

4^η Εργαστηριακή Άσκηση

Πίκουλας Κωνσταντίνος 03120112
Στάμος Ανδρέας 03120018

1^η Άσκηση

Εξήγηση

Ο Timer1 ρυθμίζεται να προκαλεί overflow κάθε 1sec (λειτουργία Fast PWM με $\text{clk}/1024$). Ο ADC ρυθμίζεται να ξεκινά μια μετατροπή κάθε φορά που συμβαίνει Timer1 overflow. Επίσης ο ADC ρυθμίζεται να διαβάζει από την είσοδο A2 (που αντιστοιχεί στο POT3), να προκαλεί διακοπή όταν ολοκληρώσει την μετατροπή και να έχει για τάση αναφοράς την $V_{\text{ref}}=5\text{V}$.

Η οθόνη αρχικοποιείται σύμφωνα με την δοθείσα ρουτίνα `lcd_init`.

Στην ρουτίνα εξυπηρέτησης διακοπής, διαβάζουμε την τιμή που μέτρησε ο ADC και έπειτα βρίσκουμε τα 3 δεκαδικά ψηφία (x.yz) της $V_{\text{in}} = \text{ADC} * V_{\text{ref}}/1024$. Αυτό επιτυγχάνεται υπολογίζοντας αρχικά:

$$A = \text{ADC} * V_{\text{ref}} = \text{ADC} * 5 = \text{ADC} \ll 2 + \text{ADC}$$

Τότε (παρακάτω οι διαιρέσεις νοούνται ως ακέραιες διαιρέσεις):

$$x = \text{ADC} * V_{\text{ref}}/1024 = A / 2^{10} = A \gg 10$$

$$\text{Έπειτα είναι } y = ((A - 1024 * x) * 10) / 1024 = ((A - (A \gg 10) \ll 10) * 10) / 1024 = ((A \& 0x3FF) * 10) / 1024 = ((A \& 0x3FF) * 10) \ll 10.$$

Όμοια αν αφαιρέσουμε πάλι το ακέραιο μέρος y και πολλαπλάσιουμε με το 10, λαμβάνουμε:

Αν $B = (A \& 0x3FF) * 10$ (υπολογισμένο από πριν) είναι όμοια:

$$z = ((B \& 0x3FF) * 10) \ll 10.$$

Τέλος ένας αποδοτικός τρόπος για τον υπολογισμό του $10 * x$ είναι ο εξής:

$$10 * x = 8 * x + 2 * x = x \ll 3 + x$$

Επειδή ο παραπάνω υπολογισμός είναι κάπως δύσκολο να επιβεβαιώσει κανείς ότι συντάχθηκε σωστά σε AVR Assembly προσομοιώσαμε την AVR Assembly σε Python instruction προς instruction. Σε 1000000 τυχαίοποιημένες μετατροπές η μετατροπή συνέβαινε σωστά.

Τέλος για την μετατροπή των x,y,z σε ASCII χαρακτήρες τους προσθέσαμε τον ASCII κωδικό του '0'=48.

Κώδικας

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16
.equ PD0=0
.equ PD1=1
.equ PD2=2
.equ PD3=3
.equ PD4=4
.equ PD5=5
.equ PD6=6
.equ PD7=7

.org 0x0
    jmp reset
.org 0x2a
    rjmp ISR_ADC

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
```

```
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 r24,0x20 ; 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 ; display 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

```

mul10:

;r31:r30 *= 10 (in-place) ($10 \cdot x = 8 \cdot x + 2 \cdot x$)

```

mov r28, r30
mov r29, r31

```

```

lsl r28
rol r29
lsl r28
rol r29
lsl r28
rol r29

```

```

lsl r30
rol r31

```

```

add r30, r28
adc r31, r29

```

```

ret

```

ISR_ADC:

```

ldi r24, 1 << TOV1

```

```
out TIFR1, r24
```

```
;calc 5*ADC  
lds r30, ADCL  
lds r31, ADCH
```

```
mov r28, r30  
mov r29, r31
```

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

```
add r30, r28  
adc r31, r29
```

```
    ;r31:r30 is 5*ADC
```

```
    ;r17=d0 <- 5*ADC << 10  
mov r17, r31  
lsr r17  
lsr r17
```

```
andi r31, 0x03  
rcall mul10
```

```
    ;r18=d1  
mov r18, r31  
lsr r18  
lsr r18
```

```
andi r31, 0x03  
rcall mul10
```

```
    ;r19=d2  
mov r19, r31  
lsr r19  
lsr r19
```

```
    ;r17,r18,r29 := d0,d1d2  
    ;convert to ascii (add '0' = 48)
```

```
subi r17, -'0'  
subi r18, -'0'  
subi r19, -'0'
```

```
rcall lcd_clear_display
```

```
mov r24, r17  
rcall lcd_data
```

```

ldi r24, '.'
rcall lcd_data

mov r24, r18
rcall lcd_data

mov r24, r19
rcall lcd_data

reti
reset:
    ;stack init
ldi r24, HIGH(RAMEND)
out SPH, r24
ldi r24, LOW(RAMEND)
out SPL, r24

    ;read adc0, pc0
clr r24
out DDRC, r24
ser r24
out DDRB, r24
out PORTB, r24

    ;setup timer1 for overflow every 1sec (used for triggering adc).
ldi r24, 1 << WGM10 | 1 << WGM11
sts TCCR1A, r24
ldi r24, 1 << WGM12 | 1 << WGM13 | 1 << CS12 | 1 << CS10
sts TCCR1B, r24
ldi r24, HIGH(FOSC_MHZ * 1000000.0 * 1.0 / 1024.0 - 1) ;TOP <- 1sec in cycles/1024 - 1
sts OCR1AH, r24
ldi r24, LOW(FOSC_MHZ * 1000000.0 * 1.0 / 1024.0 - 1)
sts OCR1AL, r24

    ;setup adc, triggered at timer1 overflow.
ldi r24, 1 << REFS0 | 1 << MUX1
sts ADMUX, r24
ldi r24, 1 << ADEN | 1 << ADSC | 1 << ADIFSC | 1 << ADIFSC | 1 << ADIFSC | 1 << ADIFSC
sts ADCSRA, r24
ldi r24, 1 << ADTS1 | 1 << ADTS2
sts ADCSRB, r24
ldi r24, 1 << ADC2D
sts DIDR0, r24

    ;setup lcd
ser r24
out DDRD, r24
clr r24
call lcd_init
ldi r24, low(100)
ldi r25, high(100) ; delay 100 mS
call wait_msec

    ;start adc
lds r24, ADCSRA
ori r24, 1 << ADSC
sts ADCSRA, r24

```

```
sei
main:
rjmp main
```

2^η Άσκηση

Εξήγηση

Αρχικά αρχικοποιούμε τον ADC μας καθώς και θέτουμε το PORTD σαν output ώστε η θύρα να στέλνει δεδομένα στην LCD οθόνη. Ο ADC μετατρέπει την τάση του ποτενσιομέτρου κάθε 1sec. Έπειτα μετατρέπουμε την τιμή του ADC σε float. Πολλαπλασιάζουμε την τιμή αυτή επί 100 ώστε τα 2 δεκαδικά ψηφία να περάσουν στο ακέραιο μέρος του αριθμού.

- Το ακέραιο μέρος του αριθμού το παίρνουμε διαιρώντας με το 100.
- Το πρώτο δεκαδικό ψηφίο το παίρνουμε βρίσκοντας το υπόλοιπο με το 100 και μετά διαιρώντας με 10.
- Το δεύτερο δεκαδικό ψηφίο το παίρνουμε βρίσκοντας το υπόλοιπο με το 10.

Χρησιμοποιώντας την lcd_data() στέλνουμε το κατάλληλο ψηφίο προς απεικόνιση κάθε φορά (αφού προσθέσουμε το '0' στο τέλος του, για την μετατροπή σε ASCII χαρακτήρα).

Κώδικας

```
/*
 * File: main_k.c
 * Author: kostas
 *
 * Created on 31 October 2023, 00:48
 */

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



```

#include<stdbool.h>

/*
 *
 */
uint8_t command;
uint8_t command_temp;
float voltage_counter;

void write_2_nibbles() {
    uint8_t r25 = PIND;
    r25 &= 0x0f;
    command_temp = command & 0xf0;
    command_temp |= r25;
    PORTD = command_temp;
    PORTD |= (1<<3); // Enable Pulse
    asm("nop");
    asm("nop");
    PORTD &= ~(1<<3); // Clear Pulse

    command_temp = (command & 0x0f) << 4;
    command_temp |= r25;
    PORTD = command_temp;
    PORTD |= (1<<3); // Enable Pulse
    asm("nop");
    asm("nop");
    PORTD &= ~(1<<3); // Clear Pulse
    return;
}

void lcd_data() {
    PORTD |= (1<<2); //RS=1 (data)
    write_2_nibbles();
    _delay_ms(0.250);
    return;
}

void lcd_command() {
    PORTD &= ~(1 << 2); // Clear Enable
    write_2_nibbles();
    _delay_ms(0.250);
    return;
}

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

int main(int argc, char** argv) {
    // REFSn[1:0]=01 => Vref = 5V
    // MUXn[3:0] = 0010

```

```

// ADC2
ADMUX = 0b01000010;

//ADEN = 1 => ADC Enable
//ADCS = 0 => No conversion (yet)
// ADIE = 0 => Disable ADC Interrupt
// ADPS[2:0]=111 => fADC = 16MHz/128=125KHz
ADCSRA = 0b10000111;

// init screen
// DDRD output
DDRD = 0xFF;
_delay_ms(200); // wait for screen to initiali\e
for(int i = 0; i < 3; ++i) {
    PORTD = 0x30; // set 8 bit mode 3 times
    PORTD |= (1 << 3); // Enable pulse
    asm("nop");
    asm("nop");
    PORTD &= ~(1 << 3); // Clear Enable
    _delay_ms(0.250);
}

// switch to 4bit mode
PORTD = 0x20;
PORTD |= (1 << 3); // Enable pulse
asm("nop");
asm("nop");
PORTD &= ~(1 << 3); // Clear Enable
_delay_ms(0.250);

command = 0x28;
lcd_command();

command = 0x0c;
lcd_command();

lcd_clear_display();

while(1) {
    _delay_ms(1000); // delay 1 sec
    ADCSRA |= (1 << ADSC); // init conversion
    while(ADCSRA & 1 << ADSC); // it means adc is not finished
    voltage_counter = (float)ADC/1024 * 5.0;
    lcd_clear_display();
    voltage_counter *= 100;

    command = (int)voltage_counter/100 + '0';
    lcd_data();
    command = '.';
    lcd_data();
    command = (int)voltage_counter%100/10 + '0';
    lcd_data();
    command = (int)voltage_counter%10 + '0';
    lcd_data();
}

```

```

}

return (0);

}

```

3^η Άσκηση

Εξήγηση

Αρχικά αρχικοποιούμε τον ADC, το PORTD και το PORTD σε output αλλά και την LCD οθόνη. Ο ADC μετατρέπει την τάση απο το ποτενσιόμετρο κάθε 100ms περίπου. Χρησιμοποιούμε την τάση απο το ποτενσιόμετρο προκειμένου να προσομειώσουμε την τάση που θα παίρναμε απο τον αισθητήρα του τύπου **ULPSM-CO-968-001**. Από το datasheet του αισθητήρα έχουμε πως:

- $M \left(\frac{V}{ppm} \right) = SensitivityCode \left(\frac{nA}{ppm} \right) * TIAGain \left(\frac{kV}{A} \right) * 10^{-9} \left(\frac{A}{nA} \right) * 10^3 \left(\frac{V}{kV} \right)$
- $Cx = \frac{1}{M} * (Vgas - Vgas0)$

Η εκφώνηση δίνει $Vgas0 = 0.1V$ και $SensitivityCode = 129nA/ppm$.

Από το datasheet έχουμε ότι $TIAGain = 100$ διότι θέλουμε να μετρήσουμε μονοξείδιο του άνθρακα (CO).

Βρίσκουμε την τιμή Cx (συγκέντρωση του CO) από την τάση του ποτενσιομέτρου.

Εαν είναι μικρότερη απο 70ppm η οθόνη γράφει 'CLEAR' και καθώς αυξάνει η συγκέντρωση αυξάνουν και τα led του PORTB. Όταν ξεπεράσει τα 70ppm τα led είναι όλα αναμμένα (διότι έχουν συμπληρωθεί όλες οι στάθμες) και αρχίζουν να αναβοσβήνουν. Επιπλέον τώρα στην οθόνη γράφουμε 'GAS DETECTED'.

Κώδικας

```

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

uint8_t command;
uint8_t command_temp;
float vGas0 = 0.1;
float voltage_counter;
float sensitivityCode = 129;
float M_inverse = 77.5193;

```

```

float gas_concentration;
float leds_level;
uint8_t TIAGain = 100;

void write_2_nibbles() {
    uint8_t r25 = PIND;
    r25 &= 0x0f;
    command_temp = command & 0xf0;
    command_temp |= r25;
    PORTD = command_temp;
    PORTD |= (1<<3); // Enable Pulse
    asm("nop");
    asm("nop");
    PORTD &= ~(1<<3); // Clear Pulse

    command_temp = (command & 0x0f) << 4;
    command_temp |= r25;
    PORTD = command_temp;
    PORTD |= (1<<3); // Enable Pulse
    asm("nop");
    asm("nop");
    PORTD &= ~(1<<3); // Clear Pulse
    return;
}

void lcd_data() {
    PORTD |= (1<<2); //RS=1 (data)
    write_2_nibbles();
    _delay_ms(0.250);
    return;
}

void lcd_command() {
    PORTD &= ~(1 << 2); // Clear Enable
    write_2_nibbles();
    _delay_ms(0.250);
    return;
}

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

int main(int argc, char** argv) {
    // REFSn[1:0]=01 => Vref = 5V
    // MUXn[3:0] = 0011
    // ADC3
    ADMUX = 0b01000011;

    //ADEN = 1 => ADC Enable
    //ADCS = 0 => No conversion (yet)

```

```

// ADIE = 0 => Disable ADC Interrupt
// ADPS[2:0]=111 => fADC = 16MHz/128=125KHz
ADCSRA = 0b10000111;

// init screen
// DDRD output
DDRD = 0xFF;
// DDRB output
DDRB = 0xFF;
_delay_ms(200); // wait for screen to initiali\
for(int i = 0; i < 3; ++i) {
    PORTD = 0x30; // set 8 bit mode 3 times
    PORTD |= (1 << 3); // Enable pulse
    asm("nop");
    asm("nop");
    PORTD &= ~(1 << 3); // Clear Enable
    _delay_ms(0.250);
}

// switch to 4bit mode
PORTD = 0x20;
PORTD |= (1 << 3); // Enable pulse
asm("nop");
asm("nop");
PORTD &= ~(1 << 3); // Clear Enable
_delay_ms(0.250);

command = 0x28;
lcd_command();

command = 0x0c;
lcd_command();

lcd_clear_display();

while(1) {
    ADCSRA |= (1 << ADSC); // init conversion
    while(ADCSRA & 1 << ADSC); // it means adc is not finished
    voltage_counter = (float)ADC/1024 * 5.0;
    gas_concentration = M_inverse * (voltage_counter - vGas0);
    lcd_clear_display();
    if (gas_concentration >= 70) {
//        command = 'GAS DETECTED';
        command = 'G'; lcd_data();
        command = 'A'; lcd_data();
        command = 'S'; lcd_data();
        command = ' '; lcd_data();
        command = 'D'; lcd_data();
        command = 'E'; lcd_data();
        command = 'T'; lcd_data();
        command = 'E'; lcd_data();
        command = 'C'; lcd_data();
        command = 'T'; lcd_data();
        command = 'E'; lcd_data();
    }
}

```

```
    command = 'D'; lcd_data();
}
else {
    command = 'C'; lcd_data();
    command = 'L'; lcd_data();
    command = 'E'; lcd_data();
    command = 'A'; lcd_data();
    command = 'R'; lcd_data();
}
leds_level = gas_concentration * 6.0 / 70.0;
if (gas_concentration >= 70) PORTB = 0x3F;
else PORTB = (1<<(int)leds_level) - 1;
if (gas_concentration >= 70) {
    _delay_ms(250);
    PORTB = 0;
    _delay_ms(250);
}
else _delay_ms(100); //delay 1 sec
}
return (0);
}
```