**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**Facultatea de Sisteme Informatice și Securitate Cibernetică**

**Departamentul de Calculatoare și Securitate Cibernetică**



***Utilizare senzor digital vibrații***

***Platforma de dezvoltare frdm-kl25z***

Std. sg. maj. Mitran Luca

Std. sg. maj. Enache Ștefan

Grupa C114D

**București**

**2024**

CUPRINS

[1. Scopul proiectului 3](#_Toc156391782)

[2. Prezentarea senzorilor din proiect 3](#_Toc156391783)

[3. Conectarea senzorilor la placa de dezvoltare 5](#_Toc156391784)

[4. Descriere program 7](#_Toc156391785)

[4.1 Funcția main 8](#_Toc156391786)

[4.2 Module 8](#_Toc156391787)

[GPIO (Secvență de led-uri) 8](#_Toc156391788)

[TPM(Rotire SG90) 13](#_Toc156391789)

[UART 17](#_Toc156391790)

[ADC 20](#_Toc156391791)

[5. Testarea aplicației 27](#_Toc156391792)

[6. Probleme întâmpinate 28](#_Toc156391793)

[7. Bibliografie 29](#_Toc156391794)

# Scopul proiectului

Sursele proiectului sunt disponibile pe următorul link:

<https://github.com/stefanenache3/FRDM-KL25Z-ROTs_SG90>

Prezentarea proiectului este disponibilă pe următorul link:

<https://www.youtube.com/watch?v=uo1cCHyWfSs>

Scopul principal proiectului este de a agrega la nivelul unui program datele primite de la cei doi senzori: un senzor de rotație și unu senzor de șoc. Se vor implementa două funcționalități în baza datelor citite în timp real:

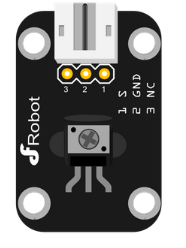
* Gradul de rotație al primului senzor se va reprezenta prin poziționarea elicei unui Micro Servo Motor(SG90) în trei poziții.Datele citite de la senzor vor fi mapate în trei intervale, reprezentate la nivelul servo motorului prin rotirea la 0°, 90°, respectiv 180°.
* Se vor analiza valorile citite de senzorul de șoc și se va implementa o funționalitate care să determine anomalii în aceste date, asociate în acest caz unei mișcări seismice.

Aplicația va avea un Graphical User Interface care să reprezinte prin două grafice valorile citite de la senzori.La detectarea unei anomalii seismice, aplicația va emite o alertă sonoră și va afișa un mesaj de avertizare. Punctele de pe graficul cu date provenite de la senzorul de rotație vor fi colorate diferit, în funcție de unghiul citit de la senzor. Astfel se definesc trei intervale colorate diferit, 0°-60°, 60°-120°,120°-180°.

# Prezentarea senzorilor din proiect

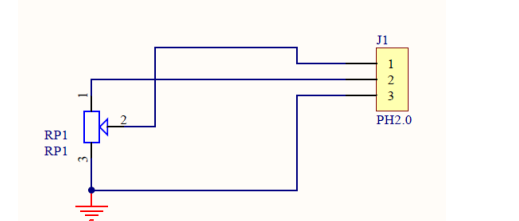
* Senzorul analog de rotație DFR0054

Senzorul DFRobot Rotation Sensor V2 este un senzor care poate fi rotit până la 300 de grade, acesta este bazat pe un potențiometru simplu.Acest senzor este folosit pentru a modifica tensiunea aplicată circuitului.Este un senzor analogic, astfel vom utiliza modulul ADC pentru a putea citi unghiul de rotație.

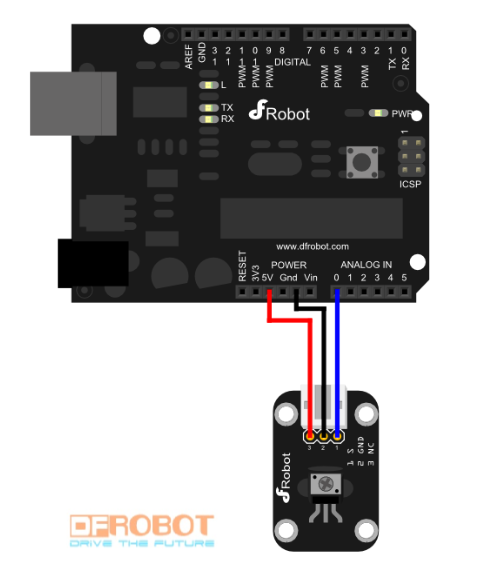
  Pinii senzorului:

Figură 1 - Pinii senzorului de rotație

* Signal Output
* GND
* Power



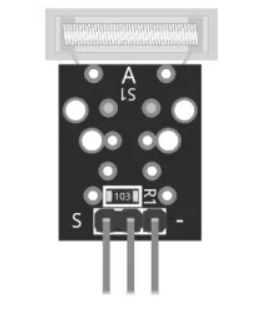
Figură 2 - Schemă electrică senzor de rotație



Figură 3 - Diagramă de conexiune senzor

* Modulul KY-031

Modulul KY-031 este un senzor de vibrații care trimite semnale auntic cand se detectează batăi.



Figură 4 - Senzor de vibrații

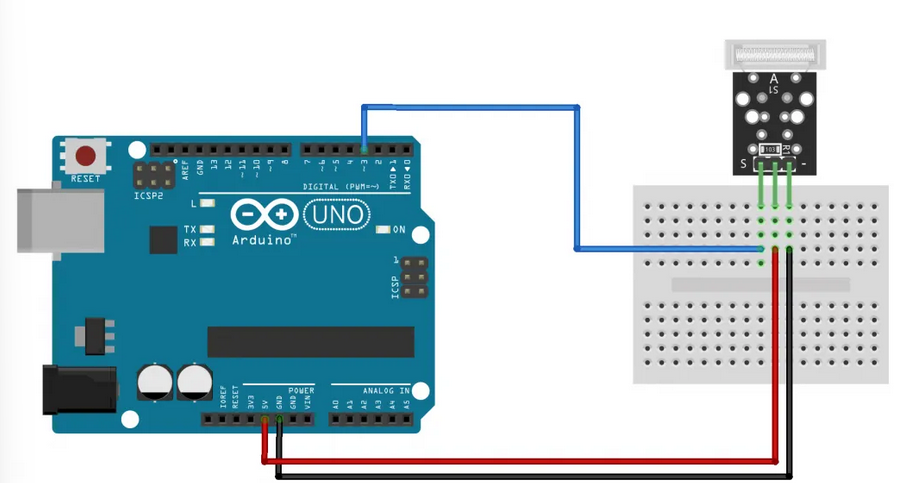
Senzorul este format dintr-un senzor cu arc, un rezistor de 10 kΩ și 3 pini tată. Arcul emite un semnal ridicat atunci când este detectată o vibrație.

Pinii senzorului:

- Signal Output

- GND

- Power

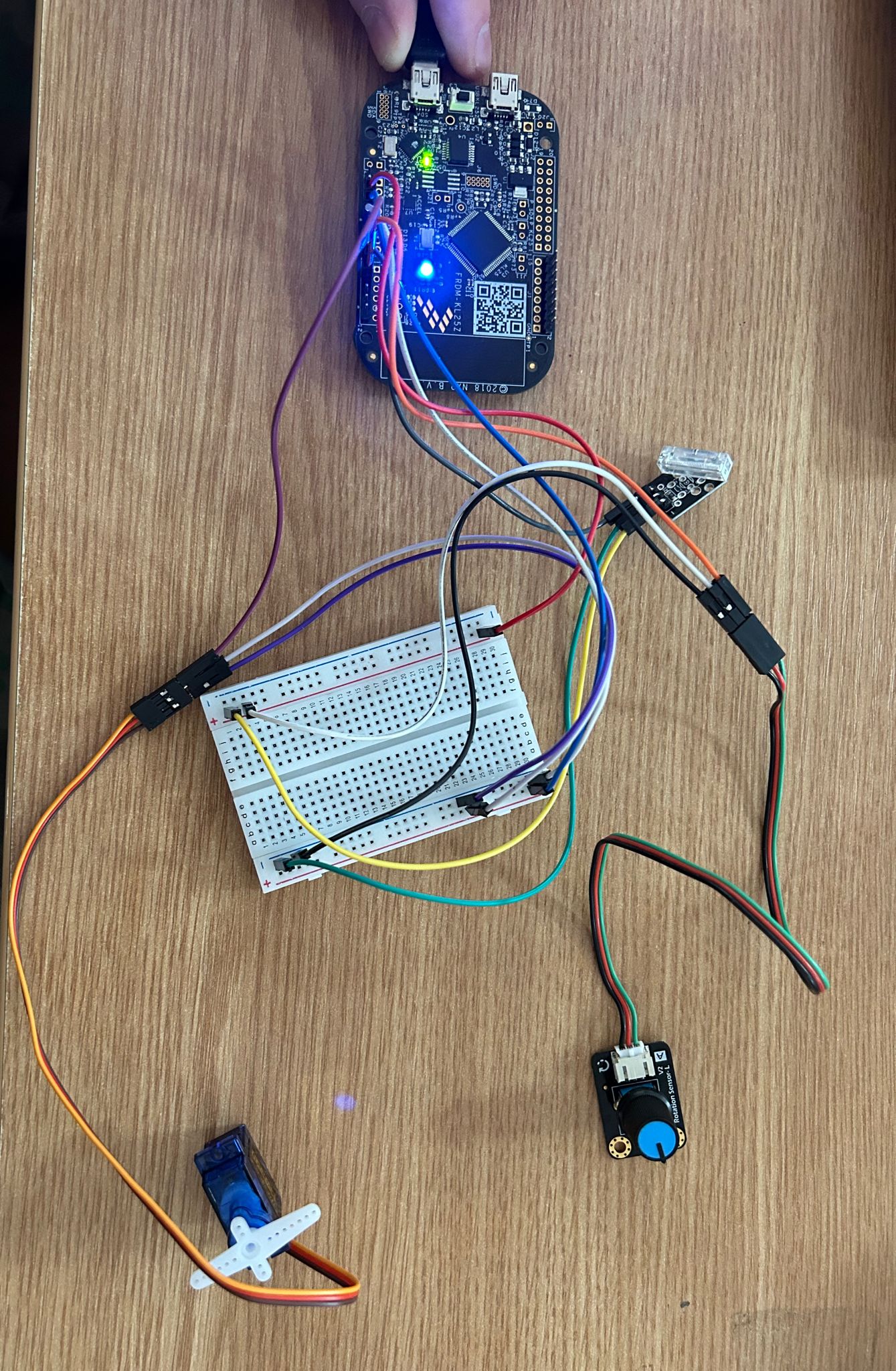


Figură 5 - Diagramă de conexiune senzor vibrații

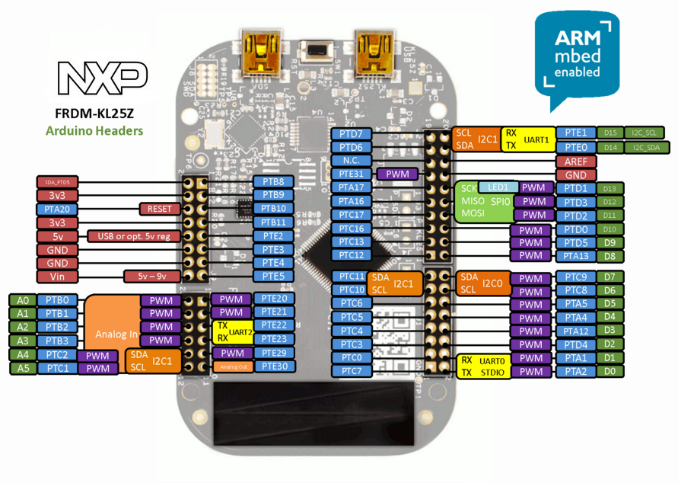
# Conectarea senzorilor la placa de dezvoltare

Vom conecta senzorii astfel:

* Pin-ul de semnal pentru SG90 : PTB2
* Senzorul de rotație : PTB0
* Senzorul de vibrații : PTB1

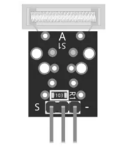
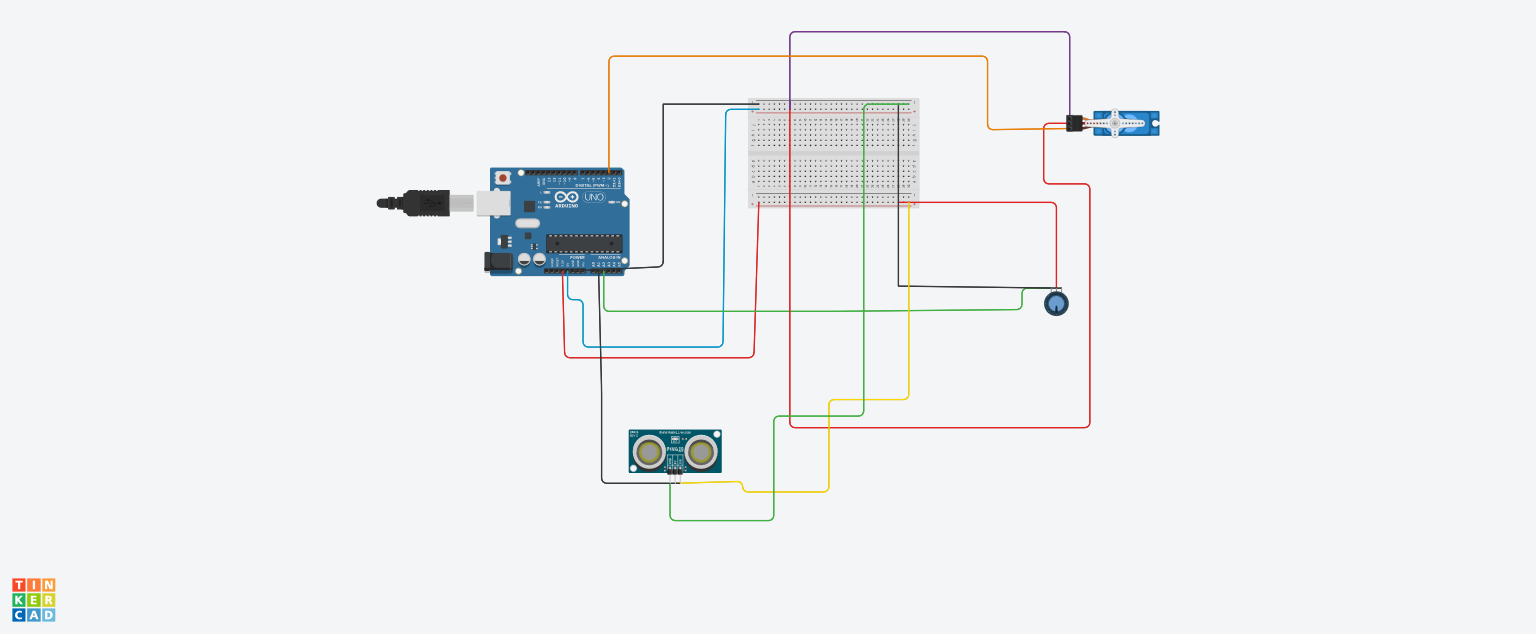


Figură 6 - Conectare senzori



Figură 7 - Pini platformă de dezvoltare

În figura de mai jos este prezentată schema in Tinkercad a proiectului.De menționat este că platforma de dezvoltare de la NXP nu era disponibilă in Tinkercard, așa că am înlocuit-o cu o plăcuță de Arduino UNO.



# Descriere program

Figură 8 - Schema proiectului

## Funcția main

În funcția main, se apelează toate funcțiile de inițializare ale modulelor necesare proiectului. Se initializează baudrate-ul pentru UART la valoarea 38400. Toate apelurile de funcții necesare rulării programului sunt preluate de către NVIC prin intermediul mecanismului de întreruperi. Întreruperile periodice din PIT acționează citirea datelor de la senzori, acționarea servomotorului și aprinderea secvenței de LED-uri în ordinea dictată prin intermediul interfeței grafice.

1. #include "MKL25Z4.h"

2. #include "Sequencer.h"

3. #include "pit.h"

4. #include "uart.h"

5. #include "adc.h"

6. #include "Servo.h"

7. int main(void){

8. init\_tpm2();

9. UART0\_Initialize(38400);

10. ADC0\_Init();

11.

12. led\_init();

13. PIT\_Init();

14.

15.

16.

17. while(1){

18.

19. }

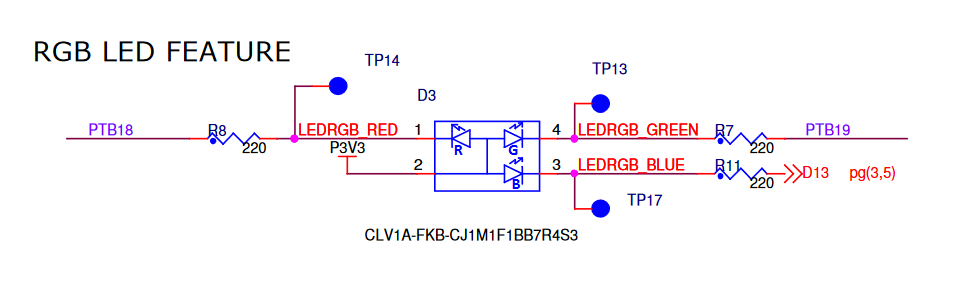
20. return 0;

21. }

## Module

### GPIO (Secvență de led-uri)

Inițializările pentru modulul GPIO se fac la nivelul surse Sequencer.c Aplicația noastră trebuie să afișeze o secvență de culori prin controlarea canalelor de culoare LED de pe placa de dezvoltare.Pentru a aprinde LED-ul se utilizează pinii de GPIO.Se vor confoigura pinii GPIO pentru a funcționa ca ieșiri.Fiecare LED va avea un pin corespunzător a cărui ieșire trebuie să o controlăm pentru a aprinde și a stinge LED-ul.



Configurarea pinilor de GPIO pentru a putea realiza secvența de led-uri se face în fișierul Sequencer.c

**Sequencer.h:**

1. #include "MKL25Z4.h"

2.

3. #define RED\_LED\_PIN 18

4. #define GREEN\_LED\_PIN 19

5. #define BLUE\_LED\_PIN 1

6.

7. void delay(long int n);

8. void led\_init();

9. void sequence\_leds(uint32\_t state);

**Sequencer.c**

1. void led\_init(){

2.

3. SIM->SCGC5 |= SIM\_SCGC5\_PORTB\_MASK | SIM\_SCGC5\_PORTD\_MASK ;

4.

5. PORTB->PCR[RED\_LED\_PIN]=~PORT\_PCR\_MUX\_MASK;

6. PORTB->PCR[RED\_LED\_PIN] |= PORT\_PCR\_MUX(1);

7.

8. PORTB->PCR[GREEN\_LED\_PIN]=~PORT\_PCR\_MUX\_MASK;

9. PORTB->PCR[GREEN\_LED\_PIN] |= PORT\_PCR\_MUX(1);

10.

11. PORTD->PCR[BLUE\_LED\_PIN] = ~PORT\_PCR\_MUX\_MASK;

12. PORTD->PCR[BLUE\_LED\_PIN] |= PORT\_PCR\_MUX(1);

13.

14. PTB->PDDR |= (1 << RED\_LED\_PIN);

15. PTB->PDDR |= (1 << GREEN\_LED\_PIN);

16. PTD->PDDR |= (1 << BLUE\_LED\_PIN);

17.

18. PTB->PSOR = (1 << RED\_LED\_PIN);

19. PTB->PSOR = (1 << GREEN\_LED\_PIN);

20. PTD->PSOR = (1 << BLUE\_LED\_PIN);

21. }

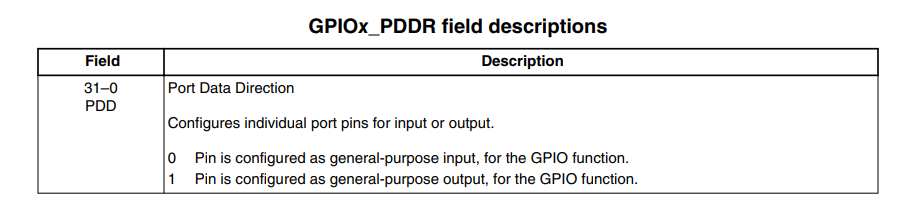
Pe linia #3 activăm semnalul de ceas pentru porturile PTB și PTD, acum le vom putea folosi pentru modulul de GPIO.

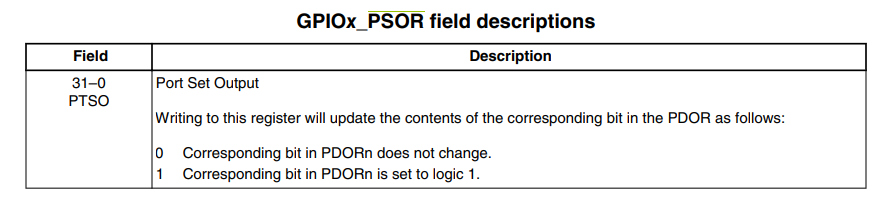
Din schema electrică de mai sus am definit valorile pentru pinurile corespunzătoare fiecărui led.

* Red roșu : PTB18
* Led verde : PTB19
* Led albastru : PTD3

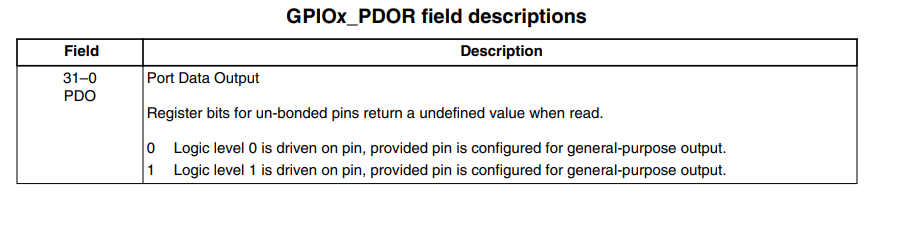
In următoarea secvență facem clear la registrul de control pentru PIN și setăm alternativa 1 de funcționare.

Între liniile #14 și #16 setăm pinii pentru general-purpose output.





În ultimele 3 linii setăm registrul PSOR cu 1 pe valorile corespunzătoare pinilor utilizați, asta produce output high pentru cei trei pini, ceea ce face ca led-urile să se stingă.



1. void sequence\_leds(uint32\_t state){

2.

3. switch(state){

4.

5. case 0:

6. PTB->PCOR=(1<<RED\_LED\_PIN);

7. PTB->PCOR=(1<<GREEN\_LED\_PIN);

8. PTD->PCOR=(1<<BLUE\_LED\_PIN);

9. break;

10.

11.

12. case 1:

13. PTB->PCOR = (1 << GREEN\_LED\_PIN);

14. PTB->PSOR = (1 << RED\_LED\_PIN);

15. PTD->PSOR = (1 << BLUE\_LED\_PIN);

16. break;

17. case 2:

18. PTB->PSOR = (1 << GREEN\_LED\_PIN);

19. PTB->PSOR = (1 << RED\_LED\_PIN);

20. PTD->PCOR = (1 << BLUE\_LED\_PIN);

21. break;

22. case 3:

23. PTB->PCOR = (1 << GREEN\_LED\_PIN);

24. PTB->PCOR = (1 << RED\_LED\_PIN);

25. PTD->PSOR = (1 << BLUE\_LED\_PIN);

26. break;

27. }

28. }

29.

În funcția sequence\_leds putem seta una din stările secvenței de leduri din enunț. Prin setarea pe 1 a bit-ului corespondent din PCOR setăm output-ul pe 0, astfel led-ul respectiv se aprinde.

Pentru a obține anumite culori trebuie sa pornim în combinație led-urile, punem output pe 0 pentru cele pe care vrem să le pornim și 1 pentru cele pe care vrem să le oprim.

Stări:

* Starea 0: Led Alb

Aprindem toate led-urile

1. PTB->PCOR=(1<<RED\_LED\_PIN);

2. PTB->PCOR=(1<<GREEN\_LED\_PIN);

3. PTD->PCOR=(1<<BLUE\_LED\_PIN);

4.

* Starea 1:Led verde

Aprindem doar led-ul verde

1. PTB->PCOR = (1 << GREEN\_LED\_PIN);

2. PTB->PSOR = (1 << RED\_LED\_PIN);

3. PTD->PSOR = (1 << BLUE\_LED\_PIN);

* Starea 2:Led albastru

Aprindem led-ul albastru

1. PTB->PSOR = (1 << GREEN\_LED\_PIN);

2. PTB->PSOR = (1 << RED\_LED\_PIN);

3. PTD->PCOR = (1 << BLUE\_LED\_PIN);

4.

* Starea 3:Led galben

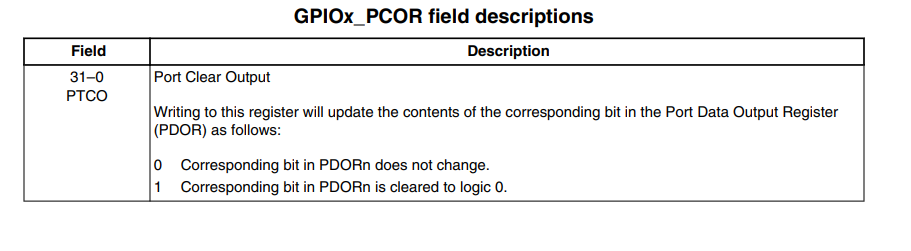
Aprindem led-ul verde și pe cel roșu

1. PTB->PCOR = (1 << GREEN\_LED\_PIN);

2. PTB->PCOR = (1 << RED\_LED\_PIN);

3. PTD->PSOR = (1 << BLUE\_LED\_PIN);

4.



### TPM(Rotire SG90)

Scopul este să generăm un semnal PWM pe care să îl utilizăm ca semnal de control pentru a roti elicea servomotorului SG90. Perioada standard a semnalului pentru a control SG90 trebuie să fie de **20ms.** Rotirea servomotorului este influențată de perioada în care semnalul va fi 1 într-un ciclu.

**Servo.h**

1. #include "MKL25Z4.h"

2. #define PG90\_PWM\_PERIOD (20) //20ms

3. #define CLOCK\_FREQ (375) //375KHz

4.

5.

6. #define DUTY\_CYCLE\_0 (365)

7. #define DUTY\_CYCLE\_90 (195)

8. #define DUTY\_CYCLE\_180 (77) //2ms

9.

10.

11. #define SERVO\_PIN (2) //PTB2

12. void init\_tpm2();

13. void setPG90\_angle(uint32\_t angle);

14.

**Inițializare modul TPM**

1. void init\_tpm2(){

2.

3. //PTB2: alternativa de functionare pentru perifericul TPM: TPM2\_CH0

4. // TPM2 Clock Gate Control - Clock enabled

5. SIM->SCGC5 |= SIM\_SCGC5\_PORTB\_MASK;

6. PORTB->PCR[SERVO\_PIN] |= PORT\_PCR\_MUX(3);

7.

8.

9. SIM->SCGC6 |= SIM\_SCGC6\_TPM2(1);

10. // Selectam sursa de ceas pentru TPM: MCGFLLCLK -> 48 MHz max

11. SIM->SOPT2 |= SIM\_SOPT2\_TPMSRC(1);

12. // Setam prescale factor : 111 -> divide by 128

13. // 48 MHz / 128 = 375 kHz

14. TPM2->SC |= TPM\_SC\_PS(7);

15.

16. TPM2->SC |= TPM\_SC\_CPWMS(0);

17.

18. // LPTPM counter increments on every LPTPM counter clock

19. TPM2->SC |= TPM\_SC\_CMOD(1);

20.

21. TPM2->MOD = PG90\_PWM\_PERIOD \* CLOCK\_FREQ;

22. //Center-aligned PWM/high-true pulses

23. TPM2->CONTROLS[0].CnSC = TPM\_CnSC\_MSB\_MASK | TPM\_CnSC\_ELSB\_MASK;

24.

25. }

26.

În liniile #5 și #6 activăm semnalul de ceas pentru PORTB și selectăm alternativa de funcționare 3 pentru pinul PTB2 ( FTM2\_CH0).Așadar, se trimite semnalul PWM pe PTB2, pin la care vom conecta servomotorul.

1. SIM->SCGC6 |= SIM\_SCGC6\_TPM2(1);

2. // Selectam sursa de ceas pentru TPM: MCGFLLCLK -> 48 MHz max

3. SIM->SOPT2 |= SIM\_SOPT2\_TPMSRC(1);

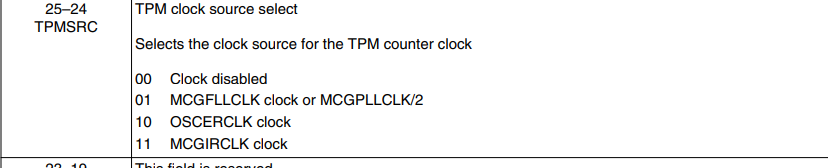
4. // Setam prescale factor : 111 -> divide by 128

5. // 48 MHz / 128 = 375 kHz

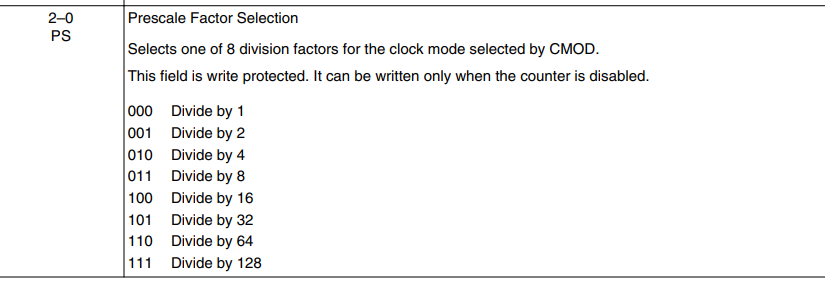
6. TPM2->SC |= TPM\_SC\_PS(7);

7.

În registrul SOPT selectăm sursa de ceas pentru modulul tpm.



Această sursă de ceas are o frecvență maximă de 48 MHz.

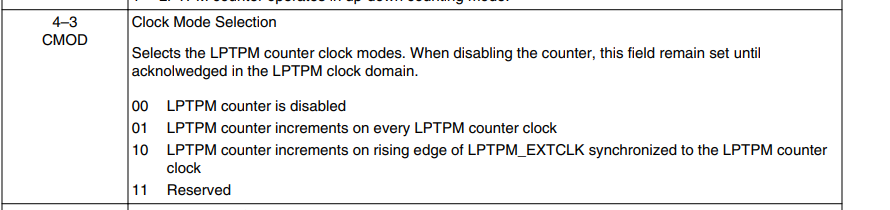


În registrul de status și control setăm bitul PS pe 7 pentru a divide frecvența de ceas cu 128.O frecvență de ceas mai mică ne permite să avem range-uri mai mari de timp.

1. 1. TPM2->SC |= TPM\_SC\_CPWMS(0);

2. 2. // LPTPM counter increments on every LPTPM counter clock

3. 3. TPM2->SC |= TPM\_SC\_CMOD(1);

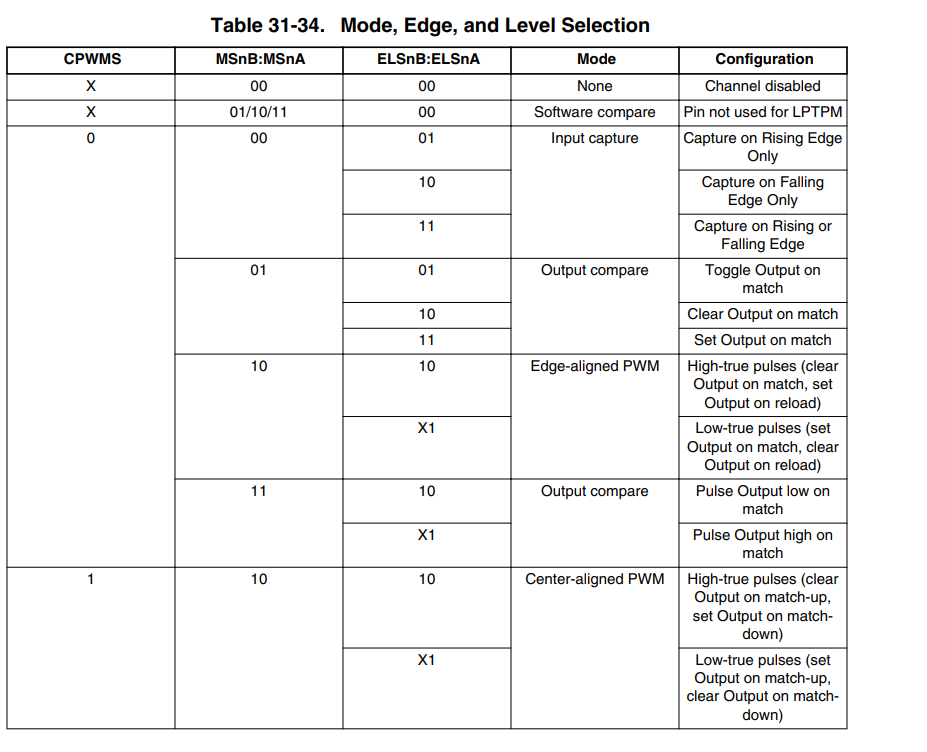
#1 Setăm counter-ul în up counting mode. 

1. TPM2->MOD = PG90\_PWM\_PERIOD \* CLOCK\_FREQ;

2. //Center-aligned PWM/high-true pulses

3. TPM2->CONTROLS[0].CnSC = TPM\_CnSC\_MSB\_MASK | TPM\_CnSC\_ELSB\_MASK;

Setăm valoarea până la care să numere numărătorul într-un ciclu de 20ms.

Setăm CPWMS pe 0, MSnb:MSnA 10 șo ELSnB:ELSnA 10

Folosim module Edge-aligned, ceea ce ne permite să modificăm precis poziția servomotorului, ajustând duty cycle-ul.

1. void setPG90\_angle(uint32\_t angle){

2.

3. uint32\_t dutyCycle;

4.

5. if(angle<90){

6. dutyCycle=DUTY\_CYCLE\_0;

7. }else if(angle<180){

8. dutyCycle=DUTY\_CYCLE\_90;

9. }else{

10. dutyCycle=DUTY\_CYCLE\_180;

11. }

12.

13. TPM2->CONTROLS[0].CnV=dutyCycle;

14. }

15.

Programul poate roti servomotorul în trei poziții, la unghiuri de 0, 90 si 180 de grade.Putem controla acest unghi prin ajustarea factorului de umplere. Am numit variabila dutyCycle, dar ea nu reprezintă chiar factorul de umplere, deoarece utilizând formula specifică (dutycycle\*T\*freq) și factorii de umplere din documentație nu se obținea rotația dorită, astfel ca am ajustat valoarea registrului CnV al canalului 0 până am obțiunut ungiurile dorite astfel:

|  |  |
| --- | --- |
| **CnV** | **Angle** |
| 365 | 0 grade |
| 195 | 90 grade |
| 77 | 180 grade |

### UART

Modulul UART este folosit în cadrul proiectului pentru trimiterea și primirea datelor de la și către MCU. Inițializarea acestui modul este prezentată în secventa următoare de cod:

1. void UART0\_Initialize(uint32\_t baud\_rate) {

2. //NU SE SETEAZA OSR IN C4

3. uint16\_t osr = 8;

4. uint16\_t sbr;

5.

6. SIM->SCGC4 = SIM->SCGC4 | SIM\_SCGC4\_UART0\_MASK;

7.

8. SIM->SCGC5 |= SIM\_SCGC5\_PORTA\_MASK;

9. UART0->C2 &= ~((UART0\_C2\_RE\_MASK) | (UART0\_C2\_TE\_MASK));

10.

11. SIM->SOPT2 |= SIM\_SOPT2\_UART0SRC(01);

12.

13. PORTA->PCR[1] = ~PORT\_PCR\_MUX\_MASK;

14. PORTA->PCR[1] = PORT\_PCR\_ISF\_MASK | PORT\_PCR\_MUX(2);

15. PORTA->PCR[2] = ~PORT\_PCR\_MUX\_MASK;

16. PORTA->PCR[2] = PORT\_PCR\_ISF\_MASK | PORT\_PCR\_MUX(2);

17.

18. sbr = (uint16\_t)((DEFAULT\_SYSTEM\_CLOCK)/(baud\_rate \* (osr)));

19. UART0->BDH &= UART0\_BDH\_SBR\_MASK;

20. UART0->BDH |= UART0\_BDH\_SBR(sbr>>8);

21. UART0->BDL = UART\_BDL\_SBR(sbr);

22. UART0->C4 &= UART0\_C4\_OSR(osr - 1);

23.

24. UART0->C1 = UART0\_C1\_M(0) | UART0\_C1\_PE(0);

25.

26. UART0->S2 |= UART0\_S2\_MSBF(0);

27.

28.

29. UART0->C2 |= UART0\_C2\_TIE(0) | UART0\_C2\_TCIE(0);

30. UART0->C2 |= UART0\_C2\_RIE\_MASK;

31.

32.

33. UART0->C2 |= ((UART\_C2\_RE\_MASK) | (UART\_C2\_TE\_MASK));

34.

35.

36. NVIC\_ClearPendingIRQ(UART0\_IRQn);

37. NVIC\_EnableIRQ(UART0\_IRQn);

38.

39. \_\_enable\_irq();

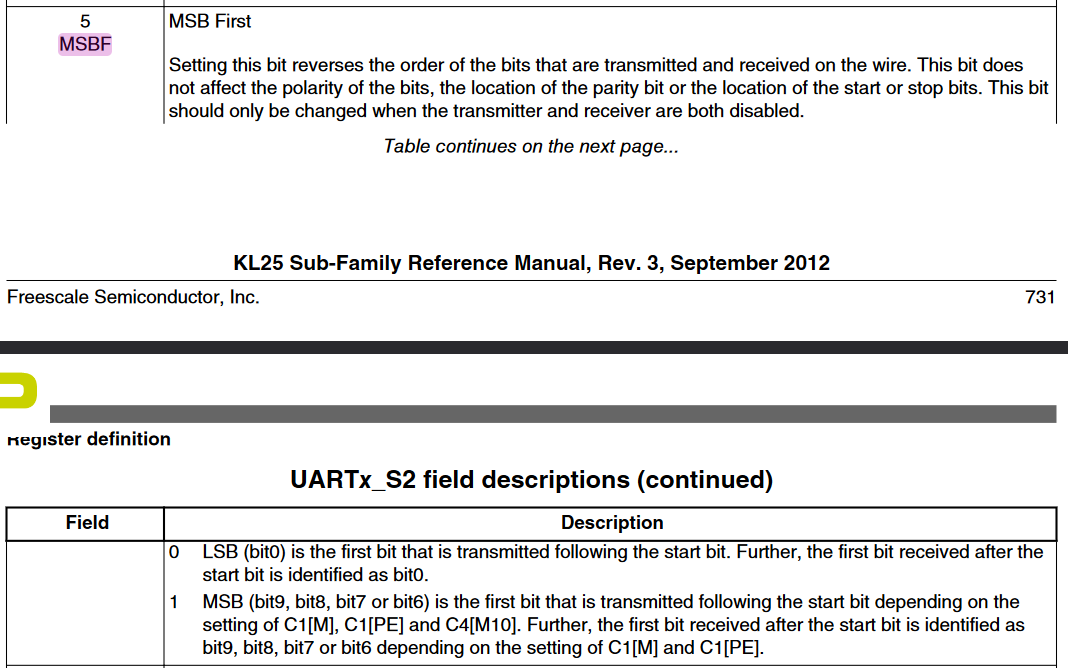
40. }

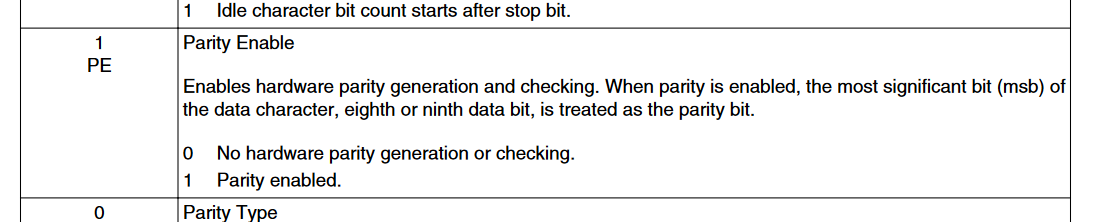
41.

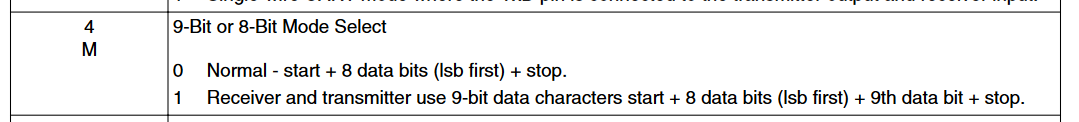
42.

Pentru satisfacerea cerinței de proiect de a avea control asupra modului în care se aprinde secvența de LED-uri am activat mecanismul de întreruperi pentru modulul UART. Portul folosit pentru receive este PTA1, iar cel pentru transmit este PTA2.

În initializarea modulului UART se seteaza urmatorii parametrii pentru comunicatii:

* Over sampling rate setat la valoarea 8. Aceasta se seteaza in registrul C4 pentru UART.
* Data frame order: LSB first setat în registrul UARTx\_S2
* Parity enable: este setat pe OFF, nu se trimit biți de paritate.



* 8-bit mode: selectarea trimiterii a 8 biti de date.
* Baud rate este setat din functia main în concordanta cu specificațiile furnizate proiectului: 38400.
* Setarea în UARTx\_C2 a pornirii întreruperii la apariția flagului RDRF, folosit pentru inversarea direcției de aprindere a LED-urilor.

#### Intrerupere UART

1. void UART0\_IRQHandler(void) {

2. if(UART0->S1 & UART0\_S1\_RDRF\_MASK) {

3. uint8\_t data=UART0->D;

4. if(data=='N')

5. {

6. led\_sequence\_direction=1;

7. }

8. if(data=='R')

9. {

10. led\_sequence\_direction=2;

11. }

12.

13. }

14. }

15.

La apariția datelor se verifica dacă s-a primit caracterul N sau R. În funcție de datele de intrare se seteaza variabila globala led\_sequence\_direction, care gestionează direcția de aprindere a LED-urilor din intermediul întreruperilor PIT.

#### UART Transmit

1. void UART0\_Transmit(uint8\_t data) {

2. while(!(UART0->S1 & UART\_S1\_TDRE\_MASK)) {}

3. UART0->D = data;

4. }

5.

Această functie este folosita pentru transmiterea datelor catre modulul de Pyhton. Funcția este apelată din adc.c, și transmite date prin UART. Datele pot fi trimise atunci când în registrul UART0\_S1 este setată masca pentru transmit data ready.

### ADC

"Analog to Digital Conversion", acest modul din cadrul proiectului convertește date analogice primite de la senzori, cuantificând tensiunile într-un anumit spectru de intervale, fiecărui interval atribuindu-i o valoare digitală. Datele de la senzorii de rotație și de șoc sunt trimise către MCU ca o tensiune între 0-3.3V. ADC (Conversor Analog-Digital) poate efectua o conversie analogic-digitală pe oricare dintre canalele selectabile software. Toate modurile realizează conversia printr-un algoritm de aproximare succesivă.

#### Initializarea ADC

1. void ADC0\_Init() {

2.

3. // Activarea semnalului de ceas pentru modulul periferic ADC

4. SIM->SCGC6 |= SIM\_SCGC6\_ADC0\_MASK;

5.

6. // Functia de calibrare

7. ADC0\_Calibrate();

8.

9. ADC0->CFG1 = 0x00;

10.

11. // Selectarea modului de conversie pe 10 biti single-ended --> MODE

12. // Selectarea sursei de ceas pentru generarea ceasului intern --> ADICLK

13. // Selectarea ratei de divizare folosit de periferic pentru generarea ceasului intern --> ADIV

14. ADC0->CFG1 |= ADC\_CFG1\_MODE(2) | ADC\_CFG1\_ADICLK(1) |ADC\_CFG1\_ADIV(3);

15.

16. // DIFF = 0 --> Conversii single-ended

17. ADC0->SC1[0] = 0x00;

18. ADC0->SC3 = 0x00;

19.

20.

21. ADC0->SC1[0] |= ADC\_SC1\_AIEN\_MASK;

22.

23. ADC0->SC1[0] &= ~ADC\_SC1\_DIFF\_MASK;

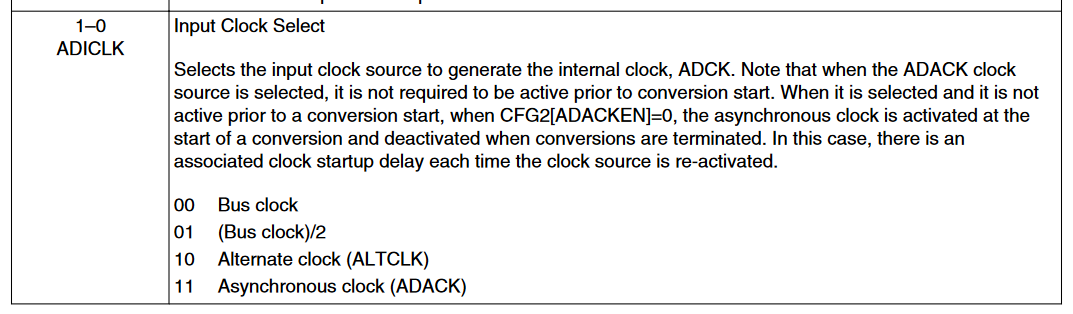
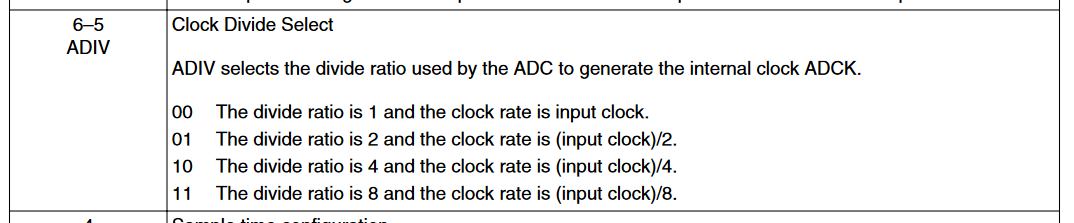
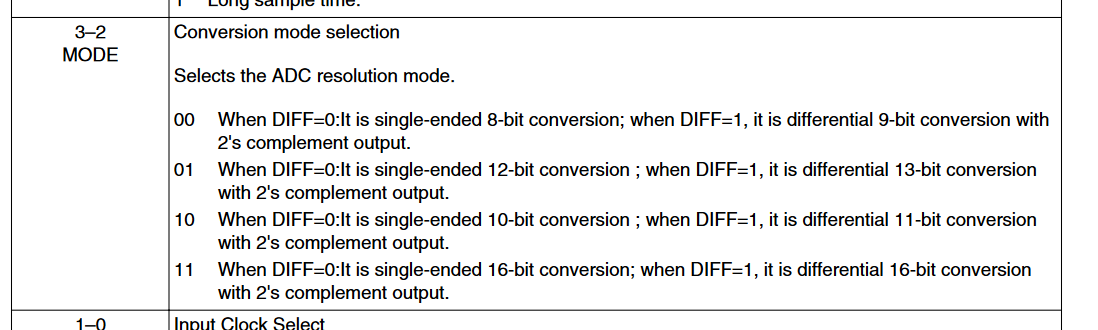
24. ADC0->SC3 &= ~ADC\_SC3\_AVGE\_MASK;

25. }

26.

Pentru calibrarea conversiei de date prin ADC se apelează funcția ADC\_Calibrate(), implementată în documentația oficială.

Registrul ADC0\_CFG1:

* Setarea conversiei cu precizie de 10 biți se efectuează setând în acest registru masca ADC\_CFG1\_MODE(2) și avand setat pentru DIFF valoarea 0, din aceasta cauza înainte de utilizare registrului i se atribuie valoarea 0.
* Sursa de ceas selectată pentru acest modul este aleasă cu masca ADC\_CFG1\_ADICLK(1), astfel se va folosi Bus Clock/2 .
* Divizarea sursei de ceas cu 8 prin intermediul folosirii maștii ADC\_CFG1\_ADIV(3), conform cu documentatia din Reference Manual.

#### Citirea datelor de pe canalele ADC

1. int16\_t ADC0\_Read(uint8\_t ch){

2.

3. // A conversion is initiated following a write to SC1A, with SC1n[ADCH] not all 1's

4. ADC0->SC1[0] = ADC\_SC1\_ADCH(ch);

5.

6. // ADACT is set when a conversion is initiated

7. // and cleared when a conversion is completed or aborted.

8. while(ADC0->SC2 & ADC\_SC2\_ADACT\_MASK);

9.

10. // A conversion is completed when the result of the conversion is transferred

11. // into the data result registers

12.

13. // If the compare functions are disabled, this is indicated by setting of SC1n[COCO]

14. // If hardware averaging is enabled, the respective SC1n[COCO] sets only if

15. // the last of the selected number of conversions is completed

16. while(!(ADC0->SC1[0] & ADC\_SC1\_COCO\_MASK));

17.

18. ADC0->SC1[0] |= ADC\_SC1\_ADCH(31);

19.

20. return (uint16\_t) ADC0->R[0];

21. }

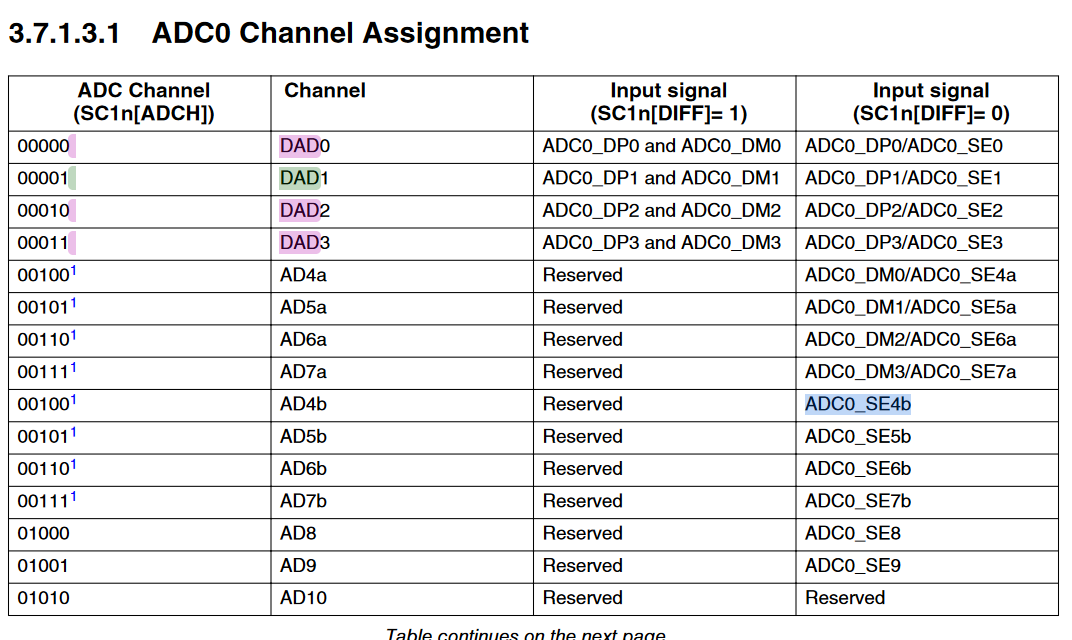
22.

Prin intermediul registrului ADC0\_SC1, se schimbă canalul de pe care se citesc datele, în campul ADCH. Alegerea canalului este influențată și de valoarea campului DIFF. Cum în proiect vom folosi conversii Single-Ended, DIFF este setat pe 0. Astfel, pentru citirea datelor, vom avea nevoie de 2 canale separate pentru ADC: un canal pentru senzorul de rotație și un canal pentru senzorul de șoc.

1. #define ADC\_CHANNEL1 (8) // PTB0

2. #define ADC\_CHANNEL2 (9) // PTB1

Canalele folosite sunt definite prin define-urile de mai sus. ADC\_CHANNEL1 corespunde lui PTB0, folosit de către senzorul de rotație, iar ADC\_CHANNEL2 corespunde lui PTB1, folosit de către senzorul de șoc. Valorile selectate sunt din următorul tabel, având în vedere că DIFF este setat pe 0. Deci vom folosi ca și input signal ADC0\_SE8 și ADC0\_SE9, care, folosind documentul pinouts, se observă că sunt expuse de porturile prezentate mai devreme.

Pentru citire, este folosită o funcție separată, deoarece având 2 canale, trebuie citirea efectuată pe fiecare în parte. Se setează canalul pe care vreau să citesc în registrul ADC0\_SC1. După selectarea canalului, aștept ca flag-ul ADC\_SC2\_ADACT\_MASK să nu mai fie în registrul ADC0\_SC2. Cât timp operația de șI între registru și flag dă ca rezultat 1, înseamnă că o conversie este în curs. Când rezultatul operației șI este 0, atunci conversia s-a efectuat. Se face busy-waiting până când conversia s-a încheiat. Apoi, verificarea setării în ADC0\_SC1 a lui COCO pe 1 înseamnă conversie terminată. După verificarea acestui flag, tot prin busy-waiting, se pot citi datele de pe canal.

Conform tabelului de mai sus, putem vedea ca pentru DIFF setat pe 0, canalele 8, respectiv 9, corespund semnalelor de input ADC0\_SE8 și ADC0\_SE9. Citirea datelor de pe canal se face din registrul ADC0\_R, rezultatul fiind un număr pe 2 octeți, de aceea rezultatul funcției este tipul de date int16\_t.

La încheierea operației de citire, se setează în registrul ADC0\_SC1 pe ADCH canalul 31, ceea ce înseamnă că modulul este dezactivat, conform figurii.

#### 



#### Utilizarea datelor de la senzori

1. void ADC0\_Func(){

2.

3. uint16\_t input\_rotation = ADC0\_Read(ADC\_CHANNEL1);

4. uint16\_t input\_soc = ADC0\_Read(ADC\_CHANNEL2);

5.

6. //Apeleaza aici functia pentru a intoarce motorul!

7. float rotation=input\_rotation\*180.0f/1024;

8. uint8\_t interval=find\_Interval((float)rotation);

9. setPG90\_angle(interval\*90);

10.

11. //Shiftare la stanga cu 6 pentru a aduce la 16 biti

12. //Rezultat pe 10 biti SE10

13. input\_rotation=(input\_rotation)<<6;

14. input\_soc=(input\_soc)<<6;

15.

16. uint8\_t firstbyte=input\_rotation>>8;

17. uint8\_t secondbyte=((input\_rotation<<8)>>8);

18. UART0\_Transmit(0x10);

19. UART0\_Transmit(0x99);

20. UART0\_Transmit(firstbyte);

21. UART0\_Transmit(secondbyte);

22.

23. firstbyte=input\_soc>>8;

24. secondbyte=((input\_soc<<8)>>8);

25. UART0\_Transmit(0x22);

26. UART0\_Transmit(0x88);

27. UART0\_Transmit(firstbyte);

28. UART0\_Transmit(secondbyte);

29.  }

30.

Datele de la senzorul de rotație se transformă într-un interval în funcție de valoarea lor, deci vom avea 3 intervale. În funcție de intervalul rezultat, se trimite către servomotor unghiul de rotație pe care trebuie să-l efectueze: 0, 90 sau 180 grade. Mai departe, datele, fiind pe 10 biți, se aliniază către stânga cu 6 biți, pentru a obține data pe 2 octeți, și se sparg în firstByte și secondByte. Datele pe 2 octeți se împart în câte 2 întregi pe 8 biți, deoarece UART transmite datele pe 1 octet.

Pentru a asigura sincronizarea dintre interfața grafică și datele trimise de la MCU, se folosește următorul protocol: după secvența /0x10/0x99 urmează 2 octeți care reprezintă unghiul citit de la senzorul de rotație, iar după secvența /0x22/0x22 urmează 2 octeți care reprezintă datele citite de la senzorul de șoc. Astfel, ne putem asigura că cele 2 module se pot sincroniza și știm cu siguranță care sunt datele trimise de fiecare senzor în parte, folosind o singură linie UART de transmisie.

PIT

#### Initializare PIT

1. void PIT\_Init(void) {

2.

3. // Activarea semnalului de ceas pentru perifericul PIT

4. SIM->SCGC6 |= SIM\_SCGC6\_PIT\_MASK;

5. // Utilizarea semnalului de ceas pentru tabloul de timere

6. PIT\_MCR &= ~PIT\_MCR\_MDIS\_MASK;

7. // Oprirea decrementarii valorilor numaratoarelor in modul debug

8. PIT->MCR |= PIT\_MCR\_FRZ\_MASK;

9. // Setarea valoarea numaratorului de pe canalul 0 la o perioada de 629 ms

10. PIT->CHANNEL[0].LDVAL = 0x64A3D6;

11.

12. // Activarea întreruperilor pe canalul 0

13. PIT->CHANNEL[0].TCTRL |= PIT\_TCTRL\_TIE\_MASK;

14. // Activarea timerului de pe canalul 0

15. PIT->CHANNEL[0].TCTRL |= PIT\_TCTRL\_TEN\_MASK;

16.

17.

18. // Setarea valoarea numaratorului de pe canalul 1 la o perioada de 30ms

19. PIT->CHANNEL[1].LDVAL = 191999\*3;

20.

21. // Activara întreruperilor pe canalul 1

22. PIT->CHANNEL[1].TCTRL |= PIT\_TCTRL\_TIE\_MASK;

23. // Activarea timerului de pe canalul 1

24. PIT->CHANNEL[1].TCTRL |= PIT\_TCTRL\_TEN\_MASK;

25.

26.

27.

28. // Activarea întreruperii mascabile si setarea prioritatii

29. NVIC\_ClearPendingIRQ(PIT\_IRQn);

30. NVIC\_EnableIRQ(PIT\_IRQn);

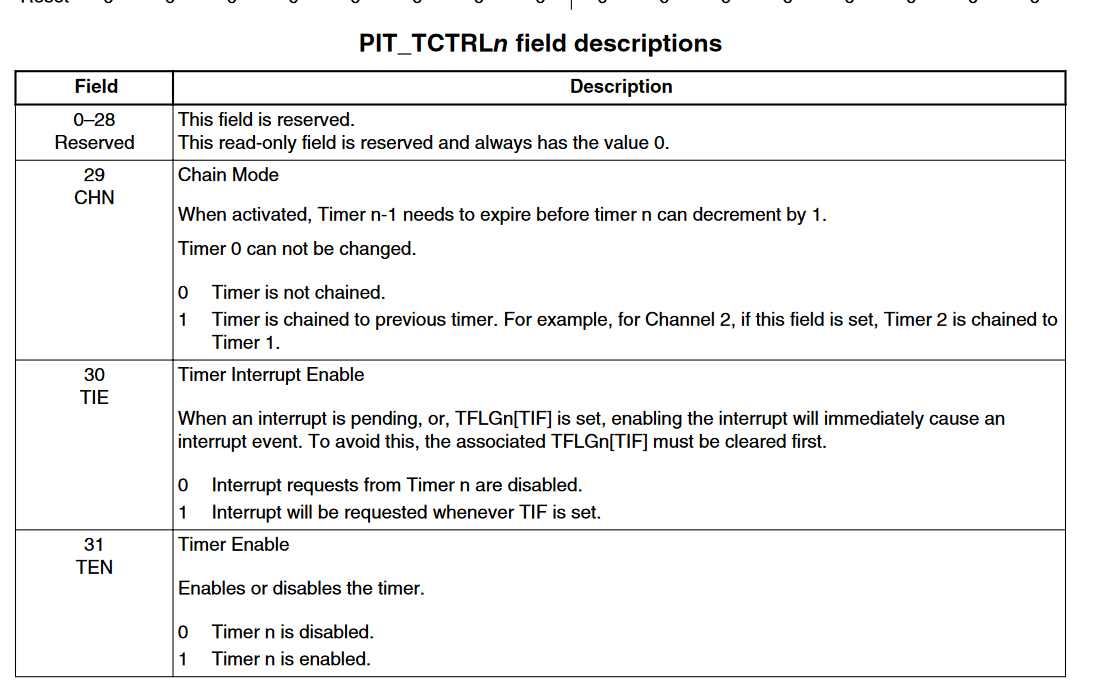
31. }

32.

Pentru PIT, am folosit 2 canale, deoarece avem nevoie de 2 întreruperi periodice: o întrerupere o dată la 629ms, conform cu specificațiile proiectului, pentru aprinderea secvenței de LED-uri, și o întrerupere la fiecare 30ms, pentru citirea datelor de la ADC, trimiterea acestor date către interfața grafică și acționarea servomotorului cu datele citite de la senzorul de rotație.

În registrul PIT\_TCTRL aferent fiecărui canal, cu masca TIE, setez activarea întreruperilor pentru canalul respectiv, iar cu masca TEN setez activarea timerului. Valoarea dată în LDVAL reprezintă un numărator modulo valoarea respectivă. Pentru calculul valorilor am folosit regula de 3 simplă, știind cât trebuie setat numărătorul pentru o secundă.

De asemenea, am pornit întreruperile pentru PIT, deoarece trebuie la fiecare perioadă pentru canalul 0 și 1 de PIT să apelez funcționalitățile precizate anterior.



În figura de mai sus se poate observa folosirea măștii TIE, pentru activarea întreruperilor și activarea măștii TEN pentru activarea timer-ului.

#### Intrerupere PIT

1. void PIT\_IRQHandler(void) {

2. static uint8\_t led\_turn=0;

3.

4.

5. if(PIT->CHANNEL[0].TFLG & PIT\_TFLG\_TIF\_MASK) {

6. if(led\_sequence\_direction==1)

7. led\_turn++;

8. else

9. led\_turn--;

10.

11. if(led\_turn == UINT8\_MAX)

12. led\_turn=3;

13. if(led\_turn==4)

14. led\_turn=0;

15. PIT->CHANNEL[0].TFLG &= PIT\_TFLG\_TIF\_MASK;

16. sequence\_leds(led\_turn);

17. }

18. if(PIT->CHANNEL[1].TFLG & PIT\_TFLG\_TIF\_MASK) {

19.

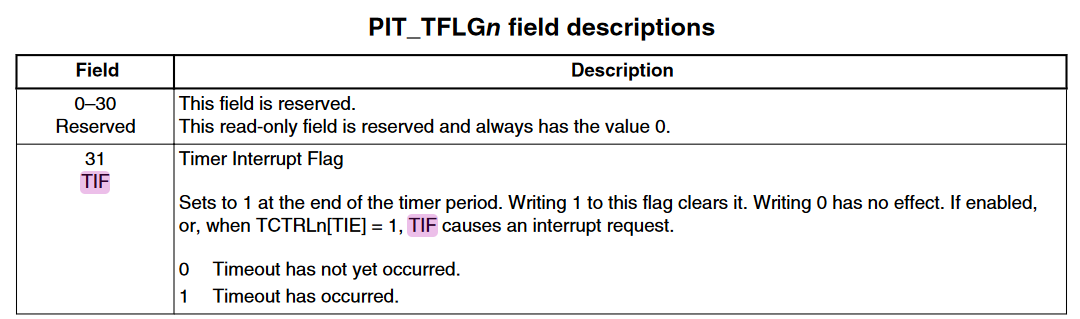
20. PIT->CHANNEL[1].TFLG &= PIT\_TFLG\_TIF\_MASK;

21. ADC0\_Func();

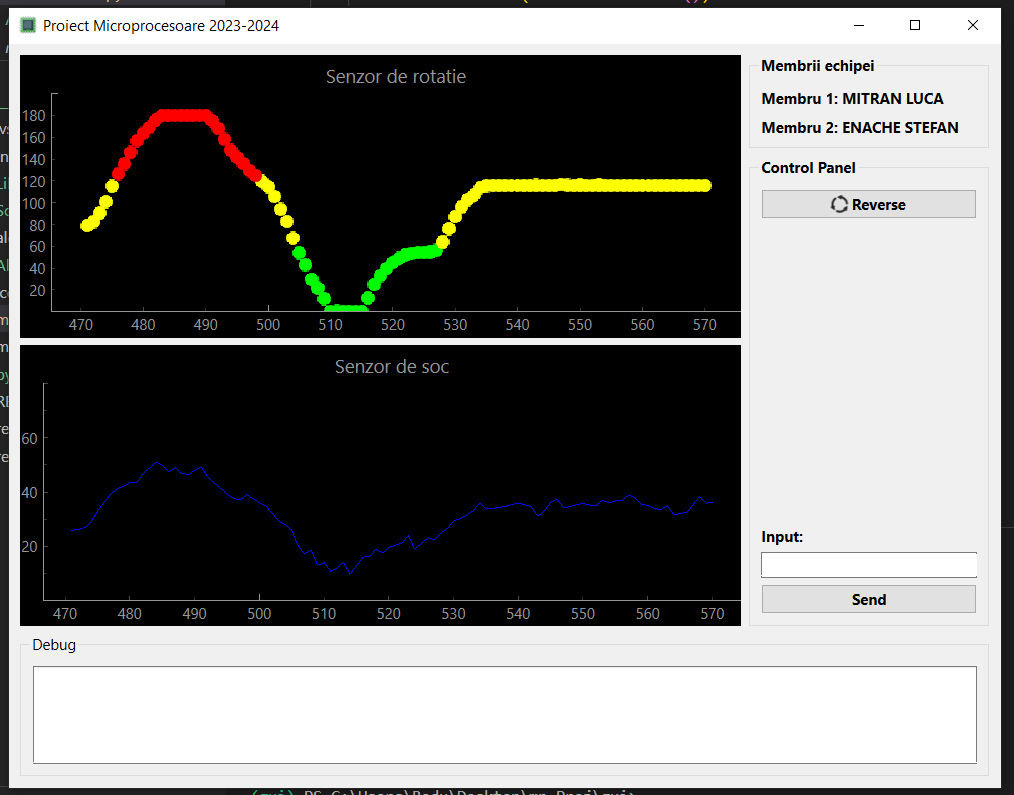
22. }

23. }

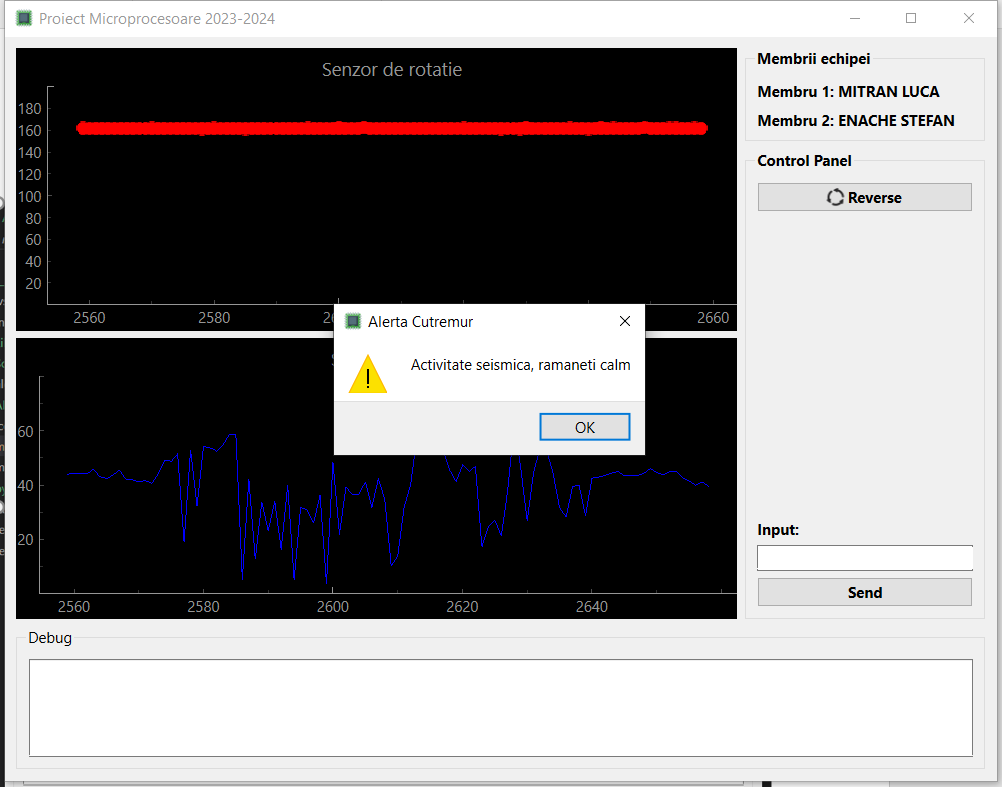
Funcția de întreruperi de PIT tratează cele 2 canale, când timerele pentru fiecare canal au ajuns la 0, adică când Timer Interrupt Flag este setat în registrul PIT\_TFLG aferent fiecărui canal. Pentru fiecare canal în parte, se apelează funcțiile specifice. Pentru canalul 1, se aprinde secvența de LED-uri în funcție de valoarea variabilei globale led\_sequence\_direction. Dacă această variabilă este setată pe 1, LED-urile se vor aprinde, fiecare pentru 629ms, în ordinea descrisă în specificațiile proiectului: Alb, Verde, Albastru și Galben; altfel, LED-urile se aprind în ordine inversă.

Pentru canalul 1, se apelează funcția din adc.c care citește datele de la senzori, le trimite prin UART și rotește servomotorul în funcție de inputul primit ca interval. În figura de mai jos este prezentată semnificația.

# Testarea aplicației



In graficele de mai sus, din interfata grafica, putem observa variatia datelor primite de la senzorul de rotatie si de la senzorul de soc.



Activarea alertei in functie de datele care au trecut peste un prag setat pentru senzoru de soc.

# Probleme întâmpinate

Sincronizarea datelor transmise de la microcontroller către interfața grafică este crucială pentru a evita erorile de desincronizare. Deoarece datele sunt reprezentate pe 2 bytes, pentru o eficiență mai mare și un overhead mai mic, acestea sunt trimise ca atare, în format binar, și nu caracter cu caracter în reprezentare zecimală. Astfel, interfața grafică, neavând un mod de a ști când se trimite primul byte al unei secvențe de 2 bytes, pot apărea erori de desincronizare și se pot citi date eronate. Spre exemplu, se poate citi ca și prim octet (fals) al doilea octet al unei citiri și ca și al doilea octet primul octet al următoarei.

Avem datele:

/0x01/0x02/0x03/0x04

Presupunem că prima valoare este /0x01/0x02, iar a doua /0x03/0x04. Pe exemplul dat, poate apărea eroarea următoare: din Python se ia ca și valoare validă /0x02/0x03.

Pentru rezolvarea acestei probleme, am folosit un "protocol" de comunicare care să anticipe primirea a 2 bytes primiți de la senzorul de rotație și care să anticipe primirea a 2 bytes de la senzorul de șoc.

Întrucât datele venite de la ADC nu sunt mereu foarte apropiate, chiar la variații foarte mici, pot apărea spike-uri, am folosit o mediere din interfața grafică a valorilor venite de la microcontroller, pentru a avea niște grafice mai line. Folosind un filtru de mediere pentru un set de valori primite obținem un grafic mai lin.

Pentru folosirea TX\_INV pentru UART nu exista suport în Python pentru clasa Serial.

În specificațiile servomotorului SG90 aveam următoarele valori de duty cycle pentru a roti elicea : 2.5% pentru 0 ° și 12.5% pentru 180 °.Aceste valori nu au fost însă corecte pentru a roti servomotorul, astfel ca am luat prin încercări valorile pentru CnV.

# Bibliografie

* APMonitor. Servo Control. <https://apmonitor.com/dde/index.php/Main/ServoControl>
* ARM mbed. FRDM-KL25Z <https://os.mbed.com/platforms/KL25Z/>
* Tinkercad <https://www.tinkercad.com>
* DFRobot Wiki. <https://wiki.dfrobot.com/>
* Qt for Python <https://doc.qt.io/qtforpython-6/index.html>
* Arduino Modules. KY-031 Knock Sensor Module <https://arduinomodules.info/ky-031-knock-sensor-module/>