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

ΑΘΗΝΑ 20-01-2022

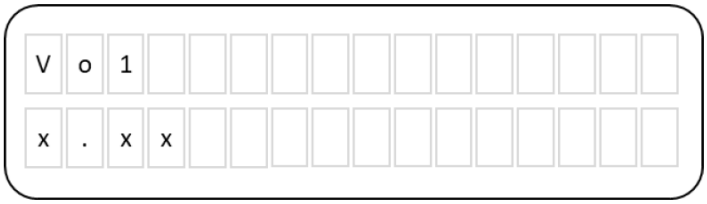
5η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ ''Εργαστήριο Μικροϋπολογιστών''
4η Εργ. Άσκ. στον Μικροελεγκτή AVR
Γεννήτρια παραγωγής μιας μεταβαλλόμενης ηλεκτρικής τάσης
(υλοποίηση στο εκπαιδευτικό σύστημα easyAVR6)

Ομάδα 41
Σιόκουρου Αιμιλία – 03117703
Πεγειώτη Νάταλυ – 03117707

Στην παρούσα άσκηση ζητείται να δημιουργηθεί κατάλληλος κώδικας σε γλώσσα C έτσι ώστε να αρχικοποιεί τον Timer/Counter1 για να παράγει μια PWM κυματομορφή με συχνότητα 4KHz.

Κάθε φορά που πιέζεται το πλήκτρο 1 αυξάνεται κατά 1 το Duty Cycle της κυματομορφής στον ακροδέκτη PB3. Κάθε φορά που πιέζεται το πλήκτρο μειώνεται κατά 1 το Duty Cycle της ίδιας κυματομορφής. Το Duty Cycle παίρνει τιμές από 0 έως 255.

Επίσης ο κώδικας αρχικοποιεί τον ενσωματωμένο ADC μετατροπέα για να διαβάζει την τιμή της τάσης στον ακροδέκτη PA0 και να την απεικονίζει στην LCD οθόνη με ακρίβεια δύο δεκαδικών ψηφίων, στη μορφή που φαίνεται στο παρακάτω σχήμα:



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

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

#define _OPEN_SYS_ITOA_EXT

int num_pressed, tmp,i;            // 16-bit number to store the key that was pressed
and tmp
char digit;// digits pressed
char x, y, z;
uint16_t adc,vout,vout1;


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
}
```

```
char keypad_to_ascii_sim(int x)    // function that makes the number pressed
{                                  //to the ascii value or 0
       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
	lcd_command_sim(0x06);	// activate automatic address increase by 1 and
deactivate screen sliding
}


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); //set PB3 pin as output
}



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


ISR(ADC_vect)
{
	vout = 0;
	char buffer[10];

	adc = ADCL|(ADCH<<8);
	vout=adc*4.88;		// 5/1024=4.88e^-3 *100 = 4.88
	vout1=vout;
	itoa(vout, buffer ,10);
```

```
        lcd_init_sim();
        lcd_data_sim('V');
        lcd_data_sim('o');
        lcd_data_sim('1');
        lcd_data_sim('\n');
        if (vout1<1000){
                lcd_data_sim('0');
                lcd_data_sim('.');
                lcd_data_sim(buffer[0]);
                lcd_data_sim(buffer[1]);
        }
        else if(vout1>999){
                lcd_data_sim(buffer[0]);
                lcd_data_sim('.');
                lcd_data_sim(buffer[1]);
                lcd_data_sim(buffer[2]);
        }


        //reset the timer
        TCNT1H = 0xFC;
        TCNT1L = 0xF3;
        return;
}



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



int main(void)
{
        unsigned char duty=0;

        lcd_init_sim();
        PWM_init();

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

        //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)
        {
                ADCSRA |= (1<<ADSC);
                do{
                        // read and store 1st digit
                        num_pressed = scan_keypad_rising_edge_sim();
                        digit = keypad_to_ascii_sim(num_pressed);
                } while(digit==0);   //repeat until first digit is valid
```

```
                scan_keypad_rising_edge_sim(); //called one more time as instructed

                if (digit=='1'){
                        cli();
                        if (duty==255){

                                duty=0;
                                OCR0=duty;
                                _delay_ms(8);

                        }
                        else{
                                duty++;
                                OCR0=duty;
                                _delay_ms(8);
                        }
                        sei();
                }
                else if(digit=='2'){
                        cli();
                        if (duty=='0'){
                                cli();
                                duty=255;
                                OCR0=duty;
                                _delay_ms(8);
                        }
                        else{

                                duty--;
                                OCR0=duty;
                                _delay_ms(8);
                        }
                        sei();
                }
        }
}
```