Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο Μικροϋπολογιστών

7ο εξάμηνο, Ακαδημαϊκή περίοδος 2025-2026

3η Εργαστηριακή Άσκηση

Χρήση Οθόνης 2×16 Χαρακτήρων στον AVR

Εργαστηριακή Αναφορά

Φοιτητές: Παπαδάτος Αναστάσιος -> **ΑΜ:** 03122847

Σέρτζιο Γκούρι -> **ΑΜ:**03122827

Ομάδα: 30

Ημερομηνία: 11/11/2025

Άσκηση 1:

Παρακάτω φαίνεται ο κώδικας **assembly** που φτιάξαμε:

```
;
; 4.1.asm
;
; Created: 5/11/2025 3:38:16 μμ
; Author : anast
;

.include "m328PBdef.inc"          ;ATmega328P microcontroller definition

.equ PD0=0
.equ PD1=1
.equ PD2=2
.equ PD3=3
.equ PD4=4
.equ PD5=5
.equ PD6=6
.equ PD7=7
.equ timer_start = 49910          ;explained below
.def result = r20                 ;to store the division result

.org 0x0
    rjmp start
.org 0x1A
    rjmp timer_interupt
.org 0x02A
    rjmp ADC_ready

start:
       ldi r29,'0'                ;for turning the digits to ASCII characters.

       ser r16
       out DDRD,r16               ;set DDRD as output
       out DDRB,r16               ;set DDRB as output
       clr r16
       out DDRC,r16               ;set DDRC as input
       out PORTB,r16              ;clear output
       out PORTD,r16              ;clear outpout

       out EIMSK,r16              ;ensure every other normal interrupt is disabled

       rcall lcd_init
       rcall lcd_clear_display

       sei                        ;enable interrupts

       ldi r16,LOW(RAMEND)
       out SPL,r16
       ldi r16,HIGH(RAMEND)
       out SPH,r16
       clr r16

;pre-building ADC
       ldi r16,0x43               ;ADC3 as input
       sts ADMUX,r16
```

```
	ldi r16,0x8F		;enable conversion-done interrupt
	sts ADCSRA,r16

;initialise counter-timer
	clr r16
	sts TCCR1B,r16		;ensuring counter is not yet counting and is frozen | WGM13,WGM12 -> 0
	sts TCCR1A,r16		;WGM11,WGM10 -> 0 WE WANT NORMAL FUNCTION...NO PWM
	ldi r16,0x01
	sts TIMSK1,r16		;allowing overflow interrupts
;we choose clk/1024 = 15.625hz and timer has 16 bits so it counts from 0 to 65.535
;to have overflow interrupt every 1 sec we need to start counting from 65.535-1*15.625 = 49910 = timer_start

;load TCNT1 with the correct timer_start
	ldi r16,HIGH(timer_start)
	sts TCNT1H,r16
	ldi r16,LOW(timer_start)
	sts TCNT1L,r16

	ldi r16,0x05
	sts TCCR1B,r16		;start timer

main:
	rjmp main		;wait for overflow interupt


timer_interupt:
	sei
	clr r16
	sts TCCR1B,r16		;stop timer
	ldi r16,HIGH(timer_start)	;reload TCNT1 to timer_start
	sts TCNT1H,r16
	ldi r16,LOW(timer_start)
	sts TCNT1L,r16
	ldi r16,0x05
	sts TCCR1B,r16		;restart timer

	ldi r16,0xCF		;setting ADSC to 1 to start convertion
	sts ADCSRA,r16
	reti			;waiting for the completion interrupt to happen

ADC_ready:
	rcall lcd_clear_display

	sei

	lds r30 , ADCL
	lds r31 , ADCH
	mov r17,r30
	mov r18,r31
	lsl r30
	rol r31
	lsl r30
	rol r31			;*4

	add r30,r17		;+1
	adc r31,r18		;so we have *5
```

```
        ldi result,0
        rcall div_1024              ;devide /1024
        mov r24,result
        add r24,r29                 ;turn result to ASCII character
        rcall lcd_data              ;output to LCD

        ldi r24,'.'
        rcall lcd_data              ;output the dot for decimals

        mov r17,r30
        mov r18,r31
        lsl r30
        rol r31
        lsl r30
        rol r31
        lsl r30
        rol r31                     ; x <<  3 = 8*x

        add r30,r17                 ; +x
        adc r31,r18

        add r30,r17                 ; +x
        adc r31,r18                 ; so we have 8x + x + x = 10x

        clr result                  ;divide with 1024 to find the first decimal
        rcall div_1024
        mov r24,result
        add r24,r29                 ;turn result to ASCII character
        rcall lcd_data              ;output to LCD

        mov r17,r30
        mov r18,r31
        lsl r30
        rol r31
        lsl r30
        rol r31
        lsl r30
        rol r31                     ; x <<  3 = 8*x

        add r30,r17                 ; +x
        adc r31,r18

        add r30,r17                 ; +x
        adc r31,r18                 ; so we have 8x + x + x = 10x

        clr result                  ;divide with 1024 to find the second decimal
        rcall div_1024
        mov r24,result
        add r24,r29                 ;turn result to ASCII character
        rcall lcd_data              ;output to LCD

        reti

;div_1024:
;       cpi r31,0x04                ;is our number smaller than 1024?
;       brlo div_done               ;if yes then we are done
;       subi r31,0x04               ;if not then subtract 1024
```

```
;       inc result                  ;and add 1 to our quotient (piliko)
;       rjmp div_1024

;div_done:
;       ret

div_1024:
        mov result, r31             ;could do 10 right rotations of r30,r31
        andi result, 0b11111100     ;or we can keep 6 MSB (/1024)
        lsr result                  ;and rotate right 2 times
        lsr result
        andi r31, 0b00000011        ;we keep the remain
        reti

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 ,PD3              ; Enable Pulse
        nop
        nop
        cbi PORTD ,PD3
        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 ,PD3              ; Enable Pulse
        nop
        nop
        cbi PORTD ,PD3
        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 ,PD3              ; Enable Pulse
        nop
        nop
        cbi PORTD ,PD3
        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 ,PD3              ; Enable Pulse
        nop
        nop
        cbi PORTD ,PD3
        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

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 ,PD3                  ; Enable Pulse
        nop
        nop
        cbi PORTD ,PD3

        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 ,PD3                  ; Enable Pulse
        nop
        nop
        cbi PORTD ,PD3
        ret

lcd_data:
        sbi PORTD ,PD2                  ; 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 ,PD2                  ; 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
```

```
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 cycle

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
       nop

       ret
```

Σχόλια/Περιγραφή του κώδικα:

Ο παραπάνω κώδικας λειτουργεί με πολύ απλό τρόπο. Όπως έχουμε δει και απ το PDF, υπάρχουν έτοιμες συναρτήσεις που χρησιμοποιούμε για να μεταφέρουμε στην οθόνη τα ζητούμενα μεγέθη. Τέτοια ειναι το write 2 nibbles που μεταφέρει 4 bits τη φορά στην οθόνη και άλλες συναρτήσεις όπως η lcd_data που είναι για μεταφορά δεδομένων, το lcd_command που περνάει την εντολή, το lcd_clear_display που καθαρίζει την οθόνη και το lcd_init που αρχικοποιεί την οθόνη(χρειάζεται και καθυστέρηση). Απο εδώ και πέρα χρησιμοποιούμε τον χρονιστή και τον ADC που χρησιμοποιήσαμε και στο προηγούμενο εργαστήριο έτσι ώστε να προκαλέσουμε διακοπές. Μετά το timer interrupt overflow το οποίο είναι επιλεγμένο με τέτοιο τρόπο ώστε να γίνεται interrupt κάθε 1 sec, μπαίνουμε μέσα στη ρουτίνα εξυπηρέτησης της διακοπής του χρονιστή και απλά κάνουμε ριστάρτ τον χρόνο. Ωστόσο η σημαντική δουλεία γίνεται μέσα στην ρουτίνα εξυπηρητέσησης της διακοπής του ADC. Προσπαθούμε απ την διακοπή του ADC να μετρήσουμε την τιμή της τάσης στο ποτενσιόμετρο POT4. Για να το κάνουμε αυτό παίρνουμε την ψηφίακη έξοδο του ADC (που αποθηκεύεται μέσα στον καταχωρήτη αυτόν), τον πολλαπλασιάζουμε με την τάση αναφοράς που ειναι τα 5 Volt και διαιρούμε με το 1024 επειδή είναι τα bits τα οποία μπορεί να δώσει ο ADC. Το κάνουμε αυτό διότι ουσιαστικά να μειώσουμε τα επιπέδα και να το φέρουμε σε πραγματική τιμή. Απο εκεί και πέρα παίρνουμε στη καινούργια τιμή. Αυτό το κάνουμε με ακρίβεια 2

δεκαδικών ψηφίων. Έπειτα μετατρέπουμε την τιμή που έχουμε σε χαρακτήρα ASCII και καλούμε τις απαραίτητες συναρτήσεις για να μεταφέρουμε την τιμή στην οθόνη.

Άσκηση 2:

Α) Παρακάτω φαίνεται ο κώδικας **C** που φτιάξαμε:

```
/*
 * main.c
 *
 * Created: 11/5/2025 5:09:50 PM
 *  Author: anast
 */

#define F_CPU 16000000UL    // 16 MHz
#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>
#include <avr/interrupt.h>

uint16_t timer_start = 49910 ;

void write_2_nibbles(uint8_t lcd_data) {
	uint8_t temp;

// Send the high nibble
	temp = (PIND & 0x0F) | (lcd_data & 0xF0);		// Keep lower 4 bits of
PIND and set high nibble of lcd_data
	PORTD = temp;						// Output the high nibble
to PORTD
	PORTD |= (1 << PD3);					// Enable pulse high
	_delay_us(1);						// Small delay to let the
signal settle
	PORTD &= ~(1 << PD3);					// Enable pulse low

// Send the low nibble
	lcd_data <<= 4;						// Move low nibble to high
nibble position
	temp = (PIND & 0x0F) | (lcd_data & 0xF0);// Keep lower 4 bits of PIND and set high
nibble of new lcd_data
	PORTD = temp;						// Output the low nibble to
PORTD
	PORTD |= (1 << PD3);					// Enable pulse high
	_delay_us(1);						// Small delay to let the
signal settle
	PORTD &= ~(1 << PD3);					// Enable pulse low
}

void lcd_data(uint8_t data)
{
	PORTD |= 0x04;						// LCD_RS = 1, (PD2 = 1) ->
For Data
	write_2_nibbles(data);					// Send data
	_delay_ms(1);
	return;
}
```

```
void lcd_command(uint8_t data)
{
	PORTD &= 0xFB;                                          // LCD_RS = 0, (PD2 = 0) ->
For Instruction
	write_2_nibbles(data);                                  // Send data
	_delay_ms(1);
	return;
}

void lcd_clear_display()
{
	uint8_t clear_disp = 0x01;                              // Clear display command
	lcd_command(clear_disp);
	_delay_ms(5);                                           // Wait 5 msec
	return;
	}

void lcd_init() {
	_delay_ms(200);                                         // Wait 5 msec

// Send 0x30 command to set 8-bit mode (three times)
	PORTD = 0x30;                                           // command to switch to 8-
bit mode
	PORTD |= (1 << PD3);                                    // Enable pulse
	_delay_us(1);
	PORTD &= ~(1 << PD3);                                   // Clear enable
	_delay_us(250);                                         // Wait 250 µs

	PORTD = 0x30;                                           // Repeat command to ensure
mode set
	PORTD |= (1 << PD3);
	_delay_us(1);
	PORTD &= ~(1 << PD3);
	_delay_us(250);

	PORTD = 0x30;                                           // Repeat once more
	PORTD |= (1 << PD3);
	_delay_us(1);
	PORTD &= ~(1 << PD3);
	_delay_us(250);

// Send 0x20 command to switch to 4-bit mode
	PORTD = 0x20;
	PORTD |= (1 << PD3);
	_delay_us(1);
	PORTD &= ~(1 << PD3);
	_delay_us(30);

// Set 4-bit mode, 2 lines, 5x8 dots
	lcd_command(0x28);

// Display ON, Cursor OFF
	lcd_command(0x0C);

// Clear display
	lcd_clear_display();

// Entry mode: Increase cursor, no display shift
```

```
	lcd_command(0x06);
}

ISR(TIMER1_OVF_vect)
{
//	ADCSRA |= (1 << ADSC);				/* Start conversion from
analog to digital.
//							* Answer saved in
ADC 16-bit register by default.
//							*/
//while(ADCSRA & (1 << ADSC));			// Wait for conversion to
end
	TCNT1H = (timer_start>>8);			//must begin-time so i have
interrupt by timer overflow every 1 second
	TCNT1L = (timer_start);				// Re-initialize again the
timer for overflow
//TCCR1B = (1 << CS10) | (1 << CS12);
	lcd_clear_display();
	ADCSRA |= 0x40;					//start ADC
	while((ADCSRA&0x40)!= 0x00);			//Wait for conversion to
end
	uint16_t temp;
	uint16_t result;

	result = ADC;
	result = result*5;

	temp = result % 1024;
	result = result/1024;
	result += '0';
	lcd_data(result);

	lcd_data('.');

	result = temp*10;
	temp = result % 1024;
	result = result/1024 ;
	result += '0';
	lcd_data(result);

	result = temp*10;
	result = result/1024;
	result += '0';
	lcd_data(result);
}

int main(){
	lcd_init();
	lcd_clear_display();

	sei();					//enable interrupts
	DDRB = 0xFF;
	PORTB = 0x00;

	DDRC = 0x00;

	DDRD = 0xff;
	PORTD = 0x00;
```

```
//ADC enable
	ADMUX = 0x43;
	ADCSRA = 0x87;                                  //disable interrupts from ADC

//time set up
	TCCR1B = 0x00;                                  //freeze timer
	TIMSK1 = 0x01;                                  //allowing overflow interrupt
	TCNT1H = (timer_start>>8);                      //set timer_start to have overflow
interrupt every 1 s
	TCNT1L = (timer_start);

	TCCR1B = 0x05;                                  //start timer with
16000000/1024=15.625 hz

	while(1);

}
```

Σχόλια:

Η περιγραφή του κώδικα είναι ίδια με αυτή της προηγούμενης άσκησης.

Άσκηση 3:

Παρακάτω φαίνεται ο κώδικας **C** που φτιάξαμε:

```
/*
 * main.c
 *
 * Created: 11/6/2025 1:15:35 PM
 *  Author: anast
 */

#define F_CPU 16000000UL                                    // 16 MHz
#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>
#include <avr/interrupt.h>

#define SENSITIVITY 0.0129                                  // Sensitivity in
A/ppm
#define VREF 5.0                                            // Reference voltage
in Volts
#define Vgas0 0.1                                           // Vgas0 in Volts
#define CO_threshold 75                                     // Threshold in ppm

uint16_t timer_start = 63972 ;

volatile float V_in = 0.0;
volatile int CO_ppm = 0;
volatile uint8_t leds = 0x00;
volatile int blink = 0;
volatile int first_time = 1;
```

```
void write_2_nibbles(uint8_t lcd_data) {
	uint8_t temp;

// Send the high nibble
	temp = (PIND & 0x0F) | (lcd_data & 0xF0);// Keep lower 4 bits of PIND and set high
nibble of lcd_data
	PORTD = temp;                                          // Output the high nibble
to PORTD
	PORTD |= (1 << PD3);                                   // Enable pulse high
	_delay_us(1);                                          // Small delay to let the
signal settle
	PORTD &= ~(1 << PD3);                                  // Enable pulse low

// Send the low nibble
	lcd_data <<= 4;                                        // Move low nibble to high
nibble position
	temp = (PIND & 0x0F) | (lcd_data & 0xF0);// Keep lower 4 bits of PIND and set high
nibble of new lcd_data
	PORTD = temp;                                          // Output the low nibble to
PORTD
	PORTD |= (1 << PD3);                                   // Enable pulse high
	_delay_us(1);                                          // Small delay to let the
signal settle
	PORTD &= ~(1 << PD3);                                  // Enable pulse low
}

void lcd_data(uint8_t data)
{
	PORTD |= 0x04;                                         // LCD_RS = 1, (PD2 = 1) ->
For Data
	write_2_nibbles(data);                                 // Send data
	_delay_ms(1);
	return;
}

void lcd_command(uint8_t data)
{
	PORTD &= 0xFB;                                         // LCD_RS = 0, (PD2 = 0) ->
For Instruction
	write_2_nibbles(data);                                 // Send data
	_delay_ms(1);
	return;
}

void lcd_clear_display()
{
	uint8_t clear_disp = 0x01;                             // Clear display command
	lcd_command(clear_disp);
	_delay_ms(5);                                          // Wait 5 msec
	return;
}

void lcd_init() {
	_delay_ms(200);                                        // Wait 5 msec

// Send 0x30 command to set 8-bit mode (three times)
	PORTD = 0x30;                                          // command to switch to 8-
bit mode
```

```
        PORTD |= (1 << PD3);                                    // Enable pulse
        _delay_us(1);
        PORTD &= ~(1 << PD3);                                   // Clear enable
        _delay_us(250);                                         // Wait 250 µs

        PORTD = 0x30;                                           // Repeat command to ensure
mode set
        PORTD |= (1 << PD3);
        _delay_us(1);
        PORTD &= ~(1 << PD3);
        _delay_us(250);

        PORTD = 0x30;                                           // Repeat once more
        PORTD |= (1 << PD3);
        _delay_us(1);
        PORTD &= ~(1 << PD3);
        _delay_us(250);

// Send 0x20 command to switch to 4-bit mode
        PORTD = 0x20;
        PORTD |= (1 << PD3);
        _delay_us(1);
        PORTD &= ~(1 << PD3);
        _delay_us(30);

// Set 4-bit mode, 2 lines, 5x8 dots
        lcd_command(0x28);

// Display ON, Cursor OFF
        lcd_command(0x0C);

// Clear display
        lcd_clear_display();

// Entry mode: Increase cursor, no display shift
        lcd_command(0x06);
}

void gas_detected()
{
        lcd_clear_display();
        lcd_data('G');
        lcd_data('A');
        lcd_data('S');
        lcd_data(' ');
        lcd_data('D');
        lcd_data('E');
        lcd_data('T');
        lcd_data('E');
        lcd_data('C');
        lcd_data('T');
        lcd_data('E');
        lcd_data('D');
        return;
}

void gas_clear()
{
```

```
        lcd_clear_display();
        lcd_data('C');
        lcd_data('L');
        lcd_data('E');
        lcd_data('A');
        lcd_data('R');
        return;
}

uint8_t calc_leds(int CO_ppm)
{
        if (CO_ppm <= 15) return 0x00;                                  // if Cx <= 15ppm,
open none
        if (CO_ppm <= 35) return 0x01;                                  // if Cx <= 35ppm,
open PB0
        if (CO_ppm <= 75) return 0x03;                                  // if Cx <= 75ppm,
open PB0-PB1
        if (CO_ppm <= 175) return 0x07;                                 // if Cx <= 175ppm,
open PB0-PB2 -> GAS DETECTED
        if (CO_ppm <= 275) return 0x0F;                                 // if Cx <= 275ppm,
open PB0-PB3
        if (CO_ppm <= 375) return 0x1F;                                 // if Cx <= 375ppm,
open PB0-PB4
        return 0x3F;                                                    // if Cx > 375ppm,
open PB0-PB5
}

int calc_CO()
{
    V_in = (ADC * VREF) / 1024.0;                                       // normalize ADC to
VREF (float)
    CO_ppm = (int)((V_in - Vgas0) / SENSITIVITY);                       // convert float to
int (from the link provided page 3)
    return CO_ppm;
}

ISR(TIMER1_OVF_vect)
{
        cli();
        TCNT1 = timer_start;
        ADCSRA |= 0x40;                                                         //start ADC
        return;
}

// Interrupt routine for ADC
ISR(ADC_vect)
{
        cli();
        CO_ppm = calc_CO();                                                     // Calculate
CO concentration
        leds = calc_leds(CO_ppm);

        if (CO_ppm > CO_threshold)
        {
                first_time = 0;
                if(blink==0)gas_detected();
                blink = 1;                                                      // Blink
necessary leds until CO_ppm drops below 75ppm
```

```
                return;
        }
        else if (first_time==1)
        {
                lcd_clear_display();
                return;
        }
        else
        {
                if(blink==1)gas_clear();
                blink = 0;
                return;
        }
}

int main(){
        lcd_init();

        sei();                                                  //enable interrupts
        DDRB = 0x3F;                                            //set PB0-PB5 as
output
        PORTB = 0x00;

        DDRC = 0x00;                                            //ADC

        DDRD = 0xFF;                                            //LCD
        PORTD = 0x00;

//ADC enable
        ADMUX = 0x43;                                           //input from POT4
and REFS0 = 1 for Voltage reference
        ADCSRA = 0x8F;                                          //enable interrupts
from ADC

//time set up
        TCCR1B = 0x00;                                          //freeze timer
        TIMSK1 = 0x01;                                          //allowing overflow
interrupt
        TCNT1 = timer_start;                                    //prescaler = 1024
so 16 MHZ/1024 = 15625 cycles for 1 s
                                                                // 0,1 * 15625 =
1562,5 cycles for 100 ms
                                                                // 65535 - 1563 =
63972 (=timer_start)
        TCCR1B = 0x05;                                          //start timer with
16000000/1024=15.625 hz
        lcd_clear_display();
        while(1)
        {
                while(blink == 0)
                {
                        PORTB = leds;                            // Steady leds
                        sei();
                }
                while(blink == 1)
                {
                        PORTB = leds;                            // Blinking leds
                        _delay_ms(50);
```

```
                    PORTB = 0x00;
                    _delay_ms(50);
                    sei();
                }
        }
}
```

Σχόλια:

Και εδώ πέρα ο κώδικας χρησιμοποιεί στοιχεία που έχουν και οι προηγούμενοι 2 κώδικες. Χρησιμοποιεί για την υπέρβαση των 75ppm και την επαναφορά του gas κάτω απ τα 75ppm 2 έτοιμες συναρτήσεις που κρατάνε τα output GAS DETECTED και CLEAR. Για τον έλεγο της υπέρβασης των 75ppm απ το POT4 μετράμε το ppm του gas εκείνη την στιγμή. Για να το κάνουμε αυτό χρησιμοποιούμε τον τύπο

$$Cx = \frac{1}{M} \cdot (Vgas - Vgas_0),$$

Όπου M φαίνεται παρακάτω:

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

Και το Vgas όπως είπαμε ειναι η τάση που παίρνουμε με τους καταλλήλους πολλαπλασιασμούς (x3.3) και διαιρέσεις (1024) όπως και στην άσκηση 1. Απο εκεί και πέρα συγκρίνουμε την τιμή Cx κάθε 100ms με τα 75ppm και άναλογα τη τιμή αυτή καλούμε την αντίστοιχη συνάρτηση για την έξοδο στην οθόνη και ανάβουμε τα κατάλληλα λαμπάκια.

****Για να επιβεβαιώσουμε ότι η σύγκριση που κάνουμε βγάζει ορθά αποτελέσματα πηγαίνουμε το ποτενσιόμετρο στην τιμή εναλλαγής του CLEAR και του GAS DETECTED (κοντά στα 75ppm) και βλέπουμε στην μνήμη RAM ποια τιμή έχει αποθηκευτεί στον καταχωρητή ADC. Η διεύθυνση του καταχωρητή αυτού βρίσκεται στις θέσεις 0x0078,0x0079 και διάβαζουμε την τιμή DA=218. Σύμφωνα με τους παραπάνω τύπους και την σχέση Vin=(ADC)(Vref)/1024 , πράγματι διασταυρώνουμε ότι το αποτέλεσμα είναι σωστό****