Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών ΕΜΠ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΑΚΑΔ. ΕΤΟΣ 2024-2025

3ή εργαστηριακή άσκηση.

Ομάδα : 58

Ονοματεπώνυμο : Φέζος Κωνσταντίνος / Α.Μ : 03118076

Ονοματεπώνυμο : Τσουκνίδας Οδυσσέας Αρθούρος Ρήγας / Α.Μ : 03120043

Ζήτημα 3.1

Στο συγκεκριμένο ζήτημα καλούμαστε να παράγουμε μια PWM κυματομορφή στον ακροδέκτη PB1 με συχνότητα 62500 Hz. Για να το πετύχουμε αυτό αρχικοποιούμε τον TMR1A σε λειτουργία 8-bit. Οι εντολές σε assembly που χρησιμοποιήσαμε θα συμπεριληφθούν στον πηγαίο κώδικα παρακάτω. Σκοπός του προγράμματος είναι να αποτυπώνεται το DUTY CYCLE μέσω της φωτεινότητας ενός LED το οποίο είναι συνδεδεμένο στον παραπάνω ακροδέκτη. Πιο συγκεκριμένα ελέγχουμε την τιμή του DUTY CYCLE μέσω των PD3 και PD4. Το πάτημα του μπουτόν PD3 αυξάνει το DUTY CYCLE κατά 8% ενώ το PD4 το μειώνει κατά την ίδια τιμή. Οι ακραίες τιμές είναι 98% και 2% αντίστοιχα. Για την καλύτερη επίλυση του προβλήματος αποθηκεύουμε τις τιμές που θα χρειαστούμε σε ένα πινακάκι που βρίσκεται στη flash memory. Όπως γνωρίζουμε από το εργαστήριο η τιμή του DUTY CYCLE ελέγχεται από τον OCR1A. Ο πηγαίος κώδικας σε assembly φαίνεται παρακάτω.

```
.include"m328PBdef.inc"
.def DC_VALUE = r18
//.equ X1 = 300

.org 0x00
    rjmp reset
.org 0x0100
table:
  .db 0, 8, 28, 48, 68, 88, 108, 128, 148, 168, 188, 208, 228, 248

reset:
; initialize stack pointer
    ldi r24, LOW(RAMEND)
    out SPL, r24
    ldi r24, HIGH(RAMEND)
    out SPH, r24

/*Set registers for delay
    ldi r24, low(X1)
    ldi r25, high(X1)
*/
    ldi r16, 0b00111111
    out DDRB, r16                   ; PB5-0 as output
    clr r16
    out DDRD, r16                   ; PORTD as input
    ;ser r16
```

```
    ;out DDRC, r16

    ldi r16, (1 << WGM10) | (1 << COM1A1)
    sts TCCR1A, r16
    ldi r16, (1 << WGM12) | (1<<CS10)
    sts TCCR1B, r16                         ; Fast PWM 8-bit,
prescaler = 1, non-inverse, connected to PB1
    ldi DC_VALUE, 7                 ; 50% duty cycle (for 8-bit, max
255) / 7th position on the table
    ldi ZL, low(table*2)
    ldi ZH, high(table*2)
    add ZL, DC_VALUE                ; Add the index (in r16) to the Z
register
    lpm    r17, Z                       ; Load the value from
program memory
    sts OCR1AL, r17                 ; Write duty cycle to OCR1A


main:
    ;out PORTC, DC_VALUE
    in r16, PIND

wait_to_unpress:
    in r19, PIND
    cpi r19, 0xFF                ;keep looping while button is still
pressed
    brne wait_to_unpress

    sbrs r16, 03               ; If PD3 is pressed then add 8%
    rjmp ADD8
    sbrs r16, 04               ; If PD4 is pressed then reduce 8%
    rjmp SUB8
    rjmp MAIN

add8:
    cpi DC_VALUE, 13                ; Check if we reached max value
    breq MAIN
    inc DC_VALUE                  ; Next table location
    ldi ZL, low(table*2)            ; Load low byte of the table
address into Z register
    ldi ZH, high(table*2)           ; Load high byte of the table
address into Z register
    add ZL, DC_VALUE                ; Add the index (in r16) to the Z
register
```

```
    lpm r17, Z                      ; Load the value from program
memory (Z points to the value)
    sts OCR1AL, r17                 ; Write duty cycle to OCR1A
    rjmp MAIN

sub8:
    cpi DC_VALUE, 1                 ; Check if we reached min value
    breq MAIN
    dec DC_VALUE                ; Previous Table Location
    ldi ZL, low(table*2)            ; Load low byte of the table
address into Z register
    ldi ZH, high(table*2)           ; Load high byte of the table
address into Z register
    add ZL, DC_VALUE                ; Add the index (in r16) to the Z
register
    lpm r17, Z                      ; Load the value from program
memory (Z points to the value)
    sts OCR1AL, r17                 ; Write duty cycle to OCR1A
    rjmp MAIN
```

Ζήτημα 3.2

Στο ζήτημα 3.2 μεταφράζουμε τον κώδικα του προηγούμενου ζητήματος σε γλώσσα C. Καθώς τώρα τα PD3 και PD4 θα είναι δεσμευμένα χρησιμοποιούμε τα PD5 και PD6 για να αυξήσουμε και να μειώσουμε το DUTY CYCLE αντίστοιχα. Σκοπός του προγράμματος τώρα είναι ο ADC να διαβάζει την DC τάση που παράγεται στην έξοδο του αναλογικού φίλτρου PB1_PWM με ακρίβεια 10 bit. Η μέση τιμή του ADC υπολογίζεται αθροίζοντας 16 διαδοχικές τιμές (η κάθε μέτρηση θα γίνεται ανά 100mS) και διαιρώντας με 16 αντίστοιχα. Η διαίρεση γίνεται με ολίσθηση κατά 4 θέσεις. Το πρόγραμμα μας ανάβει τα LED που μας ζητούνται σύμφωνα με το πινακάκι που φαίνεται παρακάτω :

Τιμή μέτρησης του ADC(ADCvalue)	LED ON
$0 \leq \text{ADCvalue} \leq 200$	PD0
$200 \leq \text{ADCvalue} \leq 400$	PD1
$400 \leq \text{ADCvalue} \leq 600$	PD2
$600 \leq \text{ADCvalue} \leq 800$	PD3
$800 \leq \text{ADCvalue}$	PD4

Ακολουθεί ο πηγαίος κώδικας σε C :

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

void adc_init(void) {
      // Initialize ADC
      // Set reference voltage to AVcc (with external capacitor at AREF
pin)
      ADMUX = (1 << REFS0);

      // Set input channel to ADC1 (=0001)
      ADMUX |= (1 << MUX0);

      // Set ADC pre-scaler to 128 (for 16 MHz clock, ADC clock = 16MHz
/128 = 125 kHz - within the range)
      ADCSRA = (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

      // Enable the ADC
      ADCSRA |= (1 << ADEN);
}

uint16_t adc_read(void) {
      // Set ADSC flag of ADSCRA
      ADCSRA |= (1 << ADSC);

      // Wait for ADSC flag to become '0'
     while (ADCSRA & (1 << ADSC));

     //Return 10-bit result
     return ADC;
}

int main(){

      adc_init();                //Initialize ADC

      int DC_VALUES[14];         //Create table for dc_values

      for (int i = 0; i < 13; i++){         //set up the DC_VALUE table
            DC_VALUES[i] = (8 + i*20);
```

```
    }

    //    set fast non-inverted PWM 8 bit and pre-scale  = 1
    TCCR1A = (1 << WGM10) | (1 << COM1A1);
    TCCR1B = (1 << WGM12) | (1 << CS10);

  DDRB = 0xFF;       //PB5-0 as output
  DDRC= 0x00;              //PORTC as input
    DDRD = 0b00011111;          //PD4-PD0 output the rest is input

    int index = 7;              //Duty Cycle =  50%
  OCR1AL = DC_VALUES[index];

    uint16_t ADC_RESULT = 0;
    int reset = 0;

    while(1){
        if(reset == 16){
            reset = 0;
            ADC_RESULT = (ADC_RESULT >> 4); //Divide by 2^4 = 16
         if (ADC_RESULT >= 0 & ADC_RESULT <= 200) {
            PORTD = 0x01;
         }
         else if(ADC_RESULT > 200 & ADC_RESULT <= 400) {
            PORTD = 0x02;
         }
         else if(ADC_RESULT > 400 & ADC_RESULT <= 600) {
            PORTD = 0x04;
         }
         else if(ADC_RESULT > 600 & ADC_RESULT <= 800) {
            PORTD = 0x08;
         }
         else {
            PORTD = 0x10;
         }
         ADC_RESULT =0;
        }
        _delay_ms(100);                         //Call delay
        ADC_RESULT += adc_read();        //Read ADC_value
        reset++;                                //Keep track of how
many values we have

      /*Increase or decrease the Duty Cycle
        based on which button is pressed*/
      unsigned char d = PIND, check = PIND;
```

```
        //wait for the button to be un-pressed
        while (!(check & (1 << PD5)) | !(check & (1<<PD6))) {
            check = PIND;
        }

        unsigned char dummy5 = d &0x20;
        unsigned char dummy6 = d &0x40;

           if((dummy5 ==0) && index < 12){ // with PD5 we increase
                index++;
                OCR1AL = DC_VALUES[index];
           }
           if((dummy6 ==0) && index > 0){ // with PD6 we decrease
                index--;
                OCR1AL = DC_VALUES[index];
        }



    }
}
```

Ζήτημα 3.3

Τέλος, στο ζήτημα 3.3 επεκτείνουμε τον παραπάνω κώδικα ώστε να υλοποιεί 2 νέες λειτουργίες. Αυτή τη φόρα θα υπάρχουν 2 modes λειτουργίας. Στο 1ο mode ελέγχουμε την τιμή του το DUTY CYCLE με το PD1 αυξάνοντας την και με το PD2 μειώνοντας την αντίστοιχα. Η υλοποίηση της συγκεκριμένης λειτουργίας γίνεται όπως και προηγουμένως. Με το πάτημα του PD7 επιλέγουμε να μπούμε στο mode 2 στο οποίο ελέγχουμε το DUTY CYCLE με το ποτενσιόμετρο 1 (POT1). Για να επανέλθουμε στο mode 1 πιέζουμε το PD6 . Ο πηγαίος κώδικας σε C ακολουθεί παρακάτω :

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

void adc_init(void) {
    // Initialize ADC
```

```
      // Set reference voltage to AVcc (with external capacitor at AREF
pin)
      ADMUX = (1 << REFS0);

      // Set ADC pre-scaler to 128 (for 16 MHz clock, ADC clock = 16MHz
/128 = 125 kHz - within the range)
      ADCSRA = (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

      // Enable the ADC
      ADCSRA |= (1 << ADEN);
}

uint16_t adc_read(void) {
      // Set ADSC flag of ADSCRA
      ADCSRA |= (1 << ADSC);

      // Wait for ADSC flag to become '0'
      while (ADCSRA & (1 << ADSC));

    //Return 10-bit result
    return ADC;
}

int main(){
      int DC_VALUES[14];          //Create table for dc_values

      for (int i = 0; i < 13; i++){          //set up the DC_VALUE table
            DC_VALUES[i] = (8 + i*20);
      }
    adc_init();             //Initialize ADC


      // set fast non-inverted PWM 8 bit and pre-scale  = 1
      TCCR1A = (1 << WGM10) | (1 << COM1A1);
      TCCR1B = (1 << WGM12) | (1<<CS10);

      DDRB = 0xFF;         //PORTB as output
    DDRC = 0x00;         //PORTC is left as input
      DDRD = 0x00;           //PORTD as input
      int index = 7;               //Duty Cycle =  50%
    OCR1AL = DC_VALUES[index];
    unsigned char mode1 = 1;
    unsigned char mode2 = 1;
    bool flag = true;
```

```
      while(1){
         // Set the desired mode
         mode1 = PIND & 0x40;
         mode2 = PIND & 0x80;
         if (mode1 ==0) {
             flag = true;
             OCR1AL = DC_VALUES[index];
         }
         if (mode2 ==0) {
             flag = false;
         }
         if(flag) {
             _delay_ms(100);
             unsigned char pd1 = PIND & 0x02, pd2 = PIND & 0x04, check
= PIND;

             //wait for the button to be un-pressed
             while (!(check & (1 << PD1)) | !(check & (1<<PD2))) {
                 check = PIND;
             }


             if((pd1 ==0) && index < 12){ // with PD1 we increase
                 index++;
                 OCR1AL = DC_VALUES[index];
             }
             if((pd2 ==0) && index > 0){ // with PD2 we decrease
                 index--;
                 OCR1AL = DC_VALUES[index];
             }
         }
         else {
             OCR1AL = adc_read() >> 2;
         }
     }
}
```