Εργαστήριο Προηγμένοι Μικροεπεξεργαστές

**Εργαστηριακή Άσκηση 3**

**Προσομοίωση κίνησης ανεμιστήρα**

**Στοιχεία Ομάδας :**

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

**Διαμαντόπουλος Ιωάννης , ΑΜ:1064281**

**Ομάδα : Β3**

**Ερώτημα 1.α - Κώδικας για απλή λειτουργία ανεμιστήρα με ενεργοποίηση διακόπτη και δύο LEDs**

#include <avr/io.h>

#include <avr/interrupt.h>

int x = 0;

int cntl = 0;

int cnth = 0;

int main(){

x=0;

PORTD.DIR = PIN0\_bm | PIN1\_bm; //PIN0 and PIN1 output

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

PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc; //init switch port5

sei(); //enable interrupts

while(x==0){ //while loop for switch

}

sei();

while (1){ //while loop for TCA

;

}

}

ISR(PORTF\_PORT\_vect){ //ISR for switch

int y = PORTF.INTFLAGS; //clear the interrupt flag

PORTF.INTFLAGS=y;

x=1;

TCA0.SPLIT.CTRLD = TCA\_SPLIT\_SPLITM\_bm; //enable split mode for TCA

TCA0.SPLIT.CTRLA=TCA\_SPLIT\_CLKSEL\_DIV1024\_gc; //prescaler=1024

TCA0.SPLIT.LPER = 20; //select the resolution for base

TCA0.SPLIT.LCMP0 = 8; //select the duty cycle for base

TCA0.SPLIT.HPER = 10; //select the resolution for blade

TCA0.SPLIT.HCMP0 = 5; //select the duty cycle for blade

TCA0.SPLIT.CTRLB = TCA\_SPLIT\_HCMP0EN\_bm | TCA\_SPLIT\_LCMP0EN\_bm;

TCA0.SPLIT.INTCTRL |= TCA\_SPLIT\_HUNF\_bm; //enable HUNF interrupt

TCA0.SPLIT.CTRLA |= TCA\_SPLIT\_CLKSEL\_DIV1024\_gc | TCA\_SPLIT\_ENABLE\_bm; //start timer, set clock

}

ISR(TCA0\_HUNF\_vect) { //ISR for TCA (HUNF)

//interrupt for high bits

if ((TCA0.SPLIT.INTFLAGS & 0b00000010)==2)

{

cnth++;

if(cnth%2==1){

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

}

else{

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

}

}

//interrupt for low bits

if ((TCA0.SPLIT.INTFLAGS & 0b00000001)==1)

{

cntl++;

if(cntl%2==1){

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

}

else{

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

}

}

int intflags = TCA0.SPLIT.INTFLAGS; //clear flag

TCA0.SPLIT.INTFLAGS = intflags;

}

**Ερώτημα 1.β - Κώδικας για λειτουργία ανεμιστήρα, με επιπρόσθετη τη λειτουργία του ADC και τη διαδικασία επανεκκίνησης μέσω του διακόπτη**

#include <avr/io.h>

#include <avr/interrupt.h>

int x = 0;

int cntl = 0;

int cnth = 0;

int stop = 0;

int main(){

PORTD.DIR = PIN0\_bm | PIN1\_bm | PIN2\_bm; //PIN0, PIN1, PIN2 output

while(1){

x=0;

stop=0;

PORTD.OUT = PIN0\_bm | PIN1\_bm | PIN2\_bm; //PIN0, PIN1, PIN2 off

PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc; //init switch port5

sei(); //enable interrupts

while(x==0){ //while loop for switch

}

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

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

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

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

//Enable Debug Mode

ADC0.DBGCTRL |= ADC\_DBGRUN\_bm;

//Window Comparator Mode

ADC0.WINLT |= 10; //Set threshold

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

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

sei(); //enable interrupts

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

while (stop==0){ //while loop for ADC

;

}

cli(); //disable interrupts

}

}

ISR(PORTF\_PORT\_vect){ //ISR for switch

int y = PORTF.INTFLAGS; //clear the interrupt flag

PORTF.INTFLAGS=y;

x=1;

TCA0.SPLIT.CTRLD = TCA\_SPLIT\_SPLITM\_bm; //enable split mode for TCA

TCA0.SPLIT.CTRLA=TCA\_SPLIT\_CLKSEL\_DIV1024\_gc; //prescaler=1024

TCA0.SPLIT.LPER = 20; //select the resolution for base

TCA0.SPLIT.LCMP0 = 8; //select the duty cycle for base

TCA0.SPLIT.HPER = 10; //select the resolution for blade

TCA0.SPLIT.HCMP0 = 5; //select the duty cycle for blade

TCA0.SPLIT.CTRLB = TCA\_SPLIT\_HCMP0EN\_bm | TCA\_SPLIT\_LCMP0EN\_bm;

TCA0.SPLIT.INTCTRL |= TCA\_SPLIT\_HUNF\_bm; //enable HUNF interrupt

TCA0.SPLIT.CTRLA |= TCA\_SPLIT\_CLKSEL\_DIV1024\_gc | TCA\_SPLIT\_ENABLE\_bm; //start timer, set clock

}

ISR(TCA0\_HUNF\_vect) { //ISR for TCA (HUNF)

//interrupt for high bits

if ((TCA0.SPLIT.INTFLAGS & 0b00000010)==2)

{

cnth++;

if(cnth%2==1){

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

}

else{

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

}

}

//interrupt for low bits

if ((TCA0.SPLIT.INTFLAGS & 0b00000001)==1)

{

cntl++;

if(cntl%2==1){

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

}

else{

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

}

}

int intflags = TCA0.SPLIT.INTFLAGS; //clear flag

TCA0.SPLIT.INTFLAGS = intflags;

}

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

int intflags = ADC0.INTFLAGS; //clear flag

ADC0.INTFLAGS = intflags;

PORTD.OUT |= PIN0\_bm | PIN1\_bm; //LED0 and LED1 off

PORTD.OUTCLR = PIN2\_bm; //LED2 on => fan off

stop=1;

TCA0.SPLIT.CTRLA = 0b00000000; //disable timer

}

**Ερώτημα 1.γ - Κώδικας για λειτουργία ανεμιστήρα, με επιπρόσθετη τη λειτουργία επιτάχυνσης των στροφών της λεπίδας και διαδικασία απενεργοποίησης**

#include <avr/io.h>

#include <avr/interrupt.h>

int x = 0;

int cntl = 0;

int cnth = 0;

int stop = 0;

int switch\_cnt = 0;

int main(){

PORTD.DIR = PIN0\_bm | PIN1\_bm | PIN2\_bm; //PIN0, PIN1, PIN2 output

while(switch\_cnt<3){

x=0;

stop=0;

switch\_cnt=0;

PORTD.OUT = PIN0\_bm | PIN1\_bm | PIN2\_bm; //PIN0, PIN1, PIN2 off

PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc; //init switch port5

sei(); //enable interrupts

while(x==0){ //while loop for switch

}

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

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

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

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

//Enable Debug Mode

ADC0.DBGCTRL |= ADC\_DBGRUN\_bm;

//Window Comparator Mode

ADC0.WINLT |= 10; //Set threshold

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

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

sei(); //enable interrupts

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

while (stop==0 && switch\_cnt!=3){

;

}

cli(); //disable interrupts

}

}

ISR(PORTF\_PORT\_vect){ //ISR for switch

int y = PORTF.INTFLAGS; //clear the interrupt flag

PORTF.INTFLAGS=y;

x=1;

switch\_cnt++; //counter for switch press

TCA0.SPLIT.CTRLD = TCA\_SPLIT\_SPLITM\_bm; //enable split mode for TCA

TCA0.SPLIT.CTRLA=TCA\_SPLIT\_CLKSEL\_DIV1024\_gc; //prescaler=1024

TCA0.SPLIT.LPER = 20; //select the resolution for base

TCA0.SPLIT.LCMP0 = 8; //select the duty cycle for base

if(switch\_cnt==1){

TCA0.SPLIT.HPER = 10; //select the resolution for blade

TCA0.SPLIT.HCMP0 = 5; //select the duty cycle for blade

TCA0.SPLIT.CTRLB = TCA\_SPLIT\_HCMP0EN\_bm | TCA\_SPLIT\_LCMP0EN\_bm;

TCA0.SPLIT.INTCTRL |= TCA\_SPLIT\_HUNF\_bm; //enable HUNF interrupt

TCA0.SPLIT.CTRLA |= TCA\_SPLIT\_CLKSEL\_DIV1024\_gc | TCA\_SPLIT\_ENABLE\_bm; //start timer, set clock

}

else if(switch\_cnt==2){

TCA0.SPLIT.HPER = 20; //select the resolution for blade

TCA0.SPLIT.HCMP0 = 10; //select the duty cycle for blade

TCA0.SPLIT.CTRLB = TCA\_SPLIT\_HCMP0EN\_bm | TCA\_SPLIT\_LCMP0EN\_bm;

TCA0.SPLIT.INTCTRL |= TCA\_SPLIT\_HUNF\_bm; //enable HUNF interrupt

TCA0.SPLIT.CTRLA |= TCA\_SPLIT\_CLKSEL\_DIV1024\_gc | TCA\_SPLIT\_ENABLE\_bm; //start timer, set clock

}

else{

TCA0.SPLIT.CTRLA = 0b00000000; //disable timer

}

}

ISR(TCA0\_HUNF\_vect) { //ISR for TCA (HUNF)

//interrupt for high bits

if ((TCA0.SPLIT.INTFLAGS & 0b00000010)==2)

{

cnth++;

if(cnth%2==1){

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

}

else{

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

}

}

//interrupt for low bits

if ((TCA0.SPLIT.INTFLAGS & 0b00000001)==1)

{

cntl++;

if(cntl%2==1){

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

}

else{

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

}

}

int intflags = TCA0.SPLIT.INTFLAGS; //clear flag

TCA0.SPLIT.INTFLAGS = intflags;

}

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

int intflags = ADC0.INTFLAGS; //clear flag

ADC0.INTFLAGS = intflags;

PORTD.OUT |= PIN0\_bm | PIN1\_bm; //LED0 and LED1 off

PORTD.OUTCLR = PIN2\_bm; //LED2 on => fan off

stop=1;

TCA0.SPLIT.CTRLA = 0b00000000; //disable timer

}

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

Σε ότι έχει να κάνει με τον κώδικα του πρώτου ερωτήματος, αρχικά, γίνεται η αρχικοποίηση της μεταβλητής *x* σε 0, η οποία, θα αξιοποιηθεί στη συνέχεια για τον διακόπτη μας (switch). Ομοίως, γίνεται η αρχικοποίηση των μεταβλητών *cntl* και *cnth* σε 0, τις οποίες θα τις αξιοποιήσουμε στην *ISR(TCA0\_HUNF\_vect)* για να ενεργοποιήσουμε ή να απενεργοποιήσουμε τα LED0 (αφορά τη λεπίδα) και LED1 (αφορά τη βάση).

Στην main(), με την εντολή *PORTD.DIR = PIN0\_bm | PIN1\_bm* ορίζουμε ότι θα χρησιμοποιήσουμε τα LED0 και LED1. Στη συνέχεια, με την εντολή *PORTD.OUT = PIN0\_bm | PIN1\_bm* απενεργοποιούμε τα προαναφερθέντα LED. Με την εντολή *PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc* αρχικοποιούμε τον διακόπτη ενεργοποίησης/απενεργοποίησηςτου ανεμιστήρα στο Port5. Όταν ο χρήστης πατήσει το PIN5 (bit5 = 1 στον INTFLAGS), θα μεταφερθεί στην *ISR(PORTF\_PORT\_vect)*.

Σε ότι έχει να κάνει με το περιεχόμενο της *ISR(PORTF\_PORT\_vect*, δηλαδή της διακοπής που προκαλείται χάρη στο switch, θέτουμε τη μεταβλητή *x* ίση με 1, ώστε μόλις τρέξουν όλες οι εντολές της ISR, το πρόγραμμα να εξέλθει της *while(x==0)* της main(). Εν συνεχεία, η εντολή *TCA0.SPLIT.CTRLD = TCA\_SPLIT\_SPLITM\_bm*, θέτει τον TCA σε split mode. Ως αποτέλεσμα, ο TCA δεν είναι 16bit πλέον, αλλά δύο 8bit timer(HUNF, LUNF). Με την εντολή *TCA0.SPLIT.CTRLA=TCA\_SPLIT\_CLKSEL\_DIV1024\_gc*, θέτουμε το ρολόι, με prescaler ίσο με 1024. Επιπρόσθετα, θέτουμε τιμές για τα LPER, LCMP0, HPER και HCMP0. Ειδικότερα, οι LPER και HPER παίρνουν τιμές που αφορούν το χρόνο του εκάστοτε παλμού (περίοδος), ενώ οι LCMP0 και HCMP0 παίρνουν τιμές που αφορούν τον κύκλο λειτουργίας (duty cycle). Παρατηρούμε ότι σχετικά με τους κύκλους λειτουργίας και με βάση την εκφώνηση, για τη βάση το duty cycle είναι 40% ενώ για την λεπίδα 50%. Αυτό αποδεικνύεται και παρακάτω στις εντολές που αναφέρουμε :

TCA0.SPLIT.LPER = 20;

TCA0.SPLIT.LCMP0 = 8;

TCA0.SPLIT.HPER = 10;

TCA0.SPLIT.HCMP0 = 5;

Με την εντολή *TCA0.SPLIT.CTRLB = TCA\_SPLIT\_HCMP0EN\_bm | TCA\_SPLIT\_LCMP0EN\_bm* ο TCA ενεργοποιεί τα κανάλια σύγκρισης high και low (HUNF, LUNF). Τέλος, με την εντολή *TCA0.SPLIT.INTCTRL |= TCA\_SPLIT\_HUNF\_bm* ενεργοποιούνται τα interrupts για ένα HUNF γεγονός, και αν γίνει το interrupt, το πρόγραμμα θα μεταφερθεί στην *ISR(TCA0\_HUNF\_vect)*.

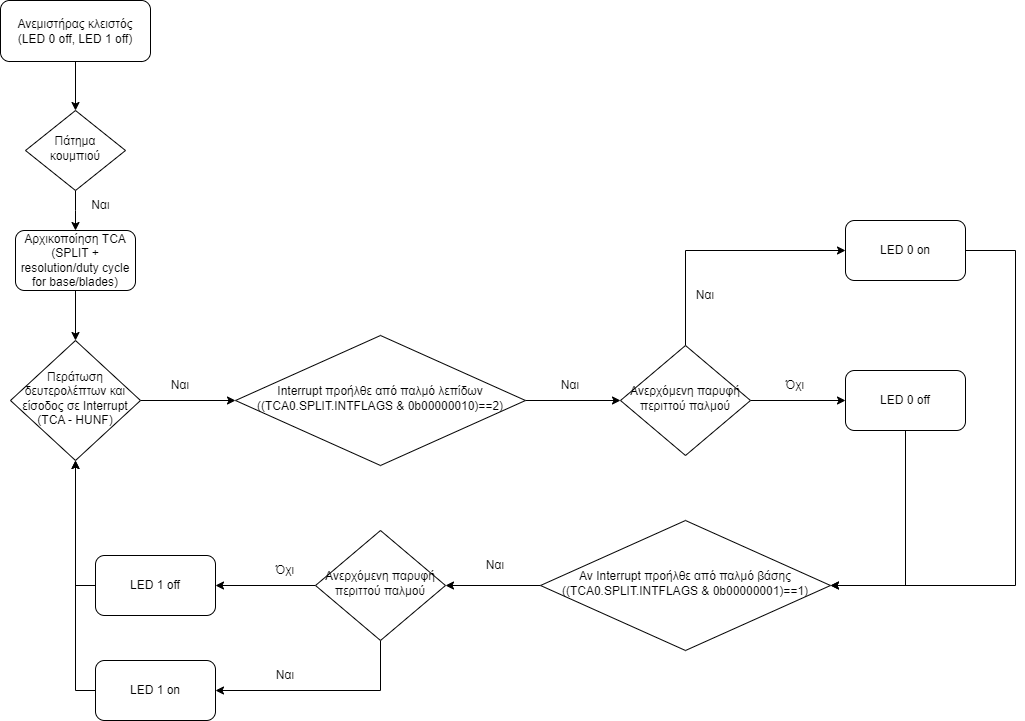
Όταν το πρόγραμμα εξέλθει της άνωθεν ISR, συνεχίζει και εγκλωβίζεται στην *while (1)*. Πρακτικά αυτό σημαίνει ότι έχει ήδη αρχίσει η λειτουργία του ανεμιστήρα και σε κάθε ανερχόμενη παρυφή του παλμού το πρόγραμμα εισέρχεται στην *ISR(TCA0\_HUNF\_vect)*. Χάρη στην εντολή if *((TCA0.SPLIT.INTFLAGS & 0b00000010)==2)*, αξιοποιώντας τη δυνατότητα της μάσκας (mask) και του INTFLAGS του TCA, ελέγχουμε αν η είσοδος στο interrupt προέρχεται από τον παλμό που δίνει η κίνηση των λεπίδων. Αν ναι, αυξάνουμε κατά ένα τον μετρητή cnth με την εντολή *cnth++*, ώστε στη συνέχεια, να ελέγχουμε με την εντολή *if(cnth%2==1)* αν βρίσκεται σε άρτια ή περιττή ανερχόμενη παρυφή παλμού. Αυτός ο έλεγχος γίνεται, καθώς επιθυμούμε στο πρώτο Rising Edge του παλμού να ανάβει το LED0, στο δεύτερο Rising Edge να σβήνει το LED0, στο τρίτο Rising Edge να ανάβει το LED0 κ.ο.κ. Τo LED ενεργοποιείται και απενεργοποιείται με τις εντολές *PORTD.OUTCLR = PIN0\_bm* και *PORTD.OUT |= PIN0\_bm* αντίστοιχα. Ομοίως, στη συνέχεια, το πρόγραμμα ελέγχει αν η πράξη *TCA0.SPLIT.INTFLAGS & 0b00000001* δίνει αποτέλεσμα 1, κάτι που σημαίνει πως βρισκόμαστε κατά τη διάρκεια της κίνησης της βάσης.

Συμπερασματικά, το πρόγραμμα περιμένει να πατηθεί το κουμπί (switch) και συνεχίζει μόνο εάν πατηθεί. Σε περίπτωση που συμβεί αυτό (πατηθεί το switch), ξεκινάει η λειτουργία του ανεμιστήρα, ο οποίος, διαθέτει δύο LED, τα οποία ενεργοποιούνται και απενεργοποιούνται κατάλληλα, με έναν σταθερό ρυθμό. Παρατηρείται ότι σε κάθε είσοδο του interrupt αλλάζει η κατάσταση του LED0 (λεπίδα) και ανά δύο interrupt αλλάζει κατάσταση το LED1.

Σε αυτό το ερώτημα, όπως και στα υπόλοιπα, έπρεπε να δοθεί προσοχή στον συγχρονισμό των παλμών. Ο λόγος είναι, πως και οι δύο timers «σκάνε» ταυτόχρονα. Έτσι το πρόγραμμα δεν ξέρει σε ποια ISR να πάει (είτε σε αυτή του HUNF, είτε σε αυτή του LUNF). Ως αποτέλεσμα, το πρόγραμμα θα πρέπει να επιλέξει τυχαία σε ποια ISR να πάει ανάμεσα στις δύο που προαναφέραμε. Το πρόβλημα λοιπόν είναι, ότι καθώς εκτελείται τυχαία μία από τις δύο ρουτίνες, η συχνότητα του LED χάνεται. Ο τρόπος που επιλέξαμε να χειριστούμε αυτό το πρόβλημα, όπως φαίνεται και στον κώδικα αλλά και στην επεξήγησή του είναι ο εξής :

* Ορίζουμε μονάχα μια ISR (εμείς την *ISR(TCA0\_HUNF\_vect)*)
* Σε αυτή την ISR πρέπει να ελέγξουμε αν πραγματοποιήθηκε interrupt από τα low bytes (LUNF). Αυτό επιτυγχάνεται με το INTFLAG. Δηλαδή, μέσα στην ISR των high bytes, ελέγχεται το συγκεκριμένο bit των INTFLAGS, το οποίο δηλώνει ότι υπάρχει interrupt.
* Παράδειγμα ελέγχου : *if ((TCA0.SPLIT.INTFLAGS & 0b00000001)==1)*
* Για τον λόγο αυτό, ο μηδενισμός του INTFLAG γίνεται μετά τους ελέγχους, δηλαδή πριν το κλείσιμο της *ISR(TCA0\_HUNF\_vect).*

Ακολουθεί το διάγραμμα ροής :



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

Σε αυτό το ερώτημα, προσθέσαμε την λειτουργία του ADC. Αρχικά, γίνεται η αρχικοποίηση της μεταβλητής *x* σε 0, η οποία, θα αξιοποιηθεί στη συνέχεια για τον διακόπτη μας (switch). Ομοίως, γίνεται η αρχικοποίηση των μεταβλητών *cntl* και *cnth* σε 0, τις οποίες θα τις αξιοποιήσουμε στην *ISR(TCA0\_HUNF\_vect)* για να ενεργοποιήσουμε ή να απενεργοποιήσουμε τα LED0 (αφορά τη λεπίδα) και LED1 (αφορά τη βάση). Επιπρόσθετα, ορίζουμε μια νέα μεταβλητή *stop*, την οποία θα αξιοποιήσουμε παρακάτω για τον ADC, ώστε το πρόγραμμά μας να εξέρχεται της *while (stop==0)*.

Στην main(), με την εντολή *PORTD.DIR = PIN0\_bm | PIN1\_bm | PIN2\_bm* ορίζουμεότι θα χρησιμοποιήσουμε τα LED0, LED1 και LED2. Ακολουθεί η *while(1)* την οποία την χρειαζόμαστε ώστε το πρόγραμμά μας να τρέχει συνέχεια, και σε περίπτωση αξιοποίησης του ADC, ο χρήστης να μπορεί να ξανά πατήσει το κουμπί (switch), ώστε να εκκινήσει και πάλι ο ανεμιστήρας. Στην αρχή της προαναφερθείσας *while(1)*, αρχικοποιούμε σε 0 τις τιμές *x* και *stop*, ώστε αν ο ανεμιστήρας ξανά ξεκινήσει να λειτουργεί με το πάτημα του κουμπιού (switch), να έχουν τεθεί οι σωστές τιμές στις μεταβλητές αυτές για τη σωστή λειτουργία του προγράμματος (reset). Στη συνέχεια, με την εντολή *PORTD.OUT = PIN0\_bm | PIN1\_bm | PIN2\_bm* απενεργοποιούμε τα προαναφερθέντα LED. Με την εντολή *PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc* δίνουμε την δυνατότητα στο πρόγραμμα, μόλις ενεργοποιηθούν οι διακοπές (interrupts) με την εντολή *sei()*, και ο χρήστης πατήσει το PIN5 στον INTFLAGS, να μεταφερθεί στην *ISR(PORTF\_PORT\_vect)*.

Εάν πατηθεί το κουμπί, το πρόγραμμα μεταφέρεται στην *ISR(PORTF\_PORT\_vect)*, σηματοδοτώντας έτσι την εκκίνηση του ανεμιστήρα. Αρχικά σε αυτή την ISR, καθαρίζονται τα flags και τίθεται η μεταβλητή *x* στην τιμή 1, ώστε μόλις εκτελεστούν όλες οι εντολές της ISR, το πρόγραμμα να εξέλθει της *while(x==0)* της main(). Επίσης, η εντολή *TCA0.SPLIT.CTRLD = TCA\_SPLIT\_SPLITM\_bm*, θέτει τον TCA σε split mode. Ως αποτέλεσμα, ο TCA δεν είναι 16bit πλέον, αλλά δύο 8bit timer(HUNF, LUNF). Με την εντολή *TCA0.SPLIT.CTRLA=TCA\_SPLIT\_CLKSEL\_DIV1024\_gc*, θέτουμε το ρολόι, με prescaler ίσο με 1024. Επιπρόσθετα, θέτουμε τιμές για τα LPER, LCMP0, HPER και HCMP0. Ειδικότερα, οι LPER και HPER παίρνουν τιμές που αφορούν το χρόνο του εκάστοτε παλμού (περίοδος), ενώ οι LCMP0 και HCMP0 παίρνουν τιμές που αφορούν τον κύκλο λειτουργίας (duty cycle). Παρατηρούμε ότι σχετικά με τους κύκλους λειτουργίας και σύμφωνα με την εκφώνηση, για τη βάση το duty cycle είναι 40% ενώ για την λεπίδα 50%. Αυτό αποδεικνύεται και παρακάτω στις εντολές που αναφέρουμε :

TCA0.SPLIT.LPER = 20;

TCA0.SPLIT.LCMP0 = 8;

TCA0.SPLIT.HPER = 10;

TCA0.SPLIT.HCMP0 = 5;

Με την εντολή *TCA0.SPLIT.CTRLB = TCA\_SPLIT\_HCMP0EN\_bm | TCA\_SPLIT\_LCMP0EN\_bm* ο TCA ενεργοποιεί τα κανάλια σύγκρισης high και low (HUNF, LUNF). Τέλος, με την εντολή *TCA0.SPLIT.INTCTRL |= TCA\_SPLIT\_HUNF\_bm* ενεργοποιούνται τα interrupts για ένα HUNF γεγονός, και αν γίνει το interrupt, το πρόγραμμα θα μεταφερθεί στην *ISR(TCA0\_HUNF\_vect)*.

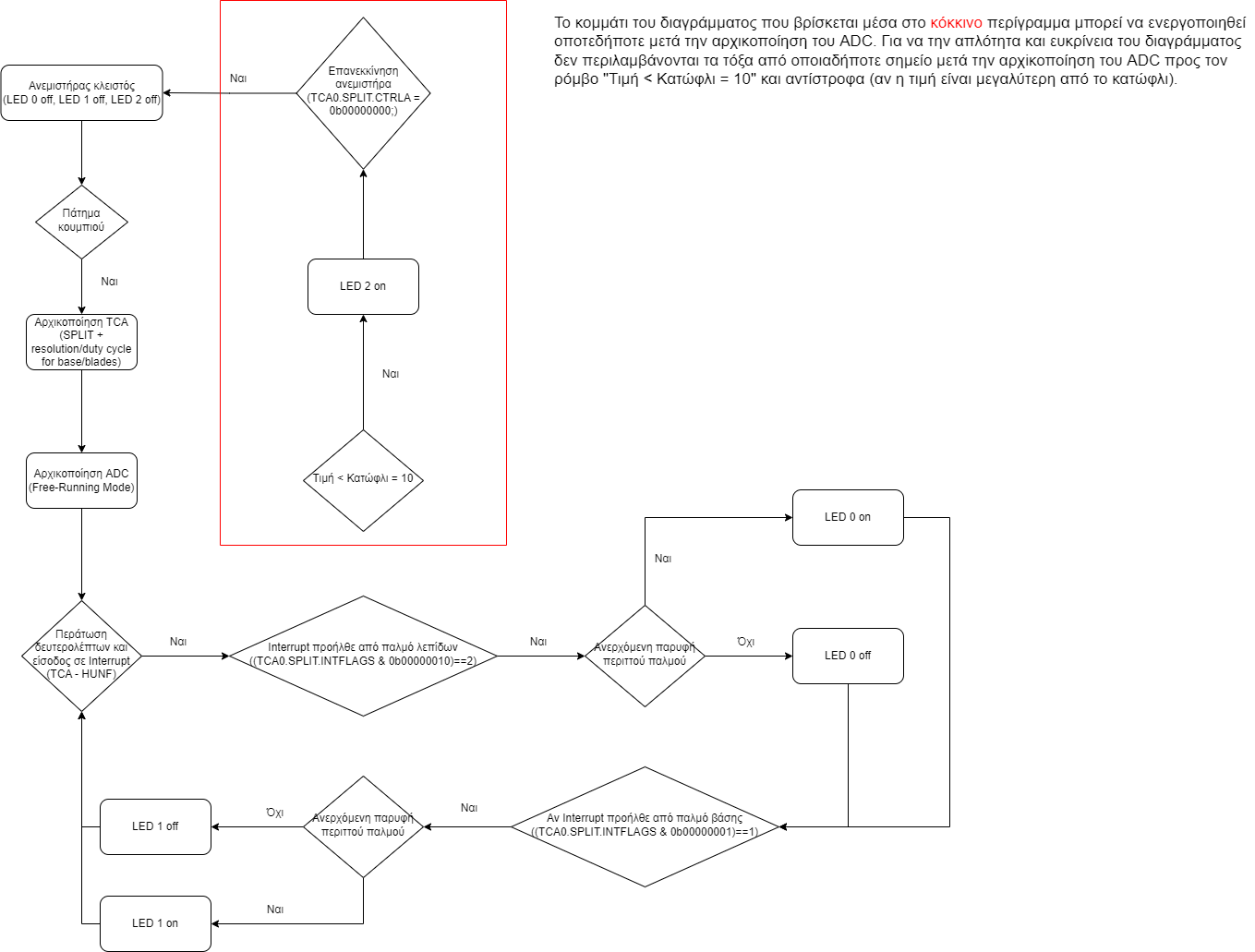
Πηγαίνοντας στη main(), αμέσως μετά την *while(x==0)* που εξήλθε το πρόγραμμα, ορίζουμε τον ADC με τις ακόλουθες εντολές :

* 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\_FREERUN\_bm* 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.WINLT |= 10* θέτουμε το κατώφλι στον καταχωρητή *ADC0.WINLT.*
* Με την εντολή *ADC0.CTRLE |= 0b00000001* ορίζουμε ότι θέλουμε να γίνεται interrupt όταν RESULT < WINLT.

Εφόσον ενεργοποιήσαμε τον ADC, ενεργοποιούμε τις διακοπές με την εντολή *sei()* και στη συνέχεια με την εντολή *ADC0.COMMAND |= ADC\_STCONV\_bm* ξεκινάει το conversion. To πρόγραμμά μας εισέρχεται στη *while (stop==0)*, και παραμένει εκεί έως ότου κάποιο αντικείμενο εντοπιστεί πολύ κοντά στις λεπίδες που περιστρέφονται, δηλαδή έως ότου η μεταβλητή *stop* γίνει 1. Σε αυτό το σημείο υπάρχουν δύο πιθανές περιπτώσεις :

1. Η τιμή του RESULT είναι μεγαλύτερη του WINLT (RESULT>WINLT) και έτσι το πρόγραμμά μας δεν εισέρχεται στην ISR του ADC *[ISR(ADC0\_WCOMP\_vect)]*, και επειδή είναι ενεργοποιημένες οι διακοπές και έχει οριστεί το ρολόι, εισέρχεται στην *ISR(TCA0\_HUNF\_vect)*. Χάρη στην εντολή if *((TCA0.SPLIT.INTFLAGS & 0b00000010)==2)*, αξιοποιώντας τη δυνατότητα της μάσκας (mask) και του INTFLAGS του TCA, ελέγχουμε αν η είσοδος στο interrupt προέρχεται από τον παλμό που δίνει η κίνηση των λεπίδων. Αν ναι, αυξάνουμε κατά ένα τον μετρητή cnth με την εντολή *cnth++*, ώστε στη συνέχεια, να ελέγχουμε με την εντολή *if(cnth%2==1)* αν βρίσκεται σε άρτια ή περιττή ανερχόμενη παρυφή παλμού. Αυτός ο έλεγχος γίνεται, καθώς επιθυμούμε στο πρώτο Rising Edge του παλμού να ανάβει το LED0, στο δεύτερο Rising Edge να σβήνει το LED0, στο τρίτο Rising Edge να ανάβει το LED0 κ.ο.κ. Τo LED ενεργοποιείται και απενεργοποιείται με τις εντολές *PORTD.OUTCLR = PIN0\_bm* και *PORTD.OUT |= PIN0\_bm* αντίστοιχα. Ομοίως, στη συνέχεια, το πρόγραμμα ελέγχει αν η πράξη *TCA0.SPLIT.INTFLAGS & 0b00000001* δίνει αποτέλεσμα 1, κάτι που σημαίνει πως βρισκόμαστε κατά τη διάρκεια της κίνησης της βάσης. Με άλλα λόγια είναι η περίπτωση χρήσης του πρώτου ερωτήματος.
2. Η τιμή του RESULT είναι μικρότερη του WINLT (RESULT<WINLT) και έτσι το πρόγραμμά μας εισέρχεται στην ISR του ADC *[ISR(ADC0\_WCOMP\_vect)]*. Για να εξέλθει της ISR πρέπει η τιμή του RESULT να γίνει μεγαλύτερη από την τιμή του WINLT. Σε αυτή την ISR, αρχικά, καθαρίζονται τα flags, και στη συνέχεια απενεργοποιούμε τα LED0 και LED1, που φανερώνουν περιστροφή λεπίδας και βάσης, και ενεργοποιούμε το LED2, που φανερώνει ότι ο ανεμιστήρας απενεργοποιήθηκε. Τα άνωθεν, γίνονται με τη χρήση των εντολών *PORTD.OUT |= PIN0\_bm | PIN1\_bm* και *PORTD.OUTCLR = PIN2\_bm* αντίστοιχα. Τέλος, ορίζουμε τη μεταβλητή *stop* στην τιμή 1 και κάνουμε disable τον TCA με την εντολή *TCA0.SPLIT.CTRLA = 0b00000000*.

Αφού το πρόγραμμα εξέλθει της ISR του ADC, παράλληλα εξέρχεται και της *while (stop==0)* και απενεργοποιούνται οι διακοπές με την εντολή *cli()*. Το πρόγραμμα συνεχίζει να βρίσκεται στην *while(1)* και κάνει reset απενεργοποιώντας όλα τα LED και περιμένοντας τον χρήστη να ξανά πατήσει του κουμπί για να επαναλειτουργήσει ο ανεμιστήρας.

****Ακολουθεί το διάγραμμα ροής :

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

Σε ότι έχει να κάνει με το ερώτημα αυτό, δίνουμε την δυνατότητα στον χρήστη να επιταχύνει την κίνηση της λεπίδας αλλά και να απενεργοποιήσει τον ανεμιστήρα. Αρχικά, γίνεται η αρχικοποίηση της μεταβλητής *x* σε 0, η οποία, θα αξιοποιηθεί στη συνέχεια για τον διακόπτη μας (switch). Ομοίως, γίνεται η αρχικοποίηση των μεταβλητών *cntl* και *cnth* σε 0, τις οποίες θα τις αξιοποιήσουμε στην *ISR(TCA0\_HUNF\_vect)* για να ενεργοποιήσουμε ή να απενεργοποιήσουμε τα LED0 (αφορά τη λεπίδα) και LED1 (αφορά τη βάση). Από το προηγούμενο ερώτημα έχουμε ήδη ορίσει τη μεταβλητή *stop*, την οποία θα αξιοποιήσουμε παρακάτω, μαζί με μία νέα μεταβλητή *switch\_cnt*, ώστε το πρόγραμμά μας να εξέρχεται της *while (stop==0 && switch\_cnt!=3)*. Ουσιαστικά, η μεταβλητή *switch\_cnt*, μετράει πόσες φορές έχει πατηθεί το κουμπί από τον χρήστη. Στο πρώτο πάτημα (*switch\_cnt=1*), ο ανεμιστήρας ενεργοποιείται, στο δεύτερο πάτημα (*switch\_cnt=2*), επιταχύνεται ο ρυθμός της λεπίδας, ενώ στο τρίτο πάτημα, ο ανεμιστήρας απενεργοποιείται (*switch\_cnt=3*).

Στην main(), με την εντολή *PORTD.DIR = PIN0\_bm | PIN1\_bm | PIN2\_bm* ορίζουμεότι θα χρησιμοποιήσουμε τα LED0, LED1 και LED2. Ακολουθεί η *while(switch\_cnt<3)* την οποία την χρειαζόμαστε ώστε το πρόγραμμά μας να τρέχει συνέχεια έως ότου πατηθεί το κουμπί τρεις φορές, χωρίς να μεσολαβήσει τιμή μικρότερη του κατωφλίου (χωρίς να ενεργοποιηθεί ο ADC). Αρχικοποιούμε σε 0 τις τιμές x, stop και *switch\_cnt*, ώστε αν ο ανεμιστήρας ξανά ξεκινήσει να λειτουργεί με το πάτημα του κουμπιού (switch), να έχουν τεθεί οι σωστές τιμές στις μεταβλητές αυτές για τη σωστή λειτουργία του προγράμματος (reset). Στη συνέχεια, με την εντολή *PORTD.OUT = PIN0\_bm | PIN1\_bm | PIN2\_bm* απενεργοποιούμε τα προαναφερθέντα LED. Με την εντολή *PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc* δίνουμε την δυνατότητα στο πρόγραμμα, μόλις ενεργοποιηθούν οι διακοπές (interrupts) με την εντολή *sei()*, και ο χρήστης πατήσει το PIN5 στον INTFLAGS, να μεταφερθεί στην *ISR(PORTF\_PORT\_vect)*.

Εάν πατηθεί το κουμπί, το πρόγραμμα μεταφέρεται στην *ISR(PORTF\_PORT\_vect)*, σηματοδοτώντας έτσι την εκκίνηση του ανεμιστήρα. Αρχικά σε αυτή την ISR, καθαρίζονται τα flags και τίθεται η μεταβλητή *x* στην τιμή 1 και η μεταβλητή *switch\_cnt* αυξάνεται κατά ένα, ώστε μόλις εκτελεστούν όλες οι εντολές της ISR, το πρόγραμμα να εξέλθει της *while(x==0)* της main(). Επίσης, η εντολή *TCA0.SPLIT.CTRLD = TCA\_SPLIT\_SPLITM\_bm*, θέτει τον TCA σε split mode. Ως αποτέλεσμα, ο TCA δεν είναι 16bit πλέον, αλλά δύο 8bit timer(HUNF, LUNF). Με την εντολή *TCA0.SPLIT.CTRLA=TCA\_SPLIT\_CLKSEL\_DIV1024\_gc*, θέτουμε το ρολόι, με prescaler ίσο με 1024. Επιπρόσθετα, θέτουμε τιμές για τα LPER, LCMP0, HPER και HCMP0. Ειδικότερα, οι LPER και HPER παίρνουν τιμές που αφορούν το χρόνο του εκάστοτε παλμού (περίοδος), ενώ οι LCMP0 και HCMP0 παίρνουν τιμές που αφορούν τον κύκλο λειτουργίας (duty cycle). Παρατηρούμε ότι το duty cycle για τα low bytes είναι 40%, δηλαδή η τιμή LPER ισούται με 20 ενώ η τιμή LCMP0 ισούται με 8. Στη συνέχεια, σε ότι αφορά τα high bytes, η περίοδος του κάθε παλμού εξαρτάται από τις πόσες φορές έχει πατηθεί το κουμπί. Ειδικότερα αν έχει πατηθεί μονάχα μία φορά το κουμπί (απλή φάση ενεργοποίησης, *switch\_cnt==1*) το HPER ισούται με 10 και το HCMP0 ισούται με 5, ενώ εάν έχει πατηθεί δύο φορές το κουμπί (γρηγορότερη στροφή λεπίδων, *switch\_cnt==2*), το HPER ισούται με 20 και το HCMP0 ισούται με 10. Και στις δύο περιπτώσεις το duty cycle ισούται με 50% . Μέσα στην else *if(switch\_cnt==2)* με την εντολή *TCA0.SPLIT.CTRLB = TCA\_SPLIT\_HCMP0EN\_bm | TCA\_SPLIT\_LCMP0EN\_bm* ο TCA ενεργοποιεί τα κανάλια σύγκρισης high και low (HUNF, LUNF). Με την εντολή *TCA0.SPLIT.INTCTRL |= TCA\_SPLIT\_HUNF\_bm* ενεργοποιούνται τα interrupts για ένα HUNF γεγονός, και αν γίνει το interrupt, το πρόγραμμα θα μεταφερθεί στην *ISR(TCA0\_HUNF\_vect)*. Τέλος, στην τρίτη περίπτωση (φάση απενεργοποίησης, *switch\_cnt==3*), κάνουμε disable τον TCA με την εντολή *TCA0.SPLIT.CTRLA = 0b00000000*.

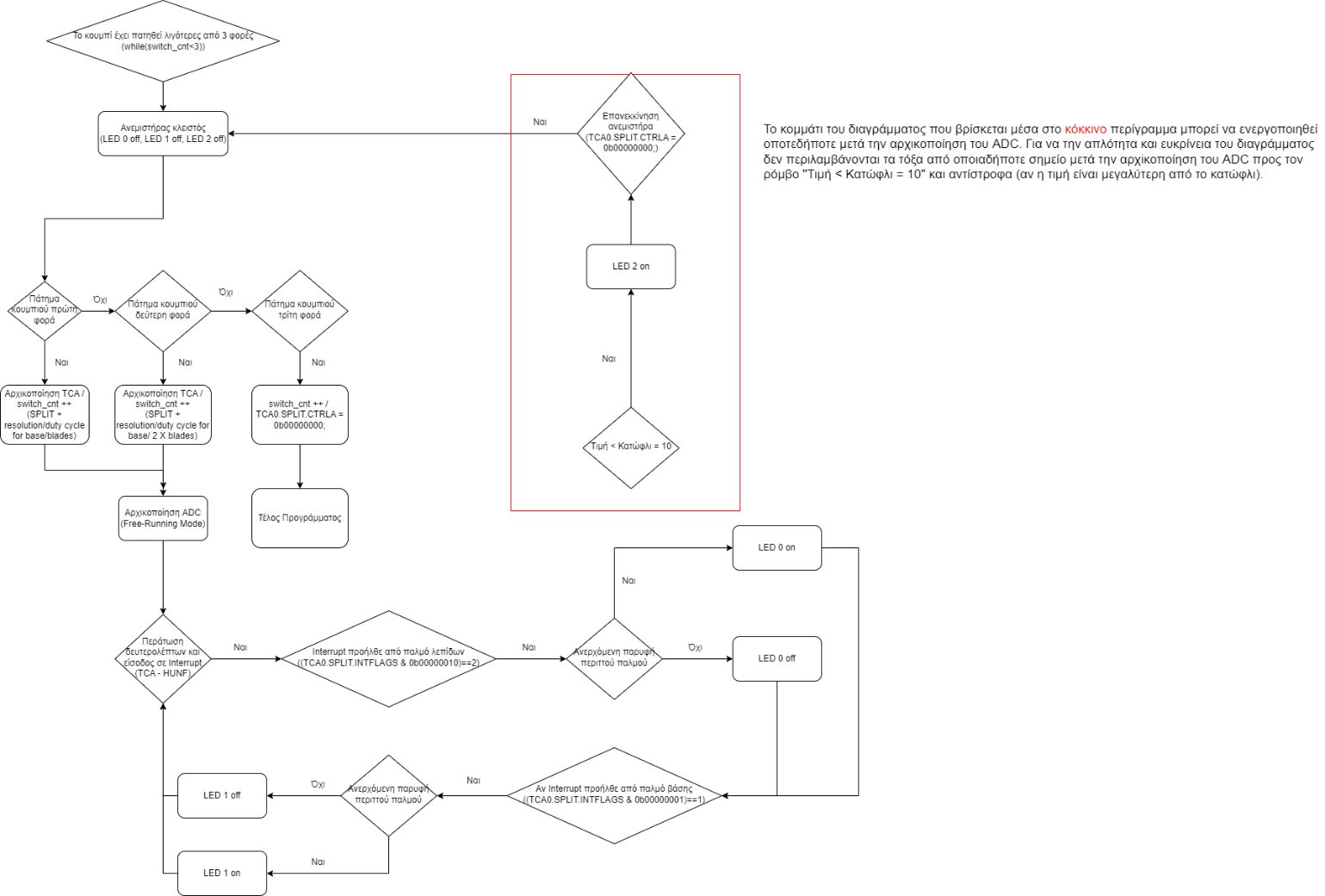
Πηγαίνοντας στη main(), αμέσως μετά την *while(x==0)* που εξήλθε το πρόγραμμα, ορίζουμε τον ADC με τις ακόλουθες εντολές :

* 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\_FREERUN\_bm* 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.WINLT |= 10* θέτουμε το κατώφλι στον καταχωρητή *ADC0.WINLT.*
* Με την εντολή *ADC0.CTRLE |= 0b00000001* ορίζουμε ότι θέλουμε να γίνεται interrupt όταν RESULT < WINLT.

Εφόσον ενεργοποιήσαμε τον ADC, ενεργοποιούμε τις διακοπές με την εντολή *sei()* και στη συνέχεια με την εντολή *ADC0.COMMAND |= ADC\_STCONV\_bm* ξεκινάει το conversion. To πρόγραμμά μας εισέρχεται στη *while (stop==0 && switch\_cnt!=3)*, και παραμένει εκεί έως ότου, είτε η μεταβλητή *stop* γίνει 1 είτε το *switch\_cnt* γίνει 3. Σε αυτό το σημείο υπάρχουν οι ακόλουθες περιπτώσεις :

1. Το κουμπί είναι πατημένο είτε μία (απλή φάση ενεργοποίησης, *switch\_cnt==1)* είτε δύο φορές (γρηγορότερη στροφή λεπίδων *switch\_cnt==2)* και RESULT < WINLT, επομένως το πρόγραμμά μας θα μεταφερθεί στην *ISR(ADC0\_WCOMP\_vect)* και θα παραμείνει εκεί έως ότου RESULT > WINLT. Σε αυτή την ISR, αρχικά, καθαρίζονται τα flags, και στη συνέχεια απενεργοποιούμε τα LED0 και LED1, που φανερώνουν περιστροφή λεπίδας και βάσης, και ενεργοποιούμε το LED2, που φανερώνει ότι ο ανεμιστήρας απενεργοποιήθηκε. Τα άνωθεν, γίνονται με τη χρήση των εντολών *PORTD.OUT |= PIN0\_bm | PIN1\_bm* και *PORTD.OUTCLR = PIN2\_bm* αντίστοιχα. Τέλος, ορίζουμε τη μεταβλητή *stop* στην τιμή 1 και κάνουμε disable τον TCA με την εντολή *TCA0.SPLIT.CTRLA = 0b00000000*. Έτσι το πρόγραμμα εξέρχεται της *while (stop==0 && switch\_cnt!=3)*, απενεργοποιούνται οι διακοπές με την εντολή *cli()*, γίνεται reset στις μεταβλητές *x*, *stop*, *switch\_cnt* και περιμένει να ξανά πατηθεί το κουμπί για να ξεκινήσει η λειτουργία του ανεμιστήρα.
2. Το κουμπί είναι πατημένο μία φορά (απλή φάση ενεργοποίησης, *switch\_cnt==1)*, η τιμή του RESULT είναι μεγαλύτερη του WINLT (RESULT>WINLT) και έτσι το πρόγραμμά μας δεν εισέρχεται στην ISR του ADC *[ISR(ADC0\_WCOMP\_vect)].* Καθώς βρίσκεται στην *ISR(PORTF\_PORT\_vect)*, και εφόσον η μεταβλητή *switch\_cnt* ισούται με 1, το πρόγραμμα έχει πάρει τις τιμές που βρίσκονται μέσα στην *if(switch\_cnt==1)*, με τον τρόπο που εξηγήσαμε παραπάνω. Επειδή είναι ενεργοποιημένες οι διακοπές και έχει οριστεί το ρολόι, το πρόγραμμά εισέρχεται στην *ISR(TCA0\_HUNF\_vect)*. Χάρη στην εντολή if *((TCA0.SPLIT.INTFLAGS & 0b00000010)==2)*, αξιοποιώντας τη δυνατότητα της μάσκας (mask) και του INTFLAGS του TCA, ελέγχουμε αν η είσοδος στο interrupt προέρχεται από τον παλμό που δίνει η κίνηση των λεπίδων. Αν ναι, αυξάνουμε κατά ένα τον μετρητή cnth με την εντολή *cnth++*, ώστε στη συνέχεια, να ελέγχουμε με την εντολή *if(cnth%2==1)* αν βρίσκεται σε άρτια ή περιττή ανερχόμενη παρυφή παλμού. Αυτός ο έλεγχος γίνεται, καθώς επιθυμούμε στο πρώτο Rising Edge του παλμού να ανάβει το LED0, στο δεύτερο Rising Edge να σβήνει το LED0, στο τρίτο Rising Edge να ανάβει το LED0 κ.ο.κ. Τo LED ενεργοποιείται και απενεργοποιείται με τις εντολές *PORTD.OUTCLR = PIN0\_bm* και *PORTD.OUT |= PIN0\_bm* αντίστοιχα. Ομοίως, στη συνέχεια, το πρόγραμμα ελέγχει αν η πράξη *TCA0.SPLIT.INTFLAGS & 0b00000001* δίνει αποτέλεσμα 1, κάτι που σημαίνει πως βρισκόμαστε κατά τη διάρκεια της κίνησης της βάσης. Με άλλα λόγια είναι η περίπτωση χρήσης του πρώτου ερωτήματος.
3. Το κουμπί είναι πατημένο δύο φορές (γρηγορότερη στροφή λεπίδων *switch\_cnt==2)*, η τιμή του RESULT είναι μεγαλύτερη του WINLT (RESULT>WINLT) και έτσι το πρόγραμμά μας δεν εισέρχεται στην ISR του ADC *[ISR(ADC0\_WCOMP\_vect)].* Καθώς το πρόγραμμά μας βρίσκεται στην *ISR(PORTF\_PORT\_vect)*, και εφόσον η μεταβλητή *switch\_cnt* ισούται με 2, το πρόγραμμα έχει πάρει τις τιμές που βρίσκονται μέσα στην *if(switch\_cnt==2)*, με τον τρόπο που εξηγήσαμε παραπάνω. Επειδή είναι ενεργοποιημένες οι διακοπές και έχει οριστεί το ρολόι, το πρόγραμμά εισέρχεται στην *ISR(TCA0\_HUNF\_vect)* και η συνέχεια είναι ανάλογη με αυτή της c) περίπτωσης.
4. Στην τελευταία περίπτωση, εάν το κουμπί είναι πατημένο τρεις φορές (φάση απενεργοποίησης, *switch\_cnt==3*) και η τιμή του RESULT είναι μεγαλύτερη του WINLT (RESULT>WINLT), τότε το πρόγραμμα εξέρχεται της *while (stop==0 && switch\_cnt!=3)*, και απενεργοποιεί τα interrupts με την εντολή *cli()*.

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.

Τιμές Ερωτήματος 1 και 2:

* Κυκλική κίνηση βάσης :

Για Τ = 2 msec έχουμε value = T\*ftimer = 0,002 (sec) \* 19.531,25 (Hz) = 39,0625

* Κυκλική κίνηση λεπίδων :

Για Τ = 1 msec έχουμε value = T\*ftimer = 0,001 (sec) \* 19.531,25 (Hz) = 19,53125

Τιμές Ερωτήματος 3:

* Κυκλική κίνηση βάσης :

Για Τ = 2 msec έχουμε value = T\*ftimer = 0,002 (sec) \* 19.531,25 (Hz) = 39,0625

* Κυκλική κίνηση λεπίδων :

Για Τ = 2 msec έχουμε value = T\*ftimer = 0,002 (sec) \* 19.531,25 (Hz) = 39,0625

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