**Εργαστήριο Προηγμένοι Μικροεπεξεργαστές**

Εργαστηριακή Άσκηση 04:

Λειτουργία Ανεμιστήρα

2024-25

**Περιεχόμενα**

[Σχεδιασμός / Διάγραμμα Ροής 3](#__RefHeading___Toc1188_1975538019)

[Περιγραφή Κώδικα – Σχεδιαστική ιδέα 4](#__RefHeading___Toc30_1975538019_Copy_1)

[main 4](#__RefHeading___Toc666_1975538019_Copy_1_)

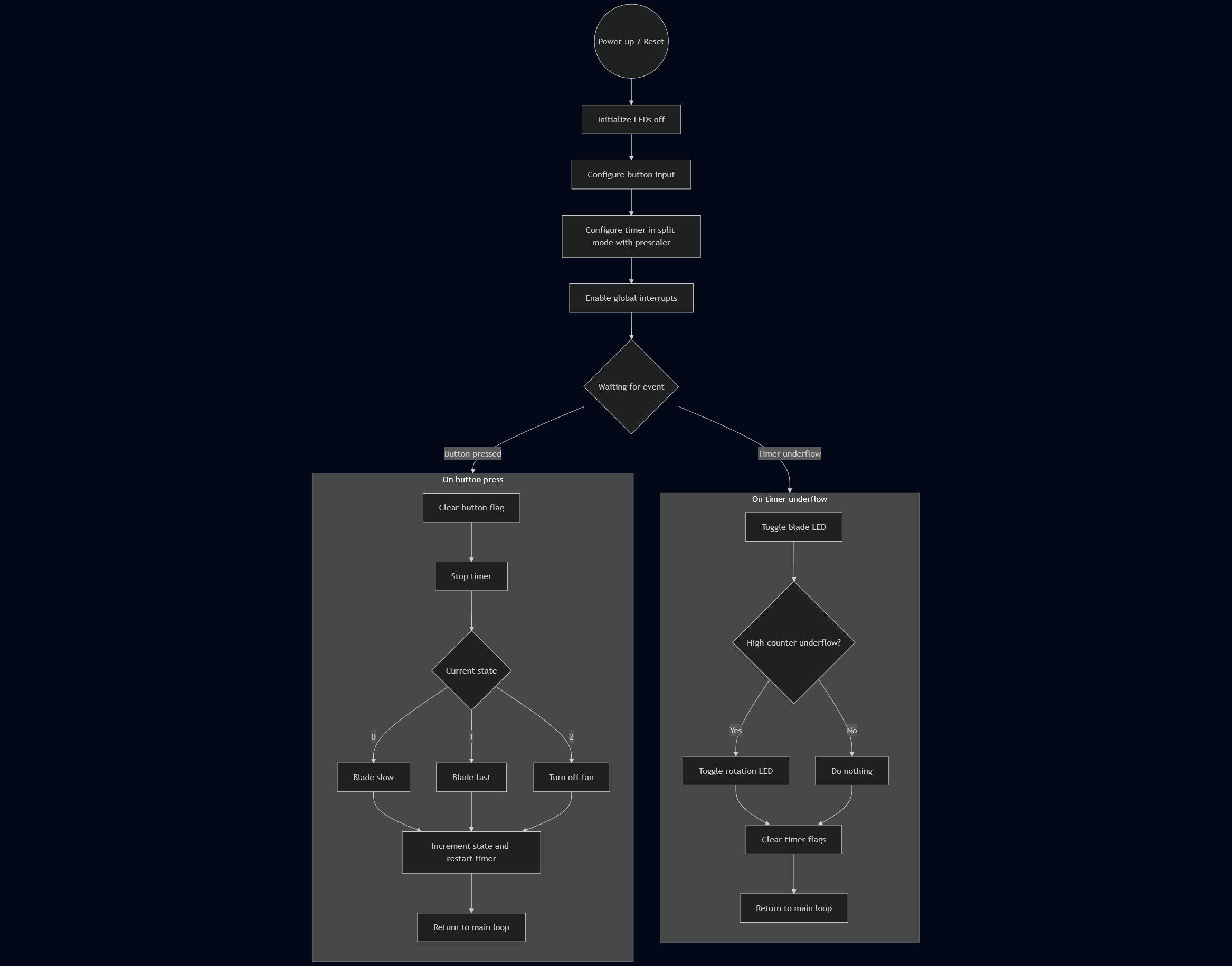
[ISR(PORTF\_PORT\_vect) 5](#__RefHeading___Toc5418_867001118)

[enable() 6](#__RefHeading___Toc666_1975538019_Copy_11)

[ISR(TCA0\_LUNF\_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 ανεμιστήρα με χρήση διαφορετικών PWM με διάφορα duty cycles.

Μετά από πολύωρη ενασχόληση και επικοινωνία με τους υπεύθυνους, συμπεράναμε πως τα ζητούμενα με τα duty cycles δεν δύναται να υλοποιηθούν σε split mode του TCA0 για τον high bit timer (καθώς δεν υπάρχει TCA\_SPLIT\_HCMPx\_bm), γιαυτό ο κώδικας υλοποιεί ένα απλό toggle των LED με prescaler, **χωρίς κάποια λειτουργία PWM.**

Για τον παραπάνω λόγο, οι γραμμές που αναφέρονται στα LCMP0 / HCMP0 **δεν κάνουν κάτι στο πρόγραμμα και βρίσκονται εκεί για ενδεικτικούς σκοπούς.**

Στο τέλος του κώδικα βρίσκεται σε σχόλιο η λειτουργία του PWM για τον low bit timer, καθώς και η ISR του HUNF για μια εναλλακτική λύση που δεν ελέγχει τα κριτήρια του high bit timer μέσα στην ISR του LUNF, με τα αρνητικά και τα θετικά της.

## main

Ορίζουμε τα pins εξόδου και τα αρχικοποιούμε κλειστά.

PORTD.DIR |= PIN0\_bm | PIN1\_bm; // Set PIN 0,1 as output

PORTD.OUT |= PIN0\_bm | PIN1\_bm; // Initialize both LEDs OFF

1. Αρχικοποιούμε το κουμπί και τον prescaler σε 256 (δεν τον ενεργοποιούμε ακόμα για λόγος εξοικονόμησης ενέργειας). Το πρόγραμμα μπαίνει σε ατέρμονο βρόγχο περιμένοντας interrupts.
2. PORTF.PIN6CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_RISING\_gc;
3. TCA0.SPLIT.CTRLD = TCA\_SPLIT\_SPLITM\_bm;
4. TCA0.SPLIT.CTRLA = TCA\_SPLIT\_CLKSEL\_DIV256\_gc;
5. sei();

while (1)

{

}

## ISR(PORTF\_PORT\_vect)

1. Καθαρίζουμε το interrupt του κουμπιού και απενεργοποιούμε τον TCA όσο αλλάζει η κατάσταση του συστήματος (αποφυγή edge cases).
2. int y = PORTF.INTFLAGS;
3. PORTF.INTFLAGS = y;
4. TCA0.SPLIT.CTRLA &= ~TCA\_SPLIT\_ENABLE\_bm;
6. Μέσω switch-case σε μία global μεταβλητή ρυθμίζουμε την περίοδο και το duty cycle του κάθε LED.
7. Στην περίπτωση 0 (πρώτο πάτημα του κουμπιού) θέτουμε τις (παρακάτω) κατάλληλες τιμές.
8. // f\_clk = 20MHz
9. // prescaler = 256
10. // ftimer = 20MHz / 256 = 38,022KHz ~= 38KHz
11. // Tb = 2ms
12. // fpwm\_ss = (f\_clk / N ) / (PER + 1) = ftimer / (PER + 1)
14. // HPER :
15. // HPER + 1 = ftimer / fpwm\_ss;
16. // HPER + 1 = 38 \* 2
17. // HPER = 75
19. // LPER :
20. // LPER + 1 = ftimer / fpwm\_ss;
21. // LPER + 1 = 38 \* 1
22. // LPER = 37
23. Στην περίπτωση 0 (πρώτο πάτημα του κουμπιού) θέτουμε τις (παρακάτω) κατάλληλες τιμές ως εξής: Στους HPER, LPER τις περιόδους και στους HCMP0, LCMP0 την τιμή του duty cycle. Στην συνέχεια αρχικοποιούμε τους μετρητές του TCA στην τιμή των HPER/LPER **(αλλιώς ενεργοποιούνται τα interrutps αμέσως)**, κάνουμε toggle τα leds μέσω BITWISE XOR (μή απαραίτητο) και καλούμε την συνάρτηση enable (ορισμένη παρακάτω).
24. switch (point) {
25. case 0:
26. // blade slow (60%) with 1ms period
27. TCA0.SPLIT.HPER = 75; // High period
28. TCA0.SPLIT.HCMP0 = 30; // 40% duty cycle
30. TCA0.SPLIT.LPER = 37; // Low period
31. TCA0.SPLIT.LCMP0 = 22; // 60% duty cycle
33. TCA0.SPLIT.LCNT = TCA0.SPLIT.LPER; // Restart low counter
34. TCA0.SPLIT.HCNT = TCA0.SPLIT.HPER; // Restart high counter
35. PORTD.OUT ^= PIN0\_bm; // Toggle LED0
36. PORTD.OUT ^= PIN1\_bm; // Toggle LED1
37. enable();
38. break;
39. Στην περίπτωση 1 (δεύτερο πάτημα του κουμπιού) θέτουμε τις κατάλληλες τιμές στον LPER και στον LCMP0, και αρχικοποιούμε τους μετρητές του TCA στην τιμή των HPER/LPER **(για να παραμείνουν συγχρονισμένα τα interrupts).** Στην συνέχεια καλούμε την συνάρτηση enable (ορισμένη παρακάτω)
40. case 1:
41. // blade fast (50%) with 0.5ms period
42. // LPER' = LPER / 2 = 18.5 ~= 19
43. TCA0.SPLIT.LPER = 19; // Low period
44. TCA0.SPLIT.LCMP0 = 10; // 50% duty cycle
45. TCA0.SPLIT.LCNT = TCA0.SPLIT.LPER; // Restart low counter
46. TCA0.SPLIT.HCNT = TCA0.SPLIT.HPER; // Restart high counter
48. enable();
49. break;
50. Στην περίπτωση 2 (τρίτο πάτημα του κουμπιού) απενεργοποιούμε τα LED και το interrupt της LUNF.
51. case 2:
52. // Turn off LEDs
53. PORTD.OUT |= PIN0\_bm | PIN1\_bm;
54. TCA0.SPLIT.INTCTRL &= ~TCA\_SPLIT\_LUNF\_bm; // Disable interrupt for underflow
55. break;
56. }

1. Τέλος, κάνουμε increment την global μεταβλητή mod2, για να κάνει loop από τα 0,1,2.

point++;

point %= 3; // Loop through 0, 1, 2

## enable()

1. Εδώ καθαρίζουμε υπολειπόμενα flags της LUNF, ενεργοποιούμε τα interrupts για την LUNF και ξεκινάμε τον χρονιστή.

TCA0.SPLIT.INTFLAGS = TCA\_SPLIT\_LUNF\_bm;

TCA0.SPLIT.INTCTRL = TCA\_SPLIT\_LUNF\_bm /\*| TCA\_SPLIT\_LCMP0\_bm \*/;

TCA0.SPLIT.CTRLA |= TCA\_SPLIT\_ENABLE\_bm;

## ISR(TCA0\_LUNF\_vect)

1. Κάνουμε toggle το LED0 και ελέγχουμε εάν έχει κάνει underflow και το high timer. Εάν έχει, κάνουμε toggle και το LED1 και καθαρίζουμε το flag του HUNF. Τέλος, καθαρίζουμε το flag του LUNF.

PORTD.OUT ^= PIN0\_bm; // Toggle PORTD0 (LED0)

if(TCA0.SPLIT.INTFLAGS & TCA\_SPLIT\_HUNF\_bm ) {

PORTD.OUT ^= PIN1\_bm; // Toggle PORTD1 (LED1)

TCA0.SPLIT.INTFLAGS = TCA\_SPLIT\_HUNF\_bm; // Clear the interrupt flag for high counter underflow

}

TCA0.SPLIT.INTFLAGS = TCA\_SPLIT\_LUNF\_bm; // Clear the interrupt flag for low counter underflow

# Kώδικας

#include <avr/io.h>

#include <avr/interrupt.h>

// LED 0 = blade movement

// LED 1 = rotation movement

int point = 0;

void enable() {

TCA0.SPLIT.INTFLAGS = TCA\_SPLIT\_LUNF\_bm; // Clear any interrupts

// TCA0.SPLIT.CTRLB = TCA\_SPLIT\_LCMP0\_bm; // Enable PWM output compare

TCA0.SPLIT.INTCTRL = TCA\_SPLIT\_LUNF\_bm /\*| TCA\_SPLIT\_LCMP0\_bm \*/; // Enable interrupts for underflow

TCA0.SPLIT.CTRLA |= TCA\_SPLIT\_ENABLE\_bm; // Enable the timer

}

int main(void)

{

PORTD.DIR |= PIN0\_bm | PIN1\_bm; // Set PIN 0,1 as output

PORTD.OUT |= PIN0\_bm | PIN1\_bm; // Initialize both LEDs OFF

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

TCA0.SPLIT.CTRLD = TCA\_SPLIT\_SPLITM\_bm; // Enable TCA Split Mode

TCA0.SPLIT.CTRLA = TCA\_SPLIT\_CLKSEL\_DIV256\_gc; // Configure clock source, not enabled

sei(); // Enable interrupts globally

while (1)

{

}

cli(); // Disable interrupts (will never be reached)

}

// Button Press ISR

ISR(PORTF\_PORT\_vect) {

// Clear the interrupt flag

int y = PORTF.INTFLAGS;

PORTF.INTFLAGS = y;

TCA0.SPLIT.CTRLA &= ~TCA\_SPLIT\_ENABLE\_bm; // Disable timer for configuration

// f\_clk = 20MHz

// prescaler = 256

// ftimer = 20MHz / 256 = 38,022KHz ~= 38KHz

// Tb = 2ms

// fpwm\_ss = (f\_clk / N ) / (PER + 1) = ftimer / (PER + 1)

// HPER :

// HPER + 1 = ftimer / fpwm\_ss;

// HPER + 1 = 38 \* 2

// HPER = 75

// LPER :

// LPER + 1 = ftimer / fpwm\_ss;

// LPER + 1 = 38 \* 1

// LPER = 37

switch (point) {

case 0:

// blade slow (60%) with 1ms period

TCA0.SPLIT.HPER = 75; // High period

TCA0.SPLIT.HCMP0 = 30; // 40% duty cycle

TCA0.SPLIT.LPER = 37; // Low period

TCA0.SPLIT.LCMP0 = 22; // 60% duty cycle

TCA0.SPLIT.LCNT = TCA0.SPLIT.LPER; // Restart low counter

TCA0.SPLIT.HCNT = TCA0.SPLIT.HPER; // Restart high counter

PORTD.OUT ^= PIN0\_bm; // Toggle LED0

PORTD.OUT ^= PIN1\_bm; // Toggle LED1

enable();

break;

case 1:

// blade fast (50%) with 0.5ms period

// LPER' = LPER / 2 = 18.5 ~= 19

TCA0.SPLIT.LPER = 19; // Low period

TCA0.SPLIT.LCMP0 = 10; // 50% duty cycle

TCA0.SPLIT.LCNT = TCA0.SPLIT.LPER; // Restart low counter

TCA0.SPLIT.HCNT = TCA0.SPLIT.HPER; // Restart high counter

enable();

break;

case 2:

PORTD.OUT |= PIN0\_bm | PIN1\_bm; // Turn off LEDs

TCA0.SPLIT.INTCTRL &= ~TCA\_SPLIT\_LUNF\_bm; // Disable interrupt for underflow

break;

}

point++;

point %= 3; // Loop through 0, 1, 2

}

// Low counter underflow ISR

ISR(TCA0\_LUNF\_vect) {

PORTD.OUT ^= PIN0\_bm; // Toggle PORTD0 (LED0)

if(TCA0.SPLIT.INTFLAGS & TCA\_SPLIT\_HUNF\_bm ) {

PORTD.OUT ^= PIN1\_bm; // Toggle PORTD1 (LED1)

TCA0.SPLIT.INTFLAGS = TCA\_SPLIT\_HUNF\_bm; // Clear the interrupt flag for high counter underflow

}

TCA0.SPLIT.INTFLAGS = TCA\_SPLIT\_LUNF\_bm; // Clear the interrupt flag for low counter underflow

}

// Alternative solution (better scalability, slightly worse time)

// Implementation -> LUNF to HUNF delay ~= 1μs

// Alternative solution -> LUNF to HUNF delay ~= 50μs

// Alternative solution has better scalability because each ISR takes up less time,

// so the system can respond quicker to other interrupts if needed

// and it makes the code easier to understand

// For this to work we would also need to enable the HUNF interrupt

/\*

// High counter underflow ISR

ISR(TCA0\_HUNF\_vect) {

TCA0.SPLIT.INTFLAGS = TCA\_SPLIT\_HUNF\_bm; // Clear the interrupt flag for high counter underflow

PORTD.OUT ^= PIN1\_bm; // Toggle PORTD1 (LED1)

}

\*/

// We would do this if we actually used the duty cycle

// which we can't do for HUNF using TCA0 in split mode

// due to it not having an equivalent TCA0\_HCMP0\_vect.

// The changes required for this solution are commented out

/\*

ISR(TCA0\_LCMP0\_vect) {

// Clear the interrupt flag for low counter underflow

TCA0.SPLIT.INTFLAGS = TCA\_SPLIT\_LCMP0\_bm;

PORTD.OUT ^= PIN0\_bm; // Toggle PORTD0 (LED0)

}

\*/