Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών ΕΜΠ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΑΚΑΔ. ΕΤΟΣ 2024-2025

4ή εργαστηριακή άσκηση.

Ομάδα : 58
Ονοματεπώνυμο : Φέζος Κωνσταντίνος / Α.Μ : 03118076
Ονοματεπώνυμο : Τσουκνίδας Οδυσσέας Αρθούρος Ρήγας / Α.Μ : 03120043

Ζήτημα 4.1

Στο ζήτημα 4.1 καλούμαστε να υλοποιήσουμε πρόγραμμα σε AVR Assembly, το οποίο διαβάζει ανά 1 δευτερόλεπτο από την είσοδο 1 του ADC την τιμή τάσης που ορίζεται μέσω του POT2 ποτενσιόμετρου, και την τυπώνει με ακρίβεια 2 δεκαδικών ψηφίων στην οθόνη LCD. Συγκεκριμένα, ανά 1 δευτερόλεπτο θέτουμε την σημαία ADSC του καταχωρητή ADCSRA, ξεκινώντας έτσι μία μετατροπή ADC, η οποία όταν ολοκληρώνεται στέλνει διακοπή στον AVR. Στον χειρισμό της διακοπής έχουμε γράψει κώδικα ο οποίος διαβάζει την τιμή του ADC, την χειρίζεται κατάλληλα ώστε να βρει τις μονάδες και τα 2 πρώτα δεκαδικά ψηφία της τάσης Vin, τα οποία τελικά στέλνει στην οθόνη, κάνοντας χρήση των ρουτινών που μας δίνονται από το εργαστήριο.

```
.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

.org 0x00
    rjmp reset
.org 0x02A
    rjmp adc_handler

reset:

    ; initialize stack pointer
    ldi r24, LOW(RAMEND)
    out SPL, r24
    ldi r24, HIGH(RAMEND)
    out SPH, r24

    ; Set PORTD as output, PORTC as input (it has to be for the ADC)
    ser r24
    out DDRD, r24
    clr r24
    out DDRC, r24
```

```
    ; Initialize ADC and LCD screen
    rcall adc_init
    rcall lcd_init

    ldi r24, low(100)
    ldi r25, high(100)
    rcall wait_msec       ; Delay

    sei                   ; Global Interrupts Enable
main:
    ldi r24, low(1000)
    ldi r25, high(1000)
    rcall wait_msec

    lds r24, ADCSRA
    ori r24, (1<<ADSC)
    sts ADCSRA, r24       ; Each second: begin to take the ADC
measurement
    rjmp main


adc_init:
    ; RESF1-0: 01 -> AVCC with external capacitor at AREF pin
    ; MUX: 0001 -> ADC1
    ; ADLAR is set -> Left adjusted
    ldi r24, (1 << MUX0) | (1 << REFS0) | (1<<ADLAR)
    sts ADMUX, r24

    ; ADEN -> ADC Enable
    ; ADPS:111 -> ADC clock=16 MHz/128= 125 kHz (withing the desired
range)
    ; ADIE -> ADC Interrupt Enable
    ldi r24, (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0) |
(1<<ADIE)
    sts ADCSRA, r24

    ret


adc_handler:
    push r25
    push r24
    in r24, SREG
    push r24          ; Push in stack
```

```
    rcall lcd_clear_display
    lds r20, ADCL
    bst r20, 07       ; Set T flag if ADCL's MSB is 1
    clr r20
    bld r20, 00       ; Make r20's LSB 0 or 1
    lds r21, ADCH

    ldi r22, 5
    mul r21,r22       ; Multiply ADC's highest 8 bits with 5 -> result
    mov r29, r1       ; The result / 256 is stored in R1 already
    ldi r22, 100
    mul r0, r22       ;  Multiply the R0 (result % 256) with 100 to
find the 2 digits after the .
    mov r21, r1
    add r21, r20; Add 1 or 0
                ; If we've missed something useful in the two lower
bits of ADC add it here

    cpi r21, 100
    breq change_unit  ; If we've reached 100 (for example 1.99 -> 2.00)
go to change_unit
    rjmp continue     ; Else continue below

change_unit:          ; Increment the first_digit and make the 2 digits
after the . '0'
    inc r29
    ori r29, 0x30     ; Create the correct code for CG_ROM
    mov r24,r29
    call lcd_data
    ldi r24, '.'
    call lcd_data
    ldi r24, '0'
    call lcd_data
    ldi r24, '0'
    call lcd_data
    rjmp exit


continue:
    ori r29, 0x30     ;  Create the correct code for CG_ROM
    mov r24,r29
    call lcd_data
    ldi r24, '.'
    call lcd_data
```

```
    ldi r22, 0        ; Counter for decades

find_tens:      ; Break number between 0-99 to two digits
    inc r22
    subi r21, 10
    brcc find_tens

    dec r22           ; Here we have decades
    ldi r16, 10
    add r21, r16 ; Here we have singles

    ori r22,0x30 ; Create the correct code for CG_ROM
    mov r24,r22
    call lcd_data

    ori r21,0x30 ; Create the correct code for CG_ROM
    mov r24,r21
    call lcd_data


exit:
    pop r24
    out SREG, r24
    pop r24
    pop r25           ; Pop from stack
    reti        ; Return + set interrupts


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 (set bit in I/O Reg)
    nop
    nop
    cbi PORTD ,PD3 ; clear bit in I/O Reg
```

```
    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
    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
    rcall lcd_command ; clear display command
    ldi r24 ,low(5) ;
    ldi r25 ,high(5)
    rcall wait_msec ; Wait 5 mSec
    ret




lcd_init:
```

```
ldi r24 ,low(200)
ldi r25 ,high(200)
rcall wait_msec ; Wait for 200mSec


; We send 3 times 0x30, the command for 8-bit
ldi r24 ,0x30     ; command to switch to 8 bit
out PORTD ,r24
sbi PORTD ,PD3    ; enable pulse
nop
nop
cbi PORTD ,PD3

ldi r24 ,250
ldi r25 ,0
rcall wait_usec ; wait for 250 uSec

ldi r24 ,0x30     ; command to switch to 8 bit
out PORTD ,r24
sbi PORTD ,PD3    ; enable pulse
nop
nop
cbi PORTD ,PD3

ldi r24 ,250
ldi r25 ,0
rcall wait_usec ; wait for 250 uSec

ldi r24 ,0x30     ; command to switch to 8 bit
out PORTD ,r24
sbi PORTD ,PD3    ; enable pulse
nop
nop
cbi PORTD ,PD3

ldi r24 ,250
ldi r25 ,0
rcall wait_usec ; wait for 250 uSec


ldi r24 ,0x20   ;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
    rcall wait_usec ; wait for 250 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

; I am not sure if this is correct
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
```

Ζήτημα 4.2

Στο ζήτημα 4.2 καλούμαστε να υλοποιήσουμε σε C το πρόγραμμα του 4.1, με την διαφορά ότι εδώ δεν χρησιμοποιούμε την διακοπή του AVR, αλλά ελέγχουμε εμείς ότι έχει ολοκληρωθεί η μετατροπή ADC, διαβάζοντας την τιμή ADSC του καταχωρητή ADCSRA. Έχουμε επίσης φροντίσει να γράψουμε σωστά σε C τις συναρτήσεις που υλοποιούν την επικοινωνία με τον εσωτερικό μικροελεγκτή της οθόνης.

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

uint8_t adc_low_val= 0;
void adc_init(void) {
	// Initialize ADC
	// Set reference voltage to AVcc (with external capacitor at AREF
pin)
	// Set ADC1
	ADMUX = (1 << REFS0) | (1<<MUX0);

	// Set ADC pre-scaler to 128 (for 16 MHz clock, ADC clock = 16MHz
/128 = 125 kHz - within the range)
	// Set ADC enable
	ADCSRA = (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0) | (1<<ADEN);
}

uint16_t adc_read(void) {
	// Set ADSC flag of ADSCRA
	ADCSRA |= (1 << ADSC);

	// Wait for ADSC flag to become '0'
	while (ADCSRA & (1 << ADSC));

	//Return 10-bit result

	return ADC;
}
```

```
void write_2_nibbles (uint8_t input) {
	unsigned char pind = PIND, temp = input & 0xF0;
	pind = pind & 0x0F;
	PORTD = pind + temp;

	PORTD |= (1<<PD3);
	asm("nop");
	asm("nop");
	PORTD &= ~(1 << PD3);

	temp = input & 0x0F;
	temp = temp << 4;
	PORTD = pind + temp;

	PORTD |= (1<<PD3);
	asm("nop");
	asm("nop");
	PORTD &= ~(1 << PD3);

	return;
}

void lcd_data (uint8_t input) {
	PORTD |= (1<<PD2);
	write_2_nibbles(input);
	_delay_ms(1);
	return;
}

void lcd_command (uint8_t input) {
	PORTD &= ~(1 << PD2);
	write_2_nibbles(input);
	_delay_ms(1);
	return;
}

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

void lcd_init () {
	_delay_ms(200);
	PORTD = 0x30;
```

```
	for (int i=0; i<3; i++) {
		PORTD |= (1<<PD3);
		asm("nop");
		asm("nop");
		PORTD &= ~(1 << PD3);
		_delay_ms(1);
	}

	PORTD = 0x20;
	PORTD |= (1<<PD3);
	asm("nop");
	asm("nop");
	PORTD &= ~(1 << PD3);
	_delay_ms(1);
	lcd_command(0x28);
	lcd_command(0x0c);
	lcd_clear_display();
	lcd_command(0x06);
	return;
}

int main() {
	// PORTD as output (to send data and commands to LCD)
	DDRD = 0xFF;
	//PORTC as input (for the correct operation of ADC)
	DDRC = 0x00;

	adc_init();
	lcd_init();
	_delay_ms(100);



	while(1) {
		_delay_ms(1000);
		uint16_t adc_value =0;
		uint8_t first_digit=0, second_digit=0, third_digit =0;
		//int dummy =0;
		lcd_clear_display();
		// Take ADC measurement
		adc_value = (adc_read()*5);

		// Find first digit
		first_digit = (adc_value / 1024);
```

```
            // Find fist digit after the decimal point (aka second
digit)
            adc_value = adc_value % 1024;
            adc_value = adc_value*10;
            second_digit =  adc_value / 1024;

            // Find second digit after the decimal point (aka third
digit)
            adc_value = adc_value % 1024;
            adc_value = adc_value*10;
            third_digit = adc_value / 1024;

            // Print in LCD screen
            lcd_data(first_digit + '0');
            lcd_data('.');
            lcd_data(second_digit+ '0');
            lcd_data(third_digit + '0');
      }

      return 0;
}
```

Ζήτημα 4.3

Στο ζήτημα 4.3 καλούμαστε να υλοποιήσουμε σε C πρόγραμμα, το οποίο διαβάζει ανά 100 ms την τιμή συγκέντρωσης CO στον χώρο, ανάβει τα λαμπάκια PB5-0 σε τιμή ανάλογη της συγκέντρωσης σε CO, και εμφανίζει κατάλληλο μήνυμα στην οθόνη εφόσον η συγκέντρωση ξεπεράσει τα 70 ppm. Για την προσομοίωση του αισθητήρα θα χρησιμοποιήσουμε το ποτενσιόμετρο POT3, το οποίο θα ορίζει μία τάση από 0-5V, την οποία θα διαβάζουμε μέσω του ADC. Συμβουλευόμενοι το datasheet του αισθητήρα και γνωρίζοντας πως Vgas0= 0.1 V, Sensitivity Code = 129 na/ppm και βρίσκοντας την κατάλληλη τιμή TIA GAIN για μονοξείδιο του άνθρακα (100 kV/A), βρίσκουμε πως συγκέντρωση 70 ppm αντιστοιχεί σε τάση περίπου ίση με 1 V. Ενημερώνουμε συνεπώς τον κώδικα μας, ώστε να εμφανίζει στην οθόνη το μήνυμα “GAS DETECTED” εάν ξεπεράσουμε την συγκεκριμένη τιμή, και να εμφανίζει στην οθόνη το μήνυμα “CLEAR” για 1 δευτερόλεπτο εφόσον πέσουμε από τιμή υψηλότερη του 1 V σε τιμή χαμηλότερη.

```
#define F_CPU 16000000UL
#include "avr/io.h"
```

```
#include <util/delay.h>
#include <stdbool.h>


void adc_init(void) {
	// Initialize ADC
	// Set reference voltage to AVcc (with external capacitor at AREF
pin)
	// Set ADC2
	ADMUX = (1 << REFS0) | (1<<MUX1);

	// Set ADC pre-scaler to 128 (for 16 MHz clock, ADC clock = 16MHz
/128 = 125 kHz - within the range)
	// Set ADC enable
	ADCSRA = (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0) | (1<<ADEN);
}

uint16_t adc_read(void) {
	// Set ADSC flag of ADSCRA
	ADCSRA |= (1 << ADSC);

	// Wait for ADSC flag to become '0'
	while (ADCSRA & (1 << ADSC));

	//Return 10-bit result
	return ADC;
}

void write_2_nibbles (uint8_t input) {
	unsigned char pind = PIND, temp = input & 0xF0;
	pind = pind & 0x0F;
	PORTD = pind + temp;

	PORTD |= (1<<PIND3);
	asm("nop");
	asm("nop");
	PORTD &= ~(1 << PIND3);

	temp = input & 0x0F;
	temp = temp << 4;
	PORTD = pind + temp;

	PORTD |= (1<<PIND3);
	asm("nop");
	asm("nop");
```

```
	PORTD &= ~(1 << PIND3);

	return;
}

void lcd_data (uint8_t input) {
	PORTD |= (1<<PIND2);
	write_2_nibbles(input);
	_delay_ms(1);
	return;
}

void lcd_command (uint8_t input) {
	PORTD &= ~(1 << PIND2);
	write_2_nibbles(input);
	_delay_ms(1);
	return;
}

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

void lcd_init () {
	_delay_ms(200);
	PORTD = 0x30;
	for (int i=0; i<3; i++) {
		PORTD |= (1<<PIND3);
		asm("nop");
		asm("nop");
		PORTD &= ~(1 << PIND3);
		_delay_ms(1);
	}

	PORTD = 0x20;
	PORTD |= (1<<PIND3);
	asm("nop");
	asm("nop");
	PORTD &= ~(1 << PIND3);
	_delay_ms(1);
	lcd_command(0x28);
	lcd_command(0x0c);
	lcd_clear_display();
```

```
	lcd_command(0x06);
	return;

}


int main() {
	// PORTD as output (to send data and commands to LCD)
	DDRD = 0xFF;
	//PORTC as input (for the correct operation of ADC)
	DDRC = 0x00;
	//PORTB as output for the LEDs
	DDRB = 0xFF;

	adc_init();
	lcd_init();

	_delay_ms(100);

	uint16_t adc_value =0;
	char clr[] = "CLEAR";
	char gas[] = "GAS DETECTED";
	bool already_on= 0, flag=0;
	int counter =0;

	while(1){
		_delay_ms(100);
		adc_value = adc_read();
		// 205 is the value for Vgas that gives density 70 ppm
		if(adc_value > 205) {  // density > 70 ppm
			flag=1;
			int dummy = adc_value / 16;
			if (already_on--) { // Turn LEDs off
				PORTB = 0x00;

			}
			else {              // Turn LEDs on
				PORTB = dummy;
				already_on++;
			}
			lcd_clear_display();
			for (int i=0; i<12; i++) {
				//int distance = gas[i] - 'A';
				lcd_data(gas[i]);
			}
```

```
            }
            else {  // density <= 70 ppm
                  int dummy = adc_value / 16;
                  PORTB = dummy;
                  if(flag) {
                        flag =0;
                        lcd_clear_display();
                        for (int i=0; i<5; i++) {
                              //int distance = clr[i] - 'A';
                              lcd_data(clr[i]);
                        }
                        counter++;
                  }
                  if(counter <= 10) {
                        counter++;
                  }
                  else{
                        lcd_clear_display();
                        counter =0;
                  }
            }
      }
      return 0;
}
```