Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Ακαδημαϊκό Έτος 2020-2021

Εξάμηνο 7ο

Εργαστήριο Μικροϋπολογιστών

2η Εργαστηριακή Αναφορά

Κωστάκη Χριστίνα (el18136)
Τσάφος Αλέξανδρος (el18211)

Ζήτημα 5.1

```
#define F_CPU 8000000UL // FREQUENCY OF ATMEGA16
#define RS PD2
#define E PD3

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

char ram[2], key[2], digit;

// Scan a row of the keypad for input
// input: row of choice
// output: row's status
char scan_row(char c)
{
	PORTC = c;
	_delay_us(500);
	asm("nop");
	asm("nop");
	return (PINC & 0x0f);
}

// swap the 4 MSB with the 4 LSB of a variable
char swap(char word)
{
	return ((word & 0x0f) << 4 | (word & 0xf0) >> 4);
}

// scan the whole keypad's status.
// input: none
// output: none
// The keypad's status is stored in key[1] and key[2]
void scan_keypad()
{
	char ret;

	ret = scan_row(0x10); // 1st line
	key[1] = swap(ret);

	ret = scan_row(0x20); // 2nd line
	key[1] += ret;

	ret = scan_row(0x40); // 3rd line
```

```
	key[0] = swap(ret);

	ret = scan_row(0x80); // 4th line
	key[0] += ret;
}

// scan the keypad for recently pressed buttons
// input: none
// output: none
void scan_keypad_rising_edge()
{
	char ret[2];

	scan_keypad(); // scan and store
	ret[0] = key[0];
	ret[1] = key[1];

	_delay_ms(15); // prevent sparkling

	scan_keypad();

	key[0] &= ret[0]; // check if the button is indeed pressed
	key[1] &= ret[1];

	ret[0] = ram[0]; // restore the last call's pressed buttons
	ret[1] = ram[1];

	ram[0] = key[0]; // store this call's pressed buttons
	ram[1] = key[1];

	key[0] &= ~ret[0]; // check if the button is newly pressed
	key[1] &= ~ret[1];
}

// match the button pressed, to it's ascii char,
// according to the manual
char keypad_to_ascii()
{
	if (key[0] & 0x01)
	return '*';

	if (key[0] & 0x02)
	return '0';

	if (key[0] & 0x04)
	return '#';
```

```
      if (key[0] & 0x08)
      return 'D';

      if (key[0] & 0x10)
      return '7';

      if (key[0] & 0x20)
      return '8';

      if (key[0] & 0x40)
      return '9';

      if (key[0] & 0x80)
      return 'C';

      if (key[1] & 0x01)
      return '4';

      if (key[1] & 0x02)
      return '5';

      if (key[1] & 0x04)
      return '6';

      if (key[1] & 0x08)
      return 'B';

      if (key[1] & 0x10)
      return '1';

      if (key[1] & 0x20)
      return '2';

      if (key[1] & 0x40)
      return '3';

      if (key[1] & 0x80)
      return 'A';

      return 0;
}

void lcd_cmd_half_byte(unsigned char cmd)
{
      unsigned char tmp;
```

```
	/*Set RS = 0, E = 1, send the hi 4 bits*/
	tmp = (cmd & ~(1<<RS)) | (1<<E);
	PORTD = tmp;
	asm("nop");
	PORTD = tmp & ~(1<<E);
	asm("nop");
}

void lcd_cmd_full_byte(unsigned char cmd)
{
	unsigned char tmp;

	/*send the first 4 bits*/
	tmp = (cmd & ~(1<<RS)) | (1<<E);
	PORTD = tmp;
	asm("nop");
	PORTD = tmp & ~(1<<E);
	asm("nop");
	_delay_us(50);

	/*send the second 4 bits*/
	tmp = ((cmd<<4) & ~(1<<RS)) | (1<<E);
	PORTD = tmp;
	asm("nop");
	PORTD = tmp & ~(1<<E);
	asm("nop");
	_delay_us(50);
}


void lcd_clr(void)
{
	/*return home an address 00 on LCD*/
	lcd_cmd_full_byte(0b10000000);
	_delay_ms(1);
	/*Clear the display*/
	lcd_cmd_full_byte(0b00000001);
}

void lcd_putchar(unsigned char data)
{
	unsigned char tmp;

	tmp = (data | (1<<RS)) | (1<<E);
	PORTD = tmp;
```

```
	asm("nop");
	PORTD = tmp & ~(1<<E);
	asm("nop");
	_delay_ms(2);

	tmp = ((data<<4) | (1<<RS)) | (1<<E);
	PORTD = tmp;
	asm("nop");
	PORTD = tmp & ~(1<<E);
	asm("nop");
	_delay_ms(2);
}

void lcd_str(char *s)
{
	while(*s)
	lcd_putchar(*s++);
}

void lcd_init(void)
{

	/* First function set */
	lcd_cmd_half_byte(0b00110000);
	_delay_ms(5);
	/* Second function set */
	lcd_cmd_half_byte(0b00110000);
	_delay_ms(100);
	/* Third function set */
	lcd_cmd_half_byte(0b00110000);
	_delay_ms(2);
	/* Last function set */
	lcd_cmd_half_byte(0b00100000);
	_delay_ms(2);

	/* Function set */
	lcd_cmd_full_byte(0x28);
	/* Display on/off */
	lcd_cmd_full_byte(0b00001100);
	/* Clear display */
	lcd_cmd_full_byte(0b00000001);
	/* Entry mode */
	lcd_cmd_full_byte(0x06);
}

void ADC_init(void) // Initialize ADC
```

```
{
	ADMUX = 0x40;
	ADCSRA = (1 << ADEN | 1 << ADIE | 1 << ADPS2 | 1 << ADPS1 | 1 <<
ADPS0);
}

ISR(ADC_vect)
{
	//after printing just the ADC value, its range is 0-175
	//that is for EasyAVR6 sim only. The ADC slider must be set at 0
	//the range changes as the slider changes
	lcd_clr();
	volatile float Vin = (ADC*5.0)/176; //should be 1024 irl
	Vin = Vin*100;
	unsigned char str[10];
	int VinDigit[3];
	VinDigit[0] = (int)Vin/100;
	VinDigit[1] = (int)(Vin - VinDigit[0]*100)/10;
	VinDigit[2] = (int)Vin%10;
	sprintf(str, "%d.%d%d", VinDigit[0], VinDigit[1], VinDigit[2]);

	lcd_str("Vo1 \n");
	lcd_str(str);
}

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){
//interrupt every 250 ms, allow ADC to convert and display
	ADCSRA |= (1<<ADSC); //allow ADC to interrupt and convert
	TCNT1 = 63685; //reset the Timer
	TCCR1B = 0x05; //flags

}

int main(void)
{
	DDRD = 0xff; // output (lcd)
	DDRC = 0xf0; // input and output (keyboard)

	unsigned char DC = 127; // set initial DC value to 0.5
```

```
	PWM_init(); // initialize timer0 for the pwm generator
	ADC_init();
	lcd_init();

	//initialize timer 1

	TIMSK = (1<<TOIE1); //TOIE1
	TCCR1B = 0x05;
	TCNT1 = 63685; //Timer set to 250ms

	asm("sei");

	while (1)
	{
		OCR0 = DC;
		_delay_us(40);

		while (1)
		{ // wait for a change
			scan_keypad_rising_edge();
			if ((digit = keypad_to_ascii()) != 0)
			break;
		}
		switch (digit)
		{
			case '1':
			if (DC == 255) // cannot decrease further
			continue;
			else
			DC++; //increase the duty cycle
			break;
			case '2':
			if (DC == 0) // cannot increase further
			continue;
			else
			DC--; //decrease the duty cycle
			default:
			continue;
		}
	}
}
```