ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 2 ΠΡΟΗΓΜΕΝΟΙ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΕΣ

Ομάδα

Ζήκος Σπύρος 1084581

Κυριακουλόπουλος Καλλίνικος 1084583

Αναλυτική Περιγραφή Κώδικα

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Παραθέτουμε δύο κώδικες: Ο πρώτος είναι για το απλό δωμάτιο χωρίς αμβλείες γωνίες και ο δεύτερος είναι για το δωμάτιο με τις αμβλείες γωνίες. Ο δεύτερος κώδικας λειτουργεί και για το πρώτο δωμάτιο. Γι’αυτό, περιγράφουμε μόνο τον κώδικα για το δωμάτιο με τις δύο αμβλείες γωνίες. Ο κώδικας χωρίς αμβλείες γωνίες είναι πιο απλός και ακολουθεί την ίδια λογική, για αυτό δεν θεωρήσαμε απαραίτητο να τον περιγράψουμε.**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ορίζουμε τους χρόνους Τ1, Τ2, Τ3 που ζητούνται στην εκφώνηση.

Υπολογισμός χρόνου:

ftimer = 20MHz/16 = 1.25 MHz

γενικά: value = T\*ftimer

Τ1 = 1 ms => value = 1250

T2 = 2 ms => value = 2500

T3 = 0.5 ms => value = 625

Επίσης, ορίζουμε κατώφλι για τον μπροστά αισθητήρα THRESHOLDL και τους πλαϊνούς αισθητήρες THRESHOLDH.

“

#define THRESHOLDH 10 // high threshold for ADC interrupt

#define THRESHOLDL 10 // low threshold for ADC interrupt

#define T1 1250 // = 1ms

#define T2 2500 // = 2ms

#define T3 625  // = 0.5ms

“

Δηλώνουμε συναρτήσεις που τις ορίζουμε αργότερα.

“

void front\_sensor(); // front sensor activation

void rl\_sensor(); // side sensor activation (right on normal route, left on inverse route)

void led\_direction(); // activates leds depending on the movement of the device

void init\_ADC(); // setup ADC

void timer\_func(); // setup timer and count T1 or T2 or T3

“

Ορίζουμε μεταβλητές (flags) οι οποίες υποδεικνύουν αν η συσκευή πρέπει να σταματήσει, ποιος timer είναι ενεργός, τον αριθμό των συνολικών στροφών, αν η συσκευή κινείται κανονικά ή ανάποδα, τις δεξιά στροφές και τις αριστερές στροφές.

”

int x = 0; // becomes 1 when device reaches it’s initial position

int timer = 1; // is 1 when counting T1, is 2 when counting T2, is 3 when counting T3

int turns\_counter = 0; // it’s counting turns of the device since it began moving

int direction = 0; // 0 = normal, 1 = inverse

int right\_turn = 0;

int left\_turn = 0;

“

int main()

Στην συνάρτηση αυτή κάνουμε setup τα LEDs του PORTD, το PIN5 του PORTF. Μετά, ρυθμίζουμε και ενεργοποιούμε τον ADC με την συνάρτηση init\_ADC(). Ύστερα, ρυθμίζουμε και ενεργοποιούμε τον timer ώστε να μετράει T1 με την συνάρτηση timer\_func(1). Τέλος, μπαίνουμε σε ατέρμονο βρόγχο ώστε το πρόγραμμα να περιμένει interrupts και να μην τελειώσει.

void initADC()

Η συνάρτηση αυτή θέλει να αρχικοποιήσει τον ADC για τον δεξιά αισθητήρα. Έτσι, ορίζει το resolution του ADC σε 8 bits, ενεργοποιεί τον ADC, ενεργοποιεί το PIN εισόδου του ADC και ενεργοποιεί το Debug Mode του ADC. Μετά, ορίζουμε το χαμηλό και υψηλό threshold, θέτουμε συγκεκριμένη τιμή στον interrupt control register ώστε να ενεργοποιηθεί το interrupt του high threshold και επιλέγουμε την λειτουργία η οποία θα προκαλέσει interrupt. Τέλος, ο ADC κάνει μία μετατροπή.

void front\_sensor()

Η συνάρτηση αυτή ενεργοποιεί το free-running mode του ADC, ορίζει την ανάλυση του ADC σε 8 bits και ενεργοποιεί τον ADC. Επίσης, ενεργοποιεί το interrupt για το low threshold και αρχίζει να κάνει μετατροπές.

void rl\_sensor()

Η συνάρτηση αυτή ορίζει την ανάλυση του ADC σε 8 bits και ενεργοποιεί τον ADC. Επίσης, ενεργοποιεί το interrupt για το high threshold και κάνει μία μετατροπή.

void led\_direction(int led)

Η συνάρτηση αυτή αναλόγως το όρισμα που θα της δώσουμε, ανάβει είτε το LED0, είτε το LED1, είτε το LED2, είτε και τα τρία LED μαζί.

void timer\_func(int t)

Η συνάρτηση αυτή ανάλογα με το όρισμα που θα της δώσουμε, βάζει τον timer TCA0 να μετράει είτε T1, είτε T2, είτε T3 και αλλάζει την μεταβλητή timer ώστε το πρόγραμμα να γνωρίζει πόσο μετράει ο timer κάθε φορά.

Στον ISR(ADC0\_WCOMP\_vect) καθαρίζουμε τα intflags στη συνέχεια ελέγχεται η κατεύθυνση της σκούπας (μπροστά ή ανάποδα) και ανάλογα αυξάνουμε ή μειώνουμε τον αριθμό των στροφών ώστε να καταλάβει η σκούπα στην ανάποδη διαδρομή πότε να σταματήσει, δηλαδή πρέπει να κάνει όσες στροφές έκανε στην κανονική πορεία για να βρεθεί εκεί που άρχισε. Έπειτα ελέγχεται αν ο counter των στροφών είναι -1 και η σκούπα εκτελεί ανάποδη πορεία, όταν ισχύει αυτή η συνθήκη το πρόγραμμα τερματίζει.

Στη συνέχεια γίνεται η διαδικασία των στροφών, ανάλογα με την κατεύθυνση και την αιτία που προκλήθηκε το interrupt (RES>WINHT ή RES<WINLT) η σκούπα εκτελεί την κατάλληλη στροφή. Επίσης στην μπροστινή πορεία αλλάζει και η τιμή των counters ώστε να γνωρίζουμε πόσες δεξιές και αριστερές στροφές έχει εκτελέσει η σκούπα για να σταματήσει όταν φτάσει στην θέση από την οποία ξεκίνησε. Στην διαδρομή με τις 2 αμβλείες γωνίες υπολογίζουμε ότι η σκούπα φτάνει στην αρχική της θέση όταν έχει κάνει 6 αριστερές και 2 δεξιές στροφές. Αυτό γενικεύεται με τη συνθήκη ότι η διαφορά αριστερών και δεξιών στροφών πρέπει να είναι 4.

Τέλος ελέγχεται στην περίπτωση της κανονικής πορείας πότε η διαφορά αριστερών και δεξιών στροφών είναι 4 ώστε να καταλάβει η σκούπα ότι έφτασε στην αρχική θέση και το πρόγραμμα να τερματίσει.

Στον ISR(TCA0\_CMP0\_vect) καθαρίζουμε τα intflags και ελέγχεται η τιμή του timer. Η τιμή που έχει δηλώνει ποιος χρόνος τελείωσε και προκλήθηκε η διακοπή. Επομένως θα πρέπει να οριστεί ο επόμενος χρόνο λόγω της εναλλαγής και να τεθεί ο κατάλληλος αισθητήρας (μπροστινός ή πλαϊνός). Όταν το interrupt έχει γίνει από τη λήξη του χρόνου Τ3 (χρόνος περιστροφής σκούπας) αλλάζει η τιμή της μεταβλητής για τη διεύθυνση της σκούπας.

Ο ISR(PORTF\_PORT\_vect) καλείται όταν προκληθεί interrupt από το PIN5 του PORTF, δηλαδή το PIN5 γίνει 1. Στην αρχή καθαρίζονται τα interrupt flags και απενεργοποιούνται οι διακοπές από το PIN5 του PORTF. Ακολούθως καλείται η συνάρτηση που ενεργοποιεί και τα 3 leds και αρχίζει η μέτρηση από τον timer για Τ3, όσο πρέπει να μείνουν και τα 3 leds ενεργά.

ΔΙΑΓΡΑΜΜΑTA ΡΟΗΣ

ROOM 1 (χωρίς αμβλείες γωνίες)

Εικόνα που περιέχει στιγμιότυπο οθόνης, διάγραμμα, κείμενο, Σχέδιο

Περιγραφή που δημιουργήθηκε αυτόματα

ROOM 2 (με δύο αμβλείες γωνίες)

Εικόνα που περιέχει διάγραμμα, κείμενο, τεχνικό σχέδιο, Σχέδιο

Περιγραφή που δημιουργήθηκε αυτόματα

Using: <https://app.diagrams.net>

Κώδικας

Δωμάτιο χωρίς αμβλείες γωνίες (μπροστινός αισθητήρας μόνο)

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#define THRESHOLDL 10

#define T3 625  // = 0.5ms

void led\_direction();

void init\_ADC();

int x = 0;

int turns\_counter = 0;

int direction = 0; // 0 = normal, 1 = inverse

int main(){

    PORTD.DIR |= 0b00000111; //PIN is output

    PORTD.OUT |= 0b00000101;//LED is off

    PORTD.OUTCLR = PIN1\_bm; //LED is on

    PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc;//enable switch

    init\_ADC();

    while(x == 0){

        sei();

    }

}

void init\_ADC() {

    //initialize the ADC for Free-Running mode

    ADC0.CTRLA |= ADC\_RESSEL\_8BIT\_gc;//10-bit resolution

    ADC0.CTRLA |= ADC\_FREERUN\_bm; //Free-Running mode enabled

    ADC0.CTRLA |= ADC\_ENABLE\_bm; //Enable ADC

    ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc;//The bit

    ADC0.DBGCTRL |= ADC\_DBGRUN\_bm; //Enable Debug Mode

    //Window Comparator Mode

    ADC0.WINLT |= THRESHOLDL; //Set threshold LOW = 10

    ADC0.INTCTRL |= ADC\_WCMP\_bm; //Enable Interrupts for WCM

    ADC0.CTRLE |= ADC\_WINCM0\_bm; //Interrupt when RESULT < WINLT

    sei();

    ADC0.COMMAND |= ADC\_STCONV\_bm; //Start Conversion

}

void led\_direction(int led) {

    switch (led) {

        case 0: //right

        PORTD.OUT |= 0b00000110;//LED is off

        PORTD.OUTCLR = PIN0\_bm; //LED is on

        break;

        case 1: //normal

        PORTD.OUT |= 0b00000101;//LED is off

        PORTD.OUTCLR = PIN1\_bm; //LED is on

        break;

        case 2: //left

        PORTD.OUT |= 0b00000011;//LED is off

        PORTD.OUTCLR = PIN2\_bm; //LED is on

        break;

        case 3: //reverse

        PORTD.OUTCLR = 0b00000111; //LED is on

    }

}

void timer\_t3() {

    TCA0.SINGLE.CNT = 0; //clear counter

    TCA0.SINGLE.CTRLB = 0; //Normal Mode

    TCA0.SINGLE.CMP0 = T3;//When CMP0 reaches this value -> interrupt

    TCA0.SPLIT.CTRLA = TCA\_SPLIT\_CLKSEL\_DIV16\_gc | TCA\_SPLIT\_ENABLE\_bm;

    TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm;

}

ISR(ADC0\_WCOMP\_vect){

    cli();

    //change RES here

    int intflags = ADC0.INTFLAGS;

    ADC0.INTFLAGS = intflags;

    if (direction == 0)

    turns\_counter++;

    else if (direction == 1)

    turns\_counter--;

    //in normal mode checks if vacuum has made 4 turns so has reached the start,

    //in reverse mode checks if vacuum has made the same number of turns as in normal

    //mode so the vacuum has reached the start

    if ((turns\_counter == -1 && direction == 1) || (turns\_counter == 4 && direction == 0)) {

        TCA0.SINGLE.CTRLA = 0; //Disable

        ADC0.INTCTRL = 0; //Disable Interrupts for WCM

        x = 1;

        return;

    }

    if (direction == 0) {

        led\_direction(2);

        led\_direction(1);

    } else if (direction == 1) {

        led\_direction(0);

        led\_direction(1);

    }

    sei();

}

ISR(TCA0\_CMP0\_vect){

    cli();

    TCA0.SINGLE.CTRLA = 0; //Disable

    int intflags = TCA0.SINGLE.INTFLAGS; //Procedure to

    TCA0.SINGLE.INTFLAGS=intflags; //clear interrupt flag

    direction = 1;

    led\_direction(1);

    sei();

}

ISR(PORTF\_PORT\_vect){

    cli();

    int y = PORTF.INTFLAGS; //Procedure to

    PORTF.INTFLAGS=y; //clear the interrupt flag

    PORTF.PIN5CTRL = 0;

    led\_direction(3);

    timer\_t3();//set timer here

    sei();

}

Δωμάτιο με 2 αμβλείες γωνίες (μπροστινός και πλαϊνοί αισθητήρες)

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#define THRESHOLDH 10 // high thresshold for ADC interrupt

#define THRESHOLDL 10 // low thresshold for ADC interrupt

#define T1 1250 // = 1ms

#define T2 2500 // = 2ms

#define T3 625  // = 0.5ms

void front\_sensor(); // front sensor activation

void rl\_sensor(); // side sensor activation (right on normal route, left on inverse route)

void led\_direction(); // activates leds depending on the movement of the device

void init\_ADC(); // setup ADC

void timer\_func(); // setup timer and count T1 or T2 or T3

int x = 0; // becomes 1 when device reaches it’s initial position

int timer = 1; // is 1 when counting T1, is 2 when counting T2, is 3 when counting T3

int turns\_counter = 0; // it’s counting turns of the device since it began moving

int direction = 0; // 0 = normal, 1 = inverse

int right\_turn = 0;

int left\_turn = 0;

int main(){

    PORTD.DIR |= 0b00000111; //PIN is output

    PORTD.OUT |= 0b00000101;//LED is off

    PORTD.OUTCLR = PIN1\_bm; //LED is on

    PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc;//enable switch

    init\_ADC();

    timer\_func(1);

    while(x == 0){

        sei();

    }

}

void init\_ADC() {

    //initialize the ADC for Single Conversion mode

    //right sensor first

    ADC0.CTRLA |= ADC\_RESSEL\_8BIT\_gc;//8-bit resolution

    ADC0.CTRLA |= ADC\_ENABLE\_bm; //Enable ADC

    ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc;//The bit

    ADC0.DBGCTRL |= ADC\_DBGRUN\_bm; //Enable Debug Mode

    //Window Comparator Mode

    ADC0.WINLT |= THRESHOLDL; //Set threshold LOW = 10

    ADC0.WINHT |= THRESHOLDH; //Set threshold HIGH = 10

    ADC0.INTCTRL |= ADC\_WCMP\_bm; //Enable Interrupts for WCM

    ADC0.CTRLE = 0x2; //RESULT > WINHT

    sei();

    ADC0.COMMAND |= ADC\_STCONV\_bm; //Start Conversion

}

void front\_sensor() {

    ADC0.CTRLA |= ADC\_FREERUN\_bm; //Free-Running mode enabled

    ADC0.CTRLA |= ADC\_RESSEL\_8BIT\_gc;//8-bit resolution

    ADC0.CTRLA |= ADC\_ENABLE\_bm; //Enable ADC

    ADC0.CTRLE = 0x1;

    ADC0.COMMAND |= ADC\_STCONV\_bm; //Start Conversion

}

void rl\_sensor() {

    ADC0.CTRLA |= ADC\_RESSEL\_8BIT\_gc;//8-bit resolution

    ADC0.CTRLA |= ADC\_ENABLE\_bm; //Enable ADC

    ADC0.CTRLE = 0x2;

    ADC0.COMMAND |= ADC\_STCONV\_bm; //Start Conversion

}

void led\_direction(int led) {

    switch (led) {

        case 0: //right

            PORTD.OUT |= 0b00000110;//LED is off

            PORTD.OUTCLR = PIN0\_bm; //LED is on

            break;

        case 1: //normal

            PORTD.OUT |= 0b00000101;//LED is off

            PORTD.OUTCLR = PIN1\_bm; //LED is on

            break;

        case 2: //left

            PORTD.OUT |= 0b00000011;//LED is off

            PORTD.OUTCLR = PIN2\_bm; //LED is on

            break;

        case 3: //reverse

            PORTD.OUTCLR = 0b00000111; //LED is on

    }

}

void timer\_func(int t) {

    timer = t;

    TCA0.SINGLE.CNT = 0; //clear counter

    TCA0.SINGLE.CTRLB = 0; //Normal Mode

    TCA0.SINGLE.CMP0 = (t==1)?T1:(t==2)?T2:(t=3)?T3:0;//When CMP0 reaches this value -> interrupt

    TCA0.SPLIT.CTRLA = TCA\_SPLIT\_CLKSEL\_DIV16\_gc | TCA\_SPLIT\_ENABLE\_bm;

    TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm;

}

ISR(ADC0\_WCOMP\_vect){

    cli();

    //change RES here

    int intflags = ADC0.INTFLAGS;

    ADC0.INTFLAGS = intflags;

    if (direction == 0)

        turns\_counter++;

    else if (direction == 1)

        turns\_counter--;

    if (turns\_counter == -1 && direction == 1) {

        TCA0.SINGLE.CTRLA = 0; //Disable

        ADC0.INTCTRL = 0; //Disable Interrupts for WCM

        x = 1;

        return;

    }

    if (direction == 0) {

        if (ADC0.RES < THRESHOLDL) {

            left\_turn++;

            led\_direction(2);

            led\_direction(1);

        } else if (ADC0.RES > THRESHOLDH) {

            right\_turn++;

            led\_direction(0);

            led\_direction(1);

        }

    } else if (direction == 1) {

        if (ADC0.RES < THRESHOLDL) {

            led\_direction(0);

            led\_direction(1);

        } else if (ADC0.RES > THRESHOLDH) {

            led\_direction(2);

            led\_direction(1);

        }

    }

    if (left\_turn-right\_turn == 4 && direction == 0) {

        TCA0.SINGLE.CTRLA = 0; //Disable

        ADC0.INTCTRL = 0; //Disable Interrupts for WCM

        x = 1;

        return;

    }

    sei();

}

ISR(TCA0\_CMP0\_vect){

    cli();

    TCA0.SINGLE.CTRLA = 0; //Disable

    int intflags = TCA0.SINGLE.INTFLAGS; //Procedure to

    TCA0.SINGLE.INTFLAGS=intflags; //clear interrupt flag

    if (timer ==  1) {

        ADC0.CTRLA = 0b00000000; //clear CTRLA

        //change RES here

        front\_sensor();

        timer\_func(2);

    } else if (timer == 2) {

        ADC0.CTRLA = 0b00000000; //clear CTRLA

        //change RES here

        rl\_sensor();

        timer\_func(1);

    } else if (timer == 3) {

        direction = 1;

        led\_direction(1);

        rl\_sensor();

        timer\_func(1);

    }

    sei();

}

ISR(PORTF\_PORT\_vect){

    cli();

    int y = PORTF.INTFLAGS; //Procedure to

    PORTF.INTFLAGS=y; //clear the interrupt flag

    PORTF.PIN5CTRL = 0;

    led\_direction(3);

    timer\_func(3);

    sei();

}