**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**Facultatea de Sisteme Informatice și Securitate Cibernetică**

**Departamentul de Calculatoare și Securitate Cibernetică**



***Utilizare senzor ANALOGIC DE ROTAȚIE***

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

Std. sg. maj. Bureacă Angela-Emilia

Std. sg. maj. Huțanu David

Grupa C114C

**București**

**2023**

Cuprins

[1. Prezentarea senzorului DFR0054 3](#_Toc155637089)

[2. Scop proiect 4](#_Toc155637090)

[3. Conectare senzor – placă de dezvoltare 5](#_Toc155637091)

[4. Diagramă sistem 6](#_Toc155637092)

[5. Descriere program 7](#_Toc155637093)

[5.1. Funcția main 7](#_Toc155637094)

[5.2. Inițializarea modulelor 7](#_Toc155637095)

[5.2.1. Inițializarea modulului UART 7](#_Toc155637096)

[5.2.2. Inițializarea modulului GPIO 9](#_Toc155637097)

[5.2.3. Inițializarea modulului PIT 12](#_Toc155637098)

[5.2.4. Inițializarea modulului ADC 16](#_Toc155637099)

[6. Descriere interfață grafică 22](#_Toc155637100)

[7. Diagrama de stări a sistemului 24](#_Toc155637101)

[8. Rezultat proiect 25](#_Toc155637102)

[9. Dificultăți întâmpinate 25](#_Toc155637103)

[10. Referințe 26](#_Toc155637104)

# Prezentarea senzorului DFR0054

Senzorul DFR0054 este un senzor analogic, un simplu potențiometru (rotary potentiometer sensor) produs de DFRobot, parte a seriei Gravity, care permite rotația de la 0 până la 300 de grade.

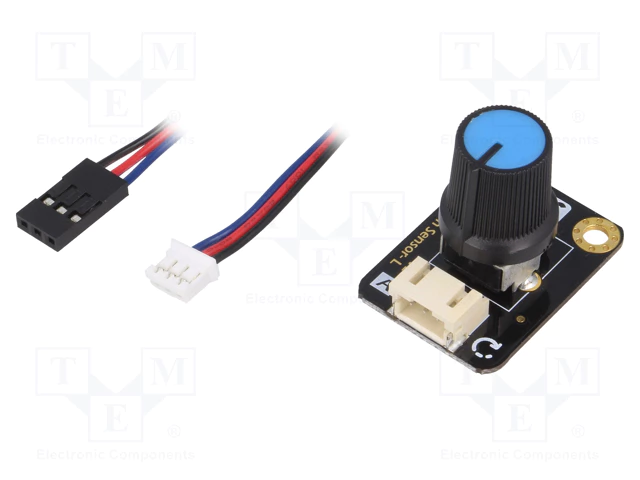


Figura 1. Senzorul DFR0045

**Descriere:**

* **Tipul Senzorului:** Potențiometru rotativ. Acesta este un senzor analogic care furnizează o valoare variabilă în funcție de poziția sau rotația unui potențiometru.
* **Interfață:** Analogică. Acest senzor se conectează la un port analogic al unui microcontroler sau unei plăci de dezvoltare pentru a transmite valori analogice.
* **Tensiune de Lucru:** Alimentare la 3.3VDC. Acesta necesită o alimentare de 3.3 volți pentru a funcționa corect.
* **Canale:** Are un singur canal (Ch: 1), ceea ce înseamnă că oferă o singură valoare de ieșire.

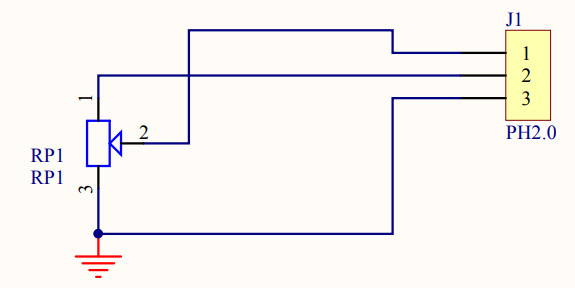


Figura 2. Circuitul senzorului DFR0045

Specificații:

* Tensiune de alimentare: 3.3V până la 5V;
* Se poate roti până la 300 de grade;
* Mărime: 22x31 mm.

|  |  |
| --- | --- |
| Red | VCC |
| Black | GND |
| Blue/Green | Analog signal output |

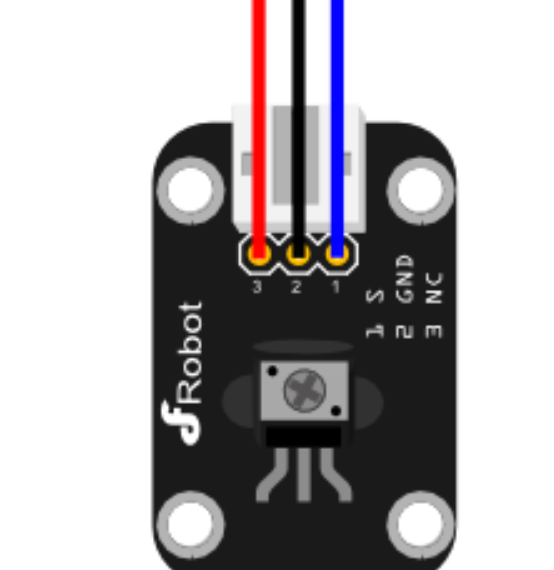


Figura 3. Conectare senzor

# Scop proiect

Scopul acestui proiect este acela de a afișa în interfață un grafic ce reprezintă datele primite de la senzorul de rotație în timp real și modificarea LED-ulurilor, atât cele de pe breadboard, cât și cel prezent pe placa de dezvoltare pusă la dispoziție în funcție de poziția senzorului.

Se va dezvolta un program care va verifica valorile primite de la senzor și va actualiza graficul.

De asemenea, sunt asignate patru culori specifice care trebuie sa fie afisate în ordine pe LED-ul dispus pe placă astfel: verde, albastru, mov si negru. Ordinea poate fi schimbată din interfață (poate fi cea dată anterior sau invers : nergu, mov, albastru și verde).

# Conectare senzor – placă de dezvoltare

Vom conecta senzorul astfel:

* Firul verde – conectează senzorul la pinul de pe placă pe care va transmite datele (în cazul prioectului nostru la pinul PTC2)
* Firul roșu – conectează senzorul la alimentarea de 3.3V
* Firul negru – conectează senzorul la GND

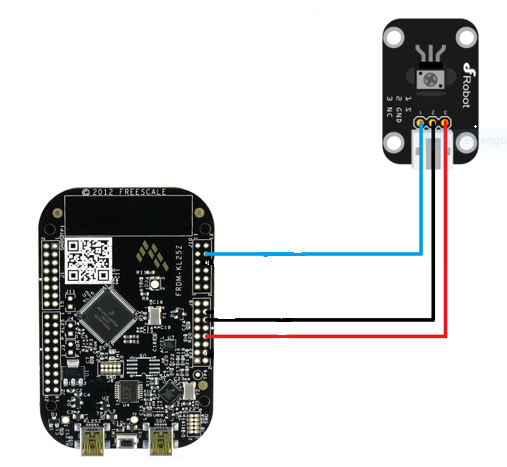


Figura 4. Conectare senzor

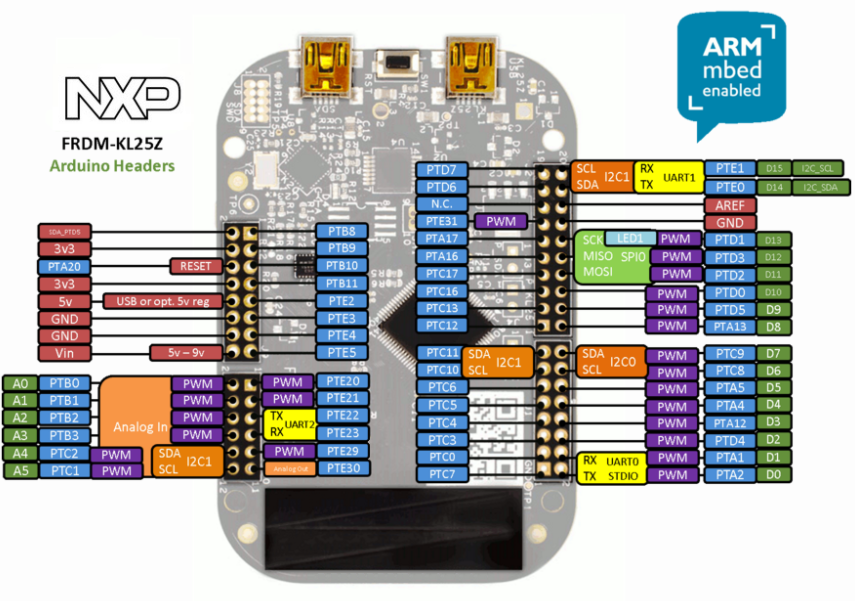


Figura 5. Pini placă dezvoltare

# Setup

Conectăm cele 3 led-uri pe breadboard și pentru fiecare avem pin asociat pe placă astfel:

* led-ul verde se va conecta la pinul PTA4 de pe J1;
* led-ul galben se va conecta la pinul PTA5 de pe J1;
* led-ul roșu se va conecta la pinul PTA12 de pe J1.

Conectam senzorul de rotație care are 3 pini asociați astfel:

* firul verde se va conecta la pinul PTC2 de pe J10;
* firul roșu se va conecta la pinul de 3.3V de pe J9;
* firul negru se va conecta la GND pe J9.

Deschidem programul in aplicația Keil, apasăm butonul de build:

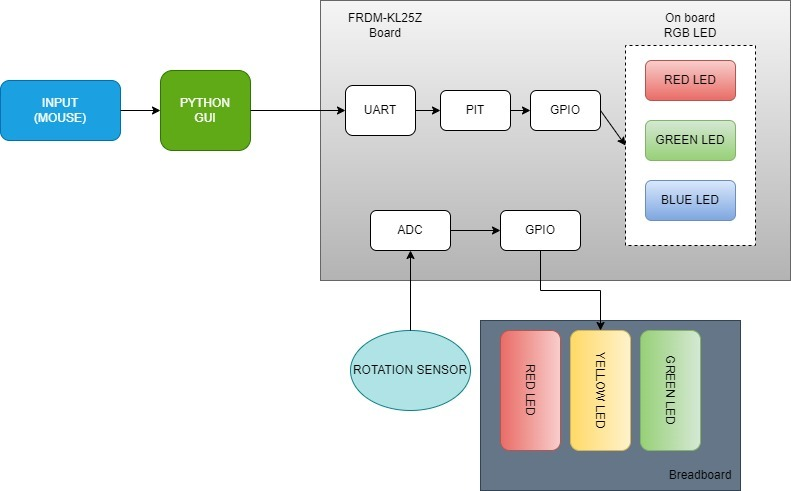


Apoi apasăm butonul de load pentru a încărca proiectul pe placa de dezvoltare:



Deschidem interfața grafică în visual code si rulăm codul.

# Diagramă sistem



*Figura 6. Diagramă sistem*

# Descriere program

## Funcția main

În fișierul *main.c* am inclus fișierele header *Pit.h, Uart.h*, *Gpio.h și Adc.h* în care sunt declarate funcții și variabile ce vor fi folosite în interiorul aplicației.

1. #include "MKL25Z4.h"

2. #include "Pit.h"

3. #include "Uart.h"

4. #include "Gpio.h"

5. #include "Adc.h"

6.

7. int main()

8. {

9. UART0\_Init(9600);

10. PIT\_Init();

11. OutputPIN\_Init();

12. ADC0\_Init();

13.

14. while (1)

15. {

16. }

17. }

Logica programului este următoarea:

* se inițializează modulul UART prin apelarea funcției UART0\_Init cu baudrate-ul de 9600;
* se inițializează modulul PIT prin apelarea funcției PIT\_Init;
* se inițializează modulul ADC prin apelarea funcției ADC0\_Init;
* se inițializează GPIO prin apelarea funcției OutputPIN\_Init;
* se intră într-un loop infinit care are scopul de a lăsa aplicația să ruleze și de a detecta întreruperile venite de la modulul ADC și modulul PIT.

## Inițializarea modulelor

### Inițializarea modulului UART

Vom folosi modulul UART pentru comunicația serială si asincrona cu PC prin cablul USB. Are doua linii: receptie si transmisie.

În funcția de inițializare, configurăm parametrii necesari funcționării modulului astfel:

1. /\*Setarea sursei de ceas pentru modulul UART\*/

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

3.

4. /\*Activarea semnalului de ceas pentru modulul UART\*/

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

6.

7. /\*Activarea semnalului de ceas pentru portul A\*/

8. /\*intrucat dorim sa folosim pinii PTA1, respectiv PTA2 pentru comunicarea UART\*/

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

10.

11. /\*Fiecare pin pune la dispozitie mai multe functionalitati \*/

12. /\*la care avem acces prin intermediul multiplexarii\*/

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

14. PORTA->PCR[1] = PORT\_PCR\_ISF\_MASK | PORT\_PCR\_MUX(2); /\* Configurare RX pentru UART0\*/

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

16. PORTA->PCR[2] = PORT\_PCR\_ISF\_MASK | PORT\_PCR\_MUX(2); /\* Configurare TX pentru UART0\*/

17.

18.

19.

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

21.

22. /\*

23. //SBR - vom retine valoarea baud rate-ului calculat pe baza frecventei ceasului de sistem

24. // SBR - b16 b15 b14 [b13 b12 b11 b10 b09 b08 b07 b06 b05 b04 b03 b02 b01] &

25. // 0x1F00 - 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0

26. // 0 0 0 b13 b12 b11 b10 b09 0 0 0 0 0 0 0 0 >> 8

27. // BDH - 0 0 0 b13 b12 b11 b10 b09

28. // BDL - b08 b07 b06 b05 b04 b03 b02 b01

29. \*/

30.

31.

32.

33. /\*-------SET OVER SAMPLING RATE TO 8--------\*/

34. uint32\_t osr = 8; /\* Over-Sampling Rate (numarul de esantioane luate per bit-time)\*/

35.

36. /\* calculeaza Baud Rate Divisor(sbr) pe baza osr dat\*/

37. uint16\_t sbr = (uint16\_t)((DEFAULT\_SYSTEM\_CLOCK) / (baud\_rate \* (osr + 1)));

38.

39. /\* extrage bitii high si low din sbr\*/

40. uint8\_t sbr\_high = (uint8\_t)((sbr & 0xFF00) >> 8);

41. uint8\_t sbr\_low = (uint8\_t)(sbr & 0xFF);

42.

43. /\* seteaza UART0\_BDH cu bitii high din sbr\*/

44. UART0->BDH = UART0\_BDH\_SBR(sbr\_high);

45.

46. /\* seteaza UART0\_BDL cu bitii low din sbr\*/

47. UART0->BDL = sbr\_low;

48.

49. /\* seteaza UART0\_C4 cu osr ales\*/

50. UART0->C4 = (UART0->C4 & ~UART0\_C4\_OSR\_MASK) | UART0\_C4\_OSR(osr);

51. /\*-------END SET OVER SAMPLING RATE TO 8--------\*/

52.

53. /\*-------SET RX INVERTED BIT------\*/

54. UART0->S2 |= UART0\_S2\_RXINV\_MASK;

55. /\*-------END SET RX INVERTED BIT-----\*/

56.

57.

58. /\*-------SET DATA FRAME ORDER TO LSB FIRST----------\*/

59. /\* seteaza LSB bit in UART0\_S2 (putea fi lasata si default, e lsb first by default)\*/

60. UART0->S2 &= ~UART0\_S2\_MSBF\_MASK;

61. /\*-------END SET DATA FRAME ORDER TO LSB FIRST----------\*/

62.

63. /\*Setare numarul de biti de date la 8 si fara bit de paritate\*/

64. UART0->C1 = 0;

65.

66. /\*Dezactivare intreruperi la transmisie\*/

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

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

69.

70. /\*Activare intreruperi la receptie\*/

71. UART0->C2 |= UART0\_C2\_RIE(1);

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

73.

74. NVIC\_SetPriority(UART0\_IRQn, 1); /\* setam prioritate mai mica decat la modulul de PIT pentru nu a interfera intre ele\*/

75. NVIC\_EnableIRQ(UART0\_IRQn);

76.

În funcția de întrerupere, care se apelează de fiecare dată când utilizatorul dorește schimbarea ordinii culorilor led-ului de pe placă, se va seta flag-ul is\_changed, flag prin care mai apoi în modulul GPIO se va interchimba ordinea.

1. void UART0\_IRQHandler(void)

2. {

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

4. {

5. c = UART0->D;

6.

7. if(c == 'g')

8. {

9. is\_changed = '1'; /\* 'g' imi spune ca s-a schimbat ordinea\*/

10. }

11. }

12. }

### Inițializarea modulului GPIO

Modulul GPIO se ocupa cu pinii expusi fizic si permite controlarea la „run-time” a comportamentului unui pin al unui circuit integrat, mai specific, a direcției de trecere a curentului electric prin el (intrare / iesire) si valoarea logică pe care acesta o are.

Un pin poate fi configurat pentru scriere sau citire.

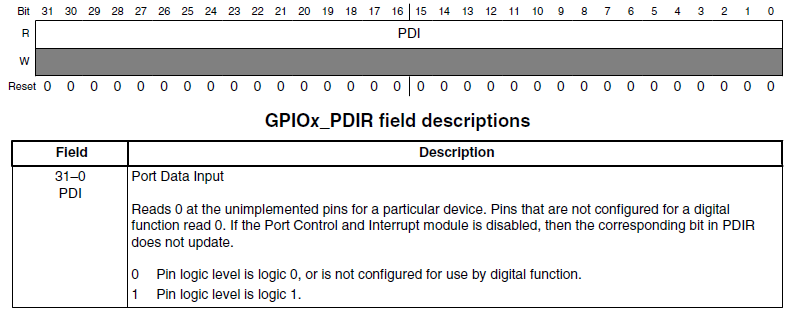
Interfațe (dispozitive externe) :

* Intrare – switch;
* Ieșire – LED;

Pe placa noi avem Port A, B, C si D. Fiecare port are 32 de biti (bit = pin fizic expus pe microcontroler), dar nu toti sunt expusi.

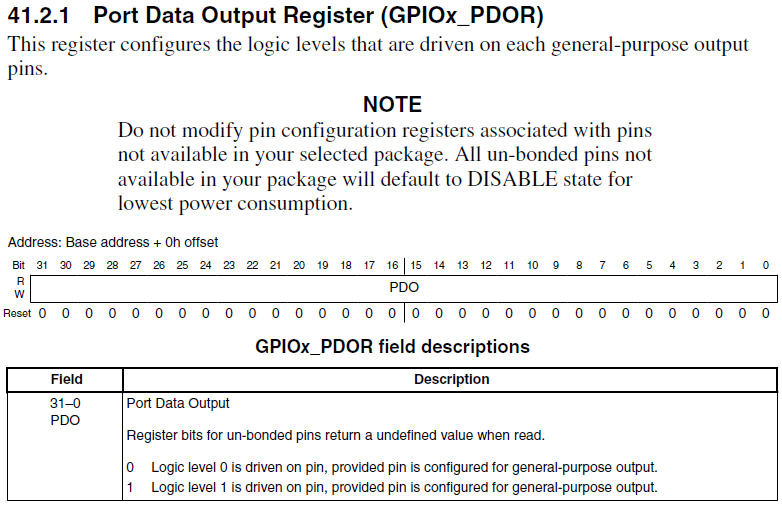
Pentru a controla un bit/pin, trebuie sa interactionam cu 3 registrii : PDIR, PDOR si PDDR:

* PDIR (GPIOx\_PDIR , unde x poate fi A, B, C sau D) : folosit pentru a mentine valoare bitului de intrare. Observam ca putem doar citi(R) din el, nu putem scrie(W).



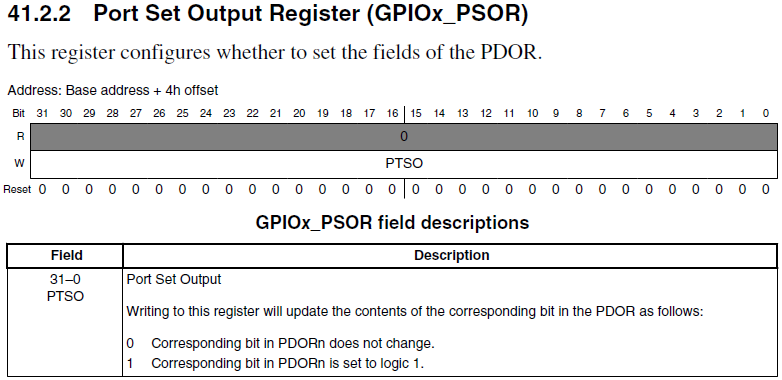
In cazul in care folosesc mai multi pini de pe acelasi port si acestia genereaza intreruperi, in handler-ul asociat pinului, cu ajutorul acestui registru GPIOx->PDIR pot sa vad verificand-ul, de pe care pin a venit intreruperea !!! (de exemplu folosim PTA4 si PTA12, pe ambii sunt conectate doua push-botton uri).

* PDOR (GPIOx\_PDOR , unde x poate fi A, B, C sau D) : este folosit pentru a scrie in el ceea ce vrem output.

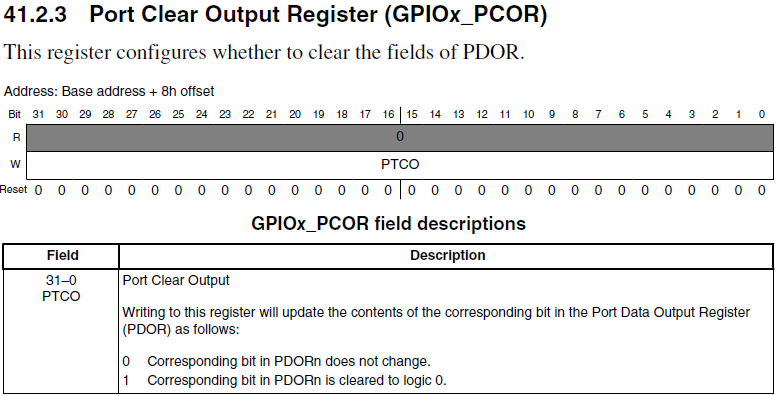


Putem modifica registru PDOR in 4 moduri :

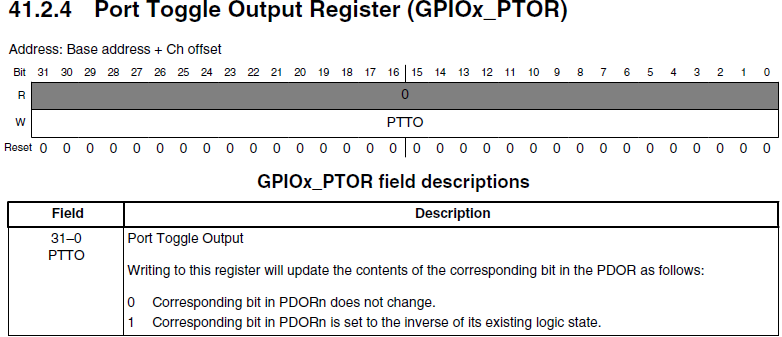
* Scriu direct in el
* Prin registrul PSOR:



* Prin registrul PCOR :

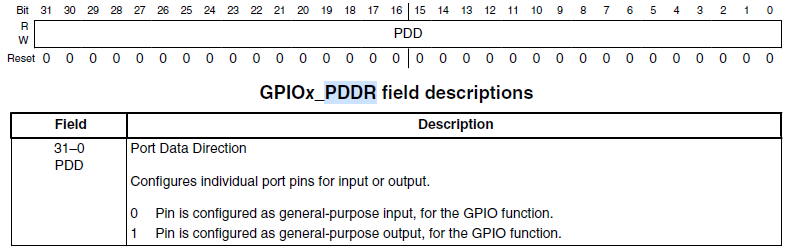


* Prin registrul PTOR:



* PDDR (GPIOx\_PDDR , unde x poate fi A, B, C sau D) : prin acest registru configurez/setez pinul ca fiind de input sau de output.

De exemplu, daca pentru pinul PTA4 registrul PDDR e setat pe 1, adica GPIOA->PDDR |= 1 << 4; atunci acel pin va fi folosit pentru output, iar daca e setat pe 0, acel pin va fi folosit pentru input.

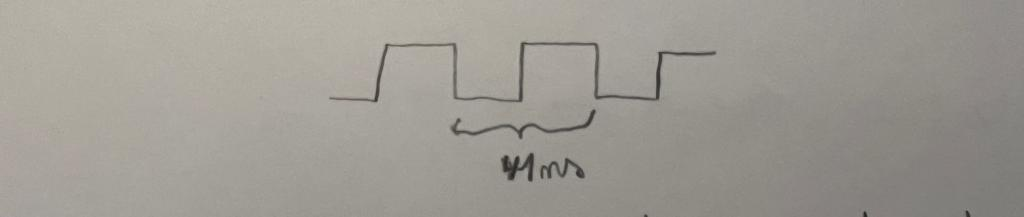


### Inițializarea modulului PIT

Perifericul PIT e o entitate separata de procesor si poate functiona in paralel chiar daca noi facem debug. Perifericul PIT nu interfatează cu pini externi. In general, pentru un numarator (PIT) avem :

* o frecventa de intrare (Fclk) = frecventa sursei de ceas
* si rezolutia = pe cati biti e acel numarator, adica pana la cat numara.

Pe aceasta placa FRDM-KL25Z avem doua surse de ceas (acele 2 cipuri cu suprafata metalica, argintie : y1 si y2). Exista doua numaratoare care sunt descrescatoare si sunt pe 32 biti => numere de la 0 pana la 2^32 – 1. Frecventa Fclk este de maxim 24 MHz = > Pclk = 1/(24\*10^5)s = 0,041 \* 10^6s = 41ns.



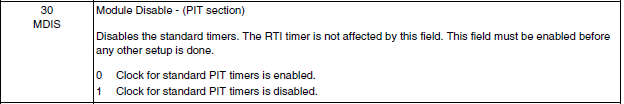
Pentru a mari rezolutia avem la dispozitie două canale care pot fi folosite pentru timere (Channel 0 si Channel 1, fiecare pe 32 biti). Daca le concatenam vom avea numarator pe 64biti = > numara de la 0 la 2^62 – 1 (se mareste plaja de valori). Accesul la un canal se face cu ajutorul sintaxei PIT->CHANNEL[x].

Fiecare timer are asociată o valoare de numărare independentă (load\_value = LDVAL). In cazul placii noastre, e o valoare de la xare incepe sa numere (sa decrementeze).

PIT ul, in general, are 3 registrii specifici:

* PIT\_MCR : This register enables or disables the PIT timer clocks and controls the timers when the PIT enters the Debug mode. Contine printre cei 32 de biti:

1. un bit pentru MDIS :

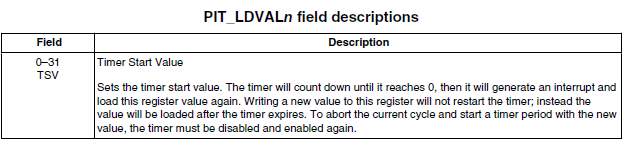


1. un bit pentru FRZ : daca e 0, timerul nu se blocheaza in cazul in care facem debug, iar daca e 1, atunci timerul este oprit.

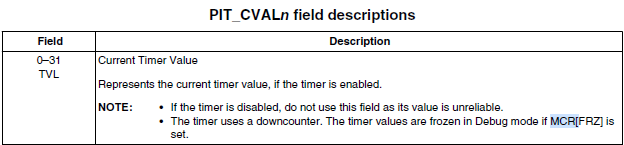
* PIT\_LTMR64H : acesta si urmatorul sunt pentru folositi pentru acea inlantuire/concatenare de timere
* PIT\_LTMR64L

Si fiecare timer/numarator/PIT la randul sau, are cate 4 registrii specifici:

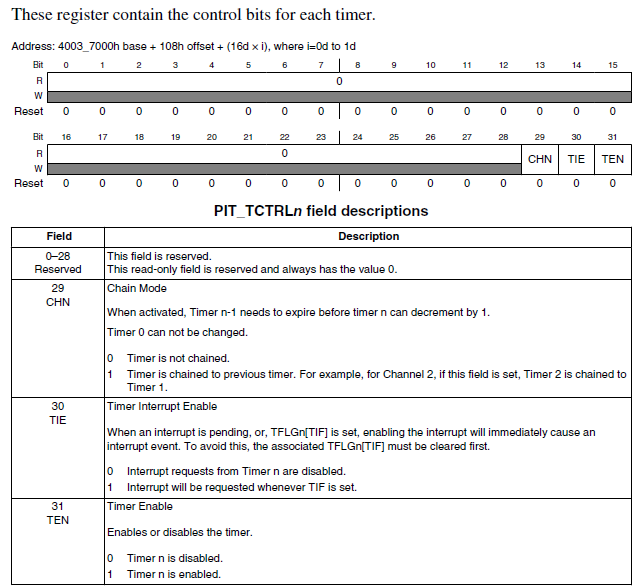
* PIT\_LDVALn (n e specific canalului, poate fi 0 sau 1) : reprezinta load\_value; prin el specific perioada dintre doua intreruperi(reseturi).



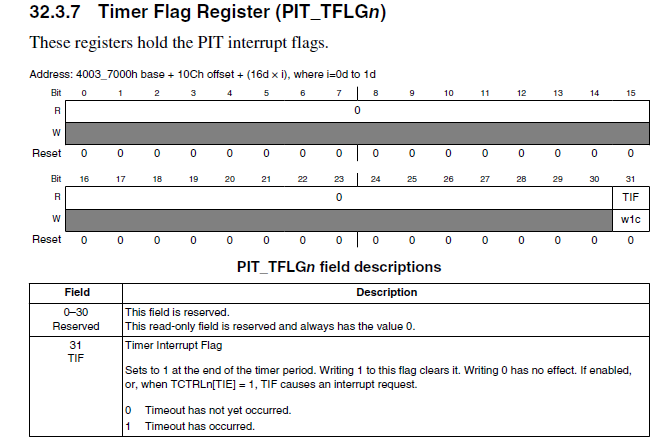
* PIT\_CVALn (n e specific canalului, poate fi 0 sau 1) : avem doar drept de read(R) si reprezinta valoarea curenta a timerului.



* PIT\_TCTRLn (n e specific canalului, poate fi 0 sau 1) :



* PIT\_TFLGn (n e specific canalului, poate fi 0 sau 1) :



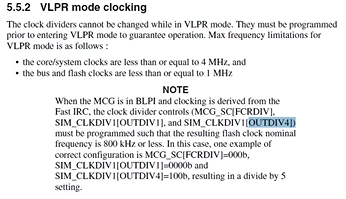
La proiectul nostru, modulul PIT este in stransa legatura cu led ul de pe placa, PIT fiind considerat device input iar ledul de pe placa, device output. Cu ajutorul lui vom afisa succesiunea de culori in oridinea : verde, albastru, mov si negru, sau invers.

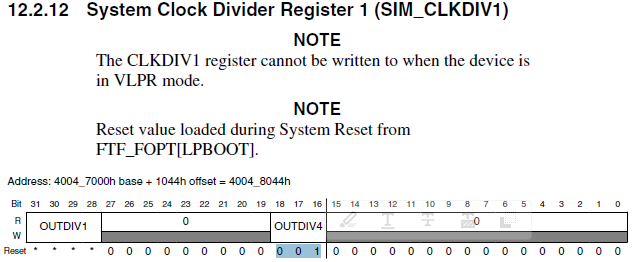
Orice periferic de pe aceasta placa va avea ca si frecventa valoarea lui Bus clock ( = frexcventa de ceas a magistralei). Pentru a calcula valoarea lui LDVAL avem nevoie sa stim valoarea lui bus clock care se calculeaza astfel (capitolul 5.4 din referance manual):



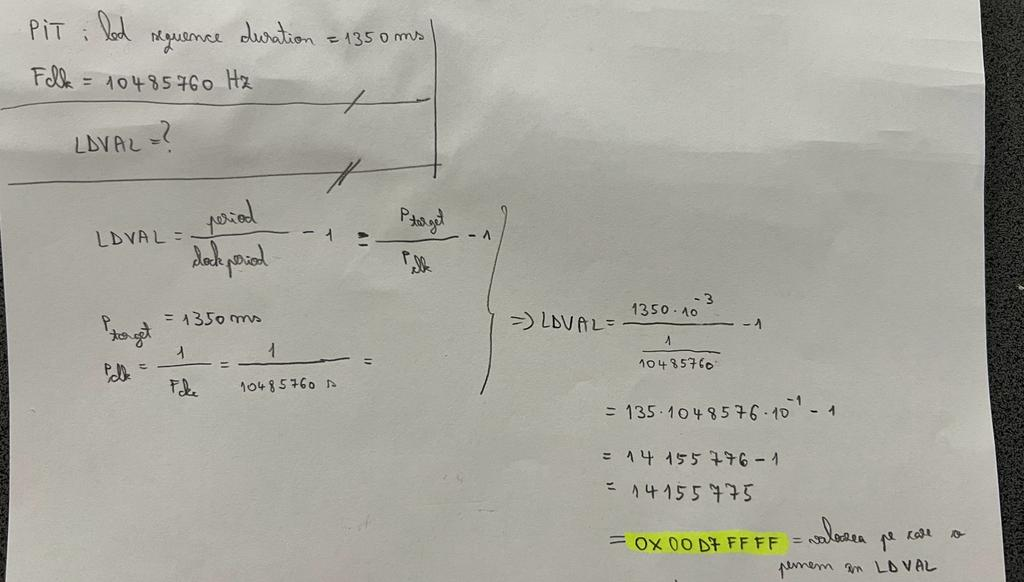
In  gasim valoarea lui system clock (e o valoare in hertz) :



Valoarea lui OUTDIV4 o vom lua cea de reset (vedem ca e 001) de unde e definita, si anume in registrul SIM\_CLKDIV : 



Ce inseamna 001? -> inseamna divide-by-2 = > ceea ce inseamna ca frecventa de intrare va fi DEFAULT\_SYSTEM\_CLOCK/2 = 20971520/2 = 10485760 Hz = frecventa maxima la care lucreaza microcontrolerul.



### Inițializarea modulului ADC

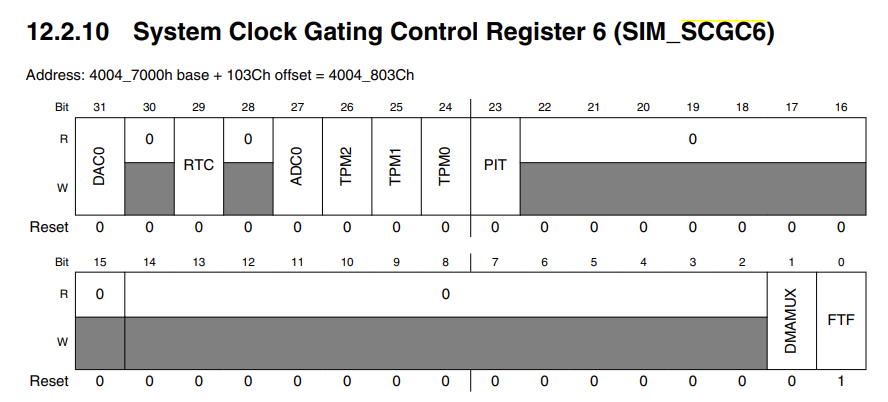
Microcontrolerul FRDM KL25Z este echipat cu un modul ADC (Convertor Analog-Digital), oferind posibilitatea de a converti semnale analogice în valori digitale. Acest modul ADC este esențial în domenii precum monitorizarea senzorilor, controlul motoarelor și în alte aplicații care necesită interacțiunea cu lumea analogică.

Principalele caracteristici ale modulului ADC pe FRDM KL25Z includ rezoluție configurabilă, multiple canale de intrare analogică, și capacitatea de a genera intreruperi (interrupts) în timpul conversiilor. Cu o rezoluție variabilă, utilizatorul poate alege între o precizie mai mare sau o viteză mai mare de conversie, în funcție de cerințele specifice ale aplicației.

Microcontrolerul oferă, de asemenea, diferite moduri de operare, cum ar fi modul single-ended și modul diferential, pentru a satisface nevoile diverse ale aplicațiilor. Utilizatorii pot selecta sursa de referință pentru a asigura stabilitatea și precizia conversiilor ADC.

În cadrul proiectului nostru, am folosit acest ADC pentru a transforma semnalele digitale venite de la senzorul de rotație în semnale analogice pe care microcontrolerul să le poată interpreta.

În acest sens, am pornit mai întâi sursa de ceas pentru ADC0 pentru a putea alimenta cu energie acest modul de pe placa. Acest lucru îl putem face setand pe 1 bitul corespunzator din SIM\_SCGC6 (system clock gating register 6).



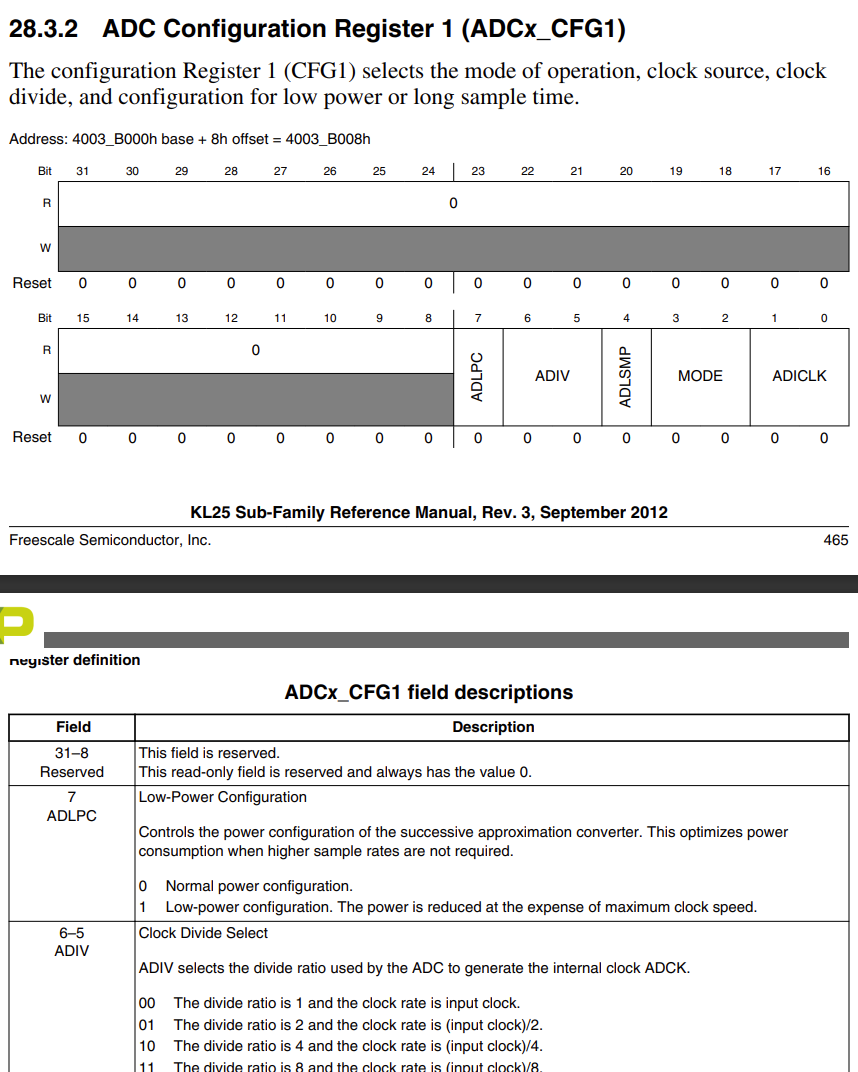
Apoi, am creat o funcție de calibrare a modulului conform indicațiilor din Reference Manual. Acest lucru este necesar pentru îmbunătățirea acurateții, micșorarea erorilor și stabilitatea pe termen lung. Considerăm că explicațiile oferite în Reference Manual sunt destul de explicite, prin urmare recomandăm cititorilor să se îndrepte spre acestea privind partea de calibrare a modulului (pag. 494).

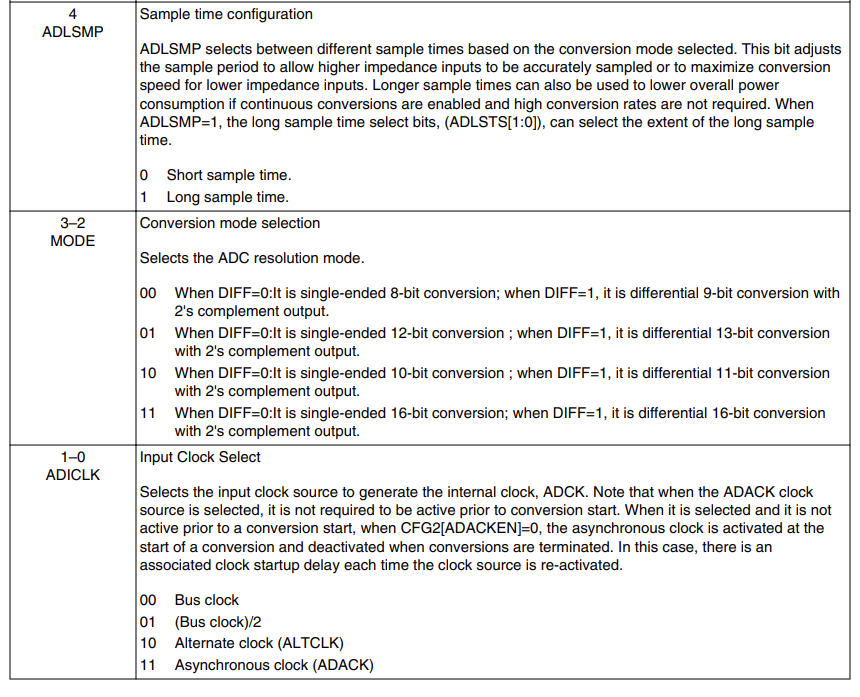
După calibrare, trebuie să configuram ADC cu opțiunile dorite. Pentru aceasta, ne îndreptam atenția spre registrul de configurare CFG1. Ne dorim să avem normal power configuration, să folosim ca sursă de ceas sursa plăcii, sursa internă a ADC sa fie sursa de ceas input divizată la 4 (pentru a avea o frecvneta mai mica sau egala cu 4MHz (pag 494)), să avem o conversie de 16 biti și să utilizăm convertorul pe un short sample time. Astfel setăm corespunzător biții acestui registru de configurare.

1. ADC0->CFG1 = 0x00; /\* dupa setarea unor biti mai jos raman pe 0 (Normal Power configuration) si (Short sample time) \*/

2. ADC0->CFG1 |= ADC\_CFG1\_MODE(3) | ADC\_CFG1\_ADICLK(0) | ADC\_CFG1\_ADIV(2);

3. /\*reg config | conv pe 16 biti | fol sursa de ceas MCU | imparte sursa de ceas la 4 (pentru a obtine o frecventa <= 4 Mz -> pag 494)\*/





De asemenea, după cum scrie și în explicația câmpului MODE de mai sus, pentru a avea single ended conversion, trebuie să setam pe 0 bitul DIFF. Acesta se află în registrul de status și control 1 (ADC0\_SC1). Tot aici putem seta ca atunci cand s-a realizat o conversie completă să se apeleze o întrerupere (câmpul AIEN) precum și canalul de input de unde se vor citi datele de la senzor (în cazul nostru 13 întrucât conectam senzorul pe pinul PTC2)(câmpul ADCH).

Totodată, setam ca, conversiile să fie continue astfel încat să putem folosi și întreruperile în același timp cu modulul ADC.

1. /\* DIFF = 0 -> selectam conversia sa fie single-ended (pag 464) \*/

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

3. ADC0->SC3 = 0x00;

4.

5. /\* dorim ca, conversia sa fie continua pentru a putea folosi si intreruperi in acelasi timp \*/

6. ADC0->SC3 |= ADC\_SC3\_ADCO\_MASK;

7.

8. /\* selecteaza canalul de input ca fiind PTC2 (la care am conectat senzorul) \*/

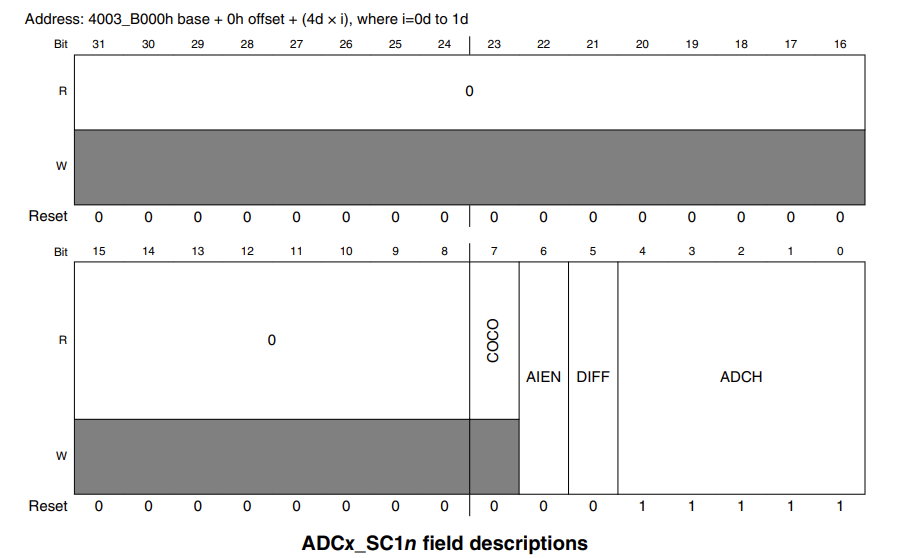
9. ADC0->SC1[0] |= ADC\_SC1\_ADCH(ROTATION\_SENSOR);

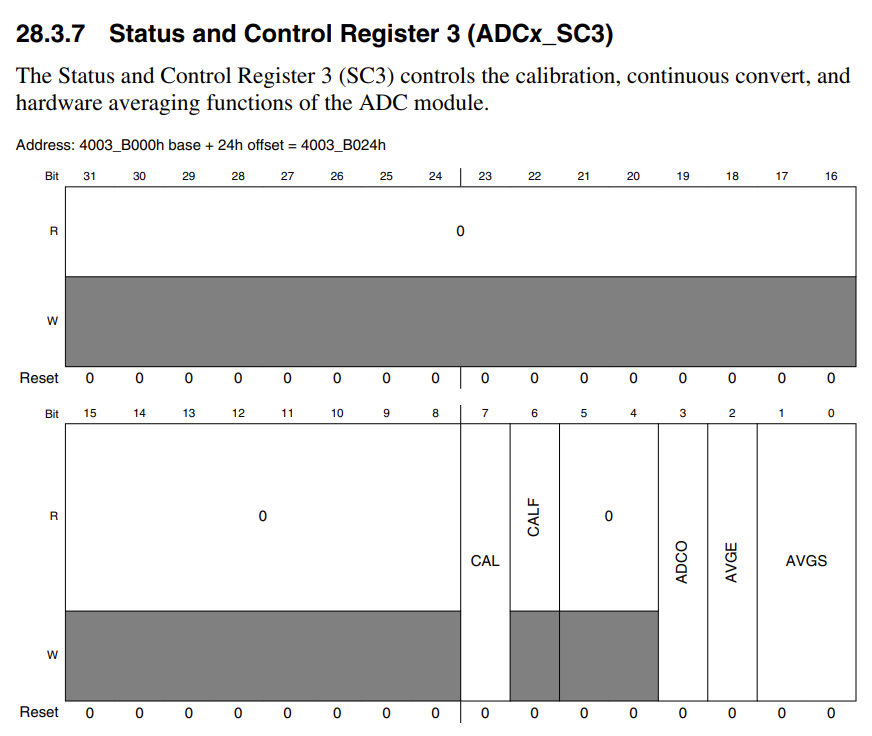
10.

11.

12. /\* Apeleaza o intrerupere dupa ce o conversie completa a fost realizata \*/

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





În continuare, am implementat funcția care citește o conversie făcută de modulul ADC și o procesează. Astfel, configurăm modulul să citească date de pe canalul pe care este conectat senzorul:

1. /\*configurare canal senzor de rotatie\*/

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

3. ADC0->SC3 = 0x00;

4. ADC0->SC3 |= ADC\_SC3\_ADCO\_MASK;

5. ADC0->SC1[0] |= ADC\_SC1\_ADCH(ROTATION\_SENSOR);

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

Citim valoarea din registrul ADC0\_R[0] și apoi o transformăm în char pentru a o putea trimite prin UART către aplicația host:

În functia de tratare a întreruperii vom prelua valoarea convertită de modulul ADC (prin funcția getRotationSensorValue) iar în funcție de această valoare vom aprinde ledurile corespunzătoare de pe breadboard.

1. /\*citirea valorii transmise de senzor\*/

2. analogValue = (uint16\_t)ADC0->R[0];

3. analogValue = analogValue\*300/65535;

4. rez = analogValue;

5.

6. /\*afisarea valorii obtinute\*/

7. while(analogValue!=0)

8. {

9. charValue[index] = analogValue%10 + 0x30;

10. analogValue = analogValue/10;

11. index = index + 1;

12. }

13.

14. for(i = 0; i < index ; i++)

15. UART0\_Transmit(charValue[index - i - 1]);

16.

17. UART0\_Transmit(0x0A);

18. UART0\_Transmit(0x0D);

19.

Funcția de întrerupere va manageria citirea valorii convertită din registru și apreinderea led-urilor necesare pe breadboard:

1. void ADC0\_IRQHandler(void)

2. {

3. uint16\_t value;

4. value = getRotationSensorValue();

5.

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

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

8.

9. if(value == 0)

10. {

11. closeLed(LED\_PIN4);

12. closeLed(LED\_PIN5);

13. closeLed(LED\_PIN12);

14. }

15.

16. if(value > 0 && value < 100)

17. {

18. openLed(LED\_PIN4);

19. closeLed(LED\_PIN5);

20. closeLed(LED\_PIN12);

21. }

22.

23. if(value >= 100 && value < 200)

24. {

25. openLed(LED\_PIN4);

26. openLed(LED\_PIN5);

27. closeLed(LED\_PIN12);

28. }

29.

30. if(value >= 200 && value <= 300)

31. {

32. openLed(LED\_PIN4);

33. openLed(LED\_PIN5);

34. openLed(LED\_PIN12);

35. }

36.

# Descriere interfață grafică

Interfața grafică a fost realizată în limbajul python folosind PySide6 și Matplotlib pentru a crea un barplot actualizat din valorile primite de la senzorul de rotație folosind o comunicare serială. Ea oferă două secțiuni principale:

* Una reprezentând graficul cu valorile primite de la senzor
* Cealaltănumele membrilor echipei precum și un buton pentru a schimba ordinea ledurilor de pe placă. Apăsând butonul se va apela funcția:

1. def send\_to\_microcontroller(self):

2.         # Sending command to the microcontroller

3.         command = b'1'

4.         self.serObj.write(command)

Se stabilește o conexiune serială cu un dispozitiv extern, microcontroller ul, configurată pentru a citi date de pe portul COM7.

1. 1. self.serObj = serial.Serial('COM7')

2. 2.         self.serObj.baudrate = 9600

**Plotarea graficului**

* Se utilizează Matplotlib pentru a crea un grafic de bare pe care se vor afișa valorile primite de la senzor.
* **self.figure** este instanțiat ca un obiect **Figure** din Matplotlib, reprezentând zona de plotare.
* Se creează un subplot (**self.ax**) în cadrul acestei figuri, pe care se vor desena barele graficului.
* Inițial, se afișează un grafic de bare gol, fără date, în cadrul ferestrei.

1. # Matplotlib plot

2.         self.figure = Figure(figsize=(5, 4), dpi=100) # it sets the figure size to be 5 inches wide and 4 inches tall

3. # DPI (dots per inch) determines the resolution or quality of the figure when it's displayed or saved as an image. A higher DPI means a higher resolution. In this case, it's set to 100.

4.         self.canvas = FigureCanvas(self.figure)

5.         self.ax = self.figure.add\_subplot(111)

6.         self.ax.set\_ylabel('Rotation Values')

7.         self.ax.set\_ylim(0, 300)  # Set y-axis limits from 0 to 300

8.

9.         self.rot\_values = []  # Store received values

10.         self.bar\_positions = []  # Store bar positions

11.         self.num\_values\_to\_display = 20  # Number of values to display on the plot

12.

13.         for i in range(self.num\_values\_to\_display):

14.             self.rot\_values.append(0)

15.             self.bar\_positions.append(i + 1)

16.

17.         self.bars = self.ax.bar(self.bar\_positions, self.rot\_values)  # Initial empty bar plot

18.         self.canvas.draw()

**Actualizarea graficului**

* **update\_plot\_data()** este apelată periodic de un timer (**QTimer**) pentru a actualiza datele și a re-desena graficul.

1. # Timer for updating graph data

2.         self.timer = QTimer(self)

3.         self.timer.setInterval(1)

4.         self.timer.timeout.connect(self.update\_plot\_data)

5.         self.timer.start()

* Se citesc 50 de valori de la senzor, dar se folosește doar ultima valoare citită pentru a actualiza graficul.
* Valorile citite sunt stocate într-un vector **rot\_values** și sunt afișate sub formă de grafic de bare.
* Se schimbă culoarea fiecărei bare în funcție de valoarea sa: verde pentru valori <= 100, galben pentru valori între 100 și 200, și roșu pentru valori > 200.
* De asemenea, se ascund etichetele și marcajele axei x folosind **self.ax.set\_xticks([])** pentru a elimina ambele.

1. def update\_plot\_data(self):

2.         # because the senzor send more values, so to display in right way we read 50 values from senzor, but only one (the last value) we use

3.         for \_ in range(50):

4.             received\_string = self.serObj.readline()

5.

6.         received\_string = received\_string.decode('utf', errors='replace').strip("\n").strip("\r")

7.

8.         try:

9.             rot = int(received\_string)

10.

11.             self.rot\_values.pop(0) # remove the first value from array of rot\_values

12.             self.rot\_values.append(rot) # add the last value readed (the last from 50 values)

13.             for i, bar in enumerate(self.bars):

14.                 bar.set\_height(self.rot\_values[i])

15.                 if self.rot\_values[i] <= 100:

16.                     bar.set\_color('green')

17.                 elif 100 < self.rot\_values[i] <= 200:

18.                     bar.set\_color('yellow')

19.                 else:

20.                     bar.set\_color('red')

21.

22.             self.ax.set\_xticks([]) # Hide both the labels and ticks

23.             self.canvas.draw()

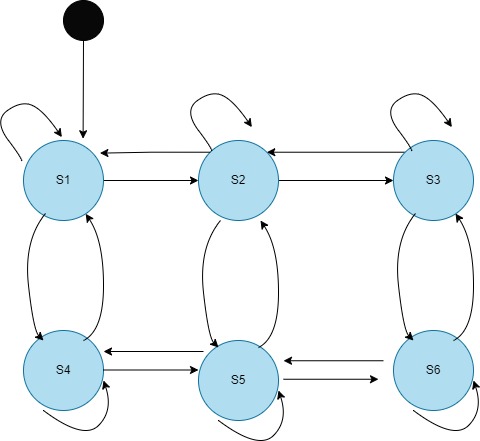
24.             print(rot)

25.

26.         except (ValueError, IndexError):

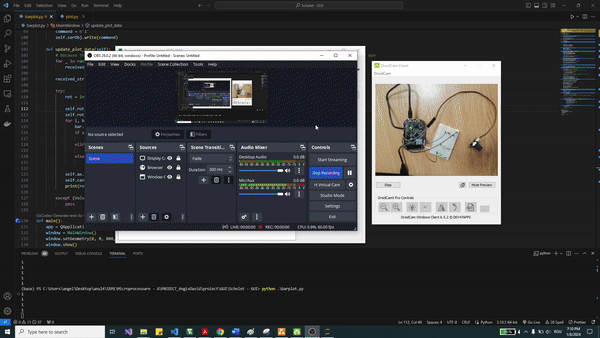
27.             pass

# Diagrama de stări a sistemului

*Figura 7. Diagrama de stări a sistemului*

# Rezultat proiect



# Dificultăți întâmpinate

În timpul realizării acestui proiect, am întâmpinat mai multe dificultăți, precum:

* Nu stiam cum sa conectam ledurile pe breadboard. Rezolvare : Majoritatea becurilor LED au polaritate, ceea ce înseamnă că trebuie conectate corect pentru a funcționa. În general, pentru un LED:

**Anodul** este partea pozitivă și se conectează la sursa de tensiune sau la polul pozitiv al sursei de alimentare; poate fi identificat adesea prin sârmă mai lungă.

**Catodul** este partea negativă și se conectează la pământ (ground) sau la polul negativ al sursei de alimentare; este partea mai scurtă a sârmei.

* Dupa ce am terminat de implementat atat modulul de PIT cat si modulul de ADC0, daca le rulam separate functionau amandoua, dar daca le rulam impreuna, nu mai mergea modulul de PIT, doar cel de ADC0. Rezolvare: am setat perioritatea handle-relor de intrerupere astfel incat intreruperile venite de la PIT sa aiba o prioritate mai mare (prioritatea 5) iar intreruperile venite de la ADC0 sa aiba o prioritate mai mica (prioritatea 6).

# Referințe

[1] <https://www.seeedstudio.com/blog/2020/01/06/how-to-use-a-breadboard-wiring-circuit-and-arduino-interfacing/#:~:text=Plug%20the%20LED's%20longer%20lead,main%20part%20of%20the%20breadboard.&text=Plug%20one%20of%20the%20leads,middle%20channel%20of%20the%20breadboard>.

[2] FRDM-KL25Z\_ReferenceManual

[3] FRDM-KL25Z\_Schematics

[4] FRDM-KL25Z\_UserManual

|  |  |
| --- | --- |
|  |  |
|  |  |