**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**FACULTATEA DE SISTEME INFORMATICE ȘI SECURITATE CIBERNETICĂ**

**Specializarea: Calculatoare şi sisteme informatice pentru apărare şi securitate naţională**



**Proiect Sisteme Specializate cu Microprocesoare**

Sd. Sg. Maj. DUMITRU Andreea-Ioana

Sd. Sg. Maj. MOCANU Răzvan

**București**

**2023**

**CUPRINS**

[**1.** **Scopul proiectului** 2](#_Toc154681091)

[**2.** **Configurare senzori** 3](#_Toc154681092)

[*2.1.* *Senzorul analog de sunet DFR0034* 3](#_Toc154681093)

[*2.2.* *Servomotorul SG90* 4](#_Toc154681094)

[**3.** **Reprezentarea grafică a modului de conectare a senzorilor și a componentelor electronice** 5](#_Toc154681095)

[**4.** **Descrierea programului** 5](#_Toc154681096)

[*4.1.* *Inițializarea modulelor* 5](#_Toc154681097)

[4.1.1. Modulul UART 5](#_Toc154681098)

[4.1.2. Inițializarea modului ADC și prelucrarea datelor înregistrate de către sensor 10](#_Toc154681099)

[4.1.3. Inițializarea modului TPM 10](#_Toc154681100)

[4.1.4. Inițializarea modului GPIO 10](#_Toc154681101)

[4.1.5. Inițializarea modului PIT 13](#_Toc154681102)

[*4.2.* *Generare PWN* 16](#_Toc154681103)

[*4.3.* *Transmitere date prin UART* 16](#_Toc154681104)

[*4.4.* *Funcția main* 16](#_Toc154681105)

[**5.** **Probleme întâmpinate** 16](#_Toc154681106)

[**6.** **Rezultate aplicație Python** 17](#_Toc154681107)

[**7.** **Referințe** 17](#_Toc154681108)

# **Scopul proiectului**

Scopul proiectului este acela de a modifica unghiul de rotație al servomotorului utilizând un senzor de sunet printr-un semnal PWN generat și manipulat în funcție de plaja de valori a senzorului de sunet.

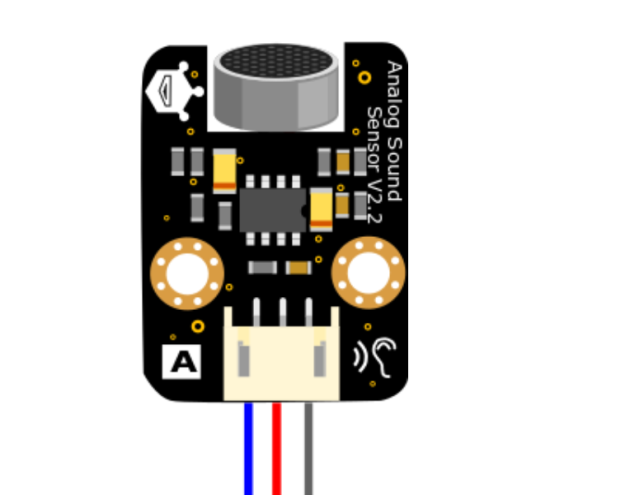
Valoarea digitală care va rezulta în urma conversiei din tensiune a datelor achiziționate de la senzorul de sunet se vor regăsi într-un interval care va fi apoi împărțit in trei subintervale care vor corespunde cu valori mici, medii și mari ale datelor achiziționate.

În funcție de subintervalul în care se regăsesc datele convertite din analogic în digital, servomotorul își va schimba poziția elicei (0°, 90° și 180°).

# **Configurare senzori**

## *Senzorul analog de sunet DFR0034*

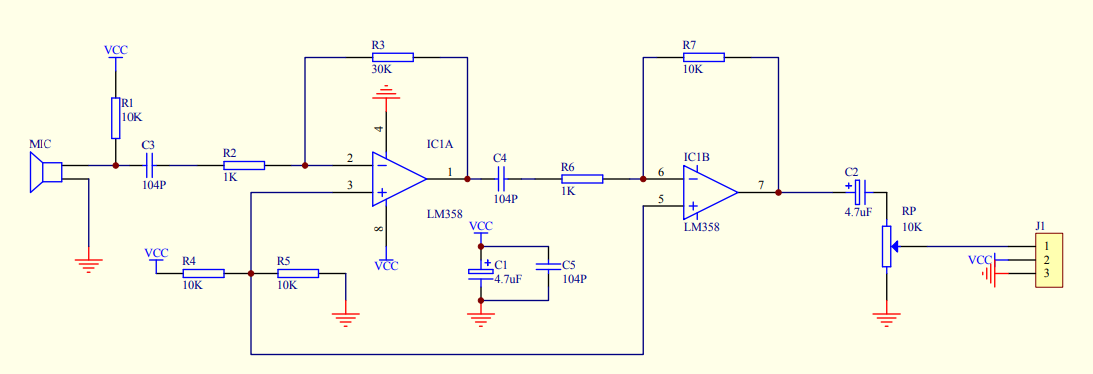
Senzorul DFR0034 este un senzor analog ce detectează zgomotul din mediul în care este pus.



Blue – Analog signal output

Red - VCC

Black – GND



Specificații:

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

Senzorul de sunet a fost conectat pe plăcuță astfel:

* **Firul roșu (VCC)** se conectează la pinul de 3V (P3V3);
* **Firul negru (GND)** se conectează la pinul GND;
* **Firul albastru (output sensor value)** se conectează la pinul PTC2.

## *Servomotorul SG90*

Servomotorul SG90 se poate roti până la aproximativ 180 grade(90 în fiecare direcție).



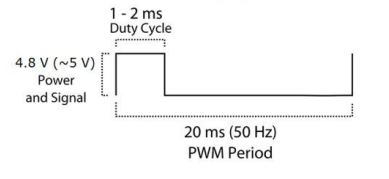
Orange – PWN

Red – VCC

Brown – GND

Specificații:

* Tensiune de alimentare: 4.8V până la 5V;
* Rotire elice;
* Mărime: 23x32 mm.



Servomotorul a fost conectat pe plăcuță astfel:

* **Firul maroniu (GND)** se conectează la pinul GND;
* **Firul roșu (VCC)** se conectează la pinul de 5V (P5V\_USB) ;
* **Firul portocaliu (PWN)** se conectează la PTA1.

# **Reprezentarea grafică a modului de conectare a senzorilor și a componentelor electronice**

# **Descrierea programului**

## *Inițializarea modulelor*

### Modulul UART

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

Prin funcția *UART0\_init* configurăm următoarele:

1. void UART0\_Init(uint32\_t baud\_rate)

2. {

3.

4. //Setarea sursei de ceas pentru modulul UART

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

6.

7. //Activarea semnalului de ceas pentru modulul UART

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

9.

10. //Activarea semnalului de ceas pentru portul A

11. //intrucat dorim sa folosim pinii PTA1, respectiv PTA2 pentru comunicarea UART

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

13.

14. //Fiecare pin pune la dispozitie mai multe functionalitati

15. //la care avem acces prin intermediul multiplexarii

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

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

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

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

20.

21.

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

23.

24. //Configurare Baud Rate

25. uint32\_t osr = 4; // Over-Sampling Rate (numarul de esantioane luate per bit-time)

26.

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

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

29. // 0x1F00 - 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0

30. // 0 0 0 b13 b12 b11 b10 b09 0 0 0 0 0 0 0 0 >> 8

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

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

33.

34. uint32\_t sbr = 48000000UL / (osr \* baud\_rate / 4);

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

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

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

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

39.

40.

41. //Setare numarul de biti de date la 8 si fara bit de paritate

42. UART0->C1 = 0;

43.

44. //Dezactivare intreruperi la transmisie

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

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

47.

48. //Activare intreruperi la receptie

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

50.

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

52.

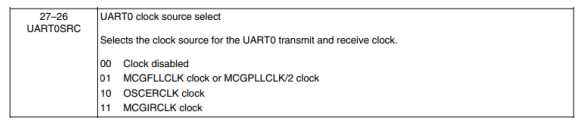
53. NVIC\_EnableIRQ(UART0\_IRQn);

54.

55. }

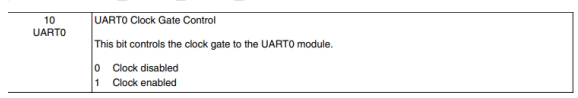
56.

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



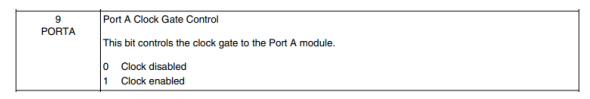
În registrul SIM\_SCGC4 (System Clock Gating Control Register 4) setăm pe 1 câmpul UART0 (bitul 10) pentru activarea ceasului pentru acest modul, folosind masca 14 SIM\_SCGC4\_UART0\_MASK. Masca are valoarea 0x400, adica 1024 în zecimal, adică are setat pe 1 al 10-lea bit (UART0).

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

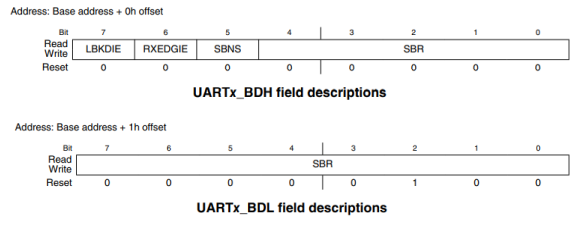


În registrul SIM\_SCGC5 (System Clock Gating Control Register 5) setăm pe 1 câmpul PORTA (bitul 9) pentru activarea ceasului acestui port, folosind masca SIM\_SCGC5\_PORTA\_MASK. Masca are valoarea 0x200, adica 512 în zecimal, adică are setat pe 1 al 9-lea bit (PORTA).

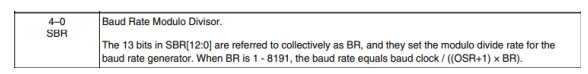
*SIM->SCGC5 |= SIM\_SCGC5\_PORTA\_MASK;*



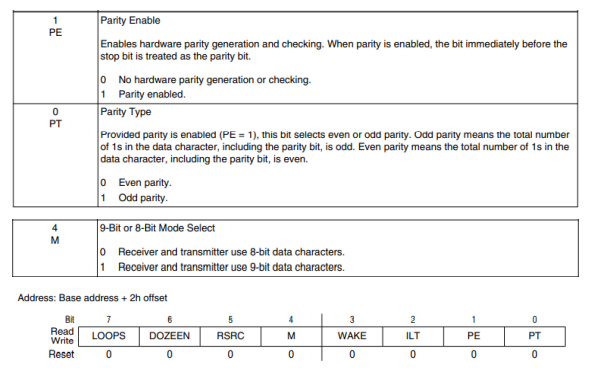
În regiștrii de control ai pinilor 1 și 2 din portul A (PORTA\_PCR1/PORTA\_PCR2), setăm câmpul MUX (biții 10-8) pe valoarea 0b010, care înseamnă folosirea acestora în modulul de UART0 (RX/TX).



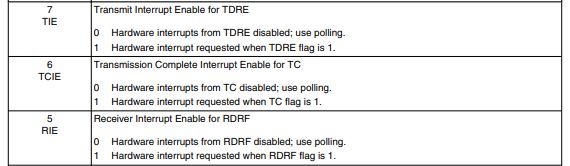
Având în vedere că baud rate-ul nostru este 115200 și over sampling rate-ul(osr) este 4, formula pentru calcularea câmpului SBR am modificat-o astfel încât să respectăm valorile SBR-ului:

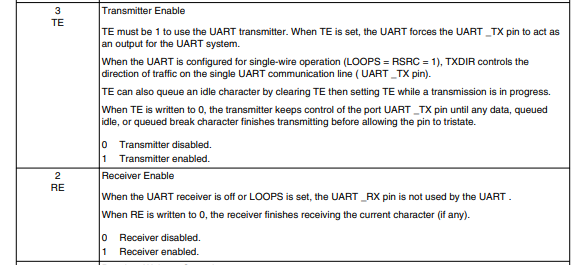
*uint32\_t sbr = 48000000UL / (osr \* baud\_rate / 4);*

Portul serial este setat implicit cu 8 biți date, niciun bit de paritate și un bit de stop.



Activăm transmiterea prin UART prin setarea câmpului TE (bitul 3, Transmitter Enable) din UART0\_C2 pe valoarea 1 cu ajutorul măștii UART\_C2\_TE\_MASK ce are valoarea 0x8u, și activăm, de asemenea, recepția prin UART prin setarea câmpului RE (bitul 2, Receiver Enable) din UART0\_C2 pe valoarea 1 cu ajutorul măștii UART\_C2\_RE\_MASK ce are valoarea 0x4u.





Funcția *UART0\_Trasmit* are rolul de a transmite serial caracterul transmis ca parametru.

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

2. {

3. //Punem in asteptare pana cand registrul de transmisie a datelor nu este gol

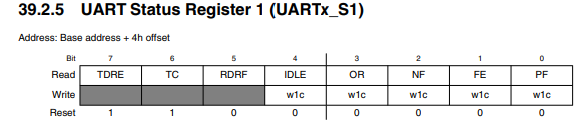
4. while(!(UART0->S1 & UART0\_S1\_TDRE\_MASK));

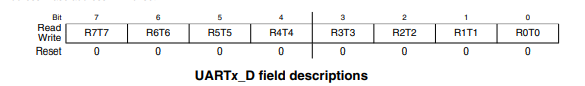
5. UART0->D = data;

6.

7. }

8.





Funcția *UART0\_receive* are rolul de a primi serial date.

1. uint8\_t UART0\_receive(void)

2. {

3. //Punem in asteptare pana cand registrul de receptie nu este plin

4. while(!(UART0->S1 & UART0\_S1\_RDRF\_MASK));

5. return UART0->D;

6.

7. }

8.

Functia *UART0\_IRQHandler* are rolul de a transmite datele cu majuscule.

1. void UART0\_IRQHandler(void) {

2.

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

4. c = UART0->D;

5. }

6.

7. if(c >= 'a' && c <= 'z') {

8. UART0\_Transmit(c - 'a' + 'A');

9. }

10. else if ( c >= 'A' && c <= 'Z') {

11. UART0\_Transmit(c - 'A' + 'a');

12. }

13. else if(c == 0X0D) {

14. UART0\_Transmit(0X0D);

15. UART0\_Transmit(0X0A);

16. }

17. }

18.

### Inițializarea modului ADC și prelucrarea datelor înregistrate de către sensor

### Inițializarea modului TPM

### Inițializarea modului GPIO

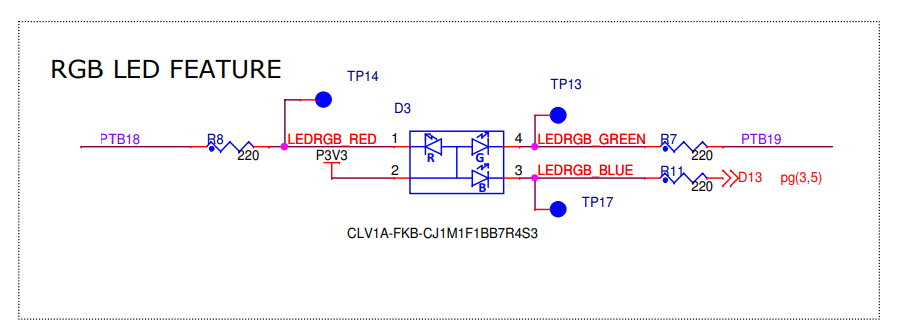
Pentru aplicația noastră avem nevoie de 3 LED-uri: roșu (Red), verde (Green), albastru (Blue) și negru (Black) care semnifică faptul că toate becurile sunt stinse.

1. #define RED\_LED\_PIN (18) // PORT B

2. #define GREEN\_LED\_PIN (19) // PORT B

3. #define BLUE\_LED\_PIN (1) // PORT D

4.



Funcția OutputPIN\_Init are rolul de a inițializa modulul GPIO pentru a controla aprinderea/stingerea celor 3 LED-uri.

1. void OutputPIN\_Init(void){

2.

3. SIM\_SCGC5 |= SIM\_SCGC5\_PORTB\_MASK | SIM\_SCGC5\_PORTD\_MASK;

4.

5. // --- RED LED ---

6.

7. // Utilizare GPIO ca varianta de multiplexare

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

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

10.

11. // Configurare pin pe post de output

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

13.

14. // Stingerea LED-ului (punerea pe 0 logic)

15. GPIOB\_PSOR |= (1<<RED\_LED\_PIN);

16.

17. // --- GREEN LED ---

18.

19. // Utilizare GPIO ca varianta de multiplexare

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

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

22.

23. // Configurare pin pe post de output

24. GPIOB\_PDDR |= (1<<GREEN\_LED\_PIN);

25.

26. // Stingerea LED-ului (punerea pe 0 logic)

27. GPIOB\_PSOR |= (1<<GREEN\_LED\_PIN); //scriu 1 stinge ledul

28.

29. // --- BLUE LED ---

30.

31. // Utilizare GPIO ca varianta de multiplexare

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

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

34.

35. // Configurare pin pe post de output

36. GPIOD\_PDDR |= (1<<BLUE\_LED\_PIN);

37.

38. // Stingerea LED-ului (punerea pe 0 logic)

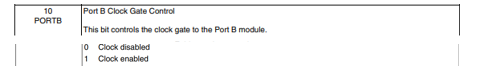
39. GPIOD\_PSOR |= (1<<BLUE\_LED\_PIN); //set e stins

40.

41. }

Se activează semnalul de ceas pentru pinii folosiți în cadrul fiecărui LED:

*SIM\_SCGC5 |= SIM\_SCGC5\_PORTB\_MASK | SIM\_SCGC5\_PORTD\_MASK;*

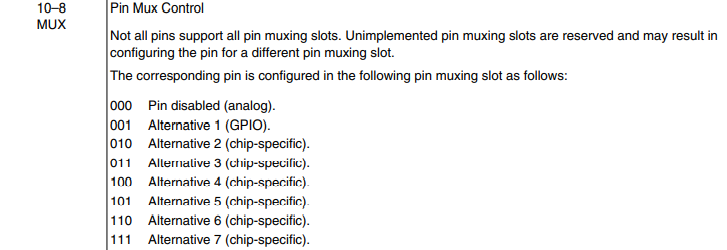


Utilizăm GPIO ca variantă de multiplexare pentru pinii corespunzători:

19. // Utilizare GPIO ca varianta de multiplexare

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

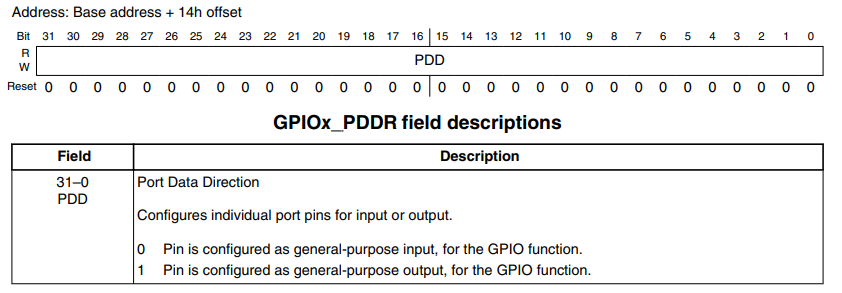
21. PORTB->PCR[GREEN\_LED\_PIN] |= PORT\_PCR\_MUX(1);



Configurăm pinii pe post de output:

35. // Configurare pin pe post de output

36. GPIOD\_PDDR |= (1<<BLUE\_LED\_PIN);



Stingerea ledului constă în punerea pe 0 logic a pinilor:

// Stingerea LED-ului (punerea pe 0 logic)

15. GPIOB\_PSOR |= (1<<RED\_LED\_PIN);

Funcția *Control\_RGB\_LEDs* are rolul de a aprinde/stinge un LED. LED-urile sunt configurate să se stingă/aprindă după un interval de 1190 ms, valoare care a fost inițializată folosind canalul 1 din PIT.

1. void Control\_RGB\_LEDs(uint8\_t red, uint8\_t green, uint8\_t blue) {

2.

3. // Setare sau stinge LED-ul rosu

4. if (red) {

5. GPIOB\_PCOR |= (1 << RED\_LED\_PIN); // Seteaza pinul

6. }

7. else {

8. GPIOB\_PSOR |= (1 << RED\_LED\_PIN); // Stinge pinul

9. }

10.

11. // Setare sau stinge LED-ul verde

12. if (green) {

13. GPIOB\_PCOR |= (1 << GREEN\_LED\_PIN); // Seteaza pinul

14. }

15. else {

16. GPIOB\_PSOR |= (1 << GREEN\_LED\_PIN); // Stinge pinul

17. }

18.

19. // Setare sau stinge LED-ul albastru

20. if (blue) {

21. GPIOD\_PCOR |= (1 << BLUE\_LED\_PIN); // Seteaza pinul

22. }

23. else {

24. GPIOD\_PSOR |= (1 << BLUE\_LED\_PIN); // Stinge pinul

25. }

26.

27. }

28.

Controlul LED-ului **roșu**:

* Dacă valoarea variabilei **red** nu este zero (adevărat), se șterge (se setează la 0) bitul corespunzător în registrul GPIOB\_PCOR. Acest lucru, în mod obișnuit, activează LED-ul roșu.
* Dacă valoarea variabilei **red** este zero (fals), se setează (se setează la 1) bitul corespunzător în registrul GPIOB\_PSOR. Acest lucru, în mod obișnuit, dezactivează LED-ul roșu.

Controlul LED-ului **verde**:

* Logica similară cu cea pentru LED-ul roșu, dar aplicată LED-ului verde pe GPIOB.

Controlul LED-ului **albastru**:

* Dacă valoarea variabilei **blue** nu este zero (adevărat), se șterge (se setează la 0) bitul corespunzător în registrul GPIOD\_PCOR. Acest lucru, în mod obișnuit, activează LED-ul albastru.
* Dacă valoarea variabilei **blue** este zero (fals), se setează (se setează la 1) bitul corespunzător în registrul GPIOD\_PSOR. Acest lucru, în mod obișnuit, dezactivează LED-ul albastru.

### Inițializarea modului PIT

Funcția *PIT\_Init* inițializeaza perifericul PIT, mai exact cele 2 canale ale sale:

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 1,119 secunde pt led-uri

10. PIT->CHANNEL[1].LDVAL = 56879999;

11. //jumatate de secuda pentru adc

12. PIT->CHANNEL[0].LDVAL = 23999999;

13.

14. // Activarea întreruperilor pe canalul 0 si 1

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

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

17.

18. // Activarea timerului de pe canalul 0 si 1

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

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

21.

22. // Activarea întreruperii mascabile si setarea prioritatiis

23. NVIC\_ClearPendingIRQ(PIT\_IRQn);

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

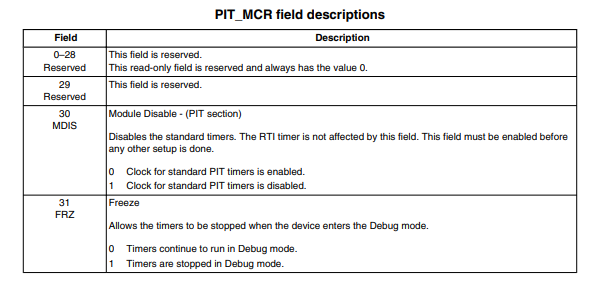
25. NVIC\_EnableIRQ(PIT\_IRQn);

26. }

27.

Se activează semnalul de ceas folosind masca SIM\_SCGC6\_PIT\_MASK:

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

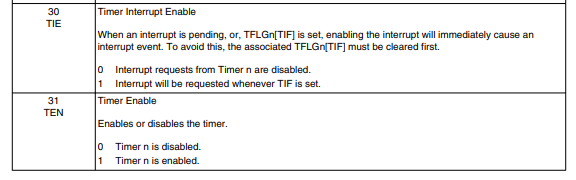
S-a setat semnalul de ceas pentru tabloul de timere(*PIT\_MCR &= ~PIT\_MCR\_MDIS\_MASK;*) și s-a oprit decrementarea valorii numărătoarelor în modul debug(*PIT->MCR |= PIT\_MCR\_FRZ\_MASK;*):

S-au setat valorile celor 2 canale ce semnifică perioada de timp pentru întreruperile timer-ului, astfel:

* Canalul 1 a fost folosit pentru controlul LED-urilor și setat la *56879999* care înseamnă 1119 ms;
* Canalul 2 a fost folosit pentru modulul ADC și este setat la *23999999* care înseamnă 0,5 s.

Cele 2 timere generează întreruperi la perioadele de timp setate in registrul LDVAL. Ele își încarcă în acest registru valoarea, scade până la 0 și apoi își încarcă valoarea de start înapoi. Mereu când un timer ajunge la 0, acesta va genera un impuls de întrerupere și va seta flag-ul de întrerupere.

Apoi, am activat întreruperile(TCTRL\_TIE) și timer-ul(TCTRL\_TEN) pe ambele canale folosindu-ne de registrul TCTRL (Timer Control Register) care conține bitul de control pentru fiecare timer în parte.



Funcția *PIT\_IRQHandler* este apelată atunci când se declanșează întreruperea de către modulul PIT:

1. uint8\_t ledState = 0;

2. void PIT\_IRQHandler(void) {

3.

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

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

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

7. }

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

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

10.

11. switch (ledState) {

12. case 0: // Red

13. Control\_RGB\_LEDs(1, 0, 0);

14. break;

15. case 1: // Green

16. Control\_RGB\_LEDs(0, 1, 0);

17. break;

18. case 2: // Blue

19. Control\_RGB\_LEDs(0, 0, 1);

20. break;

21. default: // Turn off LEDs -- Black

22. Control\_RGB\_LEDs(0, 0, 0);

23. break;

24. }

25.

26. // Increment LED state

27. ledState++;

28.

29. // Check for overflow

30. if (ledState > 3) {

31. ledState = 0;

32. }

33. }

34. }

35.

Se verifică dacă întreruperea a fost declanșată pentru canalul 0 al PIT, apoi se setează un bit pentru controlul ADC și se șterge flag-ul de întrerupere pentru canalul 0 al PIT-ului:

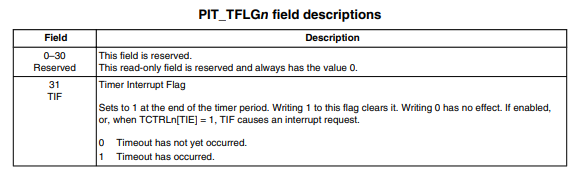
if(PIT->CHANNEL[0].TFLG & PIT\_TFLG\_TIF\_MASK) {

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

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

7. }

După se verifică dacă întreruperea a fost declanșată pentru canalul 1 al PIT, apoi se șterge flag-ul de întrerupere pentru canalul 1 al PIT-ului și începe schimbarea culorii LED-urilor în funcție de starea curentă a variabilei ledState.



## *Generare PWN*

## *Transmitere date prin UART*

## *Funcția main*

# **Probleme întâmpinate**

În cadrul elaborării acestui proiect, nu am întâmpinat dificultăți semnificative. Odată ce am înțeles modul de funcționare al senzorilor și cum să configurăm platforma de dezvoltare, întregul proiect a progresat rapid și fără probleme notabile. De asemenea, credem că atât resursele oferite în cadrul laboratorului, cât și documentația platformei utilizate ne-au fost de un real ajutor.

Considerăm că acest proiect contribuie semnificativ la asimilarea unei părți importante din conținutul predat în clasă.

# **Rezultate aplicație Python**

# **Referințe**

* Freescale Semiconductor, Inc., KL25 Sub-Family Reference Manual, 2012
* <https://datasheetspdf.com/pdf-file/791970/TowerPro/SG90/1>
* <https://wiki.dfrobot.com/Analog_Sound_Sensor_SKU__DFR0034>
* <https://learningmicro.wordpress.com/serial-communication-interface-using-uart/>