**Εργαστήριο 7**

**02/12/2020**

**ΥΠΟΡΟΥΤΙΝΕΣ, ΚΑΙ ΕΝΑΣ ΠΟΛΥ ΑΠΛΟΣ ΔΡΟΜΟΛΟΓΗΤΗΣ**

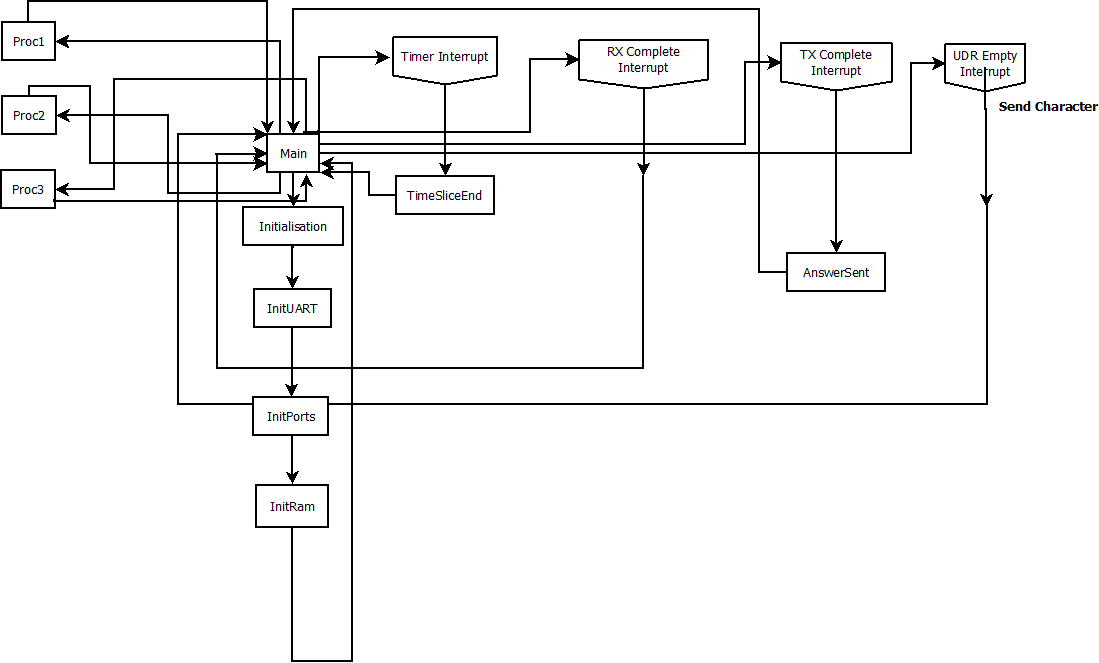
Ντουνέτας Δημήτρης

ΑΜ: 2016030141

# Εισαγωγή

Σκοπός του εργαστηρίου είναι η δημιουργία ενός απλού δρομολογητή για την δρομολόγηση 3 απλών υπορουτίνων σε Μικροελεγκτή AVR. Επιπλέον, η παρατήρηση της assembly εξόδου από τον C Compiler και πως κάνει χρήση των πόρων του συστήματος. Τέλος η παρατήρηση για τη λειτουργία χωρίς προβλήματα και πανωγράψιμο της μνήμης και χρήση USART του ATMEGA16, για παραλαβή εντολών και αποστολή απαντήσεων.

# Block Diagram Προγράμματος

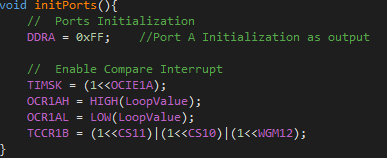


# Αρχικοποίηση του προγράμματος σε Γλώσσα C

Αρχικά το πρόγραμμα ξεκινάει από την συνάρτηση main που βρίσκεται στο αρχείο main.c . Η main συνάρτηση είναι υπεύθυνη για την αρχικοποίηση του προγράμματος και της μνήμης καθώς και τηλ λειτουργία του βρόγχου που εκτελεί τον Scheduler. Για γίνει η αρχικοποίηση η Main καλεί τη συνάρτηση initialization. O Stack Pointer δεν χρειάζεται να αρχικοποιηθεί αφού τον αρχικοποιεί ο C Compiler από μόνος του ώστε να μπορούμε να επιστρέφουμε σωστά από τις ρουτίνες και τις συναρτήσεις που καλούμε.

Η Initialization στην συνέχεια καλεί τις initUΑRT, initRam και initPorts συναρτήσεις οι οποίες κάνουν τις αρχικοποιήσεις μνήμης , καταχωρητών, θέσεων μνήμης και Ports όπως αναφέρθηκαν στο προηγούμενο εργαστήριο. Συνοπτικά χρησιμοποιούνται 3 σημεία στην μνήμη SRAM στα οποία το ένα με όνομα UARTAnsAddress όπου γράφεται η αυτοματοποιημένη απάντηση που δίνει ο Μικροελεγκτής μας όταν λάβει ένα ολόκληρο Instruction σωστά δηλαδή OK<CR><LF>. Tο δεύτερο, είναι το InstrAddress όπου αποθηκεύεται το instruction που λαμβάνουμε ώστε να μπορούμε να το διαχειριστούμε κατάλληλα όταν θέλουμε να εκτελέσουμε την εντολή. Τέλος το τρίτο είναι το κομμάτι που αποθηκεύουμε τις μεταβλητές που χρειαζόμαστε για να λειτουργούν σωστά οι υπορουτίνες μας και ο Scheduler.

# 



# RAM MAP

|  |  |  |  |
| --- | --- | --- | --- |
| 0x90 | 0x91 | 0x92 | 0x93 |
| UARTAnsByte7(O) | UARTAnsByte6(K) | UARTAnsByte5(CR) | UARTAns4(LF) |

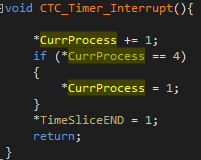
|  |  |  |  |
| --- | --- | --- | --- |
| 0xΑ0 | 0xΑ1 | 0xΑ2 | 0xΑ3 |
| instrByte0 | instrByte1 | instrByte2 | instrByte3 |

Στις θέσεις ορισμένες ως instrByte εισάγεται διαδοχικά η λέξη της κάθε εντολής που λαμβάνουμε.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0xΒ0 | 0xΒ1 | 0xΒ2 | 0xΒ3 | 0xΒ4 | 0xΒ5 | 0xΒ6 | 0xΒ7 |
| RingCounterL | RingCounterR | LedToggle | TimeSliceEnd | currProcess | Proc1Active | Proc2Active | Proc3Active |

# Υπολογισμός του TimeSlice

Για τον υπολογισμό του TimeSlice χρησιμοποιούμε τον 16-bit μετρητή Timer/Counter 1 σε CTC mode με prescale 1/64 και τιμή Compare = 15625. Έτσι όταν ο μετρητής μετρήσει μέχρι αυτή τη τιμή θα έχουν περάσει 100ms σύμφωνα με το ρολόι του Μικροελεγκτή χρονισμένο στα 10Mhz. Έτσι για να υποδηλώσουμε τη λήξη του TimeSlice κάνουμε την τιμή του 1.

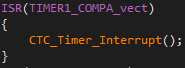


## **Λειτουργία του κύριου προγράμματος και τα Ιnterrupt του USART(UART).**

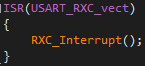
Καθώς το πρόγραμμα έχει ξεκινήσει και βρίσκεται στον ατέρμονα βρόγχο καλούνται τα κατάλληλα interrupts και αυτά χρησιμοποιούμε για να δώσουμε λειτουργικότητα στο πρόγραμμα μας. Αυτή τη στιγμή υπάρχουν 4 Interrupts τα οποία αξιοποιεί το πρόγραμμα για να δείξει στην οθόνη ότι εμείς του δίνουμε ως είσοδο μέσω του UART. Τα Interrupts αυτά είναι:

1. Timer/Counter Compare1 Interrupt
2. USART RX Complete Interrupt
3. USART DATA Registry Empty Interrupt
4. USART TX Complete Interrupt

* To Timer/Counter 1 Compare A Interrupt είναι υπεύθυνο για το σωστό χρονισμό του timeslice.

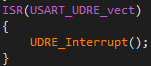


* Καλεί την συνάρτηση CTC\_TIMER\_Interrupt η οποία μας δίνει την τιμή 1 στη μεταβλητή TimeSliceEND και προσθέτει +1 στην τιμή του currProcess. Αν το currProcess πάρει την τιμή 4 το αρχικοποιεί ξανά στη τιμή 1
* Το USART RX Complete Interrupt είναι υπεύθυνο για το σωστό διάβασμα όταν σηκωθεί το Flag RXC το οποίο υποδηλώνει ότι έχουμε λάβει κάποιον χαρακτήρα έτοιμο να διαβάσουμε.



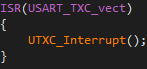
Καλεί την συνάρτηση RXC \_Interrupt η οποία λαμβάνει ένα χαρακτήρα και τον αποθηκεύει. Επίσης ελέγχει αν έχει ολοκληρωθεί η λήψη της εντολής, καθώς και ποια εντολή έχει ληφθεί.

* Το USART DATA Registry Empty Interrupt είναι υπεύθυνο να ενημερώνει όταν το UDR καταχωρητής είναι διαθέσιμος για εγγραφή ώστε να μην απανογράφουμε χωρίς να έχουν αποσταλεί πρώτα τα δεδομένα που θέλουμε.



Καλεί την συνάρτηση UDRE\_Interrupt η οποία στέλνει έναν χαρακτήρα στο UDR ώστε αυτός να αποσταλεί ορθά από τον USART.

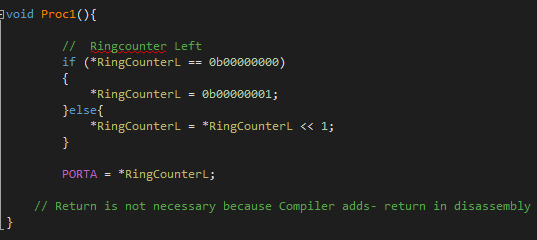
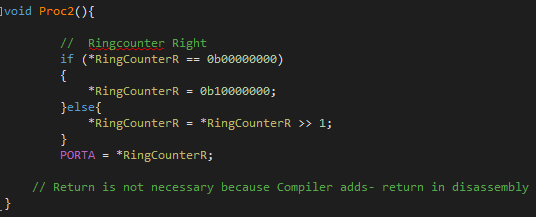
* To USART TX Complete Interrupt είναι υπεύθυνο για την ενημέρωση της αποστολής ενός χαρακτήρα. Μας ενημερώνει ότι η αποστολή έχει ολοκληρωθεί και ο χαρακτήρας έχει φτάσει στον προορισμό του. Η διαφορά με το DATA Registry Empty Interrupt είναι ότι περιμένει μέχρι το Byte να φύγει εντελώς από τον Shift Register που αποστέλλει σειριακά και έτσι μας είναι χρήσιμο μόνο σε HALF-Duplex πρωτόκολλα ή όταν θέλουμε να κάνουμε ενέργεια μετά το πέρας της αποστολής. Έτσι το χρησιμοποιούμε για να οριστικοποιήσουμε την αποστολή όλης της απάντησης του Μικροελεγκτή μας.

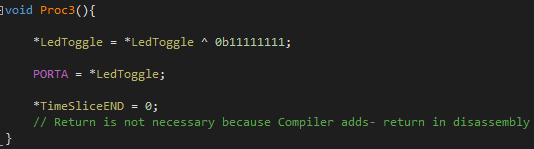


# Οι Υπορουτίνες

Οι υπορουτίνες που θα διαχειρίζεται ο απλός δρομολογητής είναι 3 απλές διεργασίες.

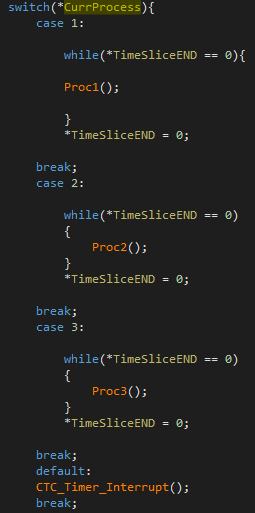
Η πρώτη είναι ένας αριστερόστροφος μετρητής δακτυλίου, η δεύτερη ένας δεξιόστροφος μετρητής δακτυλίου και η τρίτη μια διαδικασία εναλλαγής Led από 01010101 σε 10101010 και αντίστροφα. Είναι πολύ απλές διεργασίες καθώς αυτή ήταν η απαίτηση της άσκησης με τη σημαντική λεπτομέρεια ότι όλες έχουν ως έξοδο το PortA του Μικροελεγκτή. Τέλος, δεν γίνεται χρήση της εντολής Return μετά το τέλος κάθε υπορουτίνας καθώς δεν είναι αναγκαίο στον C Compiler και στο disassembly προστίθεται πάντα η εντολή Ret. Έτσι το πρόγραμμα ελέγχθηκε και δεν γεμίζει άσκοπα το Stack με διευθύνσεις επιστροφής.





# Ο απλός Δρομολογητής

Στο σώμα της συνάρτησης Main υλοποιείται ο απλός δρομολογητής που εναλλάσσει κατάλληλα τις διεργασίες κάθε 100ms. Σε αρχική φάση και για το εργαστήριο 7 δεν ενεργοποιούνται – απενεργοποιούνται διεργασίες και έτσι οι 3 διεργασίες εναλλάσσονται διαδοχικά η μια μετά την άλλη. Για να γίνει αυτό, απλά επιλέγουμε τη διεργασία που θα τρέξει σύμφωνα με τη τιμή που έχει η μεταβλητή currProcess. Η διεργασία τρέχει και αφού τελειώσει επιστρέφει πίσω εκεί που κλήθηκε όπου και ξανά καλείται για όσο διαρκεί το κάθε Timeslice. Το Timeslice τελειώνει όταν ο Timer/ Counter που έχουμε υλοποιήσει μετρήσει 100 ms και τότε αλλάζει τη τιμή του TimeSliceEND από 0 σε 1. Βγαίνουμε από τον βρόγχο κλήσης της υπορουτίνας και τότε κάνουμε ξανά το TimeSliceEND 0 ώστε να συνεχίσει η επόμενη υπορουτίνα.



# Τελικό Αποτέλεσμα

Το πρόγραμμα τρέχει συνεχόμενα χωρίς να σταματάει αφού ως βασικό μέρος έχει έναν ατέρμονα βρόγχο. Τρέχει διαδοχικά τις διεργασίες 1 έως 3 και δείχνει στο PortA τα αποτελέσματα τους. Κάθε φορά που ξανακαλείται μια διεργασία συνεχίζει από το σημείο που είχε τελειώσει προηγουμένως καθώς κρατάμε αποθηκευμένη στην SRAM του Μικροελεγκτή της πληροφορία που δημιουργεί κάθε διαεργασία. Έτσι αφού δε χρησιμοποιούμε κάποιο καταχωρητή για τη διαδικασία αυτή και αφού έχουμε δεσμεύσει μνήμη μόνο για τη συγκεκριμένη διεργασία δεν χάνουμε τα δεδομένα που έχει εκτελέσει κάθε διεργασία ούτε από πανωγράψιμο μνήμης ούτε καταχωρητή.