Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2022-2023

ΑΘΗΝΑ, 3 Νοεμβρίου 2022

3η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"
Χρήση εξωτερικών διακοπών στον Μικροελεγκτή AVR

Αναφορά 3ης Εργαστηριακής Άσκησης

Ραπτόπουλος Πέτρος (el19145)
Σαφός Κωνσταντίνος (el19172)

Ζήτημα 3.1:

Παρακάτω φαίνεται ο κώδικας σε Assembly και C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

Σημείωση: Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

Κώδικας σε Assembly

```
.include "m328PBdef.inc"    ; ATmega328P microcontroller definitions

.org 0x0
rjmp reset                  ; reset
.org 0x4
rjmp ISR1                   ; jump to INT1's interrupt routine
.org 0x1A
rjmp ISR_TIMER1_OVF         ; jump to timer's interrupt routine

.equ FOSC_MHZ = 16          ; Microcontroller operating frequency in MHz
.equ DEL_BOUNCE_mS = 5      ; Delay in mS for bouncing effect
.equ DEL_mS_ALL_LEDS = 500
.equ DEL_NU_BOUNCE=FOSC_MHZ*DEL_BOUNCE_mS       ; delay_mS routine: (1000*DEL_NU+6) cycles
.equ DEL_NU_ALL_LEDS=FOSC_MHZ*DEL_mS_ALL_LEDS

.equ TIMER_DELAY_S = 4                          ; we want a delay of 4 sec
.equ OVERFLOW_POINT = 65535                     ; overflow point of timer (16 bits)
.equ FOSC_HZ = 16000000                         ; Microcontroller operating frequency in Hz
.equ TIMER_FREQ = FOSC_HZ/1024                  ; cause of the value of TCCR1B set below
.equ DEL_NU_TIMER=OVERFLOW_POINT - TIMER_FREQ*TIMER_DELAY_S ; find the total cycles of timer's delay
; when an overflow takes place an timer interrupt is fired

.DEF temp=r22                       ; define temporary register
.DEF flag=r21                       ; flag activated if first interrupt detected

.MACRO Leds                         ; define leds macro

    ; Set frequency of timer's increase to fclock/1024
    ldi temp, (1<<CS12) | (0<<CS11) | (1<<CS10)
    sts TCCR1B, temp

    ; set timer to 4 sec
    ldi temp, HIGH(DEL_NU_TIMER)
    sts TCNT1H, temp
    ldi temp, LOW(DEL_NU_TIMER)
    sts TCNT1L, temp

    cpi flag, 0x00
    breq first_Interrupt

    ldi temp, 0xFF
    out PORTB, temp                 ;light up all leds
    ldi r24, low(DEL_NU_ALL_LEDS)
    ldi r25, high(DEL_NU_ALL_LEDS)
    rcall delay_mS                  ; delay 0.5 sec

first_Interrupt:
    inc flag
    ldi temp, 0x01
    out PORTB, temp                 ;let there be light

.ENDMACRO
```

```
reset:
    ldi temp, low(RAMEND)    ;Initialize stack pointer
    out SPL, temp
    ldi temp, high(RAMEND)
    out SPH, temp

    ; Interrupt on rising edge of INT1 pin
    ldi temp, (1 << ISC11) | (1 << ISC10)
    sts EICRA, temp

    ; Enable the INT1 interrupt (PD3)
    ldi temp, (1 << INT1)
    out EIMSK, temp

    ; Init PORTD as input
    clr temp
    out DDRD, temp

    ; Init PORTC as input
    clr temp
    out DDRC, temp

    ; Init PORTB as output
    ser temp
    out DDRB, temp

    ; Enable TCNT1 overflow timer interrupt
    ldi temp, (1<<TOIE1)
    sts TIMSK1, temp

    ; Stop the time counter
    ldi temp, (0<<CS12) | (0<<CS11) | (0<<CS10)
    sts TCCR1B, temp

    ldi flag, 0x00          ; init flag
    sei                     ; enable interrupts

main:
    in temp, PINC           ; keep the state of PINC
    andi temp, 0x20         ; isolate PC5
    cpi temp, 0x00          ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    brne main               ; if temp != 0 then continue looping
                            ; else the button is pressed
BTN_PRESSED:                ; while the button is pressed or bouncing loop
    ldi r24, low(DEL_NU_BOUNCE)
    ldi r25, high(DEL_NU_BOUNCE)
    rcall delay_mS          ; delay to overcome bouncing
    in temp, PINC           ; fetch the state of PINC
    andi temp, 0x20         ; isolate PC5
    cpi temp, 0x00          ; compare with zero
    breq BTN_PRESSED        ; if still pressed loop
    Leds                    ; otherwise call Leds macro
    rjmp main               ; jump again to main

; delay of 1000*F1 + 6 cycles (almost equal to 1000*F1 cycles)
delay_mS:
; total delay of next 4 instruction group = 1+(249*4-1) = 996 cycles
    ldi r23, 249
loop_inn:
    dec r23 ; 1 cycle
    nop             ; 1 cycle
    brne loop_inn    ; 1 or 2 cycles
    sbiw r24, 1      ; 2 cycles
    brne delay_mS   ; 1 or 2 cycles
    ret              ; 4 cycles
```

```
ISR1:             ; interrupt routine
                  ; for INT1
    push r25      ; keep delay registers
    push r24
    push temp     ; save temp
    in temp, SREG ; save SREG
    push temp

bouncing:
    ldi temp, (1<<INTF1) ; set interrupt
    out EIFR, temp       ; flag to 0
    ldi r24, low(DEL_NU_BOUNCE)
    ldi r25, high(DEL_NU_BOUNCE)
    rcall delay_mS
    in temp, EIFR
    andi temp, 0x02
    cpi temp, 0x02
    breq bouncing

    Leds          ; call macro

    pop temp
    out SREG, temp  ; restore SREG
    pop temp        ; restore temp
    pop r24
    pop r25         ; restore delay registers
    reti            ; return from interrupt

ISR_TIMER1_OVF:
    push r25        ; keep delay registers
    push r24
    push temp       ; save temp
    in temp, SREG   ; save SREG
    push temp

    ; Stop the time counter
    ldi temp, (0<<CS12) | (0<<CS11) | (0<<CS10)
    sts TCCR1B, temp

    ldi temp, 0x00
    out PORTB, temp ;lights off

    ldi flag, 0x00  ; set flag to zero

    pop temp
    out SREG, temp   ; restore SREG
    pop temp         ; restore temp
    pop r24
    pop r25          ; restore delay registers
    reti             ; return from interrupt
```

Κώδικας σε C

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

#define TIMER_DELAY_S  4                                        // we want a delay of 4 sec
#define OVERFLOW_POINT 65535                                    // overflow point of timer (16 bits)
#define TIMER_FREQ F_CPU/1024                                   // cause of the value of TCCR1B set below
#define DEL_NU_TIMER OVERFLOW_POINT - TIMER_FREQ*TIMER_DELAY_S // find the total cycles of timer's delay
// when an overflow takes place an timer interrupt is fired

long int flag; // consecutive interrupt counter

void leds() {

    TCCR1B = (1<<CS12) | (0<<CS11) | (1<<CS10);
    // Set frequency of timer's increase to fclock/1024

    // set timer to 4 sec
    TCNT1 = DEL_NU_TIMER;
    if (flag++ != 0) {              // check the interrupt counter and increase
        PORTB = 0xFF;               // light up all the leds, that's not the first interrupt
        _delay_ms(500);             // delay for 0.5 sec
    }
    PORTB = 0x01;                  // light up the lamp
}

ISR (INT1_vect) {
    do {                    // bouncing effect
        EIFR = (1<<INTF1); // clear interrupt pin
        _delay_ms(100);    // delay a small amount of time
    } while((EIFR & 0x02) == 0x02); // check if interrupt pin
                                    // is changed
    leds();
}

ISR (TIMER1_OVF_vect) {
    // Stop the time counter
    TCCR1B = (0<<CS12) | (0<<CS11) | (0<<CS10);
    PORTB = 0x00;   // lights off
    flag = 0;       // interrupts are handled
}

int main() {
    // Interrupt on rising edge of INT1 pin
    EICRA = (1 << ISC11) | (1 << ISC10);
    // Enable the INT1 interrupt (PD3)
    EIMSK = (1 << INT1);
    DDRB = 0xFF;    // Set PORTB as output
    DDRC = 0x00;    // Set PORTC as input
    DDRD = 0x00;    // Set PORTD as input
    TIMSK1 = (1<<TOIE1); // Enable TCNT1 overflow timer interrupt
    TCCR1B = (0<<CS12) | (0<<CS11) | (0<<CS10); // Stop the time counter
    flag = 0;
    sei();  // Enable global interrupts

    while(1) {                      // loop for ever
        if((PINC & 0x20) == 0x00) { // check if button pressed (logical 0)
            do { _delay_ms(100); }  // bouncing or button unpressed
            while((PINC & 0x20) == 0x00);
            leds();
        }
    }
}
```

Ζήτημα 3.2:

Παρακάτω φαίνεται ο κώδικας σε Assembly και C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

Σημείωση: Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

Κώδικας σε Assembly

```
.include "m328PBdef.inc"

.org 0x0
rjmp reset

.equ FOSC_MHZ = 16                      ; Microcontroller operating frequency in MHz
.equ DEL_BOUNCE_mS = 5                  ; Delay in mS for bouncing effect
.equ DEL_NU_BOUNCE=FOSC_MHZ*DEL_BOUNCE_mS ; delay_mS routine: (1000*DEL_NU+6) cycles

.DEF temp=r22
.DEF counter=r21
.DEF lpmReg=r0

reset:
    ldi temp, LOW(RAMEND) ;Initialize stack pointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

; Init PORTB as output
    ser temp
    out DDRB, temp

; Init PORTD as input
    clr temp
    out DDRD, temp

PWM:
    ldi temp, (1<<WGM10) | (1<<COM1A1)
    sts TCCR1A, temp

    ldi temp, (1<<WGM12) | (1<<CS11)
    sts TCCR1B, temp
    ; The above values are for fast PWM

    ldi zl, low(Table*2+6)
    ldi zh, high(Table*2+6)
    ; the double register Z keeps table's
    ; address stored in program memory
    ; each byte represent the DC that corresponds
    ; to each counter value
    ; Z initialized to Table*2+6 in order to point
    ; to 50% DC (PWM)

    ; for 8-bit PWM we have max value = 0xFF (255)
    ldi counter, 0x06       ; init counter to 50% DC
    ldi temp, high(127)     ; for 50% DC 255/2
    sts OCR1AH, temp
    ldi temp, low(127)      ; for 50% DC 255/2
    sts OCR1AL, temp
```

```
main:
    in temp, PIND    ; keep the state of PIND
    andi temp, 0x02  ; isolate PD1
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD1 ; if temp != 0 then continue looping

    in temp, PIND    ; keep the state of PIND
    andi temp, 0x04  ; isolate PD2
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD2 ; if temp != 0 then continue loopin

    rjmp main

PressD1:
    ldi r24, low(DEL_NU_BOUNCE)
    ldi r25, high(DEL_NU_BOUNCE)
    rcall delay_mS   ; delay to overcome bouncing
    in temp, PIND    ; keep the state of PIND
    andi temp, 0x02  ; isolate PD1
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD1     ; if the button is still pressed wait

    ; button unpressed, proceed
    cpi counter, 0      ; DC=50+(6*8)=98%, #table = 13 elements
    breq main           ; if at 98% dont decrease more

    dec counter             ; otherwise decrease counter


    sbiw zl, 1              ; decrease Z register one byte
                            ; to access next DC value
    lpm                     ; r0 <-- memory

    clr temp
    sts OCR1AH, temp        ; values are from 0 to 255
    sts OCR1AL, lpmReg      ; set the compare level
    rjmp main               ; continue looping

PressD2:
    ldi r24, low(DEL_NU_BOUNCE)
    ldi r25, high(DEL_NU_BOUNCE)
    rcall delay_mS          ; delay to overcome bouncing
    in temp, PIND           ; keep the state of PIND
    andi temp, 0x04         ; isolate PD2
    cpi temp, 0x00          ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD2            ; if the button is still pressed wait

    ; button unpressed, proceed
    cpi counter, 12         ; if at 2% DC dont increase more
    breq main

    inc counter             ; otherwise increase counter

    adiw zl, 1              ; increase Z register one byte
                            ; to access previous DC value
    lpm                     ;r0 <-- memory

    clr temp
    sts OCR1AH, temp        ; values are from 0 to 255
    sts OCR1AL, lpmReg      ; set the compare level
    rjmp main
```

```
; delay of 1000*F1 + 6 cycles (almost equal to 1000*F1 cycles)
delay_mS:
; total delay of next 4 instruction group = 1+(249*4-1) = 996 cycles
    ldi r23, 249
loop_inn:
    dec r23          ; 1 cycle
    nop              ; 1 cycle
    brne loop_inn    ; 1 or 2 cycles
    sbiw r24, 1      ; 2 cycles
    brne delay_mS    ; 1 or 2 cycles
    ret              ; 4 cycles

Table:
; Non-Inverted PWM, formula for OCR1AL: hex(round(maxValue-maxValue*DC))
; DCs start from 2% to 98% increasing by 8%
.DW 0xE5FA, 0xBCD1, 0x94A8, 0x6B7F, 0x4257, 0x192E, 0x0005
```

Κώδικας σε C

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

#define maxValue 255
#define step 0.08
#define start 6 // 50% PWM

int main() {
    DDRB = 0xFF;                          // Set PORTB as output
    DDRD = 0x00;                          // Set PORTD as input
    TCCR1A = (1<<WGM10) | (1<<COM1A1);    // Init control register A of Timer 1
    TCCR1B = (1<<WGM12) | (1<<CS11);      // Init control register B of Timer 1
    // with the above values we have fast PWM mode

    char table[13] = {0xFA, 0xE5, 0xD1, 0xBC, 0xA8, 0x94, 0x7F, 0x6B, 0x57, 0x42, 0x2E, 0x19, 0x05};
    // example OCR1A value for 98% is table[0]
    // keeps the values of OCR1A for desired DCs
    // type == char cause we want 1 byte values

    int counter = start;             // keeps the index of table
    OCR1AH = 0x00;                   // values from 0 to 255
    OCR1AL = table[counter];        // init PWM

    while(1) {                       // loop forever
        if((PIND & 0x02) == 0x00) { // check if button pressed (logical 0)
            do {
                _delay_ms(5);
            } while((PIND & 0x02) == 0x00); // while being pressed wait
            if(counter == 0) continue;       // dont decrease counter if DC = 98%
            OCR1AH = 0x00;                   // values from 0 to 255
            OCR1AL = table[--counter];       // decrease counter and fetch OCR1AL value
        }
        if((PIND & 0x04) == 0x00) {         // check if button pressed (logical 0)
            do {
                _delay_ms(5);
            } while((PIND & 0x04) == 0x00); // while being pressed wait
            if(counter == 12) continue;     // dont increase counter if DC = 2%
            OCR1AH = 0x00;                  // values from 0 to 255
            OCR1AL = table[++counter];      // increase counter and fetch OCR1AL value
        }
    }
}
```

Ζήτημα 3.3:

Παρακάτω φαίνεται ο κώδικας σε Assembly και C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

Σημείωση: Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

Κώδικας σε Assembly

```
.include "m328PBdef.inc"

.org 0x0
rjmp reset

.DEF temp=r22

reset:
    ldi temp, LOW(RAMEND) ;Initialize stack pointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

; Init PORTB as output
    ser temp
    out DDRB, temp

; Init PORTD as input
    clr temp
    out DDRD, temp

PWM:
    ldi temp, (0<<WGM10) | (1<<WGM11) | (1<<COM1A1)
    sts TCCR1A, temp
    ; The above values are for fast PWM with TOP = ICR1

main:

    ldi temp, (1<<WGM12) | (1<<WGM13) | (0<<CS12) | (0<<CS11) | (0<<CS10)
    ; we stop PWM setting timer's frequency to zero
    sts TCCR1B, temp

    in temp, PIND    ; keep the state of PIND
    andi temp, 0x01  ; isolate PD0
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD0 ; if temp != 0 then continue looping

    in temp, PIND    ; keep the state of PIND
    andi temp, 0x02  ; isolate PD1
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD1 ; if temp != 0 then continue looping

    in temp, PIND    ; keep the state of PIND
    andi temp, 0x04  ; isolate PD2
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD2 ; if temp != 0 then continue loopin

    in temp, PIND    ; keep the state of PIND
    andi temp, 0x08  ; isolate PD2
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD3 ; if temp != 0 then continue loopin

    rjmp main
```

```
; formula: top(fpwm, N) = fclk/(N*fpwm) - 1

PressD0: ; PWM with 125Hz
    ldi temp, (1<<WGM12) | (1<<WGM13) | (1<<CS12) | (0<<CS11) | (1<<CS10)
    ; N = 1024
    sts TCCR1B, temp

    clr temp
    sts OCR1AH, temp ; values are from 0 to 255
    ldi temp, 62
    sts OCR1AL, temp ; set the compare level

    clr temp
    sts ICR1H, temp  ; values are from 0 to 255
    ldi temp, 124
    sts ICR1L, temp  ; set the top level

    in temp, PIND    ; keep the state of PIND
    andi temp, 0x01  ; isolate PD0
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD0     ; if the button is still pressed wait
    rjmp main        ; continue looping


PressD1: ; PWM with 250Hz
    ldi temp, (1<<WGM12) | (1<<WGM13) | (1<<CS12) | (0<<CS11) | (0<<CS10)
    ; N = 256
    sts TCCR1B, temp

    clr temp
    sts OCR1AH, temp ; values are from 0 to 255
    ldi temp, 124
    sts OCR1AL, temp ; set the compare level

    clr temp
    sts ICR1H, temp  ; values are from 0 to 255
    ldi temp, 249
    sts ICR1L, temp  ; set the top level

    in temp, PIND    ; keep the state of PIND
    andi temp, 0x02  ; isolate PD1
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD1     ; if the button is still pressed wait
    rjmp main        ; continue looping

PressD2: ; PWM with 500Hz
    ldi temp, (1<<WGM12) | (1<<WGM13) | (1<<CS12) | (0<<CS11) | (0<<CS10)
    ; N = 256
    sts TCCR1B, temp

    clr temp
    sts OCR1AH, temp ; values are from 0 to 255
    ldi temp, 62
    sts OCR1AL, temp ; set the compare level

    clr temp
    sts ICR1H, temp  ; values are from 0 to 255
    ldi temp, 124
    sts ICR1L, temp  ; set the top level

    in temp, PIND    ; keep the state of PIND
    andi temp, 0x04  ; isolate PD2
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD2     ; if the button is still pressed wait
    rjmp main        ; continue looping
```

```
PressD3: ; PWM with 1000Hz
    ldi temp, (1<<WGM12) | (1<<WGM13) | (0<<CS12) | (1<<CS11) | (1<<CS10)
    ; N = 64
    sts TCCR1B, temp

    clr temp
    sts OCR1AH, temp ; values are from 0 to 255
    ldi temp, 124
    sts OCR1AL, temp ; set the compare level

    clr temp
    sts ICR1H, temp  ; values are from 0 to 255
    ldi temp, 249
    sts ICR1L, temp  ; set the top level

    in temp, PIND    ; keep the state of PIND
    andi temp, 0x08  ; isolate PD3
    cpi temp, 0x00   ; compare with zero
    ; if pushed temp == 0 (reversed logic)
    breq PressD3     ; if the button is still pressed wait
    rjmp main        ; continue looping
```

Κώδικας σε C

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

int main() {
    DDRB = 0xFF;                                    // Set PORTB as output
    DDRD = 0x00;                                    // Set PORTD as input
    TCCR1A = (0<<WGM10) | (1<<WGM11) | (1<<COM1A1); // Init control register A of Timer 1
    TCCR1B = (1<<WGM12) | (1<<WGM13);               // Init control register B of Timer 1
    // with the above values we have fast PWM mode with TOP = ICR1

    while(1) {                                       // loop forever
        TCCR1B = (1<<WGM12) | (1<<WGM13) | (0<<CS12) | (0<<CS11) | (0<<CS10);
        // when no buttons are pressed
        // we stop PWM setting timer's frequency to zero
        // we want to change only the CS12, CS11, CS10 bits of the register

        // formula: top(fpwm, N) = fclk/(N*fpwm) - 1
        if ((PIND & 0x01) == 0x00) { // check if PD0 pressed (logical 0)
            TCCR1B = (1<<WGM12) | (1<<WGM13) | (1<<CS12) | (0<<CS11) | (1<<CS10);    // N = 1024
            ICR1H = 0;                              //ICR=TOP
            ICR1L = 124;
            OCR1AH = 0x00;                          // values from 0 to 255
            OCR1AL = 62;                            // set OCR1AL = ICR/2, DC=50%
            while((PIND & 0x01) == 0x00);           // while being pressed PWM with 125Hz
        }
        else if ((PIND & 0x02) == 0x00) {           // check if PD1 pressed (logical 0)
            TCCR1B = (1<<WGM12) | (1<<WGM13) | (1<<CS12) | (0<<CS11) | (0<<CS10);    // N = 256
            ICR1H = 0;
            ICR1L = 249;
            OCR1AH = 0x00;                          // values from 0 to 255
            OCR1AL = 124;                           // set OCR1AL = TOP   round of 124.5
            while((PIND & 0x02) == 0x00);           // while being pressed PWM with 250Hz
        }
        else if ((PIND & 0x04) == 0x00) {           // check if PD2 pressed (logical 0)
            TCCR1B = (1<<WGM12) | (1<<WGM13) | (1<<CS12) | (0<<CS11) | (0<<CS10);    // N = 256
            ICR1H = 0;                              //ICR=TOP
            ICR1L = 124;
            OCR1AH = 0x00;                         // values from 0 to 255
            OCR1AL = 62;                           // set OCR1AL = TOP   round of 62.5
            while((PIND & 0x04) == 0x00);          // while being pressed PWM with 500Hz
        }
        else if ((PIND & 0x08) == 0x00) {          // check if PD3 pressed (logical 0)
            TCCR1B = (1<<WGM12) | (1<<WGM13) | (0<<CS12) | (1<<CS11) | (1<<CS10);    // N = 64
            ICR1H = 0;
            ICR1L = 249;
            OCR1AH = 0x00;                         // values from 0 to 255
            OCR1AL = 124;                          // set OCR1AL = TOP
            while((PIND & 0x08) == 0x00);          // while being pressed PWM with 1000Hz
        }
    }
}
```