[Έτος]

# Αναφορά 4ης Εργαστηριακής Άσκησης Εργαστηρίου Μικροϋπολογιστών

ΠΑΠΑΔΟΠΟΥΛΟΣ ΣΠΥΡΙΔΩΝ (ΑΜ): 03120033 ΕΜΜΑΝΟΥΗΛ ΞΕΝΟΣ (ΑΜ):03120850

## *Ζήτημα πρώτο*

Στο πρώτο ζήτημα μας ζητήθηκε να λαμβάνουμε μία μέτρηση από τον ADC, να την μετασχηματίζουμε με βάση τον ακόλουθο τύπο $V_{¿}=\frac{ADC}{1024}*V_{ref}$ και κρατώντας ακρίβεια δύο δεκαδικών ψηφίων να προβάλουμε το αποτέλεσμα στο LCD display της πλακέτας. Αυτό ζητήθηκε να υλοποιηθεί σε assembly , ενώ οι ρουτίνες οι οποίες χρησιμοποιούνται για την επικοινωνία του μικροεπεξεργαστή με το display δόθηκαν στην εκφώνηση. Αρχικά θα κάνουμε μία σύντομη αναφορά στις ρουτίνες που μας δόθηκαν και στην συνέχεια θα εξηγήσουμε το κύριο πρόγραμμα.

Η βασική ρουτίνα που μας δόθηκε είναι η write_2_nibbles, η οποία παίρνει ως όρισμα το περιεχόμενο του καταχωρητή r24 και με βάση αυτό δίνει στο PORTD σε δύο διαδοχικά βήματα το περιεχόμενο του. Το LCD display που χρησιμοποιείται στην άσκηση έχει 4 bit διάδρομο και συνεπώς πρέπει τα περιεχόμενα του καταχωρητή r24 να μεταφερθούν σε αυτό σε δύο διαφορετικές αναθέσεις των τεσσάρων LSB του PORTD ( για αυτό μέσα στην συνάρτηση αυτή γίνονται swap τα 4 LSB με τα 4 MSB του r24). Στην συνέχεια έχουμε τις lcd_data και lcd_command οι οποίες είναι πανομοιότυπες απλώς καλούν την write_2_nibbles με την μόνη διαφορά ότι η lcd_data θέτει το PD2 σε 1, ενώ η lcd_command το θέτει σε 0. Το PD2 αντιστοιχεί σε ένα flag του displaypου ονομάζεται RS και καθορίζει εάν έχουμε σκοπό να μεταφέρουμε μέσω του διαύλου δεδομένα η κάποια εντολή. Προφανώς για RS=1 μεταφέρεται ένα δεδομένο που πρέπει να απεικονιστούν στο display, ενώ για RS=0 μεταφέρεται μία εντολή που πρέπει να εκτελέσει το display. Επιπλέον, έχουμε την lcd_clear_display η οποία στέλνει στο display χρησιμοποιώντας την lcd_command μία εντολή στο display η οποία το κάνει να καθαρίσει την οθόνη του. Τέλος, έχουμε την lcd_init η οποία χρησιμοποιεί όλες τις υπόλοιπες συναρτήσεις για να αρχικοποιήσει το display και να το φέρει σε μία κατάσταση που θα είναι έτοιμο για χρήση.

Εφόσον εξηγήσαμε τις συναρτήσεις που μας δίνονται, θα παρουσιάσουμε την δική μας υλοποίηση για το ζητούμενο και μετά θα την αναλύσουμε:

```
.include "m328PBdef.inc" ;ATmega328P microcontroller definition

    .def temp=r16
    .def counter=r17
    .def low_reg=r26
    .def high_reg=r27
    .def extra_high_reg=r19
    .def extra_temp=r20

    .org 0x0
    rjmp init_progr

    .org 0x2A
```

```
    rjmp my_handler


init_progr:
    ldi temp, high(RAMEND)
    out SPH,temp
    ldi temp, low(RAMEND)
    out SPL,temp
    ser r24
    out DDRD, r24 ; set PORTD as output
    out DDRB,r24
    rcall lcd_init
    ldi r24, low(100)
    ldi r25, high(100) ; delay 100 mS
    rcall wait_msec

    ; REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0010 => select ADC2(pin PC2)
    ; ADLAR=0 => right adjust the ADC result
    ldi temp, 0b01000010 ;
    sts ADMUX, temp
    ldi temp, 0b10000111    ; ADEN=1 => ADC Enable, ADCS=0 => No Conversion,
    ; ADIE=0 => disable adc interrupt, ADPS[2:0]=111 => fADC=16MHz/128=125KHz
    sts ADCSRA, temp        ; we enable  the ADC but do not allow a conversion to happen
yet
Start_conv:
    ldi r24,low(1000)
    ldi r25,high(1000)
    rcall wait_msec     ; wait 1 sec and then enable the ADC
    rcall lcd_clear_display
    sei
    lds temp, ADCSRA ;
    ori temp, (1<<ADSC) | (1<<ADIE) ; Set ADSC flag of ADCSRA and enable the interrupt
    sts ADCSRA, temp ;
    rjmp Start_conv



my_handler:
    ; here we try to compute and ouput the result
    ldi temp,5  ; 5 is the Vref
    lds extra_temp,ADCL
    mul extra_temp,temp ; we dont include ADCL(we already shifted the result of the
conversion
    mov low_reg,r0  ; keep the result of ADCL and Vref
    mov high_reg,r1
    lds extra_temp,ADCH
    andi extra_temp,0b00000011      ; keep only the last three bbits
    mul extra_temp,temp
```

```
    add high_reg,r0 ; add the result of ADCH*Vref to the result of the previous
multiplication
    ldi temp,100    ; multiply ADC with 100 to make a number with two decimal points an
integer

    mul low_reg,temp ; we first multiply the previous low_register with 100
    mov low_reg,r0
    mov extra_temp,r1
    mul high_reg,temp ; then we multiply the previous high_register with 100
    mov high_reg,r0
    add high_reg,extra_temp ; we have to add the high result of low_reg*100 with
                            ; the low result of high_reg *100
    mov extra_high_reg,r1   ; now we have the result of all the multiplications in three
registers
    ldi counter,2
    mov low_reg,high_reg
    mov high_reg,extra_high_reg
rotate_2_times: ; now we will rotate the registers left 2 times because we devide with 2^10
                ; we rotate the registers just two times because we dont need the
low_register
                ; if we rotate 10 times we would keep the low register but its contents
would
                ; come from the other registers-> It's unnecessary to rotate 10 times
    ror high_reg
    ror low_reg
    dec counter
    brne rotate_until_1
    andi high_reg,1
    ; now we will make the binary number to a decimal
    sbrc high_reg,0
    rjmp count_hundreds
    cpi low_reg,99
    brlo zero_hundreds
    ldi counter,0
count_hundreds: ;  we perform consecutive subtractions of 100 from the current number
                ; in order to count the number of hundreds
    inc counter
    sbiw low_reg,63 ; in sbiw the immediate is at most 63
    sbiw low_reg,37 ; so we make two subtractions 63+37=100
    sbrc high_reg,0 ; check if the curent number is lower than 100
    rjmp count_hundreds
    cpi low_reg,99
    brlo output_counter1
    cpi low_reg,99
    breq output_counter1
    rjmp count_hundreds
output_counter1:
    mov r24,counter
```

```
    ori r24,0b00110000  ; code to output a number
    rcall lcd_data
continue:
    ldi r24,0b00101110 ; output a dot
    rcall lcd_data
    ldi counter,0
    cpi low_reg,10
    brlo zero_decades
    ldi counter,0
count_decades:;  we perform consecutive subtractions of 10 from the current number
                ; in order to count the number of decades
    inc counter
    subi low_reg,10
    cpi low_reg,10
    brge count_decades
output_counter2:
    mov r24,counter
    ori r24,0b00110000
    rcall lcd_data
continue_2:
    mov r24,low_reg
    ori r24,0b00110000
    rcall lcd_data
    reti
zero_hundreds:
    clr r24
    ori r24,0b00110000  ; code to output a number
    rcall lcd_data
    rjmp continue
zero_decades:
    clr r24
    ori r24,0b00110000  ; code to output a number
    rcall lcd_data
    rjmp continue_2









write_2_nibbles:
    push r24 ; save r24(LCD_Data)
    in r25 ,PIND ; read PIND
    andi r25 ,0x0f ;
    andi r24 ,0xf0 ; r24[3:0] Holds previus PORTD[3:0]
    add r24 ,r25 ; r24[7:4] <-- LCD_Data_High_Byte
```

```
    out PORTD ,r24 ;
    sbi PORTD ,3 ; Enable Pulse
    nop
    nop
    cbi PORTD ,3
    pop r24 ; Recover r24(LCD_Data)
    swap r24 ;
    andi r24 ,0xf0 ; r24[3:0] Holds previus PORTD[3:0]
    add r24 ,r25 ; r24[7:4] <-- LCD_Data_Low_Byte
    out PORTD ,r24
    sbi PORTD ,3 ; Enable Pulse
    nop
    nop
    cbi PORTD ,3
    ret


lcd_data:
    sbi PORTD ,2 ; LCD_RS=1(PD2=1), Data
    rcall write_2_nibbles ; send data
    ldi r24 ,250 ;
    ldi r25 ,0 ; Wait 250uSec
    rcall wait_usec
    ret


lcd_command:
    cbi PORTD ,2 ; LCD_RS=0(PD2=0), Instruction
    rcall write_2_nibbles ; send Instruction
    ldi r24 ,250 ;
    ldi r25 ,0 ; Wait 250uSec
    rcall wait_usec
    ret


lcd_clear_display:
    ldi r24 ,0x01 ; clear display command
    rcall lcd_command
    ldi r24 ,low(5) ;
    ldi r25 ,high(5) ; Wait 5 mSec
    rcall wait_msec ;
    ret

lcd_init:
    ldi r24 ,low(200) ;
    ldi r25 ,high(200) ; Wait 200 mSec
    rcall wait_msec ;
    ldi r24 ,0x30 ; command to switch to 8 bit mode
```

```
    out PORTD ,r24 ;
    sbi PORTD ,3 ; Enable Pulse
    nop
    nop
    cbi PORTD ,3
    ldi r24 ,250 ;
    ldi r25 ,0 ; Wait 250uSec
    rcall wait_usec ;
    ldi r24 ,0x30 ; command to switch to 8 bit mode
    out PORTD ,r24 ;
    sbi PORTD ,3 ; Enable Pulse
    nop
    nop
    cbi PORTD ,3
    ldi r24 ,250 ;
    ldi r25 ,0 ; Wait 250uSec
    rcall wait_usec ;
    ldi r24 ,0x30 ; command to switch to 8 bit mode
    out PORTD ,r24 ;
    sbi PORTD ,3 ; Enable Pulse
    nop
    nop
    cbi PORTD ,3
    ldi r24 ,250 ;
    ldi r25 ,0 ; Wait 250uSec
    rcall wait_usec
    ldi r24 ,0x20 ; command to switch to 4 bit mode
    out PORTD ,r24
    sbi PORTD ,3 ; Enable Pulse
    nop
    nop
    cbi PORTD ,3
    ldi r24 ,250 ;
    ldi r25 ,0 ; Wait 250uSec
    rcall wait_usec
    ldi r24 ,0x28 ; 5x8 dots, 2 lines
    rcall lcd_command
    ldi r24 ,0x0c ; dislay on, cursor off
    rcall lcd_command
    rcall lcd_clear_display
    ldi r24 ,0x06 ; Increase address, no display shift
    rcall lcd_command ;
    ret

wait_msec:
    push r24 ; 2 cycles
    push r25 ; 2 cycles
    ldi r24 , low(999) ; 1 cycle
```

```
    ldi r25 , high(999) ; 1 cycle
    rcall wait_usec ; 998.375 usec
    pop r25 ; 2 cycles
    pop r24 ; 2 cycles
    nop ; 1 cycle
    nop ; 1 cycle
    sbiw r24 , 1 ; 2 cycles
    brne wait_msec ; 1 or 2 cycles
    ret ; 4 cycles
wait_usec:
    sbiw r24 ,1 ; 2 cycles (2/16 usec)
    call delay_8cycles ; 4+8=12 cycles
    brne wait_usec ; 1 or 2 cycles
    ret
delay_8cycles:
    nop
    nop
    nop
    ret
```

Παραπάνω φαίνεται η υλοποίηση μας για το ζήτημα 1. Αρχικά, στο πρόγραμμα όταν έρχεται ο έλεγχος στην θέση μνήμης 0x2A κάνουμε jump σε έναν δικό μας handler,ώστε να επεξεργαζόμαστε  όπως θέλουμε τις διακοπές που του ADC. Στο κύριο πρόγραμμα αφού αρχικοποιήσουμε τον stack pointer,θέσουμε τις σωστές θύρες εξόδου, θέσουμε τα απαραίτητα flags και αρχικοποιήσουμε το lcd display μπαίνουμε σε ένα άπειρο loop. Σε αυτό το loop απλώς καθαρίζουμε το lcd_display ενεργοποιούμε τις διακοπές του ADC και εκτελούμε μία αναμονή 1 sec. Αφού ενεργοποιήσουμε τις διακοπές του και το conversion του ADC αυτός μόλις ολοκληρώσει ένα conversion θα πραγματοποιήσει μία διακοπή και θα μεταφέρει τον έλεγχο του προγράμματος στην θέση μνήμης 0x2A ,όπου εμείς έχουμε γράψει την δική μας ρουτίνα εξυπηρέτησης της διακοπής αυτής. Στην ρουτίνα αυτή αρχικά πολλαπλασιάζουμε τον ADC με το 5( που είναι το Vref). Για να γίνει αυτός ο πολλαπλασιασμός αρχικά πολλαπλασιάζουμε το ADCL με το 5. Αυτό θα μας δώσει ένα 16 bit αποτέλεσμα στους καταχωρητές r0 και r1. Αποθηκεύουμε το αποτέλεσμα στον high_reg και στο low_reg. Στην συνέχεια πολλαπλασιάζουμε το ADCH με το 5. Ωστόσο, επειδή στο ADCH μόνο τα δύο λιγότερα σημαντικά bits μας ενδιαφέρουν ξέρουμε ότι το ADCH*5 θα μας δώσει ένα αποτέλεσμα που χωράει σε έναν καταχωρητή ( εν προκειμένω τον r0). Συνεπώς προσθέτουμε στο high_reg το γινόμενο ADCH*5 και έχουμε στους δύο καταχωρητές, high_reg και low_reg , το γινόμενο ADC*5 . Με την ίδια λογική πραγματοποιούμε το γινόμενο αυτών των δύο καταχωρητών με το 100 ( ουσιαστικά συνολικά θα έχουμε κάνει το 500* ADC) . Με 100 πολλαπλασιάζουμε διότι στην εκφώνηση αναφέρεται ότι θέλουμε 2 δεκαδικά ψηφία ακρίβεια. Άρα αν πολλαπλασιάσουμε με 100 θα έχουμε έναν ακέραιο αριθμό χωρίς δεκαδικά ψηφία, γεγονός που μας διευκολύνει σημαντικά στην

assembly. Το γινόμενο με το 100 το κάνουμε όπως κάναμε το γινόμενο με το 5 σε δύο βήματα, μόνο που αυτή την φορά το γινόμενο high_reg*100 θα μας δώσει αποτέλεσμα σε δύο καταχωρητές και συνεπώς τα περισσότερο σημαντικά ψηφία αυτού του γινομένου που βρίσκονται στον r1 θα τα αποθηκεύσουμε στον extra_high_reg. Προφανώς πάλι θα προσθέσουμε τα 8 MSB του γινομένου low_reg*100 με τα 8 LSB του γινομένου high_reg*100, για να λάβουμε το σωστό αποτέλεσμα. Μετά από αυτές τις πράξεις θα έχουμε το αποτέλεσμα μας( πράξη 500*ADC) στους καταχωρητές extra_high_reg,high_reg,low_reg . Στην συνέχεια πρέπει να διαιρέσουμε με το $1024=2^{10}$ γεγονός που ισοδυναμεί με 10 shift του αποτελέσματος μας δεξιά. Βλέπουμε ότι εφόσον θα πραγματοποιήσουμε τουλάχιστον 8 shift δεξιά το περιεχόμενο του καταχωρητή low_reg θα αλλάξει εξ ολοκλήρου και θα πάρει το περιεχόμενου που είχαν οι άλλοι 2. Συνεπώς αντί για 10 shift δεξιά στην τρίλεπτά καταχωρητών μπορούμε να πραγματοποιήσουμε 2 shift δεξιά στην δυάδα καταχωρητών extra_high_reg και high_reg. Για να πραγματοποιήσουμε αυτό το shift χρησιμοποιούμε την εντολή ror πρώτα στον extra_high_reg και στην συνέχεια στον high_reg, καθώς έτσι σε κάθε shift που κάνουμε μέσω του κρατουμένου μεταφέρουμε το LSB του extra_high_reg στο MSB του high_reg. Αφού εκτελέσουμε και αυτά τα δύο shift έχουμε τον αριθμό μας( Vin) και εφόσον ξέρουμε ότι αυτός είναι από το 0 μέχρι το 500 για αυτόν απαιτούνται μόνο 9 bits, συνεπώς κρατάμε τον high_reg και από τον extra_high_reg κρατάμε μόνο το LSB(μηδενίζουμε τα υπόλοιπα).Μεταφέρουμε τα αποτελέσματα αυτά στον high_reg και στον low_reg για μεγαλύτερη απλότητα στην κατανόηση). Τώρα για να δείξουμε το αποτέλεσμα στο LCD display πρέπει να στείλουμε διαδοχικά το πλήθος των εκατοντάδων, των δεκάδων και των μονάδων. Για να βρούμε το πλήθος των εκατοντάδων αφαιρούμε διαδοχικά εκατοντάδες από τον διπλό καταχωρητή high_reg,low_reg(r27 και r26 αντίστοιχα). Αυξάνουμε κατά 1 τον counter των εκατοντάδων αφαιρούμε 100 από τον διπλό καταχωρητή και στην συνέχεια ελέγχουμε αν το LSB του high_reg είναι 1. Αν είναι 1 τότε ο αριθμός είναι μεγαλύτερος από το 100 και συνεχίζουμε τις διαδοχικές αφαιρέσεις, ενώ αν είναι 0 τότε ελέγχουμε αν ο low_reg είναι μεγαλύτερος από 100. Αν ο low reg είναι μεγαλύτερος από 100 συνεχίζουμε τις διαδοχικές αφαιρέσεις, ειδάλλως τυπώνουμε στο lcd display την τιμή του counter και μετά μία τελεία ( που υποδεικνύει την υποδιαστολή). Αφού τυπώσουμε τις εκατοντάδες ξέρουμε πλέον ότι ο αριθμός μας είναι μικρότερος του 100 και συνεπώς μόνο ο low_reg επαρκεί για αυτόν. Μετράμε τώρα τις δεκάδες με την ίδια λογική που μετρήσαμε τις εκατοντάδες, αφαιρώντας διαδοχικά δεκάδες από το low_reg μέχρι αυτός να γίνει μικρότερος του 10. Όταν γίνει μικρότερος του 10 ο counter μας θα έχει το πλήθος των δεκάδων, ενώ στον low reg θα έχουμε το πλήθος των μονάδων. Τυπώνουμε τα δύο αυτά αποτελέσματα διαδοχικά στο lcd display τυπώνουμε στο τέλος και ένα V, ως ένδειξη ότι στο display δείχνουμε Volt και επιστρέφουμε από την ρουτίνα εξυπηρέτησης της διακοπής του ADC στο κύριο πρόγραμμα. Αυτή ήταν η υλοποίησης μας για το ζήτημα 1, συνεχίζουμε με τα υπόλοιπα ζητήματα.

---

## *Ζήτημα 4.2*

---

Σε αυτό το ζήτημα υλοποιήθηκε το πρόγραμμα του προηγούμενου ζητήματος σε C με την διαφορά ότι αυτή την φορά δεν θα γίνεται χρήση της διακοπής ολοκλήρωσης μετατροπής του ADC αλλά θα περιμένει ελέγχοντας το bit ADSC του ADCSRA μέχρι το bit αυτό να γίνει 0, που σηματοδοτεί ότι η μετατροπή ADC ολοκληρώθηκε. Το αποτέλεσμα φαίνεται στην LCD οθόνη με ακρίβεια δύο δεκαδικών ψηφίων. Το πρόγραμμα φαίνεται παρακάτω:

```
#define F_CPU 16000000UL
#include "avr/io.h"
#include <util/delay.h>
#include <stdio.h>

void write_2_nibbles(uint8_t c);
void lcd_clear_display();
void lcd_command(uint8_t com);
void lcd_data(unsigned char data);
void lcd_init();
void lcd_string(char *str);

void display_num(int vin){
    int num[3];
    num[2]=vin%10;
    vin/=10;
    num[1]=vin%10;
    vin/=10;
    num[0]=vin%10;
    lcd_data(num[0] + 0x30);
    lcd_data('.');
    lcd_data(num[1] + 0x30);
    lcd_data(num[2] + 0x30);
    lcd_data('V');
}


int main()
{
    uint32_t temp;
    int vin;


    DDRD |= 0xFF;


    ADMUX |= 0b01000010; //Right adjusted, ADC2
    ADCSRA |= 0b10000111;   //Check bigger sampling rate 3 LSBs in
ADCRA register

    lcd_init();
    while (1)
```

```
    {
        lcd_clear_display();
        ADCSRA|= (1<<ADSC);    //Start ADC
        while((ADCSRA & 0x40)==0x40){}   //Wait until ADC is finished
        temp=ADC;
        vin=(temp*500)>>10;
        display_num(vin);
        _delay_ms(1000);

    }
}

void write_2_nibbles(uint8_t c){
    uint8_t temp= c;
    PORTD = (PIND & 0x0f) + (temp & 0xf0); //LCD Data High Bytes
    PORTD|=0x08;
    asm("nop");
    asm("nop");
    PORTD&=~(0x08);

    c=(c<<4)|(c>>4);
    PORTD = (PIND & 0x0f) + (c & 0xf0); //LCD Data Low Bytes

   PORTD|=0x08;
    asm("nop");
    asm("nop");
    PORTD&=~(0x08);

}

void lcd_clear_display(){
    lcd_command(0x01);
    _delay_ms(5);
}

void lcd_command(uint8_t com){
    PORTD&=~4; //LCD_RS=0 => Instruction
    write_2_nibbles(com);
    _delay_us(250);
}

void lcd_data(unsigned char data){
    PORTD|=4; //LCD_RS=1 => Data
    write_2_nibbles(data);
    _delay_us(250);
}

void lcd_init(){
```

```
    _delay_ms(200);
    int i=0;
    while(i<3){      //command to switch to 8 bit mode
        PORTD=0x30;
        PORTD|=0x08;
        asm("nop");
        asm("nop");
        PORTD&=~0x08;
        _delay_us(250);
        ++i;
    }

    PORTD=0x20;     //command to switch to 4 bit mode
    PORTD|=0x08;
    _delay_us(2);
    PORTD&=~0x08;
    _delay_us(250);

    lcd_command(0x28); //5*8 dots, 2 lines
    lcd_command(0x0c); //display on, cursor off

    lcd_clear_display();

}

void lcd_string(char *str){
    int i;
    for(i=0; str[i]!=0; i++) lcd_data(str[i]);
}
```

Οι συναρτήσεις write_2_nibbles(uint8_t c), void lcd_clear_display(), void lcd_command(uint8_t com), lcd_data(unsigned char data), lcd_init(), lcd_string(char *str) αφορούν την επικοινωνία του προγράμματος με την lcd οθόνη, τον καθαρισμό της και την αρχικοποίησή της.

Στην main του προγράμματος αρχικά θέτουμε στον ADMUX τις σημαίες τους ώστε το αποτέλεσμα του ADC να είναι Right Adjusted και η είσοδός του να είναι το ADC2 και στον ADCSRA τις σημαίες ώστε να ενεργοποιήσουμε το ADC και το prescaler να είναι 128. Επίσης θέτουμε την θύρα PORTD ως έξοδο, γιατί μέσω αυτής γίνεται η επικοινωνία με την LCD Display και αρχικοποιούμε την οθόνη μέσω της συνάρτησης lcd_init(). Στην συνέχεια μπαίνουμε σε ένα while loop που τρέχει συνεχώς. Εντός αυτού καθαρίζουμε αρχικά το display μέσω της συνάρτησης lcd_clear_display() και ξεκινάμε το ADC θέτοντας την σημεία ADSC του ADCSRA 1. Έπειτα περιμένουμε ελέγχοντας συνεχώς το τον ADSC flag για εάν έχει μηδενιστεί, που σημαίνει και ότι τελείωσε το ADC. Μόλις μηδενιστεί το ADSC παίρνουμε την τιμή του αποτελέσματος και «μετατρέπουμε» ξανά σε τάση (γνωρίζοντας ότι η τάση Vref είναι 5V). Μάλιστα επιλέγουμε να πολλαπλασιάσουμε το αποτέλεσμα του ADC με 100 προτού διαιρέσουμε με 1024 (αυτή η διαίρεση εδώ γίνεται με 10 δεξιά shift, γιατί $2^{10}$=1024) έτσι ώστε να «εμφανίσουμε» στο ακέραιο μέρος τα 2 πρώτα ψηφία του δεκαδικού μέρους. Λόγω αυτού του πολλαπλασιασμού επιλέχθηκε η μεταβλητή temp να είναι 32bit

ώστε να μην έχουμε overflow. Έπειτα από αυτή την μετατροπή πηγαίνουμε στην συνάρτηση display_num ώστε να αναπαραστήσουμε το vin και στο τέλος εισάγουμε μία διακοπή 1s όπως ζητείται από την εκφώνηση.

Στην συνάρτηση display_num, που παίρνει ως όρισμα τον τριψήφιο αριθμό του αποτελέσματος που προαναφέρθηκε, μέσω διαδοχικών mod και div στον απομονώνουμε ένα-ένα τα ψηφία και στο τέλος αυτής τα αναπαριστούμε στο lcd display εκμεταλλευόμενοι το γεγονός ότι η 8bit αναπαράσταση των ψηφίων στο lcd display είναι τα 4 MSB 0x30 και τα 4 LSB η αναπαράσταση του αριθμού στο δυαδικό σύστημα.

---

## *Ζήτημα 4.3*

---

Στο ζήτημα αυτό υλοποιήθηκε πρόγραμμα σε γλώσσα C για την επιτήρηση ενός χώρου όπου υπάρχει αυξημένος κίνδυνος ύπαρξης CO, μέσω αισθητήρα που είναι συνδεδεμένος στην αναλογική θύρα A3. Ο αισθητήρας αυτός έχει εύρος μέτρησης από 0-500ppm και η ανάγνωσή του γίνεται κάθε περίπου 100ms. Σε περίπτωση που ο αισθητήρας εντοπίσει ότι συγκέντρωση CO είναι μεγαλύτερη από 70ppm τότε στην LCD οθόνη εμφανίζεται το μήνυμα “GAS DETECTED” και αναβοσβήσουν τα LED PB0-PB5. Σε περίπτωση που η συγκέντρωση του μονοξειδίου είναι κάτω από 70ppm τότε στην LCD οθόνη φαίνεται το μήνυμα “CLEAR” και τα LED της PORTB παραμένουν σβηστά.

Για να υπολογίσουμε την τάση στην έξοδο του αισθητήρα ULPSM-CO 968-001 όταν αυτός μετράει συγκέντρωση CO στον χώρο ίση με 70ppm χρησιμοποιήσαμε τον τύπο:

$$C_X = \frac{1}{M} \cdot \left(V_{gas} - V_{gas0}\right) = ¿ V_{gas} = M \cdot C_X + V_{gas0}$$

όπου Cx είναι η συγκέντρωση του αερίου, $V_{gas0}$ δίνεται από την εκφώνηση της ασκήσεως 0.1V και M είναι ο παράγοντας βαθμονόμησης του αισθητήρα (sensor calibration factor) και υπολογίζεται από τον ακόλουθο τύπο:

$$M\left(V/_{ppm}\right) = Sensitivity\ Code\left(nA/_{ppm}\right) \times TIA\ Gain\left(kV/_{A}\right) \times 10^{-9}\left(A/_{nA}\right) \times 10^{3}\left(V/_{kV}\right)$$

όπου το Sensitivity Code δίνεται από την άσκηση 129nA/ppm και TIA Gain για CO είναι 100kV/A, σύμφωνα με το φύλλο δεδομένων.

Έτσι υπολογίζουμε M(V/ppm) ίσο με 0.0129 V/ppm και έτσι για $C_X$=70ppm έχουμε $V_{gas}$=1.003V. Για να βρούμε την τιμή του ADC όταν έχει ως είσοδο αυτή την τάση χρησιμοποιούμε τον τύπο $ADC = \frac{V_{¿}}{V_{ref}} \cdot 2^n$, όπου $V_{ref}$=5V και n=10, έτσι το αποτέλεσμα του ADC είναι 205.4, άρα 205. Έτσι για να δούμε αν η συγκέντρωση του CO είναι μεγαλύτερη από 70ppm μπορούμε απλά να ελέγχουμε αν το αποτέλεσμα του ADC είναι μεγαλύτερο του 205. Παρακάτω φαίνεται ο κώδικας.

```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

void write_2_nibbles(uint8_t c);
void lcd_clear_display();
void lcd_command(uint8_t com);
void lcd_data(unsigned char data);
void lcd_init();
void lcd_string(char *str);


uint8_t gas=0;

ISR(ADC_vect){
    gas=(ADC>205);
}

int main()
{

    DDRB |= 0xFF;
    DDRD |= 0xFF;


    ADMUX |= 0b01000011; //Right adjusted, ADC3
    ADCSRA |= 0b10001111;   //128 Prescaler and ASIE=1

    lcd_init();
    sei();
    ADCSRA|= (1<<ADSC);    //Start ADC

    while (1)
    {
        if(gas){
            lcd_clear_display();
            char message[]="GAS DETECTED";
            lcd_string(message);

            while(gas>143){
                int i;
                PORTB=0xFF;
                for(i=0; i<5; i++){
                    _delay_ms(100);
                    if(!gas) break;
                    ADCSRA|= (1<<ADSC);
                }
```

```
                if(!gas) break;

                PORTB=0;
                for(i=0; i<5; i++){
                    _delay_ms(100);
                    if(!gas) break;
                    ADCSRA|= (1<<ADSC);
                }
            }

        }else{
            lcd_clear_display();
            PORTB=0;
            char message[]="CLEAR";
            lcd_string(message);
            while(!gas){
                _delay_ms(100);
                ADCSRA|= (1<<ADSC);
            }
        }
    }
}

void write_2_nibbles(uint8_t c){
    uint8_t temp= c;
    PORTD = (PIND & 0x0f) + (temp & 0xf0); //LCD Data High Bytes
    PORTD|=0x08;
    asm("nop");
    asm("nop");
    PORTD&=~(0x08);

    c=(c<<4)|(c>>4);
    PORTD = (PIND & 0x0f) + (c & 0xf0); //LCD Data Low Bytes

   PORTD|=0x08;
    asm("nop");
    asm("nop");
    PORTD&=~(0x08);

}

void lcd_clear_display(){
    lcd_command(0x01);
    _delay_ms(5);
}

void lcd_command(uint8_t com){
    PORTD&=~4; //LCD_RS=0 => Instruction
```

```
    write_2_nibbles(com);
    _delay_us(250);
}

void lcd_data(uint8_t  data){
    PORTD|=4; //LCD_RS=1 => Data
    write_2_nibbles(data);
    _delay_us(250);
}

void lcd_init(){
    _delay_ms(200);
    int i=0;
    while(i<3){      //command to switch to 8 bit mode
        PORTD=0x30;
        PORTD|=0x08;
        asm("nop");
        asm("nop");
        PORTD&=~0x08;
        _delay_us(250);
        ++i;
    }

    PORTD=0x20;     //command to switch to 4 bit mode
    PORTD|=0x08;
    _delay_us(2);
    PORTD&=~0x08;
    _delay_us(250);

    lcd_command(0x28); //5*8 dots, 2 lines
    lcd_command(0x0c); //display on, cursor off

    lcd_clear_display();

}

void lcd_string(char *str){
    int i;
    for(i=0; str[i]!=0; i++) lcd_data(str[i]);
}
```

Οι συναρτήσεις write_2_nibbles(uint8_t c), void lcd_clear_display(), void lcd_command(uint8_t com), lcd_data(unsigned char data), lcd_init(), lcd_string(char *str) αφορούν την επικοινωνία του προγράμματος με την lcd οθόνη, τον καθαρισμό της και την αρχικοποίησή της. Η μεταβλητή gas θα χρησιμοποιηθεί ως Boolean μεταβλητή που θα μας δηλώνει αν η τιμή του ADC ξεπεράσει το 205 (δηλαδή η συγκέντρωση μονοξειδίου γίνει μεγαλύτερη από 70ppm).

Στην διακοπή του ADC το μόνο που επιτελούμε είναι να ελέγχουμε αν η τιμή του ADC αποτελέσματος είναι μεγαλύτερη από 205 και το αποτέλεσμα αυτής της συγκρίσεως αποθηκεύεται στην μεταβλητή gas.

Στην συνάρτηση main αρχικά θέτουμε της PORTD και PORTB ως εξόδους. Η PORTD χρησιμοποιείται για την επικοινωνία με την LCD display ενώ το PORTB θα χρησιμοποιηθεί για τα LED ένδειξης κινδύνου. Στην συνέχεια μέσω του καταχωρητή ADMUX θέτουμε την θύρα ADC3 ως είσοδο του ADC και το αποτέλεσμα Right Adjusted, ενώ μέσω του καταχωρητή ADCSRA επιτρέπουμε το ADC και τις διακοπές του. Έπειτα αρχικοποιούμε την lcd display, επιτρέπουμε όλες τις διακοπές μέσω της συνάρτησεις sei() και ξεκινάμε την πρώτη ADC μετατροπή. Μετά από τα παραπάνω το πρόγραμμα μπαίνει σε ένα infinite while loop. Σε αυτό αρχικά ελέγχουμε αν έχουμε συγκέντρωση αερίου μεγαλύτερη των 70ppm μέσω της μεταβλητής gas. Η μεταβλητή gas στην αρχή έχει τιμή 0 ως placeholder και θα αλλάξει πολύ σύντομα, όταν τελειώσει η πρώτη ADC μετατροπή ώστε να λάβει την κατάλληλη τιμή. Αν το gas είναι 0 τότε καθαρίζουμε την οθόνη και εμφανίζουμε το μήνυμα “GAS DETECTED” και για όσο το gas παραμένει true είμαστε μέσα σε ένα while loop. Εντός αυτού αρχικά ανάβουμε τα LED του PORTB και τα κρατάμε ανοιχτά για περίπου 500ms. Κάθε 100ms περίπου ενεργοποιούμε το ADC ώστε να ελέγχουμε αν συνεχίζουμε να έχουμε έχουμε μεγάλη συγκέντρωση μονοξειδίου. Σε περίπτωση που η συγκέντρωση του αερίου πέσει σε επιτρεπτό επίπεδο βγαίνουμε από το while loop. Εάν περάσουν 500ms περίπου και η συγκέντρωση του αερίου παραμένει υψηλή τότε κλείνουμε τα LED του PORTB και για περίπου 500ms ενεργοποιούμε το ADC κάθε 100ms. Αν κάποια στιγμή η συγκέντρωση του αερίου πέσει θα εντοπιστεί από το ADC και θα βγούμε εκτός του while loop. Αν η συγκέντρωση του αερίου μετά τα 500ms παραμένει αυξημένη τότε επιστρέφουμε στην αρχή του while loop που ελέγχει το gas. Αν η τιμή του gas είναι false τότε καθαρίζουμε την οθόνη, κλείνουμε τα LED του PORTB, εμφανίζουμε στην οθόνη το μήνυμα “CLEAR”. Για όσο το gas παραμένει false ενεργοποιούμε το ADC κάθε περίπου 100ms και ελέγχουμε τη τιμή του.