**Εργαστήριο Προηγμένοι Μικροεπεξεργαστές**

Εργαστηριακή Άσκηση 05:

Λειτουργία Έξυπνου Θερμοκηπίου

2024-25

**Περιεχόμενα**

[Σχεδιασμός / Διάγραμμα Ροής 3](#__RefHeading___Toc1188_1975538019)

[Περιγραφή Κώδικα – Σχεδιαστική ιδέα 3](#__RefHeading___Toc30_1975538019_Copy_1)

[main 4](#__RefHeading___Toc666_1975538019_Copy_1_)

[ISR(PORTF\_WCOMP\_vect) 5](#__RefHeading___Toc5418_867001118)

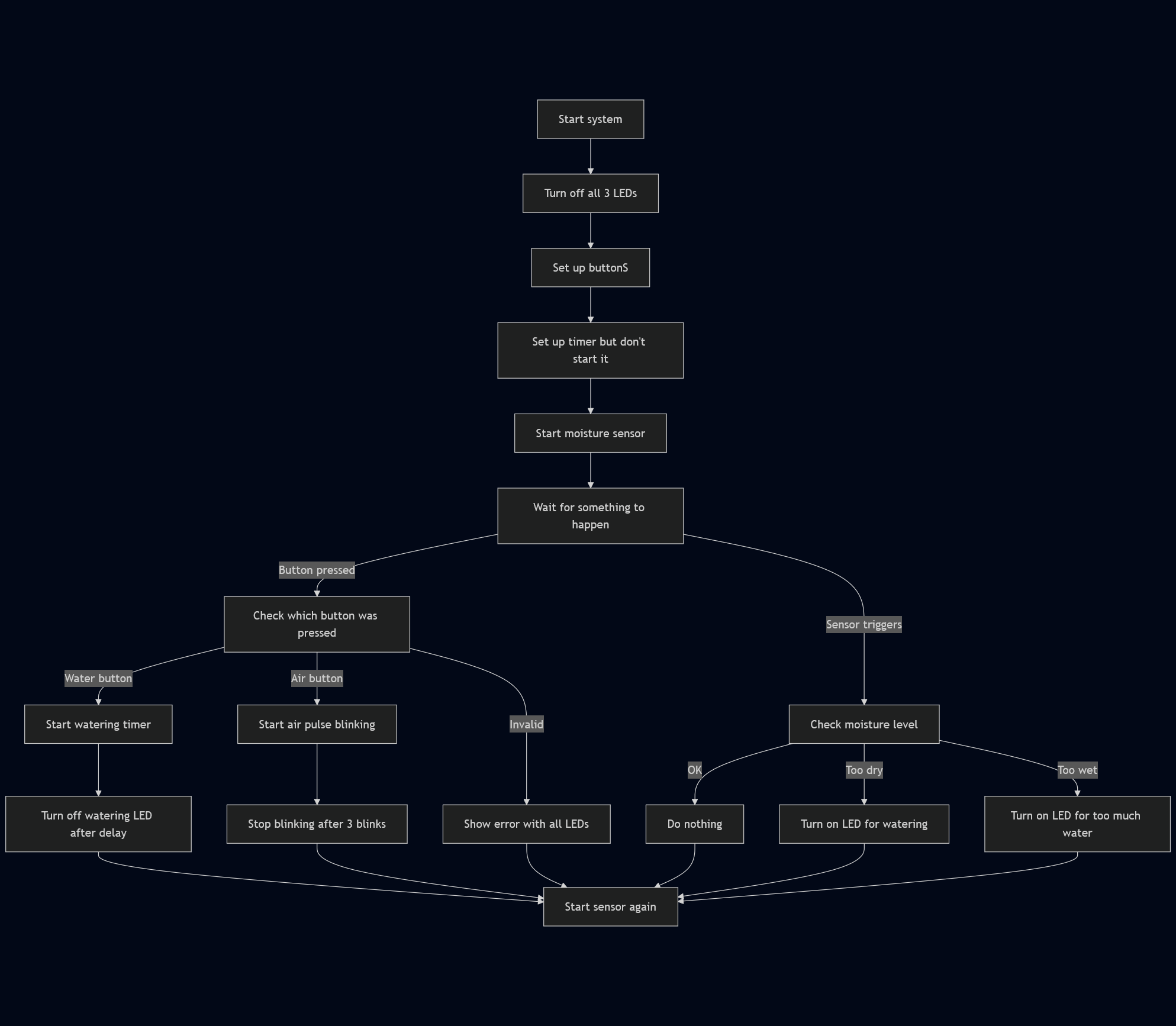
[ISR(PORTF\_PORT\_vect) 5](#__RefHeading___Toc3836_1387393539)

[ISR(TCA0\_CMP1\_vect) 7](#__RefHeading___Toc666_1975538019_Copy_12)

[ISR(TCA0\_OVF\_vect) 7](#__RefHeading___Toc666_1975538019_Copy_12)

[Kώδικας 8](#__RefHeading___Toc2477_2567516344_Copy_1)

# Σχεδιασμός / Διάγραμμα Ροής



# Περιγραφή Κώδικα – Σχεδιαστική ιδέα

Ολόκληρος ο κώδικας είναι διαθέσιμος στο τέλος της αναφοράς.

Καλούμαστε να υλοποιήσουμε την λειτουργία ενός ενός έξυπνου θερμοκηπίου με χρήση PWM, χρονιστή και ADC.

## main

Ορίζουμε τα pins εξόδου και τα αρχικοποιούμε κλειστά.

PORTD.DIR |= PIN1\_bm

| PIN0\_bm

| PIN2\_bm;

PORTD.OUT |= PIN1\_bm

| PIN0\_bm

| PIN2\_bm;

1. Αρχικοποιούμε το κουμπί και τον prescaler σε 1024 (δεν τον ενεργοποιούμε ακόμα για λόγος εξοικονόμησης ενέργειας).
2. PORTF.PIN6CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_RISING\_gc; //configure button 6
3. PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_RISING\_gc; //configure button 5
4. TCA0.SINGLE.CTRLA = TCA\_SINGLE\_CLKSEL\_DIV1024\_gc;
5. Καλούμε την συνάρτηση init\_ADC() η οποία αρχικοποιεί τον μετατροπέα σε free Running Mode με 10bit αναλυση και low / high thresholds. Η επιλογή των τιμών των thresholds έγινε τυχαία.

void init\_ADC() {

//initialize the ADC for Free-Running mode

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

ADC0.CTRLA |= ADC\_FREERUN\_bm; // Free-Running mode

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

ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc; // PD7

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

//Window Comparator Mode

ADC0.WINLT = 50; // Set low

ADC0.WINHT = 200; // Set high

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

ADC0.CTRLE |= ADC\_WINCM\_OUTSIDE\_gc; // Interrupt when

}

1. Ενεργοποιούμε τα interrupts, ξεκινάμε τον ADC, και το πρόγραμμα μπαίνει σε εναν ατέρμονο βρόγχο.

sei(); //enable interrupts

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

while(1){

;

}

## ISR(PORTF\_WCOMP\_vect)

1. Αυτή η ISR ενεργοποιείται όταν ο αισθητήρας ανιχνεύει τιμή μικρότερη ή μεγαλύτερη από το αντίστοιχο threshold. Διαβάζουμε την τιμή, καθαρίζουμε το flag και **απενεργοποιούμε** τον ADC, έτσι ώστε να μπορέσουμε να αντιμετωπίσουμε το πρόβλημα. Εάν δεν τον απενεργοποιούσαμε, το πρόγραμμα θα έμπαινε σε κάθε κύκλο στην ISR του WCOMP, οπότε δεν θα μπορούσαμε να το αντιμετωπίσουμε. Ενεργοποιούμε κατάλληλα τον ADC μετά τον κάθε χειρισμό.
2. Τέλος, ελέγχει εάν η τιμή είναι πάνω η κάτω από το αντίστοιχο threshold και ενεργοποιεί το αντίστοιχο LED.
3. cli();
4. res = ADC0.RES;
5. ADC0.INTFLAGS |= ADC\_WCMP\_bm;
6. ADC0.CTRLA &= ~ADC\_ENABLE\_bm; // Disable ADC so we can handle the issue
8. if (res < ADC0.WINLT) { // Below lower threshold (need watering)
9. PORTD.OUTCLR = PIN0\_bm; // Turn LED0 on
10. }
11. else if (res > ADC0.WINHT) { // Above upper threshold (too humid)
12. PORTD.OUTCLR = PIN1\_bm; // Turn LED1 on
13. }
14. sei();

## ISR(PORTF\_PORT\_vect)

1. Απενεργοποιούμε τις διακοπές και διαβάζουμε τα flags.
2. cli();
3. int flags = PORTF.INTFLAGS; // Read once
5. Μέσω if-elseif-else ελέγχουμε εάν το πάτημα του χρήστη ήταν σωστό (εαν αντιστοιχεί στην λειτουργία που ζητά το θερμοκήπιο).
6. Στην περίπτωση που επιλέγεται σωστά το πότισμα καθαρίζουμε πιθανά interrupt flags για το CMP1, μηδενίζουμε τον χρονιστή, απενεργοποιούμε το PWM (εάν έχει τεθεί), θέτουμε την τιμή PER > WINLT (για να φτάνει ο μετρητής την τιμή του CMP1), ενεργοποιούμε μόνο το interrupt του CMP1, και ξεκινάμε τον χρονιστή. Τελος, καθαρίζουμε το interrupt του PIN5 του PORTF.
7. if( (flags & PIN5\_bm ) && (PORTD.OUT == 0b00000110) && !(flags & PIN6\_bm) ) {
8. TCA0.SINGLE.INTFLAGS = TCA\_SINGLE\_CMP1\_bm; // Clear possible interrupt
9. TCA0.SINGLE.CTRLB &= ~TCA\_SINGLE\_WGMODE\_SINGLESLOPE\_gc; // Disable PWM
10. TCA0.SINGLE.CNT = 0;
11. TCA0.SINGLE.PER = 0xFF; // Big resolution
12. TCA0.SINGLE.CMP1 = ADC0.WINLT - ADC0.RES; // configure Timer
13. TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP1\_bm; // Enable interrupt for the timer
14. TCA0.SINGLE.CTRLA |= TCA\_SINGLE\_ENABLE\_bm; // Start the timer
15. PORTF.INTFLAGS |= PIN5\_bm; // Clear the interrupt flag
16. }
17. Στην περίπτωση που επιλέγεται σωστά ο εξαερισμός καθαρίζουμε πιθανά interrupt flags για το OVF, μηδενίζουμε τον χρονιστή, θέτουμε την τιμή PER =20 (έχει υπολογιστεί 1ms) και το CMP0 με duty cycle 50%, ενεργοποιούμε την λειτουργία PWM και τα interrupts για την OVF ISR, και ξεκινάμε τον χρονιστή. Τελος, καθαρίζουμε το interrupt του PIN6 του PORTF.
18. else if( (flags & PIN6\_bm) && (PORTD.OUT == 0b00000101) && !(flags & PIN5\_bm)) {
19. TCA0.SINGLE.INTFLAGS = TCA\_SINGLE\_OVF\_bm;
20. TCA0.SINGLE.PER = 20; // Resolution (LPER+1)/ftimer = 1 -> LPER = 19.5 ~= 20
21. TCA0.SINGLE.CMP0 = 10; // Duty Cycle 50%
22. TCA0.SINGLE.CTRLB |= TCA\_SINGLE\_WGMODE\_SINGLESLOPE\_gc; // Single-slope PWM
23. TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_OVF\_bm; // Enable interrupt for bit toggling
25. TCA0.SINGLE.CTRLA |= TCA\_SINGLE\_ENABLE\_bm; // Start PWM
26. // TCA0.SINGLE.INTCTRL |= TCA\_SINGLE\_CMP0\_bm; // Enable interrupt for handling if needed
28. PORTF.INTFLAGS |= PIN6\_bm; // Clear the interrupt flag
29. }
30. Στην περίπτωση που επιλέγεται λάθος λειτουργία, ανάβουμε στιγμιαία όλα τα LED και τα σβήνουμε βηματικά. Στην συνέχεια καθαρίζουμε τα flags, ενεργοποιούμε ξανά τον ADC και ξεκινάμε τις μετατροπές.
31. else {
32. PORTD.OUTCLR |= PIN1\_bm | PIN0\_bm| PIN2\_bm; //turn on ALL LEDs to indicate wrong handling
33. *printf*("."); // give it time to clear
34. PORTD.OUT |= PIN2\_bm;
35. PORTD.OUT |= PIN1\_bm;
36. PORTD.OUT |= PIN0\_bm; //turn off ALL LEDs step by step
37. PORTF.INTFLAGS |= PIN5\_bm | PIN6\_bm; // Clear the interrupt flag
39. ADC0.CTRLA |= ADC\_ENABLE\_bm; // Enable ADC
40. ADC0.COMMAND |= ADC\_STCONV\_bm; //Start Conversion
41. }
42. sei();

## ISR(TCA0\_CMP1\_vect)

1. Εδώ απενεργοποιούμε τον TCA και το LED 0, καθαρίζουμε το σχετικό flag, και ξεκινάμε τον ADC. Αυτή η ISR καλείται όταν ολοκληρώνεται ο κύκλος ποτίσματος.

cli();

TCA0.SINGLE.CTRLA &= ~TCA\_SINGLE\_ENABLE\_bm; // Disable TCA0

PORTD.OUTSET = PIN0\_bm; // Turn off LED 0

TCA0.SINGLE.INTFLAGS |= TCA\_SINGLE\_CMP1\_bm; // Clear interrupt flag

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

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

sei();

## ISR(TCA0\_OVF\_vect)

1. Μέσω increment ενός global value, ορίζουμε την λειτουργία του LED 2 και του PWM, έτσι ώστε να κάνει 4 toggles το LED 2. Όταν ολοκληρωθούν τα toggles, απενεργοποιούμε τα LED 1,2, μηδενίζουμε την παγκόσμια μεταβλητή και ξεκινάμε τον ADC. Αυτή η ISR καλείται σε PWM mode, όταν ο μετρητής φτάνει την PER τιμή.

cli();

PORTD.OUT ^= PIN2\_bm; // Toggle LED 2

if(pwm\_counter == 3) {

PORTD.OUTSET = PIN1\_bm; // Turn off LED 1

PORTD.OUTSET = PIN2\_bm; // Turn off LED 2

pwm\_counter = 0; // Reset count

TCA0.SINGLE.CTRLA &= ~TCA\_SINGLE\_ENABLE\_bm; // Disable TCA0

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

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

} else {

pwm\_counter++;

}

TCA0.SINGLE.INTFLAGS |= TCA\_SINGLE\_OVF\_bm; // Clear interrupt flag

sei();

# Kώδικας

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdio.h>

void init\_ADC();

*uint8\_t* pwm\_counter = 0;

*uint8\_t* res;

int main(){

PORTD.DIR |= PIN1\_bm //PIN 0-2 is output

| PIN0\_bm

| PIN2\_bm;

PORTD.OUT |= PIN1\_bm //PIN 0-2 initialize OFF

| PIN0\_bm

| PIN2\_bm;

PORTF.PIN6CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_RISING\_gc; //configure button 6

PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_RISING\_gc; //configure button 5

TCA0.SINGLE.CTRLA = TCA\_SINGLE\_CLKSEL\_DIV1024\_gc; // Configure clock source, not enabled

init\_ADC();

sei(); //enable interrupts

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

while(1){

;

}

}

ISR(ADC0\_WCOMP\_vect){

cli();

res = ADC0.RES;

ADC0.INTFLAGS |= ADC\_WCMP\_bm;

ADC0.CTRLA &= ~ADC\_ENABLE\_bm; // Disable ADC so we can handle the issue

if (res < ADC0.WINLT) { // Below lower threshold (need watering)

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

}

else if (res > ADC0.WINHT) { // Above upper threshold (too humid)

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

}

sei();

}

ISR(PORTF\_PORT\_vect) {

cli();

int flags = PORTF.INTFLAGS; // Read once

// set watering

if( (flags & PIN5\_bm ) && (PORTD.OUT == 0b00000110) && !(flags & PIN6\_bm) ) {

TCA0.SINGLE.INTFLAGS = TCA\_SINGLE\_CMP1\_bm; // Clear possible interrupt

TCA0.SINGLE.CTRLB &= ~TCA\_SINGLE\_WGMODE\_SINGLESLOPE\_gc; // Disable PWM

TCA0.SINGLE.CNT = 0;

TCA0.SINGLE.PER = 0xFF; // Big resolution

TCA0.SINGLE.CMP1 = ADC0.WINLT - ADC0.RES; // configure Timer

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP1\_bm; // Enable interrupt for the timer

TCA0.SINGLE.CTRLA |= TCA\_SINGLE\_ENABLE\_bm; // Start the timer

PORTF.INTFLAGS |= PIN5\_bm; // Clear the interrupt flag

}

// set air

else if( (flags & PIN6\_bm) && (PORTD.OUT == 0b00000101) && !(flags & PIN5\_bm)) {

TCA0.SINGLE.INTFLAGS = TCA\_SINGLE\_OVF\_bm;

TCA0.SINGLE.PER = 20; // Resolution (LPER+1)/ftimer = 1ms -> LPER = 19.5 ~= 20

TCA0.SINGLE.CMP0 = 10; // Duty Cycle 50%

TCA0.SINGLE.CTRLB |= TCA\_SINGLE\_WGMODE\_SINGLESLOPE\_gc; // Single-slope PWM

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_OVF\_bm; // Enable interrupt for bit toggling

TCA0.SINGLE.CTRLA |= TCA\_SINGLE\_ENABLE\_bm; // Start PWM

// TCA0.SINGLE.INTCTRL |= TCA\_SINGLE\_CMP0\_bm; // Enable interrupt for handling if needed

PORTF.INTFLAGS |= PIN6\_bm; // Clear the interrupt flag

}

else {

PORTD.OUTCLR |= PIN1\_bm | PIN0\_bm| PIN2\_bm; //turn on ALL LEDs to indicate wrong handling

*printf*("."); // give it time to clear

PORTD.OUT |= PIN2\_bm;

PORTD.OUT |= PIN1\_bm;

PORTD.OUT |= PIN0\_bm; //turn off ALL LEDs step by step

PORTF.INTFLAGS |= PIN5\_bm | PIN6\_bm; // Clear the interrupt flag

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

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

}

sei();

}

ISR(TCA0\_CMP1\_vect) {

cli();

TCA0.SINGLE.CTRLA &= ~TCA\_SINGLE\_ENABLE\_bm; // Disable TCA0

PORTD.OUTSET = PIN0\_bm; // Turn off LED 0

TCA0.SINGLE.INTFLAGS |= TCA\_SINGLE\_CMP1\_bm; // Clear interrupt flag

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

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

sei();

}

ISR(TCA0\_OVF\_vect) {

cli();

PORTD.OUT ^= PIN2\_bm; // Toggle LED 2

if(pwm\_counter == 3) {

PORTD.OUTSET = PIN1\_bm; // Turn off LED 1

PORTD.OUTSET = PIN2\_bm; // Turn off LED 2

pwm\_counter = 0; // Reset count

TCA0.SINGLE.CTRLA &= ~TCA\_SINGLE\_ENABLE\_bm; // Disable TCA0

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

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

} else {

pwm\_counter++;

}

TCA0.SINGLE.INTFLAGS |= TCA\_SINGLE\_OVF\_bm; // Clear interrupt flag

sei();

}

void init\_ADC() {

//initialize the ADC for Free-Running mode

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

ADC0.CTRLA |= ADC\_FREERUN\_bm; // Free-Running mode enabled

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

ADC0.MUXPOS |= ADC\_MUXPOS\_AIN7\_gc; // PD7

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

//Window Comparator Mode

ADC0.WINLT = 50; // Set low threshold

ADC0.WINHT = 200; // Set high threshold

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

ADC0.CTRLE |= ADC\_WINCM\_OUTSIDE\_gc; // Interrupt when RESULT out of bounds

}