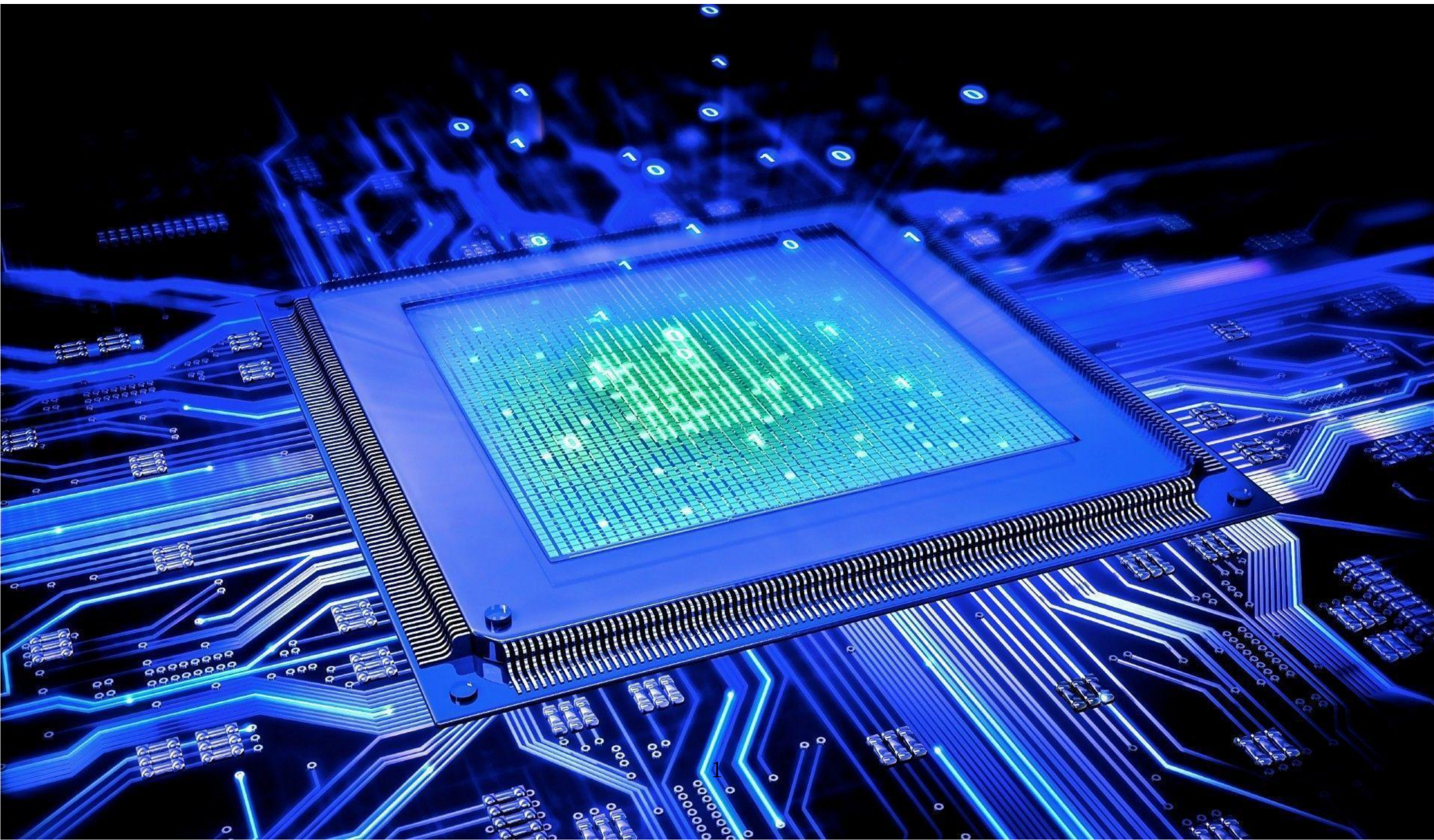
Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών &
Μηχανικών Υπολογιστών

Εργαστήριο Μικροϋπολογιστών

4η Εργαστηριακή Αναφορά - EasyAVR6

Δήμος Δημήτρης (031 17 165)
Αμπατζή Ναυσικά (031 17 198)
7ο Εξάμηνο - Ροή Υ
Χειμώνας 2020



1 Άσκηση 4.1 - Ηλεκτρονική Κλειδαριά και Ανίχνευση Μονοξειδίου του Άνθρακα CO (Assembly)

```
.include "m16def.inc"

.DSEG
        _tmp_: .byte 2

.CSEG
        .org 0x0
        rjmp MAIN
        .org 0x10
        rjmp ISR_TIMER1_OVF  ;routine for overflow interrupt of timer1
        .org 0x1c                       ;routine for ADC interupt
        rjmp ADC_SERVICE_ROUTINE


; ------------------------------
; -------- DEFINITIONS ---------
; ------------------------------

.equ one    = 0x10
.equ three = 0x40
.equ password = 0x0D
.def temp = r20
.def pressed  = r24
.def first   = r21
.def second = r22
.def counter = r16
.def temp2 = r17
.def flag = r18
.def temp1 = r19
.def flagk = r23
.def save = r15
.def clear = r26
.def gas_ = r27




; ------------------------------
; --------- ROUTINES -----------
; ------------------------------


ADC_init:ldi r24,(1<<REFS0) ; Vref: Vcc
                 out ADMUX,r24 ;MUX4:0 = 00000 for A0.
                 ;ADC is Enabled (ADEN=1)
                 ;ADC Interrupts are Enabled (ADIE=1)
                 ;Set Prescaler CK/128 = 62.5Khz (ADPS2:0=111)
```

```
                  ldi r24,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
                  out ADCSRA,r24
                  ret

scan_row_sim:
        out PORTC, r25
        push r24
        push r25
        ldi r24,low(500)
        ldi r25,high(500)
        rcall wait_usec
        pop r25
        pop r24
        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


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


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_sim:
        push r24
        push r25
        ldi r24 ,low(6000)
        ldi r25 ,high(6000)
        rcall wait_usec
        pop r25
        pop r24
        push r24
        in r25, PIND
        andi r25, 0x0f
        andi r24, 0xf0
        add r24, r25
        out PORTD, r24
        sbi PORTD, PD3
        cbi PORTD, PD3
        push r24
        push r25
        ldi r24 ,low(6000)
        ldi r25 ,high(6000)
        rcall wait_usec
        pop r25
        pop r24
        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_sim
        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_sim
        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
        push r24
        push r25
        ldi r24,low(1000)
        ldi r25,high(1000)
        rcall wait_usec
        pop r25
        pop r24
        ldi r24, 0x30
        out PORTD, r24
        sbi PORTD, PD3
        cbi PORTD, PD3
        ldi r24,39
        ldi r25,0
        rcall wait_usec
        push r24
        push r25
        ldi r24 ,low(1000)
        ldi r25 ,high(1000)
        rcall wait_usec
        pop r25
        pop r24
        ldi r24,0x20
        out PORTD, r24
        sbi PORTD, PD3
        cbi PORTD, PD3
        ldi r24,39
        ldi r25,0
        rcall wait_usec
        push r24
        push r25
        ldi r24 ,low(1000)
        ldi r25 ,high(1000)
        rcall wait_usec
        pop r25
        pop r24
        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
        brne wait_usec
        ret




ISR_TIMER1_OVF:
ldi temp,0xef
out ADCSRA,temp; anable ADC interrupt
reti

ADC_SERVICE_ROUTINE:
in temp,SREG
push temp
push r16
push r17
clr temp
clr temp2
clr r16                         ; clear input
clr r17

in r16,ADCL                     ; read ADC input
in r17,ADCH

mov temp,r16            ; save VL temporary
mov temp2,r17           ; save VH temporary

cpi r17,0x00            ; only if VH=0 we may be at zones 1 and 2 (safe zones)
brne GAS                ; if VH>0 we check for higher levels (alarm zone)
subi r16,0x15
brsh cont       ; if VL>=21 continue checking, else the value is invalid->  check again

pop r17
pop r16
pop temp
out SREG,temp

ldi temp,0xfc           ; initiallize TCNT1
out TCNT1H ,temp        ; for overflow after 0.1 sec
ldi temp ,0xf3
out TCNT1L ,temp
reti

cont:
mov r16,temp            ; reset VL
cpi r16,0x72            ; if VL < 114 (VH <= 113)
brlo PB0_set            ; Cx:[0-35], PB0 is ON (safe zone)

mov r16,temp            ; else bring back VL
cpi r16,0xce            ; if  113<VL<206 (113<VL<=205)
brlo PB1_set            ; Cx:[36-70], PB1 is ON (safe zone)
rjmp GAS                ; else check for higher levels(alarm)

PB0_set:
ldi flag,0x01
pop r17
pop r16
pop temp
out SREG,temp
ldi temp,0xfc           ; initiallize TCNT1
out TCNT1H ,temp        ; for overflow after 0.1 sec
ldi temp ,0xf3
out TCNT1L ,temp
```

```
reti

PB1_set:
ldi flag,0x02
pop r17
pop r16
pop temp
out SREG,temp
out SREG,temp
ldi temp,0xfc                   ; initiallize TCNT1
out TCNT1H ,temp                ; for overflow after 0.1 sec
ldi temp ,0xf3
out TCNT1L ,temp
reti

;----------  NOT SAFE ZONE ------------

GAS:
mov r17,temp2
andi r17,0x03
cpi r17,0x00        ; if VH is still 0 we are at level_3 (205<Cx<=298)
breq PB2_PB3_set        ; leds PB2 and PB3 are ON-> level_3

mov r17,temp2
andi r17,0x03
cpi r17,0x01            ; if VH>1 we may be at level_4
brne level_4_check
mov r16,temp
cpi r16,0x2b            ; if ADC<299 we are at level_3
brlo PB2_PB3_set
rjmp PB4_PB5            ; else we are at level_4

PB2_PB3_set:
ldi flag,0xc
pop r17
pop r16
pop temp
out SREG,temp
ldi temp,0xfc                   ; initiallize TCNT1
out TCNT1H ,temp                ; for overflow after 0.1 sec
ldi temp ,0xf3
out TCNT1L ,temp
reti

level_4_check:
mov r17,temp2                   ; restore VH
andi r17,0x03
subi r17,0x03                   ; if VH>=3, ADC>614 (max for 3 V) -> show nothing
brsh nothing
```

```
mov r16,temp                    ; restore VL
cpi r16,0x67
brlo PB4_PB5
rjmp nothing

PB4_PB5:
ldi flag,0x30
pop r17
pop r16
pop temp
out SREG,temp
ldi temp,0xfc                   ; initiallize TCNT1
out TCNT1H ,temp                ; for overflow after 0.1 sec
ldi temp ,0xf3
out TCNT1L ,temp
reti

nothing:
ldi flag,0x00
pop r17
pop r16
pop temp
out SREG,temp
ldi temp,0xfc                   ; initiallize TCNT1
out TCNT1H ,temp                ; for overflow after 0.1 sec
ldi temp ,0xf3
out TCNT1L ,temp
reti

; -----------------------------------
; --------- MAIN PROGRAM -----------
; -----------------------------------
MAIN:
; stack initialization
ldi temp,LOW(RAMEND)
out SPL, temp
ldi temp,HIGH(RAMEND)
out SPH, temp

; I/O definition. input: PORTC, output: PORTB, PORTD (LCD Display)
ldi temp, (1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4)
out DDRC, temp
ser temp
out DDRB, temp
out DDRD, temp
clr temp
out PORTB,temp
clr r24
```

```
rcall lcd_init_sim
push r24
push r25
ldi r24,low(100)
ldi r25,high(100)
rcall wait_msec
pop r25
pop r24

; enable overflow interrupt of TCNT1
ldi temp,(1<<TOIE1)
out TIMSK,temp
; CK/1024  increasing frequency
ldi temp ,(1<<CS12) | (0<<CS11) | (1<<CS10)
out TCCR1B ,temp
sei

rcall ADC_init

; ------------------------------
; --------- MAIN LOOP -----------
; ------------------------------

;------- CHECK KEYBOARD ---------
START:
clr counter                     ; clear counter
clr flagk
mov flag,temp1

digit1:
rcall scan_keypad_rising_edge_sim   ; check for pressed bottons
rcall keypad_to_ascii_sim           ; transform into ascii form
cpi pressed, 0x00
brne contt      ; loop if no button is pressed
rjmp main1
contt:
subi pressed, 0x30                  ; convert from ascii to right form
mov first, pressed                  ; save first input digit,
                                   ; before reading the second


digit2:
rcall scan_keypad_rising_edge_sim       ; check for pressed bottons
rcall keypad_to_ascii_sim               ; transform into ascii form
cpi pressed, 0x00
breq digit2                             ; loop if no button is pressed

subi pressed, 0x30                      ; convert from ascii to right form
mov second, pressed                     ; save second input digit
```

```
; check if input matches password
ldi temp, 0x0a
mul first, temp
mov first, r0
add first, second                    ; given combo = first*10 + second
cpi first, password
brne wrong_pass                      ; check if given combo matches the password


; correct given combo
ldi flagk,0x02
rjmp main1

; incorrect given combo
wrong_pass:
ldi flagk,0x01
rjmp main1


;----------- CHECK LEVELS OF CO ------------


main1:
mov save,flagk                                               ; save keyboard status
mov temp1,flag                                               ; save CO level status
cpi flag,0x00
breq no_output

cpi flagk,0x00                               ; check if any password was given
breq no_keypad                               ; if not just continue
rjmp combo_given

;-------- ADC > 614 -----------
no_output:
clr r24
rcall lcd_init_sim
clr flag
out PORTB,flag
mov flag,temp1
rjmp START


;----------------- NO PASSWORD WAS GIVEN ---------------------------
no_keypad:
mov flag,temp1
cpi flag,0x03                                ; check if we are at safe zone or not
brlo safe
```

```
rjmp ALARM

;---------- SAFE ZONE ----------

safe:
mov flag,temp1                                                  ; restore level of CO
out PORTB,flag                                                  ; level -> leds
cpi clear,0x01
breq START
clr r24
rcall lcd_init_sim
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
ldi clear,0x01

rjmp START

;--------------- CO DETECTED - ALARM  ---------------
ALARM:
keep_alarm:
ldi clear,0x00
ldi counter,0x02                        ; level of CO blinks and message 'GAS DETECTED'
;cpi gas_,0x02
;breq ALARM1
clr r24
rcall lcd_init_sim

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 gas_,0x02

ALARM1:          mov flag,temp1                                 ; loop for blink
out PORTB,temp1
push r24
push r25
ldi r24,low(200)
ldi r25,high(200)
rcall wait_msec
pop r25
pop r24

clr flag
out PORTB,flag
dec counter
cpi counter,0x00
brne ALARM1
mov flag,temp1
rjmp START

;-------------------------------------------------------


;----------------- PASSWORD WAS GIVEN ------------------

combo_given:
mov flagk,save
cpi flagk,0x02                                          ; was the password correct?
breq correct_password_before            ; if yes, go at correct_password

wrong_password:
push counter                                    ; if not PB7 starts blinking
ldi counter,0x04                                ; for 4 secs

wrong_passwrd1:
mov temp1,flag                                  ; check level of CO within these 4 secs
 cpi flag,0x01                                  ; and show the level at the leds
 breq save_zone_password
 mov flag,temp1
 cpi flag,0x02
 breq save_zone_password

;---------- alarm zone password and wrong password ----------
clr r24
rcall lcd_init_sim

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 flagk,0x80
mov flag,temp1
add flag,flagk
out PORTB,flag
push r24
push r25
ldi r24,low(500)
ldi r25,high(500)
rcall wait_msec
pop r25
pop r24
clr flag
out PORTB,flag
push r24
push r25
ldi r24,low(500)
ldi r25,high(500)
rcall wait_msec
```

```
pop r25
pop r24
dec counter

cpi counter,0x00
breq bai
rjmp wrong_passwrd1

correct_password_before: rjmp correct_password

;---------- safe zone and wrong password ----------

save_zone_password:
clr r24
rcall lcd_init_sim
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

ldi flagk,0x80
mov flag,temp1
add flag,flagk
out PORTB,flag
push r24
push r25
ldi r24,low(500)
ldi r25,high(500)
rcall wait_msec
pop r25
pop r24
out PORTB,temp1
push r24
push r25
ldi r24,low(500)
ldi r25,high(500)
rcall wait_msec
pop r25
pop r24
dec counter

cpi counter,0x00
breq bai
```

```
rjmp wrong_passwrd1



bai:
pop counter
rjmp START

;---------------- correct_password --------------------
correct_password:
mov temp1,flag
cpi flag,0xc
breq resque
mov flag,temp1
cpi flag,0x30
breq    resque

;------------ Safe zone and correct password--------------
clr r24
rcall lcd_init_sim

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
push counter
ldi counter,0x04


safe_and_correct:
mov temp1,flag
cpi counter,0x00
brne more_check
rjmp START

more_check:

mov flag,temp1
ldi flagk,0x80
```

```
add flag,flagk
out PORTB,flag

push r24
push r25
ldi r24,low(500)
ldi r25,high(500)
rcall wait_msec
pop r25
pop r24


mov temp1,flag
cpi flag,0xc
breq pass_changed
mov temp1,flag
cpi flag,0x30
breq pass_changed
dec counter
rjmp   safe_and_correct

resque: rjmp resque_team
;----------------- level has changed during the 4 secs ------------------------
pass_changed:
cpi counter,0x00
breq bai
ldi flag,0x80
out PORTB,flag
push r24
push r25
ldi r24,low(500)
ldi r25,high(500)
rcall wait_msec
pop r25
pop r24
ldi flag,0x80
add flag,temp1
out PORTB,flag

push r24
push r25
ldi r24,low(500)
ldi r25,high(500)
rcall wait_msec
pop r25
pop r24
dec counter
rjmp pass_changed
```

```
;-------- ALARM and corect password---------------
resque_team:
clr r24
rcall lcd_init_sim

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 flag,0x80
out PORTB,flag
push r24
push r25
ldi r24,low(4000)
ldi r25,high(4000)
rcall wait_msec
pop r25
pop r24
rjmp START
```

2 Άσκηση 4.2 - Ηλεκτρονική Κλειδαριά και Ανίχνευση Μονοξειδίου του Άνθρακα CO (C)

```
#define F_CPU 8000000UL
#define SPARK_DELAY 15
#define PASSWORD 13

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

unsigned char scan_row_sim(int);
unsigned int scan_keypad_sim(void);
unsigned int scan_keypad_rising_edge_sim(void);
unsigned char keypad_to_ascii_sim(unsigned int);
unsigned int keypad(void);
void ADC_init(void);
unsigned char temp,i,level,keypad_check,flag;
unsigned int previous_state = 0;
unsigned char one_two;


ISR(TIMER1_OVF_vect){

        ADCSRA =0xEF;
}

ISR(ADC_vect) { // check the CO level

        unsigned char flag_ = 0;

        if(21<=ADC & ADC<=113){ //Cx = [0,35] ppm
                level = 0x01;
                flag_ = 1;
                //PORTB = 0x01;
                //TCNT1H = 0xfc;
                //TCNT1L = 0xf3;                //Timer1 overflows in 100 msec
        }


        if(113<ADC & ADC<=205){ // Cx = [36,70] ppm
                level = 0x02;
                flag_ = 1;
                //PORTB = 0x02;

                //TCNT1H = 0xfc;
                //TCNT1L = 0xf3;                //Timer1 overflows in 100 msec
        }
```

```
         if(205<ADC & ADC<=298  ){ // Cx = [71,105] ppm
                 level = 0xc;
                 flag_ = 1;
                 //TCNT1H = 0xfc;
                 //TCNT1L = 0xf3;

         }

        if(298<ADC & ADC<=614){  // Cx = [105, ] ppm
                level = 0x30;
                flag_ = 1;
                //TCNT1H = 0xfc;
                //TCNT1L = 0xf3;                //Timer1 overflows in 100 msec
        }

        if(!flag_)
                level = 0x00;
                TCNT1H = 0xfc;
                TCNT1L = 0xf3;          //Timer1 overflows in 100 msec
        }

int main(void) {
                /* IO Settings */
                DDRB = 0xFF;
                DDRC = 0xF0;

                TIMSK = (1 << TOIE1);                          //Timer1 ,interrupt enable
            //frequency of Timer1 8MHz/1024
                TCCR1B =(1<<CS12) | (0<<CS11) | (1<<CS10);

                sei();

while(1){



flag = keypad();
if(flag){                                                  // if any password was given
        if(flag!=PASSWORD){                                // and it was wrong
                for(int i=0; i<4; i++){                    // PB7 blinks for 4 secs
                if(level==0x01 | level==0x02){             // and level of CO is also shown
//at the leds
                        PORTB = level+0x80;
                        _delay_ms(500);                    // safe zones
                        PORTB = level;
                        _delay_ms(500);
                }

                if (level==0xc | level==0x30){             // alarm zones
```

```
                        PORTB = level+0x80;
                        _delay_ms(500);
                        PORTB = 0x00;
                        _delay_ms(500);
                }

                }
        }

        if(flag==PASSWORD){      // if the password was correct PB7 is ON for 4 secs
                                //and the level of CO is also shown

                if (level==0xc | level==0x30){  // if the alarm was ON, it closes for 4 secs
                        PORTB = 0x80;
                        _delay_ms(4000);
                }

                unsigned char m=0;
                if(level==0x01 | level==0x02){           // safe levels of CO

                while(m!=8){
                        PORTB = 0x80 +level;
                        _delay_ms(500);
                        m++;
                        if ((level==0xc | level==0x30)){// if within these 4 secs the
                                                        //level of CO goes to alarm zone
                                while (m!=8)
                                {
                                        PORTB = 0x80;
                                        _delay_ms(500);
                                        //m++;                  // show it
                                        PORTB = level+0x80;
                                        _delay_ms(500);
                                        m++;

                                }
                        }

                }

        }
        }
}

// No combo was given, just show the CO levels
else{
        if(level==0x01 | level==0x02){
                PORTB = level;
        }

        if(level==0xc | level==0x30){
                for(int i=0; i<2; i++){
                        PORTB = level;
                        _delay_ms(500);
                        PORTB = 0x00;
                        _delay_ms(500);

                }
        }
}

}

}

void ADC_init(void)
{
        ADMUX = (1<<REFS0);
        ADCSRA = (1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

unsigned int keypad(void){
        int if_pressed = 0x00;
        unsigned char first, second;
        unsigned int given_combo;
                // wait, until a button is pressed
        first = scan_keypad_rising_edge_sim();
        if( first == 0){
        if_pressed = 0x00;
        return if_pressed;
        }

        else{
        first = keypad_to_ascii_sim(first) - 0x30;       // transform into the right form

        do {    // wait until a button is pressed
                second = scan_keypad_rising_edge_sim();
        }while(!second);
        second = keypad_to_ascii_sim(second) - 0x30; // transform into the right form

        given_combo = first*10 + second;
        scan_keypad_rising_edge_sim();  // no actual need - bug fix
return given_combo;
        }
}
```

```
unsigned char scan_row_sim(int row) {
        volatile unsigned char pressed_row;

        temp = 0x08;
        PORTC = temp << row;

        _delay_us(500);
        asm("nop");
        asm("nop");
        pressed_row = PINC & 0x0f;

        return pressed_row;
}


unsigned int scan_keypad_sim(void) {
        unsigned int row1, row2, row3, row4;
        int pressed_button = 0x00;

        row1 = scan_row_sim(1);
        row2 = scan_row_sim(2);
        row3 = scan_row_sim(3);
        row4 = scan_row_sim(4);

        pressed_button = (row1 << 4) | (row2);
        if(pressed_button)
        one_two = 1;
        else {
                one_two = 0;
                pressed_button = (row3 <<4 ) | (row4);
        }
        PORTC = 0x00;
        return pressed_button;
}


unsigned int scan_keypad_rising_edge_sim(void) {

        unsigned int button1, button2, current_state, final_state;

        button1 = scan_keypad_sim();
        _delay_ms(SPARK_DELAY);
        button2 = scan_keypad_sim();
        current_state = button1 & button2;
        final_state = current_state & (~ previous_state);
        previous_state = current_state;

        return final_state;
}


unsigned char keypad_to_ascii_sim(unsigned int final_state) {

        if(one_two) {
                switch (final_state) {
                        case 0x10:
                        return '1';
                        case 0x20:
                        return '2';
                        case 0x40:
                        return '3';
                        case 0x80:
                        return 'A';
                        case 0x01:
                        return '4';
                        case 0x02:
                        return '5';
                        case 0x04:
                        return '6';
                        case 0x08:
                        return 'B';
                }
        }
        else {
                switch(final_state){
                        case 0x10:
                        return '7';
                        case 0x20:
                        return '8';
                        case 0x40:
                        return '9';
                        case 0x80:
                        return 'C';
                        case 0x01:
                        return '*';
                        case 0x02:
                        return '0';
                        case 0x04:
                        return '#';
                        case 0x08:
                        return 'D';
                }
        }
}
```