Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

3η Σειρά Ασκήσεων

Μάθημα: Εργαστήριο Μικροϋπολογιστών - Ομάδα 30

Εξάμηνο: 7ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία (ΑΜ: 03120164), Γκενάκου Ζωή (ΑΜ: 03120015)

Ζήτημα 3.1

Να δημιουργηθεί κώδικας σε γλώσσα assembly, ο οποίος να αρχικοποιεί τον TMR1A σε λειτουργία 8-bit και να παράγει μία PWM κυματομορφή στον ακροδέκτη PB1. Θεωρήστε ότι BOTTOM=0. Στον ακροδέκτη PB1 είναι συνδεδεμένο ένα LED και η φωτεινότητα του μεταβάλλεται εάν μεταβληθεί το Duty Cycle της PWM κυματομορφής. Αρχικά το Duty Cycle ρυθμίζεται σε 50% αποθηκεύεται σε μία μεταβλητή με όνομα DC_VALUE. Στη συνέχεια κάθε φορά που πιέζεται το μπουτόν PD1 το Duty Cycle αυξάνεται κατά 8%. Όταν το Duty Cycle φτάσει τη μέγιστη τιμή 98% η πίεση του μπουτόν PD1 δεν το αυξάνει περεταίρω. Κάθε φορά που πιέζεται το μπουτόν PD2 το Duty Cycle μειώνεται κατά 8%. Όταν το Duty Cycle φτάσει τη ελάχιστη τιμή 2% η πίεση του μπουτόν PD2 δεν το μειώνει περεταίρω. Οι τιμές που πρέπει να πάρει ο καταχωρητής OCR1A, για τις διάφορες τιμές του Duty Cycle, να έχουν υπολογιστεί εκ των προτέρων και να έχουν τοποθετηθεί σε ένα πίνακα στη μνήμη του μικροελεγκτή, έτσι ώστε να μη χρειάζεται να υπολογιστούν κατά τη διάρκεια εκτέλεσης του κώδικα.

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

.equ FOSC_MHZ = 16                ; Microcontroller operating
```

```
frequency in MHz

; Define DC_VALUE at the beginning
.def DC_VALUE = r18

ldi DC_VALUE, 0x80                ; Set the initial duty cycle to
50%

reset:
ldi r24, (1<<WGM10) | (1<<COM1A1)
sts TCCR1A, r24                   ; Set Timer/Counter1 Control
Register A

ldi r24, (1<<WGM12) | (1<<CS11)
sts TCCR1B, r24                   ; Set Timer/Counter1 Control
Register B

; Initialize Stack Pointer
ldi r24, low(RAMEND)
out SPL, r24
ldi r24, high(RAMEND)
out SPH, r24

; Initialize PORTB as output
ser r26
out DDRB, r26

; Initialize PORTD as input
clr r27
out DDRD, r27

clr r17                           ; Clear r17 for use as a temporary
register
ldi r21, 0x06                     ; Initialize array position at the
middle, 50% DC

ldi DC_VALUE, 0x80                ; Set the initial duty cycle to
50%
sts OCR1AL, DC_VALUE              ; Set the output compare register
```

```
for 50% duty cycle

main:
; Load the address of the 'duty' array into Z register
ldi zh, high(duty*2)
ldi zl, low(duty*2)

; Read the input from PORTD (PD1 and PD2)
in r20, PIND
cpi r20, 0b11111101               ; Check if PD1 is pressed
breq butt_1
cpi r20, 0b11111011               ; Check if PD2 is pressed
breq butt_2
rjmp main

button_1:
in r20, PIND
cpi r20, 0b11111101
breq button_1                     ; Button 1 debounce, stay in
this state

cpi r21, 0x0C                     ; Check if r21 is at the
maximum value (12)
breq main
inc r21                           ; Increment r21

; Calculate the new array index and load the corresponding value
mov r22, r21
lsl r22                           ; r22 = r21 * 2
add zl, r22
adc zh, r17                       ; r17 = 0
lpm                               ; Load value from 'duty' into
r0
mov r19, r0                       ; Store the value in r19

sts OCR1AL, r19                   ; Set the duty cycle based on
the array value
rjmp main
```

```
button_2:
in r20, PIND
cpi r20, 0b11111011
breq button_2                    ; Button 2 debounce, stay in
this state

cpi r21, 0x00                    ; Check if r21 is at the minimum
value (0)
breq main
dec r21                          ; Decrement r21

; Calculate the new array index and load the corresponding value
mov r22, r21
lsl r22                          ; r22 = r21 * 2
add zl, r22
adc zh, r17                      ; r17 = 0
lpm                              ; Load value from 'duty' into r0
mov r19, r0                      ; Store the value in r19

sts OCR1AL, r19                  ; Set the duty cycle based on
the array value
rjmp main

duty:
.DW 0x0005, 0x001A, 0x002E, 0x0043, 0x0057, 0x006C, 0x0080, 0x0094,
0x00A7, 0x00BD, 0x00D2, 0x00E6, 0x00FB
```

Για το Ζήτημα 3.1 έχουμε τα παρακάτω:

Αρχικά για τον υπολογισμό των τιμών του Duty Cycle που τοποθετήσαμε στον πίνακα:

%Duty Cycle	Decimal	Hexadecimal
2	5	05
10	26	1A

18	46	2E
26	67	43
34	87	57
42	108	6C
50	128	80
58	148	94
66	167	A7
74	189	BD
82	210	D2
90	230	E6
98	251	FB

Ο κώδικας αποθηκεύει εκ των προτέρων τις τιμές του Duty Cycle με αύξουσα σειρά σε έναν πίνακα στη μνήμη του μικροελεγκτή. Αυτές οι τιμές είναι προκαθορισμένες για την αποφυγή υπολογισμών σε πραγματικό χρόνο κατά την εκτέλεση του κώδικα. Ο μέγιστος κύκλος λειτουργίας (100%) περιορίζεται σε 256 επειδή ο Timer1 (TMR1A) λειτουργεί σε λειτουργία 8 bit. Αρχικά, ο κώδικας διαμορφώνει τον πίνακα με αυτές τις προκαθορισμένες τιμές του Duty Cycle, αρχικοποιώντας τον στο 50%.
Στην συνέχεια, ο κώδικας παρακολουθεί συνεχώς το πατήματα κουμπιών PD1 ή PD2. Όταν ανιχνεύεται ένα πάτημα κουμπιού, είτε αυξάνει είτε μειώνει τη μεταβλητή `r21`. Αυτή η μεταβλητή λειτουργεί ως δείκτης, διευκολύνοντας την πρόσβαση στον πίνακα "`duty`" και, κατά συνέπεια, τη ρύθμιση της φωτεινότητας με βάση τις εισόδους κουμπιών.

Ο πίνακας περιέχει στοιχεία μεγέθους 16 bit ή 2 byte. Ο καταχωρητής Z, που αποτελείται από ZH (high byte) και ZL (low byte), χρησιμεύει ως δείκτης στον πίνακα.Διπλασιάζοντας το r21 (ουσιαστικά μετατοπίζοντάς το προς τα αριστερά κατά ένα bit), ο κώδικας υπολογίζει μια μετατόπιση που καθορίζει σε ποιο στοιχείο του πίνακα θα προσπελαστεί. Αυτή η ενέργεια ενημερώνει τη μετατόπιση που είναι αποθηκευμένη στον καταχωρητή ZL, επιτρέποντας στον κώδικα να έχει πρόσβαση στο επόμενο στοιχείο του πίνακα όσο αυξάνεται ο r21.

Ζήτημα 3.2

Συνδέστε την είσοδο A1 του ADC με το αναλογικό φίλτρο PB1_PWM. Αποσυνδέστε το led PB1 κάνοντας χρήση των dip switches SW1. Να γραφτεί ξανά ο κώδικας του ζητήματος 3.1 σε γλώσσα C, ο οποίος θα παράγει PWM έξοδο στον ακροδέκτη PB1. Επιπλέον ο ADC διαβάζει τη μεταβαλλόμενη DC τάση που παράγεται στην έξοδο του αναλογικού φίλτρου PB1_PWM κάθε 100 mSec (μικρές αποκλίσεις είναι αποδεκτές). Η τιμή μέτρησης του ADC (ADCH: ADCL) θα μετατρέπεται σε τιμή τάσης (Vadc) θα ανάβει ένα από τα led που είναι συνδεδεμένα στη θύρα PORTD σύμφωνα με τον παρακάτω πίνακα:

Τάση εισόδου στον ADC(Vadc)	**LED ON**
0 Volt ≤ Vadc ≤ 0,625 Volt	PD0
0,625 Volt < Vadc ≤ 1,25 Volt	PD1
1,25 Volt < Vadc ≤ 1,875 Volt	PD2
1,875 Volt < Vadc ≤ 2,5 Volt	PD3
2,5 Volt < Vadc ≤ 3,125 Volt	PD4
3,125 Volt < Vadc ≤ 3,75 Volt	PD5
3,75 Volt < Vadc ≤ 4,375 Volt	PD6
4,375 Volt < Vadc ≤ 5 Volt	PD7

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


unsigned char duty[] = {0x05, 0x1A, 0x2E, 0x43, 0x57, 0x6C, 0x80,
0x94, 0xA7, 0xBD, 0xD2, 0xE6, 0xFB};      // Ascending order

int main(){

    TCCR1A = (1<<WGM10)|(1<<COM1A1);  // Non-inverting Mode, Fast
PWM, 8-bit Mode
    TCCR1B = (1<<WGM12)|(1<<CS11);          // Prescaler is 1
(frequency = f.clk/8)
    //Set Vref = 5V
    ADMUX = (1 << REFS0) | (1 << MUX0) |(1 << ADLAR);
```

```
    //Enable ADC & set ADC prescaler
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (0 <<
ADPS0);


    DDRB=0b001111;                          // Set PB1 as output,
PB4,5 as input
    DDRD=0xFF;                                  // Set PORTD as
output

    unsigned int DC_VALUE = 6;
    unsigned char input , adcValue, output;
    OCR1AL = duty[DC_VALUE];                // Initial dc 50%



    while(1){
        input = PINB&0b010000;
        if((!(input))&&(DC_VALUE<12)){      // If PB4 is pressed
            DC_VALUE++ ;               // Increase dc by 8%
            _delay_ms(50);
        }
        input = PINB&0b100000;
        if((!(input))&&(DC_VALUE>0)){           // If PB5 is
pressed
            DC_VALUE--;                     // Decrease dc by 20%
            _delay_ms(50);
    }

    OCR1AL = duty[DC_VALUE];

    ADCSRA |= (1<<ADSC);

    output = ADCSRA;
    output &= 0b01000000;
    while(output){
        output = ADCSRA;
        output &= 0b01000000;
    }
```

```
    adcValue = ADCH;

    if(adcValue<32){
        output=0x01;
    }
    else if(adcValue<64){
        output=0x02;
    }
    else if(adcValue<96){
        output=0x04;
    }
    else if(adcValue<128){
        output=0x08;
    }
    else if(adcValue<160){
        output=0x10;
    }
    else if(adcValue<192){
        output=0x20;
    }
    else if(adcValue<224){
        output=0x40;
    }
    else{
        output=0x80;
    }

    PORTD = output;
    _delay_ms(100);

    }
    return 0;
}
```

Ακολουθώντας τη δομή του ζητήματος 3.1, αρχικοποιούμε κατάλληλα τους καταχωρητές TCCR1A και TCCR1B. Αρχικοποιούμε επίσης τον καταχωρητή ADMUX έτσι ώστε να ενεργοποιείται το κανάλι εισόδου ADC1, να επιλέγεται $V_{ref} = 5\ V$ και το περιεχόμενο του ADMUX να είναι left-adjusted. Επίσης, ενεργοποιούμε τον ADC και θέτουμε τον prescaler στο 64. Η θήρα PORTD

λειτουργεί εξ' ολοκλήρου ως output, ενώ η θήρα PORTB λειτουργεί ταυτόχρονα ως output (PB1) και ως input (PB4, PB5). Τα PB4 και PB5 επιτελούν τώρα τη λειτουργία των PD1 και PD2 του ζητήματος 3.1 αντίστοιχα. Έχουμε υπολογίσει την τάση ADC με βάση $V_{ref} = 5\ V$, επομένως ανά 100 ms ελέγχεται το εύρος της τάσης εξόδου ADC, σύμφωνα με το οποίο ανάβει το κατάλληλο LED της θήρας PORTD.

Ζήτημα 3.3

Να γραφτεί ξανά ο κώδικας του ζητήματος 3.1 σε γλώσσα C:

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

int main ()
{
    unsigned char duty[] = {0x05, 0x1A, 0x2E, 0x43, 0x57, 0x6C,
0x80, 0x94, 0xA7, 0xBD, 0xD2, 0xE6, 0xFB};      // Ascending order

    // set TMR1A 8 bit
    TCCR1A = (1<<WGM10) | (1<<COM1A1);
    TCCR1B = (1<<WGM12) | (1<<CS11);

    DDRB |= 0b00111111;      // Output
    DDRD |= 0b00000000;      // Input


    int DC_VALUE = 6;              // Initialize DC to 50%
    OCR1AL = duty[DC_VALUE];

    while (1)
    {
        if (PIND == 0b11111101) {       // PD1
            while (PIND == 0b11111101);
            if (DC_VALUE < 12) {
                DC_VALUE++;
                OCR1AL = duty[DC_VALUE];
            }
        }
        if (PIND == 0b11111011) {       // PD2
```

```
            while (PIND == 0b11111011);
            if (DC_VALUE > 0) {
                DC_VALUE--;
                OCR1AL = duty[DC_VALUE];
            }
        }
    }
}
```

Επιπλέον, για τη μεταβολή της φωτεινότητας του led θα υπάρχουν δύο τρόποι λειτουργίας, το mode1 και το mode2. Στο mode1 η φωτεινότητα(Duty Cycle) θα αποθηκεύεται σε μια μεταβλητή με όνομα DC_VALUE και το πάτημα του μπουτόν PD1 θα αυξάνει τη φωτεινότητα ενώ το πάτημα του μπουτόν PD2 θα μειώνει τη φωτεινότητα. Στο mode2 η φωτεινότητα (Duty Cycle) θα ελέγχεται από το ποτενσιόμετρο POT1. Εάν πατηθεί το μπουτόν PD6 τότε θα επιλέγεται το mode1 ενώ αν πατηθεί το μπουτόν PD7 τότε θα επιλέγεται το mode2.

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

unsigned char duty[] = {0x05, 0x1A, 0x2E, 0x43, 0x57, 0x6C, 0x80,
0x94, 0xA7, 0xBD, 0xD2, 0xE6, 0xFB};

int main() {

    DDRB |= 0b00111111; //Output
    DDRC |= 0b00111111;
    DDRD |= 0b00000000; //Input

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

    //Set Vref = 5V
    ADMUX = (1<<REFS0) | (1<<ADLAR);

    //Enable ADC & set ADC prescaler to 128
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) |(1 <<
ADPS0);
```

```
unsigned char DC_VALUE = 6;
unsigned char MODE = 1;
OCR1AL = duty[DC_VALUE];

while(1) {

    if (PIND == 0b10111111) {    //PD6
        MODE = 1;
    }
    if (PIND == 0b01111111) {  //PD7
        MODE = 2;
    }

    while (MODE == 1) {
            PORTC = DC_VALUE;
            if (PIND == 0b11111101) {
                _delay_ms(200);
                if (DC_VALUE >= 12) ;
                else if (DC_VALUE < 12) {
                    DC_VALUE++;
                    OCR1AL = duty[DC_VALUE];
                }
            }
            if (PIND == 0b11111011) {
                    _delay_ms(200) ;
                    if (DC_VALUE > 0) {
                        DC_VALUE--;
                        OCR1AL = duty[DC_VALUE];
                }
            }
        if (PIND == 0b01111111) {  //PD7
            MODE = 2;
        }
        //return 0;
    }
    while (MODE == 2) {

        ADCSRA |= (1 <<ADSC);
```

```
            //Read the potentiometer value
            while (ADCSRA & (1 << ADSC)) ;
            uint16_t potentiometerValue = ADC;  //Store the ADC
result

            DC_VALUE = (uint8_t)((potentiometerValue*12)/1023);

            OCR1AL = duty[DC_VALUE];

            if (PIND == 0b10111111) {    //PD6
            MODE = 1;
            }
        }
    }
    return 0;
}
```

Το ζήτημα 3.3 είναι κατά το ήμισυ ίδιο με το ζήτημα 3.1 -συνεπώς δεν είναι σκόπιμο να εμβαθύνουμε στη λειτουργία του κώδικα, όταν το κουμπι PD6 είναι πατημένο. Αξίζει, ωστόσο, να αναφερθούμε στη λειτουργία του, όταν πατάμε το κουμπί PD7.
Ο κώδικας αρχικοποιεί τους ίδιους καταχωρητές, όπως στο ζήτημα 3.2. Το default setting είναι η λειτουργία του κώδικα του ζητήματος 3.1, ωστόσο σε όλο το εύρος του κώδικα ελέγχουμε αν το mode αλλάζει, δηλαδή αν πατιέται κάποιο από τα pins PD6 ή PD7. Για τη λειτουργία mode2, ενεργοποιείται το Single Conversion mode, και η έξοδος του ADC αποθηκεύεται στη μεταβλητή `potentiometerValue`. Κάνουμε κανονικοποίηση του αποτελέσματος, πολλαπλασιάζοντας με 12 και διαιρώντας με 1023, κι έτσι λαμβάνουμε την τιμή DC_VALUE, η οποία αντιστοιχεί στη θέση του πίνακα duty[ ]. Τέλος, η φωτεινότητα του LED PB1 αποτυπώνεται με τη χρήση της εντολής `OCR1AL = duty[DC_VALUE];`.