ΕΜΠ - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2021-2022

ΑΘΗΝΑ 15-01-2022

4^{η} ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ ''Εργαστήριο Μικροϋπολογιστών''
3^{η} Εργ. Άσκ. στον Μικροελεγκτή AVR – Χρονιστές, ADC

(υλοποίηση στο εκπαιδευτικό σύστημα easyAVR6)

Ομάδα 41
Πεγειώτη Νάταλυ – 03117707

Στην παρούσα άσκηση ζητείται η υλοποίηση μίας "ηλεκτρονικής κλειδαριάς" που ενσωματώνει ελεγκτή για την συγκέντρωση του επιπέδου του CO στον χώρο που θα τοποθετηθεί.
Συγκεκριμένα, γίνεται έλεγχος του επιπέδου του CO ανά 100ms και γίνεται εμφανής η συγκέντρωση αυτή μέσω των led PB0-PB6 ανάλογα της μετρούμενης τιμής, καθώς και από το μήνυμα που εμφανίζεται στην οθόνη LCD.
Καθώς η συγκέντρωση αυξάνεται, τόσα περισσότερα leds αναβοσβήνουν.

Σε περίπτωση που πατηθεί ο κωδικός της ομάδας (41) στο πληκτρολόγιο τότε η λειτουργία του ελεγκτή σταματά, το Led PB7 ανάβει σταθερά για 4 sec και εμφανίζεται στην LCD οθόνη το μήνυμα "WELCOME 41".
Αν δοθεί λανθασμένος κωδικός, τότε δεν αλλάζει η λειτουργιά του ελεγκτή αλλά ταυτόχρονα αναβοσβήνει το led PB7 για 4 sec με συχνότητα 1 sec (0.5 sec αναμμένο και 0.5 sec σβηστό).

Για την υλοποίηση του προγράμματος γίνεται χρήση του χρονιστή timer1 και του ADC μετατροπέα, οι λειτουργίες των οποίων επεξηγούνται ακολούθως.

Ρουτίνα Εξυπηρέτησης Διακοπής Χρονιστή:
Για την υλοποίηση της ανανέωσης της τιμής του αισθητήρα ανά 100msec χρησιμοποιείται ο timer1. Για την αρχικοποίηση του χρονιστή πραγματοποιούνται οι ακόλουθες πράξεις:

- ✓ Συχνότητα Διακοπής: ftimer1 = 8MHz/1024=7812.5Hz
- ✓ Χρόνος Διακοπής: 100ms=0.1sec
- ✓ Μετρούμενοι Κύκλοι: cc_count = 0.1*7812.5=781.25cc
- ✓ Έναρξη Χρονιστή: tstart = 65536-781.25=64754.75 ~ 64755 = 0xFCF3

Έτσι εγγυάται η διακοπή ανά 100msec.
Όταν συμβαίνει η διακοπή του timer1, ο έλεγχος του προγράμματος περνάει στην ρουτίνα εξυπηρέτησης διακοπής του χρονιστή, όπου εκεί γίνεται εκκίνηση της μετατροπής του ADC. Έτσι υλοποιείται η ρουτίνα του ADC και στην συνέχεια επιστρέφει στην παρούσα ρουτίνα όπου γίνεται επαναρχικοποίηση της τιμής του χρονιστή και μεταβαίνει στο κύριο πρόγραμμα.

Ρουτίνα Εξυπηρέτησης Διακοπής του ADC

Ο έλεγχος του προγράμματος μεταβαίνει στην ρουτίνα εξυπηρέτησης διακοπής του ADC όταν ολοκληρωθεί η μετατροπή της αναλογικής τιμής που διαβάζεται από τον αισθητήρα σε ψηφιακή.

Για τον υπολογισμό της συγκέντρωσης του CO χρησιμοποιούνται οι εξής τύποι:

$$M = Sensitivity\ Code * TIA\ Gain * 10^{-9} * 10^{3} = 1.9 * 100 * 10^{-9} * 10^{3} \Rightarrow$$
$$\Rightarrow M = 0.0129\, V/ppm$$

$$C_x = \frac{V_{gas} - V_{gas0}}{M} \xrightarrow{V_{gas0}=0.1} C_x = \frac{V_{gas} - 0.1}{0.0129} \xrightarrow{V_{gas0}=\frac{5*ADC}{1024}} C_x = \frac{\frac{5*ADC}{1024} - 0.1}{0.0129} \Rightarrow$$

$$\Rightarrow C_x = \frac{5*ADC - 102.4}{13.2096} \Rightarrow C_x = 0.3785 * ADC - 7.7519 \Rightarrow$$

$$\Rightarrow ADC = \frac{C_x + 7.7519}{0.3785}$$

Επιλέγεται βήμα 35ppm, έτσι κάθε led αντιστοιχεί στο ακόλουθο εύρος τιμών:

PB0: $C_x \le 35 \Rightarrow ADC \le 113.75 \Rightarrow ADC \le 114 = 0x72$
PB1: $35 < C_x \le 70 \Rightarrow 113.75 < ADC \le 206.87 \Rightarrow 114 < ADC \le 207 = 0xCF$
PB2: $70 < C_x \le 105 \Rightarrow 206.87 < ADC \le 299.99 \Rightarrow 207 < ADC \le 300 = 0x12C$
PB3: $105 < C_x \le 140 \Rightarrow 299.99 < ADC \le 393.11 \Rightarrow 300 < ADC \le 394 = 0x18A$
PB4: $140 < C_x \le 175 \Rightarrow 393.11 < ADC \le 486.24 \Rightarrow 394 < ADC \le 487 = 0x1E7$
PB5: $175 < C_x \le 210 \Rightarrow 486.24 < ADC \le 579.36 \Rightarrow 487 < ADC \le 580 = 0x244$
PB6: $210 < C_x \le 379.4536 \Rightarrow 579.36 < ADC \le 1023 \Rightarrow 580 < ADC \le 1023 = 0x3FF$

Αφού ολοκληρωθεί η μετατροπή και ληφθεί η σωστή τιμή της συγκέντρωσης, αν αυτή είναι μικρότερη από 70 ppm, τότε ανάβει αναλόγως το PB0 ή τα PB0-PB1 και εμφανίζεται στην οθόνη η ένδειξη "CLEAR", ενώ αν είναι πάνω από 70 ppm, τότε αναβοσβήνουν όσα led προκύπτουν από τους παραπάνω τύπους και εμφανίζεται στην οθόνη η ένδειξη "GAS DETECTED".
Επίσης γίνεται έλεγχος μέσω σημαίας, ούτως ώστε η ένδειξη "CLEAR" να εμφανίζεται μόνο αν η προηγούμενη κατάσταση ήταν η "GAS DETECTED" και αντίστροφα η ένδειξη "GAS DETECTED" να εμφανίζεται μόνο αν η προηγούμενη κατάσταση ήταν "CLEAR" ή "WELCOME 41".

Ζήτημα 4.1

Υλοποίηση σε Assembly. Ο κώδικας για την υλοποίηση του προγράμματος είναι ο ακόλουθος, ο οποίος επεξηγείται με αναλυτικά σχόλια:

```
.include "m16def.inc"

.DSEG
_tmp_: .byte 2
.CSEG

.org 0x0
jmp reset
.org 0x10
rjmp ISR_TIMER1_OVF
.org 0x1c
rjmp ADC_ISR


reset: ldi r24, low(RAMEND)        ; initialize stack pointer
       out SPL, r24
       ldi r24, high(RAMEND)
       out SPH, r24

       clr r24
       rcall lcd_init_sim          ; initialize with clear screen
       call ADC_init               ; initialize ADC

       ldi r30, 0           ; initialize with flag = 0
       ldi r31, 0           ; flag to make leds blink in Gas detected state

       ser r24
       out DDRB, r24        ; initialize PORTB for output
       out DDRD, r24        ; initialize PORTD that is connected to LCD, as output

       ; set as output the 4 MSB of PORTC
       ldi r24, (1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4)
       out DDRC, r24

       ldi r24, (1 << TOIE1)       ; activate overfloat interrupt of TCNT1
       out TIMSK, r24              ; timer1
       ldi r24, (1 << CS12) | (0<<CS11) | (1<<CS10)    ; CK/1024
       out TCCR1B, r24


init:  rcall lcd_init_sim
       ; fcount = 8MHz/1024=7812.5Hz
       ; timer: 100ms=0.1sec
       ; cc_count = 0.1*7812.5=781.25cc
       ; START = 65536-781.25=64754.75 ~ 64755 = 0xFCF3
       ldi r24,0xFC  ;initialize TCNT1
       out TCNT1H, r24
       ldi r24, 0xF3
       out TCNT1L, r24
       sei    ; activate interrupts


main:  rcall scan_keypad_rising_edge_sim
       rcall keypad_to_ascii_sim          ; read 1st digit
       cpi r24, 0                         ; repeat until 1st digit is valid
       breq main
       mov r20,r24                        ; store 1st digit in r20
```

```
next:  rcall scan_keypad_rising_edge_sim
       rcall keypad_to_ascii_sim         ; read 2nd digit
       cpi r24, 0                        ; repeat until 2nd digit is valid
       breq next
       mov r21,r24                       ; store 2nd digit in r21
       rcall scan_keypad_rising_edge_sim ; we call that for safety reasons

       cpi r20, 52          ; if 1st digit != 4
       brne wrong_key
       cpi r21, 49          ; or 2nd digit != 1
       brne wrong_key       ; wrong_key given, go to wrong_key


correct_key:  cli           ; disable interupts
              ldi r30, 0    ; flag = 0
              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 r24, ' '
              rcall lcd_data_sim
              ldi r24, '4'
              rcall lcd_data_sim
              ldi r24, '1'
              rcall lcd_data_sim  ;display "WELCOME 41"

              ldi r19, (1 << PB7)
              out PORTB, r19      ; turn on PB7
              ldi r24, low(4000)
              ldi r25, high(4000)
              rcall wait_msec     ; delay 4sec
              ldi r19, (0 << PB7)
              out PORTB, r19      ; turn off PB7
              rjmp init           ; return to activate overfloat interrupt of TCNT1

wrong_key:    ldi r18, 4          ; loop for 4 times
loop:         cpi r18, 0
              breq finish
              in r19, PINB
              ori r19, 128
              out PORTB, r19      ; turn on PB7
              ldi r24, low(500)
              ldi r25, high(500)
              rcall wait_msec     ; delay 0.5sec
              in r19, PINB
              andi r19, 127
              out PORTB, r19      ; turn off PB7
              ldi r24, low(500)
              ldi r25, high(500)
              rcall wait_msec     ; delay 0.5sec
              dec r18
              rjmp loop
finish:       rjmp main           ; return to main
```

```
; interupt service routine of timer1
ISR_TIMER1_OVF:
       cli
       ldi r24,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADSC)
       out ADCSRA,r24
       ldi r24,0xFC ;initialize TCNT1
       out TCNT1H, r24
       ldi r24, 0xF3
       out TCNT1L, r24
       sei
       reti

; interupt service routine (read CO level and acts accordingly)
ADC_ISR:      in r28, ADCL
              in r29, ADCH  ; read ADC
              cpi r29, 0
              breq ZERO
              cpi r29, 1
              breq ONE
              cpi r29, 2
              breq TWO
              cpi r29, 3
              breq THREE
       ZERO:  cpi r28, 0x72
              brlo led_PB0  ; ADC <= 114 => Cx <= 35
              cpi r28, 0xcf
              brlo led_PB1  ; 114 < ADC <= 207 => 35 < Cx <= 70
       ONE:   cpi r28, 0x2c
              brlo led_PB2  ; 207 < ADC <= 300 => 70 < Cx <= 105
              cpi r28, 0x8a
              brlo led_PB3  ; 300 < ADC <= 394 => 105 < Cx <= 140
              cpi r28, 0xe7
              brlo led_PB4  ; 394 < ADC <= 487 => 140 < Cx <= 175
       TWO:   cpi r28, 0x44
              brlo led_PB5  ; 487 < ADC <= 580 => 175 < Cx <= 210
       THREE: cpi r28, 0xff
              brlo led_PB6  ; 580 < ADC <= 1023 => 210 < Cx <= 1023

led_PB0:      ldi r17, 0b00000001
              out PORTB, r17       ; turn on PB0
              rjmp CLEAR
led_PB1:      ldi r17, 0b00000011
              out PORTB, r17       ; turn on PB0 - PB1
              rjmp CLEAR
led_PB2:      ldi r17, 0b00000111
              rjmp Gas_on_led
led_PB3:      ldi r17, 0b00001111
              rjmp Gas_on_led
led_PB4:      ldi r17, 0b00011111
              rjmp Gas_on_led
led_PB5:      ldi r17, 0b00111111
              rjmp Gas_on_led
led_PB6:      ldi r17, 0b01111111
              rjmp Gas_on_led

Gas_on_led:
       cpi r31,0     ; make leds blink
       breq turn_on  ; if in previous Gas detected state leds were on, turn them off
       ldi r31,0
       ldi r17,0
       rjmp turn_off
turn_on:      ldi r31,1
turn_off:     out PORTB, r17 ; turn on or off PB0 - PB6
              rjmp GAS
```

```
CLEAR: cpi r30, 0    ; check if the previous state was GAS LEAK
       breq already_clean
       ldi r30, 0
       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   ;display "CLEAR"
       ldi r24, low(200)
       ldi r25, high(200)
       rcall wait_msec ; delay 0.2sec
       rjmp fin
already_clean:       rcall lcd_init_sim
                     rjmp fin



GAS:   cpi r30, 1    ; if previous state was Gas detected then do nothing
       breq fin
       ldi r30, 1           ; set flag = 1 if there is a GAS LEAK
       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   ;display "GAS DETECTED"
       rjmp fin



fin:   ldi r24,0xFC  ;initialize TCNT1
       out TCNT1H, r24
       ldi r24, 0xF3
       out TCNT1L, r24
       sei
       reti
```

```
; Routine: usart_init
; Description:
; This routine initializes the
; ADC as shown below.
; ------- INITIALIZATIONS -------
;
; Vref: Vcc (5V for easyAVR6)
; Selected pin is A0
; ADC Interrupts are Enabled
; Prescaler is set as CK/128 = 62.5kHz
; --------------------------------
; parameters: None.
; return value: None.
; registers affected: r24
; routines called: None
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<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
              out ADCSRA,r24
              ret


//given code for functions to be called
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


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


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_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


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


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
```

Ζήτημα 4.1

Υλοποίηση σε C. Ο κώδικας για την υλοποίηση του προγράμματος είναι ο ακόλουθος, ο οποίος επεξηγείται με αναλυτικά σχόλια:

```
#define F_CPU 8000000        // frequency is set 8MHz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

int num_pressed, tmp,i;     // 16-bit number to store the key that was pressed and tmp
char first_digit, second_digit;    // digits pressed
char x, y, z, flag_clear=0, flag_blink=0, leds;


char scan_row_sim(char y)   // y-->r25
{
       PORTC = y;    // Current line is set to 1
       _delay_us(500);      // delay for ~ 0.5usec
                            //(each 'nop' is 1/4usec so it's included to 500usec)
       return PINC & 0x0F;  // keep the 4 LSB of PORTC
}


int scan_keypad_sim(void)   // x-->r24, y-->r25, z-->r26, h-->r27
{
       char z,h;     // set as parameters so as to be topical
       int result;   // result = r25:r24 to be returned

       y = 0x10;     // check 1st line
       x = scan_row_sim(y); // keep the result
       h =  x<<4;    // and save it in the 4 MSB of h

       y = 0x20;     // check 2nd line
       x = scan_row_sim(y); // keep the result
       h = h+x;      // and save it in the 4 LSB of h

       y = 0x40;     // check 3rd line
       x = scan_row_sim(y); // keep the result
       z =  x<<4;    // and save it in the 4 MSB of z

       y = 0x80;     // check 2nd line
       x = scan_row_sim(y); // keep the result
       z = z+x;      // and save it in the 4 LSB of z

       result = h;
       return (result<<8) + z;     //return correct number
}



int scan_keypad_rising_edge_sim(void)
{
       int y = scan_keypad_sim();  // check the keypad for pressed button
       _delay_ms(15);       // delay for ~ 15msec
       int z = scan_keypad_sim();  // check the keypad again
       y = y & z;    // bitwise and, so to have the correct result
       z = tmp;      // load from RAM the previous value
       tmp = y;      // save in RAM the new value
       z = ~z;       // one's complement
       y = y & z;    // bitwise and
       return y;     // return value
}
```

```
// function that makes the number pressed to the ascii value or 0
char keypad_to_ascii_sim(int x)
{
       switch(x){
              case 0x01:return '*';
              case 0x02:return '0';
              case 0x04:return '#';
              case 0x08:return 'D';
              case 0x10:return '7';
              case 0x20:return '8';
              case 0x40:return '9';
              case 0x80:return 'C';
              case 0x100:return '4';
              case 0x200:return '5';
              case 0x400:return '6';
              case 0x800:return 'B';
              case 0x1000:return '1';
              case 0x2000:return '2';
              case 0x4000:return '3';
              case 0x8000:return 'A';
       }
       return 0;
}



void write_2_nibbles_sim(char x)
{
       char k,v;     //local variable for this program
       _delay_us(6000);     // delay for ~ 6000usec | protection for simulation
       k = (PIND & 0x0f);   // k is r25
       v =  k + (x & 0xf0);
       PORTD = v;    //output in PORTD
       v = PIND | 0x08;
       PORTD = v;
       v = PIND & 0xf7;
       PORTD = v;                  //PD3=1 and then PD3=0 (enable pulse)
       _delay_us(6000);     // delay for ~ 6000usec | protection for simulation
       v =  k + ((x >> 4 | x << 4) & 0xf0);
       PORTD = v;    //output in PORTD
       v = PIND | 0x08;
       PORTD = v;                  //PD3=1 and then PD3=0 (enable pulse)
       v = PIND & 0xf7;
       PORTD = v;
}



void lcd_data_sim(char x)
{
       PORTD = (1<<PD2);
       write_2_nibbles_sim(x);
       _delay_us(43);       // delay for ~ 43usec
}



void lcd_command_sim(char x)
{
       PORTD = (0<<PD2);
       write_2_nibbles_sim(x);
       _delay_us(39);       // delay for ~ 39usec
}
```

```
void lcd_init_sim()
{
       char v;              //local variable
       _delay_us(40);       // delay for ~ 40usec
       PORTD = 0x30;        //8-bit mode
       v = PIND | 0x08;
       PORTD = v;           //PD3=1
       v = PIND & 0xf7;
       PORTD = v;           //PD3=0
       _delay_us(39);       // delay for ~ 39usec
       _delay_us(1000);     // delay for ~ 1000usec | protection for the simulation
       PORTD = 0x30;
       v = PIND | 0x08;
       PORTD = v;           //PD3=1
       v = PIND & 0xf7;
       PORTD = v;           //PD3=0
       _delay_us(39);       // delay for ~ 39usec
       _delay_us(1000);     // delay for ~ 1000usec | protection for the simulation

       PORTD = 0x20;        //change in 4-bit mode
       v = PIND | 0x08;
       PORTD = v;           //PD3=1
       v = PIND & 0xf7;
       PORTD = v;           //PD3=0
       _delay_us(39);       // delay for ~ 39usec
       _delay_us(1000);     // delay for ~ 1000usec | protection for the simulation
       lcd_command_sim(0x28);      // choose character size 5x8
       lcd_command_sim(0x0c);      // turn on screen, hide cursor
       lcd_command_sim(0x01);      // clear screen
       _delay_us(1530);            // delay for ~ 1530usec
       // activate automatic address increase by 1 and deactivate screen sliding
       lcd_command_sim(0x06);
}


void CLEAR()
{
       lcd_init_sim();
       if (flag_clear == 1)        // if previous state was Gas Detected display Clear
       {
              flag_clear=0;
              lcd_data_sim('C');
              lcd_data_sim('L');
              lcd_data_sim('E');
              lcd_data_sim('A');
              lcd_data_sim('R');
              _delay_ms(200);
       }
       TCNT1H = 0xfc;
       TCNT1L = 0xf3;
}


void GAS()
{
       if (flag_clear == 0)        // if previous state was Clear display Gas Detected
       {
              flag_clear=1;
              lcd_init_sim();
              lcd_data_sim('G');
              lcd_data_sim('A');
              lcd_data_sim('S');
              lcd_data_sim(' ');
              lcd_data_sim('D');
              lcd_data_sim('E');
```

```c
		lcd_data_sim('T');
		lcd_data_sim('E');
		lcd_data_sim('C');
		lcd_data_sim('T');
		lcd_data_sim('E');
		lcd_data_sim('D');
	}
	TCNT1H = 0xfc;
	TCNT1L = 0xf3;
}



void Gas_on_led() // make leds blink when in Gas Detected state
{
	// if in previous Gas detected state leds were on, turn them off
	if (flag_blink == 0)
	{
		flag_blink = 1;
		PORTB = leds;
	}
	else
	{
		flag_blink = 0;
		PORTB = 0;
	}
	GAS();
}



ISR(TIMER1_OVF_vect)
{
	cli(); //deactivate interrupts
	ADCSRA = (1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADSC);
	TCNT1H = 0xfc;
	TCNT1L = 0xf3;
	sei(); //reactivate interrupts
}



ISR(ADC_vect) // turn leds on according to the CO concentration level
{
	if (ADC<=0x72)
	{
		PORTB = 0b00000001;
		CLEAR();
	}
	else if (ADC<=0xcf)
	{
		PORTB = 0b00000011;
		CLEAR();
	}
	else if (ADC<=0x12c)
	{
		leds = 0b00000111;
		Gas_on_led();
	}
	else if (ADC<=0x18a)
	{
		leds = 0b00001111;
		Gas_on_led();
	}
```

```
        else if (ADC<=0x1e7)
        {
            leds = 0b00011111;
            Gas_on_led();
        }
        else if (ADC<=0x244)
        {
            leds = 0b00111111;
            Gas_on_led();
        }
        else
        {
            leds = 0b01111111;
            Gas_on_led();
        }
}


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


int main(void)
{
        ADC_init();
        lcd_init_sim();

        DDRB = 0xFF;  // initialize PB0-7 as output
        DDRD = 0xFF;
        // initialize PC4-7 as output
        DDRC = (1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4);

        //enable interrupts for timer , set frequency and initialize
        TIMSK = (1 << TOIE1);
        TCCR1B = (1 << CS12) | (0<<CS11) | (1<<CS10);
        TCNT1H = 0xfc;
        TCNT1L = 0xf3;
        sei();

        while(1)
        {
            first_digit = 0;     //initialize first digit with 0
            second_digit = 0;    //initialize first digit with 0
            do{
                num_pressed = scan_keypad_rising_edge_sim();
                // read and store 1st digit
                first_digit = keypad_to_ascii_sim(num_pressed);
            } while(first_digit==0);    //repeat until first digit is valid
            do{
                num_pressed = scan_keypad_rising_edge_sim();
                // read and store 2nd digit
                second_digit = keypad_to_ascii_sim(num_pressed);
            } while(second_digit==0);   //repeat until second digit is valid
            scan_keypad_rising_edge_sim();     // we call that for safety reasons


            if((first_digit=='4') && (second_digit)=='1')    // if the key is correct
            {
                cli();
                lcd_init_sim();
                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('4');
                                lcd_data_sim('1');   // display "WELCOME 41"

                                PORTB = (1 << PB7);  // turn on LED in PB7
                                _delay_ms(4000);     // for ~4sec
                                PORTB = (0 << PB7);  // turn off LED in PB7

                                flag_clear = 0;

                                lcd_init_sim();
                                TCNT1H = 0xfc;       // initialize timer1
                                TCNT1L = 0xf3;
                                sei();
                        }
                        else  // if the key is wrong
                        {
                                for(i=0; i<4; i++)   //repeat 4 times --> ~4sec
                                {
                                        z = PINB | 0x80; // turn on PB7
                                        PORTB = z;
                                        _delay_ms(500);      // for ~ 0.5sec
                                        z = PINB & 0x7f ;    // turn off PB7
                                        PORTB = z;
                                        _delay_ms(500);      // for ~ 0.5sec
                                }
                        }
                }
        }
```