Εργαστήριο 2

Ονομα: Μαυρογιώργης Δημήτρης
ΑΜ: 2016030016
ΗΡΥ 411 - Ενσωματωμένα Συστήματα Μικροεπεξεργατών
Πολυτεχνειο Κρητης

April 19, 2021

Σκοπός

Σκοπός του δεύτερου εργαστηρίου είναι η περαιτέρω εξοικείωση μας με το περιβάλλον ανάπτυξης μικροελεγκτών AVR. Ειδικότερα, στόχος του εργαστηρίου είναι η δημιουργία ενός προγράμματος για οδήγηση με πολυπλεξία στον χρόνο μίας οθόνης 7-segment LED οκτώ ψηφίων. Για την πολυπλεξία το χρόνο, χρησιμοποιήθηκε ο timer/counter0 και τα interrupt από το πρώτο εργαστήριο, προκειμένου να διαιρέσουμε το χρόνο σε slots και σε κάθε slot να έχουμε ενεργό ένα LED στο οποίο εμφανίζουμε έναν αριθμό BCD.

Περιγραφή της υλοποίησης

Για το συγκεκριμένο εργαστήριο χρησιμοποιήθηκαν η θύρα A του AVR για το περιεχόμενο του κάθε 7-segment και η θύρα C για την ενεργοποίηση του κάθε ψηφίου που βγαίνει στην οθόνη. Επιπλέον, τόσο τα δεδομένα προς εμφάνιση, όσο και τα δεδομένα για το ποιο LED ανάβουμε βρίσκονται στην κυρίως μνήμη. Πιο συγκεκριμένα, για τα περιεχόμενα των 7-segment χρησιμοποιήθηκε η διεύθυνση 0x60 έτσι, ώστε κάθε φορά που διαβάζουμε έναν BCD αριθμό προς εμφάνιση να προσθέτουμε στη διεύθυνση αυτή τον αριθμο και να παίρνουμε τη διεύθυνση στην οποία υπάρχει η αποκωδικοποίηση του αριθμού για τα 7-segments. Επίσης, για τα δεδομένα προς εμφάνιση χρησιμοποιήθηκε η διεύθυνση 0x76.

Παράλληλα, επειδή ήταν αναγκαία η χρήση ενός timer, για την ανανέωση των LED, χρησιμοποιήθηκε ο κώδικας του δεύτερου μερους του πρώτου εργαστηρίου με τον timer/counter0 και τα interrupts. Πιο ειδικά, για το κάθε LED θέλαμε μια ελάχιστη συχνότητα 30 Hz. Επομένως, το 1 ms του πρώτου εργαστηρίου ήταν επαρκές για την ανανέωση του κάθε ψηφίου.

Επεξήγηση Κώδικα
Αρχικά, στο data segment πρέπει να κάνουμε memory allocation για τις δέκα αποκωδικοποιήσεις των BCD αριθμών, καθώς και για τους οχτώ αριθμούς που θέλουμε να εμφανίσουμε. Επιπρόσθετα, αρχικοποιούμε τους καταχωρητές DDRA και DDRC με 0xFF, επειδή η θύρα A και C είναι έξοδοι, καθώς και τους PORTA και PORTC με 0xFF και 0x00 αντίστοιχα.

Επίσης, οι αρχικοποιήσεις των καταχωρητών TCNT, TCCR0 και TIMSK είναι οι ίδιες με του εργαστηρίου 1, καθώς θέλουμε κάθε 1 ms να ανανεώνονται τα LED. Αφού έγινε η αποθήκευση των 18 τιμών στη μνήμη, αρχικοποιούμε τους καταχωρητές X και Y με τις διευθύνσεις 0x60 και 0x76 που αποθηκεύσαμε τα δεδομένα. Οι συγκεκριμένοι καταχωρητές θα χρησιμεύσουν

για να διαβάζουμε κάθε φορά από τη μνήμη τους οχτώ αριθμούς. Παράλληλα, όπως και στο 1ο εργαστήριο η main συνάρτηση του προγράμματος είναι ένα ατέρμονο loop στο οποίο εκτελούμε NOP, καθώς περιμένουμε να έρθει το interrupt του timer/counter0 για να εμφανίσουμε το επόμενο ψηφίο.

Όσον αφορά τον interrupt handler, αρχικά ελέγχουμε αν έχουμε ήδη εμφανίσει όλους τους αριθμούς. Στην περίπτωση αυτή, κάνουμε reset τον digit counter, καθώς και τον καταχωρητή Y με τον οποίο διαβάζουμε τους αριθμούς BCD. Αντίθετα, αν δεν έχουμε τελειώσει με όλη την ανανέωση των ψηφίων, κάθε φορά σβήνουμε τα 7-segment βγάζοντας ως έξοδο 0xFF, διαβάζουμε τον BCD αριθμό στον καταχωρητή R19, υπολογίζουμε τη διεύθυνση που βρίσκεται η αποκωδικοποίηση του προσθέτοντάς στον XL τον R19 και τη διαβάζουμε από τη μνήμη. Ωστόσο, πριν διαβάσουμε από τη μνήμη την αποκωδικοποιημένη μορφή, επαναφέρουμε στον καταχωρητή XL την τιμή 0x60 έτσι, ώστε κάθε φορά να μπορούμε να υπολογίσουμε τη σωστή διεύθυνση.

Έπειτα, για την ενεργοποίηση του κατάλληλου ψηφίου ελέγχουμε αν βρισκόμαστε το πρώτο ψηφίο ή κάποιο από τα επόμενα. Αν θέλουμε να εμφανίσουμε το πρώτο ψηφίο κάνουμε "1" το C-bit του Status Register και χρησιμοποιούμε την εντολή ROL για να ολισθήσουμε έναν "1" στα αριστερά και να επιτύχουμε τη λειτουργικότητα του ring counter. Έπειτα, κάνουμε clear το C-bit και βγάζουμε στο PORTC το αποτέλεσμα του καταχωρητή R16 στον οποίο έχουμε αποθηκεύσει την τιμή 0x00 κατα την αρχικοποίηση των θυρών. Αντίθετα, για τα υπόλοιπα ψηφία, απλά ολισθαίνουμε τον "1", επειδή κάθε φορά ενεργοποιούμε ένα LED.

Τέλος, βγάζουμε στην έξοδο τα περιεχόμενα του καταχωρητή R18 στον οποίο έχει αποθηκευτεί η αποκωδικοποιημένη μορφή, αρχικοποιούμε τον TCNT ξανά και αυξάνουμε το digit counter.

```
ldi r16, 0xC0             //bcd to 7-segment decoding and storing in sram
sts seg_dec_addr, r16

ldi r16, 0xF9
sts seg_dec_addr+1, r16

ldi r16, 0xA4
sts seg_dec_addr+2, r16

ldi r16, 0xB0
sts seg_dec_addr+3, r16

ldi r16, 0x99
sts seg_dec_addr+4, r16

ldi r16, 0x92
sts seg_dec_addr+5, r16

ldi r16, 0x82
sts seg_dec_addr+6, r16

ldi r16, 0xF8
sts seg_dec_addr+7, r16

ldi r16, 0x80
sts seg_dec_addr+8, r16

ldi r16, 0x90
sts seg_dec_addr+9, r16
```

(a) Store the decoded numbers for 7-segment

```
ldi r16, 0x01             //Start storing the 8 numbers to display from 1 to 8
sts data_addr, r16

inc r16
sts data_addr+1, r16

inc r16
sts data_addr+2, r16

inc r16
sts data_addr+3, r16

inc r16
sts data_addr+4, r16

inc r16
sts data_addr+5, r16

inc r16
sts data_addr+6, r16

inc r16
sts data_addr+7, r16
```

(b) Store the numbers to display

```
.equ F_CPU=10000000

.dseg
.org 0x60
    seg_dec_addr: .byte 10
.org 0x76
    data_addr: .byte 8

.def digit_cnt = r20

.cseg
.org 0x00           //Address of the beginning of the program
    rjmp start
.org 0x12           // Address ofthe interrupt handler
    rjmp handler
```

(c) Memory allocation for our data in SRAM

```
    ldi digit_cnt, 0x00 //Initialize digit counter to 0x00

    ldi XL, LOW(seg_dec_addr)
    ldi XH, HIGH(seg_dec_addr)

    ldi YL, LOW(data_addr)
    ldi YH, HIGH(data_addr)

    rjmp main


main:
    nop         //We are doing no op while we wait for the interrupt
    rjmp main
```

(d) Initialize X and Y register for reading data

```
handler:
        cpi digit_cnt, 0x08         //Check if we have dispaly all the digits
        brne done
        ldi digit_cnt, 0x00         //Reset digit_cnt (r20) with 0 to start counting the digits again
        ldi YL, LOW(data_addr)      //Reset YL register to the first digit data address
        ldi YH, HIGH(data_addr)     //Reset YL register to the first digit data address
    done:
        out PORTA, r17      //r17 has the value 0xFF from initialization of DDRA-C and PORTA-C

        ld r19, Y+      //Read the bcd number

        add Xl, r19     //Calculate the address in which the decoded number exists
        ld r18, X
        sub Xl, r19     //

        cpi digit_cnt, 0x00     //Check if we have to dispaly the 1st digit
        brne else
        in r16, PORTC
        sec             //Set carry for ROL instr
        rol r16
        clc             //Clear carry
        out PORTC, r16
        jmp done1
```

(a) Interrupt handler - part A

```
else:                   //We already have one "1" and we need to just shift left
    in r16, PORTC
    rol r16
    out PORTC, r16
done1:
    out PORTA, r18

    ldi r16, 0x63       //Set once again the value of TCNT0 in order to roll again after the handling
    out TCNT0, r16

    inc digit_cnt

    reti
```

(b) Interrupt handler - part B

Προσομοίωση Αποτελεσμάτων

Όπως φαίνεται στην εικόνα 3 όλοι οι καταχωρητές DDRA, DDRC, PORTA και PORTC αρχικοποιούνται με τις σωστές τιμές, όπως περιγράφτηκε παραπάνω. Στην ίδια εικόνα βλέπουμε, επίσης, ότι οι δέκα αποκωδικοποιημένοι αριθμοί, καθώς και οι οχτώ αριθμοι από 1 εως 8 που θέλουμε να εμφανίσουμε στα 7-segments, έχουν αποθηκευτεί σωστά στην SRAM του μικροελεγκτή AVR.

Παράλληλα, στις εικόνες (a) εως (h) βλέπουμε ότι κάθε φορά που έρχεται το interrupt, εκτελείται σωστά ο handler και κάθε φορά εμφανίζεται στο κατάλληλο ψηφίο η σωστή τιμή που διαβάσαμε από τη μνήμη. Επιπλέον, μια σημαντική παρατήρηση είναι ότι στο PORTC κάθε φορά υπάρχει ένας "1", δηλαδή ενεργοποιείται ένα ψηφίο τη φορά και επομένος εμφανίζουμε τον αριθμό BCD που διαβάσαμε μόνο στο ενεργοποιημένο 7-segment display.

Πιο αναλυτικά, στην εικόνα (b) μετά την εκτέλεση του handler, η έξοδος PORTA έχει την τιμή 0x79 (αριθμός 1 σε BCD) και το PORTC την τιμή 0x01 (ενεργοποιημένο το πρώτο ψηφίο). Στην επόμενη εκτέλεση του handler (εικόνα (c)), η θύρα A, αλλάζει σε 0xA4 (αριθμός 2 σε BCD) και το "1" έχει ολισθήσει στο δεύτερο ψηφίο. Η ίδια διαδικασία επαναλαμβάνεται μέχρι και τον 8ο αριθμό που διβάζουμε από τη μνήμη και ξανά από την αρχή. Αυτό έχει ως αποτέλεσμα τελικά να παρατηρούμε στα οχτώ 7-segment LED τους αριθμούς 1 έως 8 οι οποίοι αποθηκεύτηκαν στη μνήμη.

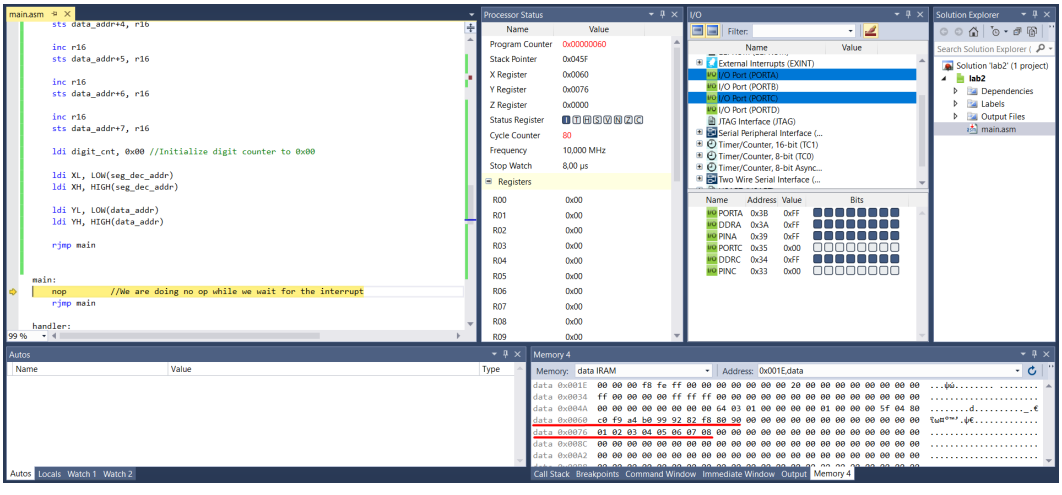
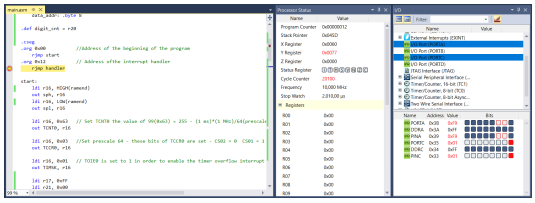
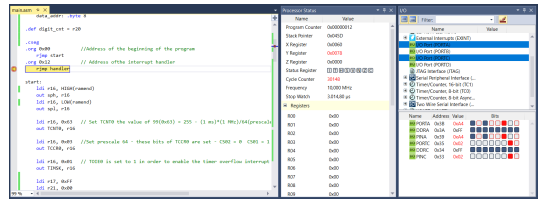


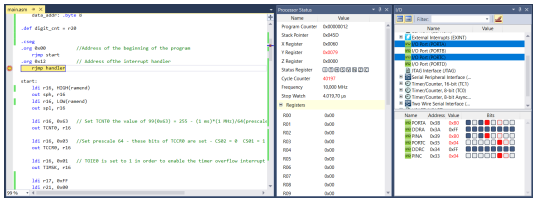
Figure 3: Results from Atmel Studio 7 - Initialization of register and memory



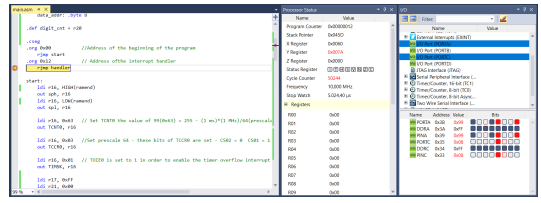
(a) Results from Atmel Studio 7 - Display digit 1



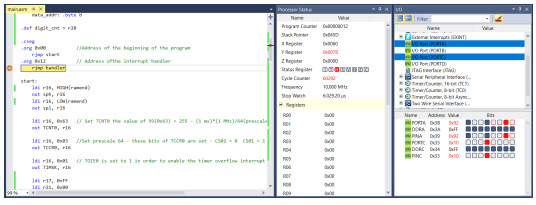
(b) Results from Atmel Studio 7 - Display digit 2



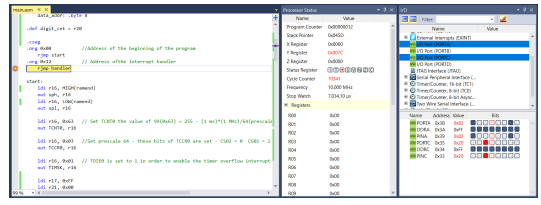
(c) Results from Atmel Studio 7 - Display digit 3



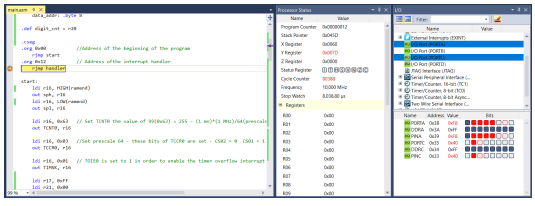
(d) Results from Atmel Studio 7 - Display digit 4



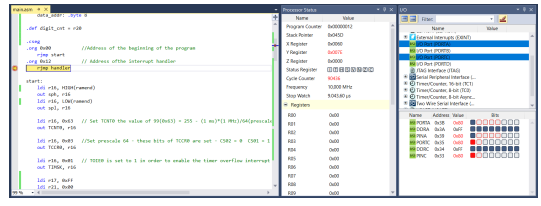
(e) Results from Atmel Studio 7 - Display digit 5



(f) Results from Atmel Studio 7 - Display digit 6



(g) Results from Atmel Studio 7 - Display digit 7



(h) Results from Atmel Studio 7 - Display digit 8