used architecture
                {
                      SWI_printString("NOT ARM target processor! \n");
                      return 0;
                }

                entry_address = ELF_header.e_entry;          //We are
storing the virtual address to which the system first transfers control

                if(entry_address == 0)
                {
                      SWI_printString("No Available Entry Point! \n");
                      return 0;
                }

                if(entry_address < 0x12000)                  //We are
checking if the loading address is at a safe location
                {
                      SWI_printString("ERROR in Loading Address...!
\n");     //else we are printing an error message
                      return 0;
                }

                if(ELF_header.e_phoff == 0)                  //We are
checking the program header table's file offset
                {
                      SWI_printString("No Available Program Header
table! \n");
                      return 0;
                }

                AXF_buffer = AXF_buffer + ELF_header.e_phoff;    //We
are moving to the program header table

                for (i = 0; i < ELF_header.e_phnum; i++)         //We
may have lot of entries
                {
                      memcpy(&ELF_PrHeadTable, AXF_buffer,
ELF_header.e_phentsize);   //We are copying the program header table into

//the ELF program header structure
                      if(ELF_PrHeadTable.p_type != 1)
//We are checking the type of the segment(loadable)
                      {
                            SWI_printString("NOT Loadable Segments!
\n");
                            return 0;
                      }

                      memory_buffer = (char *)ELF_PrHeadTable.p_paddr;
//We are setting a pointer to the physical address of the memory

                       //where the segment will be written
```

```
                        memset
(memory_buffer,0,ELF_PrHeadTable.p_memsz);//We initialize with zero all
the memory blocks that will be used

                        if(SWI_Seek(AXF_handler,
ELF_PrHeadTable.p_offset) !=0 ) //We are seeking the beginning of the
segment
                        {
                              SWI_printString("Seeking File, FAILED!
\n");
                              return 0;
                        }

                        SWI_Read(AXF_handler, memory_buffer,
ELF_PrHeadTable.p_filesz); //We are loading the segment into the memory
                       //Here while we load the segment into the memory we
see that (p_memsz-p_filesz) bytes are equal with 0
                        AXF_buffer=AXF_buffer+ELF_header.e_phentsize;
//We are moving to the next segment
                  }

      }
      if(SWI_Close(AXF_handler) != 0)
//We are closing the AXF file
            {
                  SWI_printString("Closing AXF File, FAILED! \n");
            }
      free(AXF_buffer);
//We are clearing the temporary memory space used
      return (unsigned int) entry_address;
//for the AXF file
}

unsigned int Handle_Run(unsigned int saddress) //Handle 0x109 requests
{
      if(saddress == 0)                                          //We are
checking whether the file(code) pointed by the saddress
      {                                                          //was loaded
or not
            SWI_printString("File didn't RUN! \n");
            exit(0);
      }
      return saddress;
}

void Handle_CopyMem(unsigned int address, unsigned int data) //Handle
0x200 requests
{
      unsigned int * temp_data;

      if( (address%4) !=0 )                                    //We are
checking if the address is word aligned
      {
            SWI_printString("MISALIGNED! \n");
            return;
      }
      else
```

```
	{
		temp_data = (unsigned int * ) address;		//Data
contents of the address
		SWI_printString("Data in that memory address:");
		SWI_writeInt((int )* temp_data);		//We are
printing the data contents
		*temp_data = data;				//We are
storing the new data contents in the address
		SWI_WriteC('\n');
	}
}



unsigned int SWI_Dispatcher(unsigned int SWI_num, unsigned int * ptr)
{
	int status = 0; //Flag in order to know whether the SWI request was
handled by our SWI_Handler
				//or by the Angel_Handler

	switch(SWI_num) //The SWI_num shows the code of the SWI_instruction
	{
		case (0x100):
		status = 1;
		Handle_printC(ptr[0]);
		break;

		case (0x101):
		status = 1;
		ptr[0] = Handle_getC();
		break;

		case (0x102):
		status = 1;
		Handle_printString((char *)ptr[0]);
		break;

		case (0x103):
		status = 1;
		Handle_restore();
		break;

		case (0x104):
		status = 1;
		ptr[0] = Handle_getClock();
		break;

		case (0x105):
		status = 1;
		ptr[0] = (unsigned int)Handle_findInt();
		break;

		case (0x106):
		status = 1;
		Handle_writeInt((int)ptr[0]);
		break;
```

```
            case (0x107):
            status = 1;
            Handle_CopyFile((char *)ptr[0]);
            break;

            case (0x108):
            status = 1;
            ptr[0] = Handle_loadAXF((char *)ptr[0]);
            break;

            case (0x109):
            status = 1;
            ptr[13] = Handle_Run(ptr[0]);                    //We are
storing the entry address of the code in the user link register
            break;                                           //which is
in the stack

            case (0x200):
            status = 1;
            Handle_CopyMem((unsigned int)ptr[0], (unsigned int)ptr[1]);
            break;

            default:
            status = 0;
            break;
      }

      return status;
}
```

Εφαρμογή 2η

Κώδικας Εφαρμογής 2ης

BufStatus.c

```
#include "BufStatus.h"
#include "uhal.h"

//Define output and input buffers
BufOutCharacteristics OutputBuffer;
BufInCharacteristics InputBuffer[MaxNumberOfInputBuffers];


//Function for intialization of the output buffer
void InitializingOutputBuffer(void) {
      OutputBuffer.head=0;
      OutputBuffer.tail=0;

      OutputBuffer.FreeSpace=OutputBufferLength-1;
      OutputBuffer.UsedSpace=0;

}
```

```
//Function for intialization of a specific input buffer
void InitializingInputBuffer(unsigned int InBufNumber) {
      InputBuffer[InBufNumber].head=0;
      InputBuffer[InBufNumber].tail=0;

      InputBuffer[InBufNumber].FreeSpace=InputBufferLength-1;
      InputBuffer[InBufNumber].UsedSpace=0;
}


void InitializationBothBuffers(void) {
      unsigned int i;

      for(i=0;i<MaxNumberOfInputBuffers;i++){
            InitializingInputBuffer(i);
      }

      InitializingOutputBuffer();
}

unsigned int getOutputBufferFreeSpace(void) {
      return(OutputBuffer.FreeSpace);
}

unsigned int getOutputBufferUsedSpace(void) {
      return(OutputBuffer.UsedSpace);
}

unsigned int getInputBufferFreeSpace(unsigned int InBufNumber) {
      return(InputBuffer[InBufNumber].FreeSpace);

}


// Function that returns used space in input buffer
unsigned int getInputBufferUsedSpace(unsigned int InBufNumber) {
      return(InputBuffer[InBufNumber].UsedSpace);
}

// Function that puts a character in output buffer
unsigned int PutCharacterInOutputBuffer(char Value) {
      int BufStatus=0;

      //Here first of all we are checking if there is free space for
writting a character in the output buffer
      if(OutputBuffer.FreeSpace!=0){
            //We are putting the character in the head of the buffer

            OutputBuffer.Data[OutputBuffer.tail]=Value;
            //We are shifting the head pointer 1 place in order to show
to the next empty entry
            OutputBuffer.tail=(OutputBuffer.tail+1)%OutputBufferLength;

            //We are setting the new values for the UsedSpace and
FreeSpace variables
```

```
            OutputBuffer.FreeSpace--;
            OutputBuffer.UsedSpace++;


            //We are returning 0(Successful placement)
            return(BufStatus);

      }
      else{
            BufStatus=1;
            //We are returning 1(Unsuccessful placement)
            return(BufStatus);

      }
}


// Function that gets a character from the output buffer
unsigned int PopCharacterFromOutputBuffer(char *CharValue) {
      int BufStatus=0;

      //Here first of all we are checking if there is a character for
poping from the output buffer
      if(OutputBuffer.UsedSpace!=0){
            //We are getting the character shown by the tail

            *CharValue=OutputBuffer.Data[OutputBuffer.head];
            //We are shifting the tail pointer 1 place in order to show
to the next character
            OutputBuffer.head=(OutputBuffer.head+1)%OutputBufferLength;

            //We are setting the new values for the UsedSpace and
FreeSpace variables

            OutputBuffer.FreeSpace++;
            OutputBuffer.UsedSpace--;


            //We are returning 0(Successful poping)
            return(BufStatus);

      }
      else{
            BufStatus=1;
            //We are returning 1(Unsuccessful poping)
            return(BufStatus);

      }
}



unsigned int PutCharacterInInputBuffer(unsigned int InBufNumber,char
Value) {
      int BufStatus=0;
```

```
      if(InputBuffer[InBufNumber].FreeSpace!=0){

      InputBuffer[InBufNumber].Data[InputBuffer[InBufNumber].tail]=Value;

      InputBuffer[InBufNumber].tail=(InputBuffer[InBufNumber].tail+1)%Inp
utBufferLength;

            InputBuffer[InBufNumber].FreeSpace--;
            InputBuffer[InBufNumber].UsedSpace++;
            return(BufStatus);
      }
      else{
            BufStatus=1;
            return(BufStatus);
      }
}


unsigned int PopCharacterFromInputBuffer(unsigned int InBufNumber,char
*CharValue) {
      int BufStatus=0;

      if(InputBuffer[InBufNumber].UsedSpace != 0){
            *CharValue=
InputBuffer[InBufNumber].Data[InputBuffer[InBufNumber].head];

      InputBuffer[InBufNumber].head=(InputBuffer[InBufNumber].head+1)%Inp
utBufferLength;

            InputBuffer[InBufNumber].FreeSpace++;
            InputBuffer[InBufNumber].UsedSpace--;
            return(BufStatus);
      }
      else{
            BufStatus=1;
            return(BufStatus);

      }
}



Userio.c

#include "serial.h"              //We load the libraries of which the
functions we will use
#include "uhal.h"
#include "trongame.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "my_irq.h"
#include "timer.h"
#include "screen.h"
#include "BufStatus.h"
#include "userio.h"
```

```
#include <ctype.h>
#include "brh.h"

extern int InTimer;

//Global variable declaration
timer_p Timing;                    //Structure with the tenths,secs and
mins
volatile int TimerTenths=0;
int ticks;
int GlobalTimer;

unsigned int InputMode=0;

extern ST16C550Reg *Serial1;

//-------------------------------------------------------------------------
----------------------
extern void uHALr_InitInterrupts(void);
extern void uHALr_InitTimers(void);
extern int  uHALr_RequestTimer(PrHandler,const unsigned char *);
extern int  uHALr_SetTimerInterval(unsigned int,unsigned int);
extern void uHALr_InstallTimer(unsigned int);
extern int  uHALr_FreeTimer(unsigned int);
extern void uHALir_InitSerial(unsigned int,unsigned int );
extern void uHALr_DisableInterrupt(unsigned int);
extern void uHALr_EnableInterrupt(unsigned int);
extern int uHALr_FreeInterrupt(unsigned int);


extern int NumPlayers;
extern int NumComputer;
//-------------------------------------------------------------------------
------------------------

//Function that puts a string in the output buffer(with busy waiting if
the buffer is not empty enough)
void PrintStr(char *s) {
      int i=0;
      unsigned int tmp;
      int Length;

      BRHDisableInterrupt(1);    //Disable TimerInterrupts

      Length=strlen(s);          //We are finding the length of the string

      //Wait until the transmitter holding register is empty
      while( (Serial1->ier & BIT1) != 0 );

      //If we can not put the string in the buffer...
      if( Length > getOutputBufferFreeSpace()) {
            Serial1->ier |= BIT1;    //Enable the THR interrupt
            while( (Serial1->ier & BIT1) != 0 );  //Wait until the buffer
is empty
      }

      //Put the string in the buffer
```

```
	while(s[i]!='\0') {
		tmp=PutCharacterInOutputBuffer(s[i]);
		i++;

	}

	BRHEnableInterrupt(1);		//Enable Timer Interrupts

	Serial1->ier |= BIT1;		//Enable THR interrupts

}

//Function that puts a string in the output buffer(without busy waiting
if the buffer is not empty enough)
void PrintStr_no_irq(char *s) {
	int i=0;
	char ch;

	int Length=strlen(s);

	//If we can not put the string in the buffer
	while( Length >= getOutputBufferFreeSpace() )  {
		PopCharacterFromOutputBuffer( &ch );  //Empty the buffer
until we can
		Serial1->rxtx = ch;
	}

	//Put the string in the buffer
	while(s[i]!='\0')  {
		PutCharacterInOutputBuffer(s[i]);
		i++;
	}
}

//Function that calculates the number of ticks
//It is used as the timer interrupt handler
void IncreaseTimer() {
	char String[32];

	TimerTenths++;

	//We are finding the tenths,secs and mins
	ticks=TimerTenths;
	Timing.tenths=ticks%10;

	ticks=ticks/10;
	Timing.secs=ticks%60;

	Timing.mins=ticks/60;

	// We are savinf the cureent condition,going the cursor at
	// the desirable position in terminal, putting the time in the
proper format,
	// and finally restore previous condition
    savecursor_no_irq();
	sprintf(String,"%02d:%02d.%d",Timing.mins,Timing.secs
,Timing.tenths);
```

```
	PrintStrAtLoc_no_irq( String, 70, 24 );
	restorecursor_no_irq();
}

// This function returns the char at the head of the input buffer(with
busy waiting)
char GetChar(int id) {
	char CharValue;

	//If the poping of the character is not successful do busy waiting
	while( PopCharacterFromInputBuffer(id,&CharValue) ) {
	}

	return CharValue;
}

// This function returns the char at the head of the input buffer(without
busy waiting)
char GetCharNB(int id) {
	unsigned int tmp;
	char CharValue;

	//Getting the character from the buffer
	tmp=PopCharacterFromInputBuffer(id,&CharValue);


	//Check to see if there was a character to the buffer
	if(tmp==0) {
		return CharValue;  //If there is return the character...
	}
	else {
		return 0;          //Else return 0
	}
}

//Function for initializing the timer
void init_timer(void) {
	int tmp;
	uHALr_InitInterrupts();              //We initialize the µHAL
internal interrupt structures
	uHALr_InitTimers();                  //We reset all the timers to
a known state
	TimerTenths=0;
	uHALr_printf("InitTimer!\n");
	//We install the handler for the system timer and stop the timer
	GlobalTimer=uHALr_RequestSystemTimer(IncreaseTimer,(const unsigned
char*)"test");
	if(GlobalTimer<=0) {
		uHALr_printf("Timer IRQ is already assigned!\n");
	}

	//We set the timer interval in microseconds
	tmp=uHALr_SetTimerInterval((unsigned int)GlobalTimer,100000);
	if(tmp<0) {
		uHALr_printf("Timer not found!\n");
	}
```

```
	//We are starting the timer
	uHALr_InstallTimer((unsigned int)GlobalTimer);
	BRHEnableInterrupt(PLAT_TIMERAINT);
	uHALr_printf("InitTimer out!\n");
}

//Function that implements the serial interrupt handler
void SerialHandling() {
	char CharValue;
	unsigned int tmp;
	unsigned int tmp1;

	if( Serial1->lsr & BIT0 ) {  //If we have a receive trasmit
interrupt
		CharValue= Serial1->rxtx;

		if(InputMode==0)
			PutCharacterInInputBuffer(0,CharValue);
		else
		{
			CharValue=tolower(CharValue); //in order to play even
with CAPS LOCK
			switch (CharValue) {
				case '2':
					PutCharacterInInputBuffer(1,CharValue);
					break;
				case '4':
					PutCharacterInInputBuffer(1,CharValue);
					break;
				case '6':
					PutCharacterInInputBuffer(1,CharValue);
					break;
				case '8':
					PutCharacterInInputBuffer(1,CharValue);
					break;
				case 'x':
					PutCharacterInInputBuffer(2,'2');
					break;
				case 'a':
					PutCharacterInInputBuffer(2,'4');
					break;
				case 'd':
					PutCharacterInInputBuffer(2,'6');
					break;
				case 'w':
					PutCharacterInInputBuffer(2,'8');
					break;
				case 'm':
					PutCharacterInInputBuffer(3,'2');
					break;
				case 'j':
					PutCharacterInInputBuffer(3,'4');
					break;
				case 'k':
					PutCharacterInInputBuffer(3,'6');
					break;
				case 'i':
```

```
                        PutCharacterInInputBuffer(3,'8');
                        break;
                  case 'v':
                        PutCharacterInInputBuffer(4,'2');
                        break;
                  case 'f':
                        PutCharacterInInputBuffer(4,'4');
                        break;
                  case 'g':
                        PutCharacterInInputBuffer(4,'6');
                        break;
                  case 't':
                        PutCharacterInInputBuffer(4,'8');
                        break;
            }
      }

      return;
}

//if we have a trasmitter empty interrupt
if(TX_EMPTY(UART2_BASE)) {
      if(getOutputBufferFreeSpace()!=0) {  //if there are
characters in the buffer...
            tmp= PopCharacterFromOutputBuffer(&CharValue);  //take
characters from outbuffer
            PUT_CHAR(OS_COMPORT,CharValue);
            if(getOutputBufferUsedSpace()==0) {  //if outpout
buffer empty
                  Serial1->ier &= ~ BIT1;              //Disable
THR interrupts
            }
      }
}
}

//Function that initialize the serial
void init_ser(void) {
      unsigned int tmp;

      uHALr_InitInterrupts();

      BRHDisableInterrupt(PLAT_UARTINT2);  //Disable any serial
interrupts

      //We are intializing the interrupts and the serial
      tmp=uHALr_RequestInterrupt((unsigned
int)PLAT_UARTINT2,(PrHandler)SerialHandling,(const unsigned
char*)"SerialHandling" );
      if (tmp==0){
            uHALr_printf("RequestInterrupt!!! \n");
      }

      uHALir_InitSerial(OS_COMPORT, ARM_BAUD_115200 );

      BRHEnableInterrupt(PLAT_UARTINT2);  //Enable serial interrupts
```

```
	Serial1->ier |= 1;                    //Enable THR interrupts

	return;
}
//Function that uniintialize the serial
void uninit_ser(void) {
	uHALir_InitSerial(OS_COMPORT, ARM_BAUD_115200 );   //reset port
	uHALr_DisableInterrupt(PLAT_UARTINT2);
	uHALr_FreeInterrupt(PLAT_UARTINT2);
}

//Function which is called at the end of the program for unintializing
the timer
void uninit_timer(void) {
	int tmp;

	//We disable the timer,free the interrupt and update the internal
structure
	tmp=uHALr_FreeTimer((unsigned int)GlobalTimer);
	if(tmp<0) {
		uHALr_printf("The timer is unknown!\n");
	}
}

// Function that sets which buffers received characters are placed
void SetInputMode(int mode) {
	InputMode=mode;
	return;
}

// Function that waits for "ticks" tenth of a second and returns
void Sleep(int ticks) {
	int finalTicks;

	finalTicks=TimerTenths+ticks;

	while(TimerTenths!=finalTicks){                    // Wait until
ticks time has elapsed
		//nothing
	}
	return;
}
```