3η εργαστηριακή άσκηση

Φωτεινή Κυριακοπούλου 03120182
Ιωάννης Μπουφίδης 03120162

Ζήτημα 3.1

```
.include "m328PBdef.inc"

values: .db 5, 26, 46, 66, 87, 107, 128, 148, 168, 189, 209, 230, 250

.def DC_VALUE = r18

init:
    ldi r24, low(RAMEND)
    out SPL, r24
    ldi r24, high(RAMEND)
    out SPH, r24

    ; initialize TMR1A for Fast PWM, 8-bit, prescaler = 8
    ldi r24, (1<<WGM10) | (1<<COM1A1)
    sts TCCR1A, r24
    ldi r24, (1<<WGM12) | (1<<CS11)       ; non-inverting mode
    sts TCCR1B, r24

    clr r24
    out DDRD, r24                         ; set PORTD as input
    ser r24
    out DDRB, r24                         ; set PORTB as output

    ; load the memory address of the array
    ldi ZL, low(values)
    ldi ZH, high(values)
    adiw ZL, 6                            ; offset = 6 (50% duty cycle - 128)
    lpm DC_VALUE, Z                       ; initialize DC_VALUE = 128
main:
    ldi r24, low(16*100)
    ldi r25, high(16*100)                 ; set delay (100 ms)

    sbis PIND, 1                          ; if PD1 is pressed
    rjmp increase                         ; increase duty cycle
    sbis PIND, 2                          ; if PD2 is pressed
    rjmp decrease                         ; decrease duty cycle
output:
    sts OCR1AL, DC_VALUE                  ; pwm
    rcall delay_mS                        ; delay to avoid debounce
    rjmp main                             ; repeat

increase:
    cpi DC_VALUE,250                      ; duty cycle doesn't exceed 98%
    breq inc_exit
    adiw ZL, 1                            ; next element of array
    lpm DC_VALUE, Z                       ; update DC_VALUE
```

```
inc_exit:
    rjmp output

decrease:
    cpi DC_VALUE,5                      ; duty cycle isn't below 2%
    breq dec_exit
    sbiw ZL, 1                          ; previous element of array
    lpm DC_VALUE, Z
dec_exit:
    rjmp output

; delay routine
delay_mS:
    ldi r23, 249
loop_inn:
    dec r23
    nop
    brne loop_inn

    sbiw r24, 1
    brne delay_mS

    ret
```

Ζήτημα 3.2

```
#define F_CPU 16000000UL
#include "avr/io.h"
#include <util/delay.h>

int main() {
    unsigned char DC_VALUE = 128;
    // array of duty cycles
    unsigned char values[] = {5, 26, 46, 66, 87, 107, 128, 148, 168, 189, 209, 230,
250};
    unsigned int index = 6; // index at current duty cycle
    uint16_t adc_value1, adc_value2, adc_value; // adc conversion result
    float Vadc; // corresponding voltage

    // set voltage reference to AVCC and select ADC1 channel
    ADMUX = (1 << REFS0) | (1 << MUX0);

    // enable ADC and set prescaler to 128 for maximum resolution
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

    // initialize TMR1A in Fast PWM, 8-bit, non-inverting mode
    // and prescale = 8
    TCCR1A = (1 << WGM10) | (1 << COM1A1);
    TCCR1B = (1 << WGM12) | (1 << CS11);

    DDRD = 0xFF; // led output
    DDRB |= 0b00111111; // pwm output
    DDRC &= ~((1 << PC3) | (1 << PC4)); // input from PC3 and PC4

    PORTD = 0x00;
    while (1) {
        if (!(PINC & (1 << PINC3))) { //check if PC3 is pressed
         if (index < 12) {
             index++;
             DC_VALUE = values[index]; //increase duty cycle
         }
        }
        else if (!(PINC & (1 << PINC4))) { //check if PC4 is pressed
            if (index > 0) {
             index--;
             DC_VALUE = values[index]; //decrease duty cycle
         }
        }
        OCR1AL = DC_VALUE;

        // start ADC conversion
        ADCSRA |= (1 << ADSC);
        // wait for conversion to complete
        while (ADCSRA & (1 << ADSC));
```

```
        // combine ADCL and ADCH to get the 10-bit result
        adc_value1 = ADCL | (ADCH << 8);

        ADCSRA |= (1 << ADSC);
        while (ADCSRA & (1 << ADSC));
        adc_value2 = ADCL | (ADCH << 8);

        // get mean value of two different conversions for better accuracy
        adc_value = (adc_value1 + adc_value2)/2;

        Vadc = (adc_value*5.0)/1023.0; // scale the digital ADC to a voltage level

        // display LED based on ADC value
        if (Vadc <= 0.625) {
             PORTD = (1 << PD0);
        } else if (Vadc <= 1.25) {
            PORTD = (1 << PD1);
        } else if (Vadc <= 1.875) {
            PORTD = (1 << PD2);
        } else if (Vadc <= 2.5) {
            PORTD = (1 << PD3);
        } else if (Vadc <= 3.125) {
            PORTD = (1 << PD4);
        } else if (Vadc <= 3.75) {
            PORTD = (1 << PD5);
        } else if (Vadc <= 4.375) {
            PORTD = (1 << PD6);
        } else {
            PORTD = (1 << PD7);
        }
        _delay_ms(100);
    }
    return 0;
}
```

Ζήτημα 3.3

```
#define F_CPU 16000000UL
#include "avr/io.h"
#include <util/delay.h>

int main() {
    unsigned char DC_VALUE = 128;
    unsigned char values[] = {5, 26, 46, 66, 87, 107, 128, 148, 168, 189, 209, 230,
250};
    unsigned int index = 6;
    unsigned char mode = 0;
    uint16_t pot;

    // set voltage reference to AVCC and select ADC0 channel
    ADMUX = (1 << REFS0);

    // enable ADC and set prescaler to 128
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

    // initialize TMR1A in Fast PWM, 8-bit, non-inverting mode
    // and prescale = 8
    TCCR1A = (1 << WGM10) | (1 << COM1A1);
    TCCR1B = (1 << WGM12) | (1 << CS11);

    // initialize PORTB as output and PORTD as input
    DDRB |= 0b00111111;
    DDRD &= ~((1 << PD1) | (1 << PD2) | (1 << PD6) | (1 << PD7));

    while (1) {
        // if PD6 is pressed activate mode1
        if (!(PIND & (1 << PIND6))) mode = 1;
        // else if PD7 is pressed activate mode2
        else if (!(PIND & (1 << PIND7))) mode = 2;

        if (mode == 1) {
            if (!(PIND & (1 << PIND1))) { // check if PD1 is pressed
                if (index < 12) {
                    index++;
                    DC_VALUE = values[index]; // increase duty cycle
                }
                _delay_ms(200);
            }
            else if (!(PIND & (1 << PIND2))) { // check if PD2 is pressed
                if (index > 0) {
                    index--;
                    DC_VALUE = values[index]; // decrease duty cycle
                }
                _delay_ms(200);
            }
```

```
        }
        else if (mode == 2) {
            // start an ADC conversion and wait for it to finish
            // if ADSC is set, a conversion is in progress
            ADCSRA |= (1 << ADSC);
            while (ADCSRA & (1 << ADSC));
            // collect the conversion's 10-bit result
            pot = ADCL | (ADCH << 8);
            // and transform it to a Duty Cycle value for 8-bit Fast PWM
            DC_VALUE = (pot*255) / 1023;
        }
        // update the Duty Cycle value
        OCR1AL = DC_VALUE;
    }
    return 0;
}
```