Εργαστήριο 1

9/10/2020

ΕΞΟΙΚΕΙΩΣΗ ΜΕ ΤΗΝ ΟΙΚΟΓΕΝΕΙΑ ΜΙΚΡΟΕΛΕΓΚΤΩΝ ATMEL AVR

Ντουνέτας Δημήτρης
ΑΜ: 2016030141

Εισαγωγή

Σκοπός του εργαστηρίου ήταν η εξοικείωση με το περιβάλλον ανάπτυξης για μικροελεγκτή AVR δημιουργώντας ένα χρονιστή με βρόγχο κατάλληλου αριθμού επαναλήψεων και με τη χρήση Timer/Counter με Interrupt.

Με χρήση βρόγχου

Για να επιτευχθεί η ζητούμενη καθυστέρηση του 1 ms δημιουργούμε ένα πρόγραμμα σε γλώσσα AVR Assembly που χρησιμοποιεί ένα βρόγχο ώστε όταν βγει από αυτόν να έχει χρησιμοποιήσει 10.000 κύκλους ρολογιού. Ο υπολογισμός αυτός βασίζεται στο γεγονός ότι το ρολόι του Μικροελεγκτή είναι της τάξης του 1Mhz.

Επειδή οι Register του Μικροελεγκτη μας είναι των 8 bit σημαίνει ότι η μέγιστη τιμή που μπορούμε να καταχωρήσουμε είναι ο αριθμός 255. Έτσι έχουμε 2 επιλογές. Μπορούμε είτε να χρησιμοποιήσουμε 2 Loops εμφωλευμένα ώστε να επιτύχουμε μέτρημα μεγαλύτερου του 255, είτε να «κολλήσουμε» 2 καταχωρητές κατάλληλα ώστε να δημιουργήσουμε ένα καταχωρητή που μπορεί να μετρήσει μέχρι την τιμή 65535 που είναι υπεραρκετή για τη δική μας καθυστέρηση.

Στο πρόγραμμα μου χρησιμοποίησα το δεύτερο τρόπο και έτσι αρχικοποίησα 2 καταχωρητές με συνολική τιμή 16 bit τον αριθμό 2499.

```
.def    LoopRl = r24        ; inner loop register low
.def    LoopRh = r25        ; inner loop register high

.equ    iVal    = 2499  ; inner loop value
```

Η εισαγωγή της τιμής στον LoopR καταχωρητή γίνεται ως εξής με τη χρήση των συναρτήσεων HIGH() και LOW() του Assembler:

```
ldi LoopRl,LOW(iVal)    ; intialize Low 8 bits
ldi LoopRh,HIGH(iVal)   ; intialize High 8 bits
```

Σετάρουμε την θύρα B και συγκεκριμένα το PINB0 ως έξοδο βάζοντας την τιμή 1 στον καταχωρητή DDRB.

```
ldi r17,1
out DDRB,r17            ; set PINB0 to output
```

Στη συνέχεια βεβαιωνόμαστε ότι ο LedRegister είναι άδειος και κάνοντας χρήση την εντολή out τον καταχωρούμε ως έξοδο στη θύρα B.

```
clr ledR            ; clear led register
out PORTB,ledR      ; write led register to PORTB
```

Μετά, μπαίνουμε στον βρόγχο που έχουμε φτιάξει στον οποίο αφαιρούμε από τον 16 – bit καταχωρητή 1 τιμή σε κάθε βρόγχο και ελέγχουμε αν είναι ίσος με το 0.

```
;Loop takes 4 cycles. In order to delay for 1ms we have to consume 10000 cycles. 10000 cycles / 4 = 2500 loops.
Loop:   sbiw    LoopRl,1        ; decrement inner loop registers  (2 cycles)
brne    Loop                    ; branch to iLoop if iLoop registers != 0   (2 cycles)
```

Τέλος αφού ο καταχωρητής μας γίνει 0 βγαίνουμε από το βρόγχο όπου καταχωρούμε την τιμή 1 στον καταχωρητή LedRegister και βάζουμε τη τιμή του στη θύρα B ώστε να φαίνεται στο PINB0. Ύστερα επιστρέφουμε στην αρχή του προγράμματος μας και ξανά επαναλαμβάνουμε την διαδικασία από την αρχή.

```
ldi ledR,1              ;   led logical1
out PORTB,ledR      ; write led register to PORTB
rjmp    start           ; jump back to start
```

Σημειώνουμε ότι έτσι πως έχει υλοποιηθεί το πρόγραμμά μας το PINB0 γίνεται λογικό 1 για 1 μόνο κύκλο ρολογιού και μένει σβηστό για 1 ms.

ΥΠΟΛΟΓΙΣΜΌΣ ΚΑΘΥΣΤΕΡΊΣΕΩΝ

Για να υπολογίσουμε σωστά τη καθυστέρηση που δημιουργεί το πρόγραμμα παρατηρούμε ότι:

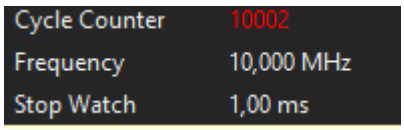
```
    ldi r17,1               ;(1 cycle)
    out DDRB,r17            ;(1 cycle) set PINB0 to output

start:
    clr ledR            ; (1 cycle) clear led register
    out PORTB,ledR      ; (1 cycle) write led register to PORTB
    ldi LoopRl,LOW(iVal)    ; (1 cycle) intialize Low 8 bits
    ldi LoopRh,HIGH(iVal)   ; (1 cycle) intialize High 8 bits

    ;Loop takes 4 cycles. In order to delay for 1ms we have to consume 10000 cycles. 10000 cycles / 4 = 2500 loops.
    Loop:   sbiw    LoopRl,1        ; decrement inner loop registers  (2 cycles)
    brne    Loop                    ; branch to iLoop if iLoop registers != 0   (2 cycles)
```

Όλες οι εντολές μέχρι να μπούμε στο βρόγχο έχουν διάρκεια ενός κύκλου οπότε σύνολο 4 κύκλοι και ο βρόγχος μας διαρκεί συνολικά 4 κύκλους επειδή η αφαίρεση μια τιμής από την λέξη 16-bit διαρκεί 2 κύκλους και ο έλεγχος άλλους 2.

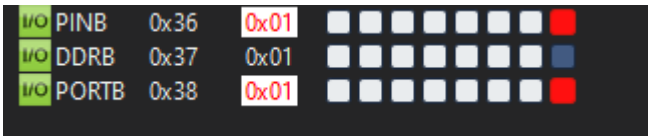
Τέλος οι εντολές που τρέχουμε για να φανεί το λογικό 1 στο Led διαρκούν άλλους 2 κύκλους. Δεν λαμβάνουμε υπο όψην την εντολή jump που μας επιστρέφει στην αρχή.

```
ldi ledR,1              ;   led logical1
out PORTB,ledR      ; write led register to PORTB
rjmp    start           ; jump back to start
```

Έτσι για να έχουμε συνολική καθυστέρηση 1 ms πρέπει να τρέξουμε 10000 συνολικούς κύκλους περίπου. Αυτό το επιτυγχάνουμε τρέχοντας τον βρόγχο μας 2499 φορές + 6 κύκλους από τις υπόλοιπες εντολές που μας κάνει 10002 κύκλους δηλαδή περίπου 1ms.



Και το PINB0 γίνεται 1 για ένα κύκλο.



Σε αυτή τη περίπτωση σε διάρκεια ενός ms τρέχουν 2499 + 6 εντολές = 2505 εντολές.

Με interrupt

Όμοια με το προηγούμενο ερώτημα για να επιτύχουμε 1 ms καθυστέρηση το πρόγραμμα μας πρέπει να καταναλώσει 10.000 κύκλους ρολογιού μέχρι να δείξει στην έξοδο του το λογικό 1.

Για να το επιτύχουμε αυτό αρχικά ξεκινάμε με ένα jump στο Main Label μας.

```
.cseg
.org 0x00

rjmp Main
```

Αυτό το κάνουμε επειδή από κάτω υλοποιούμε την διαδικασία του interrupt Handler τον οποίο θα χρησιμοποιήσουμε όταν εμφανιστεί το Interrupt που θέλουμε να υλοποιήσουμε.

```
.org OC1Aaddr       ;interrupt Handler address
rjmp ResetInterrupts
```

Στο Main Label προετοιμάζουμε τον Timer/Counter ώστε να υλοποιήσει την κατάλληλη διαδικασία για να μας δώσει το Interrupt που θέλουμε να υλοποιήσουμε.
Χρησιμοποιούμε τον Timer/Counter 1 που είναι 16- bit επειδή θα μας δώσει την μεγαλύτερη ακρίβεια στη μέτρηση με ικανό αριθμό επαναλήψεων και μπορούμε να τον χρησιμοποιήσουμε αφού δεν χρησιμοποιείται αλλού.

Αρχικοποιούμε κατάλληλα τον καταχωρητή TCCR1B με λογικό 1 στα Bits CS10 που ενεργοποιεί τον χρονιστή με Prescaler 1 δηλαδή απλό ρολόι και WGM12 που ορίζει τον χρονιστή σε CTC mode. Ύστερα αρχικοποιούμε τον καταχωρητή TIMSK με λογικό 1 στο bit OCIE1A που ενεργοποιεί το Compare Interrupt. Ύστερα βάζουμε την τιμή που θέλουμε στον καταχωρητή OCR1A με την οποία θα συγκρίνει ο TIMER/COUNTER1 και όταν θα είναι ίσοι θα σηκωθούν τα κατάλληλα flags και θα γίνει το interrupt.
Αυτό το κάνουμε όπως ορίστηκε και στη προηγούμενη άσκηση αφού και αυτός ο καταχωρητής είναι 16-bit.
Στη συνέχει με την εντολή sei ενεργοποιούμε το Global Interrupt του Μικροελεγκτή ώστε να περιμένει Interrupt και όταν συμβεί να το εξυπηρετήσει.
Τέλος μπαίνουμε σε έναν ατέρμονα βρόγχο που σπαταλάει 2 κύκλους ρολογιού τη φορά.

```
Main:

    ;Prescaler 1 Initialisation
    ldi r16, (1<<CS10)|(1<<WGM12)  ; Load immediate bits CS10 and WGM12 into TCCR1B Register (1 cycle)
    out TCCR1B, r16      ;(1 cycle)

    ;enable Compare Interrupt
    ldi r16, (1<<OCIE1A) ;Load immediate bit OCIE1A into TIMSK Register (1 cycle)
    out TIMSK, r16       ;(1 cycle)

    ldi LoopRh,HIGH(LoopValue)       ; higher 8 bits (1 cycle)
    ldi LoopRl,LOW(LoopValue)       ; lower 8 bits (1 cycle)
    out OCR1AH, LoopRh    ; Load 8 higher bytes into Compare Register 1(1 cycle)
    out OCR1AL, LoopRl    ;  Load 8 lower bytes into Compare Register 1(1 cycle)

    sei ;enable interupts

    ;infite loop
    label:
    rjmp label
```

Όλη η διαδικασία μέχρι να φτάσουμε στο Βρόγχο καταναλώνει 11 κύκλους ρολογιού. Όμως ο χρονιστής μετράει από τη στιγμή που αρχικοποιήσαμε τον καταχωρητή TCCR1B οπότε μετράει 7 κύκλους ρολογιού μέχρι να μπούμε στο βρόγχο.

Αφού μπεις το βρόγχο αρκεί να μετρήσει 9990 φορές + 3 για να εξυπηρετήσει το interrupt ώστε να πάει στο σημείο εξυπηρέτησης του Interrupt Handler για το compare όπου αρχικοποιείται και ανάβει το Led στο PINB0.

```
.org OC1Aaddr       ;interrupt Handler address

    ldi r17,1               ;(1 cycle)
    out DDRB,r17            ;(1 cycle) set PINB0 to output
    ldi r16,1               ; led logical1
    out PORTB,r16
    rjmp ResetInterrupts
```

Εδώ ομοίως ανάβει το λαμπάκι για 3 κύκλους ρολογιού μέχρι να γίνει η μετακίνηση στο Label ResetInterrupts όπου η έξοδος γίνεται 0 και όλοι οι καταχωρητές μηδενίζονται.

```
ResetInterrupts:

    ldi r16,0                ; led logical1
    out PORTB,r16

    ldi r16, 0
    out TCCR1B, r16

    ldi r16, 0
    out TIMSK, r16

    reti
```

Τελικά εκτελείται η εντολή reti και επιστέφει το πρόγραμμα στην αρχή.

Ο χρόνος καθυστέρησης μέχρι η έξοδος του Μικροελεγκτη να γίνει 1 είναι 1ms.

Cycle Counter	10003
Frequency	10,000 MHz
Stop Watch	1,00 ms

Χρόνος εκτέλεσης όλου του προγράμματος:

Cycle Counter	10011
Frequency	10,000 MHz
Stop Watch	1,00 ms

Συνολικά εκτελούνται (10 αρχικά) + (4990 στον βρόγχο) + (4 εντολές στον Interrupt Handler) + (7 εντολές στο label ResetInterrupt) = 5011 εντολές στη διάρκεια του ενός ms.

Παρατήρηση

Με τη χρήση του TIMER/COUNTER παρατηρήσαμε ότι μπορέσαμε να τρέξουμε περίπου τις διπλάσιες εντολές στην ίδια διάρκεια χρόνου. Αυτό σημαίνει ότι μπορούμε να κάνουμε καλύτερη χρήση του μικροελεγκτή μας και να έχουμε καλύτερη παραλληλία χωρίς να απασχολούμε άλλους πόρους του.