



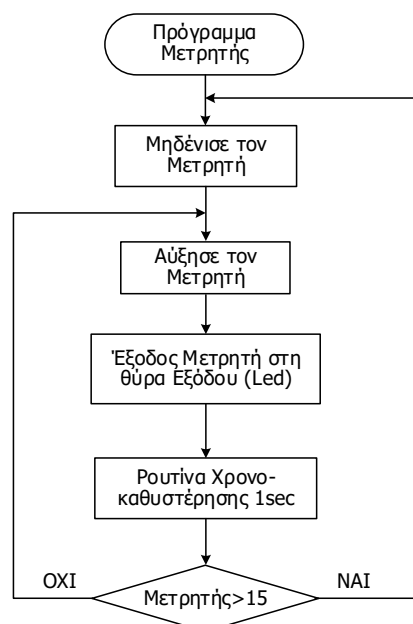
## ΠΑΡΑΔΕΙΓΜΑΤΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ AVR

### Επανάληψη από το μάθημα "Συστήματα Μικροϋπολογιστών"

#### 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<sub>10</sub>, 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.

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

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

#### Ρουτίνα: wait\_usec

Προκαλεί καθυστέρηση τόσων msec, 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 1msec. Άρα, όσες φορές εκτελεστεί ο βρόχος, τόσα msec 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  $n-1+0.875+0.500=n+0.375$  msec).

#### Ρουτίνα: wait\_msec

Προκαλεί καθυστέρηση τόσων msec, όση η τιμή του καταχωρητή r25:r24

**Είσοδος:** Ο χρόνος (1 - 65535 ms) μέσω του καταχωρητή r25:r24

**Καταχωρητές:** r25:r24

**Καλούμενες υπορουτίνες:** wait\_usec



<b>rcall off</b>	; Σβήσε τα LEDs
<b>nop</b>	; Να αντικατασταθούν κατάλληλα οι 2 εντολές nop
<b>nop</b>	; για προσθήκη καθυστέρησης 200 ms
<b>rjmp flash</b>	; Επανάλαβε

#### ; Υπορουτίνα για να ανάβουν τα LEDs

on: <b>ser r26</b>	; θέσε τη θύρα εξόδου των LED
<b>out PORTA, r26</b>	
<b>ret</b>	; Γύρισε στο κύριο πρόγραμμα

#### ; Υπορουτίνα για να σβήνουν τα LEDs

off: <b>clr r26</b>	; μηδένισε τη θύρα εξόδου των LED
<b>out PORTA, r26</b>	
<b>ret</b>	; 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 PORTA να ολισθαίνει κυκλικά το αναμμένο led της θύρας εξόδου PORTB κατά μια θέση αριστερά, λαμβάνοντας υπόψη και την πιθανότητα υπερχειλίσης.

**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 υπερχειλίσης
                x = 1;
            else
                x = x<<1;                  // Ολίσθηση αριστερά

        }

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

---