Ε . Μ . Π . - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2021-2022

ΑΘΗΝΑ 25 Μαΐου 2022

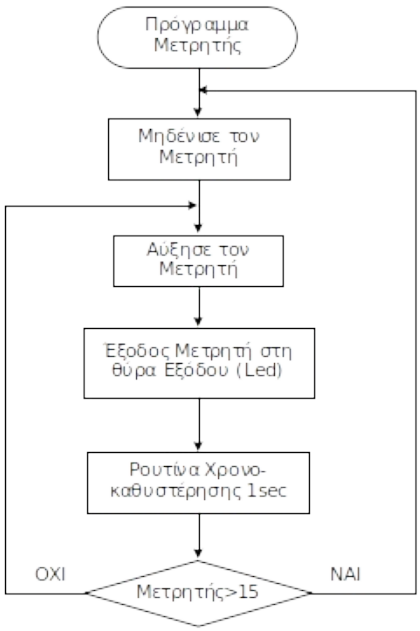
ΕΡΓΑΣΤΗΡΙΟ – ΠΑΡΑΔΕΙΓΜΑΤΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΣΕ AVR
Για το μάθημα "Συστήματα Μικροϋπολογιστών"

Εργ. Άσκ. με απλά παραδείγματα προγραμματισμού (σε assembly και C)
στον Μικροελεγκτή AVR - Χρήση χρονοκαθυστερήσεων και υπορουτινών.

Χρονοκαθυστερήσεις

Μια χρήσιμη εφαρμογή στα συστήματα μικροελεγκτών είναι η λειτουργία και η ανταπόκρισή τους σε εξωτερικές συνθήκες σε τακτά χρονικά διαστήματα. Για το σκοπό αυτό είναι πολύ χρήσιμη η ανάπτυξη σχετικού λογισμικού (υπορουτίνες) που να δημιουργεί ακριβείς και συγκεκριμένες χρονοκαθυστερήσεις και να χρησιμοποιείται από οποιαδήποτε χρονικά εξαρτώμενη εφαρμογή. Βοήθεια για την ανάπτυξη αυτού του κώδικα δίνουν τα τεχνικά χαρακτηριστικά του εκάστοτε μικροελεγκτή και συγκεκριμένα η περίοδος ρολογιού και οι κύκλοι εκτέλεσης κάθε εντολής, από τα οποία προκύπτει ο χρόνος εκτέλεσης κάθε εντολής. Η δημιουργία κώδικα χρονοκαθυστέρησης συνήθως επιτυγχάνεται με τη διαδοχική εκτέλεση μιας σειράς εντολών που δεν παράγουν κανένα χρήσιμο αποτέλεσμα (συνηθίζεται η εντολή nop). Το μέγεθος της σειράς μαζί με κατάλληλους πολλαπλασιαστικούς βρόχους δημιουργούν την επιθυμητή χρονοκαθυστέρηση. Η τεχνική αυτή φαίνεται στην παρακάτω υπορουτίνα **wait_usec**, που για τον μικροελεγκτή AVR ATmega16 και την αναπτυξιακή πλακέτα EasyAVR6 (συχνότητα ρολογιού 8MHz, περίοδος ρολογιού 0.125μsec), είναι μια χρονοκαθυστέρηση τόσων μsec, όση η δυαδική τιμή του καταχωρητή r25:r24 κατά την κλήση. Επίσης παρακάτω δίνεται η ρουτίνα **wait_msec** που αξιοποιεί την προηγούμενη και αυτή προκαλεί χρονοκαθυστέρηση τόσων msec, όση η τιμή του καταχωρητή r25:r24. Οι ρουτίνες αυτές αξιοποιούνται στο επόμενο παράδειγμα

Παράδειγμα 1.1 Να προγραμματίσετε και να επιδείξετε στο εκπαιδευτικό σύστημα easyAVR6 χρονόμετρο δευτερολέπτων που απεικονίζει το χρόνο σε δυαδική μορφή πάνω στα LED PA3-PA0. Το χρονόμετρο όταν φτάνει στην τιμή 15_{10} , στο επόμενο βήμα ξαναρχίζει από την αρχή. Όλο το πρόγραμμα σας δίνετε και το ζητούμενο είναι να περάσει από το AVRStudio αρχικά για προσομοίωση και στη συνέχεια την παραγωγή του εκτελέσιμου κώδικα που πρέπει να κατέβει στην πλακέτα για την επίδειξη της ορθής λειτουργίας στο πραγματικό σύστημα. Ακολουθούν τα αναγκαία προγράμματα και οι ρουτίνες assembly:



Σχήμα 1.1 Πρόγραμμα μετρητής modulo 15.

Πίνακας 1.1 Πρόγραμμα σε assembly μετρητή modulo 15

```
.include "m16def.inc"

reset:   ldi r24 , low(RAMEND)          ; initialize stack pointer
         out SPL , r24
         ldi r24 , high(RAMEND)
         out SPH , r24
         ser r24                        ; initialize PORTA for output
         out DDRA , r24
         clr r26                ; clear time counter

main:    out PORTA , r26
         ldi r24 , low(1000)        ; load r25:r24 with 1000
         ldi r25 , high(1000)       ; delay 1 second
         rcall wait_msec
         inc r26                    ; increment time counter, one second passed
         cpi r26 , 16               ; compare time counter with 16
         brlo main                  ; if lower goto main, else clear time counter
         clr r26                    ; and then goto main
         rjmp main

wait_msec:
         push r24           ; 2 κύκλοι (0.250 μsec)
         push r25           ; 2 κύκλοι
         ldi r24 , low(998)         ; φόρτωσε τον καταχ.  r25:r24 με 998 (1 κύκλος - 0.125 μsec)
         ldi r25 , high(998)        ; 1 κύκλος (0.125 μsec)
         rcall wait_usec            ; 3 κύκλοι (0.375 μsec), προκαλεί συνολικά καθυστέρηση 998.375 μsec
                            ; Στην προσομοίωση δεν βάζουμε χρονοκαθυστέρηση!
         pop r25            ; 2 κύκλοι (0.250 μsec)
         pop r24            ; 2 κύκλοι
         sbiw r24 , 1       ; 2 κύκλοι
         brne wait_msec             ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
         ret                ; 4 κύκλοι (0.500 μsec)

wait_usec:
         sbiw r24 ,1                ; 2 κύκλοι (0.250 μsec)
         nop                ; 1 κύκλος (0.125 μsec)
         nop                ; 1 κύκλος (0.125 μsec)
         nop                ; 1 κύκλος (0.125 μsec)
         nop                ; 1 κύκλος (0.125 μsec)
         brne wait_usec             ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
         ret                ; 4 κύκλοι (0.500 μsec)
```

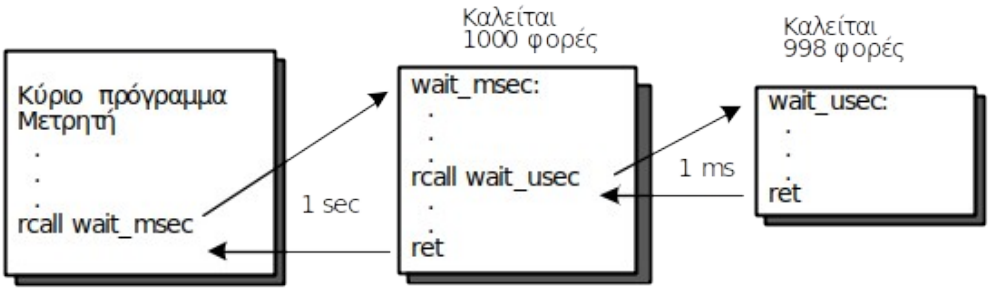
Ρουτίνα: wait_μsec
Προκαλεί καθυστέρηση τόσων μsec, όση η τιμή του καταχωρητή r25:r24
Είσοδος: Ο χρόνος (1 - 65535 μs) μέσω του καταχωρητή r25:r24
Καταχωρητές: r25:r24

Από τα σχόλια φαίνεται ότι ο παραπάνω κώδικας, όταν εκτελείται ο επαναληπτικός βρόχος, απαιτεί 8 κύκλους ρολογιού ή 1μsec. Άρα, όσες φορές εκτελεστεί ο βρόχος, τόσα μsec καθυστέρησης απαιτούνται. Η μικροδιαφορές που προκύπτουν από την μια φορά που θα εκτελεστεί η έξοδος από το βρόχο και η εντολή επιστροφής (ret), μπορούν αν απαιτηθεί να συνυπολογιστούν στον κώδικα που καλεί την υπορουτίνα wait_usec. (αναλυτικά, η υπορουτίνα wait_usec με είσοδο r25:r24=*n* καθυστερεί *n*-1+0.875+0.500=*n*+0.375 μsec).

Ρουτίνα: wait_msec

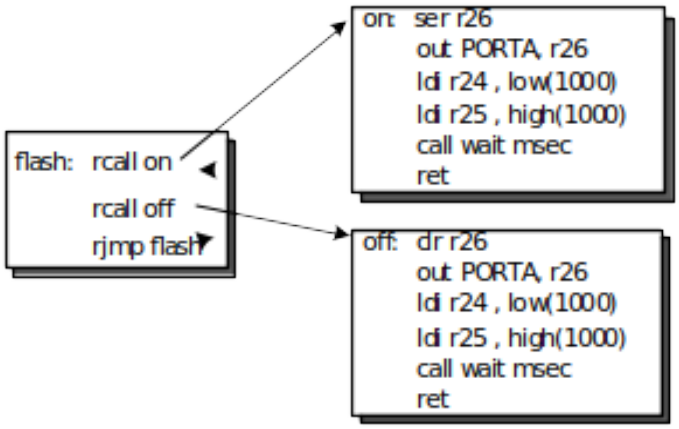
Προκαλεί καθυστέρηση τόσων msec, όση η τιμή του καταχωρητή r25:r2
Είσοδος: Ο χρόνος (1 - 65535 ms) μέσω του καταχωρητή r25:r24
Καταχωρητές: r25:r24
Καλούμενες υπορουτίνες: wait_usec

Η υπορουτίνα για τον μικροελεγκτή AVR ATmega16 και την αναπτυξιακή πλακέτα EasyAVR6 είναι μια χρονοκαθυστέρηση τόσων msec, όση η δυαδική τιμή που περιέχεται στο ζευγάρι καταχωρητών r25:r24 κατά την κλήση και βασίζεται στην προηγούμενη (wait_usec). Από τα σχόλια φαίνεται ότι η παραπάνω υπορουτίνα wait_msec, όταν εκτελείται ο επαναληπτικός βρόχος, απαιτεί 17 κύκλους ρολογιού ή 2.125μsec και μαζί με τη χρονοκαθυστέρηση της υπορουτίνας wait_usec, που με είσοδο 998 είναι 998.375μsec, συνολικά 1000.5μsec ή 1.0005msec.



Σχήμα 1.2 Κλήσεις υπορουτινών στο πρόγραμμα του μετρητή *modulo* 15.

Παράδειγμα 1.2 Δίνεται ένα παράδειγμα προγράμματος που αναβοσβήνει συνεχώς τα LEDs εξόδου του συστήματος easyAVR6. Το κύριο πρόγραμμα έχει μόνο 3 βασικές εντολές: μια που καλεί την ρουτίνα ON, μια που καλεί την ρουτίνα OFF και μια που ξαναγυρνά στην αρχή. Το Σχήμα 1.3 δείχνει πως χρησιμοποιεί υπορουτίνες για να αναβοσβήνει τα LEDs της θύρας PORTA.



Σχήμα 1.3 Πρόγραμμα που αναβοσβήνει τα LEDs.

Στη συνέχεια παρουσιάζουμε αναλυτικά την εφαρμογή.

Πίνακας 1.2 Πρόγραμμα σε assembly που αναβοσβήνει συνεχώς τα LEDs

```
reset:   ldi r24 , low(RAMEND)          ; αρχικοποίηση stack pointer
         out SPL , r24
         ldi r24 , high(RAMEND)
         out SPH , r24

         ser r26                        ; αρχικοποίηση της PORTA
         out DDRA ,  r26            ; για έξοδο

flash:       rcall  on                  ; Άναψε τα LEDs
```

```
        nop                     ; Να αντικατασταθούν κατάλληλα οι 2 εντολές nop
        nop                     ; για προσθήκη καθυστέρησης 200 ms

        rcall off               ; Σβήσε τα LEDs
        nop                     ; Να αντικατασταθούν κατάλληλα οι 2 εντολές nop
        nop                     ; για προσθήκη καθυστέρησης 200 ms

        rjmp flash              ; Επανέλαβε

; Υπορουτίνα για να ανάβουν τα LEDs
on:     ser r26                         ; θέσε τη θύρα εξόδου των LED
        out PORTA , r26
        ret                     ; Γύρισε στο κύριο πρόγραμμα

; Υπορουτίνα για να σβήνουν τα LEDs
off:    clr r26                         ; μηδένισε τη θύρα εξόδου των LED
        out PORTA , r26
        ret                     ; Γύρισε στο κύριο πρόγραμμα
```

Παράδειγμα 1.3 Δίνεται ένα παράδειγμα προγράμματος σε C που υλοποιεί βασικές λειτουργίες I/O στο σύστημα easyAVR6. Το πρόγραμμα υλοποιεί πρόσθεση 4 δεκαεξαδικών ψηφίων. Χρησιμοποιεί τις θύρες PORTA και **PORTD** ως θύρες εισόδου, απομονώνοντας ως δεκαεξαδικά ψηφία πρόσθεσης τα bit **PA7-PA4**, **PA3-PA0** της θύρας **PORTA** και τα bit **PD7-PD4**, **PD3-PD0** της θύρας **PORTD** και απεικονίζει το αποτέλεσμα της πρόσθεσης στα LEDs της θύρας **PORTB**.

Πίνακας 1.3 Πρόγραμμα σε C που προσθέτει 2 δεκαεξαδικά ψηφία

```
#include <avr/io.h>

char x, y, z, k;

int main(void)
{

        DDRB=0xFF;              // Αρχικοποίηση PORTB ως output
        DDRD=0x00;                      // Αρχικοποίηση PORTD ως input
        DDRA=0x00;                      // Αρχικοποίηση PORTA ως input

        while(1)
        {
                x = PIND & 0x0F;  // Απομόνωση PD3-PD0

                y = PIND & 0xF0;  // Απομόνωση PD7-PD4
                y = y >> 4;             // Μεταφορά ψηφίου στην ορθή του αξία

                z = PINA & 0x0F;  // Απομόνωση PA3-PA0


                k = PINA & 0xF0;  // Απομόνωση PA7-PA4
                k = k >> 4;             // Μεταφορά ψηφίου στην ορθή του αξία

                PORTB = (x+y+z+k);      // Υπολογισμός αθροίσματος και έξοδος στην
PORTB

        }

        return 0;
}
```

Παράδειγμα 1.4 Δίνεται ένα παράδειγμα προγράμματος C για το σύστημα easyAVR6 το οποίο αρχικά έχει αναμμένο το led0 που είναι συνδεδεμένο στο bit0 της θύρας εξόδου PORTB και στη συνέχεια σε κάθε επαναφορά του διακόπτη (Push-button) SW0 όπου είναι συνδεδεμένα στα αντίστοιχα bit της θύρας εισόδου PORTC να ολισθαίνει κυκλικά το αναμμένο led της θύρας εξόδου PORTD κατά μια θέση αριστερά, λαμβάνοντας υπόψη και την πιθανότητα υπερχείλισης.

Πίνακας 1.4 Πρόγραμμα σε C που περιστρέφει ένα αναμμένο led

```
#include <avr/io.h>

char x;

int main(void)
{
	DDRB=0xFF;		// Αρχικοποίηση PORTB ως output
	DDRA=0x00;		// Αρχικοποίηση PORTA ως input

	x = 1;		// Αρχικοποίηση μεταβλητής για αρχικά αναμμένο led

	while(1)
	{
		if ((PINA & 0x01) == 1){		// Έλεγχος πατήματος push-button SW0

			while ((PINA & 0x01) == 1);	// Έλεγχος επαναφοράς push-button SW0

			if (x==128)		// Έλεγχος υπεχείλισης
				x = 1;
			else
				x = x<<1;		// Ολίσθηση αριστερά

			}

			PORTB = x;			// Έξοδος σε PORTB
	}
	return 0;
}
```