ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εργαστήριο Μικροϋπολογιστών

$5^{η}$ εργαστηριακή άσκηση

Μικροελεγκτής AVR

Γεννήτρια Παραγωγής μιας μεταβαλλόμενης ηλεκτρικής τάσης

Αναστάσιος Λαγός - 03113531
Αντώνιος Δημήτριος Αλικάρης - 03118062

Γενική ιδέα

Χρησιμοποιούμε τον χρονιστή/μετρητή TIMER/COUNTER0 για να παράξουμε μία PWM κυματομορφή συχνότητας 4kHz. Επομένως, αρχικοποιούμε το prescaler στην τιμή των 8 και περιμένουμε να πατηθεί το πληκτρολόγιο για να αυξήσουμε ή να μειώσουμε την τιμή OCR0, δηλαδή το εύρος των παλμών. Επίσης, χρησιμοποιούμε και έναν χρονιστή όπου κάθε 100ms κάνει διακοπή προκειμένου να δώσει το σήμα μετατροπής του ADC.

Επίσης, σημειώνεται ότι οι συναρτήσεις για την χρήση της οθόνης LCD κρατήθηκαν σε κώδικα assembly, τις εισάγαμε στον κώδικα μας και τις προσθέσαμε ως αρχεία κατάληξης .S στον ίδιο φάκελο με την main συνάρτηση.

Κύριος Κώδικας σε c

```c
#undef F_CPU
#define F_CPU 8000000UL

#ifndef __DELAY_BACKWARD_COMPATIBLE__
#define __DELAY_BACKWARD_COMPATIBLE__
#endif

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

#define NOP(){__asm__ __volatile__("nop");} //assembly nop

/*Keypad Functions*/
unsigned int scan_row_sim(unsigned int row);
void scan_keypad_sim();
unsigned int scan_keypad_rising_edge_sim(unsigned int flick_time);
unsigned char keypad_to_ascii_sim();

unsigned int swap(unsigned int val);

/*Delay functions*/
void wait_usec(unsigned int delay);
void wait_msec(unsigned int delay);

/*LCD Assembly*/
//we kept the assembly code, given by the proffessors, for these lcd functions and we imported them from the .S files
extern void LCD_init();
extern void LCD_show(unsigned char cha);

/*Globals*/
unsigned int buttons[2], ram[2];
```

```
volatile unsigned int adc_output = 0;

/*function that delays for delay time*/
void wait_usec(unsigned int delay){
        unsigned int i;
        for(i = 0; i < (delay/10); i++) {    //10 usec delay for delay/10 times
                _delay_us(10);
        }
        if (delay % 10) {    //delay for the remainder accordingly
                _delay_us(delay % 10);
        }
}
/*same function as wait_usec but for milliseconds*/
void wait_msec(unsigned int delay) {
        unsigned int i ;
        for(i = 0; i < (delay / 10); i++){
                _delay_ms(10);
        }
        if(delay % 10) {
                _delay_ms(delay % 10);
        }
}
/*Function that does __asm__("swap")*/
unsigned int swap(unsigned int val) {
        return ((val & 0x0F) << 4 | (val & 0xF0) >> 4);
}

/*Function that returns which columns are pressed for a given row*/
unsigned int scan_row_sim(unsigned int row) {
        PORTC = row;//Search in line row.
        wait_usec(500); //Delay required for a successful remote operation

        NOP();
        NOP();//Delay to allow for a change of state

        return PINC & 0x0F; //Return 4 LSB
}
/*Function that scans the whole keypad*/
void scan_keypad_sim() {
        buttons[1] = swap(scan_row_sim(0x10));// A 3 2 1
        buttons[1] += scan_row_sim(0x20);// A 3 2 1 B 6 5 4

        buttons[0] = swap(scan_row_sim(0x40));// C 9 8 7
        buttons[0] += (scan_row_sim(0x80)); // C 9 8 7 D # 0 *

        PORTC = 0x00; // added only for the remote operation
        return;
}
```

```
/*Function that checks which buttons where pressed since its last call*/
unsigned int scan_keypad_rising_edge_sim(unsigned int flick_time) {
        scan_keypad_sim(); // do the first scan
        unsigned int temp[2]; // store first scan
        temp[0] = buttons[0];
        temp[1] = buttons[1];
        wait_msec(flick_time); //wait for flick time

        scan_keypad_sim();  //scan second time
        buttons[0] &= temp[0]; //remove flick values
        buttons[1] &= temp[1];

        temp[0] = ram[0];   //get the last state from previous call to
        ↪  rising_edge from "RAM"
        temp[1] = ram[1];
        ram[0] = buttons[0];    //update the new previous state
        ram[1] = buttons[1];
        buttons[0] &= ~temp[0]; //Keep values that change from 1 to 0
        buttons[1] &= ~temp[1];

        return (buttons[0] || buttons[1]);
}
/*Function that returns the ASCII code of button pressed*/
unsigned char keypad_to_ascii_sim() {
        unsigned int select;
        for(select = 0x01; select <= 0x80; select <<= 1) {
                switch(buttons[0] & select) {
                        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';
                }
        }
        for(select = 0x01; select <= 0x80; select <<= 1) {
                switch(buttons[1] & select) {
                        case 0x01:
```

```
                        return '4';
                        case 0x02:
                        return '5';
                        case 0x04:
                        return '6';
                        case 0x08:
                        return 'B';
                        case 0x10:
                        return '1';
                        case 0x20:
                        return '2';
                        case 0x40:
                        return '3';
                        case 0x80:
                        return 'A';
                }
        }
        return 0;
}

void PWM_init() {
        //set TMR0 in fast PWM mode with non-inverted output, prescale=8
        TCCR0 = (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS01);
        DDRB |= (1<<PB3);
}

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

ISR(TIMER1_OVF_vect){
        TCNT1 = 64755; //100ms
        ADCSRA |= (1<<ADSC);//adc start conversion enable
}

ISR(ADC_vect){
        adc_output = ADC;
}

//function which takes the ADC and shows in the lcd the converted voltage value
void convert_to_voltage_and_show(char adc_decimal_buffer[]) {
        float analog_input = (float) 5 * adc_output / 1024;

        char integer_part = (int) analog_input + '0';
        unsigned int decimal_part = (int) (analog_input * 100) % 100;
        itoa(decimal_part, adc_decimal_buffer, 10);

```

```
178         LCD_show(integer_part);
179         LCD_show('.');
180         LCD_show(adc_decimal_buffer[0]);
181         LCD_show(adc_decimal_buffer[1]);
182 
183         return;
184 }
185 
186 
187 int main(void)
188 {
189         DDRD = 0xFF;         //Initialize LCD as output
190         DDRC = 0xF0;         //Initialize PORTC and LEDs. Internal resistor pull
        ↪  up must be deactivated
191         unsigned int duty = 0;
192         char adc_decimal[2];
193         unsigned int old_output = -1;
194         ADC_init();
195         LCD_init();
196         PWM_init();
197         TIMSK = (1<<TOIE1);
198         TCCR1B = (1<<CS12)|(0<<CS11)|(1<<CS10);
199         TCNT1 = 64755; //100ms
200         sei();
201         while(1) {
202                 unsigned char button_pressed;
203                 ram[0] = 0, ram[1] = 0;
204                 while(1) {
205                         if(scan_keypad_rising_edge_sim(15)) {
206                                 button_pressed = keypad_to_ascii_sim();
207                                 break;
208                         }
209                         if (old_output != adc_output) { //if the adc hasn't
                        ↪  changed, dont show and compute the output again
210                                 old_output = adc_output;
211                                 LCD_init();
212                                 LCD_show('V');
213                                 LCD_show('o');
214                                 LCD_show('1');
215                                 LCD_show('\n');
216                                 convert_to_voltage_and_show(adc_decimal);
217                         }
218                 }
219                 if ((button_pressed == '1') && duty < 255) {
220                         duty++;//increasing the upper pulse
221                         OCR0 = duty;
222                 }
223                 else if((button_pressed == '2') && duty > 0){
```

```
                        duty--;//decreasing the upper pulse
                        OCR0 = duty;
                }
        }
}
```

Πρόσθετες συναρτήσεις σε assembly

LCD_init.S

```
#define _SFR_ASM_COMPAT 1
#define __SFR_OFFSET 0

#include <avr/io.h>

.global LCD_init

LCD_init:
        rcall lcd_init_sim
        ret

        wait_msec:
                push r24
                push r25
                ldi r24, lo8(1000)
                ldi r25, hi8(1000)
                rcall wait_usec
                pop r25
                pop r24
                sbiw r24, 1
                brne wait_msec

                ret

        wait_usec:
                sbiw r24, 1 //2 cycles
                nop
                nop
                nop
                nop
                brne wait_usec //1 cycle the majority of the time

                ret

        write_2_nibbles_sim:
        push r24
        push r25
        ldi r24 ,lo8(6000)
```

```
        ldi r25 ,hi8(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 ,lo8(6000)
        ldi r25 ,hi8(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
```

```
87          pop r25
88          pop r24
89          ret
90
91          lcd_init_sim:
92          push r24
93          push r25
94          ldi r24, 40
95          ldi r25, 0
96          rcall wait_msec
97          ldi r24, 0x30
98          out PORTD, r24
99          sbi PORTD, PD3
100         cbi PORTD, PD3
101         ldi r24, 39
102         ldi r25, 0
103         rcall wait_usec
104         push r24
105         push r25
106         ldi r24,lo8(1000)
107         ldi r25,hi8(1000)
108         rcall wait_usec
109         pop r25
110         pop r24
111         ldi r24, 0x30
112         out PORTD, r24
113         sbi PORTD, PD3
114         cbi PORTD, PD3
115         ldi r24,39
116         ldi r25,0
117         rcall wait_usec
118         push r24
119         push r25
120         ldi r24 ,lo8(1000)
121         ldi r25 ,hi8(1000)
122         rcall wait_usec
123         pop r25
124         pop r24
125         ldi r24,0x20
126         out PORTD, r24
127         sbi PORTD, PD3
128         cbi PORTD, PD3
129         ldi r24,39
130         ldi r25,0
131         rcall wait_usec
132         push r24
133         push r25
134         ldi r24 ,lo8(1000)
```

```
135         ldi r25 ,hi8(1000)
136         rcall wait_usec
137         pop r25
138         pop r24
139         ldi r24,0x28
140         rcall lcd_command_sim
141         ldi r24,0x0c
142         rcall lcd_command_sim
143         ldi r24,0x01
144         rcall lcd_command_sim
145         ldi r24, lo8(1530)
146         ldi r25, hi8(1530)
147         rcall wait_usec
148         ldi r24 ,0x06
149         rcall lcd_command_sim
150         pop r25
151         pop r24
152         ret
```

LCD_show.S

```
1  #define _SFR_ASM_COMPAT 1
2  #define __SFR_OFFSET 0
3
4  #include <avr/io.h>
5
6  .global LCD_show
7
8  LCD_show:
9          rcall lcd_data_sim
10         ret
11
12         wait_msec:
13                 push r24
14                 push r25
15                 ldi r24, lo8(1000)
16                 ldi r25, hi8(1000)
17                 rcall wait_usec
18                 pop r25
19                 pop r24
20                 sbiw r24, 1
21                 brne wait_msec
22
23                 ret
24
25         wait_usec:
26                 sbiw r24, 1 //2 cycles
27                 nop
```

```
28              nop
29              nop
30              nop
31              brne wait_usec //1 cycle the majority of the time
32
33              ret
34
35      write_2_nibbles_sim:
36      push r24
37      push r25
38      ldi r24 ,lo8(6000)
39      ldi r25 ,hi8(6000)
40      rcall wait_usec
41      pop r25
42      pop r24
43      push r24
44      in r25, PIND
45      andi r25, 0x0f
46      andi r24, 0xf0
47      add r24, r25
48      out PORTD, r24
49      sbi PORTD, PD3
50      cbi PORTD, PD3
51      push r24
52      push r25
53      ldi r24 ,lo8(6000)
54      ldi r25 ,hi8(6000)
55      rcall wait_usec
56      pop r25
57      pop r24
58      pop r24
59      swap r24
60      andi r24 ,0xf0
61      add r24, r25
62      out PORTD, r24
63      sbi PORTD, PD3
64      cbi PORTD, PD3
65      ret
66
67      lcd_data_sim:
68      push r24
69      push r25
70      sbi PORTD,PD2
71      rcall write_2_nibbles_sim
72      ldi r24,43
73      ldi r25,0
74      rcall wait_usec
75      pop r25
```

```
76          pop r24
77          ret
78
79          lcd_command_sim:
80          push r24
81          push r25
82          cbi PORTD, PD2
83          rcall write_2_nibbles_sim
84          ldi r24, 39
85          ldi r25, 0
86          rcall wait_usec
87          pop r25
88          pop r24
89          ret
90
91          lcd_init_sim:
92          push r24
93          push r25
94          ldi r24, 40
95          ldi r25, 0
96          rcall wait_msec
97          ldi r24, 0x30
98          out PORTD, r24
99          sbi PORTD, PD3
100         cbi PORTD, PD3
101         ldi r24, 39
102         ldi r25, 0
103         rcall wait_usec
104         push r24
105         push r25
106         ldi r24,lo8(1000)
107         ldi r25,hi8(1000)
108         rcall wait_usec
109         pop r25
110         pop r24
111         ldi r24, 0x30
112         out PORTD, r24
113         sbi PORTD, PD3
114         cbi PORTD, PD3
115         ldi r24,39
116         ldi r25,0
117         rcall wait_usec
118         push r24
119         push r25
120         ldi r24 ,lo8(1000)
121         ldi r25 ,hi8(1000)
122         rcall wait_usec
123         pop r25
```

```
124         pop r24
125         ldi r24,0x20
126         out PORTD, r24
127         sbi PORTD, PD3
128         cbi PORTD, PD3
129         ldi r24,39
130         ldi r25,0
131         rcall wait_usec
132         push r24
133         push r25
134         ldi r24 ,lo8(1000)
135         ldi r25 ,hi8(1000)
136         rcall wait_usec
137         pop r25
138         pop r24
139         ldi r24,0x28
140         rcall lcd_command_sim
141         ldi r24,0x0c
142         rcall lcd_command_sim
143         ldi r24,0x01
144         rcall lcd_command_sim
145         ldi r24, lo8(1530)
146         ldi r25, hi8(1530)
147         rcall wait_usec
148         ldi r24 ,0x06
149         rcall lcd_command_sim
150         pop r25
151         pop r24
152         ret
```