ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 4

ΠΡΟΗΓΜΕΝΟΙ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΕΣ

Ομάδα

Ζήκος Σπύρος 1084581

Κυριακουλόπουλος Καλλίνικος 1084583

Αναλυτική Περιγραφή Κώδικα

Υπολογισμός PER και CMP0:

Fpwm\_ss(high) = = => 20\*1000 = 1024\*PER + 1024 =>

=> PER = 20000-1024 / 1024 => PER = 18,53125 ~ 18

Duty Cycle = 50% άρα CMP0 = PER \* 0.5 = 9

Παρακάτω δηλώνονται οι τιμές των κάτω και άνω κατωφλίων και οι τιμές των καταχωρητών PER και CMP0.

“

#define LTHRESHOLD 10;

#define HTHRESHOLD 20;

#define PER\_VALUE 18;

#define CMP0\_VALUE 9; // CMP0 = PER \* 0.5

“

Η μεταβλητή latest\_RES\_value χρησιμοποιείται για την αποθήκευση της τιμής του RES που έκανε τη διακοπή έτσι ώστε να μπορούμε αφού την αποθηκεύσουμε να την αλλάξουμε και να μην δημιουργηθεί πάλι διακοπή αφού ο ADC0 είναι σε free running mode.

“

int latest\_RES\_value; // Store RES value.

“

Η μεταβλητή temp χρησιμοποιείται για την αποθήκευση της τιμής του κάτω κατωφλίου για να γίνει η αφαίρεση με το RES και να υπολογιστεί ο χρόνος που θα μετρήσει ο timer.

“

int temp; // Temporary var to store LTHRESHOLD value.

“

Η μεταβλητή counter αυξάνεται κάθε φορά που εκτελείται ο ISR για overflow. Όταν η τιμή της γίνει 3 σημαίνει ότι έχουν γίνει 4 αναβοσβησίματα του led2 και θα πρέπει να τερματίσει η λειτουργία του TCA0 (PWM).

“

int counter = 0; // Counter for the rising edges.

“

int main()

Στη main function γίνεται η αρχικοποίηση των κατευθύνσεων των leds και αρχικοποιούνται όλα σβηστά. Στη συνέχεια ενεργοποιούνται οι διακοπές για τους διακόπτες 5 και 6 του PORTF και τέλος καλούνται οι συναρτήσεις init\_TCA0 και init\_ADC0 για να αρχικοποιήσουν τον TCA0 και τον ADC0 αντίστοιχα.

void init\_ADC0()

Με αυτή τη συνάρτηση γίνεται η αρχικοποίηση του ADC0. Ο ΑDC0 θα λειτουργεί σε free running mode με τιμές των 8bits. Επίσης ορίζουμε 2 thresholds high και low με τιμές 20 και 10 αντίστοιχα (δική μας επιλογή) και ενεργοποιούμε τις διακοπές. Θα γίνεται διακοπή όταν η τιμή του RES είναι μεγαλύτερη από το άνω κατώφλι και μικρότερη από το κάτω κατώφλι. Τέλος δίνουμε την εντολή στον ADC0 να ξεκινήσει τις μετατροπές, θα λαμβάνει συνεχόμενες τιμές όχι μια φορά μόνο.

void init\_TCA0()

Ο TCA0 χρησιμοποιείται για να κάνουμε 2 λειτουργίες, timer και PWM. Σε αυτές τις δύο περιπτώσεις χρήσης έχουμε 2 κοινές εντολές αρχικοποίησης του TCA0. Η πρώτη είναι ο ορισμός του clock division σε 1024 και η δεύτερη η ενεργοποίηση του TCA0. Η συνάρτηση καλείται κάθε φορά που θα χρειαστεί ο TCA0 είτε ως timer είτε ως PWM.

void init\_PWM()

Για τη δημιουργία PWM ορίζουμε τις τιμές των PER (top value) και CMP0 (duty cycle). Επίσης θέτουμε τον CTRLB σε single slope και ενεργοποιούμε την διακοπή για overflow.

void init\_timer()

Με τη συνάρτηση αρχικοποιούμε τον TCA0 να λειτουργήσει σαν timer. Αρχικά απενεργοποιείται ο ADC0 γιατί δεν θέλουμε να προκαλεί διακοπές και στην συνέχεια καλείται η συνάρτηση init\_TCA0. Επιπλέον θα χρειαστεί να καθαρίσουμε την τιμή του CNT να θέσουμε το mode σε normal και να υπολογίσουμε την τιμή που θα πάρει ο CMP0, μέχρι την τιμή αυτή θα μετρήσει ο CNT για να κάνει διακοπή. Η τιμή που θα λάβει ο CMP0 υπολογίζεται από την αφαίρεση του κάτω κατωφλίου μείον την τιμή του RES που δημιούργησε τη διακοπή στον ADC0. Για να έχουμε διακοπή στον TCA0 όταν ο CNT φτάσει την τιμή του CMP0 θα πρέπει να ενεργοποιηθεί η διακοπή για αυτή την περίπτωση.

void restart\_ADC0()

Η συνάρτηση χρησιμοποιείται κάθε φορά που επιστρέφουμε από τη λειτουργία του TCA0 (timer ή PWM) και επειδή ο ADC0 έχει απενεργοποιηθεί πρέπει να ξαναρχικοποιηθεί. Επομένως τίθεται σε freerunning mode και resolution 8 bits και ξεκινάει τις μετατροπές.

ISR(ADC0\_WCOMP\_vect)

Η συνάρτηση αυτή διαχειρίζεται τα interrupts του ADC. Αρχικά, αποθηκεύουμε την τιμή του RES σε μία μεταβλητή επειδή η τιμή του RES μπορεί να μεταβληθεί. Μετά, καθαρίζουμε τα intflags και ύστερα σβήνουμε όλα τα LED επειδή δεν μπορούν τα LED0 και LED1 να είναι αναμένα ταυτόχρονα (επειδή το ένα δηλώνει χαμηλή υγρασία και το άλλο υψηλή υγρασία). Ύστερα, ανάλογα με το κατώφλι (high ή low) το οποίο ξεπεράστηκε και προκάλεσε το interrupt ανάβουμε το LED1 ή το LED0 αντίστοιχα.

ISR(PORTF\_PORT\_vect)

Η συνάρτηση αυτή διαχειρίζεται τα interrupts των switches 5 και 6 του PORTF. Στην αρχή αποθηκεύουμε το intflags στην μεταβλητή y.

* Αν το interrupt το προκάλεσε το switch 5 και το LED0 είναι ανοιχτό, τότε καλούμε την init\_timer()
* Αν το interrupt το προκάλεσε το switch 6 και το LED1 είναι ανοιχτό, τότε καλούμε την init\_PWM().
* Αλλιώς αναβοσβήνουμε τα LEDs 0,1,2 για να δηλώσουμε ότι έγινε λάθος

Τέλος, καθαρίζουμε τα intflags.

ISR(TCA0\_CMP0\_vect)

Η συνάρτηση αυτή διαχειρίζεται το interrupt του TCA0 που προκαλείται όταν τελειώνει η μέτρηση του timer. Αρχικά, απενεργοποιείται ο TCA0 και καθαρίζονται τα intflags. Μετά, σβήνουμε το LED0 και κάνουμε restart τον ADC0 ώστε να ξαναμπεί σε λειτουργία.

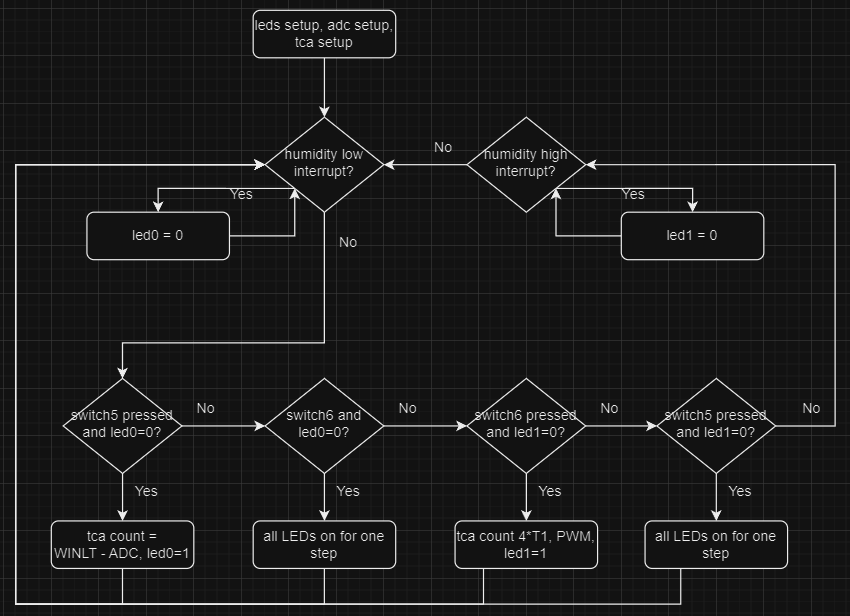
ISR(TCA0\_OVF\_vect)

Η συνάρτηση αυτή διαχειρίζεται το interrupt του TCA0 σε κάθε ανοδική ακμή του timer. Αρχικά, καθαρίζουμε τα intflags.

* Αν η τιμή του counter ειναι 3 αυτό σημαίνει ότι έχει μετρήσει ήδη 4 ανοδικές ακμές και γιαυτό απενεργοποιούμε τον TCA0, σβήνουμε τα LED 1 και 2 και θέτουμε τον counter ίσο με 0.
* Αλλιώς, κάνουμε toggle το LED2.

Τέλος, αυξάνουμε τον counter κατά 1 για να μετρήσουμε 4 ανοδικές ακμές.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ



Using: <https://app.diagrams.net>

Κώδικας

#include <avr/io.h>

#include <avr/interrupt.h>

#define LTHRESHOLD 10;

#define HTHRESHOLD 20;

#define PER\_VALUE 18;

#define CMP0\_VALUE 9; // CMP0 = PER \* 0.5

void init\_ADC0();

void init\_TCA0();

void init\_timer();

void init\_PWM();

void restart\_ADC0();

int latest\_RES\_value; // Store RES value.

int temp; // Temporary var to store LTHRESHOLD value.

int counter = 0; // Counter for the rising edges.

int main(){

PORTD.DIR |= 0b00000111; //PIN is output

PORTD.OUT |= 0b00000111; //LEDs are off

PORTF.PIN5CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc; // Enable switch5 interrupt.

PORTF.PIN6CTRL |= PORT\_PULLUPEN\_bm | PORT\_ISC\_BOTHEDGES\_gc; // Enable switch6 interrupt.

init\_ADC0(); // Initialize ADC0

while(1){

sei();

}

}

void init\_ADC0(){

//initialize the ADC for free running mode

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

ADC0.CTRLA |= ADC\_RESSEL\_8BIT\_gc; //8-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 LOW = 10

ADC0.WINHT = 20; //Set threshold HIGH = 20

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

ADC0.CTRLE = 0x4; // RESULT > WINHT or RESULT < WINLT

sei();

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

}

void init\_TCA0(){

// This part of code is used both in init\_PWM() and init\_timer()

TCA0.SINGLE.CTRLA = TCA\_SINGLE\_CLKSEL\_DIV1024\_gc; // Clock\_Division=1024

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

}

void init\_PWM(){

ADC0.CTRLA = 0; // Disable ADC

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

TCA0.SINGLE.PER = PER\_VALUE; // Select the top value

TCA0.SINGLE.CMP0 = CMP0\_VALUE; // Select the duty cycle

TCA0.SINGLE.CTRLB |= TCA\_SINGLE\_WGMODE\_SINGLESLOPE\_gc; // select Single\_Slope\_PWM

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_OVF\_bm; // enable interrupt Overflow

init\_TCA0();

}

void init\_timer(){

ADC0.CTRLA = 0; // Disable ADC

// Start timer

temp = LTHRESHOLD; // store LTHRESHOLD value to temp

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

TCA0.SINGLE.CTRLB = 0; // Normal Mode

// CMP0 = LTHREHOLD - latest\_RES\_value

TCA0.SINGLE.CMP0 = temp - latest\_RES\_value; // When CMP0 reaches this value -> interrupt

TCA0.SINGLE.INTCTRL = TCA\_SINGLE\_CMP0\_bm; // Interrupt Enable (CMP0)

init\_TCA0();

}

void restart\_ADC0(){

//start ADC

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

ADC0.CTRLA |= ADC\_RESSEL\_8BIT\_gc; // 8-bit resolution

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

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

}

ISR(ADC0\_WCOMP\_vect){

cli();

latest\_RES\_value = ADC0.RES;

int intflags = ADC0.INTFLAGS;

ADC0.INTFLAGS = intflags;

PORTD.OUT |= 0b00000111; // LEDs are off

if(latest\_RES\_value > ADC0.WINHT){

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

}else if(latest\_RES\_value < ADC0.WINLT){

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

}

sei();

}

ISR(PORTF\_PORT\_vect){

cli();

int y = PORTF.INTFLAGS;

if (((y & 0b00100000) == 0b00100000) && (PORTD.OUT == 0b00000110)){ // pin5->led0

init\_timer();

} else if (((y & 0b01000000) == 0b01000000) && (PORTD.OUT == 0b00000101)){ // pin6->led1

init\_PWM();

} else {

PORTD.OUTCLR = 0b00000111; // LED is on

cli(); // adds delay for debugging

PORTD.OUT |= 0b00000111; // LED is off

}

PORTF.INTFLAGS=y;

sei();

}

ISR(TCA0\_CMP0\_vect){

cli();

TCA0.SINGLE.CTRLA = 0; // Disable TCA0

int intflags = TCA0.SINGLE.INTFLAGS;

TCA0.SINGLE.INTFLAGS=intflags;

PORTD.OUT |= 0b00000001; // LED0 is off

restart\_ADC0(); // restart ADC

sei();

}

ISR(TCA0\_OVF\_vect){

cli();

int intflags = TCA0.SINGLE.INTFLAGS;

TCA0.SINGLE.INTFLAGS = intflags;

if (counter == 3){

TCA0.SINGLE.CTRLA = 0; // Disable TCA0

PORTD.OUT |= PIN2\_bm | PIN1\_bm; // turn off LED1&2

counter = 0;

restart\_ADC0(); // restart ADC

} else {

if ((PORTD.OUT & 0b00000100) == 0b00000100){

PORTD.OUTCLR = PIN2\_bm; // LED2 is on

} else {

PORTD.OUT |= PIN2\_bm; // LED2 is off

}

counter++; // rising edge

}

sei();

}