Εργαστήριο 2

16/10/2020

ΕΞΟΙΚΕΙΩΣΗ ΜΕ ΤΗΝ ΟΙΚΟΓΕΝΕΙΑ ΜΙΚΡΟΕΛΕΓΚΤΩΝ ATMEL AVR – Μία Απλή Οθόνη 7-Segment LED

Ντουνέτας Δημήτρης
ΑΜ: 2016030141

Εισαγωγή

Σκοπός του εργαστηρίου είναι η περαιτέρω εξοικείωση με το περιβάλλον ανάπτυξης για μικροελεγκτή AVR, με την δημιουργία απλού προγράμματος για οδήγηση με πολυπλεξία στον χρόνο μίας οθόνης 7-segment LED για (έως) οκτώ ψηφία.

Αρχικοποίηση της SRAM

Για την λειτουργία την οθόνης μας πρέπει αρχικά να αρχικοποιήσουμε 8 θέσεις μνήμης με τα 8 ψηφία που θέλουμε να δείξουμε. Στην συγκεκριμένη υλοποίηση το MSB βρίσκεται στην μεγαλύτερη θέση μνήμης ενώ το LSB βρίσκεται στην μικρότερη. Χρησιμοποιούμε την SRAM για την αποθήκευση των δεδομένων μας που στο datasheet φαίνεται ότι ξεκινά από το σημείο 0X60.

```
;RAM Initialisation
    ldi temp , 0
    sts display_address,temp
    ldi temp , 1
    sts display_address+1,temp
    ldi temp , 2
    sts display_address+2,temp
    ldi temp , 3
    sts display_address+3,temp
    ldi temp , 4
    sts display_address+4,temp
    ldi temp , 5
    sts display_address+5,temp
    ldi temp , 6
    sts display_address+6,temp
    ldi temp , 7
    sts display_address+7,temp
```

Γενικές Αρχικοποιήσεις και Main Πρόγραμμα

Στην αρχή του προγράμματος αρχικοποιούμε όσους καταχωρητές θα χρησιμοποιήσουμε στη συνέχεια. Αρχικά, αρχικοποιούμε το καταχωρητή Y στον οποίο αποθηκεύουμε τη θέση μνήμης του LSB ψηφίου μας. Ύστερα αρχικοποιούμε τους καταχωρητές που χρειάζεται για να δουλέψει ο TIMER/COUNTER 1 σε λειτουργία CTC. Τελικά το πρόγραμμα μπαίνει σε έναν ατέρμονα βρόγχο όπου και παραμένει μέχρι να περάσει το κατάλληλο χρονικό διάστημα το οποίο έμεινε το ίδιο με το προηγούμενο δηλαδή 1 ms που είναι 4 φορές πιο μικρό από το απαιτούμενο των 4 ms δηλαδή συχνότητας 240Hz. Αυτό υλοποιήθηκε με τη λογική ότι αφού ήταν επιτρεπτό θα μας δώσει καλύτερη εικόνα στην οθόνη μας όσο αναφορά το Framerate αφού αυτό θα είναι 120FPS. Ακόμη αρχικοποιείται ένας καταχωρητής τον οποίο χρησιμοποιούμε ως τον Ring Counter που θα εμφανίζεται στο PortC.

Το Interrupt

Κατά τη διάρκεια που το πρόγραμμα είναι στον ατέρμονα βρόγχο και οι κύκλοι ρολογιού περνάνε, ο μετρητής μετράει ώσπου φτάνει στην τιμή που έχουμε δώσει στον OCR1A και σηκώνεται ένα interrupt που διακόπτει αυτό το μέτρημα όπως και στο προηγούμενο εργαστήριο. Τότε ο Program Counter μεταφέρεται στη μνήμη που έχουμε ορίσει το interrupt Handling του συγκεκριμένου Interrupt. Σε αυτό το σημείο υλοποιείται το πιο ενδιαφέρον κομμάτι αυτού του εργαστηρίου. Αρχικοποιούμε ένα καταχωρητή στον οποίο αποθηκεύουμε το digit το οποίο δείχνουμε αυτή τη στιγμή. Έτσι όταν δείξουμε και τα 8 μπορούμε να γυρίσουμε πίσω και να τα ξαναδείξουμε μέχρι αυτά να αλλάξουν. Στη συνέχεια, κάνουμε Load το Bit-0 από τη SRAM που το έχουμε αποθηκευμένο και καλούμε την ρουτίνα μετατροπής αυτού από BCD σε 7-segment. Τέλος το αποτέλεσμα το εμφανίζουμε στο Port A αφού πρώτα του βάλουμε την τιμή 0XFF ώστε να αποφύγουμε δυνατόν λάθη κατά την απεικόνιση. Τέλος κάνουμε αριστερή ολίσθηση στον καταχωρητή που απεικονίζει τον RingCounter μας ώστε στο επόμενο interrupt να απεικονιστεί το επόμενο digit.

```
;interrupt Handler address
    .org OC1Aaddr

    ;Loop the same 8 digits
        cpi digitCounter , 8
        breq Reset_digit_Counter

End_Reset_digit_Counter:
    ;load From Memory next byte
        ld digitReg, Y+
        inc digitCounter
        rcall BCD_TO_7_seg

        ldi temp,0XFF
        out PORTA,temp
        out PORTC,RingCounter
        out PORTA,Seg_data_out

        rol RingCounter

        reti
```

Η Ρουτίνα BCD TO 7-SEGMENT

Αυτή η ρουτίνα είναι ένας απλός αποκωδικοποιητής που ελέγχει αν το ψηφίο που τραβάμε από τη μνήμη είναι σε BCD μορφή και ύστερα το μετατρέπει σε 7-segment μορφή. Αφού γίνει ο έλεγχος πηγαίνει στο κατάλληλο σημείο του προγράμματος όπου το κατάλληλο 7-segment μεταφέρεται στον καταχωρητή που εμφανίζεται στο PortA.

```
BCD_TO_7_seg:
        cpi digitReg,0
        breq ZeroDigit
        cpi digitReg,1
        breq OneDigit
        cpi digitReg,2
        breq TwoDigit
        cpi digitReg,3
        breq ThreeDigit
        cpi digitReg,4
        breq FourDigit
        cpi digitReg,5
        breq FiveDigit
        cpi digitReg,6
        breq SixDigit
        cpi digitReg,7
        breq SevenDigit
        cpi digitReg,8
        breq EightDigit
        cpi digitReg,9
        breq NineDigit
```

```
ZeroDigit:
    ldi Seg_data_out,seg_zero
    ret
OneDigit:
    ldi Seg_data_out,seg_one
    ret
TwoDigit:
    ldi Seg_data_out,seg_two
    ret
ThreeDigit:
    ldi Seg_data_out,seg_three
    ret
FourDigit:
    ldi Seg_data_out,seg_four
    ret
FiveDigit:
    ldi Seg_data_out,seg_five
    ret
SixDigit:
    ldi Seg_data_out,seg_six
    ret
SevenDigit:
    ldi Seg_data_out,seg_seven
    ret
EightDigit:
    ldi Seg_data_out,seg_eight
    ret
NineDigit:
    ldi Seg_data_out,seg_nine
    ret
```

Τελικό Αποτέλεσμα

Το πρόγραμμα μας δουλεύει συνεχώς χωρίς να τερματίζει αφού ως βασικό πρόγραμμα είναι ένας ατέρμονας βρόγχος. Με σκοπό το πρόγραμμα να μπορεί να δείξει για αρκετή ώρα και πολλές φορές τις θέσεις που έχουμε αρχικοποιήσει στην SRAM αλλά και να δείξει και καινούργια έξοδο σε περίπτωση που αλλάξουμε τις τιμές της SRAM. Επειδή δεν αρχικοποιούμε stack pointer στην αρχή του προγράμματος ώστε να γνωρίζει το πρόγραμμα που να επιστρέψει μετά την κλήση του Interrupt, κάθε φορά που επιστρέφουμε από αυτό το πρόγραμμα ξεκινάει από την αρχή και κάνει Reset. Αυτή είναι μια πολύ χρήσιμη ιδιότητα που χρησιμοποιούμε στο πρόγραμμα μας. Κάθε 8 interrupts που δηλαδή έχει δείξει μια φορά όλα τα στοιχεία στην οθόνη ο Μικροελεγκτής, γίνεται μια εκ νέου αρχικοποίηση του ώστε να φορτωθούν στην RAM τυχόν νέες τιμές και να σεταριστούν εκ νέου οι καταχωρητές. Αυτό γίνεται μέσω ενός καταχωρητή που αφού γίνει η πρώτη αρχικοποίηση παίρνει τιμή και κάθε φορά που επιστρέφουμε από το Interrupt ελέγχουμε την τιμή του. Αν αυτή είναι διαφορετική από 1 κάνουμε αρχικοποίηση αλλιώς την αποφεύγουμε και πάμε κατευθείαν στο ατέρμονα βρόγχο.