Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Σημειώσεις Διδασκαλίας

Μάθημα

Ενσωματωμένα Υπολογιστικά Συστήματα

Εισαγωγικός Οδηγός Χρήσης Αναπτυξιακού Συστήματος EVK1100 για Μικροελεγκτές Atmel AVR32

Συγγραφή Σημειώσεων

Δάκης Παυλίδης, Εργαστηριακό Διδακτικό Προσωπικό
Μιχάλης Ψαράκης, Επίκουρος Καθηγητής

2013-2014

Περιεχόμενα

Πρόλογος

Αυτός ο οδηγός χρήσης απευθύνεται στους μεταπτυχιακούς φοιτητές τους Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς (ΜΠΣ «Προηγμένα Συστήματα Πληροφορικής») και έχει σαν σκοπό την εισαγωγή στους μικροελεγκτές 32bit της Atmel (σειρά AT32UC3A) και την ανάπτυξη εφαρμογών για αυτούς με τη βοήθεια της πλακέτας αξιολόγησης EVK1100 και των συνοδευτικών εργαλείων λογισμικού. Η συγγραφή του κρίθηκε απαραίτητη εξ' αιτίας της πληθώρας βιβλίων δεδομένων και εγχειριδίων από την Atmel, πολλά από τα οποία δεν είναι ιδιαίτερα κατατοπιστικά σε πρώτη ανάγνωση, με αποτέλεσμα να απαιτείται η συνεχής μετάβαση από το ένα κείμενο στο άλλο ή ακόμα και η επιστράτευση «αντίστροφής μηχανικής» (reverse enginnering) πάνω στα παραδείγματα που προσφέρονται από την εταιρεία.
Γίνεται μία εισαγωγή στο υλικό (μικροελεγκτής, πλακέτα) και στο λογισμικό ανάπτυξης και προγραμματισμού που προσφέρεται, με παράλληλες αναφορές στα αρχεία της εταιρείας που περιέχουν όλη τη σχετική λεπτομέρεια. Περιγράφεται η μεθοδολογία ανάπτυξης του λογισμικού με βάση τα εργαλεία που παρέχονται χρησιμοποιώντας έτοιμα παραδείγματα και αργότερα αναπτύσσοντας απλές εφαρμογές ξεκινώντας από το μηδέν μέχρι την φόρτωση του κώδικα πάνω στον μικροελεγκτή αλλά και δείχνοντας κάποιες έννοιες σχετικά με τη διαδικασία εκσφαλμάτωσης (debugging).
Οι γνώσεις που απαιτούνται είναι πάνω σε θέματα ψηφιακής λογικής, αρχιτεκτονικής υπολογιστών, προγραμματισμού σε γλώσσα C (η γλώσσα Assembly αποφεύγεται γιατί η εταιρεία δίνει πληθώρα βιβλιοθηκών σε C που υποστηρίζουν στην πλακέτα και τον μικροελεγκτή). Επίσης, χρήσιμες αλλά όχι απαραίτητες είναι βασικές γνώσεις σε θέματα ηλεκτρονικής.
Ο οδηγός αυτός συνοδεύεται και από CD που έχει ταξινομημένα πολλά έγγραφα τελευταίας έκδοσης (κατάλογος `Documentation`) που έχουν ανακτηθεί από την ιστοθέση της εταιρείας (`http://www.atmel.com`) καθώς και τις τελευταίες εκδόσεις του απαραίτητου λογισμικού υποστήριξης (κατάλογος `Software`), επειδή τα συνοδευτικά CD της αναπτυξιακής πλακέτας περιέχουν παλαιότερες εκδόσεις. Ο πηγαίος κώδικας των παραδειγμάτων που αναπτύσσονται στο Κεφάλαιο ΙΙΙ βρίσκεται στον κατάλογο `Development` και αποτελείται από ένα αρχείο `.c` για κάθε παράδεγμα.
Επί πλέον πληροφορίες, εκτός από τα αρχεία που βρίσκονται στην ιστοθέση της εταιρείας, μπορούν να βρεθούν και στην ειδική ιστοσελίδα της εταιρείας για υποστήριξη `http://support.atmel.no/bin/customer` καθώς και στην ιστοσελίδα `http://www.avrfreaks.net` και ιδαίτερα στον πίνακα συζητήσεων (forum).

I Το υλικό.

I.1 Γενικά για τους μικροελεγκτές και την οικογένεια Atmel AVR32.

Οι μικροελεγκτές είναι ειδικές κατηγορίες μικροεπεξεργαστών που ενσωματώνουν στο ίδιο ολοκληρωμένο κύκλωμα (chip) μία πληθώρα περιφερειακών κυκλωμάτων όπως ψηφιακές θύρες εισόδου/εξόδου (input/output ports), μετατροπείς αναλογικού σήματος σε ψηφιακό (analog-to-digital converters – ADC) και ψηφιακού σήματος σε αναλογικό (digital-to-analog converters – DAC), χρονόμετρα/χρονομετρητές (timers), μετρητές (counters) διάφορους ελεγκτές διασύνδεσης περιφερειακών συσκευών (USB, SPI, RS232 κ.α.) και πολλά άλλα.
Τα πεδία εφαρμογών των μικροελεγκτών είναι εφαρμογές αυτοματισμών (οικιακών, βιομηχανικών, καταναλωτικών) όπως για παράδειγμα η ηλεκτρονική μονάδα ελέγχου ψεκασμού και ανάφλεξης σε ένα σύγχρονο αυτοκίνητο (ECU), η μονάδα ελέγχου ενός πλυντηρίου ή η μονάδα ελέγχου μίας αντλίας που γεμίζει μία δεξαμενή με νερό και άλλα πολλά όπου χρειάζεται η αλληλεπίδραση παραμέτρων του περιβάλλοντος.
Η χρήση μικροελεγκτών αντί μικροεπεξεργαστών γενικού σκοπού σε εφαρμογές όπως οι παραπάνω έχει το πλεονέκτημα του μικρότερου όγκου, κατανάλωσης και φυσικά κόστους κατασκευής και σχεδίασης αφού τα περισσότερα από τα απαιτούμενα κυκλώματα εμπεριέχονται ήδη στο ολοκληρωμένο κύκλωμα του μικροελεγκτή, ενώ η χρήση γενικών μικροεπεξεργαστών θα απαιτούσε τη σχεδίαση και χρήση επί πλέον κυκλωμάτων (π.χ. ADC για ανάγνωση σημάτων από αισθητήρες θερμοκρασίας). Φυσικά, ο προσανατολισμός των μικροελεγκτών σε τέτοιες εφαρμογές γίνεται σε βάρος της απόδοσης (ταχύτητας) σε σχέση με έναν γενικού σκοπού μικοεπεξεργαστή αλλά η ταχύτητα δεν είναι το πρώτο ζητούμενο σε τέτοιες περιπτώσεις.
Έτσι, οι μικροελεγκτές γίνονται ένα βασικό στοιχείο σε ένα ενσωματωμένο σύστημα (embedded system), δηλαδή ένα σύστημα που υλοποιεί μία εφαρμογή για κάποιον χρήστη, περιέχει στοιχεία υλισμικού και λογισμικού, χωρίς όμως αυτά τα στοιχεία να είναι άμεσα ορατά στον χρήστη.
Οι μικροελεγκτές της οικογένειας AVR32 της εταιρείας Atmel έχουν στον πυρήνα τους έναν RISC επεξεργαστή των 32 bits, και εσωτερική μνήμη flash προγράμματος και δεδομένων. Επί πλέον, εκτός από την πληθώρα περιφερειακών κυκλωμάτων εισόδου/εξόδου που διαθέτουν, έχουν ελεγκτή διακοπών (interrupt controller), δυνατότητα για σύνδεση με εξωτερική μνήμη, διασύνδεση Direct Memory Access (DMA) και ανάλογα με την υποκατηγορία ή σειρά που ανήκει ο μικροελεγκτής (π.χ. στην AT32AP7xxx) είναι δυνατόν να υπάρχει υποστήριξη για χρήση λειτουργικού συστήματος εφ' όσον υπάρχει Μονάδα Διαχείριση Μνήμης (Memory Management Unit – MMU), συνεπεξεργαστής γραφικών και ενσωματωμένος ελεγκτής οθόνης, εκτεταμένο σύνολο εντολών για υποστήριξη Java και πολλά άλλα.
Το αναπτυξιακό σύστημα EVK1100 που διαθέτει το Εργαστήριο Υπολογιστικών Συστημάτων χρησιμοποιεί τον μικροελεγκτή AT32UC3A0512ES (ανήκει στην υποκατηγορία AT32UC3A). Αυτή η σειρά μικροελεγκτών είναι προσανατολισμένη για εφαρμογές χαμηλών ως μεσαίων απαιτήσεων.

I.2 Η αρχιτεκτονική της σειράς μικροελεγκτών AT32UC3A.

Η σειρά AT32UC3A διαθέτει έξι (6) μικροελεγκτές που διαφοροποιούνται μεταξύ τους στο μέγεθος της μνήμης που διαθέτουν, στο αν μπορεί να επεκταθεί εξωτερικά η μνήμη δεδομένων τους και ως προς τον αριθμό των ακροδεκτών τους και της συσκευασίας τους (chip package). Τα μέλη της σειράς παρουσιάζονται στον παρακάτω πίνακα μαζί με τα βασικά χαρακτηριστικά που τα διαφοροποιούν.

Εξάρτημα	Μνήμη Προγράμματος (Kbytes)	Μνήμη Δεδομένων (Kbytes)	Επέκταση Μνήμης	Συσκευασία Ακροδέκτες
AT32UC3A0128	128	32	NAI	LQFP 144
AT32UC3A0256	256	64	NAI	LQFP 144
AT32UC3A0512*	512	64	NAI	LQFP 144
AT32UC3A1128	128	32	OXI	TQFP 100
AT32UC3A1256	256	64	OXI	TQFP 100
AT32UC3A1512	512	64	OXI	TQFP 100

Εικόνα 1 : Η σειρά AT32UC3A

Η μείωση του αριθμού των ακροδεκτών από 144 σε 100 στην περίπτωση των τριών τελευταίων τύπων οφείλεται στην έλλειψη δυνατότητας επέκτασης της μνήμης δεδομένων με εξωτερική. Να σημειωθεί ότι κάθε τύπος μπορεί να έχει διαφορετικό κωδικό αναθεώρησης (revision) που συνεπάγεται κάποιες διαφοροποιήσεις κατά την ανάπτυξη των προγραμμάτων. Π.χ. η αναθεώρηση ES (Engineering Sample) που διαθέτουν η πλακέττες EVK1100 του εργαστηρίου μας απαιτούν διαφορετική εγκατάσταση λογισμικού βιβλιοθηκών (Software Framework στην ορολογία της Atmel) καθώς και διαφορετική παραμετροποίηση κατά το κτίσιμο (build) του εκτελέσιμου κώδικα και του προγραμματισμού του πάνω στο εξάρτημα. Από εδώ και στο εξής θα αναφερόμαστε στο εξάρτημα με κωδικό AT32UC3A0512ES που είναι ο μικροελεγκτής που διαθέτουν οι πλακέτες του εργαστηρίου (*).
Οι ελεγκτές της σειράς AT32UC3A μπορούν να χρονιστούν με ρολόι συχνότητας μέχρι 33MHz χωρίς κύκλους αναμονής (wait cycles) από τη μνήμη προγράμματος (Flash) ή μέχρι 66MHz με εισαγωγή όμως μέχρι το πολύ ενός κύκλου αναμονής. Η μνήμη δεδομένων είναι εσωτερική SRAM και μπορεί να επεκταθεί εξωτερικά με χρήση SDRAM ή SRAM σε ελεγκτές που το υποστηρίζουν.
Ο πυρήνας του επεξεργαστή επεξεργάζεται τα δεδομένα σε ομάδες των 32 bits αν και μπορούν να φορτωθούν και να αποθηκευτούν από τη μνήμη μισές λέξεις των 16 bits (half words) και λέξεις των 8 bits (bytes). Ο χώρος των διευθύνσεων είναι και αυτός 32 bits (0x00000000 – 0xFFFFFFFF). Στο χώρο των διευθύνσεων προβάλλεται η μνήμη (εσωτερική, προγράμματος και δεδομένων και εξωτερική) αλλά και οι περιφερειακές συσκευές που διεθέτει ο επεξεργαστής (memory mapped I/O).

Καταχωρητές.
Η σειρά AT32UC3A υλοποιεί τη γενική αρχιτεκτονική συνόλου εντολών της Atmel AVR32. Αυτή η αρχιτεκτονική διαθέτει 16 καταχωρητές γενικού σκοπού των 32 bits, συμπεριλαβανομένων του μετρητή προγράματος (program counter – PC), δείκτη στοίβας (stack pointer – SP), και καταχωρητή σύνδεσης/επιστροφής (link register – LR) από κλήση υπορουτίνας. Επί πλέον διαθέτει καταχωρητή κατάστασης (status register – SR) με κάθε bit του να έχει ειδική σημασία, όπως σημαία υπερχείλισης (overflow – O), κρατουμένου (carry – C), μάσκες για τα διάφορα επίπεδα διακοπών κ.τ.λ. Επίσης υπάρχει και μία πληθώρα ειδικών καταχωρητών συστήματος. Περισσότερες λεπτομέρεις μπορούν να βρεθούν στο τμήμα 9.3 του εγχειριδίου [1].

Καταστάσεις επεξεργαστή.
Ο επεξεργαστής μπορεί να εκτελεί εντολές κάθε στιγμή σε μία από οκτώ διαφορετικές καταστάσεις (modes): τις Non Maskable Interrupt, Exception, Interrupt3 - Interrupt0, Supervisor και την Application. Μεγαλύτερη προτεραιότητα έχει η Non Maskable Interrupt και μικρότερη οι Supervisor και Application. Όλες οι εντολές είναι διαθέσιμες σε όλες τις καταστάσεις εκτός από την κατάσταση Application. Η αλλαγή καταστάσεων γίνεται με λογισμικό, με διακοπές εξωτερικές ή με εξαιρέσεις λογισμικού (exceptions). Οι προτεραιότητες θεωρούνται υψηλότερες όταν έχουν χαμηλότερο αριθμό. Ο επεξεργαστής τίθεται σε κατάσταση

που έχει προτεραιότητα 1 έως 6 αν γίνει διακοπή που δεν επιδέχεται απενεργοποίηση (masking), εξαίρεση (exception), και διακοπή επιπέδου 3 ως 0, αντίστοιχα. Δεν μπορεί να γίνει αλλαγή μίας κατάστασης από διακοπή που αντιστοιχεί σε χαμηλότερη προτεραιότητα. Ο παρακάτω πίνακας συνοψίζει τις διάφορες καταστάσεις (mode) που μπορεί να βρεθεί ο επεξεργαστής.

Priority	Mode	Security	Description
1	Non Maskable Interrupt	Privileged	Non Maskable high priority interrupt mode
2	Exception	Privileged	Execute exceptions
3	Interrupt 3	Privileged	General purpose interrupt mode
4	Interrupt 2	Privileged	General purpose interrupt mode
5	Interrupt 1	Privileged	General purpose interrupt mode
6	Interrupt 0	Privileged	General purpose interrupt mode
N/A	Supervisor	Privileged	Runs supervisor calls
N/A	Application	Unprivileged	Normal program execution mode

Εικόνα 2 : Οι καταστάσεις του μικροελεγκτή

Εκτός από τις παραπάνω οκτώ καταστάσεις που θεωρούνται κανονικές (normal states), ο επεξεργαστής μπορεί να μπει και σε κατάσταση εκσφαλμάτωσης (debug state) που χρησιμοποιείται κατά τη διάρκεια ανάπτυξης μίας εφαρμογής.

Διακοπές και εξαιρέσεις.

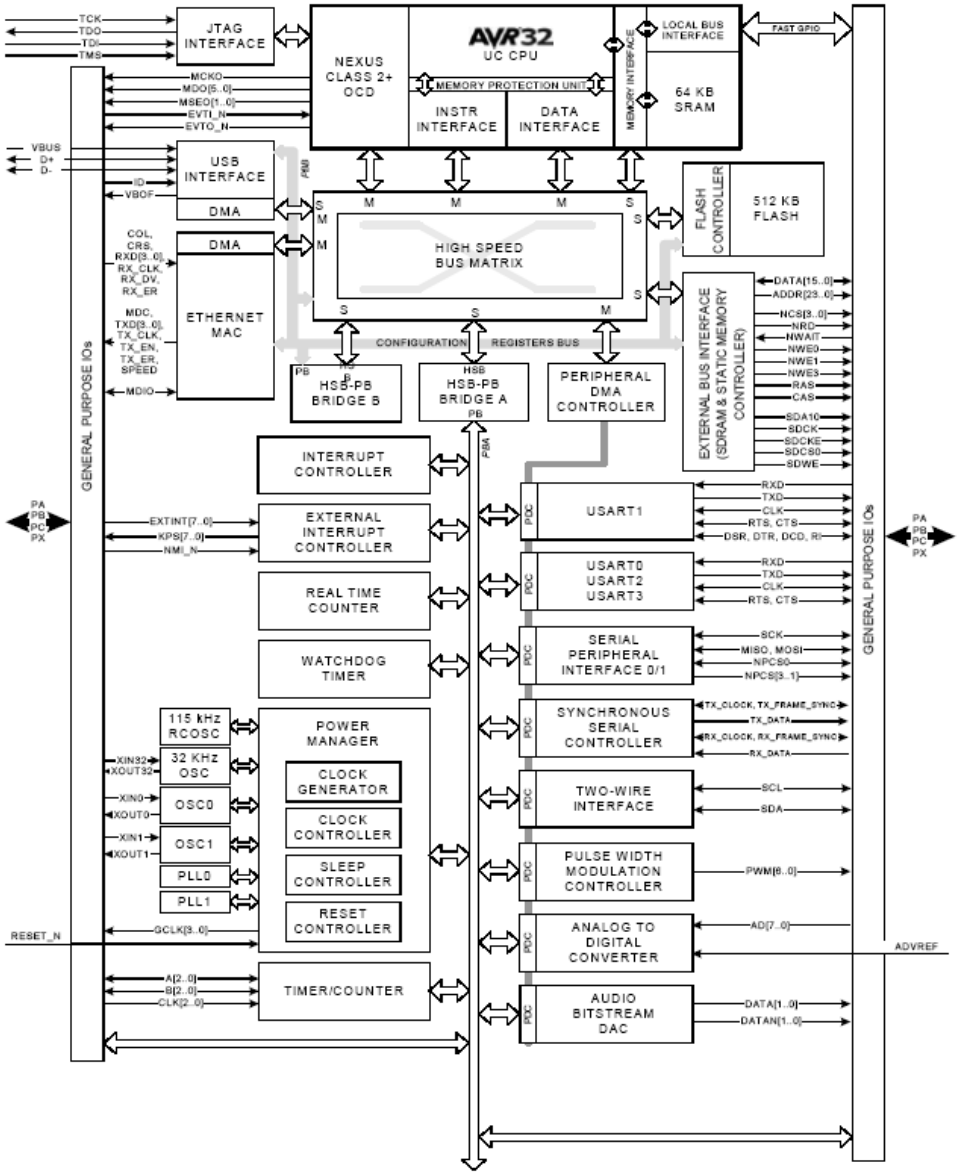
Οι μικροελεγκτές AVR32 έχουν τη δυνατότητα να διακόπτουν την κανονική ροή του προγράμματος και χειρίζονται άμεσα διάφορα έκτατα συμβάντα. Τα συμβάντα αυτά μπορεί να είναι σήματα διακοπής που προέρχονται από τους εξωτερικούς ακροδέκτες, από τα εσωτερικά περιφερειακά κυκλώματα ή από το ίδιο το λογισμικό όταν συμβαίνει ένα σφάλμα. Οι δύο πρώτες περιπτώσεις λέγονται διακοπές και η τελευταία λέξεται εξαίρεση. Οι διακοπές επεξεργάζονται από τα περιφερειακά κυκλώματα Interrupt Controller και External Interrupt Controller. Τα σήματα που τις προκαλούν μπορεί να προέρχονται από ένα περιφερειακό κύκλωμα που παρέλαβε κάποια δεδομένα και χρειάζονται άμεση επεξεργασία ή από ένα εξωτερικό κύκλωμα συνδεδεμένο σε κάποιον ακροδέκτη, π.χ. το πλήκτρο κάποιου πληκτρολογίου. Οι εξαιρέσεις δημιουργούνται από τον ίδιο τον πυρήνα του επεξεργαστή και μπορεί να είναι για παράδειγμα η διαίρεση ενός αριθμού με το μηδέν ή η προσπάθεια μή ευθυγραμμισμένης προσπέλασης στο χώρο μνήμης (unaligned access).

Όταν συμβεί ένα γεγονός διακοπής η ροή του προγράμματος (δηλαδή ο καταχωρητής PC) μετατίθεται σε μία ρουτίνα χειρισμού διακοπών (interrupt handler), δηλαδή σε μία διεύθυνση που εξαρτάται από το είδος της διακοπής. Πριν να γίνει όμως αυτό κάποιοι κρίσιμοι καταχωρητές όπως ο ίδιος ο PC, ο SR, και στις πιο πολλές περιπτώσεις οι R8-R12 και LR, αποθηκεύονται στη στοίβα, δηλαδή διαδοχικά στις διευθύνσεις που δείχνει ο δείκτης SP. Έτσι όταν τελειώσει η εξυπηρέτηση της διακοπής, δηλαδή όταν συναντηθεί η εντολή rete (return from exception), ανακτώνται αυτοί οι καταχωρητές από τη στοίβα και συνεχίζει κανονικά η ροή του προγράμματος έχοντας την ίδια κατάσταση (SR).

Οι διακοπές (εσωτερικές ή εξωτερικές) έχουν επίπεδα προτεραιότητας, και αν κατά τη διάρκεια εξυπηρέτησης μίας διακοπής έρθει μία καινούργια διακοπή, η δεύτερη εξυπηρετείται μόνο αν είναι υψηλότερης προτεραιότητας από την πρώτη. Ο πίνακας στην Εικόνα 2 καταγράφει τις προτεραιότητες (υψηλότερη προτεραιότητα αυτή με τον χαμηλότερο αριθμό). Ένα τέτοιο σχήμα επιτυγχάνεται με τη χρήση bits μασκαρίσματος που βρίσκονται στον SR. Έτσι, όταν ξεκινάει η εξυπηρέτηση μίας διακοπής τίθενται σε κατάσταση απενεργοποίησης («μασκαρίσματος», masking) όλα τα bits που αφορούν διακοπές χαμηλότερης προτεραιότητας ώστε αν αυτές συμβούν πριν τελειώσει η εξυπηρέτηση της τρέχουσας τότε, να αγνοηθούν μέχρι αυτή να τελειώσει.

Περιφερειακά κυκλώματα.

Εκτός από τον πυρήνα του επεξεργαστή και τις μνήμες,, οι μικροελεγκτές AT32UC3A διαθέτουν μία πληθώρα από περιφερειακά κυκλώματα, εσωτερικούς διαύλους (buses) για την επικοινωνία τους με τον πυρήνα και πολλούς ακροδέκτες εισόδου – εξόδου για την επικοινωνία με τον έξω κόσμο. Μία γενική άποψη της εσωτερικής δομής του μικροελεγκτή φαίνεται στην Εικόνα 3.



Εικόνα 3 : Εσωτερική δομή των μικροελεγκτών

Ακολουθεί σύντομη περιγραφή των περιφερειακών κυκλωμάτων.

- **PM** (Power Manager): Διαθέτει μεταξύ άλλων διάφορες γεννήτριες ρολογιών που χρησιμοποιούν κυκλώματα RC (σταθερά χρόνου ταλάντωσης ρυθμιζόμενη από αντίσταση και πυκνωτή), κυκλώματα κρυστάλλων για υψηλής ακρίβειας χρονισμό, PLLs (Phase Locked Loops) για πολλαπλασιασμό συχνότητας χρονισμού. Διαφορετικά ρολόγια μπορούν να οδεύσουν σε διαφορετικά περιφερειακά κυκλώματα και στον πυρήνα του επεξεργαστή ανάλογα με τις ανάγκες της εφαρμογής, ή να διακοπούν για τα κυκλώματα που δε χρησιμοποιούνται στην εφαρμογή προκειμένου να μειωθεί η κατανάλωση ενέργειας.

- **USART0-3** (Universal Synchronous Asynchronous Receiver Transmitter): Πρόκειται για κυκλώματα που χρησιμοποιούνται όταν χρειάζεται να υπάρχει σειριακή επικοινωνία ασύγχρονη (χωρίς ρολόι) ή σύγχρονη (με ρολόι). Ένα παράδειγμα εφαρμογής είναι η κατασκευή μίας θύρας RS-232 για επικοινωνία με άλλα κυκλώματα ή modems.
- **SPI** (Serial Peripheral Interface): Υλοποιεί το πρωτόκολλο SPI για επικοινωνία μικρών αποστάσεων με εξωτερικές συσκευές. Πρόκειται για δίαυλο τεσσάρων γραμμών σειριακής επικοινωνίας, όπου ένας master μπορεί να επικοινωνήσει με πολλούς slaves (έναν κάθε φορά) χρησιμοποιώντας κάποιο σχήμα διευθυνσιοδότησης. Συσκευές που υποστηρίζουν το πρωτόκολλο SPI είναι αισθητήρες, LCD, μνήμες FLASH, EEPROM, SD, MMC κ.ά.
- **SSC** (Synchronous Serial Controller): Ειδικό σειριακό πρωτόκολλο που χρησιμοποιείται κυρίως σε εφαρμογές ήχου και τηλεπικοινωνιών.
- **TWI** (Two Wire Interface): Ένα άλλο πρωτόκολλο σειριακής επικοινωνίας που χρησιμοποιεί όπως λέει και το όνομά του μόνο δύο αγωγούς. Είναι σχεδόν συμβατό με το πρωτόκολλο I^2C και οι εφαρμογές είναι η διασύνδεση με συσκευές παρόμοιες με αυτές του SPI.
- **PWM** (Pulse Width Modulator): Παρέχει μέχρι 7 ανεξάρτητα κανάλια τετραγωνικών κυματομορφών. Οι κυματομορφές είναι περιοδικές με ρυθμιζόμενη περίοδο, διάρκεια ενεργού παλμού, και φάσης. Τέτοιες κυματομορφές χρησιμοποιούνται στον έλεγχο βηματικών κινητήρων και σε ρύθμιση ισχύος (π.χ. ένταση μίας φωτεινής πηγής).
- **ADC** (Analog to Digital Converter): Μετατρέπει μέχρι 8 αναλογικά σήματα που έρχονται από τον έξω κόσμο σε ψηφιακά ακρίβειας 10 bits. Έτσι, μπορούν να μεταφερθούν στον επεξεργαστή σήματα από αισθητήρες, ήχος και άλλα αναλογικά σήματα.
- **Audio BitStream DAC** (Digital to Analog Converter): Μετατροπέας ψηφιακού σε αναλογικό δύο καναλιών για εφαρμογές ήχου. Η μετατροπή γίνεται με υπερδειγματοληψία θεωρώντας ότι το ψηφιακό σήμα έχει προέλθει από Δ-διαμόρφωση. Το κύκλωμα περιέχει επί πλέον FIR φίλτρα παρεμβολής.
- **TC** (Timer-Counter): Τρία ανεξάρτητα κανάλια χρονομέτρων/μετρητών των 16 bits για εφαρμογές μέτρησης συχνότητας, περιόδου, χρονικής απόστασης, καταμέτρησης συμβάντων κ.λπ.
- **RTC** (Real Time Clock): Ρολόι πραγματικού χρόνου για ώρα και ημερομηνία. Η ακρίβεια μέτρησης μπορεί να φτάσει τα 2 μsec περίπου ή να γίνει 1 sec με δυνατότητα απαρίθμησης 272 ετών ($\sim 2^{32}$ secs).
- **WDT** (WatchDog Timer): Ειδικό χρονόμετρο που αν ενεργοποιηθεί πρέπει ανά τακτά διαστήματα να μηδενίζεται από το πρόγραμμα, αλλιώς κάνει επανεκκίνηση του προγράμματος. Χρησιμοποιείται για να γίνεται βέβαιο ότι ο επεξεργαστής δε θα «κολλήσει» αν συμβούν απρόβλεπτα γεγονότα.
- **INTC** (Interrupt Controller): Κύκλωμα χειρισμού διακοπών που παράγονται από τα εσωτερικά περιφερειακά κυκλώματα που ήδη περιγράφονται.
- **EIC** (External Interrupt Controller): Κύκλωμα χειρισμού διακοπών που προέρχονται από εξωτερικούς ακροδέκτες της συσκευής.
- **Ethernet MAC**: Διασύνδεση για σύνδεση του μικροελεγκτή σε δίκτυα τύπου IEEE 802.3.
- **USB**: Διασύνδεση USB. Είναι συμβατό με την προδιαγραφή USB 2.0 εκτός από την ταχύτητα που περιορίζεται στα 12 MBits/s.
- **EIB** (External Bus Interface): Χρησιμοποιείται για τη σύνδεση εξωτερικής μνήμης (SRAM ή SDRAM) στον μικροελεγκτή.
- **JTAG**: Χρησιμοποιείται για τον προγραμματισμό του μικροελεγκτή. Επίσης είναι η διασύνδεση για να γίνει ο εργοστασιακός έλεγχος ορθής λειτουργίας του μικροελεγκτή (manufacturing testing).

Σύνδεση περιφερειακών με τον εξωτερικό κόσμο.
Σχεδόν όλα τα περιφερειακά κυκλώματα που προαναφέρθηκαν δεν έχουν απ' ευθείας σύνδεση με τους ακροδέκτες του μικροελεγκτή. Αυτό συμβαίνει γιατί τότε θα χρειάζονταν πάρα πολλοί ακροδέκτες. Αντί για αυτό, υπάρχουν τέσσερις ομάδες από ακροδέκτες (PA00-PA30, PB00-PB31, PC00-05, PX00-PX39). Κάθε ακροδέκτης έχει τρία διαφορετικά σήματα περιφερειακών συσκευών από τα οποία μπορεί να κάνει επιλογή σύνδεσης. Επίσης, είναι δυνατόν να ελεγχθεί ο ακροδέκτης απ' ευθείας από τον επεξεργαστή, δηλαδή να γίνει σήμα εξόδου που θα τεθεί σε λογικό 0 ή 1 ή σήμα εισόδου το οποίο μπορεί να διαβάζεται. Η δυνατότητες επιλογής που έχει ο κάθε ακροδέκτης παρουσιάζονται στις ενότητες 12.7 και 12.8 της αναφοράς [1].
Η δρομολόγηση αυτή των σημάτων γίνεται από ένα ειδικό ενσωματωμένο περιφερειακό που λέγεται GPIO Controller (ελεγκτής εισόδου/εξόδου γενικού σκοπού) με κατάλληλο προγραμματισμό του από την πυρήνα του επεξεργαστή.

Σύνδεση περιφερειακών με τον πυρήνα του επεξεργαστή.
Τα διάφορα περιφερειακά κυκλώματα συνδέονται με την κεντρική μονάδα επεξεργασίας με τη χρήση δύο διαύλων όπως φαίνεται στην Εικόνα 3. Οι δίαυλοι αυτοί με ονόματα PBA και PBB (Peripheral Bus A και B) συνδέονται σε έναν μεταγωγέα διαύλων (High Speed Bus Matrix) ο οποίος με τη σειρά του συνδέεται με άλλους διαύλους αποκλειστικής χρήσης για τις διεπαφές USB, Ethernet, Μνήμης Flash καθώς και με τους διαύλους εντολών και δεδομένων της μονάδα επεξεργασίας.

Χάρτης μνήμης και πρόσβαση στα περιφερειακά.
Η κεντρική μονάδα βλέπει με ενιαίο τρόπο όλους τους διαύλους και επομένως οι μνήμες και τα περιφερειακά έχουν συγκεκριμένες διευθύνσεις. Ο πίνακας στην Εικόνα 4 δείχνει τις διευθύνσεις έναρξης και το εύρος των διευθύνσεων για τη συσκευή AT32UC3A512.
Το κάθε περιφερειακό έχει ορισμένους καταχωρητές που καθορίζουν τη λειτουργία του (configuration registers), δείχνουν την κατάστασή του (status registers) και άλλους όπου γράφονται τα δεδομένα εισόδου ή εξόδου. Κάποιοι από αυτούς είναι μόνο ανάγνωσης, μόνο εγγραφής ή ανάγνωσης και εγγραφής. Ο κάθε καταχωρητής έχει μία σχετική διεύθυνση (offset address) η οποία πρέπει να προστεθεί στη διεύθυνση βάσης του συγκεκριμένου περιφερειακού για να γίνει πρόσβαση. Έτσι, η προσπέλαση των περιφερειακών γίνεται σαν να πρόκειται για απλή μνήμη δεδομένων με εντολές τύπου load και store. Η διαδικασία αυτή ονομάζεται είσοδος/έξοδος με χαρτογράφηση (ή απεικόνιση) μνήμης (memory-mapped I/O).

Περιφερειακή Συσκευή	Διεύθυνση Βάσης	Μέγεθος
Embedded SRAM	0x00000000	64 kbytes
Embedded Flash	0x80000000	512 kbytes
SRAM CS0	0xC0000000	16 Mbytes
SRAM CS2	0xC8000000	16 Mbytes
SRAM CS3	0xCC000000	16 Mbytes
SRAM CS1 /SDRAM CS0	0xD0000000	128 Mbytes
USB Configuration	0xE0000000	64 kbytes
Peripheral Bus A	0xFFFE0000	64 kbytes
Peripheral Bus B	0xFFFF0000	64 kbytes

Εικόνα 4 : Χάρτης Μνήμης

Ο πίνακας στην Εικόνα 5 απεικονίζει τη διεύθυνση βάσης για κάθε περιφερειακό μαζί με τον δίαυλο μέσω του οποίου γίνεται η διασύνδεση.

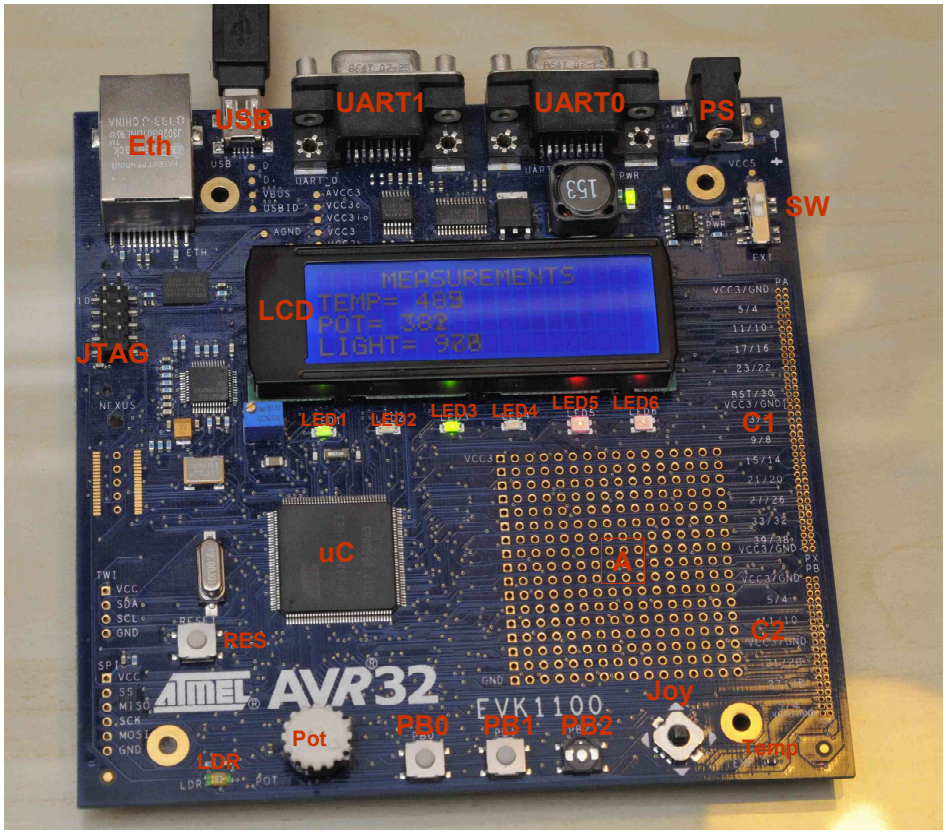
Περιφερειακή συσκευή	Διεύθυνση Βάσης	Δίαυλος
USB Slave	0xE0000000	HSB
USB Configuration	0xFFFE0000	PBB
High Speed Matrix	0xFFFE1000	PBB
Flash Controller	0xFFFE1400	PBB
MAC Configuration	0xFFFE1800	PBB
SRAM Controller	0xFFFE1C00	PBB
SDRAM Controller	0xFFFE2000	PBB
Peripheral DMA Controller	0xFFFF0000	PBA
Interrupt Controller	0xFFFF0800	PBA
Power Manager	0xFFFF0C00	PBA
Real Time Clock	0xFFFF0D00	PBA
WatchDog Timer	0xFFFF0D30	PBA
External Interrupt Controller	0xFFFF0D80	PBA
General Purpose IO Controller	0xFFFF1000	PBA
USART0	0xFFFF1400	PBA
USART1	0xFFFF1800	PBA
USART2	0xFFFF1C00	PBA
USART3	0xFFFF2000	PBA
Serial Peripheral Interface 0	0xFFFF2400	PBA
Serial Peripheral Interface 1	0xFFFF2800	PBA
Two Wire Interface	0xFFFF2C00	PBA
Pulse Width Modulation	0xFFFF3000	PBA
Synchronous Serial Controller	0xFFFF3400	PBA
Timer Counter	0xFFFF3800	PBA
Analog to Digital Converter	0xFFFF3C00	PBA

Εικόνα 5 : Διευθύνσεις Βάσης Περιφερειακών

Έτσι, αν πρέπει για παράδειγμα να γίνει ανάγνωση των δεδομένων που έχουν προέλθει από την μετατροπή του αναλογικού σήματος σε ψηφιακό του καναλιού 2 του ADC θα πρέπει να γίνει φόρτωση από τη διεύθυνση 0xFFFF3C38 επειδή η διεύθυνση βάσης του ADC είναι 0xFFFF3C00 και η διεύθυνση μετατόπισης του καταχωρητή ADC_CDR2 του ADC που περιέχει τα δεδομένα είναι 0x38 (βλ. Ενότητα 33.7 της αναφοράς [1]).

I.3 Η πλακέτα ανάπτυξης εφαρμογών EVK1100.

Η Atmel παρέχει την πλακέτα ανάπτυξης EVK1100 για τους μικροελεγκτές AVR32. Με αυτή την πλακέτα μπορεί κάποιος αρχάριος χωρίς ιδιαίτερες γνώσεις ηλεκτρονικών να αναπτύξει και να δοκιμάσει γρήγορα μία εφαρμογή και να εστιαστεί κυρίως στο λογισμικό, χωρίς να απαιτείται να σχεδιάζει εξ' αρχής κύκλωμα που θα συνδυάζει τη συσκευή του μικροελεγκτή μαζί με τα απαραίτητα στοιχεία για τη λειτουργία του, όπως συνδετήρες (connectors), κυκλώματα τροφοδοσίας και χρονισμού και άλλα κυκλώματα διασύνδεσης. Άλλωστε η συσκευασία (package) του μικροελεγκτή είναι τεχνολογίας SMD (Surface Mounted Device) κάτι που κάνει δύσκολες τις συγκολλήσεις χωρίς ειδικούς σταθμούς κόλλησης. Αλλά και ο σχεδιασμός της πλακέτας δεν είναι απλή υπόθεση γιατί δύσκολα θα αποφευχθεί η χρήση λιγότερων από τρία επίπεδα αγωγών, λόγω της πληθώρας των σημάτων του μικροελεγκτή.



Εικόνα 6 : Η πλακετα EVK1100

Στην Εικόνα 6 παρουσιάζεται η πλακέτα EVK1100 μαζί με ετικέτες πάνω σε εξαρτήματά της, των οποίων η λειτουργία περιγράφεται πιο κάτω:

- **uC**: Είναι η ο μικροελεγκτής που στην περίπτωση των πλακετών του εργαστηρίου είναι ο AT32UC3A0512ES (Engineering Sample).
- **PS**: Συνδετήρας δύο επαφών τύπου 5.5/2.1mm για την τροφοδοσία των κυκλωμάτων με συνεχές ρεύμα τάσης 8-20V. Η θετική πολικότητα εφαρμόζεται στον κεντρικό αγωγό. Κυκλώματα σταθεροποίησης πάνω στην πλακέτα υποβιβάζουν την τάση στην τάση λειτουργίας του κυκλώματος που είναι 3.3V. Αν και υπάρχει προστατευτική δίοδος για την περίπτωση λανθασμένης εφαρμογής πολικότητας καλό είναι να προσεχθεί η πολικότητα.
- **USB**: Θύρα USB. Μπορεί να χρησιμοποιηθεί για σύνδεση με προσωπικό υπολογιστή προκειμένου να γίνει μέσω αυτής ο προγραμματισμός του μικροελεγκτή. Επίσης, η θύρα αυτή μπορεί να χρησιμοποιηθεί για την τροφοδοσία της πλακέτας σε περίπτωση που δεν είναι διαθέσιμο εξωτερικό τροφοδοτικό.
- **SW**: Διακόπτης επιλογής τροφοδοσίας, είτε από το εξωτερικό τροφοδοτικό (θέση EXT), είτε από τη θύρα USB (θέση USB).
- **RES**: Διακόπτης πίεσης (push button) για Reset. Κάνει επανεκκίνηση του επεξεργαστή.
- **UART0** και **UART1**: Σειριακές θύρες RS-232 που συνδέονται στα αντίστοιχα εσωτερικά περιφερειακά για τη διασύνδεση του μικροελεγκτή.
- **Eth**: Διασύνδεση Ethernet για σύνδεση του μικροελεγκτή με δίκτυο. Εκμεταλεύεται το σχετικό περιφερειακό του μικροελεγκτή.
- **JTAG**: Χρησιμοποιείται σε συνδυασμό με άλλες συσκευές της Atmel όπως ή JTAG ICE MkII για τον προγραμματισμό του μικροελεγκτή αλλά και για εκσφαλμάτωση κατά τη διαδικασία ανάπτυξης.

- **A**: Περιοχή γενικής χρήσης στην πλακέτα για κατασκευή επί πλεόν κυκλωμάτων.
- **C1**: Θέση για συγκόλληση συνδετήρα 2x40 ώστε να επεκταθεί το κύκλωμα προς άλλες πλακέτες. Τα σήματα που είναι διαθέσιμα στον συνδετήρα είναι κυρίως σήματα από τις θύρες εισόδου/εξόδου του μικροελεγκτή PA, PX.
- **C2**: Θέση για συγκόλληση συνδετήρα 2x20 ώστε να επεκταθεί το κύκλωμα προς άλλες πλακέτες. Τα σήματα που είναι διαθέσιμα στον συνδετήρα είναι κυρίως σήματα από τις θύρες εισόδου / εξόδου του μικροελεγκτή PB.
- **LCD:** Οθόνη 4 γραμμών και 20 χαρακτήρων ανά γραμμή για εμφάνιση μηνυμάτων. Η οθόνη συνδέεται στη διασύνδεση SPI1 του μικροελεγκτή.
- **LED1, LED2,.., LED6**: Ενδεικτικά LEDs που συνδέονται σε ακροδέκτες της γενικής θύρας PB. Από αυτά τα δύο πρώτα μπορούν να ανάψουν ή να σβήσουν μόνο. Τα δύο επόμενα μπορούν να οδηγηθούν και από PWM κανάλια, επομένως μπορεί να ρυθμιστεί η φωτεινότητά τους. Τα δύο τελευταία είναι δίχρωμα LED (πράσινο και κόκκινο) με το κάθε χρώμα να έχει δυνατότητα σύνδεσης σε ένα κανάλι PWM και έτσι και αυτά μπορούν να έχουν ρυθμιζόμενη ένταση για το κάθε χρώμα.
- **PB0, PB1, PB2**: Διακόπτες πίεσης που οδηγούνται σε ακροδέκτες της θύρας PX. Χρησιμοποιούνται σαν ψηφιακά σήματα εισόδου για τον μικροελεγκτή. Οι είσοδοι όπου αυτοί οδηγούνται έχουν pull-up αντίσταση, έτσι αν δεν είναι πατημένοι το σήμα είναι σε λογικό ‘1’. Αν πατηθούν τότε, γίνεται λογικό ‘0’.
- **Joy**: Μικρός μοχλοδιακόπτης (joystick). Μπορεί να κινηθεί πάνω-κάτω και αριστερά-δεξιά. Επίσης μπορεί να πιεστεί. Τα πέντε σήματα οδηγούνται σε ακροδέκτες της θύρας PA. Όπως και πριν αν είναι ελεύθερος από πίεση δίνει λογικό ‘1’ στις γραμμές εισόδου του μικροελεγκτή, διαφορετικά (αν πατιέται) δίνει λογικό ‘0’.
- **Pot**: Ποτενσιόμετρο που μπορεί να ρυθμίζει μία αναλογική τάση προς κάποιο κανάλι του ADC.
- **LDR**: Αντίσταση που μεταβάλλεται από το φως (Light Dependent Resistor). Το σήμα που δίνει οδηγείται σε ένα κανάλι του ADC και δίνει τόσο μεγαλύτερη ψηφιακή τιμή όσο αυξάνεται το φως που πέφτει πάνω της.
- **Temp**: Θερμίστορ, δηλαδή αντίσταση με μεγάλο συντελεστή θερμοκρασιακής μεταβολής. Το σήμα οδηγείται από αυτήν την αντίσταση προς ένα κανάλι του ADC και μεταβάλλεται αντιστρόφως ανάλογα με τη θερμοκρασία.

Με τα παραπάνω περιφερειακά μπορεί να προσομοιωθεί μία εφαρμογή χωρίς να χτιστούν όλα τα μέρη της. Π.χ μπορεί ένας ανιχνευτής στάθμης υγρού να προσομοιωθεί από το ποτενσιόμετρο και να αναπτυχθεί σε αυτή τη βάση το λογισμικό. Όταν πιστοποιηθεί η λειτουργία του τότε, μπορεί να σχεδιαστεί και να συνδεθεί το κύκλωμα του αισθητήρα της στάθμης υγρού. Η φωτεινότητα των LEDs θα μπορούσε να προσομοιώνει την ταχύτητα βηματικών κινητήρων ελέγχου ενός ταινιοδρόμου κ.ο.κ.
Το αναλυτικό κύκλωμα της πλακέτας EVK1100 φαίνεται στην αναφορά [2].
Η πλακέτα είναι πακεταρισμένη μαζί με καλώδιο USB που επιτρέπει τη σύνδεση με το PC, καλώδιο δικτύου, καλώδιο τροφοδοσίας (χωρίς τροφοδοτικό) και CD (Technical Library 2007) με τεκμηρίωση και λογισμικό της Atmel, CD με διαφορετικό λογισμικό ανάπτυξης από αυτό της Atmel (IAR Embedded Workbench) και CD με λειτουργικό σύστημα πραγματικού χρόνου ThreadX RTOS V5.0.
Ο μικροελεγκτής έρχεται προγραμματισμένος με εφαρμογή (Control Panel Demo) που επιδεικνύει μεγάλο μέρος από τις δυνατότητες της πλακέτας (είσοδος από όλους τους αισθητήρες και διακόπτες, έξοδος προς τα LED, το LCD, επικοινωνία με το PC με χρήση δικτύου, USB ή σειριακής θύρας κ.τ.λ.). Περισσότερες πληροφορίες για την εφαρμογή επίδειξης υπάρχει στις αναφορές [3] και [4]. Η αναφορά [4] είναι μέρος της τεκμηρίωσης html που υπάρχει στο Software Framework.

I.4 Ο προγραμματιστής/εκσφαλματωτής JTAGICE mkII.

Η συσκευή JTAGICE mkII μπορεί να προγραμματίσει οποιονδήποτε μικροελεγκτή AVR32 με τη χρήση του JTAG interface, αλλά το σημαντικότερο είναι ότι μπορεί να συνεργαστεί με το κύκλωμα OCD (On-Chip Debug) που διαθέτουν οι μικροελεγκτές και να επιτρέψει την εκσφαλμάτωση μίας εφαρμογής με τεχνικές που θα περιγραφούν στην Ενότητα III.7.

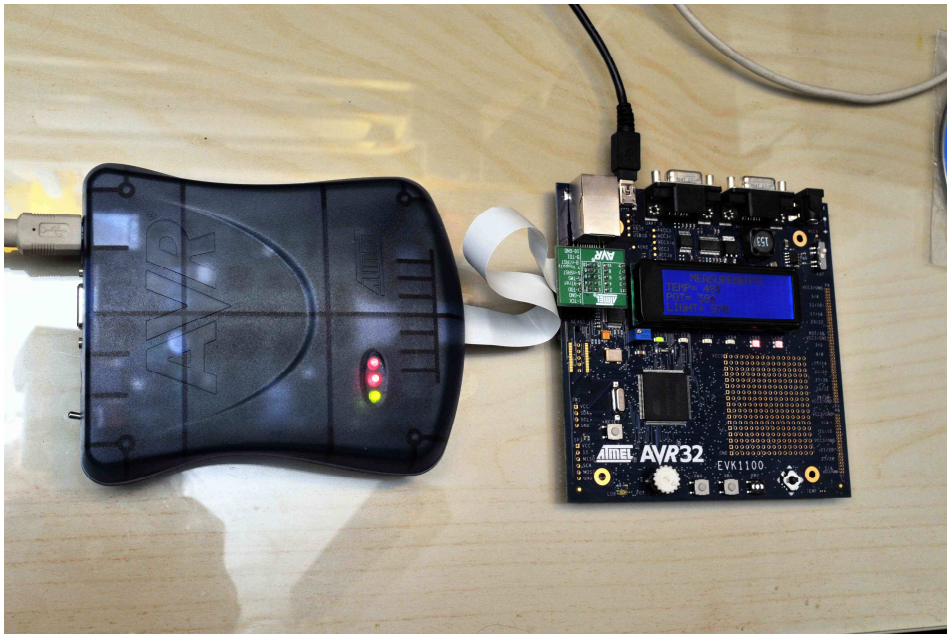
Εικόνα 7 : Ο προγραμματιστής JTAGICE mkII

Η Εικόνα 7 δείχνει τη συσκευή και τις υποδοχές διασύνδεσης με τον υπολογιστή και την πλακέτα που φιλοξενεί τον μικροελεγκτή, και τα ενδεικτικά LEDs. Η διασύνδεση με τον υπολογιστή γίνεται είτε με USB είτε με RS232 θύρα. Στην περίπτωση της USB διασύνδεσης μπορεί να υποστηριχθεί και η τροφοδοσία της εφ' όσον η θύρα USB μπορεί να δώσει ρεύμα 500mA που απαιτεί η συσκευή. Σε διαφορετική περίπτωση μπορεί να χρησιμοποιηθεί η υποδοχή για εξωτερική τροφοδοσία (PS) από συνεχή τάση 9V-15V ή από εναλλασόμενη 9V. Ο διακόπτης (SW) ανοίγει και κλείνει την παροχή της τροφοδοσίας.
Τα ενδεικτικά LEDs έχουν τη σημασία που δείχνει ο πίνακας στην Εικόνα 8.

LED	Χρώμα	Σημασία
1	Πράσινο	Υπάρχει σύνδεση με τον μικροελεγκτή
2	Κόκκινο	JTAGICE τροφοδοτείται
3	Πράσινο Κίτρινο Κόκκινο Σβηστό	Μεταφορά δεδομένων Αναβάθμιση firmware/Εκκίνηση Ανενεργό, Χωρίς Σύνδεση με AVR32 Studio Ανενεργό, Σύνδεση με AVR32 Studio
4	Αναβοσβήνει	Μεταφορά δεδομένων

Εικόνα 8 : Ενδεικτικά LEDs του JTAGICE mkII

Η θύρα RS232 μπορεί να φτάσει την ταχύτητα 115200bps και η αρχική της ρύθμιση είναι 19200bps με χρήση RTS/CTS (Hardware Control).
Η σύνδεση με την πλακέτα EVK1100 γίνεται με τη χρήση της καλωδιοταινίας (JTAG) όπως φαίνεται στην Εικόνα 9.



Εικόνα 9 : Σύνδεση EVK1100 – JTAGICE mkII

Λεπτομερής περιγραφή των δυνατοτήτων και της χρήσης της συσκευής δίνονται στην ιστοσελίδα http://support.atmel.no/bin/customer (επιλογή AVR Tools Online Help → JTAGICE mkII).

II Το λογισμικό.

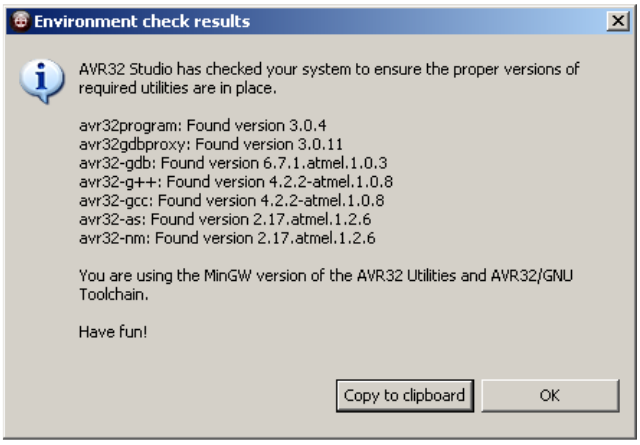
II.1 Επισκόπηση των εργαλείων.

Η ανάπτυξη πολύπλοκων εφαρμογών απαιτεί τη χρήση γλωσσών υψηλού επιπέδου γιατί είναι ευκολότερη η επαναχρησιμοποίηση κομματιών κώδικα από την ίδια ή από άλλες εφαρμογές. Επιπλέον, συνήθως σε ένα περιβάλλον γλώσσας υψηλού επιπέδου προσφέρονται βιβλιοθήκες με έτοιμες συναρτήσεις για την επίλυση συνηθισμένων προβλημάτων. Το σύνολο των εργαλείων που προσφέρονται από την Atmel είναι προσανατολισμένο στη χρήση της γλώσσας C. Η χρήση συμβολικής γλώσσας (assembly) της αρχιτεκτονικής AVR32 δε συνίσταται παρά μόνο σε κρίσιμα από απόψεως ταχύτητα σημεία. Θα φανεί άλλωστε πόσο διευκολύνει το χρήστη η χρήση της γλώσσας C, αποκρύπτοντας από τον προγραμματιστή αρκετές λεπτομέρειες από το υλικό, όπως για παράδειγμα ποιοι ακριβώς καταχωρητές του ADC πρέπει να προγραμματιστούν πριν να μπορέσει να γίνει εκμετάλλευσή του ή με ποιον τρόπο δέχεται εντολές η μονάδα εμφάνισης χαρακτήρων LCD. Αυτό συμβαίνει γιατί υπάρχουν ήδη έτοιμες βιβλιοθήκες με οδηγούς (drivers) που υποστηρίζουν την πλακέτα EVK1100 και τα εσωτερικά περιφερειακά κυκλώματα.
Τα απαραίτητα εργαλεία για την ανάπτυξη μίας εφαρμογής περιγράφονται παρακάτω (δίνονται και οι σχετικές αναφορές για μία πρώτη ανάγνωση και βοήθεια στην εγκατάσταση). Οι εκδόσεις που αναφέρονται είναι οι τρέχουσες μέχρι και το Σεπτέμβριο 2008.

- AVR 32 Studio [5].
  Είναι ένα ολοκληρωμένο γραφικό περιβάλλον γραφής κώδικα σε C, C++, Assembly, εκσφαλμάτωσης, και προγραμματισμού συσκευών της Atmel. Μεταξύ άλλων προσφέρει θέαση καταχωρητών και μνήμης του μικροελεγκτή και έχει ενσωματωμένο εγχειρίδιο βοήθειας (on-line help). Αυτό το περιβάλλον πρέπει να συνεργάζεται με έναν μεταγλωτιστή που στην προκειμένη περίπτωση είναι μέρος του πακέτου AVR 32 Gnu Toolchain. Η τρέχουσα έκδοση του AVR 32 Studio είναι η 2.0.2.
- AVR 32 Gnu Toolchain
  Είναι μία σειρά από εργαλεία γραμμής εντολών, όπως compilers και debuggers, assemblers, προγραμματιστές συσκευών AVR32 και βιβλιοθήκες C, που το ένα συνεργάζεται με το άλλο. Αυτό το πακέτο τρέχει σε λειτουργικά συστήματα Windows και Linux. Η τρέχουσα έκδοση είναι η 2.0.3.
- AVR 32 UC3A Software Framework [6]
  Πρόκειται για μία συλλογή βιβλιοθηκών και οδηγών λογισμικού σε γλώσσα C για την υποστήριξη όλων των εσωτερικών περιφερειακών μικροελεγκτή και των περιφερειακών κυκλωμάτων της πλακέτας. Περιέχει επίσης και έτοιμες εφαρμογές επίδειξης. Το λογισμικό ενσωματώνεται στο AVR 32 Studio και προσφέρει εκτενή βοήθεια μέσω του περιβάλλοντος. Η τρέχουσα έκδοση που είναι κατάλληλη για τον μικροελεγκτή της πλακέτας είναι η "AVR32 UC3A Software Framework 1.2.1ES for Engineering Samples".
- Flip [7]
  Είναι λογισμικό που τρέχει αυτόνομα σε Windows ή Linux και μπορεί να κάνει τον προγραμματισμό ενός AVR32 ενώ αυτός είναι πάνω σε πλακέτα μέσω της θύρας USB (FLexible In system Programmer). Επίσης, ενσωματώνεται και στο AVR 32 Studio για να μπορεί να γίνει ο προγραμματισμός μέσα από το ολοκληρωμένο περιβάλλον. Η τρέχουσα έκδοση του Flip είναι η 3.3.1.

ΙΙ.2 Διαδικασία εγκατάστασης.

Το συνοδευτικό CD του οδηγού χρήσης περιέχει στον κατάλογο `Software` τις τελευταίες εκδόσεις που υπήρχαν κατά τη στιγμή συγγραφής. Ένας έλεγχος στην ιστοθέση της Atmel (`http://www.atmel.com/avr32`) θα επιβεβαιώσει αν υπάρχουν ή όχι νεότερες εκδόσεις.
Η εγκατάσταση ξεκινάει με το πακέτο AVR32 Gnu Toolchain. Οι απαιτήσεις για την εγκατάσταση είναι Windows 2000 ή XP ή Linux Fedora Core 4, 5 ή 6, Ubuntu Linux 6.06 (Dapper) ή SUSE Linux 10.2. Το αρχείο εγκατάστασης έχει όνομα `avr32-gnu-toolchain-2.0.3.exe` και η εγκατάσταση γίνεται εξ' ορισμού κάτω από τον κατάλογο `C:\Program Files\Atmel\AVR Tools\AVR32 Toolchain`.
Κατόπιν, γίνεται η εγκατάσταση του AVR 32 Studio. Από απόψεως υλικού απαιτείται επεξεργαστής Pentium 4 με χρονισμό 1GHz και μνήμη 512Mbytes. Το πρόγραμμα είναι αρκετά βαρύ οπότε σε υποδεέστερο υπολογιστή θα υπάρχουν αρκετές καθυστερήσεις. Από απόψεως λογισμικού οι απαιτήσεις είναι ίδιες με τις προηγούμενες. Σύμφωνα με την εταιρεία δεν έχει πιστοποιηθεί η συμβατότητα με Windows Vista. Επίσης, πρέπει να υπάρχει Sun Java 2 Platform 1.5 ή νεότερη στα Linux συστήματα. Στα Windows είναι ενσωματωμένη στο αρχείο εγκατάστασης. Επί πλέον χρειάζεται κάποιος φυλλομετρητής Internet για την εμφάνιση των αρχείων της ενσωματωμένης βοήθειας (online help). Το AVR 32 Studio ενημερώνεται αυτόματα μέσω δικτύου για την ύπαρξη νεότερων εκδόσεων για τα υποσυστήματά του.
Η εγκατάσταση γίνεται τρέχοντας απλώς το αρχείο `AVR32Studio-2.0-Setup.exe`. Η εγκατάσταση γίνεται εξ' ορισμού κάτω από τον κατάλογο `C:\Program Files\Atmel\AVR Tools\AVR32 Studio 2.0` . Την πρώτη φορά χρήσης του προγράμματος ζητείται από το χρήστη να ορίσει έναν κατάλογο βάσης στο σύστημα αρχείων όπου θα αποθηκεύονται όλες οι εργασίες του (workspace).
Μετά την εγκατάσταση για την πιστοποίηση ότι και τα δύο πρώτα πακέτα έχουν εγκατασταθεί σωστά από το AVR 32 Studio γίνεται έλεγχος με την επιλογή `Help → AVR32 Studio → Check Environment`. Η Εικόνα 10 δείχνει ένα παράδειγμα σωστής εγκατάστασης. Επίσης ένας έλεγχος στα εγκατεστημένα προγράμματα θα δείξει ότι υπάρχει εγκατεστημένη και η Java.
Το AVR 32 UC3A Software Framework έχει ήδη εγκατασταθεί σαν μέρος της προηγούμενης εγκατάστασης, παρόλα αυτά δίνεται και ξεχωριστά σαν ένα συμπιεσμένο αρχείο προκειμένου ο χρήστης να μελετήσει τους πηγαίους κώδικες των παραδειγμάτων και να έχει ανεξάρτητη πρόσβαση στα σχετικά εγχειρίδια.
Ακολουθεί η εγκατάσταση του Flip. Το Flip είναι απαραίτητο σε περίπτωση που δε διατίθεται προγραμματιστής όπως ο JTAGICE mkII και θέλουμε να γίνεται ο προγραμματισμός από το περιβάλλον AVR 32 Studio με χρήση USB ή RS232 θύρας. Οι απαιτήσεις του είναι Windows 9x/Me/NT/2000/XP ή Linux καθώς και η ύπαρξη Java Runtime Environment. Σε περίπτωση που το τελευταίο δεν υπάρχει, διατίθεται αρχείο εγκατάστασης που περιλαμβάνει το Flip και το JRE. Το αρχείο αυτό έχει όνομα `JRE - Flip Installer - 3.3.1.exe`. Η εγκατάσταση γίνεται κάτω από τον κατάλογο `C:\Program Files\Atmel\Flip 3.3.1`.
Το πρόγραμμα Flip είναι αυτόνομο, αλλά εδώ χρειάζεται να γίνει η σύνδεση ενός υποσυνόλου του με όνομα BatchISP στο AVR32 Studio. Για να γίνει αυτό, πρέπει να συνδεθεί η πλακέτα με το USB καλώδιο στο PC και να τεθεί σε κατάσταση αναμονής για ISP (In System Programming). Αυτό γίνεται πατώντας το διακόπτη Reset, ύστερα το μοχλοδιακόπτη (joystick) και έχοντας κρατημένο το μοχλοδιακόπτη να απελευθερωθεί ο Reset. Τότε το PC θα αναγνωρίσει μία νέα συσκευή και θα αναζητήσει λογισμικό οδήγησης για αυτό. Ο οδηγός θα πρέπει να αναζητηθεί στον κατάλογο `C:\Program Files\Atmel\Flip 3.3.1\usb`. Ο τρόπος αναζήτησης διαφέρει ανάλογα με την έκδοση των Windows. Βλέπε αναλυτική περιγραφή στην αναφορά [7].



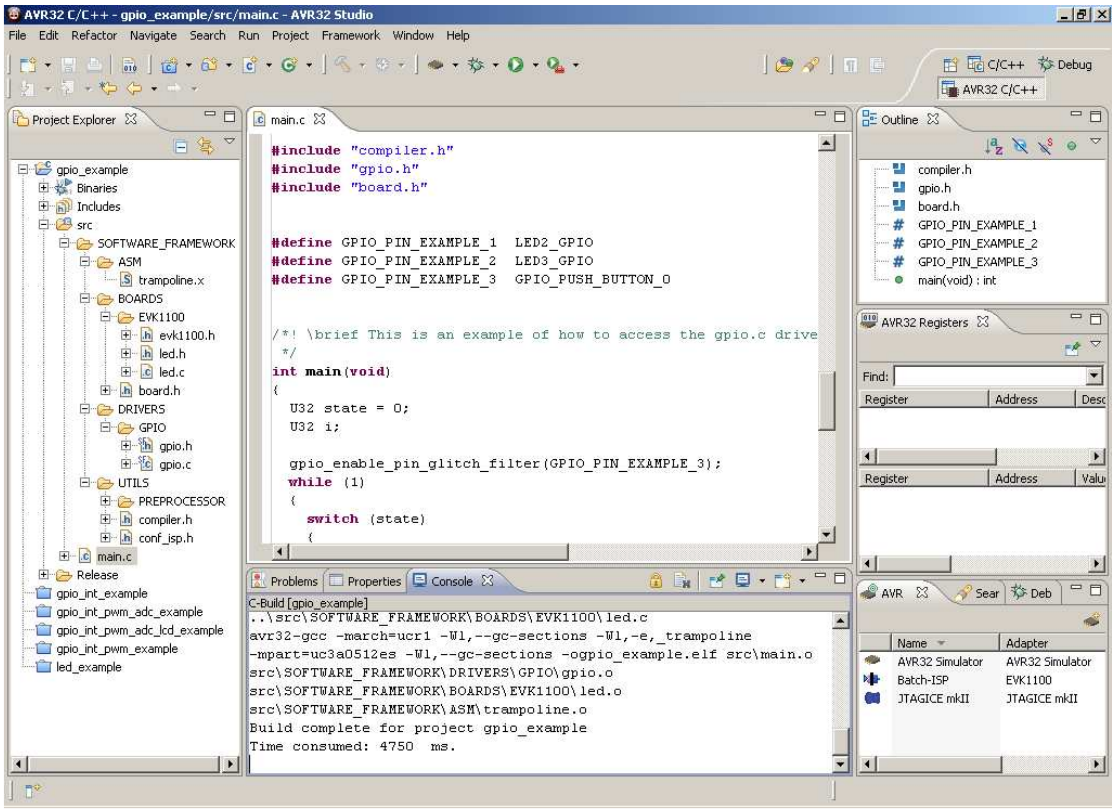
Εικόνα 10 : Μήνυμα ελέγχου σωστής εγκατάστασης

Από τη στιγμή που θα γίνουν οι εγκαταστάσεις το λογισμικό που εμφανίζεται στα Windows με την επιλογή `Control Panel → Add or Remove Programs` θα αποτελείται από τα: AVR32 Studio 2.0, AVR32 Toolchain, AVR USB (ανήκει στο Toolchain) και Flip 3.3.1. Έτσι, αν απαιτηθεί εγκατάσταση από την αρχή, καλό είναι να απεγκατασταθούν αυτά τα προγράμματα και να σβηστούν και οι κατάλογοι στους οποίους είχαν εγκατασταθεί.

II.3 Ροή ανάπτυξης (από τη γραφή κώδικα μέχρι τον προγραμματισμό).

Το Software Framework διαθέτει αρκετά παραδείγματα και η αναφορά [5] μαζί με το on-line help περιγράφουν πως γίνεται η διαδικασία της γραφής μίας εφαρμογής μέχρι τον προγραμματισμό της συσκευής. Σε αυτό το τμήμα θα αναπτυχθεί μία πολύ απλή εφαρμογή, χωρίς να εξηγηθεί καθόλου ο κώδικας, με σκοπό να δειχθεί πως ξεκινάει κανείς από το μηδέν μέχρι να προγραμματίσει τον μικροελεγκτή από το USB καλώδιο. Το παράδειγμα που δίνεται απλώς ανάβει μόνο το LED3 της πλακέτας και ανάβει ή σβήνει το LED4 ανάλογα με τη θέση του διακόπτη πίεσης PB0. Ο κώδικας και οι οδηγοί που υπάρχουν για τα διάφορα περιφερειακά θα εξηγηθούν στο επόμενο κεφάλαιο.
Το πρόγραμμα σε γλώσσα C αποτελείται από ένα μόνο αρχείο που έχει όνομα `gpio_example.c` και παρουσιάζεται στο Παράρτημα IV.1. Μπορεί να βρεθεί κάτω από τη διαδρομή `DRIVERS/GPIO/EXAMPLES` του αποσυμπιεσμένου αρχείου `AVR32-SoftwareFramework-1.2.1ES-AT32UC3A.zip` που περιέχει τις βιβλιοθήκες που υποστηρίζουν την πλακέτα EVK1100. Η διαδικασία που περιγράφεται παρακάτω μπορεί να ξεκινήσει και χρησιμοποιώντας έτοιμο το project που υπάρχει ενσωματωμένο στο περιβάλλον του AVR 32 Studio, επιλέγοντας File → New → Example, αλλά δεν είναι σκόπιμο γιατί έτσι θα παραλειφθούν κάποια βήματα.
Όλη η διαδικασία θα γίνει στο βασικό εργαλείο AVR 32 Studio το οποίο φαίνεται σε πλήρη ανάπτυξη στην Εικόνα 11.
Αρχικά πρέπει να δημιουργηθεί ένα νέο έργο (Project) που θα περιλαμβάνει όλα τα απαραίτητα αρχεία για τη μεταγλώττιση. Αυτό γίνεται με την επιλογή File → New → AVR32 C Project from Template. Η κάρτα που ανοίγει (Εικόνα 12) ζητάει το όνομα του έργου (Project Name) και τη θέση που θα αποθηκευτεί. Είναι προεπιλεγμένος με μαρκάρισμα (Use default location) ο κατάλογος που είχε δημιουργηθεί κατά την αρχική εγκατάσταση του AVR 32 Studio σαν κατάλογος βάσης (workspace) όλων των projects. Αν ο χρήστης επιθυμεί διαφορετικό κατάλογο πρέπει να ξεμαρκάρει το Default location και να επιλέξει με το Browse ένα δικό του. Σε κάθε περίπτωση θα δημιουργηθεί ένας κατάλογος με όνομα το όνομα του έργου που δόθηκε και κάτω από αυτόν θα μπαίνουν όλα τα σχετικά αρχεία. Επιπλέον πρέπει να οριστεί για ποιον μικροελεγκτή προορίζεται αυτό το έργο (Target MCU) καθώς και ο τύπος του (Project Type).

Στην συγκεκριμένη περίπτωση είναι ο μικροελεγκτής που είναι πάνω στην EVK1100, δηλαδή ο UC3A0512ES. Ο τύπος του είναι αυτόνομο εκτελέσιμο αρχείο (AVR32 Standalone Executable) το οποίο προορίζεται για την πλακέτα EVK1100. Αυτή η επιλογή έχει σαν συνέπεια την αυτόματη ενσωμάτωση στην ιεραρχία του project των αρχείων υποστήριξης της συγκεκριμένης πλακέτας (π.χ. `board.h`) και την αποφυγή της χειροκίνητης προσθήκης τους αργότερα. Κάτι τέτοιο θα ήταν απαραίτητο στην περίπτωση που το project ξεκινούσε με την απλή επιλογή File → New → AVR32 C Project.



Εικόνα 11 : Το AVR 32 Studio

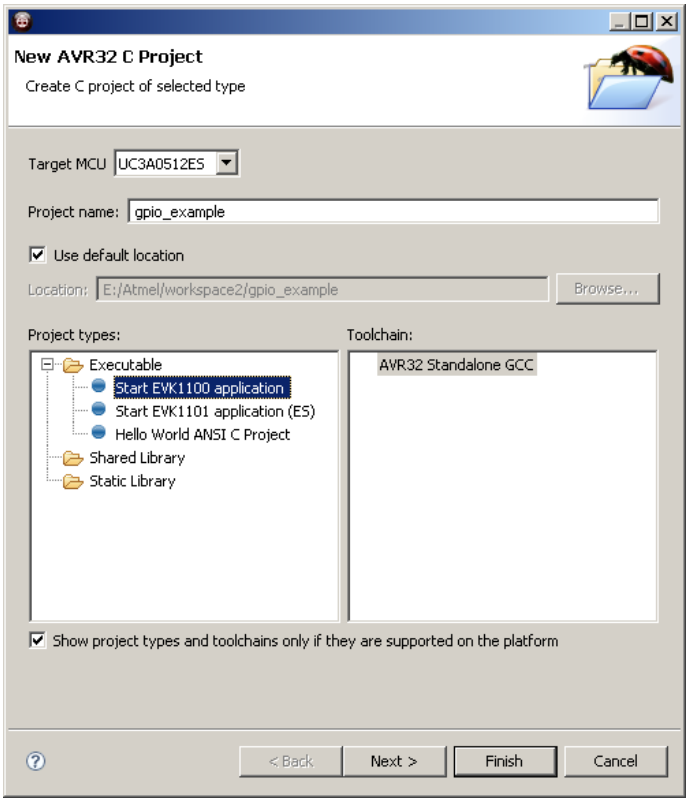
Αφού έχει αρχικοποιηθεί το project, μπορεί να αρχίσει η γραφή του κώδικα ο οποίος μπορεί να αποτελείται από πολλά αρχεία C (source ή header) και βιβλιοθήκες. Σε αυτό το παράδειγμα θα χρησιμοποιηθεί μόνο ένα αρχείο C που ήδη υπάρχει στην ιεραρχία με όνομα `main.c`. Με διπλό αριστερό κλικ στο όνομα του αρχείου στο παράθυρο Project Explorer ανοίγει το αρχείο στο κεντρικό παράθυρο του AVR32 Studio. Στην συγκεκριμένη περίπτωση θα αντιγραφεί έτοιμος (με copy-paste από κάποιον editor) ο κώδικας από το αρχείο του παραδείγματος που προαναφέρθηκε (`gpio_example.c`).

Δημιουργία – Γραφή Κώδικα

Η δημιουργία νέου αρχείου C γίνεται με την επιλογή File → New → Source File από το μενού ή με δεξί κλικ πάνω στο project που επιλέγεται στο παράθυρο Project Explorer. Στην κάρτα που αναδύεται θα πρέπει να εισαχθεί το όνομα του αρχείου C με κατάληξη `.c` (Source file) και σε ποιο project ανήκει αν δεν είναι προεπιλεγμένο το σωστό (Source Folder). Με αυτόν τον τρόπο δημιουργούνται και όλα τα υπόλοιπα επιθυμητά αρχεία .c και .h (File → New → Header File) για το εκάστοτε project.

Μία γρήγορη επισκόπηση του κώδικα C που μόλις ενσωματώθηκε στο έργο, θα δείξει ότι γίνονται αναφορές σε αρχεία επικεφαλίδας (header files) και συναρτήσεις που δεν έχουν οριστεί. Αυτά είναι κομμάτια των βιβλιοθηκών που υπάρχουν στο Software Framework της Atmel και ορίζουν οδηγούς για τα εσωτερικά περιφερειακά (στη συγκεκριμένη περίπτωση τον

GPIO controller) αλλά και περιέχουν ορισμούς και για τα εξωτερικά (π.χ. σε ποιον ακροδέκτη είναι συνδεδεμένο το LED3). Με αυτόν τον τρόπο διευκολύνεται κατά πολύ ο προγραμματισμός και δεν είναι αναγκαία η γνώση κάθε λεπτομέρειας σχετικά με το υλικό.



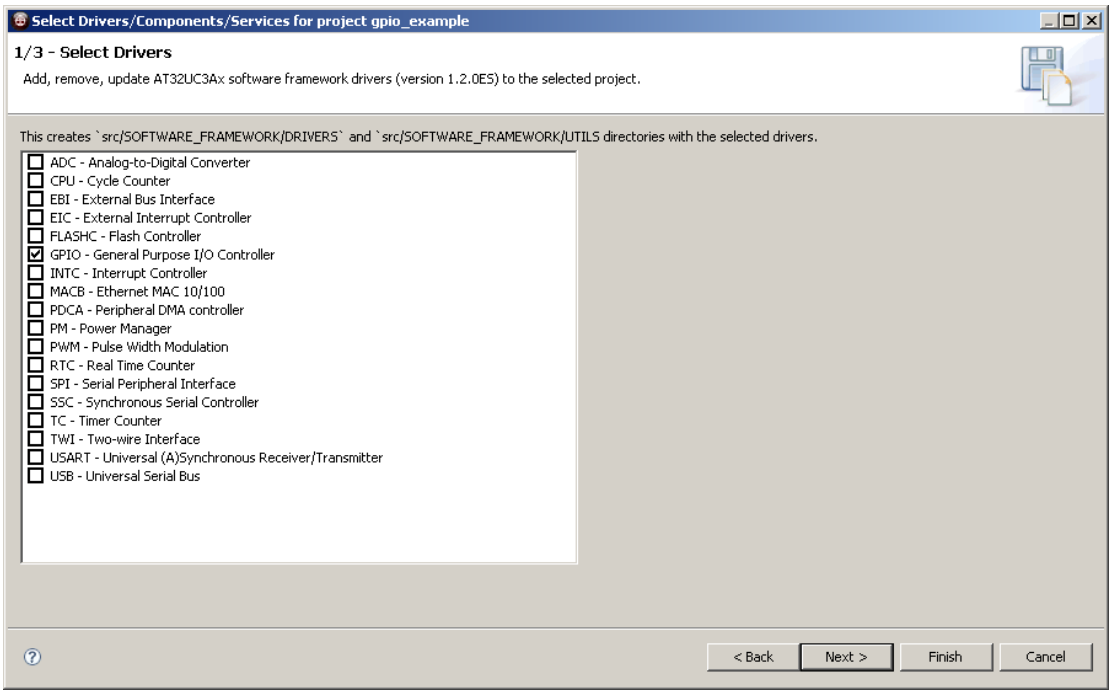
Εικόνα 12 : Δημιουργία νέου έργου

Η εισαγωγή των σχετικών βιβλιοθηκών στο project γίνεται με την επιλογή Select → Components/Drivers/Services. Στη δεύτερη κάρτα επιλέγουμε τον οδηγό για τον GPIO controller και τελειώνουμε με το κουμπί Finish (Εικόνα 13).
Τώρα στο project έχουν ενσωματωθεί τα απαραίτητα αρχεία `compiler.h`, `gpio.h` και `gpio.c` όπως φαίνεται και στο παράθυρο Project Explorer κάτω από το φάκελο src.

Μεταγλώττιση

Τώρα το project είναι έτοιμο για μεταγλώττιση. Πρώτα θα πρέπει να επιλεγεί το είδος της διαρρύθμισης (configuration) με βάση το οποίο θα παραχθεί το εκτελέσιμο αρχείο για φόρτωση. Υπάρχουν δύο είδη, ένα για εκσφαλμάτωση (Debug) και ένα για τελική χρήση (Release). Εδώ θα επιλεγεί η διαρρύθμιση για τελική χρήση. Αυτή η διαδικασία γίνεται έχοντας επιλέξει από το παράθυρο Project Explorer το project που ενδιαφέρει (στο Project Explorer εμφανίζονται όλα τα projects που είναι αποθηκευμένα στον κατάλογο βάσης) και κατόπιν επιλέγοντας από το μενού (ή και με δεξί κλικ πάνω στο project) την επιλογή Project → Build Configurations → Set Active → Release. Το επόμενο βήμα είναι η επιλογή Project → Build All ή Project → Build Project (γίνεται και με δεξί κλικ πάνω στο επιλεγμένο project). Η πορεία και τα μηνύματα σχετικά με τη μεταγλώττιση φαίνονται στο κάτω παράθυρο (στήλη Console). Η στήλη Problems θα εμφανίσει πιθανές προειδοποιήσεις και σφάλματα. Αν υπάρχουν προβλήματα εμφανίζονται διάφορα σημάδια με υπερδεσμούς στις σχετικές γραμμές των αρχείων που παρατηρήθηκαν προκειμένου να διορθωθούν. Αν δεν υπήρξε πρόβλημα τότε, το δυαδικό αρχείο που πρόκειται να φορτωθεί πάνω στον μικροελεγκτή είναι έτοιμο και είναι αποθηκευμένο μέσα στον υποκατάλογο `Release` της ιεραρχίας του project με κατάληξη `.elf` (στην συγκεκριμένη περίπτωση είναι το αρχείο `gpio_example.elf`). Φυσικά, το

αρχείο αυτό, όπως και τα υπόλοιπα της ιεραρχίας, είναι ορατό και εκτός του AVR 32 Studio, αν είναι γνωστός ο κατάλογος βάσης αποθήκευσης των projects.



Εικόνα 13 : Εισαγωγή οδηγών/βιβλιοθηκών

Φόρτωμα προγράμματος στον μικροελεγκτή

Το τελευταίο βήμα είναι ο προγραμματισμός του μικροελεγκτή με το δυαδικό αρχείο (`.elf`) που έχει δημιουργήσει ο μεταγλωττιστής. Θα περιγραφούν δύο μέθοδοι προγραμματισμού. Η μία είναι χωρίς τη χρήση ειδικής συσκευής προγραμματισμού, με τη χρήση διασύνδεσης USB μεταξύ της πλακέτας EVK1100 και του υπολογιστή (ISP – In System Programming). Η άλλη μέθοδος είναι με τη χρήση ειδικής συσκευής όπως ο JTAGICE mkII με τη χρήση της διασύνδεσης JTAG.

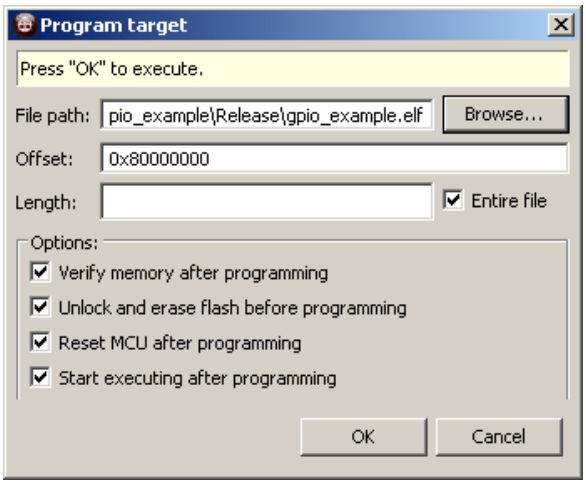
Ο χειρισμός και των δύο τρόπων γίνεται από το κάτω δεξιά παράθυρο του AVR32 Studio (στήλη AVR32 Targets). Αρχικά θα περιγραφεί το ISP με χρήση της USB διασύνδεσης (BatchISP) και ύστερα ο προγραμματισμός με τη συσκευή JTAGICE mkII. Λεπτομέρειες για τον τρόπο λειτουργίας του προγραμματισμού με τη χρήση της θύρας USB υπάρχουν στην αναφορά [7].

BatchISP

Η κάθε συσκευή ATUC3A έχει αποθηκευμένο ένα πρόγραμμα εκκίνησης στη μνήμη Flash που λέγεται Bootloader. Ο Bootloader ελέγχει κάποιες συνθήκες και αποφασίζει αν θα εκτελεστεί το κυρίως πρόγραμμα ή θα αναμένει προγραμματισμό της μνήμης Flash μέσω της θύρας USB. Στην πλακέτα EVK1100 η συνθήκη για να ενεργοποιηθεί η διαδικασία ISP στον μικροελεγκτή είναι να γίνει επανεκκίνηση (κουμπί RES) ενώ ο μοχλοδιακόπτης (Joy) είναι πιεσμένος κάτω. Μόλις γίνει επανεκκίνηση (αφεθεί το κουμπί RES) και αφεθεί και ο μοχλοδιακόπτης τότε αρχίζει η διαδικασία προγραμματισμού. Από την άλλη μεριά ο υπολογιστής πρέπει να διαθέτει ένα αντίστοιχο πρόγραμμα συνεργασίας με τον bootloader το οποίο λέγεται BatchISP. Το BatchISP θα πρέπει να έχει εγκατασταθεί και αυτό γίνεται αφού έχει εγκατασταθεί το πρόγραμμα Flip. Η διαδικασία εγκατάστασης του BatchISP έχει ως εξής (βλ. και Ενότητα 7.3 της αναφοράς [7]):

Υποθέτουμε ότι έχει γίνει η σύνδεση USB της πλακέτας και του υπολογιστή. Τότε γίνεται ενεργοποίηση του ISP στον μικροελεγκτή (RES-Joy) και εμφανίζεται στον υπολογιστή η

γνωστή κάρτα των Windows για την εγκατάσταση νέου οδηγού υλικού (Found New Hardware Wizard). Θα πρέπει να αναζητηθεί ο σχετικός οδηγός στον κατάλογο `usb` που υπάρχει στην ιεραρχία των αρχείων του προγράμματος Flip (π.χ. `Program Files\Flip 3.3.1\usb`). Αφού γίνει η εγκατάσταση του προγράμματος-οδηγού τότε, μπορεί να γίνει η χρήση του είτε αυτόνομα είτε μέσα από το περιβάλλον του AVR 32 Studio.
Στο κάτω δεξί παράθυρο του περιβάλλοντος AVR 32 Studio και στη στήλη AVR 32 Targets επιλέγεται με δεξί κλικ το Scan Targets. Αν είναι συνδεδεμένη η πλακέτα EVK1100 με τον υπολογιστή τότε θα βρεθεί και εμφανιστεί σε μία γραμμή στο ίδιο παράθυρο σαν AVR 32 Simulator. Έχοντας επιλεγμένη αυτή τη γραμμή θα πρέπει να γίνουν κάποιες ρυθμίσεις στο κάτω παράθυρο – στήλη Properties. Αυτές είναι οι εξής:
Target → Name ένα διαφορετικό όνομα από AVR 32 Simulator, π.χ. BatchISP γιατί είναι δυνατόν να υπάρχουν διαφορετικοί τρόποι προγραμματισμού (αν χρησιμοποιείται και το JTAGICE για παράδειγμα).
Adapter → Adapter να επιλεγεί η πλακέτα EVK1100.
Board → Board να επιλεγεί η πλακέτα EVK1100.
Board → MCU να επιλεγεί ο μικροελεγκτής UC3A0512ES.
Οι υπόλοιπες επιλογές στη στήλη Properties δεν είναι αναγκαίο να αλλαχθούν (βλ. και Ενότητα 7.4 της αναφοράς [7].
Αφού έχει γίνει και η εγκατάσταση του BatchISP στο AVR 32 Studio όπως περιγράφηκε προηγουμένως, μπορεί να γίνει ο προγραμματισμός του μικροελεγκτή που βρίσκεται πάνω στην πλακέτα. Η διαδικασία ξεκινάει και τελειώνει ενεργοποιώντας την κάρτα Program Target που φαίνεται στην Εικόνα 14 κάνοντας δεξί κλικ στον προγραμματιστή που έχει εγκατασταθεί από το προηγούμενο βήμα στο κάτω δεξιά παράθυρο (AVR 32 Targets). Αυτό που χρειάζεται η κάρτα σαν πληροφορία είναι το αρχείο `.elf` που θα φορτωθεί στον μικροελεγκτή (`gpioEx.elf` στο παράδειγμά μας). Άλλα στοιχεία είναι η διεύθυνση στην οποία θα ξεκινήσει να γράφεται το πρόγραμμα και η οποία είναι η 0x80000000 που είναι η πρώτη διεύθυνση της μνήμης προγράμματος (Εικόνα 4). Οι υπόλοιπες επιλογές που φαίνονται στην Εικόνα 14 είναι για επιβεβαίωση ότι το πρόγραμμα φορτώθηκε σωστά, δηλαδή σύγκριση περιεχομένων μνήμης Flash μικροελεγκτή με αρχείο `.elf` (Verify memory after programming), ξεκλείδωμα και διαγραφή της μνήμης flash πριν τον προγραμματισμό (Unlock and erase flash before programming), και ο μηδενισμός και επανεκκίνηση του μικροελεγκτή μετά τον προγραμματισμό (Reset MCU after programming και Start executing after programming).



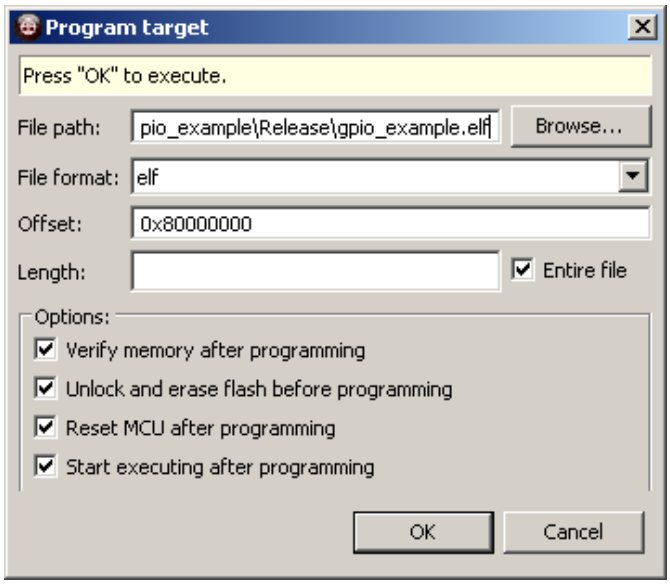
Εικόνα 14 : Προγραμματισμός του μικροελεγκτή από USB

Η πρόοδος και τυχόν προβλήματα που εμφανίζονται κατά τη διάρκεια της φόρτωσης του προγράμματος φαίνονται στο κάτω παράθυρο και στη στήλη Console. Με τις επιλογές που φαίνονται στην Εικόνα 14 και αν όλα πήγαν καλά το πρόγραμμα θα φορτωθεί στον μικροελεγκτή και θα αρχίσει να εκτελείται. Το αποτέλεσμα για το παράδειγμα που περιγράφεται θα είναι να αναβοσβήνει μόνιμα το πράσινο LED3 και να ανάβει ή να σβήνει το πράσινο LED4 ανάλογα με το αν είναι πιεσμένος ο διακόπτης PB0.

JTAGICE mkII

Ο προγραμματιστής JTAGICE mkII μπορεί να συνδεθεί στον υπολογιστή είτε με USB θύρα, είτε με RS-323 θύρα. Παρακάτω θα περιγραφεί η περίπτωση της USB θύρας. Η περίπτωση χρήσης RS-232 είναι ανάλογη. Πριν να μπορέσει να γίνει συνεργασία του JTAGICE mkII με το AVR Studio 32, πρέπει να εγκατασταθεί ο οδηγός της συσκευής στα Windows. Ο οδηγός συμπεριλαμβάνεται στο λογισμικό AVR 32 Gnu Toolchain, έτσι όταν γίνεται πρώτη φορά η σύνδεση με το καλώδιο στον υπολογιστή, ο οδηγός εγκατάστασης νέου λογισμικού (Found New Hardware Wizard) θα πρέπει να καθοδηγηθεί να κάνει την εγκατάσταση αυτόματα (χωρίς αναζήτηση δηλαδή σε κάποιο συγκεκριμένο μέρος του συστήματος αρχείων).
Τώρα, και αφού έχει γίνει η σύνδεση της καλωδιοταινίας του JTAGICE mkII με την EVK1100 όπως φαίνεται στην Εικόνα 9 και τροφοδοτούνται και οι δύο συσκευές μπορεί να γίνει η εισαγωγή του προγραμματιστή στο περιβάλλον AVR 32 Studio.
Κάνοντας την επιλογή Scan Targets στο κάτω δεξί παράθυρο θα εμφανιστεί μία νέα γραμμή με όνομα AVR32 Simulator που θα αναπαριστά τον προγραμματιστή και θα πρέπει να ρυθμιστούν κάποιες ιδιότητες κατ΄ αναλογία με την περίπτωση του BatchISP. Οι ιδιότητες αυτές (στο κάτω παράθυρο – στήλη Properties και έχοντας επιλεγμένο τον προγραμματιστή) είναι:
Target → Name ένα διαφορετικό όνομα από AVR 32 Simulator, π.χ. JTAG.
Adapter → Adapter να επιλεγεί JTAGICE mkII.
Adapter → Serial Number: Πρόκειται για τον σειριακό αριθμό της συσκευής προγραμματισμού ο οποίος μπορεί να βρεθεί στην ετικέτα κάτω από τη συσκευή. Ο αριθμός που πρέπει να εισαχθεί είναι 12 δεκαεξαδικών ψηφίων. Η ετικέτα όμως δείχνει μόνο τα 10 τελευταία, έτσι θα πρέπει μπροστά να προστεθούν 2 μηδενικά. Δηλαδή αν η ετικέτα δείχνει B000003EBE θα πρέπει να εισαχθεί ο αριθμός 00B000003EBE. Σε κάθε περίπτωση ο πλήρης αριθμός μπορεί να φανεί με τη βοήθεια του Device Manager των Windows.
Adapter → Connection να επιλεγεί usb
Board → Board να επιλεγεί η πλακέτα EVK1100.
Board → MCU να επιλεγεί ο μικροελεγκτής UC3A0512ES.
Πριν χρησιμοποιηθεί ο προγραμματιστής θα πρέπει να πιστοποιηθεί ότι διαθέτει το τελευταίο firmware ή έστω firmware που είναι συμβατό με την έκδοση του AVR 32 Studio που χρησιμοποιείται. Στην περίπτωσή μας το firmware πρέπει να είναι έκδοσης 5.23. Η αναβάθμιση του firmware γίνεται κάνοντας Update Firmware με δεξί κλικ στο κάτω δεξιά παράθυρο και είναι απαραίτητη η σύνδεση με το διαδίκτυο. Μηνύματα στο κάτω παράθυρο (Console) ενημερώνουν για την πορεία της αναβάθμισης.
Τώρα ο προγραμματιστής JTAGICE mkII είναι έτοιμος να χρησιμοποιηθεί μέσα από το AVR 32 Studio. Η χρήση του είναι ανάλογη με αυτήν της χρήσης του BatchISP που περιγράφηκε σε προηγούμενες παραγράφους. Ενδεικτικά παρουσιάζονται οι επιλογές που πρέπει να έχει η κάρτα που αναδύεται με την επιλογή Program… στην Εικόνα 15.
Υπάρχει η περίπτωση να χρειαστεί να γίνει διαγραφή ολόκληρου του μικροελεγκτή (Chip Erase) για να πετύχει ο προγραμματισμός. Σε αυτήν την περίπτωση διαγράφεται και ο bootloader. Επομένως, η συσκευή θα μπορεί να επαναπρογραμματιστεί μόνο με τον JTAGICE mkII και όχι με τον BatchISP.
Η επαναφόρτιση του bootloader μπορεί να γίνει μόνο με τη συσκευή JTAGICE mkII. Η αναφορά [7] στην Ενότητα 7.1 περιγράφει τη διαδικασία η οποία όμως ισχύει για την περίπτωση του Linux λειτουργικού συστήματος. Για την περίπτωση χρήσης Windows υπάρχει έτοιμο πακέτο με οδηγίες που δίνεται και στο CD σαν συμπιεσμένο αρχείο με όνομα

AVR32Studio20-BachISP-V10.zip. Το αρχείο αυτό μπορεί να βρεθεί στην ιστοθέση υποστήριξης της Atmel (http://support.atmel.no/bin/customer) στην ιεραρχία των συχνών ερωτήσεων (FAQ → AVR32 32-bit MCU → UC3 devices→ UC3 - USB Bootloader Reprogramming).



Εικόνα 15 : Προγραμματικός μικροελεγτή με το JTAGICE mkII

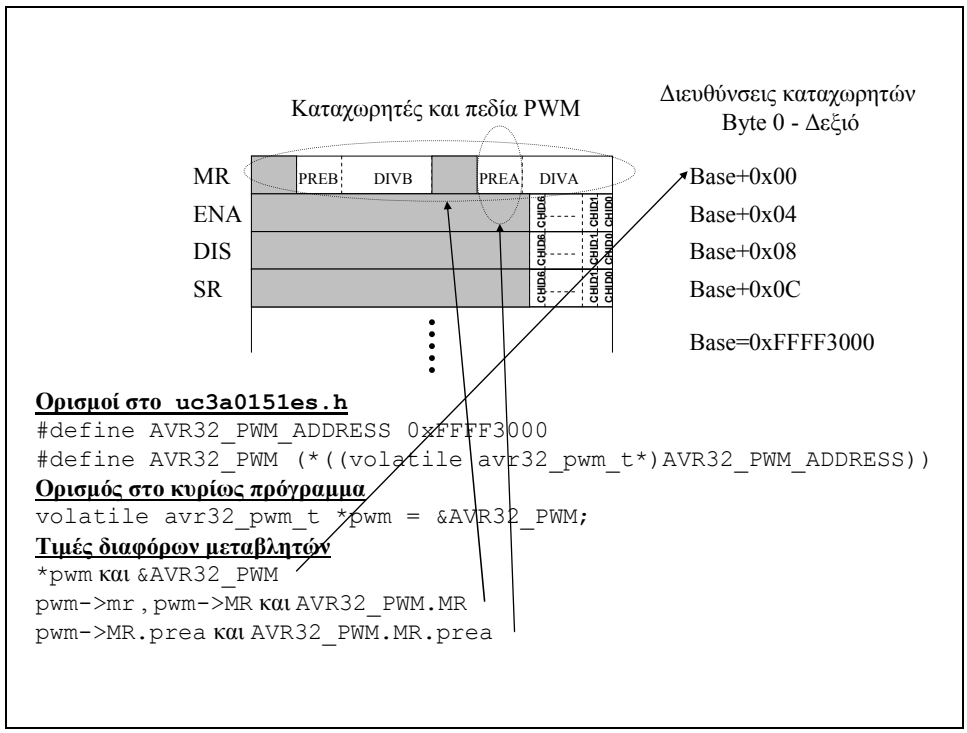
III Ανάπτυξη εφαρμογών.

Σε αυτό το κεφάλαιο θα περιγραφούν πιο αναλυτικά ορισμένα από τα εσωτερικά περιφερειακά κυκλώματα του AVR32 και θα αναπτυχθούν απλές εφαρμογές για κάθε ένα από αυτά σε γλώσσα C και με τη χρήση των έτοιμων οδηγών και συναρτήσεων που προσφέρει η Atmel μέσω του Software Framework. Δίνεται έμφαση στα περιφερειακά κυκλώματα γιατί αυτά είναι ουσιαστικά τα στοιχεία που διαφοροποιούν τον προγραμματισμό σε C του μικροελεγκτή σε σχέση με τον προγραμματισμό ενός οποιουδήποτε άλλου μικροεπεξεργαστή. Άλλωστε οι γλώσσες ανώτερου επιπέδου όπως η C αναπτύχθηκαν μεταξύ άλλων και για να κρύβουν τις εσωτερικές διαφορές των διαφόρων επεξεργαστών. Έτσι, αυτό που μένει είναι τα επί πλέον διαφορετικά στοιχεία που έχει ο μικροελεγκτής σε σχέση με οποιονδήποτε άλλο μικροεπεξεργαστή και αυτά τα στοιχεία είναι τα περιφερειακά κυκλώματά του και όχι τόσο η αρχιτεκτονική του πυρήνα του. Με αυτή την ανάπτυξη θα φανεί και το μεγάλο πλεονέκτημα της χρήσης γλώσσας C αντί της συμβολικής γλώσσας για τον προγραμματισμό, αφού πολύπλοκες διαδικασίες προγραμματισμού ενός κυκλώματος αντικαθίστανται πολλές φορές από μία συνάρτηση. Μία σύντομη εισαγωγή σε θέματα ανάπτυξης μίας εφαρμογής και επιλογών του μεταγλωττιστή δίνεται στις αναφορές [8] και [9]. Επίσης, πληροφορίες για τους οδηγούς του κάθε περιφερειακού υπάρχουν και στην επιλογή βοήθειας (on-line help) του περιβάλλοντος AVR 32 Studio (βιβλίο AT32UC3AES Software Framework). Τα ίδια έγγραφα της βοήθειας υπάρχουν σε html μορφή μέσα στο συμπιεσμένο αρχείο `AVR32-SoftwareFramework-1.3.0-AT32UC3A.zip` εκκινώντας την περιήγηση από το αρχείο `AVR32_Readme.html`. Σε κάθε παράδειγμα που ακολουθεί, εκτός από τις αναφορές που υποδεικνύονται, θα πρέπει να γίνεται και η αναζήτηση των σχετικών εγγράφων στις επιλογές βοήθειας που διαθέτει το περιβάλλον AVR 32 Studio.

III.1 Προσπέλαση σε καταχωρητές περιφερειακών.

Η χρήση των εσωτερικών περιφερειακών του μικροελεγκτή γίνεται με την προσπέλαση των ειδικών καταχωρητών τους. Η προσπέλαση γίνεται με ανάγνωση/εγγραφή σε συγκεκριμένη διεύθυνση μνήμης που έχει ο κάθε καταχωρητής (memory-mapped I/O). Για την ευκολία προγραμματισμού με τη χρήση της γλώσσας C έχουν οριστεί σε ειδικά αρχεία επικεφαλίδων (αρχεία με κατάληξη `.h`) τύποι μεταβλητών της γλώσσας C που είναι δομές (struct) και έχουν ορισμένες στα πεδία και υποπεδία τους τους καταχωρητές και τα πεδία των καταχωρητών τους. Στα ίδια αρχεία ορίζονται και σταθερές που δείχνουν τη διεύθυνση βάσης του κάθε περιφερειακού, τη διεύθυνση μετατόπισης των καταχωρητών και των πεδίων του και άλλες πληροφορίες.
Σαν παράδειγμα θα αναφέρουμε την περίπτωση του περιφερειακού PWM (Pulse Width Modulation). Η χρησιμοποίησή του προϋποθέτει τη χρήση της επικεφαλίδας `pwm.h`. Εκεί γίνεται αναφορά στην επικεφαλίδα `<avr32/io.h>` όπου ανάλογα με τον μικροελεγκτή προορισμού που έχει επιλεγεί σαν στόχος προγραμματισμού (Target MCU) γίνεται επίκληση μίας άλλης επικεφαλίδας. Στην περίπτωσή μας πρόκειται για το αρχείο `uc3a0512.h` . Σε αυτό το αρχείο ορίζεται (με την οδηγία `#define`) στο σχετικό τμήμα για το PWM το σύμβολο `AVR32_PWM_ADDRESS` που αντιστοιχεί στη διεύθυνση βάσης του PWM, δηλαδή την 0xFFFF3000. Επίσης γίνεται αναφορά και στο αρχείο `pwm130.h` στο οποίο είναι ορισμένη μία ολόκληρη δομή που περιέχει όλους τους καταχωρητές του PWM. Αυτή η δομή είναι τύπου `avr32_pwm_t` και τα πεδία της που είναι των 32 bits έχουν τα ονόματα των καταχωρητών. Π.χ. το πρώτο πεδίο της δομής με όνομα `mr` ή εναλλακτικά `MR` είναι τα πρώτα 32 bits (offset 0 σε σχέση με τη διεύθυνση βάσης) που αναπαριστούν τον πρώτο καταχωρητή του PWM. Η προσπέλαση μπορεί να γίνει με χρήση του mr πεδίου ταυτόχρονα σε όλα τα bits είτε με χρήση

του MR πεδίου για προσπέλαση σε υποπεδία του καταχωρητή (έχει προηγηθεί ο ορισμός της δομής `avr32_pwm_mr_t` που ορίζει τα υποπεδία που έχει ο καταχωρητής MR).
Ο ορισμός μίας μεταβλητής στο κυρίως πρόγραμμα, έστω της `pwm`, με τον τρόπο `volatile avr32_pwm_t *pwm = &AVR32_PWM;` θα έχει ως αποτέλεσμα η μεταβλητή pwm να δείχνει σε μία δομή τύπου `avr32_pwm_t` με αρχική διεύθυνση 0xFFFF3000 που είναι η βάση του PWM. Δηλαδή η μεταβλητή `pwm` είναι δείκτης (pointer) τύπου `avr32_pwm_t`. Αρχικοποιείται στην τιμή `&AVR32_PWM` όπου η έκφραση AVR32_PWM έχει αρχικοποιηθεί στο αρχείο `pwm130.h` και ουσιαστικά είναι η διεύθυνση βάσης του PWM που έχει γίνει casting («χύτευση» μεταβλητής, μετατροπή τύπου) στον παραπάνω τύπο.
Τώρα πια η προσπέλαση σε κάθε καταχωρητή εξ΄ολοκλήρου μπορεί να γίνει με τη χρήση των συμβόλων `->` και `.` της C. Για παράδειγμα η προσπέλαση στον καταχωρητή MR γίνεται με τη χρήση της παράστασης `pwm->mr`. Η προσπέλαση στον καταχωρητή CUPD του καναλιού 4 γίνεται με τη χρήση του `pwm->channel[4].cupd`.
Εναλλακτικά μπορεί να χρησιμοποιηθεί ο συμβολισμός `AVR32_PWM.mr` απ' ευθείας, εφ' όσον στην επικεφαλίδα έχει οριστεί η `AVR32_PWM`.
Για πληρέστερη κατανόηση η Εικόνα 16 δείχνει μερικούς καταχωρητές και πεδία του PWM και τις τιμές που παίρνουν κάποιες παραστάσεις της C.



Εικόνα 16 : Παράδειγμα αναφορών σε πεδία καταχωρητών περιφερειακών

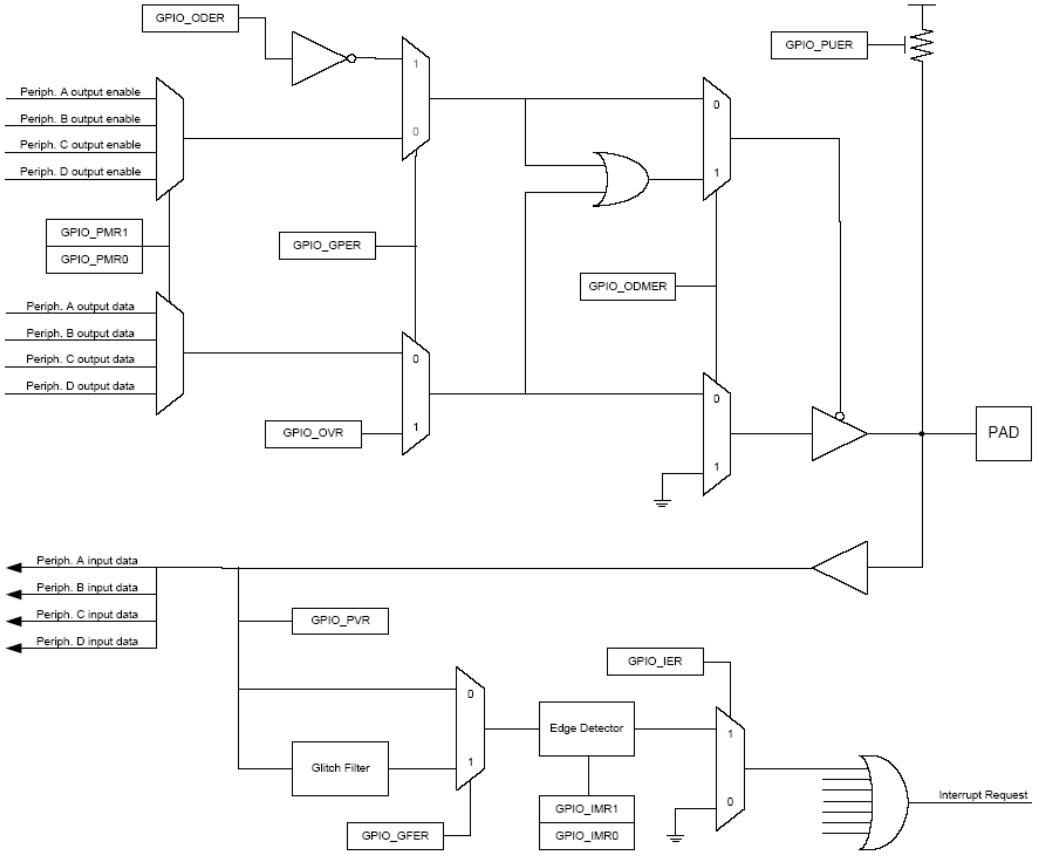
Στην αναφορά [10] δίνονται περισσότερες λεπτομέρειες για τις επικεφαλίδες και περιγράφονται και άλλοι τρόποι για προσπέλαση σε μεμονωμένα πεδία καταχωρητών μέσω της χρήσης macros και άλλων ορισμών μέσα στις επικεφαλίδες.
Εκτός από τους ορισμούς μέσω της οδηγίας `#define` και τους ορισμούς των δομών, υπάρχουν και ορισμοί συναρτήσεων που βοηθούν στην χρήση του κάθε περιφερειακού. Ο ορισμός της συνάρτησης γίνεται στο αρχείο επικεφαλίδας του περιφερειακού (`.h`) σε συνδυασμό με ένα αρχείο ίδιου ονόματος που περιέχει κώδικα C (`.c`) όπου υπάρχει αναλυτικά ο κώδικας της συνάρτησης. Για το παράδειγμα του PWM το αρχείο που περιέχει τις συναρτήσεις είναι το `pwm.c`. Οι πιο πολλές συναρτήσεις περιέχουν διαδοχικά διάφορες προσπελάσεις σε καταχωρητές και πεδία καταχωρητών. Π.χ. η συνάρτηση αρχικοποίησης καναλιού

`pwm_channel_init` επηρεάζει τους καταχωρητές CMR, CPRD, CDTY του καναλιού με αριθμό που περνάει σαν παράμετρος στη συνάρτησης. Πάντως οι συναρτήσεις αυτές στις πιο πολλές περιπτώσεις απαιτούν τη χρήση μεταβλητών που είναι δείκτες στις δομές που ορίζονται στις επικεφαλίδες, επομένως δεν αποφεύγεται ούτε με αυτό τον τρόπο η χρήση δεικτών και δομών. Για παράδειγμα ο ορισμός της `pwm_channel_init` είναι `int pwm_channel_init(unsigned int channel_id, const avr32_pwm_channel_t *pwm_channel)`.
Η επιλογή του τρόπου προσπέλασης των καταχωρητών αφήνεται στον προγραμματιστή ανάλογα με την περίπτωση. Όλα τα αρχεία των επικεφαλίδων και των συναρτήσεων είναι προσπελάσιμα και από τον Project Explorer του AVR32 Studio και επίσης εκτός περιβάλλοντος είναι προσπελάσιμα στην ιεραρχία εγκατάστασης του AVR32 Toolchain.

III.2 Ελεγκτής εισόδου – εξόδου (GPIO).

Ο ελεγκτής εισόδου – εξόδου συνδέει τους ακροδέκτες εισόδου – εξόδου (PA..,PB..,PC.. και PX..) του μικροελεγκτή με σήματα των εσωτερικών περιφερειακών ή δίνει στον απ' ευθείας στον επεξεργαστή τον έλεγχο των ακροδεκτών. Για κάθε έναν ακροδέκτη υπάρχει αφιερωμένο ένα πανομοιότυπο βασικό κύκλωμα που επιτρέπει τη διασύνδεση του με ένα από τρία διαφορετικά σήματα περιφερειακών. Η ακριβής λειτουργία του κάθε στοιχειώδους κυκλώματος καθορίζεται από τους καταχωρητές ελέγχου του GPIO (του 1 bit) που φαίνονται στην Εικόνα 17 μαζί με το πλήρες κύκλωμα. Το κύκλωμα αποτελείται κυρίως από πολυπλέκτες που ελέγχονται από τα σήματα των προγραμματιζόμενων καταχωρητών. Ο ακροδέκτης του ολοκληρωμένου (PAD) μπορεί να λειτουργήσει είτε σαν είσοδος είτε σαν έξοδος. Ακολουθεί μία σύντομη περιγραφή της λειτουργίας των καταχωρητών ελέγχου του περιφερειακού GPIO. Απαραίτητες πληροφορίες για τον προγραμματισμό του GPIO δίνονται στο κεφάλαιο 22 της αναφοράς [1] και επίσης στην αναφορά [11].

Κύκλωμα εξόδου.
Οι PMR0 και PMR1 ελέγχουν από κοινού ποιο σήμα εξόδου περιφερειακού θα συνδεθεί στον ακροδέκτη. Στη γενική αρχιτεκτονική του AVR32 μέχρι 4 διαφορετικά σήματα μπορούν να επιλεγούν, στον UC3A0512 όμως μπορούν μέχρι 3 διαφορετικά σήματα να επιλεγούν (σήματα Periph. *X* ouput data στο σχήμα). Η Ενότητα 12.7 της αναφοράς [1] περιγράφει αναλυτικά ποια σήματα μπορούν να επιλεγούν για κάθε έναν ακροδέκτη. Η λειτουργία (function) A του σχετικού πίνακα επιλέγεται με τιμή 00 στο ζεύγος καταχωρητών [PMR1:PMR0], η λειτουργία B με την τιμή 01 και η λειτουργία C με την τιμή 10. Ο καταχωρητής GPER επιλέγει αν το σήμα εξόδου θα προέρχεται από κάποια από τις 3 γραμμές περιφερειακών (0) ή θα ελέγχεται απ' ευθείας από τον καταχωρητή OVR (τιμή 1). Ο καταχωρητής ODER ελέγχει αν θα οδηγείται ο ακροδέκτης (τιμή 1) ή όχι (τιμή 0). Η δεύτερη περίπτωση συμβαίνει αν ο ακροδέκτης θα λειτουργεί σαν είσοδος οπότε δε θα πρέπει ο ακροδέκτης να οδηγείται και από το εσωτερικό κύκλωμα και από το συνδεδεμένο εξωτερικό. Σε περίπτωση που έχει επιλεγεί κάποιο περιφερειακό, το ρόλο του καταχωρητή ODER παίζει το σήμα Periph. *X* output enable. Δηλαδή τα περιφερειακά εκτός από τα σήματα εξόδου τους καθορίζουν και αν αυτά τα σήματα θα ενεργοποιούνται στον έξω κόσμο ή θα βρίσκονται σε κατάσταση υψηλής σύνθετης αντίστασης. Υπάρχει επί πλέον και η δυνατότητα η έξοδος να είναι οδηγείται σαν open-drain (καταχωρητής ODMER σε κατάσταση 1), δηλαδή να είναι σε κατάσταση υψηλής σύνθετης αντίστασης (Hi-Z) όταν πρόκειται για λογικό 1 και να πεφτει σε 0 στην αντίθετη περίπτωση. Αυτή η χρήση συμβαίνει αν πολλά σήματα εξόδου πρέπει να συνδεθούν μαζί σε συνδεσμολογία καλωδιωμένου-AND (wired-AND). Τότε, μπορεί να εκμεταλλευτούμε και την εσωτερική pull-up αντίσταση που ελέγχεται από τον καταχωρητή PUER. Η αντίσταση θα ανεβάζει σε υψηλό δυναμικό («ασθενές» όμως) τον ακροδέκτη που θα βρισκόταν αλλιώς σε απροσδιόριστη κατάσταση υψηλής αντίστασης.



Εικόνα 17 : Ένα στοιχειώδες κύτταρο GPIO

Κύκλωμα εισόδου.

Για την διαδρομή εισόδου από τον ακροδέκτη προς τα εσωτερικά περιφερειακά υπάρχει μία άλλη ομάδα καταχωρητών του 1-bit για κάθε ακροδέκτη που κυρίως αφορά θέματα διακοπών. Το σήμα εισόδου οδηγείται χωρίς πολυπλεξία κατ' ευθείαν και ταυτόχρονα προς όλα τα περιφερειακά που είναι συνδεδεμένα (δεν υπάρχει λόγος για επιλογή αφού κάθε περιφερειακό είναι υπεύθυνο αν θα αποδεχθεί ό,τι λαμβάνει). Το ίδιο σήμα οδηγείται και καταγράφεται στον καταχωρητή PVR για απ' ευθείας ανάγνωση της τιμής του σήματος. Ειναι προφανές ότι για να επιτραπεί η είσοδος σήματος από τον εξωτερικό κόσμο, θα πρέπει ο απομονωτής εξόδου τριών καταστάσεων να είναι απομονωμένος με κατάλληλο προγραμματισμό. Το σήμα εισόδου οδεύει επίσης και μέσα από κατάλληλα κυκλώματα προς τον ελεγκτή διακοπών. Ο καταχωρητής GFER καθορίζει αν το σήμα θα φιλτραριστεί (1) ή όχι (0) μέσα από το φίλτρο Glitch που καταστέλλει τυχόν αιχμές που πιθανόν να δημιουργηθούν όταν το σήμα έχει προέλθει από μηχανικούς διακόπτες. Ο καταχωρητής IER ενεργοποιεί (με λογικό 1) τις διακοπές από τον συγκεκριμένο ακροδέκτη και ο από κοινού συνδυασμός των [IMR1:IMR0] καθορίζει με τι είδους μετάβαση του σήματος θα προκαλείται διακοπή (00 για μετάβαση προς τα πάνω και κάτω, 01 για μετάβαση μόνο προς τα πάνω, και 10 για μετάβαση μόνο προς τα κάτω). Τα σήματα διακοπών από πολλούς ακροδέκτες οδεύουν προς τον ελεγκτή διακοπών μέσω μίας πύλης OR. Η απόφαση για το ποιος ακροδέκτης προκάλεσε τη διακοπή λαμβάνεται αφού αναγνωστεί ο καταχωρητής PVR.

Προγραμματισμός καταχωρητών του GPIO

Οι ακροδέκτες ομαδοποιούνται σε ομάδες των 32 (η ομαδοποίηση αυτή δεν έχει απόλυτη ταύτιση με την ομαδοποίηση στις πόρτες PA,PB,PC και PX). Κάθε ομάδα έχει ξεχωριστό καταχωρητή διαρρύθμισης (των 32 bits) για κάθε λειτουργία ελέγχου του κυκλώματος. Έτσι για παράδειγμα δημιουργείται ένας μεγάλος καταχωρητής GPER για κάθε ομάδα που ελέγχει 32

ακροδέκτες. Κάθε ομάδα (θύρα – port – στην ορολογία του GPIO) έχει μία διεύθυνση βάσης η οποία προστίθεται στη διεύθυνση βάσης του GPIO (Εικόνα 5). Στη διεύθυνση που προκύπτει πρέπει να προστεθεί με τη σειρά της η διεύθυνση μετατόπισης του καταχωρητή στον οποίο πρέπει να γίνει προσπέλαση. Οι διευθύνσεις βάσης της κάθε θύρας και μετατόπισης κάθε καταχωρητή του GPIO δίνονται στην ενότητα 22.5 της αναφοράς [1]. Εκεί περιγράφεται και ο τρόπος υπολογισμού που δίνει σε ποια ομάδα και σε ποιο bit των καταχωρητών διαρρύθμισης ανήκει ο κάθε ακροδέκτης. Για να μπορέσει να γίνει ο υπολογισμός αυτός πρέπει να βρεθεί από τον πίνακα 12-9 της αναφοράς [1] ο αριθμός του GPIO pin που έχει ο ακροδέκτης. Για παράδειγμα ο εξωτερικός ακροδέκτης PX01 αντιστοιχεί στον GPIO ακροδέκτη 99 (GPIO pin). Με βάση τους τύπους που δίνονται στο τμήμα 22.5 βρίσκουμε ότι ο ακροδέκτης αυτός αντιστοιχεί στην θύρα 3 γιατί floor(99/32)=3. Η διαρρύθμιση των ακροδεκτών αυτής της θύρας έχει διεύθυνση βάσης 0x0300 η οποία προστιθέμενη στη βάση του GPIO (0xFFFF1000) δίνει 0xFFFF1300. Το bit σε κάθε καταχωρητή διαρρύθμισης για τον ακροδέκτη PX00 είναι 99 mod 32 = 3. Έτσι θα πρέπει να προγραμματίζεται κάθε φορά το τέταρτο bit όποιου καταχωρητή ενδιαφέρει. Κάθε καταχωρητής έχει διάφορους τρόπους προσπέλασης ανάλογα με το αν θέλουμε να κάνουμε ανάγνωση ή εγγραφή όλων των bits, να σβήσουμε (λογικό 0), να ανάψουμε (λογικό 1) και να αντιστρέψουμε (toggle) συγκεριμένα bits. Συνεχίζοντας το προηγούμενο παράδειγμα, βλέπουμε ότι ο ακροδέκτης PX99 μπορεί να χρησιμοποιηθεί σαν ακροδέκτης εξόδου του USART0 και συγκεκριμένα του σήματος TXD (εκπομπής σειριακών δεδομένων). Η επιλογή αυτή είναι η δεύτερη, επομένως θα πρέπει το bit 4 του καταχωρητή PMR0 της θύρας 3 να τεθεί σε λογικό 0, και το αντίστοιχο του PMR1 σε λογικό 1 (ο συνδυασμός [PMR1(3):PMR0(3)]=10_2=2_{10} επιλέγει τη δεύτερη διασύνδεση). Η διεύθυνση μετατόπισης για λειτουργία μηδενισμού bit (clear) του καταχωρητή PMR0 είναι 0x18 (βλ. Πίνακα 22-2 της αναφοράς [1]). Αντίστοιχα, η διεύθυνσης μετατόπισης για λειτουργία ενεργοποίησης bit (set) του καταχωρητή PMR1 είναι 0x14. Στις λειτουργίες ενεργοποίησης, μηδενισμού και αντιστροφής (toggle) ο αντίστοιχος καταχωρητής γράφεται με 1 στα bits που είναι επιθυμητή η λειτουργία και 0 όπου δεν πρέπει να αλλαχθεί το σχετικό bit. Με άλλα λόγια θα πρέπει να γραφεί η λέξη 0x00000008 (0000 0000 0000 0000 0000 0000 0000 1000 σε δυαδικό σύστημα) στις διευθύνσεις 0xFFFF1314 και 0xFFFF1318 για να επιλεγεί η λειτουργία του PX01 σαν σήμα TXD του USART0. Φυσικά, δεν αρκεί αυτό, αλλά πρέπει το bit 3 του καταχωρητή GPER να τεθεί σε τιμή 0 (αν δεν είναι ήδη), κ.ο.κ.
Εναλλακτικά, θα μπορούσαν να είχαν χρησιμοποιηθεί οι σχετικές διευθύνσεις 0x10 και 0x20 που αντιστοιχούν στις λειτουργίες Read/Write των ίδιων καταχωρητών (PMR0 και PMR1). Τότε όμως, θα έπρεπε να ληφθεί μέριμνα για να μην αλλάξουν τα υπόλοιπα bits, δηλαδή να γίνει ανάγνωση, λογικές πράξεις μασκαρίσματος και επανεγγραφή. Αυτά αποφεύγονται όμως με την ύπαρξη διαφορετικών διευθύνσεων για διαφορετικές λειτουργίες πάνω στον ίδιο καταχωρητή.

Το παράδειγμα `gpio_example.c`

Από τα προηγούμενα φαίνεται πόσο πολύπλοκο είναι να γραφεί σε Assembly έστω και μία απλή εφαρμογή σαν αυτή που παρουσιάστηκε στο τμήμα II.3. Επί πλέον θα πρέπει να υπάρχει και γνώση της συνδεσμολογίας πάνω στην πλακέτα. Από το φύλλο 7 της αναφοράς [2] που περιέχει το πλήρες κύκλωμα της πλακέτας βρίσκουμε ότι τα LED 3 και 4 είναι συνδεδεμένα στους ακροδέκτες PB29 και PB30, αντίστοιχα (το διάγραμμα ξεκινάει την αρίθμηση των LEDs από το 0 και όχι από το 1 όπως είναι στην άνω όψη της πλακέτας). Επί πλέον τα LED είναι συνδεδεμένα με την κάθοδό τους στους ακροδέκτες, ενώ η άνοδος συνδέεται σε θετική τροφοδοσία μέσω αντίστασης περιορισμού ρεύματος. Επομένως ανάβουν όταν οι αντίστοιχοι ακροδέκτες είναι σε λογικό 0. Ο πιεστικός διακόπτης PB0 (αντιστοιχεί στον SW4 του κυκλωματικού διαγράμματος) συνδέεται στον ακροδέκτη PX16 με τέτοιο τρόπο ώστε αν πιέζεται να τον οδηγεί σε λογικό 0.
Όλες αυτές οι πληροφορίες είναι ενσωματωμένες σε αρχεία επικεφαλίδας γλώσσας C που δίνονται στο Software Framework. Σε ανάλογα αρχεία επικεφαλίδας και πηγαίου κώδικα

υπάρχουν έτοιμες συναρτήσεις για το GPIO (αλλά και για κάθε άλλο περιφερειακό) που διευκολύνουν τον προγραμματισμό.
Ο τρόπος που επιτυγχάνεται η λειτουργία των αναλαμπών των LED αναπτύσσεται παρακάτω. Σε έναν ατέρμονο βρόχο γίνονται διαδοχικά τα εξής βήματα: Άναμα ή σβήσιμο του LED3 ανάλογα με την τιμή της μεταβλητής state, έλεγχος αν έχει πατηθεί ο διακόπτης PB0, απόφαση για άναμα η σβήσιμο του LED4 και πάλι πίσω. Το τελευταίο βήμα βρίσκεται σε έναν άλλο βρόχο που παίζει και το ρόλο της χρονικής καθυστέρησης για την μετάβαση των καταστάσεων που επηρρεάζουν το LED3. Οι καταστάσεις που ορίζονται από την `state` είναι 0, 1, 2, 3 και η κάθε μία διαδοχικά ανάβει και σβήνει το LED3. η ακολουθία των καταστάσεων είναι 0→1→2→3→0. Χρησιμοποιούνται έτοιμες συναρτήσεις της C που ορίζονται στην επικεφαλίδα `gpio.h` και ο κώδικάς τους βρίσκεται στο `gpio.c` του Software Framework. Αυτά τα δύο αρχεία συμπεριλαμβάνονται στο project με τον τρόπο που έχει περιγραφεί στο τμήμα II.3.
Οι συναρτήσεις που χρησιμοποιούνται στο παράδειγμα φαίνονται στην Εικόνα 18 μαζί με τη λειτουργία τους και το ποιους καταχωρητές επηρρεάζουν.

Συνάρτηση ορισμένη στα gpio.h και gpio.c	Λειτουργία στον Ακροδέκτη	Καταχωρητές GPIO (bit)
`gpio_get_pin_value(uint)`	Διαβάζει την τιμή του	←PVR
`gpio_clr_gpio_pin(uint)`	Τον μηδενίζει	OVR←0 ODER←1 GPER←1
`gpio_set_gpio_pin(uint)`	Τον θέτει σε τιμή 1	OVR←1 ODER←1 GPER←1
`gpio_tgl_gpio_pin(uint)`	Αλλάζει την τιμή του	OVR←not(OVR) ODER←1 GPER←1
`gpio_enable_pin_glitch_filter(uint)`	Ενεργοποιεί το φίλτρο	GFER←1

Εικόνα 18 : Μερικές από τις συναρτήσεις της βιβλιοθήκης `gpio.c`

Όλες οι παραπάνω συναρτήσεις παίρνουν σαν παράμετρο έναν μη προσημασμένο ακέραιο που αναπαριστά τον GPIO ακροδέκτη και γράφουν ή διαβάζουν με τον κατάλληλο τρόπο τους καταχωρητές που απαιτούνται σε κάθε περίπτωση. Οι συναρτήσεις που διαβάζουν καταχωρητές (όπως η `gpio_get_pin_value`) επιστρέφουν έναν ακέραιο που μπορεί να είναι 0 ή 1.
Σαν παράδειγμα ας δούμε πως ανάβει το LED4 αν πατηθεί ο διακόπτης PB0. Το τμήμα κώδικα που το επιτυγχάνει είναι το παρακάτω:

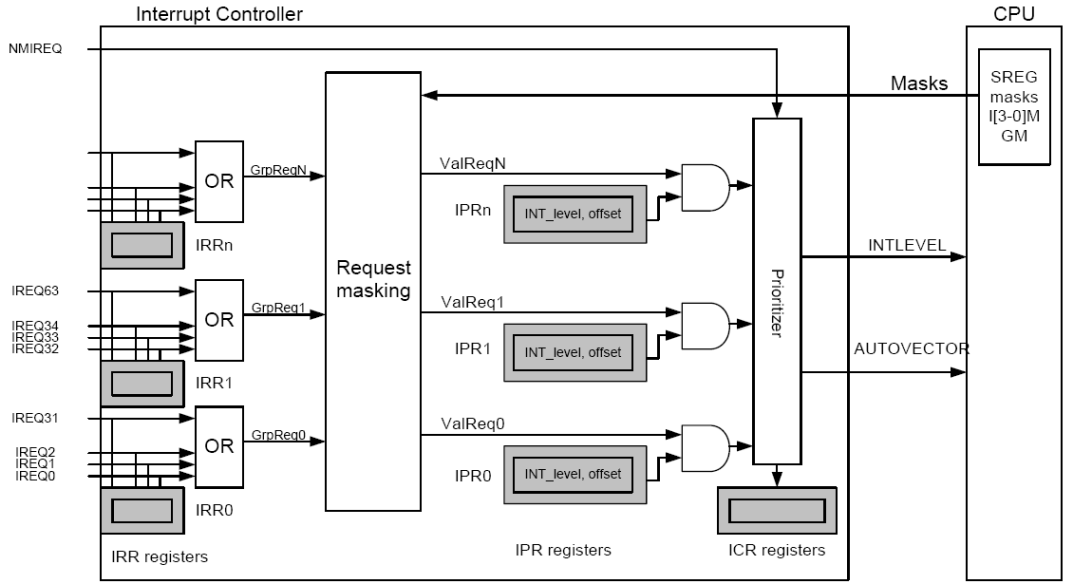
```
if (gpio_get_pin_value(GPIO_PIN_EXAMPLE_3) == 0)
      gpio_clr_gpio_pin(GPIO_PIN_EXAMPLE_2);
```

Αν διαβαστεί τιμή 0 (διακόπτης πατημένος) τότε, ο ακροδέκτης του LED τίθεται σε τιμή 0, δηλαδή το LED μπορεί να ανάψει. Σαν παράμετρος στην `gpio_clr_gpio_pin` δίνεται η σταθερά `GPIO_PIN_EXAMPLE_2` που είναι ορισμένη στο πάνω μέρος του κώδικα σαν `LED3_GPIO`. Η σταθερά `LED3_GPIO` είναι με τη σειρά της ορισμένη στο αρχείο `evk1100.h` που ενσωματώνεται (`#include`) στο `board.h` και αυτό με τη σειρά του στο κυρίως αρχείο (η αρίθμηση ξεκινάει από `LED0_GPIO` έτσι το LED4 αντιστοιχεί στο `LED3_GPIO` του κώδικα). Η τιμή της σταθεράς `LED3_GPIO` βρίσκεται ότι είναι `AVR32_PIN_PB30`, για την περίπτωση που η πλακέτα δεν είναι Revision A, κάτι που ισχύει στην περίπτωσή μας. Το αρχείο `evk1100.h` ουσιαστικά περιέχει κυρίως την πληροφορία διασύνδεσης διαφόρων εξωτερικών περιφερειακών της πλακέτας όπως τα LED, οι διακόπτες, οι διάφορες θύρες επικοινωνίας, με τους ακροδέκτες του μικροελεγκτή. Γι΄αυτό είναι απαραίτητη η μελέτη του από τον προγραμματιστή. Συνεχίζοντας, η τιμή `AVR32_PIN_PB30` είναι ορισμένη στο αρχείο `avr/uc3a0512es.h` που καλείται από το `avr32/io.h` και τελικά

είναι ίση με 62. Δηλαδή η συνάρτηση παίρνει το όρισμα 62. Πράγματι, ο πίνακας 12.9 της αναφοράς [1] δείχνει ότι ο ακροδέκτης PB30 αντιστοιχεί στον ακροδέκτη GPIO με αριθμό 62. Η συνάρτηση `gpio_clr_gpio_pin` πλέον αναλαμβάνει τα υπόλοιπα, δηλαδή να υπολογίσει την θύρα GPIO που θα προγραμματιστεί καθώς και το bit από τους σχετικούς καταχωρητές που θα πρέπει να επηρρεαστεί.
Με αυτό τον τρόπο, δηλαδή μέσω των ορισμών στις επικεφαλίδες ο προγραμματιστής δε χρειάζεται να υπολογίζει κάθε φορά ποιο bit θα προγραμματίσει και σε ποια θύρα διαρρυθμισης θα απευθυνθεί. Οι απαραίτητες επικεφαλίδες εισάγονται κάθε φορά στο project με τον τρόπο που περιγράφηκε στο τμήμα II.3. Μελετώντας τα αρχεία `gpio.h` και `gpio.c` μπορούν να βρεθούν και οι υπόλοιπες συναρτήσεις που προσφέρει η Atmel για τον προγραμματισμό του GPIO ελεγκτή. Ο προγραμματισμός του GPIO ελεγκτή είναι απαραίτητος κάθε φορά που χρειάζεται αν χρησιμοποιηθεί κάποιο εσωτερικό περιφερειακό, γιατί μέσω του GPIO θα επικοινωνήσει με τον εξωτερικό κόσμο. Ο προγραμματισμός των υπόλοιπων περιφερειακών είναι ανάλογος με αυτόν που περιγράφηκε για το GPIO και γι' αυτό στα επόμενα τμήματα η περιγραφή θα είναι πιο συνοπτική.

III.3 Ελεγκτής εσωτερικών διακοπών (INTC).

Ο ελεγκτής εσωτερικών διακοπών παίρνει σήματα που παράγουν τα διάφορα περιφερειακά (όπως ο GPIO ή ένα ADC) και προωθεί διακοπές με προτεραιότητες προς την μονάδα επεξεργασίας, εφ' όσον το επίπεδο προτεραιότητας επιτρέπεται, δηλαδή δεν έχει απενεργοποιηθεί. Επίσης προωθεί και το σήμα διακοπής ανώτατης προτεραιότητας NMI που δεν επιδέχεται μασκάρισμα. Η Εικόνα 19 δείχνει σε κυκλωματικό διάγραμμα τον ελεγκτή διακοπών (βλ. αναφορές [1] (Κεφ.16) και [12]).



Εικόνα 19 : Ο ελεγκτής εσωτερικών διακοπών

Ο ελεγκτής ομαδοποιεί τα σήματα που προκαλούν τις διακοπές σε ομάδες των 32 σημάτων. Ο αριθμός των ομάδων είναι 64 δίνοντας τη δυνατότητα χειρισμού μέχρι 2048 σημάτων. Οι είσοδοι των σημάτων είναι μόνιμα δρομολογημένες με συγκεκριμένα περιφερειακά, δηλαδή δεν υπάρχει η δυνατότητα επαναπρογραμματισμού τους από το χρήστη. Η δρομολόγηση των σημάτων φαίνεται στον πίνακα 12.3 της αναφοράς [1]. Εκεί καθορίζεται το σήμα διακοπής που ανήκει σε κάποια ομάδα (στήλη Group - μέχρι 64 ομάδες) και έχει κάποιο αύξοντα αριθμό γραμμής (στήλη Line – μέχρι 32 γραμμές) σε ποιο σήμα περιφερειακού συνδέεται (στήλες

Module και Signal). Από τον πίνακα φαίνεται ότι δεν χρησιμοποιούνται όλες οι γραμμές εισόδου του ελεγκτή διακοπών. Έτσι βλέπουμε ότι για παράδειγμα το σήμα διακοπής που παράγει η μονάδα σειριακής επικοινωνίας USART0 οδεύει στο πρώτο σήμα (Line 0) της έκτης ομάδας (Group 5) του ελεγκτή.

Τα σήματα κάθε ομάδας προωθούνται από κοινού μέσω μιας πύλης OR προς την υπομονάδα Request Masking που ελέγχει αν επιτρέπεται να προωθηθούν προς τον πυρήνα του μικροελεγκτή. Η απόφαση αυτή γίνεται στη βάση του επιπέδου προτεραιότητας διακοπής που ανήκει το σήμα. Αν το σήμα έχει υψηλότερο επίπεδο προτεραιότητας από αυτό που έχει καθοριστεί στην υπομονάδα Request Masking τότε, θα γίνει προώθηση. Οι προτεραιότητες καθορίζονται ανά ομάδα, δηλαδή τα σήματα μίας ομάδας έχουν κοινή προτεραιότητα. Τα σήματα κάθε ομάδας καταγράφονται σε έναν ειδικό καταχωρητή των 32 bits IRR που έχει η κάθε ομάδα με σκοπό το διαχωρισμό της πηγής προτεραιότητας από την υπορουτίνα χειρισμού διακοπών (η πύλη OR καταστρέφει τη δυνατότητα διαχωρισμού). Επίσης, η κάθε ομάδα έχει δικό της διάνυσμα διακοπής (Autovector) κοινό για όλα τα σήματα που της ανήκουν. Με άλλα λόγια ο χειριστής διακοπών θα πρέπει να είναι σχεδιασμένος έτσι ώστε να μπορεί να χειρίζεται όλα αυτά τα σήματα, εφ' όσον χρησιμοποιούνται. Ο διαχωρισμός στις διάφορες περιπτώσεις θα γίνεται με τη βοήθεια του καταχωρητή IRR.

Η υπομονάδα Request Masking σε συνεργασία με τους καταχωρητές IPR που περιέχουν το επίπεδο προτεραιότητας και το διάνυσμα διακοπής για κάθε ομάδα αποφασίζει ποιες ομάδες θα προωθήσουν τις διακοπές τους. Η μονάδα επεξεργασίας κρατάει ενήμερη την υπομονάδα μασκαρίσματος για την τρέχουσα κατάσταση του πυρήνα (βλ. και Εικόνα 2). Έτσι, αν για παράδειγμα η CPU βρίσκεται σε κατάσταση (mode) Interrupt1 (βλ. Εικόνα 2) μόνο διακοπές που προέρχονται από ομάδες που έχουν προτεραιότητα Interrupt2 και Interrupt3 θα προωθηθούν.

Ο διαιτητής προτεραιοτήτων αποφασίζει τελικά ποιας ομάδας τη διακοπή θα προωθήσει, στην περίπτωση που έχει παραλάβει ταυτόχρονα πάνω από μία διακοπές, στη βάση των μεταξύ τους προτεραιοτήτων. Αν πρόκειται για ίδιες προτεραιότητες προηγείται η ομάδα με τον μεγαλύτερο αύξοντα αριθμό. Τότε στέλνει στην CPU ένα σήμα διακοπής που δείχνει το επίπεδό της καθώς και τα διάνυσμα διακοπής (που είναι μία διεύθυνση μετατόπισης των 14 bits η οποία γίνεται λογικό-OR με τον καταχωρητή συστήματος EVBA) προκειμένου να εκτραπεί η ροή του προγράμματος προς την υπορουτίνα χειρισμού. Ταυτόχρονα ο καταχωρητής ICR περιέχει και την πληροφορία του ποια ομάδα έδωσε διακοπή.

Η μονάδα επεξεργασίας τότε, τίθεται σε κατάσταση προτεραιότητας ίδια με αυτήν της διακοπής έτσι ώστε να μην ξαναγίνει διακοπή από την ίδια πηγή ή άλλη πηγή χαμηλότερης ή ίδιας προτεραιότητας και τότε αρχίζει την εξυπηρέτηση της διακοπής. Είναι στην ευθύνη του προγραμματιστή να άρει την αιτία της διακοπής εφ'όσον αυτή έχει εξυπηρετηθεί (π.χ. για την περίπτωση του ελεγκτή GPIO πρέπει να καθαριστεί ο καταχωρητής IFR) ώστε να μην ξαναπροκληθεί διακοπή αμέσως μετά την επιστροφή από την υπορουτίνα χειρισμού της.

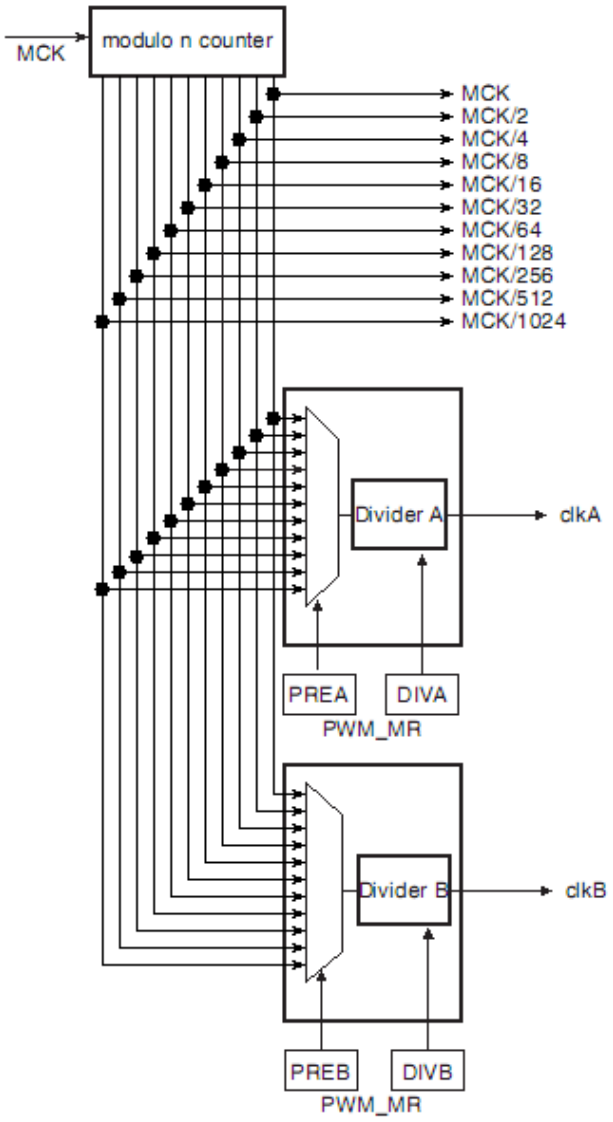
Σαν παράδειγμα προγραμματισμού με χρήση διακοπών θα γίνει μία παραλλαγή του προηγούμενου προγράμματος `gpio_example.c`. Στο προηγούμενο πρόγραμμα ο έλεγχος για το αν πατήθηκε ο διακόπτης γινόταν διαρκώς μέσα σε ένα βρογχο. Αυτή η μέθοδος (polling) έχει το μειονέκτημα ότι απασχολείται συνεχώς ο επεξεργαστής. Στην νέα παραλλαγή θα γίνει χρήση διακοπών, έτσι ώστε αν πατηθεί ή αφεθεί ο διακόπτης, ο επεξεργαστής να παραλαμβάνει μία διακοπή και να την εξυπηρετεί με σκοπό να σβήσει ή να ανάψει το LED4. Το διάστημα που δε συμβαίνει κάποιο τέτοιο γεγονός ο μικροελεγκτής θα είναι ελεύθερος να κάνει άλλες εργασίες. Στην συγκεκριμένη περίπτωση απλώς να αλλάζει τις καταστάσεις όπως πριν προκειμένου να αναβοσβήνει το LED3.

Ο νέος κώδικας με όνομα `gpio_int.c` παρουσιάζεται στο Παράρτημα. IV.2. Οι απαραίτητοι οδηγοί είναι οι GPIO και INTC. Η συνάρτηση με όνομα `PB0_int_handler` που ορίζεται στον κώδικα είναι ο χειριστής διακοπών που προκαλούνται από την πίεση ή την απελευθέρωση του διακόπτη PB0. Μέσα στη συνάρτηση γίνεται έλεγχος για το αν ο διακόπτης είναι σε λογικό

μηδέν (ελεύθερος) ή 1 (πατημένος) και ανάλογα σβήνει ή ανάβει, αντίστοιχα, το LED D4 (`LED3_GPIO`).
Είναι σημαντικό πριν την έξοδο από μία ρουτίνα χειρισμού διακοπών να αίρεται η αιτία της διακοπής, ώστε μετά την επιστροφή στον κυρίως κορμό του προγράμματος να μην προκληθεί εκ νέου διακοπή. Αυτό γίνεται με την κλήση της συνάρτησης `gpio_clear_pin_interrupt_flag(GPIO_PUSH_BUTTON_0)` η οποία και απενεργοποιεί το IFR bit (Interrupt Flag Register) που αντιστοιχεί στον πιεστικό διακόπτη. Να σημειωθεί ότι επειδή ανά οκτώ είσοδοι του GPIO συνδυάζονται με λογικό-OR για να δώσουν μία διακοπή, κανονικά τα IFR bits θα πρέπει να ελέγχονται και στην αρχή της ρουτίνας εξυπηρέτησης προκειμένου να διαπιστωθεί ποια από τις οκτώ γραμμές ήταν αυτή που προκάλεσε την διακοπή. Εδώ αυτό δε συμβαίνει γιατί περιμένουμε μόνο ενός είδους διακοπή. Να σημειωθεί επίσης ότι πριν από τον ορισμό του κάθε χειριστή διακοπών πρέπει να μπαίνει η γραμμή `__attribute__((__interrupt__))` (στον Gnu C Compiler, βλ. και αναφορές [8] και [9]) προκειμένου να μπορεί να γίνει επιστροφή στο κυρίως πρόγραμμα.
Προχωρώντας στον κυρίως κώδικα, γίνεται αρχικά απενεργοποίηση των διακοπών με την `Disable_global_interrupt()`. Κατόπιν γίνεται αρχικοποίηση του ελεγκτή διακοπών με την `INTC_init_interrupts()`. Η αρχικοποίηση συνίσταται στο να καταχωρηθεί ένας εξ' ορισμού (default) χειριστής διακοπών για όλες τις δυνατές διακοπές και να τεθεί προτεραιότητα INT0 για όλες τις διακοπές. Στη συνέχεια ενεργοποιείται το φίλτρο αιχμών (glitch) (κλήση συνάρτησης `gpio_enable_pin_glitch_filter`) και κατόπιν ενεργοποιούνται (`gpio_enable_pin_interrupt`) οι καταχωρητές IER του GPIO για τον διακόπτη και IMR0 και IMR1 του GPIO για το είδος του γεγονότος που θα προκαλέσει τη διακοπή. Στην προκειμένη περίπτωση διακοπή προκαλείται είτε με πίεση είτε με απελευθέρωση του διακόπτη PB0. Η αντιστοίχιση της γραμμής που θα δώσει διακοπή με τη ρουτίνα που θα τη χειριστεί επιτυγχάνεται με τη συνάρτηση `INTC_register_interrupt`. Έτσι, η συνάρτηση `PB0_int_handler` θα κληθεί αν η γραμμή που αντιστοιχεί στη σταθερά `AVR32_GPIO_IRQ_11` ενεργοποιηθεί. Η προτεραιότητα αυτής της διακοπής θα έχει επίπεδο `INT0`. Στον πίνακα 12-3 της αναφοράς [1] που περιγράφει τη μόνιμη διασύνδεση μεταξύ σημάτων των περιφερειακών και γραμμών/ομάδων στον ελεγκτή διακοπών παρατηρούμε ότι όλα τα σήματα του GPIO οδεύουν στην τρίτη ομάδα γραμμών (Group2) και συνολικά υπάρχουν 14 γραμμές αυτής της ομάδας αφιερωμένες σε αυτό το σκοπό. Ο διακόπτης πίεσης PB0 είναι συνδεδεμένος στον ακροδέκτη PX16, που αντιστοιχεί (από τον πίνακα 12.9 της [1]) στο GPIO_PIN88. Επειδή, οι γραμμές του GPIO ανά οκτώ συνδέονται μέσω της OR σε μία γραμμή, συμπεραίνουμε ότι αυτή η γραμμή είναι η 11, για αυτό και η συνάρτηση `PB0_int_handler` καλείται με δεύτερη παράμετρο τη σταθερά `AVR32_GPIO_IRQ_11`. Η σταθερά αυτή είναι ορισμένη στο αρχείο επικεφαλίδα `avr32/uc3a0512es.h` και έχει τιμή 75. Πράγματι, 2·Group+Line για την περίπτωση του PB0 δίνει 2·32+11=75. Φυσικά, όλοι αυτοί οι υπολογισμοί δεν είναι απαραίτητο να γίνονται από τον προγραμματιστή αφού περιέχονται στα αρχεία επικεφαλίδων. Έτσι, αρκεί η μελέτη τους για την επιλογή της κατάλληλης συμβολικής σταθεράς κάθε φορά. Τελειώνοντας, γίνεται ενεργοποίηση των διακοπών με την `Enable_global_interrupt()`.
Ο κυρίως κώδικας τελειώνει με ένα ατέρμονο βρόχο που αναβοσβήνει το LED3 χρησιμοποιώντας μία καθυστέρηση ανάμεσα στις αλλαγές κατάστασης. Η χρήση της `r=rand()` μέσα στο βρόχο της καθυστέρησης γίνεται για να μην ακυρώσει το βρόχο ο μεταγλωττιστής σε περίπτωση που έχει κληθεί (αυτή είναι η αρχική του ρύθμιση) με την επιλογή της βελτιστοποίησης. Αν χρησιμοποιηθεί κάποια άλλη εντολή όπως η κενή (`;`) ή μία πράξη με κανονικότητα (π.χ `c=c+a*b;`) ο βρόχος θα αναιρεθεί, εκτός αν δοθεί επιλογή μη χρήσης βελτιστοποίησης που είναι -O0 για τον gcc (βλ. [9]).

III.4 Διαμορφωτής εύρους παλμών (Pulse Width Modulator – PWM).

Ο μικροελεγκτής διαθέτει 7 γεννήτριες τετραγωνικών κυματομορφών οι οποίες μπορούν να προγραμματιστούν ανεξάρτητα ως προς τις εξής παραμέτρους: ρολόι χρονισμού, περίοδος μετρημένη σε κύκλους χρονισμού, διάρκεια ενεργού παλμού (δηλαδή μπορει να ρυθμιστεί το duty cycle), πολικότητα κυματομορφής και ένα βαθμό ελευθερίας σε σχέση με τη φάση της κυματομορφής που δίνεται σαν ευθυγράμμιση ως προς το κέντρο της περιόδου (center aligned) ή την έναρξη της περιόδου (left aligned).



Εικόνα 20 : Η γεννήτρια ρολογιών του PWM

Μέρος του υποκυκλώματος PWM είναι η γεννήτρια ρολογιών που φαίνεται στην Εικόνα 20. η γεννήτρια λαμβάνει ένα σήμα ρολογιού από τον διαχειριστή ισχύος (Power Manager) ο οποίος μεταξύ άλλων επιλέγει τι είδους ρολόι θα αποσταλλεί σε κάθε υποκύκλωμα. Το ρολόι αυτό υπόκειται σε δύο διαδοχικές επεξεργασίες: Πρώτα διαίρεση συχνότητας με δυνάμεις του δύο (1, 2, 4, 8 ως 1024) και ύστερα διαίρεση με ακεραίους (2, 3, 4, 5, ως 255). Υπάρχουν δύο ανεξάρτητοι διαιρέτες συχνότητας με ακεραίους που επιλέγει ο καθένας πιο από τα σήματα MCK, MCK/2,…, MCK/1024 θα διαιρέσει. Ο προγραμματισμός των διαιρετών με ακεραίους

επομένως χρειάζεται 4 bits για την επιλογή της πρώτης διαίρεσης (αυτής με δυνάμεις του 2) και άλλα 8 bits για την επιλογή της δεύτερης διαίρεσης για κάθε ακέραιο διαιρέτη. Αυτά τα bits εμφανίζονται σαν πεδία PREA, DIVA, PREB, DIVB του καταχωρητή MR. Η τελική επιλογή της συχνότητας ρολογιού του κάθε PWM καναλιού γίνεται με τα πεδία CPRE του καταχωρητή CMRx (x=0, 1, ..., 6) και μπορεί να είναι είτε από κάποια γραμμή του πρώτου διαιρέτη με δυνάμεις του 2, είτε από την έξοδο ενός από του δύο επόμενους διαιρέτες με ακεραίους. Λεπτομέρειες δίνονται στην ενότητα 32 της αναφοράς [1].
Το κυρίως κύκλωμα του κάθε PWM καναλιού αποτελείται από έναν δυαδικό μετρητή των 20 bits και έναν ψηφιακό συγκριτή. Η συνεργασία τους καθορίζει πότε και για πόση διάρκεια θα είναι ενεργός ο κάθε παλμός αλλά και την περίοδο της επανάληψης.
Η περίοδος της κυματομορφής για κάθε κανάλι προγραμματίζεται στον καταχωρητή CPRDx που έχει το κάθε κανάλι x. Μόνο τα 20 χαμηλά bits λαμβάνονται υπ' όψη και η περίοδος σε πραγματικό χρόνο είναι για την περίπτωση της ευθυγραμμισμένης στην αρχή κυματομορφής ίση με CPRDx φορές την περίοδο του ρολογιού με το οποίο χρονίζεται το κανάλι. Για την περίπτωση της κεντρικά ευθυγραμμισμένης κυματομορφής η περίοδος διπλασιάζεται.
Οι καταχωρητές CDTYx (πάλι μόνο τα 20 χαμηλά bits) καθορίζουν το ποσοστό της περιόδου που είναι **δεν** είναι ενεργός (duty cycle) η κυματομορφή. Η σχέση που δίνει το duty cycle είναι για την περίπτωση της αριστερά ευθυγραμμισμένης κυματομορφής: duty cycle=1-CDTYx/CPRDx. Φυσικά, πρέπει η τιμή που αποθηκεύεται στον καταχωρητή CDTYx να είναι μεταξύ 0 και CPRDx. Επομένως η ακρίβεια καθορισμού του duty cycle βελτιώνεται όσο αυξάνεται η τιμή του CPRDx.
Το κύκλωμα λειτουργεί ως εξής: Όσο αυξάνεται ο καταμετρητής γίνεται σύγκριση με την τιμή που είναι αποθηκευμένη στο CPRDx. Όταν γίνει ίση με αυτήν την τιμή ο καταμετρητής είτε μηδενίζεται (αριστερή ευθυγράμμιση – πεδίο CALG του καταχωρητή CMRx), είτε αντιστρέφεται η φορά καταμέτρησης προς τα κάτω (κεντρική ευθυγράμμιση). Αυτό σημαίνει ότι στη δεύτερη περίπτωση η περίοδος διπλασιάζεται και γίνεται το γινόμενο 2·(Περίοδος Εισερχόμενου Ρολογιού)·(Τιμή CPRDx). Όταν ο καταμετρητής γίνεται μηδέν, επαναλαμβάνονται τα προηγούμενα. Ταυτόχρονα, αποφασίζεται η τιμή της εξόδου του PWM καναλιού. Όταν η τιμή του καταμετρητή γίνει μεγαλύτερη της τιμής του καταχωρητή CDTYx τότε, η έξοδος του καναλιού είναι σε λογικό «1» ή λογικό «0», ανάλογα με την τιμή που έχει το πεδίο CPOL του καταχωρητή CMRx. Η διάρκεια του ενεργού παλμού (0 ή 1) σε κύκλους ρολογιού είναι 1-CDTYx/CPRDx ή 1-2·CDTYx/CPRDx, ανάλογα με την ευθυγράμμιση.
Όσο λειτουργεί η παραγωγή της κυματομορφής, είναι δυνατόν να αλλάξουν οι παράμετροι περιόδου και διάρκειας παλμών. Προκειμένου να μην υπάρχουν ασυνέχειες στο μέση μίας περιόδου, υπάρχει ο καταχωρητής CUPDx που σκοπό έχει να παραλάβει τις νέες παραμέτρους και να ανανεώσει έναν από τους καταχωρητές CPRDx ή CDTYx στο τέλος μίας περιόδου κυματομορφής ανάλογα με την τιμή του πεδίου CPD του καταχωρητή CMRx. Αυτός είναι και ο προτεινόμενος τρόπος επαναπρογραμματισμού κατά τη διάρκεια λειτουργίας του PWM καναλιού.Επί πλέον υπάρχουν καταχωρητές ενεργοποίησης./απενεργοποίησης/κατάστασης καναλιών (ENA/DIS/SR) και καταχωρητές ενεργοποίησης/απενεργοποίσης/μάσκας διακοπών (IER/IDR/IMR).
Το πρόγραμμα που υπάρχει στο Παράρτημα IV.3 χρησιμοποιεί του οδηγούς για τη χρήση της μονάδας PWM που προσφέρονται από την Atmel. Η λειτουργία του προγράμματος είναι η εξής: Το LED3 αναβοσβήνει μονίμως. Το κόκκινο LED του δίχρωμου LED6 οδηγείται από ένα PWM κανάλι (συγκεκριμένα το κανάλι 2 γιατί η κόκκινη δίοδος του LED6 είναι συνδεδεμένη στον ακροδέκτη PB21 του μικροελεγκτή και αυτός μπορεί να οδηγηθεί μεταξύ άλλων από το κανάλι 2 του PWM). Συνεπώς η φωτεινότητα του LED6 θα μεταβάλλεται με τη διάρκεια που έχει ο κάθε παλμός σε μία περίδο. Η φωτεινότητα αυτή μειώνεται με διαδοχικά πατήματα του διακόπτη PB0 και αυξάνεται με διαδοχικά πατήματα του PB1. Όπως και πριν, γίνεται χρήση διακοπών για την ανίχνευση της κατάστασης των διακοπτών. Επί πλέον το LED2 ανάβει όσο είναι πατημένος ο διακόπτης PB0 και το LED4 ανάβει όσο είναι πατημένος ο PB1.
Το πρόγραμμα θα πρέπει να εισαχθεί σε ένα έργο του AVR32 Studio κατάλληλο για την πλακέτα EVK1100 στο οποίο θα έχουν προστεθεί οι οδηγοί GPIO,INTC και PWM (βλ. II.3).

Στο κυρίως πρόγραμμα (συνάρτηση `main`) η δομή τύπου `pwm_opt_t` που είναι ορισμένη στο `pwm.h` αναπαριστά τον καταχωρητή MR Χρησιμοποιείται σε συνδυασμό με τη συνάρτηση `pwm_init` για την αρχικοποίηση της γεννήτριας ρολογιών του PWM. Έχει προηγηθεί η απενεργοποίηση των διακοπών και η αρχικοποίηση του σχετικού ελεγκτή INTC και έχει γίνει σύνδεση της εξόδου του καναλιού 2 του PWM στον ακροδέκτη PB21 που ελέγχει το κόκκινο LED6 με τη συνάρτηση `gpio_enable_pin`.
Ακολουθεί ο προγραμματισμός του καναλιού PWM 2, δηλαδή ο καθορισμός του τύπου της κυματομορφής (ευθυγράμμιση, πολικότητα), η επιλογή συχνότητας από την γεννήτρια ρολογιών, η περίοδος και η διάρκεια παλμών. Για την εγγραφή στα πεδία των σχετικών καταχωρητών χρησιμοποείται μία δομή τύπου `avr32_pwm_channel_t` (είναι ορισμένη στο `avr32/pwm_120.h`) και η συνάρτηση `pwm_channel_init`. Η περίοδος έχει τεθεί σε 20 κύκλους ρολογιού συχνότητας 115200/256 Hz, δηλαδή περίπου 44msec (115200Hz είναι η συχνότητα του ταλαντωτή RC που είναι το εξ' ορισμού ρολόι για τον μικροελεγκτή). Η διάρκεια του παλμού είναι πολύ μικρή (ο παλμός είναι ενεργός όσο ο καταμετρητής είναι μεγαλύτερος της τιμής CDTY (=19), κάτι που συμβαίνει σε έναν μόνο κύκλο από τους 20. Επειδή η πολικότητα είναι κανονική, κάθε περίοδος θα βρίσκεται το μεγαλύτερο ποσοστό του χρόνου σε λογικό 0, επομένως το LED6 θα έχει σχεδόν πλήρη φωτεινότητα αρχικά. Από τα προηγούμενα συμπεραίνουμε ότι η φωτεινότητα είναι ανάλογη της τιμής του πεδίου CDTY και μπορεί να πάρει 20 τιμές, από το 0 ως το 19. Αν χρειαζόταν μεγαλύτερη ακρίβεια στη ρύθμιση θα έπρεπε να αυξηθεί η περίοδος. Επίσης εξ' αιτίας της πραγματικής περιόδου που είναι σχετικά μεγάλη (44msec), το LED6 φαίνεται να τρεμοπαίζει. Για να εκμεταλευθούμε το φαινόμενο του μεταισθήματος πρέπει να μειωθεί η περίοδος, δηλαδή να αυξηθεί η συχνότητα ρολογιού που ρυθμίζει το PWM κανάλι (Π.χ. θα μπορούσε να γίνει διαίρεση του κεντρικού ρολογιού με 64 αντί για 256.
Μετά τον προγραμματισμό του καναλιού PWM (`pwm_channel_init`) ακολουθεί η ενεργοποίησή του με την συνάρτηση `pwm_start_channels` . Αυτή η συνάρτηση ουσιαστικά γράφει bits στον καταχωρητή ENA.
Ύστερα γίνεται ο προγραμματισμός των δύο διακοπών (μία για κάθε πιεστικό διακόπτη) όπως και στο προηγούμενο παράδειγμα και η συσχέτιση με τους χειριστές τους. Ένας έλεγχος στον πίνακα των ακροδεκτών (πίνακας 12.9 αναφορά [1]) και στον πίνακα δρομολόγησης διακοπών (πίνακας 12.3 αναφορά [1]) θα δείξει ότι η διακοπή που οδεύει από τον PB1 (`AVR32_GPIO_IRQ_10`) είναι διαφορετική από αυτήν του PB0. Για αυτό το λόγο είναι καλύτερα να χρησιμοποιηθούν δύο ξεχωριστές ρουτίνες χειρισμού (`PB0_int_handler` και `PB1_int_handler`). Και οι δύο ελέγχουν ότι είναι πατημένος ο αντίστοιχος διακόπτης (γιατί η διακοπή μπορεί να προέλθει και από απελευθέρωση προκειμένου να σβήσει το LED2 ή LED4) και ανάλογα αυξομειώνουν τον καταχωρητή CUPD όπως είναι η σωστή μέθοδος αλλαγής παραμέτρου. Το πεδίο CPD του καταχωρητή CMR είχε καθορίσει η μεταβολή που θα προκαλεί ο CUPD να αφορά τον καταχωρητή διάρκεια παλμού CDTY και όχι τον καταχωρητή περιόδου CPRD. Και τέλος, αφού καθαριστεί το αίτιο της διακοπής (`gpio_clear_pin_interrupt_flag`), γίνεται επιστροφή από τον χειριστή διακοπών στο κυρίως πρόγραμμα.

III.5 Μετατροπέας αναλογικού σε ψηφιακό (ADC).

Ο μικροελεγκτής διαθέτει μετατροπέα αναλογικού σήματος σε ψηφιακό τύπου SAR (Successive Approximation Register) ακρίβειας 10 bits η οποία μπορεί να πέσει στα 8 bits για γρηγορότερη μεταφορά με κατάλληλο προγραμματισμό. Ο μετατροπέας δέχεται μέχρι 8 αναλογικές εισόδους από εξωτερικούς ακροδέκτες (PA21-PA28, Function B), τις οποίες επεξεργάζεται διαδοχικά με χρήση πολυπλεξίας.
Η μετατροπή γίνεται σε κύκλους μετατροπής πυροδοτούμενη (triggered) με τρεις διαφορετικούς τρόπους:
α) Λογισμικό, δηλαδή εγγραφή στο bit START του καταχωρητή CR της τιμής 1

β) Σήμα που προέρχεται από το κύκλωμα των μετρητών/χρονιστών
γ) Σήμα που προέρχεται από εξωτερικό ακροδέκτη (σήμα ADC TRIGGER στο PB18 – Function B).
Η επιλογή της προέλευσης του σήματος διέγερσης για τις περιπτώσεις β) και γ) γίνεται με προγραμματισμό των 3 bits του πεδίου TRGSEL του καταχωρητή MR. Στον ίδιο καταχωρητή γίνεται και η ενεργοποίηση της δυνατότητας διέργεσης με εσωτερικό ή εξωτερικό σήμα (πεδίο TRGEN). Ακόμα και αν είναι ενεργοποιημένη η δυνατότητα διέγερσης με σήμα, δεν αναιρείται η δυνατότητα για διέγερση από το λογισμικό.
Ο κάθε κύκλος μετατροπής αποτελείται από διαδοχικές μετατροπές των καναλιών ADC που είναι ενεργοποιημένα (καταχωρητές CHER, CHDR και CHSR). Μόλις τελειώσει ο κάθε κύκλος, ο ADC παραμένει ανενεργός μέχρι να δεχτεί νέο σήμα έναρξης. Μέσα σε κάθε κύκλο, μόλις ολοκληρώνεται η μετατροπή ενός καναλιού, η ψηφιακή τιμή του αποθηκεύεται σε έναν από τους καταχωρητές CDRx (ανάλογα με το κανάλι) και ταυτόχρονα στον καταχωρητή LCDR (Last Converted Data Register).
Ο καταχωρητής SR δίνει πληροφορίες για το αν υπάρχουν έτοιμα δεδομένα σε κάποιο κανάλι (bits EOC0-EOC7 και DRDY) καθώς και αν έγινε προσπάθεια νέας μετατροπής χωρίς να έχουν αποσυρθεί τα τελευταία δεδομένα που υπήρχαν (bits OVR0-OVR7 και GOVRE). Επίσης είναι δυνατόν να προκαλούνται διακοπές με βάση τα bits του SR (καταχωρητές IER,IDR,ISR). Δηλαδή μπορεί να παράγεται σήμα διακοπής όποτε τελειώνει η μετατροπή ενός καναλιού ή όποτε τελειώνει ένας ολόκληρος κύκλος.
Ο χρονισμός του ADC γίνεται από το κεντρικό ρολόι του μικροελεγκτή με τη δυνατότητα όμως να διαιρεθεί η συχνότητά του από με ένα συντελεστή μεταξύ 1 και 64 (πεδίο PRESCAL του καταχωρητή MR). Η μέγιστη συχνότητα ρολογιού για ανάλυση 10 bits είναι 5MHz ενω για 8bits ανεβαίνει στα 8MHz.
Το κύκλωμα του μετατροπέα, όπως όλα τα αναλογικά κυκλώματα σε ολοκληρωμένα μικτής (αναλογικής και ψηφιακής) σχεδίασης έχει ξεχωριστή τροφοδοσία για λόγους αποφυγής παρεμβολών και παρασίτων (ακροδέκτες VDDANA και GNDANA). Η τάση αναφοράς σε σχέση με τη γείωση που χρησιμοποιείται για την μετατροπή δίνεται στον ακροδέκτη ADVREF ο οποίος στην πλακέτα είναι σε δυναμικό 3.3V. Έτσι η περιοχή μετατροπής για κάθε κανάλι είναι από 0V ως 3.3V. Τιμή 0V σε ένα κανάλι θα μετατραπεί στην ψηφιακή τιμή 0000000000 και τιμή 3.3V θα μετατραπεί στην ψηφιακή τιμή 1111111111. Ενδιάμεσες τιμές τάσεων μετατρέπονται στις ενδιάμεσες ψηφιακές τιμές με γραμμικό τρόπο.
Το πρόγραμμα `gpio_int_pwm_adc.c` (χρειάζεται τους οδηγούς GPIO,INTC,PWM και ADC) που υπάρχει στο Παράρτημα IV.4 είναι μία εφαρμογή που χρησιμοποιεί 3 κανάλια του ADC για να πάρει αναλογικές μετρήσεις από τον αισθητήρα θερμοκρασίας, την φωτοαντίσταση και το ποτενσιόμετρο. Η λειτουργία του είναι ίδια με αυτήν του προγράμματος `gpio_int_pwm.c` με τη διαφορά ότι χρησιμοποιούνται τρία κανάλια PWM για να οδηγήσουν τρία ισάριθμα LED σε φωτεινότητα ανάλογη με την μέτρηση του κάθε καναλιού. Συγκεκριμένα το LED1 φωτοβολεί ανάλογα με τη θέση του ποτενσιόμετρου (πιο φωτεινό δεξιόστροφα), το κόκκινο LED5 φωτοβολεί ανάλογα με το φως που προσπίπτει στην φωτοαντίσταση LDR, και το κόκκινο LED6 φωτοβολεί αντιστρόφως ανάλογα της θερμοκρασίας που βρίσκεται το θερμίστορ. Στην τελευταία περίπτωση η αλλαγή φωτεινότητας δεν είναι εμφανής γιατί χρειάζονται μεγάλες διαφορές θερμοκρασίας για να γίνει αισθητή.
Ο ατέρμονος βρόχος αναβοσβήνει το LED3 και ταυτόχρονα εκκινεί και έναν νέο κύκλο μετατροπής από αναλογικό σε ψηφιακό. Έχουν ενεργοποιηθεί τρία κανάλια ADC (`adc->cher = 7;`) και έχει οριστεί ότι μόλις τελειώνει η μετατροπή και του τρίτου καναλιού να προκαλείται μία διακοπή (`adc->ier |= 1 << AVR32_ADC_IER_EOC2_OFFSET;`). Ο χειριστής διακοπής `ADC_int_handler` έχει οριστεί να καλείται από τη διακοπή που παράγεται από τη μονάδα ADC (`INTC_register_interrupt(&ADC_int_handler,AVR32_ADC_IRQ, AVR32_INTC_INT0);`). Κάθε φορά που καλείται γίνεται διαδοχικά ανάγνωση των τριών καναλιών με τη βοήθεια της συνάρτησης βιβλιοθήκης `adc_get_value` και αφού γίνει η

μετατροπή του αριθμού εύρους 0..1023 σε αριθμό εύρους 0..PWM_DUR (0..127) γίνεται αλλαγή της διάρκειας των PWM παλμών με τη χρήση του καταχωρητή CUPD. Ο χειριστής διακοπών δε χρειάζεται να καθαρίσει το bit που προκάλεσε τη διακοπή με τη χρήση της συνάρτησης `gpio_clear_pin_interrupt_flag` γιατί αυτό μηδενίζεται αυτόματα με την ανάγνωση των δεδομένων από τον ADC.

III.6 Η οθόνη χαρακτήρων LCD.

Η οθόνη χαρακτήρων υγρών κρυστάλλων είναι η DIP204-4 της εταιρείας Electronic Assembly Gmbh. Προσφέρεται από το εργοστάσιο σε δύο εκδόσεις, η μία μπορεί να προγραμματιστεί με παράλληλο τρόπο ανά 4 ή 8 bits τη φορά και η άλλη σε σειριακό τρόπο χρησιμοποιώντας το πρωτόκολλο SPI bus. Η έκδοση της πλακέτας είναι αυτή που δέχεται προγραμματισμό μέσω του SPI bus. Τα φυλλάδια με τα χαρακτηριστικά και τον τρόπο προγραμματισμού διατίθενται στο διαδίκτυο και βρίσκονται στις αναφορές [13] και [14]. Το δεύτερο φυλλάδιο αφορά τον ελεγκτή της Samsung που είναι ενσωματωμένος στο LCD, και είναι χρήσιμος για την περίπτωση του σειριακού προγραμματισμού. Σε κάθε περίπτωση ο προγραμματιστής λίγα πράγματα πρέπει να ξέρει σε σχέση με τον ακριβή τρόπο προγραμματισμού αφού οι βιβλιοθήκες της Atmel διευκολύνουν κατά πολύ τον προγραμματισμό.
Για να χτιστεί μία εφαρμογή που χρησιμοποιεί την οθόνη υγρών κρυστάλλων θα πρέπει να χρησιμοποιηθούν οι βιβλιοθήκες για το SPI από τους οδηγούς (drivers) και το Display (DIP204) από τα εξαρτήματα (components). Τα σχετικά αρχεία είναι τα `spi.h`, `spi.c`, `dip204.h` και `dip204.c`.
Ο δίαυλος SPI λειτουργεί στη βάση του αφέντη/σκλάβου (master/slave), δηλαδή μία συσκευή ή περιφερειακό αφέντη που ελέγχει το δίαυλο και τη διατησία του και έναν ή περισσότερους σκλάβους που επιλέγονται από τον αφέντη για την αμφίδρομη επικοινωνία. Χρησιμοποιεί τέσσερις γραμμές για τη λειτουργία του, μία αποστολής δεδομένων από τον αφέντη (MOSI-Master Output Slave Input), μία για λήψη δεδομένων από τον αφέντη (MISO-Master Input Slave Output), ένα σήμα ρολογιού (SPCK) και μία ή περισσότερες γραμμές επιλογής σκλάβων (NPCS0-3). Ο μικροελεγκτής διαθέτει δύο ελεγκτές SPI που μπορούν να χρησιμοποιηθούν ανεξάρτητα έχοντας τους δικούς τους καταχωρητές ελέγχου και δεδομένων. Λεπτομέρειες για τον τρόπο λειτουργίας και προγραμματισμού του SPI bus υπάρχουν στο κεφάλαιο 32 της αναφοράς [1] και στην αναφορά [15]. Εδώ δε θα αναφερθούν λεπτομέρειες για τη λειτουργία του SPI, παρά μόνο θα δειχθεί η λειτουργία της οθόνης μέσα από το παράδειγμα `gpio_int_pwm_adc_lcd.c` που βρίσκεται στο Παράρτημα IV.5.
Το πρόγραμμα χρησιμοποιεί διαφορετική συχνότητα λειτουργίας η οποία καθορίζεται από τον ταλαντωτή 0 που χρησιμοποιεί κρύσταλλο στα 12MHz. Για την αλλαγή του βασικού ρολογιού χρειάζεται η επέμβαση στον Διαχειριστή Ισχύος με τη βοήθεια των σχετικών οδηγών (συνάρτηση `pm_switch_to_osc0`).
Οι απαραίτητες αρχικοποιήσεις για το SPI είναι πανομοιότυπες κάθε φορά που χρειάζεται να χρησιμοποιηθεί η οθόνη υγρών κρυστάλλων. Το πρόγραμμα αυτό είναι επέκταση του προηγούμενου (`gpio_int_pwm_adc.c`), με τη διαφορά ότι έχει προστεθεί και αποστολή των αποτελεσμάτων ανάγνωσης από τα τρία αναλογικά κανάλια προς την οθόνη.
Για να χρησιμοποιηθεί η οθόνη πρέπει πρώτα να γίνουν οι απαραίτητες ρυθμίσεις για το SPI. Αυτές αποτελούνται από τις εξής γραμμές:

```
static const gpio_map_t DIP204_SPI_GPIO_MAP =
{ {DIP204_SPI_SCK_PIN,  DIP204_SPI_SCK_FUNCTION },
  {DIP204_SPI_MISO_PIN, DIP204_SPI_MISO_FUNCTION},
  {DIP204_SPI_MOSI_PIN, DIP204_SPI_MOSI_FUNCTION},
  {DIP204_SPI_NPCS_PIN, DIP204_SPI_NPCS_FUNCTION} };
spi_options_t spiOptions =
 { .reg          = DIP204_SPI_NPCS,.baudrate     = 1000000,
```

```
   .bits         = 8,.spck_delay   = 0,.trans_delay  = 0,
   .stay_act     = 1,.spi_mode     = 0,.fdiv         = 0,
   .modfdis      = 1};
gpio_enable_module(DIP204_SPI_GPIO_MAP,sizeof(DIP204_SPI_GPIO_MAP) /
sizeof(DIP204_SPI_GPIO_MAP[0]));
spi_initMaster(DIP204_SPI, &spiOptions);
spi_selectionMode(DIP204_SPI, 0, 0, 0);
spi_enable(DIP204_SPI);
spi_setupChipReg(DIP204_SPI, &spiOptions, FOSC0);
```

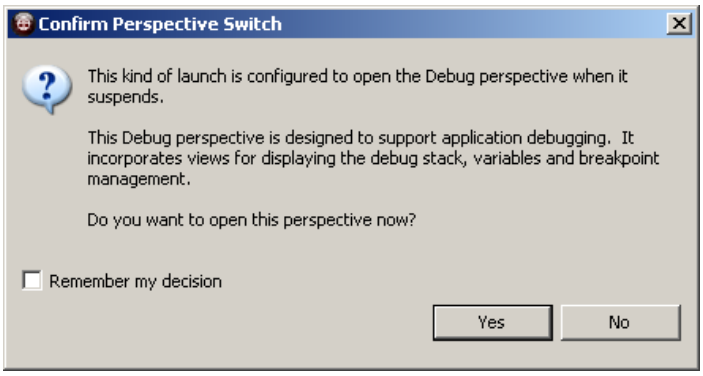
Ο ορισμός της σταθεράς `DIP204_SPI_GPIO_MAP` σε συνδυασμό με τη συνάρτηση `gpio_enable_module` κάνει τη σύνδεση μεταξύ του εσωτερικού SPI περιφερειακού και των ακροδεκτών στους οποίους είναι συνδεδεμένη η οθόνη. Ακολουθεί η αρχικοποίηση των παραμέτρων του SPI. Ακολουθούν οι συναρτήσεις `spi_initMaster` και `spi_selectionMode` που μαζί με τη δομή `spiOptions` ουσιαστικά αρχικοποιούν τον καταχωρητή MR του SPI περιφερειακού ρυθμίζοντας θέματα χρονισμού. Η συνάρτηση `spi_enable` εκκινεί τη λειτουργία του SPI και τέλος η `spi_setupChipReg` επηρεάζει τον καταχωρητή CSR2, αφού το LCD όπως είναι συνδεδεμένο επιλέγεται με το σήμα NPCS2 (σταθερά `DIP204_SPI_NPCS`).
Τώρα πλέον είναι δυνατή η αποστολή εντολών και δεδομένων στην οθόνη μέσω του διαύλου SPI. Πρώτα όμως πρέπει να αρχικοποιηθεί η οθόνη με τη συνάρτηση `dip204_init`. Αυτή η συνάρτηση καθορίζει διάφορες παραμέτρους της οθόνης, όπως πόσες γραμμές θα χρησιμοποιούνται, αν θα γίνεται αντιστροφή χρώματος χαρακτήρων, αν θα υπάρχει δρομέας και πολλά άλλα. Αν ο χρήστης επιθυμεί διαφορετικό προγραμματισμό θα πρέπει να δημιουργήσει δική του συνάρτηση αρχικοποίησης, αφού η συγκεκριμένη δεν είναι παραμετροποιήσιμη. Η μόνη παράμετρος που μπορεί να εισαχθεί είναι αυτή που καθορίζει αν ο φωτισμός της οθόνης θα ρυθμίζεται από PWM κανάλι (σταθερά `backlight_PWM` στην πρώτη παράμετρο) ή θα είναι σταθερός (`backlight_IO` ρυθμιζόμενος μόνο από το ποτενσιόμετρο που βρίσκεται κάτω αριστερά από την οθόνη). Η δεύτερη παράμετρος TRUE (ή FALSE στην αντίθετη περίπτωση) καθορίζει αν θα υπάρχει φωτισμός από πίσω στην περίπτωση της επιλογής σταθερού φωτισμού. Στη συγκεκριμένη οθόνη δε είναι ευανάγνωστοι οι χαρακτήρες αν δεν υπάρχει φωτισμός, επομένως η επιλογή θα πρέπει πάντα να είναι TRUE).
Η συνάρτηση `dip204_init` (όπως και η `dip204_clear_display`) απαιτεί τη χρήση καθυστέρησης η οποία υλοποιείται με την `delay_ms`. Η `delay_ms` όμως δεν είναι μέρος της βιβλιοθήκης που προσφέρει η Atmel στο αρχείο `dip204.c`. Η `delay_ms` ορίζεται στο αρχείο παραδείγματος που προσφέρει η Atmel με όνομα `dip204_example.c`. Εκεί η συνάρτηση αυτή υλοποιείται με τη χρήση των καταχωρητών συστήματος COUNT και COMPARE (Ενότητα 2.11 αναφοράς [16]) και με τη χρήση συναρτήσεων από τους οδηγούς που βρίσκονται στα αρχεία `cycle_counter.h` και `cycle_counter.c`. Αυτός ο τρόπος υλοποίησης καθυστέρησης απαιτεί τη χρήση διακοπής προσθέτοντας πολυπλοκότητα, όμως παρέχει το πλεονέκτημα της ακρίβειας. Εναλλακτικοί τρόποι πραγματοποίησης καθυστέρησης θα ήταν με το περιφερειακό Χρονιστών/Μετρητών ή με τη χρήση ενός απλού βρόχου. Στο παράδειγμα που παρουσιάζεται χρησιμοποιείται η χρήση των καταχωρητών συστήματος COUNT και COMPARE. Μέρος της λειτουργίας καθυστέρησης, εκτός από τον ορισμό της συνάρτησης `delay_ms`, είναι και ο ορισμός του χειριστή διακοπής (`compare_irq_handle`) που θα προκληθεί όταν οι δύο καταχωρητές έχουν ίσες τιμές καθώς και η συσχέτιση της διακοπής που θα προκληθεί με τον χειριστή (`AVR32_CORE_COMPARE_IRQ`). Είναι σημαντικό ο μεταγλωττιστής να μην κάνει καμμία βελτιστοποίηση (`-O0`) γιατί αλλιώς θα δημιουργηθεί ατέρμονος βρόχος στην `delay_ms` και συγκεκριμένα στο σημείο `while (!TimeOut);`.
Έχοντας αρχικοποιήσει την οθόνη, αυτή είναι έτοιμη τώρα να δεχθεί εντολές εγγραφής συμβολοσειρών με τις συναρτήσεις που καθορίζουν τη θέση που θα γίνει η εγγραφή

(dip204_set_cursor_position) και τη συμβολοσειρά που θα γραφεί (dip204_write_string). Αυτές είναι οι βασικές συναρτήσεις που χρησιμοποιούνται για την εγγραφή των σταθερών μηνυμάτων καθώς και την εγγραφή των ακεραίων τιμών που διαβάζονται μέσα στο χειριστή διακοπών από τον ADC.

III.7 Διαδικασία εκσφαλμάτωσης (debugging).

Ένα από τα βασικά μέρη της ανάπτυξης ενός προγράμματος είναι και η εκσφαλμάτωση, δηλαδή η ανεύρεση λογικών λαθών μέσα στο πρόγραμμα που έχουν σαν αποτέλεσμα την διαφορετική συμπεριφορά από την αρχικά σχεδιασμένη. Η ανεύρεση των σφαλμάτων γίνεται με τη βοήθεια ειδικού λογισμικού που λέγεται εκσφαλματωτής (debugger) και αν πρόκειται για πρόγραμμα που προορίζεται για κάποια ειδική συσκευή όπως ο μικροελεγκτής τότε, χρειάζεται η σύνδεση σε πραγματικό χρόνο της συσκευής με το περιβάλλον εκσφαλμάτωσης. Μία από τις συσκευές που διατίθενται για εκσφαλμάτωση είναι η JTAGICE mkII που έχει ήδη αναφερθεί στο μέρος που αναφέρεται για τον προγραμματισμό των μικροελεγκτών. Η βασική ομως λειτουργία της συσκευής δεν είναι ο προγραμματισμός, αλλά η εκσφαλμάτωση.
Η συσκευή JTAGICE mkII συνδέεται με το μικροελεγκτή προς έλεγχο μέσω της JTAG επαφής και με τον σταθμό εργασίας με USB ή RS232 καλώδιο, με τον ίδιο τρόπο που έχει παρουσιαστεί η χρήση της ως προγραμματιστή (Εικόνα 9). Η λειτουργία της είναι να συνεργάζεται με τον μικροελεγκτή προς εκσφαλμάτωση και να τον θέτει σε ειδική κατάσταση (debug state). Όποτε ο μικροελεγκτής τίθεται σε μία τέτοια κατάσταση η συσκευή μπορεί να ελέγξει όλους τους καταχωρητές του (γενικούς, συστήματος και περιφερειακών), να διαβάσει και να αλλάξει τα περιεχόμενά τους, να διαβάσει οποιαδήποτε θέση μνήμης και να την αλλάξει και γενικά να έχει τον πλήρη έλεγχο του μικροελεγκτή.
Σε μία διαδικασία εκσφαλμάτωσης υπάρχουν διάφορες μέθοδοι που μπορούν να ακολουθηθούν. Εδώ θα περιγραφούν δύο από τις βασικές που είναι η βηματική εκτέλεση του προγράμματος (step) και η εκτέλεση με καθορισμένα σημεία ελέγχου και διακοπής (breakpoints). Και στις δύο περιπτώσεις το πρόγραμμα «παγώνει» σε κάποια εντολή και ο χρήστης μπορεί να παρατηρήσει διάφορες μεταβλητές ή καταχωρητές, να αλλάξει τις τιμές τους και να συνεχίσει. Τα σημεία διακοπής καθορίζονται στον πηγαίο κώδικα του προγράμματος. Όταν η εκτέλεση φτάσει σε εκείνο το σημείο η εκτέλεση διακόπτεται προσωρινά και αφού γίνουν οι απαραίτητοι έλεγχοι τότε, μπορεί να επαναληφθεί (resume) η εκτέλεση μέχρι το επόμενο σημείο διακοπής είτε να γίνει εκτέλεση με βηματικό τρόπο, εντολή προς εντολή. Ο βηματικός τρόπος έχει διάφορες παραλλαγές, όπως το να θεωρηθεί μία συνάρτηση σαν μία ενιαία εντολή και να εκτελεστεί σε ένα βήμα, ή να εκτελεστούν οι εσωτερικές εντολές η κάθε μία ξεχωριστά σε διαφορετικά βήματα.
Η διαδικασία περιγράφεται σύντομα θεωρώντας ότι πρόκειται να γίνει έλεγχος του τελευταίου προγράμματος που περιγράφηκε στο προηγούμενο τμήμα, δηλαδή του gpio_int_pwm_adc_lcd.c.
Για να ξεκινήσει η εκσφαλμάτωση πρέπει να γίνει μεταγλώττιση του όλου έργου σε μορφή κατάλληλη για εκσφαλμάτωση επιλέγοντας Build Configurations → Set Active → Debug και κατόπιν κάνοντας την μεταγλώττιση με Build. Θα έχει δημιουργηθεί ένα νέο δυαδικό αρχείο τύπου .elf. Αυτός ο εκτελέσιμος δυαδικός κώδικας θα πρέπει να περάσει στον μικροελεγκτή με τη χρήση του JTAGICE mkII ως προγραμματιστή. Πριν από αυτό θα πρέπει να διαγραφεί πλήρως ολόκληρος ο μικροελεγκτής με δεξί κλικ πάνω στον JTAG προγραμματιστή και επιλογή του Chip Erase. Αυτή η επιλογή έχει ως αποτέλεσμα τη διαγραφή του bootloader, επομένως σε επόμενη φάση και αν είναι επιθυμητό θα πρέπει να γίνει ο επαναπρογραμματισμός του όπως έχει περιγραφεί στο II.3. Φυσικά στην επιλογή για τον προγραμματισμό του μικροελεγκτή θα πρέπει να δειχθεί το κατάλληλο .elf αρχείο που θα βρεθεί κάτω από τον κατάλογο Debug (και όχι Release) της ιεραρχίας του έργου. Η ίδια διαδικασία

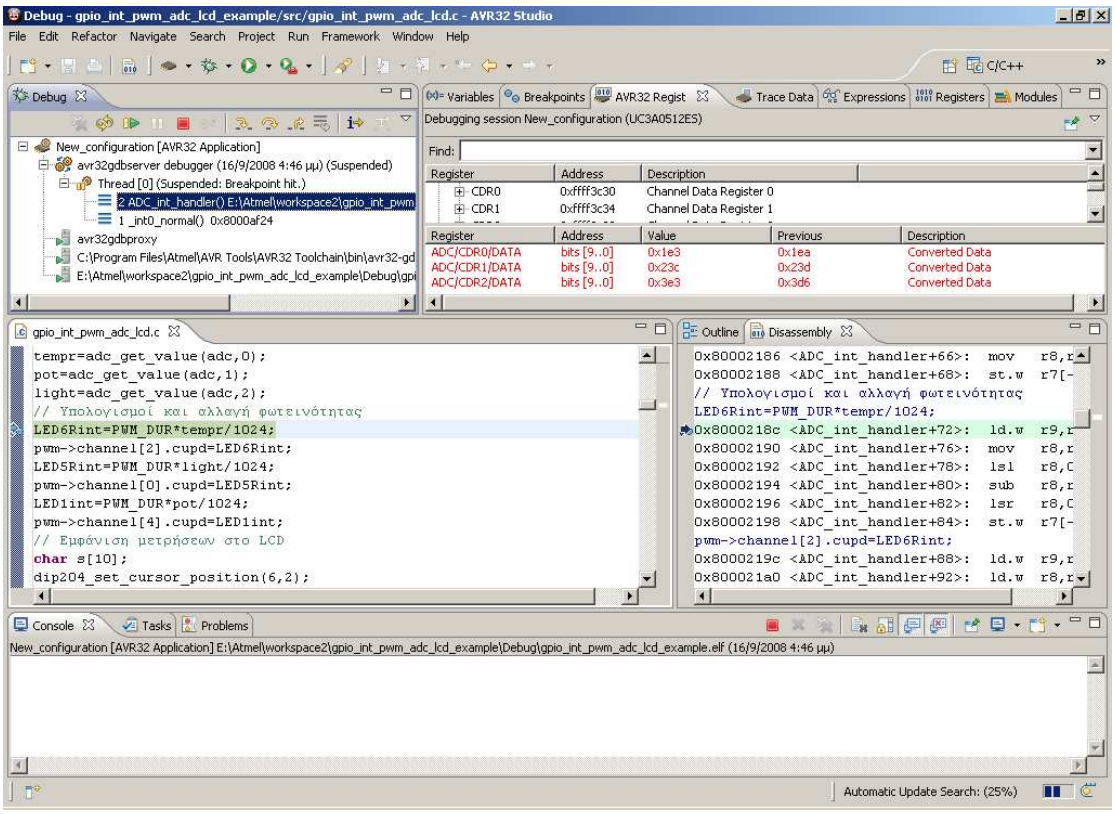
προγραμματισμού μπορεί να γίνει και επιλέγοντας Run → Open Debug Dialog όπου θα ανοίξει μία κάρτα με πολλές επιλογές μεταξύ των οποίων και το αρχείο προς προγραμματισμό.



Εικόνα 21 : Παράθυρο επιβεβαίωσης αλλαγής προοπτικής

Όταν τελειώσει η αποστολή του αρχείου προς τον μικροελεγκτή μία νέα κάρτα απευθύνει ερώτηση για αλλαγή της προοπτικής των παραθύρων (Confirm perspective switch) η οποία φαίνεται στην Εικόνα 21. Το AVR 32 Studio έχει διάφορες «προοπτικές» ανάλογα με τη χρήση που πρόκειται να γίνει και η κάθε προοπτική παρουσιάζει τα καταλληλότερα παράθυρα για κάθε χρήση. Εδώ η προοπτική θα πρέπει να αλλάξει και να γίνει Debug. Η προοπτική των παραθύρων αλλάζει οποτεδήποτε είναι επιθυμητό με την επιλογή Window → Open Perspective → ...

Η προοπτική Debug φαίνεται στην Εικόνα 22 και το βασικό παράθυρο ελέγχου τιμών είναι το πάνω δεξιά με τις διάφορες στήλες (tabs). Μεταξύ άλλων υπάρχει η στήλη των μεταβλητών που μπορούν να παρατηρηθούν (Variables), η στήλη των καταχωρητών συστήματος και περιφερειακών (AVR32 Registers), η στήλη των γενικών καταχωρητών (Registers) κ.α.



Εικόνα 22 : Προοπτική εκσφαλμάτωσης του AVR 32 Studio

Στο πρόγραμμα μπορούν να μπούν διάφορα σημεία διακοπής (breakpoints) επιλέγοντας την γραμμή στην οποία θα πρέπει να γίνει η διακοπή (το πρόγραμμα θα εκτελεστεί μία εντολή ακριβώς πριν το σημείο διακοπής) και επιλέγοντας Run → Toggle Breakpoint. Το σημείο διακοπής θα προστεθεί στη στήλη Breakpoints. Η αναίρεσή του σημείου διακοπής γίνεται με τον ίδιο ακριβώς τρόπο (toggle).
Σαν υπόθεση εργασίας θεωρούμε ότι για κάποιο λόγο οι μετρήσεις που φαίνονται στην οθόνη δεν είναι οι αναμενόμενες. Για αυτό το λόγο θα μπεί ένα σημείο διακοπής στη γραμμή του κώδικα που βρίσκεται στη συνάρτηση `ADC_int_handler` αμέσως μετά την ανάγνωση και των τριών καναλιών του ADC, δηλαδή στη γραμμή `LED6Rint=PWM_DUR*tempr/1024;`.
Επίσης θα γίνει έλεγχος του περιεχομένου των καταχωρητών του ADC που αποθηκεύουν τις μετρήσεις, δηλαδή των CDR0, CDR1 και CDR2. Η επιλογή τους γίνεται από τη στήλη AVR32 Registers και μάλιστα υπάρχει η δυνατότητα επιλογής μόνο των 10 bits από 32 (πεδίο DATA).
Το πρόγραμμα ξεκινάει και σταματάει στο σημείο διακοπής. Ο προγραμματιστής ελέγχει της μετρήσεις, και αφού τις αλλάξει εξωτερικά (π.χ. μεταβάλλοντας το ποτενσιόμετρο) επιλέγει την συνέχιση της ροής (Run → Resume ή από το σχετικό κουμπί στη γραμμή εργαλείων) και ξαναελέγχει τη μέτρηση.
Σε περίπτωση όπου όλα είναι εντάξει, μπορεί να διαπιστώσει ότι το πρόβλημα βρίσκεται στη μεταφορά της ανάγνωσης προς την οθόνη. Η διαδικασία μπορεί να προχωρήσει βάζοντας μία μεταβλητή σε θέση παρατήρησης προκειμένου να διαπιστωθεί ότι κρατάει την κατάλληλη τιμή. Π.χ. με την επιλογή της μεταβλητής `pot` που βρίσκεται στη γραμμή `pot=adc_get_value(adc,1);` και την προσθήκη της με Add Watch Expression… στη στήλη παρατήρησης των εκφράσεων (Expressions).
Με αυτό τον τρόπο είναι δυνατόν διαδοχικά να ελεχθεί η πορεία επεξεργασίας των δεδομένων μέσα στο πρόγραμμα και να εντοπιστεί το προβληματικό σημείο.
Η συνέχεια σε ένα σταματημένο πρόγραμμα μπορεί να δωθεί είτε με την εντολή Resume όπως έχει αναφερθεί, είτε με βηματική εκτέλεση με τους τρεις διαφορετικούς τύπους (Step Into, Step και Step Return) είτε με εκτέλεση μέχρι μία συγκεκριμένη γραμμή (Run to Line).
Πολλές από τις δυνατότητες εκσφαλμάτωσης παρουσιάζονται στην ενσωματωμένη βοήθεια (On Line Help) που διαθέτει το περιβάλλον προγραμματισμού.

IV Παράρτημα.

Στο Παράρτημα παρατίθενται πηγαίοι κώδικες σε γλώσσα C των παραδειγμάτων που αναπτύσσονται στο κεφάλαιο III. Για να μεταγλωττιστούν θα πρέπει να μπουν στο `src` φάκελο ενός έργου του AVR32 Studio που έχει αρχικοποιηθεί με τις κατάλληλες βιβλιοθήκες που χρειάζονται για την κάθε περίπτωση. Όλοι οι κώδικες βρίσκονται και στο CD που συνοδεύει αυτό τον οδηγό κάτω από τον κατάλογο `Development`.
Σε κάθε παράδειγμα αναφέρονται και οι οδηγοί που θα πρέπει να συμπεριληφθούν προκειμένου να γίνει σωστή η μεταγλώττιση (επιλογή Framework→Select Drivers/Components/Services στο AVR 32 Studio).

IV.1 Κώδικας παραδείγματος χρήσης GPIO.

Απαραίτητοι οδηγοί για να τρέξει το παράδειγμα στο AVR 32 Studio : **GPIO.**

```
#include "compiler.h"
#include "gpio.h"
#include "board.h"
#define GPIO_PIN_EXAMPLE_1  LED2_GPIO
#define GPIO_PIN_EXAMPLE_2  LED3_GPIO
#define GPIO_PIN_EXAMPLE_3  GPIO_PUSH_BUTTON_0

/*! \brief This is an example of how to access the gpio.c driver to
set, clear, toggle... the pin GPIO_PIN_EXAMPLE. */
int main(void)
{
  U32 state = 0;
  U32 i;

  gpio_enable_pin_glitch_filter(GPIO_PIN_EXAMPLE_3);
  while (1)
  {
    switch (state)
    {
    case 0:
      // Access with GPIO driver gpio.c with clear and set access.
      gpio_clr_gpio_pin(GPIO_PIN_EXAMPLE_1);
      state++;
      break;

    case 1:
      gpio_set_gpio_pin(GPIO_PIN_EXAMPLE_1);
      state++;
      break;

    case 2:
      // Note that it is also possible to use the GPIO toggle feature.
      gpio_tgl_gpio_pin(GPIO_PIN_EXAMPLE_1);
      state++;
      break;

    case 3:
    default:
      gpio_tgl_gpio_pin(GPIO_PIN_EXAMPLE_1);
      state = 0;
      break;
    }

    // Poll push button value.
    for (i = 0; i < 1000; i += 4)
    {
      if (gpio_get_pin_value(GPIO_PIN_EXAMPLE_3) == 0)
        gpio_clr_gpio_pin(GPIO_PIN_EXAMPLE_2);
      else
        gpio_set_gpio_pin(GPIO_PIN_EXAMPLE_2);
    }
  }
}
```

IV.2 Κώδικας παραδείγματος χρήσης διακοπών.

Απαραίτητοι οδηγοί για να τρέξει το παράδειγμα στο AVR 32 Studio : **GPIO**, **INTC.**

```
/*****************************************************
 Name          : gpio_int.c
 Author        : adp@unipi.gr
 Copyright     :
 Description   : ---LED3 αναβοσβήνει μόνιμα,
                 ---PB0 όσο είναι πατημένος
                    ανάβει το LED4
 *****************************************************/
#include "compiler.h"
#include "intc.h"
#include "gpio.h"
#include "board.h"
#if __GNUC__
      __attribute__((__interrupt__))
#elif __ICCAVR32__
__interrupt
#endif

static void PB0_int_handler(void)
      {
            if (gpio_get_pin_value(GPIO_PUSH_BUTTON_0) == 1)
                  gpio_set_gpio_pin(LED3_GPIO);
            else
                  gpio_clr_gpio_pin(LED3_GPIO);
            gpio_clear_pin_interrupt_flag(GPIO_PUSH_BUTTON_0);
      }

// Έναρξη Προγράμματος
int main(void)
      {
      U32 i,r;
      // Απενεργοποίηση όλων των διακοπών
      Disable_global_interrupt();
      // Αρχικοποίηση διακοπών (οδηγούνται σε "κενό χειριστή"
      INTC_init_interrupts();
      // Ενεργοποίηση φίλτρου παρασίτων για το PB0
      gpio_enable_pin_glitch_filter(GPIO_PUSH_BUTTON_0);
      // Ενεργοποίηση παραγωγής διακοπών για το PB0
      gpio_enable_pin_interrupt (GPIO_PUSH_BUTTON_0,GPIO_PIN_CHANGE);
      // Συσχέτιση ρουτίνας χειρισμού με διακοπή
      INTC_register_interrupt(&PB0_int_handler,AVR32_GPIO_IRQ_11,
AVR32_INTC_INT0);
      // Ενεργοποίηση όλων των διακοπών
      Enable_global_interrupt();
      while (1) // Ατέρμονος βρόχος
      {
      // Τεχνητή καθυστέρηση
    for (i = 0; i < 1000; i += 1)
      r=rand();
    // Αλλαγή κατάστασης (toggle) LED2
      gpio_tgl_gpio_pin(LED2_GPIO);
  }
}
```

IV.3 Κώδικας παραδείγματος διαμόρφωσης εύρους παλμών (PWM).

Απαραίτητοι οδηγοί για να τρέξει το παράδειγμα στο AVR 32 Studio : **GPIO**, **INTC,PWM.**

```
/*****************************************************
 Name          : gpio_int_pwm.c
 Author        : adp@unipi.gr
 Description   : ---LED3 αναβοσβήνει μόνιμα,
                 ---PB0 ανάβει LED2 και
                    μειώνει την ένταση του κόκκινου LED6,
                 ---PB1 ανάβει LED4 και
                    αυξάνει την ένταση του κόκκινου LED6.
 *****************************************************/
#include "compiler.h"
#include "intc.h"
#include "gpio.h"
#include "pwm.h"
#include "board.h"

unsigned int channel_id;

// Ρουτίνα διακοπής για τον διακόπτη PB0
// (μειώνει την φωτεινότητα του πράσινου LED6)
#if __GNUC__
      __attribute__((__interrupt__))
#elif __ICCAVR32__
__interrupt
#endif

static void PB0_int_handler(void)
{
volatile avr32_pwm_t *pwm = &AVR32_PWM;
extern unsigned int channel_id;
int intensity;

if (gpio_get_pin_value(GPIO_PUSH_BUTTON_0) == 1)
      gpio_set_gpio_pin(LED1_GPIO);
else
      {
      gpio_clr_gpio_pin(LED1_GPIO);
      intensity=pwm->channel[channel_id].cdty;
      if (intensity>0)
            intensity--;
      pwm->channel[channel_id].cupd=intensity;
      }
gpio_clear_pin_interrupt_flag(GPIO_PUSH_BUTTON_0);
}

// Ρουτίνα διακοπής για τον διακόπτη PB1
// (αυξάνει την φωτεινότητα του πράσινου LED6)
#if __GNUC__
      __attribute__((__interrupt__))
#elif __ICCAVR32__
__interrupt
#endif

static void PB1_int_handler(void)
```

```
{
volatile avr32_pwm_t *pwm = &AVR32_PWM;
extern unsigned int channel_id;
int intensity;

if (gpio_get_pin_value(GPIO_PUSH_BUTTON_1) == 1)
    gpio_set_gpio_pin(LED3_GPIO);
else
    {
    gpio_clr_gpio_pin(LED3_GPIO);
    intensity=pwm->channel[channel_id].cdty;
    if (intensity<pwm->channel[channel_id].cprd-1)
      intensity++;
    pwm->channel[channel_id].cupd=intensity;
    }

gpio_clear_pin_interrupt_flag(GPIO_PUSH_BUTTON_1);
}

// Έναρξη Προγράμματος
int main(void)
{
U32 i,r;
pwm_opt_t pwm_opt;
avr32_pwm_channel_t pwm_channel;
extern unsigned int channel_id;


Disable_global_interrupt();
INTC_init_interrupts();

// Σύνδεση καναλιού 2 του PWM με το GPIO pin 53 (PB21-LED6 κόκκινο)
gpio_enable_module_pin(AVR32_PWM_2_PIN, AVR32_PWM_2_FUNCTION);

// Αρχικοποίηση PWM MR
//Ρολόι του PWM είναι το κύριο ρολόι χωρίς διαίρεση
pwm_opt.diva = AVR32_PWM_DIVA_CLK_OFF;
pwm_opt.divb = AVR32_PWM_DIVB_CLK_OFF;
pwm_opt.prea = AVR32_PWM_PREA_MCK;
pwm_opt.preb = AVR32_PWM_PREB_MCK;
pwm_init(&pwm_opt);

// Αρχικοποίηση καναλιού PWM 2
channel_id = 2;
pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED;       // Αριστερή
ευθυγράμμιση
pwm_channel.CMR.cpol = PWM_POLARITY_LOW;            // Κανονική
πολικότητα
pwm_channel.CMR.cpd = PWM_UPDATE_DUTY;              // Ενημέρωση duty
cycle
pwm_channel.CMR.cpre = AVR32_PWM_CPRE_MCK_DIV_256;  // Επιλογή
συχνότητας 115200/256 Hz
pwm_channel.cdty = 19;  // duty cycle = 1-19/20
pwm_channel.cprd = 20;  // Περίοδος 20*256/115200 ~= 44msecs
pwm_channel.cupd = 0;   // Channel update is not used here.
pwm_channel_init(channel_id, &pwm_channel); // Set channel
configuration to channel 0.
pwm_start_channels(1 << channel_id);  // Start channel 3.
```

```
// Ρύθμιση διακοπών για PB0 και PB1
INTC_register_interrupt(&PB0_int_handler,AVR32_GPIO_IRQ_11,
AVR32_INTC_INT0);
INTC_register_interrupt(&PB1_int_handler,AVR32_GPIO_IRQ_10,
AVR32_INTC_INT0);
gpio_enable_pin_glitch_filter(GPIO_PUSH_BUTTON_0);
gpio_enable_pin_interrupt(GPIO_PUSH_BUTTON_0,GPIO_PIN_CHANGE);
gpio_enable_pin_glitch_filter(GPIO_PUSH_BUTTON_1);
gpio_enable_pin_interrupt(GPIO_PUSH_BUTTON_1,GPIO_PIN_CHANGE);
Enable_global_interrupt();

while (1)
      {
    for (i = 0; i < 1000; i += 1)
      r=rand();
      gpio_tgl_gpio_pin(LED2_GPIO);
      }
}
```

IV.4 Κώδικας παραδείγματος χρήσης ADC.

Απαραίτητοι οδηγοί για να τρέξει το παράδειγμα στο AVR 32 Studio : **GPIO**, **INTC,PWM,ADC.**

```
/*****************************************************
 Name          : gpio_int_pwm_adc.c
 Author        : adp@unipi.gr
 Copyright     :
 Description   : ---LED3 αναβοσβήνει μόνιμα,
                 ---LED1 φωτεινότητα εξαρτώμενη από ποτενσιόμετρο,
                 ---LED5κόκκινο φωτεινότητα εξαρτώμενη από το φως,
                 ---LED6κόκκινο φωτεινότητα εξαρτώμενη
                    από τη θερμοκρασία
 *****************************************************/

#include "compiler.h"
#include "intc.h"
#include "gpio.h"
#include "pwm.h"
#include "adc.h"
#include "board.h"

#define PWM_PER 128
#define PWM_DUR 127
unsigned int channel_id;


// Ρουτίνα διακοπής για τον χειρισμό των δεδομένων του ADC
// και την αλλαγή φωτεινότητας των LED1,LED5κόκκινο και LED6κόκκινο
#if __GNUC__
      __attribute__((__interrupt__))
#elif __ICCAVR32__
__interrupt
#endif

static void ADC_int_handler(void)
{
volatile avr32_pwm_t *pwm = &AVR32_PWM;
volatile avr32_adc_t *adc = &AVR32_ADC;
int LED6Rint,LED5Rint,LED1int;
int tempr,light,pot;

// Ανάγνωση ADC
tempr=adc_get_value(adc,0);
pot=adc_get_value(adc,1);
light=adc_get_value(adc,2);
// Υπολογισμοί και αλλαγή φωτεινότητας
LED6Rint=PWM_DUR*tempr/1024;
pwm->channel[2].cupd=LED6Rint;
LED5Rint=PWM_DUR*light/1024;
pwm->channel[0].cupd=LED5Rint;
LED1int=PWM_DUR*pot/1024;
pwm->channel[4].cupd=LED1int;
}
```

```
// Έναρξη Προγράμματος
int main(void)
{
pwm_opt_t pwm_opt;
avr32_pwm_channel_t pwm_channel;
extern unsigned int channel_id;

Disable_global_interrupt();
INTC_init_interrupts();

// Σύνδεση καναλιού 2 του PWM με το GPIO pin 53 (PB21-LED6 κόκκινο)
gpio_enable_module_pin(AVR32_PWM_2_PIN, AVR32_PWM_2_FUNCTION);
// Σύνδεση καναλιού 0 του PWM με το GPIO pin 51 (PB19-LED5 κόκκινο)
gpio_enable_module_pin(AVR32_PWM_0_PIN, AVR32_PWM_0_FUNCTION);
// Σύνδεση καναλιού 4 του PWM με το GPIO pin 59 (PB27-LED1)
gpio_enable_module_pin(AVR32_PWM_4_1_PIN, AVR32_PWM_4_1_FUNCTION);

// Σύνδεση καναλιού 0 του ADC με το GPIO pin 21 (PA21-θερμιστορ)
gpio_enable_module_pin(AVR32_ADC_AD_0_PIN, AVR32_ADC_AD_0_FUNCTION);
// Σύνδεση καναλιού 1 του ADC με το GPIO pin 22 (PA22-ποτενσιόμετρο)
gpio_enable_module_pin(AVR32_ADC_AD_1_PIN, AVR32_ADC_AD_1_FUNCTION);
// Σύνδεση καναλιού 2 του ADC με το GPIO pin 23 (PA23-φωτοαντίσταση)
gpio_enable_module_pin(AVR32_ADC_AD_2_PIN, AVR32_ADC_AD_2_FUNCTION);

// Αρχικοποίηση PWM MR
//Ρολόι του PWM είναι το κύριο ρολόι χωρίς διαίρεση
pwm_opt.diva = AVR32_PWM_DIVA_CLK_OFF;
pwm_opt.divb = AVR32_PWM_DIVB_CLK_OFF;
pwm_opt.prea = AVR32_PWM_PREA_MCK;
pwm_opt.preb = AVR32_PWM_PREB_MCK;
pwm_init(&pwm_opt);

// Αρχικοποίηση καναλιού PWM 2 (κόκκινο LED6 - θερμοκρασία)
channel_id = 2;
pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED;       // Αριστερή
ευθυγράμμιση
pwm_channel.CMR.cpol = PWM_POLARITY_LOW;            // Κανονική
πολικότητα
pwm_channel.CMR.cpd = PWM_UPDATE_DUTY;              // Ενημέρωση duty
cycle
pwm_channel.CMR.cpre = AVR32_PWM_CPRE_MCK_DIV_4;    // Επιλογή
συχνότητας 115200/4 Hz
pwm_channel.cdty = PWM_DUR;
pwm_channel.cprd = PWM_PER;
pwm_channel.cupd = 0;   // Ο UPD θα ενημερώνει τον CDTY
pwm_channel_init(channel_id, &pwm_channel); // Αρχικοποίηση καναλιού 2
pwm_start_channels(1 << channel_id);  // Εκκίνηση καναλιού 2

// Αρχικοποίηση καναλιού PWM 0 (κόκκινο LED5 - φωτεινότητα)
channel_id = 0;
pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED;       // Αριστερή
ευθυγράμμιση
pwm_channel.CMR.cpol = PWM_POLARITY_LOW;            // Κανονική
πολικότητα
pwm_channel.CMR.cpd = PWM_UPDATE_DUTY;              // Ενημέρωση duty
cycle
pwm_channel.CMR.cpre = AVR32_PWM_CPRE_MCK_DIV_4;    // Επιλογή
συχνότητας 115200/4 Hz
pwm_channel.cdty = PWM_DUR;
```

```
pwm_channel.cprd = PWM_PER;
pwm_channel.cupd = 0;   // Ο UPD θα ενημερώνει τον CDTY
pwm_channel_init(channel_id, &pwm_channel); // Αρχικοποίηση καναλιού 0
pwm_start_channels(1 << channel_id);  // Εκκίνηση καναλιού 0

// Αρχικοποίηση καναλιού PWM 4 (LED1 - ποτενσιόμετρο)
channel_id = 4;
pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED;       // Αριστερή
ευθυγράμμιση
pwm_channel.CMR.cpol = PWM_POLARITY_LOW;            // Κανονική
πολικότητα
pwm_channel.CMR.cpd = PWM_UPDATE_DUTY;              // Ενημέρωση duty
cycle
pwm_channel.CMR.cpre = AVR32_PWM_CPRE_MCK_DIV_4;    // Επιλογή
συχνότητας 115200/4 Hz
pwm_channel.cdty = PWM_DUR;
pwm_channel.cprd = PWM_PER;
pwm_channel.cupd = 0;   // Ο UPD θα ενημερώνει τον CDTY
pwm_channel_init(channel_id, &pwm_channel); // Αρχικοποίηση καναλιού 0
pwm_start_channels(1 << channel_id);  // Εκκίνηση καναλιού 0

// Ρύθμιση ADC
volatile avr32_adc_t *adc = &AVR32_ADC;
// set Sample/Hold time to max so that the ADC capacitor should be
loaded entirely
adc->mr |= 0xF << AVR32_ADC_SHTIM_OFFSET;
// set Startup to max so that the ADC capacitor should be loaded
entirely
adc->mr |= 0x1F << AVR32_ADC_STARTUP_OFFSET;
// Χρονισμός ADC = MCK/64
adc->mr |= 0x3F << AVR32_ADC_PRESCAL_OFFSET;
// Δημιουργία διακοπής με το τρίτο κανάλι
adc->ier |= 1 << AVR32_ADC_IER_EOC2_OFFSET;
// Ενεργοποίηση καναλιών 0,1,2
adc->cher = 7; //(1+2+4)

// Ρύθμιση διακοπών
INTC_register_interrupt(&ADC_int_handler,AVR32_ADC_IRQ,
AVR32_INTC_INT0);
Enable_global_interrupt();

int i;
while (1)                    // Ατέρμονος βρόχος
      {
      for (i = 0; i < 500; i += 1)
      adc_start(adc);               // Έναρξη νέων μετρήσεων
      gpio_tgl_gpio_pin(LED2_GPIO); // Αναβόσημα LED3
      }
}
```

IV.5 Κώδικας παραδείγματος χρήσης LCD

Απαραίτητοι οδηγοί για να τρέξει το παράδειγμα στο AVR 32 Studio : **GPIO**, **INTC,PWM,ADC,SPI,CPU,PM** και το εξάρτημα **DIP204.**

```
/*****************************************************
 Name          : gpio_int_pwm_adc.c
 Author        : adp@unipi.gr
 Copyright     :
 Description   : ---LED3 αναβοσβήνει μόνιμα,
                 ---LED1 φωτεινότητα εξαρτώμενη από ποτενσιόμετρο,
                 ---LED5κόκκινο φωτεινότητα εξαρτώμενη από το φως,
                 ---LED6κόκκινο φωτεινότητα εξαρτώμενη
                    από τη θερμοκρασία
                 ---Οθόνη LCD δείχνει συνεχώς τις τρεις μετρήσεις
 *****************************************************/

#include "compiler.h"
#include "board.h"
#include "intc.h"
#include "gpio.h"
#include "pwm.h"
#include "adc.h"
#include "spi.h"
#include "pm.h"
#include "cycle_counter.h"
#include <avr32/io.h>
#include "dip204.h"
#include <stdio.h>

#define PWM_PER 128
#define PWM_DUR 127
unsigned int channel_id;

// Ρουτίνα διακοπής για τον χειρισμό των δεδομένων του ADC
// και την αλλαγή φωτεινότητας των LED1,LED5κόκκινο και LED6κόκκινο
#if __GNUC__
      __attribute__((__interrupt__))
#elif __ICCAVR32__
__interrupt
#endif

static void ADC_int_handler(void)
{
volatile avr32_pwm_t *pwm = &AVR32_PWM;
volatile avr32_adc_t *adc = &AVR32_ADC;
int LED6Rint,LED5Rint,LED1int;
unsigned int tempr,light,pot;

// Ανάγνωση ADC
tempr=adc_get_value(adc,0);
pot=adc_get_value(adc,1);
light=adc_get_value(adc,2);
// Υπολογισμοί και αλλαγή φωτεινότητας
LED6Rint=PWM_DUR*tempr/1024;
pwm->channel[2].cupd=LED6Rint;
LED5Rint=PWM_DUR*light/1024;
```

```
pwm->channel[0].cupd=LED5Rint;
LED1int=PWM_DUR*pot/1024;
pwm->channel[4].cupd=LED1int;
// Εμφάνιση μετρήσεων στο LCD
char s[10];
dip204_set_cursor_position(6,2);
sprintf(s,"%4d",tempr);
dip204_write_string(s);
dip204_set_cursor_position(5,3);
sprintf(s,"%4d",pot);
dip204_write_string(s);
dip204_set_cursor_position(7,4);
sprintf(s,"%4d",light);
dip204_write_string(s);
dip204_hide_cursor();
}

// Ρουτίνα διακοπής που χρησιμοποιείται από την delay_ms
unsigned char TimeOut = 0;
#if __GNUC__
  __attribute__((__interrupt__))
#elif __ICCAVR32__
  interrupt
#endif
static void compare_irq_handler(void)
{
TimeOut = 1;  // Κάνε τον ατέρμονο βρόχο της delay_ms να "σπάσει"
Set_sys_compare(0); // COMPARE=0 δηλαδή απενεργοποίση διακοπών τύπου
COMPARE
}

// Συνάρτηση καθυστέρσησης - απαραίτητη στην dip204_init
void delay_ms(unsigned short time_ms)
{
unsigned long u32CountVal,u32CompareVal;
TimeOut = 0;
u32CountVal = Get_sys_count();
u32CompareVal = u32CountVal  + ((unsigned long)time_ms * (FOSC0 /
1000)); // WARNING: MUST FIT IN 32bits.
Set_sys_compare(u32CompareVal); //COMPARE=COUNT+delay
while (!TimeOut); // Μέχρι να γίνει 1 από τον χειριστή
compare_irq_handler
                                // δηλαδή COUNT=COMPARE
}

// Έναρξη Προγράμματος
int main(void)
{
pwm_opt_t pwm_opt;
avr32_pwm_channel_t pwm_channel;
extern unsigned int channel_id;




// Κύριο Ρολόι από τον ταλαντωτή  0
 pm_switch_to_osc0(&AVR32_PM, FOSC0, OSC0_STARTUP);

// Σύνδεση καναλιού 2 του PWM με το GPIO pin 53 (PB21-LED6 κόκκινο)
```

```
gpio_enable_module_pin(AVR32_PWM_2_PIN, AVR32_PWM_2_FUNCTION);
// Σύνδεση καναλιού 0 του PWM με το GPIO pin 51 (PB19-LED5 κόκκινο)
gpio_enable_module_pin(AVR32_PWM_0_PIN, AVR32_PWM_0_FUNCTION);
// Σύνδεση καναλιού 4 του PWM με το GPIO pin 59 (PB27-LED1)
gpio_enable_module_pin(AVR32_PWM_4_1_PIN, AVR32_PWM_4_1_FUNCTION);

// Σύνδεση καναλιού 0 του ADC με το GPIO pin 21 (PA21-θερμιστορ)
gpio_enable_module_pin(AVR32_ADC_AD_0_PIN, AVR32_ADC_AD_0_FUNCTION);
// Σύνδεση καναλιού 1 του ADC με το GPIO pin 22 (PA22-ποτενσιόμετρο)
gpio_enable_module_pin(AVR32_ADC_AD_1_PIN, AVR32_ADC_AD_1_FUNCTION);
// Σύνδεση καναλιού 2 του ADC με το GPIO pin 23 (PA23-φωτοαντίσταση)
gpio_enable_module_pin(AVR32_ADC_AD_2_PIN, AVR32_ADC_AD_2_FUNCTION);

// Αρχικοποίηση PWM MR
//Ρολόι του PWM είναι το κύριο ρολόι χωρίς διαίρεση
pwm_opt.diva = AVR32_PWM_DIVA_CLK_OFF;
pwm_opt.divb = AVR32_PWM_DIVB_CLK_OFF;
pwm_opt.prea = AVR32_PWM_PREA_MCK;
pwm_opt.preb = AVR32_PWM_PREB_MCK;
pwm_init(&pwm_opt);

// Αρχικοποίηση καναλιού PWM 2 (κόκκινο LED6 - θερμοκρασία)
channel_id = 2;
pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED;       // Αριστερή
ευθυγράμμιση
pwm_channel.CMR.cpol = PWM_POLARITY_LOW;            // Κανονική
πολικότητα
pwm_channel.CMR.cpd = PWM_UPDATE_DUTY;              // Ενημέρωση duty
cycle
pwm_channel.CMR.cpre = AVR32_PWM_CPRE_MCK_DIV_4;    // Επιλογή
συχνότητας 115200/4 Hz
pwm_channel.cdty = PWM_DUR;
pwm_channel.cprd = PWM_PER;
pwm_channel.cupd = 0;   // Ο UPD θα ενημερώνει τον CDTY
pwm_channel_init(channel_id, &pwm_channel); // Αρχικοποίηση καναλιού 2
pwm_start_channels(1 << channel_id);  // Εκκίνηση καναλιού 2

// Αρχικοποίηση καναλιού PWM 0 (κόκκινο LED5 - φωτεινότητα)
channel_id = 0;
pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED;       // Αριστερή
ευθυγράμμιση
pwm_channel.CMR.cpol = PWM_POLARITY_LOW;            // Κανονική
πολικότητα
pwm_channel.CMR.cpd = PWM_UPDATE_DUTY;              // Ενημέρωση duty
cycle
pwm_channel.CMR.cpre = AVR32_PWM_CPRE_MCK_DIV_4;    // Επιλογή
συχνότητας 115200/4 Hz
pwm_channel.cdty = PWM_DUR;
pwm_channel.cprd = PWM_PER;
pwm_channel.cupd = 0;   // Ο UPD θα ενημερώνει τον CDTY
pwm_channel_init(channel_id, &pwm_channel); // Αρχικοποίηση καναλιού 0
pwm_start_channels(1 << channel_id);  // Εκκίνηση καναλιού 0

// Αρχικοποίηση καναλιού PWM 4 (LED1 - ποτενσιόμετρο)
channel_id = 4;
pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED;       // Αριστερή
ευθυγράμμιση
pwm_channel.CMR.cpol = PWM_POLARITY_LOW;            // Κανονική
πολικότητα
```

```
pwm_channel.CMR.cpd = PWM_UPDATE_DUTY;              // Ενημέρωση duty
cycle
pwm_channel.CMR.cpre = AVR32_PWM_CPRE_MCK_DIV_4;    // Επιλογή
συχνότητας 115200/4 Hz
pwm_channel.cdty = PWM_DUR;
pwm_channel.cprd = PWM_PER;
pwm_channel.cupd = 0;   // Ο UPD θα ενημερώνει τον CDTY
pwm_channel_init(channel_id, &pwm_channel); // Αρχικοποίηση καναλιού 0
pwm_start_channels(1 << channel_id);  // Εκκίνηση καναλιού 0

// Ρύθμιση ADC
volatile avr32_adc_t *adc = &AVR32_ADC;
// set Sample/Hold time to max so that the ADC capacitor should be
loaded entirely
adc->mr |= 0xF << AVR32_ADC_SHTIM_OFFSET;
// set Startup to max so that the ADC capacitor should be loaded
entirely
adc->mr |= 0x1F << AVR32_ADC_STARTUP_OFFSET;
// Χρονισμός ADC = MCK/64
adc->mr |= 0x3F << AVR32_ADC_PRESCAL_OFFSET;
// Δημιουργία διακοπής με το τρίτο κανάλι
adc->ier |= 1 << AVR32_ADC_IER_EOC2_OFFSET;
// Ενεργοποίηση καναλιών 0,1,2
adc->cher = 7; //(1+2+4)

// Ρύθμιση διακοπών
Disable_global_interrupt();
INTC_init_interrupts();

//Χειριστής διακοπών από καταχωρητές COUNT/COMPARE
INTC_register_interrupt(&compare_irq_handler, AVR32_CORE_COMPARE_IRQ,
AVR32_INTC_INT0);
//Χειριστής διακοπών από ADC
INTC_register_interrupt(&ADC_int_handler,AVR32_ADC_IRQ,
AVR32_INTC_INT0);

Enable_global_interrupt();

//Ορισμός συσχέτισης SPI με εξωτερικούς ακροδέκτες
static const gpio_map_t DIP204_SPI_GPIO_MAP =
{
  {DIP204_SPI_SCK_PIN,  DIP204_SPI_SCK_FUNCTION },  // SPI Clock.
  {DIP204_SPI_MISO_PIN, DIP204_SPI_MISO_FUNCTION},  // MISO.
  {DIP204_SPI_MOSI_PIN, DIP204_SPI_MOSI_FUNCTION},  // MOSI.
  {DIP204_SPI_NPCS_PIN, DIP204_SPI_NPCS_FUNCTION}   // Chip Select
NPCS.
};

//  Ορισμός παραμέτρων που αφορούν επικοινωνία με οθόνη
 spi_options_t spiOptions =
 {
   .reg          = DIP204_SPI_NPCS,
   .baudrate     = 1000000,
   .bits         = 8,
   .spck_delay   = 0,
   .trans_delay  = 0,
   .stay_act     = 1,
   .spi_mode     = 0,
   .fdiv         = 0,
```

```
   .modfdis      = 1
 };

 // Σύνδεση SPI με εξωτερικούς ακροδέκτες (αυτούς που οδηγούνται στην
οθόνη
 gpio_enable_module(DIP204_SPI_GPIO_MAP,
                    sizeof(DIP204_SPI_GPIO_MAP) /
sizeof(DIP204_SPI_GPIO_MAP[0]));

 // Το SPI του μικροελεγκτή είναι master
 spi_initMaster(DIP204_SPI, &spiOptions);

 // Ρύθμιση γενικών παραμέτρων
 spi_selectionMode(DIP204_SPI, 0, 0, 0);

 // Ενεργοποίηση SPI
 spi_enable(DIP204_SPI);

 // Ρύθμιση παραμέτρων που αφορούν επικοινωνία με οθόνη
 spi_setupChipReg(DIP204_SPI, &spiOptions, FOSC0);

 // Αρχικοποίηση Οθόνης LCD
dip204_init(backlight_IO, TRUE);



 // Εμφάνιση Σταθερών μηνυμάτων στην οθόνη.
dip204_set_cursor_position(1,1);
dip204_write_string("    MEASUREMENTS");
dip204_set_cursor_position(1,2);
dip204_write_string("TEMP=");
dip204_set_cursor_position(1,3);
dip204_write_string("POT=");
dip204_set_cursor_position(1,4);
dip204_write_string("LIGHT=");
dip204_set_cursor_position(3,4);
dip204_hide_cursor();


int i;
while (1)                    // Ατέρμονος βρόχος
      {
      for (i = 0; i < 50000; i += 1)
      adc_start(adc);               // Έναρξη νέων μετρήσεων
      gpio_tgl_gpio_pin(LED2_GPIO); // Αναβόσημα LED3
      }
}
```

V Αναφορές.

Α. Έγγραφα σε ηλεκτρονική μορφή.

[1] "*AVR® 32 : 32-Bit Microcontroller (UC3A Series)*", αρχείο `doc32058.pdf` , Rev.E - 04/08.
[2] "*EVK1100 Schematics*", αρχείο `EVK1100_SCHEMATICS_REVC.pdf`, Rev. C - 12/08.
[3] "*AVR32 EVK1100 Getting Started Guide*", αρχείο `doc7713.pdf`, Rev. A – 05/07.
[4] "AVR32 UC3 EVK1100 Control Panel Tutorial".
[5] *"AVR32 Studio getting started"*, αρχείο `doc.32086.pdf`, Rev. C – 04/08.
[6] "*AV32 UC3 Software Framework : User Guide*", αρχείο `doc7732.pdf`, 05/07.
[7] "*AVR32 UC3 USB DFU Bootloader*", αρχείο `doc7745.pdf`, 07/07.
[8] "*Getting Started with AVR32 UC3A Microcontrollers*", αρχείο `doc32078.pdf`,12/07.
[9] "*Getting Started with GCC for AVR32*", αρχείο `doc32074.pdf`,12/07.
[10] "*Introduction to AVR32 header files*", αρχείο `doc32005.pdf`,05/08.
[11] "*Using the AVR32 PIO Controller*", `doc32014.pdf`, 05/06.
[12] "*Configuring the AVR32 Interrupt Controller*",αρχείο `doc32010.pdf`,04/06.
[13] http://www.lcd-module.de/eng/pdf/doma/dip204-4e.pdf
[14] http://www.lcd-module.de/eng/pdf/zubehoer/ks0073.pdf
[15] "*Master and Slave SPI Driver*", αρχείο `32017.pdf`, 05/06.
[16] "AVR32 Architecture Document", αρχείο `doc32000.pdf`, Rev. B, 11/07.

Β. Ιστοσελίδες.

`http://www.atmel.com`
`http://support.atmel.no/bin/customer`
`http://www.avrfreaks.net`