



Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

ΗΡΥ 411 – Ενσωματωμένα Συστήματα Μικροεπεξεργαστών

Χειμερινό εξάμηνο ακαδ. έτους 2021-22

Διδάσκων: Καθ. Απ. Δόλλας

## Εργαστήριο 5

### Κώδικας C στον Atmel AVR

Ονοματεπώνυμο φοιτητή...: Λιουδάκης Εμμανουήλ

Αρ. Μητρώου φοιτητή.....: 2018030020

*Χρησιμοποιήθηκε το λογισμικό Microchip Studio 7.0 με επιλεγμένο μικροελεγκτή τον "ATmega 16".*

## Σκοπός της άσκησης

Σκοπός της άσκησης ήταν η περαιτέρω εξοικείωση με την ανάπτυξη κώδικα C (που είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου) για τον Atmel AVR, αλλά και την εποπτεία της χρήσης των πόρων του σε χαμηλό επίπεδο.

## Τεχνολογία που χρησιμοποιήθηκε

Χρησιμοποιήθηκε ο μικροελεγκτής Atmel ATmega16 για τις προσομοιώσεις του προγράμματος, σε περιβάλλον ανάπτυξης Microchip Studio 7. Η λειτουργικότητα του κώδικα επιβεβαιώθηκε μέσω προσομοιώσεων, με αρχεία stimuli (.stim), όπως και στα εργαστήρια 3 και 4.

## Υλοποίηση

### Τροποποιήσεις στον κώδικα του εργαστηρίου 4

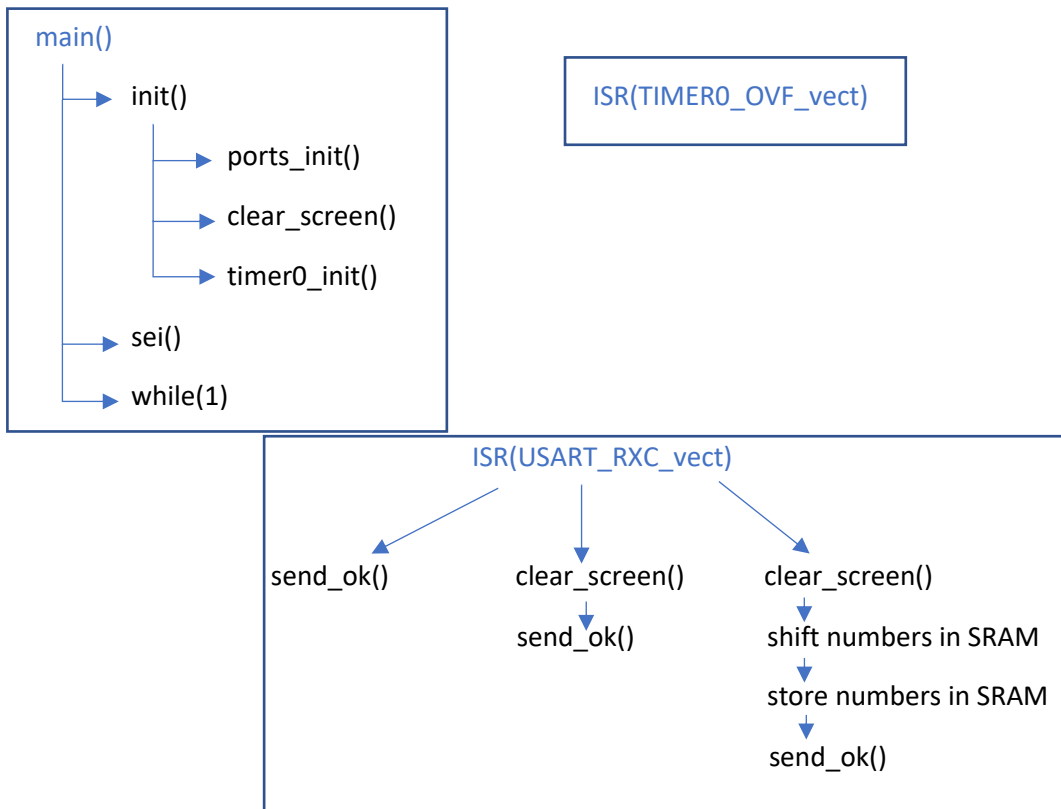
Η λειτουργικότητα του κώδικα παραμένει ακριβώς η ίδια. Απλά, τα δυο τμήματα που είχαν μείνει σε assembly, ο interrupt handler για τον Timer0, ο οποίος ανανεώνει τα ψηφία της οθόνης, και η εγγραφή των δεδομένων για την αποκωδικοποίηση στη flash, επανασυντάχθηκαν σε C. Για την εγγραφή/ανάγνωση δεδομένων στη flash, έγινε χρήση της ενότητας 3.5 του Application Note “Atmel AVR4027: Tips and Tricks to Optimize Your C Code for 8-bit AVR Microcontrollers”, διαθέσιμο στο σύνδεσμο:

<https://ww1.microchip.com/downloads/en/AppNotes/doc8453.pdf>

Επίσης, στο εργαστήριο 4 χρησιμοποιήθηκε ο καταχωρητής `r17` ως global μεταβλητή. Πλέον, η global μεταβλητή συνεχίζει να υπάρχει, αλλά «αποσυνδέεται» από τον καταχωρητή. Ο compiler θα δεσμεύσει χώρο για αυτή στη μνήμη SRAM, όπως θα εξηγηθεί παρακάτω, στο χάρτη μνήμης.

### Σύνοψη των βασικών δομικών στοιχείων του κώδικα

Ένα απλοποιημένο block diagram που περιγράφει τη λειτουργικότητα του κώδικα είναι το εξής:



Η μόνη διαφορά σε σχέση με το εργαστήριο 4, είναι η αλλαγή του ονόματος του interrupt handler για το συμβάν `TIMERO_OVF`.

Το κυρίως πρόγραμμα `main()` καλεί ορισμένες συναρτήσεις αρχικοποίησης, ενεργοποιεί τα interrupts και στη συνέχεια εκτελεί έναν ατέρμονα βρόχο. Έπειτα, οι δυο βασικές διεργασίες `ISR(TIMERO_OVF_vect)` και `ISR(USART_RXC_vect)` αναλαμβάνουν η μια την ανανέωση της οθόνης (χωρίς να καλεί άλλες συναρτήσεις) και η άλλη την επικοινωνία με τη σειριακή θύρα και την αποθήκευση των ληφθέντων δεδομένων (καλώντας ορισμένες βοηθητικές συναρτήσεις).

#### *Προσομοίωση και έλεγχος λειτουργίας του προγράμματος*

Η προσομοίωση του προγράμματος γίνεται ακριβώς όπως και στο εργαστήριο 4, με τα ίδια αρχεία `stimuli`. Δημιουργήθηκε ένα αντίγραφο του κώδικα C, το `main_sim.c`, στο οποίο αποφεύγεται η χρήση του καταχωρητή `UDR`, ώστε να είναι δυνατή η προσομοίωση.

Για τη λήψη δεδομένων θεωρούμε ότι αυτά έρχονται από τον καταχωρητή `TCNT2`, ενώ για την αποστολή δεδομένων, τα τοποθετούμε στον καταχωρητή `TCNT1L`.

Στον interrupt handler για τη λήψη δεδομένων από τη σειριακή θύρα διαβάζεται δυο φορές η τιμή του καταχωρητή `UDR` (θα διαβάζουμε πάντα `0x00` κατά την προσομοίωση). Αυτό πρέπει να γίνει δυο φορές, ώστε να απενεργοποιηθεί το flag `RXC` (`USART Receive Complete`). Ύστερα, διαβάζονται από τον καταχωρητή `TCNT2` τα δεδομένα, σαν να έρχονταν από τον `UDR`.

Έπειτα, στην συνάρτηση `send_ok`, αντί οι χαρακτήρες προς αποστολή να γράφονται στον καταχωρητή `UDR`, γράφονται στον `TCNT1L`, ώστε να είναι δυνατή η παρακολούθηση της εξόδου μέσω ενός απλού log file.

#### *Επαλήθευση λειτουργίας με το stimuli file “simulation.stim”*

Όπως και στο εργαστήριο 4, για την επαλήθευση της λειτουργίας του προγράμματος, συντάχθηκε το αρχείο `simulation.stim` το οποίο αποστέλλει την παρακάτω σειρά εντολών στο πρόγραμμα:

`AT<CR><LF>`

`C<CR><LF>`

`N123<CR><LF>`

`N01229763<CR><LF>`

Κατά την προσομοίωση, παρέχεται ένας ASCII χαρακτήρας ανά 10000 κύκλους του ρολογιού, όπως έχει εξηγηθεί και στην προηγούμενη αναφορά.

Οι απαντήσεις που δίνει το πρόγραμμα στη σειριακή θύρα (στον καταχωρητή `TCNT1L` για την προσομοίωση) θα έπρεπε να γράφονται στο αρχείο `simulation.log`. Ενώ ο `TCNT1L` λαμβάνει τις σωστές τιμές (ένα – ένα τους χαρακτήρες της απάντησης), δεν κατάφερα να κάνω log τις τιμές του καταχωρητή αυτού σε ένα αρχείο, ούτε σε αυτό το εργαστήριο, οπότε η παρατήρηση γίνεται χειροκίνητα, κατά την κλήση της συνάρτησης `send_ok`. Επίσης, για την επιβεβαίωση της λειτουργίας του προγράμματος, πρέπει να παρακολουθούνται:

- Τα περιεχόμενα του array `digits[0]...digits[7]` της μνήμης, που περιέχουν τα ψηφία που θα εμφανιστούν στην οθόνη (κατά την εκτέλεση εντολών “N” και “C”).

- Οι τιμές των PortA και PortC, ώστε να βεβαιωθεί ότι ανανεώνεται σωστά η οθόνη.

Για την απόλυτη επιβεβαίωση της λειτουργίας του προγράμματος, μπορούμε επίσης να παρακολουθούμε τις τιμές των τοπικών μεταβλητών όποτε καλείται μια συνάρτηση, όπως παρακολουθήσαμε στα προηγούμενα εργαστήρια τις τιμές των καταχωρητών που χρησιμοποιούσαμε έμμεσα σαν τοπικές μεταβλητές (όταν είχαμε κώδικα σε assembly).

Προφανώς η προσομοίωση για οποιαδήποτε εντολή μας δίνει τα ίδια αποτελέσματα με το εργαστήριο 4. Παρέχεται επίσης το αρχείο stim\_report.stim, το οποίο τροφοδοτεί την εφαρμογή με την εντολή N123<CR><LF> όπως και στο προηγούμενο εργαστήριο. Εκτελώντας την προσομοίωση, παρατηρείται η ίδια ακολουθία γεγονότων όπως και στην προηγούμενη αναφορά.

#### Σχολιασμός του κώδικα Assembly που προέκυψε

Αφού γίνει η μεταγλώττιση του προγράμματος, προκύπτει ο κώδικας assembly στο αρχείο .iss.

Πλέον όλος ο assembly κώδικας παράγεται μόνο από τον compiler, επομένως μπορούμε να παρατηρήσουμε τα εξής όσον αφορά τη μορφή του:

- Οι εντολές αρχικοποίησης της στοίβας παράγονται αυτόματα από τον compiler.
- Στις συναρτήσεις, δεν αποθηκεύει στη στοίβα την τιμή του SREG, όπως έκανα εγώ χειροκίνητα στα προηγούμενα εργαστήρια, ούτε και τις τιμές των καταχωρητών που χρησιμοποίησε. Αυτό οφείλεται στις συμβάσεις χρήσης καταχωρητών που ακολουθεί ο compiler.

Σε γενικές γραμμές πάντως, ο κώδικας που παράχθηκε παρουσιάζει μεγάλη ομοιότητα με τον πλήρη κώδικα assembly που είχε συνταχθεί στο εργαστήριο 3.

#### Χρήση πόρων – χάρτης μνήμης

Ο πίνακας των δεδομένων της αποκωδικοποίησης έχει αποθηκευτεί στις διευθύνσεις 0x0054...0x005F της flash.

Στην SRAM, έχουμε τέσσερις μεταβλητές, οι οποίες έχουν αποθηκευτεί ως εξής:

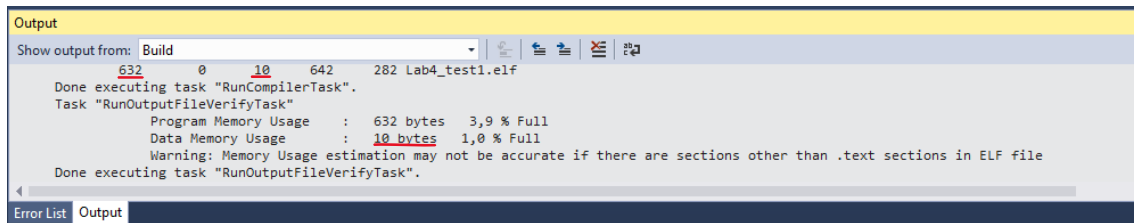
- addr\_show : στη διεύθυνση 0x0060
- num\_cnt : στη διεύθυνση 0x0061
- flag\_reg : στη διεύθυνση 0x0062
- digits : στις διευθύνσεις 0x0063 έως 0x0070 (πρόκειται για array 8 θέσεων)

Τις πληροφορίες αυτές τις λαμβάνουμε προσθέτοντας τις παραπάνω μεταβλητές στο παράθυρο παρακολούθησης (Watch), κατά τη διάρκεια του debugging:

Watch 1		
Name	Value	Type
flag_reg	0x00	uint8_t(data)@0x0062
num_cnt	0x00	uint8_t(data)@0x0061
addr_show	0x00	uint8_t(data)@0x0060
digits	0x0063	uint8_t[8](data)@0x0063
dec_data	0x0054	uint8_t[12](prog)@0x0054

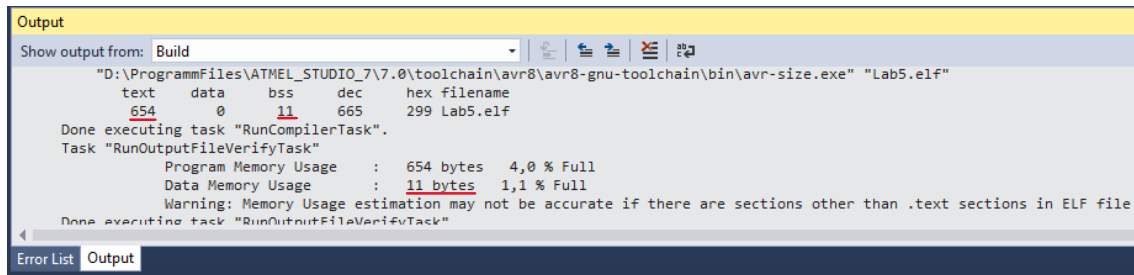
## Σύγκριση της χρήσης πόρων με το προηγούμενο εργαστήριο

Στο εργαστήριο 4, η χρήση των πόρων του AVR ήταν:



```
Output
Show output from: Build
632 0 10 642 282 Lab4_test1.elf
Done executing task "RunCompilerTask".
Task "RunOutputFileVerifyTask"
Program Memory Usage : 632 bytes 3,9 % Full
Data Memory Usage : 10 bytes 1,0 % Full
Warning: Memory Usage estimation may not be accurate if there are sections other than .text sections in ELF file
Done executing task "RunOutputFileVerifyTask".
Error List Output
```

Ενώ στο παρόν εργαστήριο, έχει μεταβληθεί σε:



```
Output
Show output from: Build
"D:\ProgramFiles\ATMEL_STUDIO_7\7.0\toolchain\avr8\avr8-gnu-toolchain\bin\avr-size.exe" "Lab5.elf"
text data bss dec hex filename
654 0 11 665 299 Lab5.elf
Done executing task "RunCompilerTask".
Task "RunOutputFileVerifyTask"
Program Memory Usage : 654 bytes 4,0 % Full
Data Memory Usage : 11 bytes 1,1 % Full
Warning: Memory Usage estimation may not be accurate if there are sections other than .text sections in ELF file
Done executing task "RunOutputFileVerifyTask".
Error List Output
```

Η χρήση της SRAM έχει μεταβληθεί από 10 σε 11 bytes, εξαιτίας της αποσύνδεσης της global μεταβλητής `addr_show` από τον καταχωρητή `r17`. Αυτό φαίνεται στη γραμμή “Data Memory Usage”, αλλά και 4 γραμμές παραπάνω, στο μέγεθος του `bss` memory section.

Αντίστοιχα, η χρήση της flash μνήμης έχει αυξηθεί από 632 σε 654 bytes, πράγμα που οφείλεται στις επιπλέον εντολές που παράχθηκαν από τη μεταγλώττιση του `interrupt handler` για τον `Timer0` (όλος ο υπόλοιπος κώδικας είναι ίδιος στα δυο εργαστήρια).

## Συμπέρασμα

Όταν αναπτύσσουμε κώδικα για τον AVR σε γλώσσα C, δημιουργείται η εντύπωση ότι είναι μικρότερος σε έκταση. Αυτό δεν ισχύει, διότι με τη μεταγλώττιση πιθανώς να παραχθεί εκτενέστερος κώδικας. Εξαιρετικό παράδειγμα αποτελεί η εντολή ανάγνωσης δεδομένων από τη flash μνήμη, `pgm_read_byte(<addr>)`, η οποία κατέληξε σε 4 εντολές `assembly`, μία `ldi`, μία `sbc`, μία `subi` και τελικά μια `lpm`. Αν είχαμε συντάξει εξ αρχής τον κώδικα σε `assembly`, πιθανώς να υπολογίζαμε τη διεύθυνση με μία μόνο εντολή π.χ. μια `addi`, άρα συνολικά δυο εντολές (`addi` και `lpm`).

Τέλος, γίνεται αντιληπτό ότι σε πολλές περιπτώσεις ίσως η συγγραφή κώδικα σε `assembly` μας εξυπηρετεί, όσον αφορά τη χρήση πόρων. Στο προσεχές ερχόμενο project του μαθήματος, πιθανώς κάποια τμήματα του κώδικα να συνταχθούν σε `assembly`, είτε για εξοικονόμηση χώρου (αν δεν χωράει όλο το πρόγραμμά μας στη flash), είτε για ταχύτερη εκτέλεση (διότι μπορεί η μεταγλώττιση να αυξήσει το πλήθος των εντολών (τη χρονικό κόστος) κάποιας συνάρτησης). Από το εργαστήριο 3, με χρήση flash 512 bytes, φτάσαμε στο εργαστήριο 5 στα 654 bytes, για το ίδιο πρόγραμμα. Εφόσον η flash ενός τυπικού μικροελεγκτή είναι της τάξεως των 8KB, αυτή η διαφορά των 142 bytes δεν είναι καθόλου αμελητέα.