ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ

Τμήμα Μηχανικών Πληροφορικής

Εργαστήριο Σχεδιασμού Ενσωματωμένων Συστημάτων & Εφαρμογών

http://esda-lab.tesyd.teimes.gr

ΒΙΒΛΙΟ ΕΡΓΑΣΤΗΡΙΟΥ

ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ Ι

Δρ Νικόλαος Σπ. Βώρος

Επίκουρος Καθηγητής

Έκδοση 1η

Μάρτιος 2014

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΙΝΑΚΑΣ ΕΡΩΤΗΜΑΤΩΝ

ΠΙΝΑΚΑΣ ΣΧΗΜΑΤΩΝ

ΕΙΣΑΓΩΓΗ

- **Τι είναι οι μικροελεγκτές και ποιος ο σκοπός τους**
- **Τι είναι τα αναπτυξιακά και ποιος ο σκοπός τους**

Οι μικροελεγκτές, microcontrollers ή MCUs, αποτελούνται από έναν πυρήνα-μικροεπεξεργαστή, (ARM, C51 κτλ) ο οποίος συνδέεται με διάφορα περιφερειακά μέσω διαύλων επικοινωνίας. Ένας μικροελεγκτής είναι ουσιαστικά ένας μικρός υπολογιστής αποτελούμενος από επεξεργαστή, μνήμη και περιφερειακά, κατασκευασμένος όμως για συγκεκριμένες εφαρμογές. Επειδή οι μικροελεγκτές είναι κυρίως κατασκευασμένοι για συγκεκριμένες εφαρμογές, το κόστος μπορεί να παραμείνει χαμηλό όπως επίσης η κατανάλωση τους, γιατί μόνο τα πλήρως απαραίτητα στοιχεία για την συγκεκριμένη εφαρμογή εμπεριέχονται.

Ποιός πυρήνας και τι περιφερειακά επιλέγονται για να αποτελέσουν έναν μικροελεγκτή, εξαρτάται από τον σκοπό για τον οποίο θα κατασκευαστεί ο μικροελεγκτής αυτός. Έτσι για παράδειγμα εάν ο σκοπός του κατασκευαστή είναι να δημιουργήσει έναν μικροελεγκτή για ένα κινητό τηλέφωνο, ανάλογα με τα χαρακτηριστικά και τις δυνατότητες που θα θέλει να έχει το τηλέφωνο, θα επιλέξει εκείνο τον πυρήνα και εκείνα τα περιφερειακά που θα δώσουν στο κινητό τηλέφωνο τα χαρακτηριστικά που χρειάζονται με το χαμηλότερο κόστος. Εάν θέλει, για παράδειγμα, ένα απλό τηλέφωνο χωρίς κάμερες, ραδιόφωνο κτλ, θα επιλέξει να κατασκευάσει ένα μικροελεγκτή με μικρό και φθηνό πυρήνα. Τα περιφερειακά του προφανώς θα είναι ένας LCD controller ένας SIM card controller και κάποια ADCs. Αρκετές φορές, οι

εταιρίες κατασκευάζουν έναν μικροελεγκτή ο οποίος μπορεί να χρησιμοποιηθεί σε μεγαλύτερη γκάμα σειράς προϊόντων. Δηλαδή στο παράδειγμα με το κινητό μπορεί η εταιρία να επιλέξει να κατασκευάσει μικροελεγκτή που θα έχει παραπάνω δυνατότητες από αυτές που χρειάζονται σε ένα απλό κινητό. Σκοπός θα είναι να μπορεί να χρησιμοποιηθεί ο ίδιος μικροελεγκτής μετέπειτα σε ένα πιο high-end κινητό. Έτσι η εταιρία θα θυσιάσει το υψηλό κόστος κατασκευής για ένα απλό κινητό αλλά θα γλιτώσει την παραπάνω έρευνα για κατασκευή ενός δεύτερου μικροελεγκτή που θα καλύψει μελλοντικές ανάγκες αναβάθμισης.

Υπάρχουν βέβαια και εταιρίες που κατασκευάζουν μικροελεγκτές γενικού σκοπού. Οι εταιρίες αυτές, όπως η STMicroelectronics η Atmel κ.α., δεν κατασκευάζουν μικροελεγκτές με σκοπό συγκεκριμένα προϊόντα αλλά έχουν σκοπό στην πώληση σε μια ευρύτερη αγορά. Έτσι μια εταιρία που για παράδειγμα κατασκευάζει πλυντήρια, ο μικροελεγκτής που θα χρησιμοποιήσει δεν χρειάζεται να περιέχει κάποια εξειδικευμένα περιφερειακά. Μπορεί έτσι να επιλέξει εκείνον τον μικροελεγκτή, γενικού σκοπού, που πληρή τις προδιαγραφές με το χαμηλότερο κόστος. Σαφώς το κόστος δεν αποτελεί πάντα πρωταρχικό ρόλο, καθώς μια εταιρία μπορεί να επιλέξει ένα μικροελεγκτή γενικού σκοπού από έναν κατασκευαστή που έχει καλύτερη τεχνική υποστήριξη και μικρότερο χρόνο παράδοσης.

Οι αναπτυξιακές πλακέτες χρησιμοποιούνται κυρίως για τους εξής λόγους:

- Αποσφαλμάτωση των MCU. Οι εταιρίες, κατά την διάρκεια σχεδίασης των MCU ή και γενικώς κατασκευής οποιαδήποτε άλλου IC, κατασκευάζουν παράλληλα και αναπτυξιακές πλακέτες για να διαπιστώσουν την εύρυθμη λειτουργία τους. Η πρώτη παρτίδα των

IC που θα βγει από το εργοστάσιο (pilot run tape-out) δοκιμάζεται σε τέτοιου είδους πλακέτες.

- Επίδειξη της σωστής και εύρυθμης λειτουργίας του μικροελεγκτή και των περιφερειακών του (demonstration). Συνήθως οι κατασκευαστές των IC, κατασκευάζουν αναπτυξιακές πλακέτες για να δείξουν στους πελάτες τους ότι το IC τους δουλεύει βάση των αναφερόμενων προδιαγραφών.
- Ανάπτυξη εφαρμογών λογισμικού για συσκευές/προϊόντα που χρησιμοποιούν τον συγκεκριμένο η παραπλήσιο μικροελεγκτή.
- Επίδειξη της εύρυθμης λειτουργίας εφαρμογών λογισμικού (demonstration) για μελλοντική χρήση τους σε συσκευές/προϊόντα που χρησιμοποιούν τον συγκεκριμένο η παραπλήσιο μικροελεγκτή.

1 ΕΡΓΑΣΤΗΡΙΟ 1

♦ **Μαθαίνοντας το hardware του αναπτυξιακού**

Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Είναι ένας πλήρης οδηγός του αναπτυξιακού που ο σπουδαστής πρέπει να διαβάσει πριν προχωρήσει στην υλοποίηση των εργαστηρίων.

- **Αρχείο MCBSTR9_schematic.pdf,** που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Είναι το σχηματικό της αναπτυξιακής πλακέτας MCBSTR9.

- **Αρχείο STR91xFAxxx.pdf,** που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Σελίδες 14, 56, 57 και από σελίδα 15 – 39 μια ανάγνωση για τα κύρια χαρακτηριστικά της κάθε μονάδας του μικροελεγκτή.

- **Αρχείο STR91xFA_Reference_Manual.pdf**, που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Το χρησιμοποιούμε ως αναφορά για τους καταχωρητές και περιφερειακά του μικροελεγκτή.

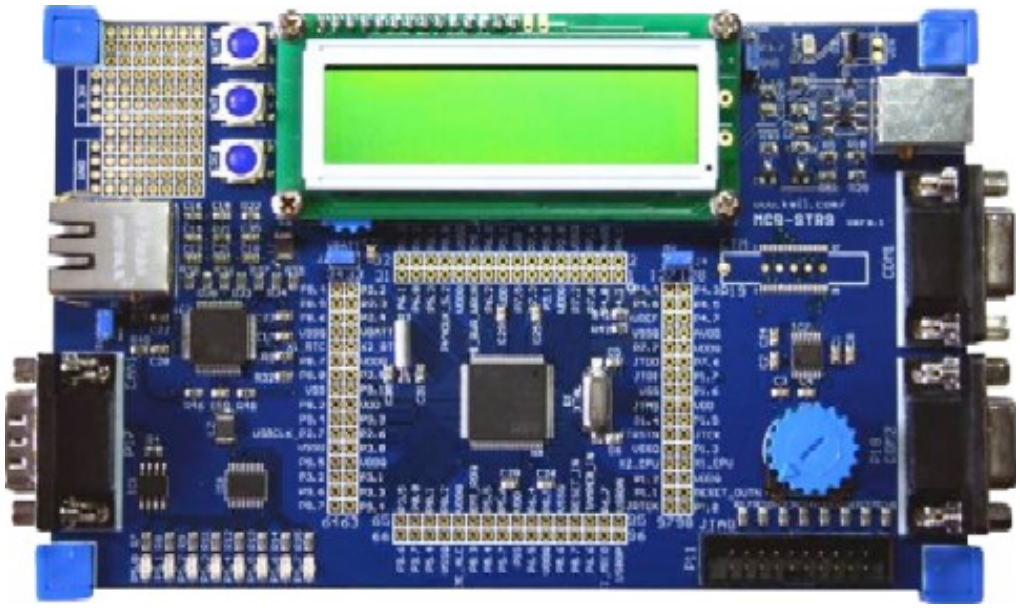
- **Βιβλίο Θεωρίας Wayne Wolf, "Οι Υπολογιστές ως Συστατικά Στοιχεία".** Κεφάλαιο 2, παράγραφος 2.2. Ο Επεξεργαστής ARM.

Εισαγωγή

Στο εργαστήριο αυτό θα προσπαθήσουμε να αποκτήσουμε μια σφαιρική εικόνα για το αναπτυξιακό πακέτο MCBSTR9 της Keil και κυρίως για το **υλικό** που χρησιμοποιείται **(hardware)**.

Πρώτα θα ασχοληθούμε με τα **περιφερειακά** της πλακέτας και μετά με τον **μικροελεγκτή** (MCU) του συστήματος, τον STR912FAW44 της STMicroelectronic's.

Η αναπτυξιακή πλακέτα MCBSTR9 της Keil που φαίνεται στο παρακάτω σχήμα, αποτελεί μια ολοκληρωμένη λύση για ανάπτυξη προγραμμάτων πάνω στους μικροελεγκτές (MCUs) **STR912FAW44 της STMicroelectronic's**, βασισμένοι σε πυρήνες της ARM. Έχει όλα τα απαραίτητα στοιχεία, όπως USB, UART, CAN και ETHERNET διεπαφές αλλά και περιοχή για πρωτότυπα κυκλώματα, ώστε ο χρήστης να μπορέσει να ξεδιπλώσει όλες τις δυνατότητες του μικροελεγκτή.



Σχήμα 1: MCBSTR9 Keil Development Board

1.1 Το Υλικό του Αναπτυξιακού (Hardware)

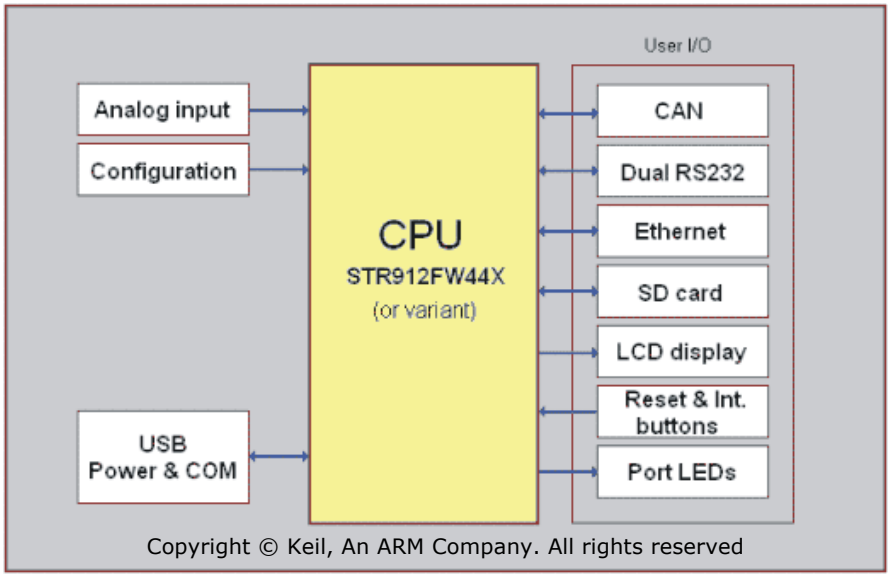
Η πλακέτα του αναπτυξιακού αποτελείται από έναν μικροελεγκτή και τα περιφερειακά του όπως φαίνεται στο παρακάτω σχήμα (Σχήμα 2). Τα περιφερειακά της αναπτυξιακής πλακέτας, όπως για παράδειγμα η USB, θύρα, προσφέρουν ουσιαστικά μια πρόσβαση στους αντίστοιχους περιφερειακούς ελεγκτές που βρίσκονται εσωτερικά στον μικροελεγκτή.

Μικροελεγκτής:

- STR912FAW4XB της STMicroelectronics.

Περιφερειακά:

- LCD, 2x16 αλφαριθμητικό
- USB 2.0 θύρα
- Ethernet θύρα
- 3 Πλήκτρα
- 7 LEDs
- CAN 2.0B θύρα
- 2x RS-232 σειριακές θύρες
- 1x Μετατροπέας αναλογικής τάσης
- JTAG θύρα
- SD μετατροπέας



Σχήμα 2: Σχηματικό διάγραμμα της MCBSTR9 πλακέτας

1.1.1 Διαβάζοντας το σχηματικό

Το σχηματικό, που δίνεται στο MCBSTR9_schematic.pdf, αποτελεί μια από τις βασικότερες πληροφορίες και χρησιμοποιείται για να βρούμε πως τα διάφορα στοιχεία πάνω στην πλακέτα συνδέονται μεταξύ τους. Το σχηματικό πρέπει να αποτελεί πάντα την βάση της οποιαδήποτε υλοποίηση μας.

Στο σχηματικό, κάθε στοιχείο έχει ένα όνομα. Για παράδειγμα, στην πρώτη σελίδα, του MCBSTR9_schematic.pdf, κάτω αριστερά, μπορούμε να δούμε κάποιους πυκνωτές, Τα ονόματα τους είναι C30 και C31. Αντίστοιχα, ονόματα έχουν τα ICs οι θύρες και γενικά οτιδήποτε υπάρχει πάνω σε μια πλακέτα, όπως μία τρύπα για βίδα ή μια θύρα για USB κτλ.

Παρατηρώντας το σχηματικό απαντήστε τα παρακάτω ερωτήματα.

Ερώτημα 1	
Βρείτε ποια ονόματα έχουν δοθεί στο σχηματικό για τα παρακάτω στοιχεία.	
USB θύρα	
7x LED	
LCD Οθόνη	
3x Πλήκτρα	
JTAG θύρα	
Ποτενσιόμετρο	

Παρακάτω θα δώσουμε μια σύντομη περιγραφή του μικροελεγκτή και των περιφερειακών του.

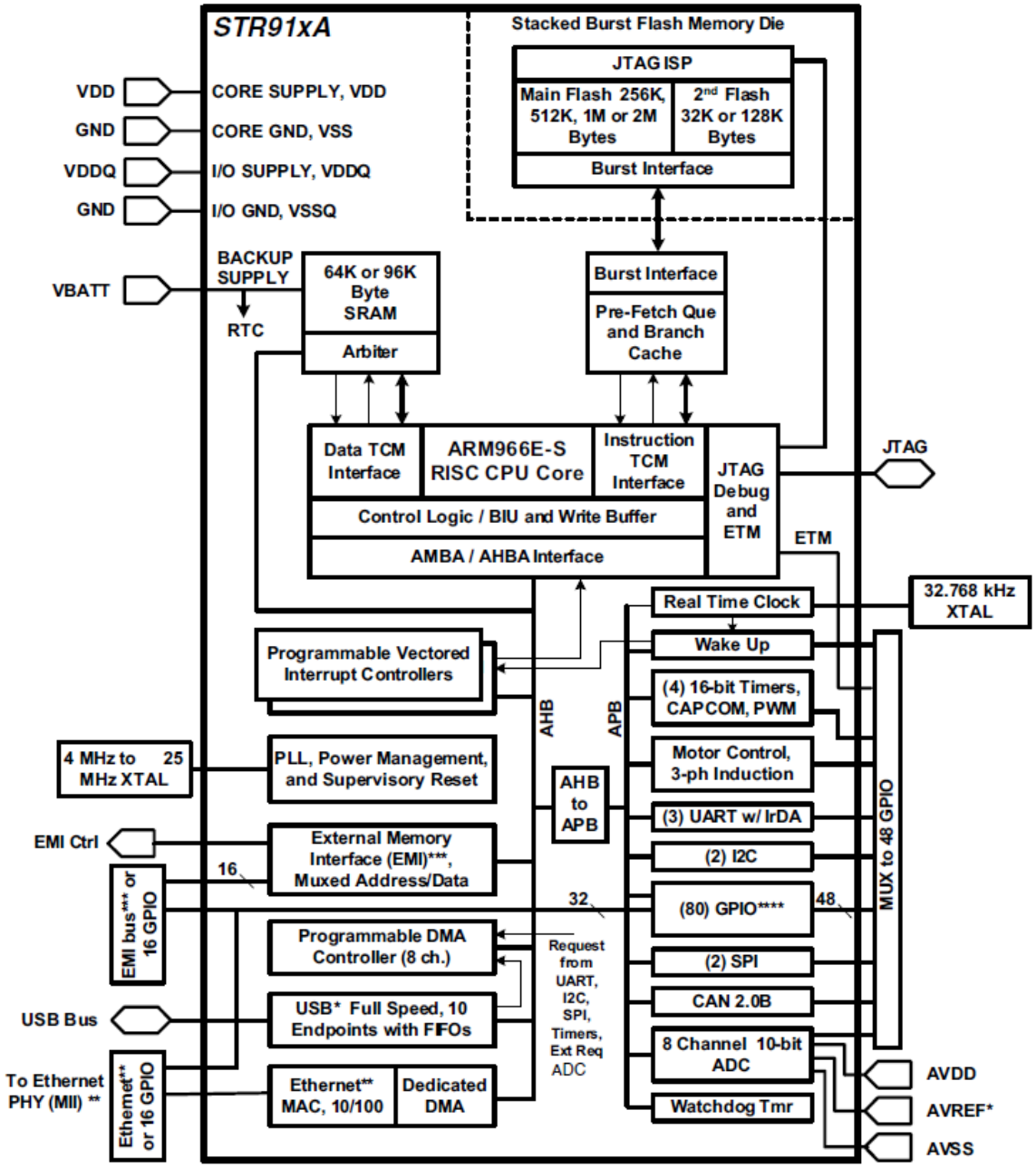
1.1.2 STR912FAW44XB

Στα συνημμένα έγγραφα του εργαστηρίου θα βρείτε δύο κείμενα STR91xFAxxx.pdf και STR91xFA_Reference_Manual.pdf. Τα κείμενα αυτά περιέχουν την πλήρη περιγραφή του μικροελεγκτή. Εδώ θα περιγράψουμε τα βασικότερα στοιχεία του.

Πυρήνας (CPU)	16/32-bit 96 MHz ARM966E-S RISC
2 εσωτερικές μνήμες Flash	• 512Kb πρωτεύουσα • 32Kb δευτερεύουσα
Εσωτερική Static RAM	64Kb
Εσωτερικός ταλαντωτής ρολογιού	Λειτουργεί με εξωτερικό κρύσταλλο 4-25MHz. Ο ταλαντωτής τροφοδοτεί το εσωτερικό PLL, το οποίο μπορεί να δώσει ταχύτητες έως 96MHz
Εξωτερική διεπαφή μνήμης	External Memory Interface (EMI)
80 I/O Γενικού σκοπού	General Purpose I/O (GPIO)
RTC (Real Time Clock)	Ρολόι πραγματικού χρόνου
10-bit ADC 8-καναλιών	Analogue to Digital Converter.
10 Communication Interfaces	Διασυνδέσεις επικοινωνίας • 10/100 Ethernet MAC • USB 2.0 slave • CAN 2.0B Active • 3x UART • 2x Fast I^2C 400KHz
4x 16-bit Timers	Χρονιστές
3-Phase Induction Motor Controller	Ελεγκτής επαγωγικού κινητήρα 3-φάσεων
JTAG	Διεπαφή ελέγχου και προγραμματισμού

Πίνακας 1: Βασικά χαρακτηριστικά του STR912FAW44XB MCU

Το επόμενο σχήμα δείχνει την βασική δομή του STR912F44XB. Εδώ παρουσιάζονται τα βασικά στοιχεία του MCU σε blocks και παράλληλα δίδεται σχηματικά πως αυτά επικοινωνούν μεταξύ τους.



Σχήμα 3: Διάγραμμα MCU της οικογένειας STR9xFAxxx

Παρατηρώντας το σχήμα 3, προσπαθήστε να απαντήσετε στα επόμενα ερωτήματα.

Ερώτημα 2
Τι αρχιτεκτονικής είναι ο ARM966E-S; Δικαιολογήστε το.

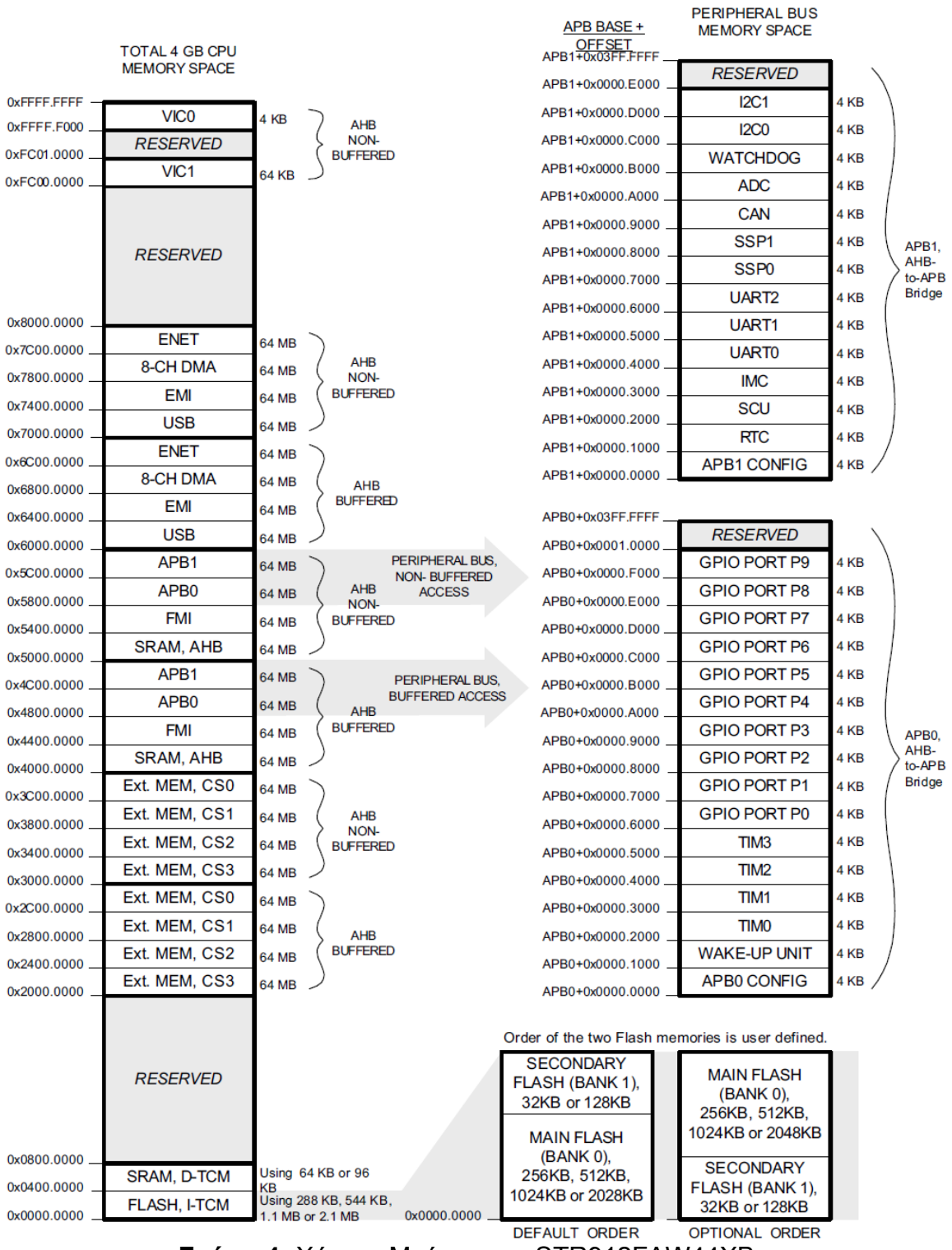
Ερώτημα 3
Πως συνδέετε ο ελεγκτής CAN με τον πυρήνα ARM966E-S;

Ερώτημα 4
Πόσοι και ποιοι τύποι μνήμης υπάρχουν εσωτερικά στον μικροελεγκτή;

Ερώτημα 5
Πόσα I/O γενικού σκοπού υπάρχουν (GPIO);

1.1.3 Χάρτης Μνήμης του STR912FAW44XB (Memory Map)

Κάθε στοιχείο του μικροελεγκτή είναι ορισμένο σε 32-bit καταχωρητές, ξεκινώντας από την θέση 0 έως την θέση 0xFFFFFFFF. Οτιδήποτε έχει να κάνει με τα περιφερειακά, τα interrupts και τις εσωτερικές ή εξωτερικές μνήμες ρυθμίζεται ή απεικονίζεται μέσω των καταχωρητών αυτών.



Σχήμα 4: Χάρτης Μνήμης του STR912FAW44XB

Έτσι για παράδειγμα, η θύρα I/O 3 γενικού σκοπού (General Perpose Input/Output ή GPIO Port 3) έχει καταχωρητές από την θέση μνήμης APB0 + 0x00009000 εώς APB0 + 0x000093FC. Δηλαδή στις θέσεις 0x48009000-0x480093FC. Στην θέση μνήμης 0x4800A000 ξεκινάνε οι καταχωρητές για το GPIO Port 4.

Το αρχείο STR91xFA_Reference_Manual.pdf περιέχει την πλήρη περιγραφή όλων των καταχωρητών του STR912FAW44XB. Ανοίγοντας το στην σελίδα 116 μπορούμε να διαβάσουμε την λειτουργία των καταχωρητών για τα GPIO Ports.

Ανατρέχοντας στο STR91xFA_Reference_Manual.pdf, απαντήστε στις παρακάτω ερωτήσεις.

Ερώτημα 6
Ποια η διεύθυνση των καταχωρητών για το RTC (Real Time Clock); **Πόσοι 32-bit καταχωρητές υπάρχουν για το RTC;**

Ερώτημα 7
Ποια η αρχική διεύθυνση των καταχωρητών για την UART1; **Πόσοι 32-bit καταχωρητές υπάρχουν για την UART1;**

Ερώτημα 8
Ανατρέχοντας στην σελίδα 83, περιγράψτε των σκοπό του SCU (System Control Unit).

1.1.4 LCD, 2x16 αλφαριθμητικό

Ένα από τα περιφερειακά της πλακέτας ανάπτυξης MCBSTR9 είναι το LCD. Τα LCD αυτού του τύπου αποτυπώνουν αλφαριθμητικούς χαρακτήρες οι οποίοι βρίσκονται σε εσωτερική μνήμη.

Ερώτημα 9
Παρατηρώντας το σχηματικό MCBSTR9_schematic.pdf, βρείτε σε ποιούς ακροδέκτες συνδέετε το LCD με τον μικροελεγκτή.

1.1.5 Ethernet θύρα

Το τμήμα Ethernet του μικροελεγκτή υλοποιεί το υποεπίπεδο Ελέγχου Προσπέλασης Μέσου (Medium Access Control ή MAC) που ανήκει στο επίπεδο Σύνδεσης Δεδομένων. Το Φυσικό επίπεδο δεν υλοποιείται από τον μικροελεγκτή και έτσι απαιτείται εξωτερική λύση. Για να συνδεθεί το Φυσικό επίπεδο με το επίπεδο Σύνδεσης Δεδομένων, το STE100P chip της STMicroelectronics χρησιμοποιείται. Το STE100P είναι εάν IC υψηλών επιδόσεων που υλοποιεί την διεπαφή μεταξύ του Φυσικού και του επιπέδου Σύνδεσης Δεδομένων για 10-100Base-T Ethernet εφαρμογές.

Ερώτημα 10
Παρατηρώντας το σχηματικό MCBSTR9_schematic.pdf, βρείτε σε ποιούς ακροδέκτες συνδέετε η θύρα Ethernet με το STE100P.

Ερώτημα 11
Παρατηρώντας το σχηματικό MCBSTR9_schematic.pdf, βρείτε σε ποιούς ακροδέκτες του μικροελεγκτή συνδέετε ο STE100P.

1.1.6 CAN 2.0B θύρα

Ένας κλασικός DB9 connector χρησιμοποιείται για την σύνδεση συσκευών CAN στο αναπτυξιακό. Όπως στην περίπτωση του Ethernet έτσι και στο CAN ένα εξωτερικό chip χρειάζεται για να μετατρέψει το διαφορικό σήμα του CAN (δύο συμπληρωματικά σε τάση, ξεχωριστά σήματα) (differential) σε ένα μόνο σήμα (single-ended). Το chip αυτό είναι το SN65HVD230 της Texas Instruments.

Ερώτημα 12
Παρατηρώντας το σχηματικό MCBSTR9_schematic.pdf, βρείτε σε ποιούς ακροδέκτες του μικροελεγκτή συνδέετε το SN65HVD230.

1.1.7 2x RS-232 σειριακές θύρες

Αντίστοιχα με το CAN και το Ethernet, για να συνδεθούν συσκευές με το σειριακό πρωτόκολλο RS-232, χρειάζεται μετατροπέας. Η κύρια λειτουργία

του μετατροπέα είναι να μετατρέψει τα επίπεδα τάσης του RS-232 (μέγιστο +-25V) σε TTL/CMOS επίπεδα (3.3-5V). Το ST3232 είναι το chip που χρησιμοποιείται στην αναπτυξιακή πλακέτα για να μετατρέψει τα επίπεδα της τάσης και ουσιαστικά να συνδέσει το Φυσικό επίπεδο με το επίπεδο Σύνδεσης Δεδομένων.

Ερώτημα 13
Παρατηρώντας το σχηματικό MCBSTR9_schematic.pdf, βρείτε σε ποιούς ακροδέκτες του μικροελεγκτή συνδέετε το ST3232.

1.1.8 USB 2.0 θύρα

Η MCBSTR9 πλακέτα έχει μια USB 2.0 θύρα. Η θύρα αυτή συνδέεται με τον μικροελεγκτή μέσω ενός IC, του USBLC6-2. Το εσωτερικό USB τμήμα του μικροελεγκτή προσφέρει έναν USB ελεγκτή ο οποίος εφαρμόζει το Φυσικό και το επίπεδο Σύνδεσης Δεδομένων των OSI επιπέδων.

Ερώτημα 14
Βρείτε από το διαδίκτυο ποια η λειτουργία του IC USBLC6-2 και δικαιολογήστε εάν το IC αυτό αποτελεί κομμάτι του Φυσικού επιπέδου για το USB 2.0 πρωτόκολλο.

Ερώτημα 15
Παρατηρώντας το σχηματικό MCBSTR9_schematic.pdf, βρείτε σε ποιούς ακροδέκτες του μικροελεγκτή συνδέετε η USB θύρα. Από το STR91xFAxxx.pdf βρείτε την λειτουργία των ακροδεκτών αυτών.

1.1.9 Μετατροπέας αναλογικής τάσης (Ποτενσιόμετρο)

Ένας μετατροπέας αναλογικής τάσης, που βρίσκεται πάνω στην αναπτυξιακή πλακέτα μπορεί να χρησιμοποιηθεί για να δοκιμάσουμε τους ADCs του μικροελεγκτή. Ο μετατροπέας αυτός μπορεί να μεταβάλει την τάση εισόδου από 0 εώς 3,3V.

Ερώτημα 16
Παρατηρώντας το σχηματικό MCBSTR9_schematic.pdf, βρείτε σε ποιό ακροδέκτη του μικροελεγκτή συνδέετε το Ποτενσιόμετρο. Από το STR91xFAxxx.pdf βρείτε την λειτουργία του ακροδέκτη αυτού.

1.1.10 Πλήκτρα και LEDs

Τρία πλήκτρα και επτά LED υπάρχουν πάνω στην αναπτυξιακή πλακέτα. Ένα πλήκτρο είναι συνδεδεμένο στο Reset pin του MCU και τα υπόλοιπα δύο είναι συνδεδεμένα σε δύο ακροδέκτες γενικού σκοπού εισόδου/εξόδου (GPIO) του μικροελεγκτή τα οποία μπορούν να προγραμματιστούν και ως πηγές εξωτερικών διακοπών (External Interrupts).

Ερώτημα 17
Παρατηρώντας το σχηματικό MCBSTR9_schematic.pdf, βρείτε σε ποιούς ακροδέκτες του μικροελεγκτή συνδέονται τα πλήκτρα. Από το STR91xFAxxx.pdf βρείτε την λειτουργία των ακροδεκτών αυτών.

1.1.11 JTAG θύρα

Η θύρα JTAG προσφέρει μια κύρια μονάδα αποσφαλμάτωσης και προσομοίωση του συστήματος σε πραγματικό χρόνο. Επίσης χρησιμοποιείται για προγραμματισμό της εσωτερικής flash. Εμείς θα χρησιμοποιήσουμε το JTAG για να κατεβάζουμε κώδικα στον μικροελεγκτή.

Ερώτημα 18
Βάση των βασικών χαρακτηριστικών του μικροελεγκτή αναφέρεται οκτώ προϊόντα στα οποία θα μπορούσε να χρησιμοποιηθεί και γιατί;
1.
2.
3.
4.
5.
6.
7.
8.

2 ΕΡΓΑΣΤΗΡΙΟ 2

- **Μαθαίνοντας το λογισμικό του αναπτυξιακού**
- **Απλή χρήση του LCD**

Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

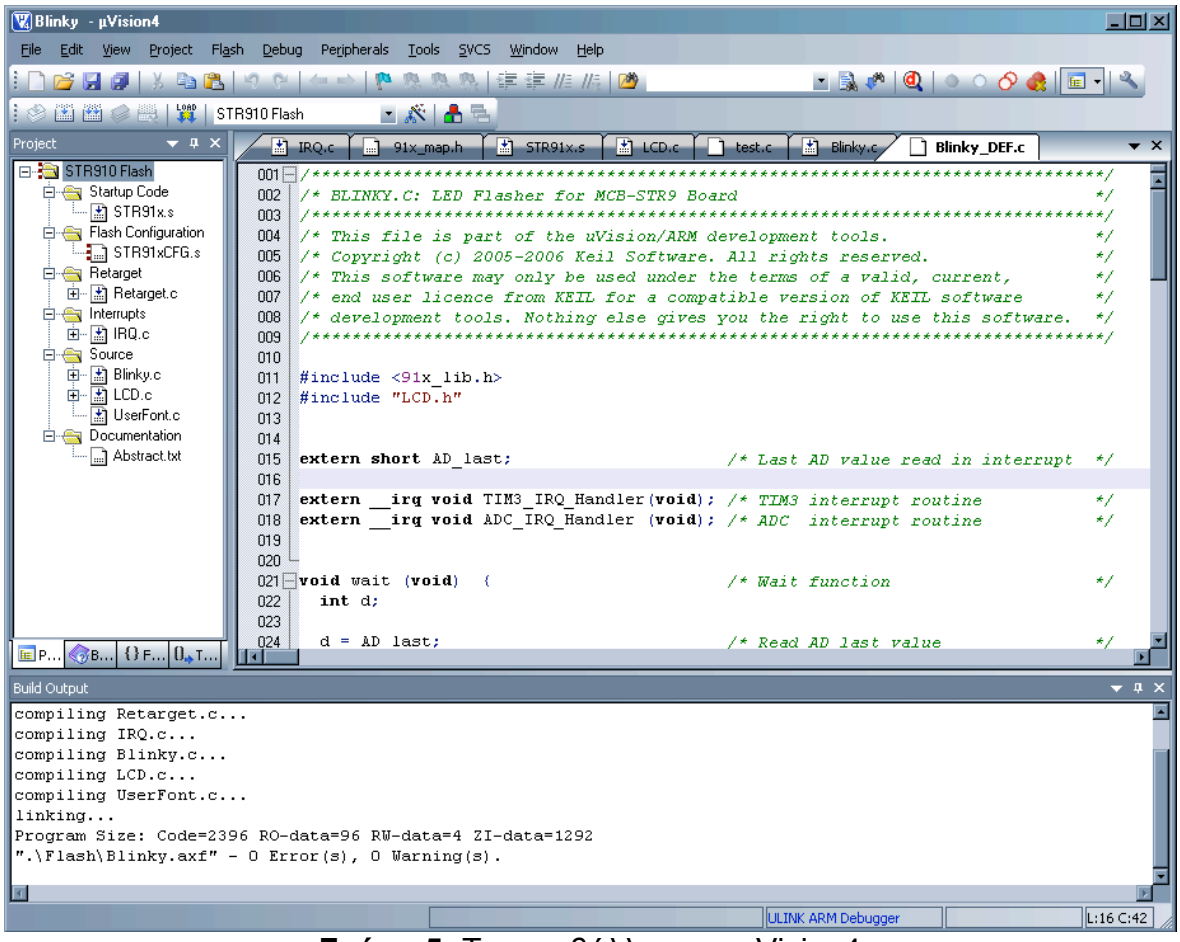
- **Αρχείο mcbstr9.chm HTML,** που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Είναι ένας πλήρης οδηγός του αναπτυξιακού που ο σπουδαστής πρέπει να διαβάσει πριν προχωρήσει στην υλοποίηση των εργαστηρίων. Κεφάλαιο: “Writing Programs”.

- **Αρχεία Blinky.c και LCD.c,** μέσα από τον φάκελο **Lab2_Blinky.rar**, που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου.

- **Βιβλίο Θεωρίας Wayne Wolf, “Οι Υπολογιστές ως Συστατικά Στοιχεία”.** Κεφάλαιο 4, παράγραφος 4. Συσκευές Εισόδου-Εξόδου, Οθόνες.

Εισαγωγή

Το λογισμικό που περιλαμβάνεται στο πακέτο του αναπτυξιακού και που θα χρησιμοποιήσουμε στο εργαστήριο, είναι το μVision4 της Keil. Το λογισμικό αυτό παρέχει ένα πλήρη περιβάλλον για δημιουργία και επεξεργασία κώδικα, απασφαλμάτωση προγραμμάτων με προσομοίωση πραγματικού χρόνου ή εξομοίωση και προγραμματισμό flash.



Σχήμα 5: Το περιβάλλον του uVision4

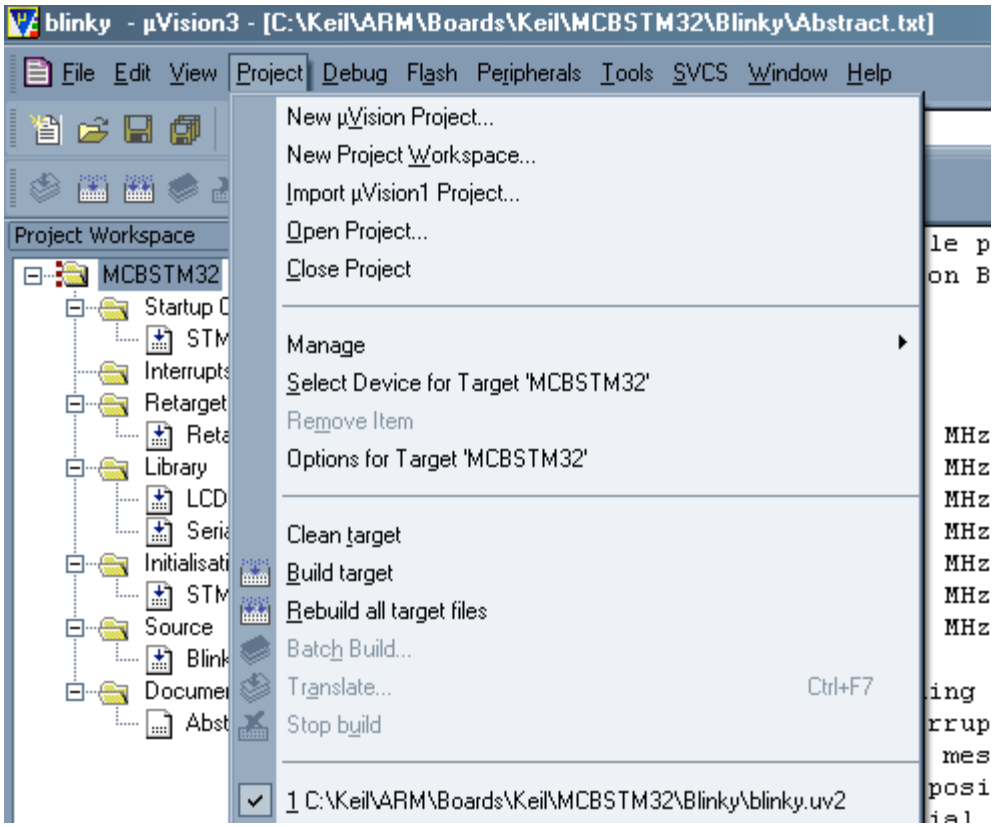
2.1 Το παράδειγμα Blinky

Παρακάτω θα αναλύσουμε τα βήματα που χρειάζονται για να εκτελέσουμε το παράδειγμα Blinky που περιέχεται στο πακέτο λογισμικού της Keil.

a. Πριν τρέξουμε το μVision4 αντιγράφουμε τον φάκελο: **Lab2_Blinky.rar** στην επιφάνεια εργασίας και τον αποσυμπιέζουμε για να μπορέσουμε να τον επεξεργαστούμε.

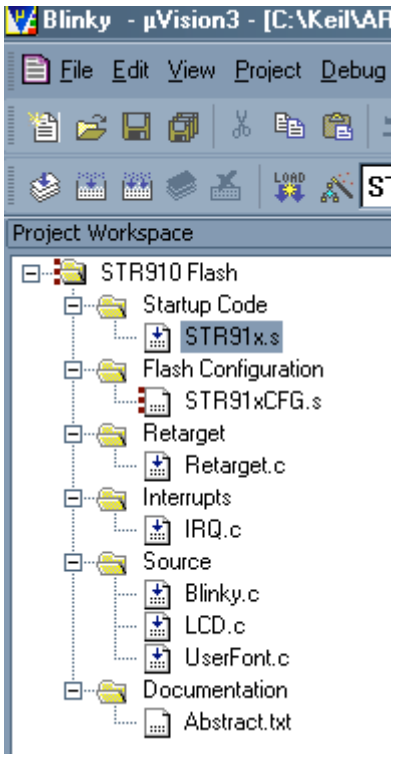
b. Ξεκινάμε το μVision4 από το **Start->All Programs->Keil uVision4** είτε από την συντόμευση στην επιφάνεια εργασίας.

c. Από την επιλογή **Project** επιλέγουμε **Open Project** όπως φαίνεται στο παρακάτω σχήμα**.** Αναζητούμε το σημείο που προηγουμένως αντιγράψαμε τον φάκελο Lab2_Blinky (επιφάνεια εργασίας) και ανοίγουμε το **Blinky.uvproj.**



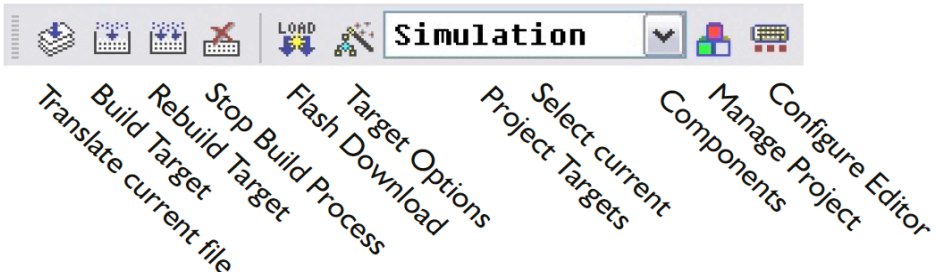
Σχήμα 6: Επιλογή Project στο uVision3

d. Στο **Project Workspace** παρατηρούμε τα διάφορα αρχεία του project (Σχήμα 7). Πατώντας διπλό κλικ στα αρχεία αυτά μπορούμε να τα ανοίξουμε, να δούμε το περιεχόμενο τους και να τα επεξεργαστούμε.



Σχήμα 7: Παράθυρο Project Workspace

e. Χτίζουμε τον κώδικα πηγαίνοντας στο **Build Toolbar.** Βεβαιωθείτε ότι στην επιλογή **Select Target** είναι επιλεγμένο **STR 910 Flash.** Πατάμε το εικονίδιο του **Build Target.**



Σχήμα 8: Build Toolbar

Εάν το χτίσιμο του κώδικα έχει ολοκληρωθεί χωρίς λάθη, στο παράθυρο Build Output θα πρέπει να έχουμε τα αποτελέσματα του παρακάτω σχήματος (Σχήμα 9).

Build Output

```
Build target 'STR910 Flash'
assembling STR91x.s...
compiling Retarget.c...
compiling IRQ.c...
compiling Blinky.c...
compiling LCD.c...
compiling UserFont.c...
linking...
Program Size: Code=2260 RO-data=96 RW-data=4 ZI-data=1292
".\Flash\Blinky.axf" - 0 Error(s), 0 Warning(s).
```

Σχήμα 9: Παράθυρο Build Output μετά από επιτυχές build

2.2 Εκτελώντας το παράδειγμα Blinky στο αναπτυξιακό

Έχοντας επιτυχώς τελειώσει το παραπάνω παράδειγμα, μπορούμε να κατεβάσουμε το **binary Blinky.axf** αρχείο που δημιουργήθηκε, στην πλακέτα. Για να το κάνουμε αυτό πρώτα βεβαιωνόμαστε για τα εξής πράγματα:

- Δεν υπάρχουν πάνω ή κοντά στην πλακέτα άλλα μεταλλικά αντικείμενα ή υγρά.
- Δεν ακουμπάμε με τα χέρια μας την πλακέτα, παρά μόνο τα πλήκτρα και τις θύρες (από πλαστικό) που βρίσκονται σε αυτή.

Αφού βεβαιωθούμε για τα παραπάνω τροφοδοτούμε και συνδέουμε το JTAG στο αναπτυξιακό, κάνοντας τα εξής:

a. Συνδέουμε το USB καλώδιο στην USB θύρα.

b. Το LED Power, χρώματος κόκκινο, ανάβει.

c. Συνδέουμε την συσκευή ULINK2 της Keil, στην JTAG θύρα.

Τώρα είμαστε έτοιμοι να κατεβάσουμε το binary αρχείο στην αναπτυξιακή πλακέτα. Η διαδικασία που ακολουθείται από τα εργαλεία της Keil για να κατεβάσει το Binary στην flash είναι η εξής:

- Erase: Σβήνει τα προηγούμενα περιεχόμενα της flash
- Program: Προγραμματίζει την flash με το νέο binary
- Verify: Πιστοποιεί ότι ο προγραμματισμός της Flash έγινε σωστά.

Τα στάδια αυτά μπορούμε να τα δούμε στο Build Output παράθυρο. Όταν το κατέβασμα τελειώσει επιτυχώς, το Build Output παράθυρο θα μας ενημερώσει αναλόγως, όπως φαίνεται στο παρακάτω σχήμα (Σχήμα 10).

Πατάμε το πλήκτρο **Flash Download** στο **Build Toolbar** για να κατεβάσουμε το binary στη flash.

Build Output

```
Build target 'STR910 Flash'
linking...
Program Size: Code=2260 RO-data=96 RW-data=4 ZI-data=1292
".\Flash\Blinky.axf" - 0 Error(s), 0 Warning(s).
Load "C:\\Keil\\ARM\\Boards\\Keil\\MCBSTR9\\Blinky\\Flash\\Blinky.AXF"
Erase Done.
Programming Done.
Verify OK.
Application running ...
```

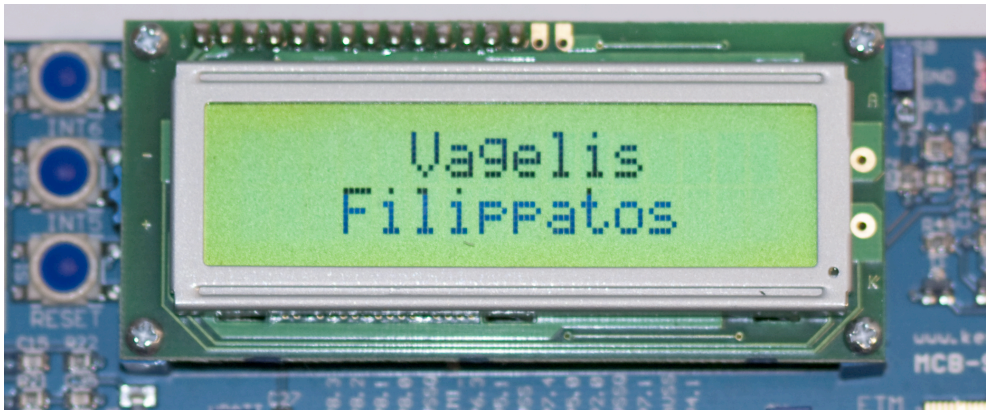
Σχήμα 10: Παράθυρο Build Output μετά από επιτυχές binary κατέβασμα

Στο παράδειγμα που εκτελούμε, ο ρυθμός ανακύλισης των LEDs και η ένδειξη του LCD αυξομειώνεται ανάλογα με την τάση εισόδου του ADC0, δηλαδή ανάλογα με την ρύθμιση του ποτενσιόμετρου POT1.

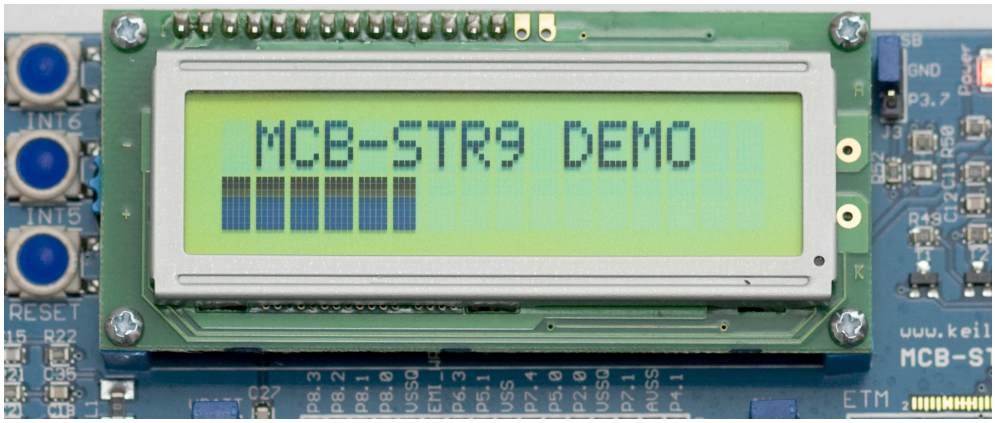
Ερώτημα 19
Χρησιμοποιώντας ένα πολύμετρο βρείτε την σχέση της ανακύλισης των LEDs και της τάσης εξόδου του ποτενσιόμετρου. Όταν αυξάνεται η τάση αυξάνεται ο ρυθμός ανακύλισης ή το αντίθετο συμβαίνει; Ανατρέξτε στο σχηματικό, MCBSTR9_schematic.pdf, για να επιλέξετε σημείο μέτρησης.

2.3 Το παράδειγμα Blinky. Απλή χρήση του LCD.

Θα κάνουμε αλλαγές στον κώδικα του παραδείγματος Blinky ώστε στην πρώτη γραμμή της οθόνης του LCD να τυπωθεί το μικρό μας όνομα και στην δεύτερη το επώνυμο μας, όπως φαίνεται παρακάτω.



Σχήμα 11: Το όνομα μας στο LCD

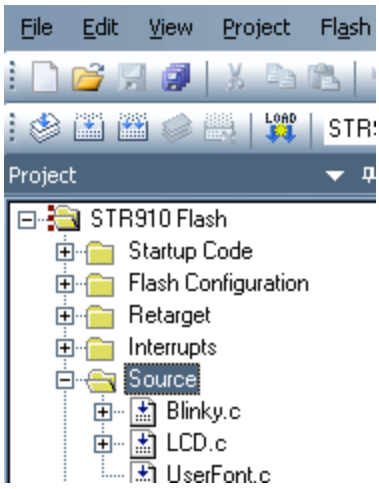


Σχήμα 12: Αρχική ένδειξη του LCD

Για να γράψουμε στο LCD πρέπει να χρησιμοποιήσουμε την συνάρτηση **lcd_print.**

Ερώτημα 20
Ανοίγοντας το αρχείο LCD.c βρείτε και γράψτε τι παραμέτρους δέχεται η lcd_print και τι επιστρέφει.

Ανοίγουμε το Blinky.c το οποίο βρίσκουμε στο Project, στο αριστερό παράθυρο του uVision4 (Σχήμα 13).



Σχήμα 13: Παράθυρο Project

Στην main συνάρτηση του Blinky.c αρχείου, μπορούμε να βρούμε που αλλού αλλά και πώς χρησιμοποιείται η lcd_print και να το χρησιμοποιήσουμε ως παράδειγμα.

Για να γράψουμε το όνομα μας στην LCD οθόνη ακολουθήσουμε τα παρακάτω βήματα:

a. Σβήνουμε τον ατέρμονο βρόχο (while(1)) και ότι αυτός περιέχει. Δεν θέλουμε να υπάρχει πλέον η ανακύλιση στα LED αλλά ούτε να γεμίζει με μαύρα κουτία η δεύτερη γραμμή του LCD. Σβήνοντας τον ατέρμονο βρόχο δεν σταματάμε την λειτουργία του ADC. Απλώς η ένδειξη για την τάση του ποτενσιόμετρου μέσω του Analoge to Digital Converter (ADC) δεν θα απεικονίζεται ούτε στο LCD αλλά ούτε μέσω των LED. Ο ADC θα λειτουργεί κανονικά αλλά εμείς δεν θα διαβάζουμε την τιμή του.

b. Προσθέτουμε την lcd_print συνάρτηση με το όνομα μας. Συμπληρωματικές συναρτήσεις, που υπάρχουν στο αρχείο LCD.c, μπορεί να χρειαστούν.

c. Χτίζουμε και κατεβάζουμε τον κώδικα όπως κάναμε στην 2.1 και 2.2 παράγραφο.

d. Εάν δεν λειτουργεί όπως θέλουμε, τρέχουμε πάλι τα βήματα b και c.

Ερώτημα 21
Εδώ συμπληρώστε τον τελικό κώδικα σας.

3 ΕΡΓΑΣΤΗΡΙΟ 3

♦ **Μέθοδοι απασφαλμάτωσης**

Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαιο "Writing Programs->Debugging Programs"
- **Αρχείο debug_test.c,** μέσα από τον φάκελο **Lab3_Debug.rar**.
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για καταχωρητές.
- **Βιβλίο Θεωρίας Wayne Wolf, "Οι Υπολογιστές ως Συστατικά Στοιχεία".** Κεφάλαιο 4, παράγραφος 7. Ανάπτυξη και Αποσφαλμάτωση.

Εισαγωγή

Στο εργαστήριο αυτό θα ακολουθήσουμε διάφορες μεθόδους απασφαλμάτωσης, χρησιμοποιώντας τον απασφαλματωτή (debugger) του μVision4 σε συνδυασμό με την συσκευή ULINK2. Η συσκευή αυτή είναι ουσιαστικά ένας USB-JTAG μετατροπέας και επιτρέπει στο λογισμικό μVision4 να επικοινωνεί με την JTAG διεπαφή του μικροελεγκτή. Χρησιμοποιεί **JTAG (Joint Test Action Group) εντολές** διεπαφής για να πραγματοποιήσει τα παρακάτω:

- Φόρτωση προγραμμάτων λογισμικού στη πλακέτα
- Απασφαλμάτωσης του τρέχοντος προγράμματος με διάφορες μεθόδους.
- Προγραμματισμό της εσωτερικής μνήμης FLASH του μικροελεγκτή.
- Προγραμματισμό εξωτερικής μνήμης FLASH στην πλακέτα.

Σχήμα 14: ULINK2 USB-JTAG μετατροπέας

Παράλληλα το λογισμικό μVision4 μπορεί να χρησιμοποιηθεί κάποια πλακέτα και μικροελεγκτή, επιλέγοντας το “Use Simulator” στο παράθυρο “Options for Target “STR910 Flash”, όπως φαίνεται στο επόμενο σχήμα. Με την επιλογή αυτή, ο χρήστης μπορεί να δοκιμάσει τον κώδικα του καθώς το μVision4 έχει την δυνατότητα να εξομοιώσει το μεγαλύτερο κομμάτι του μικροελεγκτή.

Σχήμα 15: Επιλογή εξομοίωσης στο uVision4

Παρακάτω, θα χρησιμοποιήσουμε τρεις διαφορετικές μεθόδους απασφαλμάτωσης.

- Άμεση παρακολούθηση των καταχωρητών του μικροελεγκτή.
- Breakpoints.
- Παράθυρα Dissasembly και Register.

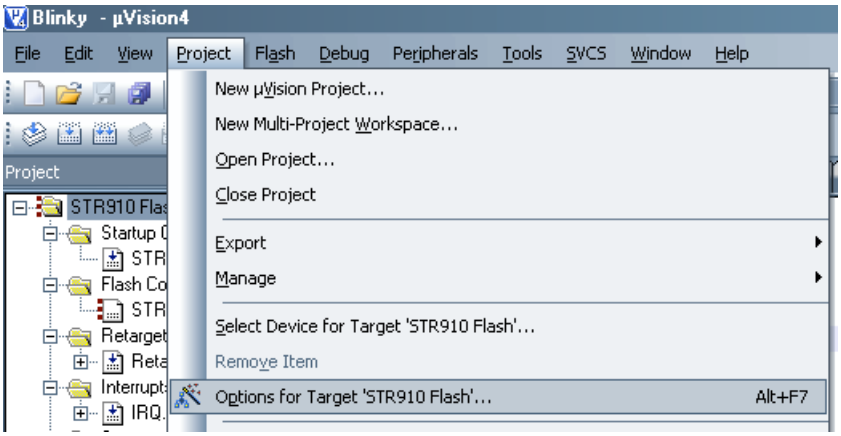
3.1 Ρυθμίσεις uVision για χρήση του αποσφαλματωτή

Ανοίγουμε το **Project debug_test.uvproj** αποσυμπιέζοντας το **Lab3_Debug.rar** αρχείο, που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Ανοίγουμε το αρχείο **debug_test.c**

```
int main (void)
{
 int i, dummy_loop;

 for (i=0; i<100; i++)
  dummy_loop = i;

 /* System Control Unit Set-up */
 SCU->PRR1          =  0x00EE2803;
 SCU->PCGR1         =  0x00EE2803;

 /* Buttons Setup */
 SCU->GPIOIN[3]     = 0x00;       /* P3.5 input - mode 1    */
 SCU->GPIOOUT[3]  &= 0xF3FF;  /* P3.5 input - mode 0    */
 GPIO3->DDR         &= 0xDF;      /* P3.5 direction - input  */
}
```

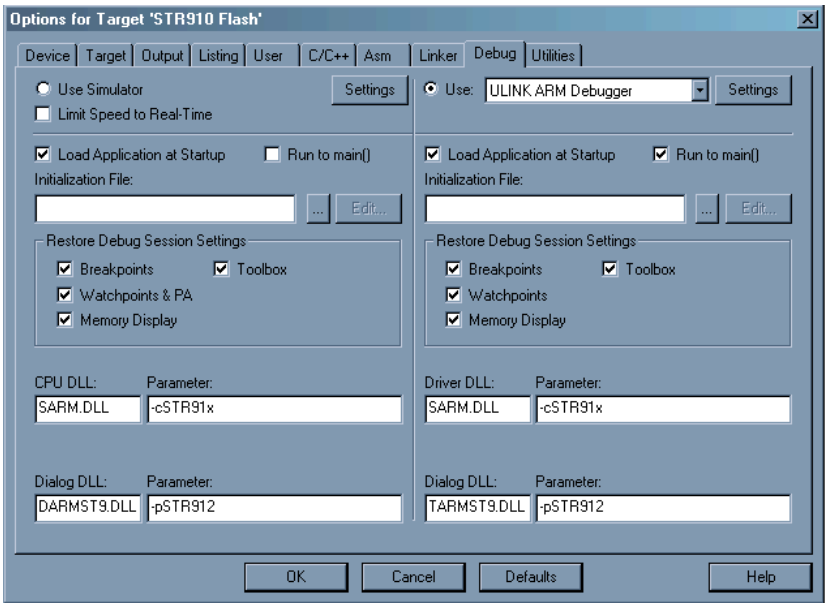
Χτίζουμε και κατεβάζουμε τον κώδικα στο αναπτυξιακό ακολουθώντας τα βήματα των Κεφαλαίων 2.1 και 2.2.

Επιλέγουμε το Options for Target 'STR910 Flash' όπως φαίνεται στο επόμενο σχήμα.



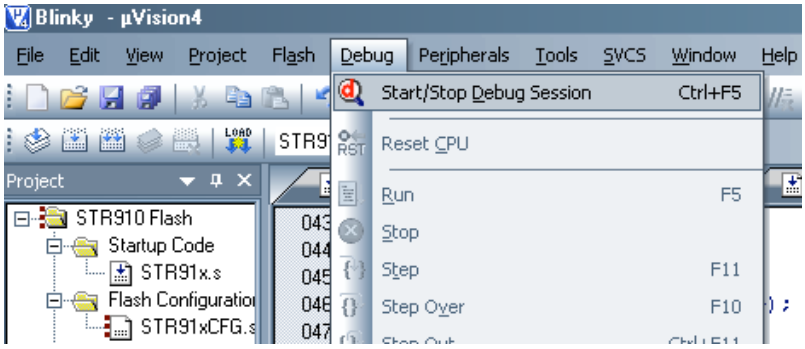
Σχήμα 16: Επιλογές debugger

Από το παράθυρο που αναδύεται επιλέγουμε το tab **Debug** και το ρυθμίζουμε όπως φαίνεται στο επόμενο σχήμα.



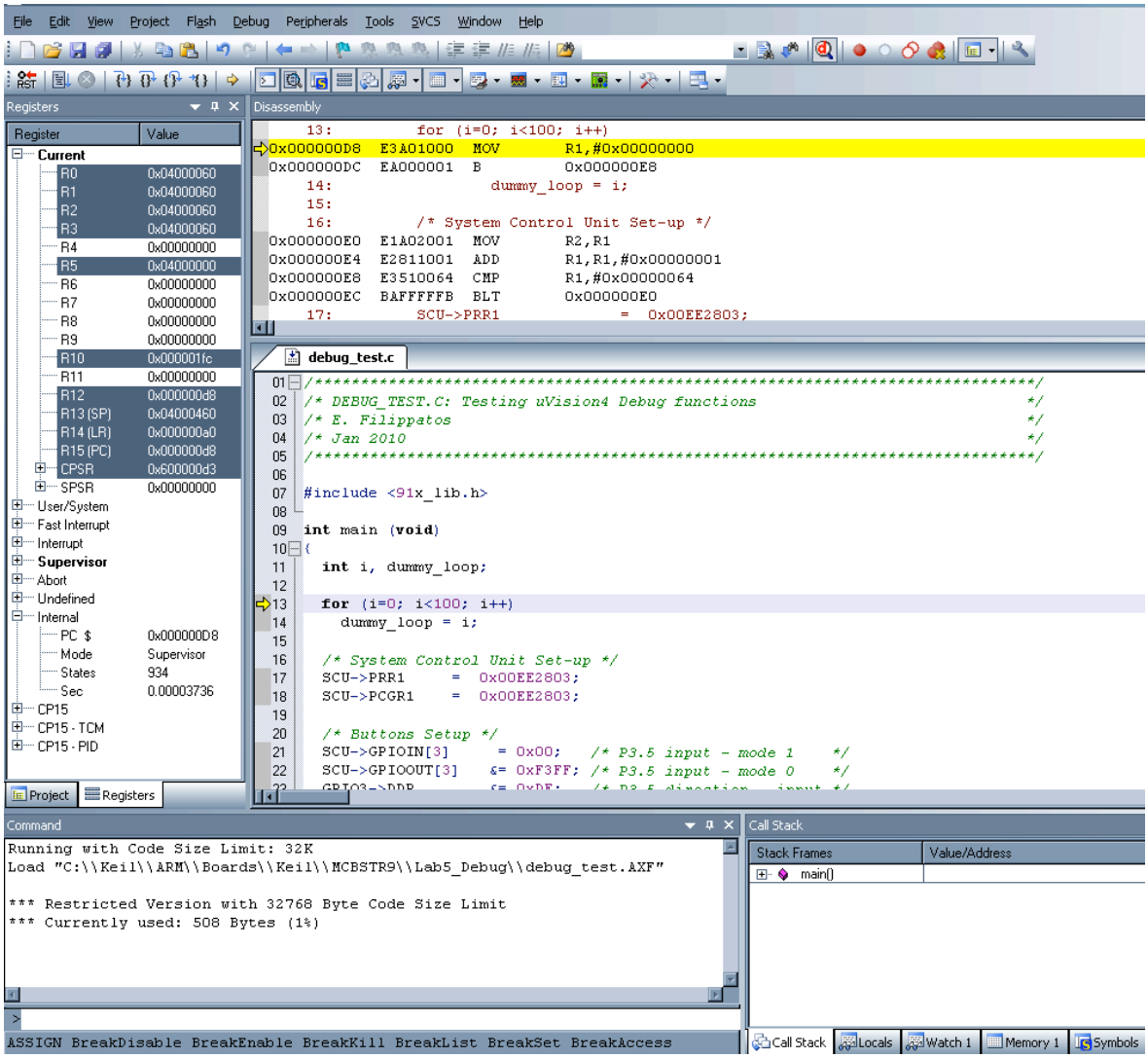
Σχήμα 17: Επιλογές debugger

Κλείνουμε το παράθυρο και επιλέγουμε το **Start/Stop Debug Session** όπως φαίνεται στην παρακάτω εικόνα.



Σχήμα 18: Επιλογή Εκκίνησης Debugger

Πατάμε ΟΚ στο μήνυμα: *EVALUATION MODE, Running with Code Size Limit : 32K*. Ο Debugger ξεκινάει και αφού κάνει ένα reset στον μικροελεγκτή, περιμένει στην πρώτη εντολή της main του προγράμματος μας. Αυτό φαίνεται παρακάτω.



Σχήμα 19: Παράθυρο Debugger μVision4

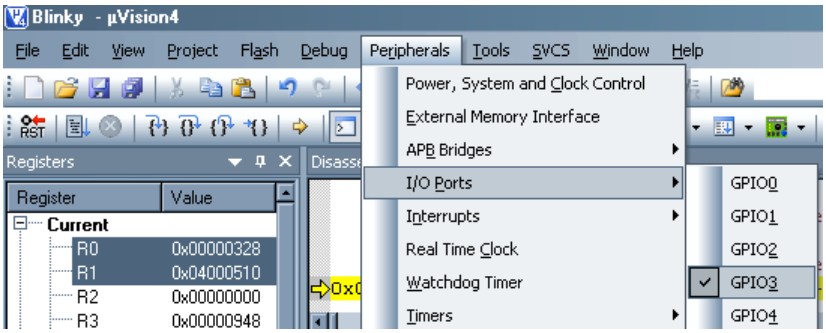
3.2 Παρακολούθηση κατάστασης καταχωρητών

Μέσω της χρήσης του debugger θα προσπαθήσουμε να ελέγξουμε την κατάσταση του GPIO 3.5 στο οποίο είναι συνδεδεμένο το πλήκτρο INT5. Για να μπορέσουμε να παρακολουθούμε την κατάσταση του GPIO 3.5, πρέπει να παρακολουθούμε τον αντίστοιχο καταχωρητή που κρατάει τις καταστάσεις ή αλλιώς τα δεδομένα του GPIO Port 3.

Ερώτημα 22
Σε ποιόν καταχωρητή αποθηκεύεται η κατάσταση του GPIO 3.5;

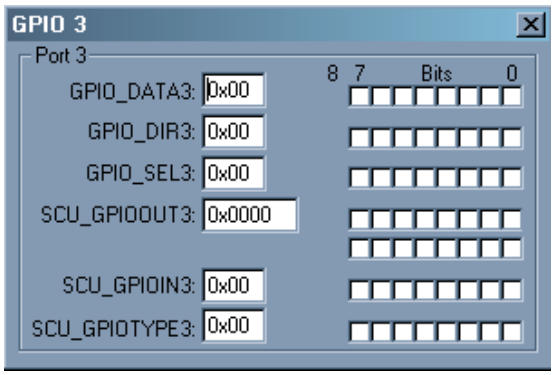
Ο έλεγχος του καταχωρητή μπορεί να γίνει με δύο διαφορετικούς τρόπους.

a. Επιλέγουμε τα παράθυρα των περιφερειακών όπως φαίνεται παρακάτω.



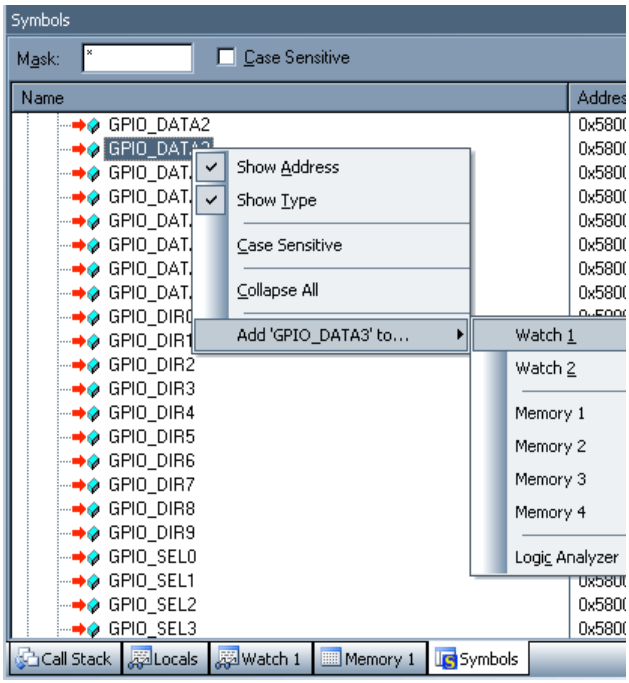
Σχήμα 20: Επιλογή παρακολούθηση κατάστασης GPIO

Το επόμενο παράθυρο θα πρέπει να εμφανιστεί.



Σχήμα 21: Παράθυρο καταχωρητών για το GPIO3

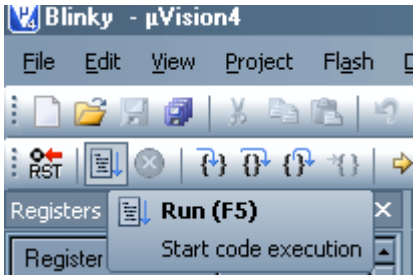
b. Επιλέγουμε να παρακολουθούμε τον καταχωρητή GPIO_DATA3 μέσα από την επιλογή Symbols -> Peripheral Registers όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 22: Επιλογή παρακολούθησης καταχωρητή

Στο παράθυρο Watch 1 θα έχουμε τώρα τον καταχωρητή GPIO_DATA3. Έχοντας και τα δυο παράθυρα παρακολούθησης του καταχωρητή GPIO_DATA3 μπορούμε να παρατηρούμε την το GPIO3.5, αλλά και όλων των GPIO, από 0 έως 7, στο Port 3. Λογικά, **αρχικά το GPIO 3.5 πρέπει να έχει τιμή 1 και κρατώντας πατημένο το πλήκτρο να πάρει την τιμή 0.**

- Ο μικροελεγκτής, με το που ξεκινάμε την διαδικασία του Debug γίνεται reset. Πατάμε το **Run ή το F5** για να ξεκινήσει να εκτελείται ο κώδικας. Η ανακύλιση των LED ξεκινάει και θα πρέπει να αυξομειώνεται καθώς μεταβάλουμε το ποτενσιόμετρο. Δοκιμάστε το.



Σχήμα 23: Εκτέλεση κώδικα μέσω του Debugger

- Πατήστε το **Stop**, που βρίσκεται δίπλα στο Run.

- Παρακολουθήστε την τιμή του GPIO_DATA3. Θα πρέπει να είναι 0x7D.

- Πατήστε πάλι το Run ή το F5.

- Κρατήστε πατημένο το πλήκτρο INT5 στο αναπτυξιακό και παράλληλα σταματήστε την εκτέλεση του κώδικα πατώντας το Stop στο uVision4.

Ερώτημα 23
Ποιά η τιμή του GPIO3.5 στον καταχωρητή GPIO_DATA3 καθώς το πλήκτρο INT5 είναι πατημένο;

3.3 Χρησιμοποίηση των breakpoints

Τα breakpoints μπορούν να οριστούν σε οποιαδήποτε μέρος του κώδικα μας. Η λειτουργία των breakpoint είναι αρκετά απλή. Όταν η εκτέλεση του κώδικα μας φθάσει στο σημείο στο οποίο έχουμε ορίσει το breakpoint και πριν εκτελεστεί ο κώδικας στο σημείο αυτό, τότε η εκτέλεση σταματάει ή αλλιώς κάνει παύση. Μπορούν να οριστούν και breakpoints υπό-συνθήκη

(conditional breakpoints ή αλλιώς watchpoints), τα οποία μπορούν να ελέγχουν εναλλαγή συγκεκριμένων τιμών σε καταχωρητές, σε μεταβλητές του κώδικα μας και άλλα πολλά.

Παρακάτω θα ορίσουμε ένα απλό breakpoint στο σημείο του κώδικα που προγραμματίζουμε τον Peripheral Reset Register 1, δηλαδή στην εντολή: **SCU->PRR1 = 0x00EE2803;**

Ερώτημα 24
Ποιος ο σκοπός της SCU->PRR1 = 0x00EE2803 εντολής

Για να ορίσουμε και να ελέγξουμε το breakpoint ακολουθούμε τα παρακάτω βήματα:

- Σταματάμε την εκτέλεση του κώδικα μας πατώντας το **Stop.**

- Κάνουμε δεξί κλικ στην γραμμή του κώδικα που θέλουμε να ορίσουμε το Breakpoint, στην γραμμή SCU->PRR1 = 0x00EE2803 και επιλέγουμε το **Insert/Remove Breakpoint** ή πατάμε το F9.

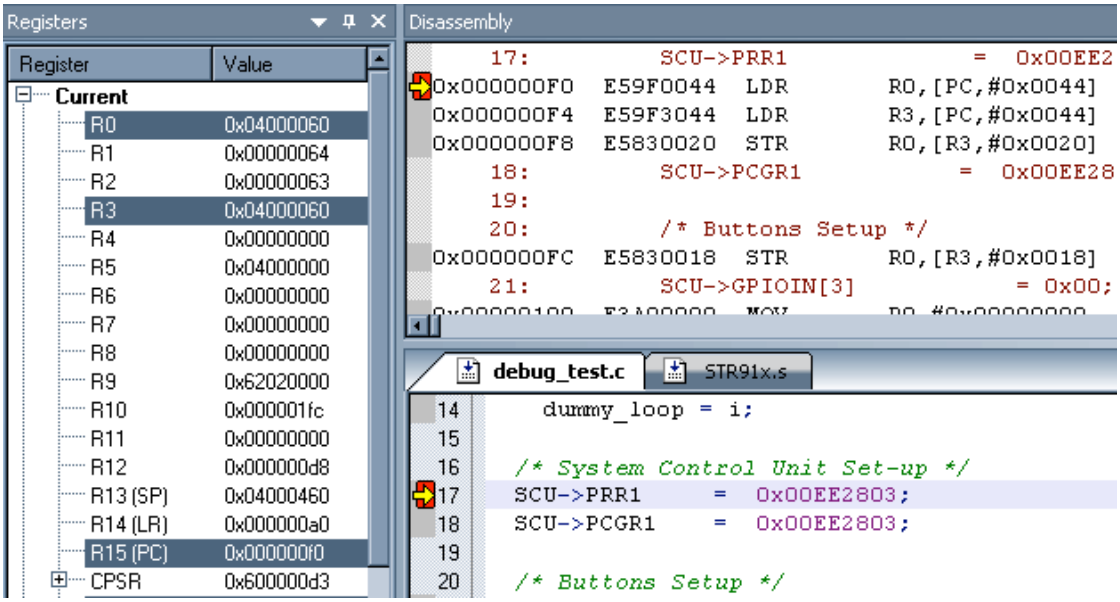
- Πατάμε το **Run** για να ξεκινήσει πάλι η εκτέλεση.
- Παρατηρούμε εάν και σε ποιο σημείο έχει σταματήσει η εκτέλεση του κώδικα μας.

- Πατάμε το πλήκτρο **Rst** και μετά το **Run** για επανεκκίνηση του κώδικα και επανέλεγχο της λειτουργία του breakpoint.

3.4 Χρησιμοποίηση των παραθύρων Disassembly και Registers

Το παράθυρο Dissasembly μας δείχνει τον C κώδικα μας και την αντίστοιχη μετάφραση του σε assembly ή μόνο την assembly εάν εκτελούμε απλή assembly και όχι C. Το παράθυρο Registers μας δείχνει τους καταχωρητές του ARM CPU που χρησιμοποιούνται για την επεξεργασία των δεδομένων και για την εκτέλεση του κώδικα μας. Για αυτούς θα μιλήσουμε αργότερα. Τώρα θα προσπαθήσουμε να παρατηρήσουμε το παράθυρο Dissasembly και παράλληλα την λειτουργία του παραθύρου Registers.

Εκτελούμε τον κώδικα μας με τις ρυθμίσεις της προηγούμενης παραγράφου. Η εκτέλεση, λόγο του breakpoint, σταματάει πριν εκτελεστεί η εντολή: SCU->PRR1 = 0x00EE2803.



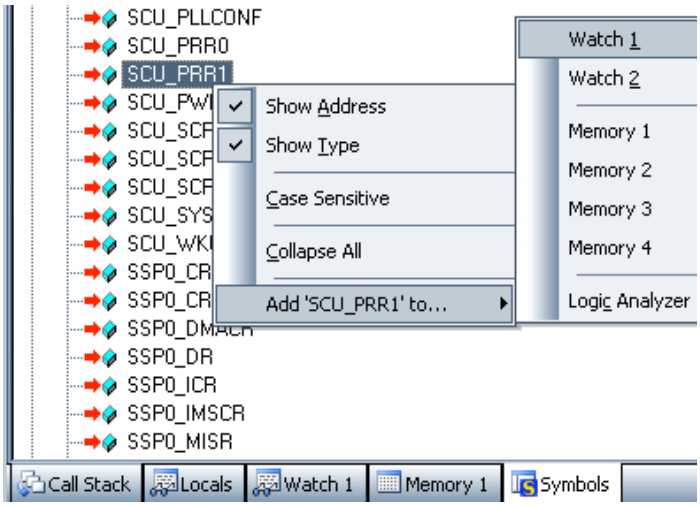
Σχήμα 24: Dissasembly και Registers

Στο προηγούμενο σχήμα παρατηρούμε δύο κίτρινα βελάκια. Τα βελάκια αυτά υποδεικνύουν ποιά είναι η επόμενη εντολή που ο CPU θα εκτελέσει. Επίσης παρατηρούμε ότι βρίσκονται πάνω στα κόκκινα κουτιά του breakpoint γιατί η εκτέλεση έχει σταματήσει στο breakpoint που έχουμε ορίσει, δηλαδή στην εντολή SCU->PRR1 = 0x00EE2803.

Ερώτημα 25
Σε ποια θέση μνήμης του MCU βρίσκεται ο καταχωρητής SCU_PRR1;

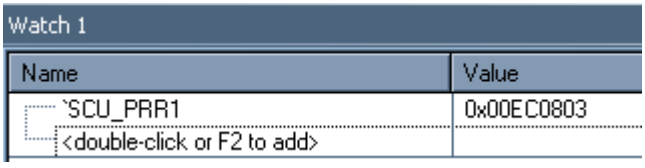
Η παραπάνω εντολή της C, στην οποία ο CPU θα καταχωρήσει στην θέση μνήμης του SCU->PRR1 την τιμή 0x00EE2803, έχει μεταφραστεί σε μια σειρά εντολών assembly. Για να δούμε ποιες εντολές είναι αυτές πρέπει να παρακολουθήσουμε τα παράθυρα Disassembly και Registers. Επίσης πρέπει να ορίσουμε τον SCU_PRR1 καταχωρητή στο Watch 1 παράθυρο.

Για να θέσουμε τον καταχωρητή SCU_PRR1 στο Watch 1 παράθυρο ακολουθούμε την διαδικασία της παραγράφου 5.2. Πηγαίνουμε στο παράθυρο Symbols και από την επιλογή Peripheral Registers θέτουμε τον καταχωρητή SCU_PRR1 στο παράθυρο Watch 1 όπως φαίνεται παρακάτω:



Σχήμα 25: Καταχωρητής στο παράθυρο Watch 1

Η αρχική τιμή του SCU_PRR1, είναι αυτή που φαίνεται στο Watch 1.



Σχήμα 26: Παράθυρο Watch 1 και αρχική τιμή του SCU_PRR1

Λόγω του breakpoint, η εκτέλεση του κώδικα μας έχει σταματήσει ακριβώς πριν εκτελεστεί η εντολή: SCU->PRR1 = 0x00EE2803. Η μετάφραση της εντολής αυτής από C σε assembly αποτελείται από τις παρακάτω τρεις εντολές, που διακρίνονται στο Disassembly παράθυρο.

```
0x000000F0  E59F0044  LDR     R0,[PC,#0x0044]
0x000000F4  E59F3044  LDR     R3,[PC,#0x0044]
0x000000F8  E5830020  STR     R0,[R3,#0x0020]
```

Η πρώτη στήλη περιέχει την θέση του Μετρητή Προγράμματος – Program Counter (PC). Ο Program Counter αυξάνεται κατά 4bytes (32bit) για κάθε μια εντολή που εκτελείται εκτός και εάν εκτελεστεί κάποιο παρακλάδι (Branch). Την τρέχουσα θέση του Program Counter (PC) μπορούμε να την δούμε από τον καταχωρητή R15 (PC) στο παράθυρο των καταχωρητών (Registers).

Ερώτημα 26
Σε ποια θέση βρίσκεται ο PC όταν η εκτέλεση σταματήσει λόγω του breakpoint που έχουμε θέσει;

Μπορούμε επίσης να βρούμε την τρέχουσα τιμή του PC χρησιμοποιώντας το Disassembly παράθυρο και κοιτώντας την πρώτη στήλη στην οποία το κίτρινο βέλος δείχνει. Παρατηρώντας το Σχήμα 24 μπορούμε να δούμε ότι ο PC, το κίτρινο βέλος, έχει σταματήσει στο breakpoint, στο κόκκινο κουτί του Disassembly παραθύρου. Η διεύθυνση του PC την στιγμή αυτή είναι η 0xF0. Η ίδια διεύθυνση υπάρχει και στον καταχωρητή R15 (PC) στο παράθυρο Registers.

Η assembly εντολή που θα εκτελεστεί στην διεύθυνση 0x214 είναι η:

```
LDR     R0,[PC,#0x0044]
```

Η οποία σημαίνει:
Φόρτωσε τον καταχωρητή R0 με ένα word (4bytes ή 32bits) από την διεύθυνση PC + 0x0044 + 8.

Πηγαίνοντας στην διεύθυνση PC + 0x0048 + 8= 0xF0 0x0044 + 8 = 0x13C θα βρούμε την εξής assembly εντολή:

0x0000013C 00EE2803 DD 0x00EE2803

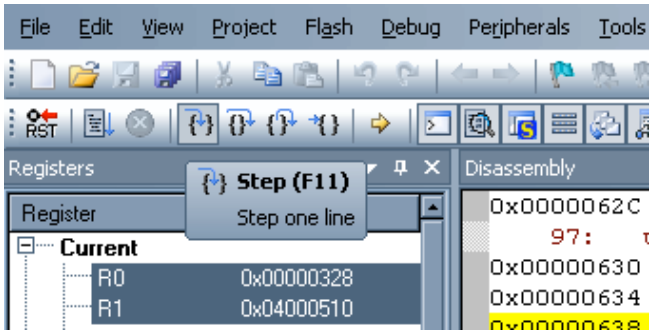
Οι ARM assembly εντολές DD ορίζουν ένα literal pool. Το literal pool είναι μια περιοχή στην μνήμη που χρησιμοποιείται για να αποθηκευτούν σταθερές τιμές, όπως είναι η τιμή 0x00EE2803 που θα καταχωρηθεί στον SCU_PRR1.

Οπότε η εντολή:

LDR R0,[PC,#0x00044]

πηγαίνει στην περιοχή μνήμης εκείνη, που προκύπτει από την πρόσθεση PC + 0x0044 + 8, στην οποία έχει αποθηκευτεί η τιμή 0x00EE2803. Την τιμή αυτή την φορτώνει στον καταχωρητή R0.

Παρακολουθώντας τα δεδομένα του καταχωρητή R0 πατήστε το πλήκτρο **Step (F11)**, όπως φαίνεται στο επόμενο σχήμα. Ο CPU θα εκτελέσει ένα βήμα, δηλαδή θα εκτελέσει μόνο την τρέχουσα εντολή του PC και θα κάνει παύση.



Σχήμα 27: Εκτέλεση Step

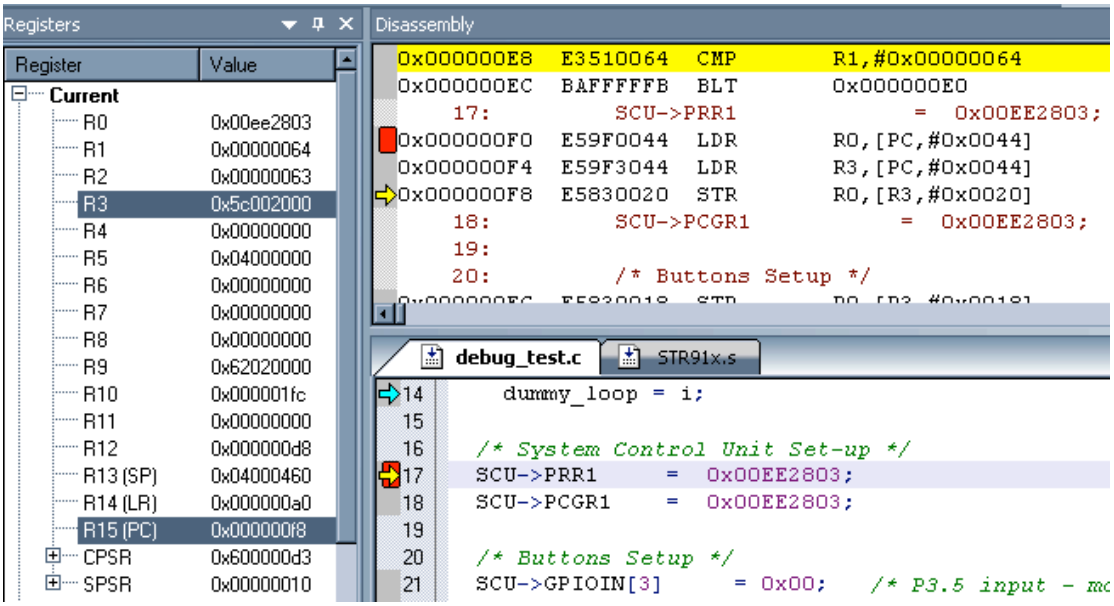
Ερώτημα 27
Ποια εντολή θα εκτελεστεί όταν πατηθεί το Step (F11); Σε ποια θέση βρισκόταν ο PC και σε ποια θέση βρίσκεται τώρα;

Ερώτημα 28
Τι τιμή έχει τώρα ο R0; Ήταν αυτό αναμενόμενο; Δικαιολογήστε το.

Ερώτημα 29
Ποια η εντολή που υπάρχει στην νέα θέση του PC; **Ποια η λειτουργία της;**

Ξαναπατάμε το πλήκτρο Step ή το F11. Η τρέχουσα εντολή στην διεύθυνση του PC εκτελείται.

Ερώτημα 30
Ποια είναι τα αποτελέσματα της νέας εντολής που εκτελέστηκε; Που βλέπουμε τα αποτελέσματα αυτά;



Σχήμα 28: Παράθυρα στην επιλογή Debug

Τώρα ο Program Counter πρέπει να βρίσκεται στην διεύθυνση 0xF8. Τα debug παράθυρα πρέπει να είναι όπως φαίνεται παρακάτω.

Παρατηρούμε ότι στο παράθυρο με τον C κώδικα το κίτρινο βέλος έχει παραμείνει στην εντολή SCU->PRR1 = 0x00EE2803, παρόλο που εκτελέσαμε 2 εντολές assembly. Αυτό συμβαίνει γιατί η εντολή αυτή

χρειάζεται 3 assembly εντολές να πραγματοποιηθεί και εμείς έχουμε εκτελέσει τις 2 από αυτές. Όταν λογικά εκτελέσουμε και την επόμενη και ο PC φτάσει στην διεύθυνση 0xFC τότε η εντολή θα έχει πλήρως εκτελεστεί.

Στο παράθυρο Registers παρατηρούμε ότι ο R0 έχει την τιμή 0x00EE2803. Την απέκτησε όταν εκτελέστηκε ή εντολή LDR R0,[PC,#0x0044] στην διεύθυνση 0xF0. Αντίστοιχα ο R1 έχει την τιμή 0x5C002000 και την απέκτησε όταν εκτελέστηκε η εντολή LDR R1,[PC,#0x0044].

Η assembly εντολή στην διεύθυνση 0xF8, στην διεύθυνση στην οποία πρέπει να βρίσκεται τώρα ο PC, είναι η εξής:

STR R0,[R1,#0x0020]

Η λειτουργία της είναι να αποθηκεύσει (STR = store) τα δεδομένα του καταχωρητή R0 στην διεύθυνση που υπάρχει στα περιεχόμενα του R1 συν 0x20.

Ερώτημα 31
Ποια διεύθυνση τελικά προκύπτει από το [R1, #0x0020]. **Ποια διεύθυνση καταχωρητή του MCU είναι αυτή;**

Ερώτημα 32
Τι αποτέλεσμα θα περιμέναμε με την εκτέλεση της STR R0,[R1,#0x0020] εντολής;

Πατάμε για τελευταία φορά το F11 ώστε να εκτελεστεί η παραπάνω εντολή;

Ερώτημα 33
Πώς μπορούμε να δούμε εάν ο καταχωρητής SCU_PRR1 πήρε τελικά την τιμή που περιμέναμε;

4 ΕΡΓΑΣΤΗΡΙΟ 4

- **Μοντέλο προγραμματισμού του ARM**
- **Εισαγωγή στην assembly του ARM**

Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαιο "Writing Programs".
- **Αρχείο ARM.pdf,** κεφάλαιο A2, Programmers' Model
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για καταχωρητές.
- **Βιβλίο Θεωρίας Wayne Wolf, "Οι Υπολογιστές ως Συστατικά Στοιχεία".** Κεφάλαιο 3, παράγραφος 2.4. Διακοπές.

Εισαγωγή

Στο εργαστήριο αυτό θα αναλύσουμε τον μοντέλο προγραμματισμού του ARM. Θα μας είναι χρήσιμο για την περαιτέρω κατανόηση και εφαρμογή των εργαστηρίων. Θα αναφέρουμε τα βασικά για τους καταχωρητές γενικού και ειδικού σκοπού. Οι καταχωρητές αυτοί χρησιμοποιούνται απο το CPU για την διαδικασία εκτέλεσης ενός προγράμματος.

Στην συνέχεια θα εκτελέσουμε ένα απλό πρόγραμμα και με την βοήθεια του debugger θα προσπαθήσουμε να κατανοήσουμε την ροή εκτέλεσης του προγράμματος και την χρήση των καταχωρητών γενικού σκοπού.

4.1 Τύποι δεδομένων

Ο ARM υποστηρίζει τους εξής τύπους δεδομένων:

Byte: 8 Bits
Halfword 16 bits
Word 32 Bits

4.2 Καταχωρητές προγραμματισμού

Ο ARM έχει συνολικά 31 καταχωρητές γενικού σκοπού. Σε οποιαδήποτε στιγμή 17 από αυτούς είναι προσπελάσιμοι. Οι υπόλοιποι χρησιμοποιούνται για να αυξήσουν την ταχύτητα εκτέλεσης των **Εξαιρέσεων (exceptions)**. Από το Σχήμα 29 μπορείτε να δείτε ποιες καταστάσεις θεωρούνται Εξαιρέσεις (exceptions). Έτσι ο προγραμματιστής μπορεί να χρησιμοποιήσει οποιοδήποτε από τους 17 προσπελάσιμους καταχωρητές. Τρεις από τους προσπελάσιμους καταχωρητές, ο R13, R14 και R15, έχουν ειδικό ρόλο.

Stack Pointer (R13): Ο καταχωρητής αυτός καταχωρεί τοπικές μεταβλητές συναρτήσεων με την μέθοδο LIFO (Last In First Out – τα τελευταία δεδομένα που μπαίνουν βγαίνουν πρώτα). Χρησιμοποιείται από τις εντολές PUSH, για να εισχωρησεί στην κορυφή της στοίβας (Stack) ένα νέο στοιχείο και POP για να μετακινήσει ένα στοιχείο από την κορυφή.

Link Register (R14): Ο καταχωρητής αυτός κρατάει την διεύθυνση της επόμενης εντολής μετά από μια Branch and Link (Bl ή BLX) εντολή, η οποία είναι ή εντολή που χρησιμοποιείται για να καλεστεί μια υπορουτίνα. Με άλλα λόγια κρατάει την διεύθυνση επιστροφής από την υπορουτίνα. Όταν η υπορουτίνα πρέπει να επιστρέψει, η τιμή του LR αντιγράφεται στον Program Counter. Επίσης χρησιμοποιείται για να κρατάει την διεύθυνση επιστροφής σε περίπτωση που το σύστημα μπει σε κάποια κατάσταση εξαίρεσης. Τις υπόλοιπες φορές μπορεί να χρησιμοποιηθεί σαν καταχωρητής γενικού σκοπού.

Program counter (R15): Κρατάει την διεύθυνση της επόμενης εντολής που θα εκτελεστεί.

CPSR (Current Program Status Register): Κρατάει την τρέχουσα κατάσταση λειτουργίας του CPU όπως αυτές αναφέρονται στην επόμενη παράγραφο.

SPSR (Saved Program Status Register): Χρησιμοποιείται στις εξαιρέσεις και κρατάει την τιμή του CPSR πριν το σύστημα εξυπηρετήσει μια εξαίρεση. Ουσιαστικά χρησιμοποιείται για να σώζει ποια ήταν η προηγούμενη κατάσταση λειτουργίας της CPU ώστε να επιστρέψει σε αυτήν μετά την εξυπηρέτηση της εξαίρεσης.

4.3 Καταστάσεις λειτουργίας

Ο ARM έχει επτά καταστάσεις λειτουργίας συν μια ειδική λειτουργία λογισμικού που ονομάζεται THUMB που μπορεί να χρησιμοποιηθεί παράλληλα με οποιαδήποτε άλλη από τις επτά καταστάσεις. Οι διαφορετικοί τρόποι λειτουργίας παρέχουν έξτρα χαρακτηριστικά για ομαλή λειτουργία του συστήματος και εκτέλεση διαφορετικών λειτουργιών του CPU ανάλογα με την κατάσταση.

	ΠΡΟΝΟΜΙΟΥΧΕΣ ΚΑΤΑΣΤΑΣΕΙΣ					
		ΕΞΑΙΡΕΣΕΙΣ				
User	**System**	**FIQ**	**IRQ**	**SVC**	**Undef**	**Abort**
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5

R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8_fiq	R8	R8	R8	R8
R9	R9	R9_fiq	R9	R9	R9	R9
R10	R10	R10_fiq	R10	R10	R10	R10
R11	R11	R11_fiq	R11	R11	R11	R11
R12	R12	R12_fiq	R12	R12	R12	R12
R13/SP	R13/SP	R13_fiq	R13_irq	R13_svc	R13_undef	R13_abort
R14/LR	R14/LR	R14_fiq	R14_irq	R14_svc	R14_undef	R14_abort
R15/PC	R15/PC	R15/PC	R15/PC	R15/PC	R15/PC	R15/PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
-	-	SPSR_fiq	SPSR_irq	SPSR_svc	SPSR_undef	SPSR_abort

Σχήμα 29: Πίνακας καταχωρητών ARM

Υπάρχουν 17 καταχωρητές σε κανονική λειτουργία (User ή System). Στις υπόλοιπες καταστάσεις λειτουργίας υπάρχει ένας ακόμα, ο SPSR. Σε privileged κατάσταση, οι Exception καταστάσεις σύν την System (FIQ, ISQ, Supervisor, Undefined και Abort), οι καταχωρητές R8-R14 για το FIQ και R13-R14 για τις υπόλοιπες καταστάσεις, είναι διαφορετικοί από τους αντίστοιχους σε κανονική λειτουργία. Όταν ο CPU μπει σε κάποια privileged κατάσταση λειτουργίας τότε οι διαφορετικοί καταχωρητές (οι σκιασμένοι στο Σχήμα 29), εναλλάσσονται με τους αντιστοίχους χωρίς να χάσουν τα δεδομένα τους οι πρώτοι. Έτσι για παράδειγμα εάν το σύστημα είναι σε κανονική λειτουργία (user mode) και εκτελέσει την εντολή MOV R13, #0x01, ο καταχωρητής R13 θα πάρει την τιμή 0x01. Εάν αμέσως μετά το σύστημα μπει σε κατάσταση IRQ και εκτελεστεί η εντολή MOV R13, #0x02, τότε η τιμή 0x02 θα γραφτεί στον R13_irq και όχι στον R13. Όταν το σύστημα επανέλθει σε κανονική λειτουργία ο R13 θα έχει ακόμα την αρχική τιμή 0x01.

Κατάσταση	Χρήση	CPSR
User	Κανονική εκτέλεση προγράμματος	0x10

FIQ	Fast Interrupt. Εξυπηρέτηση γρήγορων διακοπών	0x11
IRQ	Εξυπηρέτηση απλών διακοπών	0x12
Supervisor	Reset, Εκκίνηση και εξυπηρέτηση διακοπών λογισμικού (software interrupts)	0x13
Abort	Διαδικασίες λαθών μνήμης	0x17
Undefined	Εξυπηρέτηση μη ορισμένων εντολών	0x1B
System	Προνομιούχες διαδικασίες συστήματος	0x1F

Σχήμα 30: Πίνακας καταστάσεων ARM

4.4 Εξαιρέσεις (Exceptions)

Εξαιρέσεις μπορούν να δημιουργηθούν από εσωτερικές ή και εξωτερικές πηγές, όπως μια διακοπή μέσω του USB ή μέσω της UART. Η προηγούμενη κατάσταση πρίν την εξυπηρέτηση της εξάιρεσης σώζεται ώστε το σύστημα να μπορεί να επανέλθει σε κανονική λειτουργία. Επειδή παραπάνω από μια εξαίρεση μπορεί να δημιουργηθεί την ίδια στιγμή, υπάρχουν προτεραιότητες. Στον παρακάτω πίνακα μπορούμε να δούμε τις εξαιρέσεις του ARM και τις αντίστοιχες διευθύνσεις τους.

Εξαίρεση	**Κατάσταση λειτουργίας**	**Vector Interrupt Enable**	**Κανονική διεύθυνση**
Reset	Supervisor		0x00000000
Undefined	Undefined		0x00000004
Software Interrupt	Supervisor		0x00000008
Instruction Fetch Abort	Abort		0x0000000C
Data Access Memory Abort	Abort		0x00000010
IRQ	IRQ	0	0x00000018
		1	Χρήστης
FIQ	FIQ	0	0x0000001C
		1	Χρήστης

Σχήμα 31: Πίνακας Εξαιρέσεων ARM

Οι εξαιρέσεις IRQ προγραμματίζονται σαν **Vectored Interrupts**, θέτοντας το bit VE = 1. Το bit αυτό μπορούμε να το θέσουμε, προγραμματίζοντας τον

Control καταχωρητή (Register 1) του **CP15 coprocessor**. Ο CP15 είναι υπεύθυνος για το MMU (memory management unit) , δηλαδή για την μονάδα ελέγχου της μνήμης, την cache και άλλα. Περιέχει 16 καταχωρητές και μπορεί να προγραμματιστεί με συγκεκριμένες εντολές όπως η MCR για εγγραφή ή η MRC για διάβασμα.

4.5 Παρατηρώντας την εκτέλεση ενός προγράμματος

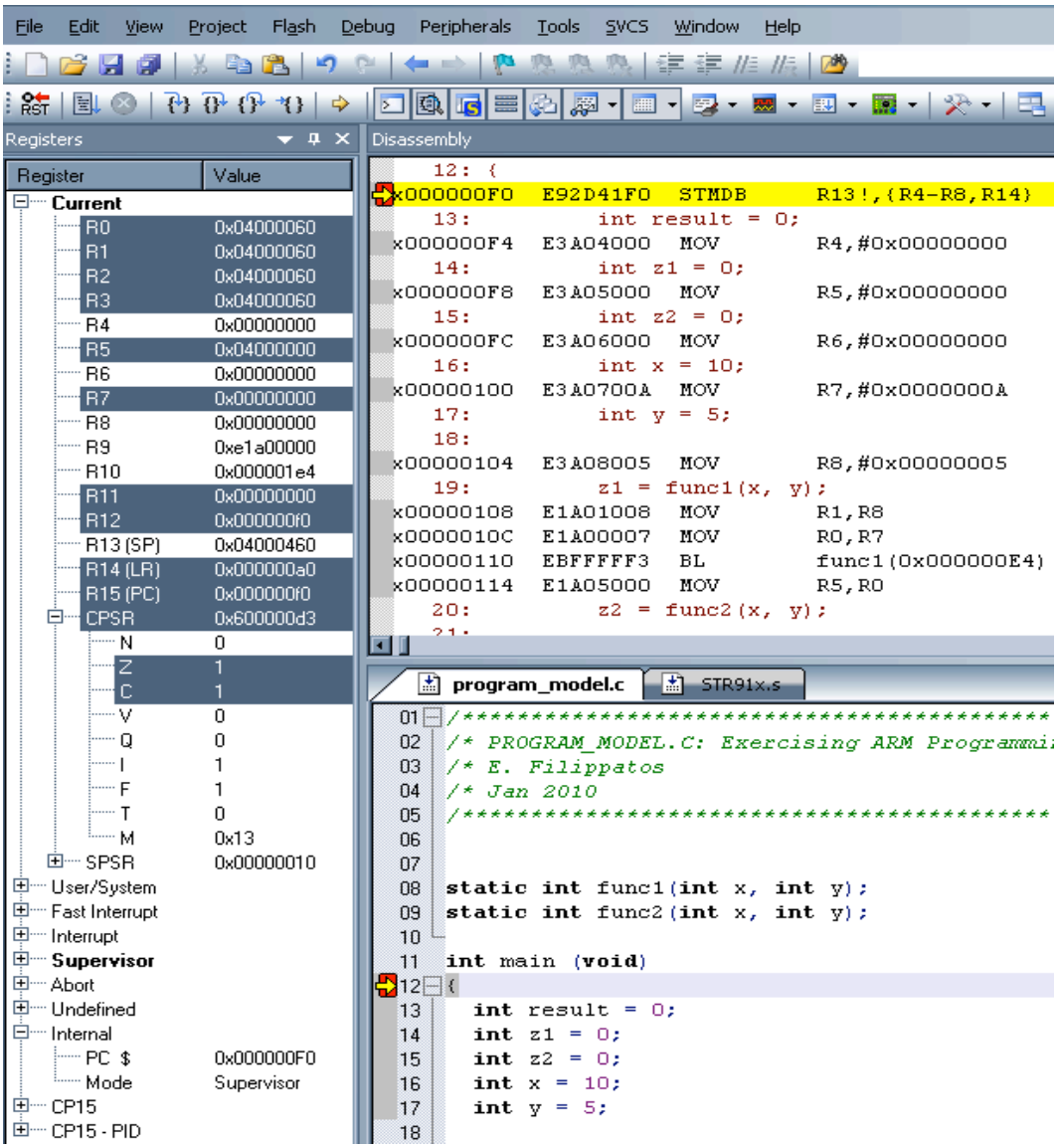
Ανοίγουμε το **Project Lab4_Program_Model.uvproj** αποσυμπιέζοντας το **Lab4_Program_Model.rar** αρχείο, που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Ανοίγουμε το αρχείο **program_model.c.**

```
static int func1(int x, int y);
static int func2(int x, int y);

int main (void)
{
        int result = 0;
        int z1 = 0;
        int z2 = 0;
        int x = 10;
        int y = 5;

        z1 = func1(x, y);
        z2 = func2(x, y);

        result = z1 - z2;

        return result;
}

static int func1(int x, int y)
{
        int result;
        result = x + y;
        return result;
}

static int func2(int x, int y)
{
        int result;
        result = x - y;
        return result;
}
```

a. Χτίζουμε και κατεβάζουμε τον κώδικα στο αναπτυξιακό ακολουθώντας τα βήματα των Κεφαλαίων 2.1 και 2.2.

b. Στα Options for Target 'Target 1' ελέγχουμε ότι το στο tab Debug είναι επιλεγμένο το Use: ULINK ARM Debugger και όχι το Use Simulation.

c. Ξεκινάμε το debugger και τον ρυθμίζουμε ώστε να φαίνονται τα παράθυρα Disassembly, Registers και ο κώδικας program_model.c, όπως φαίνεται παρακάτω.



Σχήμα 32: Παράθυρα στην διαδικασία αποσφαλμάτωσης

d. Προσθέτουμε ένα breakpoint στην αρχή της main όπως φαίνεται στο προηγούμενο σχήμα.

e. Πατάμε το Run. Η εκτέλεση θα σταματήσει. Οι εντολές που θα εκτελεστούν πρώτα φαίνονται στο παράθυρο disassembly.

f. Ο κώδικας την στιγμή αυτή βρίσκεται στην πρώτη αγκύλη της main. Παρ' όλ' αυτά ο CPU θα εκτελέσει την πρώτη του εντολή.

0x000000F0 E92D41F0 STMDB R13!,{R4-R8,R14}

Η εντολή αυτή αποθηκεύει τα περιεχόμενα πολλαπλών καταχωρητών στον καταχωρητή SP (Stack Pointer) , δηλαδή στην στοίβα (Stack). Είναι ουσιαστικά μια εντολή PUSH.

g. Πατάμε το Step ή το F11 για να εκτελείται ο κώδικας μας με βήματα, μια assembly εντολή σε κάθε βήμα. Για κάθε πάτημα του F11 διαβάζουμε τα παρακάτω και παρατηρούμε τους Registers. Πρέπει να γνωρίζουμε ότι ορισμένες εντολές στην C μπορεί να χρειάζονται παραπάνω από μια assembly εντολή για να ολοκληρωθούν.

h.

0x000000F4 E3A04000 MOV R4,#0x00000000

Μετακινεί στον R4 την τιμή 0x00000000.

Παρατηρούμε τα δεδομένα του R4.

Πατάμε το F11 και ελέγχουμε την νεα τιμή του R4.

i.

0x000000F8 E3A05000 MOV R5,#0x00000000

Μετακινεί στον R5 την τιμή 0x00000000.

Παρατηρούμε τα δεδομένα του R5.

Πατάμε το F11 και ελέγχουμε την νεα τιμή του R5.

j.

0x000000FC E3A06000 MOV R6,#0x00000000

Μετακινεί στον R6 την τιμή 0x00000000.

Παρατηρούμε τα δεδομένα του R6.

Πατάμε το F11 και ελέγχουμε την νεα τιμή του R6

k.

0x00000100 E3A0700A MOV R7,#0x0000000A

Μετακινεί στον R7 την τιμή 0x00000000.

Παρατηρούμε τα δεδομένα του R7.

Πατάμε το F11 και ελέγχουμε την νεα τιμή του R7.

l.

0x00000104 E3A08005 MOV R8,#0x00000005

Μετακινεί στον R8 την τιμή 0x00000000.

Παρατηρούμε τα δεδομένα του R8.

Πατάμε το F11 και ελέγχουμε την νεα τιμή του R8.

Ερώτημα 34

Στο σημείο αυτό, ποιό το περιεχόμενο των καταχωρητών R4-R8;
Τί ουσιαστικά έχει αποθηκευτεί στου καταχωρητές αυτούς;

ΤΙΜΗ	ΕΡΜΗΝΕΙΑ
R4:	
R5:	
R6:	
R7:	
R8:	

m.

0x00000104 E3A08005 MOV R8,#0x00000005

Μετακινεί στον R8 την τιμή 0x00000000.

Παρατηρούμε τα δεδομένα του R8.

Πατάμε το F11 και ελέγχουμε την νεα τιμή του R8.

n.

0x00000108 E1A01008 MOV R1,R8

Μετακινεί στον R1 τα περιεχόμενα του R8.

Παρατηρούμε τα δεδομένα του R1.

Πατάμε το F11 και ελέγχουμε την νεα τιμή του R1.

o.

0x0000010C E1A00007 MOV R0,R7

Μετακινεί στον R0 τα περιεχόμενα του R7.

Παρατηρούμε τα δεδομένα του R0.

Πατάμε το F11 και ελέγχουμε την νεα τιμή του R0.

Ερώτημα 35
Γιατί γίνονται οι δύο τελευταίες μετακινήσεις;

p.

0x00000110 EBFFFFF3 BL func1(0x000000E4)

Παρατηρούμε τα δεδομένα των LR, PC.

Branch with Link: Αποθηκεύει στον καταχωρητή R14 (LR - Link Register) την διεύθυνση της εντολής που υπάρχει μετά το Branch και μεταπηδάει στην διεύθηνση 0x000000E4 για να εκτελέσει τον κώδικα της func1.
Μετά την ολοκληρωση των εντολών που υπάρχουν στο Branch μια μετακίνηση της τιμής του LR στο Programm Counter (PC ή R15) θα επαναφέρει την εκτέλεση στην εντολή αμέσως μετά το Branch.

Ερώτημα 36
Η προηγούμενη εντολή βρίσκεται στην διεύθυνση 0x00000110. Μετά την εκτέλεση της, τί τιμή θα περιμένουμε να έχει ο LR (R14) και γιατί;

q.

0x000000E4 E1A02000 MOV R2,R0
0x000000E8 E0820001 ADD R0,R2,R1

Παρατηρούμε τα δεδομένα των R0, R2.

Μετακινεί στον R2 τα περιεχόμενα του R0.
Πατάμε το F11 και ελέγχουμε την νεα τιμή του R2.

Προσθέτει τα περιεχόμενα των R1, R2 και τα αποθηκευει στον R0.
Πατάμε το F11 και ελέγχουμε την νεα τιμή του R0.

r.

0x000000EC E12FFF1E BX R14

Επιστρέφει από ένα κάλεσμα συνάρτησης στην διεύθηνση που ορίζεται από τον R14 (Link Register). Ουσιαστικά τα περιεχόμενα του R14 μετακινούνται στο PC.

Παρατηρούμε τα δεδομένα των PC, LR.

Πατάμε το F11 και ελέγχουμε την νεα τιμή του PC. Θα πρέπει να είναι ίδια με τα δεδομένα του LR.

Ερώτημα 37

Η συνάρτηση func1 έχει εκτελεστεί. Ποιο είναι το αποτέλεσμα της και σε ποιόν καταχωρητή έχει αποθηκευθεί;

s.

0x00000114 E1A05000 MOV R5,R0

Παρατηρούμε τα δεδομένα του R5.
Μετακινεί το περιεχόμενο του R0 στον R5.
Πατάμε το F11 και ελέγχουμε την νεα τιμή του R5.

Στο σημείο αυτό ξεκινάει η εκτέλεση της δεύτερης συνάρτησης.

Ερώτημα 38

Προσθέστε παρακάτω τις εντολές που εκτελούνται και αντίστοιχα σχόλια ώστε να δείξετε ότι έχετε καταλάβει τις εντολές assembly του ARM που χρησιμοποιούνται.

5 ΕΡΓΑΣΤΗΡΙΟ 5

- **Χρησιμοποιώντας τα πλήκτρα**
- **Είσοδοι/Εξοδοι Γενικού Σκοπού (GPIO) Μέρος 1ο**

Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαια "Writing Programs" και "Theory of Operation->Push Buttons".
- **Αρχείο Blinky.c,** μέσα από τον φάκελο **Lab2_Blinky.rar**.
- **Αρχείο MCBSTR9_schematic.pdf.** Το σχηματικό της αναπτυξιακής πλακέτας MCBSTR9 για να δούμε τις συνδέσεις των πλήκτρων.
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για τους GPIO καταχωρητές.
- **Βιβλίο Θεωρίας Wayne Wolf, "Οι Υπολογιστές ως Συστατικά Στοιχεία".** Κεφάλαιο 4, παράγραφος 4. Συσκευές Εισόδου/Εξόδου. Πληκτρολόγια.

Εισαγωγή

Το αναπτυξιακό MCBSTR9 παρέχει τρία πλήκτρα. Το όνομα των πλήκτρων τα έχουμε βρει στο Ερώτημα 1 από το σχηματικό MCBSTR9_schematic.pdf. Επίσης στο Ερώτημα 17 έχουμε βρει σε ποια pins (GPIO) του μικροελεγκτή είναι τα πλήκτρα αυτά συνδεδεμένα. Στο κομμάτι αυτό του εργαστηρίου θα χρησιμοποιήσουμε ένα από τα πλήκτρα αυτά για να σταματάμε και να ξεκινάμε την διεργασία του ADC στο παράδειγμα Blinky.

Το πλήκτρο INT5 (S2 στο σχηματικό) και οπότε το αντίστοιχο GPIO, τροφοδοτείται με τάση 3,3V μέσω μίας weak pull-up αντίστασης (R43). Δείτε τα στο MCBSTR9_schematic.pdf σχηματικό. Πατώντας το INT5 πλήκτρο το αντίστοιχο GPIO από 1 γίνεται 0. Η ρουτίνα μας πρέπει να μπορεί να ελέγχει την κατάσταση του GPIO. Εάν το πλήκτρο κρατείται πατημένο, δηλαδή είναι σε κατάσταση 0, τότε θα πρέπει να τυπώσουμε στο LCD “Paused” και τα LED να σταματούν την ανάκληση, ενώ όταν το αφήνουμε, δηλαδή είναι στην κατάσταση 1, θα πρέπει να συνεχίζεται η προηγούμενη διαδικασία.

5.1 Διαδικασία προγραμματισμού GPIO

Για να μπορέσουμε να διαβάζουμε μέσα στον κώδικα μας την κατάσταση του GPIO, με το οποίο συνδέεται το πλήκτρο INT5, θα πρέπει πρώτα να το ενεργοποιήσουμε και να το ρυθμίσουμε καταλλήλως. Αυτό θα γίνει προγραμματίζοντας τους κατάλληλους καταχωρητές. Στον παρακάτω πίνακα θα συμπληρώσουμε του καταχωρητές που χρησιμοποιούνται για τον προγραμματισμό των GPIO.

Ερώτημα 39	
Συγκεντρωτικός πίνακας καταχωρητών για τα GPIO	
Καταχωρητής	**Περιγραφή**

Παρακάτω θα συμπληρώσουμε τις τιμές που πρέπει να πάρουν οι παραπάνω καταχωρητές ώστε το GPIO 3.5 να λειτουργεί ως απλή είσοδος.

Ερώτημα 40
Βγάζουμε το GPIO Port 3 από κατάσταση reset:

Ερώτημα 41
Ενεργοποιούμε το ρολόι του GPIO Port 3:

Ερώτημα 42
Απενεργοποιούμε εναλλακτική έξοδο του GPIO: (SCU GPIOOUT)

Ερώτημα 43
Ρυθμίζουμε το GPIO ως είσοδο:

Αντίστοιχες ρυθμίσεις, που μπορούμε να πάρουμε ως παράδειγμα, υπάρχουν για το GPIO 0 του Port 4 (GPIO 4.0) που χρησιμοποιείται ως είσοδος για τον ADC (Analogue to Digital Converter).

Έχοντας καταλλήλως ρυθμίσει τους καταχωρητές του GPIO, μπορούμε τώρα να ελέγχουμε την κατάσταση του. Αυτό γίνεται ελέγχοντας την τιμή στον καταχωρητή:

GPIO3->DR[0x80]

Ο λόγος που βάζουμε 0x80 εξηγείται στις σελίδες 116-117 του STR91xFA_Reference_Manual.pdf.

Επειδή το GPIO δεν το έχουμε συνδέσει με κάποιο interrupt, πρέπει ο κώδικας μας να ελέγχει τον καταχωρητή αυτόν συνεχώς (polling). **Συνεπώς πρέπει ο έλεγχος να μπει μέσα στο ατέρμονο βρόγχο του Blinky παραδείγματος.** Σε κάθε κύκλο του βρόγχου η κατάσταση του GPIO πρέπει να ελέγχεται χρησιμοποιώντας μια από τις δύο παρακάτω εντολές και πράττοντας αναλόγως:

```
if (GPIO3->DR[0x80] == 0x20) /* Εάν η κατάσταση του GPIO 3.5  είναι 1 */
if (GPIO3->DR[0x80] == 0x00) /* Εάν η κατάσταση του GPIO 3.5  είναι 0 */
```

Ανοίγουμε το **Project Blinky** (με την διαδικασία που αναφέρεται στην παράγραφο 2.1) και στο αρχείο **Blinky.c,** υλοποιούμε τον κώδικα μας.

Ερώτημα 44

Εδώ γράψτε τον κώδικα για το έλεγχο του πλήκτρου. Προσθέστε μόνο τις επιπλέον εντολές που γράψατε για την αρχικοποίηση και έλεγχο του GPIO.

6 ΕΡΓΑΣΤΗΡΙΟ 6

- **ARM9 Assembly Μέρος 1ο**
- **LEDs**
- **Είσοδοι/Εξοδοι Γενικού Σκοπού (GPIO) Μέρος 2ο**

Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαιο “Writing Programs”.
- **Αρχείο Lab6_GPIOs_assembly.s,** μέσα από τον φάκελο **Lab6_GPIOs_assembly.rar**.
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για καταχωρητές.
- **Βιβλίο Θεωρίας Wayne Wolf, “Οι Υπολογιστές ως Συστατικά Στοιχεία”.** Κεφάλαιο 4, παράγραφος 4. Συσκευές Εισόδου / Εξόδου – LEDs.

Εισαγωγή

Στο εργαστήριο αυτό θα χρησιμοποιήσουμε ARM assembly γλώσσα προγραμματισμού για να ρυθμίσουμε το πλήκτρο INT5 ως είσοδο. Το πλήκτρο INT5 συνδέεται με το GPIO 3.5, δηλαδή με τον 5^{ο} ακροδέκτη του Port 3. Ο κώδικας μας θα παρατηρεί την κατάσταση του GPIO 3.5. Εάν το πλήκτρο INT5 κρατείται πατημένο πρέπει να γνωρίζουμε ότι το GPIO 3.5 γίνεται 0 ενώ στην αντίθετη περίπτωση που το πλήκτρο δεν είναι πατημένο, το GPIO 3.5 έχει την κατάσταση 1.

Το πρόγραμμα που θα εκτελέσουμε παρατηρεί την κατάσταση του GPIO 3.5 και αναλόγως την κατάσταση του πλήκτρου ανάβει ή σβήνει το LED 0. Εάν το πλήκτρο είναι πατημένο (GPIO 3.5 = 0), τότε το LED 0 (P 7.0 στο σχηματικό) ανάβει, ενώ στην αντίθετη περίπτωση (GPIO 3.5 = 1) το LED 0 σβήνει.

Το συγκεκριμένο κομμάτι κώδικα είναι ήδη υλοποιημένο και δίδεται με τα συνημμένα αρχεία του εργαστηρίου. Παρ' όλ' αυτά ο φοιτητής, αφού πρώτα διαβάσει, κατανοήσει και εκτελέσει την έτοιμη υλοποίηση για το GPIO 3.5 και LED 0, πρέπει να **προσθέσει** δικό του κώδικα σε assembly ώστε επιπλέον το πλήκτρο INT6 να ανάβει το LED 1 (P 7.1 στο σχηματικό).

6.1 Ανάλυση του κώδικα

Ανοίγουμε το Project **Lab6_GPIOs_assembly.uvproj** αποσυμπιέζοντας το **Lab6_GPIOs_assembly.rar** αρχείο, που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Ανοίγουμε το αρχείο **Lab6_GPIOs_assembly.s**. Διαβάζουμε τον κώδικα και τα σχόλια του. Παρακάτω θα δώσουμε μια σύντομη περιγραφή του κώδικα.

```
; APB0, 1 Bridge definition (Peripherals)
APB0_NBUF_BASE  EQU   0x58000000 ; APB Bridge 0 Non-buffered Base Address
APB1_NBUF_BASE  EQU   0x5C000000 ; APB Bridge 1 Non-buffered Base Address
```

Δηλώνουμε την αρχική διέυθυνση των γεφυρών APB0, 1 (non-buffered). Η διεύθυνση θα λειτουργεί ως η βάση στην οποία οι υπόλοιπες διευθύνσεις θα χτίζονται.

```
; System Control Unit (SCU) definitions
SCU_BASE      EQU   APB1_NBUF_BASE + 0x2000    ; SCU Base Address
SCU_PCGR1_OFS EQU   0x18          ; Peripheral Clock Gating Register 1 Offset
SCU_PRR1_OFS  EQU   0x20          ; Peripheral Reset Register 1 Offset
SCU_GPIOUT7_OFS EQU 0x60          ; GPIO Out Register 7
```

Δηλώνουμε την αρχική διέυθυνση της Μονάδας Ελέγχου του Συστήματος (System Control Unit - SCU), η οποία ξεκινάει 0x2000 bytes από την αρχική διέυθυνση του APB1. Επίσης δηλώνουμε τις μετατοπίσεις των υπολοίπων καταχωρητών που θα χρησιμοποιήσουμε.

Από το Ερώτημα 39 μπορούμε να καταλάβουμε γιατί μας χρειάζονται οι παραπάνω καταχωρητές.

Ερώτημα 45
Γιατί στις παραπάνω δηλώσεις δεν έχουμε δηλώσει την απόκλιση του καταχωρητή SCU_GPIOOUT3, μιας και το GPIO Port 3 το χρειαζόμαστε για το πλήκτρο INT5;

```
; Values for the SCU equivalen registers
SCU_PCGR1_Val   EQU    0x00220000
SCU_PRR1_Val    EQU    0x00220000
```

Δηλώνουμε τις τιμές που που πρέπει να προγραμματίσουμε στους καταχωρητές **SCU_PCGR1** και **SCU_PRR1.**

Ερώτημα 46
Γιατί πρέπει να προγραμματίσουμε τους καταχωρητές SCU_PCGR1 και SCU_PRR1 με τις τιμές 0x00220000;

```
; GPIO Port 7 Addresses
GPIO7_DATA_ADDR   EQU    APB0_NBUF_BASE + 0x0000D000
GPIO7_DIR_OFS     EQU    0x00000400
```

Δηλώνουμε τις διευθύνσεις για τους καταχωρητές του GPIO Port 7. Ο συνολικός αριθμός των καταχωρητών για κάθε GPIO είναι 8. Εμείς εδώ δηλώνουμε 2 σύν 3 στο SCU παραπάνω. Αυτούς τους 5 καταχωρητές θα χρειαστούμε μόνο διότι οι υπόλοιποι 3 έχουν προκαθορισμένες τιμές 0x00000000 και δεν χρειάζεται να τους αλλάξουμε.

```
; GPIO Port 7 Program values
SCU_GPIOUT7_Val     EQU          0x00005555
GPIO7_DIR_Val       EQU          0xFF
```

Δηλώνουμε τις τιμές που πρέπει να προγραμματίσουμε στους καταχωρητές του GPIO 7.

```
; GPIO Port 3 Data Address
GPIO3_DATA_ADDR   EQU          APB0_NBUF_BASE + 0x00009000
```

Δηλώνουμε την διεύθυνση του καταχωρητή των δεδομένων του GPIO P3.

Ερώτημα 47
Πόσους καταχωρητές για το GPIO Port 3 έχουμε δηλώσει; Γιατί δεν τους έχουμε δηλώσει όλους;

```
; Area Definition and Entry Point
; Startup Code must be linked first at Address at which it expects to run.
        AREA    Reset, CODE, READONLY
```

Η εντολή AREA σημαδεύει την αρχή ενός ξεχωριστού νέου κομματιού κώδικα ή δεδομένων. Πληροφορεί τον συμβολομεταφραστή (assembler) να μεταφράσει ένα νέο κομμάτι κώδικα. Η εντολή AREA έχει την μορφή

`AREA` *`sectionname`*`{,`*`attr`*`}{,`*`attr`*`}...`

`sectionname`: το όνομα που θα δώσουμε στο νέο κομματι κώδικα ή δεδομένων.

`attr`: Ιδιότητες του νέου κομματιού. Π.χ. εάν είναι κώδικας ή δεδομένα.

```
; Area Definition and Entry Point
; Startup Code must be linked first at Address at which it expects to run.
                ENTRY


        NOP     ; Wait for OSC stabilization
        NOP
```

Η εντολή ENTRY σημαδεύει την πρώτη εντολή που θα εκτελεστεί. Στον κώδικα αυτό η πρώτη εντολή θα είναι μια NOP. Η NOP σημαίνει No Operation, καμία λειτουργία και διαρκεί 2 κύκλους ρολογιού. Οπότε χρησιμοποιείται όπου χρειάζονται καθυστερήσεις. Εδώ έχουν μπεί 10 NOP και θα είναι οι πρώτες εντολές που θα εκτελεστούν αμέσως μόλις η πλακέτα τροφοδοτηθεί με ρεύμα. Στις περιπτώσεις αυτές, επειδή υπάρχουν μεταβατικά φαινόμενα τάσεων και επειδή οι κρύσταλοι θέλουν κάποιο χρόνο για να σταθεροποιηθούν, NOP εντολές χρησιμοποιούνται για δημιουργία μαις αρχικής καθυστέρησης μέχρι το σύστημα να σταθεροποιηθεί.

Ερώτημα 48
Εάν το σύστημα τρέχει στα 25MHz πόσο χρόνο θα διαρκέσουν οι 10 NOP εντολές;

```
; Load SCU Base address to R0
; Then we can move by using offsets from that address
        LDR             R0, =SCU_BASE
; Enable GPIO Port 3 and Port 7 clocks
        LDR     R1, =SCU_PCGR1_Val
        STR     R1, [R0, #SCU_PCGR1_OFS]
```

Φορτώνεται στον R0 η τιμή της σταθεράς SCU_BASE.

Φορτώνεται στον R1 η τιμή της σταθεράς SCU_PCGR1_Val.

Αποθηκεύεται στον R0 + SCU_PCGR1_OFS η τιμή του R1.

Ερώτημα 49
Ποια θα περιμέναμε να είναι τα αποτελέσματα των τριών παραπάνω εντολών;

```
; Set GPIO Port 3 and Port 7 out of reset
        LDR     R1, =SCU_PRR1_Val
        STR     R1, [R0, #SCU_PRR1_OFS]
```

Φορτώνεται στον R0 η τιμή της σταθεράς SCU_BASE.

Φορτώνεται στον R1 η τιμή της σταθεράς SCU_PRR1_Val.

Αποθηκεύεται στον R0 + SCU_PRR1_OFS η τιμή του R1.

Ερώτημα 50
Ποιά θα περιμέναμε να είναι τα αποτελέσματα των δύο παραπάνω εντολών; Τα δεδομένα του R0 δεν έχουν μεταβληθεί από την πρώτη εντολή: **LDR R0, =SCU_BASE**

```
; Set SCU GPIO Port7 Out at Mode 1: Normal Output
        LDR    R1, =SCU_GPIOUT7_Val
        STR    R1, [R0, #SCU_GPIOUT7_OFS]

; Set GPIO Port7 Output
        LDR    R0, =GPIO7_DATA_ADDR
        LDR    R1, =GPIO7_DIR_Val
        STR    R1, [R0, #GPIO7_DIR_OFS]
```

Αντίστοιχα, όπως δείξαμε προηγουμένως, δουλεύουν και οι παραπάνω εντολές.

```
Check_Button
                LDR         R0, =GPIO3_DATA_ADDR
                LDRB        R0, [R0, #0x080]
                CMP         R0, #0x00
                BNE         Led0_OFF
                B           Led0_ON
```

Το Check_Button είναι μια ετικέτα η οποία χρησιμοποιείται για να ονοματίσει το συγκεκριμένο σημείο του κώδικα. Ο ετικέτες χρησιμοποιούνται κυρίως στα branches (διακλάδωση) ώστε αντί να καλούνται branches με αριθμό διευθύνσεως να καλούνται με το ονομά τους. Για παράδειγμα:

B Led0_ON αντί για B 0x00000092

Το παραπάνω κομμάτι του κώδικα είναι πολύ σημαντικό.

Διαβάζει τον καταχωρητή των δεδομένων του GPIO Port 3

```
        LDR         R0, =GPIO3_DATA_ADDR
```

και κοιτάει εάν το GPIO 3.5 έιναι 0.

```
        CMP         R0, #0x00000000
```

Δηλαδή ελέγχει έαν το κουμπί είναι πατημένο. Εάν είναι 0 τότε κάνει ένα branch στην διεύθηνση Led0_OFF διαφορετικά στην διεύθηνση Led0_ON.

```
; Set GPIO Port7 GPIO1 High
Led0_ON
        LDR    R0, =GPIO7_DATA_ADDR
        LDR    R1, =0x01
        STR    R1, [R0, #0x004]
        B      Check_Button

Led0_OFF
        LDR    R0, =GPIO7_DATA_ADDR
        LDR    R1, =0x00
        STR    R1, [R0, #0x004]
        B      Check_Button
```

Τα δύο αυτά κομμάτια κώδικα ανάβουν ή σβήνουν το LED 0 (P 7.0 στο σχηματικό).

LDR R0, =GPIO7_DATA_ADDR

Φορτώνεται, στον R0, η διεύθηνση του καταχωρητή των δεδομένων του GPIO Port 7.

LDR R1, =0x01

ή

LDR R1, =0x00

Φορτώνεται, στον R1, η τιμή η οποία θέλουμε να περάσουμε στον R0 καταχωρητή. Η τιμή είναι 1 ή 0.

STR R1, [R0, #0x004]

Αποθηκεύονται τα περιεχόμενα του R1, στην διεύθηνση του καταχωρητή που προκείπτει από τα δεδομένα του R0 + 0x004.

B Check_Button

Αφού τελιώσουμε με το LED επιστρέφουμε στον έλεγχο του κουμπίου.

6.2 Εκτέλεση του κώδικα

Έχοντας διαβάσει και κατανοήσει την δομή του κώδικα, μπορούμε να τον εκτελέσουμε. Χρησιμοποιώντας τον debugger μπορούμε να δούμε την χρησιμοποίηση των καταχωρητών προγραμματισμού R0 και R1 καθώς επίσης και τον Program Counter στα καλέσματα των branches.

- **Βάλτε ένα breakpoint στην διεύθυνση Led0_ON**.
- Πατήστε το πλήκτρο της πλακέτας.
- Θα πρέπει η εκτέλεση του κώδικα να σταματήσει με το πάτημα του πλήκτρου.
- Εκτελέστε τον κώδικά από το σημείο αυτό βηματικά, πατώντας το F11.
- Παρατηρήστε τους καταχωρητές R0, R1.
- Παρατηρήστε τους καταχωρητές περιφερειακών GPIO, όπως φαίνεται στο Σχήμα 23.
- Δείτε ποιά στιγμή το LED ανάβει.

6.3 Δημιουργία κώδικα για το πλήκτρο INT6

Ερώτημα 51

Συνεχίστε τον Lab6_GPIOs_assembly.s κώδικα ώστε το πάτημα του πλήκτρου INT6 να ανάβει το LED 1 (P 7.1 στο σχηματικό).

7 ΕΡΓΑΣΤΗΡΙΟ 7

- **ARM9 Assembly Μέρος 2ο**
- **Εξωτερικές Διακοπές – External Interrupts Μέρος 1ο**

Προϋποθέσεις

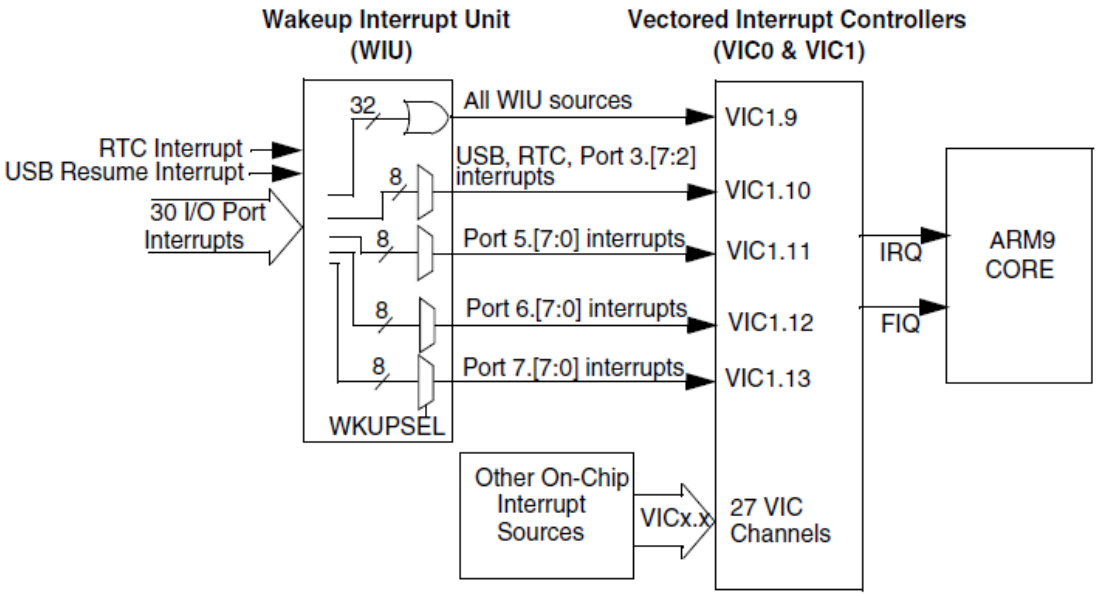
Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαιο "Writing Programs".
- **Αρχείο Lab7_Ext_Int.s,** μέσα από τον φάκελο **Lab7_Ext_Int.rar**.
- **Αρχείο STR9_Programming_Manual.pdf**
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για καταχωρητές.
- **Βιβλίο Θεωρίας Wayne Wolf, "Οι Υπολογιστές ως Συστατικά Στοιχεία".** Κεφάλαιο 3, παράγραφος 2.4. Διακοπές.

Εισαγωγή

Οι διακοπές είναι μια σηματοδοσία προς τον CPU. Ονομάζονται έτσι διότι διακόπτουν την εκτέλεση του τρέχοντος προγράμματος και εξαναγκάζουν τον CPU να εκτελέσει τον κώδικα που είναι γραμμένος ειδικά για την εξυπηρέτηση της συγκεκριμένης διακοπής. Όταν η εξυπηρέτηση ολοκληρωθεί, δηλαδή όταν το κομμάτι κώδικα γραμμένο ειδικά για την συγκεκριμένη διακοπή εκτελεστεί, η ροή του προγράμματος επιστρέφει στην θέση που ήταν πριν την διακοπή αυτή.

Ανάλογα με τον μικροελεγκτή και τον CPU, υπάρχει διαφορετικός αριθμός αλλά και είδος διακοπών που μπορούν να υποστηριχθούν. Ο STR921FAW44 υποστηρίζει συνολικά 32 εισόδους διακοπών, από τις οποίες οι 5 μπορεί να προέρχονται από εξωτερικές πηγές όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 33: Διάγραμμα ελέγχου διακοπών

Τα εσωτερικά περιφερειακά, όπως οι μετρητές, ο ADC, ή UART μπορούν να διακόψουν τον CPU.

Ο ARM CPU άμεσα μπορεί να δεχθεί ένα FIQ ή ένα IRQ Interrupt. Τα υπόλοπα IRQs μπαίνουν σε προτεραιότητα ανάλογα με το αριθμό του

Vector (το μικρότερο έχει μεγαλύτερη προτεραιότητα) ή ανάλογα με τον προγραμματισμό που υπάρχει για κάθε vector. Vector Interrupt σημαίνει ότι η ρουτίνα εξυπηρέτησης μπορεί να οριστεί έτσι ώστε να βρίσκεται σε οποιαδήποτε θέση μνήμης. Αυτό όμως δημιουργεί κάποια καθυστέρηση, γιατί η διεύθυνση της ρουτίνας εξυπηρέτησης του Interrupt πρέπει να προσδιοριστεί. Απο την άλλη το FIQ Interrupt έχει πάντα προτεραιότητα απο όλα τα υπόλοιπα Interrupts. Δεν μπαίνει σε κάποια σειρά προτεραιότητας και επίσης η διεύθυνση της ρουτίνας εξυπηρέτησης του FIQ βρίσκεται πάντα σε προκαθορισμένο σημείο. Όλα αυτά έχουν ώς αποτέλεσμα το FIQ να είναι ένα πολύ γρήγορο Interrupt. Παρ' όλ' αυτά μόνο μία πηγή μπορεί να το χρησιμοποιεί.

Εμείς θα ασχοληθούμε με τα εξωτερικά Interrupts, τα οποία συνδέονται στα Vectored IRQs. Συγκεκριμένα θα ορίσουμε ένα εξωτερικό interrupt στο πλήκτρο INT5 (GPIO 3.5). Αντί να κάνουμε polling συνεχώς για να ελέγχουμε την κατάσταση του πλήκτρου θα δημιουργήσουμε μια ρουτίνα διακοπής (interrupt routine) η οποία θα καλείται μόνο όταν πατηθεί το πλήκτρο.

Το πρόγραμμα μας θα εκτελεί έναν κύριο βρόχο. Σε κάθε κύλιση του βρόχου τα δεδομένα του R2 ελέγχονται και ανάλογα με την τιμή του τα LED στην οθόνη ανάβουν ή σβήνουν. Η τιμή του R2 αλλάζει κάθε φορά που πατιέται το πλήκτρο INT5, μέσα στην εξυπηρέτηση του Interrupt.

7.1 Ανάλυση του κώδικα

Ανοίγουμε το Project **Lab7_Ext_Int.uvproj** αποσυμπιέζοντας το **Lab7_Ext_Int.rar** αρχείο, που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Ανοίγουμε το αρχείο **Lab7_Ext_Int.s**. Διαβάζουμε τον κώδικα και τα σχόλια του. Παρακάτω θα δώσουμε μια σύντομη περιγραφή των ποιό βασικών σημείων του κώδικα.

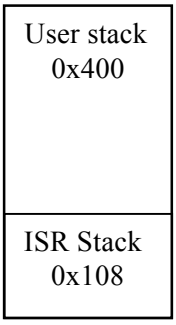
```
Mode_USR    EQU   0x10
Mode_FIQ    EQU   0x11
Mode_IRQ    EQU   0x12
Mode_SVC    EQU   0x13
Mode_ABT    EQU   0x17
Mode_UND    EQU   0x1B
Mode_SYS    EQU   0x1F

I_Bit       EQU   0x80
F_Bit       EQU   0x40
```

Δηλώσεις σταθερών τιμών που ορίζουν την τιμή του CPSR ή των αντίστοιχων SPSR για τις διάφορες καταστάσεις του ARM.

Τα I_Bit και F_Bit, χρησιμοποιούνται για να απενεργοποιήσουν τα αντίστοιχα Interrupts.

```
UND_Stack_Size      EQU    0x00000000
SVC_Stack_Size      EQU    0x00000008
ABT_Stack_Size      EQU    0x00000000
FIQ_Stack_Size      EQU    0x00000000
IRQ_Stack_Size      EQU    0x00000100
USR_Stack_Size      EQU    0x00000400

ISR_Stack_Size      EQU    (UND_Stack_Size + SVC_Stack_Size +
ABT_Stack_Size + FIQ_Stack_Size + IRQ_Stack_Size)

                AREA    STACK, NOINIT, READWRITE, ALIGN=3

Stack_Mem       SPACE   USR_Stack_Size
__initial_sp    SPACE   ISR_Stack_Size

Stack_Top
```

Το κομμάτι αυτό ορίζει το μέγεθος της στοίβας (Stack) για την κατάσταση χρήστη και για οποιαδήποτε κατάσταση εξαίρεσης. Η στοίβα είναι ένας προκαθορισμένος χώρος μνήμης που δεσμεύεται στην αρχή εκκίνησης του συστήματος. Ο χώρος αυτός χρησιμοποιείται από τις διάφορες συναρτήσεις για να αποθηκεύουν τρέχουσες μεταβλητές που άμεσα χρησιμοποιούν, αλλά παράλληλα δεν χρειάζονται μετά το πέρας της εκτέλεσης τους. Όταν μια συνάρτηση επιστρέψει, η μεταβλητή αυτή και οποιαδήποτε άλλη μεταβλητη αποδεσμεύονται από την στοίβα. Ανάλογα με την κατάσταση λειτουργίας του MCU υπάρχει και η ανάλογη στοίβα. Εδώ για παράδειγμα ορίζεται ένα Stack για την κατάσταση χρήστη και ένα για τις εξαιρέσεις. Στη στοίβα αποθηκεύονται δεδομένα από την μεγαλύτερη προς τη μικρότερη διεύθυνση και έτσι το 0x400 και 0x108 είναι τα μεγιστα όρια για τις δύο στοίβες. Για παράδειγμα ένα PUSH μιας 32-bit μεταβλητής στο User Stack θα αποθηκευθεί στην διεύθυνση 0x39C-0x400.



Η επόμενη στην 0x390-39C κτλ.

Η εντολή SPACE δεσμεύει χώρο για την Στοίβα της κατάστασης χρήστη και για την στοίβα των Εξαιρέσεων.

```
IRQHandler
                    STMDB    R13!,{R0-R1}

; Clear VIC interrupt
                    LDR    R0, =VIC1_BASE
                    LDR    R1, =0x00000000
                    STR    R1, [R0, #VIC1_VAR_OFS]
; Clear WIU Interrupt
                    LDR    R0, =WIU_CTRL_BASE
                    LDR    R1, =WIU_PR_Val
                    STR    R1, [R0, #WIU_PR_OFS]

                    MVN      R2, R2
                    LDMIA    R13!,{R0-R1}
                    SUBS     PC, R14, #0x4
```

Εδώ ορίζεται η εξυπηρέτηση του IRQ Interrupt. Όταν το κουμπί INT5 πατηθεί, ένα Interrupt θα δημιουργηθεί στο Vector Interrupt 1.10 και ο παραπάνω κώδικας θα εκτελεστεί. Στο τέλος, η εκτέλεση του προγράμματος θα επιστρέψει στην θέση στην οποία ήταν πριν την εξυπηρέτηση του Interrupt αυτού. Η εντολή MVN R2, R2 αντιστρέφει τα δεδομένα του R2 κάθε φορά που πατιέται το κουμπί ώστε ο κύριος βρόχος να γνωρίζει εάν τα LED είναι αναμμένα ή όχι. Ετσι ο R2 μπορεί να είται 0xFFFFFFFF είται 0x00000000.

```
Main_Loop
                    CMP    R2, #0xFFFFFFFF
                    BNE    LED_Off
                    B      LED_On
LED_On
                    LDR    R0, =GPIO7_DATA_ADDR
                    LDR    R1, =0xFF
                    STR    R1, [R0, #0x3FC]

                    B      Main_Loop
LED_Off
                    LDR    R0, =GPIO7_DATA_ADDR
                    LDR    R1, =0x00
                    STR    R1, [R0, #0x3FC]
                    B      Main_Loop
```

Το σύστημα ξεκινάει με τα LED σβηστά και τον R2=0. Με το πρώτο πάτημα του κουμπίου INT5 ένα Interrupt θα διακόψει τον CPU και η εξυπηρετησή του θα εντιστρέψει τον R2 από 0x0 σε 0xFFFFFFFF.

Μετά η εκτέλεση του κώδικα θα επιστρέψει στο σημείο του Main_Loop που βρισκόταν πριν το Interrupt.
Το Main_Loop συγκρίνει την κατάσταση του R2 εάν είναι 0xFFFFFFFF. Εάν είναι 0xFFFFFFFF, σημαίνει ότι το κουμπί έχει πατηθεί και οπότε ανάβει τα LED. Διαφορετικά τα κρατάει σβηστά.

Πριν τον ορισμό του Main_Loop γίνονται αρκετές αρχικοποιήσεις σε καταχωρητές. Τον σκοπό των αρχικοποιήσεων αυτών θα τον συζητήσουμε και θα τον συμπληρώσουμε παρακάτω.

Ερώτημα 52	
SCU_PCGR0	
Τιμή	**Εξήγηση**

Ερώτημα 53	
SCU_PRR0	
Τιμή	**Εξήγηση**

Ερώτημα 54	
SCU_PRR1	
Τιμή	**Εξήγηση**

Ερώτημα 55	
SCU_WKUPSEL	
Τιμή	**Εξήγηση**

Ερώτημα 56	
VIC_INTER	
Τιμή	**Εξήγηση**

Ερώτημα 57	
VIC1_VA10R	
Τιμή	**Εξήγηση**

Ερώτημα 58	
VIC1_VC10R	
Τιμή	**Εξήγηση**

Ερώτημα 59	
SCU_GPIOOUT7	
Τιμή	**Εξήγηση**

Ερώτημα 60	
GPIO7_DIR	
Τιμή	**Εξήγηση**

Ερώτημα 61	
WIU_CTRL	
Τιμή	**Εξήγηση**

Ερώτημα 62	
WIU_MASK (WIU_MR)	
Τιμή	**Εξήγηση**

Ερώτημα 63	
WIU_TRIG (WIU_TR)	
Τιμή	**Εξήγηση**

8 ΕΡΓΑΣΤΗΡΙΟ 8

- **ARM9 Assembly Μέρος 3ο**
- **Εξωτερικές Διακοπές – External Interrupts Μέρος 2ο**

Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαιο "Writing Programs".
- **Αρχείο Lab7_Ext_Int.s,** μέσα από τον φάκελο **Lab7_Ext_Int.rar**.
- **Αρχείο STR9_Programming_Manual.pdf**
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για καταχωρητές.
- **Βιβλίο Θεωρίας Wayne Wolf, "Οι Υπολογιστές ως Συστατικά Στοιχεία".** Κεφάλαιο 3, παράγραφος 2.4. Διακοπές.

Εισαγωγή

Το εργαστήριο αυτό είναι συνέχεια του προηγούμενου εργαστηρίου.

Στο προηγούμενο εργαστήριο αναλύσαμε τον κώδικα, γραμμή-γραμμή για να κατανοήσουμε τις ρυθμίσεις και τον προγραμματισμό που απαιτείται για να ρυθμίσουμε τις εξωτερικές διακοπές στο STR912FAW44 μικροελεγκτή.

Σε αυτό το εργαστήριο θα εκτελέσουμε τον κώδικα μας και θα χρησιμοποιήσουμε τον Debugger για να παρακολουθήσουμε τους καταχωρητές των Interrupt και την διαδικασία που ακολουθείται. Θα χρησιμοποιήσουμε Breakpoints και θα παρακολουθούμε τους Registers γενικού σκοπού.

Τέλος θα προσπαθήσουμε να μετατρέψουμε τον κώδικα μας ώστε πλέον το Interrupt να έρχεται από το πλήκτρο INT6 και όχι από το πλήκτρο 5.

8.1 Εκτέλεση του κώδικα

Ανοίγουμε το Project **Lab7_Ext_Int.uvproj** αποσυμπιέζοντας το **Lab7_Ext_Int.rar** αρχείο, που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Ανοίγουμε το αρχείο **Lab7_Ext_Int.s**.

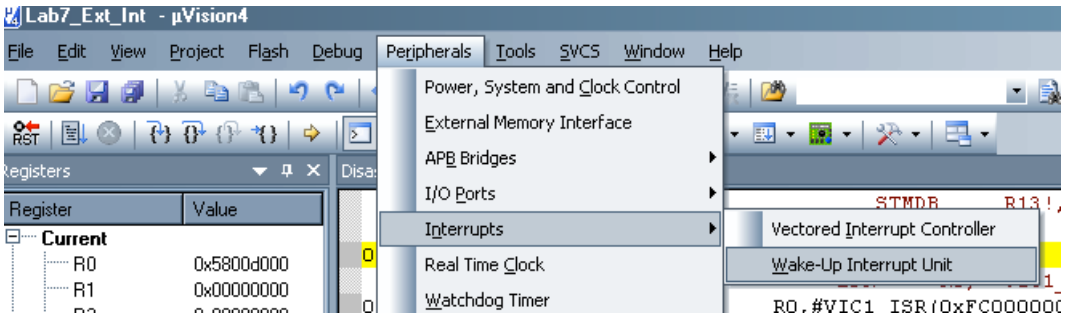
Εκτελούμε τον κώδικα σύμφωνα με την διαδικασία που αναφέρεται στην παράγραφο 2.1 και 2.2.

Πατάμε το πλήκτρο INT5 και παρατηρούμε τα LED να ανάβουν. Ξαναπατώντας το πλήκτρο INT5 τα πλήκτρα θα πρέπει να σβήνουν.

8.2 Έλεγχος μέσο του αποσφαλματωτή

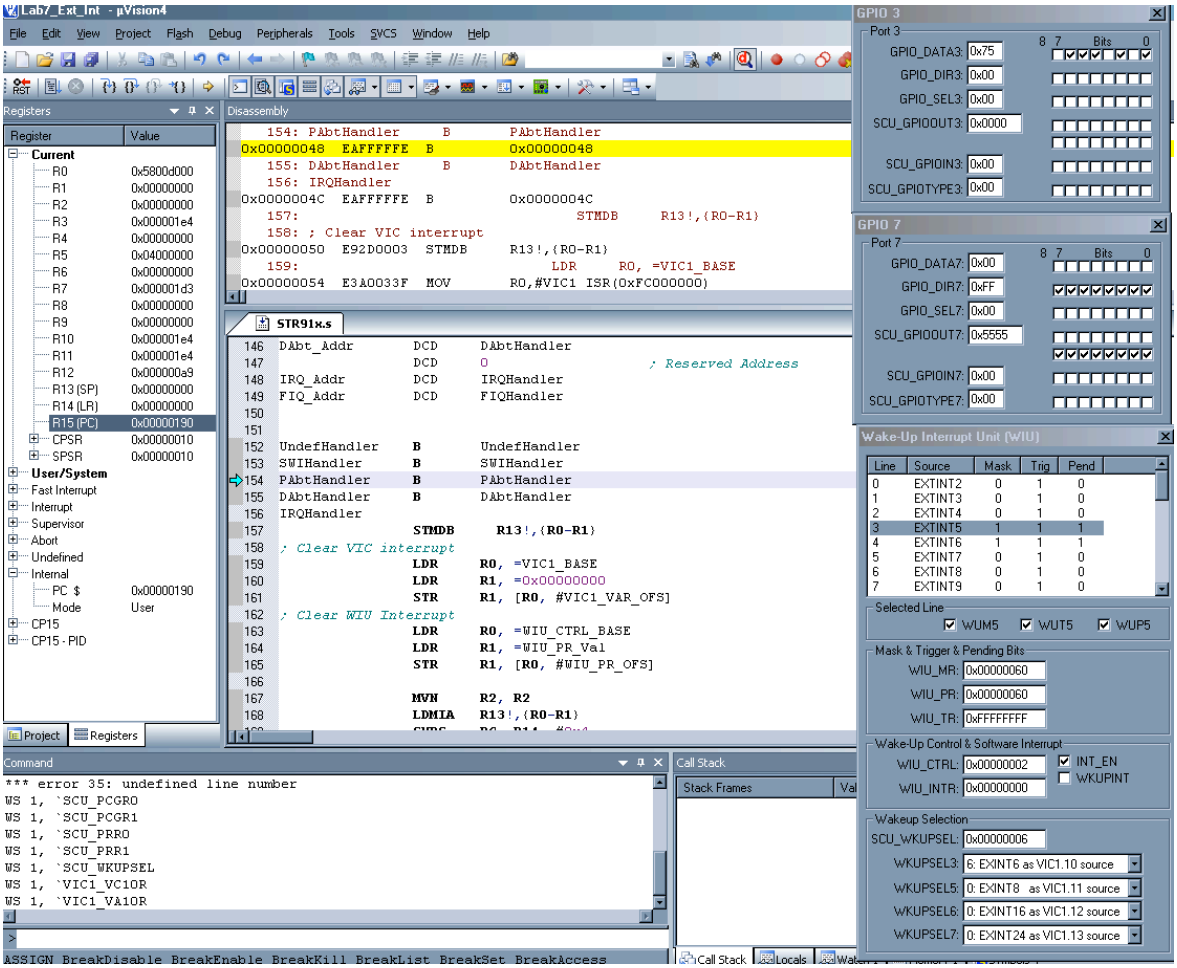
Ανοίγουμε τον απασφαλματωτή αφού σιγουρευτούμε ότι έχουμε επιλέξει το *Use ULINK ARM Debugger* κάτω από το Debug tab στο παράθυρο *'Options for Target 'Target 1'*. Το *Run to main* πρέπει να είναι επίσης επιλεγμένο.

Από την επιλογή Peripherals ανοίγουμε τα εξής παράθυρα:
Wake-Up Interrupt Controller, GPIO3 και GPIO7



Σχήμα 34: Επιλογή Wake Up Interrupt Controller

Η τελική μορφή του Debugger πρέπει να έχει την δομή που φαίνεται στο παρακάτω σχήμα.

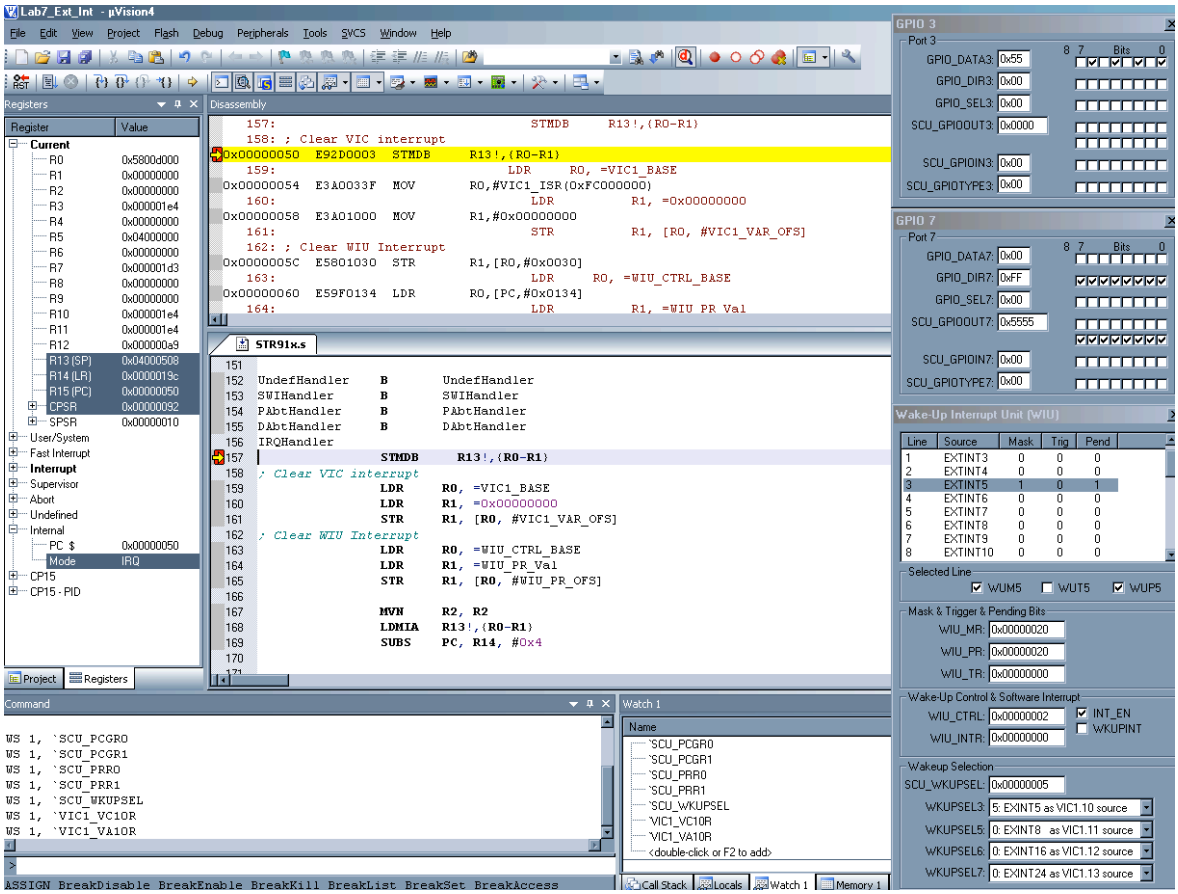


Σχήμα 35: Δομή παραθύρων Debugger για έλεγχο του Εξωτερικού Interrupt

Από το GPIO3 παράθυρο παρατηρούμε ότι το GPIO 3.5 είναι 1. Με το πάτημα γίνεται 0. Αυτό είναι αναμενόμενο και για τον λόγο αυτό έχουμε ορίσει στον WIU Interrupt Trigger Register να ενεργεί το Interrupt με την πτίτουσα ακμή του GPIO.

Στο παράθυρο **Wake-UP Interrupt Unit** επιλέγουμε το **EXTINT5**. Παρατηρούμε την μάσκα (Mask) ότι είναι 1, και το INT_EN bit επίσης.

Θέτουμε ένα breakpoint στην αρχή του IRQHandler και εκτελούμε τον κώδικα πατώντας το Run ή το F5.
Πατάμε το πλήκτρο και ο κώδικας σταματάει στο breakpoint λόγω του Interrupt.



Σχήμα 36: Breakpoint στην εξυπηρέτηση του Interrupt

Στην παραπάνω εικόνα έχουμε τα στοιχεία του debbuger την στιγμή που πατήθηκε το πλήκτρο. Στο παράθυρο GPIO3, το GPIO3.5 έχει γίνει 0. Επίσης στο παράθυρο Wake-UP Interrupt Unit το bit WUIP5 έχει γίνει 1 (Pending = 1).

Ερώτημα 64
Ποια η ερμηνεία του bit WUIP5;

Από το σημείο που ο κώδικας μας έχει σταματήσει λόγω του Interrupt και του breakpoint, τον εκτελούμε βηματικά πατώντας το F11. Παρατηρούμε

τον καθαρισμό του WIU EXTINT5 Pending bit μετά την εκτέλεση των παρακάτω εντολών:

```
LDR     R0, =WIU_CTRL_BASE
LDR     R1, =WIU_PR_Val
STR     R1, [R0, #WIU_PR_OFS]
```

Ερώτημα 65
Τι ακριβώς κάνουν οι παραπάνω εντολές ώστε να καθαρίσει το Pending bit στον WIU EXTINT5 καταχωρητή.

Όταν ο PC βρίσκεται στην εντολή:

```
MVN     R2, R2
```

παρατηρήστε τον R12 πριν και μετά την εκτέλεση της.

Συνεχίστε την βηματική εκτέλεση του κώδικα έως ότου επιστρέψουμε στην Main_Loop. Όταν φτάσετε την εντολή:

```
CMP     R2, #0xFFFFFFFF
```

παρατηρήστε τις δύο επόμενες εντολές

```
BNE     LED_Off
B       LED_On
```

και τα περιεχόμενα του R2 και απαντήστε:

Ερώτημα 66

Ποια είναι τα περιεχόμενα του R2, και σε ποιο από τα δύο Branches θα περιμένατε ο κώδικας να μεταπηδήσει και γιατί;

Μπορείτε να πειραματιστείτε αλλάζοντας τον αριθμό των LED που ανάβουμε και σβήνουμε.

Ερώτημα 67
Ποιες αλλαγές πρέπει να γίνουν στον κώδικα μας ώστε το External Interrupt να έρχεται πλέον από το INT6 και όχι από το INT5; Σημειώστε παρακάτω τις αλλάγες/αντικαταστήσεις.

9 ΕΡΓΑΣΤΗΡΙΟ 9

♦ Εισαγωγή στους ADCs

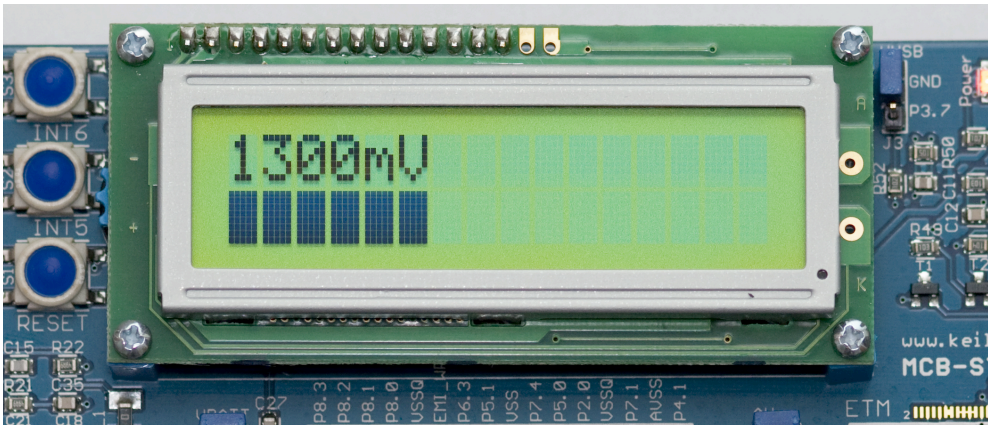
Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαια “Writing Programs” και “Theory of Operation->Potentiometer”.
- **Αρχεία Blinky.c και LCD.c,** μέσα από τον φάκελο **Lab2_Blinky.rar**.
- **Αρχείο MCBSTR9_schematic.pdf.** Το σχηματικό της αναπτυξιακής πλακέτας MCBSTR9 για να δούμε τις συνδέσεις του ποτενσιόμετρου.
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για τους ADC καταχωρητές.
- **Βιβλίο Θεωρίας Wayne Wolf, “Οι Υπολογιστές ως Συστατικά Στοιχεία”.** Κεφάλαιο 4, παράγραφος 4. Συσκευές Εισόδου-Εξόδου, Μετατροπείς A/D και D/A.

Εισαγωγή

Στο εργαστήριο αυτό θα χρησιμοποιήσουμε το παράδειγμα Blinky της Keil. Θα κάνουμε αλλαγές στον κώδικα ώστε στην πρώτη γραμμή της LCD οθόνης, να τυπώνετε η πραγματική τιμή της τάσης του ποτενσιόμετρου POT1 σε mV, όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 37: Ένδειξη τάσης ποτενσιόμετρου στο LCD

Σχήμα 38: Ένδειξη LCD πριν τις αλλαγές μας

Στο αρχείο STR91xFA_Reference_Manual.pdf, σελίδα 463, υπάρχουν πληροφορίες για τους ADCs του μικροελεγκτή. Σε γενικές γραμμές η λειτουργία των ADC έχει ως εξής.

- Ο STR912FAW44 έχει 8 κανάλια ADC.
- Κάθε ADC έχει διακριτότητα 10-bit.

- Λόγω ότι η διακριτότητα των ADC είναι 10-bit, η μέγιστη τιμή που δίνουν ως αποτέλεσμα μετατροπής είναι η 2^{10} = 1023.
- Η μέγιστη και ελάχιστη τιμή που μπορεί ο ADC να μετατρέψει εξαρτάται από την ειδική παροχή τροφοδοσίας στον μικροελεγκτή, VREF. Έτσι για παράδειγμα εάν το VREF είναι 1023mV, τότε η μέγιστη μετατρεπόμενη τιμή του ADC, το 1023, θα αντιστοιχεί σε 1023mV και εάν το VREF είναι 3300mV τότε η μέγιστη τιμή του ADC, το 1023, θα αντιστοιχεί σε 3300mV. Στην περίπτωση αυτή όμως επειδή ο ADC μπορεί να δώσει 1023 τιμές, η αύξηση των τιμών του ADC θα γίνεται κάθε 3300mV/1023 = 3,2mV. Ή διαφορετικά για να υπολογίσουμε την τάση σε mV πολλαπλασιάζουμε την δεκαδική τιμή του ADC με 3,2. Αυτό σημαίνει ότι έχουμε 3,2mV διαφορά ανά βήμα του ADC.

Ερώτημα 68
Έστω ότι ο ADC ήταν 12-bit και το VREF στον μικροελεγκτή είναι 10V. Υπολογίστε το βήμα του ADC ανά mV αλλαγής στην είσοδο του.

Για να μπορέσουμε να δούμε την τιμή της τάσης του ποτενσιόμετρου στην οθόνη του LCD πρέπει πρώτα να μετατρέψουμε την τιμή του ADC σε πραγματικά mV. Είδαμε πριν ότι με VREF 3,2V για κάθε 3,2mV αύξηση της τάσης μετρήσεως του ADC, έχουμε ένα βήμα αύξηση. Δηλαδή την τιμή που μας δίνει ο ADC ως αποτέλεσμα εάν την πολλαπλασιάσουμε με 3,2 θα πάρουμε την πραγματική τιμή της τάσης εισόδου στον ADC.

Ανοίγουμε το **Project Blinky** (με την διαδικασία που αναφέρεται στην παράγραφο 2.1) και το αρχείο **Blinky.c.**

9.1 Ένδειξη της τάσης του ποτενσιόμετρου στην οθόνη LCD

Στο αρχείο **Blinky.c** πρέπει να προσθέσουμε κώδικα ώστε να τυπώσουμε την τάση του ποτενσιόμετρου στο LCD.

Η μεταβλητή **AD_last** κρατάει την τελευταία μέτρηση του ADC. Η μεταβλητή αυτή είναι εξωτερική μεταβλητή και ανανεώνεται μέσα στο αρχείο **IRQ.c**, στην συνάρτηση **ADC_IRQ_Handler.** Όταν μια μετατροπή έχει τελειώσει από τον ADC ένα ειδικό **interrupt** χτυπάει και η εξυπηρέτηση του interrupt αυτού γίνεται στην συνάρτηση αυτή.

Το πρόγραμμα Blinky τρέχει ουσιαστικά έναν κλειστό ατέρμονο βρόχο. Όταν ξεκινήσει το πρόγραμμα γίνονται πρώτα κάποιες αρχικοποιήσεις στο σύστημα, προγραμματίζοντας κάποιους καταχωρητές. Οι καταχωρητές ή αλλιώς ο χάρτης μνήμης **(Memory Map)** του μικροελεγκτή, υλοποιείται στο αρχείο **C:\Keil\ARM\INC\ST\91x\91x_map.h.** Χρησιμοποιώντας τις δομές μέσα από το αρχείο αυτό, μπορούμε να έχουμε πρόσβαση σε όλους τους καταχωρητές του συστήματος.

Έτσι για παράδειγμα η εντολή:

SCU->GPIOIN[4] |= 0x01;

πηγαίνει στον καταχωρητή GPIOIN4 του SCU (SCU_GPIOIN4 σελίδα 115 στο STR91xFA_Reference_Manual.pdf) και θέτει το πρώτο bit, το bit 0, με την τιμή 1. Αφήνει τα υπόλοιπα bit ως είχαν.

Ερώτημα 69
Ποιες αρχικοποιήσεις γίνονται στην αρχή του Blink.c και γιατί;

Μετά εκτελείτε ο κώδικας μέσα στον βρόχο, όπως φαίνεται στον επόμενο πίνακα.

```
while (1) {                    /* Loop forever                */
  for (n = 0x01; n <= 0xFF; n <<= 1) {
    GPIO7->DR[0x3FC] = n;          /* Turn on LED              */
    wait();                    /* Delay                       */
    AD_value = AD_last;            /* Read AD_last value      */
    if (AD_value != AD_last)       /* Make sure that AD interrupt did  */
      AD_value = AD_last;          /* not interfere with value reading */
    AD_value /= 13;                /* Scale to AD_Value to 0 - 78      */
    if (AD_old != AD_value)  {     /* If AD value has changed
      set_cursor (0, 1);
      AD_old = AD_value;
      for (i = 0; i < 16; i++)  {      /* Disp bargraph according to AD    */
        if (AD_value > 5)  {
          lcd_putchar (0x05);
          AD_value -= 5;
           } else  {
          lcd_putchar (AD_value);
                        AD_value = 0;
        }
      }
```

Ερώτημα 70
Περιγράψτε τα βήματα που ακολουθούνται μέσα στον ατέρμονο βρόχο.

Ερώτημα 71
Εφόσον το πρόγραμμα μας εκτελεί τον κώδικα μέσα στον ατέρμονο βρόχο, πως η τιμή του ADC (AD_last) αλλάζει;

Για να γράψουμε στο LCD θα χρησιμοποιήσουμε την συνάρτηση **lcd_print**. Με την συνάρτηση **set_cursor** επιλέγουμε σε ποια γραμμή και θέση θα βρίσκεται ο cursor. Ουσιαστικά επιλέγουμε που θα πάει να γράψει η lcd_print. Το LCD του αναπτυξιακού έχει 2 γραμμές με 16 θέσεις για χαρακτήρες σε κάθε γραμμή. Έτσι για παράδειγμα εάν θέλουμε να γράψουμε στην θέση 5 της πρώτης γραμμής, επιλέγουμε πρώτα την θέση με την συνάρτηση set_cursor(4, 0) (4 και όχι 5 γιατί η πρώτη θέση είναι το 0 και όχι το 1) και μετά γράφουμε με την lcd_print. Σε κάθε γράψιμο ο cursor αυξάνεται αυτομάτως.

Η συνάρτηση **lcd_print** παίρνει ως όρισμα **string**. Η μεταβλητή AD_value είναι **δεκαδικός** και οπότε πρέπει να μετατραπεί η μεταβλητή AD_value σε string.

Ερώτημα 72

Εδώ γράψτε τον κώδικα, ώστε να βλέπουμε στην LCD οθόνη την τάση του ποτενσιόμετρου σε mV.

10 ΕΡΓΑΣΤΗΡΙΟ 10

- **Real Time Clock Μέρος 1ο**
- **Alarm και Periodic Interrupts Μέρος 1ο**

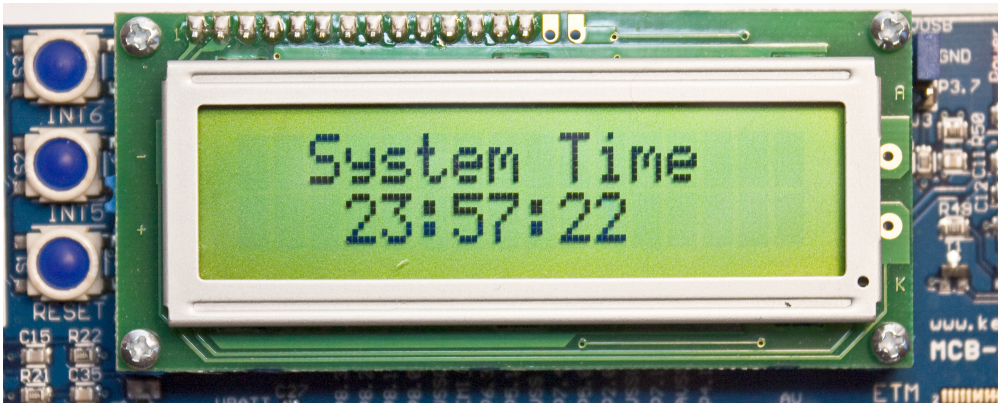
Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαια “Writing Programs”.
- **Αρχεία RTC.c και IRQ.c,** μέσα από τον φάκελο **Lab10_RTC.rar**.
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για τους καταχωρητές του MCU αλλά και για την μονάδα RTC.
- **Βιβλίο Θεωρίας Wayne Wolf, “Οι Υπολογιστές ως Συστατικά Στοιχεία”.** Κεφάλαιο 4, παράγραφος 9. Ρολόι Ξυπνητήρι.

Εισαγωγή

Στο εργαστήριο αυτό θα χρησιμοποιήσουμε το Real Time Clock του μικροελεγκτή και θα υλοποιήσουμε ένα ψηφιακό ρολό. Το ψηφιακό ρολόι έχει την δυνατότητα προγραμματισμού ενός Alarm κάποια συγκεκριμένη χρονική στιγμή. Το Alarm αυτό είναι συνδεδεμένο με ένα Interrupt το οποίο θα ανάβει τα LED για 8 δευτερόλεπτα. Η ώρα εμφανίζετε στο LCD όπως φαίνεται στην παρακάτω φωτογραφία.



Σχήμα 39: Οθόνη Ψηφιακού Ρολογιού

Το Real Time Clock του μικροελεγκτή είναι ένα ενσωματωμένο κομμάτι υλικού και το μόνο που χρειάζεται για να λειτουργήσει εκτός από την ανάλογη τάση, που μπορεί να δοθεί και ξεχωριστά από το υπόλοιπο IC, είναι ένας απλός κρύσταλλος 32,768KHz.

Τα κύρια χαρακτηριστικά του RTC είναι:

- Ημερολόγιο 9999 ετών σε BCD (Binary Coded Decimal).
- Φορμάτ 24 ωρών σε BCD (Binary Coded Decimal).
- Ξεχωριστή είσοδος ρολογιού και τάσεως ώστε να μπορεί να λειτουργεί ολοκληρωτικά αυτόνομα ακόμα και από μία μπαταρία χωρίς να το επηρεάζει εάν το υπόλοιπο κομμάτι του μικροελεγκτή είναι σε τάση.
- Υποστήριξη δίσεκτου έτους.

- Προγραμματιζόμενο Alarm Interrupt μέγιστης διάρκειας ενός μήνα.
- Ευκρίνεια σε millisecond.
- Τα περιεχόμενα των καταχωρητών της ώρας και της ημερομηνίας είναι τα μόνα στον μικροελεγκτή STR912FAW44 που δεν επιστρέφουν στις προκαθορισμένες (default) τιμές τους μετά απο Reset.
- Όταν ο μικροελεγκτής είναι σε Sleep Mode οι ταχύτητες των εσωτερικών ρολογιών μειώνονται αρκετά για εξοικονόμηση ενέργειας. Η λειτουργία του RTC όμως δεν επηρεάζεται μιας και έχει δικό του εσωτερικό δίκτυο ρολογιού που δημιουργείται από τον εξωτερικό κρύσταλλο.
- Εκτός από το Alarm Interrupt μπορεί να δημιουργήσει και περιοδικά Interrupts ανά 1024Hz, 128Hz, 16Hz και 2Hz.

Ερώτημα 73
Βρείτε από το STR91xFAxxx.pdf σε ποια pins του μικροελεγκτή πρέπει να συνδεθεί ο 32,768KHz κρύσταλλος. Στην συνέχεια βρείτε από το σχηματικό MCBSTR9_schematic.pdf εάν στα αντίστοιχα pins είναι συνδεδεμένος κάποιος κρύσταλλος και σημειώστε το όνομα που έχει στο σχηματικό.

Στο εργαστήριο αυτό θα εκτελέσουμε το πρόγραμμα και θα δούμε την εναλλαγή της ώρας στο LCD. Το Alarm είναι προγραμματισμένο να δώσει ένα Interrupt στα 30sec από την αρχική ώρα εκκίνησης εκτέλεσης του προγράμματος. Θα παρατηρήσουμε ότι μετά από αυτά τα 30sec όλα τα LED ανάβουν. Παραμένουν αναμμένα για 8sec και μετά σβήνουν.

Στην συνέχεια θα κάνουμε αλλαγές στον κώδικά μας για να ξεκινάμε το ρολόι από διαφορετική ώρα αλλά και για να πειραματιστούμε με το Alarm Interrupt.

10.1 Εκτέλεση του κώδικα

Ανοίγουμε το Project **Lab10_RTC.uvproj** αποσυμπιέζοντας το **Lab10_RTC.rar** αρχείο, που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Ανοίγουμε τα αρχεία **RTC.c** και **IRQ.c**.

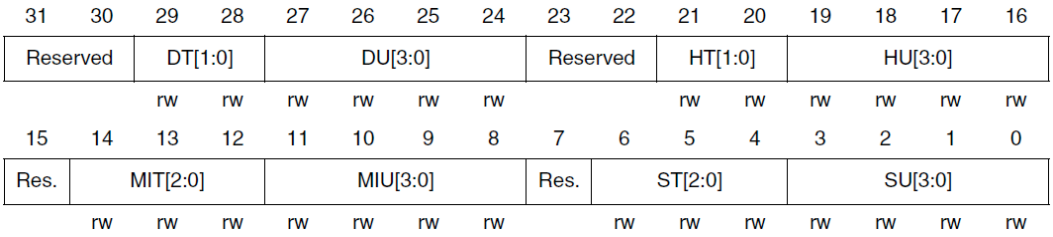
Εκτελούμε τον κώδικα σύμφωνα με την διαδικασία που αναφέρεται στην παράγραφο 2.1 και 2.2.

Παρατηρούμε την εναλλαγή της ώρας. Περιμένουμε 30sec για να χτυπήσει το Alarm Interrupt και ανάψουν τα LED. Πατάμε το Reset για επανεκκίνηση.

10.2 Ανάλυση του κώδικα

Παρακάτω θα αναλύσουμε την διαδικασία προγραμματισμού του μικροελεγκτή για το παράδειγμα του Ψηφιακού Ρολογιού. Το σύστημα βασίζεται στην λειτουργία του Real Time Clock (RTC) και στις δυνατότητες που έχει για προγραμματισμό Alarm και παραγωγή Interrupt.

Δύο ειδών Interrupt έχουν προγραμματιστεί στο RTC. Το Alarm Interrupt και το Periodic Interrupt. Το Periodic Interrupt είναι υπεύθυνο για την ανανέωση της ώρας στην οθόνη. Είναι προγραμματισμένο να χτυπάει κάθε 2Hz (0.5sec) και οπότε κάθε 2Hz η τιμή του καταχωρητή RTC_TR που κρατάει την ώρα, διαβάζετε και τυπώνετε στην οθόνη. Το διάβασμα του καταχωρητή RTC_TR γίνεται με έναν ιδιαίτερο τρόπο μιας και ο καταχωρητής κρατάει την ώρα σε μορφή BCD με τα χαμηλά bit να αντιστοιχούν στις μονάδες και τα υψηλά στις δεκάδες. **Διαβάζουμε το STR91xFA_Reference_Manual.pdf σελίδα 147.** Στο επόμενο σχήμα φαίνεται η δομή του καταχωρητή RTC_TR.



Σχήμα 40: Καταχωρητής RTC_TR

T

Bytes	Περιγραφή	Τιμές
0-3	Seconds Μονάδες	0-9
4-6	Seconds Δεκάδες	0-5
8-11	Minutes Μονάδες	0-9
12-14	Minutes Δεκάδες	0-5
16-19	Hour Μονάδες	0-9
20-21	Hour Δεκάδες	0-2
24-27	Date Μονάδες	1-9
28-29	Date Δεκάδες	0-3

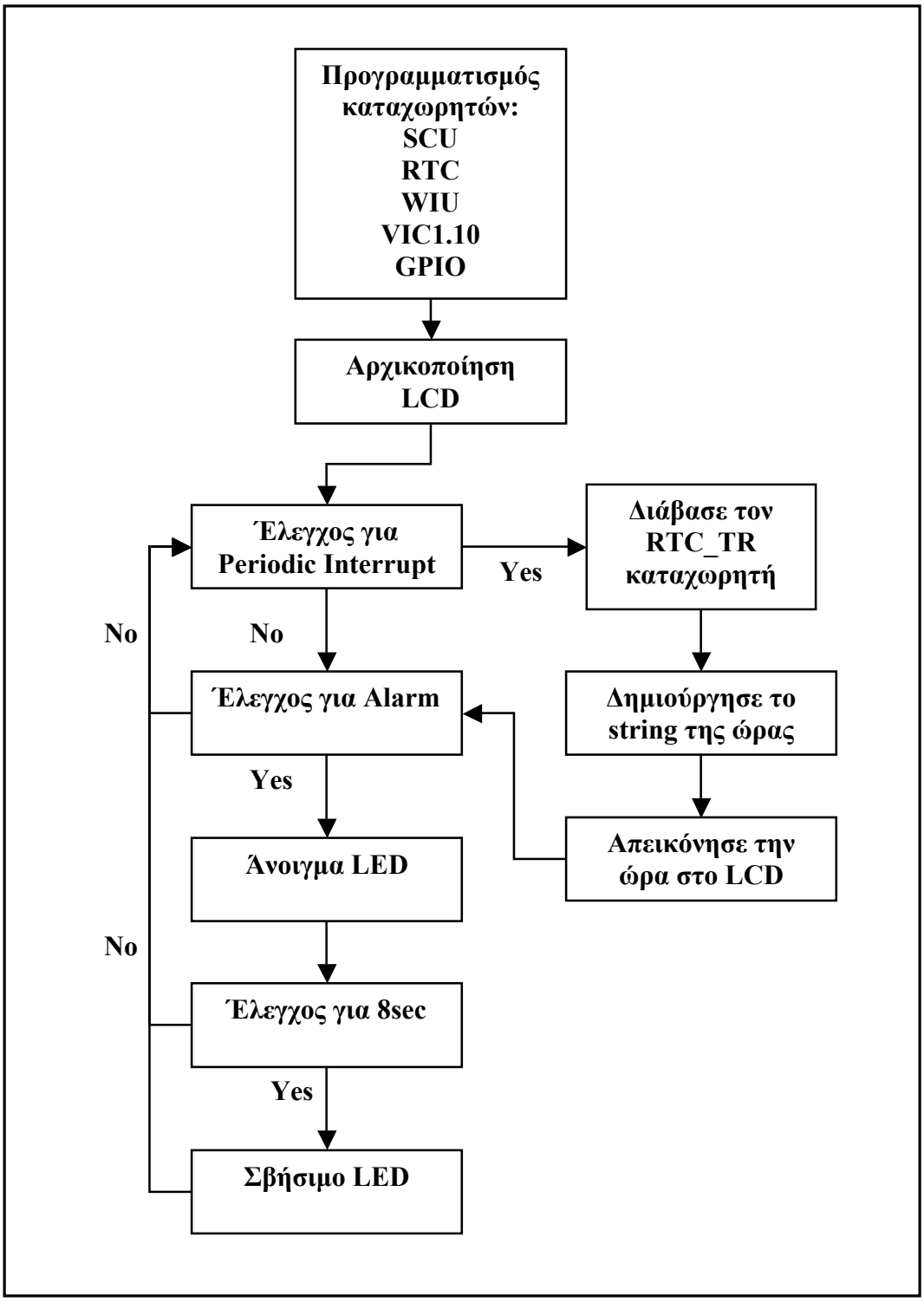
Σχήμα 41: Πίνακας τιμών καταχωρητή RTC_TR

10.2.1 Αποτύπωση της ώρας

Η ώρα αποτυπώνεται στην οθόνη με την συνάρτηση **lcd_print** μετά από την παρακάτω διαδικασία:

```
if (new_time_value){
        get_time_str(time_str);
        set_cursor(3, 1);
        lcd_print(time_str);
        new_time_value = 0;
}
```

Το παρακάτω σχηματικό μας δείχνει την ροή του προγράμματος μας.



Σχήμα 42: Ροή προγράμματος Ψηφιακού Ρολογιού

Η **lcd_print** συνάρτηση, κοιτώντας μέσα στο αρχείο **LCD.c** ορίζεται ως

void lcd_print (unsigned char const *string)

και οπότε δέχεται ως όρισμα string. Άρα τα περιεχόμενα του RTC_TR καταχωρητή πρέπει να μετατραπούν στην αντίστοιχη μορφή Για να μπορέσουν να τυπωθούν.

Η συνάρτηση που καλούμε για να μας διαβάσει τις τιμές του καταχωρητή RTC_TR και να τις μετατρέψει σε μορφή string είναι η εξής:

```
int get_time_str(char *time)
{

sprintf(time,"%02d:%02d:%02d",get_hour(),get_minutes(),get_seconds());
return 0;

}
```

Η get_time_str με την σειρά της καλεί τρεις επιπλέον συναρτήσεις για να δημιουργήσει το string με την χρήση της **sprintf**. Οι τρεις αυτές συναρτήσεις είναι οι παρακάτω:

```
int get_seconds(void)
{
        int seconds = 0;
        seconds = bcd2dec(RTC->TR & 0x7F);
        return seconds;
}

int get_minutes(void)
{
        int minutes = 0;
        minutes = bcd2dec((RTC->TR & 0x7F00) >> 8);
        return minutes;
}

int get_hour(void)
{
        int hour = 0;
        hour = bcd2dec((RTC->TR & 0x3F0000) >> 16);
        return hour;
}
```

Κάθε μία από τις παραπάνω συναρτήσεις διαβάζει το αντίστοιχο μέρος του RTC_TR καταχωρητή που χρειάζεται. Έτσι για παράδειγμα, η συνάρτηση **get_seconds** θα διαβάσει τα πρώτα 7bits του καταχωρητή για να πάρει μόνο την τιμή των seconds. Η τιμή αυτή είναι σε μορφή BCD και πρέπει να

μετατραπεί σε δεκαδικό (decimal). Η παρακάτω συνάρτηση, μετατρέπει μια τιμή της μορφής BCD σε δεκαδική και μας την επιστρέφει.

```
int bcd2dec(int bcd)
{
        bcd = bcd & 0xFF;
        return ((bcd>>4)*10) + bcd%16;
}
```

Εφόσον η συνάρτηση get_time_str δημιουργήσει το string της ώρας με την παραπάνω διαδικασία, το επιστρέφει έμμεσα μέσα από την ίδια της την παράμετρο.

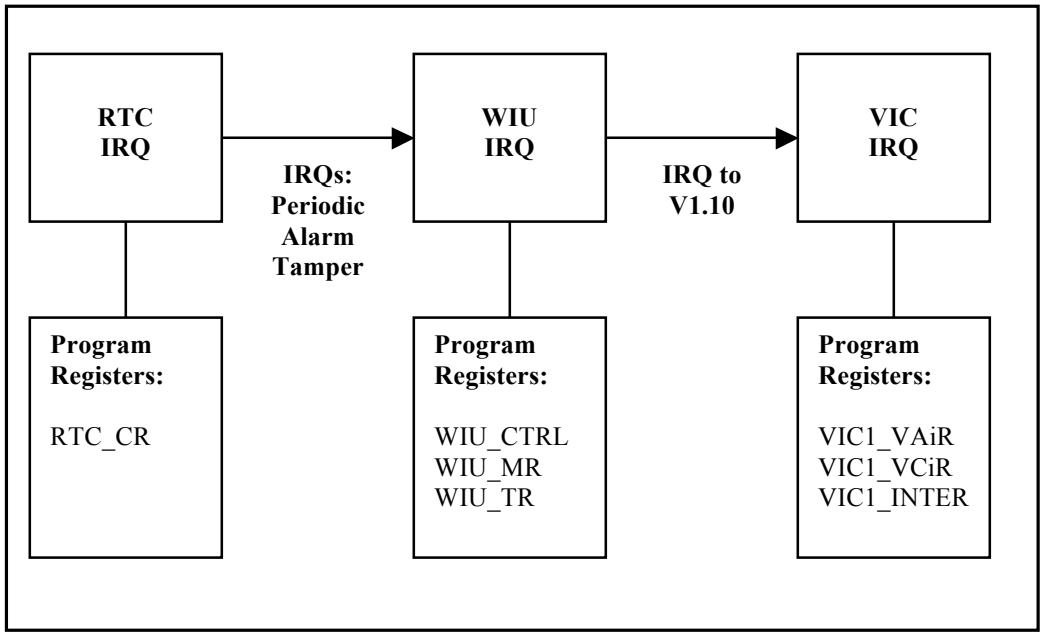
Για να τυπωθεί η ώρα στην οθόνη πρέπει η συνθήκη,

if (new_time_value)

να είναι αληθής. Η τιμή της μεταβλητής new_time_value γίνεται 1 όταν ένα Periodic Interrupt εξυπηρετηθεί.

10.2.2 Periodic RTC Interrupt

Όπως ήδη έχουμε αναφέρει το RTC μπορεί να δημιουργεί περιοδικές διακοπές σε χρόνους 1024Hz, 128Hz, 16Hz και 2Hz. Για να συμβεί αυτό ο ανάλογος προγραμματισμός των απαραίτητων καταχωρητών πρέπει να προηγηθεί. Τα RTC Interrupts συνδέονται στα Vector Interrupts μέσω της μονάδας Wake-UP Interrupt όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 43: Διαδικασία RTC Interrupt

Την περιοδική διακοπή την προγραμματίσαμε για τον λόγο να την χρησιμοποιούμε στην ανανέωση της οθόνης. Μας παρέχει έναν σταθερό χρόνο καθυστέρησης χωρίς την χρήση κάποιου for ή while loop που θα μας μπλόκαρε το σύστημα. Ο σταθερός χρόνος καθυστέρησης, δηλαδή της περιοδικής διακοπής είναι 0,5sec. Οπότε κάθε 0,5sec διαβάζουμε την ώρα και την γράφουμε στο LCD. Θα μπορούσαμε βέβαια μέσα σε έναν ατέρμονο βρόχο να διαβάζουμε συνεχώς τον καταχωρητή των δεδομένων της ώρας και κάθε φορά να γράφουμε στο LCD είτε έχει αλλάξει η ώρα είτε όχι, αλλά αυτό θα δημιουργούσε μεγάλο φόρτο.

10.2.3 Alarm RTC Interrupt

Το Alarm Interrupt του RTC ακολουθεί την ίδια λογική με αυτή των περιοδικών διακοπών όπως φαίνεται στο Σχήμα 42. Δηλαδή περνάει μέσα από την μονάδα WIU για να φθάσει στα Vector Interrupts. Η κύρια διαφορά ανάμεσα στα Alarm και στα Periodic Interrupts είναι ότι ο Alarm Interrupt πρέπει να ξαναπρογραμματιστεί εφόσον χτυπήσει, εάν πρέπει να τον ξαναχρησιμοποιήσουμε μιας και ο χρόνος που είχε οριστεί για να χτυπήσει έχει πλέον περάσει.

10.3 Δοκιμή στις αλλαγές ώρας και Alarm

Παρακάτω θα προσπαθήσουμε να κάνουμε αλλαγές στον κώδικα ώστε να δώσουμε διαφορετική αρχική τιμή στην ώρα. Επίσης θα πρέπει να διαβάσουμε το Κεφάλαιο 5 RTC από το κείμενο STR91xFA_Reference_Manual.pdf για να απαντήσουμε μερικές από τις παρακάτω ερωτήσεις.

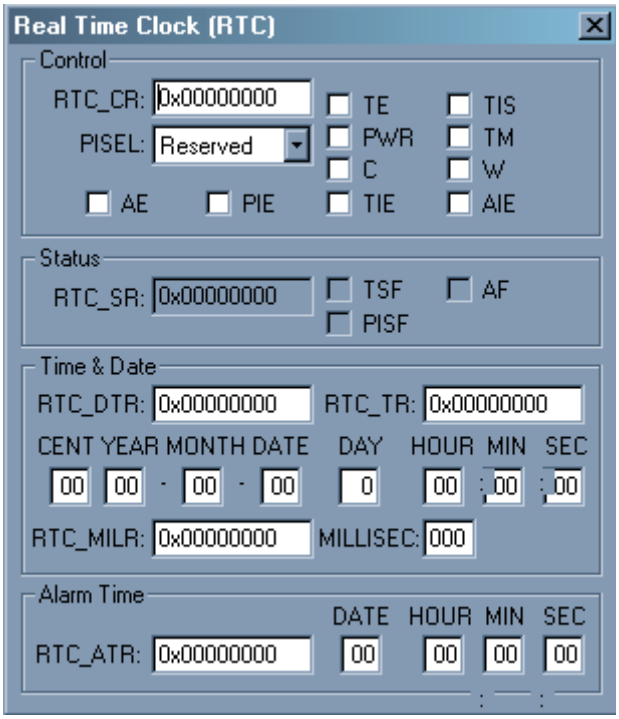
- Βρείτε στον κώδικα, στο αρχείο RTC.c, που ορίζεται η αρχικοποίηση της ώρας και αλλάξτε την.

Ερώτημα 74
Ποιες είναι οι μεταβλητές που αρχικοποιούν την ώρα;

Ερώτημα 75
Ποια είναι ή διαδικασία που πρέπει να ακολουθείτε για να προγραμματίζετε η ώρα, η ημερομηνία και ο συναγερμός στο RTC;

- Κατεβάστε στην πλακέτα τον κώδικα με τις αλλαγές σας και παρατηρήστε εάν οι αλλαγές σας προγραμματίστηκαν κανονικά.

- Πειραματιστείτε με την συχνότητα του Periodic Interrupt δοκιμάζοντας και τις 4 διαφορετικές συχνότητες.
- από το παράθυρο Real Time Clock (RTC) στην Debug λειτουργία του uVision4 μπορούμε να παρακολουθούμε τους καταχωρητές της μονάδας RTC κατά την διάρκεια εκτέλεσης του προγράμματος μας.



Σχήμα 44: Διάβασμα των RTC καταχωρητών στην Debug λειτουργία

11 ΕΡΓΑΣΤΗΡΙΟ 11

- **Real Time Clock Μέρος 2ο**
- **Alarm και Periodic Interrupts Μέρος 2ο**

Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαια “Writing Programs”.
- **Αρχεία RTC.c, LCD.c και IRQ.c,** μέσα από τον φάκελο **Lab10_RTC.rar**.
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για τους καταχωρητές του MCU αλλά και για την μονάδα RTC.
- **Βιβλίο Θεωρίας Wayne Wolf, “Οι Υπολογιστές ως Συστατικά Στοιχεία”.** Κεφάλαιο 4, παράγραφος 9. Ρολόι Ξυπνητήρι.

Εισαγωγή

Στο εργαστήριο αυτό θα αναπτύξουμε το προηγούμενο παράδειγμα του ψηφιακού ρολογιού φτιάχνοντας νέες συναρτήσεις για την ρύθμιση του Alarm αλλά και της ημερομηνίας.

11.1 Προγραμματισμός Alarm

Χρησιμοποιώντας τις παρακάτω σταθερές, όπως αντιστοίχως και με την ώρα, και δημιουργώντας νέες συναρτήσεις προγραμματίστε την ώρα και την ημερομηνία του Alarm.

```
#define ALARM_SECONDS
#define ALARM_MINUTES
#define ALARM_HOUR
#define ALARM_DAY
```

Ερώτημα 76

Εδώ συμπληρώστε τον επιπλέον κώδικα που χρειάζεται για την υλοποίηση.

11.2 Απεικόνιση του Alarm στο LCD

Επιπλέον προσθέστε λογική η οποία θα χρησιμοποιεί το πλήκτρο INT5 ως απλό GPIO (όχι ως WIU Interrupt). Πατώντας το και για όσο χρόνο το κρατάτε πατημένο απεικονίστε στο LCD της ρύθμισης του Alarm όπως φαίνεται στην παρακάτω φωτογραφία. Οι ρυθμίσεις του Alarm πρέπει να διαβάζονται από τον αντίστοιχο καταχωρητή στην μονάδα RTC.

Σχήμα 45: Απεικόνιση Alarm ενώ το πλήκτρο INT5 είναι πατημένο

Ερώτημα 77
Εδώ συμπληρώστε τον επιπλέον κώδικα που χρειάζεται για την υλοποίηση του πατήματος του πλήκτρου και ένδειξη του Alarm.

12 ΕΡΓΑΣΤΗΡΙΟ 12

- **Real Time Clock Μέρος 3ο**
- **Alarm και Periodic Interrupts Μέρος 3ο**

Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαια “Writing Programs”.
- **Αρχεία RTC.c και IRQ.c,** μέσα από τον φάκελο **Lab10_RTC.rar**.
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για τους καταχωρητές του MCU αλλά και για την μονάδα RTC.
- **Βιβλίο Θεωρίας Wayne Wolf, “Οι Υπολογιστές ως Συστατικά Στοιχεία”.** Κεφάλαιο 4, παράγραφος 9. Ρολόι Ξυπνητήρι.

Εισαγωγή

Στο εργαστήριο αυτό θα αναπτύξουμε το προηγούμενο παράδειγμα του ψηφιακού ρολογιού φτιάχνοντας νέες συναρτήσεις και λογική για την ρύθμιση και απεικόνιση της ημερομηνίας στο LCD.

12.1 Προγραμματισμός και απεικόνιση ημερομηνίας

Χρησιμοποιώντας τις παρακάτω σταθερές, όπως και με το Alarm, δημιουργήστε νέες συναρτήσεις ώστε να προγραμματίσετε την ημερομηνία στο RTC.

```
#define DATE
Π
#define MONTH
α
#define YEAR
ρ
#define CENTURY
ά
#define WEEKDAY
λ
```

ληλα απεικονίστε την πληροφορία της ημερομηνίας στην πρώτη γραμμή του LCD όπως φαίνεται στην παρακάτω φωτογραφία. Στην δεύτερη γραμμή του LCD παραμένει η απεικόνιση της ώρας.

Σχήμα 46: Απεικόνιση της ώρας και της ημερομηνίας στο LCD

Ερώτημα 78

Εδώ συμπληρώστε τον επιπλέον κώδικα που χρειάζεται για την υλοποίηση του προγραμματισμού και απεικόνιση της ημερομηνίας.