Α.Τ.Ε.Ι. ΗΡΑΚΛΕΙΟΥ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΠΟΛΥΜΕΣΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΤΙΤΛΟΣ :

« ΕΚΤΙΜΗΣΗ ΕΠΙΔΟΣΕΩΝ ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΝΘΕΣΙΜΩΝ ΕΠΕΞΕΡΓΑΣΤΩΝ SOFT-PROCESSORS ΜΕ ΕΦΑΡΜΟΓΕΣ BENCHMARKS »

ΦΟΙΤΗΤΕΣ :

ΚΩΝΣΤΑΝΤΙΝΟΣ ΜΑΥΡΟΚΩΣΤΑΣ ΑΜ 912

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΕΝΗΣ ΑΜ 941

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ : *ΓΕΩΡΓΙΟΣ ΚΟΡΝΑΡΟΣ*

- ΗΡΑΚΛΕΙΟ 2009 -

ΕΥΧΑΡΙΣΤΙΕΣ

Στην πτυχιακή αυτή εργασία, βοήθησε σημαντικά με την καθοδήγηση του και τον τρόπο του ο καθηγητής μας Γιώργος Κορνάρος. Με τη συμβολή και τη συνεχόμενη του προσπάθεια δημιουργήθηκε η συγκεκριμένη εργασία.

Τον ευχαριστούμε θερμά.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ ΤΡΙΤΟ

ΚΕΦΑΛΑΙΟ ΤΕΤΑΡΤΟ

ΠΑΡΑΡΤΗΜΑ Α

ΠΑΡΑΡΤΗΜΑ Β

ΠΑΡΑΡΤΗΜΑ Γ

ΚΕΦΑΛΑΙΟ ΠΡΩΤΟ

ΕΙΣΑΓΩΓΗ

[1.1] Ενσωματωμένα συστήματα (Embedded Systems)

Ένα ενσωματωμένο σύστημα είναι ένα υπολογιστικό σύστημα ειδικού σκοπού, σχεδιασμένο έτσι ώστε να εκτελεί μια ή και περισσότερες συναρτήσεις, συνήθως σε σταθερές πραγματικού χρόνου. Είναι συνήθως ενσωματωμένο σαν ένα μέρος μιας ολοκληρωμένης συσκευής, περιλαμβάνοντας hardware καθώς και μηχανικά μέρη. Σε αντίθεση, ένα υπολογιστικό σύστημα γενικού σκοπού, όπως ένας προσωπικός υπολογιστής, ανάλογα με τον προγραμματισμό που του έχει γίνει, μπορεί να κάνει πολλά διαφορετικά καθήκοντα. Τα ενσωματωμένα συστήματα ελέγχουν πολλές από τις κοινές καθημερινές συσκευές που χρησιμοποιούμε σήμερα. Από τη στιγμή που ένα σύστημα είναι προσανατολισμένο σε συγκεκριμένα καθήκοντα, οι μηχανικοί σχεδίασης μπορούν να το βελτιστοποιήσουν, μειώνοντας το μέγεθος και το κόστος του.

1.1.1 Εφαρμογή και χρήση των ενσωματωμένων συστημάτων

- Μηχανήματα αναλήψεως (ATMs)
- Ναυσιπλοΐα, such as inertial guidance systems, hardware/software συστημάτων ελέγχου τάσεων, συστήματα αεροναυπηγικής, πυραύλους, GPS
- ελεγκτές κινητήρων και ABS αυτοκινήτων
- Προϊόντα αυτοματισμού σπιτιού, όπως θερμοστάτες, κλιματιστικά, συστήματα συναγερμού
- αριθμομηχανές
- οικιακές συσκευές, όπως φούρνοι μικροκυμάτων, συσκευές DVD , μαγνητόφωνα
- ιατρικός εξοπλισμός
- παιχνιδομηχανές
- περιφερειακά υπολογιστών, όπως εκτυπωτές και routers
- Σε βιομηχανικά συστήματα, κυρίως σε ελεγκτές για χειρισμού των μηχανημάτων.

[1.2] Ενσωματωμένοι Επεξεργαστές

Το hardware των ενσωματωμένων συστημάτων είναι συνήθως διαφορετικό από το hardware ενός κλασικού συστήματος. Τους ενσωματωμένους επεξεργαστές μπορούμε να τους χωρίσουμε σε 2 μεγάλες κατηγορίες : Στους μικροεπεξεργαστές (μP) και στους μικροελεγχτές (μC), οι οποίοι έχουν πολλά περισσότερα περιφερειακά στο chip, μειώνοντας έτσι το κόστος και το μέγεθος.

Οι περισσότερες αρχιτεκτονικές είναι μεγάλες σε αριθμό διαφορετικών παραλλαγών και μορφών, πολλές από τις οποίες κατασκευάζονται επίσης από διάφορες διαφορετικές εταιρίες. Κάποιες από αυτές τις αρχιτεκτονικές είναι οι παρακάτω :

65816, 65C02, 68HC08, 68HC11, 68k, 8051, ARM, AVR, AVR32, Blackfin, C167, Coldfire, COP8, eZ8, eZ80, FR-V, H8, HT48, M16C, M32C, MIPS, MSP430, PIC, PowerPC, R8C, SHARC, ST6, SuperH, TLCS-47, TLCS-870, TLCS-900, Tricore, V850, x86, XE8000, Z80, AsAP και άλλες πολλές.

1.2.1 Η λύση της ASIC και της FPGA

Μια κοινή διαμόρφωση για τα μεγάλου όγκου ενσωματωμένα συστήματα είναι το σύστημα σε ένα τσιπ (SoC) που περιέχει ένα πλήρες σύστημα που αποτελείται από πολλαπλούς επεξεργαστές, πολυπλέκτες, caches και διεπαφές σε ένα μόνο chip.

Τα συστήματα σε ένα τσιπ μπορούν να εφαρμοστούν ως ASIC (application-specific integrated circuit) ή χρησιμοποιώντας μια FPGA (field-programmable gate array).

[1.3] Τι είναι μια FPGA

Μια FPGA είναι μια συσκευή ημιαγωγών και αποτελείται από πολλά blocks προγραμματιζόμενης λογικής, καθώς και από τις διασυνδέσεις μεταξύ τους. Αυτά τα λογικά blocks μπορούν να προγραμματιστούν για να εκτελέσουν τις λογικές πράξεις AND και XOR καθώς και ποιο περίπλοκους συνδυασμούς των πράξεων αυτών. Στις περισσότερες FPGAs κάποια από αυτά τα blocks υποκαθιστούν στοιχεία μνήμης, δηλαδή απλά flip – flops ή ποιο σύνθετα blocks μνήμης. Δεν προσφέρονται για τον σχεδιασμό πολύ πολύπλοκων κυκλωμάτων, αλλά σε αντιπαράθεση με αυτό μας προσφέρουν το πλεονέκτημα του επαναπρογραμματισμού από την αρχή. Υπάρχουν δύο τρόποι για να τις προγραμματίσουμε: είτε με τη βοήθεια κυκλωματικών σχεδιαγραμμάτων, είτε μέσω HDL που είναι μια περιγραφική γλώσσα προγραμματισμού για FPGAs και συνίσταται για μεγαλύτερα και πολυπλοκότερα projects. Μετατρέποντας είτε τα σχεδιαγράμματα είτε τον κώδικα της HDL σε εκτελέσιμα κυκλώματα χρειαζόμαστε είτε ένα σειριακό interface τύπου JTAG, Είτε τη βοήθεια εξωτερικής μνήμης τύπου EEPROM.

[1.4] Διαφορά μεταξύ των FPGAs & CPLDs

Οι κυριότερες διαφορές ανάμεσα στα CPLDs και τις FPGAs είναι σε θέματα αρχιτεκτονικής. Ένα CPLD έχει περιορισμένη δομή διότι αποτελείται από έναν ή περισσότερους λογικούς πίνακες τροφοδοτώντας έτσι ένα σχετικά μικρό αριθμό από χρονισμένους καταχωρητές. Ως αποτέλεσμα, έχουμε μείωση της ευελιξίας στο σύστημα, μπορούμε όμως πιο εύκολα να προβλέψουμε τις χρονικές καθυστερήσεις. Η αρχιτεκτονική των FPGA από την άλλη βασίζονται στις διασυνδέσεις. Αυτό τις κάνει πολύ πιο ευέλικτες (μεγαλύτερη γκάμα λογικών συνδυασμών που μπορούν να προκύψουν) καθώς και πιο περίπλοκες στο σχεδιασμό τους.

Άλλη μία ειδοποιός διαφορά ανάμεσα στα CPLDs και τις FPGAs είναι η παρουσία κάποιων ενσωματωμένων μνημών και ενσωματωμένων λειτουργιών υψηλού επιπέδου (όπως προσθέτες και πολλαπλασιαστές) στις περισσότερες FPGAs, καθώς και το να διαθέτουν λογικά block για τη σύνθεση κάποιων αποκωδικοποιητών ή μαθηματικών συναρτήσεων.

[1.5] Η ιστορία τής FPGA

Η βιομηχανία των FPGA προήλθε από PROM μνήμες (programmable read only memory) και PLDs (programmable logic devices). Πρόσφεραν τη δυνατότητα να προγραμματίζονται μαζικά σ'ένα εργοστάσιο. Ωστόσο η λογική του προγραμματισμού, ήταν πολύ δύσκολη μεταξύ λογικών κυκλωμάτων.

Η βιομηχανία τών FPGA προήλθε από PROM μνήμες (programmable read only memory) και PLDs (programmable logic devices). Πρόσφεραν τη δυνατότητα να προγραμματίζονται μαζικά σ'ένα εργοστάσιο. Ωστόσο η λογική του προγραμματισμού, ήταν πολύ δύσκολη μεταξύ λογικών κυκλωμάτων.

Οι ιδρυτές της Xilinx, Ross Freeman και Bernard Vonderschmitt, εφηύραν την πρώτη εμπορικά εφαρμόσιμη FPGA το 1985 - την XC2064. Η XC2064 διέθετε προγραμματιζόμενες πύλες καθώς και προγραμματιζόμενες διασυνδέσεις μεταξύ των πυλών και υπήρξε η απαρχή για μια καινούρια τεχνολογία και μια καινούρια αγορά. Περιείχε 64 διαμορφούμενα λογικά block (CLBs), με δύο πίνακες αναζήτησης 3 εισόδων. 20 χρόνια μετά, ο Freeman μπήκε στο Hall of Fame των εφευρετών για την επινόησή του.

Κάποιες από τις θεμελιώδεις έννοιες και τεχνολογίες για τους προγραμματιζόμενους λογικούς πίνακες, πύλες και λογικά block θα τις βρει κανείς σε ευρεσιτεχνίες για τις οποίες βραβεύτηκαν οι David W. Page και LuVerne R. Peterson το 1985.

Στα τέλη της δεκαετίας του 1980 το υπουργείο των ναυτικών ενόπλων δυνάμεων της Αμερικής χρηματοδότησε ένα πείραμα του οποίου εισηγητής ήταν ο Steve Casselman. Σκοπός του πειράματος ήταν να αναπτυχθεί ένα σύστημα με 600.000 πύλες. Ο Casselman τελικά τα κατάφερε και το σύστημά του επιβραβεύτηκε με κατοχύρωση ευρεσιτεχνίας το 1992. Η Xilinx συνέχισε την εξέλιξη χωρίς ανταγωνισμό από το 1985 μέχρι και τα μέσα της δεκαετίας του '90, όταν άρχισαν να ξεπετάγονται ανταγωνιστικές εταιρίες οι οποίες σκοπό είχαν να καταλάβουν ένα σημαντικό κομμάτι της αγοράς. Το 1993 η Actel εξυπηρετούσε το 18 % της αγοράς. Η δεκαετία του '90 υπήρξε μια τρομερή περίοδος ανάπτυξης για τις FPGA και ως προς την εξέλιξη αλλά και ως προς το πλήθος των ανθρώπων που άρχισαν να ασχολούνται μ'αυτόν τον τομέα. Ενώ στις αρχές τις δεκαετίας οι FPGA

χρησιμοποιούνταν κυρίως στις τηλεπικοινωνίες και τα δίκτυα, προς το τέλος τρύπωσαν στις καταναλωτικές, αυτοκινητιστικές και βιομηχανικές εφαρμογές.

Οι FPGA είδαν για λίγο το φως της δημοσιότητας όταν το 1997, ο Adrian Thompson συνένωσε τεχνολογίες γενετικών αλγορίθμων και FPGAs για να κατασκευάσει μια συσκευή αναγνώρισης προτύπων ήχου. Ο αλγόριθμος του Thomson επέτρεπε σε μια FPGA της Xilinx που διέθετε έναν πίνακα 64 x 64 κελιών να υπολογίζει τις συνθέσεις που χρειάζονταν για να διεκπεραιώνει μια διεργασία αναγνώρισης ήχων.

[1.6] Μοντέρνες τεχνικές ανάπτυξης

Μια πρόσφατη τάση είναι να πάει η ακατέργαστη αρχιτεκτονική προσέγγιση ένα βήμα παραπέρα με το να συνδυάζονται τα λογικά block και οι διασυνδέσεις των παραδοσιακών FPGA με ενσωματωμένους μικροεπεξεργαστές και τα σχετικά περιφερειακά για να σχηματιστεί ένα πλήρες σύστημα πάνω σ'ένα προγραμματιζόμενο chip. Αυτή η τάση αντικατοπτρίζει την αρχιτεκτονική των Ron Perlof και Hana Potash του Burroughs Advanced Systems Group που συνδύαζε μια επαναπρογραμματιζόμενη CPU πάνω σ'ένα μόνο chip, το SB24. Αυτός ο σκοπός επιτεύχθηκε το 1982. Παραδείγματα τέτοιων υβριδικών τεχνολογιών μπορούμε να βρούμε σε συσκευές της σειράς Virtex-4 της Xilinx, οι οποίες περιέχουν έναν ή περισσότερους επεξεργαστές PowerPC ενσωματωμένους μέσα στη λογική δομή των FPGA. Μια ακόμη συσκευή τέτοιου είδους είναι η Atmel FPSLIC, διότι χρησιμοποιεί έναν AVR επεξεργαστή σε συνδυασμό με την αρχιτεκτονική προγραμματιζόμενης λογικής της Atmel.

[1.7] Μια εναλλακτική προσέγγιση

Όπως αναφέραμε παραπάνω, πολλές σύγχρονες FPGA έχουν τη δυνατότητα να επαναπρογραμματίζονται κατά το χρόνο εκτέλεσης, και αυτό οδηγεί στην ιδέα των επαναπρογραμματιζόμενων συστημάτων-CPUs οι οποίες αναδιαμορφώνονται για να προσαρμοστούν στην εννίοτε διεργασία. Ο Mitrion Virtual Processor της Mitrionics είναι ένα καλό παράδειγμα επαναπρογραμματιζόμενου softcore επεξεργαστή και ουσιαστικά αποτελείται από ένα σύνολο πυλών μιας FPGA. Παρόλα αυτά, δεν υποστηρίζει δυναμική διαμόρφωση κατά το χρόνο εκτέλεσης-έχει όμως τη δυνατότητα να προσαρμόζεται ανάλογα με το πρόγραμμα που πρόκειται να χρειαστεί να εκτελέσει.

Επιπροσθέτως, σε πειραματικό στάδιο μελετούνται καινούριες αρχιτεκτονικές χωρίς καν τη βοήθεια των FPGA. Μικροεπεξεργαστές όπως ο Stretch 5000 είναι παραμετροποιήσιμοι μόνο σε επίπεδο software υιοθετώντας μια πιο υβριδική προσέγγιση : αποτελούνται από έναν πίνακα πυρήνων του επεξεργαστή καθώς και από επαναπρογραμματιζόμενους πυρήνες-παρόμοιους μ'αυτούς μιας FPGA-στο ίδιο chip.

ΚΕΦΑΛΑΙΟ ΔΕΥΤΕΡΟ

ΕΝΣΩΜΑΤΩΜΕΝΟΙ ΕΠΕΞΕΡΓΑΣΤΕΣ ΤΗΣ FPGA

Η ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥΣ ΔΙΑΣΥΝΔΕΣΗ

[2.1] hardware

Ο ML403 εμπεριέχει μια FPGA τύπουVirtex 4, η οποία είναι μια οικογένεια ολοκληρωμένων που εμπεριέχουν ενσωματωμένους επεξεργαστές. Η δική μας FPGA ενσωματώνει τους powerPC (hard Processor) και MicroBlaze (soft processor).

[2.2] PowerPC Hard Core Επεξεργαστής

Ο PowerPC 405 της IBM είναι ένας 32-μπιτος CPU αρχιτεκτονικής RISC σχεδιασμένος απευθείας για τις FPGAs της Xilinx και πιο συγκεκριμένα για την οικογένεια Virtex-4, έτσι ώστε να εκτελούν ενσωματωμένες εφαρμογές υψηλών προδιαγραφών. Ο συνδυασμός ενός PowerPC με ολοκληρωμένες προδιαγραφές συνεπεξεργασίας προϋποθέτουν μια μεγάλη γκάμα επιλογών παραμετροποίησης. Υποστηρίζει βαθμιδωτό pipelining 5 επιπέδων, ξεχωριστές cache για εντολές και για δεδομένα, μια JTAG θύρα, FIFO προτεραιότητες, πολλαπλούς χρονιστές καθώς και μια MMU.

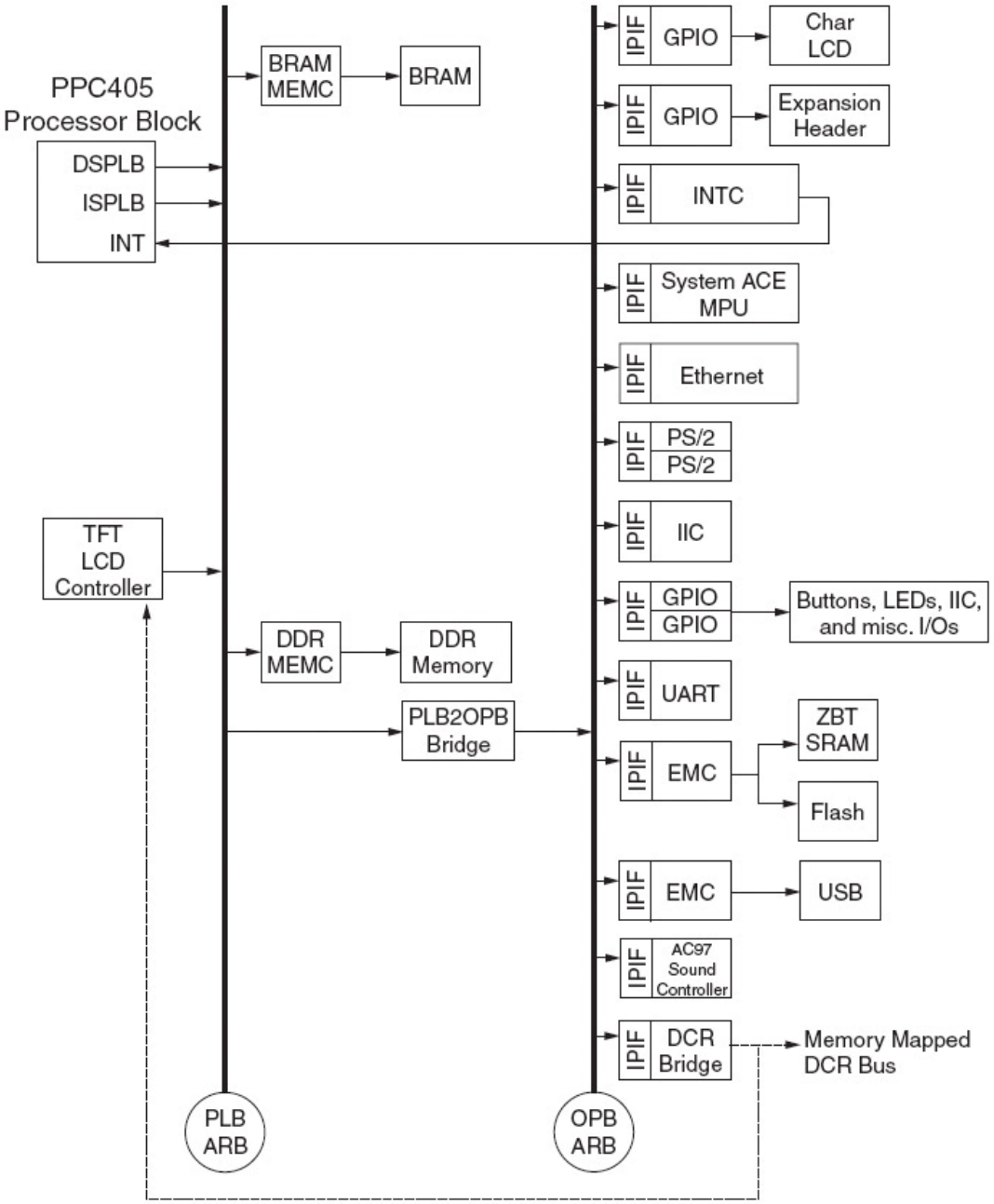
[2.3] MicroBlaze Soft Core Επεξεργαστής

Ένας Soft-Core επεξεργαστής συντίθεται χρησιμοποιώντας τα λογικά blocks της FPGA. Δουλεύοντας με έναν Soft-Core επεξεργαστή, μας επιτρέπει ένα μεγάλο βαθμό ευελιξίας και παραμετροποίησης. Ο MicroBlaze είναι ένας 32-μπιτος επεξεργαστής RISC αρχιτεκτονικής και αποτελεί μέρος του EDK (Embedded Development Kit). Το EDK είναι μέρος της σχεδιαστικής πλατφόρμας που μας προσφέρει η Xilinx, για το software σχεδιασμό ενσωματωμένων συστημάτων.

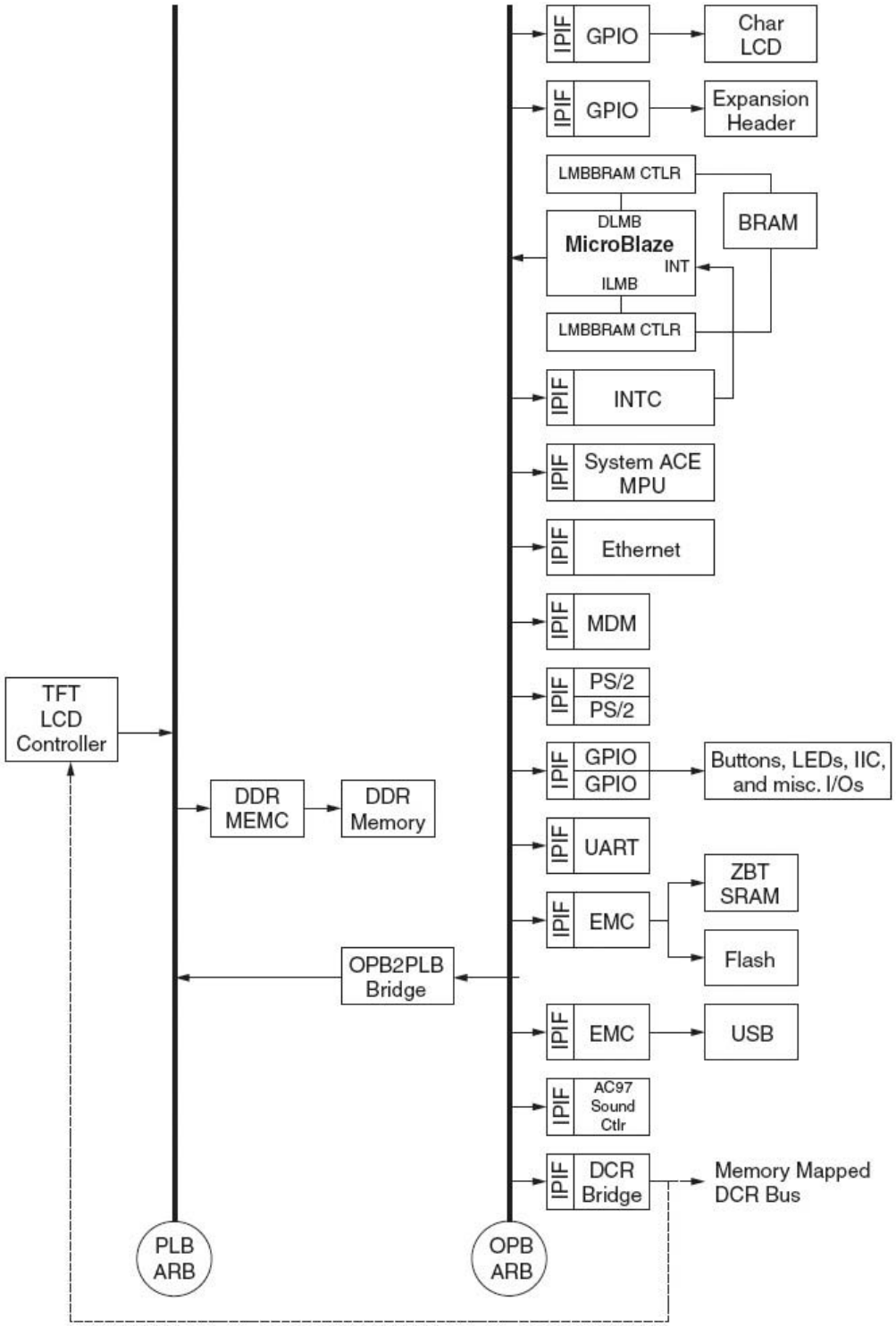
[2.4] Η διασύνδεση των PowerPC και MicroBlaze επεξεργαστών

Η διασύνδεση που χρησιμοποιείται είναι τύπου IBM core connect, συνδέει τη κάθε CPU με τις διαφορές περιφερικές μονάδες, χρησιμοποιώντας Processor Local Bus(PLB), On-Chip Peripheral Bus (OPB), και Device Control Register buses (DCR).Η FPGA μας ακλουθεί διαφορετική αρχιτεκτονική για να ενσωματώσει τον MicroBlaze επεξεργαστή και διαφορετική για τον PowerPC.Παρακάτω παραθέτουμε την αρχιτεκτονική της διασύνδεσης με καθένα από τους δύο επεξεργαστές.

Απεικόνιση της αρχιτεκτονικής του PowerPC hard core επεξεργαστή



Απεικόνιση της αρχιτεκτονικής του MicroBlaze soft core επεξεργαστή



2.4.1 Processor Local Bus (PLB)

Ο δίαυλος PLB διασυνδέει τη CPU με συσκευές υψηλής απόδοσης, όπως ελεγκτές μνήμης. Το πρωτόκολλο PLD υποστηρίζει μεταφορές δεδομένων που απαιτούν μεγάλο bandwidth(σε σχέση με τα OPB και DCR).Τα κυριότερα σημεία του πρωτοκόλλου PLB περιλαμβάνουν τη

σύγχρονη αρχιτεκτονική, ανεξάρτητα data paths ανάγνωσης - γραφής, και ξεχωριστές ενέργειες στα buses διευθύνσεων και δεδομένων. Ο σχεδιασμός της αρχιτεκτονικής περιλαμβάνει 64-bit PLB υποδομή με 64-bit συνδεδεμένες master και slave συσκευές. Όλες οι PLB συσκευές ρυθμίζονται κατάλληλα, γύρω από την αρχιτεκτονική της FPGA και χρησιμοποιούν pipelining για να αυξήσουν τις μέγιστες συχνότητες του ρολογιού και να μειώσουν την απαιτουμένη υπολογιστική ισχύ για λογικές πράξεις.

Οι PLB συσκευές περιλαμβάνουν:

• PLB Masters

♦ 640x480 VGA ελεγκτής

♦ OPB-to-PLB Bridge (για MicroBlaze σύστημα)

♦ PPC403 ISPLB (Instruction-Side PLB) Interface (γιά PPC403 σύστημα)

♦ PPC403 DSPLB (Data-Side PLB) Interface (για PPC403 σύστημα)

• PLB Slaves

♦ DDR SDRAM ελεγκτής

♦ BRAM (για PPC403 σύστημα)

♦ PLB-to-OPB Bridge (για PPC403 σύστημα)

• PLB Arbiter

♦ 64-bit Xilinx PLB Arbiter

2.4.2 On-Chip Peripheral Bus (OPB)

Το OPB συνδέει τις περιφερικές συσκευές χαμηλής απόδοσης με το σύστημα. Το OPB υλοποιεί μια λιγότερο πολύπλοκη αρχιτεκτονική απλοποιώντας τη περιφερική επέκταση. Οι PLB και οι OPB συσκευές επικοινωνούν μεταξύ τους με τη βοήθεια των OPBtoPLB είτε των PLBtoOPB γεφυρών. Όλες οι OPB συσκευές ρυθμίζονται κατάλληλα, γύρω από την αρχιτεκτονική της FPGA και χρησιμοποιούν pipelining για να αυξήσουν τις μέγιστες συχνότητες του ρολογιού και να μειώσουν την απαιτούμενη υπολογιστική ισχύ για λογικές πράξεις. The OPB devices in the reference design make use of Intellectual Property InterFace (IPIF) modules to further simplify IP development. Το IPIF μετατρέπει το OPB πρωτόκολλο σε κοινά interfaces όπως ένα SRAM πρωτόκολλο ή μια διεπαφή ελέγχου των register. Κάποιες IPIF ενότητες υποστηρίζουν επίσης DMA και interrupt λειτουργίες. Άλλες IPIF ενότητες απλοποιούν την ανάπτυξη λογισμικού διότι το πλαίσιο εργασίας έχει πολλά κοινά χαρακτηριστικά.

Το IPIF είναι σχεδιασμένο κυρίως για να υποστηρίζει μια ευρεία γκάμα διεπαφών αλλά αυτό δεν σημαίνει ότι είναι η καλύτερη λύση για όλες τις περιπτώσεις. Όπου χρειάζεται περισσότερη απόδοση ή λειτουργικότητα ο χρήστης μπορεί να αναπτύξει μια δική τουOPB διεπαφή. Τα πρωτόκολλα IPIF μπορούν επίσης να αναπτυχθούν έτσι ώστε να υποστηρίζουν διαφορετικά πρότυπα διαύλων όπως PLB. Αυτό επιτρέπει στο backend interface του Instruction Pointer να παραμείνει στη θέση που βρίσκεται ενώ μεταβάλλεται το interface διαύλου στο IPIF. Έτσι παρέχεται ένα αποτελεσματικό μέσο για την υποστήριξη διαφορετικών προτύπων διαύλου με τον ίδιο IP μηχανισμό.

Οι OPB συσκευές περιλαμβάνουν:

• OPB Masters

♦ Ethernet ελεγκτής

♦ Instruction-Side Interface (για MicroBlaze σύστημα)

♦ Data-Side Interface (για MicroBlaze σύστημα)

♦ PLB-to-OPB Bridge (για PPC403 σύστημα)

• OPB Slaves

♦ IIC Controller (για PPC403 σύστημα)

♦ General-Purpose Input/Output (GPIO) x3

♦ 16450 UART

♦ Interrupt Controller

♦ External Memory Controller x2

♦ Microprocessor Debug Module (για MicroBlaze σύστημα)

♦ AC97 Sound Controller

♦ OPB-to-DCR Bridge

♦ Ethernet Controller

♦ Dual PS/2 Controller

♦ System ACE™ MPU Interface

♦ OPB-to-PLB Bridge-In (για MicroBlaze σύστημα)

2.4.3 Device Control Register (DCR)

Ο DCR δίαυλος προσφέρει ένα πολύ απλό πρωτόκολλο διεπαφής και χρησιμοποιείται για να έχουμε πρόσβαση σε καταχωρητές ελέγχου και καταστάσεων για διάφορες συσκευές (χωρίς να υπερφορτώνει τις διεπαφές OPB και PLB). Μιάς και οι συσκευές DCR χρησιμοποιούνται σπάνια και δεν έχουν μεγάλες απαιτήσεις, χρησιμοποιούνται καθόλα τη διάρκεια της φάσης σχεδιασμού για συναρτήσεις, όπως καταχωρητές καταστάσεων λάθους, ελεγκτές interrupt και συσκευές αρχικοποίησης των λογικών πράξεων. Οι DCR προδιαγραφές προβλέπουν ότι τα DCR master και slave ρολόγια να είναι συγχρονισμένα μεταξύ τους και συσχετισμένα σε συχνότητα από ένα πολλαπλό ακέραιο. Είναι σημαντικό να γνωρίζουμε το πεδίο ορισμού του ρολογιού της κάθε DCR συσκευής για να διασφαλίσουμε τη σωστή λειτουργία.

Για να εντοπιστεί ένα 4-kB DCR μέσα στο χώρο της μνήμης, φτιάχνεται ένα αντίγραφο μιας OPB-to-DCR bridge. Οι DCR slave συσκευές που είναι συνδεδεμένες πάνω στην OPB-to-DCR bridge, περιλαμβάνουν:

• PLB Arbiter (αν γίνει enable)

• VGA TFT LCD ελεγκτής

[2.5] Υλοποίηση των παραπάνω αρχιτεκτονικών

Η Xilinx προσφέρει ένα εργαλείο για το σχεδιασμό της αρχιτεκτονικής του hardware, το EDK (Embeded Development Kit). Αυτό το εργαλείο προσφέρεται για το σχεδιασμό ενσωματωμένων συστημάτων.

Το λογισμικό σύστημα που θέλουμε να εγκαταστήσουμε στο ML403 είναι το uCLinux.

2.5.1 µCLinux

είναι ένα Linux για μικροελεγκτές. Προέκυψε από τον αυθεντικό kernel του Linux για μικροεπεξεργαστές χωρίς MMU (μονάδα διαχείρισης μνήμης). Ξεκίνησε υποστηρίζοντας kernel 2,0 και σήμερα υποστηρίζει 2,4 και 2,6. Αυτή η διανομή συνεχίζεται να εξελίσσεται και να υποστηρίζει εργαλεία για διάφορες συσκευές όπως routers, κάμερες ασφαλείας, DVD/MP3 players, VoIP τηλέφωνα, scanners και αναγνώστες καρτών μνήμης. Οι αρχιτεκτονικές των επεξεργαστών που υποστηρίζει είναι οι εξής:

* Altera NIOS
* ADI Blackfin
* ARM
* ETRAX
* Freescale M68K
* Fujitsu FRV
* Hitachi H8
* Intel i960
* MIPS
* NEC V850E
* Xilinx MicroBlaze

Επειδή υποστηρίζει τις παραπάνω αρχιτεκτονικές, ενώ το ML403 υποστηρίζει PowerPC και MicroBlaze αρχιτεκτονικές, κάνουμε υλοποίηση του uCLinux σε MicroBlaze.

ΚΕΦΑΛΑΙΟ ΤΡΙΤΟ

Η ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ MICROBLAZE ΕΠΕΞΕΡΓΑΣΤΗ

[3.1] Εντολές (Instructions)

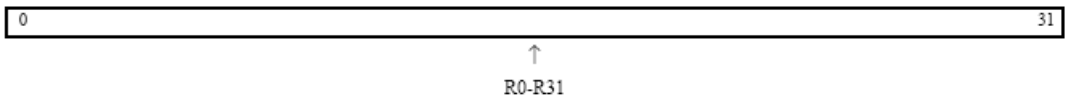
Όλες οι εντολές του MicroBlaze έχουν εύρος 32 bit και ορίζονται ως τύπου Α ή τύπου Β. Οι εντολές τύπου Α έχουν έναν ή δύο καταχωρητές-τελεστές και έναν καταχωρητή-προορισμό. Οι εντολές τύπου Β έχουν έναν καταχωρητή-τελεστή και ένα offset εύρους 16 bit (που μπορεί να φτάσει τα 32 bit αν προηγηθεί η κατάλληλη εντολή). Έχουν και έναν μόνο καταχωρητή-προορισμό. Οι εντολές χωρίζονται στις εξής κατηγορίες : αριθμητικές, λογικές, διακλάδωσης, εγγραφής/ανάγνωσης και ειδικές.

[3.2] Καταχωρητές (Registers)

Το σέτ εντολών(instruction set) μιας MicroBlaze έχει ορθογώνια αρχιτεκτονική. Έχει τριάντα δύο 32-bit καταχωρητές γενικού σκοπού, και έχει και πάνω από δεκαοχτώ 32-bit καταχωρητές ειδικού σκοπού, εξαρτώμενους από παραμετροποιημένες επιλογές.

3.2.1 Καταχωρητές γενικού σκοπού

Οι τριάντα δύο 32-bit καταχωρητές γενικού σκοπού ξεκινούν να αριθμούνται από τον R0 έως τον R31. Το αρχείο των καταχωρητών γίνεται reset στο bit stream κατεβάσματος. Η reset τιμή είναι 0x00000000. Στην από κάτω εικόνα απεικονίζονται οι καταχωριτές γενικού σκοπού



3.2.2 Καταχωρητές ειδικού σκοπού

- Program Counter (PC)
- Machine Status Register (MSR)
- Exception Address Register (EAR)
- Exception Status Register (ESR)
- Branch Target Register (BTR)
- Floating Point Status Register (FSR)
- Processor Version Register (PVR)

[3.3] Αρχιτεκτονική μνήμης

Ο MicroBlaze έχει σχεδιαστεί με βάση την αρχιτεκτονική μνήμης Harvard : η πρόσβαση εντολών και δεδομένων γίνονται σε διαφορετικές θέσεις μνήμης. Κάθε θέση μνήμης έχει εύρος 32 bit. Η μνήμη των εντολών και των δεδομένων μπορούν να συνδιαστούν είτε σε σειρά είτε παράλληλα και να χρησιμοποιηθούν σαν ένα block φυσικής μνήμης. Αυτό μπορεί να φανεί χρήσιμο στο debugging του λογισμικού. Εκτός απ'το ότι έχουν και τα 2 block εύρος 32 bit, χρησιμοποιούν και τα 2 big endian format. Ο MicroBlaze υποστηρίζει πρόσβαση δεδομένων εύρους μιας ψηφιολέξης, μισής ψηφιολέξης και ενός byte. Οι προσβάσεις δεδομένων πρέπει να γίνονται με την απαραίτητη ευθυγράμμιση (π.χ. δεν μπορούμε να διαβάσουμε μισή ψηφιολέξη με το δεύτερο και τρίτο byte), εκτός και αν ο επεξεργαστής είναι ρυθμισμένος να υποστηρίζει μη ευθυγραμμισμένες εξαιρέσεις. Όλες οι προσβάσεις εντολών όμως πρέπει να είναι αυστηρά ευθυγραμμισμένες ως προς το εύρος της μιας ψηφιολέξης.

Ο MicroBlaze δεν ξεχωρίζει προσβάσεις δεδομένων σε εισόδου/εξόδου και μνήμης. Ο επεξεργαστής έχει maximum 3 interface για πρόσβαση μνήμης :

• Local Memory Bus (LMB)

• Processor Local Bus (PLB) or On-Chip Peripheral Bus (OPB)

• Xilinx CacheLink (XCL)

[3.4] Cache Δεδομένων

Ο MicroBlaze μπορεί να χρησιμοποιηθεί με μια προαιρετική cache δεδομένων προκειμένου να αυξήσουμε την απόδοσή του. Το εύρος της μνήμης που έχει γίνει cache δεν πρέπει να περιλαμβάνει διευθύνσεις από την περιοχή του LMB. Η cache δεδομένων έχει τα εξής χαρακτηριστικά :

- Ευθεία χαρτογράφηση (συσχετισμός προς μια κατεύθυνση)
- Write-through
- Προσδιορισμός από το χρήστη για το εύρος διευθύνσεων μνήμης που μπορούν να γίνουν cache
- Παραμετροποιήσιμο μέγεθος και ετικέτες cache
- Caching με το interface της Xilinx CacheLink (XCL)
- Επιλογή για χρήση γραμμών cache 4 ή 8 ψηφειολέξεων
- Cache on και off ελεγχόμενη από ένα bit στο MSR

3.4.1 Η λειτουργικότητα της Cache Δεδομένων

Όταν χρησιμοποιείται η cache δεδομένων, το block διευθύνσεων μνήμης χωρίζεται σε 2 τμήματα : ένα cacheable τμήμα κι ένα non-cacheable τμήμα. Η cacheable περιοχή προσδιορίζεται από 2 παραμέτρους : C_DCACHE_BASEADDR και C_DCACHE_HIGHADDR. Όλες οι διευθύνσεις μέσα σε αυτό το τμήμα ανταποκρίνονται στο cacheable διάστημα διευθύνσεων. Όλες οι υπόλοιπες διευθύνσεις είναι non-cacheable.

Η cacheable διεύθυνση δεδομένων αποτελείται από 2 μέρη : τη διεύθυνση cache και τη διεύθυνση ετικέτας. Η cache δεδομένων του MicroBlaze μπορεί να οριστεί από 64 bytes έως 64 Kbytes. Αυτό έχει ως επακόλουθο μια cache διευθύνσεων από 6 έως 16 bits. Αθροίζοντας τη διεύθυνση ετικέτας και τη διεύθυνση cache θα πρέπει να οδηγούμαστε στην πλήρη διεύθυνση της cacheable μνήμης.

3.4.2 Λειτουργίες της Cache δεδομένων

Η cache δεδομένων του MicroBlaze υλοποιεί ένα write-through πρωτόκολλο. Δεδομένου ότι η cache είναι ενεργοποιημένη, μια ενέργεια αποθήκευσης σε μια διεύθυνση που βρίσκεται μέσα στην cacheable περιοχή δημιουργεί αντίστοιχα μια εγγραφή ψηφειολέξης, μισής ψηφειολέξης ή ενός byte μέσω του CacheLink δεδομένων στην εξωτερική μνήμη. Η εγγραφή ενημερώνει επίσης τα cached δεδομένα αν η διεύθυνση στην οποία πρόκειται να γίνει αποθήκευση βρίσκεται μέσα στην cache.

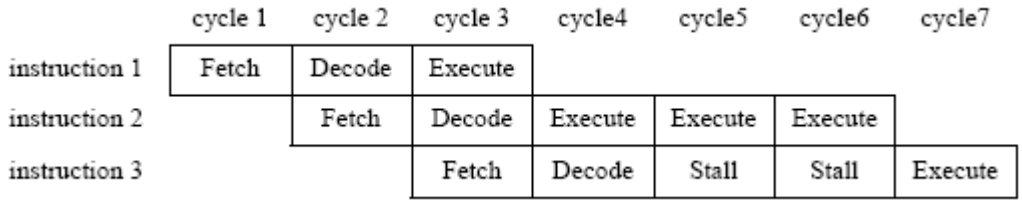
Δεδομένου ότι η cache είναι ενεργοποιημένη, μια ενέργεια φόρτωσης από μια διεύθυνση μέσα στην cacheable περιοχή σηματοδοτεί έναν έλεγχο για να δούμε αν τα δεδομένα που έχουν ζητηθεί βρίσκονται σε cached κατάσταση. Αν είναι, η υλοποίηση εξελίσσεται ως έχει. Αν όχι, ζητείται η διεύθυνση μέσω του CacheLink δεδομένων, και το pipelining του επεξεργαστή παγώνει μέχρι να συσχετιστεί μια γραμμή cache με τη ζητούμενη διεύθυνση, και να επιστρέψει από τον ελεγκτή εξωτερικής μνήμης.

[3.5] Η Pipeline αρχιτεκτονική

Η δομή του MicroBlaze βασίζεται στο pipelining. Για τις περισσότερες εντολές, κάθε βήμα διαρκεί έναν κύκλο ρολογιού. Συνεπώς, ο αριθμός των κύκλων ρολογιού που χρειάζονται για να εκτελεστεί μια εντολή είναι ίσος με τον αριθμό των βημάτων του pipelining, και μια εντολή ολοκληρώνεται σε κάθε κύκλο, με εξαίρεση κάποιες περίπλοκες εντολές που απαιτούν περισσότερους κύκλους. Εκεί αναστέλλεται η λειτουργία του pipelining. Όταν η επεξεργασία γίνεται σε πιο αργές μνήμες, οι εντολές χρειάζονται περισσότερους από έναν κύκλους. Η επιπλέον καθυστέρηση επηρεάζει άμεσα την αποτελεσματικότητα του pipelining. Αν και η λειτουργία του pipelining αναστέλλεται από εντολές που θέλουν περισσότερους από έναν κύκλους για να ολοκληρωθούν, ο buffer εξακολουθεί να γεμίζει με τις επόμενες εντολές. Στη συνέχεια, προς αποφυγήν της επιπλέον καθυστέρησης φορτώνονται απευθείας από τον buffer οι επόμενες εντολές αντί για περαιτέρω αναμονή έως ότου να δοθεί πρόσβαση για ανάγνωση της μνήμης.

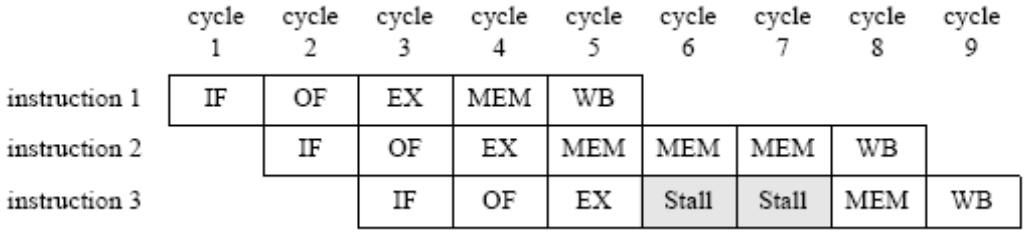
3.5.1 Pipeline τριών σταδίων

Όταν βελτιστοποιείται στο μέγιστο η αξιοποίηση του χώρου, το pipelining χωρίζεται σε 3 στάδια για να ελαχιστοποιήσουμε το κόστος του hardware : Fetch, Decode, and Execute.



3.5.2 Pipeline πέντε σταδίων

Στην αντίθετη περίπτωση με τα παραπάνω, το pipelining χωρίζεται σε 5 στάδια για να μεγιστοποιήσουμε την απόδοση : Fetch (IF), Decode (OF), Execute (EX), Access Memory (MEM), and Writeback (WB).



[3.6] Μονάδα κινητής υποδιαστολής (FPU)

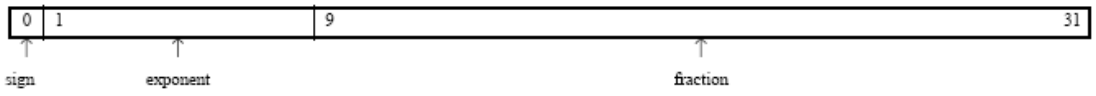
Η μονάδα κινητής υποδιαστολής FPU (Floating Point Unit) του MicroBlaze, βασίζεται στο πρότυπο IEEE 754 :

- Χρησιμοποιεί διάταξη κινητής υποδιαστολής απλής ακρίβειας IEEE 754, συμπεριλαμβανομένων και κάποιων ορισμών για το άπειρο, τον προσδιορισμό NaN (not-a-number) και το μηδέν
- Υποστηρίζει εντολές πρόσθεσης, αφαίρεσης, πολλαπλασιασμού, διαίρεσης, σύγκρισης, μετατροπής και τετραγωνικής ρίζας
- Υλοποιεί λειτουργία round-to-nearest

[3.7] Μορφή (Format)

Το format με το οποίο μπορούμε να αναπαραστήσουμε έναν αριθμό κινητής υποδιαστολής απλής ακρίβειας αποτελείται από τα εξής 3 πεδία :

1. πρόσημο (1 bit/θέση 0)
2. Εκθέτης (8 bit/θέση 1-8)
3. ακέραιο ή/και δεκαδικό μέρος (23 bit/θέση 9-31)



[3.8] Πράξεις

Όλες οι FPU πράξεις του MicroBlaze χρησιμοποιούν καταχωρητές γενικού σκοπού, αντί για έναν υποτιθέμενο καταχωρητή αποκλειστικά για πράξεις κινητής υποδιαστολής.

3.8.1 Αριθμητικές πράξεις

Αριθμητικές πράξεις της FPU :

- πρόσθεση, fAdd
- αφαίρεση, f Sub
- πολλαπλασιασμός, f Mul
- διαίρεση, f Div
- τετραγωνική ρίζα, f Sqrt

3.8.2 Συγκρίσεις

Λογικές πράξεις της FPU :

- μικρότερο από, fCmp.lt
- ίσο, fCmp.eq
- μικρότερο ή ίσο, fCmp.le
- μεγαλύτερο από, fCmp.gt
- διάφορο, fCmp.ne
- μεγαλύτερο ή ίσο, fCmp.ge
- μη σε σειρά, fCmp.un

3.8.3 Μετατροπές

Πράξεις μετατροπής της FPU :

- από προσημασμένο ακέραιο σε κινητής υποδιαστολής, flt
- από κινητής υποδιαστολής σε προσημασμένο ακέραιο, fint

[3.9] Fast Simplex Link (FSL)

Ο MicroBlaze μπορεί να υποστηρίξει έως και 16 Fast Simplex Link (FSL) λειτουργίες, αποτελούμενες η καθεμία από μία θύρα εισόδου και μία θύρα εξόδου. Τα κανάλια για το FSL είναι στην πραγματικότητα μονοκατευθυντικές επαφές από σημείο σε σημείο (point-to-point) για εκπομπή δεδομένων και έχουν εύρος 32 bit. Ένα ξεχωριστό bit συμβολίζει το αν η ψηφιολέξη είναι πληροφορία ελέγχου ή δεδομένων. Η εντολή get στον ISA του MicroBlaze μας επιτρέπει να μεταφέρουμε πληροφορίες από μια θύρα FSL σε έναν καταχωρητή γενικού σκοπού. Η εντολή put μας επιτρέπει να κινηθούμε στην αντίστροφη διαδρομή. Οι 2 παραπάνω εντολές έχουν ένα flag με 4 επιλογές : εμπόδισης δεδομένων, μη εμπόδισης δεδομένων, εμπόδισης ελέγχου και μη εμπόδισης ελέγχου.

[3.10] Debugging

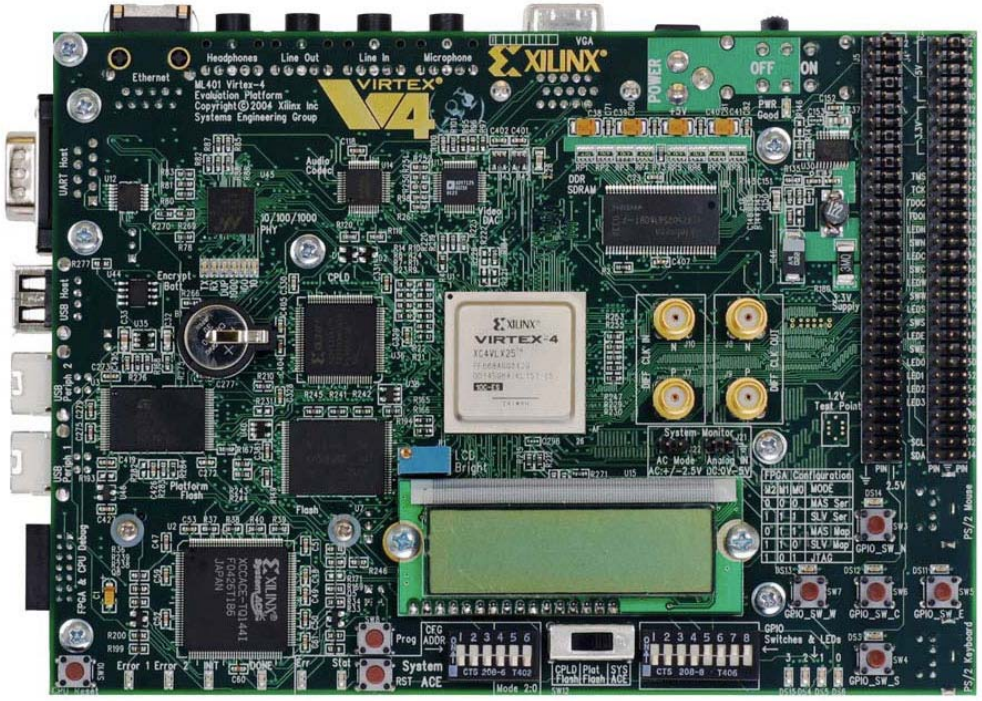
Ο MicroBlaze διαθέτει ένα interface για την υποστήριξη του debugging μέσω του JTAG βασισμένο σε εργαλεία για software debugging (κοινώς γνωστά και ως BDM-Background Debug Mode debuggers) όπως το Xilinx Microprocessor Debug. Είναι σχεδιασμένο για να συνδέεται με τον πυρήνα του MDM (Xilinx Microprocessor Debug Module), ο οποίος επικοινωνεί με τη θύρα του JTAG που βρίσκεται πάνω στο board των FPGA της Xilinx. Πολλαπλά αντίγραφα του MicroBlaze μπορούν να συνδεθούν με ένα και μόνο MDM για να επιτρέψουν το debugging πολλαπλών πυρήνων. Τα χαρακτηριστικά του debugging περιέχουν τα εξής :

- Έναν διαμορφούμενο αριθμό από hardware breakpoints και watchpoints, καθώς και άπειρα software breakpoints
- Εξωτερικός έλεγχος του επεξεργαστή επιτρέπει στα εργαλεία του debugging λειτουργίες stop, reset και single step
- Ανάγνωση από και εγγραφή σε : μνήμη, καταχωρητές γενικού σκοπού και ειδικού σκοπού, εκτός από EAR, EDR, ESR, BTR και PVR0 εως PVR11 όπου επιτρέπουν μόνο ανάγνωση
- Υποστήριξη για πολλαπλούς επεξεργαστές50
- Εγγραφή σε cache εντολών και δεδομένων

ΚΕΦΑΛΑΙΟ ΤΕΤΑΡΤΟ

ΔΗΜΙΟΥΡΓΙΑ MICROBLAZE ΕΠΕΞΕΡΓΑΣΤΗ ΚΑΙ ΛΕΙΤΟΥΡΓΙΚΟΥ ΣΥΣΤΗΜΑΤΟΣ (UCLINUX) ΠΡΑΚΤΙΚΗ ΕΦΑΡΜΟΓΗ ΣΤΗΝ FPGA

Η πλακέτα FPGA ML403



Οι προδιαγραφές και τα επιμέρους κομμάτια που απαρτίζουν το FPGA board ML403 αναφέρονται στο ΠΑΡΑΡΤΗΜΑ Β στη σελίδα 79

Το παράρτημα είναι χωρισμένο στα παρακάτω σε 3 μέρη :

- *Μονάδες επεξεργασίας και μνήμης*
- *Μονάδες εισόδου και εξόδου*
- *Leds, Buttons & Switches*

[4.1] ΠΕΙΡΑΜΑΤΙΚΟ ΜΕΡΟΣ

Στο πειραματικό μέρος θέλουμε τη δημιουργία ενός MicroBlaze soft επεξεργαστή και εν συνεχεία την εγκατάσταση του uCLinux λογισμικού συστήματος στο ML403 board.

Για να τα πετύχουμε το σκοπό μας χρειάζεται αρχικά να υπάρχουν οπωσδήποτε τα παρακάτω:

- Χρήση ενός Linux λογισμικού για υπολογιστή
- Xilinx EDK 8.1
- Linux kernel 2.6
- uCLinux source
- BSP

Αναλυτικότερα

4.1.1 Χρήση ενός Linux λογισμικού για υπολογιστή

Για τη δημιουργία του uCLinux, χρειάζεται η χρήση ενός UNIX περιβάλλοντος. Για αυτόν τον λόγο χρειαζόμαστε μια διανομή των Linux για να χτίσουμε το καινούργιο λειτουργικό. Στη προκειμένη περίπτωση κάναμε εγκατάσταση του Linux Fedora 8 στον προσωπικό μας υπολογιστή.

4.1.2 Xilinx EDK 8.1

Το EDK προέρχεται από τις λέξεις Embedded Developers Kit που σημαίνει εργαλείο υποστήριξης για ανάπτυξη λογισμικού για ενσωματωμένων επεξεργαστών. Χρησιμοποιείται κυρίως για την υποστήριξη του soft core επεξεργαστή MicroBlaze, καθώς και του ενσωματωμένου hard core PowerPC405 και PowerPC440. Μας δίνει τη δυνατότητα μέσω του system generator της Xilinx να σχεδιάζουμε soft core επεξεργαστές, και στη προκειμένη περίπτωση μας βοηθάει στο σχεδιασμό του MicroBlaze επεξεργαστή.

4.1.3 Linux Kernel 2.6

Ο Linux Kernel είναι ο πυρήνας οποιουδήποτε Linux λειτουργικού συστήματος. Είναι το ποιό βασικό συστατικό για τη δημιουργία και την ανάπτυξη ενός καινούργιου Linux λειτουργικού, σε συνδυασμό στη προκείμενη περίπτωση με τον πηγαίο κώδικα του uCLinux.

4.1.4 uCLinux source files

Δημιουργήσαμε ένα φάκελο με την ονομασία uclinux-dist, στη παρακάτω διαδρομή /petalinux/software/. Εν συνεχεία ανοίγουμε μια κονσόλα του UNIX όπου γράφουμε τις παρακάτω εντολές για να λάβουμε τα πηγαία αρχεία του uCLinix, για το χτίσιμο ενός τροποποιημένου uCLinux για το ML403 board.

```
$ cd /petalinux/software
$ mkdir uclinux-dist
$ cd uclinux-dist
```

Κάνουμε είσοδο στον CVS server με τη παρακάτω εντολή :

```
$ cvs –d:pserver:anonymous@cvs.uclinux.org:/var/cvs
login
```

(πατάμε enter όταν μας ζητηθεί το password)

Κατεβάζουμε τη διανομή του uCLinux και τον 2.6 Linux kernel με τις εξής εντολές :

```
$ cvs –z3 –d:pserver:anonymous@cvs.uclinux.org
:/var/cvs co uclinux-dist
$ cvs –z3 –d:pserver:anonymous@cvs.uclinux.org
:/var/cvs co uclinux-2.6.x
```

4.1.5 BSP

Στα ενσωματωμένα συστήματα ένα BSP (Board Support Package), είναι στην ουσία η υλοποίηση του κώδικα υποστήριξης για ένα συγκεκριμένο Board, και βοηθάει στο να παντρεύονται οι πόροι του hardware με ένα συγκεκριμένο λειτουργικό σύστημα. Συνήθως χτίζεται (γίνεται build) για την κατασκευή ενός bootloader, ο οποίος περιέχει την ελάχιστη υποστήριξη συσκευών για να φορτώσει το λειτουργικό μας σύστημα, καθώς και τους drivers όλων των περιφεριακών πάνω στο board. Στη δική μας περίπτωση, το BSP του uCLinux για το δικό μας board, παρέχεται από τη petalogix, με την ονομασία petalinux BSP.

[4.2] Δημιουργία του βασικού συστήματος

ΔΗΜΙΟΥΡΓΙΑ MICROBLAZE

Εδώ κάνουμε τη δημιουργία του hardware σχεδιασμού του βασικού συστήματος χρησιμοποιώντας το EDK 8.1.

4.2.1 Χρήση του Base System Builder (BSB)

Από το EDK ανοίγουμε το BSB (Base System Builder) που είναι ένα εργαλείο για να φτιάξουμε ένα καινούργιο project.

Επιλογή του Board και ρυθμίσεις επεξεργαστή

Attribute	**Value**
Board Select	
Vendor	Xilinx
Board	Virtex 4 ML401 Evaluation Platform
Version	1
Processor Configuration	
Reference Clock Frequency	100.00 Mhz
Processor-Bus Clock Frequency	100.00 MHz
Reset Polarity	Active LOW
On-chip H/W debug module	Enable
Local Memory	8K
Cache	Enable Cache link
Cached Memory	DDR_SDRAM_64x32*

Πρόσθεση περιφεριακών μονάδων

Attribute	Value
RS232 Uart	
Device	Enable
Baudrate	9600
Interrupt	Enable
Flash 2Mx32	
Device	Enable
DDR SDRAM 64Mx32	
Device	Enable
Ethernet MAC	
Device	Enable
DMA mode	Simple DMA
Interrupt	Enable
OPB TIMER	
Device	Enable
One timer present	Enable
Bit Width	32
Interrupt	Enable

Ρύθμιση της μνήμης cache του επεξεργαστή

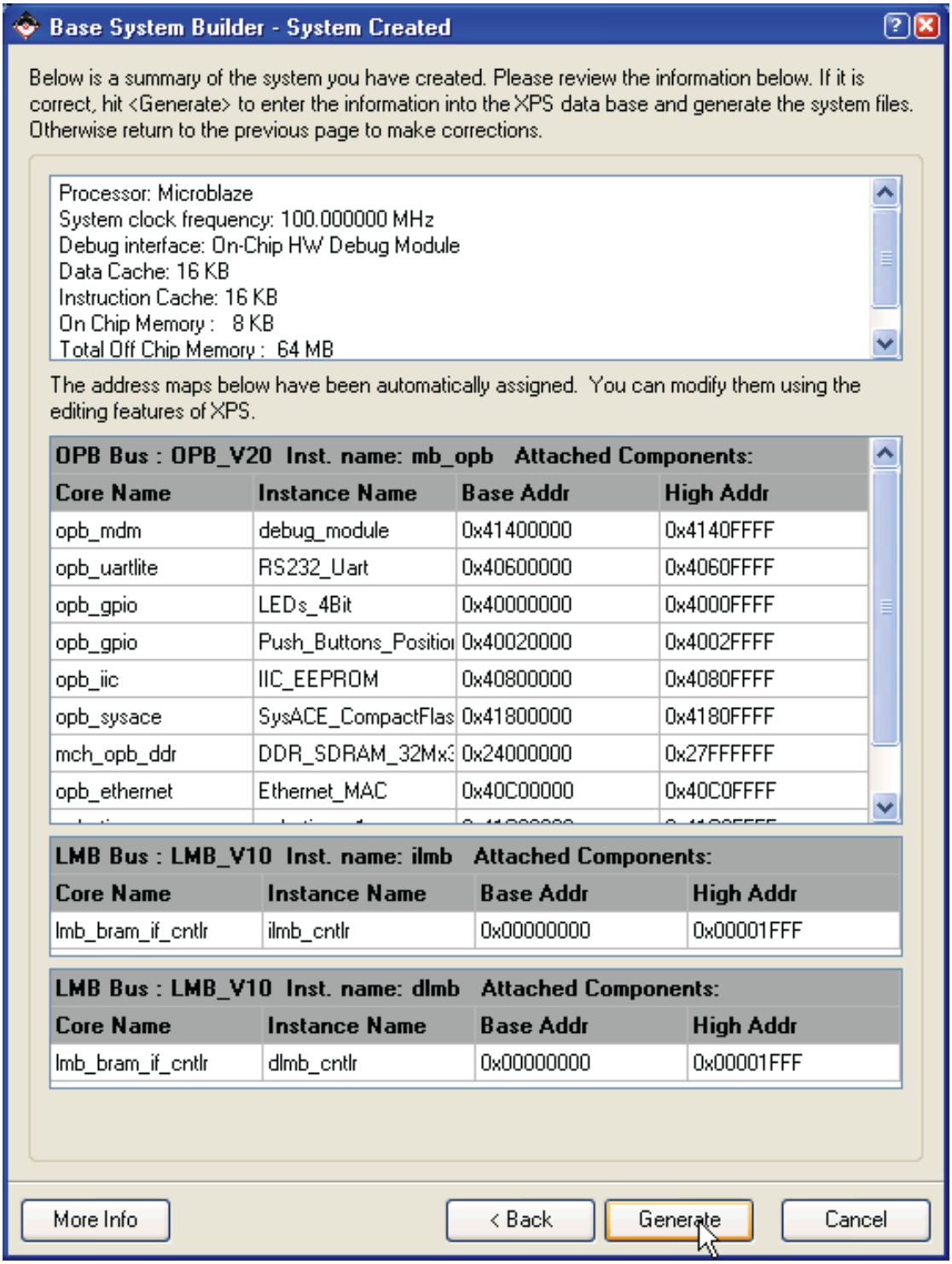
Attribute	Value
Dcache	8k
ICache	8k
Dcache from	DDR_SDRAM_64MBx32#
Icache from	DDR_SDRAM_64MBx32#

Επικοινωνία Input / Output

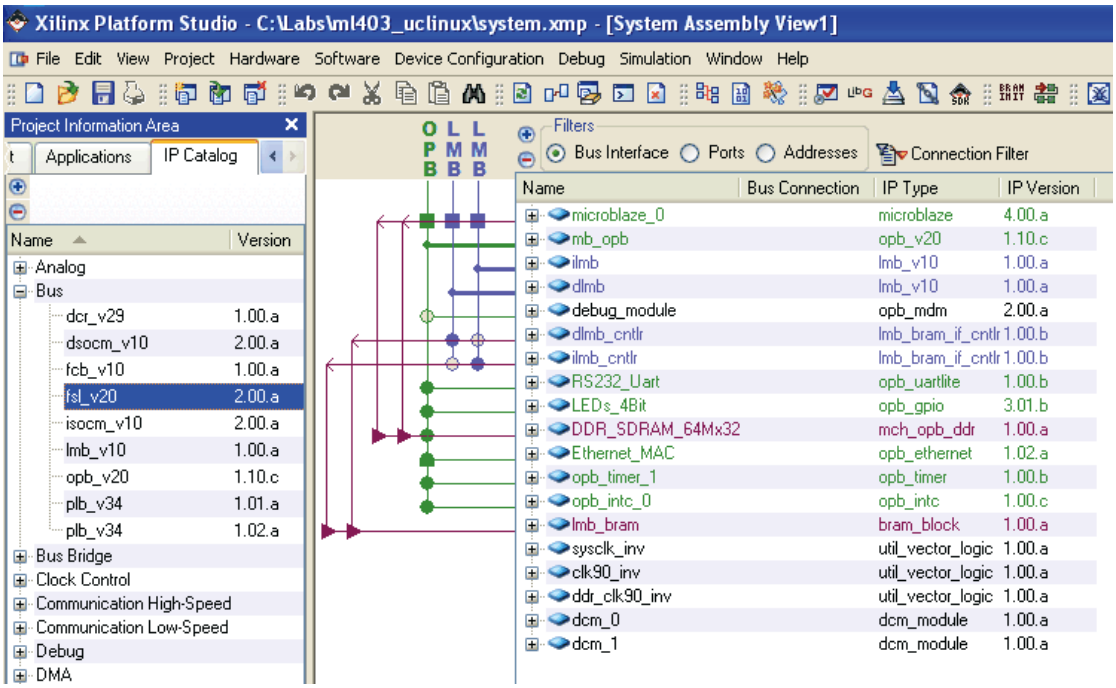
Attribute	Value
STDIN	RS232_Uart
STDOUT	RS232_Uart

Έχοντας ολοκληρώσει τις ρυθμίσεις για τις βασικές προδιαγραφές του hardware εμφανίζεται ένα report όπου επαληθεύω τις ρυθμίσεις που επέλεξα, και στη συνέχεια

κάνω ένα “generate” όπου δημιουργείται το base system. Το αρχείο που παράγεται είναι το system.xmp



Το Base System εμφανίζεται σε μορφή system assembly view



Στη συνέχεια προσθέτω και γρήγορο download interface και ενεργοποιώ την υποστήριξη για FSL.

4.2.2 Χρήση του Petalinux BSP

Για να συμπεριλάβουμε το BSP της Petalinux στο project μας εισάγουμε την μεταβλητή ModuleSearchPath στο system.xmp που δημιουργήθηκε παραπάνω από το "generate". Έτσι ενημερώνουμε το EDK για το path του BSP.

4.2.3 Προσθήκη Bootloader

Στο ήδη υπάρχων project δημιουργούμε ένα software application project για τον Bootloader. Ο Bootloader χρειάζεται για να φορτώσει το σύστημα μας με το που του δώσουμε ρεύμα ή μετά από κάποιο reset. Η Petalinux προσφέρει μια λύση για Bootloader σε 2 φάσεις.

Η πρώτη φάση ονομάζεται FS-Boot (First Stage Boot), όπου ξεκινάει ο βασικός Bootloader, όπου στέλνει στο Board ένα αρχικό Image του Bootloader, για την εκκίνηση του Board.

Για τον βασικό Bootloader η Petalinux χρησιμοποιεί έναν open source Bootloader τον U-Boot. Ο πηγαίος κώδικας για το U-Boot είναι ενσωματωμένος στο κώδικα του Petalinux ο οποίο ρυθμίζεται και μεταγλωττίζεται ατιμότατα σαν ένα ξεχωριστό κομμάτι του όλου συστήματος.

Όταν ξεκινάει να δουλεύει ο MicroBlaze επεξεργαστής ξεκινούν οι εντολές (instruction code) από την αρχή της μνήμης (0x00000000). Τυπικά από αυτή τη διεύθυνση ξεκινάει η BRAM (που είναι μνήμη πάνω στον MicloBlaze). Αν και η BRAM προσφέρεται για να τοποθετήσουμε έναν με πολλαπλές λειτουργίες Bootloader πάνω της, αυτό θεωρείται σπατάλη πόρων. Γι' αυτό και χρησιμοποιούμε έναν μικρότερο στην αρχή, για να εκκινήσει έναν μεγαλύτερο ο οποίος βρίσκεται σε μια flash μνήμη.

4.2.4 fs-boot

Στο software application project που δημιουργούμε στο EDK και το ονομάζουμε “fs-boot” κάνουμε χρήση των παρακάτω αρχείων:

Attribute	**Value**
Source File	fs-boot.c
Source File	srec.c
Source File	time.c
Header File	fs-boot.h
Header File	srec.h
Header File	time.h

κάνουμε τις εξής ρυθμίσεις για τη μεταγλώττιση του project “fs-boot”:

Attribute	**Value**
Environment	
Application Mode	executable
Output ELF file	default value
Linker Script	Use default Linker Script
Stack Size	1K
Debug and Optimisation	
Optimization Level	Size Optimized (-Os)
Advance	
Other Compiler Options to Append	-Wall

Αφότου κάνουμε τις παραπάνω ρυθμίσεις, κάνουμε και αρχικοποίηση (Initialize) της BRAM.

Ρυθμίσεις Software από το EDK :

Attribute	**Value**
Processor, Driver Parameters and Interrupt Handlers	
CORE_CLOCK_FREQUENCY	66666667
xmdstub_peripheral	none
OS & Library Settings	petalinux
PetaLinux version	1.00.b
OS and Library	
lmb_memory	dlmb_cntlr
main_memory	DDR_SDRAM_64MBx32
main_memory_bank	0
flash_memory	FLASH_2MBx32
flash_memory_bank	0
stdout	RS232_Uart
stdin	RS232_Uart

Έχοντας δώσει τις παραπάνω ρυθμίσεις, καθώς και από όλες τις ρυθμίσεις που έχουμε κάνει στο EDK, κάνοντας κάποια “Generate” από τα αντίστοιχα menus του software και του hardware, έχουμε φτάσει στο σημείο να παράγουμε τις βιβλιοθήκες που χρειάζονται, το BSP, το Netlist και τέλος το Bitstream αρχείο.

ΚΕΦΑΛΑΙΟ ΠΕΜΠΤΟ

ΕΛΕΓΧΟΣ ΕΠΙΔΟΣΕΩΝ

(BENCHMARKING)

[5.1] Benchmark (έλεγχος επιδόσεων)

Το benchmark είναι ο έλεγχος κατά τον οποίο τρέχοντας ένα πρόγραμμα, μια συστάδα προγραμμάτων ή άλλες λειτουργίες, καθορίζεται η σχετική επίδοση ενός αντικειμένου. Αυτό επιτυγχάνεται κυρίως με το να κάνουμε κάποιες πραγματικές δοκιμές καθώς και με κάποια δοκιμαστικά τεστ. Ο όρος "benchmark" αναφέρεται επίσης και στα προγράμματα που είναι σχεδιασμένα αποκλειστικά για σκοπούς δοκιμών εξέλιξης. Το benchmarking συνήθως σχετίζεται με τον προσδιορισμό χαρακτηριστικών των επιδόσεων του hardware ενός υπολογιστή. Για παράδειγμα, τις μαθηματικές πράξεις κινητής υποδιαστολής που μπορεί να κάνει ένας επεξεργαστής. Παρόλα αυτά, υπάρχουν περιπτώσεις όπου η τεχνική του benchmarking μπορεί να εφαρμοστεί και σε εφαρμογές (software). Για παράδειγμα, σε περιπτώσεις που θέλουμε να αντιπαραθέσουμε compilers ή συστήματα διαχείρισης βάσεων δεδομένων. Ένας άλλος λόγος που μπορεί να χρησιμοποιήσει κάποιος software benchmarks-και πιο συγκεκριμένα ολόκληρες σουίτες από benchmarks-είναι για να εξασφαλίσει την εγκυρότητα/ορθότητα των προγραμμάτων του.

[5.2] Σκοπός

Καθώς η αρχιτεκτονική υπολογιστών εξελισσόταν, γινόταν όλο και πιο δύσκολο να βγάλει κανείς σωστά συμπεράσματα για τις επιδόσεις διαφόρων υπολογιστικών συστημάτων απλά με το να επισημαίνει τα τεχνικά τους χαρακτηριστικά. Γι'αυτόν ακριβώς το λόγο αναπτύχθηκαν κάποια τεστ που επέτρεπαν στους κατασκευαστές να συγκρίνουν διαφορετικές αρχιτεκτονικές. Για παράδειγμα, οι επεξεργαστές της οικογένειας Pentium 4 της Intel κατά γενική ομολογία λειτουργούσαν σε υψηλότερες συχνότητες απ'ότι οι Athlon XP της AMD, χωρίς αυτό να σημαίνει ότι οι πρώτοι υπερτερούσαν σε υπολογιστική ισχύ σε σχέση με τους δεύτερους δίχως αμφισβήτηση. Ένας πιο αργός επεξεργαστής-ως αναφορά στη συχνότητα ρολογιού-είναι εφικτό να αποδίδει εξίσου καλά μ'έναν επεξεργαστή που λειτουργεί σε υψηλότερες συχνότητες.

Τα benchmarks είναι σχεδιασμένα για να προσομοιώνουν ένα ιδιαίτερο είδος "φορτίου"(επομένως διεργασία/διεργασίες στη γλώσσα της Πληροφορικής) πάνω σ'ένα στοιχείο ή ένα σύστημα. Τα benchmarks-εφαρμογές τρέχουν πραγματικά προγράμματα και δίνουν μια πιο ολοκληρωμένη εικόνα για ένα δεδομένο σύστημα ενώ τα βιομηχανικά benchmarks είναι χρησιμότερα για τον ποιοτικό έλεγχο μεμονωμένων στοιχείων όπως ένας σκληρός δίσκος ή μια κάρτα ήχου.

Τα benchmarks είναι εξίσου σημαντικά στο σχεδιασμό των επεξεργαστών, παρέχοντας στους σχεδιαστές την ικανότητα να παίρνουν μετρήσεις και να κάνουν εναλλαγές σε αποφάσεις μικροαρχιτεκτονικού σχεδιασμού. Για παράδειγμα, αν ένα benchmark εξάγει τους αλγορίθμους κλειδιών από μία εφαρμογή, θα εμπεριέχει ως παράγωγα τις πληροφορίες που είναι σημαντικές ως προς τις επιδόσεις του. Τρέχοντας αυτό το ακόμα μικρότερο κομμάτι σε έναν προσομοιωτή με ακρίβεια ενός κύκλου ρολογιού, μπορούμε να αντλήσουμε πληροφορίες για το πώς θα αυξήσουμε την απόδοση του.

Πριν το 2000, η αρχιτεκτονική υπολογιστών και μικροεπεξεργαστών χρησιμοποιούσε τα benchmarks της SPEC (τα οποία ήταν βασισμένα στο Unix) : ήταν μακροσκελέστατα και με τεραστίων διαστάσεων ρουτίνες από κώδικα κι επομένως ήταν σχεδόν ακατόρθωτο να τα χρησιμοποιήσει κανείς σχεδόν ανέπαφα. Είναι γνωστό ότι οι κατασκευάστριες εταιρίες υπολογιστών στήνουν με τέτοιο τρόπο τα συστήματά τους έτσι ώστε να αποδίδουν εξωπραγματικά(κάτι που δεν ανταποκρίνεται στην πραγματικότητα). Για παράδειγμα, στη δεκαετία του '80 κάποιοι compilers μπορούσαν να ανιχνεύσουν μια συγκεκριμένη μαθηματική πράξη ευρύτερα διαδεδομένη σ'ένα benchmark κινητής υποδιαστολής και να την αντικαταστήσουν με μία ισοδυνάμως σωστή πράξη που θα έφερνε ταχύτερα αποτελέσματα αυτή τη φορά. Ωστόσο, ένα τέτοιο τέχνασμα μάλλον δεν ήταν χρήσιμο εκτός των benchmarks μέχρι τα μέσα της δεκαετίας του '90, όταν οι αρχιτεκτονικές RISC και VLIW τόνισαν τη σπουδαιότητα της τεχνολογίας των compilers καθώς αυτή αναφερόταν στην απόδοση. Τα benchmarks στις μέρες μας χρησιμοποιούνται συνηθέστατα από εταιρίες μεταγλωττιστών όχι μόνο για να βελτιώσουν την αξιοπιστία τους αυτή καθαυτή, αλλά επιδόσεις πραγματικών εφαρμογών.

Με δεδομένο το μεγάλο αριθμό διαθεσίμων benchmarks, ο κατασκευαστής μπορεί συνήθως να βρίσκει τουλάχιστον ένα benchmark που να υποδεικνύει ότι το σύστημά

του θα υπερβεί σε απόδοση κάποιο άλλο: τα άλλα συστήματα μπορεί να αποδειχθεί ότι υπερέχουν με διαφορετικό benchmark.

Οι κατασκευαστές συνήθως αναφέρουν μόνο εκείνα τα benchmarks (ή παραλλαγές τους) που παρουσιάζουν τα προϊόντα τους όσο πιο ευπαρουσίαστα γίνεται. Επίσης είναι γνωστό ότι δεν παρουσιάζουν σωστά τη σπουδαιότητα των benchmarks πάλι για τον ίδιο λόγο. Συνοψίζοντας, αυτές οι πρακτικές ονομάζονται bench-marketing.

Τα ιδανικά benchmarks θα έπρεπε απλά να υποκαθιστούν τις πραγματικές εφαρμογές (σε περίπτωση που η εφαρμογή δεν είναι διαθέσιμη), ή πάρα πολύ δύσκολη ή δαπανηρή(ως προς την εφαρμογή της πάνω σ'ένα συγκεκριμένο επεξεργαστή ή υπολογιστή). Αν η απόδοση είναι βαρύνουσας σημασίας, το μόνο benchmark που μετράει είναι για το σκόπιμο φορτίο.

[5.3] Τύποι benchmarks

1. Πραγματικού προγράμματος
   - Λογισμικό επεξεργασίας κειμένου
   - Λογισμικά εργαλεία απο CDA
   - Λογισμικό εφαρμογής από το χρήστη (MIS)

2. Kernel (πυρίνα συστήματος)
   - Περιέχει τους βασικούς κώδικες
   - Popular kernel: Βρόγχος Livermore
   - Linpack benchmark (Περιέχει τη βασική γραμμική υπορουτίνα άλγεβρας που γράφεται σε γλώσσα FORTRAN)
   - Τα αποτελέσματα προβάλλονται σε MFLOPS

3. Benchmark βασικών εξαρτημάτων / micro-benchmark
   - Προγράμματα σχεδιασμένα να μετρούν τις επιδόσεις των βασικών τμημάτων που απαρτίζουν τον υπολογιστή
   - αυτόματη ανίχνευση των hardware παραμέτρων του υπολογιστή, όπως το πλήθος των καταχωρητών, το μέγεθος της cache μνήμης

4. Συνθετικό Benchmark
   - Διαδικασία για τον προγραμματισμό συνθετικού Benchmark
     - Λαμβάνει όλα τα στατιστικά στοιχεία του τύπου των πράξεων από το πλήθος των προγραμμάτων εφαρμογής
     - Καταγράφει σε ένα μέρος για κάθε πράξη που γίνεται
     - Γράφεται ένα πρόγραμμα που βασίζεται στα παραπάνω
   - Οι τύποι των συνθετικών Benchmark είναι :
     - Whetstone
     - Dhrystone
   - Αυτό ήταν το πρώτο γενικού σκοπού bencmark που βγήκε για προσωπικούς υπολογιστές παραγωγής. Δεν χρησιμοποιείται πλέον για σύγχρονους υπολογιστές.

5. I/O benchmarks
   - Παράλληλο Benchmark : Χρησιμοποιείται σε μηχανές με πολλαπλούς υπολογιστές ή σε συστήματα που κάνουν χρήση πολλαπλών μηχανών.

[5.4] Ποιες είναι οι επιδόσεις που μπορεί να πετύχει ένας MicroBlaze επεξεργαστής

Πόσα Dhrystone MIPS μπορεί να πετύχει στο συνθετικό Benchmark

Οι επιδόσεις του MicroBlaze εξαρτώνται από τη σύνθεση του επεξεργαστή, καθώς και την αρχιτεκτονική και την ταχύτητα της FPGA στην οποία πρόκειται να υλοποιηθεί. Όλοι αυτοί οι παράγοντες επηρεάζουν το F_{max} (μέγιστη ταχύτητα ρολογιού) που μπορεί να επιτευχθεί από το σχεδιασμό του ενσωματωμένου επεξεργαστή. Τα Dhrystone MIPS (DMIPS) σχετίζονται άμεσα με το F_{max} του επεξεργαστή και τα DMIPS/MHz είναι ένας αριθμός που συχνά σχετίζεται με τους ενσωματωμένους επεξεργαστές.

Στην περίπτωση του MicroBlaze, έχουμε μια έκδοση για pipeline 3 σταδίων και μια 5 έκδοση σταδίων και τα στατιστικά τους έχουν ως εξής :

3-stage pipeline	0.95 DMIPS/MHz
5-stage pipeline	1.19 DMIPS/MHz

Πόσους MicroBlaze επεξεργαστές μπορεί να υποστηρίξει μια FPGA

Ο αριθμός των MicroBlaze επεξεργαστών που μπορεί να υποστηρίξει μια FPGA εξαρτάται μόνο από το μέγεθος της FPGA, ενώ το MDM (MicroBlaze Debug Module) επιτρέπει debugging έως και για 8 MicroBlaze ταυτοχρόνως.

ΚΕΦΑΛΑΙΟ ΕΚΤΟ

ΑΝΑΠΤΥΞΗ ΚΩΔΙΚΑ ΚΑΙ ΤΡΟΠΟΣ ΕΦΑΡΜΟΓΗΣ ΤΟΥ ΓΙΑ ΔΟΚΙΜΗ ΚΑΙ ΕΛΕΓΧΟ ΕΠΙΔΟΣΕΩΝ

[6.1] Κώδικας

Οι αλγόριθμοι που χρησιμοποιήθηκαν είναι γραμμένοι σε γλώσσα C, σε συνδυασμό με κάποιες εντολές της Xilinx που χρησιμεύουν στον έλεγχο των επιδόσεων.

Για τις δοκιμές μας χρησιμοποιούμε τους εξής αλγορίθμους :

- Ταξινόμησης
- επεξεργασίας εικόνας
- αναζήτησης
- μαθηματικών συναρτήσεων

Το πλήθος των αλγόριθμος που χρησιμοποιήσαμε είναι 12 στο σύνολο, ώστε να πετύχουμε αξιοπιστία στα αποτελέσματα μας. Κυρίως θέλαμε να δούμε μέχρι που φτάνουν οι δυνατότητες αυτού του επεξεργαστή, δοκιμάζοντας σύνθετους και απλούς κώδικες.

Όλοι αυτοί οι αλγόριθμοι, έχουν κάποια κοινά στοιχεία.

I. βιβλιοθήκες της Xilinx
   Γίνεται χρήση της ”xtmrctr.h” και της “xparameters.h” για να χρησιμοποιήσουμε κάποιες από τις συναρτήσεις και κάποιες εντολές που μας προσφέρει η Xilinx στους κώδικες μας.

II. Συναρτήσεις
   Χρησιμοποιούμε τις παρακάτω συναρτήσεις σε όλους τους κώδικες : “XTmrCtr_Initialize()” , “XTmrCtr_SetResetValue()” , “XTmrCtr_Reset()” , “XTmrCtr_Start()” , “XTmrCtr_Stop()” , ”XTmrCtr_GetValue()”. Αυτές οι συναρτήσεις μας χρησιμεύουν στο να πάρουμε χρονικές μετρήσεις, ώστε να καταγράψουμε το χρόνο που χρειάζεται για να ολοκληρωθεί η εκτέλεση του προγράμματος.

6.1.1 Περιγραφή των Xilinx Βιλιοθηκών

Η βιβλιοθήκη “xtmrctr.h” εμπεριέχει όλες τις απαραίτητες συναρτήσεις της Xilinx για χρονισμούς και μετρήσεις (όπως εισαγωγή delay, χρονικά interupts, μέτρηση παλμών με pwm).

Η βιβλιοθήκη “xparameters.h” είναι η ποιο βασική βιβλιοθήκη της Xilinx. Περιέχει όλες τις σταθερές εκείνες οι οποίες καθορίζουν από ποιά διεύθυνση ξεκινάει το κάθε module (όπως ο επεξεργαστής μας, η μνήμη μας, τα Leds και τα switches). Οδηγεί τις περιφεριακές συσκευές που χρησιμοποιουμε στο board σύμφωνα με τον κώδικα που εμείς γράφουμε.

6.1.2 Περιγραφή των Xilinx συναρτήσεων

- XTmrCtr_Initialize(&XPS_Timer, XPAR_XPS_TIMER_0_DEVICE_ID) : κάνει αρχικοποίηση όλων των ορισμάτων που απαιτούνται για μια χρονική μέτριση
- XTmrCtr_SetResetValue(&XPS_Timer, 0, 0x00000000) : μηδενίζουμε τον καταχωρητή στον οποίο αποθηκεύεται η χρονική μέτρηση. Η εντολή περιγράφει τα εξής : με σχετική διευθυνσιοδότηση GOTO όσες θέσεις λέει το τρίτο όρισμα, και καταχώρησε τη τιμή του δευτέρου ορίσματος (δηλαδή 0).
- XTmrCtr_Reset(&XPS_Timer, 0) : μηδενίζει το χρονόμετρο και θέτει τους κύκλους του ρολογιού σε ένα T_0.
- XTmrCtr_Start(&XPS_Timer, 0) : ξεκινάμε τη χρονομέτρηση σε κύκλους ρολογιού
- XTmrCtr_Stop(&XPS_Timer, 0) : σταματάμε τη τρέχουσα χρονομέτρηση
- XTmrCtr_GetValue(&XPS_Timer, 0) : μας δίνει τη τιμή που αποθηκεύτηκε τελευταία στο καταχωρητή

- microblaze_init_icache_range
  (0,XPAR_MICROBLAZE_0_CACHE_BYTE_SIZE);
  microblaze_enable_icache();
  microblaze_init_dcache_range
  (0,XPAR_MICROBLAZE_0_DCACHE_BYTE_SIZE);
  microblaze_enable_dcache();
  Το παραπάνω block κώδικα, κάνει αρχικοποίηση καθώς και ενεργοποίηση των ICACHE και DCACHE μνημών, όταν αυτό είναι αναγκαίο (για το benchmarking με χρήση CACHE μνήμης)

  (ενδεικτική χρήση της μέτρησης CACHE μνήμης σε κώδικα βρίσκεται στο παράρτημα Γ.13, σελίδα 119)

[6.2] Περιγραφή των αλγορίθμων που χρησιμοποιήθηκαν

Αλγόριθμος αναζήτησης (anazhthsht.c) (παράρτημα Γ.1, σελίδα 90)

Στον κώδικα αυτόν, γεμίζουμε τον πίνακα pin 100 στοιχείων, με χρήση του βρόγχου for. Στη συνέχεια κάνουμε αναζήτηση για το στοιχείο εκείνο του πίνακα, το οποίο περιέχει τον αριθμό 33.

Αλγόριθμος συνάρτησης υπερβολής (Hyperbolict.c) (παράρτημα Γ.2, σελίδα 91)

Σε αυτόν τον κώδικα ξεκινάμε χρησιμοποιώντας τη μεταβλητή val με αρχική τιμή -1, και βρίσκουμε όλα τα σημεία της εφαπτομένης μιας συνάρτησης υπερβολής μέχρι η μεταβλητή να έχει τιμή 1, με βήμα +0,1.

Αλγόριθμος εφαπτόμενων (tantantanR1t.c) (παράρτημα Γ.3, σελίδα 92)

Σε αυτόν το κώδικα ξεκινάμε χρησιμοποιώντας τη μεταβλητή val με αρχική τιμή -1 και βήμα +0.1, υπολογίζουμε το τόξο εφαπτομένης (atan) της τρέχουσας τιμής της val, καθώς και τη τιμή της παραλλαγής του τόξου εφαπτομένης (atan2) που έχει να κάνει με ακτίνια. Οι υπολογισμοί συνεχίζονται μέχρις ότου η μεταβλητή val πάρει τιμή +1.

Αλγόριθμος εφαπτόμενων (tantantan1t.c) (παράρτημα Γ.4, σελίδα 93)

Στον κώδικα αυτόν υπολογίζουμε το τόξο εφαπτομένης μιας μεταβλητής v με αρχική τιμή 5 και το μετατρέπουμε σε μοίρες. Επίσης υπολογίζουμε τη παραλλαγή του τόξου εφαπτομένης (atan2) δύο μεταβλητών (χ , y) και το μετατρέπουμε σε μοίρες.

Αλγόριθμος ύψωσης σε δύναμη (randPowers.c) (παράρτημα Γ.5, σελίδα 94)

Στον κώδικα αυτόν γίνεται επιλογή ενός τυχαίου αριθμού, τον οποίο τον μετατρέπουμε σε ακέραιο. Στην συνέχεια βρίσκουμε τις δυνάμεις αυτού του αριθμού υψωμένο στο τετράγωνο, υψωμένο στο 12 και υψωμένο στο 1.54. Η διαδικασία αυτή επαναλαμβάνεται για 30 φορές.

Αλγόριθμος του Sobel (sobel.c) (παράρτημα Γ.6, σελίδα 95)

Χρησιμοποιείται στην επεξεργασία εικόνας, και πιο συγκεκριμένα στους αλγορίθμους ανίχνευσης ακμών. Στον κώδικα αυτόν κάνουμε επεξεργασία μιας εικόνας 7x16 με βάση τον αλγόριθμο του Sobel, εφαρμόζοντας μια μάσκα που προκαλεί όξυνση της εικόνας.

Η συνάρτηση υπολογίζει την κλίση της έντασης της εικόνας σε κάθε σημείο, δίνοντάς μας έτσι την κατεύθυνση της μεγαλύτερης πιθανής αύξησης από τα φωτεινά στα σκοτεινά καθώς και το ρυθμό μεταβολής προς αυτή την κατεύθυνση. Ως αποτέλεσμα παίρνουμε το πόσο απότομα ή ομαλά αλλάζει η εικόνα μας ως προς αυτό το σημείο, οπότε και το κατά πόσο είναι πιθανό αυτό το σημείο να αντιπροσωπεύει μια ακμή. Στην πράξη, μπορούμε να ερμηνεύσουμε πιο αξιόπιστα αποτελέσματα και με μεγαλύτερη ευκολία προσπαθώντας να υπολογίσουμε την πιθανότητα μιας ακμής παρά προσπαθώντας να υπολογίσουμε την κατεύθυνση της ακμής (και πάλι ως ακμή αντιλαμβανόμαστε το ρυθμό της μεταβολής προς μια περιοχή της εικόνας).

Αλγόριθμος δρομολόγησης Dijkstra (Dijkstra1t.c) (παράρτημα Γ.7, σελίδα 101)

Ο Dijkstra είναι ένας αλγόριθμος δρομολόγησης που χρησιμοποιείται στα δίκτυα.

Για κάθε ένα από τους δεδομένους κόμβους σε ένα γράφημα, ο αλγόριθμος βρίσκει το μονοπάτι με το χαμηλότερο κόστος (δηλαδή τη συντομότερη διαδρομή) μεταξύ της εν λόγω κορυφής και κάθε άλλης κορυφής. Μπορεί επίσης να χρησιμοποιηθεί για την εύρεση του κόστους της συντομότερης διαδρομής από μια μόνο κορυφή προς μία μόνο κορυφή προορισμού σταματώντας τον αλγόριθμο, από τη στιγμή που έχει καθοριστεί η συντομότερη διαδρομή προς την κορυφή του προορισμού. Στον κώδικα μας τον εφαρμόζουμε για να βρούμε το συντομότερο μονοπάτι ανάμεσα σε 100 κόμβους.

Αλγόριθμος ταξινόμησης της φυσαλίδας (bublesort1t.c) (παράρτημα Γ.8, σελίδα 104)

Στον κώδικα αυτόν καλείται η συνάρτηση Fill_array(), για να γεμίσει ένα μονοδιάστατο πίνακα 100 στοιχείων. Στη συνέχεια καλείται η συνάρτηση ArraySort(my_array, cmpfun, ARRAY_SIZE), η οποία κάνει την ταξινόμηση της φυσαλίδας.
Ο αλγόριθμος ταξινόμησης bubble sort λειτουργεί συγκρίνοντας κάθε στοιχείο με το επόμενό του, ξεκινώντας από την μία άκρη του πίνακα μέχρι την άλλη. Γενικά για να ταξινομηθεί ο πίνακας χρειάζονται N-1 «περάσματα» του πίνακα, όπου N το πλήθος των στοιχείων. Η διαδικασία αυτή επαναλαμβάνεται για 80000 φορές.
Είναι η πιο γνωστή τεχνική ταξινόμησης. Το όνομα οφείλεται στο ότι σε διαδοχικές επαναλήψεις του πίνακα οι μικρότερες τιμές αναδύονται σαν φυσαλίδες σε κατάλληλες θέσεις.

Αλγόριθμος ταξινόμησης με συγχώνευση (mergeSort1t.c) (παράρτημα Γ.9, σελίδα 106)

Στον κώδικα αυτόν καλείται η συνάρτηση Fill_array(), για να γεμίσει ένα μονοδιάστατο πίνακα 100 στοιχείων. Στη συνέχεια καλείται η συνάρτηση ArraySort(my_array, cmpfun, ARRAY_SIZE), η οποία κάνει την ταξινόμηση με συγχώνευση. Χρησιμοποιούμε δύο δείκτες, ένα για κάθε πίνακα, και ένα τρίτο πίνακα για την αποθήκευση της συγχώνευσης των δύο αρχικών πινάκων. Συγκρίνουμε τις δύο τιμές των πινάκων εισόδου που δείχνουν οι δείκτες και αντιγράφουμε την μικρότερη τιμή στην κατάλληλη θέση του πίνακα εξόδου.

Η ταξινόμηση με συγχώνευση ενός πίνακα υλοποιείται χρησιμοποιώντας επαναλαμβανόμενες συγχωνεύσεις με αναδρομική κλήση της διαδικασίας συγχώνευσης ως εξής: χωρίζουμε τον πίνακα σε δύο μέρη και ταξινομούμε πρώτα το αριστερό μισό μέρος του πίνακα και κατόπιν το δεξιό. Ακολουθεί η συγχώνευση των δύο πινάκων. Επαναλαμβάνουμε την διαδικασία αναδρομικά για τα δύο μέρη του πίνακα μέχρι το μήκος των υπό-πινάκων οι οποίοι προκύπτουν να είναι μικρότερο ή ίσο του ένα.

Αλγόριθμος γρήγορης ταξινόμησης (quickSort1t.c) (παράρτημα Γ.10, σελίδα 110)

Στον κώδικα αυτόν καλείται η συνάρτηση Fill_array(), για να γεμίσει ένα μονοδιάστατο πίνακα 100 στοιχείων. Στη συνέχεια καλείται η συνάρτηση ArraySort(my_array, cmpfun, ARRAY_SIZE), η οποία κάνει την γρήγορη ταξινόμηση. Βασική αρχή είναι ότι προσπαθούμε να χωρίσουμε τα στοιχεία του πίνακα, σε δύο υπό-πίνακες έτσι ώστε ο ένας να περιέχει όλα τα στοιχεία μικρότερα από κάποια τιμή, ενώ ο άλλος όλα τα μεγαλύτερα από αυτή την τιμή. Αυτή η τιμή μπορεί να είναι οποιαδήποτε τιμή του πίνακα. Στην γρήγορη ταξινόμηση η τιμή αυτή ονομάζεται κεντρική. Το πρώτο βήμα είναι να χωρίσουμε τον πίνακα σε δύο υπό-πίνακες. Η διαδικασία αυτή επαναλαμβάνεται για 80000 φορές.

Όπως είπαμε ένα βασικό χαρακτηριστικό των δύο πινάκων είναι ότι κάθε στοιχείο του αριστερού πίνακα είναι μικρότερο από κάθε στοιχείο του δεξιού πίνακα.

Αλγόριθμος ταξινόμησης με επιλογή (SelectionSort1t.c) (παράρτημα Γ.11, σελίδα 114)

Στον κώδικα αυτόν καλείται η συνάρτηση Fill_array(), για να γεμίσει ένα μονοδιάστατο πίνακα 100 στοιχείων. Στη συνέχεια καλείται η συνάρτηση ArraySort(my_array, cmpfun, ARRAY_SIZE), η οποία κάνει ταξινόμηση με επιλογή. Η ταξινόμηση με επιλογή λειτουργεί τοποθετώντας κάθε τιμή (μία κάθε φορά) στην σωστή, τελική και ταξινομημένη θέση. Με άλλα λόγια, για κάθε θέση του πίνακα ο αλγόριθμος επιλέγει την τιμή η οποία θα πρέπει να πάει σε εκείνη την θέση.

Ο αλγόριθμος γενικά λειτουργεί ως εξής: Βρίσκουμε την μικρότερη τιμή του πίνακα και την ανταλλάσσουμε με την τιμή στην πρώτη θέση του πίνακα. Μετά βρίσκουμε την μικρότερη τιμή από τις υπόλοιπες (εκτός της πρώτης) και την ανταλλάσουμε με την δεύτερη θέση του πίνακα. Η διαδικασία συνεχίζεται για κάθε θέση του πίνακα.

Αλγόριθμος ταξινόμησης με εισαγωγή (insertSort1t.c) (παράρτημα Γ.12, σελίδα 116)

Στον κώδικα αυτόν καλείται η συνάρτηση Fill_array(), για να γεμίσει ένα μονοδιάστατο πίνακα 100 στοιχείων. Στη συνέχεια καλείται η συνάρτηση ArraySort(my_array, cmpfun, ARRAY_SIZE), η οποία κάνει ταξινόμηση με εισαγωγή. Η ταξινόμηση με εισαγωγή λειτουργεί ως εξής: Παίρνουμε κάθε στοιχείο και το τοποθετούμε στη σωστή θέση στον ταξινομημένο πίνακα αριστερά του τρέχοντος στοιχείου. Εάν ξεκινήσουμε από το πρώτο στοιχείο, τότε αφού δεν υπάρχουν άλλα στοιχεία αριστερά, το στοιχείο αυτό βρίσκεται στη σωστή θέση (μέχρι τώρα). Επομένως δεν χρειάζεται να ξεκινήσουμε από το πρώτο στοιχείο.

[6.3] Μετρήσεις

Όλες οι δοκιμές έγιναν με τον MicroBlaze επεξεργαστή που συνθέσαμε στο 4ο Κεφάλαιο. Κρατώντας σταθερά τα χαρακτηριστικά του συστήματος, με επεξεργαστή συχνότητας 100 MHz, χρησιμοποιούμε μια από τις τρείς παρακάτω διαφορετικές προσεγγίσεις που έχουν να κάνουν κυρίως με τη προσωρινή αποθήκευση.

Ποιο συγκεκριμένα :

I. Χρήση της εξωτερικής DDR_SDRAM μεγέθους 64M x 32 (2 DIMMs των 128MB, σύνολο 256MB)

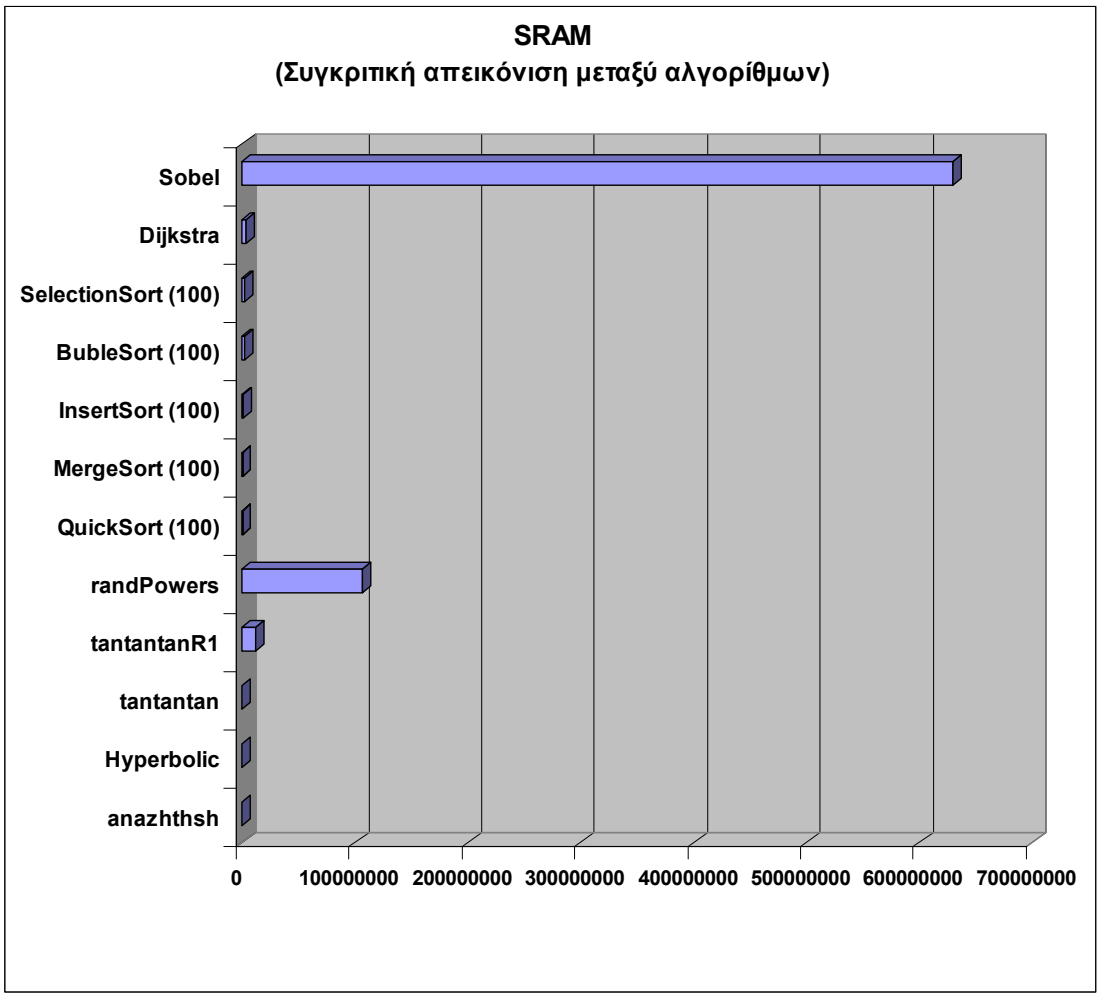
II. Χρήση της εσωτερικής SRAM μεγέθους 10MB

III. Χρήση της on-chip Cache μνήμης του επεξεργαστή μεγέθους 4KB συνολικά

Compiled -O2						
	SRAM		**DDR**		**DDR with cache 1KB (XCL links), DATA & INSTRUCTION CACHE**	
	Min	Max	Min	Max	Min	Max
anazhthsh	18156		27425		17406	
Hyperbolic	2595		4030		528	
tantantan	19792		29918		8416	
tantantanR1	12245280		18684059		3465790	
randPowers	106467996		164848363		39957983	
QuickSort (100)	332050	350727	488119	526381	19021	22413
MergeSort (100)	392905	426687	577925	604261	126370	134316
InsertSort (100)	1299468	1542975	1905407	2304629	570630	662156
BubleSort (100)	2334269	2420459	3325438	3655350	112462	120766
SelectionSort (100)	2369890	2377250	3746976	3759146	996148	997294
Dijkstra	3869081		3876622		3854920	
Sobel	630612775		966829054		255090	

6.3.1 Συγκριτικά γραφήματα

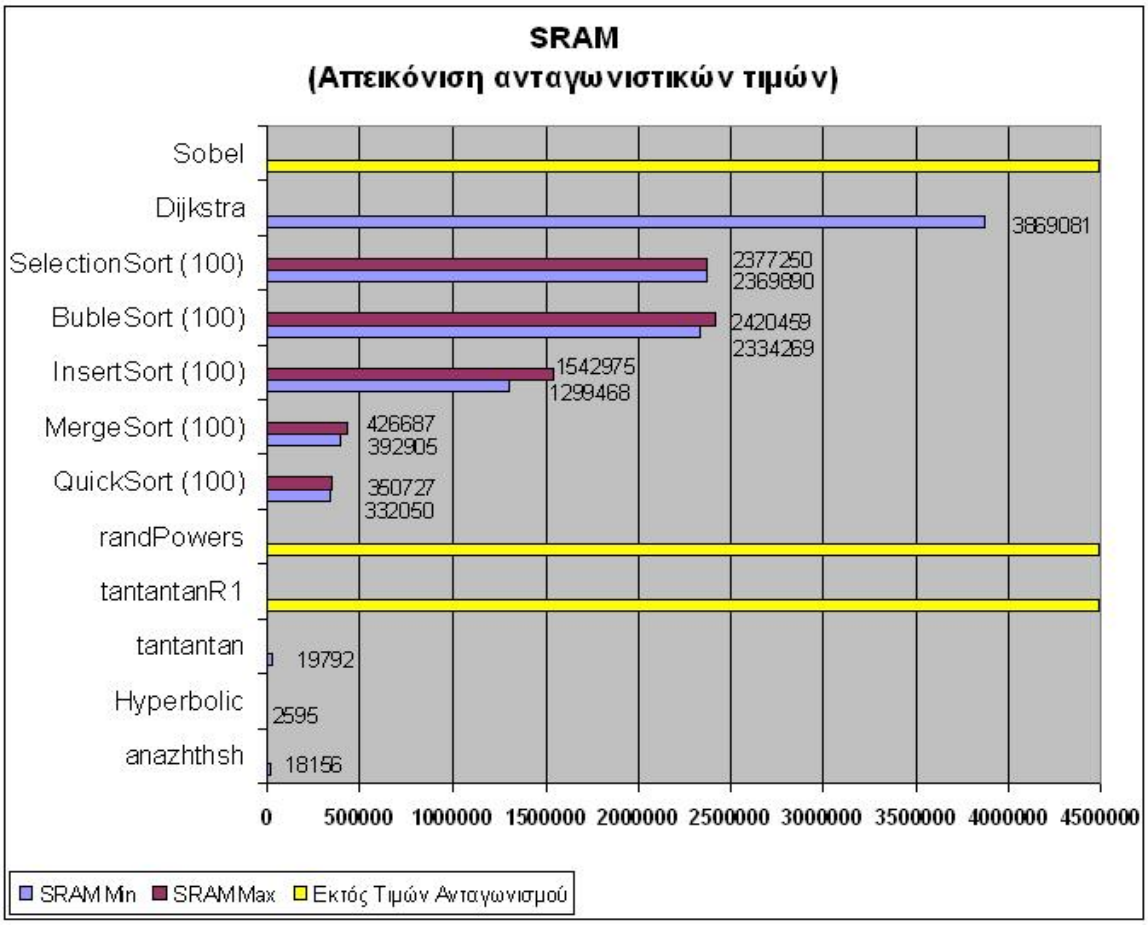
1η συγκριτική δοκιμή – μνήμη SRAM

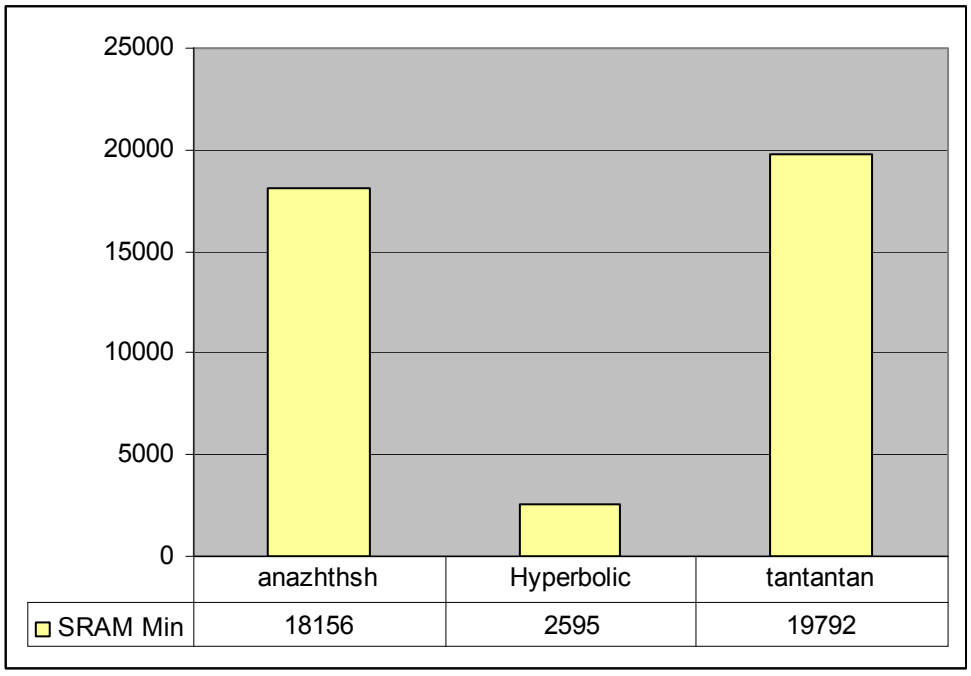


Στο παραπάνω γράφημα φαίνονται οι μετρήσεις που πραγματοποιήθηκαν για την SRAM μνήμη. Το γράφημα αυτό βοηθάει στο να γίνουν αντιληπτές οι μεγάλες αποκλίσεις των τιμών που παίρνουμε, από τη διαφορετικότητα των αλγορίθμων που χρησιμοποιήσαμε για αυτή τη μέτρηση.

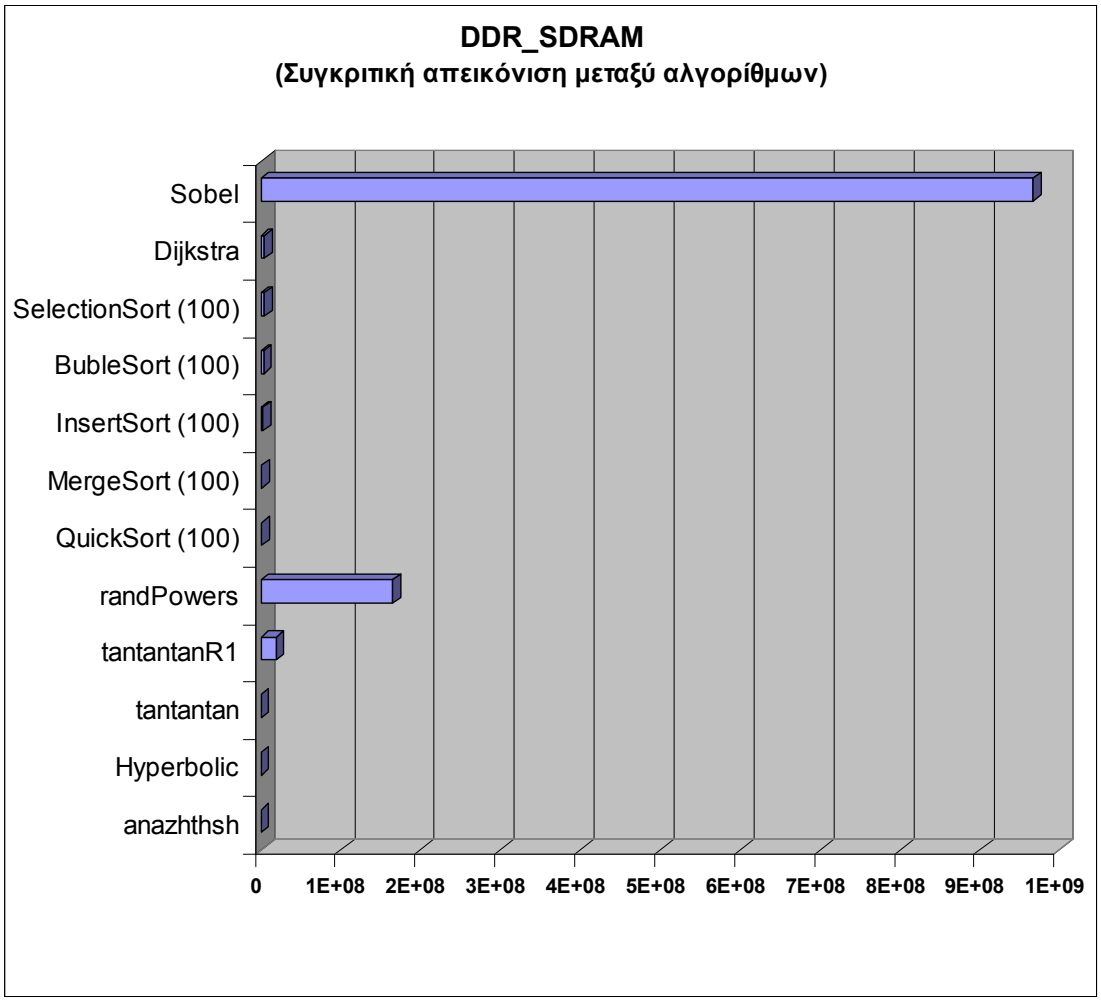
- Ελάχιστη τιμή : 2595 κύκλοι ρολογιού (Αλγόριθμος Hyperbolic)
- Μέγιστη τιμή : 630612775 κύκλοι ρολογιού (αλγόριθμος Sobel)

Στο γράφημα που ακολουθεί, μικρύνουμε τη μέγιστη τιμή του, με αποτέλεσμα να εμφανιστούν οι ανταγωνιστικές τιμές των μετρήσεων που δεν ήταν δυνατό να εμφανιστούν με το παραπάνω γράφημα. Ακολουθεί και γράφημα με ακόμα μικρότερες τιμές για τον ίδιο σκοπό.





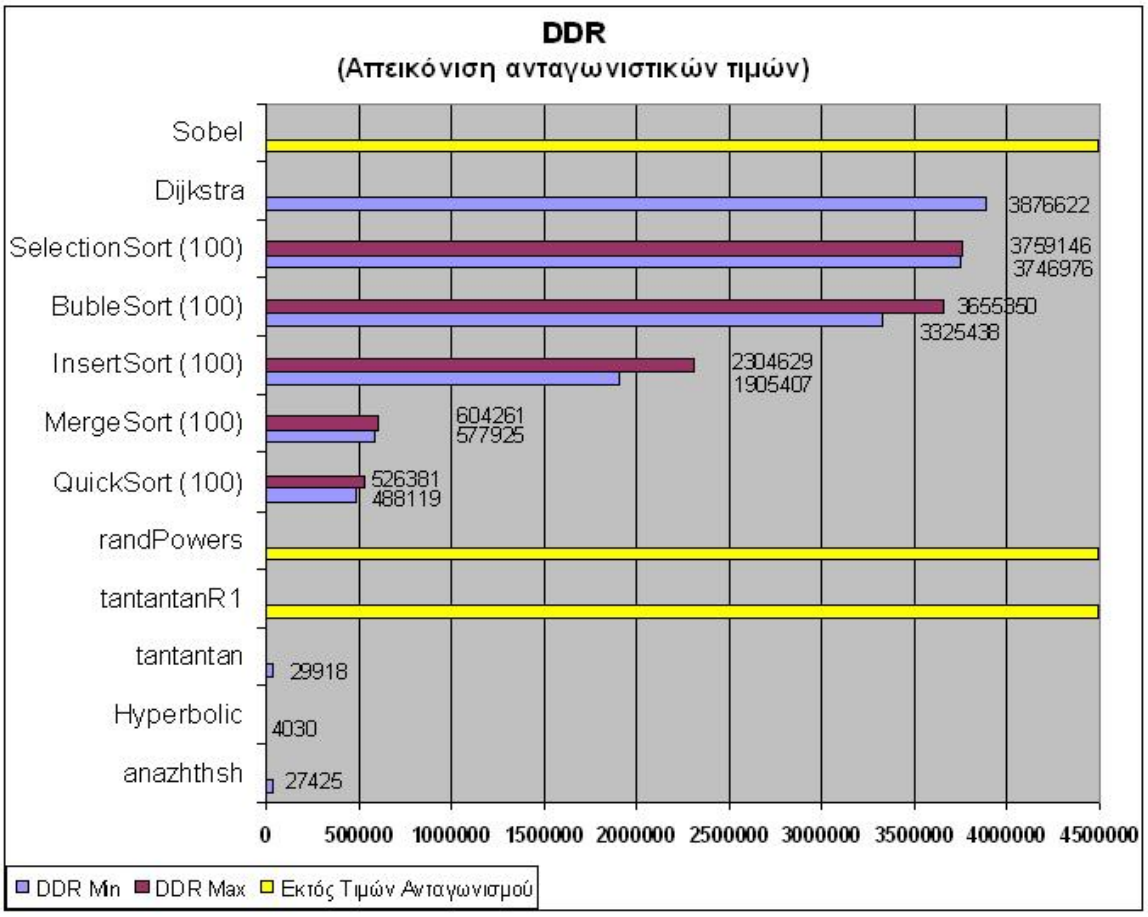
2η συγκριτική δοκιμή – μνήμη DDR_SDRAM

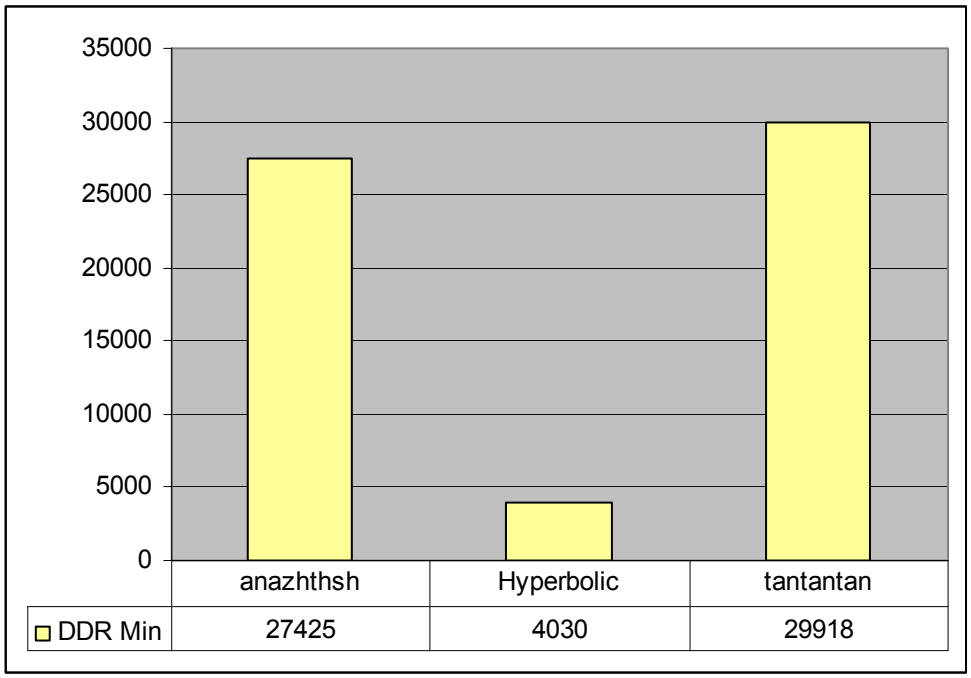


Στο παραπάνω γράφημα φαίνονται οι μετρήσεις που πραγματοποιήθηκαν για την DDR μνήμη. Το γράφημα αυτό βοηθάει στο να γίνουν αντιληπτές οι μεγάλες αποκλίσεις των τιμών που παίρνουμε, από τη διαφορετικότητα των αλγορίθμων που χρησιμοποιήσαμε για αυτή τη μέτρηση.

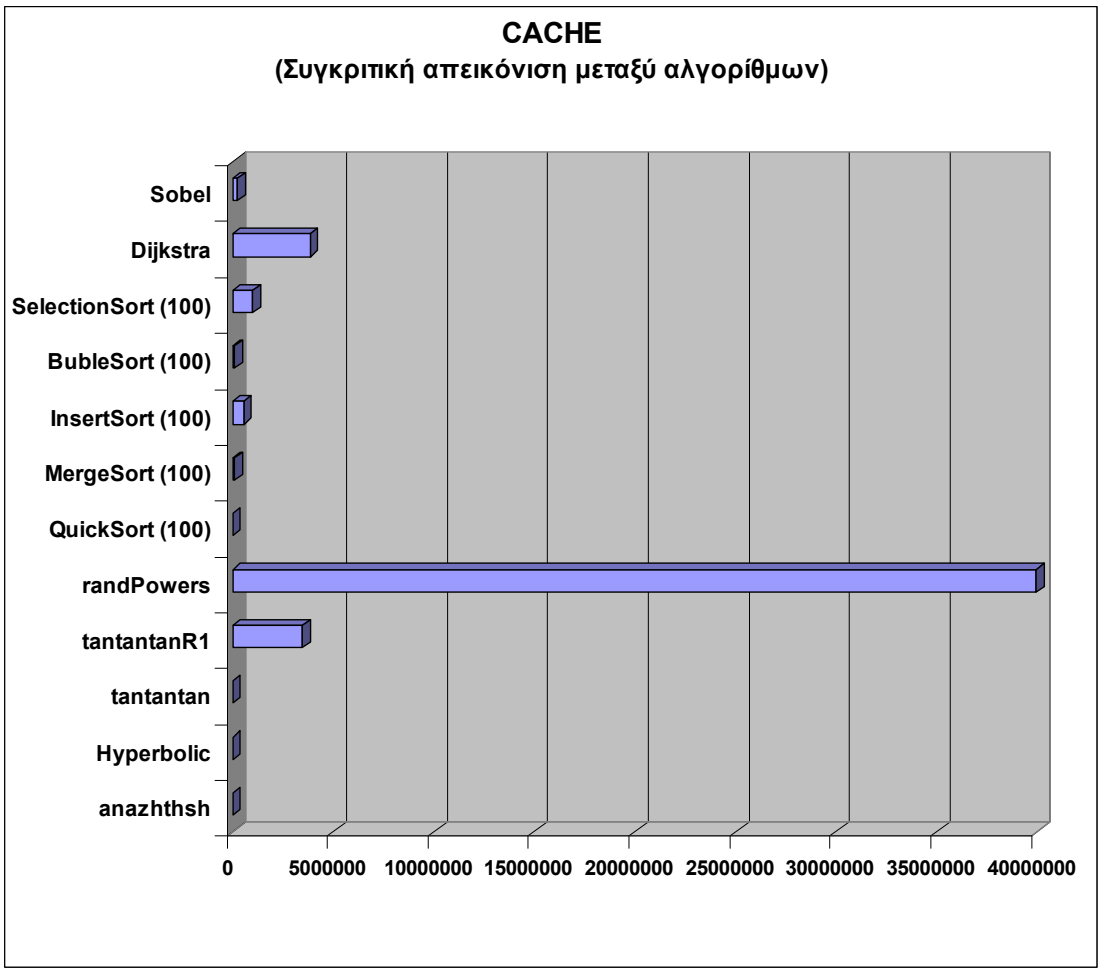
- Ελάχιστη τιμή : 4030 κύκλοι ρολογιού (Αλγόριθμος Hyperbolic)
- Μέγιστη τιμή : 966829054 κύκλοι ρολογιού (αλγόριθμος Sobel)

Στο γράφημα που ακολουθεί, μικρύνουμε τη μέγιστη τιμή του, με αποτέλεσμα να εμφανιστούν οι ανταγωνιστικές τιμές των μετρήσεων που δεν ήταν δυνατό να εμφανιστούν με το παραπάνω γράφημα. Ακολουθεί και γράφημα με ακόμα μικρότερες τιμές για τον ίδιο σκοπό.





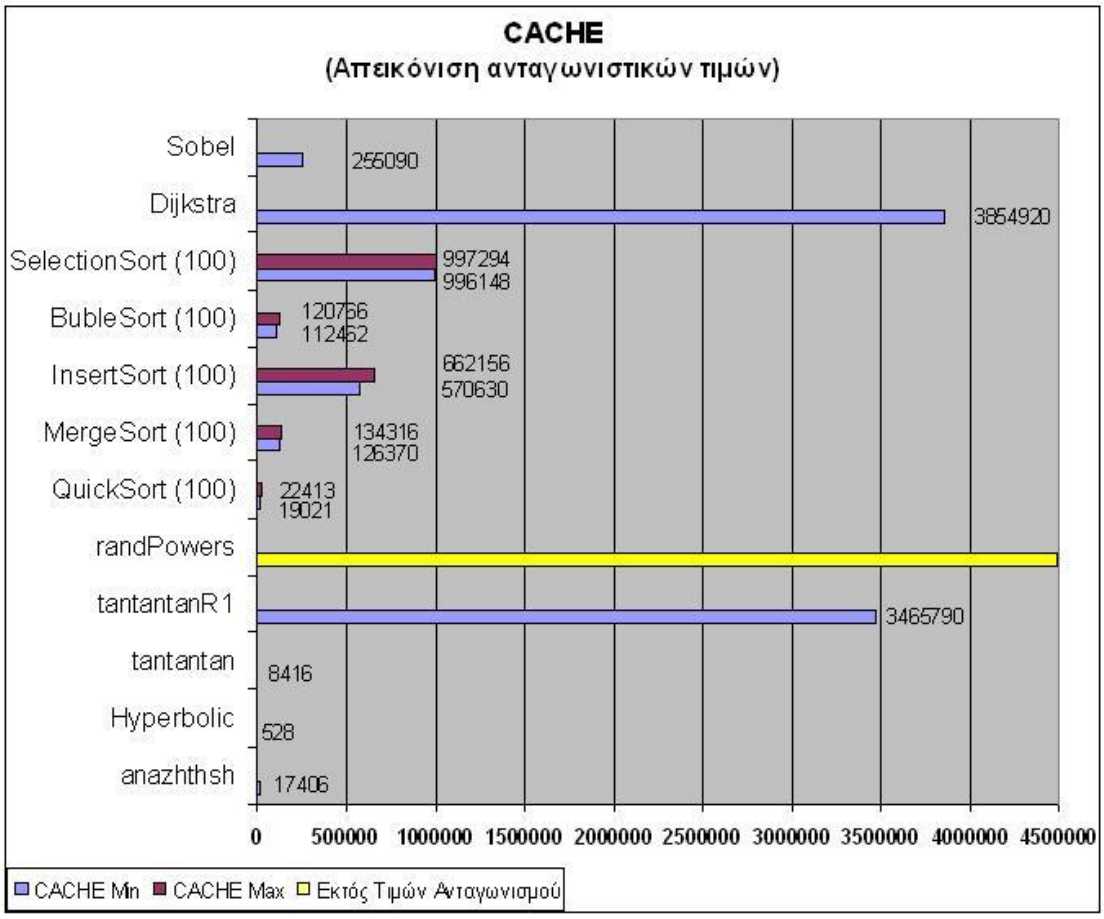
3η συγκριτική δοκιμή – μνήμη CACHE

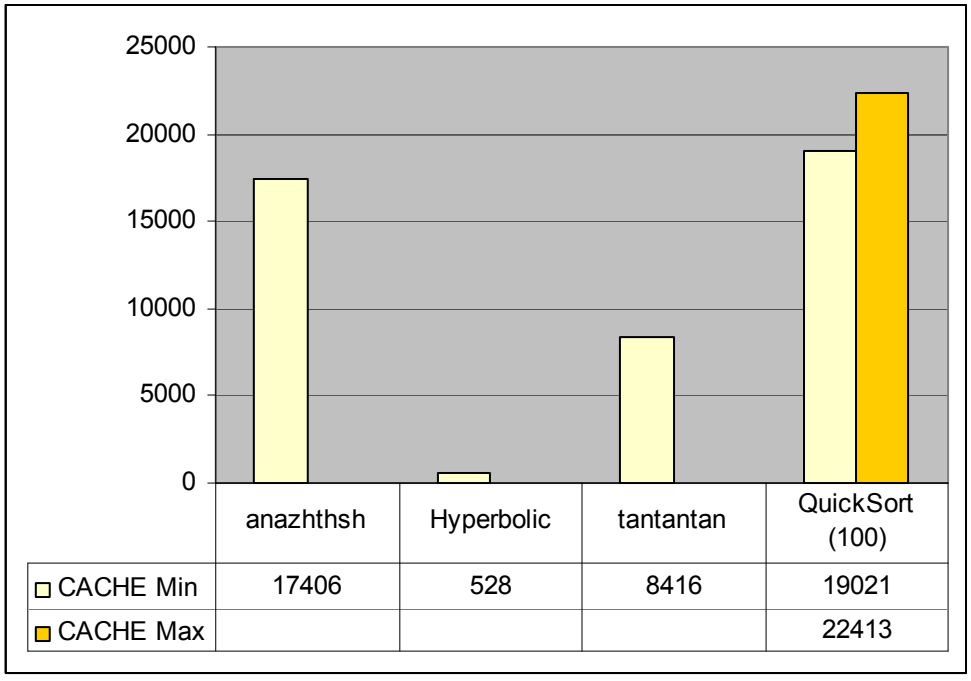


Στο παραπάνω γράφημα φαίνονται οι μετρήσεις που πραγματοποιήθηκαν για την DDR μνήμη. Το γράφημα αυτό βοηθάει στο να γίνουν αντιληπτές οι μεγάλες αποκλίσεις των τιμών που παίρνουμε, από τη διαφορετικότητα των αλγορίθμων που χρησιμοποιήσαμε για αυτή τη μέτρηση.

- Ελάχιστη τιμή : 528 κύκλοι ρολογιού (Αλγόριθμος Hyperbolic)
- Μέγιστη τιμή : 39957983 κύκλοι ρολογιού (αλγόριθμος randPowers)

Στο γράφημα που ακολουθεί, μικρύνουμε τη μέγιστη τιμή του, με αποτέλεσμα να εμφανιστούν οι ανταγωνιστικές τιμές των μετρήσεων που δεν ήταν δυνατό να εμφανιστούν με το παραπάνω γράφημα. Ακολουθεί και γράφημα με ακόμα μικρότερες τιμές για τον ίδιο σκοπό.





[6.4] Συμπεράσματα

Αθροιστικά, για όλους τους αλγορίθμους με μια γρήγορη ματιά βλέπει κανείς ότι όσο πιο κοντά βρίσκεται ο προορισμός, τόσο μεγαλύτερες επιδόσεις επιτυγχάνονται.
Πιο συγκεκριμένα :

I. Η πιο γρήγορη είναι η μνήμη Cache με διαφορά.
II. Δεύτερη έρχεται η μνήμη SRAM
III. και τρίτη η μνήμη DDR_SDRAM.

Σχετικά με τη μνήμη CACHE

Εσωτερικά στην FPGA, ένα μέρος της cache μνήμης χωρίζεται σε ICACHE (instruction Cache ή Cache εντολών), και DCACHE (data Cache ή Cache δεδομένων). Κάθε μια από τις παραπάνω έχει μέγεθος 128 Bytes. Επειδή το μέγεθος της Cache είναι τέτοιο ώστε να μη χωράει τον κώδικα μας, δίνεται η οδηγία από την ICACHE προς την εξωτερική DDR μνήμη (όπου ο κώδικα είναι προσωρινά αποθηκευμένος), να στέλνεται σε πακέτα των 128 Bytes στην Cache μνήμη του MicroBlaze επεξεργαστή της FPGA. Αυτό καθιστά την μνήμη CACHE τη πιο γρήγορη στις μετρήσεις που εκτελέσθηκαν.

Σχετικά με τους αλγόριθμους ταξινόμησης

Στους αλγορίθμους που έχουν να κάνουν με την ταξινόμηση, σαν κριτήριο κρατήσαμε σταθερό το μέγεθος του πίνακα προς ταξινόμηση στα 100 στοιχεία. Επίσης για κάθε προσέγγιση έγινε επανάληψη της κάθε μέτρησης, για τον λόγο ότι κάθε φορά ο πίνακας γεμίζει με μια συνάρτηση τυχαίας επιλογής. Αυτό έχει ως αποτέλεσμα να επηρεάζεται ο χρόνος που χρειάστηκε να γίνει η κάθε ταξινόμηση. Από τις διαφορετικές αυτές μετρήσεις ήταν αναγκαίο να κρατηθεί μια ελάχιστη και μια μέγιστη τιμή με σκοπό να εστιάσουμε στο μέγεθος των αποκλίσεων για κάθε αλγόριθμο.

Παρατηρούμε πως :

I. Ο ποιο γρήγορος από τους αλγορίθμους ταξινόμησης είναι ο QuickSort
II. Ο πιο αργός είναι ο SelectionSort
III. Ο πιο αξιόπιστος (αυτός με τις μικρότερες αποκλίσεις μεταξύ μέγιστης από ελάχιστης τιμής μέτρησης με χρήση τυχαίων αριθμών) είναι ο SelectionSort με απόκλιση 0,3 %
IV. Ο πιο αναξιόπιστος είναι ο InsertSort με απόκληση 18,7 %

Σχετικά με τους αλγόριθμους μαθηματικών συναρτήσεων

Στους αλγορίθμους που έχουν να κάνουν με μαθηματικές συναρτήσεις παρατηρείται πως όταν ο αριθμός των επαναλήψεων είναι μικρός, τα αποτελέσματα είναι τα αναμενόμενα, ενώ όταν οι επαναλήψεις αυξάνονται δραματικά τότε ο χρόνος εκτέλεσης μεγαλώνει με γεωμετρική πρόοδο. Αυτό παρατηρείται στους δύο διαφορετικούς κώδικες με την ονομασία tantantan & tantantanR1. Ενώ η φυσιολογία των υπολογισμών και των δύο είναι παρεμφερής, αυτός που έχει επιβαρυνθεί με τις 20πλάσιες επαναλήψεις σε σχέση με τον άλλο, έχει διαρκέσει περίπου 620 φορές περισσότερο.

Στον αλγόριθμο του Sobel, παρατηρήθηκε το εξής φαινόμενο :

- Είναι ο πιο αργός αλγόριθμος σε σύγκριση με τους υπόλοιπους, όσον αφορά τις δοκιμές με DDR_SDRAM και SRAM μνήμες.
- Ενώ θα περιμέναμε να είναι και οι τιμές των μετρήσεων για την CACHE αναντιστοιχία μεγαλύτερες απ’αυτές των υπολοίπων benchmarks, παρατηρήσαμε μια δραματική, και μη αναμενόμενη αύξηση στις επιδόσεις.

Αυτό οφείλεται κυρίως στις διαφορές που έχει ένας αλγόριθμος επεξεργασίας εικόνας σε σχέση με όλα τα υπόλοιπα προγράμματα που χρησιμοποιήσαμε : πιο πολύπλοκη δομή σε βαθμό να αξιοποιείται ουσιαστικά η χρήση της CACHE μνήμης (η οποία ως γνωστόν είναι η πιο γρήγορη απ’αυτές που είχαμε στη διάθεσή μας).

ΠΑΡΑΡΤΗΜΑ Α

ΧΡΗΣΗ UNIX ΓΙΑ ΤΗ ΣΥΝΘΕΣΗ ΤΟΥ uCLinux

Αρχικά ανοίγουμε μια κονσόλα σε UNIX περιβάλλον όπου πηγαίνουμε στο directory στο οποίο έχουμε τοποθετήσει το uCLinux. Είναι αναγκαίο να τρέξουμε ένα script με το όνομα “settings.sh” ώστε να σοτάρουμε τις κατάλληλες μεταβλητές περιβάλλοντος ώστε να ενημερώσουμε την κονσόλα για το πώς να φορτώσει τα κατάλληλα εργαλεία που πρόκειται να χρειαστούν, καθώς και τη τοποθεσία στην οποία βρίσκεται το σύστημα που κατασκευάζουμε.

```
$ cd /petalinux
$ source ./settings.sh
```

Α.1 Δημιουργία μιας νέας πλατφόρμας

Εδώ έχουμε να εκτελέσουμε 2 καθήκοντα.

Το πρώτο είναι η δημιουργία μιας πλατφόρμας, μέσω ενός script που μας παρέχει το Petalinux για το κατασκευαστή μας, που στη προκείμενη περίπτωση είναι η Xilinx.

Το δεύτερο είναι ο σχεδιασμός της κατασκευής του συστήματος μας.

Εδώ δηλώνουμε τον κατασκευαστή, τον τύπο της πλατφόρμας και την έκδοση του kernel που πρόκειται να χρησιμοποιήσουμε. Αυτό γίνεται με την παρακάτω εντολή :

$ petalinux-new-platform -v MyVendorName -p MyPlatformName -k 2.6

Όπου MyVendorName βάζουμε Xilinx και όπου MyPlatformName βάζουμε ML403.

```
$ cd /petalinux/software/uclinux-dist
$ petalinux-new-platform –v xilinx –p ML403K9 –k 2.6
New platform for MyVendorName MyPlatformName
successfully created
```

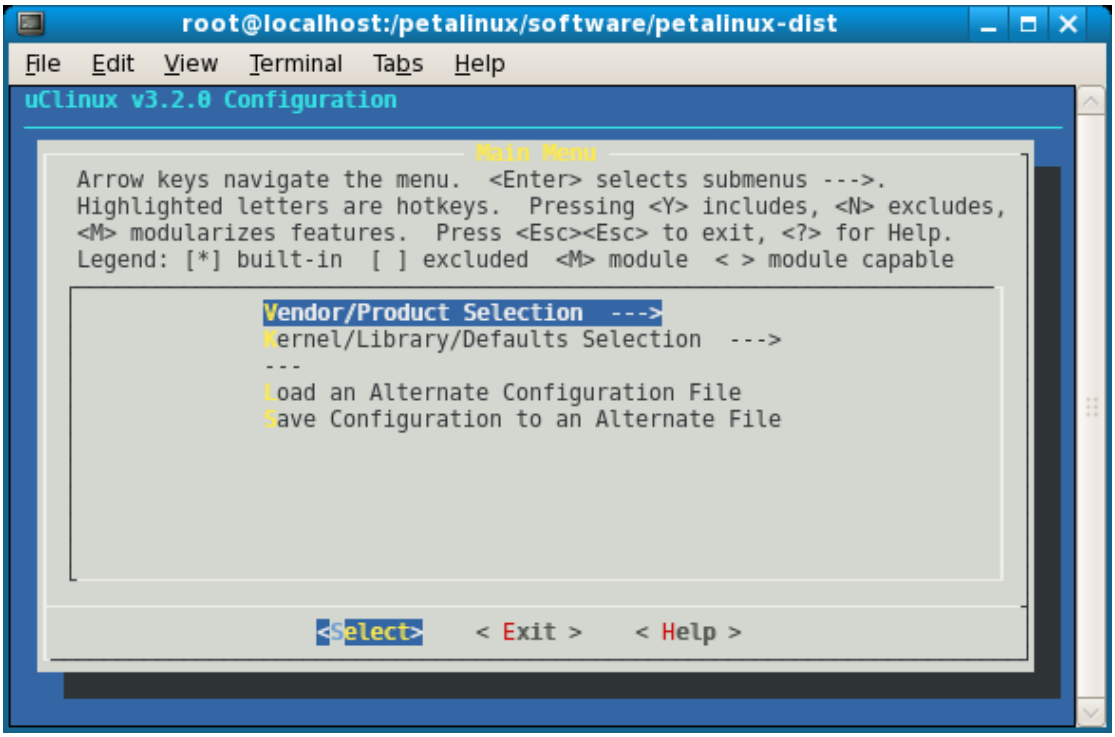
Δημιουργήσαμε την επιλογή της πλατφόρμας στο petalinux-dist με την ονομασία ML403K9 του κατασκευαστή της Xilinx με την έκδοση του 2.6 kernel

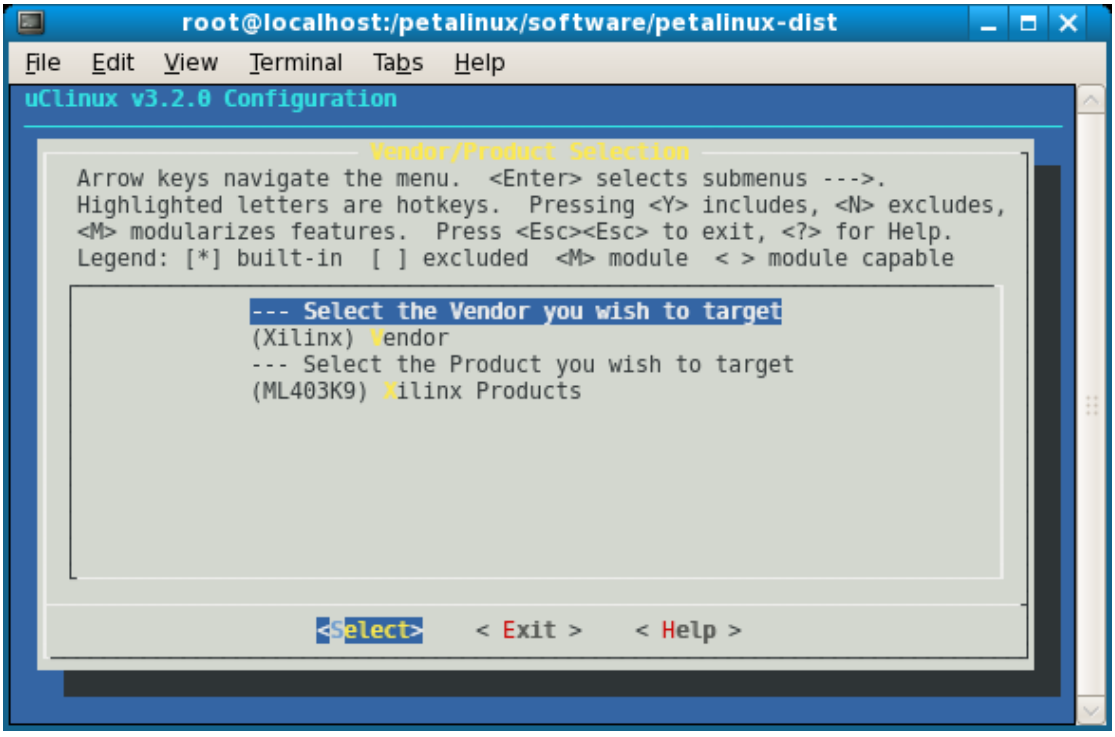
Α.2 Επιλογή κατασκευαστή και πλατφόρμας

Τώρα που τα συστατικά του software και του hardware έχουν προστεθεί στο project του PetaLinux, το επόμενο βήμα είναι η επιλογή του κατασκευαστή και της πλατφόρμας ως το **current build target,** επιτρέποντας μας να κάνουμε τις ρυθμίσεις που χρειάζονται για το χτίσιμο του συστήματος μας. Αυτό γίνεται με τη χρήση του εργαλείου menuconfig, χρησιμοποιόντας τις παρακάτω εντολές :

```
$ cd /petalinux/software/uclinux-dist
$ make menuconfig
```

Στη συνέχεια εμφανίζεται το παρακάτω γραφικό από όπου και κάνουμε τις επιλογή της πλατφόρμας που δημιουργήσαμε σαν επιλογή στο προηγούμενο βήμα





Η αρχική διαμόρφωση που χρησιμοποιούμε για την καινούρια πλατφόρμα είναι η εξορισμού διαμόρφωση και παρέχει στο χρηστή ένα σκελετό για να ξεκινήσει να παραμετροποιεί την πλατφόρμα του, έτσι ώστε να συμπτύξει τη διαδικασία της διαμόρφωσης.

Α.3 Δημιουργία του directory για την τοποθέτηση του hardware project

Φτιάχνουμε τον φάκελο “ML403K9” που θα εμπεριέχει το project στη παρακάτω τοποθεσία

```
$ cd /petalinux/hardware/user-platforms/
$ mkdir ML403K9
```

Στη συνέχεια αντιγράφουμε το “dk_user_repository/bsp” στο φάκελο του project μας “/opt/petalinux/hardware/user-platforms/ML403K9/bsp” και δημιουργούμε ένα link ώστε τα εργαλεία του EDK na μπορούν να έχουν εύκολη πρόσβαση στα αρχεία του BSP.

```
$ cd /petalinux/hardware/user-platforms/Customer-
Project-demo-edk91
$ ln -s ../../edk_user_repository edk_user_repository
```

Α.4 Ρυθμίζοντας το AutoConfig

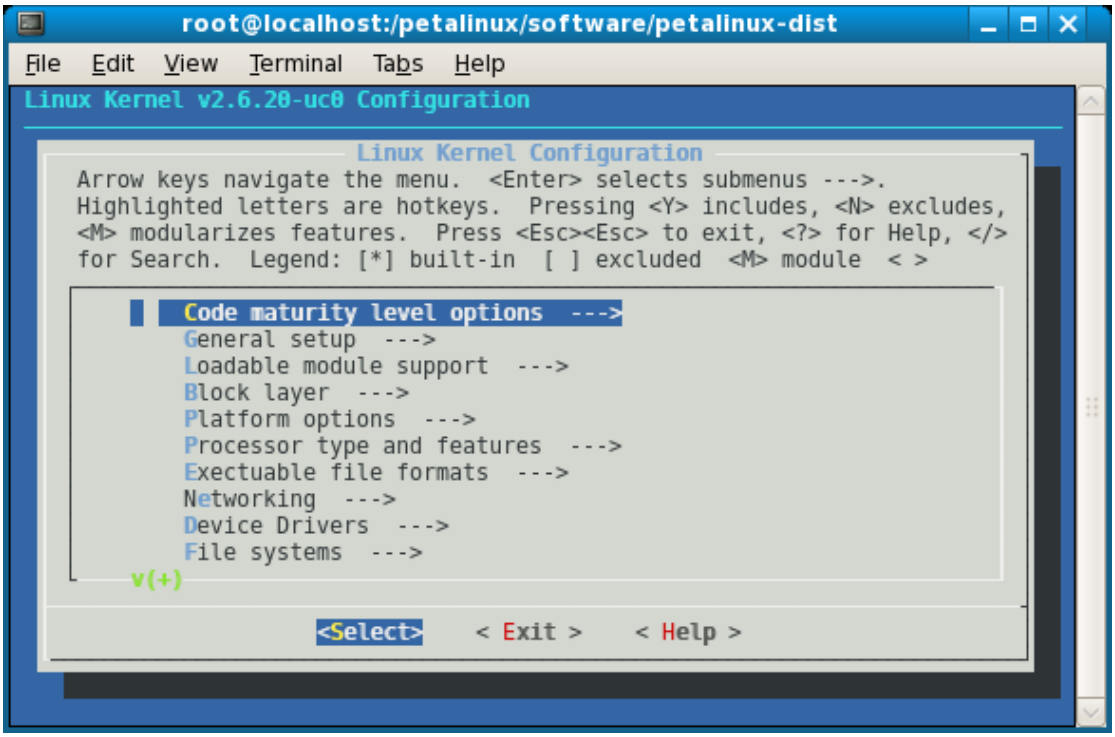
Το AutoConfig μας δίνει τη δυνατότητα να συγχρονίσουμε το software και το hardware του συστήματος. Αυτό γίνεται εκτελώντας την παρακάτω εντολή στο directory που έχομε φτιάξει μέσα στους φακέλους του uCLinux όπου δουλεύουμε το project μας :

```
$ petalinux-copy-autoconfig
```

Με την παραπάνω εντολή δημιουργείται ένα link ανάμεσα στο software και στο hardware project (που αφορά ρυθμίσεις που έγιναν απ'το EDK, καθώς και ρυθμίσεις που έγιναν από το UNIX σε μια προσπάθεια να ενοποιηθούν). Εντοπίζει αυτόματα τον κατασκευαστή και την πλατφόρμα που επιλέξαμε προηγουμένως.

Α.5 Ρυθμίζοντας τον Linux kernel του σύστηματος

Για να μπούμε στις ρυθμίσεις του kernel γράφουμε πάλι στην κονσόλα make menuconfig όπως προηγουμένως και επιλέγουμε Customize Kernel Settings. Στη συνέχεια κάνουμε save and exit και με αυτό τον τρόπο μας εμφανίζει το παρακάτω μενού :



Στο κεντρικό μενού επιλέγουμε Processor type and features ---> , όπου κάνουμε disable το GPIO driver και ενεργοποιούμε το Console on UARTLITE.

Στο κεντρικό μενού πάλι επιλέγουμε General Setup --->, όπου κάνουμε enable το Networking support.

Πάλι στο κεντρικό μενού επιλέγουμε Processor type and features --->, πηγαίνουμε στο μενού Memory Technology Devices (MTD) που μας οδηγεί στο μενού RAM/ROM/Flash chip drivers ---> στο οποίο κάνουμε enable "Support for Intel/Sharp flash chips"

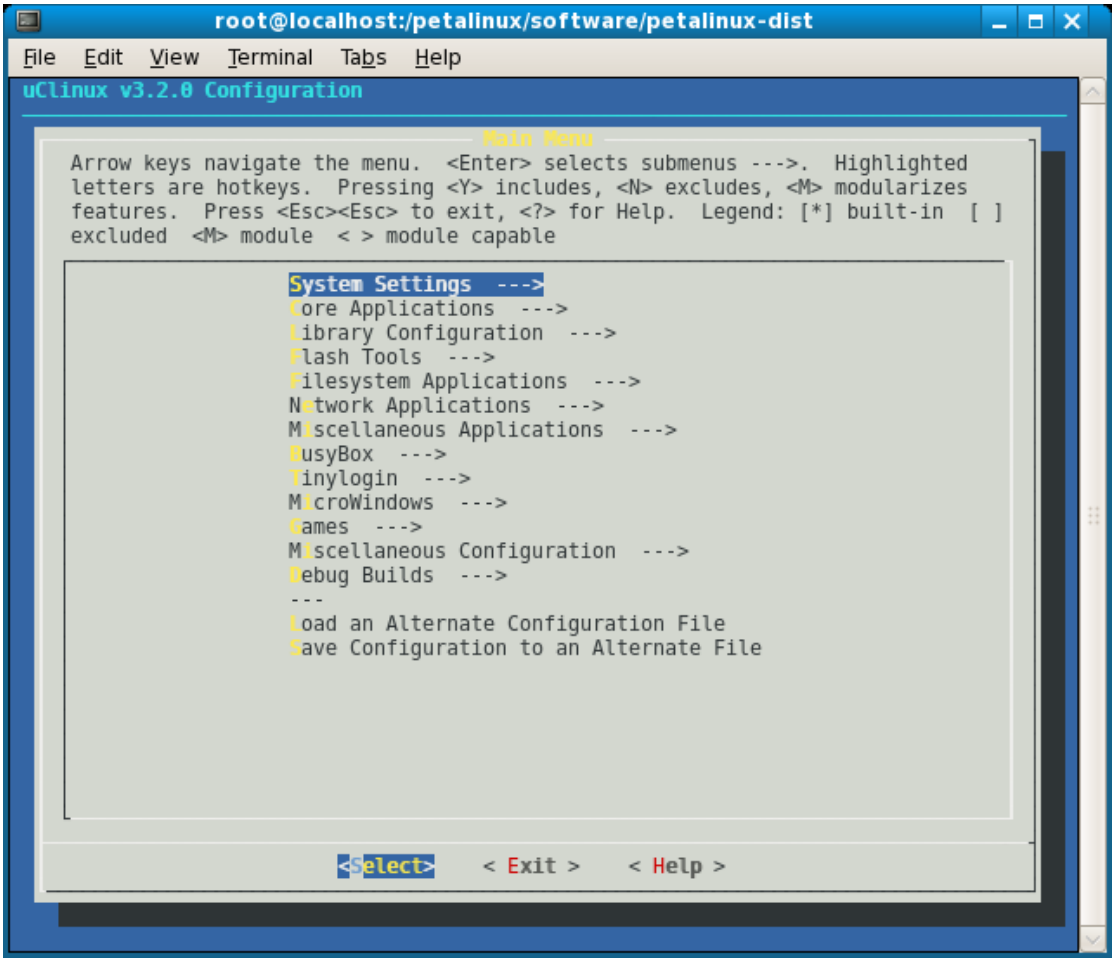
Πίσω στο "Memory Technology Devices (MTD)" πηγαίνουμε στο μενού "Mapping drivers for chip access ---> " , όπου επιλέγουμε μόνο "CFI Flash device PetaLinux AutoConfig" και "Generic uCLinux RAM/ROM filesystem support"

Στο κεντρικό μενού επιλέγουμε "Netwrok device Support “ και ενεργοποιούμε το "Netwrok device Support”. Από εδώ πηγαίνουμε στο μενού "Ethernet (10 or 100Mbit) --->" και κάνουμε enable τα "Ethernet (10 or 100Mbit)" και "Xilinx Ethernet driver".

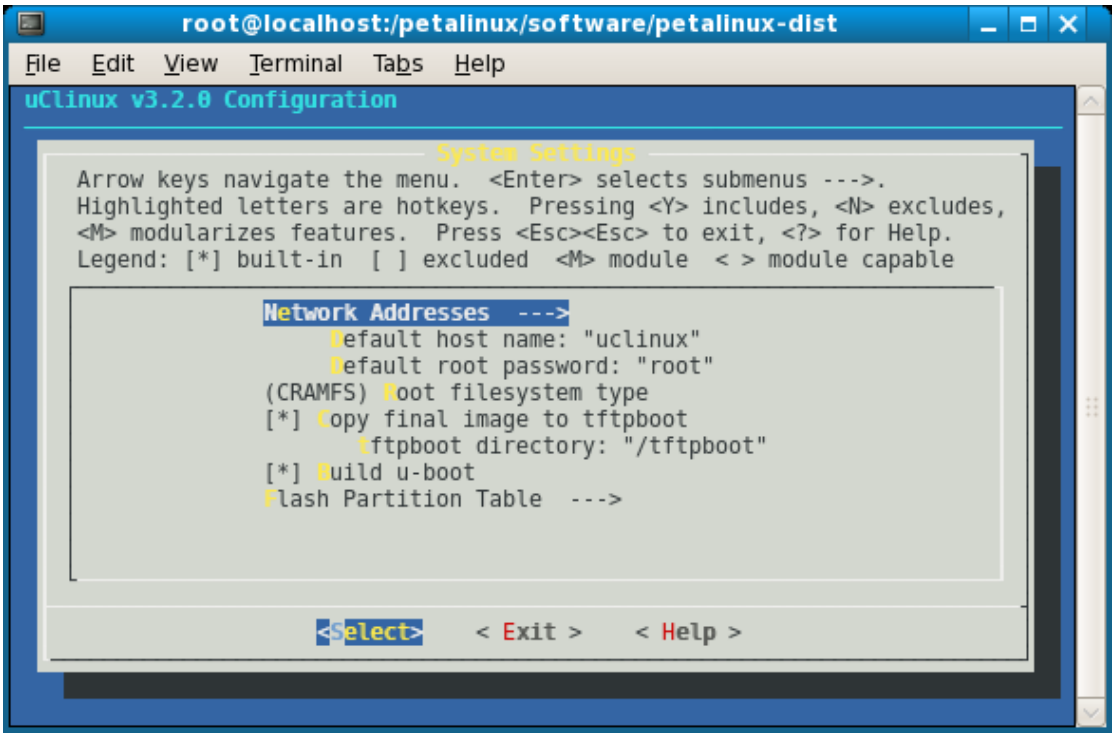
Στη συνέχεια κάνουμε save and exit configuration.

Α.6 Καθορισμός ρυθμίσεων του συστήματος

Για να μπούμε στις ρυθμίσεις του συστήματος γράφουμε πάλι στην κονσόλα make menuconfig όπως προηγουμένως και επιλέγουμε Customize Vendor/User Settings. Στη συνέχεια κάνουμε save and exit και με αυτό τον τρόπο μας εμφανίζει το παρακάτω μενού :

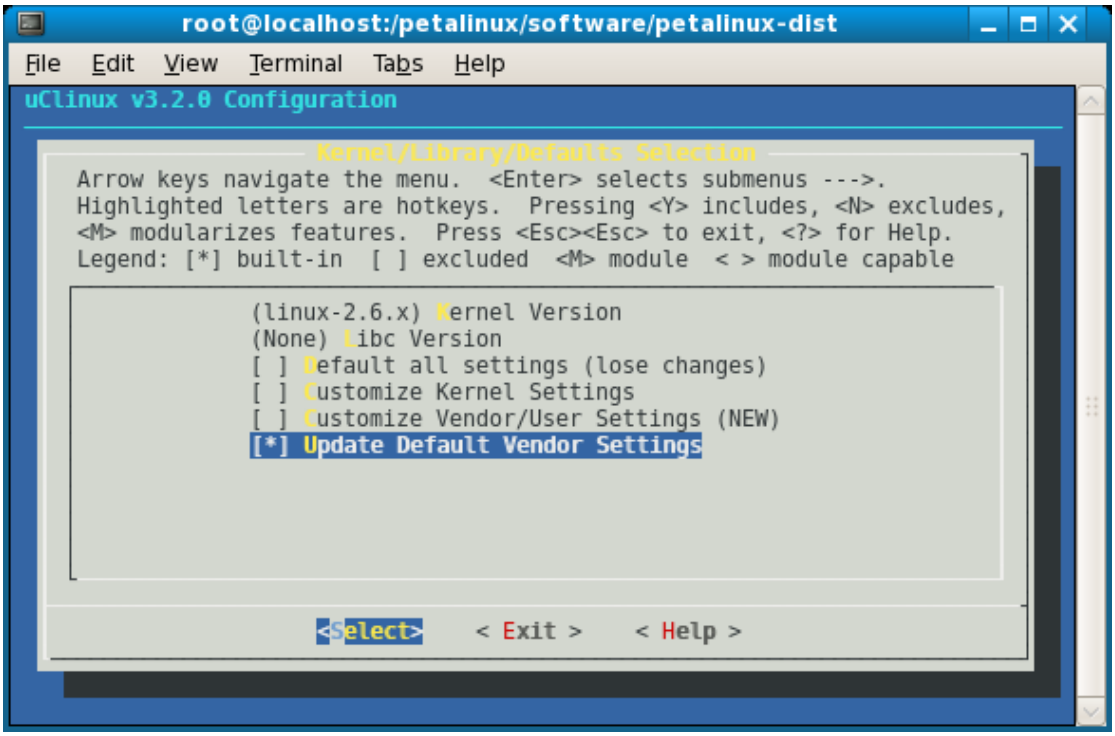


Επιλέγουμε το system settings και στο μενού που ακολουθεί ενημερώνουμε τις ρυθμίσεις του Network Address συμπεριλαμβανομένου και του MAC address. Έπειτα save and exit configuration.



A.7 Ανανέωση των Default ρυθμίσεων του κατασκευαστή

Μόλις ολοκληρωθούν οι ρυθμίσεις, θα πρέπει να σωθούν ως default για αυτή τη πλατφόρμα. Αυτό γίνεται με την επιλογή “Update Default Vendor Settings option”, όπως φαίνεται παρακάτω:



A.8 Φτιάχνοντας το Root Filesystem and Kernel Image

Για να ξεκινήσουμε τη διαδικασία αφού έχουμε σεττάρει την πλατφόρμα μας, μπαίνουμε στο subdirectory petalinux-dist

1) Ορίζουμε τις εξαρτήσεις του συστήματος με την εξής εντολή :

```
$ yes "" | make oldconfig dep
```

2) Κάνουμε build τα images μας με την εξής εντολή :

```
$ make all
```

3) 'Ελεγχος για το αν φτιάχτηκαν όλα σωστά και αν όχι επαναλαμβάνουμε το βήμα 2

4) Output image

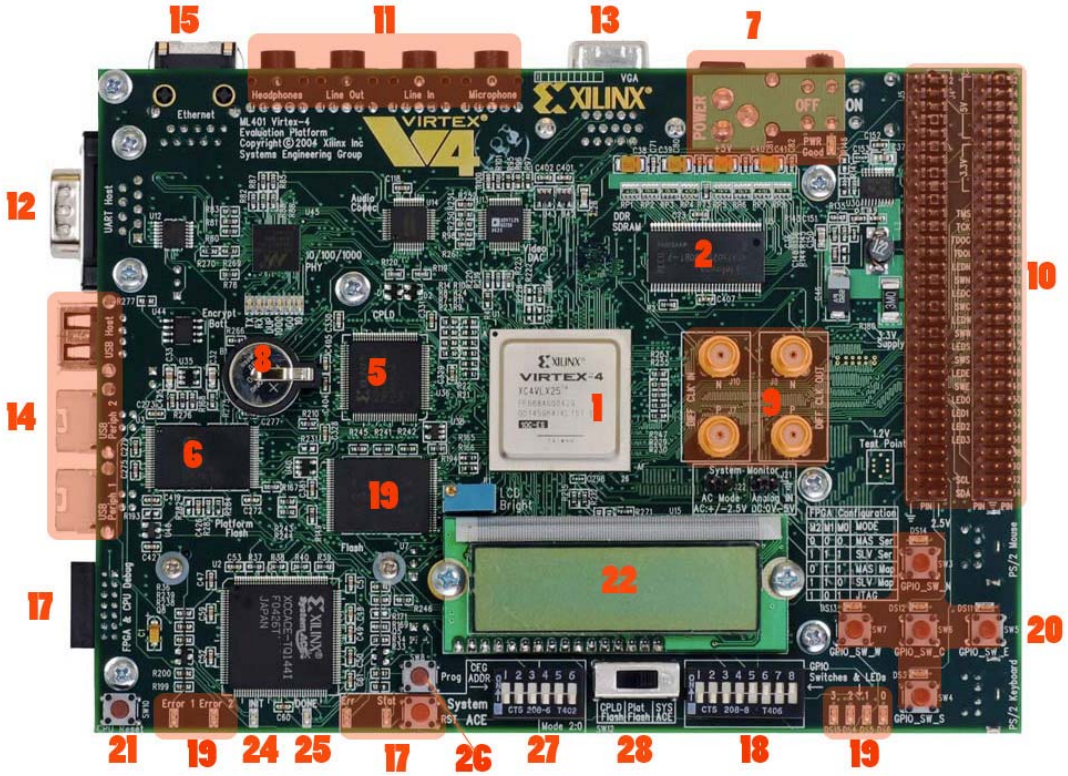
Το image που δημιουργήθηκε βρίσκεται στο παρακάτω directory :

/petalinux/uclinux-dist/images

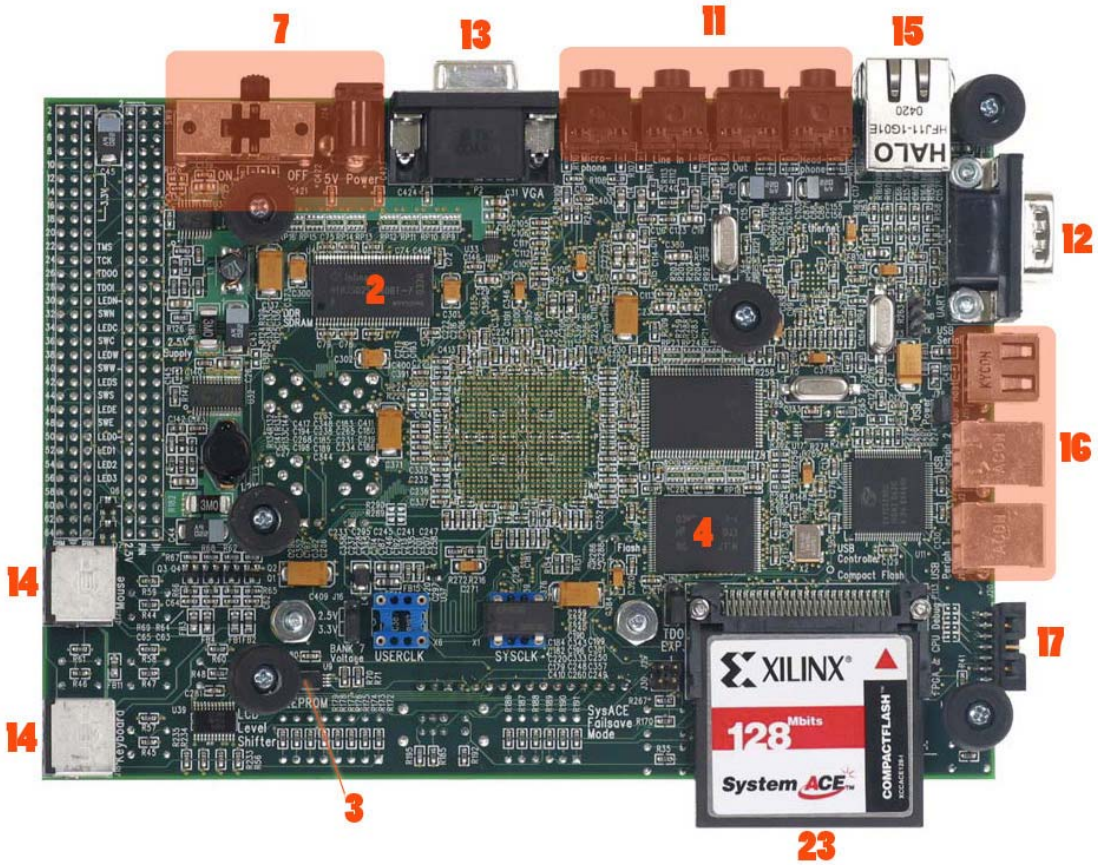
ΠΑΡΑΡΤΗΜΑ Β

Η ΑΝΑΤΟΜΙΑ ΤΟΥ FPGA BOARD ML403

Η πλακέτα FPGA ML403



(Μπροστά Όψη)



(Πίσω Όψη)

Παρακάτω περιγράφονται τα μέρη που απαρτίζουν το board σύμφωνα με τους αριθμούς που αναγράφονται στις εικόνες

Β.1 Μονάδες επεξεργασίας και μνήμες

1. Virtex-4 FPGA

Μια FPGA είναι μια συσκευή ημιαγωγών και αποτελειται απο πολλα blocks προγραμματιζόμενης λογικής, καθώς και απο τις διασυνδεσεις μεταξύ τους. Αυτά τα λογικά blocks μπορούν να προγραμματιστούν για να εκτελέσουν τις λογικές πράξεις AND και XOR καθώς και ποιο περίπλοκους συνδυασμούς των πράξεων αυτών. Στις περισσότερες FPGAs κάποια από αυτά τα blocks υποκαθιστούν στοιχεία μνήμης, δηλαδή απλά flip – flops ή ποιο σύνθετα blocks μνήμης. Δεν προσφέρονται για τον σχεδιασμό πολύ πολύπλοκων κυκλωμάτων, αλλά σε αντιπαράθεση με αυτό μας προσφέρουν το πλεονέκτημα του επαναπρογραμματισμού από την αρχή. Υπάρχουν δύο τρόποι για να τις προγραμματίσουμε: είτε με τη βοήθεια κυκλωματικών σχεδιαγραμμάτων, είτε μέσω HDL που είναι μια περιγραφική γλώσσα προγραμματισμού για FPGAs και συνίσταται για μεγαλύτερα και πολυπλοκότερα projects. Μετατρέποντας είτε τα σχεδιαγράμματα είτε τον κώδικα της HDL σε εκτελέσιμα κυκλώματα χρειαζόμαστε είτε ένα σειριακό interface τύπου JTAG, Είτε τη βοήθεια εξωτερικής μνήμης τύπου EEPROM.

2. DDR SDRAM

Το board περιέχει 64 MB DDR SDRAM σε δύο chip της Infineon (HYB25D256160BT-7). Κάθε chip έχει εύρος 16 bit και μαζί σχηματίζουν ένα 32μπιτο data bus που μπορεί να φτάσει μέχρι και τα 266 MHz. Όλα τα DDR SDRAM σήματα τερματίζονται μέσα από αντιστάσεις των 47 Ohm σε τάση αναφοράς 1.25 volt(V_{tt}). Το board είναι σχεδιασμένο για να μπορεί να υποστηρίξει μέχρι 256 MB DDR SDRAM μνήμης συνολικά σε περίπτωση που αντικαταστήσουμε τα υπάρχοντα chip με μεγαλύτερα. Ένα επιπλέον pin διεύθυνσης μπαίνει πάνω στο board όταν θέλουμε να υποστηρίξουμε chip του 1 GB.

DDR Clock Signal

The DDR clock signal is broadcast from the FPGA as a single differential pair that drives both DDR chips. The delay on the clock trace is designed to match the delay of the other DDR control and data signals. The DDR clock is also fed back to the FPGA to allow for clock deskew using Virtex-4 DCMs. The board is designed so that the DDR clock signal reaches the FPGA clock feedback pin at the same time as it arrives at the DDR chips.

3. IIC Bus with 4-Kb EEPROM

Μια IIC EEPROM παρέχεται στο board για να αποθηκεύει αμετάβλητα δεδομένα όπως η MAC address της ethernet. Η προστασία εγγραφής της EEPROM γίνεται disable για να εξουδετερωθεί η προστασία εγγραφής μέσω hardware. Ο δίαυλος IIC χρησιμοποιεί σήματα των 2.5 volt και μπορεί να λειτουργήσει μέχρι και στα 400 KHz. Το board είναι εξοπλισμένο με αντιστάσεις pull-up του διαύλου του IIC.

Ο δίαυλος IIC εκτείνεται μέχρι το ανάπτυγμα της ζεύξης έτσι ώστε ο χρήστης να μπορεί να προσθέσει επιπρόσθετες IIC συσκευές καθώς και να προσθέσει τον ελεγκτή IIC στην FPGA. Όταν ο χρηστής θελήσει να αξιοποιήσει την επέκταση του διαύλου IIC, θα πρέπει να έχει στη διάθεσή του επιπλέον pull-up αντιστάσεις πάνω στην κάρτα επέκτασης.

4. Linear Flash Chips

Στο board είναι εγκατεστημένες και 2 γραμμικές μνήμες flash των 32 Mbit παρέχοντας έτσι 8 MB flash μνήμης συνολικά. Οι μνήμες αυτές είναι συμβατές με το StrataFlash της Intel και χρησιμοποιούνται για μόνιμη αποθήκευση δεδομένων, λογισμικού ή ακόμα και bitstream.

Κάθε chip έχει εύρος 16 bit και μαζί σχηματίζουν ένα δίαυλο δεδομένων των 32 bit, ο οποίος διαμοιράζεται στην SRAM. Σε συζυγία με ένα CPLD, η flash μνήμη μπορεί να χρησιμοποιηθεί και για να φορτώσουμε ένα πρόγραμμα για την FPGA.

5. Xilinx XC95144XL CPLD

Το CPLD XC95144XL της Xilinx είναι συνδεδεμένο με τη flash μνήμη και τα σήματα διασύνδεσης της FPGA. Αυτή η διασύνδεση του CPLD χρησιμεύει σε εφαρμογές όπου η flash μνήμη προγραμματίζει την FPGA. Tp CPLD είναι προγραμματισμένο από την κύρια JTAG αλυσίδα του board. Το CPLD είναι σεταρισμένο με τέτοιο τρόπο έτσι ώστε να υποστηρίζει master και slave διαμόρφωση σε σειριακή και παράλληλη διάταξη(SelectMap). Για τη διαμόρφωση της FPGA μέσω του CPLD και της flash ο configuration selector(δηλαδή ο switch 12) πρέπει να είναι σε κατάσταση up για να επικοινωνεί το CPLD με τη flash.

6. Xilinx XCF32P Platform Flash Configuration Storage Device

Προσφέρει μια άνετη και φιλική προς χρήση λύση για τη διαμόρφωση της FPGA. Η flash μνήμη της πλατφόρμας μπορεί να αποθηκεύσει μέχρι και 4 images διαμόρφωσης(τα 2 στο board της ML402) τα οποία μπορούν να προσπελαθούν μέσω των address switches διαμόρφωσης. Για τη διαμόρφωση της FPGA με τη βοήθεια της flash της πλατφόρμας ο configuration selector(δηλαδή ο switch 12) πρέπει να είναι σε κατάσταση down.

Η flash της πλατφόρμας είναι σεταρισμένη με τέτοιο τρόπο έτσι ώστε να υποστηρίζει master και slave διαμόρφωση σε σειριακή και παράλληλη διάταξη(SelectMap). Η flash είναι προγραμματισμένη μέσω του λογισμικού της Xilinx impact, μέσω του καλωδίου JTAG.

7. AC Adapter and Input Power Switch/Jack

Ο διακόπτης λειτουργίας όταν βρίσκεται στη θέση ON δίνει τροφοδοτεί το board με 5Volt. Το AC καλώδιο τροφοδοσίας παρέχει 15Walt στα 3Amber.

8. Encryption Key Battery

Η μπαταρία που βρίσκεται στο Board, συνδέεται στο VBATT pin και χρησιμοποιείται για να διατηρεί το κλειδί κρυπτογράφησης για την FPGA. Η μπαταρία που χρησιμοποιείται είναι μια 12-mm lithium coin battery (3V).

9. Differential Clock Input And Output With SMA Connectors

Σήματα ρολογιού υψηλής ακριβείας μπορούν να γίνουν είσοδοι στην FPGA χρησιμοποιώντας διαφορετικά σήματα ρολογιού στα οποία ασκούνται 50 Ω μέσω ομοαξονικής σύνδεσης. Αυτό επιτρέπει σε μια εξωτερική γεννήτρια συχνοτήτων να οδηγήσει τις διάφορες εισόδους ρολογιού που τροφοδοτούν τις εισόδους του ρολογιού από τα pin της FPGA. Η FPGA μπορεί να ρυθμιστεί ώστε να παρουσιάσει μέχρι και 100Ω Αντίσταση.

Ένα διαφορικό ρολόι εξόδου από την FPGA οδηγείται από ένα δεύτερο ζευγάρι από ομοαξονικές συνδέσεις. Αυτό επιτρέπει στην FPGA να οδηγήσει ένα ακριβές ρολόι σε μια εξωτερική συσκευή όπως ένα κομμάτι από διαγνωστικά τεστ.

10. Expansion Headers

Το Board χρησιμοποιεί θύρες επέκτασης για την εύκολη επέκταση και προσαρμογή του Board με διάφορες εφαρμογές.

11. Stereo AC97 Audio Codec

Το board χρησιμοποιεί AC97 audio codec που του επιτρέπει την επεξεργασία του ήχου. Υποστηρίζει 16-bit στερεοφωνικό ήχο με δειγματοληψία πάνω από τα 48kHz. Υπάρχουν υποδοχές ήχου για μικρόφωνο, Line-in, Line-out και για ακουστικά

12. RS-232 Serial Port

Το board σειράς ML40x περιέχει μια αρσενική σειριακή θύρα τύπου DB-9 RS-232 επιτρέποντας έτσι στην FPGA να δέχεται και να λαμβάνει σειριακά δεδομένα από άλλες συσκευές. Η σειριακή θύρα είναι συνδεδεμένη σαν μια περιφερειακή μονάδα(DCE), όποτε χρειαζόμαστε ένα καλώδιο τύπου null modem για να επικοινωνεί το board με τη σειριακή θύρα ενός υπολογιστή. Η σειριακή θύρα είναι σχεδιασμένη για να λειτουργεί μέχρι τα 115200 baud. Για να ρυθμίσουμε το επίπεδο της τάσης των σημάτων ανάμεσα στην FPGA και την RS-232 απαιτείται ένα chip διασύνδεσης.

13. VGA Output

Η VGA θύρα εξόδου υποστηρίζει διασύνδεση με οθόνη υπολογιστή

Board	Speed	Description	Video monitor
ML403	140 MHz	15-bit video data bus connected to FPGA	Analog Devices

14. PS/2 θύρες για ποντίκι και πληκτρολόγιο

Το board εμπεριέχει δύο PS/2 θύρες: Μία για κέρσορα και μια για πληκτρολόγιο. Οι PS/2 θύρες τροφοδοτούνται κατευθείαν από το καλώδιο τροφοδοσίας που παρέχει ρεύμα σε όλο το board.

15. 10/100/1000 Tri-Speed Ethernet PHY

Το board εμπεριέχει τη συσκευή Marvell Alaska PHY device (88E1111), η οποία κάνει διαχείριση της ταχύτητας του Ethernet στα 10/100/1000 Mb/s.

16. USB Controller with Host and Peripheral Ports

Ο Ενσωματωμένος Cypress CY7/C67300 USB host controller παρέχει τη USB συνδεσιμότητα στο board. Ο ελεγκτής USB υποστηρίζει τον host και το περιφερειακό τρόπο λειτουργίας. Ο ελεγκτής USB έχει 2 μηχανές διεπαφής (SIE) που μπορούν να χρησιμοποιηθούν ανεξάρτητα. Το SIE_1 είναι συνδεδεμένο με το USB host_1 σύνδεσμο και με το USB περιφεριακο_1 σύνδεσμο. Το SIE_2 είναι συνδεδεμένο μόνο με το USB περιφεριακο_2 σύνδεσμο.

17. JTAG Configuration Port

Η JTAG θύρα, επιτρέπει μέσω του ειδικού καλωδίου που ονομάζεται "προγραμματιστής", το προγραμματισμό καθώς και το debug της FPGA. Η θύρα JTAG έχει παράλληλη συνδεσιμότητα.

18. DIP Switches (Active-High)

Στην FPGA 8 switch γενικού σκοπού(DIP switches) συνδέονται στα pins εισόδου/εξόδου του χρήστη.

19. User and Error LEDs (Active-High)

Υπάρχουν συνολικά 11 active-High LEDs άμεσα ελεγχόμενα από την FPGA:

• Τέσσερα πράσινα LEDs, που είναι LEDs γενικού σκοπού ταξινομημένα σε σειρά

• Πέντε πράσινα LEDs είναι τοποθετημένα δίπλα σε κουμπιά του board.

• Δύο κόκκινα LEDs που προορίζονται να χρησιμοποιηθούν για συνθήκες σφαλμάτων, όπως για παράδειγμα σε σφάλμα του διαύλου επικοινωνιών (bus), μπορούν να χρησιμοποιηθούν και για οποιοδήποτε άλλο σκοπό.

20. User Push Buttons (Active-High)

Έχουμε 5 μπουτόν που χρησιμεύουν σε υλοποιήσεις γενικού σκοπού και βρίσκονται τοποθετημένα βορειοανατολικά/νοτιοδυτικά του κέντρου.

21. CPU Reset Button (Active-Low)

Το μπουτόν reset της CPU είναι σχεδιασμένο για να χρησιμοποιείται σαν system reset ή use reset. Το μπουτόν αυτό είναι συνδεδεμένο μόνο με ένα pin εισόδου/εξόδου της FPGA οπότε μπορεί να χρησιμοποιηθεί και σαν γενικού σκοπού.

22. 16-Character x 2-Line LCD

Το ML403 διαθέτει μια οθόνη υγρών κρυστάλλων (LCD) που μπορεί να απεικονίσει δύο γραμμές δεκαέξι χαρακτήρων. Η διεπαφή των δεδομένων στην LCD που συνδέεται στην FPGA υποστηρίζει πληροφορία των 4-bit.

23. System ACE and CompactFlash Connector

Επιτρέπει στην FPGA να προγραμματίζεται από εξωτερική κάρτα μνήμης flash τύπου CompactFlash(Type I ή Type II) μέσω της θύρας JTAG, χάρη στην οποία μπορούμε να κατεβάσουμε είτε hardware είτε software data. Ο ελεγκτής του System ACE υποστηρίζει έως και 8 διαφορετικά image διαμόρφωσης σε μια και μόνο κάρτα, καθένα από τα οποία μπορούν να επιλεγούν από τα 8 DIP switches.

Τα LEDs του System ACE controller υποδεικνύουν τις λειτουργικές καταστάσεις στις οποίες μπορεί να βρεθεί :

- όταν αναβοσβήνει κόκκινο το error LED σημαίνει πως απουσιάζει η κάρτα(CompactFlash)
- όταν ανάβει σταθερά κόκκινο το error LED σημαίνει πως προέκυψε λάθος κατά τη διαμόρφωση
- όταν αναβοσβήνει πράσινο το status LED σημαίνει πως η διαδικασία διαμόρφωσης βρίσκεται σε εξέλιξη
- όταν ανάβει σταθερά πράσινο το status LED σημαίνει πως έχει γίνει επιτυχές download από την κάρτα

Κάθε φορά που εισέρχεται μια CompactFlash στον card reader της, αρχικοποιείται μια λειτουργία διαμόρφωσης. Πατώντας το μπουτόν reset, επαναπρογραμματίζεται η FPGA.

Το board διαθέτει και ένα failsafe mode : όταν ο controller ανιχνεύει μια εσφαλμένη προσπάθεια διαμόρφωσης, αυτόματα επανεκινείται για να επανέλθει σε μια προκαθορισμένη κατάσταση.

24. INIT LED

Η ένδειξη INIT LED ανάβει μόνο όταν όλα στην πλακέτα δουλεύουν σωστά κατά το άναμμα της FPGA.

25. DONE LED

Όταν η ένδειξη DONE LED ανάψει, τότε αυτό σηματοδοτεί ότι η FPGA έχει ρυθμιστεί επιτυχώς.

26. Program Switch

Ο διακόπτης αυτός όταν πατηθεί, γειώνεται το prog pin της FPGA. Αυτή η ενέργεια "καθαρίζει" την FPGA.

27. Configuration Address and Mode DIP Switches

Η θέση των έξι DIP διακοπτών, ελέγχουν τη διεύθυνση διαμόρφωσης καθώς και τους FPGA τρόπους διαμόρφωσης. Οι τρείς διακόπτες από τα αριστερά δίνουν τη δινατότητα της επιλογής μίας από τις οκτώ διευθύνσεις διαμόρφωσης. Αυτοί οι τρείς διακόπτες παρέχουν στον System ACE ελεγκτή και τη CPLD τη δυνατότητα χρήσης έως και οκτώ διαφορετικών εικόνων διαμόρφωσης όπως ορίζεται από αυτούς τους τρείς διακόπτες. Η πλατφόρμα της Flash μνήμης υποστηρίζει έως και τέσσερις διαφορετικές εικόνες. Οι τρείς δεξιότεροι DIP διακόπτες, διαμορφώνουν στην FPGA τα pin : M2, M1, M0 όπως φαίνεται στον παρακάτω πίνακα.

M2	**M1**	**M0**	**Διαμόρφωση**
0	0	0	Master Serial
1	1	1	Slave Serial
0	1	1	Master Parallel (SelectMAP)
1	1	0	Slave Parallel (SelectMAP)
1	0	1	JTAG

28. Configuration Source Selector Switch

Ο διακόπτη επιλογής (SW12), δίνει την επιλογή μεταξύ του System ACE, της Flash πλατφόρμας, και μεταξύ της γραμμικής flash / CPLD μεθόδου προγραμματισμού της FPGA. Η PC4 υποδοχή επιτρέπει μέσω του JTAG το κατέβασμα και τον εντοπισμό σφαλμάτων, ανεξάρτητα από τη ρύθμιση του διακόπτη επιλογής της πηγής διαμόρφωσης, μέσω του DIP διακόπτη.

ΠΑΡΑΡΤΗΜΑ Γ

Ο ΚΩΔΙΚΑΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ

ΣΕ ΓΛΩΣΣΑ C

Γ.1 anazhthsht.c

```
#include "xtmrctr.h"
#include "xparameters.h"
//#include "lib.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
        int pin[100];
        int i;
        int ti ;
   XTmrCtr XPS_Timer ;

        XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
   XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
   XTmrCtr_Reset(&XPS_Timer,0) ;
   XTmrCtr_Start(&XPS_Timer,0) ;

        for(i=0;i<100;i++)
        {
                pin[i]=i++ * 2 + 3;
        }

        for(i=0;i<100;i++)
        {
                if(pin[i] == 33)
                {
                        print("h anazhthsh oloklhrw8hke epityxws kai o ari8mos 33 bre8hke
                        sth 8esh tou pinaka pin ");
                        putnum(i);
                        break;
                }
                else
                {
                        print("den bre8hke o ari8mos 33 sth 8esh ");
                        putnum(i);
                }
        }

        XTmrCtr_Stop(&XPS_Timer,0) ;
   ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
   putnum(ti) ;
   print("\r\n");
}
```

Γ.2 hyperbolic1t.c

```
#include <math.h>
#include <stdio.h>
#include "xtmrctr.h"
#include "xparameters.h"


int main(int argc, char *argv[])
{
  double val = -1.0, val2;
  int ti ;
  XTmrCtr XPS_Timer ;


    XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

            do {

    //print("Hyperbolic tangent of ");
            //putnum(val);
    //print(" is ");
            val2 = tanh(val);
            //print(".\n");
            val += 0.1;

            } while(val<=1.0);

            XTmrCtr_Stop(&XPS_Timer,0) ;
    ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(ti) ;
    print("\r\n");


  return 0;
}
```

Γ.3 tantantanR1t.c

```
#include <stdio.h>
#include <math.h>
#include "xtmrctr.h"
#include "xparameters.h"

// upologismos efaptomenhs kai tokso efaptomenhs pollaplwn arithmwn


int main(int argc, char *argv[])
{

double val = -1.0;
int ti ;
XTmrCtr XPS_Timer ;

  do {

    XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

            //atan
            print("Arc tangent of ");
            putnum(val);
            print(" is ");
            putnum(atan(val));
            print(".\n");

            //atan2
            print("Atan2 of ");
            putnum(val);
            print(" is ");
            putnum(atan2(val, 1.0));
            print(".\n");

            val += 0.1;

    XTmrCtr_Stop(&XPS_Timer,0) ;
    ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(ti) ;
    print("\r\n");

  } while(val<=1.0);

  return 0;
}
```

Γ.4 tantantan1t.c

```
#include <stdio.h>
#include <math.h>
#include "xtmrctr.h"
#include "xparameters.h"

#define PI 3.14159265

// aplos upologismos efaptomenhs kai tokso efaptomenhs

int main(int argc, char *argv[])
{
  double v, result, x, y, result2;
  v = 5;
  x = 0;
  y = 6;
 int ti ;
 XTmrCtr XPS_Timer ;

    XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

  result = atan ( v ) * 180 / PI;
  print("Arctangent of ");
  putnum(v);
  print(" is ");
  putnum(result);
  print(" degrees\n");

  result2 = atan2 (y, x) * 180 / PI;
  print("Arctangent for (x = ");
  putnum(x);
  print(", y = ");
  putnum(y);
  print(") is ");
  putnum(result2);
  print(" degrees\n");

  XTmrCtr_Stop(&XPS_Timer,0) ;
    ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(ti) ;
    print("\r\n");

  return 0;
  }
```

Γ.5 randPowers1t.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "xtmrctr.h"
#include "xparameters.h"

// tuxaioi arithmoi se sundiasmo me dunameis

const int MAX = 40;
int main(int argc, char *argv[])
{
   int index, number, newNumber;
            int ti ;
            XTmrCtr XPS_Timer ;

   for (index = 0; index < 30; ++index) {

            XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
   XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
   XTmrCtr_Reset(&XPS_Timer,0) ;
   XTmrCtr_Start(&XPS_Timer,0) ;

     number = rand();
               newNumber = 1 + (number % (MAX-1));
               print("H dunamh tou arithmou ");
               putnum(newNumber);
               print(" einai \n gia ^2 = ");
               putnum(pow(newNumber, 2));
               print(", gia ^12 = ");
               putnum(pow(newNumber, 12));
               print(" kai gia ^1.54 = ");
               putnum(pow(newNumber, 1.54));
               print(".\n");
               print(".\n");

               XTmrCtr_Stop(&XPS_Timer,0) ;
   ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
   putnum(ti) ;
   print("\r\n");

   }
   return (0);
}
```

Γ.6 sobel.c

```
#define MAX
#include "xtmrctr.h"
#include "xparameters.h"
//#include "lib.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
//#include<string.h>

struct header {
        int nr, nc;          /* Rows and columns in the image */
        int oi, oj;          /* Origin */
};

/*    The IMAGE data structure      */
struct image {
                struct header *info;          /* Pointer to header */
                unsigned char **data;         /* Pixel values */
};

typedef struct image * IMAGE;
void sys_abort (int val, char *mess)
{
        xil_printf ("**** System library ABORT %d: %s ****\n", val, mess);
        exit (2);
}
/*****************EXPORT TO FILE******************/
/*
IMAGE Output_PBM (IMAGE image, char *filename){
        FILE *f;
        int i,j,k, perline;
        char buf1[64];

        strcpy (buf1, filename);
        if (image->info->nc > 20) perline = 20;
         else perline = image->info->nc-1;
        f = fopen (buf1, "w");
        if (f == 0) sys_abort (0, "Can't open output file.");

        fprintf (f,"P2\n#origin %d %d\n",image->info->oj,image->info->oi);
        fprintf (f, "%d %d %d\n", image->info->nc, image->info->nr, 255);
        k = 0;
        for (i=0; i<image->info->nr; i++)
          for (j=0; j<image->info->nc; j++)
          {
                fprintf (f, "%d ", image->data[i][j]);
                k++;
                if (k > perline)
                {
                  fprintf (f, "\n");
                  k = 0;
                }
          }
        fprintf (f, "\n");
        fclose (f);
        return image;
}
```

```
*/
/*****************************************************/



struct image  *newimage (int nr, int nc)
{
        struct image  *x;               /* New image */
        //unsigned char *ptr;             /* new pixel array */
        int i;


        if (nr < 0 || nc < 0) {
                xil_printf ("Error: Bad image size\n");
                return 0;
        }

/*      Allocate the image structure    */
        x = (struct image  *) malloc( sizeof (struct image) );
        if (!x) {
                xil_printf ("Out of storage in NEWIMAGE (1).\n");
                return 0;
        }

/*      Allocate and initialize the header      */

        x->info = (struct header *)malloc( sizeof(struct header) );
        if (!(x->info)) {
                xil_printf ("Out of storage in NEWIMAGE (2).\n");
                return 0;
        }
        x->info->nr = nr;       x->info->nc = nc;
        x->info->oi = x->info->oj = 0;

/*      Allocate the pixel array        */

        x->data = (unsigned char **)malloc(sizeof(unsigned char *)*nr);

/* Pointers to rows */
        if (!(x->data)) {
                xil_printf ("Out of storage in NEWIMAGE (3).\n");
                return 0;
        }

        for (i=0; i<1; i++)
        {
          x->data[i] = (unsigned char *)malloc(nc*sizeof(unsigned char));
          if (x->data[i]==0)
          {
                xil_printf ("Out of storage. Newimage - row %d\n", i);
                exit(1);
          }
        }
        return x;
}
```

```
void freeimage (struct image  *z)
{

/*      Free the storage associated with the image Z    */
        int i;

        if (z != 0)
        {
          for (i=0; i<z->info->nr; i++)
              free (z->data[i]);
           free (z->info);
           free (z->data);
           free (z);
        }
}


void CopyVarImage (IMAGE *a, IMAGE *b)
{
        int i,j;


        for (i=0; i<(*b)->info->nr; i++)
          for (j=0; j< (*b)->info->nc; j++)
            (*a)->data[i][j] = (*b)->data[i][j];
        (*a)->info->oi = (*b)->info->oi;
        (*a)->info->oj = (*b)->info->oj;
}


/*
void copy (IMAGE *a, IMAGE b)
{
        CopyVarImage (a, &b);
}
*/

void thresh (IMAGE z){
        int histo[256];
        int i,j,t;

/* Compute a grey level histogram */
        for (i=0; i<256; i++) histo[i] = 0;
        for (i=1; i<z->info->nr-1; i++)
          for (j=1; j<z->info->nc-1; j++){
            histo[z->data[i][j]]++;
          }

/* Threshold at the middle of the occupied levels */
        i = 255;
        while (histo[i] == 0) i--;
        j = 0;
        while (histo[j] == 0) j++;
        t = (i+j)/2;

/* Apply the threshold */
        for (i=1; i<z->info->nr-1; i++)
          for (j=1; j<z->info->nc-1; j++)
            if (z->data[i][j] >= t) z->data[i][j] = 0;
            else z->data[i][j] = 255;
}
```

```
/*          Apply a Sobel edge mask to the image X      */

IMAGE sobel (struct image *x, struct image *z){
            int i,j,n,m,k;
            CopyVarImage (&z, &x);

            for (i=0; i<x->info->nr; i++)
              for (j=0; j<x->info->nc; j++)
                z->data[i][j] = 255;

/* Now compute the convolution, scaling */
            for (i=1; i<x->info->nr-1; i++)
              for (j=1; j<x->info->nc-1; j++) {
                n = (x->data[i-1][j+1]+2*x->data[i][j+1]+x->data[i+1][j+1]) -
                    (x->data[i-1][j-1]+2*x->data[i][j-1]+x->data[i+1][j-1]);
                m = (x->data[i+1][j-1]+2*x->data[i+1][j]+x->data[i+1][j+1])-
                    (x->data[i-1][j-1]+2*x->data[i-1][j]+x->data[i-1][j+1]);
                k = (int)( sqrt( (double)(n*n + m*m) )/4.0 );
                z->data[i][j] = k;
              }

            thresh (z);

       return ( z );
}



 unsigned char imgdata[7][16] = {
{  34,  35,  38,  83, 153, 172, 175, 175, 174, 172, 172, 174, 172, 169, 172, 175},
{  37,  38,  46, 102, 152, 161, 158, 156, 151, 147, 144, 146, 144, 142, 143, 144},
{  82,  83,  76,  79,  91,  90,  87,  85,  84,  81,  79,  78,  75,  72,  72,  73},
{ 148, 139,  97,  56,  41,  36,  35,  35,  35,  36,  35,  35,  34,  33,  33,  33},
{ 167, 142,  87,  48,  35,  31,  31,  31,  32,  33,  33,  31,  31,  31,  31,  31},
{ 165, 126,  69,  40,  33,  31,  31,  31,  31,  32,  32,  30,  30,  29,  30,  31},
{ 166, 113,  59,  38,  33,  31,  32,  31,  33,  33,  33,  31,  31,  30,  31,  32}
};



#define NCOLS 181
#define NROWS 146

#define MAXUBLAZE 4
#define UBLAZE 0

#define DATAWORDS 4

//int main (int argc, char *argv[]){
int main ( void ) {

            int ti;

  XTmrCtr XPS_Timer ;

   microblaze_init_icache_range(0, XPAR_MICROBLAZE_0_CACHE_BYTE_SIZE);
     microblaze_enable_icache();

     microblaze_init_dcache_range(0, XPAR_MICROBLAZE_0_DCACHE_BYTE_SIZE);
     microblaze_enable_dcache();
```

```
  u32 cycles ;

  u32 *IsendIMG ;
  u32 *Isendflag ;

//SRAM_C_MEM0_BASEADDR : ORIGIN = 0x84500000, LENGTH = 0x00100000

  // use 256 bytes from the end of SRAM :
//  Isendflag = (u32 *) XPAR_SRAM_MEM0_BASEADDR+0x000FFF00 ; //0x84500000 ;

  Isendflag = (u32 *) XPAR_SRAM_MEM0_BASEADDR ; //0x84500000 ;
  IsendIMG = (u32 *) (XPAR_SRAM_MEM0_BASEADDR  + MAXUBLAZE*4 ) ;



            int rows;
            int currow;

            int i;

   for (i=0; i<MAXUBLAZE; i++)
           *(Isendflag+i) = 0;

   rows = (int) ( NROWS / MAXUBLAZE ) + 1 ;
   currow = 0 ;
   for (i=0; i<MAXUBLAZE; i++) {
     *(IsendIMG+i*DATAWORDS) = NCOLS ;  // columns
     *(IsendIMG+i*DATAWORDS+2) = &imgdata[currow] ;

     if (i==0) {
       *(IsendIMG+i*DATAWORDS+1) = rows  ;
       currow = currow + rows - 2 ;
     } else {
       if (i==MAXUBLAZE-1) {
        *(IsendIMG+i*DATAWORDS+1) = NROWS - (currow + rows) +1  ;
       } else {
        *(IsendIMG+i*DATAWORDS+1) = rows+1  ;
        currow = currow + rows - 1 ;
       }
     }
    //putnum ( currow ) ;
    xil_printf (" ") ;
    //putnum ( rows ) ;
    print ("\r\n") ;

   }

/******** Arxikopoihsh x1**/
            IMAGE x1=(IMAGE)malloc(sizeof(struct image));
       x1->info=(struct header *)malloc( sizeof(struct header) ); // gia ton epexergasti 1

            x1->info->nc=181;
            x1->info->nr=37;
//x1->info->oi=x1->info->oj= 0;

            x1->data=(unsigned char **) malloc(sizeof(unsigned char *)*37); //pinakas deiktwn unsigned
char 40 thesewn
```

```
            for(i=0;i<37;i++)
                        //x1->data[i] =  (unsigned char *) x->data[i] ;
                  x1->data[i] =  (unsigned char *) imgdata[i] ;


        IMAGE z1 = newimage ( x1->info->nr, x1->info->nc);
        if (z1 == 0) sys_abort (0, "No more storage.\n");

        *(IsendIMG+0*DATAWORDS+3) = z1 ;

xil_printf (" 1_") ;
// ////////////////////////////////////////////////////////////////////////////////////////

xil_printf ("\n") ;

   for (i=0; i<MAXUBLAZE; i++)
            *(Isendflag+i) = 1;

/*
             Output_PBM(x1,"sobelpic.pgm");

*/
XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

x1 = sobel (x1,z1);

XTmrCtr_Stop(&XPS_Timer,0) ;
    ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(ti) ;
    print("\r\n");

xil_printf (" 1.") ;
//x2 = sobel (x2);
//xil_printf (" 2.") ;
//x3 = sobel (x3);
//xil_printf (" 3.") ;
//x4 = sobel (x4);
//xil_printf (" 4.") ;



//sobel (x);


/*
Output_PBM(x1,"metaX1.pgm");
*/

             xil_printf ("process done\n");

        while(1) ;

             return 0;
}
```

Γ.7 dijkstra1t.c

```
#include <stdio.h>
#include <stdlib.h>
#include "xtmrctr.h"
#include "xparameters.h"

#define NUM_NODES               100
#define NONE                    9999

struct _NODE
{
  int iDist;
  int iPrev;
};
typedef struct _NODE NODE;

struct _QITEM
{
  int iNode;
  int iDist;
  int iPrev;
  struct _QITEM *qNext;
};
typedef struct _QITEM QITEM;

QITEM *qHead = NULL;

    int AdjMatrix[NUM_NODES][NUM_NODES];
int g_qCount = 0;
NODE rgnNodes[NUM_NODES];
int ch;
int iPrev, iNode;
int i, iCost, iDist;


void print_path (NODE *rgnNodes, int chNode)
{
  if (rgnNodes[chNode].iPrev != NONE)
    {
      print_path(rgnNodes, rgnNodes[chNode].iPrev);
    }
putnum(chNode);
fflush(stdout);
}

void enqueue (int iNode, int iDist, int iPrev)
{
  QITEM *qNew = (QITEM *) malloc(sizeof(QITEM));
  QITEM *qLast = qHead;

  if (!qNew)
    {
      //fprintf(stderr, "Out of memory.\n");
      exit(1);
    }
  qNew->iNode = iNode;
  qNew->iDist = iDist;
  qNew->iPrev = iPrev;
  qNew->qNext = NULL;
```

```
  if (!qLast)
    {
      qHead = qNew;
    }
  else
    {
      while (qLast->qNext) qLast = qLast->qNext;
      qLast->qNext = qNew;
    }
  g_qCount++;
  //              ASSERT(g_qCount);
}


void dequeue (int *piNode, int *piDist, int *piPrev)
{
  QITEM *qKill = qHead;

  if (qHead)
    {
      //                ASSERT(g_qCount);
      *piNode = qHead->iNode;
      *piDist = qHead->iDist;
      *piPrev = qHead->iPrev;
      qHead = qHead->qNext;
      free(qKill);
      g_qCount--;
    }
}

int qcount (void)
{
  return(g_qCount);
}

int dijkstra(int chStart, int chEnd)
{



  for (ch = 0; ch < NUM_NODES; ch++)
    {
      rgnNodes[ch].iDist = NONE;
      rgnNodes[ch].iPrev = NONE;
    }

  if (chStart == chEnd)
    {
      print("Shortest path is 0 in cost. Just stay where you are.\n");
    }
  else
  {
      rgnNodes[chStart].iDist = 0;
      rgnNodes[chStart].iPrev = NONE;
      enqueue (chStart, 0, NONE);
```

```
    while (qcount() > 0)
    {
      dequeue (&iNode, &iDist, &iPrev);
      for (i = 0; i < NUM_NODES; i++) {
           if ((iCost = AdjMatrix[iNode][i]) != NONE)  {
                if ((NONE == rgnNodes[i].iDist) || (rgnNodes[i].iDist > (iCost + iDist))) {
                     rgnNodes[i].iDist = iDist + iCost;
                     rgnNodes[i].iPrev = iNode;
                     enqueue (i, iDist + iCost, iNode);
                }
           }
      }
   }
     print("Shortest path is ");
     putnum(rgnNodes[chEnd].iDist);
     print(" in cost. ");
     print("Path is: ");
     print_path(rgnNodes, chEnd);
     printf("\n");
   }
}

int main(int argc, char *argv[]) {
  int i,j,k;
  int ti ;
  XTmrCtr XPS_Timer ;
  FILE *fp;

  if (argc<2) {
    //fprintf(stderr, "Usage: dijkstra <filename>\n");
    //fprintf(stderr, "Only supports matrix size is #define'd.\n");
  }

  /* open the adjacency matrix file */
  fp = fopen (argv[1],"r");


    XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

     dijkstra(i,j);

    XTmrCtr_Stop(&XPS_Timer,0) ;
    ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(ti);
    print("\r\n");
  }
  exit(0);
}
```

Γ.8 BubbleSort1t.c

```
#include <stdlib.h>
#include <stdio.h>
#include "xtmrctr.h"
#include "xparameters.h"

#define uint32 unsigned int

typedef int (*CMPFUN)(int, int);



void ArraySort(int This[], CMPFUN fun_ptr, uint32 ub)
{
  /* bubble sort */

  uint32 indx;
  uint32 indx2;
  int temp;
  int temp2;
  int flipped;

  if (ub <= 1)
    return;

  indx = 1;
  do
  {
    flipped = 0;
    for (indx2 = ub - 1; indx2 >= indx; --indx2)
    {
      temp = This[indx2];
      temp2 = This[indx2 - 1];
      if ((*fun_ptr)(temp2, temp) > 0)
      {
        This[indx2 - 1] = temp;
        This[indx2] = temp2;
        flipped = 1;
      }
    }
  } while ((++indx < ub) && flipped);
}

#define ARRAY_SIZE 100

int my_array[ARRAY_SIZE];

void fill_array()
{
  int indx;

  for (indx=0; indx < ARRAY_SIZE; ++indx)
  {
    my_array[indx] = rand();
  }
  /* my_array[ARRAY_SIZE - 1] = ARRAY_SIZE / 3; */
}

int cmpfun(int a, int b)
```

```
{
  if (a > b)
    return 1;
  else if (a < b)
    return -1;
  else
    return 0;
}

int main(int argc, char *argv[])
{
  int indx;
  int indx2;
  int i ;
  XTmrCtr XPS_Timer ;

  print("Check BubbleSort:\r\n");

    fill_array();

    XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

       ArraySort(my_array, cmpfun, ARRAY_SIZE);

    XTmrCtr_Stop(&XPS_Timer,0) ;
    i = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(i) ;
    print("\r\n");
    //delay :
    for (i=0; i<8000000; ++i) ;

    for (indx=1; indx < ARRAY_SIZE; ++indx)
    {
      if (my_array[indx - 1] > my_array[indx])
      {
        print("bad sort\n");
        return(1);
      }
    }


}
```

Γ.9 mergeSort1t.c

```
#include <stdlib.h>
#include <stdio.h>
#include "xtmrctr.h"
#include "xparameters.h"


#define uint32 unsigned int

typedef int (*CMPFUN)(int, int);



#define INSERTION_SORT_BOUND 8 /* boundary point to use insertion sort */

void ArraySort(int This[], CMPFUN fun_ptr, uint32 the_len)
{
  uint32 span;
  uint32 lb;
  uint32 ub;
  uint32 indx;
  uint32 indx2;

  if (the_len <= 1)
    return;

  span = INSERTION_SORT_BOUND;

  /* insertion sort the first pass */
  {
    int prev_val;
    int cur_val;
    int temp_val;

    for (lb = 0; lb < the_len; lb += span)
    {
      if ((ub = lb + span) > the_len) ub = the_len;

      prev_val = This[lb];

      for (indx = lb + 1; indx < ub; ++indx)
      {
        cur_val = This[indx];

        if ((*fun_ptr)(prev_val, cur_val) > 0)
        {
          /* out of order: array[indx-1] > array[indx] */
          This[indx] = prev_val; /* move up the larger item first */

          /* find the insertion point for the smaller item */
          for (indx2 = indx - 1; indx2 > lb;)
          {
            temp_val = This[indx2 - 1];
            if ((*fun_ptr)(temp_val, cur_val) > 0)
            {
              This[indx2--] = temp_val;
              /* still out of order, move up 1 slot to make room */
            }
            else
              break;
```

```
      }
      This[indx2] = cur_val; /* insert the smaller item right here */
    }
    else
    {
      /* in order, advance to next element */
      prev_val = cur_val;
    }
   }
  }
}

/* second pass merge sort */
{
  uint32 median;
  int* aux;

  aux = (int*) malloc(sizeof(int) * the_len / 2);

  while (span < the_len)
  {
    /* median is the start of second file */
    for (median = span; median < the_len;)
    {
      indx2 = median - 1;
      if ((*fun_ptr)(This[indx2], This[median]) > 0)
      {
        /* the two files are not yet sorted */
        if ((ub = median + span) > the_len)
        {
          ub = the_len;
        }

        /* skip over the already sorted largest elements */
        while ((*fun_ptr)(This[--ub], This[indx2]) >= 0)
        {
        }

        /* copy second file into buffer */
        for (indx = 0; indx2 < ub; ++indx)
        {
          *(aux + indx) = This[++indx2];
        }
        --indx;
        indx2 = median - 1;
        lb = median - span;
        /* merge two files into one */
        for (;;)
        {
          if ((*fun_ptr)(*(aux + indx), This[indx2]) >= 0)
          {
            This[ub--] = *(aux + indx);
            if (indx > 0) --indx;
            else
            {
              /* second file exhausted */
              for (;;)
              {
                This[ub--] = This[indx2];
                if (indx2 > lb) --indx2;
                else goto mydone; /* done */
```

```
            }
          }
        }
        else
        {
          This[ub--] = This[indx2];
          if (indx2 > lb) --indx2;
          else
          {
            /* first file exhausted */
            for (;;)
            {
              This[ub--] = *(aux + indx);
              if (indx > 0) --indx;
              else goto mydone; /* done */
            }
          }
        }
      }
    }
    mydone:
    median += span + span;
  }
  span += span;
 }

 free(aux);
 }
}

#define ARRAY_SIZE 100

int my_array[ARRAY_SIZE];

uint32 fill_array()
{
 int indx;
 uint32 sum = 0;

 for (indx=0; indx < ARRAY_SIZE; ++indx)
 {
  sum += my_array[indx] = rand();
 }
 return sum;
}

int cmpfun(int a, int b)
{
 if (a > b)
  return 1;
 else if (a < b)
  return -1;
 else
  return 0;
}

int main()
{
 int indx;
 uint32 checksum, checksum2;
 int ti ;
```

```
  XTmrCtr XPS_Timer ;

  checksum = fill_array();

  XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

  ArraySort(my_array, cmpfun, ARRAY_SIZE);

  XTmrCtr_Stop(&XPS_Timer,0) ;
    ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(ti) ;
    print("\r\n");

  checksum2 = my_array[0];

  for (indx=1; indx < ARRAY_SIZE; ++indx)
  {
    checksum2 += my_array[indx];
    if (my_array[indx - 1] > my_array[indx])
    {
      print("bad sort\n");
      return(1);
    }
  }

  if (checksum != checksum2)
  {
    print("bad checksum");
    putnum(checksum1);
    putnum(checksum2);
    print(".\n");
    return(1);
  }
  return(0);
}
```

Γ.10 quickSort1t.c

```
#include <stdlib.h>
#include <stdio.h>
#include "xtmrctr.h"
#include "xparameters.h"

#define INSERTION_SORT_BOUND 16 /* boundary point to use insertion sort */

#define uint32 unsigned int

typedef int (*CMPFUN)(int, int);

/* explain function
 * Description:
 *   fixarray::Qsort() is an internal subroutine that implements quick sort.
 *
 * Return Value: none
 */
void Qsort(int This[], CMPFUN fun_ptr, uint32 first, uint32 last)
{
  uint32 stack_pointer = 0;
  int first_stack[32];
  int last_stack[32];

  for (;;)
  {
    if (last - first <= INSERTION_SORT_BOUND)
    {
      /* for small sort, use insertion sort */
      uint32 indx;
      int prev_val = This[first];
      int cur_val;

      for (indx = first + 1; indx <= last; ++indx)
      {
        cur_val = This[indx];
        if ((*fun_ptr)(prev_val, cur_val) > 0)
        {
          /* out of order: array[indx-1] > array[indx] */
          uint32 indx2;
          This[indx] = prev_val; /* move up the larger item first */

          /* find the insertion point for the smaller item */
          for (indx2 = indx - 1; indx2 > first; )
          {
            int temp_val = This[indx2 - 1];
            if ((*fun_ptr)(temp_val, cur_val) > 0)
            {
              This[indx2--] = temp_val;
              /* still out of order, move up 1 slot to make room */
            }
            else
              break;
          }
          This[indx2] = cur_val; /* insert the smaller item right here */
        }
        else
        {
```

```
      /* in order, advance to next element */

      prev_val = cur_val;
    }
  }
}
else
{
  int pivot;

  /* try quick sort */
  {
    int temp;
    uint32 med = (first + last) >> 1;
    /* Choose pivot from first, last, and median position. */
    /* Sort the three elements. */
    temp = This[first];
    if ((*fun_ptr)(temp, This[last]) > 0)
    {
      This[first] = This[last]; This[last] = temp;
    }
    temp = This[med];
    if ((*fun_ptr)(This[first], temp) > 0)
    {
      This[med] = This[first]; This[first] = temp;
    }
    temp = This[last];
    if ((*fun_ptr)(This[med], temp) > 0)
    {
      This[last] = This[med]; This[med] = temp;
    }
    pivot = This[med];
  }
  {
    uint32 up;
    {
                        uint32 down;

      /* First and last element will be loop stopper. */
                        /* Split array into two partitions. */

                        down = first;
                        up = last;
                        for (;;)
                        {
                          do
                          {
                            ++down;
                          } while ((*fun_ptr)(pivot, This[down]) > 0);

                          do
                          {
                            --up;
                          } while ((*fun_ptr)(This[up], pivot) > 0);

                          if (up > down)
                          {
                            int temp;
                            /* interchange L[down] and L[up] */
                            temp = This[down]; This[down]= This[up]; This[up] = temp;
                          }
```

```
                        else
                          break;
                      }
                    }
                    {
                      uint32 len1; /* length of first segment */
                      uint32 len2; /* length of second segment */
                      len1 = up - first + 1;
                      len2 = last - up;
                      /* stack the partition that is larger */
                      if (len1 >= len2)
                      {
                        first_stack[stack_pointer] = first;
                        last_stack[stack_pointer++] = up;

                        first = up + 1;
                        /*  tail recursion elimination of
                         *  Qsort(This,fun_ptr,up + 1,last)
                         */
                      }
                      else
                      {
                        first_stack[stack_pointer] = up + 1;
                        last_stack[stack_pointer++] = last;

                        last = up;
                      }
                    }
        continue;
      }
      /* end of quick sort */
    }
    if (stack_pointer > 0)
    {
      /* Sort segment from stack. */
      first = first_stack[--stack_pointer];
      last = last_stack[stack_pointer];
    }
    else
      break;
  } /* end for */
}

void ArraySort(int This[], CMPFUN fun_ptr, uint32 the_len)
{
  Qsort(This, fun_ptr, 0, the_len - 1);
}

#define ARRAY_SIZE 100

int my_array[ARRAY_SIZE];

uint32 fill_array()
{
  int indx;
  uint32 checksum = 0;
  for (indx=0; indx < ARRAY_SIZE; ++indx)
  {
    checksum += my_array[indx] = rand();
  }
  return checksum;
```

```
}

int cmpfun(int a, int b)
{
  if (a > b)
    return 1;
  else if (a < b)
    return -1;
  else
    return 0;
}

int main()
{
  int indx;
  uint32 checksum1;
  uint32 checksum2 = 0;
  checksum1 = fill_array();
  int ti ;
  XTmrCtr XPS_Timer ;

  XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

  ArraySort(my_array, cmpfun, ARRAY_SIZE);

  XTmrCtr_Stop(&XPS_Timer,0) ;
    ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(ti) ;
    print("\r\n");

  for (indx=1; indx < ARRAY_SIZE; ++indx)
  {
    if (my_array[indx - 1] > my_array[indx])
    {
      //printf("bad sort\n");
      return(1);
    }
  }

  for (indx=0; indx < ARRAY_SIZE; ++indx)
  {
    checksum2 += my_array[indx];
  }

  if (checksum1 != checksum2)
  {
    print("bad checksum");
                        putnum(checksum1);
                        putnum(checksum2);
                        print(".\n");
    return(1);
  }

  return(0);

}
```

Γ.11 SelectionSort1t.c

```
#include <stdlib.h>
#include <stdio.h>
#include "xtmrctr.h"
#include "xparameters.h"

#define uint32 unsigned int

typedef int (*CMPFUN)(int, int);



void ArraySort(int This[], CMPFUN fun_ptr, uint32 the_len)
{
  /* selection sort */

  uint32 indx;
  uint32 indx2;
  uint32 large_pos;
  int temp;
  int large;

  if (the_len <= 1)
    return;

  for (indx = the_len - 1; indx > 0; --indx)
  {
    /* find the largest number, then put it at the end of the array */
    large = This[0];
    large_pos = 0;

    for (indx2 = 1; indx2 <= indx; ++indx2)
    {
      temp = This[indx2];
      if ((*fun_ptr)(temp ,large) > 0)
      {
        large = temp;
        large_pos = indx2;
      }
    }
    This[large_pos] = This[indx];
    This[indx] = large;
  }
}

#define ARRAY_SIZE 100

int my_array[ARRAY_SIZE];

void fill_array()
{
  int indx;

  for (indx=0; indx < ARRAY_SIZE; ++indx)
  {
    my_array[indx] = rand();
  }

  /* my_array[ARRAY_SIZE - 1] = ARRAY_SIZE / 3; */
}
```

```
int cmpfun(int a, int b)
{
  if (a > b)
    return 1;
  else if (a < b)
    return -1;
  else
    return 0;
}

int main()
{
  int indx;
  int indx2;
  int ti ;
  XTmrCtr XPS_Timer ;


    fill_array();

    XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

    ArraySort(my_array, cmpfun, ARRAY_SIZE);

    XTmrCtr_Stop(&XPS_Timer,0) ;
    ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(ti) ;
    print("\r\n");

    for (indx=1; indx < ARRAY_SIZE; ++indx)
    {
      if (my_array[indx - 1] > my_array[indx])
      {
        print("bad sort\n");
        return(1);
      }
    }


  return(0);
}
```

Γ.12 insertSort1t.c

```
#include <stdlib.h>
#include <stdio.h>
#include "xtmrctr.h"
#include "xparameters.h"


#define uint32 unsigned int

typedef int (*CMPFUN)(int, int);



void ArraySort(int This[], CMPFUN fun_ptr, uint32 the_len)
{
  /* insertion sort */

  uint32 indx;
  int cur_val;
  int prev_val;

  if (the_len <= 1)
    return;

  prev_val = This[0];

  for (indx = 1; indx < the_len; ++indx)
  {
    cur_val = This[indx];
    if ((*fun_ptr)(prev_val, cur_val) > 0)
    {
      /* out of order: array[indx-1] > array[indx] */
      uint32 indx2;
      This[indx] = prev_val; /* move up the larger item first */

      /* find the insertion point for the smaller item */
      for (indx2 = indx - 1; indx2 > 0;)
      {
        int temp_val = This[indx2 - 1];
        if ((*fun_ptr)(temp_val, cur_val) > 0)
        {
          This[indx2--] = temp_val;
          /* still out of order, move up 1 slot to make room */
        }
        else
          break;
      }
      This[indx2] = cur_val; /* insert the smaller item right here */
    }
    else
    {
      /* in order, advance to next element */
      prev_val = cur_val;
    }
  }
}
```

```
#define ARRAY_SIZE 100
 int my_array[ARRAY_SIZE];

uint32 fill_array()
{
  int indx;
  uint32 checksum = 0;
  for (indx=0; indx < ARRAY_SIZE; ++indx)
  {
    checksum += my_array[indx] = rand();
  }
  return checksum;
}

int cmpfun(int a, int b)
{
  if (a > b)
    return 1;
  else if (a < b)
    return -1;
  else
    return 0;
}

int main()
{
  int indx;
  int indx2;
  uint32 checksum1;
  uint32 checksum2;
  int ti ;
  XTmrCtr XPS_Timer ;


    checksum1 = fill_array();

    XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
    XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
    XTmrCtr_Reset(&XPS_Timer,0) ;
    XTmrCtr_Start(&XPS_Timer,0) ;

    ArraySort(my_array, cmpfun, ARRAY_SIZE);

                        XTmrCtr_Stop(&XPS_Timer,0) ;
    ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
    putnum(ti) ;
    print("\r\n");

    for (indx=1; indx < ARRAY_SIZE; ++indx)
    {
      if (my_array[indx - 1] > my_array[indx])
      {
        print("bad sort\n");
        return(1);
      }
    }
    checksum2 = 0;
    for (indx=0; indx < ARRAY_SIZE; ++indx)
    {
      checksum2 += my_array[indx];
    }
```

```
    if (checksum1 != checksum2)
    {
      print("bad checksum");
                          putnum(checksum1);
                          putnum(checksum2);
                          print(".\n");
    }

  return(0);
}
```

Γ.13 anazhthshCache.c

```
#include "xtmrctr.h"
#include "xparameters.h"
//#include "lib.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   int pin[50];
   int i;
   int ti ;
   XTmrCtr XPS_Timer ;

    microblaze_init_icache_range(0, XPAR_MICROBLAZE_0_CACHE_BYTE_SIZE);
    microblaze_enable_icache();

    microblaze_init_dcache_range(0, XPAR_MICROBLAZE_0_DCACHE_BYTE_SIZE);
    microblaze_enable_dcache();

   XTmrCtr_Initialize(&XPS_Timer,XPAR_XPS_TIMER_0_DEVICE_ID) ;
   XTmrCtr_SetResetValue(&XPS_Timer,0,0x00000000) ;
   XTmrCtr_Reset(&XPS_Timer,0) ;
   XTmrCtr_Start(&XPS_Timer,0) ;

   for(i=0;i<50;i++)
   {
         pin[i]=i++ * 2 + 3;
   }

    for(i=0;i<50;i++)
   {
         if(pin[i] == 33)
         {
                  print("h anazhthsh oloklhrw8hke epityxws kai o ari8mos 33 bre8hke sth 8esh tou
                  pinaka pin ");
                  putnum(i);
                  break;
         }
        else
         {
                  print("den bre8hke o ari8mos 33 sth 8esh ");
                  putnum(i);
         }
    }

   XTmrCtr_Stop(&XPS_Timer,0) ;
   ti = XTmrCtr_GetValue(&XPS_Timer,0) ;
   putnum(ti) ;
   print("\r\n");
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Οι πηγές της βιβλιογραφίας μας είναι ηλεκτρονικές και οι κύριες πηγές της πτυχιακής μας εργασίας είναι οι παρακάτω :

- Η εταιρία Xilinx που είναι η κατασκευάστρια εταιρία του FPGA board ML403
- Η uCLinux διανομή των Linux που ασχολείται κυρίως με λογισμικό, με εφαρμογή σε μικροεπεξεργαστές.
- Η Petalogix που ασχολείται με τη δημιουργία bsp για μικροεπεξεργαστές, κυρίως στη διανομή του uCLinux
- Η ηλεκτρονική εγκυκλοπαίδια Wikipedia
- Η μηχανή αναζήτησης της Google

Ηλεκτρονικοί σύνδεσμοι

http://www.google.com

http://www.uclinux.org/

http://www.xilinx.com/products/design_resources/proc_central/index.htm

http://www.xilinx.com/products/design_resources/proc_central/microblaze_faq.pdf

http://www.xilinx.com/support/documentation/sw_manuals/edk92i_mb_ref_guide.pdf

http://www.xilinx.com/support/documentation/ml403-edk.htm

http://www.xilinx.com/support/documentation/boards_and_kits/ug083.pdf

http://www.xilinx.com/onlinestore/v4_boards.htm

http://www.petalogix.com/

http://www.itee.uq.edu.au/~listarch/microblaze-uclinux/archive/2009/06/msg00023.htmlhttp://www.mhl.tuc.gr/Controller

http://www.microprocessor.sscc.ru/

http://www.jlc.tcu.edu.tw/Embedded System/952/Embedded System Design.ppt

http://elinux.org/upload/f/f8/TC-presentation-OLS-2008.ppt

http://www.eng.auburn.edu/~vagrawal/COURSE/E6200_06/STUDENT_TALKS/POWERPC(2).ppt

http://www.uclinux.org/pub/uClinux/dist/patches/

http://www.uclinux.org/pub/uClinux/dist/patches/

http://www.uclinux.org/ucsimm/

http://sourceforge.net/projects/uclinux/files/

http://en.wikipedia.org/wiki/Benchmark_(computing)

http://en.wikipedia.org/wiki/Field-programmable_gate_array

http://www.microcontroller.com/

http://www.howstuffworks.com/microcontroller.htm

http://www.kmitl.ac.th/~kswichit/

http://www.atmel.com/products/AVR/

http://www.cs.sunysb.edu/~cse320/Slides/PowerPC.ppt

http://aetos.it.teithe.gr/~adamidis/Prog_II/Sorting.pdf

http://delab.csd.auth.gr/~manolopo/Analysis/chapter8.pdf