A close up of text

Description automatically generated

ΑΘΗΝΑ 18 Οκτωβρίου 2024

**3η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ**

**ΓΙΑ ΤΟ ΜΑΘΗΜΑ “Εργαστήριο Μικροϋπολογιστών”**

**ΟΜΑΔΑ 23**

**Συνεργάτες**

Νικόλαος Αναγνώστου Νικόλαος Λάππας

03121818 03121098

**Ζήτημα 3.1:**

Στην συγκεκριμένη άσκηση υπολογίσαμε αρχικά ένα table με τις ζητούμενες φωτεινότητες για το LED PB1, οι οποίες αποτελούν ποσοστό (8% διαφορά οι διαδοχικές τιμές) της μέγιστης επιτρεπτής φωτεινότητας (255 λόγω της 8-bit PWM). Αφού επιλέξουμε την κατάλληλη συχνότητα λειτουργίας του timer/counter1 (TCNT1) μέσω του καταχωρητή TCCR1A και ρυθμίσουμε τον ζητούμενο τρόπο λειτουργίας του timer/counter1 (TCNT1) μέσω του καταχωρητή TCCR1B (φαίνονται στα σχόλια του κώδικα οι αιτιολογήσεις), προχωράμε με το κύριο πρόγραμμα. Η όλη δουλειά γίνεται σε αυτή την ετικέτα με όνομα . Ο καταχωρητής DC\_VALUE περιέχει το index της εκάστοτε τιμής του πίνακα (αρχική τιμή 0x80, index = 7) και με βάση αυτόν γίνεται η αυξομείωση της φωτεινότητας αναλόγως πιο PIND έχει πατηθεί. Η διαδικασία φαίνεται εύκολα μέσα από τα σχόλια που παρατίθενται στον κώδικα.

.include "m328pbdef.inc"

.equ FOSC\_MHZ = 16 ; Microcontroller operating frequency in MHz

.equ DEL\_mS = 500 ; Delay in mS (valid number from 1 to 4095)

.equ DEL\_NU = FOSC\_MHZ \* DEL\_mS ; \_delay\_mS routine: (1000\*DEL\_NU+6) cycles

.equ req = 100 ;100 msec delay for better visualization

.def DC\_VALUE = r16 ; For Duty Cycle, the index for the correct position in the table

.org 0x0

rjmp reset

reset:

; Define a table that contains precomputed values of OCR1A for various Duty Cycles

OCR\_table:

.db 0x05, 0x1A, 0x2E, 0x43, 0x57, 0x6C, 0x80, 0x94, 0xA7, 0xBD, 0xD2, 0xE6, 0xFB

; Stack initialisation

ldi r24,LOW(RAMEND)

out SPL,r24

ldi r24,HIGH(RAMEND)

out SPH,r24

clr r24

; Initialize PB1 (OC1A) as output

ser r24

out DDRB, r24

; Initialize Timer1 -> TCCR1A in FAST PWM, 8-bit

; The OC1A output of PWM (non-inverting) timer1 will be connected to the PB1

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

sts TCCR1A, r24

; f\_PWM = 16MHz / (prescaler \* 256)

; 256 because we have 8-bit timer = (1 + TOP), where TOP = 255

; So if we choose prescaler = 1 (CS10 = 1), f\_PWM = 62,5KHz

; Initialize Timer1 -> TCCR1B in FAST PWM, 8-bit

ldi r24, (1 << CS10) | (1 << WGM12)

sts TCCR1B, r24

; Initialize PD3, PD4 as inputs

ldi r24, 0xE7

out DDRD, r24

clr r24

out PORTD,r24

ser r24

out DDRC,r24

clr r24

ldi DC\_VALUE, 7 ; Value 128 from the table (50%)

rcall update\_PWM ; Go to the table, index 7, and ‘print’ the desired value

loop:

in r26, PIND ; Read the state of PORTD

com r26 ; Reverse logic, so now if pressed = 1

andi r26, 0x18 ; Mask for PD3, PD4

cpi r26, 0x18 ; If both PINs are pressed, do nothing

breq loop ; Loop again

;LDI R31,0xFA

;STS OCR1AL,R31

out PORTC, DC\_VALUE

sbrc r26, 3 ; Skip the next instruction if PD3 is clear

rcall increaseDC\_triggered ; Jump to 'increaseDC\_triggered', return with ret

; We have reverse logic in input, so if PD3

; is pressed, then microprocessor understands it as 0

; and it does not skip the next operation

sbrc r26, 4 ; Skip the next instruction if PD4 is clear

rcall decreaseDC\_triggered ; Jump to 'decreaseDC\_triggered', return with ret

rjmp loop

; If PD3 is pressed

increaseDC\_triggered:

cpi DC\_VALUE, 13 ; Compare the two indexes, and if DC\_VALUE is greater

breq end\_increasing ; Don't increase (branch if greater or equal)

inc DC\_VALUE ; Increase the index, means increase another 8% the DC

; The following code in red is used to set the delay of 100 msec for better and more discrete visualization

push r24

push r25

push r26

push r27

in r1,sreg

push r1

ldi r24,LOW(req)

ldi r25,HIGH(req)

rcall wait\_x\_msec ;3 cycles

pop r1

out sreg,r1

pop r27

pop r26

pop r25

pop r24

rcall update\_PWM ; Go update the value from the table

end\_increasing:

ret

; If PD4 is pressed

decreaseDC\_triggered:

cpi DC\_VALUE, 1 ; Compare the two indexes, and if DC\_VALUE is less

breq end\_decreasing ; Don't decrease (branc if less or equal)

dec DC\_VALUE ; Decrease the index, means decrease another 8% the DC

push r24

push r25

push r26

push r27

in r1,sreg

push r1

ldi r24,LOW(req)

ldi r25,HIGH(req)

rcall wait\_x\_msec ;3 cycles

pop r1

out sreg,r1

pop r27

pop r26

pop r25

pop r24

rcall update\_PWM ; Go update the value from the table

end\_decreasing:

ret

; This label loads from program memory the precomputed values for the PWM Duty Cycle.

; We use DC\_VALUE as the index to select the appropriate value each time, which is

; the written to the OCR1A register -> adjust the brightness of LED PB1

; We use Z register because we address the program memory where we

; have stored the table's values

**update\_PWM:**

ldi ZH, high(OCR\_table) ; Load high byte of table address into ZH

ldi ZL, low(OCR\_table) ; Load low byte of table address into ZL

add ZL, DC\_VALUE ; Add the index (DC\_VALUE to the low byte of Z)

lpm r10,Z ; Loads the value from program memory LOW (OCR\_TABLE) into R0 by default

out PORTC,r10

STS OCR1AL,r10 ; Update the OCR1A register with the new value

ret

; The previous values we have calculated:

; For 8-bit PWM, Timer1 has a bandwith from 0 to 255 (0xFF)

; 2%: OCR1A = 2%\*255 = 5

; 10%: OCR1A = 10%\*255 = 25

; 18%: OCR1A = 46

; 26%: OCR1A = 66

; 34%: OCR1A = 86

; 42%: OCR1A = 107

; 50%: OCR1A = 128

; 58%: OCR1A = 148

; 66%: OCR1A = 168

; 74%: OCR1A = 188

; 82%: OCR1A = 209

; 90%: OCR1A = 229

; 98%: OCR1A = 250

wait\_x\_msec: (δεν παρατίθεται, είναι η ίδια με την αυτή που υλοποιήσαμε στην 1η εργαστηριακή άσκηση)

**Ζήτημα 3.2:**

Σε αυτή την άσκηση ζητήθηκε να υλοποιηθεί ίδια λογική με την άσκηση 3.1 με την διαφορά ότι τώρα κάθε 1600msec θα υπολογίζουμε την μέση τιμή 16 συνεχόμενων ADC τιμών της παραγόμενης PWM, και ανάλογα με την τιμή που θα προκύψει να ανάβουμε ένα συγκεκριμένο led της PORTD (pd0-pd4) σύμφωνα με τον πίνακα της εκφώνησης. Φροντίσαμε κάθε 100msec να ελέγχουμε αν έχει πατηθεί το PD5 (μείωση του PWM) ή το PD6 (αύξηση PWM) και μετά το πέρας της \_delay\_ms(100) να κάνουμε έναν υπολογισμό της ADC τιμής της εκάστοτε PWM και να το προσθέτουμε στην μεταβλητή result, η οποία περιέχει τις προηγούμενες μετατροπές (μέγιστο πλήθος = 16). Μετά το πέρας των κάθε 1600msec (16 μετρήσεις) κάνουμε right shift κατά 4 θέσεις την result για να υπολογίσουμε την μέση τιμή (πρακτικά διαιρέσαμε έτσι με τον συνολικό αριθμό των δειγμάτων), και ανάλογα με την τιμή της ανάβουμε το ειδικό λαμπάκι της PORTD (pd0-pd4). Έπειτα μηδενίζουμε το result και η διαδικασία επαναλαμβάνεται όσο το πρόγραμμα τρέχει. Παρακάτω ακολουθεί ο κώδικας που χρησιμοποιήθηκε:

#define F\_CPU 16000000UL // 16 MHz

#include <avr/io.h>

#include <util/delay.h>

#include <stdint.h>

uint16\_t result;

uint8\_t input;

int current\_index = 6;

unsigned const int OCR\_table[13] = {5, 25, 46, 66, 86, 107, 128, 148, 168, 188, 209, 229, 250};

int main(void){

DDRB = 0xFF; //PORTB outputs

DDRD = 0x9F; //5 first bits as output and the PD7 (last)

PORTB = 0x00;

PORTD = 0x00;

// Initialize TMR1A in fast PWM 8-bit mode with non-inverted output

// Prescaler = 1, to get 62.5kHz waveform in PB1

//TCCR1A = (1 << WGM10) | (1 << COM1A1);

//TCCR1B = (1 << CS10) | (1 << WGM12);

TCCR1B = 0x0C;

TCCR1A = 0x81;

// I have deleted the (0 << ... ) i had in assembly code, non necessary ones

/\* Chose ADC channel (ADC1) to read from PB1\_PWM, ends in ...0001

\* For voltage reference selection: REFS0 = 1, REFS1 = 0

\* Right adjustment: ADLAR = 0

\* ADC1: MUX3 = 0, MUX2 = 0, MUX1 = 0, MUX0 = 1

\*/

ADMUX = (1 << REFS0) | (1 << MUX0);

// Same as the above ADMUX = 0b01000001;

/\* Enable ADC: ADEN = 1

\* No conversion from analog to digital yet: ADSC = 0

\* Disable ADC interrupt: ADIE = 0

\* Prescaler: f\_ADC = 16MHz / prescaler and

\* 50kHz <= f\_ADC <= 200kHz for 10-bits accuracy. So,

\* division factor = 128 -> gives f\_ADC = 125kHz

\*/

ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); ;

// Same as the above ADCSRA = 0b10000111;

while(1)

{

int counter = 0;

result = 0;

while (counter!=16)

{

input = ~PIND;

input = (input & 0x60);

OCR1AL = OCR\_table[current\_index];

if(input == 0x60) continue;

else if ((input == 0x20) && (current\_index != 0)) { //lower if pd5

current\_index--;

//PORTC = OCR\_table[current\_index];

OCR1AL = OCR\_table[current\_index];

}

else if ((input == 0x40) && (current\_index != 12)){ //higher if pd6 is pressed

current\_index++;

//PORTC = OCR\_table[current\_index];

OCR1AL = OCR\_table[current\_index];

}

\_delay\_ms(100);

ADCSRA |= 0x40; //start ADC

while((ADCSRA&0x40)!= 0x00){} //while the conversion last hold fast

result = result + ADC;

counter++;

}

result = (result >> 4);

if ((result >= 0) && (result <= 200)) PORTD = 0x01;

else if (result <= 400) PORTD = 0x02;

else if (result <= 600) PORTD = 0x04;

else if (result <= 800) PORTD = 0x08;

else PORTD = 0x10;

}

}

**Ζήτημα 3.3:**

Στην άσκηση αυτή ακολουθούμε το ίδιο σκεπτικό με το ζήτημα 3.1, μόνο που πλέον έχουμε δυο modes για την αλλαγή της φωτεινότητας του LED PB1. Αρχική φωτεινότητα το 50%.

Mode1: Αν πατηθεί το PIN PD6, τότε τα PD1-PD2 καθορίζουν την αυξομείωση του PB1. Στην περίπτωση αυτή ελέγχουμε πιο από τα δυο έχει πατηθεί και εκτελούμε αύξηση κατά 8% της φωτεινότητας ή μείωση κατά 8%. Την αλλαγή πραγματοποιεί ο καταχωρητής ( με την βοήθεια ενός πίνακα που έχουμε ορίσει, ο οποίος περιέχει τις προϋπολογισμένες τιμές του ποσοστού φωτεινότητας κάθε φορά. Μέγιστη τιμή το 255 (αφού επιλέξαμε 8-bit έξοδο PWM, και κάναμε στρογγυλοποίηση προς τα κάτω).

Mode2: Αν πατηθεί το PIN PD7, τότε το POT1 καθορίζει την φωτεινότητα του PB1. Συγκεκριμένα καλείται η συνάρτηση , η οποία εκκινεί την μετατροπή του αναλογικού σήματος σε ψηφιακό, και περιμένει για όσο διαρκέσει η δειγματοληψία που πραγματοποιείται. Όταν ολοκληρωθεί μας επιστρέψει την digital τιμή που θέλουμε στον και αναθέτουμε αυτή την τιμή πάλι στον .

Την εναλλαγή των modes υλοποιεί η συνάρτηση , η οποία καλείται μέσα στο και μέσα σε κάθε προκειμένου να μπορεί να αλλάξει το mode ομαλά όποτε εμείς επιθυμούμε (φαίνονται με κόκκινο χρώμα οι θέσεις αυτές).

Ακολουθεί ο κώδικας που χρησιμοποιήσαμε:

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <util/delay.h>

unsigned char OCR\_table[13] = {0x05, 0x19, 0x2E, 0x42, 0x56, 0x6B, 0x80, 0x94, 0xA8, 0xBC, 0xD1, 0xE5, 0xFA};

int mode = 0; // Variable that indicates the mode

void PWM\_init()

{

// Initialize TMR1A in fast PWM 8-bit mode with non-inverted output

// Prescaler = 1, to get 62.5kHz waveform in PB1

TCCR1A = (1 << WGM10) | (1 << COM1A1);

TCCR1B = (1 << CS10) | (1 << WGM12);

}

void ADC\_init()

{

/\* Chose ADC channel (ADC0) to read from POT1, ends in ...0000

\* For voltage reference selection: REFS0 = 1, REFS1 = 0

\* Right adjustment: ADLAR = 0

\* ADC0: MUX3 = 0, MUX2 = 0, MUX1 = 0, MUX0 = 0

\*/

ADMUX = (1 << REFS0);

// Same as the above ADMUX = 0b01000000;

/\* Enable ADC: ADEN = 1

\* No conversion from analog to digital yet: ADSC = 0

\* Disable ADC interrupt: ADIE = 0

\* Prescaler: f\_ADC = 16MHz / prescaler and

\* 50kHz <= f\_ADC <= 200kHz for 10-bits accuracy. So,

\* division factor = 128 -> gives f\_ADC = 125kHz

\*/

ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); ;

// Same as the above ADCSRA = 0b10000111;

}

// Read the DC voltage from PB1\_PWM analog filter, and take the digital result

uint16\_t ADC\_conversion()

{

ADCSRA |= (1 << ADSC); // Start conversion from analog to digital

while(ADCSRA & (1 << ADSC)); // Wait for conversion to end

return ADC; // Return the value, (ADCH:ADCL)

}

void select\_mode()

{

if(PIND == 0b10111111) // If PD6 pressed, mode1

{

\_delay\_ms(100);

while(PIND != 0b11111111) {};// For debouncing, wait till all PINs unpressed

mode = 1; // Mode 1, waiting for PD1-PD2

}

if(!(PIND & (1 << PD7))) // If PD7 pressed, mode2

{

\_delay\_ms(100);

while(PIND != 0b11111111) {}; // For debouncing, wait till all PINs unpressed

mode = 2; // Mode 2, waiting for POT1

}

}

int main(void)

{

PWM\_init(); // Initialize PWM

// Initialize PD6-PD7 as input (clear their values)

// Indicate mode1 or mode2

// Initialize PD1-PD2 as input (clear their values)

// The ones that trigger increase-decrease of Duty Cycle

DDRB = 0b00111111; // as output

DDRD = 0x00000000; // as input

ADC\_init(); // Initialize ADC

int DC\_VALUE = 0x80; // Default Duty Cycle (50%)

int duty\_cycle\_index = 6; // Index in OCR\_table corresponding to 50% duty cycle

OCR1A = DC\_VALUE; // Set initial duty cycle

\_delay\_ms(100);

while(1)

{

select\_mode();

if(mode == 1){

while(mode == 1)

{

// Check if PD1 is pressed (reverse logic -> PD1 = 0)

if(PIND == 0b11111101)

{

if(duty\_cycle\_index < 12) // If DC < 98%

{

duty\_cycle\_index++;

DC\_VALUE = OCR\_table[duty\_cycle\_index];

OCR1A = DC\_VALUE;

\_delay\_ms(100);

}

}

// Check if PD2 is pressed (reverse logic -> PD2 = 0)

if(PIND == 0b11111011)

{

if(duty\_cycle\_index > 0) // If DC > 2%

{

duty\_cycle\_index--;

DC\_VALUE = OCR\_table[duty\_cycle\_index];

OCR1A = DC\_VALUE;

\_delay\_ms(100);

}

}

select\_mode();

}

//------------------------------------------------------------------

}

else if(mode == 2) // If PD7 pressed, mode2

{

while(mode == 2)

{

// Connection of ADC0 with POT1

uint16\_t ADC\_value = ADC\_conversion(); // Read POT1

DC\_VALUE = ADC\_value;

OCR1A = DC\_VALUE;

//\_delay\_ms(100); // Small delay for better performance

select\_mode();

}

}

}

}