ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2024-2025

ΑΘΗΝΑ 25 Οκτωβρίου 2024

# 4η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

## ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

## ΟΜΑΔΑ 23

## Συνεργάτες

Νικόλαος Αναγνώστου
03121818

Νικόλαος Λάππας
03121098

**Ζήτημα 4.1:**

Σε αυτήν την άσκηση έπρεπε να χρησιμοποιήσουμε το POT1 σαν αναλογική είσοδο και αυτή την είσοδο να την κάνουμε ADC μετατροπή και κατόπιν χρησιμοποιώντας τον ακόλουθο τύπο: $V_{IN} = \frac{ADC}{1024} V_{REF}$

να εκτυπώνουμε το Vin με ακρίβεια 2 δεκαδικών.

Επίσης, ζητήθηκε να μην περιμένουμε με polling τον adc και να ελέγχουμε με επανάληψη πότε θα ολοκληρώσει την μετατροπή ο adc, αλλά να χρησιμοποιήσουμε την διακοπή ολοκλήρωσης του adc.

Να τονιστεί ότι οι πολλαπλασιασμοί γίνανε με ολισθήσεις του r30 και r31 αριστερά και ανάλογες προσθέσεις και όχι με την εντολή mul/muli.

Ο κώδικας του προγράμματος σε assembly φαίνεται παρακάτω:

```
.include "m328pbdef.inc"


.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 time_essential = 49910
.def div1 = r20
.def initiator = r16 ;used for pre-building and setting purposes...

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

  start:
        ldi r27,10 ;for multiplying purposes
        ldi r29,'0';for turning the digits to ascii characters.

        ser initiator

        out DDRD,initiator

        out DDRB,initiator

        clr initiator
        out DDRC,initiator


        out PORTB,initiator

        out PORTD,initiator

        out EIMSK,initiator ;ensure every other normal interrupt is disabled
```

```
        rcall lcd_init
        rcall lcd_clear_display
        sei ;enable interrupts

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

        ;pre-building ADC
        ldi initiator,0x41 ;ADC1 as input
        sts ADMUX,initiator
        ldi initiator,0x8F ;enable conversion-done interrupt
        sts ADCSRA,initiator;sts it to ADCSRA
        ;adc completed


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

        ;preload the tcnt1 with the correct time_essential
        ldi initiator,HIGH(time_essential)
        sts TCNT1H,initiator
        ldi initiator,LOW(time_essential)
        sts TCNT1L,initiator

        ldi initiator,0x05
        sts TCCR1B,initiator ;timer has started now



        main:
        rjmp main ;looping

time_has_come:
        sei
        clr initiator ;we can test it without stopping the timer on ntuaboard for fun! But this is more secure
        sts TCCR1B,initiator ;defuse timer
        ldi initiator,HIGH(time_essential) ;reload TCNT1 to time_essential
        sts TCNT1H,initiator
        ldi initiator,LOW(time_essential)
        sts TCNT1L,initiator
        ldi initiator,0x05
        sts TCCR1B,initiator ;restart



        ldi initiator,0xCF ;setting adsc to 1...now the convertion has been started
        sts ADCSRA,initiator
        ;the job now is done...waiting for the completion interrupt to happen...nothing we can do
        reti
```

```
ADC_ready:
        rcall lcd_clear_display
        sei
        lds r30 , ADCL
        lds r31 , ADCH


        rcall lcd_clear_display

        mov r17,r31
        mov r18,r30

        lsl r30
        rol r31

        lsl r30
        rol r31

        add r30,r18
        adc r31,r17 ;*5


        ;/1024
        ;Vin = 5*adc/2^10
        ;mov r24,r30
        ;add r24,r29
        ;rcall lcd_data

        ldi div1,0
        rcall helper_div
        mov r24,div1
        add r24,r29
        rcall lcd_data


        ldi r24,'.'
        rcall lcd_data


        ;multiplying to get first decimal
        mov r17,r31
        mov r18,r30

        lsl r30
        rol r31
        lsl r30
        rol r31
        lsl r30
        rol r31

        add r30,r18
        adc r31,r17

        add r30,r18
        adc r31,r17

        clr div1
        rcall helper_div
        mov r24,div1
        add r24,r29
        rcall lcd_data
```

```
            mov r17,r31
            mov r18,r30

            lsl r30
            rol r31
            lsl r30
            rol r31
            lsl r30
            rol r31

            add r30,r18
            adc r31,r17

            add r30,r18
            adc r31,r17

            clr div1
            rcall helper_div
            mov r24,div1
            add r24,r29
            rcall lcd_data

            reti

  helper_div:
  cpi r31,0x04; -1024
  brlo helper_done
  subi r31,0x04
  inc div1
  rjmp helper_div

  helper_done:
            ret

wait_x_msec:
ldi r26,LOW(15984);1 cycle
ldi r27,HIGH(15984);1 cycle
helper:
  sbiw r26,4 ;2 cycles
  brne helper ;2 cycles or 1 cycle for the last iteration
;15984 -> helper consumes 15983 cycles
;so after helper we consume totally 15985 cycles

sbiw r24,1 ;2 cycle
breq last_msec ;1 cycle but if last msec 2 cycles

;for all msec except from the last -> 15985 + 2 + 1 = 15988 cycles
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop ;10 cycles
brne wait_x_msec ;2 cycles total 16000 cycles with this operation

last_msec:
```

```
;in the last iteration (last msec) we have 15989 cycles
nop
nop
nop
nop

;extra 4 cycles -> 15993 cycles
ret ;4 cycles

;with ret and rcall we calculated exactly 16000 cycles again
;so in both cases we end up having 16000 cycles -> 1 msec * (desired time)

lcd_init:

                    ldi r24 ,low(200) ;
                    ldi r25 ,high(200) ; Wait 200 mSec
                    rcall wait_x_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 ,30  ;
                    ldi r25 ,0  ; Wait 250uSec
                    rcall wait_x_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 ,30  ;
                    ldi r25 ,0  ; Wait 250uSec
                    rcall wait_x_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 ,30             ;
                    ldi r25 ,0
                    rcall wait_x_msec

                    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 ,30             ;
                    ldi r25 ,0
                    rcall wait_x_msec

                    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 ,30         ;
                ldi r25 ,0
                rcall wait_x_msec
                ret

lcd_command:
                cbi PORTD ,PD2
                rcall write_2_nibbles
                ldi r24 ,30
                ldi r25 ,0
                rcall wait_x_msec
                ret

lcd_clear_display:
                ldi r24 ,0x01
                rcall lcd_command
                ldi r24 ,low(5)
                ldi r25 ,high(5)
                rcall wait_x_msec
                ret
```

**Ζήτημα 4.2:**

Σε αυτή την άσκηση έπρεπε να κάνουμε το ίδιο ακριβώς σε C αλλά τώρα έπρεπε να περιμένουμε τον adc να ολοκληρώσει την μετατροπή με την τεχνική polling.

Αξίζει να σημειώσουμε ότι τόσο στην 4.2 όσο και στην 4.1 αξιοποιήσαμε τον timer1 για να επιβάλλουμε καθυστέρηση 1 sec.

Παρακάτω δίνεται ο κώδικας σε C:

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

uint16_t counter_begin = 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(5);
  return;
}
void lcd_command(uint8_t data)
{
  PORTD &= 0xFB;          // LCD_RS = 0, (PD2 = 0) -> For Instruction
  write_2_nibbles(data);    // Send data
  _delay_ms(5);
  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);

  // Send 0x30 command to set 8-bit mode (three times)
  PORTD = 0x30;          // Set command to switch to 8-bit mode
  PORTD |= (1 << PD3);     // Enable pulse
  _delay_us(1);
  PORTD &= ~(1 << PD3);    // Clear enable
  _delay_us(30);         // Wait 250 Âµs

  PORTD = 0x30;          // Repeat command to ensure mode set
  PORTD |= (1 << PD3);
  _delay_us(1);
  PORTD &= ~(1 << PD3);
  _delay_us(30);

  PORTD = 0x30;          // Repeat once more
  PORTD |= (1 << PD3);
  _delay_us(1);
  PORTD &= ~(1 << PD3);
  _delay_us(30);

  // 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: Increment 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 = (counter_begin>>8); //must begin-time so i have interrupt by timer overflow every 1 second
  TCNT1L = (counter_begin);               // Re-initialize again the timer for overflow
  //TCCR1B = (1 << CS10) | (1 << CS12);
  lcd_clear_display();
  ADCSRA |= 0x40;            //start ADC
  while((ADCSRA&0x40)!= 0x00){}  //while ?he conversion last hold fast
  uint16_t save1;
  uint16_t result;

  result = ADC;
  result = result*5;
```

```
  save1 = result % 1024;
  result = result/1024;
  result += '0';
  lcd_data(result);


  lcd_data('.');


  result = save1*10;
  save1 = result % 1024;
  result = result/1024 ;
  result += '0';
  lcd_data(result);


  result = save1*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 = 0x41;
  ADCSRA = 0x87;//ban interrupts from adc


  //time set up
  TCCR1B = 0x00; //freeze timer
  TIMSK1 = 0x01; //allowing overflow interrupt
  TCNT1H = (counter_begin>>8); //must begin-time so i have interrupt by timer overflow every 1 second
  TCNT1L = (counter_begin);

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


   while(1){}
```

**Ζήτημα 4.3:**

Ζητούμενο της συγκεκριμένης άσκησης είναι να προσομοιώσουμε έναν αισθητήρα για μονοξείδιο του άνθρακα (CO). Αυτό το επιτυγχάνουμε ρυθμίζοντας τον αισθητήρα ως την αναλογική είσοδο A2 του μικροελεγκτή (POT3). Για την μετατροπή της αναλογικής τάσης εισόδου από Volt στην δεδομένη μονάδα μέτρησης που θέλουμε, (ppm), χρησιμοποιούμε τον σύνδεσμο που μας δίνεται, και συγκεκριμένα, κάνουμε χρήση των τύπων:

$$V_{in} = \frac{(ADC \ * \ VREF)}{1024} \ και \ CO_{ppm} = \frac{(V_{in} - \ Vgas0)}{SENSITIVITY}$$

όπου έχουμε ορίσει κατάλληλα τις σταθερές με βάση τα δεδομένα. Τις σταθερές στο πρόβλημά μας της ορίσαμε με $\#define$. Τους υπολογισμούς αυτούς τους κάνουμε μέσα στην συνάρτηση $int \ calc\_CO\_concentration(uint16\_t \ ADC\_value)\{\}$ και ανάλογα με το τι τιμή θα επιστρέψει, (συγκριτικά πάντα με το 70ppm), ανάβουμε συγκεκριμένα λαμπάκια τις επιλογής μας. Η επιλογή των κατάλληλων leds αναλόγως του επιπέδου επικινδυνότητας φαίνεται στην συνάρτηση $uint8\_t \ open\_LEDs(int \ Cx)\{\}$. Όταν έχουμε τιμή συγκέντρωσης του μονοξειδίου πάνω από τα 70ppm, τότε τα αντίστοιχα λαμπάκια που έχουμε ορίσει αναβοσβήνουν.

Για την επικοινωνία με την οθόνη LCD της πλακέτας, έχουμε μετατρέψει τον δοθέντα κώδικα της assembly σε c και προσέχουμε να κάνουμε κάθε φορά που φορτώνουμε νέο ή επόμενο μήνυμα να έχει προηγηθεί καθαρισμός της οθόνης, αλλιώς θα τυπώνονται συνεχόμενα τα μηνύματα. Όταν έχουμε CO_ppm > 7ppm, τυπώνεται στην οθόνη το μήνυμα ‘GAS DETECTED’ και μένει εκεί μέχρι να πέσει η τιμή σε ppm. Αντίστοιχα, όταν έχουμε τιμή μικρότερη από 70 ppm, τότε τυπώνεται στην οθόνη το μήνυμα ‘CLEAR’. Αυτό με το συγκεκριμένο μήνυμα θα γίνεται μόνο αν έχει προηγηθεί το μήνυμα ‘GAS DETECTED’. Αν ξεκινήσει η υλοποίηση της άσκησης και προκύψει τιμή απευθείας μικρότερη των 70ppm, τότε κανένα μήνυμα δεν θα τυπωθεί στην οθόνη. Αν, όμως, απευθείας προκύψει >70ppm, τότε θα τυπωθεί το ‘GAS DETECTED’ και μετά όταν πέσει κάτω από τα 70ppm, θα τυπώσουμε ‘CLEAR’.

Τέλος, αξίζει να πούμε ότι προκαλούμε διακοπές με χρονιστή (TIMER1) κάθε 100ms. Αυτό το επιτυγχάνουμε αρχικοποιώντας τον καταχωρητή TCNT1 στην τιμή 63972. Άρα κάθε 100msec θα προκαλείται διακοπή υπερχείλισης του timer1, η οποία θα μεταφέρει την εκτέλεση του κώδικα στην ρουτίνα εξυπηρέτησης της διακοπής αυτής. Στην ρουτίνα αυτή αρχικοποιούμε την ADC μετατροπή (δειγματοληπτούμε την τάση που διαβάζει ο μικροελεγκτής στο POT3) και μόλις γίνει η μετατροπή, επειδή έχουμε ενεργοποιήσει το interrupt flag του καταχωρητή ADCSRA, θα γίνει trigger και της διακοπής ADC, και θα μεταφερθούμε στην ρουτίνα εξυπηρέτησης αυτής της διακοπής (αφού ολοκληρώσει η προηγούμενη). Στην ρουτίνα εξυπηρέτησης της ADC διακοπής έχουμε βάλει όλο τον κώδικα που καλούμαστε να υλοποιήσουμε. Ο κώδικάς μας φαίνεται παρακάτω:

```
#define F_CPU 16000000UL  // 16 MHz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <stdbool.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 70          // Threshold in ppm

volatile float V_in = 0.0;
volatile int CO_ppm = 0;
volatile uint8_t leds = 0x00;
volatile bool gas_detected = false;

void ADC_init() // Must add the interrupts
{
  /* Chose ADC channe2 (ADC2) to read from POT3, ends in ...0010
   * For voltage reference selection: REFS0 = 1, REFS1 = 0
   * Right adjustment: ADLAR = 0
   * ADC0: MUX3 = 0, MUX2 = 0, MUX1 = 1, MUX0 = 0
   */
  ADMUX = (1 << REFS0) | (1 << MUX1);
  // Same as the above ADMUX = 0b01000010;

  /* Enable ADC: ADEN = 1
   * No conversion from analog to digital YET: ADSC = 0
   * Enable ADC interrupt: ADIE = 1

   * Prescaler: f_ADC = 16MHz / prescaler and
   * 50kHz <= f_ADC <= 200kHz for 10-bits accuracy. So,
   * division factor = 128 -> gives f_ADC = 125kHz -> may NOT be needed here
   */
  ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0) | (1<<ADIE);
  // Same as the above ADCSRA = 0b10000111;

  // Only using ADC2, so for less power consumption set ADCi to disable their
  // digital input that we don't use
  //DIDR0 = (1 << ADC2D); // Only disable digital input on ADC2
}

// Cx -> CO concentration
uint8_t open_LEDs(int Cx)
{
  if (Cx <= 10) return 0x00;       // if Cx <= 10ppm, open none
  if (Cx <= 30) return 0x01;       // if Cx <= 30ppm, open PB0
  if (Cx <= 70) return 0x03;       // if Cx <= 70ppm, open PB0-PB1
  if (Cx <= 170) return 0x07;      // if Cx <= 170ppm, open PB0-PB2 -> GAS DETECTED
  if (Cx <= 270) return 0x0F;      // if Cx <= 270ppm, open PB0-PB3
  if (Cx <= 370) return 0x1F;      // if Cx <= 370ppm, open PB0-PB4
  return 0x3F;                     // if Cx > 370ppm, open PB0-PB5
}

int calc_CO_concentration()
{
  /* V_in: (normalized) voltage from analog input A2 of microprocessor
   * ADC_value: value (10-bits = 1024) that ADC conversion gets
   * VREF: voltage reference by default
   */
  V_in = (ADC * VREF) / 1024.0;            // float

  /* The target gas concentration CO_ppm is calculated by the following
   * method (from the link provided page 3)
   */
```

```
  CO_ppm = (int)((V_in - Vgas0) / SENSITIVITY);   // convert float to int

  return CO_ppm;
}

/* Timer Interrupt routine
 * When overflow of TCNT1 occurs, the program will come here
 * and we want the ADC conversion to start.
 */
ISR(TIMER1_OVF_vect)
{
  ADCSRA |= (1 << ADSC);        /* Start conversion from analog to digital.
                                 * Answer saved in ADC 16-bit register by default.
                                 */
  sei();                        // Because interrupts are disabled
  TCNT1 = 63972;                // Re-initialize again the timer for overflow
}

// Interrupt routine for ADC
ISR(ADC_vect)
{
  CO_ppm = calc_CO_concentration();// Calculate CO concentration

  if (CO_ppm > CO_threshold)              // Blink necessary leds until CO_ppm drops down to 70ppm or less
  {
    if(gas_detected)                      // If already gas detected
    {
      leds = open_LEDs(CO_ppm);           // Indicate which leds should be on

      if(PORTB == leds)
      {
        PORTB = 0x00;
        _delay_ms(50);
        PORTB = leds;
      }
      else
      {
        PORTB = leds;
        _delay_ms(50);
        PORTB = 0x00;
        _delay_ms(50);
        PORTB = leds;
      }
    }
    else                  // If first time gas detection
    {
      gas_detected = true;
      leds = open_LEDs(CO_ppm);           // Indicate which leds should be on
      PORTB = leds;
      detected_gas();                     // Display 'GAS DETECTED'
    }
  }
  else if (CO_ppm <= CO_threshold) // Just open necessary leds
  {

    leds = open_LEDs(CO_ppm);   // Indicate which leds should be on
    PORTB = leds;               // Steady led display without blinking

    if(gas_detected)            // If we have detected gas, clear screen and display 'CLEAR'
    {
      clear_gas();              // Display 'CLEAR'
```

```
    }
    gas_detected = false;
  }
}


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(5);                  // Wait 5 msec
  return;
}

void lcd_command(uint8_t data)
{
  PORTD &= 0xFB;                 // LCD_RS = 0, (PD2 = 0) -> For Instruction
  write_2_nibbles(data);         // Send data
  _delay_ms(5);                  // Wait 5 msec
  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);

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

  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_ms(5);

  // 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: Increment cursor, no display shift
  lcd_command(0x06);
}

void clear_gas()
{
  lcd_clear_display();   // Clear display before new output
  lcd_data('C');
  lcd_data('L');
  lcd_data('E');
  lcd_data('A');
  lcd_data('R');
  return;
}

void detected_gas()
{
  lcd_clear_display();   // Clear display before new output
  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;
}


int main ()
```

```
{
  DDRB = 0x3F;                // Initialize (set) PB0-PB5 as output
  DDRD = 0xFF;                // LCD
  DDRC = 0x00;                // ADC

  ADC_init();                 // Initialize ADC

  TIMSK1 = (1 << TOIE1);          // Enable interrupts of TCNT1 (overflow)
  TCCR1B = (1 << CS10) | (1 << CS12); // Frequency of Timer1 16MHz/1024
  TCNT1 = 63972;                  /* This is because i want interrupts to happen
                                   * every 100msec. I have 16MHz microprocessor
                                   * frequency, so with prescaler = 1024 every
                                   * second occurs after 16MHz/1024 = 15625 cycles.
                                   * I want after each 0,1 second to trigger interrupt
                                   * so 0,1 * 15625 = 1562,5 equals almost to
                                   * 1563 cycles. So TCNT1 initial value should
                                   * be 65535-1563=63972.
                                   * That means that when 1563 cycles pass
                                   * there will occur overflow that will trigger interrupt.
                                   */
  lcd_init();
  _delay_ms(100);
  lcd_clear_display();

  sei();                      // Enable global interrupts


  while(1)                    // Loop infinitly enabling interrupts
  {
    sei();
  }
}
```