Ενσωματωμένα Συστήματα Μικροεπεξεργαστών – ΗΡΥ411

Αναφορά Εργαστηρίου 1 – Εξοικείωση με την Οικογένεια Μικροελεγκτών Atmel AVR

Ομάδα: LAB41145877
Κιούλος Ευάγγελος 2016030056

Εισαγωγή:

Σκοπός της πρώτης εργαστηριακής άσκησης ήταν η εξοικείωση με την οικογένεια μικροελεγκτών AVR και με το περιβάλλον ανάπτυξης Atmel Studio 7. Κληθήκαμε να υλοποιήσουμε έναν απλό χρονιστή του 1 msec, σε AVR Assembly, με δύο τρόπους. Πρώτον, χρησιμοποιώντας έναν βρόγχο επαναλήψεων και δεύτερων, χρησιμοποιώντας τους πόρους TIMER/COUNTER του μικροελεγκτή. Επιπλέων, χρειάστηκε και η προσομοίωση των προγραμμάτων με την χρήση του προσομοιωτή που παρέχει το Atmel Studio 7. Για την άσκηση χρησιμοποιήθηκε ο ATmega16 με ρολόι 10 MHz

Υλοποίηση – Προσομοίωση:

- **<u>Με χρήση βρόγχου επαναλήψεων:</u>**

Αρχικά πρέπει να υπολογισθεί ο αριθμός των επαναλήψεων που χρειαζόμαστε για να συμπληρωθεί το 1 msec έχοντας ρολόι 10 MHz. Έχουμε:

$$clk = 10\ MHz \Rightarrow 1\ cycle = \frac{1}{10\ Mhz} = 0.1\ \mu sec.$$

Επομένως για να μετρήσουμε 1 msec θα χρειαστούμε:

$$\frac{1\ msec}{0.1\ \mu sec} = 10000\ cycles.$$

Αρά για να μετρήσουμε 1 msec θα χρειαστούμε ένα πρόγραμμα που να εκτελεί 10000 εντολές, θεωρώντας ότι οι βασικές εντολές κοστίζουν από 1 κύκλο ενώ τα branch από 2. Παρατηρούμε ότι οι βασικοί καταχωρητές γενικής χρήσης είναι των 8 bit, επομένως η μέγιστη τιμή που μπορούν να πάρουν είναι το 255, επομένως συμπεραίνουμε ότι θα χρειαστούμε εμφολευμένους βρόγχους για την εκτέλεση των 10000 εντολών.

Αρχικά, ορίσαμε το πρώτο pin του PORTB ως έξοδο, μετατρέποντας το πρώτο bit του καταχωρητή DDRB σε ‘1’.(Ref. Table 20 pg. 52, ATmega16 manual)

```
ldi r16,(1<< PINB0)          ; insert 0x00000001 in register r16
out DDRB, r16                ; insert value of register r16 in DDRB,
                             ; makes pin 0 of port b an output
```

Όπως βλέπουμε και στην παρακάτω εικόνα, δημιουργήθηκε ένας κόμβος των που κοστίζει 4 κύκλους σε κάθε επανάληψη και επαναλαμβάνεται 250 φορές. Επομένως, όταν ο βρόγχος αυτός τελειώσει θα έχουμε εκτελέσει 1000 κύκλους.

```
loop2:
    nop                         ; no operation (1 cycle)
    dec r19                     ; decriment register r19 (1 cycle)
    brne loop2                  ; branch to label loop2 when r19 is not 0 (2 cycles)
    ; This loop(loop2) costs 4 cycles on each iterration. It's repeted 250 times
    ; so when the loop ends we have 1000 cycles
```

Στην συνέχεια, όπως φαίνεται στο επόμενο απόσπασμα κώδικα, δημιουργήθηκε ένας εξωτερικός βρόγχος που επαναλαμβάνει τον παραπάνω βρόγχο 10 φορές. Επομένως, όταν τελειώσουν όλες οι επαναλήψεις θα έχουμε εκτελέσει περίπου 10000 κύκλους.

```
    ldi r18,10                  ; insert value 10 in register r18
loop1:
    ldi r19, 250                ; insert value 250 in register r19
loop2:
    nop                         ; no operation (1 cycle)
    dec r19                     ; decriment register r19 (1 cycle)
    brne loop2                  ; branch to label loop2 when r19 is not 0 (2 cycles)
    ; This loop(loop2) costs 4 cycles on each iterration. It's repeted 250 times
    ; so when the loop ends we have 1000 cycles
    dec r18                     ; decriment register r18
    brne loop1                  ; branch to label loop1 when r18 is not 0
    ; loop1 repeats loop2, which costs 1000 cycles, 10 times. When loop1 ends we have 10000 cycles
    out PORTB, r16              ; insert value of register r16 in PORTB, makes pin 0 of port b "1"
```

Έπειτα με την εντολή *“out PORTB, r16”,* εισάγουμε την τιμή του καταχωρητή r16(σε αυτή την περίπτωση εισάγουμε το 0x00000001), στον καταχωρητή PORTB, το οποίο έχει σαν αποτέλεσμα το pin 0 του port b να βγάζει στην έξοδο ‘1’. Τέλος, το πρόγραμμα μένει σε έναν ατέρμονο βρόγχο.

<u>Αποτελέσματα Προσομοίωσης:</u>

Αρχικά, παρατηρούμε στην ένδειξη I\O view ότι μετά από την εκτέλεση της εντολής *“out DDRB, r16”*, το πρώτο bit του Data Direction Register, έχει την τιμή ‘1’ και επομένως το πρώτο pin του port B λειτουργεί ως έξοδος, το οποίο αρχικά δεν έχει καμία τιμή στην έξοδο του.

Name	Address	Value	Bits
I/O PINB	0x36	0x00	▢▢▢▢▢▢▢▢
I/O DDRB	0x37	0x01	▢▢▢▢▢▢▢■
I/O PORTB	0x38	0x00	▢▢▢▢▢▢▢▢

Στην συνέχεια, μετά την ολοκλήρωση των δύο βρόγχων επανάληψης, παρατηρούμε ότι το πρόγραμμα έχει εκτελέσει λίγο παραπάνω από 10000 κύκλους οι οποίοι αντιστοιχούν σε 1,003 msec, σύμφωνα με την ένδειξη “Stop Watch”.

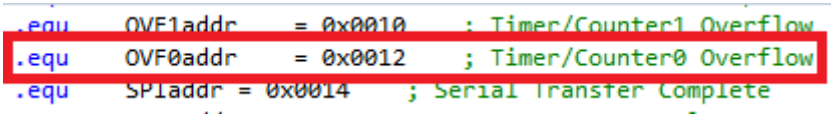
Status Register	I T H S V N Z C
Cycle Counter	10032
Frequency	10,000 MHz
Stop Watch	1.003,20 µs

Έπειτα, παρατηρούμε ότι μετά την εκτέλεση της εντολής *“out PORTB, r16”,* το πρώτο bit του port B έχει γίνει ‘1’, επομένως, το pin 0 του port B έχει ‘1’ στην έξοδο του.

Name	Address	Value	Bits
I/O PINB	0x36	0x01	▢▢▢▢▢▢▢■
I/O DDRB	0x37	0x01	▢▢▢▢▢▢▢■
I/O PORTB	0x38	0x01	▢▢▢▢▢▢▢■

- **<u>Με χρήση TIMER/COUNTER και interrupt:</u>**

Για αυτή την υλοποίηση χρησιμοποιήθηκε ο TIMER/COUNTER 0. Αρχικά, στο πρόγραμμα μας, με την εντολή *".org OVF0addr",* ορίζουμε την διεύθυνση του interrupt vector για τον interrupt που προκαλείται από την υπερχείλιση του timer 0. Το label "OVF0addr" ορίζεται στο αρχείο "Dependencies" που υπάρχει στο project.



Το πρόγραμμα μας, θα πρέπει να πηγαίνει σε αυτή την διεύθυνση μόνο όταν προκαλείται το interrupt για την υπερχείλιση του timer0. Επομένως, πριν την εντολή *".org OVF0addr",* κάνουμε *"rjmp start"*, το οποίο μας οδηγεί αρχικά στην αρχικοποίηση των καταχωρητών που θα χρειαστούμε και στην συνέχεια στο κυρίως πρόγραμμα. Στο παρακάτω απόσπασμα κώδικα βλέπουμε τις αρχικές σειρές του κώδικα, όπου υπάρχει το "jump" στο κυρίως πρόγραμμα αλλά και η δήλωση του interrupt vector.

```
.org 0x00           ; We specify the address 0x00 to put our code in
                    ; the beginning of the flash memory.

rjmp start          ; Jump to label "start" which starts the initialization
                    ; the registers we need and the main program.

; We get here only when the interrupt occurs.
.org OVF0addr       ; Specify the address of Timer/Counter 0 Overflow interrupt.
rjmp ISR            ; Jump to Interrupt service routine.
```

Στην συνέχεια, ορίζουμε τον stack pointer στην αρχή της μνήμης RAM. Παρατηρούμε ότι ο stack pointer αποτελείται από 2 καταχωρητές, τους SPH, SPL. Αυτό γίνεται γιατί ο stack pointer είναι 16 bit ενώ ο καταχωρητής r16 είναι 8 bit. Έπειτα, ενεργοποιούμε τα interrupts μέσω του *"sei"*.

```
start:
    ldi r17, high(RAMEND)          ; Set stack pointer at the top of RAM
    out SPH, r17                   ; Upper byte of stack pointer
    ldi r17, low(RAMEND)
    out SPL, r17                   ; Lower byte of stack pointer
    sei                            ; Global Interrupt Enable
```

Έπειτα, ορίζουμε το pin 0 του port B ως έξοδο και αρχικοποιούμε τους καταχωρητές που χρειαζόμαστε για τον timer0 και τον prescaler.

Για τον prescaler, θα χρησιμοποιήσουμε τον τύπο:

$$f_{OCnA} = \frac{f_{clk}}{2N(1 + OCRnA)}.$$

(Ref. pg. 102, ATmega16 manual.)

Έχουμε:

$$f_{OCnA} = \frac{f_{clk}}{2N(1 + OCRnA)} \Rightarrow \frac{f_{clk}}{10000} = \frac{f_{clk}}{2N(1 + OCRnA)} \Rightarrow$$

$$2N(1 + OCRnA) = 10000 \xrightarrow{\max(1+OCRnA)=256} 2\,N\,256 = 10000.$$

Για τον prescaler θα πρέπει να χρησιμοποιήσουμε το μικρότερο δυνατό N. Παρατηρούμε ότι Για N=1 => 512<<10000 και για N=8 => 4096<<10000. Επομένως ο prescaler θα είναι $f_{clk}/64$ καθώς: N=64 => 32768>10000.

Επομένως, με N=64 έχουμε: (1+OCRnA) = 10000/(2*64) = 78.125 ≈ 78.

Στην συνέχεια, θα πρέπει να αρχικοποιήσουμε τους καταχωρητές που θα χρειαστούμε για τον timer και τον prescaler. Αρχικά, θα πρέπει να θέσουμε το bit TOIE0 του καταχωρητή TIMKSK στην τιμή '1' καθώς ενεργοποιεί το Timer/Counter 0 Overflow interrupt(Ref. pg. 86, ATmega16 manual).

```
ldi r16,(1<<TOIE0)              ; insert 0x00000001 in register r16.
out TIMSK, r16                  ; Makes Value of bit TOIE0 '1' which enables Timer/Counter 0
                                ; Overflow interrupt
```

Ο timer που χρησιμοποιούμε θα αυξάνεται κάθε 64 κύκλους ρολογιού. Θέλουμε ο TCNT0 να υπερχειλίζει όταν έχει αυξηθεί η τιμή του 78 φορές. Επομένως, στον καταχωρητή TCNT0 θα εισάγουμε την τιμή 256-78 = 156. Έτσι, όταν η τιμή του TCNT0 αυξηθεί άλλες 78 φορές, θα πάρει την τιμή 256 η οποία θα προκαλέσει το Timer/Counter 0 Overflow interrupt.

```
ldi r16,0x64                    ; Initialize TCNT0 to 256-2*78=100. This is because 256-78=156
out TCNT0, r16                  ; overflows TCNT0 in 5000 cycles and we need 10000 cycles.
```

ΠΑΡΑΤΉΡΗΣΗ: Χρησιμοποιούμε στην πραγματικότητα την τιμή 2*78, επομένως 256-2*78=100, καθώς με την τιμή 78 ο TCNT0 υπερχειλίζει στους 5000 κύκλους ενώ θέλουμε να υπερχειλίζει στους 10000 κύκλους.

Για να ορίσουμε τον prescaler $f_{clk}/64$ θα πρέπει να εισάγουμε την τιμή '1' στα bit CS00 και CS01 του καταχωρητή TCCR0, όπως φαίνεται παρακάτω. Επίσης, θέλουμε ο timer0 να λειτουργεί σε normal mode(δηλαδή η σύγκριση με τον καταχωρητή OCR0 πρέπει να είναι απενεργοποιημένη), οπότε, όλα τα υπόλοιπα bit του TCCR0 θα μείνουν με την τιμή '0'. (Ref. pg. 83-85 Table 38-42, ATmega16 manual.)

```
ldi r16,(1<<CS01)|(1<<CS00)     ; insert 0x00000011 in register TCCR0
out TCCR0, r16                  ; Sets prescaler fclk/64
```

Τέλος το πρόγραμμα, μένει σε έναν ατέρμονο βρόγχο μέχρι να προκληθεί το interrupt.

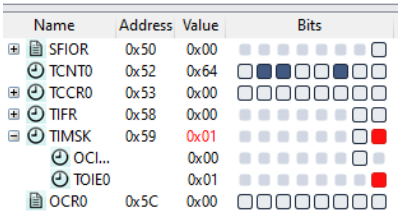
```
main:
    nop                         ; loops endlessly until timer0 ovf interrupt occurs
    rjmp main
```

Αποτελέσματα Προσομοίωσης:

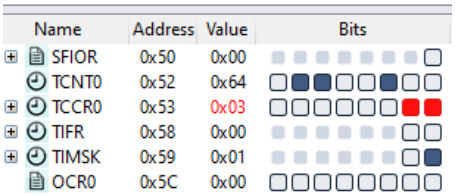
Το πρόγραμμα ξεκινά να εκτελεί από την διεύθυνση του label "start", αρχικοποιεί τον stack pointer και ενεργοποιεί τα interrupts με την εντολή "sei". Στην συνέχεια αφού το pin0 του port B οριστεί ως έξοδος, αρχικοποιείται η τιμή του TCNT0, όπως φαίνεται παρακάτω.

Name	Address	Value	Bits
SFIOR	0x50	0x00	
TCNT0	0x52	0x64	
TCCR0	0x53	0x00	
TIFR	0x58	0x00	
TIMSK	0x59	0x00	
OCR0	0x5C	0x00	

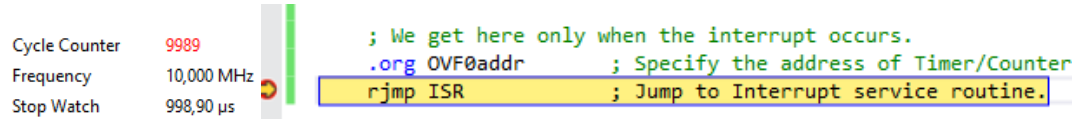
Ενεργοποιείται το πρώτο bit(TOIE0) του TIMSK:



Στην συνέχεια παρατηρούμε ότι αρχικοποιείται και η τιμή του prescaler στον καταχωρητή TCCR0:

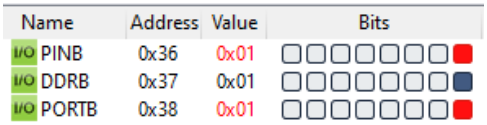


Έπειτα το πρόγραμμα μεταβαίνει σε έναν ατέρμονο βρόγχο μέχρι να προκληθεί το interrupt. Παρατηρούμε ότι όταν εκτελεσθούν περίπου 10000 κύκλοι το πρόγραμμα μεταβαίνει στην διεύθυνση του interrupt handler.



Και ο καταχωρητής TCNT0 κάνει reset στην τιμή “0x00”

Στην συνέχεια, μεταβαίνουμε στη ρουτίνα εξυπηρέτησης του interrupt handler όπου θέτουμε στην έξοδο του pin0 του port B την τιμή ‘1’ και θέτουμε ξανά την τιμή 256-2*78 στον TCNT0.



Τέλος, με *“reti”,* το πρόγραμμα επιστρέφει στο ατέρμονο loop που βρισκόταν πριν προκληθεί το interrupt και μένει εκεί.

Συμπεράσματα:

Παρατηρούμε ότι στην υλοποίηση με το βρόγχο επανάληψης χρειαζόμαστε 10032 κύκλους για να μετρήσουμε 1msec ενώ στην υλοποίηση με τον Timer/Counter0 χρειαζόμαστε 9989 κύκλους. Επομένως, η υλοποίηση με την χρήση Timer/Counter είναι πιο κοντά στην ζητούμενη τιμή, δηλαδή το 1 msec. Επιπλέων, η χρήση του timer/counter είναι πιο αποδοτική καθώς χρησιμοποιούμε τους υπάρχοντες πόρους του μικροελεγκτή για την δημιουργία χρονιστή χωρίς να σπαταλάμε τους υπόλοιπους πόρους.