A close up of text

Description automatically generated

ΑΘΗΝΑ 25 Οκτωβρίου 2024

**4η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ**

**ΓΙΑ ΤΟ ΜΑΘΗΜΑ “Εργαστήριο Μικροϋπολογιστών”**

**ΟΜΑΔΑ 23**

**Συνεργάτες**

Νικόλαος Αναγνώστου Νικόλαος Λάππας

03121818 03121098

**Ζήτημα 4.1:**

**Ζήτημα 4.2:**

**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 Α2 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:

όπου έχουμε ορίσει κατάλληλα τις σταθερές με βάση τα δεδομένα. Τις σταθερές στο πρόβλημά μας της ορίσαμε με . Τους υπολογισμούς αυτούς τους κάνουμε μέσα στην συνάρτηση 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 . 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();

}

}