Εργαστήριο Μικροϋπολογιστών: 4η Εργαστηριακή Άσκηση (2022)

Ομάδα 64

Ιωάννου Κωνσταντίνος AM:031-19840

Μυτιληναίος Γιώργος AM:031-19841

Ζήτημα 4.1

```
.def ADC_L= r21
.def ADC_H= r22
.def count = r24
 .def VIN = r19

.org 0x00
    jmp reset

.org 0x2A
    jmp Int_ADC


   reset:
    ldi temp ,high(RAMEND)
    out SPH,temp
    ldi temp ,low(RAMEND)
    out SPL,temp

    ldi temp,0xFF
    out DDRB,temp   ; set PORTB as output
    out PORTB,temp  ; off all the leds

    ; here we need to make A2 input ?
    ;REFDSn[1:0]=01 => select Vref = 5 V
    ;ADLAR = 0 => Right adjust the ADC result
    ;Muxn[4:0]= 0010 => select  ADC2 (POT3 => PA2 input)
    ldi temp,0b01000010
    sts ADMUX,temp

    ;ADEN=1 => ADC enable  ,ADSC = 0 ,ADATE = 0
    ;ADIF= 0 => off conversion DC is done
    ;ADIE =1 =>  the ADC interrupt is enabled.
    ;ADPS =111  =>  FADC =16MHz/128 =125KHz
    ldi temp,0b10001111
    sts ADCSRA ,temp
```

```
39
40      Start_conv:
41          lds temp,ADCSRA
42          ori temp,(1<<ADSC) ;Set ADSC flag of ADCSRA
43          sts ADCSRA,temp
44          sei
45
46      loop1:
47          clt
48
49      loop2:
50          subi count,-1
51          out PORTB , count
52          cpi count , 64
53          breq CLEAR
54       //   jmp delay_1
55           brtc loop2  ; jmp to loop2 until a interupt comes
```

a)Assemply

Αρχικά δίνουμε τις κατάλληλες τιμές στους καταχωρητές ADMUX και ADCSRA ,έτσι ώστε να έχουμε κατάλληλη τάση αναφοράς ,να λαμβάνουμε την είσοδο από το POT3(PA2) αλλά και να ενεργοποιήσουμε τις διακοπές του ADC όπως φαίνεται από τα σχόλια του κώδικα. Στην συνέχεια ξεκινάμε την μετατροπή ADC (στην ρουτίνα Start_conv) .Στην loop_1 με εντολή **clt** γίνεται μηδενισμός σημαίας T του καταχωρητή SREG .Στο loop2 έχουμε έναν μετρητή μέχρι το 64 που όταν γεμίσει ξεκινάει πάλι από την αρχή και η έξοδος του φαίνεται στα led του PORTB .Η loop2 είναι ενας κλειστός βρόγχος μέχρι να γίνει interrupt και να πάμε στην ρουτίνα ADC_INT στην οποία διαβάζουμε τις τιμές του ADC και με το **SET** θέτουμε τον T flag ίσων με άσσο.

```
    LCD_screen:
    rcall lcd_init
    ; αρχικοποίηση οθόνης
    ldi r24, low(2)
    ldi r25, high(2)
    ; Αναμονή 2 msec
    rcall wait_msec
    mov r24, VIN //??
    rcall lcd_data
    ; αποστολή ενός byte δεδομένων στον ελεγκτή της οθόνης lcd
    ldi r24, low(2000)
    ldi r25, high(2000)
    ; Αναμονή 2 sec
    rcall wait_msec
    jmp LCD_screen


    CLEAR: ldi count,0
           reti


    //////// Απο εδώ και κάτω όπως μας δόθηκαν στις διαφάνειες.

     .equ PD3=3
     .equ PD2=2
```

Έπειτα μετατρέπουμε την ADC σε τάση όπως μας δίνεται από τύπο για πολλαπλασιασμό εκμεταλλευόμαστε το **lsr** ενώ για διαίρεση την εντολή **lsl .**Προσοχή δεν πρέπει να παραβλέψουμε το γεγονός ότι το αποτέλεσμα είναι 16 bit και χρειάζεται δύο καταχωρητές των 8 bit . Η ρουτίνα LCD_screen παίρνει τα 3 πρώτα ψηφία της τάσης που βρήκαμε και τα εμφανίζει στην οθόνη .Χρησιμοποιούμε τις ρουτίνες που δόθηκαν στις διαφάνειες του μαθήματος (δεν τις παρουσιάζουμε εδώ για λόγους συντομίας) ,δηλαδή τις write_2_nibbles , lcd_command, lcd_init, wait_msec και wait_usec:

```
void LCD_Char (unsigned char char_data) /* LCD data write function */
{
    LCD_Data_Port= char_data;
    LCD_Command_Port |= (1<<RS);    /* RS=1 Data reg. */
    LCD_Command_Port &= ~(1<<RW);   /* RW=0 write operation */
    LCD_Command_Port |= (1<<EN);    /* Enable Pulse */
    _delay_us(1);
    LCD_Command_Port &= ~(1<<EN);
    _delay_ms(1);
}

void LCD_String (char *str)     /* Send string to LCD function */
{
    int i;
    for(i=0;str[i]!=0;i++)      /* Send each char of string till the NULL */
    {
        LCD_Char (str[i]);
    }
}

void LCD_print(float x ) {
int temp;  //βρες τα 3 ψηφία που θέλουμε
temp = 100*x ;
int d , d1,d2,d3;
d1= temp/100 ;
d = temp -d1*100 ;
d2 = d/10 ;
d3 = d -d2*10;

char A[100];
sprintf(A,"%d,%d%d" ,d1,d2,d3);
LCD_String(A);
}
```

```
void LCD_Command( unsigned char cmnd )
{
    LCD_Port = (LCD_Port & 0x0F) | (cmnd & 0xF0);/* Sending upper nibble */
    LCD_Port &= ~ (1<<RS);      /* RS=0, command reg. */
    LCD_Port |= (1<<EN);        /* Enable pulse */
    _delay_us(1);
    LCD_Port &= ~ (1<<EN);
    _delay_us(200);
    LCD_Port = (LCD_Port & 0x0F) | (cmnd << 4);/* Sending lower nibble */
    LCD_Port |= (1<<EN);
    _delay_us(1);
    LCD_Port &= ~ (1<<EN);
    _delay_ms(2);
}

void LCD_Init (void)  /* LCD Initialize function */
{
    LCD_Dir = 0xFF;     /* Make LCD port direction as o/p */
    _delay_ms(20);      /* LCD Power ON delay always >15ms */

    LCD_Command(0x33);
    LCD_Command(0x32);  /* Send for 4 bit initialization of LCD  */
    LCD_Command(0x28);  /* 2 line, 5*7 matrix in 4-bit mode */
    LCD_Command(0x0c);  /* Display on cursor off */
    LCD_Command(0x06);  /* Increment cursor (shift cursor to right) */
    LCD_Command(0x01);  /* Clear display screen */
}
```

b)Σε γλώσσα C

Αρχικά υπενθυμίζουμε ότι σε C , δεν ενεργοποιούμε τις διακοπες αλλά περιμένουμε να τελειώσει η ADC μετατροπή δηλαδή κάνουμε Polling.Οι συναρτήσει LCD INIT,command ,string ,char είναι γενικά τετριμμένες συναρτήσεις των οποία ο σκοπός περιγράφεται στα σχόλια-(μια μετάφραση των συναρτήσεων που μας δόθηκαν σε assemply).Η **συνάρτηση LCD_print**(float) παίρνει την float τιμή της τάσης (που από ADC μετατράπηκε σε τάση) και απομονώνει τα τρία σημαντικότερα ψηφία που θέλουμε .Εκμεταλλευόμαστε ότι η διαίρεση στην C έτσι όπως την εκτελούμε κρατάει μόνο το ακέραιο μέρος και ακολουθούμε μια απλή λογική όπως φαίνεται στον κώδικα. Έπειτα με την Sprintf βάζουμε σε μορφή char τους αριθμούς μας καθώς και την υποδιαστολή και χρησιμοποιούμε την συνάρτηση που δείχνει την lcd οθόνη strings.

```
int ADC_check()
{
    ADCSRA = 0b10000111;
    ADMUX  = 0b01000011;
    int ADSC_check = 1;
    float value;
    while(1)
    {
        if (ADSC_check)
        {
            bit_is_set(ADCSRA, ADSC); // C3:: start converting voltage on A0
            ADSC_check = 0;
        }
        if (bit_is_clear(ADCSRA, ADSC)) //when conv is over
        {
            value = (ADC*5)/1024;
            ADSC_check = 1;
            return value;
        }
    }
    value = (value*5)/(2^(10));
    return value;
}
```

```
main(int argc, char** argv) {

    float ADC_val;

    LCD_Init();

    while(1)
    {

        ADC_val = ADC_check();
        LCD_print(ADC_val) ;

    }

    return (EXIT_SUCCESS);
}
```

Η συνάρτηση **ADC_check()** αφού αρχίσει την μετατροπή ADC δίνοντας τις κατάλληλες τιμές στους ADMUX και ADCRA , χρησιμοποιούμε και το ADSC_check ώστε την πρώτη φορά να αρχίσει η μετατροπή ADC ,ενώ στην συνέχεια όταν το bit ADSC γίνει μηδέν , δηλαδή γνωρίζουμε ότι έχει ολοκληρωθεί η μετατροπή να μετατρέψουμε την ADC σε τάση και η συνάρτηση να επιστρέψει αυτήν την float τιμή της τάσης. Η main στο πρόγραμμα μας αφού αρχικοποιήσεις την LCD οθόνη ενεργοποιεί συνεχώς την μετατροπή όταν αυτή ολοκληρωθεί εκτυπώνει την τιμή και ξανά από αρχή.

Ζήτημα 4.2

```
.def temp = r16
.def ADC_L = r21
.def ADC_H = r22

.org 0x00
    jmp reset
.org 0x2A
    reti

reset:
    ldi temp, low(RAMEND)
    out SPL, temp
    ldi temp, high(RAMEND)
    out SPH, temp
    ser temp
    out DDRB, temp
    clr temp
    out DDRC, temp
    out DDRD, temp
    ldi temp, 0b11000000
    sts ADMUX, temp
    ldi temp, 0b10000111
    sts ADCSRA, temp
    ldi zh, high(Table)
    ldi zl, low(Table)

setup_LCD:
    rcall lcd_init
    ldi r24, low(2)
    ldi r25, high(2)
    rcall wait_msec


 ////////////////////////////ADC_CHECK
 main:
     lds temp, ADCSRA
     ori temp, (1<<ADSC)
     sts ADCSRA, temp
     wait_adc:
     lds temp, ADCSRA
     sbrc temp,ADSC
     rjmp wait_adc
     lds ADC_L,ADCL
     lds ADC_H,ADCH
     cpi ADC_H, 5
     brsh warning
     rcall clear
     ldi r24, low(100)
     ldi r25, high(100)
     rcall wait_msec
     rjmp main
 ////////////////////////////

 ////////////////////////////WARNING_SETUP
 warning:
     rcall LCD_warning
     cpi ADC_H, 255
     brsh five
     cpi ADC_H, 204
     brsh four
     cpi ADC_H, 153
     brsh three
     cpi ADC_H, 102
     brsh two
     cpi ADC_H, 51
     brsh one
     ldi temp, 0b00000001
     out PORTB, temp
 exit_blink:
 ret
 ////////////////////////////
```

a)Assemply

Αρχικά δεν παρουσιάζουμε τις τετριμμένες συναρτήσεις που υπάρχουν στις διαφάνειες για λόγους συντομίας. Δίνουμε κατάλληλες τιμές στους ADMUX και ADCRA και αρχικοποιούμε την lcd οθόνη . Στην main ξεκινάμε την μετατροπή ,διαβάζουμε τις τιμές ADC και περιμένουμε μέχρι να γίνει κάποια διακοπή. Έπειτα εξετάζουμε την τιμή ADC για να ελέγξουμε τα στάδια του “CO” και να ανάψουμε τα ανάλογα leds .H blink () προφανώς αναβοσβήνει όσα leds της PORTB αναλογούν στο επίπεδο του “CO” για μερικά sec ώστε να γίνουν αντιληπτά. Τέλος η CLEAR καθαρίζει την οθόνη LCD .

```
//////////////////////////////WARNING_LCD
LCD_warning:
    cpi r24, 0x47
    breq exit_warning
    ldi r24, 1
    rcall lcd_command
    ldi zh, high(Table)
    ldi zl, low(Table)
    adiw zl, 12
    lpm r24, z
    loop_warning:
    cpi r24, 0x47
    breq exit_warning
    lpm r24, z
    rcall lcd_data
    sbiw zl, 1
    rjmp loop_warning
exit_warning:
ret
//////////////////////////////

//////////////////////////////WARNING_BLINK
blink:
    out PORTB, temp
    ldi r24, low(10)
    ldi r25, high(10)
    rcall wait_msec
    clr temp
    out PORTB, temp
    ldi r24, low(10)
    ldi r25, high(10)
    rcall wait_msec
    ret
//////////////////////////////

//////////////////////////////WARNING_BLINK_LEVEL
five:
    ldi temp, 0b00111111
    rcall blink
    rjmp exit_warning

four:
    ldi temp, 0b00011111
    rcall blink
    rjmp exit_warning

three:
    ldi temp, 0b00001111
    rcall blink
    rjmp exit_warning

two:
    ldi temp, 0b00000111
    rcall blink
    rjmp exit_warning

one:
    ldi temp, 0b00000011
    rcall blink
    rjmp exit_warning
//////////////////////////////

//////////////////////////////CLEAR
clear:
    lpm r24, z
    cpi r24, 0x43
    breq exit_clear
    ldi r24, 1
    rcall lcd_command
    ldi zh, high(Table)
    ldi zl, low(Table)
    adiw zl, 17
    lpm r24, z
loop_clear:
    cpi r24, 0x43
    breq exit_clear
    lpm r24, z
    rcall lcd_data
    sbiw zl, 1
    rjmp loop_clear
exit_clear:
ret
```

```c
#define En 1 //enable
#define Rs 0

int ADSC_check;

int LCD_init()
{...11 lines }

void blink(int x)
{...44 lines }

int ADC_check()
{...22 lines }

void Pulse()
{
    bit_is_set(PORTB,En); // take LCD enable line high
    _delay_us(40); // wait 40 microseconds
    bit_is_clear(PORTB,En); // take LCD enable line low
}

int write_2_nibbles(uint8_t data)
{...9 lines }

int send_2_nibbles(uint8_t data)
{...6 lines }

void command (uint8_t cmd)
{...4 lines }
void write_2_lcd (uint8_t x)
{
    bit_is_set(PORTB,Rs);
    send_2_nibbles(x);
}



void display_string(const char *txt) // display string on LCD
{

   while (*txt)
   {
       write_2_lcd(*txt++);
   }

}
void Clear() // clear the LCD display
{
    command(0x01);
    _delay_ms(5);
}

gas_level(int ADC_val){

    if(ADC_val < 0.1)
    {
        display_string("CLEAR");
        blink(0);
    }

    display_string("GAS DETECTED");

    if(ADC_val < 1)
    {

        blink(1);
    }

    if(ADC_val < 2)
    {
        blink(2);
    }

    if(ADC_val < 3)
    {
        blink(3);
    }

    if(ADC_val < 4)
    {
        blink(4);
    }

    if(ADC_val >= 5)
    {
        blink(5);
    }
```

```c
int main(int argc, char** argv) {
    DDRB = 0xFF;
    DDRC = 0x00;
    DDRD = 0x00;
    PORTB = 1;
    int ADC_val;
    LCD_init();
    while(1)
    {

        ADC_val = ADC_check();
        gas_level(ADC_val);

    }

    return (EXIT_SUCCESS);
}
```

b) Σε γλώσσα C

Στην ίδια λογική με την assemply και χρησιμοποιώντας τις “κλασικές» συναρτήσεις για αρχικοποίηση και εκτύπωση στην lcd οθόνη ,αξίζει μόνο να αναφέρουμε την display_string η οποία εκτυπώνει στην οθόνη το κατάλληλο μήνυμα αν εντοπίζεται GAS ή όχι .Προφανώς στην c με την συνάρτηση gas_level το πρόγραμμα γίνεται πιο εύκολα κατανοητό καθώς όλες εκδοχές εμφανίζονται μεσα στην συνάρτηση.

Ζήτημα 4.3

```
int LCD_init()
{...11 lines }
int ADC_check()
{...19 lines }

void Pulse()
{...5 lines }

void cursor_pos(uint8_t x, uint8_t y)
{
 uint8_t addr = 0;
 switch (y)
 {
 case 1: addr = 0x40; break;
 case 2: addr = 0x14; break;
 //case 3: addr = 0x54; break;
 }
 command(Set_Cursor+addr+x);
}

void select_line(uint8_t row)
{
 cursor_pos(0,row);
}


int write_2_nibbles(uint8_t data)
{...9 lines }


int send_2_nibbles(uint8_t data)
{
    write_2_nibbles(data);   // send upper 4 bits
    write_2_nibbles(data<<4); // send lower 4 bits
    bit_is_clear(PORTB,5);
    return 0;
}

void command (uint8_t cmd)
{...4 lines }
void write_2_lcd (uint8_t x)
{...4 lines }

void display_string(const char *txt) // display string on LCD
{...8 lines }
void Clear() // clear the LCD display
{...4 lines }


int main(int argc, char** argv)
{
    int duty[5] = {51, 102, 153, 204, 0};
    TCCR1A = (1<<WGM10) | (1<<COM1A1);
    TCCR1B = (1<<WGM12) | (0<<CS12) | (1<<CS11) | (0<<CS10);
    DDRB = 0b00000011;
    DDRD = 0b0000000;
    DDRC = 0b0000000;
    ICR1 = 399;
    const char buffer[sizeof(char)*4];
    int i, x, ADC_val;
    LCD_init();
    while(1)
    {
        switch(PINB)
        {

            case 2: i = 0;
            Clear();
            display_string("20%");
            break;

            case 4: i = 1;
            Clear();
            display_string("40%");
            break;

            case 8: i = 2;
            Clear();
            display_string("60%");
            break;

            case 16: i = 3;
            Clear();
            display_string("80%");
            break;

            default:
            Clear();
            i = 4;
            break;

        }

        OCR1A = duty[i];
        select_line(2);
        ADC_val = ADC_check();
        itoa(ADC_val, buffer, 10);
        display_string(buffer);
    }
    return (EXIT_SUCCESS);
}
```

Σε αυτό το ζήτημα εκμεταλλευόμαστε όλα τα παραπάνω προγράμματα καθώς και το ζήτημα 3 της 3^{ης} σειράς ασκήσεων. Αποκρύπτουμε τις συναρτήσεις που δείξαμε σε προηγούμενα ζητήματα για πρακτικούς λόγους .Επομένως αρκεί να δείξουμε τη συνάρτηση main του προγράμματος που απλώς ανάλογα με το μπουτόν της PORTB που είναι πατημένο δείχνει στην οθόνη στην πρώτη γραμμή το dc και στη δεύτερη γραμμή την τάση .