Εργαστήριο Προηγμένοι Μικροεπεξεργαστές

**Εργαστηριακή Άσκηση 2**

**Έξυπνη Οικιακή Συσκευή με Κίνηση στο Χώρο**

**Στοιχεία Ομάδας :**

**Ταμβάκης Θωμάς-Χρυσοβαλάντης, ΑΜ:1072631**

**Διαμαντόπουλος Ιωάννης , ΑΜ:1064281**

**Ομάδα : Β3**

**Ερώτημα 1.α - Κώδικας για τη κίνηση της οικιακής συσκευής όταν το δωμάτιο είναι τετράγωνο.**

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdio.h>

int turns=0; //Left turns

int endOfTurn = 0;

#define turn\_time 20 //Time for Turn

int main(){

PORTD.DIR |= 0b00000111; //PINS 012 are output

PORTD.OUT |= 0b00000111; //left turn off / forward movement off / right turn off

///////////// Initialize the ADC for single conversion mode /////////////

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

ADC0.WINLT |= 10; //Set threshold

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

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

sei(); // Enable Interrupts

while(turns<4){ //When do 4 turns, stop

PORTD.OUTCLR = 0b00000010; //left turn off / forward movement on / right turn off

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

while(endOfTurn == 0 && ADC0.RES < ADC0.WINLT){ //Block program while turn has not completed

;

}

*printf*("Turn %d", turns); //print in the console the number of turns

endOfTurn = 0;

}

cli(); //Disable Interrupts

}

ISR(ADC0\_WCOMP\_vect){ //ISR for ADC

int intflags = ADC0.INTFLAGS;

ADC0.INTFLAGS = intflags;

PORTD.OUT |= 0b00000010; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000100; //left turn on / forward movement off / right turn off

//////////////////// TIMER for left turn ////////////////////////////////

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = turn\_time; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

}

ISR(TCA0\_CMP0\_vect){ //ISR for turn Timer

TCA0.SINGLE.CTRLA = 0; //Disable

//clear flag

int intflags = TCA0.SINGLE.INTFLAGS;

TCA0.SINGLE.INTFLAGS=intflags;

endOfTurn = 1; //Turn has completed

turns++;

PORTD.OUT |= 0b00000100; //left turn off / forward movement off / right turn off

}

**Ερώτημα 1.β - Κώδικας για τη κίνηση της οικιακής συσκευής όταν το δωμάτιο είναι τυχαίο (Γίνεται χρήση και των δύο λειτουργιών του ADC).**

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdio.h>

int leftturns=0; //number of left turns

int w=0;

int adc = 0;

#define turn\_time 10

#define t 30

#define t2 20

int main(){

PORTD.DIR |= 0b00000111; //PINS 012 are output

PORTD.OUT |= 0b00000111; //left turn off / forward movement off / right turn off

//////////////// Initialize the ADC for Free-Running Mode conversion 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

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

//Window Comparator Mode

ADC0.WINHT |= 10; //Set threshold

ADC0.WINLT |= 10; //Set threshold

sei(); //Enable Interrupts

while(leftturns<4){ //When do 4 leftturns, stop

PORTD.OUTCLR = 0b00000010; //left turn off / forward movement on / right turn off

w=0;

adc=0;

//T=3 sec

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = t; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

while(w==0){

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

ADC0.CTRLE = 0b00000010; //Interrupt when RESULT > WINHT

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

if(adc==1){

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = turn\_time; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

PORTD.OUT |= 0b00000010; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000001; //left turn off / forward movement off / right turn on

while(w==0){ //Block program while timer has not completed

}

leftturns--;

}

}

PORTD.OUT |= 0b00000001; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000010; //left turn off / forward movement on / right turn off

adc=0;

w=0;

//T2=2 sec

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = t2; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

while(w==0){

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

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

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

ADC0.CTRLE = 0b00000001; //Interrupt when RESULT < WINLT

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

if(adc==1){

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = turn\_time; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

PORTD.OUT |= 0b00000010; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000100; //left turn on / forward movement off / right turn off

while(w==0){ //Block program while timer has not completed

}

leftturns++;

}

}

PORTD.OUT |= 0b00000001; //left turn off / forward movement off / right turn off

*printf*("Turn %d", leftturns);

}

cli();

}

ISR(ADC0\_WCOMP\_vect){ //ISR for ADC

int intflags = ADC0.INTFLAGS;

ADC0.INTFLAGS = intflags;

ADC0.CTRLA &= ~ADC\_ENABLE\_bm; //Disable ADC

adc=1;

}

ISR(TCA0\_CMP0\_vect){ //ISR for turn Timer

TCA0.SINGLE.CTRLA = 0; //Disable

//clear flag

int intflags = TCA0.SINGLE.INTFLAGS;

TCA0.SINGLE.INTFLAGS=intflags;

w=1;

PORTD.OUT |= 0b00000100; //left turn off / forward movement off / right turn off

}

**Ερώτημα 1.γ - Κώδικας για τη κίνηση της οικιακής συσκευής με ανάποδη πορεία**

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdio.h>

int leftturns=0; //number of left turns

int rightturns=0; //number of left turns

int buttonIsPressed=0;

int w=0;

int adc = 0;

#define turn\_time 10

#define t 30

#define t2 20

#define t3 20

int main(){

PORTD.DIR |= 0b00000111; //PINS 012 are output

PORTD.OUT |= 0b00000111; //left turn off / forward movement off / right turn off

//////////////// Initialize the ADC for Free-Running Mode conversion 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

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

//Window Comparator Mode

ADC0.WINHT |= 10; //Set threshold

ADC0.WINLT |= 10; //Set threshold

sei(); //Enable Interrupts

while(leftturns<4 && rightturns<leftturns){ //When do 4 leftturns or as many rightturns as left, stop

if(buttonIsPressed == 0){

w=0;

adc=0;

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

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

while (buttonIsPressed==0) {

PORTD.OUTCLR = 0b00000111; //left turn on / forward movement on / right turn on

}

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = t3; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

while(w==0){

;

}

PORTD.OUT |= 0b00000111; //left turn off / forward movement off / right turn off

}

if(buttonIsPressed == 0){

PORTD.OUTCLR = 0b00000010; //left turn off / forward movement on / right turn off

w=0;

adc=0;

//T=3 sec

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = t; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

while(w==0){

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

ADC0.CTRLE = 0b00000010; //Interrupt when RESULT > WINHT

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

if(adc==1){

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = turn\_time; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

PORTD.OUT |= 0b00000010; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000001; //left turn off / forward movement off / right turn on

while(w==0){ //Block program while timer has not completed

}

leftturns--;

}

}

PORTD.OUT |= 0b00000001; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000010; //left turn off / forward movement on / right turn off

adc=0;

w=0;

//T2=2 sec

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = t2; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

while(w==0){

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

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

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

ADC0.CTRLE = 0b00000001; //Interrupt when RESULT < WINLT

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

if(adc==1){

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = turn\_time; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

PORTD.OUT |= 0b00000010; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000100; //left turn on / forward movement off / right turn off

while(w==0){ //Block program while timer has not completed

}

leftturns++;

}

}

PORTD.OUT |= 0b00000001; //left turn off / forward movement off / right turn off

printf("Turn %d", leftturns);

} else { /////////////////////////////////////////////////////////////////////////////////////////////////button is pressed ///////////////////////////////////////////////////////////

PORTD.OUTCLR = 0b00000010; //left turn off / forward movement on / right turn off

w=0;

adc=0;

//T=3 sec

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = t; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

while(w==0){

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

ADC0.CTRLE = 0b00000001; //Interrupt when RESULT < WINLT

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

if(adc==1){

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = turn\_time; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

PORTD.OUT |= 0b00000010; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000100; //left turn on / forward movement off / right turn off

while(w==0){ //Block program while timer has not completed

}

rightturns--;

}

}

PORTD.OUT |= 0b00000001; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000010; //left turn off / forward movement on / right turn off

adc=0;

w=0;

//T2=2 sec

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = t2; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

while(w==0){

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

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

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

ADC0.CTRLE = 0b00000010; //Interrupt when RESULT > WINHT

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

if(adc==1){

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

TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA\_SINGLE\_WGMODE\_NORMAL\_gc)

TCA0.SINGLE.CMP0 = turn\_time; //When reaches this value -> interrupt CLOCK FREQUENCY/1024

TCA0.SINGLE.CTRLA = 0x7<<1; //TCA\_SINGLE\_CLKSEL\_DIV1024\_gc

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

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; //Interrupt Enable (=0x10)

PORTD.OUT |= 0b00000010; //left turn off / forward movement off / right turn off

PORTD.OUTCLR = 0b00000001; //left turn off / forward movement off / right turn on

while(w==0){ //Block program while timer has not completed

}

rightturns++;

}

}

PORTD.OUT |= 0b00000001; //left turn off / forward movement off / right turn off

printf("Turn %d", leftturns);

}

}

cli();

}

ISR(ADC0\_WCOMP\_vect){ //ISR for ADC

int intflags = ADC0.INTFLAGS;

ADC0.INTFLAGS = intflags;

ADC0.CTRLA &= ~ADC\_ENABLE\_bm; //Disable ADC

adc=1;

}

ISR(TCA0\_CMP0\_vect){ //ISR for turn Timer

TCA0.SINGLE.CTRLA = 0; //Disable

//clear flag

int intflags = TCA0.SINGLE.INTFLAGS;

TCA0.SINGLE.INTFLAGS=intflags;

w=1;

PORTD.OUT |= 0b00000100; //left turn off / forward movement off / right turn off

}

ISR(PORTF\_PORT\_vect){

//clear the interrupt flag

int y = PORTF.INTFLAGS;

PORTF.INTFLAGS=y;

buttonIsPressed=1;

}

**Ερώτημα 2.α – Επεξήγηση κώδικα και διάγραμμα ροής**

Σχετικά με τον κώδικα, αρχικά, με την εντολή *PORTD.DIR |= 0b00000111* θέτουμε τις εξόδους. Πιο συγκεκριμένα, το LED0 αφορά τη δεξιά κίνηση, το LED1 αφορά τη μπροστά κίνηση, ενώ το LED2 αφορά την αριστερή κίνηση (το LED0 δεν χρησιμοποιείται στο συγκεκριμένο ερώτημα). Στη συνέχεια, με την εντολή *PORTD.OUT |= 0b00000111* απενεργοποιούμε και τα τρία LED. Επιπρόσθετα, ορίζουμε τον ADC με τις ακόλουθες εντολές :

* Με την εντολή *ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc* ορίζουμε ότι το Resolution θα είναι 10 bit για τη μετατροπή της αναλογικής τιμής σε ψηφιακή.
* Λόγω της μη χρησιμοποίησης της εντολής *ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc* ο ADC θα λειτουργήσει σε Single Conversion Mode (προκαθορισμένη λειτουργία).
* Με την εντολή *ADC0.CTRLA |= ADC\_ENABLE\_bm* ενεργοποιείται ο ADC.
* Με την εντολή *ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc* ορίζεται το bit με το οποίο συνδέεται ο ADC.
* Με την εντολή *ADC0.WINLT |= 10* θέτουμε το κατώφλι στον καταχωρητή *ADC0.WINLT.*
* Με την εντολή *ADC0.INTCTRL |= ADC\_WCMP\_bm* ενεργοποιείται η λειτουργία διακοπτών.
* Τέλος, με την εντολή *ADC0.CTRLE |= ADC\_WINCM0\_bm* στον ADC0.CTRLE θέτουμε το mode που επιθυμούμε (δηλαδή επειδή έχουμε ενεργοποιήσει το bit0 θα γίνει η σύγκριση RES < WINLT, και αν αληθεύει θα πραγματοποιηθεί διακοπή)

Εφόσον ενεργοποιήσαμε τον ADC, ενεργοποιούμε τις διακοπές με την εντολή *sei()*. Εν συνεχεία, το πρόγραμμά μας θα εισαχθεί σε μία while loop, στην οποία θα βρίσκεται έως ότου κάνει την περίμετρο του τετράγωνου δωματίου, δηλαδή έως ότου κάνει τέσσερις αριστερές στροφές. Όταν κάνει τέσσερίς στροφές, δηλαδή όταν ολοκληρώσει την περίμετρο του δωματίου το πρόγραμμα εξέρχεται της while loop και οι διακοπές απενεργοποιούνται με την εντολή *cli()*. Για το περιεχόμενο της while loop θα μιλήσουμε στη συνέχεια. Ας αναφερθούμε στις δύο ISR που βρίσκονται έξω από τη main() του προγράμματος.

Στην πρώτη ISR μεταφέρεται το πρόγραμμα όταν γίνεται η διακοπή στον ADC, δηλαδή όταν κάποια συνθήκη στον καταχωρητή ADC0.CTRLE προκαλεί διακοπή (πχ. *RESULT <* WINLT). Σε αυτό το χρονικό διάστημα με την εντολή *PORTD.OUT |= 0b00000010* απενεργοποιούνται όλα τα LED, ενώ με την εντολή *PORTD.OUTCLR = 0b00000100* ενεργοποιείται το LED2 το οποίο φανερώνει ότι η συσκευή κάνει αριστερή στροφή σε αυτό το χρονικό διάστημα (*turn\_time*) που έχουμε ορίσει.

Η δεύτερη ISR έχει να κάνει με το χρόνο που η συσκευή κάνει την αριστερή στροφή. Αξίζει να τονίσουμε πως όταν το πρόγραμμα εισέρχεται σε αυτή την ISR, με την εντολή *turns++* αυξάνεται κατά ένα η μεταβλητή turns και με την εντολή *PORTD.OUT |= 0b00000100* απενεργοποιείται το LED2 που αφορά την αριστερή στροφή (καθώς τη συγκεκριμένη χρονική στιγμή η συσκευή έχει ολοκληρώσει την αριστερή στροφή).

Σε ότι αφορά το περιεχόμενο της while loop, αρχικά θέτουμε το LED1 να είναι ανοιχτό καθώς ακόμα δεν έχει πραγματοποιηθεί κάποια στροφή (δεν έχει εμφανιστεί εμπόδιο μπροστά στη συσκευή) και η συσκευή κινείται ευθεία. Ακόμη, με την εντολή *ADC0.COMMAND |= ADC\_STCONV\_bm* ξεκινάει η διαδικασία του conversion. Με την εντολή *while(endOfTurn == 0 && ADC0.RES < ADC0.WINLT)* το πρόγραμμα περιμένει να ολοκληρωθεί η στροφή. Όταν συμβεί αυτό, και μεταφερθεί στην ISR, η συσκευή ολοκληρώνει τη στροφή και επιστρέφοντας στη main() εκτυπώνεται στην κονσόλα με την εντολή *printf("Turn %d", turns)* η μεταβλητή turns, που μας φανερώνει τον αριθμό των στροφών που έχει πραγματοποιήσει η συσκευή.

Ακολουθεί το διάγραμμα ροής :

**Εικόνα που περιέχει διάγραμμα

Περιγραφή που δημιουργήθηκε αυτόματα**

**Ερώτημα 2.β – Επεξήγηση κώδικα και διάγραμμα ροής**

Σχετικά με τον κώδικα, αρχικά θέτουμε μηδέν μια μεταβλητή *w*, την οποία αξιοποιούμε για να «σκάσει» ο timer. Επίσης η μεταβλητή *adc,* η οποία στην αρχή έχει τεθεί ‘0’, γίνεται ‘1’ κάθε φορά που το πρόγραμμα μπαίνει στην ISR του ADC. Ακόμη, η μεταβλητή *leftturns*, έχει τεθεί και αυτή ‘0’, και μετράει τον αριθμό των αριστερών στροφών.

Στην main(), με την εντολή *PORTD.DIR |= 0b00000111* θέτουμε τις εξόδους. Πιο συγκεκριμένα, το LED0 αφορά τη δεξιά κίνηση, το LED1 αφορά τη μπροστά κίνηση, ενώ το LED2 αφορά την αριστερή κίνηση. Στη συνέχεια, με την εντολή *PORTD.OUT |= 0b00000111* απενεργοποιούμε και τα τρία LED. Επιπρόσθετα, ορίζουμε τον ADC με τις ακόλουθες εντολές :

* Με την εντολή *ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc* ορίζουμε ότι το Resolution θα είναι 10 bit για τη μετατροπή της αναλογικής τιμής σε ψηφιακή.
* Με την εντολή *ADC0.CTRLA |= ADC\_RESSEL\_10BIT\_gc* ο ADC θα λειτουργήσει σε Free-Running Mode (στον ADC.CTRLA τίθεται ‘1’ στο Free-Running bit.
* Με την εντολή *ADC0.CTRLA |= ADC\_ENABLE\_bm* ενεργοποιείται ο ADC.
* Με την εντολή *ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc* ορίζεται το bit με το οποίο συνδέεται ο ADC.
* Με την εντολή *ADC0.WINHT |= 10* θέτουμε το κατώφλι στον καταχωρητή *ADC0.WINHT.*
* Αντίστοιχα, με την εντολή *ADC0.WINLT |= 10* θέτουμε το κατώφλι στον καταχωρητή *ADC0.WINLT.*

Εφόσον ενεργοποιήσαμε τον ADC, ενεργοποιούμε τις διακοπές με την εντολή *sei()*. Εν συνεχεία, το πρόγραμμά μας θα εισαχθεί σε μία while loop, στην οποία θα βρίσκεται έως ότου κάνει την περίμετρο του τυχαίου δωματίου, δηλαδή έως ότου η διαφορά αριστερών με δεξιών στροφών γίνει 4 (left turns – right turns). Αναλυτικότερα, ο υπολογισμός της διαφοράς γίνεται μειώνοντας κατά ένα τη μεταβλητή *leftturns* σε κάθε δεξιά στροφή και αυξάνοντάς τη κατά ένα σε κάθε αριστερή στροφή. Όταν η διαφορά αυτή, γίνει 4, δηλαδή όταν η συσκευή ολοκληρώσει την περίμετρο του τυχαίου δωματίου το πρόγραμμα εξέρχεται της while loop και οι διακοπές απενεργοποιούνται με την εντολή *cli()*. Για το περιεχόμενο της while loop θα μιλήσουμε στη συνέχεια. Ας αναφερθούμε στις δύο ISR που βρίσκονται έξω από τη main() του προγράμματος.

Στην πρώτη ISR μεταφέρεται το πρόγραμμα όταν γίνεται η διακοπή στον ADC, δηλαδή όταν κάποια συνθήκη στον καταχωρητή ADC0.CTRLE προκαλεί διακοπή (πχ. *RESULT <* WINLT ή *RESULT > WINHT*). Σε αυτή, η μεταβλητή *adc*, γίνεται ‘1’, έτσι ώστε ο χρόνος του timer να αρχίσει αφού έχουμε επιστρέψει από την ISR (δηλαδή έχει βρεθεί εμπόδιο). Στη συνέχεια αφού μπει στην *if(adc==1)* θα πάει στην ISR του timer και θα γίνουν οι αλλαγές στα LED (πχ θα ανάψει το LED2 για την αριστερή στροφή ή θα ανάψει το LED0 για την δεξιά στροφή με τις εντολές *PORTD.OUTCLR = 0b00000001* και *PORTD.OUTCLR = 0b00000100*)*.*

Στην δεύτερη ISR μεταφέρεται το πρόγραμμα όταν γίνεται διακοπή λόγω του timer. Το πρόγραμμα θέτει τη μεταβλητή *w* ίση με ‘1’, ώστε να εξέλθει της loop ‘while(w==0)’ όταν επιστρέψει στην main(). Επιπρόσθετα, σε αυτή την ISR, με την εντολή *PORTD.OUT |= 0b00000100* απενεργοποιείται το LED2, το οποίο αφορά την αριστερή στροφή.

Σε ότι αφορά το περιεχόμενο της while loop (*while(leftturns<4)*), με την εντολή *PORTD.OUTCLR = 0b00000010* θέτουμε το LED1 να είναι ανοιχτό όσο δεν έχει πραγματοποιηθεί κάποια στροφή (δεν έχει εμφανιστεί εμπόδιο μπροστά στη συσκευή) και η συσκευή κινείται ευθεία.

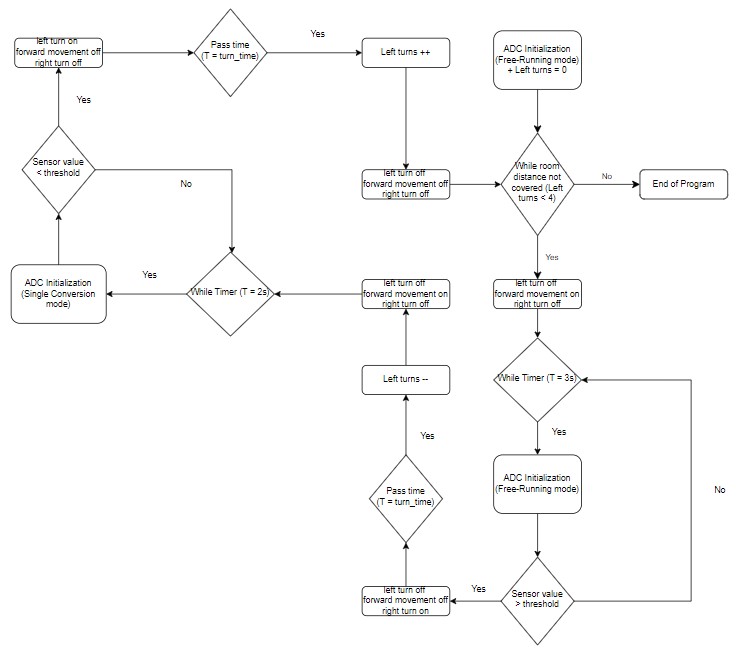
Στη συνέχεια, πραγματοποιούνται οι εναλλαγές από Free-Running Mode σε Single Conversion και αντίστροφα. Πιο συγκεκριμένα, ξεκινάμε σε Free-Running Mode και σε διάστημα Τ = 3 sec ελέγχουμε συνεχόμενα αν έρθει τιμή μεγαλύτερη του κατωφλίου (RESULT > WINHT). Αν γίνει αυτό, τότε πραγματοποιείται η δεξιά στροφή όπως περιγράφηκε προηγουμένως. Αν παρέλθει το διάστημα των 3 sec, ή αν παρουσιαστεί κάποιο εμπόδιο, τότε το πρόγραμμα εξέρχεται από το Free – Running Mode. Ακολουθεί η εναλλαγή σε Single Conversion Mode (αρχικοποίηση ADC χωρίς την εντολή *ADC0.CTRLA |= ADC\_FREERUN\_bm*) με την οποία ελέγχουμε μία και μόνο φορά, αν η τιμή του RES (τιμή αισθητήρα) είναι μικρότερη του κατωφλίου (RESULT < WINLT). Το σύστημα βρίσκεται σε Single Conversion για Τ2 = 2 sec, ή μέχρι να βρεθεί εμπόδιο.

Τέλος, στη main() εκτυπώνεται στην κονσόλα με την εντολή *printf("Turn %d", leftturns)* η μεταβλητή leftturns, που μας φανερώνει την διαφορά (left turns – right turns) που επεξηγήθηκε παραπάνω.

Ακολουθούν οι 4 περιπτώσεις εναλλαγής των LED κατά τη διάρκεια εκτέλεσης του προγράμματος :

* Ευθεία 🡪 Αριστερά :
* Με την εντολή *PORTD.OUT |= 0b00000010* απενεργοποιείται το LED1 (εμπρός κίνηση)
* Με την εντολή *PORTD.OUTCLR = 0b00000100* ενεργοποιείται το LED2 (στροφή αριστερά)
* Ευθεία 🡪 Δεξιά :
* Με την εντολή *PORTD.OUT |= 0b00000010* απενεργοποιείται το LED1 (εμπρός κίνηση)
* Με την εντολή *PORTD.OUTCLR = 0b00000001* ενεργοποιείται το LED0 (στροφή δεξιά)
* Αριστερά 🡪 Ευθεία :
* Με την εντολή *PORTD.OUT |= 0b00000100* απενεργοποιείται το LED2 (στροφή αριστερά)
* Με την εντολή *PORTD.OUTCLR = 0b00000010* ενεργοποιείται το LED1 (εμπρός κίνηση)
* Δεξιά 🡪 Ευθεία :
* Με την εντολή *PORTD.OUT |= 0b00000001* απενεργοποιείται το LED0 (στροφή δεξιά)
* Με την εντολή *PORTD.OUTCLR = 0b00000010* ενεργοποιείται το LED1 (εμπρός κίνηση)

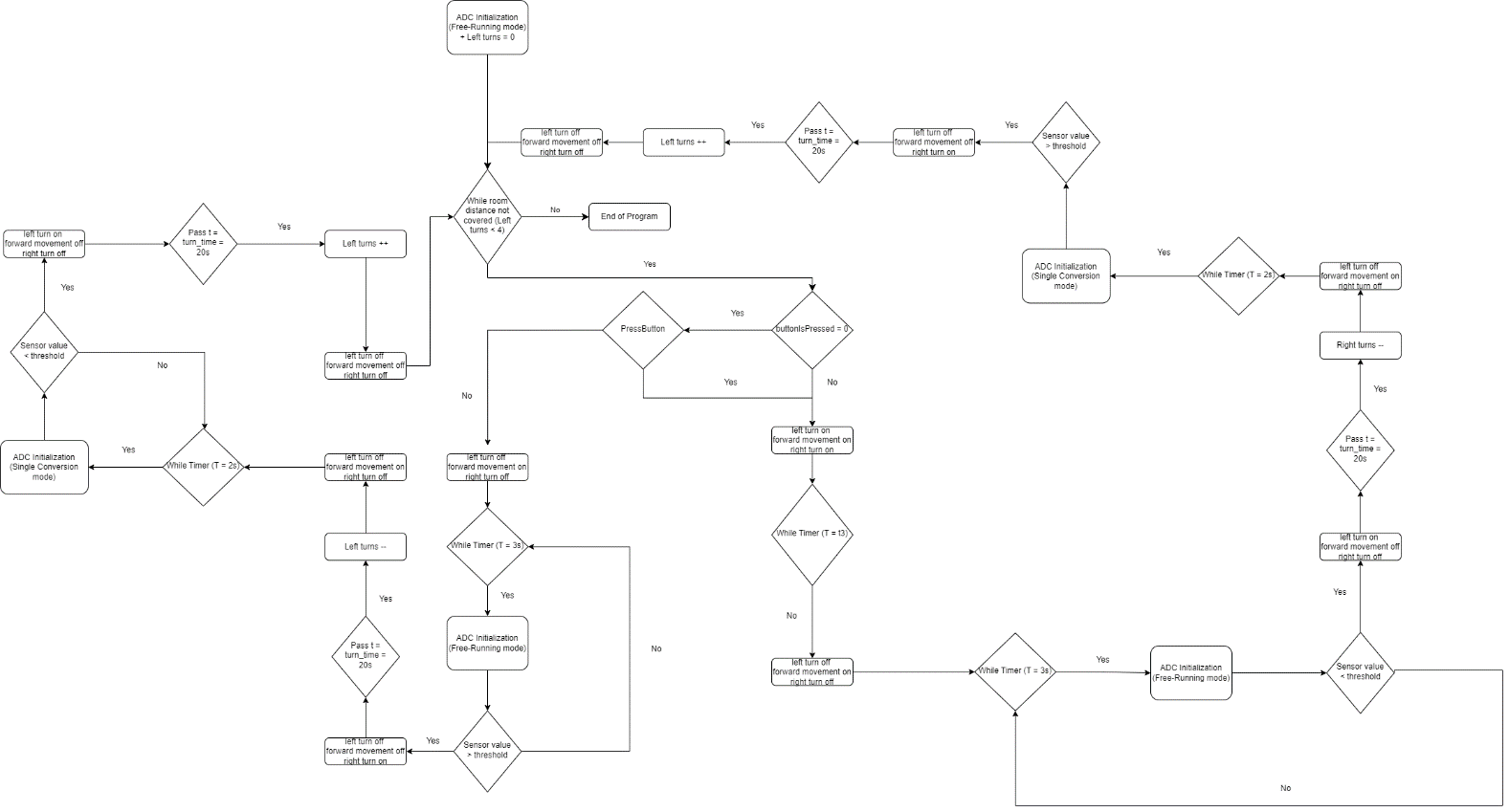
Ακολουθεί το διάγραμμα ροής :



**Ερώτημα 2.γ – Επεξήγηση κώδικα και διάγραμμα ροής**

Σε ότι αφορά τον κώδικα αυτό, έχουμε αξιοποιήσει το μεγαλύτερο μέρος του κώδικα από το Ερώτημα 2.β και έχουμε κάνει ορισμένες προσθήκες. Ειδικότερα, υποθέτουμε ότι το σύστημα μπορεί να κάνει αναστροφή 180 μοιρών μονάχα όταν κινείται ευθεία, και όχι κατά τη διάρκεια κάποια αριστερής ή δεξιάς στροφής. Έχουμε ορίσει μία καινούρια μεταβλητή *rightturns*, η οποία μετράει τις δεξιές στροφές όταν έχει πατηθεί ο διακόπτης (switch) για να κάνει αναστροφή 180 μοιρών. Με την εντολή *while(leftturns<4 && rightturns<leftturns)* ελέγχουμε το σύστημα μας να τερματίσει στο σημείο εκκίνησης. Πιο συγκεκριμένα, το κομμάτι που προστέθηκε στη while loop σε σχέση με το Ερώτημα 2 είναι το *rightturns<leftturns,* τo οποίο μας χρησιμεύει όταν έχει πατηθεί ο διακόπτης και ουσιαστικά ελέγχει αν γίνουν τόσες δεξιές όσες και αριστερές πριν πατηθεί το κουμπί. Δηλαδή, σε περίπτωση που *rightturns = leftturns* το πρόγραμμα εξέρχεται από την while loop. Εν συνεχεία, ακόμα μια αλλαγή που παρατηρούμε σε σχέση με τον κώδικα του Ερωτήματος 2, είναι η εντολή *if(buttonIsPressed == 0)* η οποία ελέγχει κατά πόσο έχει, ή δεν έχει πατηθεί ο διακόπτης αναστροφής 180 μοιρών της συσκευής. Ακόμη, έχει προστεθεί η εντολή *PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc* χάρη στην οποία, από τη στιγμή που είναι ενεργοποιημένες οι διακοπές με την εντολή *sei()*, μπορεί να προκληθεί διακοπή και το πρόγραμμα να μεταφερθεί στην καινούρια ISR (*ISR(PORTF\_PORT\_vect)*). Σε αυτή την ISR, εάν πατηθεί ο διακόπτης, η μεταβλητή *buttonIsPressed* γίνεται 1 και έτσι μπορεί να περάσει στην ανάποδη πορεία. Επίσης, να σημειωθεί ότι έχουν γίνει οι απαραίτητες αλλαγές, ώστε στο Free-Running να ελέγχεται ο τοίχος αριστερά της συσκευής και στρίβει κατάλληλα (αριστερά). Τέλος, έχει προστεθεί η μεταβλητή *t3,* στην οποία εκχωρείται ο χρόνος στροφής 180 μοιρών.

Ακολουθεί το διάγραμμα ροής :



**Μετατροπή Χρόνων για την Εξομοίωση του Κώδικα**

Σε ότι έχει να κάνει με τους χρόνους και τις μετατροπές, χρησιμοποιούμε τους τύπους ftimer  = και value = T\*ftimer και μπορούμε να υπολογίσουμε την τιμή του value που θα θέσουμε στο πρόγραμμα γνωρίζοντας το χρόνο Τ. Αρχικά, χρησιμοποιούμε τον τύπο ftimer  = . Θέτουμε στο fsystem την τιμή 20ΜΗz και στον Nprescaler την τιμή 1024. Έτσι από τον άνωθεν τύπο έχουμε ftimer  = 19.531,25 Hz. Εν συνεχεία, αξιοποιούμε τον δεύτερο τύπο. Ειδικότερα, θέτουμε για ftimer την τιμή που υπολογίσαμε προ λίγου. Για την άσκηση 2 του εργαστηρίου χρησιμοποιούμε δύο χρόνους. Έτσι :

1. Για Τ = 2 sec έχουμε value = 39.062,50
2. Για Τ = 3 sec έχουμε value = 58.593,75

Αξίζει να σημειωθεί, πως για την πιο γρήγορη εξομοίωση των προγραμμάτων μας, θέσαμε μικρότερες τιμές αντί των παραπάνω.