Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Προηγμένοι Μικροεπεξεργαστές

Εργαστηριακή Άσκηση 2

Κατσαρός Ανδρέας 1084522
Ποταμιάνος Άγγελος Νικόλαος 1084537

Πάτρα, 2023-24

Ερωτήματα Εργαστηριακής Άσκησης 2

1 Υλοποιήστε τον κώδικα της κίνησης της οικιακής συσκευής, όταν το δωμάτιο είναι τετράγωνο με 90^{o} γωνίες (επομένως δεν χρειάζεται ο δεξιά αισθητήρας και η δεύτερη λειτουργία του ADC).

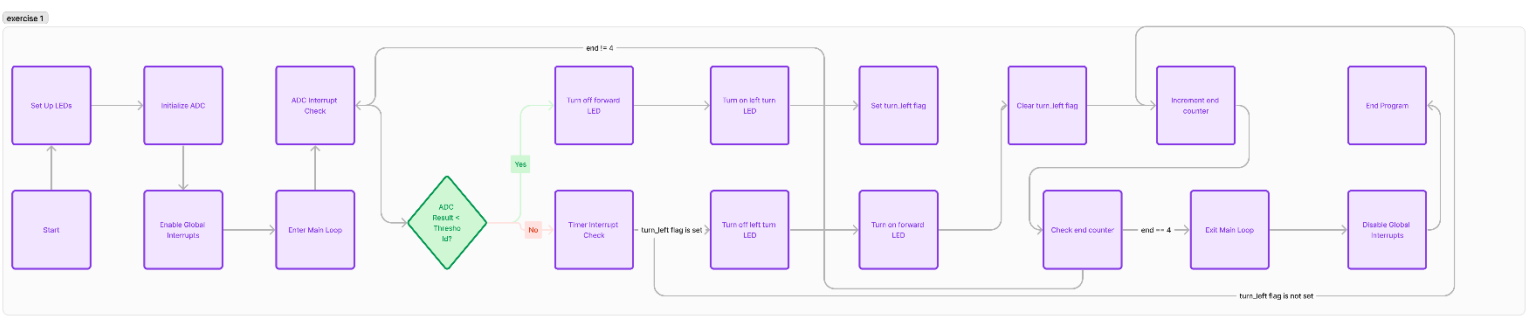
2 Υλοποιήστε τον κώδικα της κίνησης για ένα τυχαίο δωμάτιο που περιέχει και δύο αμβλείες (270ο) γωνίες, εδώ εισάγεται και η δεύτερη λειτουργία του ADC. Στο παρακάτω σχήμα 2.2 παρουσιάζεται ένα παραδειγματικό τέτοιο δωμάτιο για την σωστή κατανόηση του προβλήματος. Ωστόσο, η συσκευή θα πρέπει να μπορεί να ανταποκριθεί σωστά και σε ένα άγνωστο δωμάτιο.

3 Υλοποιήστε την ανάποδη λειτουργία.

Μπορείτε να έχετε πρόσβαση και σε ολόκληρο τον κώδικα στο τέλος της αναφοράς . Παρακάτω επεξηγούμε αναλυτικά τον κώδικα για την υλοποίηση των ερωτημάτων .

Ερώτημα 1:

Παράθεση Διαγράμματος Ροής:



Σκοπός του κώδικα:

Σκοπός είναι η προσομοίωση της λειτουργίας μιας έξυπνης οικιακής συσκευής που κινείται στον χώρο ενός άδειου δωματίου. Ξεκινάει από μία γωνία του δωματίου και ο σκοπός της είναι να σχεδιάσει το περίγραμμά του όταν το δωμάτιο είναι τετράγωνο με 90^o γωνίες..

Επεξήγηση βασικών σημείων του κώδικα:

```
// Initialize the ADC for Free-Running mode
ADC0.CTRLA = ADC_RESSEL_10BIT_gc | ADC_FREERUN_bm | ADC_ENABLE_bm;
ADC0.MUXPOS = ADC_MUXPOS_AIN7_gc; // Select AIN7 for the front sensor
ADC0.DBGCTRL = ADC_DBGRUN_bm; // Enable Debug Mode
ADC0.WINLT = 5; // Set threshold
ADC0.INTCTRL = ADC_WCMP_bm; // Enable Interrupts for Window Comparator Mode
ADC0.CTRLE = ADC_WINCM0_bm; // Interrupt when RESULT < WINLT
sei(); // Enable global interrupts
ADC0.COMMAND = ADC_STCONV_bm; // Start Conversion
```

Σύμφωνα με τις διαφάνειες έχουμε ότι ο κώδικας:

-Διαμορφώνει τον ADC σε λειτουργία Free-Running με ανάλυση 10 bit και τον ενεργοποιεί.

-Ενεργοποιείται η λειτουργία δημιουργίας διακοπών (interrupts) → Window Comparator Interrupt Enable bit (WCOMP) στον Interrupt Control register (ADCn.INTCTRL).

-Επιλέγει το AIN7 ως κανάλι εισόδου για τον μπροστινό αισθητήρα.

-Ορίζει μια τιμή threshhold (5) για τη λειτουργία Window Comparator Mode, ενεργοποιώντας μια διακοπή όταν το αποτέλεσμα του ADC είναι κάτω από αυτό το κατώφλι.

-Ενεργοποιούνται οι interrupts ώστε να είναι δυνατή η κλήση των isr.

```
if (turn_left) {
    // Configure timer for left turn delay
    TCA0.SINGLE.CNT = 0; // Clear counter
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc; // Set to Normal Mode
    TCA0.SINGLE.CMP0 = ped; // Set compare value for interrupt
    TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc | 1; // Set clock and enable
    TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm; // Enable compare interrupt
    while (turn_left) {
        // Wait for turn to complete, interrupt will clear turn_left
    }
    PORTD.OUT |= (PIN0_bm | PIN1_bm | PIN2_bm); // Turn off all movement LEDs
    PORTD.OUTCLR = PIN1_bm; // Resume forward movement
    forward = 1; // Set forward movement flag
}
```

-Ελέγχει συνεχώς αν η συσκευή έχει ολοκληρώσει την πλοήγηση στο τετράγωνο

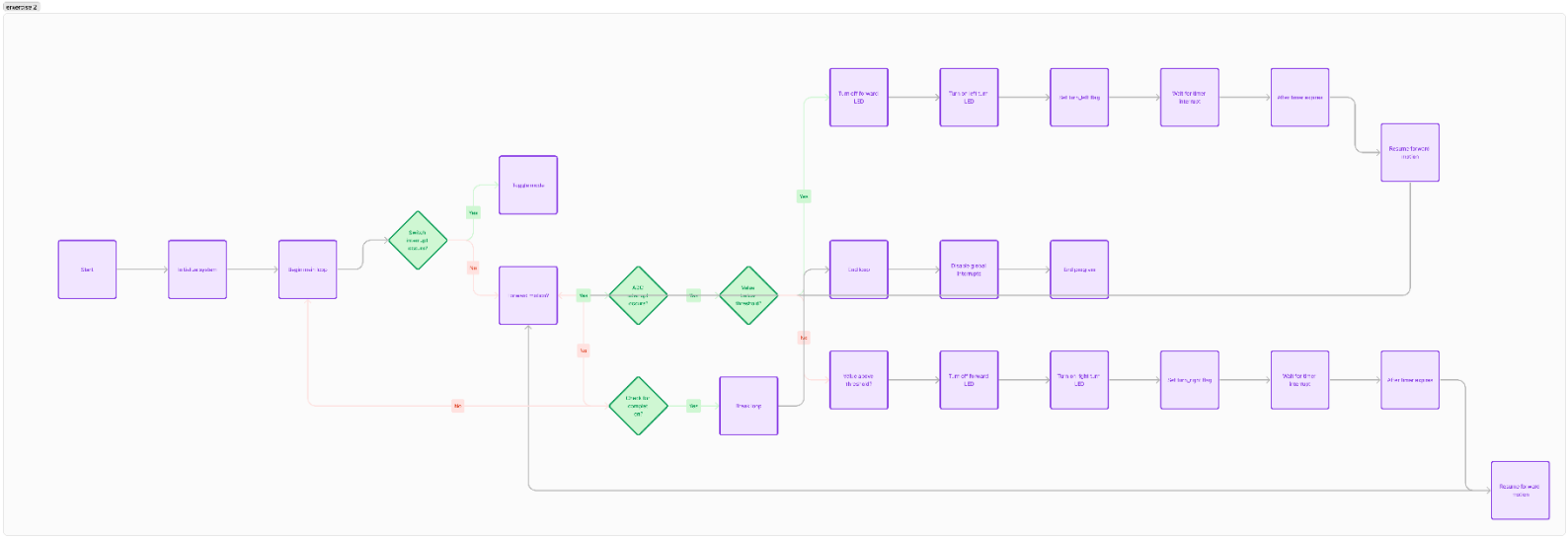
-Εάν κινείται προς τα εμπρός, διασφαλίζει ότι το LED προς τα εμπρός είναι αναμμένο.

-Εάν στρίβει αριστερά, ρυθμίζει ένα χρονόμετρο για καθυστέρηση, προσομοιάζοντας το χρόνο που απαιτείται για μια αριστερή στροφή, και περιμένει το χρονόμετρο να ολοκληρώσει τη στροφή.

Τέλος έχουμε δημιουργήσει τις απαραίτητες ISR , timer & adc,οι οποίες παρατίθενται στον κώδικα.

Ερώτημα 2:

Παράθεση Διαγράμματος Ροής:



Σκοπός του κώδικα:

Εδώ, είμαστε σε τυχαίο δωμάτιο που περιέχει και δύο αμβλείες (270ο) γωνίες, εδώ εισάγεται και η δεύτερη λειτουργία του ADC.

Επεξήγηση βασικών σημείων του κώδικα:

```
void check_right(){
    // Temporarily disable free-running mode to check right
    ADC0.CTRLA &= ~ADC_FREERUN_bm; // Disable free-running
    ADC0.MUXPOS &= ~ADC_MUXPOS_AIN7_gc; // Change MUXPOS to right sensor if needed
    // ADC0.MUXPOS |= ADC_MUXPOS_AINx_gc; // Adjust to select the right sensor
    ADC0.COMMAND = ADC_STCONV_bm; // Start single conversion for right sensor
    _delay_ms(1); // Delay for ADC conversion to complete
    frunning_ADC0(); // Return to free-running mode for front sensor
}

void turning_timer(){
    // Set the timer for turning
    TCA0.SINGLE.CNT = 0; // Clear counter
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc; // Normal Mode
    TCA0.SINGLE.CMP0 = ped; // When reaches this value -> interrupt CLOCK Frequency/1024
    TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc | TCA_SINGLE_ENABLE_bm; // Set clock and enable
    TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm; // Interrupt Enable
}
```

- **`check_right()` Λειτουργία**:

    - Απενεργοποιεί προσωρινά τη λειτουργία Free-Running του ADC για να εκτελέσει μια απλή μετατροπή, ελέγχοντας αν υπάρχει χώρος στα δεξιά της συσκευής για να στρίψει.
    - Αυτή η λειτουργία είναι κρίσιμη για την πλοήγηση σε χώρους με αμβλείες γωνίες, επιτρέποντας στη συσκευή να αποφασίσει πότε θα στρίψει δεξιά με βάση την απουσία τοίχου.
    - Μετά τον έλεγχο της δεξιάς πλευράς, ενεργοποιεί εκ νέου τη λειτουργία Free-Running για συνεχή παρακολούθηση του μπροστινού αισθητήρα.

- **λειτουργία `turning_timer()`**:
    - Χρησιμοποιείται για την προσομοίωση της διάρκειας τόσο των αριστερών όσο και των δεξιών στροφών.

```
void initialize_ADC0(){
    ADC0.CTRLA = ADC_RESSEL_10BIT_gc | ADC_ENABLE_bm; // 10-bit resolution, Enable ADC
    ADC0.MUXPOS = ADC_MUXPOS_AIN7_gc; // Select ADC channel (front sensor)
    ADC0.DBGCTRL = ADC_DBGRUN_bm; // Enable debug mode
}

void frunning_ADC0(){
    ADC0.CTRLA |= ADC_FREERUN_bm; // Free-running mode enabled
    ADC0.WINLT = limit; // Set the lower threshold for the front sensor
    ADC0.WINHT = 0xFFFF; // Set the upper threshold to max (unused in this context)
    ADC0.INTCTRL = ADC_WCMP_bm; // Enable interrupts for Window Comparator Mode
    ADC0.CTRLE = ADC_WINCM0_bm; // Interrupt when RESULT < WINLT (front sensor)
    ADC0.MUXPOS = ADC_MUXPOS_AIN7_gc; // Select the front sensor channel
}

void start_ADC0(){
    ADC0.COMMAND = ADC_STCONV_bm; // Start Conversion (front sensor)
}
```

- **Λειτουργίες `initialize_ADC0()`, `free_mode_ADC0()` και `start_ADC0()`**:
    - Αυτές οι συναρτήσεις έχουν προσαρμοστεί για τη διαμόρφωση του ADC τόσο για τις μετρήσεις που απαιτούνται για τον έλεγχο της δεξιάς πλευράς.
    - Επιτρέπουν τη δυναμική λήψη αποφάσεων κατά τη διάρκεια της πλοήγησης.

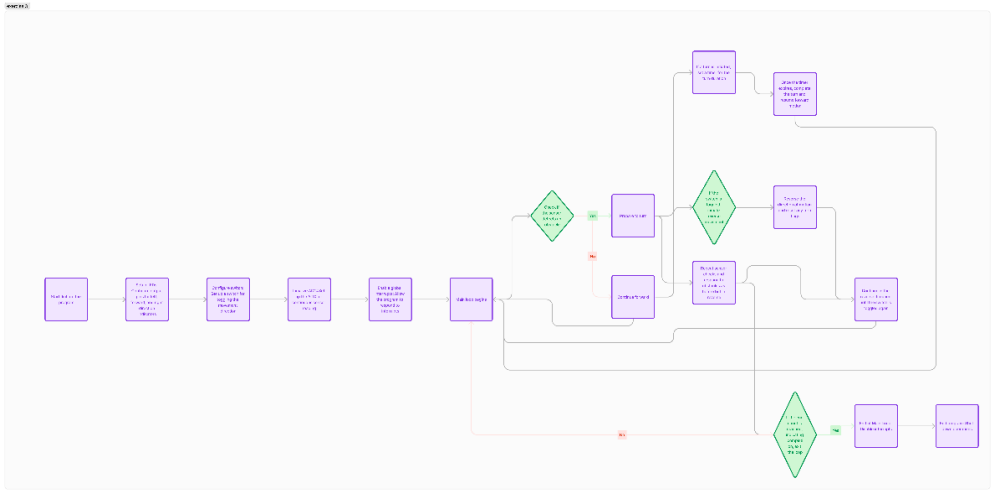
```
// Complete the ISR for the TCA0 compare match
ISR(TCA0_CMP0_vect) {
    TCA0.SINGLE.CTRLA = 0; // Disable the timer
    TCA0.SINGLE.INTFLAGS = TCA0.SINGLE.INTFLAGS; // Clear the timer interrupt flag

    // Decide what to do based on the flags set
    if (turn_left) {
        // Turn left logic
        turn_left = 0; // Reset the turn_left flag
        forward = 1; // Set forward to indicate moving straight after the turn
        end++; // Increment the end counter as we've completed a left turn
        } else if (turn_right) {
        // Turn right logic
        turn_right = 0; // Reset the turn_right flag
        forward = 1; // Set forward to indicate moving straight after the turn
        // Note: Depending on how you want to handle turns, you may want to adjust the end counter differently
        end--; // Decrement the end counter as we've completed a right turn
    }

    // Turn off all LEDs and then turn on the forward LED
    PORTD.OUT |= (PIN0_bm | PIN1_bm | PIN2_bm);
    PORTD.OUTCLR = PIN1_bm; // Forward LED on
}
```

- **ISR σύγκρισης χρονοδιακόπτη (`ISR(TCA0_CMP0_vect)`):**
    - Χειρίζεται την ολοκλήρωση τόσο των αριστερών όσο και των δεξιών στροφών, σταματώντας τον timer, κάνοντας clear τα flags.

    - Αυτό το ISR διασφαλίζει ότι μετά την ολοκλήρωση μιας στροφής, η συσκευή μπορεί να συνεχίσει την κίνησή της προς τα εμπρός, ενημερώνοντας ανάλογα την κατάσταση κατεύθυνσής της.

Ερώτημα 3:

Παράθεση Διαγράμματος Ροής:



Σκοπός του κώδικα:

Εδώ, είμαστε σε τυχαίο δωμάτιο και προσπαθούμε να υλοποιήσουμε την ανάποδη λειτουργία.

Επεξήγηση βασικών σημείων του κώδικα:

```
    PORTD.DIRSET = PIN0_bm | PIN1_bm | PIN2_bm; // 2:left, 1:forward, 0:right
    PORTD.OUT |= (PIN0_bm | PIN2_bm); // Start going forward

    // Configure switch on PORTF
    PORTF.DIR &= ~PIN5_bm; // Set PIN5 as input
    PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc; // Enable pull-up and interrupt on both edges
```

```
// Function to toggle the movement direction
void toggle_direction() {
    // Change the direction of movement
    forward = !forward; // Toggle the forward flag
    // Update LEDs or other indicators if needed
    // ... (additional logic to physically reverse the direction, if needed)
    inverse = 0; // Reset the inverse flag after handling the reverse operation
}
```

Διαμόρφωση διακόπτη pin: Διαμορφώνεται ένας διακόπτης στο **`PORTF`**, συγκεκριμένα στο **`PIN5`**.

Όπως εξηγήθηκε στις διαλέξεις, αυτός ο ακροδέκτης ορίζεται ως είσοδος με ενεργοποιημένη αντίσταση pull-up και οι διακοπές ρυθμίζονται ώστε να ενεργοποιούνται και στις δύο ακμές. Αυτή η ρύθμιση χρησιμεύει ως εντολή για την εναλλαγή της κατεύθυνσης κίνησης της συσκευής.

- **`toggle_direction()` Λειτουργία**: Αυτή η συνάρτηση συμβολίζει τη φυσική ενέργεια της αντιστροφής της κατεύθυνσης, η οποία, σε μια πραγματική εφαρμογή, θα περιελάμβανε την αντιστροφή των κατευθύνσεων των κινητήρων ή παρόμοιες ενέργειες.

```
ISR(TCA0_CMP0_vect) {
    TCA0.SINGLE.CTRLA = 0; // Disable the timer
    TCA0.SINGLE.INTFLAGS = TCA0.SINGLE.INTFLAGS; // Clear the timer interrupt flag

    // Decide what to do based on the flags set
    if (turn_left) {
        // Turn left logic
        turn_left = 0; // Reset the turn_left flag
        forward = 1; // Set forward to indicate moving straight after the turn
        end++; // Increment the end counter as we've completed a left turn
        } else if (turn_right) {
        // Turn right logic
        turn_right = 0; // Reset the turn_right flag
        forward = 1; // Set forward to indicate moving straight after the turn
        // Note: Depending on how you want to handle turns, you may want to adjust the end counter differently
        end--; // Decrement the end counter as we've completed a right turn
    }

    // Turn off all LEDs and then turn on the forward LED
    PORTD.OUT |= (PIN0_bm | PIN1_bm | PIN2_bm);
    PORTD.OUTCLR = PIN1_bm; // Forward LED on
}
```

- **ISR σύγκρισης (`ISR(TCA0_CMP0_vect)`)**: Οι προσαρμογές σε αυτό το ISR προσαρμόζονται στην ολοκλήρωση των ενεργειών στροφής, είτε πρόκειται για αριστερή, είτε για δεξιά, είτε για αντιστροφή κατεύθυνσης. Ο ISR διασφαλίζει ότι μετά την ολοκλήρωση μιας στροφής ή την αντιστροφή της κατεύθυνσης, η συσκευή συνεχίζει την προβλεπόμενη κίνησή της, με τη σημαία **`προώθησης`** να προσαρμόζεται ανάλογα.

ΠΛΉΡΗΣ ΚΏΔΙΚΑΣ:

1.c:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

int turn_left = 0;
int forward = 1;
int turn_right = 0;

int end = 0;

#define ped 10

int main() {
      PORTD.DIRSET = PIN0_bm | PIN1_bm | PIN2_bm; // 2:left 1:forward
0:right
      PORTD.OUT &= ~(PIN1_bm); // brosta kai anoigei LED1

      // arxikopoihsh ADC gia Free-Running mode
      ADC0.CTRLA = ADC_RESSEL_10BIT_gc | ADC_FREERUN_bm | ADC_ENABLE_bm;
      ADC0.MUXPOS = ADC_MUXPOS_AIN7_gc; // epilogi AIN7 gia brostino sensor
      ADC0.DBGCTRL = ADC_DBGRUN_bm; // Enable Debug Mode
      ADC0.WINLT = 5; // thetw katofli
      ADC0.INTCTRL = ADC_WCMP_bm; // Enable Interrupts for Window
Comparator Mode
      ADC0.CTRLE = ADC_WINCM0_bm; // Interrupt otan RESULT < WINLT
      sei(); // Enable global interrupts
```

```
	ADC0.COMMAND = ADC_STCONV_bm; // ksekinaei metatropi

	while (1) {
		if (end == 4) break; // stamataei afou kanei to tetragwno
dwmatio

		if (forward) {
			PORTD.OUT &= ~(PIN1_bm); // Epivevaiwsi oti einai anoixto
to led
		}

		if (turn_left) {
			TCA0.SINGLE.CNT = 0; // katharismos counter
			TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc; // Set
se Normal Mode
			TCA0.SINGLE.CMP0 = ped; // Set compare value gia
interrupt
			TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc | 1; //
Set clock kai enable
			TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm; // energopoisi
compare interrupt
			while (turn_left) {
				// perimenw na stripsei,to interrupt tha katharisei
to turn_left
			}
			PORTD.OUT |= (PIN0_bm | PIN1_bm | PIN2_bm); // Turn off
all movement LEDs
			PORTD.OUTCLR = PIN1_bm; // sinexizei brosta
			forward = 1; // Set forward movement flag
		}
	}
	cli(); // apenergopoihsh global interrupts
	return 0;
}

ISR(ADC0_WCOMP_vect) {
	cli();
	// katharismos ADC interrupt flag
	ADC0.INTFLAGS = ADC0.INTFLAGS;

	PORTD.OUT |= (PIN0_bm | PIN1_bm | PIN2_bm); // apenergopoihsh olwn
twn LED kinisewn
	PORTD.OUTCLR = PIN2_bm; // anoigo led aristeris kinisis
	forward = 0;
	turn_left = 1; // deixnw oti kanei aristera
	sei();
}

ISR(TCA0_CMP0_vect) {
	// Disable timer and clear flag
	TCA0.SINGLE.CTRLA = 0;
	TCA0.SINGLE.INTFLAGS = TCA0.SINGLE.INTFLAGS;

	turn_left = 0; // katharismos left flag
	end++; // auksanw metriti gwniwn
}
```

2.c:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

void check_right();
void turning_timer();
void initialize_ADC0();
void free_mode_ADC0();
void start_ADC0();

int turn_left=0;
int forward=1;
int turn_right=0;

int end=0;

#define ped 10
#define limit 10  // katofli gia entopismo tixou

int main(){
    PORTD.DIRSET = PIN0_bm|PIN1_bm|PIN2_bm; // 2:left, 1:forward, 0:right
    PORTD.OUT |= (PIN0_bm|PIN2_bm); // brosta

    initialize_ADC0();
    free_mode_ADC0();
    start_ADC0();
    sei(); // Enable global interrupts

    while(1){
        check_right();

        if (end == 4) break; // stamataei afou kanei to perigramma

        while (turn_left){
            end++;
            turning_timer();
            while (turn_left){}
        }

        while (turn_right){
            end--;
            turning_timer();
            while (turn_right){}
        }
    }

    cli(); // Disable global interrupts
    return 0;
}

ISR(ADC0_WCOMP_vect){
    cli(); // apenergopoisi interrupts
    // katharismos ADC interrupt flags
    ADC0.INTFLAGS = ADC0.INTFLAGS;

    // apofasi gia deksia I aristeri strofi
    if (ADC0.RES < limit) { //entopizei tixo = etoimazei aristeri strofi
        PORTD.OUT |= (PIN0_bm|PIN1_bm|PIN2_bm); // menei akinito
```

```
        PORTD.OUTCLR |= PIN2_bm; // strivei aristera
        forward = 0;
        turn_left = 1;
        turn_right = 0;
    } else { //den entopizei tixo = strivei deksia
        PORTD.OUT |= (PIN0_bm|PIN1_bm|PIN2_bm); // menei akinito
        PORTD.OUTCLR |= PIN0_bm; // strivei deksia
        forward = 0;
        turn_right = 1;
        turn_left = 0;
    }
    sei(); // Re-enable interrupts
}


void check_right(){
    // apenergopoiw proswrina to free running gia na dw ti kinisi tha kanei
    ADC0.CTRLA &= ~ADC_FREERUN_bm; // apenergopoiw free-running
    ADC0.MUXPOS &= ~ADC_MUXPOS_AIN7_gc; // allazei MUXPOS se deksi sensor
an xreaizetai
    // ADC0.MUXPOS |= ADC_MUXPOS_AINx_gc; // epilegw ton swsto sensora
ADC0.COMMAND = ADC_STCONV_bm; // ksekianaw single conversion gia deksi
sensor
    _delay_ms(1); // kathisterisi wste na oloklirwthei ADC conversion
    free_modeADC0(); // epistrefw se free-running mode gia brosta sensor
}

void turning_timer(){
    // Set the timer for turning
    TCA0.SINGLE.CNT = 0; // katharismos counter
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc; // Normal Mode
    TCA0.SINGLE.CMP0 = ped; // otan ftasei tin timi -> interrupt CLOCK
Frequency/1024
    TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc |
TCA_SINGLE_ENABLE_bm; // Set clock kai enable
       TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm; // Interrupt Enable
}

void initialize_ADC0(){
    ADC0.CTRLA = ADC_RESSEL_10BIT_gc | ADC_ENABLE_bm; // 10-bit resolution,
Enable ADC
    ADC0.MUXPOS = ADC_MUXPOS_AIN7_gc; // epilegw ADC channel (brosta
sensor)
    ADC0.DBGCTRL = ADC_DBGRUN_bm; // Enable debug mode
}

void free_mode_ADC0(){
    ADC0.CTRLA |= ADC_FREERUN_bm; // energopoiw Free-running mode
    ADC0.WINLT = limit; // thetw to xamilo orio/katofli gia ton brosta
sensor
    ADC0.INTCTRL = ADC_WCMP_bm; // energopoiw interrupts for Window
Comparator Mode
    ADC0.CTRLE = ADC_WINCM0_bm; // Interrupt otan RESULT < WINLT (brosta
sensor)
    ADC0.MUXPOS = ADC_MUXPOS_AIN7_gc; // epilegw to kanali gia ton brosta
sensor
}

void start_ADC0(){
```

```
    ADC0.COMMAND = ADC_STCONV_bm; // Start Conversion (brosta sensor)
}

ISR(TCA0_CMP0_vect) {
      TCA0.SINGLE.CTRLA = 0; // apenergopoisi timer
      TCA0.SINGLE.INTFLAGS = TCA0.SINGLE.INTFLAGS; // katharismos timer
interrupt flag

      // apofasizei basi twn flags
      if (turn_left) {
            // stripse aristera
            turn_left = 0; // Reset turn_left flag
            forward = 1; // Set forward gia na paei brosta meta tin strofi
            end++; // auksanw ton end counter otan teleiwnei aristera
strofi
            } else if (turn_right) {
            // stripse deksia
            turn_right = 0; // Reset the turn_right flag
            forward = 1; // Set forward gia na paei brosta meta tin strofi
            end--; // meiwnw ton end counter otan teleiwnei aristera strofi
      }

      // kleinw ola ta LEDs kai meta anoigw to brosta led
      PORTD.OUT |= (PIN0_bm | PIN1_bm | PIN2_bm);
      PORTD.OUTCLR = PIN1_bm; // brosta LED anoixto
}
```

3.c:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

void check_right();
void turning_timer();
void initialize_ADC0();
void free_mode_ADC0();
void start_ADC0();

int turn_left=0;
int forward=1;
int turn_right=0;
int inverse=0;

int end=0;

#define ped 10
#define limit 10  // katofli gia entopismo tixou

int main() {
    PORTD.DIRSET = PIN0_bm | PIN1_bm | PIN2_bm; // 2:left, 1:forward,
0:right
    PORTD.OUT |= (PIN0_bm | PIN2_bm); // brosta

    //configure to port f
    PORTF.DIR &= ~PIN5_bm; // Set PIN5 ws eisodo
    PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc; // Enable
pull-up kai interrupt
    initialize_ADC0();
    free_mode_ADC0();
```

```
    start_ADC0();
    sei(); // Enable global interrupts

    while (1) {
        check_right();

        if (end == 4) break; // stamataei afou kanei to perigramma

        while (turn_left) {
            turning_timer();
            while (turn_left) { }
        }

        while (turn_right) {
            turning_timer();
            while (turn_right) { }
        }

        while (inverse) {
            toggle_direction();
            turning_timer();
            while (inverse) { }
        }
    }

    cli(); // Disable global interrupts
    return 0;
}

ISR(ADC0_WCOMP_vect){
    cli(); // apenergopoisi interrupts
    // katharismos ADC interrupt flags
    ADC0.INTFLAGS = ADC0.INTFLAGS;

    // apofasi gia deksia I aristeri strofi
    if (ADC0.RES < limit) { //entopizei tixo = etoimazei aristeri strofi
        PORTD.OUT |= (PIN0_bm|PIN1_bm|PIN2_bm); // menei akinito
        PORTD.OUTCLR |= PIN2_bm; // strivei aristera
        forward = 0;
        turn_left = 1;
        turn_right = 0;
    } else { //den entopizei tixo = strivei deksia
        PORTD.OUT |= (PIN0_bm|PIN1_bm|PIN2_bm); // menei akinito
        PORTD.OUTCLR |= PIN0_bm; // strivei deksia
        forward = 0;
        turn_right = 1;
        turn_left = 0;
    }
    sei(); // Re-enable interrupts
}


void check_right(){
    // apenergopoiw proswrina to free running gia na dw ti kinisi tha kanei
    ADC0.CTRLA &= ~ADC_FREERUN_bm; // apenergopoiw free-running
    ADC0.MUXPOS &= ~ADC_MUXPOS_AIN7_gc; // allazei MUXPOS se deksi sensor
an xreaizetai
    ADC0.MUXPOS |= ADC_MUXPOS_AINx_gc; // epilegw ton swsto sensora
    ADC0.COMMAND = ADC_STCONV_bm; // ksekianaw single conversion gia deksi
```

```
sensor
    _delay_ms(1); // kathisterisi wste na oloklirwthei ADC conversion
    free_mode_ADC0(); // epistrefw se free-running mode gia brosta sensor}

void turning_timer(){
    // Set the timer for turning
    TCA0.SINGLE.CNT = 0; // katharismos counter
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc; // Normal Mode
    TCA0.SINGLE.CMP0 = ped; // otan ftasei tin timi -> interrupt CLOCK
Frequency/1024
    TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc |
TCA_SINGLE_ENABLE_bm; // Set clock kai enable
       TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm; // Interrupt Enable
}

void initialize_ADC0(){
    ADC0.CTRLA = ADC_RESSEL_10BIT_gc | ADC_ENABLE_bm; // 10-bit resolution,
Enable ADC
    ADC0.MUXPOS = ADC_MUXPOS_AIN7_gc; // epilegw ADC channel (brosta
sensor)
    ADC0.DBGCTRL = ADC_DBGRUN_bm; // Enable debug mode
}

void free_mode_ADC0(){
    ADC0.CTRLA |= ADC_FREERUN_bm; // energopoiw Free-running mode
    ADC0.WINLT = limit; // thetw to xamilo orio/katofli gia ton brosta
sensor
    ADC0.INTCTRL = ADC_WCMP_bm; // energopoiw interrupts for Window
Comparator Mode
    ADC0.CTRLE = ADC_WINCM0_bm; // Interrupt otan RESULT < WINLT (brosta
sensor)
    ADC0.MUXPOS = ADC_MUXPOS_AIN7_gc; // epilegw to kanali gia ton brosta
sensor}

void start_ADC0(){
    ADC0.COMMAND = ADC_STCONV_bm; // Start Conversion (brosta sensor)
}


ISR(TCA0_CMP0_vect) {
      TCA0.SINGLE.CTRLA = 0; // apenergopoisithe timer
      TCA0.SINGLE.INTFLAGS = TCA0.SINGLE.INTFLAGS; // katharismos timer
interrupt flag

      // apofasizei basi twn flags
      if (turn_left) {
            // stripse aristera
            turn_left = 0; // Reset the turn_left flag
            forward = 1; // Set forward gia na paei brosta meta tin strofi
            end++; // auksanw ton end counter otan teleiwnei aristera
strofi
            } else if (turn_right) {
            // stripse deksia
            turn_right = 0; // Reset the turn_right flag
            forward = 1; // Set forward gia na paei brosta meta tin strofi
            end--; //  meiwnw ton end counter otan teleiwnei aristera
strofi      }

      // kleinw ola ta LEDs kai meta anoigw to brosta led
```

```
      PORTD.OUT |= (PIN0_bm | PIN1_bm | PIN2_bm);
      PORTD.OUTCLR = PIN1_bm; // Forward LED on
}

// ISR gia energopoisi reverse mode
ISR(PORTF_PORT_vect) {
    if (PORTF.INTFLAGS & PIN5_bm) { // elegxei an switch interrupt flag
einai set
        inverse = !inverse; // energopoiw/apenergopoiw inverse flag
        end = 0; // Reset end count otan I kateuthinsi allaksei
        PORTF.INTFLAGS = PIN5_bm; // katharismos the interrupt flag
    }
}

// katheuthinsi kinisis (reverse)
void toggle_direction() {

    forward = !forward; // energopoiw/apenergopoiw inverse flag

    inverse = 0; // Reset inverse flag meta to reverse
}
```

ΤΕΛΟΣ ΑΝΑΦΟΡΆΣ