**Εργαστήριο Προηγμένων Μικροεπεξεργαστών**

ΟΜΑΔΑ Β4 - ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 2

Μιχαηλίδης Νεκτάριος – 1072594 – 4ο έτος

Περικλής Κύρκος – 1067524 – 5ο έτος

Παραδοχές :

1. Ο τρόπος που οραματιστήκαμε την συσκευή να γνωρίζει πότε φτάνει στην θέση από την οποία ξεκίνησε είναι ο εξής :

* Το πρόγραμμα έχει μια μεταβλητή που ονομάζεται corner\_degrees και είναι αρχικά 0
* Κάθε φορά που η συσκευή κάνει μια δεξιόστροφη στροφή η μεταβλητή αυξάνεται κατά 90
* Κάθε φορά που η συσκευή κάνει μια αριστερόστροφη στροφή η μεταβλητή μειώνεται κατά 90
* Αν η μεταβλητή ποτέ φτάσει τον αριθμό 360 ( μοίρες ) τότε σημαίνει ότι έχει φτάσει στην αρχική της θέση (καθώς έχει κάνει έναν κύκλο)

2. Όταν πατηθεί το κουμπί η συσκευή, αντί για 180 μοίρες στροφή, γυρνάει συνέχει αριστερά μέχρι να δει τοίχο. Αυτό υλοποιήθηκε για την περίπτωση που κάποιος πατήσει το κουμπί κατά την διάρκεια μιας στροφής για να μην χρειαστούν παραπάνω κινήσεις.

3. Όταν πατηθεί το κουμπί και η συσκευή ξεκινήσει την αντίθετη πορεία της θα πραγματοποιήσει όσες στροφές έκανε στην φυσιολογική της πορεία. Ο τρόπος που το υλοποιήσαμε αυτό είναι με το να θυμάται η εφαρμογή τον αριθμό των στροφών που έχει πραγματοποιήσει. Συγκεκριμένα :

* Το πρόγραμμα έχει μια μεταβλητή που ονομάζεται turns και είναι αρχικά 0
* Κάθε φορά που στρίβει προσθέτει 1 στην μεταβλητή turns

Κατά την αντίθετη κίνηση , κάθε φορά που στρίβει μειώνει 1 από την μεταβλητή turns.

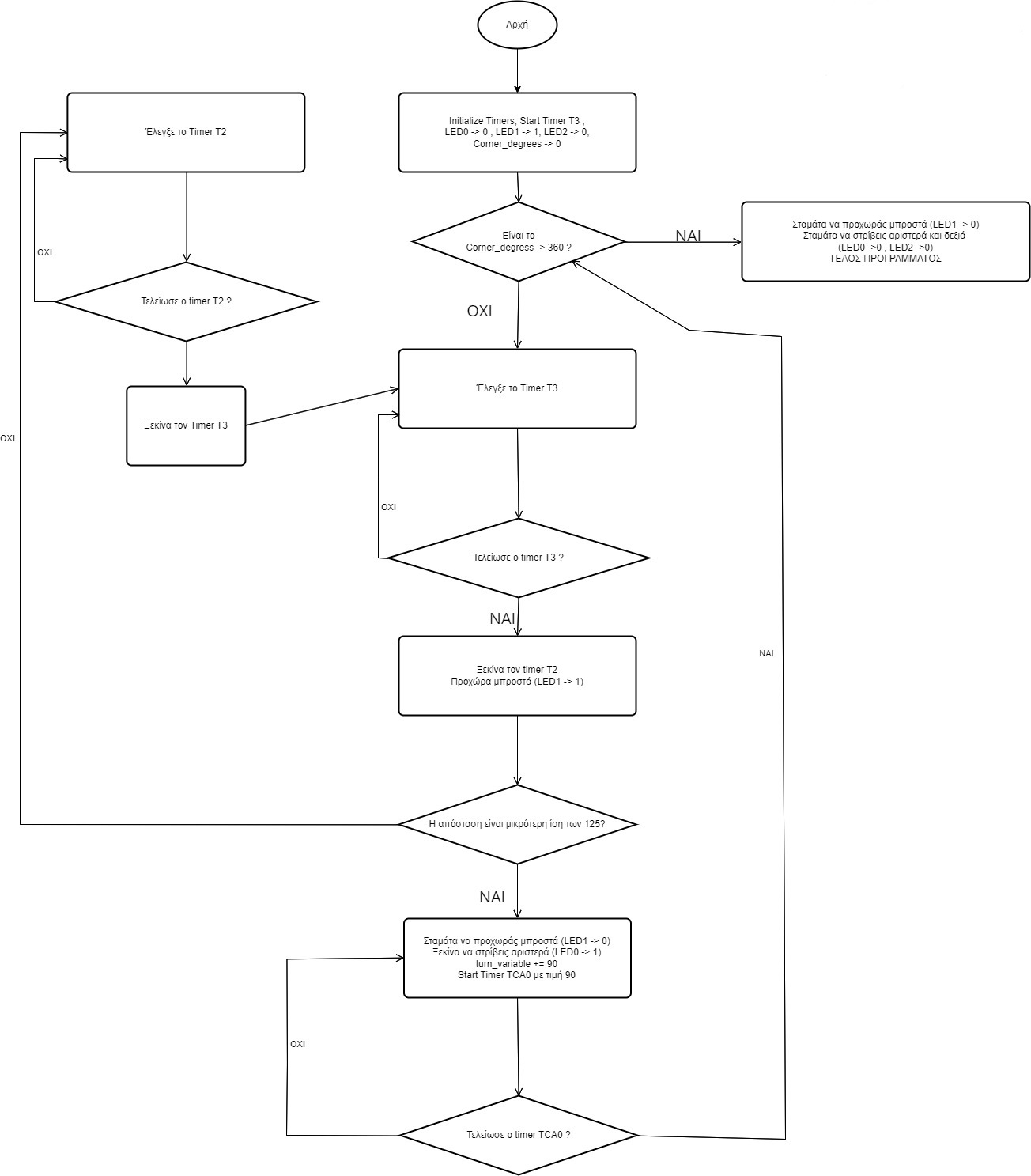
Μόλις η μεταβλητή turns ξαναφτάσει στο 0 , τότε η συσκευή έχει φτάσει στην αρχική της θέση και το πρόγραμμα σταματάει.

4. Ορίσαμε την threshold τιμή του ADC στην λειτουργεία free-running στην τιμή 50 ως high boundary καθώς αν η τιμή φτάσει ποτέ μεγαλύτερη από αυτήν τότε η συσκευή πρέπει να αρχίσει να στρίβει δεξιά (ή αριστερά στην περίπτωση που έχει πατηθεί το κουμπί ) και στην λειτουργία Single Conversion την τιμή 10 ως low boundary όπου αν η τιμή είναι μικρότερη από αυτήν σημαίνει ότι υπάρχει τοίχος κοντά στο μπροστινό μέρος της συσκευής οπότε πρέπει να στρίψει αριστερά (ή δεξιά σε περίπτωση που έχει πατηθεί το κουμπί).

5. Χρησιμοποιήσαμε τον TCB0 για τον χρονισμό των στροφών και τον TCA0 για τα T2 και T3 της εκφώνησης.

Ερωτήματα :

1. Διάγραμμα ροής για το ερώτημα 1:



ΚΩΔΙΚΑΣ ΕΡΩΤΗΜΑΤΟΣ 1 :

Περιγραφή προγράμματος :

Αρχικά το πρόγραμμα ξεκινάει τον TCA0 timer και μπαίνει στο loop της main και καλεί την συνάρτηση go\_Straight() συνέχεια. Δηλαδή η συσκευή πηγαίνει ευθεία μέχρι να χτυπήσει ο πρώτος timer (T3). Όταν τελειώσει το T3 καλείτε το interrupt του timer T3 όπου ενεργοποιείτε ο ADC Single Conversion. Αν το value του ADC είναι μικρότερο από το threshold μας (10) τότε καλείτε το interrupt του ADC όπου η συσκευή ξεκινάει να στρίβει αριστερά για T1 seconds (χρησιμοποιώντας τον timer TCB0 ). Αυτή είναι δική μας προσθήκη καθώς η εκφώνηση δεν εξακρίβωνε αν όλες οι ευθείες είναι όρθιες (90 μοίρες) οπότε χρησιμοποιήσαμε τον timer TCB0 για να κάνουμε την συσκευή να στρίβει με όσες μοίρες θέλουμε χρησιμοποιώντας την τιμή T1 για να ελέγξουμε πόση ώρα θα στρίβουμε. Όσο στρίβουμε έχουμε θέση την τιμή της μεταβλητής TURNING σε TRUE οπότε όταν το πρόγραμμα βγει από το interrupt του ADC να μην εκτελέσει την συνάρτηση go\_Straight() και αντ αυτού να περιμένει μέχρι να σκάσει ο timer T1 που συμβολίζει το τέλος της στροφικής κίνησης. Όταν τελειώσει ο timer T1 εκτελείτε το interrupt του TCB0 όπου θέτουμε την μεταβλητή TURNING πάλι FALSE. Και το πρόγραμμα συνεχίζει να τρέχει στο main. Όταν τελειώσει ο timer T2 απλώς ξαναξεκινάνε οι timers T3 και T2. Κάθε φορά που εκτελείτε η συνάρτηση go\_Left() προσθέτουμε στην global μεταβλητή Corner\_Degrees τον αριθμό 90 (που συμβολίσει τις 90 μοίρες της στροφής) Όταν η μεταβλητή φτάσει τον αριθμό 360 ξέρουμε ότι η συσκευή βρίσκεται στην αρχική της θέση οπότε το main loop σταματάει , η συσκευή κλείνει και τα 3 led και το πρόγραμμα τερματίζει.

/\*

 \* GccApplication1.c

 \*

 \* Author : nektar24

 \*/

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdbool.h>

#define T1 90;

#define T2 50;

#define T3 30;

void go\_Straight();

void go\_Left();

void start\_timer\_t2\_and\_t3();

void start\_turning();

int Corner\_degrees = 0; // when it reaches or surpasses 360 degrees it knows it has done a circle

bool T2\_PAUSE = false;

bool TURNING = false;

int main() {

    PORTD.DIR |= PIN0\_bm; //PIN0 is output

    PORTD.DIR |= PIN1\_bm; //PIN1 is output

    PORTD.DIR |= PIN2\_bm; //PIN2 is output

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

    //iniitalize the ADC

    ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc;

    ADC0.CTRLA |= ADC\_FREERUN\_bm;

    ADC0.CTRLA |= ADC\_ENABLE\_bm;

    ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc;

    ADC0.DBGCTRL |= ADC\_DBGRUN\_bm;

    ADC0.WINLT |= 10;

    ADC0.INTCTRL |= ADC\_WCMP\_bm;

    ADC0.CTRLE |= ADC\_WINCM0\_bm;

    // Enable Interrupt

    sei();

    start\_timer\_t2\_and\_t3();

    while (Corner\_degrees < 360)

    {

        while (!T2\_PAUSE && !TURNING)

        {

            go\_Straight();

        }

    }

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

    // Disable Interrupt

    cli();

}

// ADC Single Conversion interrupt

ISR(ADC0\_WCOMP\_vect){

    go\_Left();

    start\_turning();

    // Reset Interrupt Flag

    int intflags = ADC0.INTFLAGS;

    ADC0.INTFLAGS = intflags;

    ADC0.CTRLA &= ~ADC\_ENABLE\_bm;

}

// TIMER T3 INTERRUPT

ISR(TCA0\_CMP0\_vect){

    T2\_PAUSE = true;

    ADC0.COMMAND |= ADC\_STCONV\_bm;

    // Reset Interrupt Flag

    int intflags2 = TCA0.SINGLE.INTFLAGS;

    TCA0.SINGLE.INTFLAGS = intflags2;

}

// TIMER T2 INTERRUPT

ISR(TCA0\_CMP1\_vect){

    T2\_PAUSE = false;

    // Reset Interrupt Flag

    int intflags3 = TCA0.SINGLE.INTFLAGS;

    TCA0.SINGLE.INTFLAGS = intflags3;

    start\_timer\_t2\_and\_t3();

}

// TURNING TIMER INTERRUPT - TURNING FINISHED

ISR(TCB0\_CCMP\_vect){

    go\_Straight();

    TURNING = false;

    // Reset Interrupt Flag

    int intflags = TCB0.INTFLAGS;

    TCB0.INTFLAGS = intflags;

}

// TURN ON LED 1 AND TURN OFF LEDS 0 AND 2

void go\_Straight(){

    PORTD.OUTCLR = PIN0\_bm;

    PORTD.OUTCLR = PIN1\_bp; //LED1 is on

    PORTD.OUTCLR = PIN2\_bm;

    PORTD.OUT |= PIN1\_bm; //LED1 is on

}

// TURN ON LED 2 AND TURN OFF LEDS 0 AND 1

void go\_Left(){

    PORTD.OUTCLR = PIN0\_bm;

    PORTD.OUTCLR = PIN1\_bm;

    PORTD.OUTCLR = PIN2\_bp; //LED2 is on

    PORTD.OUT |= PIN2\_bm; //LED2 is on

    Corner\_degrees += 90;

}

void start\_timer\_t2\_and\_t3(){

    //-----initialize timer

    TCA0.SINGLE.CNT = 0;

    TCA0.SINGLE.CTRLB = 0;

    // TRIGGER INTERRUPT AT T3

    TCA0.SINGLE.CMP0 = T3;

    // TRIGGER INTERRUPT AT T2+T3

    TCA0.SINGLE.CMP1 = T2;

    TCA0.SINGLE.CTRLA = 0x7<<1;

    TCA0.SINGLE.CTRLA |= 1;

    TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm | TCA\_SINGLE\_CMP1\_bm;

}

void start\_turning(){

    TURNING = true;

    //-----initialize turning time to T1

    TCB0.CNT = 0;

    TCB0.CTRLB = 0;

    // TRIGGER INTERRUPT AT T1

    TCB0.CCMP = T1;

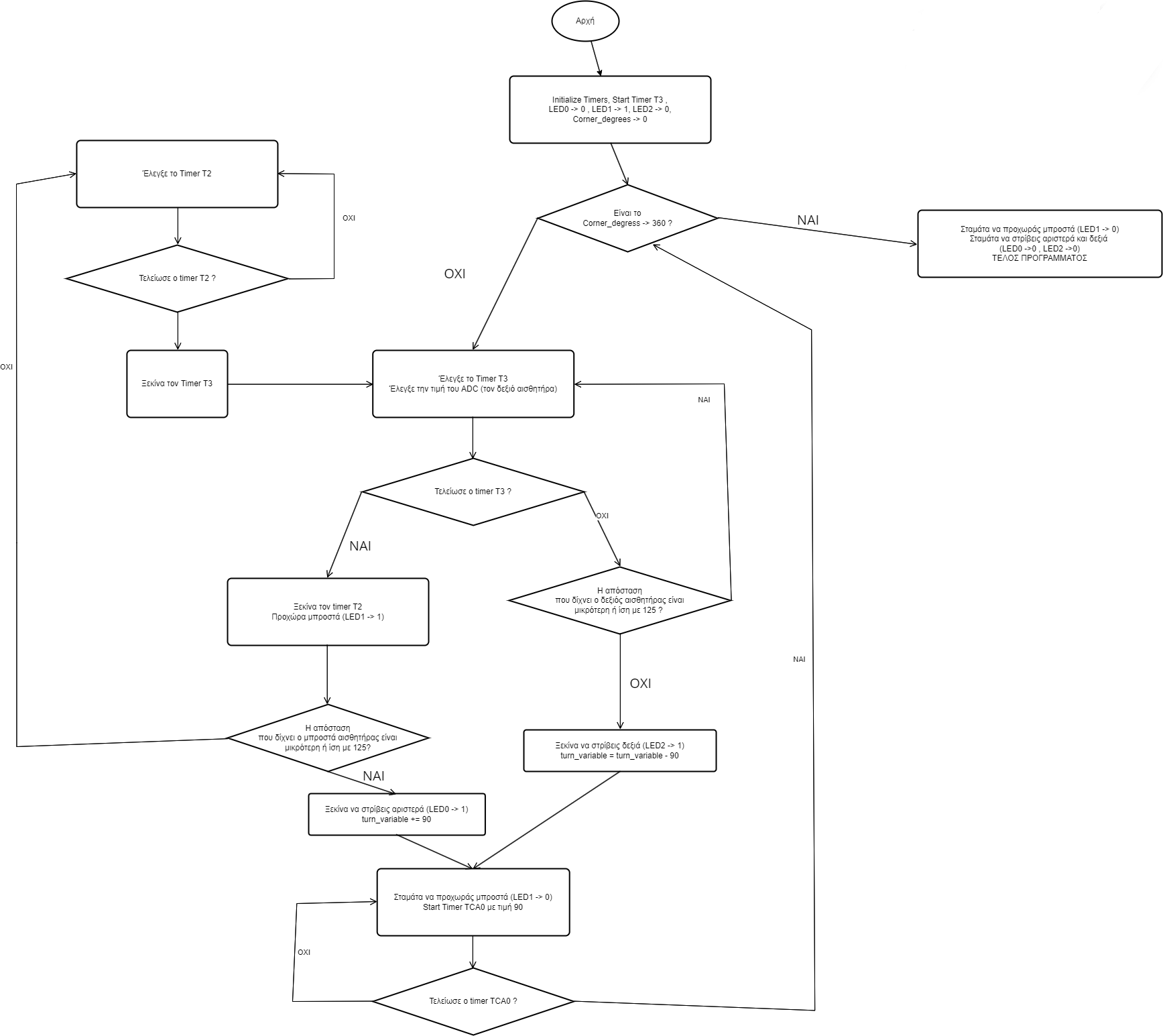
    TCB0.CTRLA = 0x7<<1;

    TCB0.CTRLA |= 1;

    TCB0.INTCTRL = TCB\_CCMPINIT\_bm;

}

1. Διάγραμμα ροής για το ερώτημα 2: (παρακαλώ καντε zoom για να το διαβάσετε)



ΚΩΔΙΚΑΣ ΕΡΩΤΗΜΑΤΟΣ 2 :

Περιγραφή Προγράμματος :

Η προσθήκη σε αυτό το ερώτημα είναι η διαφορετική χρήση του ADC καθώς και η προσθήκη της συνάρτησης go\_Right(). Στην main προσθέτουμε το high boundary 50 το οποίο επίσης θα καλεί το interrupt του ADC όταν η τιμή του είναι μεγαλύτερη από 50. Επίσης ενεργοποιούμε τον ADC μέσα στην main στην αρχή του προγράμματος σε free-running mode. Έτσι όταν η τιμή του ADC βρίσκεται ανάμεσα σε 10 και 50 (δυο αυθαίρετες τιμές που έχουν εξηγηθεί στην παραδοχή) το πρόγραμμα τρέχει μέσα στην main καλώντας την συνάρτηση go\_Straight() συνέχεια. Μόλις βαρέσει ο ADC εκτελείτε το interrupt του ADC όπου εκεί μέσα ελέγχουμε την τιμή RESULT και αν είναι μεγαλύτερη του 50 ξέρουμε ότι προήλθε από το free-running mode ( και συνεπώς από τον πλαϊνό αισθητήρα ) και έτσι εκτελούμε την συνάρτηση go\_Right() (μαζί με την start\_turning()) , ενώ αν η τιμή RESULT είναι μικρότερη από 10 ξέρουμε ότι προήλθε από την διεργασία που εξηγήσαμε στο πρώτο ερώτημα (Single Conversion Interrupt) άρα η συσκευή εντόπισε τον τοίχο χρησιμοποιώντας τον μπροστά αισθητήρα και έτσι θα κάνουμε ότι κάνει ότι έκανε στο πρώτο ερώτημα και θα καλέσει την συνάρτηση go\_Left().

Κάθε φορά που εκτελείτε η συνάρτηση go\_Right() το πρόγραμμα αφαιρεί την τιμή 90 (συμβολίζοντας τις μοίρες της στροφής) από την global μεταβλητή Corner\_degrees και έτσι ξέρουμε ότι όταν φτάσει 360 θα βρισκόμαστε στην αρχική θέση.

Αξιοσημείωτο επίσης ότι όταν χτυπήσει ο timer T3 η λειτουργεία του ADC αλλάζει από free-running mode σε single-conversion και όταν χτυπήσει ο T2 αλλάζει πίσω σε free-running mode .

/\*

 \* GccApplication1.c

 \*

 \* Author : nektar24

 \*/

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdbool.h>

#define T1 90;

#define T2 50;

#define T3 30;

void go\_Straight();

void go\_Left();

void go\_Right();

void start\_timer\_t2\_and\_t3();

void start\_turning();

int Corner\_degrees = 0; // when it reaches or surpasses 360 degrees it knows it has done a circle

bool TURNING = false;

int main() {

    PORTD.DIR |= PIN0\_bm; //PIN0 is output

    PORTD.DIR |= PIN1\_bm; //PIN1 is output

    PORTD.DIR |= PIN2\_bm; //PIN2 is output

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

    //iniitalize the ADC for Free-Running mode

    ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc;

    ADC0.CTRLA |= ADC\_FREERUN\_bm;

    ADC0.CTRLA |= ADC\_ENABLE\_bm;

    ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc;

    ADC0.DBGCTRL |= ADC\_DBGRUN\_bm;

    ADC0.WINLT |= 10;

    ADC0.WINHT |= 50;

    ADC0.INTCTRL |= ADC\_WCMP\_bm

    ADC0.CTRLE |= ADC\_WINCM0\_bm;

    // Enable Interrupt

    sei();

    // ENABLING Free-Running mode

    ADC0.COMMAND |= ADC\_STCONV\_bm;

    start\_timer\_t2\_and\_t3();

    while (Corner\_degrees < 360)

    {

        while (!TURNING) // when turning process is in order do nothing in main , wait for TCB0 to finish.

        {

            go\_Straight();

        }

    }

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

    // Disable Interrupt

    cli();

}

// ADC Single Conversion interrupt

ISR(ADC0\_WCOMP\_vect){

    if (ADC0.RES < 10) {

        go\_Left();

    } else if (ADC0.RES > 50) {

        go\_Right();

    }

    // Reset Interrupt Flag

    int intflags = ADC0.INTFLAGS;

    ADC0.INTFLAGS = intflags;

}

// TIMER T3 INTERRUPT

ISR(TCA0\_CMP0\_vect){

    // DISABLING Free-Running mode

    ADC0.COMMAND |= ADC\_STCONV\_bp;

    // Enable ADC Single Conversion

    ADC0.CTRLA &= ~ADC\_FREERUN\_bm;

    // Reset Interrupt Flag

    int intflags2 = TCA0.SINGLE.INTFLAGS;

    TCA0.SINGLE.INTFLAGS = intflags2;

}

// TIMER T2 INTERRUPT

ISR(TCA0\_CMP1\_vect){

    // ENABLING Free-Running mode

    ADC0.COMMAND |= ADC\_STCONV\_bm;

    // Reset Interrupt Flag

    int intflags3 = TCA0.SINGLE.INTFLAGS;

    TCA0.SINGLE.INTFLAGS = intflags3;

}

// TURNING TIMER INTERRUPT - TURNING FINISHED

ISR(TCB0\_CCMP\_vect){

    go\_Straight();

    TURNING = false;

    // Reset Interrupt Flag

    int intflags = TCB0.INTFLAGS;

    TCB0.INTFLAGS = intflags;

}

// TURN ON LED 1 AND TURN OFF LEDS 0 AND 2

void go\_Straight(){

    PORTD.OUTCLR = PIN0\_bp;

    PORTD.OUTCLR = PIN1\_bm; //LED1 is on

    PORTD.OUTCLR = PIN2\_bp;

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

}

// TURN ON LED 0 AND TURN OFF LEDS 1 AND 2

void go\_Right(){

    PORTD.OUTCLR = PIN0\_bm; //LED0 is on

    PORTD.OUTCLR = PIN1\_bp;

    PORTD.OUTCLR = PIN2\_bp;

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

    Corner\_degrees = Corner\_degrees - 90;

    start\_turning();

}

// TURN ON LED 2 AND TURN OFF LEDS 0 AND 1

void go\_Left(){

    PORTD.OUTCLR = PIN0\_bp;

    PORTD.OUTCLR = PIN1\_bp;

    PORTD.OUTCLR = PIN2\_bm; //LED2 is on

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    Corner\_degrees += 90;

    start\_turning();

}

void start\_timer\_t2\_and\_t3(){

    //-----initialize timer

    TCA0.SINGLE.CNT = 0;

    TCA0.SINGLE.CTRLB = 0;

    // TRIGGER INTERRUPT AT T3

    TCA0.SINGLE.CMP0 = T3;

    // TRIGGER INTERRUPT AT T2+T3 ([#DEFINE T2] IS T2+T3)

    TCA0.SINGLE.CMP1 = T2;

    TCA0.SINGLE.CTRLA = 0x7<<1;

    TCA0.SINGLE.CTRLA |= 1;

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

}

void start\_turning(){

    TURNING = true;

    //-----initialize turning time to T1

    TCB0.CNT = 0;

    TCB0.CTRLB = 0;

    // TRIGGER INTERRUPT AT T1

    TCB0.CCMP = T1;

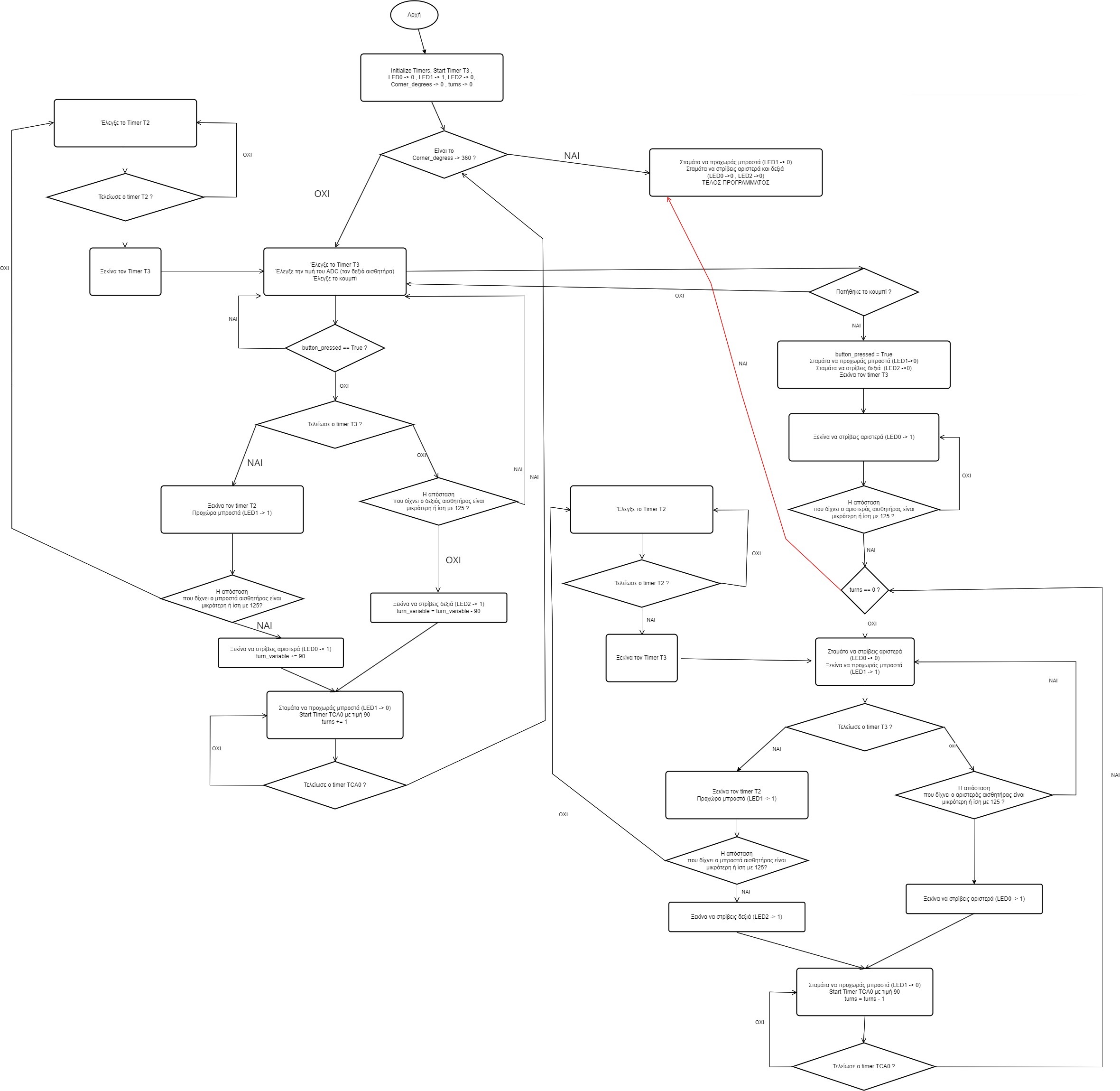
    TCB0.CTRLA = 0x7<<1;

    TCB0.CTRLA |= 1;

    TCB0.INTCTRL = TCB\_CCMPINIT\_bm;

}

1. Διάγραμμα ροής για το ερώτημα 3: (παρακαλώ κάντε zoom για να το διαβάσετε)



ΚΩΔΙΚΑΣ ΕΡΩΤΗΜΑΤΟΣ 3 :

Περιγραφή Προγράμματος :

Η προσθήκη αυτού του ερωτήματος είναι το κουμπί. Το οποίο προσθέτει ένα Interrupt , καθώς και μια μεταβλητή turns που στην αρχή είναι μηδέν και αυξάνετε μετά από κάθε στροφή , ανεξαρτήτως αν αυτή είναι αριστερή ή δεξιά. Προσθέτουμε αντίστοιχο κώδικα στην main κάτω από το corner\_degrees loop που θα τρέχει για το ταξίδι της επιστροφής σε περίπτωση που πατηθεί το κουμπί. Μόλις πατηθεί το κουμπί εκτελείτε το interrupt του κουμπιού η συσκευή εκτελεί την εντολή go\_Left() δυο φορές (για να προσομοιώσουμε 180 μοίρες. Ιδανικά θα έθετε την τιμή του T1 -> 180 και θα καλούσε την συνάρτηση μια φορά αλλά δεν κατάφερα να βρω τρόπο να το κάνω αυτό.) και επιστρέφει στο main loop (όπου είναι παγωμένο κατά την διάρκεια της στροφής επειδή η μεταβλητή TURNING είναι FALSE αλλά μόλις βαρέσει ο timer T1 του TCB0 η μεταβλητή θα γίνει true ) και ξεκινάει ο κώδικας την επιστροφής όπου λειτουργεί με τον ίδιο τρόπο πηγαίνοντας ευθεία αλλά όταν η τιμή του ADC ξεπερνάει την high boundary η συσκευή θα στρίβει αριστερά και όταν είναι μικρότερη από την low boundary θα στρίβει δεξιά. Αγνοούμε τις αλλαγές στην μεταβλητή Corner\_degrees όταν έχει πατηθεί το κουμπί , δεν μας ενδιαφέρει η τιμή της , αλλά αξιοσημείωτο ότι με κάθε στροφή είτε αριστερή είτε δεξιά ελαττώνουμε την τιμή της μεταβλητής turns . Μόλις η μεταβλητή turns φτάσει στο 0 ξέρουμε ότι φτάσαμε στην αρχική μας θέση και το πρόγραμμα φτάνει στην δεύτερη τερματική του κατάσταση (όπως φαίνεται στο κόκκινο βελάκι στο διάγραμμα ροής) και το πρόγραμμα σταματάει.

/\*

 \* GccApplication1.c

 \*

 \* Author : nektar24

 \*/

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdbool.h>

#define T1 90;

#define T2 50;

#define T3 30;

void go\_Straight();

void go\_Left();

void go\_Right();

void start\_timer\_t2\_and\_t3();

void start\_turning();

int Corner\_degrees = 0; // when it reaches or surpasses 360 degrees it knows it has done a circle

bool TURNING = false;

int turns = 0; // keeps track of the turns so it can return back when button is pressed

bool button\_pressed = false;

int main() {

    PORTD.DIR |= PIN0\_bm; //PIN0 is output

    PORTD.DIR |= PIN1\_bm; //PIN1 is output

    PORTD.DIR |= PIN2\_bm; //PIN2 is output

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

    //iniitalize the ADC for Free-Running mode

    ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc;

    ADC0.CTRLA |= ADC\_FREERUN\_bm;

    ADC0.CTRLA |= ADC\_ENABLE\_bm;

    ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc;

    ADC0.DBGCTRL |= ADC\_DBGRUN\_bm;

    ADC0.WINLT |= 10;

    ADC0.WINHT |= 50;

    ADC0.INTCTRL |= ADC\_WCMP\_bm

    ADC0.CTRLE |= ADC\_WINCM0\_bm;

    // Listen for interrupt on bit 5

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

    // Enable Interrupt

    sei();

    // ENABLING Free-Running mode

    ADC0.COMMAND |= ADC\_STCONV\_bm;

    start\_timer\_t2\_and\_t3();

    while (Corner\_degrees < 360 && !button\_pressed)

    {

        while (!TURNING ) // when turning process is in order do nothing in main , wait for TCB0 to finish.

        {

            go\_Straight();

        }

    }

    while (turns > 0 && button\_pressed){

        while (!TURNING ) // when turning process is in order do nothing in main , wait for TCB0 to finish.

        {

            go\_Straight();

        }

    }

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

    // Disable Interrupt

    cli();

}

// Button Press INTERRUPT

ISR(PORTF\_PORT\_vect){

    button\_pressed = true;

    go\_Left();

    start\_turning();

    go\_Left();

    start\_turning();

    // Reset Interrupt Flag

    int intflags = PORTF.INTFLAGS;

    PORTF.INTFLAGS = intflags;

}

// ADC Single Conversion interrupt

ISR(ADC0\_WCOMP\_vect){

    if (ADC0.RES < 10) {

        if (button\_pressed){

            turns = turns - 1;

            go\_Right();

        } else {

            go\_Left();

            turns += 1

        }

    } else if (ADC0.RES > 50) {

        if (button\_pressed){

            turns = turns - 1;

            go\_Left();

        } else {

            go\_Right();

            turns += 1

        }

    // Reset Interrupt Flag

    int intflags = ADC0.INTFLAGS;

    ADC0.INTFLAGS = intflags;

}

// TIMER T3 INTERRUPT

ISR(TCA0\_CMP0\_vect){

    // DISABLING Free-Running mode

    ADC0.COMMAND |= ADC\_STCONV\_bp;

    // Enable ADC Single Conversion

    ADC0.CTRLA &= ~ADC\_FREERUN\_bm;

    // Reset Interrupt Flag

    int intflags2 = TCA0.SINGLE.INTFLAGS;

    TCA0.SINGLE.INTFLAGS = intflags2;

}

// TIMER T2 INTERRUPT

ISR(TCA0\_CMP1\_vect){

    // ENABLING Free-Running mode

    ADC0.COMMAND |= ADC\_STCONV\_bm;

    // Reset Interrupt Flag

    int intflags3 = TCA0.SINGLE.INTFLAGS;

    TCA0.SINGLE.INTFLAGS = intflags3;

}

// TURNING TIMER INTERRUPT - TURNING FINISHED

ISR(TCB0\_CCMP\_vect){

    go\_Straight();

    NOTINATURN = true;

    // Reset Interrupt Flag

    int intflags = TCB0.INTFLAGS;

    TCB0.INTFLAGS = intflags;

}

// TURN ON LED 1 AND TURN OFF LEDS 0 AND 2

void go\_Straight(){

    PORTD.OUTCLR = PIN0\_bp;

    PORTD.OUTCLR = PIN1\_bm; //LED1 is on

    PORTD.OUTCLR = PIN2\_bp;

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

}

// TURN ON LED 0 AND TURN OFF LEDS 1 AND 2

void go\_Right(){

    PORTD.OUTCLR = PIN0\_bm; //LED0 is on

    PORTD.OUTCLR = PIN1\_bp;

    PORTD.OUTCLR = PIN2\_bp;

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    PORTD.OUT |= PIN2\_bm; //LED2 is off

    Corner\_degrees = Corner\_degrees - 90;

}

// TURN ON LED 2 AND TURN OFF LEDS 0 AND 1

void go\_Left(){

    PORTD.OUTCLR = PIN0\_bp;

    PORTD.OUTCLR = PIN1\_bp;

    PORTD.OUTCLR = PIN2\_bm; //LED2 is on

    PORTD.OUT |= PIN0\_bm; //LED0 is off

    PORTD.OUT |= PIN1\_bm; //LED1 is off

    Corner\_degrees += 90;

}

void start\_timer\_t2\_and\_t3(){

    //-----initialize timer

    TCA0.SINGLE.CNT = 0;

    TCA0.SINGLE.CTRLB = 0;

    // TRIGGER INTERRUPT AT T3

    TCA0.SINGLE.CMP0 = T3;

    // TRIGGER INTERRUPT AT T2+T3 ([#DEFINE T2] IS T2+T3)

    TCA0.SINGLE.CMP1 = T2;

    TCA0.SINGLE.CTRLA = 0x7<<1;

    TCA0.SINGLE.CTRLA |= 1;

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

}

void start\_turning(){

    TURNING = true;

    //-----initialize turning time to T1

    TCB0.CNT = 0;

    TCB0.CTRLB = 0;

    // TRIGGER INTERRUPT AT T1

    TCB0.CCMP = T1;

    TCB0.CTRLA = 0x7<<1;

    TCB0.CTRLA |= 1;

    TCB0.INTCTRL = TCB\_CCMPINIT\_bm;

}