



3^η Εργαστηριακή Άσκηση

Πίκουλας Κωνσταντίνος 03120112
Στάμος Ανδρέας 03120018

1^η Άσκηση

Εξήγηση

Χρησιμοποιήσαμε τον Timer1 σε λειτουργία Fast PWM με την έξοδο σε non-inverting mode (η έξοδος μηδενίζει όταν ο μετρητής έχει τιμή ίσης με OCR1A -- στο Compare Match A -- και τίθεται σε 1 στην μηδενική τιμή του μετρητή -- στο BOTTOM). Η συχνότητα του Timer1 τίθεται στην συχνότητα ρολογιού της ΚΜΕ (16 MHz) για το ελάχιστο ripple της παραγόμενης τάσης. Βέβαια, πιθανώς, να μπορούσαμε να χρησιμοποιήσουμε μικρότερη συχνότητα μετρητή για να μειώσουμε την κατανάλωση ηλεκτρικής ισχύος, καθώς, σε συνδυασμό με το αναλογικό φίλτρο εξομάλυνσης, το λίγο αυξημένο ripple μπορεί να μην επηρεάζει το LED.

Το Duty Cycle ελέγχεται από τον χρόνο εντός της περιόδου του μετρητή όπου ο μετρητής θα μηδενίζει την έξοδο. Οι τιμές των χρόνων αυτών, αποθηκεύτηκαν στην μνήμη προγράμματος και ανακαλούνται κάθε φορά που ο χρήστης πατάει το κουμπί αλλαγής φωτεινότητας.

Με στόχο την ελάχιστη κατανάλωση ηλεκτρικής ισχύος, ανάμεσα στα πατήματα των κουμπιών αλλαγής φωτεινότητας βάζουμε τον AVR σε Idle Sleep απενεργοποιώντας, επίσης, όλα τα εξωτερικά περιφερειακά εκτός του Timer1. Για να λειτουργήσει αυτό, αντί να κάνουμε polling στην main για πατήματα των PD1/PD2 ρυθμίσαμε τις εισόδους PD1/PD2 να προκαλούν διακοπές (PCINT17/PCINT18) και βάλαμε όλη την λογική εξυπηρέτησής τους μέσα στην ρουτίνα εξυπηρέτησης διακοπής (μαζί με το debouncing). Μόλις ολοκληρωθεί η ρουτίνα εξυπηρέτησης διακοπής ξανακοιμίζουμε τον AVR. Στο Idle Sleep ο Timer1 εξακολουθεί να λειτουργεί, οπότε εξακολουθούμε να λαμβάνουμε έξοδο PWM.

Κώδικας

```
.include "m328Pbdef.inc"
.def temp = r17
.equ FOSC_MHZ=16
.equ DEBOUNCE_ms=5
.equ DEBOUNCE_NU=FOSC_MHZ*DEBOUNCE_ms
.org 0x0
    rjmp reset
.org 0x0a
    rjmp ISR_PCINT2
reset:
    //stack setup
    ldi temp, HIGH(RAMEND)
    out SPH, temp
    ldi temp, LOW(RAMEND)
    out SPL, temp
    //timer and fast-pwm setup
    ldi temp, (1 << WGM10 | 1 << COM1A1)
    sts TCCR1A, temp
    ldi temp, (1 << CS10 | 1 << WGM12)
    sts TCCR1B, temp
    //portd set as input, portb set as output
    clr temp
    out DDRD, temp
    ser temp
    out PORTD, temp
    out DDRB, temp //is that necessary?
    ;setup pcint at pd1 and pd2
    ldi temp, 1 << PCIE2
    sts PCICR, temp
    ldi temp, 1 << PCINT17 | 1 << PCINT18
    sts PCMSK2, temp
    //init at 50% dc
    ldi ZH, HIGH(DC_VALUES*2+6)
    ldi ZL, LOW(DC_VALUES*2+6)
    lpm temp, z
```

```

    sts OCR1AL, temp
    //setup sleep in idle mode only with timer1 turned on
    ldi temp, 1 << SE
    out SMCR, temp
    ldi temp, ~(1<< PRTIM1)
    sts PRRO, temp
    sei
main:
    sleep
    rjmp main

ISR_PCINT2:
    sbis PIND, 1
    rjmp inc_dc

    sbis PIND, 2
    rjmp dec_dc

    reti
inc_dc:
    //debouncing
    ldi r24, LOW(DEBOUNCE_NU)
    ldi r25, HIGH(DEBOUNCE_NU)
    rcall delay_mS
    sbis PIND, 1
    rjmp inc_dc
    //if 98% do nothing
    cpi ZL, LOW(DC_VALUES*2+DC_SIZE-1)
    brne do_inc
    cpi ZH, HIGH(DC_VALUES*2+DC_SIZE-1)
    brne do_inc
    reti
do_inc:
    adiw ZL, 1
    lpm temp, z
    sts OCR1AL, temp
    reti
dec_dc:
    ldi r24, LOW(DEBOUNCE_NU)
    ldi r25, HIGH(DEBOUNCE_NU)
    rcall delay_mS
    sbis PIND, 2
    rjmp dec_dc
    //if 2% do nothing
    cpi ZL, LOW(DC_VALUES*2)
    brne do_dec
    cpi ZH, HIGH(DC_VALUES*2)
    brne do_dec
    reti
do_dec:
    sbiw ZL, 1
    lpm temp, z
    sts OCR1AL, temp
    reti
.equ DC_SIZE = 13
DC_VALUES: .DW 0x1a05,0x432e,0x6c57,0x9480,0xbda9,0xe6d2,0x00fb
delay_mS:
    ldi r23, 249
loop_inn:
    dec r23
    nop
    brne loop_inn
    sbiw r24, 1
    brne delay_mS
    ret

```

2^η Άσκηση

Εξήγηση

Αρχικά σετάρουμε τον timer1 ώστε να παράγουμε τους pwm παλμούς. Έχουμε μια μεταβλητή «array_ptr» που δείχνει στο στοιχείο του πίνακα DC_VALUES που παράγει το ανάλογο duty_cycle. Οι τιμές έχουν επιλεχθεί ώστε να παράγουν κατά προσέγγιση το ποσοστό duty cycle που πρέπει κάθε φορά. Σετάρουμε τον adc να διαβάζει από το κανάλι εισόδου ADC1. Είναι right adjusted και θέτουμε prescaler 128. Άρα η συχνότητα τους adc είναι 16MHz/128 = 125 KHz. Κάθε φορά που πατάμε το μπουτόν PD4 (αφού το κάνουμε debounce) αυξάνουμε το Duty Cycle γράφοντας την τιμή από τον πίνακα DC_VALUES στον καταχωρητή OCR1AL (μόνο αυτόν χρειαζόμαστε καθώς ο pwm παλμός είναι 8 bits). Μετατρέπουμε την τιμή που διαβάζει ο ADC σε τάση εισόδου χρησιμοποιώντας την σχέση $V_{in} = ADC * \frac{V_{ref}}{2^n}$. Ύστερα ανάβουμε το ανάλογο LED της PORTD.

Κώδικας

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<util/delay.h>
#include<avr/interrupt.h>
#include<stdbool.h>
unsigned const char DC_VALUES[] = {5,26,47,68,89,110,128,152,173,194,215,236,250};
float voltage_input = 0.0;

int main(int argc, char** argv) {
    int array_ptr = 6;
    TCCR1A = (1 << WGM10) | (1 << COM1A1);
    TCCR1B = (1 << WGM12) | (1 << CS11);
    DDRB = 0xFF; // OUTPUT
    DDRD = 0xFF; // OUTPUT
    DDRC = 0; //INPUT
    OCR1AL = DC_VALUES[array_ptr];

    // REFSn[1:0]=01 => Vref = 5V
    // MUXn[3:0] = 0001
    // ADLAR = 1 => Left adjusted
    ADMUX = 0b01000001;

    //ADEN = 1 => ADC Enable
    //ADCS = 0 => No conversion (yet)
    // ADIE = 0 => Disable ADC Interrupt
    // ADPS[2:0]=111 => fADC = 16MHz/128=125KHz
    ADCSRA = 0b10000111;
    while(1) {
        if (!(PINC & 1<<4)) {
            while(!(PINC & 1<<4)) _delay_ms(5);
            if (!(array_ptr == 12))
                OCR1AL = DC_VALUES[++array_ptr];
        }
        if (!(PINC & 1<<5)) {
            while(!(PINC & 1<<5)) _delay_ms(5);
            if (!(array_ptr == 0))
                OCR1AL = DC_VALUES[--array_ptr];
        }
        _delay_ms(100);
        // Start conversion
        ADCSRA |= (1 << ADSC); // init conversion
        while(ADCSRA & 1<<ADSC); // it means adc is not finished
        // we need to divide by 2^10
        // dividing by 1024 means 10 shift to the right
        // we have ADCH:ADCL 16 bits and 10bits used
        // so when shifting 10 times right we get the value of ADCH in the ADCL reg
        // so just mess with ADCH reg
```

```

voltage_input = (float)ADC/1024 * 5.0;
if (voltage_input > 0 && voltage_input <= 0.625) PORTD = 0x01;
else if (voltage_input > 0.625 && voltage_input <= 1.25) PORTD = 0x02;
else if (voltage_input > 1.25 && voltage_input <= 1.875) PORTD = 0x04;
else if (voltage_input > 1.875 && voltage_input <= 2.5) PORTD = 0x08;
else if (voltage_input > 2.5 && voltage_input <= 3.125) PORTD = 0x10;
else if (voltage_input > 3.125 && voltage_input <= 3.75) PORTD = 0x20;
else if (voltage_input > 3.75 && voltage_input <= 4.375) PORTD = 0x40;
else if (voltage_input > 4.375 && voltage_input <= 5) PORTD = 0x80;
}
return 0;
}

```

3^η Άσκηση

Εξήγηση

Έχουμε ένα boolean flag «mode1» το οποίο όταν είναι true τότε τρέχει το mode1 ενώ όταν είναι false τρέχει το mode2. Στο mode1 όταν πατάμε το μπουτον PD1 τότε αυξάνεται το duty cycle κατά 8% ενώ όταν πατάμε το μπουτον PD2 μειώνεται κατά 8%. Όταν πατάμε το μπουτόν PD7 το mode αλλάζει σε mode2. Στο mode2 τρέχουμε τον ADC κάθε 100msec και θέτουμε το duty cycle από την τιμή που δίνει το ποτενσιόμετρο και διαβάζουμε από τον ADC. Όταν πατάμε το μπουτόν PD6 επιστρέφουμε στο mode1.

Κώδικας

```

#define F_CPU 16000000UL
#include<avr/io.h>
#include<util/delay.h>
#include<avr/interrupt.h>
#include<stdbool.h>
unsigned const char DC_VALUES[] = {5,26,47,68,89,110,128,152,173,194,215,236,250};
bool mode1 = true; // if mode1 = true then mode 1 else mode 2
int main(int argc, char** argv) {
    int array_ptr = 6;
    TCCR1A = (1 << WGM10) | (1 << COM1A1);
    TCCR1B = (1 << WGM12) | (1 << CS11);
    DDRD = 0x00; // INPUT
    DDRB = 0xFF; // OUTPUT
    OCR1AL = DC_VALUES[array_ptr];
    // init ADC
    ADMUX = 0b01100000;

    //ADEN = 1 => ADC Enable
    //ADCS = 0 => No conversion (yet)
    // ADIE = 0 => Disable ADC Interrupt
    // ADPS[2:0]=111 => fADC = 16MHz/128=125KHz
    ADCSRA = 0b10000111;
    while(1){
        while(mode1) {
            if (!(PIND & 1<<1)) {
                while(!(PIND & 1<<1)) _delay_ms(5);
                if (!(array_ptr == 12))
                    OCR1AL = DC_VALUES[++array_ptr];
            }
            if(!(PIND & 1<<2)) {
                while(!(PIND & 1<<2)) _delay_ms(5);
                if(!(array_ptr == 0))
                    OCR1AL = DC_VALUES[--array_ptr];
            }
        }
    }
}

```

```
    if (!(PIND & 1<<7)) {mode1 = false;}  
}  
while(!mode1) {  
    _delay_ms(100);  
    ADCSRA |= (1 << ADSC); // init conversion  
    while(ADCSRA & 1<<ADSC); // it means adc is not finished  
    OCR1AL = ADCH;  
    if (!(PIND & 1<<6)) {mode1 = true;}  
}  
}  
return 0;  
}
```