ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ – 7ο ΕΞΑΜΗΝΟ

Δημητρίου Αγγελική – ΑΜ: 03117106

Τζομάκα Αφροδίτη – ΑΜ: 03117107

Ομάδα Β8

4η Εργαστηριακή Άσκηση

ΖΗΤΗΜΑ 4.1

Στην υλοποίηση αυτή έχουμε την ανάγνωση των δεδομένων του ADC στην ρουτίνα εξυπηρέτησής του.

Αφού συμβούν οι απαραίτητες αρχικοποιήσεις για τη στοίβα, τον χρονιστή και τον μετατροπέα ADC, τίθενται οι γενικές διακοπές και το πρόγραμμα ακολουθεί δύο μονοπάτια. Το πρώτο είναι εκείνο της main συνάρτησης η οποία είναι υπεύθυνη για τον έλεγχο πατήματος κουμπιού από το πληκτρολόγιο καθώς και των συναρτήσεων alarm_on και welcome οι οποίες χειρίζονται τις περιπτώσεις λάθος και σωστού κωδικού αντίστοιχα. Το δεύτερο αποτελείται από τις ρουτίνες εξυπηρέτησης των διακοπών του Timer και του ADC. Πιο συγκεκριμένα, αρχικά σε κάθε διακοπή του timer ξεκινάμε μια μετατροπή ADC η οποία μόλις ολοκληρωθεί καλεί την αντίστοιχη ρουτίνα εξυπηρέτησής της. Εκεί προσδιορίζεται ανάλογα με την τιμή του ADC το επίπεδο της συγκέντρωσης για τα LED και καλούνται σύμφωνα με τις συμβάσεις της άσκησης τα μηνύματα CLEAR ή GAS DETECTED!. Σημειώνεται ότι αναφορικά με το επίπεδο στα LED για την περίπτωση του CLEAR η έξοδος δίνεται μέσα στην ρουτίνα εξυπηρέτησης του ADC ενώ στην περίπτωση GAS DETECTED! η έξοδος στα LED υλοποιείται στον χρονιστή καθώς προκειμένου να πετύχουμε το αναβόσβημα θέλουμε να εκμεταλλευτούμε την ιδιότητα του να μετράει χρόνο.

```
.DSEG
 _tmp_: .byte 2

.CSEG
.include "m16def.inc"
.org  0x0
  rjmp   reset
.org 0x10
  rjmp ISR_TIMER1_OVF
.org 0x1C
  rjmp ADC_ISR
```

```
reset:
  cli
;----Initialize stack pointer----
  ldi r24,low(RAMEND)
  out SPL,r24
  ldi r24,high(RAMEND)
  out SPH,r24
;----Initialize Timer----
  ldi r24, (1<<TOIE1)                      ;Overflow interrupt enable for timer1
  out TIMSK, r24
  ldi r24, (1<<CS12)|(0<<CS11)|(1<<CS10)  ;Scaling Ck/1024
  out TCCR1B, r24
  ldi r24, 0xFC                            ;Initialize TCNT1 overflow after 100msecs
  out TCNT1H, r24
  ldi r24, 0xF3
  out TCNT1L, r24
;----Initialize LEDs & LCD----
  ser r24
  out DDRD, r24                 ;Set PORTD as output for LCD
  out DDRB, r24                 ;Set PORTB as output for LEDS
  clr r24
  ldi r24,(1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4)
  out DDRC,r24                  ;Set the 4 MSBs as output for keyboard
  clr r24
  rcall lcd_init_sim

  clr r31                   ;Holds the level
  clr r30                   ;LSB Flag for blinking, 2nd LSB previous state of alarm,
                            ;3rd LSB for coming from welcome
  clr r29                   ;Flag for coming from clear(0) or gasd(1)
  clr r21                   ;Counter for time blinking
  rcall ADC_init
  ;rcall clear_msg
  sei

;####### MAIN ########
main:
  rcall scan_keypad_rising_edge_sim      ;Read the first num
  rcall keypad_to_ascii_sim
  cpi   r24,0x00                         ;If 0 is returned then nothing was given
  breq  main                             ;Check again
  mov   r16,r24                          ;Store the first number at r16
read_second:
  rcall scan_keypad_rising_edge_sim      ;Read the second number
  rcall keypad_to_ascii_sim              ;Same logic as the first
  cpi   r24,0x00
  breq  read_second
  mov   r17,r24                          ;Store the second number at r17

  cpi   r16,0x30            ;Check the 1st number (hex code for ascii '0' is 30)
  breq  check1
wrong:
  rcall alarm_on
  jmp   end
check1:
  cpi   r17,0x38            ;Check the 2nd number
  breq  success
  jmp   wrong
success:
  rcall welcome
end:
  jmp main
```

```
;##### WRONG PSW #####
alarm_on:
  ldi r19,4                              ;Initialize counter for blinking at r19
rep:
;------- ON -------
  sbr r30, 0x02                          ;Alarm - MSB ON
  sbr r31, 0x80
  out PORTB, r31
  ldi r24,low(500)
  ldi r25,high(500)
  rcall wait_msec                        ;PB0:6 & PB7 ON for 0.5msecs
  rcall scan_keypad_rising_edge_sim      ;Needed for remote access
;------- OFF -------
;alarm_clear:
  cbr r30, 0x02                           ;Alarm - MSB OFF
  cbr r31, 0x80
  out PORTB,r31
  ldi r24,low(500)
  ldi r25,high(500)
  rcall wait_msec
  rcall scan_keypad_rising_edge_sim      ;Needed for remote access
;--------------------
  dec r19
  brne rep
  cbr r31, 0x80
  out PORTB,r31                          ;Turn PB at its level and return
 ret

;##### RIGHT PSW #####
 welcome:
  cli
  rcall welcome_msg
  sbr r30, 0x04             ;We are coming from Welcome
  cpi r29, 0x00             ;Is our state Clear?
  breq welcome_clear
  ldi r29, 0x00
  ldi r18, 0x80             ;Else we show only the welcome led
  out PORTB, r18
  rjmp endw
welcome_clear:              ;If we are in CLEAR state we want to preserve the level
  ldi r29, 0x01             ;For the clear message to be triggered again
  sbr r31, 0x80             ;So we change PB7 on r31
  out PORTB, r31
endw:
  ldi r24,low(4000)
  ldi r25,high(4000)
  rcall wait_msec                        ;PB7 ON for 4secs
  rcall scan_keypad_rising_edge_sim
  ldi r24, 0x01
  rcall lcd_command_sim
  ldi r24, low(1530)
  ldi r25, high(1530)
  rcall wait_usec
  cbr r31, 0x80
  out PORTB,r31                          ;Turn PB on its level & lcd OFF and return
  sei
 ret
```

```
;###### TIMER SERVICE ROUTINE ######
ISR_TIMER1_OVF:
  ldi r20, 0xFC                    ;Initialize TCNT1 for overflow after 100msecs
  out TCNT1H, r20
  ldi r20, 0xF3
  out TCNT1L, r20
  ldi r20, (1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADSC)
;Start conversion
  out ADCSRA, r20
  clr r20

  inc r21
  cpi r21, 0x05                    ;500msec have passed
  brne endt
blinkt:
  clr r21                          ;Clear counter to start counting again 500msec
  cpi r29, 0x00                    ;If we came from CLEAR then do nothing and return
  breq endt
  sbrs r30, 0               ;If previous state was ON
  rjmp blon                 ;If previous state was OFF
  out PORTB, r21            ;PB0:6 OFF
  ldi r30, 0x00
  rjmp endt
blon:
  ;rcall welcome_msg - WARNING! ADVANCED DEBUGGING!
  out PORTB, r31            ;PB0:6 ON
  ldi r30, 0x01
endt:
reti

;###### ADC SERVICE ROUTINE ######
ADC_ISR:
  push r19
  push r17
  push r18
  push r24
  push r25
  in r17, ADCL                     ;Read the output of the ADC
  in r18, ADCH                     ;First we MUST read the ADCL
  mov r19, r17
  lsl r17
  rol r18
  andi r18, 0x07
;----LEVEL CHECKING----
  ldi r31, 0x01
  cpi r18, 0x01
  brlo clear
  ldi r31, 0x03
  cpi r18, 0x02
  brlo clear
  ldi r31, 0x07
  cpi r18, 0x03
  brlo gasd
  ldi r31, 0x0F
  cpi r18, 0x04
  brlo gasd
  ldi r31, 0x1F
  cpi r19, 0x00
  brlo gasd
  ldi r31, 0x3F
  cpi r19, 0x67
  brlo gasd
```

```
  ldi r31, 0x7F
  rjmp gasd
;---End of Level Checking---
clear:
  sbrc r30, 1              ;Check for alarm
  sbr r31, 0x80
  out PORTB, r31            ;SHOW THE LEVEL
  sbrc r30, 2              ;Check for welcome
  rjmp showc
  cpi r29, 0x00            ;Check for clear
  breq endi
showc:
  rcall clear_msg
  ldi r29, 0x00             ;DO THE STATE CLEAR (r29 = 0)
  rjmp endi
gasd:
  ;out PORTB, r31 – WARNING ADVANCED DEBUGGING
  sbrc r30, 2
  rjmp showg
  cpi r29, 0x01
  breq endi
showg:
  rcall gasd_msg
  ldi r29, 0x01             ;STATE WAS GAS DETECTED SO BEGIN BLINKING IN TIMER
endi:
  cbr r30, 0x04             ;Welcome state cleared
  pop r25
  pop r24
  pop r18
  pop r17
  pop r19
reti


ADC_init:
  push r24
  ldi r24, (1<<REFS0)                    ;Vref = Vcc
  out ADMUX, r24                         ;MUX4:0 = 00000 for A0
  ;ADC Enabled, ADC Interrupts Enabled, Set Prescaler CLK/128 = 62.5KHz
  ldi r24, (1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  out ADCSRA, r24
  pop r24
  ret


welcome_msg:
  push r24
  ldi r24, 0x01
  rcall lcd_command_sim
  ldi r24, low(1530)
  ldi r25, high(1530)
  rcall wait_usec
  ldi r24,'W'
  rcall lcd_data_sim
  ldi r24,'E'
  rcall lcd_data_sim
  ldi r24,'L'
  rcall lcd_data_sim
  ldi r24,'C'
  rcall lcd_data_sim
  ldi r24,'O'
  rcall lcd_data_sim
```

```
  ldi r24,'M'
  rcall lcd_data_sim
  ldi r24,'E'
  rcall lcd_data_sim
  ldi r24,' '
  rcall lcd_data_sim
  ldi r24,'B'
  rcall lcd_data_sim
  ldi r24,'8'
  rcall lcd_data_sim
  pop r24
ret

gasd_msg:
  push r24
  ldi r24, 0x01
  rcall lcd_command_sim
  ldi r24, low(1530)
  ldi r25, high(1530)
  rcall wait_usec
  ldi r24,'G'
  rcall lcd_data_sim
  ldi r24,'A'
  rcall lcd_data_sim
  ldi r24,'S'
  rcall lcd_data_sim
  ldi r24,' '
  rcall lcd_data_sim
  ldi r24,'D'
  rcall lcd_data_sim
  ldi r24,'E'
  rcall lcd_data_sim
  ldi r24,'T'
  rcall lcd_data_sim
  ldi r24,'E'
  rcall lcd_data_sim
  ldi r24,'C'
  rcall lcd_data_sim
  ldi r24,'T'
  rcall lcd_data_sim
  ldi r24,'E'
  rcall lcd_data_sim
  ldi r24,'D'
  rcall lcd_data_sim
  ldi r24,'!'
  rcall lcd_data_sim
  pop r24
reti

clear_msg:
  push r24
  ldi r24, 0x01
  rcall lcd_command_sim
  ldi r24, low(1530)
  ldi r25, high(1530)
  rcall wait_usec
  ldi r24,'C'
  rcall lcd_data_sim
  ldi r24,'L'
  rcall lcd_data_sim
  ldi r24,'E'
  rcall lcd_data_sim
```

```
  ldi r24,'A'
  rcall lcd_data_sim
  ldi r24,'R'
  rcall lcd_data_sim
  pop r24
reti

scan_keypad_rising_edge_sim:
    push r22
       push r23
       push r26
       push r27
       rcall scan_keypad_sim
       push r24
       push r25
       ldi r24, 15
       ldi r25, 0
       rcall wait_msec
       rcall scan_keypad_sim
       pop r23
       pop r22
       and r24 ,r22
       and r25 ,r23
       ldi r26 ,low(_tmp_)
       ldi r27 ,high(_tmp_)
       ld r23 ,X+
       ld r22 ,X
       st X ,r24
       st -X ,r25
       com r23
       com r22
       and r24 ,r22
       and r25 ,r23

       pop r27
       pop r26
       pop r23
       pop r22
       ret

scan_row_sim:
       out PORTC , r25
       ;remote access cseg
       push r24
       push r25
       ldi r24, low(500)
       ldi r25, high(500)
       rcall wait_usec
       pop r25
       pop r24
       ;end remote access cseg
       nop
       nop
       in r24 , PINC
       andi r24 ,0x0f
       ret

scan_keypad_sim:
       push r26
       push r27
       ldi r25 , 0x10
       rcall scan_row_sim
```

```
        swap r24
        mov r27, r24
        ldi r25 ,0x20
        rcall scan_row_sim
        add r27, r24
        ldi r25 , 0x40
        rcall scan_row_sim
        swap r24
        mov r26, r24
        ldi r25 ,0x80
        rcall scan_row_sim
        add r26, r24
        movw r24, r26
        clr r26
        out    PORTC,r26
        pop r27
        pop r26
        ret


keypad_to_ascii_sim:
        push r26
        push r27
        movw r26 ,r24
        ldi r24 ,'*'
        sbrc r26 ,0
        rjmp return_ascii
        ldi r24 ,'0'
        sbrc r26 ,1
        rjmp return_ascii
        ldi r24 ,'#'
        sbrc r26 ,2
        rjmp return_ascii
        ldi r24 ,'D'
        sbrc r26 ,3
        rjmp return_ascii
        ldi r24 ,'7'
        sbrc r26 ,4
        rjmp return_ascii
        ldi r24 ,'8'
        sbrc r26 ,5
        rjmp return_ascii
        ldi r24 ,'9'
        sbrc r26 ,6
        rjmp return_ascii ;
        ldi r24 ,'C'
        sbrc r26 ,7
        rjmp return_ascii
        ldi r24 ,'4'
        sbrc r27 ,0
        rjmp return_ascii
        ldi r24 ,'5'
        sbrc r27 ,1
        rjmp return_ascii
        ldi r24 ,'6'
        sbrc r27 ,2
        rjmp return_ascii
        ldi r24 ,'B'
        sbrc r27 ,3
        rjmp return_ascii
        ldi r24 ,'1'
        sbrc r27 ,4
```

```
        rjmp return_ascii ;
        ldi r24 ,'2'
        sbrc r27 ,5
        rjmp return_ascii
        ldi r24 ,'3'
        sbrc r27 ,6
        rjmp return_ascii
        ldi r24 ,'A'
        sbrc r27 ,7
        rjmp return_ascii
        clr r24
        rjmp return_ascii
return_ascii:
        pop r27
        pop r26 ret

write_2_nibbles:
        ;remote access cseg
        push r24
        push r25
        ldi r24, low(6000)
        ldi r25, high(6000)
        rcall wait_usec
        pop r25
        pop r24
        ;end remote access cseg
        push r24
        in r25 ,PIND
        andi r25 ,0x0f
        andi r24 ,0xf0
        add r24 ,r25
        out PORTD ,r24
        sbi PORTD ,PD3
        cbi PORTD ,PD3
        ;remote access cseg
        push r24
        push r25
        ldi r24, low(6000)
        ldi r25, high(6000)
        rcall wait_usec
        pop r25
        pop r24
        ;end remote access cseg
        pop r24
        swap r24
        andi r24 ,0xf0
        add r24 ,r25
        out PORTD ,r24
        sbi PORTD ,PD3
        cbi PORTD ,PD3
        ret

lcd_data_sim:
        push r24
        push r25
        sbi PORTD ,PD2
        rcall write_2_nibbles
        ldi r24 ,43
        ldi r25 ,0
        rcall wait_usec
        pop r25
        pop r24
```

```
        ret

lcd_command_sim:
        push r24
        push r25
        cbi PORTD, PD2
        rcall write_2_nibbles
        ldi r24, 39
        ldi r25, 0
        rcall wait_usec
        pop r25
        pop r24
        ret

lcd_init_sim:
        push r24
        push r25
        ldi r24 ,40
        ldi r25 ,0
        rcall wait_msec

        ldi r24 ,0x30
        out PORTD ,r24
        sbi PORTD ,PD3
        cbi PORTD ,PD3
        ldi r24 ,39
        ldi r25 ,0
        rcall wait_usec

        ;remote access cseg
        push r24
        push r25
        ldi r24, low(1000)
        ldi r25, high(1000)
        rcall wait_usec
        pop r25
        pop r24
        ;end remote access cseg
        ldi r24 ,0x30
        out PORTD ,r24
        sbi PORTD ,PD3
        cbi PORTD ,PD3
        ldi r24 ,39
        ldi r25 ,0
        rcall wait_usec
        ;remote access cseg
        push r24
        push r25
        ldi r24, low(1000)
        ldi r25, high(1000)
        rcall wait_usec
        pop r25
        pop r24
        ;end remote access cseg

        ldi r24 ,0x20
        out PORTD ,r24
        sbi PORTD ,PD3
        cbi PORTD ,PD3
        ldi r24 ,39
        ldi r25 ,0
        rcall wait_usec
```

```
        ;remote access cseg
        push r24
        push r25
        ldi r24, low(1000)
        ldi r25, high(1000)
        rcall wait_usec
        pop r25
        pop r24
        ;end remote access cseg

        ldi r24 ,0x28
        rcall lcd_command_sim
        ldi r24 ,0x0c
        rcall lcd_command_sim
        ldi r24 ,0x01
        rcall lcd_command_sim
        ldi r24 ,low(1530)
        ldi r25 ,high(1530)
        rcall wait_usec

        ldi r24 ,0x06
        rcall lcd_command_sim

        pop r25
        pop r24
        ret

wait_msec:
        push r24
        push r25
        ldi r24 , low(998)
        ldi r25 , high(998)
        rcall wait_usec
        pop r25
        pop r24
        sbiw r24 , 1
        brne wait_msec
        ret
wait_usec:
        sbiw r24 ,1
        nop
        nop
        nop
        nop
        nop
        brne wait_usec
        ret
```

ΖΗΤΗΜΑ 4.2

Στην υλοποίηση αυτή έχουμε την ανάγνωση των δεδομένων του ADC στην ρουτίνα εξυπηρέτησής του timer με τη μέθοδο polling.

Ξεκινάμε αρχικοποιώντας όπως και πριν χωρίς όμως να ενεργοποιούμε τις διακοπές του ADC. Το πρώτο κομμάτι που αφορά τη main συνάρτηση και το διάβασμα από το πληκτρολόγιο παραμένει στην ίδια λογική με του προηγούμενου ζητήματος την υλοποίηση. Ωστόσο στο δεύτερο κομμάτι, εξαιτίας της μεθόδου polling για το διάβασμα από τον ADC, δεν υπάρχει η ρουτίνα εξυπηρέτησης για αυτόν και όλες οι διαδικασίες εύρεσης επιπέδων, ανάμματος των LED και τυπώματος στην οθόνη γίνονται στον timer. Σημειώνεται ότι σε αυτήν την υλοποίηση, δεν απενεργοποιούνται οι διακοπές όσο διαχειριζόμαστε την περίπτωση WELCOME και ελέγχεται και αυτή μέσα στον timer με την χρήση κατάλληλων flags.

```
#define       F_CPU   8000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

int _tmp_ = 0x0000;
int flag=0;
int level=0x00;
int in_welcome,out_welcome=0;
int counter=0;

/*keyboard routines*/
int scan_row_sim(char row_no)
{
       PORTC = (0x08<<row_no);    //PC4-PC7
       _delay_us(500);            //wait 500 usec for remote access

       asm volatile("nop");       //compiler shouldn't move these nops they're crucial
       asm volatile("nop");
       return (PINC&0x0F);        //4 switches in LSBs
}

int scan_keypad_sim()
{
       int keys_pressed = scan_row_sim(4) | (scan_row_sim(3)<<4) |
(scan_row_sim(2)<<8) | (scan_row_sim(1)<<12);
//for rows 1(PC4) to 4(PC7) keep all switches on
       PORTC=0x00;              //for remote access
       return keys_pressed;
}

int scan_keypad_rising_edge_sim()
{
       int scan1 = scan_keypad_sim();
       _delay_ms(15);
       int scan2 = scan_keypad_sim();
       int result = scan1 & scan2;           //scan twice with 10-20 ms in between
                                             //to eliminate debounce flaws
       int final_result = result & ~(_tmp_);   //get different switches pressed
                                                //comparing to old state
       _tmp_=result;                            //update state of switches
```

```
	return final_result;
}

char keypad_to_ascii(int pressed_key)
{
	if((pressed_key&0x0001)==0x0001) return '*';
	if((pressed_key&0x0002)==0x0002) return '0';
	if((pressed_key&0x0004)==0x0004) return '#';
	if((pressed_key&0x0008)==0x0008) return 'D';
	if((pressed_key&0x0010)==0x0010) return '7';
	if((pressed_key&0x0020)==0x0020) return '8';
	if((pressed_key&0x0040)==0x0040) return '9';
	if((pressed_key&0x0080)==0x0080) return 'C';
	if((pressed_key&0x0100)==0x0100) return '4';
	if((pressed_key&0x0200)==0x0200) return '5';
	if((pressed_key&0x0400)==0x0400) return '6';
	if((pressed_key&0x0800)==0x0800) return 'B';
	if((pressed_key&0x1000)==0x1000) return '1';
	if((pressed_key&0x2000)==0x2000) return '2';
	if((pressed_key&0x4000)==0x4000) return '3';
	if((pressed_key&0x8000)==0x8000) return 'A';
	return 0x00;
}

/*lcd routines*/

void write_2_nibbles_sim(char input)
{

	_delay_us(6000);                        //remote access
	char prev = PIND;
	PORTD = (input&0xf0) | (prev&0x0f);     //keep previous state of D (LSBs)
                                            //and resend
                                        // Get the 4 msb of input and send to PORTD
	PORTD |= (1<<PD3);                  //enable pulse
	PORTD &= ~(1<<PD3);
	_delay_us(6000);                        //remote access
	PORTD = ((input&0x0f)<<4) | (prev&0x0f);//Get the 4 lsb of input
                                            //and send to PORTD as msb


	PORTD |= (1<<PD3);                      //enable pulse
	PORTD &= ~(1<<PD3);

	return;
}

void lcd_data_sim(unsigned char input)
{
	PORTD = (1 << PD2 );        //PD2=1 for data
	write_2_nibbles_sim(input);
	_delay_us(43);              //data delay
	return ;
}

void lcd_command_sim(unsigned char command)
{
	PORTD = (0 << PD2);            //PD2=0 for commands
	write_2_nibbles_sim(command);
	_delay_us(39);                 //general command delay
	return ;
}
```

```
void lcd_init_sim()
{
                                    //initialize using required commands
       _delay_ms(40);               //wait for controller

       PORTD = 0x30;
       PORTD |=(1 << PD3);
       PORTD &=~(1 << PD3);
       _delay_us(39);

       _delay_us(1000);             //remote access
       PORTD = 0x30;
       PORTD |= (1 << PD3);
       PORTD &=~ (1 << PD3);
       _delay_us(39);

       _delay_us(1000);           //remote access
       PORTD = 0x20;
       PORTD |= (1 << PD3);
       PORTD &=~ (1 << PD3);
       _delay_us(39);

       _delay_us(1000);           //remote access
       lcd_command_sim(0x28);
       lcd_command_sim(0x0c);
       lcd_command_sim(0x01);
       _delay_us(1530);

       lcd_command_sim(0x06);
       return ;
}

void ADC_init()
{
       ADMUX = 1<<REFS0;                                    //pick A0
       ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADSC);
//enable ADC, set prescaler to 128 (don't enable interrupts for ADC)
//set ADSC to start conversion
       return;
}

ISR (TIMER1_OVF_vect)    // Timer1 ISR
{
       int ADC_val;      //value read from ADC
       int new_flag;     //flag for new state to be determined (clear - gas)

    if((ADCSRA & 0x40) == 0x00)  //if ADSC is 0 conversion is done and we can read the
measurement
       {

              ADC_val = ADCL | (ADCH << 8);   //read ADC value
              if(ADC_val < 103){              //find correct level and if state will
be clear or gas
                     level=0x01;
                     new_flag=0;
              }
              else if(ADC_val < 206){
                     level=0x03;
                     new_flag=0;
              }
              else if(ADC_val < 308){
                     level=0x07;
```

```
                        new_flag=1;
                }
                else if(ADC_val < 411){
                        level=0x0F;
                        new_flag=1;
                }
                else if(ADC_val < 513){
                        level=0x1F;
                        new_flag=1;
                }
                else if(ADC_val < 615){
                        level=0x3F;
                        new_flag=1;
                }
                else {
                        level=0x7F;
                        new_flag=1;
                }

/*FIX LCD*/
                if(new_flag == 1 ) {                        //case gas detected
//only if we came from clear or welcome we must write GAS DETECTED message on screen
                        if(flag==0 || out_welcome==1){
                              lcd_command_sim(0x01);
                               _delay_us(1530);
                               lcd_data_sim('G');
                               lcd_data_sim('A');
                               lcd_data_sim('S');
                               lcd_data_sim(' ');
                               lcd_data_sim('D');
                               lcd_data_sim('E');
                               lcd_data_sim('T');
                               lcd_data_sim('E');
                               lcd_data_sim('C');
                               lcd_data_sim('T');
                               lcd_data_sim('E');
                               lcd_data_sim('D');
                               lcd_data_sim('!');
                               out_welcome=0;
                               in_welcome=0;
                        }
                }
                else {                                      //case clear
                        if(flag==1 || out_welcome == 1){    //same logic as before
                               lcd_command_sim(0x01);
                               _delay_us(1530);
                               lcd_data_sim('C');
                               lcd_data_sim('L');
                               lcd_data_sim('E');
                               lcd_data_sim('A');
                               lcd_data_sim('R');
                               out_welcome=0;
                               in_welcome=0;
                        }
                }

/*FIX LEDS*/
                if(new_flag==0){                            //case clear
                               PORTB = (PORTB & 0x80)|level;  //new led value, changing
only 6 LSBs
                               counter = 0;
                }
```

```
		else{                                           //case gas
			if(counter < 5){                        //counter<5 -> LEDs ON
				if(in_welcome == 0){       //don't do this when IN welcome
				PORTB = (PORTB & 0x80)|level;  //new led value
				counter++;
				}
				else{
					PORTB =0x80;           //IN welcome, only light MSB
				}
			}
			else if(counter < 10){                  //counter>5 -> LEDs OFF
				if(in_welcome == 0){            //same as before
				PORTB = PORTB & 0x80;           //don't show level
				counter++;
				}
				else{
					PORTB=0x80;
				}
			}
			if(counter == 10)
			counter=0;
		}


		flag=new_flag;
	}

	TCNT1 = 64755;
	ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADSC);
//enable ADC, set prescaler to 128 (don't enable interrupts for ADC)
}

int main(void)
{
	char number1,number2;
	cli();
	DDRC=0xF0;                      //PC4-PC7 -row detection are outputs and
                                        //PC3-PC0 inputs -column detection (for keypad)
	DDRB=0xFF;                      //PORTB is output
	DDRD=0xFF;                      //PORTD output for lcd
	lcd_init_sim();
	ADC_init();
	TIMSK = (1 << TOIE1) ;
// Enable timer1 overflow interrupt(TOIE1)
	TCCR1B = (1<<CS12) | (0<<CS11) | (1<<CS10);  // Timer mode with 1024 prescler
	TCNT1 = 64755;                                  //65535 - 0,1* (8*10^6 /1024)
	sei();

	while (1)
	{
		_tmp_=0x0000;     //reset tmp before reading the 2 nums
		//loop until sth is read
		while((number2=keypad_to_ascii(scan_keypad_rising_edge_sim())) == 0x00);
		while((number1=keypad_to_ascii(scan_keypad_rising_edge_sim())) == 0x00);
		if(number1 == '8' && number2 == '0')
		{
			in_welcome=1;           //signifies we have entered welcome
			lcd_command_sim(0x01);
			_delay_us(1530);
			lcd_data_sim('W');
			lcd_data_sim('E');
			lcd_data_sim('L');
```

```
                        lcd_data_sim('C');
                        lcd_data_sim('O');
                        lcd_data_sim('M');
                        lcd_data_sim('E');
                        lcd_data_sim(' ');
                        lcd_data_sim('B');
                        lcd_data_sim('8');
                        PORTB=0x80|level;               //turn MSB on
                        _delay_ms(4000);
                        level=level&0x7F;
                        PORTB=level;                    //turn MSB off
                        scan_keypad_rising_edge_sim();  //for remote access
                        out_welcome=1;                  //welcome is done

                }
                else
                {
                        int cnt=0;
                        while(cnt<4)                    //4*1 secs
                        {
                                PORTB=level|0x80;       //turn MSB on
                                _delay_ms(500);
                                PORTB = (PORTB&0x7F);   //turn MSB off
                                _delay_ms(500);
                                scan_keypad_rising_edge_sim(); //for remote access
                                cnt++;
                        }
                }

        }
}
```