Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Πανεπιστήμιο Πατρών
Πολυτεχνική Σχολή

Τομέας Υλικού και Αρχιτεκτονικής των Υπολογιστών
Διδάσκουσα: Μαριλένα Δούναβη
Ακαδημαϊκό Έτος: 2023 – 2024
Ημ/νία Παράδοσης: 11/03/2024

Εργαστήριο Προηγμένων Μικροϋπολογιστών

2η Εργαστηριακή Άσκηση

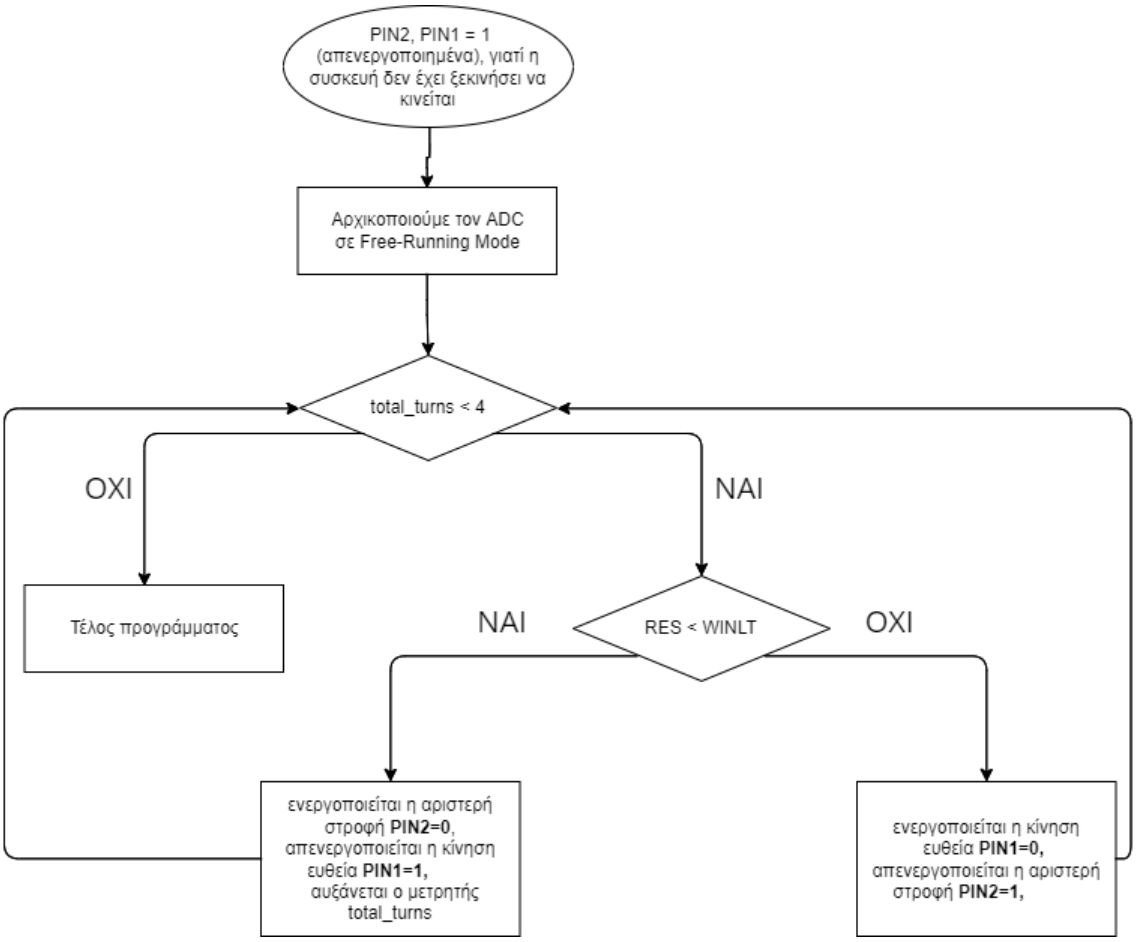
Έξυπνη Οικιακή Συσκευή, με Κίνηση στο Χώρο

Μάθημα Κορμού – CEID_NY463
Εαρινό Εξάμηνο 2024

Στοιχεία Φοιτητών (ΤΜΗΜΑ Α2):

Ονοματεπώνυμο:	Μηλτιάδης Μαντές	Χρυσαυγή Πατέλη
Α.Μ.:	1084661	1084513
E – mail:	up1084661@ac.upatras.gr	up1084513@ac.upatras.gr
Εξάμηνο:	Η’	Η’

2.1.0 Διάγραμμα Ροής



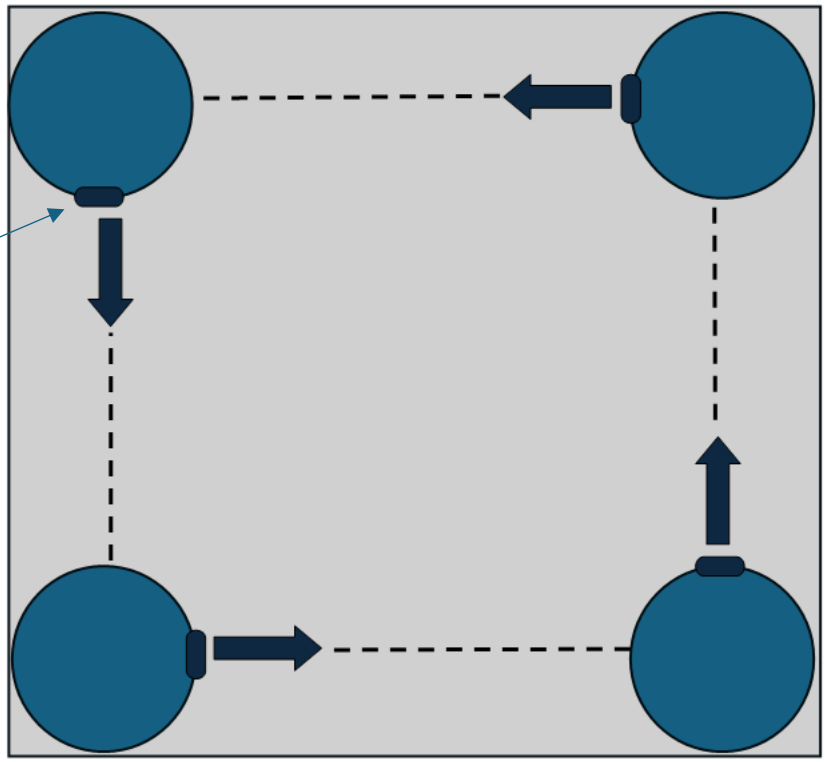
2.1.1 Σχόλια

Αρχικά, ορίζουμε την λογική σημαία **total_turns** (αρχικά είναι 0 γιατί η σκούπα δεν έχει κάνει καμία στροφή)**,** που είναι απαραίτητη για την προσομοίωση που θα εκτελέσουμε. Μετά, ορίζουμε την συνάρτηση **setup_FRM ( )** την οποία θα χρησιμοποιήσουμε για να κατασκευάσουμε τον μετατροπέα αναλογικού σε ψηφιακό σήμα.

Στην συνάρτηση αρχικά ορίζουμε πόσα bit θέλουμε να είναι ο μετατροπέας ADC και τον ενεργοποιούμε. Επειδή θέλουμε να λειτουργεί σε Free Running Mode ενεργοποιούμε και αυτή την λειτουργία θέτοντας '1' στο Free-Running bit (FREERUN) στο ADCn.CTRLA. Ορίζουμε σε ποιο bit θα συνδεθεί ο ADC και ενεργοποιούμε το debug mode. Στην συνέχεια, εισάγουμε ως κατώφλι στον καταχωρητή ADC.WINLT την τιμή 10 και ενεργοποιούμε τις διακοπές στην περίπτωση που το RESULT<10.

Στη συνάρτηση **main( )** αρχικά ορίζουμε με την εντολή **PORT.DIRSET** τα bit που θα χρησιμοποιήσουμε ως έξοδο, δηλαδή τα PIN1 και PIN2. Στη συνέχεια, με την εντολή **PORT.OUT** απενεργοποιούμε τα LEDs στα PIN1 και PIN2, έτσι ώστε να έχουμε ως αρχική συνθήκη ότι όλα τα pins είναι απενεργοποιημένα, καθώς η συσκευή δεν έχει ξεκινήσει να κινείται. Ακολούθως, αρχικοποιούμε τον ADC σε free running mode καλώντας την συνάρτηση setup_FRM. Στην συνέχεια ελέγχουμε πόσες στροφές έχουν πραγματοποιηθεί, αν το σύνολό τους είναι μικρότερο του 4 τότε το πρόγραμμα εισέρχεται στον βρόγχο while. Με την εντολή **sei()** ξεκινάμε να δεχόμαστε διακοπές και επειδή η κύρια συνθήκη είναι ότι η σκούπα εκτελεί ευθεία κίνηση ενεργοποιούμε το PIN1. Στην περίπτωση που το σύνολο των στροφών είναι 4 ή μεγαλύτερο τότε το πρόγραμμα δεν εισέρχεται στον βρόγχο while, απενεργοποιεί τις διακοπές με την εντολή **cli()** και το πρόγραμμα τερματίζει.

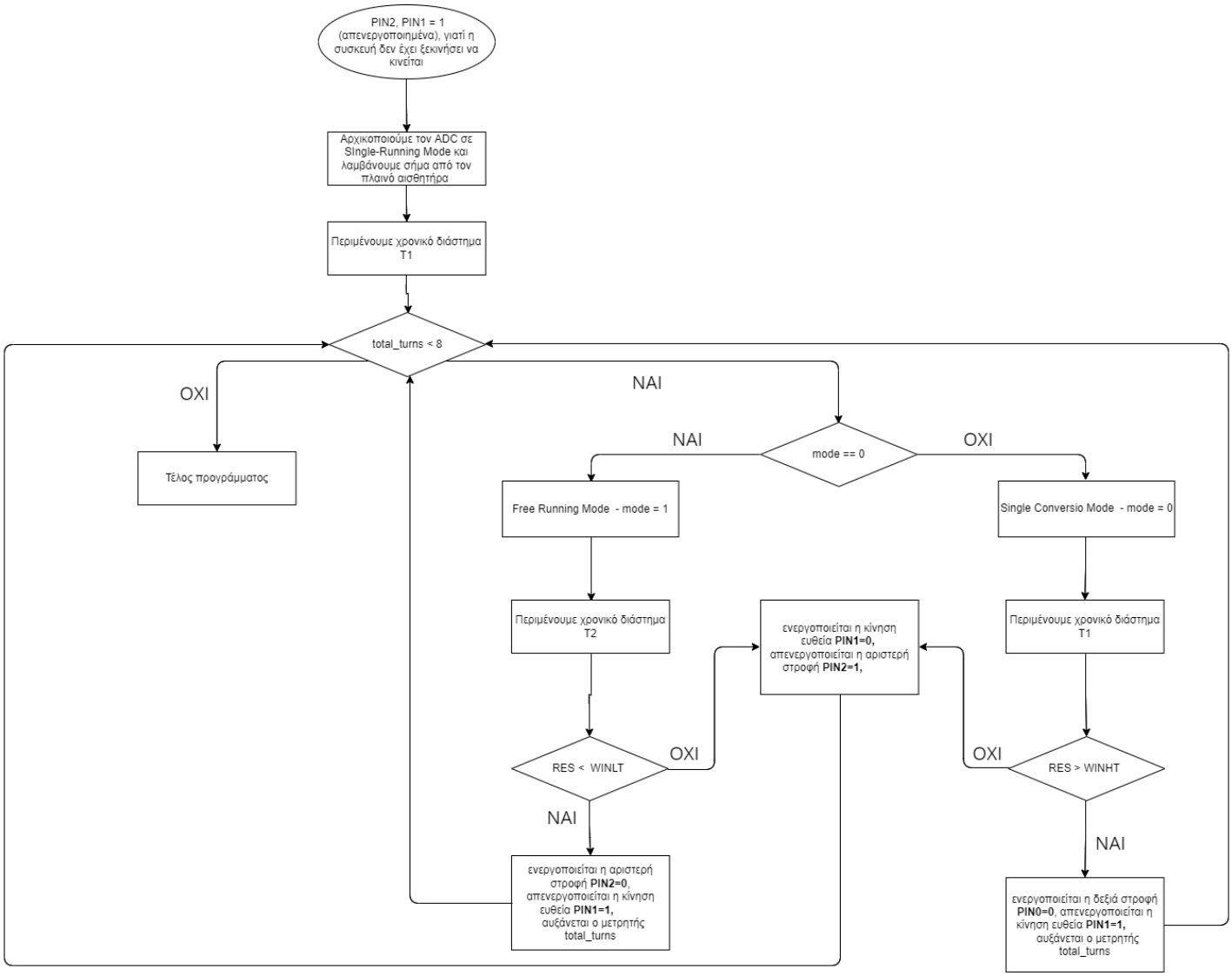
Η **ISR(ADC0_WCOMP_vect)** υλοποιεί τη διακοπή που προκύπτει όταν ο καταχωρητής RESULT έχει μικρότερη τιμή από τον καταχωρητή WINLT, δηλαδή όταν ο μπροστινός αισθητήρας της σκούπας εντοπίσει τοίχο. Τότε θέλουμε η σκούπα να σταματήσει να κινείται ευθεία, άρα απενεργοποιούμε το LED1=1 και να στρίψει αριστερά, άρα ενεργοποιούμε το LED2=0. Εφόσον η σκούπα έστριψε αυξάνουμε και τον μετρητή total_turns κατά 1.



2.1.2 Κώδικας

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

int total_turns = 0; // Μετρητής που αποθηκεύει το πλήθος των αριστερών στροφών που έχει πραγματοποιήσει η συσκευή

void setup_FRM(void){
	ADC0.CTRLA |= ADC_RESSEL_10BIT_gc; // 10-bit resolution
	ADC0.CTRLA |= ADC_ENABLE_bm; // Ενεργοποίηση του ADC
	ADC0.CTRLA |= ADC_FREERUN_bm; // Ενεργοποιούμε το Free-Running Mode του ADC
	ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc; // Ορίζεται το bit με το οποίο θα συνδεθεί ο ADC
	ADC0.DBGCTRL |= ADC_DBGRUN_bm;    // Ενεργοποίηση Debug Mode

	// Ορίζουμε το Window-Comparator Mode
	ADC0.WINLT |= 10; // Θέτουμε το κατώφλι
	ADC0.INTCTRL |= ADC_WCMP_bm; // Ενεργοποιούμε ις διακοπές για το Window-Comparator Mode
	ADC0.CTRLE |= ADC_WINCM0_bm; // Πραγματοποίηση διακοπης όταν RES < WINLT
}

int main(void)
{
	PORTD.DIRSET = PIN1_bm | PIN2_bm; // Ρυθμίζουμε τα PIN1 και PIN2 ως output
	PORTD.OUT |= PIN2_bm | PIN1_bm; // Αρχικά όλα τα pins είναι απενεργοποιημένα γιατί η συσκευή δεν έχει ξεκινήσει να κινείται
	setup_FRM(); // Αρχικοποιούμε τον ADC σε Free-Running Mode

	while (total_turns < 4) // Κάνουμε σάρωση μέχρι να ολοκληρώσουμε τέσσερις αριστερές στροφές
	{
		sei(); // Αρχίζουμε να δεχόμαστε σήματα διακοπής
		PORTD.OUTCLR = PIN1_bm; // Το PIN1 ανάβει για να σηματοδοτήσει τη μπροστά κίνηση
		PORTD.OUT |= PIN2_bm; // Το PIN2 σβήνει για να εμποδίσει την αριστερή κίνηση
		ADC0.COMMAND |= ADC_STCONV_bm; // Αρχίζει η μετατροπή
	}
	cli(); // Σταματάμε να δεχόμαστε σήματα διακοπής
}

// Διακοπή για χειρισμό του ADC και αριστερή στροφή της συσκευής
ISR(ADC0_WCOMP_vect){
	// Καθαρισμός σημαίας διακοπής
	int intflags = ADC0.INTFLAGS;
	ADC0.INTFLAGS = intflags;

	PORTD.OUT |= PIN1_bm; // Σβήνουμε το PIN1 για να σταματήσει η μπροστά κίνηση
	PORTD.OUTCLR = PIN2_bm; // Ανάβουμε το PIN2 για να σηματοδοτήσουμε την αριστερή κίνηση-στροφή
	total_turns++; // Αυξάνουμε το πλήθος των αριστερών στροφών κατά 1
}
```

2.2.0 Διάγραμμα Ροής



2.2.1 Σχόλια

Αρχικά, ορίζουμε τις χρονικές σταθερές **T1, T2** και τις λογικές σημαίες **total_turns** (αρχικά είναι 0 γιατί η σκούπα δεν έχει κάνει καμία στροφή) **, mode** (το αρχικοποιούμε σε 0 για να δείξουμε ότι ξεκινάει ως SCM), που είναι απαραίτητες για την προσομοίωση που θα εκτελέσουμε. Μετά, ορίζουμε την συνάρτηση **setup_FRM ( )** την οποία θα χρησιμοποιήσουμε για να κατασκευάσουμε τον μετατροπέα αναλογικού σε ψηφιακό σήμα όταν βρίσκεται σε Free Running Mode, και την συνάρτηση **setup_SCM( )** την οποία θα χρησιμοποιήσουμε για να κατασκευάσουμε τον μετατροπέα και να προσομοιώσουμε την λειτουργία όταν θα βρίσκεται σε Single Conversion Mode. Τέλος, ορίζουμε την συνάρτηση **setup_TCA0( ),** η οποία θα χρησιμοποιηθεί για να κατασκευάσουμε τους μετρητές.

Στην συνάρτηση **setup_FRM ( )** αρχικά ορίζουμε πόσα bit θέλουμε να είναι ο μετατροπέας ADC και τον ενεργοποιούμε. Επειδή θέλουμε να λειτουργεί σε Free Running Mode ενεργοποιούμε και αυτή την λειτουργία θέτοντας '1' στο Free-Running bit (FREERUN) στο ADCn.CTRLA. Ορίζουμε σε ποιο bit θα συνδεθεί ο ADC και ενεργοποιούμε το debug mode. Στην συνέχεια, εισάγουμε ως κατώφλι στον καταχωρητή ADC.WINLT την τιμή 10 και ενεργοποιούμε τις διακοπές στην περίπτωση που το RESULT<10. Η συγκεκριμένη συνάρτηση προσομοιώνει το σήμα που λαμβάνει ο μπροστινός αισθητήρας της σκούπας και αν εντοπίσει τοίχο σημαίνει ότι ο καταχωρητής RES έχει ικρότερη τιμή από το κατώφλι οπότε θα εισάγεται στην διακοπή.

Στην συνάρτηση **setup_SCM( )** αρχικά ορίζουμε πόσα bit θέλουμε να είναι ο μετατροπέας ADC και τον ενεργοποιούμε. Ορίζουμε σε ποιο bit θα συνδεθεί ο ADC και ενεργοποιούμε το debug mode. Στην συνέχεια, εισάγουμε ως κατώφλι στον καταχωρητή ADC.WINHT την τιμή 10 και ενεργοποιούμε τις διακοπές στην περίπτωση που το RESULT>10. Η συγκεκριμένη συνάρτηση προσομοιώνει το σήμα που λαμβάνει ο πλαϊνός (δεξιός) αισθητήρας της σκούπας και δεν έχει τοίχο στα δεξιά της σημαίνει ότι ο καταχωρητής RES έχει μεγαλύτερη τιμή από το κατώφλι οπότε θα εισάγεται στην διακοπή.

Στην συνάρτηση **setup_TCA0(**) αρχικά θέτουμε τον χρονιστή σε κανονική λειτουργία και θέτουμε την τιμή του ίση με το μηδέν. Στην συνέχεια, ορίζουμε την τιμή στην οποία όταν φτάσει ο χρονιστής θα προκληθεί διακοπή, η τιμή αυτή θα δίνεται ως όρισμα στη συνάρτηση για να μπορούμε να την αλλάζουμε κάθε φορά. Επίσης, ενεργοποιούμε τον χρονιστή μέσω του καταχωρητή CTRLA. Τέλος, ενεργοποιούμε τις διακοπές μέσω του καταχωρητή INTCTRL.

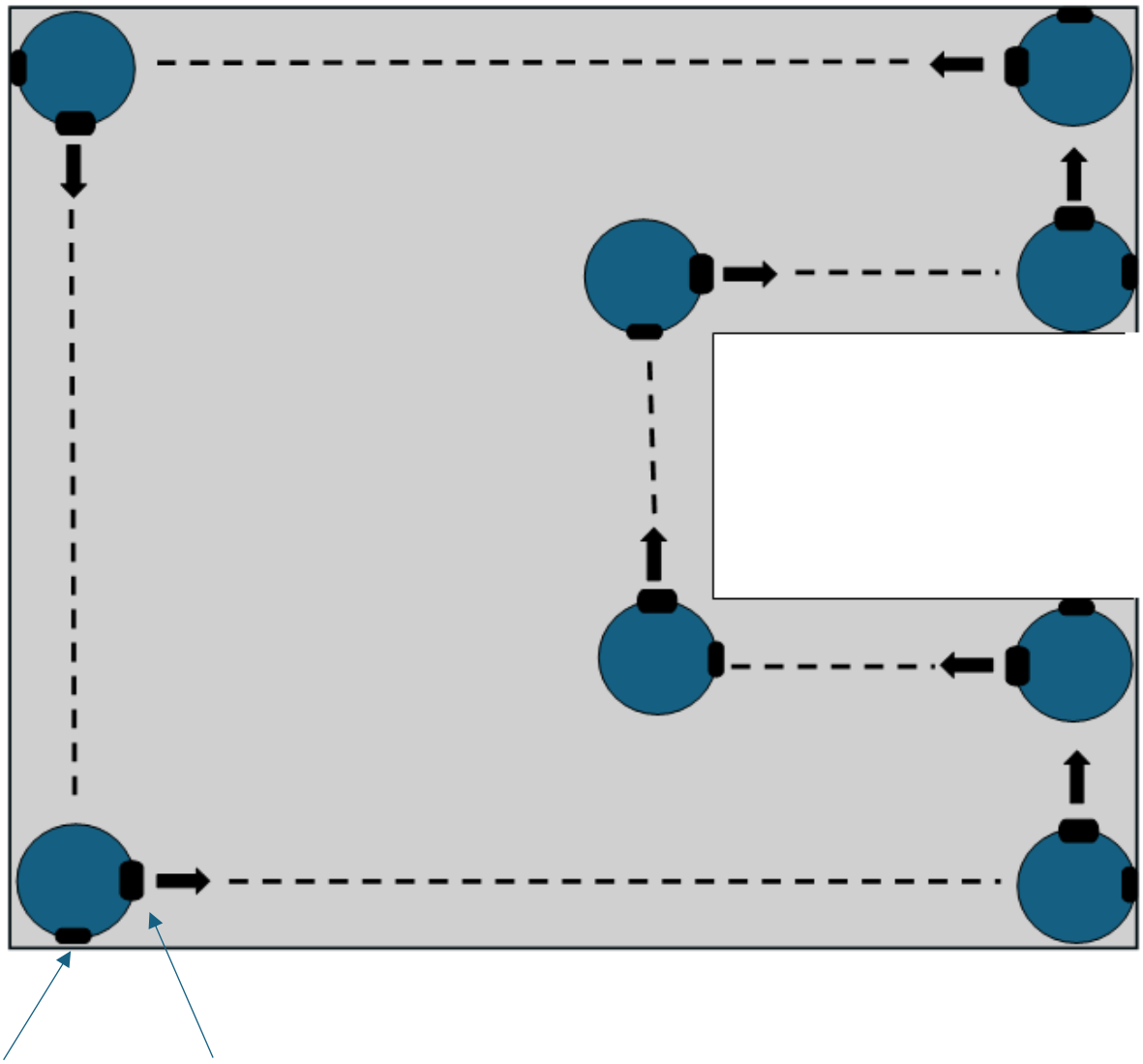
Στη συνάρτηση **main( )** αρχικά ορίζουμε με την εντολή **PORT.DIRSET** τα bit που θα χρησιμοποιήσουμε ως έξοδο, δηλαδή τα PIN1, PIN2 και PIN0. Στη συνέχεια, με την εντολή **PORT.OUT** απενεργοποιούμε τα LEDs στα PIN1, PIN2 και PIN0, έτσι ώστε να έχουμε ως αρχική συνθήκη ότι όλα τα pins είναι απενεργοποιημένα, καθώς η συσκευή δεν έχει ξεκινήσει να κινείται. Ακολούθως, αρχικοποιούμε τον ADC σε single conversion mode καλώντας την συνάρτηση setup_SCM, καθώς αρχικά λαμβάνει σήμα από τον πλαϊνό αισθητήρα(mode =0) για χρονικό διάστημα T1=1sec γι'αυτό καλούμε την συνάρτηση setup_TCA0(T1). Στην συνέχεια ελέγχουμε πόσες στροφές έχουν πραγματοποιηθεί, αν το σύνολό τους είναι μικρότερο του 8 τότε το πρόγραμμα εισέρχεται στον βρόγχο while. Με την εντολή **sei()** ξεκινάμε να δεχόμαστε διακοπές και επειδή η κύρια συνθήκη είναι ότι η σκούπα εκτελεί ευθεία κίνηση ενεργοποιούμε το PIN1 και απενεργοποιούμε τα PIN0, PIN2. Στην περίπτωση που το σύνολο των στροφών είναι 8 ή μεγαλύτερο τότε το πρόγραμμα δεν εισέρχεται στον βρόγχο while, απενεργοποιεί τις διακοπές με την εντολή **cli()** και το πρόγραμμα τερματίζει.

Η **ISR(ADC0_WCOMP_vect)** διακοπή ενεργοποιείται σε δύο περιπτώσεις:

- Υλοποιεί τη διακοπή που προκύπτει όταν ο καταχωρητής RESULT έχει μεγαλύτερη τιμή από τον καταχωρητή WINHT, δηλαδή όταν ο πλαϊνός αισθητήρας της σκούπας δεν εντοπίσει τοίχο. Τότε θέλουμε η σκούπα να σταματήσει να κινείται ευθεία, άρα απενεργοποιούμε το LED1=1 και να στρίψει δεξιά, άρα ενεργοποιούμε το LED0=0. Εφόσον η σκούπα έστριψε αυξάνουμε και τον μετρητή total_turns κατά 1. Η συγκεκριμένη περίπτωση προσομοιώνει την λειτουργία όταν η σκούπα κινείται σε single conversion mode, δηλαδή δέχεται σήμα από τον πλαϊνό αισθητήρα άρα όταν mode==0.
- Υλοποιεί τη διακοπή που προκύπτει όταν ο καταχωρητής RESULT έχει μικρότερη τιμή από τον καταχωρητή WINLT, δηλαδή όταν ο μπροστινός αισθητήρας της σκούπας εντοπίσει τοίχο. Τότε θέλουμε η σκούπα να σταματήσει να κινείται ευθεία, άρα απενεργοποιούμε το LED1=1 και να στρίψει αριστερά, άρα ενεργοποιούμε το LED2=0. Εφόσον η σκούπα έστριψε αυξάνουμε και τον μετρητή total_turns κατά 1. Η συγκεκριμένη περίπτωση προσομοιώνει την λειτουργία όταν η σκούπα κινείται σε free running mode, , δηλαδή δέχεται σήμα από τον μπροστινό αισθητήρα άρα όταν mode==1.

Η **ISR(TCA0_CMP0_vect)** υλοποιεί τη διακοπή που προκύπτει όταν υπερχειλίσει ο χρονιστής TCA0, δηλαδή όταν φτάσουμε στη χρονική στιγμή t = T1 ή t=T2. Η συγκεκριμένη διακοπή χρησιμοποιείται για να πραγματοποιεί τις εναλλαγές από SCM σε FRM και το αντίθετο ανά τακτά χρονικά διαστήματα μέχρι η σκούπα να φτάσει στην αρχική της θέση:

- Όταν η σκούπα βρίσκεται σε SCM, δηλαδή δέχεται σήμα από τον πλαϊνό αισθητήρα (mode==0) τότε εναλλάσσουμε το mode σε 1 και τον ADC από SCM σε FRM και περιμένουμε χρονικό διάστημα T2.
- Ενώ όταν η σκούπα βρίσκεται σε FRM, δηλαδή δέχεται σήμα από τον μπροστινό αισθητήρα (mode==1) τότε εναλλάσσουμε το mode σε 0 και τον ADC από FRM σε SCM και περιμένουμε χρονικό διάστημα T1.

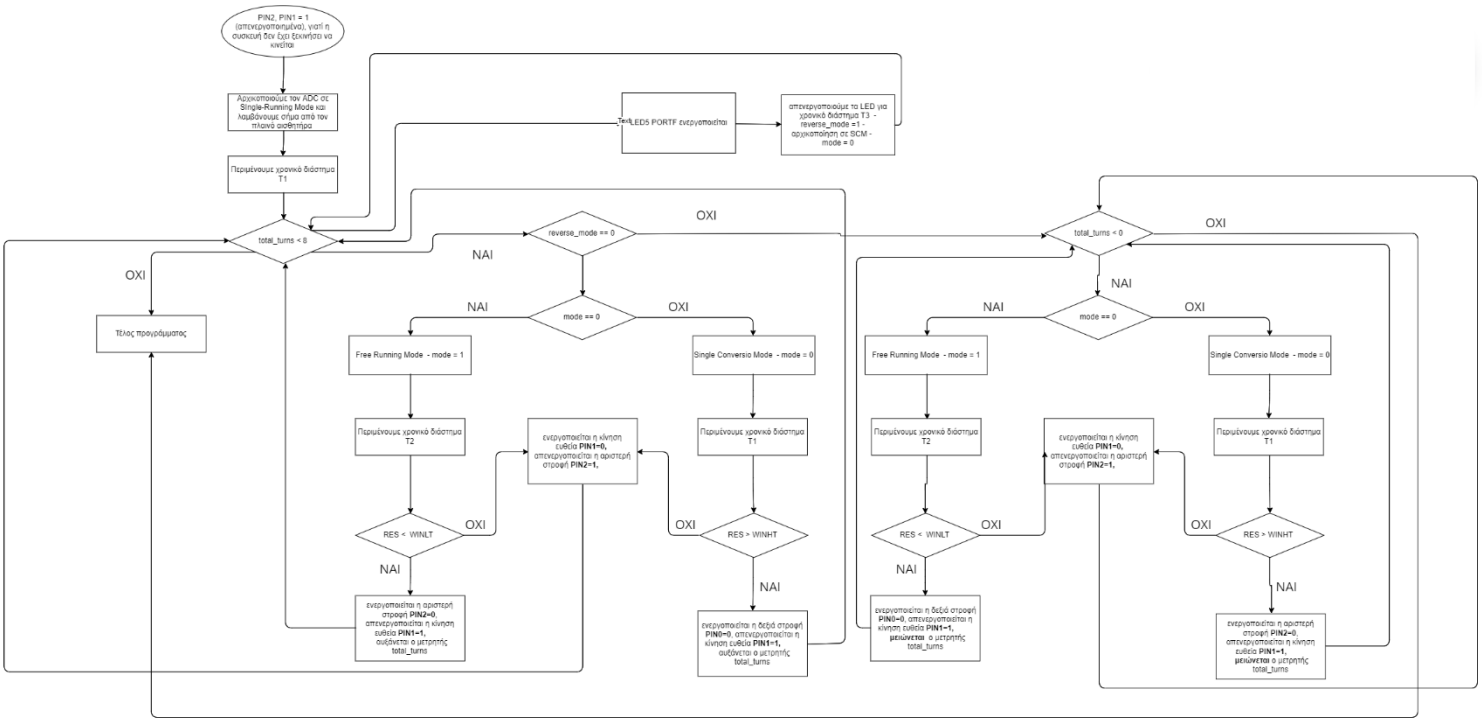


2.2.2 Κώδικας

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

#define T1 1 // T1 = 1 msec
#define T2 2 // T2 = 2 msec
int total_turns = 0; // Μετρητής που αποθηκεύει το πλήθος των αριστερών στροφών που έχει
πραγματοποιήσει η συσκευή
int mode = 0; // Αρχικά βρισκόμαστε σε Single-Conversion Mode

void setup_TCA0(int t){
	TCA0.SINGLE.CNT = 0;
	TCA0.SINGLE.CTRLB = 0;
	TCA0.SINGLE.CMP0 = t;
	TCA0.SINGLE.CTRLA = 0x7<<1;
	TCA0.SINGLE.CTRLA |=1;
	TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm;
}

void setup_SCM(void){
	ADC0.CTRLA |= ADC_RESSEL_10BIT_gc; // 10-bit resolution
	ADC0.CTRLA |= ADC_ENABLE_bm; // Ενεργοποίηση του ADC
	ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc; // Ορίζεται το bit με το οποίο θα συνδεθεί ο ADC
	ADC0.DBGCTRL |= ADC_DBGRUN_bm;    // Ενεργοποίηση Debug Mode

	//Window Comparator Mode
	ADC0.WINHT |= 10; //Set threshold
	ADC0.INTCTRL |= ADC_WCMP_bm; //Enable Interrupts for WCM
	ADC0.CTRLE = 0b00000010; //Interrupt when RESULT > WINHT
	ADC0.COMMAND |= ADC_STCONV_bm; // Αρχίζει η μετατροπή
}

void setup_FRM(void){
	ADC0.CTRLA |= ADC_RESSEL_10BIT_gc; // 10-bit resolution
	ADC0.CTRLA |= ADC_ENABLE_bm; // Ενεργοποίηση του ADC
	ADC0.CTRLA |= ADC_FREERUN_bm; // Ενεργοποιούμε το Free-Running Mode του ADC
	ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc; // Ορίζεται το bit με το οποίο θα συνδεθεί ο ADC
	ADC0.DBGCTRL |= ADC_DBGRUN_bm;    // Ενεργοποίηση Debug Mode

	//Window Comparator Mode
	ADC0.WINLT |= 10; //Set threshold
	ADC0.INTCTRL |= ADC_WCMP_bm; //Enable Interrupts for WCM
	ADC0.CTRLE = 0b00000001; //Interrupt when RESULT < WINLT
	ADC0.COMMAND |= ADC_STCONV_bm; // Αρχίζει η μετατροπή
}

int main(void)
{
	PORTD.DIRSET =  PIN1_bm | PIN2_bm | PIN0_bm; // Ρυθμίζουμε τα pins 0, 1, 2 ως output
	PORTD.OUT |= PIN2_bm | PIN1_bm | PIN0_bm;  // Αρχικά όλα τα pins είναι απενεργοποιημένα γιατί
η συσκευή δεν έχει ξεκινήσει να κινείται


	setup_SCM(); // Ξεκινάμε λαμβάνοντας σήμα από τον πλάγιο αισθητήρα
	setup_TCA0(T1); // Περιμένουμε χρονικό διάστημα T1

	while (total_turns < 8) // Κάνουμε σάρωση μέχρι να ολοκληρώσουμε 8 στροφές
	{

		sei(); // Αρχίζουμε να δεχόμαστε σήματα διακοπής
		PORTD.OUTCLR |= PIN1_bm; // Το pin 1 της μπροστά κίνησης ενεργοποιείται για να
προχωρήσει η συσκευή ευθεία
		PORTD.OUT |= PIN2_bm |PIN0_bm; // Το pin 2 της αριστερής στροφής και το pin 1 της
δεξιάς στροφής  απενεργοποιείται
	}
```

```
	cli();
}

// Διακοπή για χειρισμό του ADC όταν ξεπεράσουμε το κατώφλι
ISR(ADC0_WCOMP_vect){
	// Καθαρισμός σημαίας διακοπής
	int intflags = ADC0.INTFLAGS;
	ADC0.INTFLAGS = intflags;
	if (mode==0){
		PORTD.OUTCLR |= PIN0_bm; // Το pin 0  ενεργοποιείται για να στρίψει η συσκευή δεξιά
		PORTD.OUT |= PIN1_bm |PIN2_bm; // Το pin 1 της μπροστά κίνησης απενεργοποιείται
		total_turns++; // Αυξάνουμε το πλήθος των στροφών που έχουμε πραγματοποιήσει κατά 1

	}
	else{
		PORTD.OUTCLR |= PIN2_bm; // Το pin 2  ενεργοποιείται για να στρίψει η συσκευή αριστερά
		PORTD.OUT |= PIN1_bm|PIN0_bm; // Το pin 1 της μπροστά κίνησης απενεργοποιείται
		total_turns++; // Αυξάνουμε το πλήθος των στροφών που έχουμε πραγματοποιήσει κατά 1
	}

}

// Διακοπή υπερχείλισης για τον χρονομετρητή TCA0
ISR(TCA0_CMP0_vect){
	// Καθαρισμός σημαίας διακοπής
	int intflags = TCA0.SINGLE.INTFLAGS;
	TCA0.SINGLE.INTFLAGS=intflags;

	if (mode == 0)
	{
		// Εναλλάσσουμε τον ADC από Single-Conversion σε Free-Running Mode
		mode = 1;
		setup_FRM();
		setup_TCA0(T2); // Περιμένουμε χρονικό διάστημα T2

	}
	else
	{
		// Εναλλάσσουμε τον ADC από Free-Running σε Single-Conversion Mode
		mode = 0;
		setup_SCM();
		setup_TCA0(T1); // Περιμένουμε χρονικό διάστημα T2

	}
}
```

2.3.0 Διάγραμμα Ροής



2.3.1 Σχόλια

Αρχικά, ορίζουμε τις χρονικές σταθερές **T1, T2,T3** και τις λογικές σημαίες **total_turns** (αρχικά είναι 0 γιατί η σκούπα δεν έχει κάνει καμία στροφή) **, mode** (το αρχικοποιούμε σε 0 για να δείξουμε ότι ξεκινάει ως SCM)**, reverse_mode** (το αρχικοποιούμε σε 0 γιατί δεν έχει πατηθεί ακόμα ο διακόπτης για την αντίστροφη πορεία) που είναι απαραίτητες για την προσομοίωση που θα εκτελέσουμε. Μετά, ορίζουμε την συνάρτηση **setup_FRM ( )** την οποία θα χρησιμοποιήσουμε για να κατασκευάσουμε τον μετατροπέα αναλογικού σε ψηφιακό σήμα όταν βρίσκεται σε Free Running Mode, και την συνάρτηση **setup_SCM( )** την οποία θα χρησιμοποιήσουμε για να κατασκευάσουμε τον μετατροπέα και να προσομοιώσουμε την λειτουργία όταν θα βρίσκεται σε Single Conversion Mode. Τέλος, ορίζουμε την συνάρτηση **setup_TCA0( ),** η οποία θα χρησιμοποιηθεί για να κατασκευάσουμε τους μετρητές.

Στην συνάρτηση **setup_FRM ( )** αρχικά ορίζουμε πόσα bit θέλουμε να είναι ο μετατροπέας ADC και τον ενεργοποιούμε. Επειδή θέλουμε να λειτουργεί σε Free Running Mode ενεργοποιούμε και αυτή την λειτουργία θέτοντας '1' στο Free-Running bit (FREERUN) στο ADCn.CTRLA. Ορίζουμε σε ποιο bit θα συνδεθεί ο ADC και ενεργοποιούμε το debug mode. Στην συνέχεια, εισάγουμε ως κατώφλι στον καταχωρητή ADC.WINLT την τιμή 10 και ενεργοποιούμε τις διακοπές στην περίπτωση που το RESULT<10. Η συγκεκριμένη συνάρτηση προσομοιώνει το σήμα που λαμβάνει ο μπροστινός αισθητήρας της σκούπας και αν εντοπίσει τοίχο σημαίνει ότι ο καταχωρητής RES έχει ικρότερη τιμή από το κατώφλι οπότε θα εισάγεται στην διακοπή.

Στην συνάρτηση **setup_SCM( )** αρχικά ορίζουμε πόσα bit θέλουμε να είναι ο μετατροπέας ADC και τον ενεργοποιούμε. Ορίζουμε σε ποιο bit θα συνδεθεί ο ADC και ενεργοποιούμε το debug mode. Στην συνέχεια, εισάγουμε ως κατώφλι στον καταχωρητή ADC.WINHT την τιμή 10 και ενεργοποιούμε τις διακοπές στην περίπτωση που το RESULT>10. Η συγκεκριμένη συνάρτηση προσομοιώνει το σήμα που λαμβάνει ο πλαϊνός (δεξιός) αισθητήρας της σκούπας και δεν έχει τοίχο στα δεξιά της σημαίνει ότι ο καταχωρητής RES έχει μεγαλύτερη τιμή από το κατώφλι οπότε θα εισάγεται στην διακοπή.

Στην συνάρτηση **setup_TCA0(**) αρχικά θέτουμε τον χρονιστή σε κανονική λειτουργία και θέτουμε την τιμή του ίση με το μηδέν. Στην συνέχεια, ορίζουμε την τιμή στην οποία όταν φτάσει ο χρονιστής θα προκληθεί διακοπή, η τιμή αυτή θα δίνεται ως όρισμα στη συνάρτηση για να μπορούμε να την αλλάζουμε κάθε φορά. Επίσης, ενεργοποιούμε τον χρονιστή μέσω του καταχωρητή CTRLA. Τέλος, ενεργοποιούμε τις διακοπές μέσω του καταχωρητή INTCTRL.

Στη συνάρτηση **main( )** αρχικά ορίζουμε με την εντολή **PORT.DIRSET** τα bit που θα χρησιμοποιήσουμε ως έξοδο, δηλαδή τα PIN1, PIN2 και PIN0 και με την εντολή **PORT.DIRCLR** τα bit που θα χρησιμοποιήσουμε ως είσοδο, δηλαδή το PIN5 του PORTF. Στη συνέχεια, με την εντολή **PORT.OUT** απενεργοποιούμε τα LEDs στα PIN1, PIN2 και PIN0, έτσι ώστε να έχουμε ως αρχική συνθήκη ότι όλα τα pins είναι απενεργοποιημένα, καθώς η συσκευή δεν έχει ξεκινήσει να κινείται και ενεργοποιούμε το pull-up resistor για το PIN5 του PORTF ώστε να ρυθμίζεται η συμπεριφορά του PIN5 για να προκαλεί διακοπή σε κάθε αλλαγή.. Ακολούθως, αρχικοποιούμε τον ADC σε single conversion mode καλώντας την συνάρτηση setup_SCM, καθώς αρχικά λαμβάνει σήμα από τον πλαϊνό αισθητήρα(mode =0) για χρονικό διάστημα T1=1sec γι'αυτό καλούμε την συνάρτηση setup_TCA0(T1). Στην συνέχεια ελέγχουμε πόσες στροφές έχουν πραγματοποιηθεί, αν το σύνολό τους είναι μικρότερο του 8 τότε το πρόγραμμα εισέρχεται στον βρόγχο while.

- Στην πρώτη περίπτωση που δε έχει πατηθεί ακόμα ο διακόπτης για να αλλάξει η σκούπα πορεία ξεκινάμε να δεχόμαστε διακοπές με εντολή **sei()** και επειδή η κύρια συνθήκη είναι ότι η σκούπα εκτελεί ευθεία κίνηση ενεργοποιούμε το PIN1 και απενεργοποιούμε τα PIN0, PIN2.
- Στην περίπτωση που πατηθεί ο διακόπτης (reverse_mode==1) η σκούπα θέλουμε να αλλάξει πορεία, να ακολουθήσει την πορεία που έχει κάνει και να γυρίσει στην αρχική της θέση, γι'αυτό προσθέτουμε έναν επιπλέον βρόγχο while που θα εκτελείται όσο μειώνεται το πλήθος των στροφών και όταν φτάσει στο 0 σημαίνει ότι η σκούπα βρίσκεται στην αρχική της θέση, το πρόγραμμα εξέρχεται από τον βρόγχο, σταματάει να δέχεται διακοπές και τερματίζει.
- Τέλος, στην περίπτωση που το σύνολο των στροφών είναι 8 ή μεγαλύτερο τότε το πρόγραμμα δεν εισέρχεται στον βρόγχο while, απενεργοποιεί τις διακοπές με την εντολή **cli()** και το πρόγραμμα τερματίζει.

Η **ISR(ADC0_WCOMP_vect)**:

1. Στην περίπτωση που δε έχει πατηθεί ακόμα ο διακόπτης για να αλλάξει η σκούπα πορεία (reverse_mode==0)

- υλοποιεί τη διακοπή που προκύπτει όταν ο καταχωρητής RESULT έχει μεγαλύτερη τιμή από τον καταχωρητή WINHT, δηλαδή όταν ο πλαϊνός αισθητήρας της σκούπας δεν εντοπίσει τοίχο. Τότε θέλουμε η σκούπα να σταματήσει να κινείται ευθεία, άρα απενεργοποιούμε το LED1=1 και να στρίψει δεξιά, άρα ενεργοποιούμε το LED0=0. Εφόσον η σκούπα έστριψε αυξάνουμε και τον μετρητή total_turns κατά 1. Η συγκεκριμένη περίπτωση προσομοιώνει την λειτουργία όταν η σκούπα κινείται σε single conversion mode, δηλαδή δέχεται σήμα από τον πλαϊνό αισθητήρα άρα όταν mode==0.
- υλοποιεί τη διακοπή που προκύπτει όταν ο καταχωρητής RESULT έχει μικρότερη τιμή από τον καταχωρητή WINLT, δηλαδή όταν ο μπροστινός αισθητήρας της σκούπας εντοπίσει τοίχο. Τότε θέλουμε η σκούπα να σταματήσει να κινείται ευθεία, άρα απενεργοποιούμε το LED1=1 και να στρίψει αριστερά, άρα ενεργοποιούμε το LED2=0. Εφόσον η σκούπα έστριψε αυξάνουμε και τον μετρητή total_turns κατά 1. Η συγκεκριμένη περίπτωση προσομοιώνει την λειτουργία όταν η σκούπα κινείται σε free running mode, , δηλαδή δέχεται σήμα από τον μπροστινό αισθητήρα άρα όταν mode==1.

2. Στην περίπτωση που έχει πατηθεί ο διακόπτης για να αλλάξει η σκούπα πορεία (reverse_mode==1) θέλουμε να μειώνονται οι στροφές μέχρι να μηδενιστούν άρα η σκούπα θα βρίσκεται στην αρχική της θέση.

- υλοποιεί τη διακοπή που προκύπτει όταν ο καταχωρητής RESULT έχει μεγαλύτερη τιμή από τον καταχωρητή WINHT, δηλαδή όταν ο πλαϊνός αισθητήρας της σκούπας δεν εντοπίσει τοίχο. Τότε θέλουμε η σκούπα να σταματήσει να κινείται ευθεία, άρα απενεργοποιούμε το LED1=1 και να στρίψει αριστερά, άρα ενεργοποιούμε το LED2=0. Εφόσον η σκούπα έστριψε μειώνουμε και τον μετρητή total_turns κατά 1. Η συγκεκριμένη περίπτωση προσομοιώνει την λειτουργία όταν η σκούπα κινείται σε single conversion mode, δηλαδή δέχεται σήμα από τον πλαϊνό αισθητήρα άρα όταν mode==0.
- υλοποιεί τη διακοπή που προκύπτει όταν ο καταχωρητής RESULT έχει μικρότερη τιμή από τον καταχωρητή WINLT, δηλαδή όταν ο μπροστινός αισθητήρας της σκούπας εντοπίσει τοίχο. Τότε θέλουμε η σκούπα να σταματήσει να κινείται ευθεία, άρα απενεργοποιούμε το LED1=1 και να στρίψει δεξιά, άρα ενεργοποιούμε το LED0=0. Εφόσον η σκούπα έστριψε αυξάνου μειώνουμε και τον μετρητή total_turns κατά 1. Η συγκεκριμένη περίπτωση προσομοιώνει την λειτουργία όταν η σκούπα κινείται σε free running mode, , δηλαδή δέχεται σήμα από τον μπροστινό αισθητήρα άρα όταν mode==1.

Η **ISR(TCA0_CMP0_vect)** υλοποιεί τη διακοπή που προκύπτει όταν υπερχειλίσει ο χρονιστής TCA0, δηλαδή όταν φτάσουμε στη χρονική στιγμή t = T1 ή t=T2. Η συγκεκριμένη διακοπή χρησιμοποιείται για να πραγματοποιεί τις εναλλαγές από SCM σε FRM και το αντίθετο ανά τακτά χρονικά διαστήματα μέχρι η σκούπα να φτάσει στην αρχική της θέση:

- Όταν η σκούπα βρίσκεται σε SCM, δηλαδή δέχεται σήμα από τον πλαϊνό αισθητήρα (mode==0) τότε εναλλάσσουμε το mode σε 1 και τον ADC από SCM σε FRM και περιμένουμε χρονικό διάστημα T2.
- Ενώ όταν η σκούπα βρίσκεται σε FRM, δηλαδή δέχεται σήμα από τον μπροστινό αισθητήρα (mode==1) τότε εναλλάσσουμε το mode σε 0 και τον ADC από FRM σε SCM και περιμένουμε χρονικό διάστημα T1.

Η **ISR(PORTF_PORT_vect)** υλοποιεί τη διακοπή που προκύπτει αν πατηθεί το κουμπί της ανάποδης πορείας. Τότε θέλουμε να απενεργοποιηθούν όλα τα LEDs (LED1, LED2, LED0) για χρονικό διάστημα T3 (setup_TCA0(T3)). Επίσης ενεργοποιούμε το flag που σηματοδοτεί την ανάποδη πορεία (reverse_mode = 1). Η ανάποδη λειτουργία ξεκινά λαμβάνοντας σήμα από τον πλαϊνό αισθητήρα mode=0 και καλούμε την συνάρτηση setup_SCM για χρονικό διάστημα T1 (setup_TCA0(T1)).

2.3.2 Κώδικας

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

#define T1 1 // T1 = 1 msec //SCM
#define T2 2 // T2 = 2 msec //FRM
#define T3 3 // T3 = 3 msec
int total_turns = 0; // Μετρητής που αποθηκεύει το πλήθος των αριστερών στροφών που έχει
πραγματοποιήσει η συσκευή
int mode = 0; // Αρχικά βρισκόμαστε σε Single-Conversion Mode
int reverse_mode = 0; // μεταβλητη που θα καθορίζει την αντίστροφη πορεία

void setup_TCA0(int t){
	TCA0.SINGLE.CNT = 0;
	TCA0.SINGLE.CTRLB = 0;
	TCA0.SINGLE.CMP0 = t;
	TCA0.SINGLE.CTRLA = 0x7<<1;
	TCA0.SINGLE.CTRLA |=1;
	TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm;
}

void setup_SCM(void){
	ADC0.CTRLA |= ADC_RESSEL_10BIT_gc; // 10-bit resolution
	ADC0.CTRLA |= ADC_ENABLE_bm; // Ενεργοποίηση του ADC
	ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc; // Ορίζεται το bit με το οποίο θα συνδεθεί ο ADC
	ADC0.DBGCTRL |= ADC_DBGRUN_bm;    // Ενεργοποίηση Debug Mode

	//Window Comparator Mode
	ADC0.WINHT |= 10; //Set threshold
	ADC0.INTCTRL |= ADC_WCMP_bm; //Enable Interrupts for WCM
	ADC0.CTRLE = 0b00000010; //Interrupt when RESULT > WINHT
	ADC0.COMMAND |= ADC_STCONV_bm; // Αρχίζει η μετατροπή
}

void setup_FRM(void){
	ADC0.CTRLA |= ADC_RESSEL_10BIT_gc; // 10-bit resolution
	ADC0.CTRLA |= ADC_ENABLE_bm; // Ενεργοποίηση του ADC
	ADC0.CTRLA |= ADC_FREERUN_bm; // Ενεργοποιούμε το Free-Running Mode του ADC
	ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc; // Ορίζεται το bit με το οποίο θα συνδεθεί ο ADC
	ADC0.DBGCTRL |= ADC_DBGRUN_bm;    // Ενεργοποίηση Debug Mode

	//Window Comparator Mode
	ADC0.WINLT |= 10; //Set threshold
	ADC0.INTCTRL |= ADC_WCMP_bm; //Enable Interrupts for WCM
	ADC0.CTRLE = 0b00000001; //Interrupt when RESULT < WINLT
	ADC0.COMMAND |= ADC_STCONV_bm; // Αρχίζει η μετατροπή
}

int main(void)
{
	PORTD.DIRSET =  PIN1_bm | PIN2_bm | PIN0_bm; // Ρυθμίζουμε τα pins 0, 1, 2 ως output
	PORTF.DIRCLR = PIN5_bm; // Ρυθμίζουμε το pin 5 ως input
	PORTD.OUT |= PIN2_bm | PIN1_bm | PIN0_bm;  // Αρχικά όλα τα pins είναι απενεργοποιημένα γιατί
η συσκευή δεν έχει ξεκινήσει να κινείται
	PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc; // Ενεργοποιούμε το pull-up
resistor

	setup_SCM(); // Ξεκινάμε λαμβάνοντας σήμα από τον πλάγιο αισθητήρα
	setup_TCA0(T1); // Περιμένουμε χρονικό διάστημα T1

	while (total_turns < 8) // Κάνουμε σάρωση μέχρι να ολοκληρώσουμε 8 στροφές
	{
		if (reverse_mode==0){

			sei(); // Αρχίζουμε να δεχόμαστε σήματα διακοπής
```

```
                        PORTD.OUTCLR |= PIN1_bm; // Το pin 1 της μπροστά κίνησης ενεργοποιείται για να
προχωρήσει η συσκευή ευθεία
                        PORTD.OUT |= PIN2_bm |PIN0_bm; // Το pin 2 της αριστερής κίνησης
απενεργοποιείται
                }
                else{
                        while(total_turns > 0){
                                sei(); // Αρχίζουμε να δεχόμαστε σήματα διακοπής
                                PORTD.OUTCLR |= PIN1_bm; // Το pin 1 της μπροστά κίνησης ενεργοποιείται
για να προχωρήσει η συσκευή ευθεία
                                PORTD.OUT |= PIN2_bm |PIN0_bm; // Το pin 2 της αριστερής κίνησης
απενεργοποιείται

                        }
                        cli();
                        break;
                }

        }
        cli();
}

// Διακοπή για χειρισμό του ADC όταν ξεπεράσουμε το κατώφλι
ISR(ADC0_WCOMP_vect){
        // Καθαρισμός σημαίας διακοπής
        int intflags = ADC0.INTFLAGS;
        ADC0.INTFLAGS = intflags;
        if (reverse_mode==0){
                if (mode==0){
                        PORTD.OUTCLR |= PIN0_bm; // Το pin 0  ενεργοποιείται για να στρίψει η συσκευή
δεξιά
                        PORTD.OUT |= PIN1_bm |PIN2_bm; // Το pin 1 της μπροστά κίνησης απενεργοποιείται
                        total_turns++; // Αυξάνουμε το πλήθος των στροφών που έχουμε πραγματοποιήσει
κατά 1

                }
                else{
                        PORTD.OUTCLR |= PIN2_bm; // Το pin 2  ενεργοποιείται για να στρίψει η συσκευή
αριστερά
                        PORTD.OUT |= PIN1_bm|PIN0_bm; // Το pin 1 της μπροστά κίνησης απενεργοποιείται
                        total_turns++; // Αυξάνουμε το πλήθος των στροφών που έχουμε πραγματοποιήσει
κατά 1
                }
        }
        else{
                if (mode==0){
                        PORTD.OUTCLR |= PIN2_bm; // Το pin 2 της αριστερής κίνησης ενεργοποιείται για
να στρίψει η συσκευή αριστερά
                        PORTD.OUT |= PIN1_bm|PIN0_bm; // Το pin 1 της μπροστά κίνησης απενεργοποιείται
                        total_turns--; // Αυξάνουμε το πλήθος των στροφών που έχουμε πραγματοποιήσει
κατά 1
                }
                else{
                        PORTD.OUTCLR |= PIN0_bm; // Το pin 0  ενεργοποιείται για να στρίψει η συσκευή
δεξιά
                        PORTD.OUT |= PIN1_bm |PIN2_bm;  // Το pin 1 της μπροστά κίνησης
απενεργοποιείται
                        total_turns--; // Αυξάνουμε το πλήθος των στροφών που έχουμε πραγματοποιήσει
κατά 1

                }

        }

}

// Διακοπή υπερχείλισης για τον χρονομετρητή TCA0
ISR(TCA0_CMP0_vect){
```

```
	// Καθαρισμός σημαίας διακοπής
	int intflags = TCA0.SINGLE.INTFLAGS;
	TCA0.SINGLE.INTFLAGS=intflags;

	if (mode == 0)
	{
		// Εναλλάσσουμε τον ADC από Single-Conversion σε Free-Running Mode
		mode = 1;
		setup_FRM();
		setup_TCA0(T2); // Περιμένουμε χρονικό διάστημα T2

	}
	else
	{
		// Εναλλάσσουμε τον ADC από Free-Running σε Single-Conversion Mode
		mode = 0;
		setup_SCM();
		setup_TCA0(T1); // Περιμένουμε χρονικό διάστημα T2

	}
}

// Διακοπή για το πάτημα του κουμπιού.
ISR(PORTF_PORT_vect) {
	int y = PORTF.INTFLAGS;
	PORTF.INTFLAGS=y;
	PORTD.OUTCLR |= PIN1_bm|PIN0_bm|PIN2_bm; //απενεργοποιούμε και τα 3 LEDs για χρονικό διάστημα
T3
	setup_TCA0(T3);
	reverse_mode = 1; //ενεργοποιούμε το flag της ανάποδης πορείας
	// Ξεκινάμε λαμβάνοντας σήμα από τον πλάγιο αισθητήρα
	mode=0;
	setup_SCM();
	setup_TCA0(T1); // Περιμένουμε χρονικό διάστημα T1

}
```