**Προηγμένοι Μικροεπεξεργαστές**

**Εργαστηριακή Άσκηση 2**

**Μέλη Ομάδας**

**1ο μέλος: 2ο μέλος:**

**ΟΝΟΜΑ: ΛΟΥΚΑΣ ΟΝΟΜΑ: ΕΥΑΓΓΕΛΟΣ**

**ΕΠΩΝΥΜΟ: ΤΡΙΑΝΤΑΦΥΛΛΟΠΟΥΛΟΣ ΕΠΩΝΥΜΟ: ΜΠΕΝΕΤΟΣ**

**ΕΞΑΜΗΝΟ: 8 ΕΞΑΜΗΝΟ: 10**

**ΑΜ: 1084495 ΑΜ: 1072628**

1) Για το πρώτο ερώτημα της άσκησης γράφουμε ένα κώδικα για μία συσκευή που θα ακολουθεί μια διαδρομή που σχηματίζει ένα τετράγωνο μέσα σε έναν χώρο. Ο κώδικας ξεκινά με την χρήση των ζητούμενων LEDs ως εξόδων για τον συμβολισμό της ευθείας κίνησης (LED1) και της αριστερής στροφής (LED2). Στην αρχή δηλώνουμε τα δύο LED ότι είναι σβηστά και στην συνέχεια ανάβουμε μόνο το LED1 για να δείξουμε ότι εκτελείται η κίνηση προς τα μπρος. Έπειτα ορίζουμε μια ακέραια μεταβλητή (turn\_count) και την αρχικοποιούμε με 0. Σκοπός αυτής, είναι να επαναλάβει την διαδικασία 4 φορές, δηλαδή να κινηθεί μπροστά και να στρίψει προς τα αριστερά.

Επειδή η μόνη κίνηση που θα γίνεται, είναι αυτή προς τα εμπρός κι επειδή θα χρησιμοποιηθεί μόνο ένας αισθητήρας, θα γίνει χρήση μόνο της λειτουργίας Free running mod. Συγχρόνως ορίζουμε ως threshold την τιμή 20 για να θέσουμε την μέγιστη απόσταση που θα διανύει η συσκευή πριν έρθει αντιμέτωπη με τοίχο αλλά και για να γίνεται η σύγκριση μεταξύ του καταχωρητή RES και του WINLT για να δούμε αν θα εισέρχεται, ή όχι, στην αντίστοιχη συνάρτηση ISR. Για να σταματήσει η οποιαδήποτε διαδικασία από πλευράς αισθητήρα, θέτουμε μια μικρή δομή επανάληψης while με όρισμα μια μεταβλητή aux\_flag.

Για την ISR του αισθητήρα, πέραν από τις εντολές που προϋπήρχαν από το παράδειγμα που διδάχθηκε στην 2η διάλεξη, προσθέσαμε τις συναρτήσεις sei(); και cli(); για να γίνεται reset ο μετατροπέας. Στο τέλος της int\_main(), θα προσθέσουμε την μεταβλητή turn\_count και θα την αυξάνουμε κατά ένα, κάθε φορά που κάνει το κύκλο όλης της διαδικασίας. Όσο η turn\_count είναι μικρότερη του 4, η διαδικασία θα συνεχίσει να επαναλαμβάνεται μέχρις ότου να γίνει το τετράγωνο. Με την εκτέλεση του κώδικα, όταν εισάγουμε χειροκίνητα την τιμή του καταχωρητή RES, μέσω επιλογής ενός bit από τον πίνακα, γίνεται η σύγκριση για το αν η τιμή που διαλέξαμε είναι μικρότερη ή μεγαλύτερη του threshold που έχουμε ορίσει. Αν είναι μικρότερη, τότε με το run κουμπί εισέρχεται στην ISR. Όταν μπει στην συνάρτηση αλλάζουμε την τιμή του RES με μια τιμή μεγαλύτερη του threshold, έτσι ώστε όταν εξέλθει της ISR, να μπορέσει να συνεχίσει στις εντολές της main που βρίσκονται μετά την while(aux\_flag == false) και να αυξηθεί κατά ένα η turn\_count.

Όταν η διαδικασία επαναληφθεί 4 φορές (turn\_count < 4), η διαδικασία τερματίζεται. Επομένως με βάση όλες τις λεπτομέρειες που αναφέραμε παραπάνω, στην συνέχεια παραθέτουμε τον ακόλουθο κώδικα:

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdbool.h>

int turn\_count = 0; //turn counter to make a route of a square

bool aux\_flag = false; // auxiliary flag for the loop

int main()

{

PORTD.DIR|= PIN1\_bm; //Set pin1 as output to move forward

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

PORTD.DIR|= PIN2\_bm;// Set pin2 as output to move left

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

while(turn\_count<4)

{

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

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

//initialize the ADC for Free-Running mode

ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc; //10-bit resolution

ADC0.CTRLA |= ADC\_FREERUN\_bm; //ADC Freerun mode bit mask

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

ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc; //Analog Channel Selection Bits select

ADC0.DBGCTRL |= ADC\_DBGRUN\_bm; //Debug run bit mask

//Window Comparator Mode

ADC0.WINLT |= 20; //Set threshold

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

while(aux\_flag == false)

{

;//similar to a nop function

}

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

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

cli();

turn\_count++; // Increase the counter until it makes a square route

aux\_flag = false;

}

}

ISR(ADC0\_WCOMP\_vect)

{

//cli();

int intflags = ADC0.INTFLAGS;

ADC0.INTFLAGS = intflags;

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

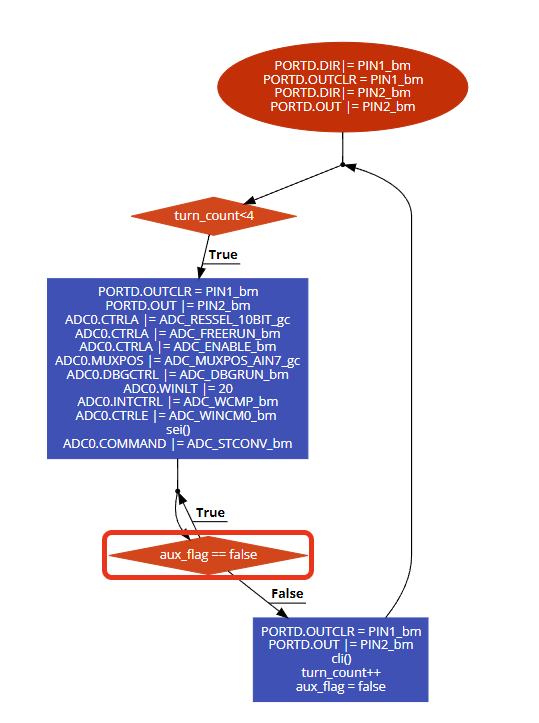
PORTD.OUT |= PIN1\_bm; //LED for forward movement is off

aux\_flag = true;

//sei();

}

Και εδώ παραθέτουμε το διάγραμμα ροής για το ερώτημα 1:



2) Στο δεύτερο ερώτημα ζητείται να εισάγουμε την δεύτερη λειτουργία του μετατροπέα και να κάνει την περίμετρο ενός τυχαίου δωματίου με 2 αμβλείες γωνίες. Αρχικά εισάγουμε κάποιες νέες μεταβλητές που θα χρειαστούν στην συνέχεια του κώδικα. Πρώτα ορίζουμε τις Τ1 και Τ2 που θα αξιοποιηθούν για τους δύο χρονιστές. Στην συνέχεια ορίζουμε δύο νέους χρονιστές (turn\_left\_count και turn\_right\_count) για να μετράμε πόσες αριστερές και πόσες δεξιές στροφές έχουμε διανύσει. Για κάθε μία στροφή, αυξάνεται και κατά ένα και ο ανάλογος χρονιστής. Επιπλέον ορίζουμε μια μεταβλητή turn\_difference που θα ισούται με την διαφορά των δύο μετρητών (turn\_left\_count και turn\_right\_count).

Για την main ορίζουμε μια συνθήκη όπου η διαφορά των δύο μετρητών των δύο κατευθύνσεων θα είναι μικρότερη του 4, για την οποία θα εκτελείται η δομή επανάληψης για όσο ισχύει. Πριν την δομή επανάληψης while θέτουμε αρχικά ως κλειστά LEDs τα PIN0 και PIN2 (δεξιά και αριστερά) και ανοικτό το PIN1 (μπροστά). Εντός της δομής βάζουμε την μετατροπέα σε λειτουργία single conversion όπως ζητείται και τον χρησιμοποιούμε ως αισθητήρα για εντοπισμό τοίχου προς τα δεξιά. Αμέσως μετά βάζουμε και τον χρονιστή με ένα αντιπροσωπευτικό χρόνο κοντά σε αυτόν που θέτει η άσκηση. Έπειτα ορισμός της εμφωλευμένης δομής while έτσι ώστε όταν γίνει step into και run να εισέλθει σε ADC και TCA ISR αντίστοιχα. Μετά θα προσθέσουμε και μια συνθήκη if, η οποία θα συγκρίνει την χειροκίνητη και τρέχουσα τιμή του καταχωρητή RES με τον WINHT. Αν η τιμή του RES είναι μικρότερη, τότε θα ανάβει το επιθυμητό λαμπάκι για την δεξιά στροφή. Αφού εξέλθει της if, τα LEDs ξαναορίζονται έτσι ώστε να έχουμε την εμπρός κίνηση. Η ίδια διαδικασία επαναλαμβάνεται αμέσως μετά και για την στροφή προς τα αριστερά. Για τα αριστερά θέτουμε το κατώφλι WINLT και λειτουργία μετατροπέα free running. Μετά ξαναορίζουμε τα λαμπάκια να αντιστοιχούν σε εμπρός κίνηση. Τέλος, βάζουμε κατάλληλη εντολή που θα αλλάζει τη λειτουργία του μετατροπέα πίσω σε single conversion έτσι ώστε να τον αξιοποίησουμε ξανά όταν θα επαναληφθεί από την αρχή η while.

Λαμβάνοντας υπόψιν όσα όλα ειπώθηκαν παραπάνω, στην συνέχεια δίνουμε τον κώδικα που αναλογεί σε αυτήν την περιγραφή:

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdbool.h>

#define T1 15 //Time for Single Conversion

#define T2 15 //Time for Free Running

int turn\_difference = 0; //turn counter to make a route of a random room

int turn\_left\_count = 0; //integer variable to count the left turns

int turn\_right\_count = 0; //integer variable to count the right turns

bool aux\_flag = false; // auxiliary flag for the loop

bool aux\_flag2 = false; // auxiliary flag for the loop

int main()

{

PORTD.DIR |= PIN0\_bm;//Set pin0 as output to move right

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

PORTD.DIR |= PIN1\_bm; //Set pin1 as output to move forward

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

PORTD.DIR |= PIN2\_bm; //Set pin2 as output to move left

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

while(turn\_difference < 4)

{

//We use single conversion

ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc; //10-bit resolution

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

ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc; //Analog Channel Selection Bits select

ADC0.DBGCTRL |= ADC\_DBGRUN\_bm; //Debug run bit mask

//Window Comparator Mode

ADC0.WINHT |= 8; //Set threshold for right movement

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

ADC0.CTRLE = ADC\_WINCM1\_bm; //Interrupt when RESULT > WINHT

sei();

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

TCA0.SINGLE.CNT = 0; //Clear Counter

TCA0.SINGLE.CTRLB = TCA\_SINGLE\_WGMODE\_NORMAL\_gc; //Waveform generation mode selection ==> Normal Mode

TCA0.SINGLE.CMP0 = T1; //When CMP0 reaches this value, then interrupt will occur ==> CLOCK\_FREQUENCY/1024

TCA0.SINGLE.CTRLA = TCA\_SINGLE\_CLKSEL\_DIV1024\_gc; //Clock Selection mode ==> System Clock/1024

TCA0.SINGLE.CTRLA |= 1; //Enable

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Compare 0 Interrupt bit mask

while(aux\_flag == false)

{

sei();//similar to a nop function

}

if(ADC0.RES>8)

{

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

PORTD.OUT |= PIN1\_bm; //LED for forward movement is off

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

turn\_right\_count++;

}

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

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

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

aux\_flag = false;

aux\_flag2 = false;

//initialize the ADC for Free-Running mode

ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc; //10-bit resolution

ADC0.CTRLA |= ADC\_FREERUN\_bm; //ADC Freerun mode bit mask

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

ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc; //Analog Channel Selection Bits select

ADC0.DBGCTRL |= ADC\_DBGRUN\_bm; //Debug run bit mask

//Window Comparator Mode

ADC0.WINLT |= 8; //Set threshold for left

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

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

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

TCA0.SINGLE.CNT = 0; //Clear Counter

TCA0.SINGLE.CTRLB = TCA\_SINGLE\_WGMODE\_NORMAL\_gc; //Waveform generation mode selection ==> Normal Mode

TCA0.SINGLE.CMP0 = T2; //When CMP0 reaches this value, then interrupt will occur ==> CLOCK\_FREQUENCY/1024

TCA0.SINGLE.CTRLA = TCA\_SINGLE\_CLKSEL\_DIV1024\_gc; //Clock Selection mode ==> System Clock/1024

TCA0.SINGLE.CTRLA |= 1; //Enable

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Compare 0 Interrupt bit mask

while(aux\_flag2 == false)

{

sei();

}

if(ADC0.RES<8)

{

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

PORTD.OUT |= PIN1\_bm; //LED for forward movement is off

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

turn\_left\_count++;

}

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

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

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

cli();

turn\_difference = turn\_left\_count - turn\_right\_count; //If turn\_difference reaches

//a certain value the program will end

ADC0.CTRLA = ADC0.CTRLA & 0b11111101; //We set again ADC to single conversion to start

//the loop all over again if needed

aux\_flag = false;

aux\_flag2 = false;

}

}

ISR(ADC0\_WCOMP\_vect)

{

cli();

int intflags = ADC0.INTFLAGS;

ADC0.INTFLAGS = intflags;

ADC0.CTRLA = 0;

sei();

}

ISR(TCA0\_CMP0\_vect)

{

TCA0.SINGLE.CTRLA = 0; //Disable

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

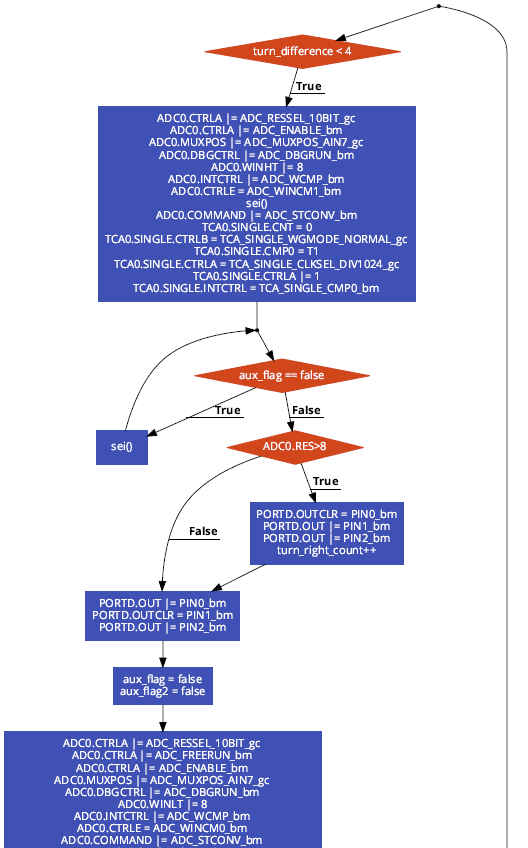
TCA0.SINGLE.INTFLAGS=intflags; //clear interrupt flag

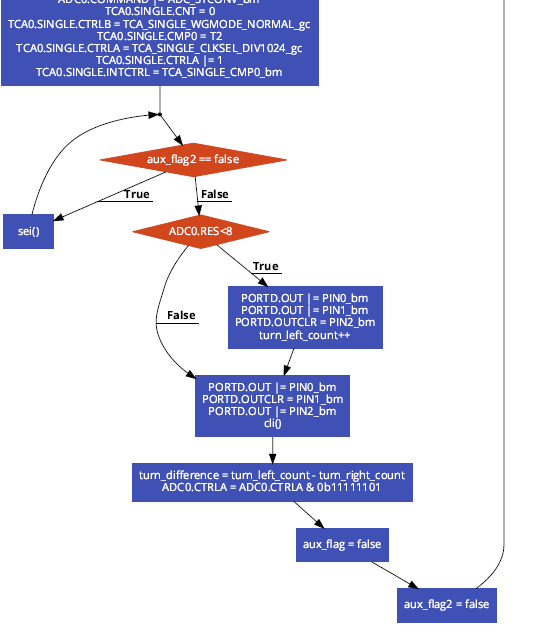
aux\_flag = true;

aux\_flag2 = true;

}

Και παράλληλα παραθέτουμε και το διάγραμμα ροής που αντιστοιχεί στο παραπάνω ερώτημα:





3) Για το τελευταίο ερώτημα, πρέπει να προσθέσουμε και ένα κουμπί (διακόπτης PIN5) κι όταν το πατήσουμε, να γίνεται στροφή 180 μοίρες και να ακολουθεί την ανάποδη διαδρομή. Για να το πετύχουμε αυτό ορίσαμε μια επιπλέον μεταβλητή, την switch\_is\_on που γίνεται αληθής όταν πατήσουμε το PIN5 και εισέλθει στην ISR που αντιστοιχεί για τον διακόπτη (PORTF\_PORT\_vect). Επιπλέον ορίζουμε μια συνάρτηση ονόματι reverse\_mod. Σε αυτή την συνάρτηση βάλαμε την δομή της while(turn\_difference < 4) από το προηγούμενο ερώτημα και αλλάξαμε την σειρά των LEDs και των μετρητών στροφών, δηλαδή τα αριστερά αντικαθίστανται με τα δεξιά και τα δεξιά με τα αριστερά. Για να μπορούμε να εκτελέσουμε την αντίστροφη λειτουργία από όποιο σημείο του κώδικα κι αν πατηθεί το PIN5, προσθέσαμε μια δομή επανάληψης while με συνθήκη switch\_is\_on == true και μέσα σε αυτήν καλείται η συνάρτηση reverse\_mod. Αυτή η δομή επανάληψης έχει τοποθετηθεί μετά από κάθε μετατροπέα, χρονιστή και συνθήκης ανισότητας μεταξύ καταχωρητή RES και κατωφλίου για να καλύψουμε όσο το δυνατόν περισσότερες περιπτώσεις. Όταν εισερχόμαστε στην συνάρτηση reverse\_mod, η διαδικασία που ακολουθούμε είναι η ίδια, με την διαφορά ότι οι κατευθύνσεις είναι αντεστραμμένες και όταν εκτελεστούν όλες οι εντολές, ο κώδικας παραμένει εγκλωβισμένος στην αντίστροφη λειτουργία.

Επομένως, με βάσει όλες αυτές τις νέες λεπτομέρειες, ο κώδικας για το 3ο ερώτημα είναι ο παρακάτω:

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdbool.h>

#define T1 15 //Time for Single Conversion

#define T2 15 //Time for Free Running

int turn\_difference = 0; //turn counter to make a route of a random room

int turn\_left\_count = 0; //integer variable to count the left turns

int turn\_right\_count = 0; //integer variable to count the right turns

bool aux\_flag = false; // auxiliary flag for the loop

bool aux\_flag2 = false; // auxiliary flag for the loop

bool switch\_is\_on = false; //check wether the switch for reverse mod is on/off

void reverse\_mod(); //this function is called when the button is pressed (switch\_is\_on => true)

int main()

{

PORTD.DIR |= PIN0\_bm;//Set pin0 as output to move right

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

PORTD.DIR |= PIN1\_bm; //Set pin1 as output to move forward

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

PORTD.DIR |= PIN2\_bm; //Set pin2 as output to move left

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

//pullup enable and Interrupt enabled with sense on both edges

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

while(turn\_difference < 4)

{

//we set single conversion

ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc; //10-bit resolution

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

ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc; //Analog Channel Selection Bits select

ADC0.DBGCTRL |= ADC\_DBGRUN\_bm; //Debug run bit mask

//Window Comparator Mode

ADC0.WINHT |= 8; //Set threshold for right movement

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

ADC0.CTRLE = ADC\_WINCM1\_bm; //Interrupt when RESULT > WINHT

sei();

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

while (switch\_is\_on == true)

{

reverse\_mod();

}

TCA0.SINGLE.CNT = 0; //Clear Counter

TCA0.SINGLE.CTRLB = TCA\_SINGLE\_WGMODE\_NORMAL\_gc; //Waveform generation mode selection ==> Normal Mode

TCA0.SINGLE.CMP0 = T1; //When CMP0 reaches this value, then interrupt will occur ==> CLOCK\_FREQUENCY/1024

TCA0.SINGLE.CTRLA = TCA\_SINGLE\_CLKSEL\_DIV1024\_gc; //Clock Selection mode ==> System Clock/1024

TCA0.SINGLE.CTRLA |= 1; //Enable

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Compare 0 Interrupt bit mask

while (switch\_is\_on == true)

{

reverse\_mod();

}

while(aux\_flag == false)

{

sei();//similar to a nop function

}

if(ADC0.RES>8)

{

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

PORTD.OUT |= PIN1\_bm; //LED for forward movement is off

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

turn\_right\_count++;

}

while (switch\_is\_on == true)

{

reverse\_mod();

}

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

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

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

aux\_flag = false;

aux\_flag2 = false;

while (switch\_is\_on == true)

{

reverse\_mod();

}

//initialize the ADC for Free-Running mode

ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc; //10-bit resolution

ADC0.CTRLA |= ADC\_FREERUN\_bm; //ADC Freerun mode bit mask

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

ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc; //Analog Channel Selection Bits select

ADC0.DBGCTRL |= ADC\_DBGRUN\_bm; //Debug run bit mask

//Window Comparator Mode

ADC0.WINLT |= 8; //Set threshold for left

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

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

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

while (switch\_is\_on == true)

{

reverse\_mod();

}

TCA0.SINGLE.CNT = 0; //Clear Counter

TCA0.SINGLE.CTRLB = TCA\_SINGLE\_WGMODE\_NORMAL\_gc; //Waveform generation mode selection ==> Normal Mode

TCA0.SINGLE.CMP0 = T2; //When CMP0 reaches this value, then interrupt will occur ==> CLOCK\_FREQUENCY/1024

TCA0.SINGLE.CTRLA = TCA\_SINGLE\_CLKSEL\_DIV1024\_gc; //Clock Selection mode ==> System Clock/1024

TCA0.SINGLE.CTRLA |= 1; //Enable

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Compare 0 Interrupt bit mask

while (switch\_is\_on == true)

{

reverse\_mod();

}

while(aux\_flag2 == false)

{

sei();

}

if(ADC0.RES<8)

{

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

PORTD.OUT |= PIN1\_bm; //LED for forward movement is off

PORTD.OUTCLR |= PIN2\_bm; //LED for left movement is on

turn\_left\_count++;

}

while (switch\_is\_on == true)

{

reverse\_mod();

}

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

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

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

cli();

ADC0.CTRLA = ADC0.CTRLA & 0b11111101; //We set again ADC to single conversion to start

//the loop all over again if needed

turn\_difference = turn\_left\_count - turn\_right\_count; //If turn\_difference reaches

//a certain value the program will end

aux\_flag = false;

aux\_flag2 = false;

}

}

ISR(ADC0\_WCOMP\_vect)

{

cli();

int intflags = ADC0.INTFLAGS;

ADC0.INTFLAGS = intflags;

ADC0.CTRLA = 0;

sei();

}

ISR(TCA0\_CMP0\_vect)

{

TCA0.SINGLE.CTRLA = 0; //Disable

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

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

aux\_flag = true;

aux\_flag2 = true;

}

ISR(PORTF\_PORT\_vect)

{

int y = PORTF.INTFLAGS; //Procedure to

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

switch\_is\_on = true; //the switch turned to on

}

void reverse\_mod()

{

//everything in here will be exexuted

//when the switch is pressed

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

PORTD.OUT |= PIN1\_bm; //LED for forward movement is off

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

//we set single conversion

ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc; //10-bit resolution

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

ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc; //Analog Channel Selection Bits select

ADC0.DBGCTRL |= ADC\_DBGRUN\_bm; //Debug run bit mask

//Window Comparator Mode

ADC0.WINHT |= 8; //Set threshold for left movement

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

ADC0.CTRLE = ADC\_WINCM1\_bm; //Interrupt when RESULT > WINHT

sei();

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

TCA0.SINGLE.CNT = 0; //Clear Counter

TCA0.SINGLE.CTRLB = TCA\_SINGLE\_WGMODE\_NORMAL\_gc; //Waveform generation mode selection ==> Normal Mode

TCA0.SINGLE.CMP0 = T1; //When CMP0 reaches this value, then interrupt will occur ==> CLOCK\_FREQUENCY/1024

TCA0.SINGLE.CTRLA = TCA\_SINGLE\_CLKSEL\_DIV1024\_gc; //Clock Selection mode ==> System Clock/1024

TCA0.SINGLE.CTRLA |= 1; //Enable

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Compare 0 Interrupt bit mask

while(aux\_flag == false)

{

sei();//similar to a nop function

}

if(ADC0.RES>8)

{

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

PORTD.OUT |= PIN1\_bm; //LED for forward movement is off

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

turn\_left\_count++;

}

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

PORTD.OUTCLR |= PIN1\_bm; //LED for forward movement is on

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

aux\_flag = false;

aux\_flag2 = false;

//initialize the ADC for Free-Running mode

ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc; //10-bit resolution

ADC0.CTRLA |= ADC\_FREERUN\_bm; //ADC Freerun mode bit mask

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

ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc; //Analog Channel Selection Bits select

ADC0.DBGCTRL |= ADC\_DBGRUN\_bm; //Debug run bit mask

//Window Comparator Mode

ADC0.WINLT |= 8; //Set threshold for right movement

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

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

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

TCA0.SINGLE.CNT = 0; //Clear Counter

TCA0.SINGLE.CTRLB = TCA\_SINGLE\_WGMODE\_NORMAL\_gc; //Waveform generation mode selection ==> Normal Mode

TCA0.SINGLE.CMP0 = T2; //When CMP0 reaches this value, then interrupt will occur ==> CLOCK\_FREQUENCY/1024

TCA0.SINGLE.CTRLA = TCA\_SINGLE\_CLKSEL\_DIV1024\_gc; //Clock Selection mode ==> System Clock/1024

TCA0.SINGLE.CTRLA |= 1; //Enable

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Compare 0 Interrupt bit mask

while(aux\_flag2 == false)

{

sei();

}

if(ADC0.RES<8)

{

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

PORTD.OUT |= PIN1\_bm; //LED for forward movement is off

PORTD.OUT |= PIN2\_bm; //LED for left movement is on

turn\_right\_count++;

}

PORTD.OUT |= PIN0\_bm; //LED for right movement is off

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

PORTD.OUT |= PIN2\_bm; //LED for left movement is off

cli();

ADC0.CTRLA = ADC0.CTRLA & 0b11111101; //We set again ADC to single conversion to start

//the loop all over again if needed

turn\_difference = turn\_left\_count - turn\_right\_count; //If turn\_difference reaches

//a certain value the program will end

aux\_flag = false;

aux\_flag2 = false;

}

Και το διάγραμμα ροής για τον παραπάνω κώδικα έχει ως εξής:

