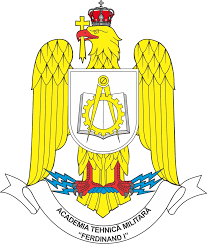
**Academia Tehnică Militară**



**,,Ferdinand I”**

|  |
| --- |
|  |

**-Proiect-**

**Sisteme Specializate cu Microprocesoare**

**Istrate Stefan gr. C114B**

**Pascu Andrei gr. C114B**

**București**

**2024**

**CUPRINS**

[**1.** **Scopul proiectului** 3](#_Toc156409267)

[**2.** **Platforma de dezvoltare FRDM-KL25Z** 3](#_Toc156409268)

[**3.** **Prezentare senzori** 4](#_Toc156409269)

[**3.1. Senzorul analog de sunet DFR0034** 4](#_Toc156409270)

[**4.** **Conectare senzori** 5](#_Toc156409271)

[**5.** **Descriere program principal** 7](#_Toc156409272)

[**5.1.** **Functia main** 7](#_Toc156409273)

[**5.2. Inițializarea modulului UART (*uart.c*)** 7](#_Toc156409274)

[**5.3. Inițializarea modulului PIT(*Pit.c*)** 13](#_Toc156409275)

[**5.4. Inițializarea modulului GPIO (*gpio.c*)** 16](#_Toc156409276)

[**5.5. Inițializarea modulului ADC și prelucrarea datelor înregistrate de către senzor (*Adc.c*)** 19](#_Toc156409277)

[**6.** **Descrierea programului de afisare grafica a valoriilor** 26](#_Toc156409278)

[**6.1 Rezultate rulare program** 28](#_Toc156409279)

[**7.** **Probleme întâmpinate și sugestii** 29](#_Toc156409280)

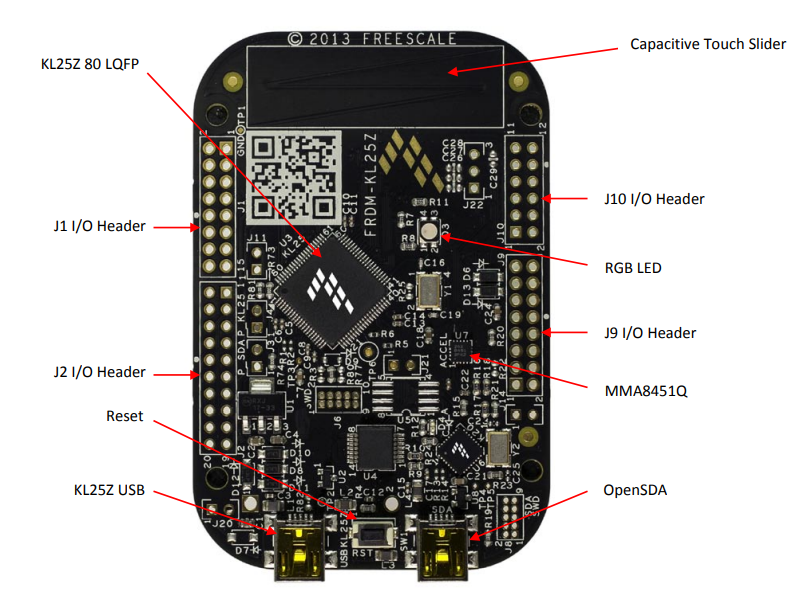
[**8.** **Github** 30](#_Toc156409281)

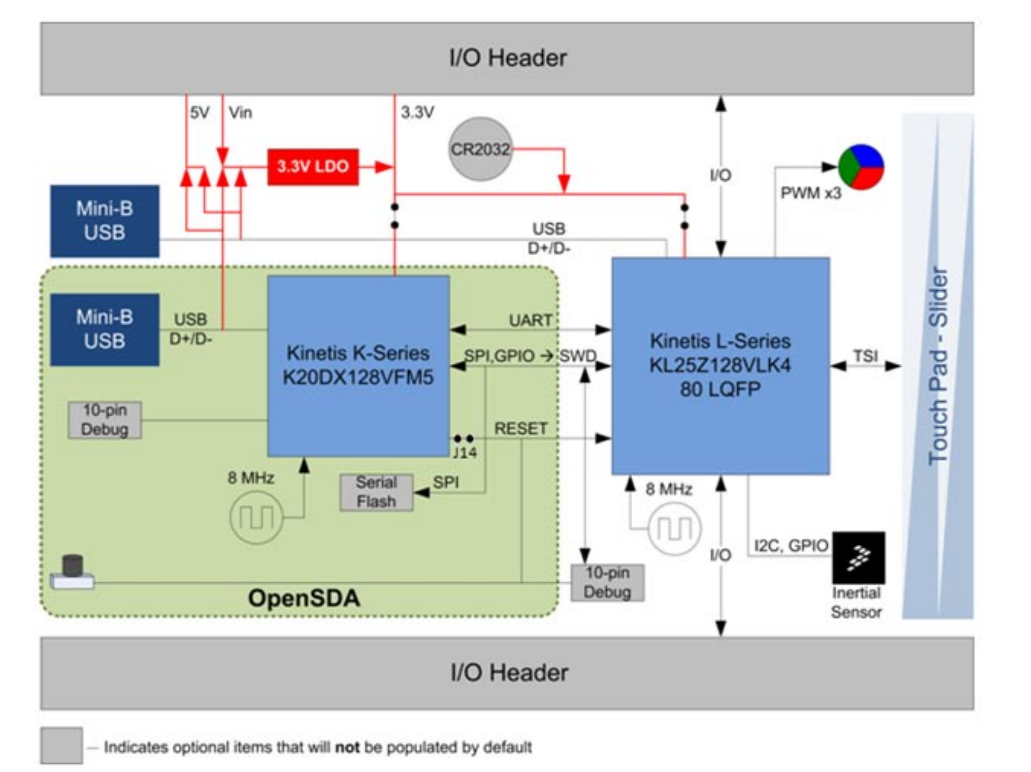
# **Scopul proiectului**

Scopul acestui proiect este de a aprinde 3 led-uri externe puse pe un breadboard în funcție de prelucrarea datelor obținute de la senzorul de sunet. De asemenea, se vor afișa grafic valorile obținute în urma prelucrării.

Astfel, în funcție de pragurile alese de noi empiric, valorile dezvoltate de către senzorul de sunet vor aprinde (in functie de intervalul unde se incardreaza) ledul verde (pentru valorile mici), verde si galben (pentru valorile medii) si toate 3 (pentru valorile mari). Valorile înregistrate se vor transmite prin UART către PC și se vor afișa printr-un grafic.

# **Platforma de dezvoltare FRDM-KL25Z**



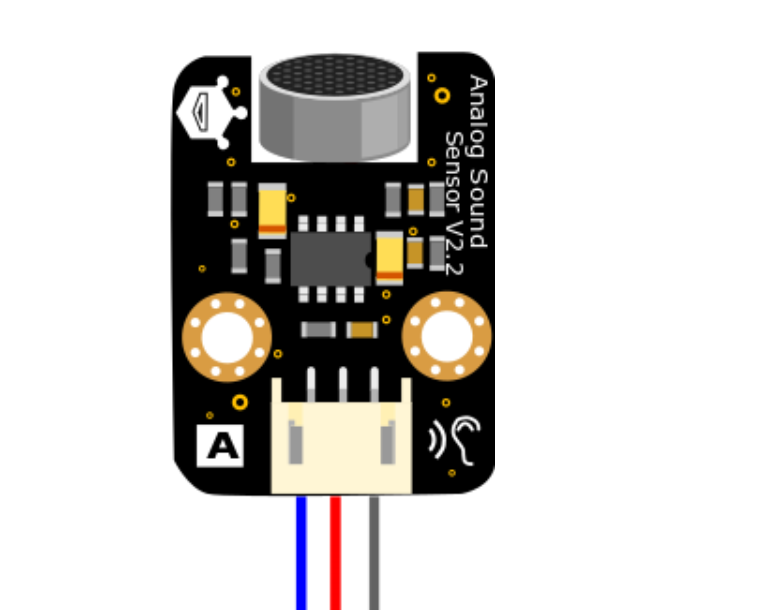


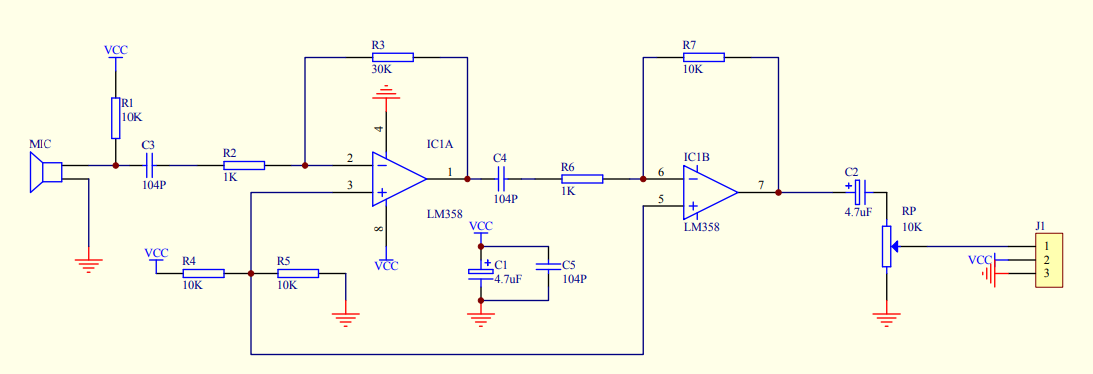
# **Prezentare senzori**

## **3.1. Senzorul analog de sunet DFR0034**

Senzorul DFR0034 este un senzor analog ce detectează zgomotul din mediul în care este pus.

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





Specificații:

* Tensiune de alimentare: 3.3V până la 5V;
* Detectează intensitatea sunetului;
* Mărime: 22x30 mm.

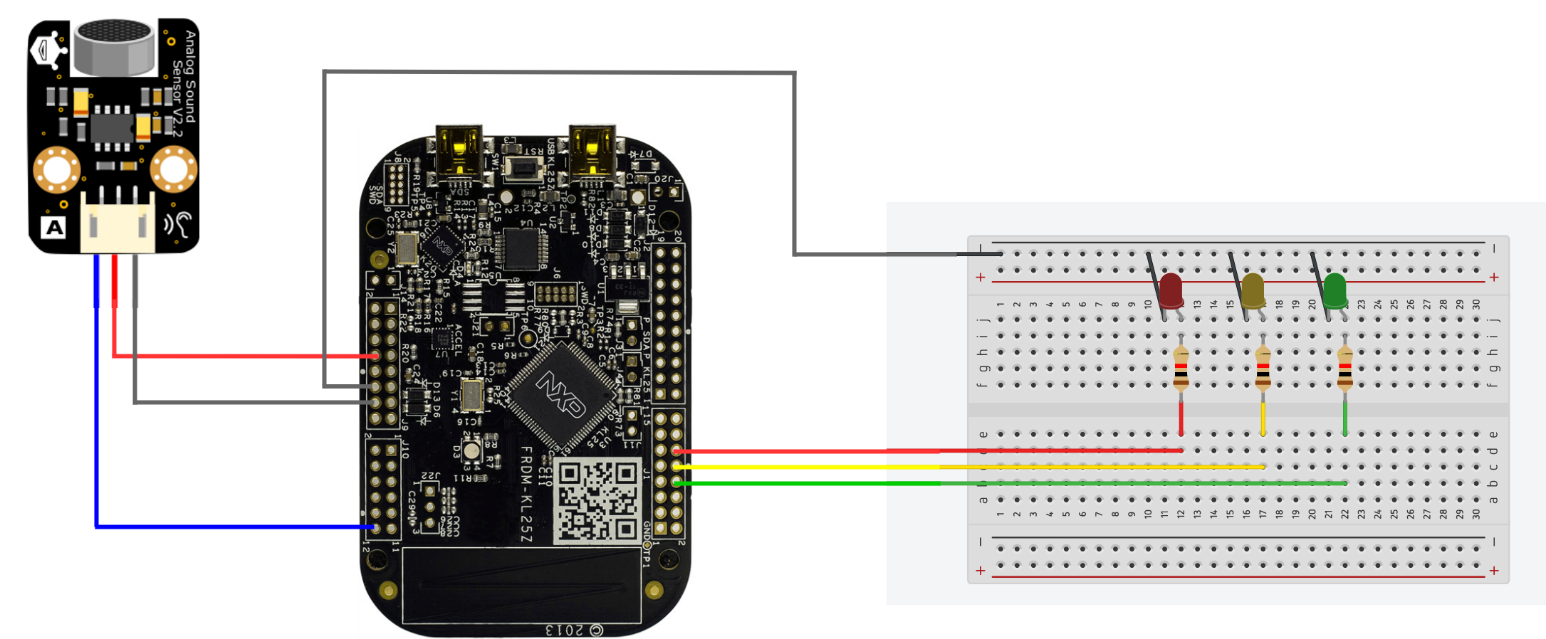
# **Conectare senzori**

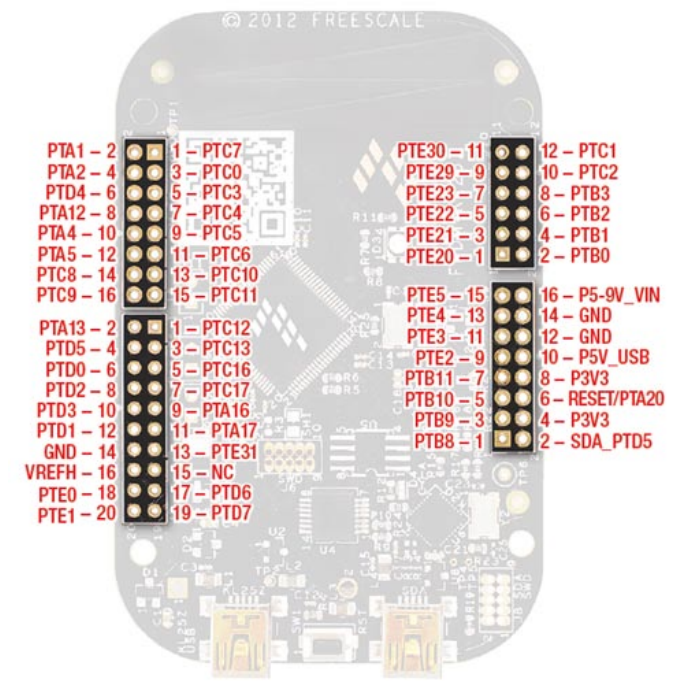
Senzorul de sunet:

* Firul roșu (VCC) se conectează la pinul J9 08 (P3V3);
* Firul negru (GND) se conectează la pinul J9 14 (GND);
* Firul verde (output sensor value) se conectează la pinul J10 12 (PTC1).

Led-uri externe:

* Led-ul rosu se conecteaza la pinul J1 12 (PTA5)
* Led-ul galben se conecteaza la pinul J1 10 (PTA4)
* Led-ul verde se conecteaza la pinul J1 8 (PTA12)





# **Descriere program principal**

## **Functia main**

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

1. #include "Adc.h"

2. #include "Uart.h"

3. #include "Pit.h"

4. #include "gpio.h"

5.

6. int main(void) {

7. UART0\_Initialize(38400);

8. PIT\_Init();

9. ADC0\_Init();

10. RGBLed\_Init();

11.

12. for(;;){

13.

14. }

15.

16. }

Logica programului este următoarea:

* se inițializează modulul UART prin apelarea funcției UART0\_Init cu baudrate-ul de 38400;
* se inițializează modulul ADC prin apelarea funcției ADC0\_Init;
* se inițializează ledurile (atat cele 3 externe, cat si cele 3 RGB de pe placa FRDM) prin apelarea funcției RGBLed\_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 datorită datelor primite de la senzori.

## **5.2. Inițializarea modulului UART (*uart.c*)**

Vom folosi modulul UART0 pentru comunicația serială cu PC prin cablul USB.

Prin funcția UART0\_Initialize configurăm următoarele:

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

2.

3. uint16\_t osr = 3;

4. uint16\_t sbr;

5.

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

7.

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

9. /\* PTA1 - receptie UART0 \*/

10. /\* PTA2 - transmisie UART0 \*/

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

12. /\* Dezactivare receptor si emitator pentru realizarea configuratiilor\*/

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

14.

15. /\* Setarea sursei de ceas pentru modulul UART la 48MHz \*/

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

17.

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

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

20.

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

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

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

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

25.

26.

27. /\* Setarea baud rate-ului si a ratei de supraesantionare \*/

28. sbr = (uint16\_t)((DEFAULT\_SYSTEM\_CLOCK)/(baud\_rate \* (osr+1))/4);

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

30. uint8\_t temp = UART0->BDH & ~(UART0\_BDH\_SBR(0x1F));

31. UART0->BDH = temp | UART0\_BDH\_SBR(((sbr & 0x1F00)>> 8));

32. UART0->BDL = (uint8\_t)(sbr & UART\_BDL\_SBR\_MASK);

33. UART0->C4 |= UART0\_C4\_OSR(osr);

34.

35. /\* Setarea numarului de biti de date la 8 si fara bit de paritate \*/

36. UART0->C1 = 0;

37.

38. /\* Transmisie incepand cu MSB \*/

39. UART0->S2 = UART0\_S2\_MSBF(1);

40.

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

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

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

44.

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

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

47.

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

49.

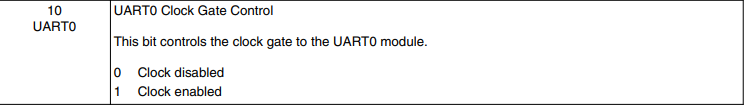
50. NVIC\_EnableIRQ(UART0\_IRQn);

51.

52. }

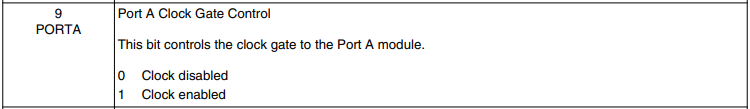
În registrul SIM\_SCGC4 (System Clock Gating Control Register 4) setăm pe 1 câmpul UART0 (bitul 10) pentru activarea ceasului pentru acest modul.

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



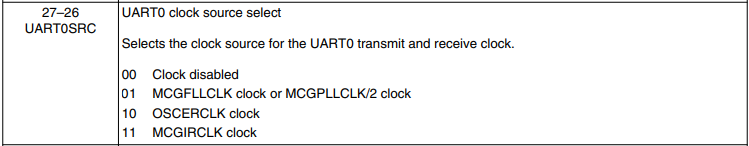
În registrul SIM\_SCGC5 (System Clock Gating Control Register 5) setăm pe 1 câmpul PORTA (bitul 9) pentru activarea ceasului acestui port.

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



În registrul SIM\_SOPT2 (System Options Register 2) setăm pe 0b01 câmpul UART0SRC (biții 27-26) pentru selectarea ca ceas al modulului MCGFLLCLK.

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



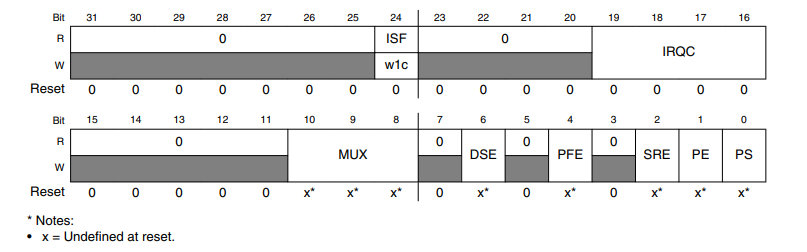
Fiecare pin pune la dispoziție mai multe funcționalități la care avem acces prin intermediul multiplexării. Astfel, configurăm pinul 1 din portul A pe modulul de UART0\_RX și pinul 2 din portul A pe modulul de UART0\_TX prin selectarea alternativei 2 folosindu-ne de registrul de control.

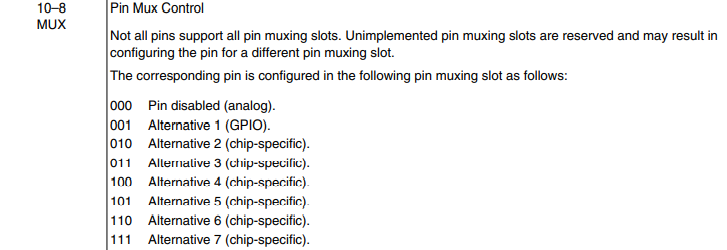
PORTA->PCR[1] = ~PORT\_PCR\_MUX\_MASK;

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

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

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

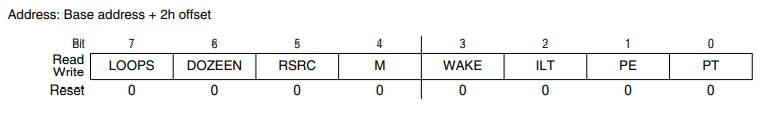


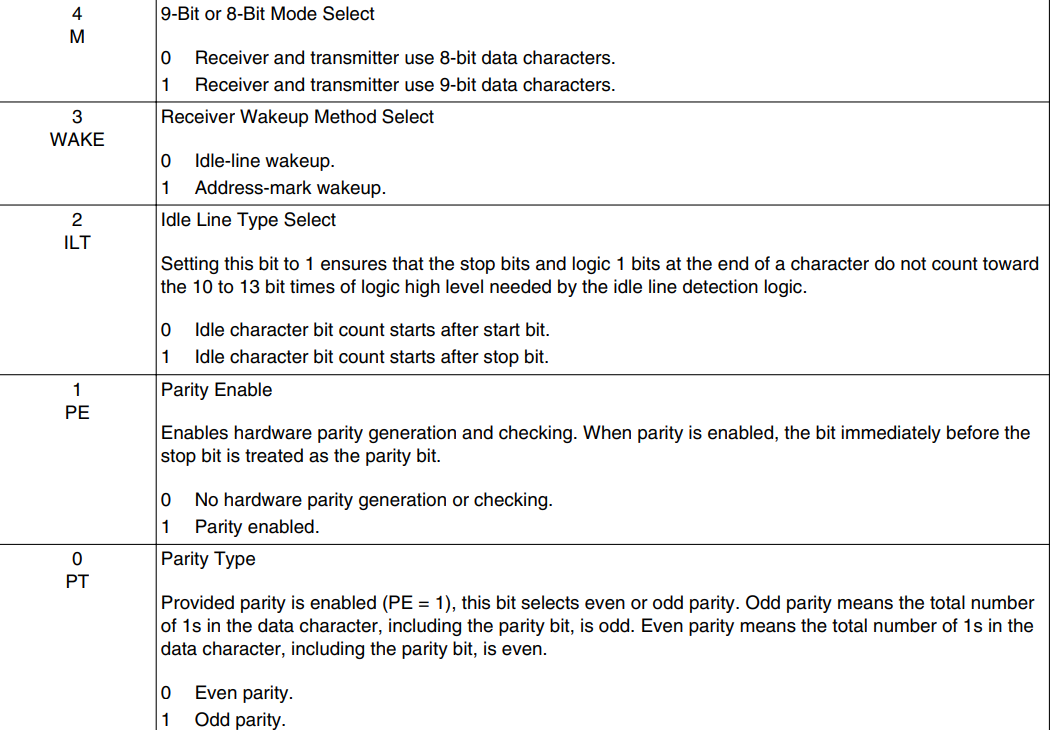




Setăm numărul de biți de date la 8 și fără biți de paritate.

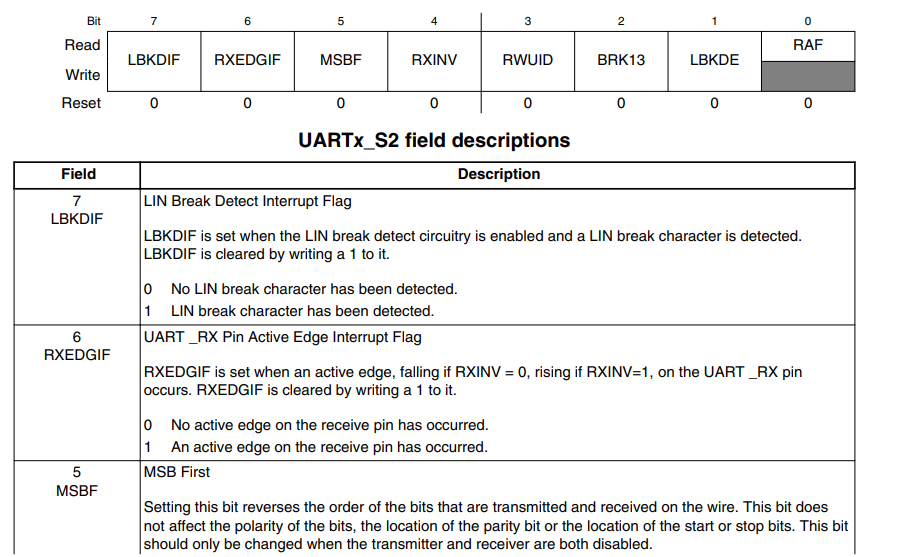
UART0->C1 = 0;



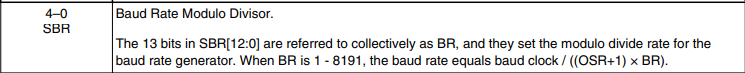


Setăm ca transmisia sa se faca MSB first.

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



Pentru baud rate-ul de 38400 trebuie sa configurăm câmpul SBR de la regiștrii UART0\_BDH și UART0\_BDL astfel încât să satisfacem urmatoarea formulă:



OSR (Over Sampling Ratio) are valoarea implicită 0b0011 adică 3.

uint16\_t osr = 3;

uint16\_t sbr;

(uint16\_t)((DEFAULT\_SYSTEM\_CLOCK)/(baud\_rate \* (osr+1))/4);

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

uint8\_t temp = UART0->BDH & ~(UART0\_BDH\_SBR(0x1F));

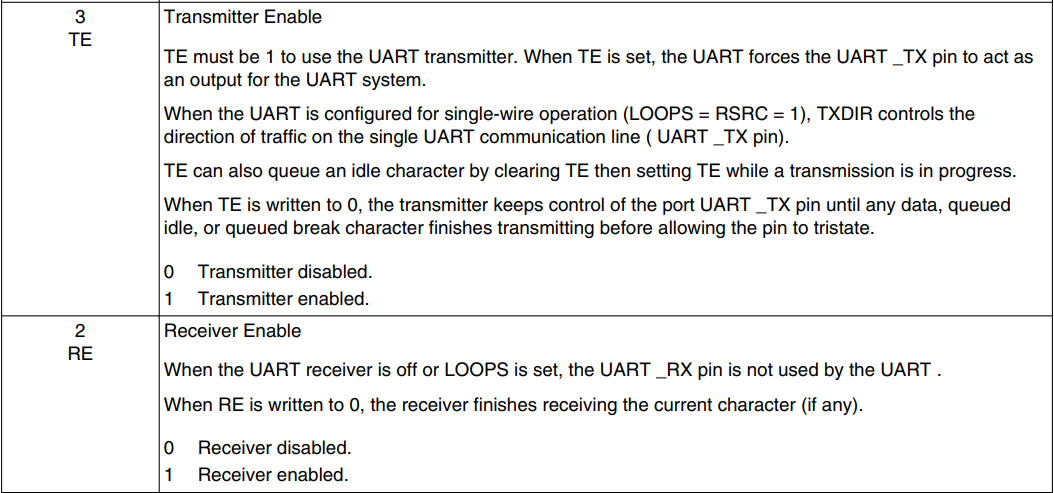
UART0->BDH = temp | UART0\_BDH\_SBR(((sbr & 0x1F00)>> 8));

UART0->BDL = (uint8\_t)(sbr & UART\_BDL\_SBR\_MASK);

UART0->C4 |= UART0\_C4\_OSR(osr);

Activăm transmiterea si receptia prin UART:

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

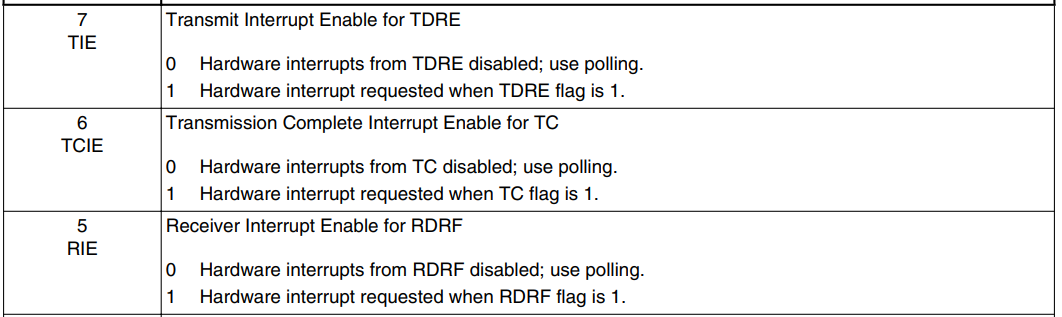


Dezactivăm întreruperea la transmisie și activăm întreruperea la recepție

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

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

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





Funcțiile UART0\_Transmit și UART0\_IRQHandler ,care inversează biții înainte de transmite și recepție întrucât se folosește MSB first:

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

2. /\* Utilizam masca TDRE pentru a verifica \*/

3. /\* disponibilitatea buffer-ului de transmisie \*/

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

5. data = (uint8\_t)((data & 0x01) << 7) |

6. (uint8\_t)((data & 0x02) << 5) |

7. (uint8\_t)((data & 0x04) << 3) |

8. (uint8\_t)((data & 0x08) << 1) |

9. (uint8\_t)((data & 0x10) >> 1) |

10. (uint8\_t)((data & 0x20) >> 3) |

11. (uint8\_t)((data & 0x40) >> 5) |

12. (uint8\_t)((data & 0x80) >> 7);

13.

14. UART0->D = data;

15. }

1. void UART0\_IRQHandler(void) {

2.

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

4. sens = UART0->D;

5. sens = (uint8\_t)((sens & 0x01) << 7) |

6. (uint8\_t)((sens & 0x02) << 5) |

7. (uint8\_t)((sens & 0x04) << 3) |

8. (uint8\_t)((sens & 0x08) << 1) |

9. (uint8\_t)((sens & 0x10) >> 1) |

10. (uint8\_t)((sens & 0x20) >> 3) |

11. (uint8\_t)((sens & 0x40) >> 5) |

12. (uint8\_t)((sens & 0x80) >> 7);

13. }

14. UART0->S1 &= ~UART0\_S1\_RDRF\_MASK;

15. }

16.

## **5.3. Inițializarea modulului PIT(*Pit.c*)**

Prin funcția PIT\_Init configurăm următoarele:

1. void PIT\_Init(void) {

2. /\* Activarea semnalului de ceas pentru perifericul PIT \*/

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

4. /\* Utilizarea semnalului de ceas pentru tabloul de timere \*/

5. PIT->MCR &= ~PIT\_MCR\_MDIS\_MASK;

6. /\* Oprirea decrementarii valorilor numaratoarelor in modul debug \*/

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

8.

9. /\* Setarea valoarea numaratorului de pe canalul 0 la o perioada de 0.1 secunde \*/

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

11.

12. /\* Activarea intreruperilor 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. /\* Setarea valoarea numaratorului de pe canalul 1 la o perioada de 0.711 secunde \*/

18. PIT->CHANNEL[1].LDVAL = 0x71C28E;

19.

20. /\* Activara ?ntreruperilor pe canalul 1 \*/

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

22. /\* Activarea timerului de pe canalul 1 \*/

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

24.

25. /\* Activarea intreruperii mascabile si setarea prioritatiis \*/

26. NVIC\_ClearPendingIRQ(PIT\_IRQn);

27. NVIC\_SetPriority(PIT\_IRQn,5);

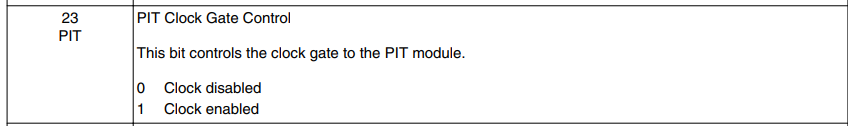
28. NVIC\_EnableIRQ(PIT\_IRQn);

29. }

30.

În registrul SIM\_SCGC6 (System Clock Gating Control Register 6) setăm pe 0b01 câmpul PIT (bitul 23) pentru a activa semnalul de ceas pentru perifericul PIT.

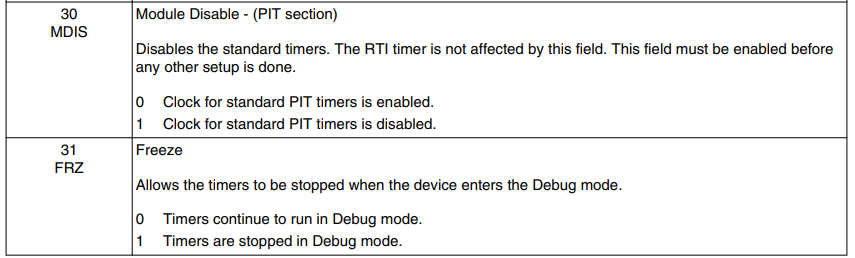
SIM->SCGC6 |= SIM\_SCGC6\_PIT\_MASK;



În registrul PIT\_MCR (PIT Module Control Register) setăm pe 0b00 câmpul MDIS (bitul 30) pentru utilizarea semnalului de ceas pentru tabloul de timere și pe 0b01 câmpul FRZ (bitul 31) pentru oprirea decrementării valorilor numărătoarelor în modul debug.

PIT->MCR &= ~PIT\_MCR\_MDIS\_MASK;

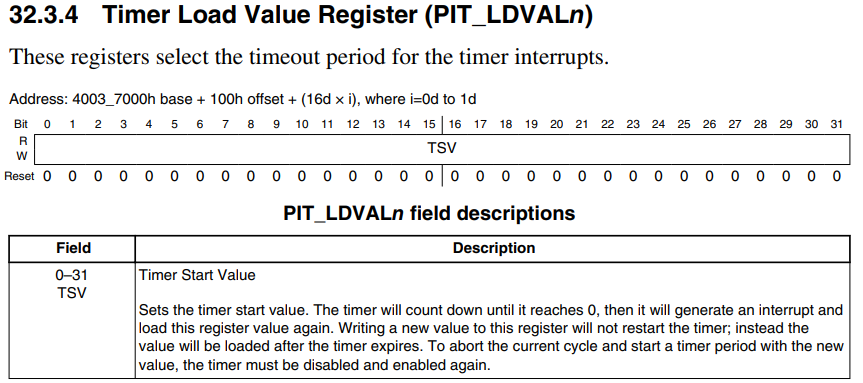
PIT->MCR |= PIT\_MCR\_FRZ\_MASK;



Setăm valoarea numărătorului de pe canalul 0 la o perioadă de 0.1 secunde prin atribuirea valorii 0xFFFFE (valoare folosita pentru sampling al semnalului analogic si de a transmite interefetei grafice valoarea citita) și pentru canalul 1 la o perioadă de 0.711 secunde prin atribuirea valorii 0x71C28E (valoare specificata in cerintele proiectului) registrului PIT\_LDVAL ( Timer Load Value Register).

PIT->CHANNEL[0].LDVAL = 0xFFFFE;

PIT->CHANNEL[1].LDVAL = 0x71C28E;



Valorile au fost calculate folosind formula: Load Value = Nr. sec. \* Bus Clock Freq - 1, unde Bus Clock Freq = 10485760 Hz și apoi convertită in hex.

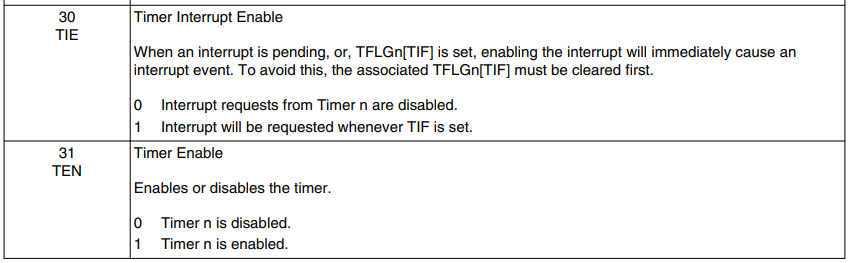
În registrul PIT\_TCTRL (Timer Control Register) setăm pe 0b01 câmpul TIE (bitul 30) pentru activarea întreruperilor și câmpul TEN (bitul 31) pentru activarea timerului pe canalul 0 și pe canalul 1.

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

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

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

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



1. void PIT\_IRQHandler(void) {

2.

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

4.

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

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

7.

8. }

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

10. state+=1;

11. state=state%4;

12.

13. if(sens==0){

14. switch (state) {

15. case 0:

16. /\* Alb \*/

17. GPIOB->PCOR |= (1<< SEQ\_RED\_LED\_PIN);

18. GPIOB->PCOR |= (1<< SEQ\_GREEN\_LED\_PIN);

19. GPIOD->PCOR |= (1<< SEQ\_BLUE\_LED\_PIN);

20. break;

21. case 1:

22. /\* Magenta \*/

23. GPIOB->PCOR |= (1<< SEQ\_RED\_LED\_PIN);

24. GPIOB->PSOR |= (1<< SEQ\_GREEN\_LED\_PIN);

25. GPIOD->PCOR |= (1<< SEQ\_BLUE\_LED\_PIN);

26. break;

27. case 2:

28. /\* Galben \*/

29. GPIOB->PCOR |= (1<< SEQ\_RED\_LED\_PIN);

30. GPIOB->PCOR |= (1<< SEQ\_GREEN\_LED\_PIN);

31. GPIOD->PSOR |= (1<< SEQ\_BLUE\_LED\_PIN);

32. break;

33. case 3:

34. /\* Stins \*/

35. GPIOB->PSOR |= (1 << SEQ\_RED\_LED\_PIN);

36. GPIOB->PSOR |= (1 << SEQ\_GREEN\_LED\_PIN);

37. GPIOD->PSOR |= (1 << SEQ\_BLUE\_LED\_PIN);

38. break;

39. }

40. }

41. if(sens==1)

42. {

43. switch (state) {

44. case 3:

45. /\* Alb \*/

46. GPIOB->PCOR |= (1<< SEQ\_RED\_LED\_PIN);

47. GPIOB->PCOR |= (1<< SEQ\_GREEN\_LED\_PIN);

48. GPIOD->PCOR |= (1<< SEQ\_BLUE\_LED\_PIN);

49. break;

50. case 2:

51. /\* Magenta \*/

52. GPIOB->PCOR |= (1<< SEQ\_RED\_LED\_PIN);

53. GPIOB->PSOR |= (1<< SEQ\_GREEN\_LED\_PIN);

54. GPIOD->PCOR |= (1<< SEQ\_BLUE\_LED\_PIN);

55. break;

56. case 1:

57. /\* Galben \*/

58. GPIOB->PCOR |= (1<< SEQ\_RED\_LED\_PIN);

59. GPIOB->PCOR |= (1<< SEQ\_GREEN\_LED\_PIN);

60. GPIOD->PSOR |= (1<< SEQ\_BLUE\_LED\_PIN);

61. break;

62. case 0:

63. /\* Stins \*/

64. GPIOB->PSOR |= (1 << SEQ\_RED\_LED\_PIN);

65. GPIOB->PSOR |= (1 << SEQ\_GREEN\_LED\_PIN);

66. GPIOD->PSOR |= (1 << SEQ\_BLUE\_LED\_PIN);

67. break;

68. }

69. }

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

71. }

72. }

73.

In functia de intrerupere a modulului PIT au loc 2 scenarii, unul pentru fiecare canal:  
Intreruperea canalului 0 are rolul de a da enable intreruperilor modulului ADC (pentru a nu inregistra si transmite valori continuu, ci doar la o perioada dorita de noi).  
Intreruperea canalului 1 are rolul de a aprinde led-urile de pe placa FRDM conform secventei specificate in cerintele proiectului. Variabila *sens* specifica sensul de parcurgere al secventei de led-uri. Aceasta variabila este modificata in functia UART0\_IRQHandler, in functie de input-ul din interfata grafica.

## **5.4. Inițializarea modulului GPIO (*gpio.c*)**

Pentru aplicația noastră, avem nevoie doar de culorile roșu si verde.

1. #define RED\_LED\_PIN (5) /\* PORT A \*/

2. #define YELLOW\_LED\_PIN (4) /\* PORT A \*/

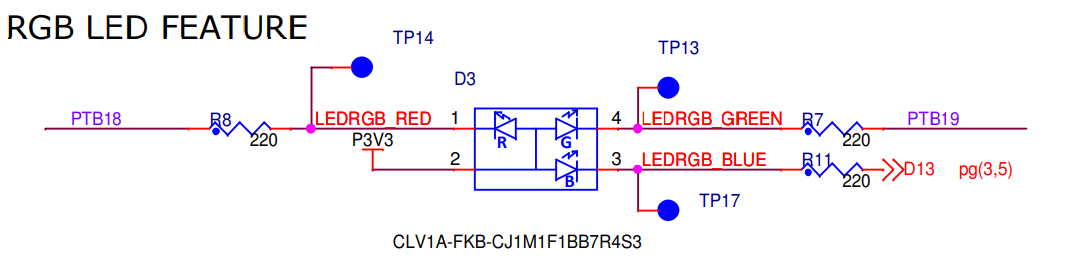
3. #define GREEN\_LED\_PIN (12) /\* PORT A \*/

4.

5. #define SEQ\_RED\_LED\_PIN (18) /\* PORT B \*/

6. #define SEQ\_GREEN\_LED\_PIN (19) /\* PORT B \*/

7. #define SEQ\_BLUE\_LED\_PIN (1) /\* PORT D \*/



Activarea semnalului de ceas pentru pinii folosiți în cadrul led-urilor RGB si cele 3 externe.

1. SIM\_SCGC5 |= SIM\_SCGC5\_PORTA\_MASK | SIM\_SCGC5\_PORTB\_MASK | SIM\_SCGC5\_PORTD\_MASK;



Utilizare GPIO ca variantă de multiplexare pentru pinii corespunzători.

1. PORTA->PCR[RED\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

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

3.

4. PORTA->PCR[YELLOW\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

5. PORTA->PCR[YELLOW\_LED\_PIN] |= PORT\_PCR\_MUX(1);

6.

7. PORTA->PCR[GREEN\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

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

9.

10. PORTB->PCR[SEQ\_RED\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

11. PORTB->PCR[SEQ\_RED\_LED\_PIN] |= PORT\_PCR\_MUX(1);

12.

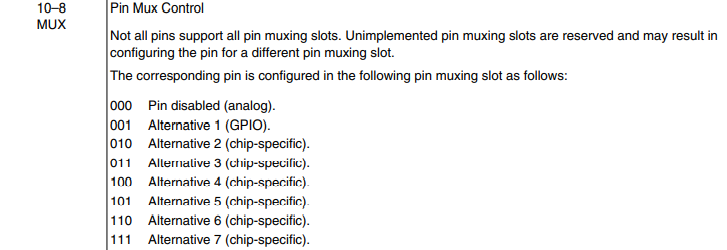
13. PORTB->PCR[SEQ\_GREEN\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

14. PORTB->PCR[SEQ\_GREEN\_LED\_PIN] |= PORT\_PCR\_MUX(1);

15.

16. PORTD->PCR[SEQ\_BLUE\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

17. PORTD->PCR[SEQ\_BLUE\_LED\_PIN] |= PORT\_PCR\_MUX(1);



Configurare pini pe post de output.

1. GPIOA\_PDDR |= (1<<RED\_LED\_PIN);

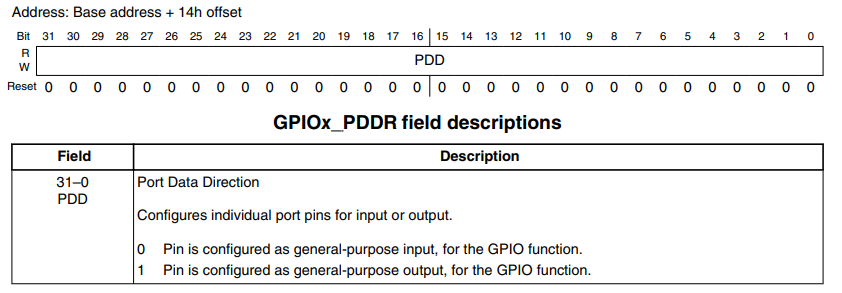
2. GPIOA\_PDDR |= (1<<YELLOW\_LED\_PIN);

3. GPIOA\_PDDR |= (1<<GREEN\_LED\_PIN);

4. GPIOB\_PDDR |= (1<<SEQ\_RED\_LED\_PIN);

5. GPIOB\_PDDR |= (1<<SEQ\_GREEN\_LED\_PIN);

6. GPIOD\_PDDR |= (1<<SEQ\_BLUE\_LED\_PIN);



sau cele de pe placa FRDM).

1. GPIOA\_PCOR |= (1<<RED\_LED\_PIN);

2. GPIOA\_PCOR |= (1<<YELLOW\_LED\_PIN);

3. GPIOA\_PCOR |= (1<<GREEN\_LED\_PIN);

/\*SEQ\_X\_LED reprezinta ledurile RGB de pe placa FRDM – active pe 1 logic \*/

4. GPIOB\_PSOR |= (1<<SEQ\_RED\_LED\_PIN);

5. GPIOB\_PSOR |= (1<<SEQ\_GREEN\_LED\_PIN);

6. GPIOD\_PSOR |= (1<<SEQ\_BLUE\_LED\_PIN);

## **5.5. Inițializarea modulului ADC și prelucrarea datelor înregistrate de către senzor (*Adc.c*)**

Setarea canalelor specifice pinilor la care sunt conectați senzorii

1. #define ADC\_CHANNEL (15) /\* PORT C PIN 1 \*/





Inițializarea modulului ADC.

1. void ADC0\_Init(void) {

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 16 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. /\* Setarea frecventei mai mica sau egale cu 4MHz (PG. 494) \*/

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

16.

17. /\* DIFF = 0 --> Conversii single-ended (PG. 464) \*/

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

19. ADC0->SC3 = 0x00;

20.

21. /\* Selectarea modului de conversii continue, \*/

22. /\* pentru a-l putea folosi in tandem cu mecanismul de intreruperi \*/

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

24.

25. /\* Activarea subsistemului de conversie prin aproximari succesive pe un anumit canal (PG.464) \*/

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

27.

28.

29. /\* Permite intreruperi dupa o conversie completa \*/

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

31.

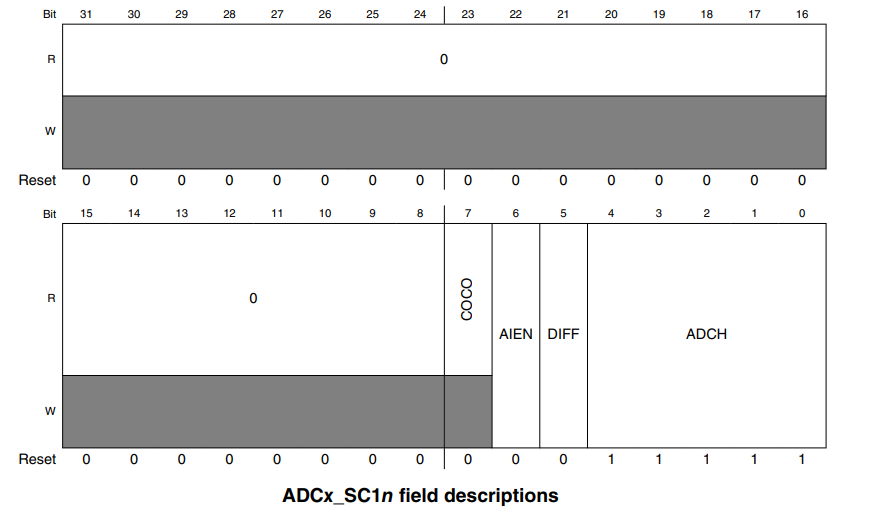
32. NVIC\_ClearPendingIRQ(ADC0\_IRQn);

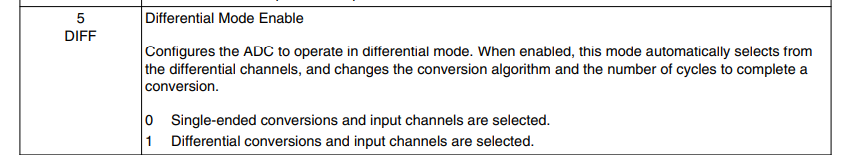
33. NVIC\_EnableIRQ(ADC0\_IRQn);

34. }

Activăm conversiile single-ended.

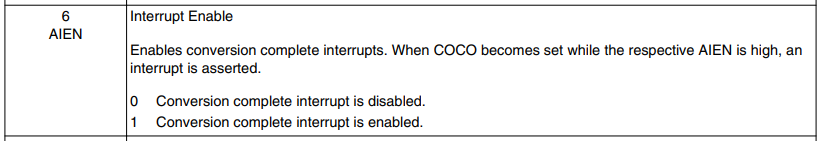
1. ADC0->SC1[0] = 0x00;





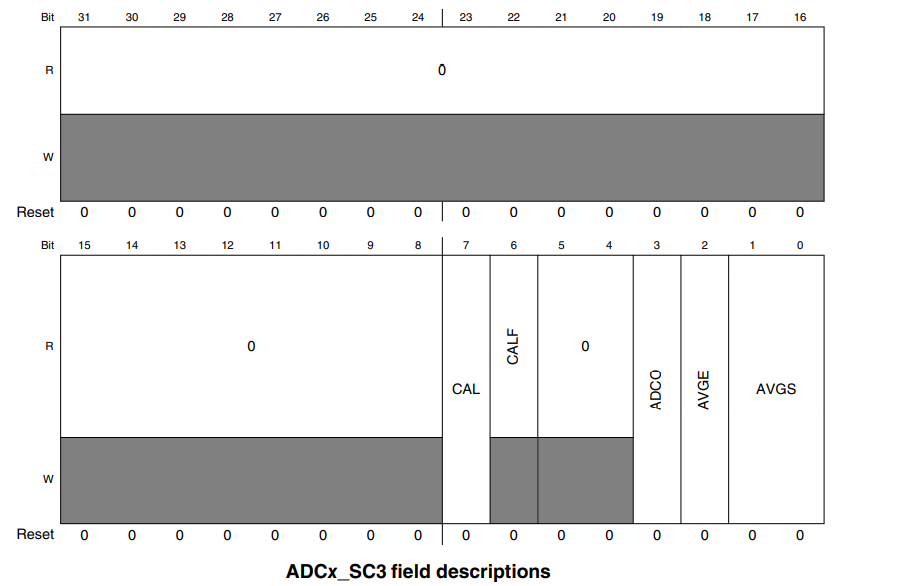
Activăm întreruperile la terminarea conversiilor.

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



Ne asigurăm că registrul de status și control 3 este 0 înainte de a face calibrările necesare.

1. ADC0->SC3 = 0x00;

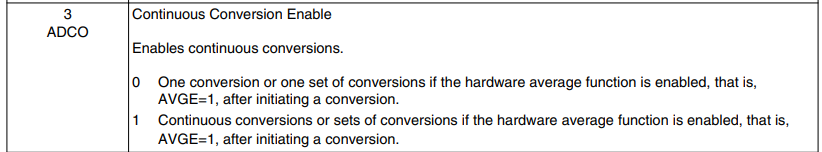


Activăm conversiile continue.

1. /\* Selectarea modului de conversii continue, \*/

2. /\* pentru a-l putea folosi in tandem cu mecanismul de intreruperi \*/

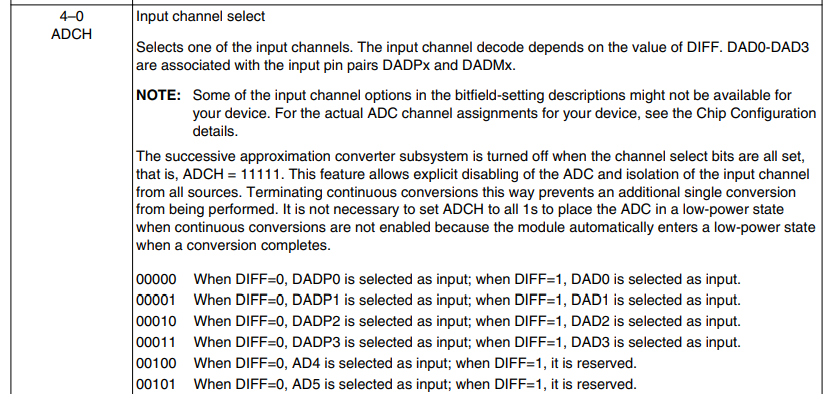
3. ADC0->SC3 |= ADC\_SC3\_ADCO\_MASK;



Selectăm canalul specific senzorului (15 pentru senzorul de sunet ).

1. /\* Activarea subsistemului de conversie prin aproximari succesive pe un anumit canal (PG.464) \*/

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



Funcția de calibrare pentru conversii ale modulului ADC, conform manualului de referință al platformei:

1. int ADC0\_Calibrate(void) {

2.

3. uint16\_t calibration\_var;

4. /\* ===== For best calibration results ===== \*/

5.

6. ADC0\_CFG1 |= ADC\_CFG1\_MODE(3) | /\* 16 bits mode \*/

7. ADC\_CFG1\_ADICLK(1)| /\* Input Bus Clock divided by 2 \*/

8. ADC\_CFG1\_ADIV(3); /\* Clock divide by 8 \*/

9.

10. /\* The calibration will automatically begin if the SC2[ADTRG] is 0. (PG. 495) \*/

11. ADC0->SC2 &= ~ADC\_SC2\_ADTRG\_MASK;

12.

13. /\* Set hardware averaging to maximum, that is, SC3[AVGE]=1 and SC3[AVGS]=0x11 for an average of 32 (PG. 494) \*/

14. ADC0->SC3 |= (ADC\_SC3\_AVGE\_MASK | ADC\_SC3\_AVGS(3));

15.

16. /\* To initiate calibration, the user sets SC3[CAL] (PG. 495) \*/

17. ADC0->SC3 |= ADC\_SC3\_CAL\_MASK;

18.

19. /\* At the end of a calibration sequence, SC1n[COCO] will be set (PG. 495) \*/

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

21.

22. /\* At the end of the calibration routine, if SC3[CALF] is not \*/

23. /\* set, the automatic calibration routine is completed successfully. (PG. 495) \*/

24. if(ADC0->SC3 & ADC\_SC3\_CALF\_MASK){

25. return (1);

26. }

27.

28. /\* ====== CALIBRATION FUNCTION (PG.495) ===== \*/

29.

30. /\* 1. Initialize or clear a 16-bit variable in RAM. \*/

31. calibration\_var = 0x0000;

32.

33. /\* 2. Add the plus-side calibration results CLP0, CLP1, CLP2, CLP3, CLP4, and CLPS to the variable. \*/

34. calibration\_var += ADC0->CLP0;

35. calibration\_var += ADC0->CLP1;

36. calibration\_var += ADC0->CLP2;

37. calibration\_var += ADC0->CLP3;

38. calibration\_var += ADC0->CLP4;

39. calibration\_var += ADC0->CLPS;

40.

41. /\* 3. Divide the variable by two. \*/

42. calibration\_var /= 2;

43.

44. /\* 4. Set the MSB of the variable. \*/

45. calibration\_var |= 0x8000;

46.

47. /\* 5. Store the value in the plus-side gain calibration register PG. \*/

48. ADC0->PG = ADC\_PG\_PG(calibration\_var);

49.

50. /\* 6. Repeat the procedure for the minus-side gain calibration value. \*/

51. calibration\_var = 0x0000;

52.

53. calibration\_var += ADC0->CLM0;

54. calibration\_var += ADC0->CLM1;

55. calibration\_var += ADC0->CLM2;

56. calibration\_var += ADC0->CLM3;

57. calibration\_var += ADC0->CLM4;

58. calibration\_var += ADC0->CLMS;

59.

60. calibration\_var /= 2;

61.

62. calibration\_var |= 0x8000;

63.

64. ADC0->MG = ADC\_MG\_MG(calibration\_var);

65.

66. /\* Incheierea calibrarii \*/

67. ADC0->SC3 &= ~ADC\_SC3\_CAL\_MASK;

68.

69. return (0);

70. }

71.

La fiecare întrerupere, programul citește valoarea de la senzorul de sunet. În funcție de valoarea citita va aprinde 1,2 sau 3 leduri externe. Intervalele au fost setate empiric. Pentru valori mici ( <0.2 V) se va aprinde doar ledul verde. Pentru valori medii (0.2<x<0.4) se vor aprinde ledurile verde si galben. Pentru valori mari (>0.4) se vor aprinde toate cele 3 leduri. Dupa aprinderea ledurilor se transmite prin comunicatia seriala valorea citita si se dezactiveaza intreruperile pentru modulul ADC. Acestea vor fi reactivate de catre PIT0 in intreruperea sa, precum a fost explicat in sectiunea alocata modului.

1. void ADC0\_IRQHandler(void){

2.

3. float measured\_voltage;

4. uint8\_t parte\_zecimala, parte\_fractionara1, parte\_fractionara2;

5.

6. analog\_input = (uint16\_t) ADC0->R[0];

7.

8. /\* conversia valorii primite in voltaj din intervalul 0-VREF \*/

9. measured\_voltage = 3.3f - (analog\_input \* 3.3f) / 65535;

10.

11. parte\_zecimala = (uint8\_t) measured\_voltage;

12. parte\_fractionara1 = ((uint8\_t)(measured\_voltage \* 10)) % 10;

13. parte\_fractionara2 = ((uint8\_t)(measured\_voltage \* 100)) % 10;

14.

15. if(parte\_fractionara1 < 2){

16. GPIOA\_PSOR |= (1<<GREEN\_LED\_PIN);

17. GPIOA\_PCOR |= (1<<YELLOW\_LED\_PIN);

18. GPIOA\_PCOR |= (1<<RED\_LED\_PIN);

19. }

20.

21. if(parte\_fractionara1 >= 2 && parte\_fractionara1 < 4){

22. GPIOA\_PSOR |= (1<<GREEN\_LED\_PIN);

23. GPIOA\_PSOR |= (1<<YELLOW\_LED\_PIN);

24. GPIOA\_PCOR |= (1<<RED\_LED\_PIN);

25. }

26.

27. if(parte\_fractionara1 >= 4){

28. GPIOA\_PSOR |= (1<<GREEN\_LED\_PIN);

29. GPIOA\_PSOR |= (1<<YELLOW\_LED\_PIN);

30. GPIOA\_PSOR |= (1<<RED\_LED\_PIN);

31. }

32.

33.

34. UART0\_Transmit(parte\_zecimala + 0x30);

35. UART0\_Transmit('.');

36. UART0\_Transmit(parte\_fractionara1 + 0x30);

37. UART0\_Transmit(parte\_fractionara2 + 0x30);

38. /\*UART0\_Transmit(0x0A);

39. UART0\_Transmit(0x0D);\*/

40.

41.

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

43.

44. }

45.

Preluăm datele din senzor din registrul ADC0\_R.



Salvăm datele citite într-o variabila ajutătoare, dupa care facem conversia la un voltaj in intervalul [0,VREF], unde VREF este tensiunea la care este conectat senzorul de sunet. Apoi transformam voltajul cu ajutorul altor variabile ajutatoare (lucru necesar pentru transmiterea ulterioara prin UART a cate 1 caracter) intr-un numar cu 2 zecimale unde fiecare cifra este stocata separat.

1. analog\_input = (uint16\_t) ADC0->R[0];

2. measured\_voltage = 3.3f - (analog\_input \* 3.3f) / 65535;

3. parte\_zecimala = (uint8\_t) measured\_voltage;

4. parte\_fractionara1 = ((uint8\_t)(measured\_voltage \* 10)) % 10;

5. parte\_fractionara2 = ((uint8\_t)(measured\_voltage \* 100)) % 10;

Urmatoarele linii de cod au rolul de a transmite către PC prin UART a valorii date de senzor (variabila este un număr între 0 si 3.3).

1. UART0\_Transmit(parte\_zecimala + 0x30);

2. UART0\_Transmit('.');

3. UART0\_Transmit(parte\_fractionara1 + 0x30);

4. UART0\_Transmit(parte\_fractionara2 + 0x30);

In functie de valoarea citita aprindem 1,2 sau 3 leduri asa cum a fost explicat anterior. In functie de cate leduri trebuie aprinse, celelalte sunt stinse.

1. if(parte\_fractionara1 < 2){

2. GPIOA\_PSOR |= (1<<GREEN\_LED\_PIN);

3. GPIOA\_PCOR |= (1<<YELLOW\_LED\_PIN);

4. GPIOA\_PCOR |= (1<<RED\_LED\_PIN);

5. }

6.

7. if(parte\_fractionara1 >= 2 && parte\_fractionara1 < 4){

8. GPIOA\_PSOR |= (1<<GREEN\_LED\_PIN);

9. GPIOA\_PSOR |= (1<<YELLOW\_LED\_PIN);

10. GPIOA\_PCOR |= (1<<RED\_LED\_PIN);

11. }

12.

13. if(parte\_fractionara1 >= 4){

14. GPIOA\_PSOR |= (1<<GREEN\_LED\_PIN);

15. GPIOA\_PSOR |= (1<<YELLOW\_LED\_PIN);

16. GPIOA\_PSOR |= (1<<RED\_LED\_PIN);

17. }

# **Descrierea programului de afisare grafica a valoriilor**

Fișierul *main.py*:

from main\_window import MainWindow

from PySide6.QtWidgets import QApplication

from PySide6.QtGui import QScreen

app = QApplication([])

window = MainWindow()

window.setGeometry(0, 0, 800, 600)

screen\_geometry = QScreen.availableGeometry(QApplication.primaryScreen())

screenX = (screen\_geometry.width() - window.width()) / 2

screenY = (screen\_geometry.height() - window.height()) / 2

window.move(screenY, screenY)

window.show()

app.exec()

Importarea librăriilor și modulelor necesare rulării programului în *main\_window.py*:

from PySide6.QtWidgets import QMainWindow, QHBoxLayout, QVBoxLayout, QWidget, QGroupBox, QLabel, QPushButton, QLineEdit, QTextEdit

from PySide6.QtGui import QIcon, QPalette, QColor, QFont

from PySide6.QtCore import Qt,QTimer

import pyqtgraph as pg

import serial

import numpy as np

Inițializarea conexiunii seriale:

self.ser = serial.Serial()

self.ser.baudrate = 38400

Portul este configurat ulterior prin funcția send\_input:

def send\_input(self):

        com\_port = self.line\_edit.text().strip()

        self.ser.port = 'COM' + com\_port

Pentru schimbarea sensului secvenței de led-uri vom avea 2 butoane:

        button1 = QPushButton("A-M-G-N")

        button2 = QPushButton("N-G-M-A")

        button1.clicked.connect(lambda: self.update\_sens(0))

        button2.clicked.connect(lambda: self.update\_sens(1))

Acestea apelează funcția care va trimite un caracter pe serial:

    def update\_sens(self, new\_sens):

        if self.ser.is\_open:

            self.sens = new\_sens

            self.text\_edit.insertPlainText(f"Sense updated to: {new\_sens}\n")

            self.ser.write(bytes([new\_sens]))

Pentru reprezentarea valorilor senzorilor avem 2 vectori de dimensiune 10.Valorile de pe axa Y vor fi reprezentate sub forma unui bar-graph, iar culoarea acestuia va fi dată dinamic in funcție de valoare.

        self.plot\_widget = pg.PlotWidget()

        self.time = [0,0.25,0.5,0.75,1,1.25,1.5,1.75,2,2.25]

        self.voltage=[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

        self.plot\_widget.getPlotItem().setRange(yRange=[0, 1])

        self.plot\_widget.setBackground("w")

        self.plot\_widget.setLabel('left','Voltage')

        self.plot\_widget.setLabel('bottom','Time')

        self.color\_gradient = pg.ColorMap(pos=np.array([0, self.low\_medium, self.medium\_high]), color=[(0, 255, 0), (255, 255, 0), (255, 0, 0)])

        self.colors = self.color\_gradient.map(self.voltage)

        self.bar\_item = pg.BarGraphItem(x=self.time, height=self.voltage, width=0.2, brushes=[QColor(\*c) for c in self.colors])

        self.plot\_widget.addItem(self.bar\_item)

Acestea sunt pragurile:

self.low\_medium = 0.2

self.medium\_high = 0.4

Folosim un timer care va apela metoda update\_value la fiecare 100ms:

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_value)

self.timer.start(100)

Funcția *update\_value*:

def update\_value(self):

        if self.ser.is\_open:

            received\_data = self.ser.read(4)

            self.ser.reset\_input\_buffer()

            decoded\_data = received\_data.decode('utf-8')

            sensor\_value=float(decoded\_data)

            self.text\_edit.insertPlainText(f"Received value : {sensor\_value}\n")

            self.time=self.time[1:]

            self.time.append(self.time[-1]+0.25)

            self.voltage=self.voltage[1:]

            self.voltage.append(sensor\_value)

            self.colors = self.color\_gradient.map(self.voltage)

            if hasattr(self, 'bar\_item'):

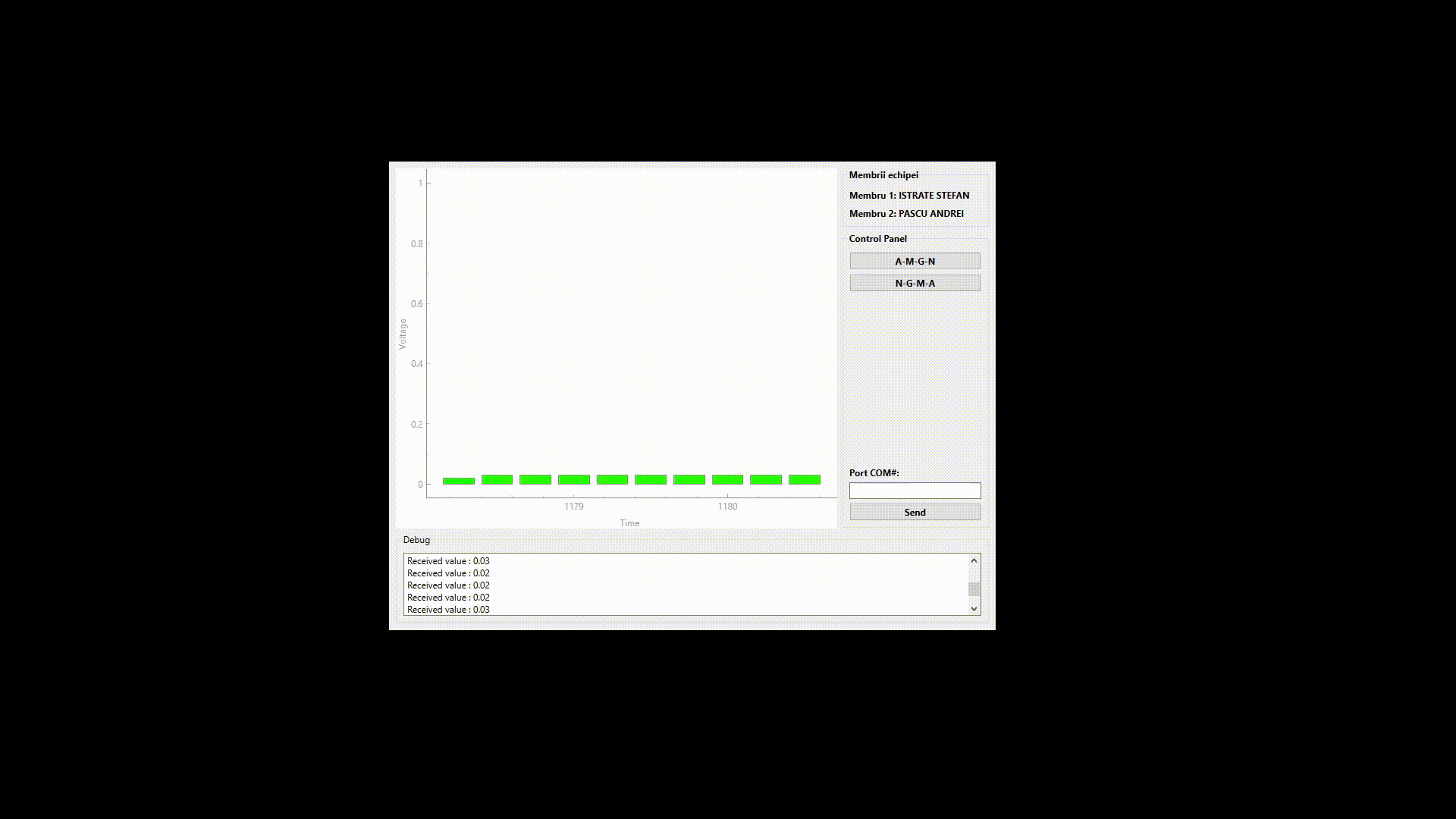
                self.plot\_widget.removeItem(self.bar\_item)

            self.bar\_item = pg.BarGraphItem(x=self.time, height=self.voltage, width=0.2, brushes=[QColor(\*c) for c in self.colors])

            self.plot\_widget.addItem(self.bar\_item)

Se citește de pe serial valoare transmisă de senzor și se convertește în float.Se translatează atât vectorul time, cât și vectorul voltage cu 1 poziție și se adaugă noua valoare.Se reconstruiește graficul.

## **6.1 Rezultate rulare program**



# **Probleme întâmpinate și sugestii**

De-a lungul dezvoltarii aplicatiei am intampinat o serie de probleme, fie ele tehnice sau din neatentie. Pentru a scuti din timpul irosit de noi rezolvandu-le, vom expune atat problemele cat si solutiile acestora intr-o maniera in care sa evidentiem pe ce aspecte trebuie pus accent inca de la inceput, pentru a nu fi nevoie sa se revina la o parte de cod deja scrisa de exemplu.

* Configurare GPIO

**Sugestie:** Atentie la pinii pe care doriti sa ii folositi pentru GPIO sa nu fie folositi de alt modul, de exemplu UART.

– initial am ales 3 pini oarecare pentru cele 3 leduri externe si am uitat pe ce pini este configurat UART-ul in functie de alternativa, astfel ca 1 led parea ca este ars, dar de fapt era conectat necorspunzator

* Configurare ADC

**Sugestie:** Atentie la rezolutia configurata pentru ADC si tipul de date al variabilelor in care se citeste valoarea, cat si al variabilelor ajutatoare folosite in calcule

* initial am folosit uint8\_t, uint16\_t, int, double, float, dar apoi am standardizat la uint8\_t si float. Faptul ca foloseam mai multe tipuri de date a dus la o inconsistenta a datelor in momentul in care am incercat conversia valorii citite de la senzorul de sunet in decibeli, lucru la care am si renuntat dupa ce am realizat ca este un overhead inutil, nefiind necesara valorea transformata pentru incadrarea intr-un anumit interval.
* In plus, intervalele pentru valori mici, medii si mari au fost determinate prin experimentare cu sunete, senzorul nefiind capabil sa returneze valoarea maxima teoretica
* Configurare PIT

**Sugestie:** Atentie la sincronizarea intre conversiile ADC, transmiterea prin UART si receptionarea prin UART pe PC

* Initial intervalul la care PIT-ul activa intreruperile pentru ADC nu era acelasi cu intervalul la care interfata receptiona date. Acest lucru era important pentru ca transmiterea datelor se face in intreruperea ADC.
* Dupa sincronizarea timing-ului am intampinat probleme de delay in citire, de parca se faceau mai multe conversii decat ce afisam. Acest lucru ne-a dus cu gandul la nevoia unui trigger hardware pentru ADC care sa aiba ca declansator PIT-ul ( pe canalul destinat activarii intreruperilor ADC ), insa dupa activarea acestuia problema a ramas aceeasi. Solutia a fost golirea fortata a bufferului de receptie din Python dupa fiecare citire.
* Dupa ultimul retus, o noua problema a aparut, ocazional la intervale random, caracterele pareau ca nu sunt citite in ordinea corecta, asa ca am adaugat o conditie de sanitizare in codul din interfata.

# **Github**

În următorul repository se află toate sursele cod ale proiectului realizat: https://github.com/AndreiPascu180/MicrocontrollerProject