ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ

ΟΜΑΔΑ Β1 - ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 2

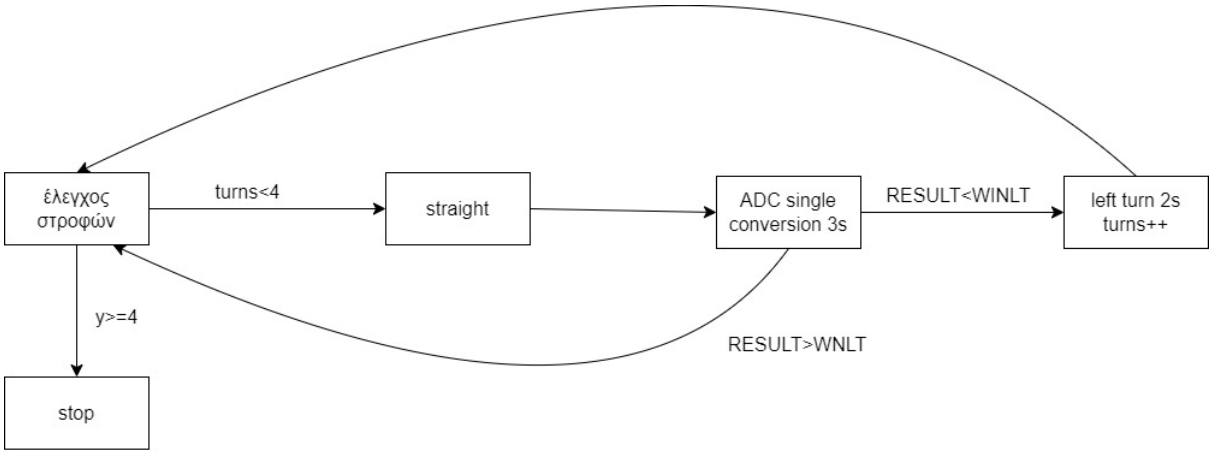
Στοιχεία Φοιτητών:

Αλευράς Ηλίας up1069667@upnet.gr 1069667

Σάββας Στυλιανού up1069661@upnet.gr 1069661

Κανονική λειτουργία

Διάγραμμα Ροής:



Κώδικας

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

#define T1 10
#define T2 10
int zero = 0;
int p;
int z;
int k;
int t;
int y;
int main(void){
	//PIN is output
	PORTD.DIR |= 0b00000001; //PIN0_bm right
	//PIN is output
	PORTD.DIR |= 0b00000010; //PIN1_bm straight
	//PIN is output
	PORTD.DIR |= 0b00000100; //PIN2_bm left
	//LED is off
	PORTD.OUT |= 0b00000001; //PIN0_bm
	//LED is off
	PORTD.OUT |= 0b00000010; //PIN1_bm
	//LED is off
	PORTD.OUT |= 0b00000100; //PIN2_bm

	CLOCK_init();
	//interrupt
	PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;

	//TIMER freerunning mode
	/* Load the Compare or Capture register with the timeout value*/
	TCB0.CCMP |= T1;
	/* Enable Capture or Timeout interrupt */
```

```
        TCB0.INTCTRL |= TCB_CAPT_bm;

        TCB0.CTRLB |= TCB_CNTMODE_INT_gc;

        //TIMER single convertion
        /* Load the Compare or Capture register with the timeout value*/
        TCB1.CCMP = T2;
        /* Configure TCB in Periodic Timeout mode */
        TCB1.CTRLB = TCB_CNTMODE_INT_gc;
        /* Enable Capture or Timeout interrupt */
        TCB1.INTCTRL = TCB_CAPT_bm;

        //TIMER strofis
        /* Load the Compare or Capture register with the timeout value*/
        TCB2.CCMP = T2;
        /* Configure TCB in Periodic Timeout mode */
        TCB2.CTRLB = TCB_CNTMODE_INT_gc;
        /* Enable Capture or Timeout interrupt */
        TCB2.INTCTRL = TCB_CAPT_bm;
        PORTD.DIR |= PIN1_bm; //PIN is output

        //initialize the ADC for Free-Running mode
        ADC0.CTRLA |= ADC_RESSEL_10BIT_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 //Enable Debug Mode
        ADC0.DBGCTRL |= ADC_DBGRUN_bm; //Window Comparator Mode
        ADC0.WINLT |= 10; //Set threshold
        ADC0.INTCTRL |= ADC_WCMP_bm; //Enable Interrupts for WCM
        ADC0.CTRLE |= ADC_WINCM0_bm; //Interrupt when RESULT < WINLT
        sei();
        y=0;
        while(y<4){//to robot tha kanei 4 strofes kai tha stamatisi
                p=0;
                t=1;
                //energopiisi mprostinou esthitira
                straight_on();
                ADC0.CTRLA &= ~ADC_FREERUN_bm; //Free-Running mode disabled
                ADC0.COMMAND |= ADC_STCONV_bm; //Start Conversion
                //energopiisi timer
                TCB1.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ;
                TCB1.CTRLA |= TCB_ENABLE_bm;
                while(p==0){//perimene timer
                        ;
                }
                if(t==0){ // mpeni mono otan energopiithi i isr tou adc
                        TCB2.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina to t2
                        TCB2.CTRLA |= TCB_ENABLE_bm;
                        while (k==0){ //perimene timer strofis t2
                                ;
                        }
                        left_off();
                        straight_on();
                        y++; //metritis strofon
                }


        }

}

void  right_on(){
```

```
	//on
	PORTD.OUTCLR= 0b00000001; //PIN0_bm
}
void right_off(){
	//LED is off
	PORTD.OUT |= 0b00000001; //PIN0_bm
}

void straight_on(){
	//on
	PORTD.OUTCLR= 0b00000010; //PIN1_bm
}
void straight_off(){
	//LED is off
	PORTD.OUT |= 0b00000010; //PIN1_bm
}

void left_on(){
	//on
	PORTD.OUTCLR= 0b00000100; //PIN2_bm
}
void left_off(){
	//LED is off
	PORTD.OUT |= 0b00000100; //PIN2_bm
}

void all_on(){
	//on
	PORTD.OUTCLR= 0b00000001; //PIN0_bm
	//on
	PORTD.OUTCLR= 0b00000010; //PIN1_bm
	//on
	PORTD.OUTCLR= 0b00000100; //PIN2_bm
}

void all_off(){
	//LED is off
	PORTD.OUT |= 0b00000001; //PIN0_bm
	//LED is off
	PORTD.OUT |= 0b00000010; //PIN1_bm
	//LED is off
	PORTD.OUT |= 0b00000100; //PIN2_bm
}

void CLOCK_init (void)
{
	/* Enable writing to protected register */
	CPU_CCP = CCP_IOREG_gc;
	/* Enable Prescaler and set Prescaler Division to 64 */
	CLKCTRL.MCLKCTRLB = CLKCTRL_PDIV_64X_gc | CLKCTRL_PEN_bm;

	/* Enable writing to protected register */
	CPU_CCP = CCP_IOREG_gc;
	/* Select 32KHz Internal Ultra Low Power Oscillator (OSCULP32K) */
	CLKCTRL.MCLKCTRLA = CLKCTRL_CLKSEL_OSCULP32K_gc;

	/* Wait for system oscillator changing to finish */
	while (CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
	{
		;
	}
}
```

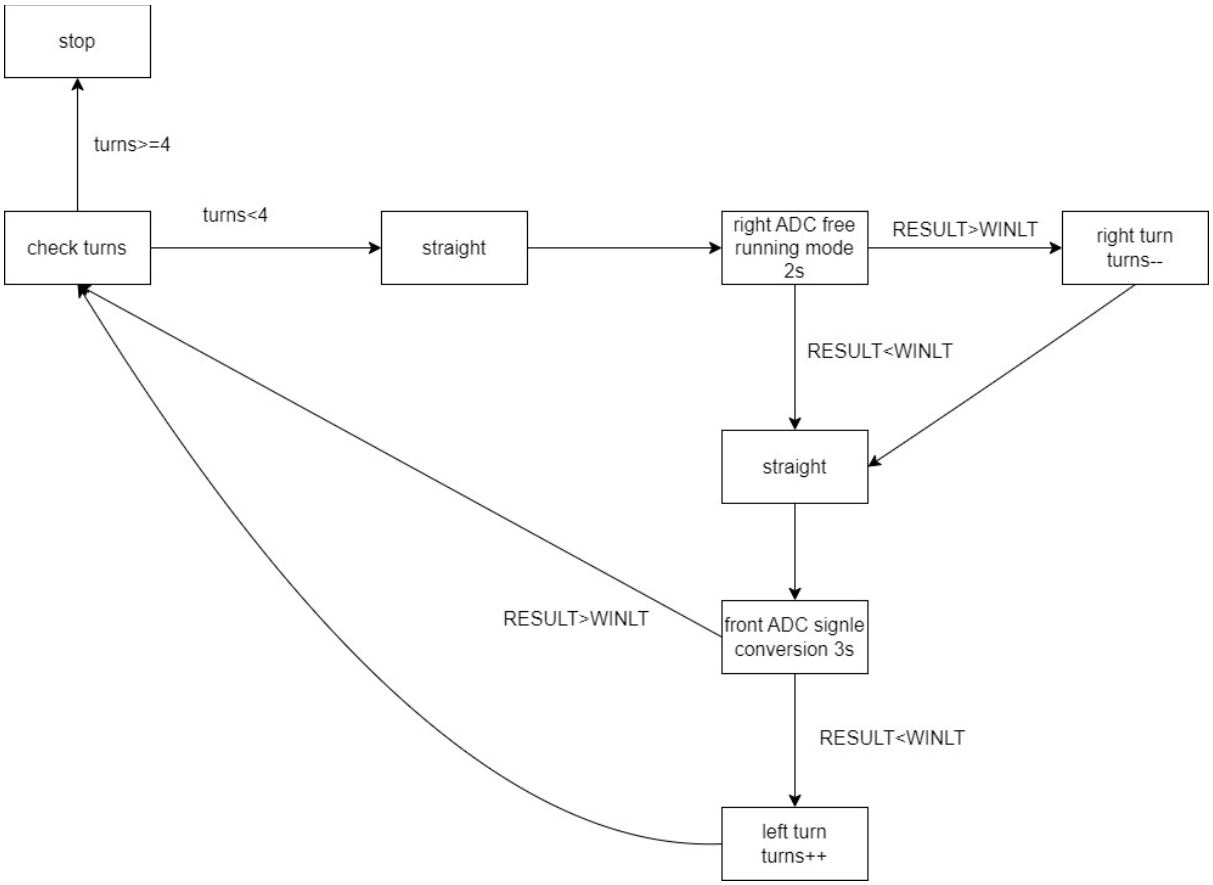
```
ISR(TCB0_INT_vect){
	TCB0.INTFLAGS |= TCB_CAPT_bm; /* Clear the interrupt flag */
	TCB0.CTRLA &= ~TCB_ENABLE_bm;
	z=1;

}
ISR(TCB1_INT_vect){
	TCB1.INTFLAGS |= TCB_CAPT_bm; /* Clear the interrupt flag */
	TCB1.CTRLA &= ~TCB_ENABLE_bm;
	p=1;
}
ISR(TCB2_INT_vect){
	TCB2.INTFLAGS |= TCB_CAPT_bm; /* Clear the interrupt flag */
	TCB2.CTRLA &= ~TCB_ENABLE_bm;
	k=1;
}

ISR(PORTF_PORT_vect){
	//clear the interrupt flag
	int z = PORTF.INTFLAGS;
	PORTF.INTFLAGS=z;


}

ISR(ADC0_WCOMP_vect){
	int intflags = ADC0.INTFLAGS;
	ADC0.INTFLAGS = intflags;
	t=0; // gia na mpi stin if tis strofis
	straight_off();
	left_on();


}
```

Περιγραφή

Στο πρόγραμμα μας χρησιμοποιούμε 2 χρονίστες, τον tcb1 ο οποίος μετρά και μέχρι να δεχτεί τιμή ο μπροστινός αισθητήρας και tcb2 ο οποίος μετρά μέχρι να γίνει η στροφή. Επίσης χρησιμοποιούμε ένα adc ο οποίος είναι συνεχώς σε single conversion γιατί χρησιμοποιούμε μόνο τον μπροστινό αισθητήρα. Αρχικά η λειτουργίες του προγράμματος μας εμπεριέχονται μέσα σε μια while που μετρά μέχρι 4 όσες είναι δηλαδή και οι στροφές οπού θα κάνει. Ξεκινούμε με το LED της ευθείας αναμμένο και ξεκινούμε τον adc. Στην συνέχεια ξεκινούμε και τον timer οπού περιμένει τον adc να πάρει τιμή. Εάν ο adc πάρει τιμή RESULT < WINLT τότε εισέρχεται μέσα στην if οπού γίνεται η στροφή. Στην if ενεργοποιείται το LED τις αριστερής στροφής και ο timer του. Αφού τελειώσει ο timer το LED γίνεται off και προσθέτουμε 1 στην μεταβλητή στροφών y.

Δυο αμβλείες γωνίες

Διάγραμμα Ροής:



Κώδικας

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

#define T1 10
#define T2 10
int zero = 0;
int p;//gia na stamatisi i while
int z;//gia na stamatisi i while
int y=0;//metritis strofon
int x;//dixni pou na gini strofi aristera i deksia
int t;
int k;
int main(void){
	//PIN is output
	PORTD.DIR |= 0b00000001; //PIN0_bm right
	//PIN is output
	PORTD.DIR |= 0b00000010; //PIN1_bm straight
	//PIN is output
```

```
PORTD.DIR |= 0b00000100; //PIN2_bm left
//LED is off
PORTD.OUT |= 0b00000001; //PIN0_bm
//LED is off
PORTD.OUT |= 0b00000010; //PIN1_bm
//LED is off
PORTD.OUT |= 0b00000100; //PIN2_bm

CLOCK_init();
//interrupt
PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;

//TIMER freerunning mode
/* Load the Compare or Capture register with the timeout value*/
TCB0.CCMP |= T1;
/* Enable Capture or Timeout interrupt */
TCB0.INTCTRL |= TCB_CAPT_bm;

TCB0.CTRLB |= TCB_CNTMODE_INT_gc;

//TIMER single convertion
/* Load the Compare or Capture register with the timeout value*/
TCB1.CCMP = T2;
/* Configure TCB in Periodic Timeout mode */
TCB1.CTRLB = TCB_CNTMODE_INT_gc;
/* Enable Capture or Timeout interrupt */
TCB1.INTCTRL = TCB_CAPT_bm;
//TIMER strofis
/* Load the Compare or Capture register with the timeout value*/
TCB2.CCMP = T2;
/* Configure TCB in Periodic Timeout mode */
TCB2.CTRLB = TCB_CNTMODE_INT_gc;
/* Enable Capture or Timeout interrupt */
TCB2.INTCTRL = TCB_CAPT_bm;

PORTD.DIR |= PIN1_bm; //PIN is output

//initialize the ADC for Free-Running mode
ADC0.CTRLA |= ADC_RESSEL_10BIT_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 //Enable Debug Mode
ADC0.DBGCTRL |= ADC_DBGRUN_bm; //Window Comparator Mode
ADC0.WINLT |= 10; //Set threshold
ADC0.INTCTRL |= ADC_WCMP_bm; //Enable Interrupts for WCM

sei();
y=0;
while(y<4){ //metritis strofon
	p=t=z=k=0;//arxikopiisi timon
	//deksia strofi
	//deksia esthitiras
	straight_on();
	ADC0.CTRLE = 0x02; //Interrupt when RESULT > WINLT
	ADC0.CTRLA |= ADC_FREERUN_bm; //Free-Running mode enabled
	ADC0.COMMAND |= ADC_STCONV_bm; //Start Conversion

	TCB0.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //timer dekia esthitira
	TCB0.CTRLA |= TCB_ENABLE_bm;
	while(z==0){ //perimer timer
		;
	}
```

```
		if(t==1){//mpeni mes stin if mono ean exei gini interrupt apo ton adc

			ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion
			right_on();
			TCB2.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina timer strofis
			TCB2.CTRLA |= TCB_ENABLE_bm;
			while(k==0){//perimene timer strofis
				;
			}
			right_off();
			y--; //metritis strofis

		}
		straight_on();
		ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion
		ADC0.CTRLA |= ADC_ENABLE_bm; //start conversion
		t=0;
		k=0;
		//aristeri strofi
		//mprosta esthitiras
		ADC0.CTRLA &= ~ADC_FREERUN_bm; //Free-Running mode disabled
		ADC0.CTRLE = 0x01; //Interrupt when RESULT < WINLT
		ADC0.COMMAND |= ADC_STCONV_bm; //Start Conversion

		TCB1.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina timer mprosta esthitira
		TCB1.CTRLA |= TCB_ENABLE_bm;
		while(p==0){ //perimene timer
			;
		}
		if(t==1){ //mpeni mesa otan energopiithi to interrupt tis adc

			left_on();
			TCB2.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina timer strofis
			TCB2.CTRLA |= TCB_ENABLE_bm;
			while(k==0){ //perimene timer
				;
			}
			left_off();
			y++;

		}


	}

}

void  right_on(){
	//on
	PORTD.OUTCLR= 0b00000001; //PIN0_bm
}
void right_off(){
	//LED is off
	PORTD.OUT |= 0b00000001; //PIN0_bm
}

void straight_on(){
	//on
	PORTD.OUTCLR= 0b00000010; //PIN1_bm
}
void straight_off(){
	//LED is off
```

```
	PORTD.OUT |= 0b00000010; //PIN1_bm
}

void left_on(){
	//on
	PORTD.OUTCLR= 0b00000100; //PIN2_bm
}
void left_off(){
	//LED is off
	PORTD.OUT |= 0b00000100; //PIN2_bm
}

void all_on(){
	//on
	PORTD.OUTCLR= 0b00000001; //PIN0_bm
	//on
	PORTD.OUTCLR= 0b00000010; //PIN1_bm
	//on
	PORTD.OUTCLR= 0b00000100; //PIN2_bm
}

void all_off(){
	//LED is off
	PORTD.OUT |= 0b00000001; //PIN0_bm
	//LED is off
	PORTD.OUT |= 0b00000010; //PIN1_bm
	//LED is off
	PORTD.OUT |= 0b00000100; //PIN2_bm
}

void CLOCK_init (void)
{
	/* Enable writing to protected register */
	CPU_CCP = CCP_IOREG_gc;
	/* Enable Prescaler and set Prescaler Division to 64 */
	CLKCTRL.MCLKCTRLB = CLKCTRL_PDIV_64X_gc | CLKCTRL_PEN_bm;

	/* Enable writing to protected register */
	CPU_CCP = CCP_IOREG_gc;
	/* Select 32KHz Internal Ultra Low Power Oscillator (OSCULP32K) */
	CLKCTRL.MCLKCTRLA = CLKCTRL_CLKSEL_OSCULP32K_gc;

	/* Wait for system oscillator changing to finish */
	while (CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
	{
		;
	}
}


ISR(TCB0_INT_vect){
	TCB0.INTFLAGS |= TCB_CAPT_bm; /* Clear the interrupt flag */
	TCB0.CTRLA &= ~TCB_ENABLE_bm;
	z=1;

}
ISR(TCB1_INT_vect){
	TCB1.INTFLAGS |= TCB_CAPT_bm; /* Clear the interrupt flag */
	TCB1.CTRLA &= ~TCB_ENABLE_bm;
	p=1;
}
ISR(TCB2_INT_vect){
```

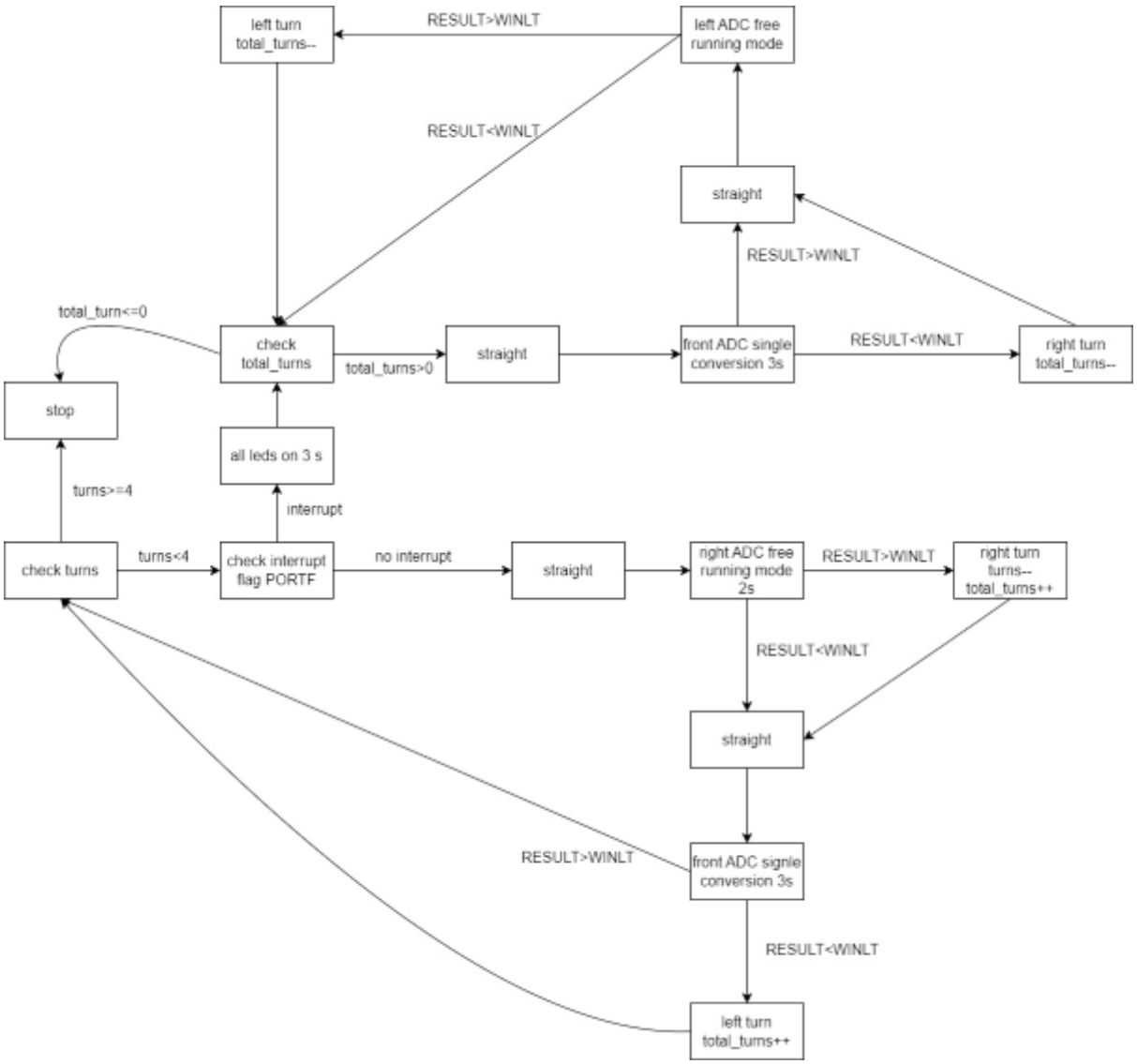
```
        TCB2.INTFLAGS |= TCB_CAPT_bm; /* Clear the interrupt flag */
        TCB2.CTRLA &= ~TCB_ENABLE_bm;
        k=1;
}


ISR(ADC0_WCOMP_vect){
        int intflags = ADC0.INTFLAGS;
        ADC0.INTFLAGS = intflags;
        straight_off();
        t=1; //gia na mpi stin if tis strofis
        ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion
}
```

Περιγραφή

Στο πρόγραμμα μας χρησιμοποιούμε 3 χρονίστες, τον tcb0 ο οποίος μετρά μέχρι να δεχτεί τιμή ο δεξιά αισθητήρας, τον tcb1 ο οποίος μετρά μέχρι να δεχτεί τιμή ο μπροστινός αισθητήρας και tcb2 ο οποίος μετρά μέχρι να γίνει η στροφή. Επίσης χρησιμοποιούμε ένα adc ο οποίος δουλεύει ενάλλαξα single conversion και free running mode ανάλογα με τον πιο αισθητήρα θέλουμε να ελέγξουμε. Δηλώνουμε όλους τους χρονιστες, τον adc και όλα τα flags. Το πρόγραμμα που εκτελείται βρίσκεται σε μια while η οποία μετρά μέχρι το 4. Αρχικά αρχικοποιουμε τα flags p t z k με 0, ανάβουμε το LED για ευθεία κίνηση και ξεκινούμε τον δεξιό αισθητήρα adc σε free running mode με interrupt να γίνεται όταν RESULT > WINLT. Ακόλουθος ξεκινούμε τον tcb0 και περιμένουμε στην while μέχρι να τελειώσει. Αφού τελειώσει η while του tcb0 και έχει συμβεί interrupt από τον adc το πρόγραμμα θα εισέλθει στην if(t==1){ οπου είναι ο κώδικας για να στρίψει δεξιά και θα αφαιρέσει 1 από την μεταβλητή y, εάν δεν συμβεί interrupt θα συνέχιση ευθεία. Στην συνέχεια απενεργοποιούμε τον adc για να σταματήσει το conversion και τον ξανά ενεργοποιούμε, αρχικοποιουμε τα flags και αλλάζουμε τον adc να κάνει interrupt όταν RESULT < WINLT και να τρέχει σε single conversion mode. Ακόλουθος ξεκινούμε τον tcb1 και τον περιμένουμε να τελειώσει. Παράλληλα περιμένουμε να γίνει interrupt από τον adc. Όταν τελειώσει ο tcb1 εάν έχει γίνει interrupt από τον adc τότε θα εισέλθουμε στο if οπού γίνεται η αριστερή στροφή και θα προσθέσουμε αλλιώς συνεχίζει ευθεία.

Ανάποδη λειτουργία

Διάγραμμα Ροής:



Κώδικας

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

#define T1 10
#define T2 10
int zero = 0;
int p;//gia na stamatisi i while
int z;//gia na stamatisi i while
int y=0;//metritis strofon
int x;//dixni pou na gini strofi aristera i deksia
int t;
int k;
int strofes=0;
int main(void){
	//PIN is output
	PORTD.DIR |= 0b00000001; //PIN0_bm right
	//PIN is output
	PORTD.DIR |= 0b00000010; //PIN1_bm straight
	//PIN is output
	PORTD.DIR |= 0b00000100; //PIN2_bm left
	//LED is off
	PORTD.OUT |= 0b00000001; //PIN0_bm
	//LED is off
	PORTD.OUT |= 0b00000010; //PIN1_bm
	//LED is off
	PORTD.OUT |= 0b00000100; //PIN2_bm

	CLOCK_init();
	//interrupt
	PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;

	//TIMER freerunning mode
	/* Load the Compare or Capture register with the timeout value*/
	TCB0.CCMP |= T1;
	/* Enable Capture or Timeout interrupt */
	TCB0.INTCTRL |= TCB_CAPT_bm;

	TCB0.CTRLB |= TCB_CNTMODE_INT_gc;

	//TIMER single convertion
	/* Load the Compare or Capture register with the timeout value*/
	TCB1.CCMP = T2;
	/* Configure TCB in Periodic Timeout mode */
	TCB1.CTRLB = TCB_CNTMODE_INT_gc;
	/* Enable Capture or Timeout interrupt */
	TCB1.INTCTRL = TCB_CAPT_bm;
	//TIMER strofis
	/* Load the Compare or Capture register with the timeout value*/
	TCB2.CCMP = T2;
	/* Configure TCB in Periodic Timeout mode */
	TCB2.CTRLB = TCB_CNTMODE_INT_gc;
	/* Enable Capture or Timeout interrupt */
	TCB2.INTCTRL = TCB_CAPT_bm;

	PORTD.DIR |= PIN1_bm; //PIN is output

	//initialize the ADC for Free-Running mode
	ADC0.CTRLA |= ADC_RESSEL_10BIT_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 //Enable Debug Mode
	ADC0.DBGCTRL |= ADC_DBGRUN_bm; //Window Comparator Mode
	ADC0.WINLT |= 10; //Set threshold
	ADC0.INTCTRL |= ADC_WCMP_bm; //Enable Interrupts for WCM

	sei();
	y=0;

	while(y<4){ // to programma tha trexi mexri na gini to y 4 dld ftasi eki pou
ksekinise

		if(x==0){ // KANONIKI LITOURGIA to x einai 0 otan den exei energopiithi
to interrupt sto portf
			p=t=z=k=0; //arxikopiisi timon
			//deksia strofi
			straight_on();
			//deksios esthitiras
			ADC0.CTRLA |= ADC_ENABLE_bm; //start conversion
			ADC0.CTRLE = 0x02; //Interrupt when RESULT > WINLT
			ADC0.CTRLA |= ADC_FREERUN_bm; //Free-Running mode enabled
			ADC0.COMMAND |= ADC_STCONV_bm; //Start Conversion

			TCB0.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //timer aristerou esthitia
			TCB0.CTRLA |= TCB_ENABLE_bm;
			while(z==0){ //perimene timer
				;
			}
			if(t==1){ //ean exi gini interupt apo to adc to t tha gini 1 kai
tha stripsi deksia

				ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion
				right_on();
				TCB2.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //timer strofis
				TCB2.CTRLA |= TCB_ENABLE_bm;
				while(k==0){ //perimene timer strofis
					;
				}
				right_off();
				y--; //kathe fora pou strifi deksia aferoume 1 apo to y
				strofes++;

			}
			straight_on();
			ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion
			ADC0.CTRLA |= ADC_ENABLE_bm; //start conversion
			p=t=z=k=0; //arxikopiisi timon
			//aristeri strofi
			//mprosta esthitiras
			ADC0.CTRLA &= ~ADC_FREERUN_bm; //Free-Running mode disabled
			ADC0.CTRLE = 0x01; //Interrupt when RESULT < WINLT
			ADC0.COMMAND |= ADC_STCONV_bm; //Start Conversion

			TCB1.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina to timer tou
mprosta esthitira
			TCB1.CTRLA |= TCB_ENABLE_bm;
			while(p==0){ //perimene timer mprosta esthitira
				;
			}
			if(t==1){ //ean exi gini interrupt apo ton adc tha gini to t 1
kai tha stripsi aristera
```

```
                        left_on();
                        TCB2.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina timer
strofis
                        TCB2.CTRLA |= TCB_ENABLE_bm;
                        while(k==0){ //perimene timer strofis
                                ;
                        }
                        left_off();
                        y++;
                        strofes++;
                }


        }else if(x==1){ //ANAPODI LITOURGIA mpeni sto if mono ean exi
energopiithi to portf
        TCB2.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina timer gia ola ta fota
anixta
        TCB2.CTRLA |= TCB_ENABLE_bm;
        while(k==0){ //perimene timer
                ;
        }
        all_off();
        while(strofes>0){ //otan oi strofes einai mikroteres i ises tou 0 exei
ftasi stin arxiki tou thesi
                p=t=z=k=0; //arxikopiisi timon
                ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion


                //deksia strofi
                //mprosta esthitiras
                ADC0.CTRLA |= ADC_ENABLE_bm; //start conversion
                ADC0.CTRLA &= ~ADC_FREERUN_bm; //Free-Running mode disabled
                ADC0.CTRLE = 0x01; //Interrupt when RESULT < WINLT
                ADC0.COMMAND |= ADC_STCONV_bm; //Start Conversion

                TCB1.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina timer
                TCB1.CTRLA |= TCB_ENABLE_bm;
                while(p==0){//perimene timer
                        ;
                }
                if(t==1){ // mpeni ean exei energopiithi interrupt apo ton adc

                        right_on();
                        TCB2.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina timer
                        TCB2.CTRLA |= TCB_ENABLE_bm;
                        while(k==0){
                                ;
                        }
                        right_off();

                        strofes--;
                }
                ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion
                p=t=z=k=0; //arxikopiisi timon
                //aristeri strofi
                straight_on();
                ADC0.CTRLA |= ADC_ENABLE_bm; //start conversion
                ADC0.CTRLE = 0x02; //Interrupt when RESULT > WINLT
                ADC0.CTRLA |= ADC_FREERUN_bm; //Free-Running mode enabled
                ADC0.COMMAND |= ADC_STCONV_bm; //Start Conversion
```

```
                        TCB0.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ;  //ksekina timer
                        TCB0.CTRLA |= TCB_ENABLE_bm;
                        while(z==0){ //perimene timer
                                ;
                        }
                        if(t==1){ //mpeni mono ean exei energopiithi to timer apo ton adc

                                ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion
                                left_on();
                                TCB2.CTRLA |= TCB_CLKSEL_CLKDIV1_gc ; //ksekina timer
                                TCB2.CTRLA |= TCB_ENABLE_bm;
                                while(k==0){
                                        ;
                                }
                                left_off();

                                strofes--;



                        }
                        straight_on();

                }

        }
        }
}
void  right_on(){
        //on
        PORTD.OUTCLR= 0b00000001; //PIN0_bm
}
void right_off(){
        //LED is off
        PORTD.OUT |= 0b00000001; //PIN0_bm
}

void straight_on(){
        //on
        PORTD.OUTCLR= 0b00000010; //PIN1_bm
}
void straight_off(){
        //LED is off
        PORTD.OUT |= 0b00000010; //PIN1_bm
}

void left_on(){
        //on
        PORTD.OUTCLR= 0b00000100; //PIN2_bm
}
void left_off(){
        //LED is off
        PORTD.OUT |= 0b00000100; //PIN2_bm
}

void all_on(){
        //on
        PORTD.OUTCLR= 0b00000001; //PIN0_bm
        //on
        PORTD.OUTCLR= 0b00000010; //PIN1_bm
        //on
```

```
	PORTD.OUTCLR= 0b00000100; //PIN2_bm
}

void all_off(){
	//LED is off
	PORTD.OUT |= 0b00000001; //PIN0_bm
	//LED is off
	PORTD.OUT |= 0b00000010; //PIN1_bm
	//LED is off
	PORTD.OUT |= 0b00000100; //PIN2_bm
}

void CLOCK_init (void)
{
	/* Enable writing to protected register */
	CPU_CCP = CCP_IOREG_gc;
	/* Enable Prescaler and set Prescaler Division to 64 */
	CLKCTRL.MCLKCTRLB = CLKCTRL_PDIV_64X_gc | CLKCTRL_PEN_bm;

	/* Enable writing to protected register */
	CPU_CCP = CCP_IOREG_gc;
	/* Select 32KHz Internal Ultra Low Power Oscillator (OSCULP32K) */
	CLKCTRL.MCLKCTRLA = CLKCTRL_CLKSEL_OSCULP32K_gc;

	/* Wait for system oscillator changing to finish */
	while (CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
	{
		;
	}
}


ISR(TCB0_INT_vect){
	TCB0.INTFLAGS |= TCB_CAPT_bm; /* Clear the interrupt flag */
	TCB0.CTRLA &= ~TCB_ENABLE_bm;
	z=1;

}
ISR(TCB1_INT_vect){
	TCB1.INTFLAGS |= TCB_CAPT_bm; /* Clear the interrupt flag */
	TCB1.CTRLA &= ~TCB_ENABLE_bm;
	p=1;
}
ISR(TCB2_INT_vect){
	TCB2.INTFLAGS |= TCB_CAPT_bm; /* Clear the interrupt flag */
	TCB2.CTRLA &= ~TCB_ENABLE_bm;
	k=1;
}
ISR(PORTF_PORT_vect){
	//clear the interrupt flag
	int z = PORTF.INTFLAGS;
	PORTF.INTFLAGS=z;
	all_on();
	x=1; //gia na mpi sto if tis anapodis litourgias
	k=0;
	ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion
}

ISR(ADC0_WCOMP_vect){
	int intflags = ADC0.INTFLAGS;
	ADC0.INTFLAGS = intflags;
	straight_off();
```

```
        t=1; //gia na mpi sto if tis strofis
        ADC0.CTRLA &= ~ADC_ENABLE_bm; //stop conversion
}
```

Περιγραφή

Η ανάποδη λειτουργία ενεργοποιείται από interrupt στο PORTF. Η κανονική λειτουργία βρίσκεται στο if(x==0){ και η ανάποδη στο }else if(x==1){. Όταν γίνει 1 η μεταβλητή x από το interrupt στο PORTF τότε ενεργοποιείται. Επίσης κρατάμε μια μεταβλητή με όνομα strofes οπού στην κανονική λειτουργία κάθε φορά που κάνουμε μια στροφή προσθέτουμε 1 σε αυτή ώστε στην ανάποδη να γνωρίζουμε πόσες στροφές θα κάνουμε.