ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2022-2023

Αναφορά

3η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Ανάπτυξη κώδικα για το μικροελεγκτή ATmega328PB και προσομοίωση της εκτέλεσης του στο αναπτυξιακό περιβάλλον MPLAB X

Ομάδα 1

Ξανθόπουλος Παναγιώτης (03119084)

Παπαναστάσης Αθανάσιος (03113197)

Ζήτημα 3.1

Στο πρόγραμμα αρχικά επιτρέπουμε την διακοπή Timer1 overflow ώστε όταν ο μετρητής TCNT1 πάρει την τιμή 0xFFFF, να συνεχίσει το πρόγραμμα από την ρουτίνα timer1. Επίσης, επιτρέπουμε την διακοπή INT1 σε ανερχόμενη ακμή και ορίζουμε τον μετρητή να τρέχει με prescaler /1024. Εφόσον θέλουμε ο μετρητής να κάνει overflow στα 4 δευτερόλεπτα, υπολογίζουμε την αρχική του τιμή ως εξής:

Η συχνότητα αύξησης του μετρητή είναι 16MHz / 1024 = 15625 Hz. Εφόσον θέλουμε να διαρκέσει 4 δευτερόλεπτα, θέλουμε ο μετρητής να μετρήσει 4 * 15625 περιόδους, δηλαδή 62500 περιόδους. Εφόσον το overflow γίνεται κάθε 65536 περιόδους του μετρητή, αφαιρούμε το 62500 και προκύπτει το 3036 ως αρχική τιμή του μετρητή.

Στο κύριο πρόγραμμα θέτουμε την PORTB ως έξοδο και την PORTC ως είσοδο. Στη συνέχεια περιμένουμε μέχρι να πατηθεί το PC5. Όταν πατηθεί, περιμένουμε να το αφήσει ο χρήστης. Όταν το αφήσει, ανανεώνουμε τον μετρητή στο 3036, ανάβουμε όλα τα LED της PORTB και περιμένουμε 500ms (μέσω της δοσμένης ρουτίνας delay_mS). Όταν περάσουν, αφήνουμε μόνο το LSB αναμμένο και επιστρέφουμε ξανά στο loop που περιμένει να πατήσουμε το PC5.

Στην ρουτίνα εξυπηρέτησης της διακοπής INT1, αφού αποθηκεύσουμε στη στοίβα κάποιους καταχωρητές που δεν θέλουμε να χάσουμε την τιμή τους, ελέγχουμε για σπινθηρισμό (ο κώδικας που ελέγχει για σπινθηρισμό είναι ίδιος με αυτόν της άσκησης 2). Στη συνέχεια, ανανεώνουμε τον μετρητή στο 3036, ανάβουμε όλα τα LED της PORTB και περιμένουμε 500ms (μέσω της δοσμένης ρουτίνας delay_mS). Όταν περάσουν, αφήνουμε μόνο το LSB αναμμένο, επανακτούμε τις τιμές των καταχωρητών από την στοίβα και επιστρέφουμε.

Στην ρουτίνα εξυπηρέτησης της διακοπής υπερχείλισης του μετρητή, απλά κλείνουμε όλα τα LED.

```
.include "m328PBdef.inc"

.org 0x0
   rjmp reset
.org 0x4
   rjmp ISR1
.org 0x1A
   rjmp timer1

reset:
   ldi r24, low(RAMEND)
   out SPL, r24
   ldi r24, high(RAMEND)
   out SPL, r24

   ldi r26,(1<<TOIE1)                ;enable timer1 overflow interrupt
   sts TIMSK1, r26

   ldi r26, (1 << ISC11) | (1 << ISC10)
   sts EICRA, r26

   ldi r26, (1 << INT1)
   out EIMSK, r26
                                     ;INT1 is enabled at rising edge
   ldi r26 ,(1<<CS12) | (0<<CS11) | (1<<CS10) ; CK/1024
   sts TCCR1B, r26
```

```
sei

init:
   ser r26              ;PORT B is set as output
   out DDRB, r26
   clr r26
   out DDRC, r26     ;PORT C is set as input

loop:
   in r20, PINC        ;wait for PC5 to be pressed
   sbrc r20, 5                  ;sbrs for sim, sbrc for lab
   rjmp loop

loop1:
   in r20, PINC        ;wait for PC5 to be released
   sbrs r20, 5                  ;sbrc for sim, sbrs for lab
   rjmp loop1

   ldi r26, HIGH(3036)    ;reset TCNT1
   sts TCNT1H, r26
   ldi r26, LOW(3036)
   sts TCNT1L, r26

   ser r26                 ;turn on all leds for 500ms
   out PORTB, r26

   ldi r24, LOW(16*500)
   ldi r25, HIGH(16*500)
   rcall delay_mS

   ldi r26, 0x01          ;turn on only LSB
   out PORTB, r26

   rjmp loop



ISR1:
   push r24
   push r25
   in r26, SREG
   push r26

debounce:                        ;debounce avoidance algorithm (given)
   ldi r26, (1 << INTF1)
   out EIFR, r26

   ldi r24, low(16*5)
   ldi r25, high(16*5)
   rcall delay_mS

   in r26, EIFR
   lsr r26
   lsr r26
   brcs debounce

   ldi r26, HIGH(3036)    ;reset TCNT1
   sts TCNT1H, r26
   ldi r26, LOW(3036)
   sts TCNT1L, r26
```

```
    ser r26                 ;turn on all leds for 500ms
    out PORTB, r26

    ldi r24, LOW(16*500)
    ldi r25, HIGH(16*500)
    rcall delay_mS

    ldi r26, 0x01           ;turn on only LSB
    out PORTB, r26

    pop r26
    out SREG, r26
    pop r25
    pop r24
    reti



timer1:
    in r26, SREG
    push r26

    clr r26
    out PORTB, r26

    pop r26
    out SREG, r26
    reti



delay_mS:               ;delay of 1000*DEL_NU_X + 6 cycles
    ldi r23, 249    ;default is 249, for simulator set to 15
loop_inn:
    dec r23
    nop
    brne loop_inn

    sbiw r24, 1
    brne delay_mS

    ret
```

Ο κώδικας σε C λειτουργεί με αντίστοιχο τρόπο.

```
#define F_CPU 16000000UL

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

ISR(INT1_vect)
{
   while (EIFR==0x02) {
      EIFR = 0x02;
      _delay_ms(5);
   }
   TCNT1 = 3036;
   PORTB = 0xFF;
   _delay_ms(500);
   PORTB = 0x01;
}

ISR(TIMER1_OVF_vect)
{
   PORTB = 0x00;
}

static unsigned char x;

int main(void)
{
   TIMSK1 = (1<<TOIE1);                  //timer1 overflow interrupt enabled

   EICRA = (1 << ISC11) | (1 << ISC10);

   EIMSK = (1 << INT1);                  //int1 enabled at rising edge

   TCCR1B = (1<<CS12) | (0<<CS11) | (1<<CS10);    //CK/1024

   sei();

   DDRB = 0xFF;       //PORTB as output
   DDRC = 0x00;       //PORTC as input

   PORTB = 0x00;

   while (1)
   {
      x = PINC;
      if ((x & 0x20) == 0x20) {         //check if PC5 was pressed
         while ((PINC & 0x20) == 0x20);   //wait for PC5 to be released
         _delay_ms(10);                 //debounce avoidance
         TCNT1 = 3036;
         PORTB = 0xFF;
         _delay_ms(500);
         PORTB = 0x01;
      }
   }
}
```

Ζήτημα 3.2

Στο πρόγραμμα αυτό, ο μετρητής TCNT1 λειτουργεί σε fast PWM mode – 8bit σύμφωνα με τα bits WGM1[3:0] = 0b0101 των καταχωρητών TCCR1A και TCCR1B, δηλαδή ξεκινάει από το 0x00 και πηγαίνει μέχρι το 0xFF. Επίσης, η έξοδος του PWM είναι το pin OC1A, που αντιστοιχεί στο pin PB1. Σύμφωνα με τα bits COM1A[1:0] = 0b10, το OC1A κάνει override την κανονική λειτουργία του PB1 και βγάζει στην έξοδο λογικό 1 όταν ο μετρητής γίνει 0, ενώ όταν ο μετρητής γίνει ίσος με την τιμή του καταχωρητή OCR1A, η έξοδος γίνεται λογικό 0. Έτσι, παράγεται ένας τετραγωνικός παλμός με DC που εξαρτάται από την τιμή του OCR1A σε σχέση με την τιμή 0xFF (το max του μετρητή).

Αρχικά θέλουμε το DC να είναι 50%. Εφόσον ο μετρητής πάει από 0 έως 255, θέλουμε η τιμή του OCR1A να είναι 50% · 255 = 128. Ομοίως υπολογίζουμε τις υπόλοιπες τιμές:

2%: OCR1A = 5 (0x05)

10%: OCR1A = 26 (0x1A)

18%: OCR1A = 46 (0x2E)

26%: OCR1A = 66 (0x42)

34%: OCR1A = 87 (0x57)

42%: OCR1A = 107 (0x6B)

50%: OCR1A = 128 (0x80)

58%: OCR1A = 148 (0x94)

66%: OCR1A = 168 (0xA8)

74%: OCR1A = 189 (0xBD)

82%: OCR1A = 209 (0xD1)

90%: OCR1A = 230 (0xE6)

98%: OCR1A = 250 (0xFA)

Οργανώνουμε τις τιμές αυτές σε λέξεις των 16 bit όπως φαίνεται στο τέλος του προγράμματος και τις τοποθετούμε στη μνήμη προγράμματος. Η πρόσβαση στις τιμές αυτές γίνεται μέσω του διπλού καταχωρητή Z (R31:R30) και της εντολής LPM.

Αρχικοποιούμε τον OCR1A στην τιμή που αντιστοιχεί σε DC 50% και θέτουμε την PORTD ως είσοδο και την PORTB ως έξοδο.

Επίσης, χρησιμοποιούμε τον r21 ως μετρητή ώστε να μη ξεφεύγουμε από τα όρια του πίνακα.

Η ρουτίνα main ελέγχει συνεχώς αν έχει πατηθεί ένα εκ των PD1 και PD2. Αν πατηθεί κάποιο, έστω το PD1, το πρόγραμμα πάει στην ρουτίνα wait_for_PD1, η οποία περιμένει να αφήσουμε το PD1. Όταν το αφήσουμε, το πρόγραμμα πάει στην ρουτίνα inc_DC ώστε να αυξηθεί το DC.

Στην ρουτίνα αυτή, αρχικά έχουμε delay 10ms ώστε να αποφευχθεί τυχόν σπινθηρισμός στο κουμπί PD1. Στη συνέχεια, ελέγχουμε αν έχουμε φτάσει στο μέγιστο DC, βλέποντας αν η τιμή του μετρητή είναι 13. Αν είναι, επιστρέφουμε απευθείας στην main. Αν δεν είναι, αφού τον

αυξήσουμε, βάζουμε το Z (R31:R30) να δείχνει στο επόμενο byte, το φορτώνουμε στον καταχωρητή OCR1A (στο low byte του) και επιστρέφουμε στην main.

Η ρουτίνα για το PD2 λειτουργεί πανομοιότυπα με τις διαφορές ότι ο μετρητής ελέγχεται αν είναι ίσος με την τιμή 1 (δηλαδή αν το DC είναι το μικρότερο δυνατό) και ο Z θα δείχνει στο προηγούμενο byte αν μειώσουμε το DC.

```
.include "m328PBdef.inc"

.org 0x0
   rjmp reset

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

   ldi r24, (1 << WGM10) | (1 << COM1A1)
   sts TCCR1A, r24

   ldi r24, (1 << WGM12) | (1 << CS11)
   sts TCCR1B, r24

   ldi zh, high(table*2)
   ldi zl, low(table*2)
   adiw zl, 6

   lpm r23, z
   sts OCR1AL, r23          ;DC initalized at 50%

   clr r24
   out DDRD, r24            ;set D as input

   ser r24
   out DDRB, r24

   ldi r21, 7                     ;counter

main:
   in r25, PIND             ;check if PD1 was pressed
   lsr r25
   lsr r25
   brcc wait_for_PD1              ;brcs for sim, brcc for lab

   lsr r25                  ;check if PD2 was pressed
   brcc wait_for_PD2              ;brcs for sim, brcc for lab

   rjmp main                      ;if none, check again


wait_for_PD1:                     ;wait for PD1 to be released
   in r25, PIND             ;check if PD1 was released
   lsr r25
   lsr r25
   brcs inc_DC                    ;brcc for sim, brcs for lab
   rjmp wait_for_PD1              ;if not, check again
```

Ο κώδικας σε C λειτουργεί με αντίστοιχο τρόπο.

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

int main(void) {

   static unsigned char table[13] = {5, 26, 46, 66, 87, 107, 128, 148, 168, 189, 209, 230, 250};
   static unsigned char x;

   TCCR1A = (1<<WGM10) | (1<<COM1A1);
   TCCR1B = (1<<WGM12) | (1<<CS11);

   DDRB |= 0b00111111;

   static int counter = 7;
   OCR1AL = table[counter - 1];

   DDRD = 0b00000000;

   while (1) {              // != on the conditions for lab, == for sim
      x = PIND;
      if ((x & 0x02) != 0x02) {        //check if PD1 was pressed
         while ((PIND & 0x02) != 0x02);   //wait for PD1 to be released
         _delay_ms(10);               //debounce avoidance
         if (counter != 13) {
            counter++;
            OCR1AL = table[counter - 1];
         }
      }
      else if ((x & 0x04) != 0x04) {    //check if PD2 was pressed
         while ((PIND & 0x04) != 0x04);   //wait for PD2 to be released
         _delay_ms(10);               //debounce avoidance
         if (counter != 1) {
            counter--;
            OCR1AL = table[counter - 1];
         }
      }
   }
}
```

Ζήτημα 3.3

Στο πρόγραμμα αυτό, αρχικά θέτουμε την PORTB ως είσοδο και την PORTD ως έξοδο. Στη συνέχεια, στην ρουτίνα none θέτουμε τον μετρητή στο 0. Τα bits ελέγχου του μετρητή και του PWM είναι ως εξής:

- WGM1[3:0] = 0b1110 ώστε ο μετρητής να ξεκινάει από το bottom (0) και να πηγαίνει μέχρι την τιμή του ICR1.

- COM1A[1:0] = 0b00 ώστε το pin OC1A να μην είναι συνδεδεμένο με το PB1 και να μην υπάρχει έξοδος.
- CS1[2:0] = 0b000 ώστε ο μετρητής να είναι σταματημένος

Η ρουτίνα none (μαζί με την recheck) τρέχει όταν δεν είναι πατημένο κάποιο κουμπί. Η recheck διαβάζει την είσοδο και ελέγχει αν έχει πατηθεί ένα από τα κουμπιά. Αν έχει πατηθεί κάποιο, πηγαίνει στην αντίστοιχη ρουτίνα, αλλιώς ξαναελέγχει.

Αναλύουμε την ρουτίνα PD0_125 η οποία αντιστοιχεί στο κουμπί PD0. Οι υπόλοιπες λειτουργούν ανάλογα.

Αρχικά θέτουμε τα bits ελέγχου του PWM ως εξής:

- WGM1[3:0] = 0b1110 ώστε ο μετρητής να ξεκινάει από το bottom (0) και να πηγαίνει μέχρι την τιμή του ICR1.
- COM1A[1:0] = 0b10 ώστε το pin OC1A να μην είναι συνδεδεμένο με το PB1 και να γίνεται λογικό 1 όταν ο μετρητής γίνει 0 και λογικό 0 όταν ο μετρητής γίνει ίσος με την τιμή του καταχωρητή OCR1A
- CS1[2:0] = 0b010 ώστε ο μετρητής τρέχει με prescaler /8.

Η συχνότητα υπολογίζεται ως εξής: $f_{PWM} = \frac{f_{CLK}}{N \cdot (1+TOP)}$. Εφόσον $f_{CLK} = 16MHz$, N=8 kai θέλουμε $f_{PWM} = 125Hz$, προκύπτει TOP = ICR1 = 15999. Οπότε, φορτώνουμε την τιμή αυτή στον ICR1 και το μισό της στον OCR1A ώστε το DC να είναι 50%. Στη συνέχεια το πρόγραμμα πηγαίνει στην ρουτίνα release_PD0, η οποία περιμένει μέχρι να αφήσουμε το PD0. Όταν το αφήσουμε, επιστρέφει στην none, ώστε να μην παράγεται πια κυματομορφή.

Οι υπόλοιπες ρουτίνες λειτουργούν με τον ίδιο τρόπο.

```
.include "m328PBdef.inc"

.org 0x0
   rjmp reset

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

   ser r24
   out DDRB, r24

   clr r24
   out DDRD, r24        ;set PORTD as input

   ;WGM1[3:0] = 0xE for fast PWM, bottom = 0, top = ICR1
   ;CS1[2:0] = 0x0 for timer stop, 0x2 for prescaler 8 (default)
   ;COM1A[1:0] = 0 for no output, 0x2 for non inverting fast PWM
```

```
none:                           ;no waveform output but keep checking continuously
   ldi r24, 0x00          ;reset timer
   sts TCNT1H, r24
   sts TCNT1L, r24

   ldi r24, (1 << WGM11)   ;no waveform and timer is stopped
   sts TCCR1A, r24

   ldi r24, (1 << WGM12) | (1 << WGM13 )
   sts TCCR1B, r24

recheck:
   in r24, PIND

   sbrs r24, 0             ;sbrc for sim, sbrs for lab
   jmp PD0_125

   sbrs r24, 1             ;sbrc for sim, sbrs for lab
   jmp PD1_250

   sbrs r24, 2             ;sbrc for sim, sbrs for lab
   jmp PD2_500

   sbrs r24, 3             ;sbrc for sim, sbrs for lab
   jmp PD3_1000

rjmp recheck



PD0_125:
   ldi r24, (1 << WGM11) | (1 << COM1A1) ;fast PWM at 125Hz, non inverted, 50%DC, prescaler 8
   sts TCCR1A, r24

   ldi r24, (1 << WGM12) | (1 << WGM13 ) | (1 << CS11)
   sts TCCR1B, r24

   ldi r24, 0x3E
   ldi r25, 0x7F
   sts ICR1H, r24
   sts ICR1L, r25          ;ICR1 at 15999 for 125Hz

   ldi r24, 0x1F
   ldi r25, 0x40
   sts OCR1AH, r24
   sts OCR1AL, r25       ;OCR1A at 8000 for 50% DC

release_PD0:                    ;wait for PD0 to be released
   in r24, PIND
   lsr r24
   brcc release_PD0    ;if not yet released, check again
                                ;brcs for sim, brcc for lab
   jmp none                     ;if released, stop waveform
```

```
PD1_250:
   ldi r24, (1 << WGM11) | (1 << COM1A1) ;fast PWM at 125Hz, non inverted, 50%DC, prescaler 8
   sts TCCR1A, r24

   ldi r24, (1 << WGM12) | (1 << WGM13 ) | (1 << CS11)
   sts TCCR1B, r24

   ldi r24, 0x1F
   ldi r25, 0x3F
   sts ICR1H, r24
   sts ICR1L, r25        ;ICR1 at 7999 for 125Hz

   ldi r24, 0x0F
   ldi r25, 0xA0
   sts OCR1AH, r24
   sts OCR1AL, r25      ;OCR1A at 4000 for 50% DC

release_PD1:                    ;wait for PD1 to be released
   in r24, PIND
   lsr r24
   lsr r24
   brcc release_PD1   ;if not yet released, check again
                                   ;brcs for sim, brcc for lab
   jmp none                        ;if released, stop waveform



PD2_500:
   ldi r24, (1 << WGM11) | (1 << COM1A1) ;fast PWM at 125Hz, non inverted, 50%DC, prescaler 8
   sts TCCR1A, r24

   ldi r24, (1 << WGM12) | (1 << WGM13 ) | (1 << CS11)
   sts TCCR1B, r24

   ldi r24, 0x0F
   ldi r25, 0x9F
   sts ICR1H, r24
   sts ICR1L, r25        ;ICR1 at 3999 for 125Hz

   ldi r24, 0x07
   ldi r25, 0xD0
   sts OCR1AH, r24
   sts OCR1AL, r25      ;OCR1A at 2000 for 50% DC

release_PD2:                    ;wait for PD2 to be released
   in r24, PIND
   lsr r24
   lsr r24
   lsr r24
   brcc release_PD2   ;if not yet released, check again
                                   ;brcs for sim, brcc for lab
   jmp none                        ;if released, stop waveform
```

```
PD3_1000:
   ldi r24, (1 << WGM11) | (1 << COM1A1) ;fast PWM at 125Hz, non inverted, 50%DC, prescaler 8
   sts TCCR1A, r24

   ldi r24, (1 << WGM12) | (1 << WGM13 ) | (1 << CS11)
   sts TCCR1B, r24

   ldi r24, 0x07
   ldi r25, 0xCF
   sts ICR1H, r24
   sts ICR1L, r25        ;ICR1 at 1999 for 125Hz

   ldi r24, 0x03
   ldi r25, 0xE8
   sts OCR1AH, r24
   sts OCR1AL, r25      ;OCR1A at 1000 for 50% DC

release_PD3:                    ;wait for PD3 to be released
   in r24, PIND
   lsr r24
   lsr r24
   lsr r24
   lsr r24
   brcc release_PD3   ;if not yet released, check again
                                ;brcs for sim, brcc for lab
   jmp none                     ;if released, stop waveform
```

Ο κώδικας σε C λειτουργεί με αντίστοιχο τρόπο. Σημειώνουμε ότι το break μετά από τις κλήσεις στις συναρτήσεις pd_ υπάρχει ώστε όταν επιστρέψουμε από τις συναρτήσεις αυτές, να βγούμε από το εσωτερικό while ώστε να γίνει reset ο μετρητής και να σταματήσει να αυξάνεται, όπως και να σταματήσει η παραγωγή κυματομορφής.

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

void pd0(void) {
   TCCR1A = (1 << WGM11) | (1 << COM1A1);
   TCCR1B = (1 << WGM12) | (1 << WGM13 ) | (1 << CS11);
   ICR1 = 0x3E7F;
   OCR1A = 0x1F40;

   while ((PIND & 0x01) != 0x01);
   return;
}

void pd1(void) {
   TCCR1A = (1 << WGM11) | (1 << COM1A1);
   TCCR1B = (1 << WGM12) | (1 << WGM13 ) | (1 << CS11);
   ICR1 = 0x1F3F;
```

```
   OCR1A = 0x0FA0;

   while ((PIND & 0x02) != 0x02);
   return;
}

void pd2(void) {
   TCCR1A = (1 << WGM11) | (1 << COM1A1);
   TCCR1B = (1 << WGM12) | (1 << WGM13 ) | (1 << CS11);
   ICR1 = 0x0F9F;
   OCR1A = 0x07D0;

   while ((PIND & 0x04) != 0x04);
   return;
}

void pd3(void) {
   TCCR1A = (1 << WGM11) | (1 << COM1A1);
   TCCR1B = (1 << WGM12) | (1 << WGM13 ) | (1 << CS11);
   ICR1 = 0x07CF;
   OCR1A = 0x03E8;

   while ((PIND & 0x08) != 0x08);
   return;
}

int main(void) {

   DDRD = 0b00000000;
   DDRB |= 0b00111111;
   static unsigned char x;

   while (1) {
      TCNT1 = 0x0000;
      TCCR1A = (1 << WGM11);
      TCCR1B = (1 << WGM12) | (1 << WGM13 );
      while (1) {
         x = PIND;
         if ((x & 0x01) != 0x01) {   //check if PD0 was pressed
            pd0();
            break;
         }
         if ((x & 0x02) != 0x02) {   //check if PD1 was pressed
            pd1();
            break;
         }
         if ((x & 0x04) != 0x04) {   //check if PD2 was pressed
            pd2();
            break;
         }
         if ((x & 0x08) != 0x08) {   //check if PD3 was pressed
            pd3();
            break;
         }
      }
   }
}
```