**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](#_Toc154780247)

[**2.** **Configurare senzori** 3](#_Toc154780248)

[*2.1.* *Senzorul analog de sunet DFR0034* 3](#_Toc154780249)

[*2.2.* *Servomotorul SG90* 4](#_Toc154780250)

[**3.** **Reprezentarea grafică a modului de conectare a senzorilor și a componentelor electronice** 5](#_Toc154780251)

[**4.** **Descrierea programului** 5](#_Toc154780252)

[*4.1.* *Inițializarea modulelor* 5](#_Toc154780253)

[4.1.1. Modulul UART 5](#_Toc154780254)

[4.1.2. Inițializarea modului GPIO 10](#_Toc154780255)

[4.1.3. Inițializarea modului PIT 13](#_Toc154780256)

[4.1.4. Inițializarea modului ADC și prelucrarea datelor înregistrate de către senzor 16](#_Toc154780257)

[4.1.5. Inițializarea modului TPM și generare PWM 22](#_Toc154780258)

[*4.2.* *Configurarea ceasului* 25](#_Toc154780259)

[*4.3.* *Funcția main* 28](#_Toc154780260)

[**5.** **Set-up** 30](#_Toc154780261)

[**6.** **Schemă bloc** 31](#_Toc154780262)

[**7.** **Diagrame de stări** 31](#_Toc154780263)

[**8.** **Rezultate aplicație Python** 32](#_Toc154780264)

[**9.** **Probleme întâmpinate** 33](#_Toc154780265)

[**10.** **Referințe** 33](#_Toc154780266)

# **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.

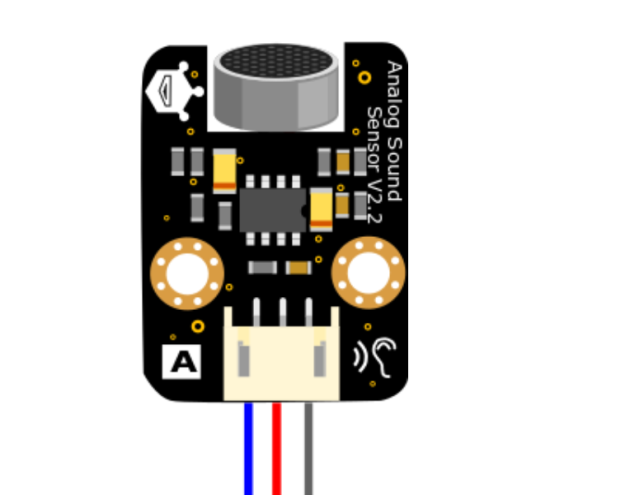
Valoarea digitală care va rezulta în urma conversiei din tensiune a datelor achiziționate de la senzorul de sunet se vor regăsi într-un interval care va fi apoi împărțit in trei subintervale care vor corespunde cu valori mici, medii și mari ale datelor achiziționate.

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*

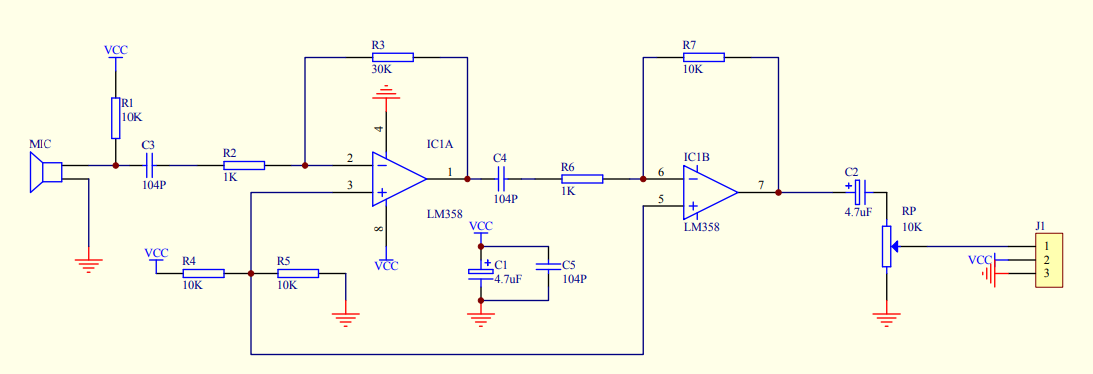
Senzorul DFR0034 este un senzor analog ce detectează zgomotul din mediul în care este pus.



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).



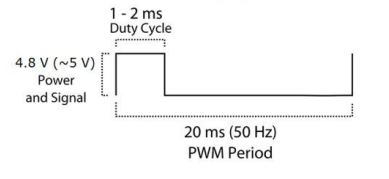
Orange – PWN

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**

TINKERCAD dar nu gasesc ce avem noi??????????/ TO DO

# **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));

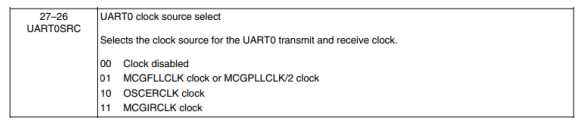
52.

53. NVIC\_EnableIRQ(UART0\_IRQn);

54.

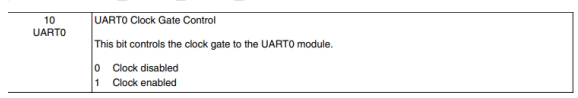
55. }

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.

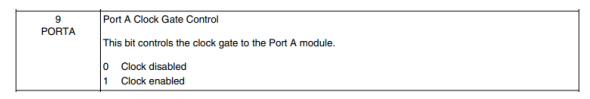
În registrul SIM\_SCGC4 (System Clock Gating Control Register 4) setăm pe 1 câmpul UART0 (bitul 10) pentru activarea ceasului pentru acest modul, folosind masca 14 SIM\_SCGC4\_UART0\_MASK. Masca are valoarea 0x400, adica 1024 în zecimal, adică are setat pe 1 al 10-lea bit (UART0).

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

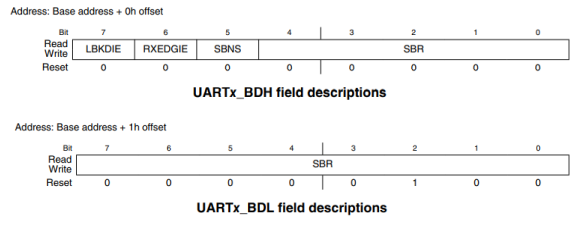


În registrul SIM\_SCGC5 (System Clock Gating Control Register 5) setăm pe 1 câmpul PORTA (bitul 9) pentru activarea ceasului acestui port, folosind masca SIM\_SCGC5\_PORTA\_MASK. Masca are valoarea 0x200, adica 512 în zecimal, adică are setat pe 1 al 9-lea bit (PORTA).

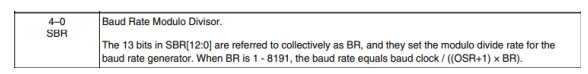
*SIM->SCGC5 |= SIM\_SCGC5\_PORTA\_MASK;*



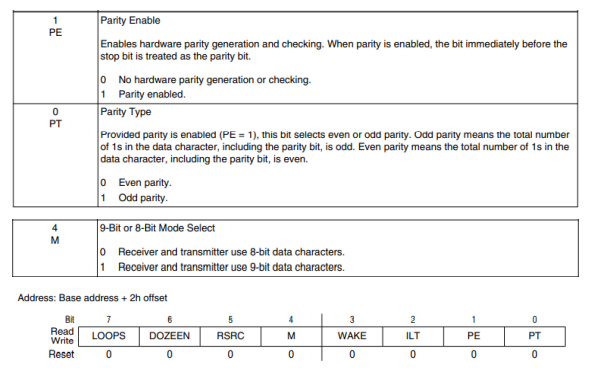
În regiștrii de control ai pinilor 1 și 2 din portul A (PORTA\_PCR1/PORTA\_PCR2), setăm câmpul MUX (biții 10-8) pe valoarea 0b010, care înseamnă folosirea acestora în modulul de UART0 (RX/TX).



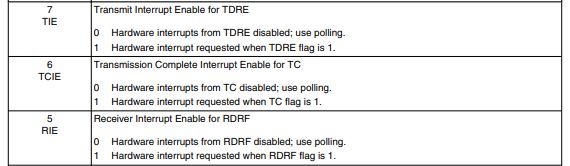
Având în vedere că baud rate-ul nostru este 115200 și over sampling rate-ul(osr) este 4, formula pentru calcularea câmpului SBR am modificat-o astfel încât să respectăm valorile SBR-ului:

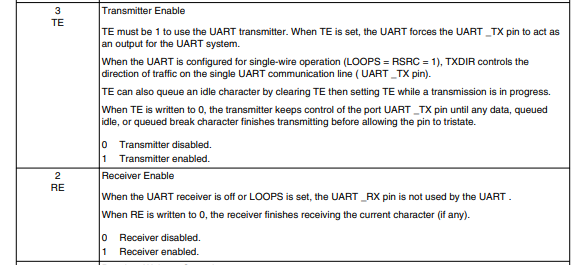
*uint32\_t sbr = 48000000UL / (osr \* baud\_rate / 4);*

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

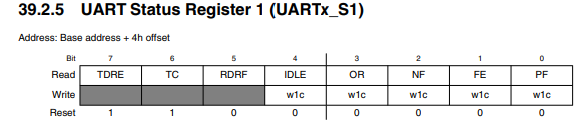
4. while(!(UART0->S1 & UART0\_S1\_TDRE\_MASK));

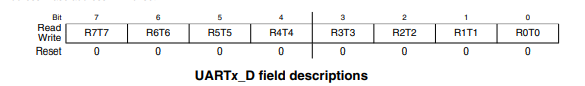
5. UART0->D = data;

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. }

### Inițializarea modului GPIO

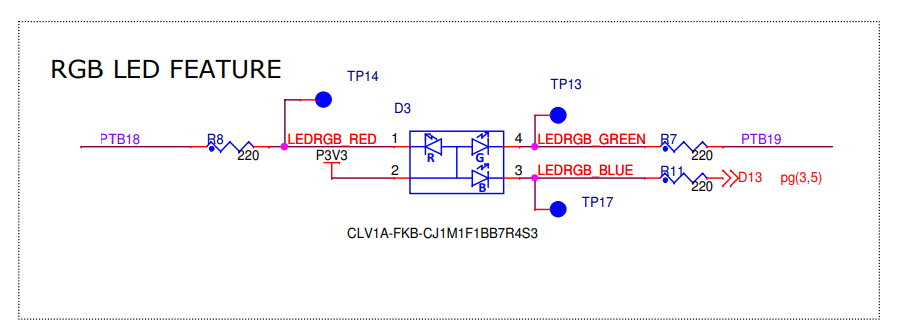
Pentru aplicația noastră avem nevoie de 3 LED-uri: roșu (Red), verde (Green), albastru (Blue) și negru (Black) care semnifică faptul că toate becurile sunt stinse.

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)

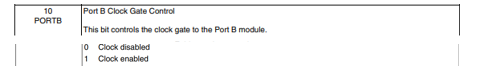
39. GPIOD\_PSOR |= (1<<BLUE\_LED\_PIN); //set e stins

40.

41. }

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

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

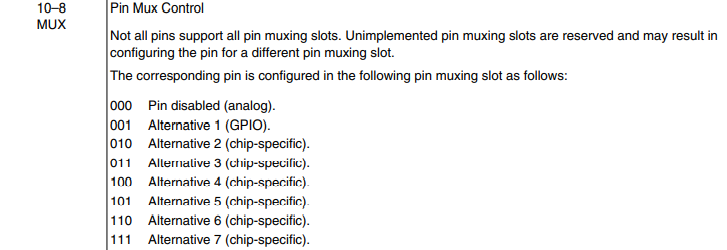


Utilizăm GPIO ca variantă de multiplexare pentru pinii corespunzători:

19. // Utilizare GPIO ca varianta de multiplexare

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

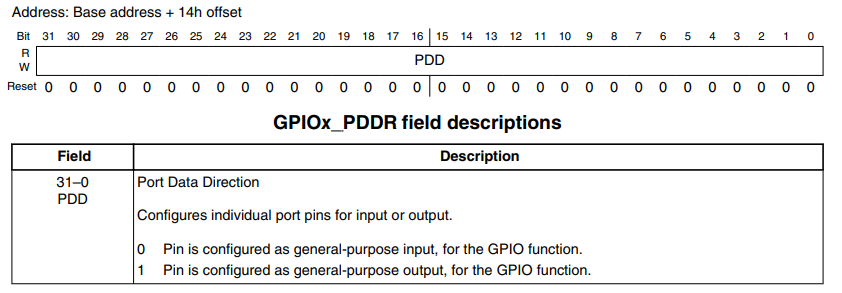
21. PORTB->PCR[GREEN\_LED\_PIN] |= PORT\_PCR\_MUX(1);



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. }

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);

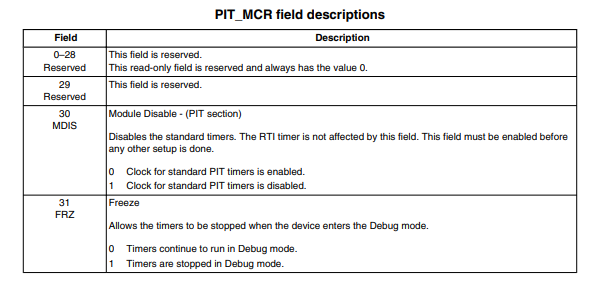
25. NVIC\_EnableIRQ(PIT\_IRQn);

26. }

27.

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

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

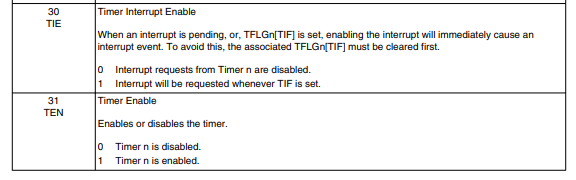
S-a setat semnalul de ceas pentru tabloul de timere(*PIT\_MCR &= ~PIT\_MCR\_MDIS\_MASK;*) și s-a oprit decrementarea valorii numărătoarelor în modul debug(*PIT->MCR |= PIT\_MCR\_FRZ\_MASK;*):

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:

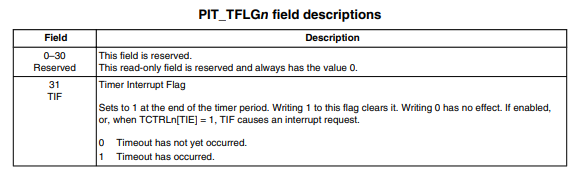
if(PIT->CHANNEL[0].TFLG & PIT\_TFLG\_TIF\_MASK) {

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.



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

Setarea canalului pentru senzorul de sunet și inițializarea unor variabile:

1. #define ADC\_CHANNEL (11) // PORT C PIN 2
2. volatile uint8\_t flag=0;
3. uint8\_t analog\_input=0;



Inițializarea modului 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 8 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. // Set ADC clock frequency fADCK less than or equal to 4 MHz (PG. 494)

15. ADC0->CFG1 |= ADC\_CFG1\_MODE(0) |

16. ADC\_CFG1\_ADICLK(0) |

17. ADC\_CFG1\_ADIV(2);

18.

19. // DIFF = 0 --> Conversii single-ended (PG. 464)

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

21. ADC0->SC3 = 0x00;

22.

23. // Selectarea modului de conversii continue,

24. // pentru a-l putea folosi in tandem cu mecanismul de intreruperi

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

26.

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

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

29. // Enables conversion complete interrupts

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

31.

32. NVIC\_ClearPendingIRQ(ADC0\_IRQn);

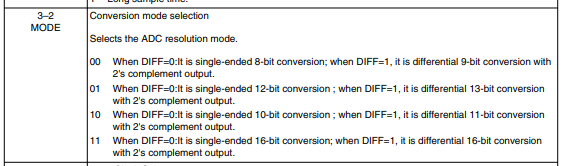
33. NVIC\_EnableIRQ(ADC0\_IRQn);

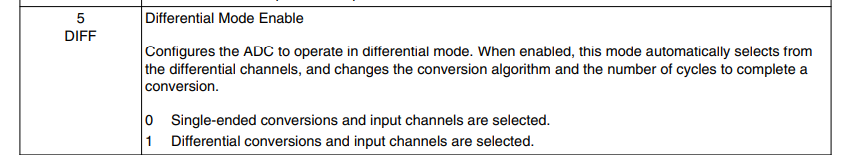
34. //NVIC\_DisableIRQ(ADC0\_IRQn);

35. }

S-a setat modul pe 8 biți single-ended când DIFF=0:

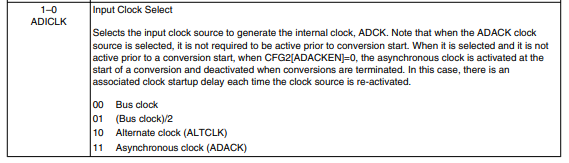
*ADC\_CFG1\_MODE(0)*



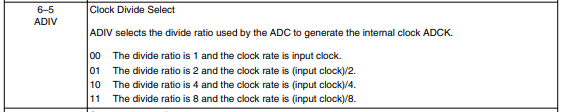


S-a setat ca sursă de ceas bus clock:

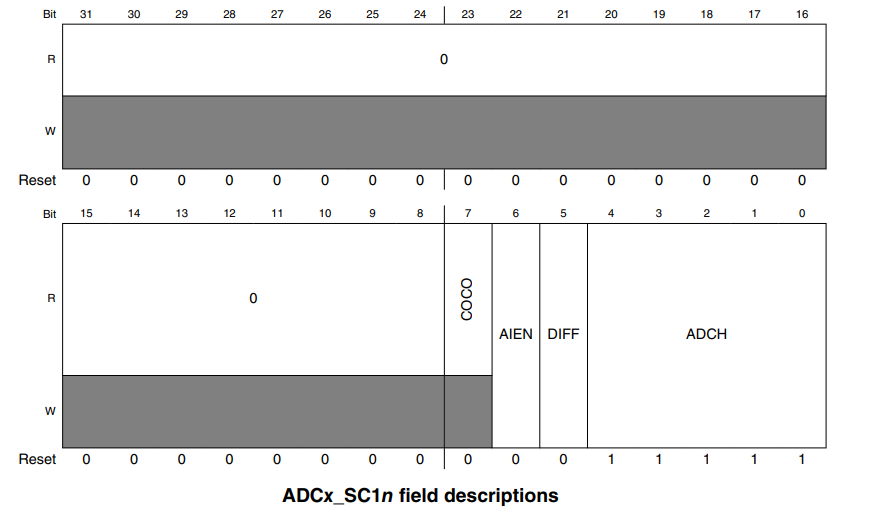
*ADC\_CFG1\_ADICLK(0)*



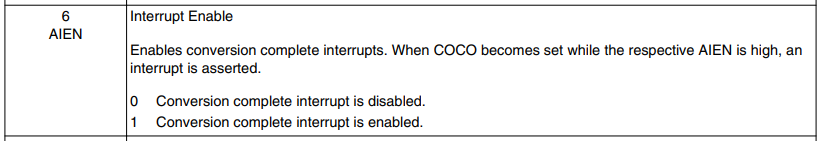
și ca rată de divizare:

 *ADC\_CFG1\_ADIV(2)*

Activăm conversiile single-ended.

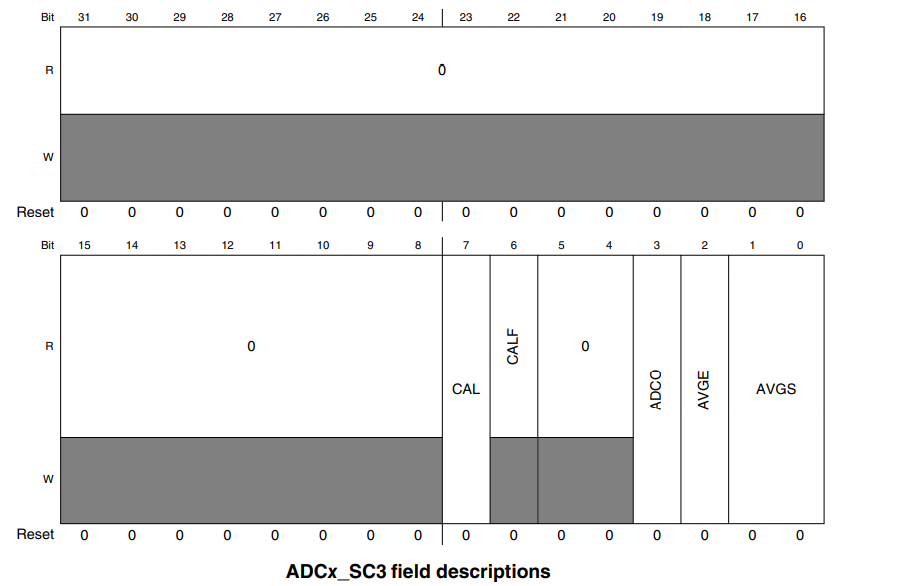


Activăm întreruperile la terminarea conversiilor:

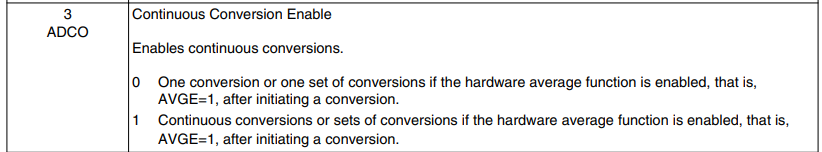
*ADC0->SC1[0] |= ADC\_SC1\_AIEN\_MASK;*

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

*ADC0->SC3 = 0x00;*

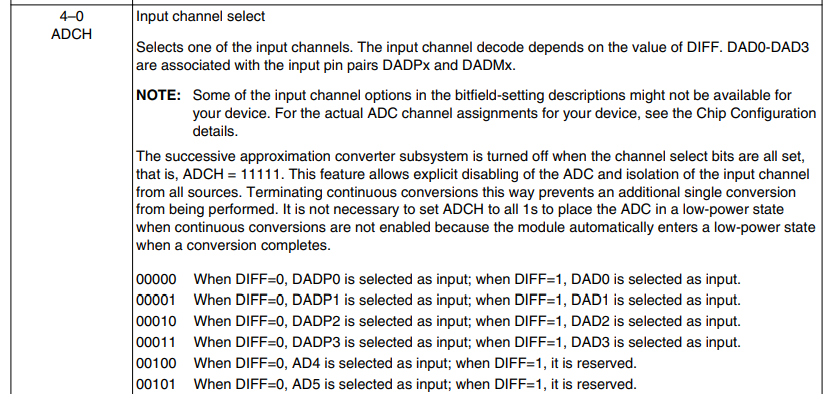


Activăm conversiile continue pentru a-l putea folosi in tandem cu mecanismul de intreruperi:

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

Selectăm canalul specific senzorului (11 pentru senzorul de sunet):

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



Funcția de calibrare pentru conversii ale modulului ADC, conform manualului de referință al platformei doar că s-a setat modul pe 8 biți single-ended:

*ADC\_CFG1\_MODE(0)*

1. int ADC0\_Calibrate() {

2.

3. // ===== For best calibration results =====

4.

5. ADC0\_CFG1 |= ADC\_CFG1\_MODE(0) | // 8 bits mode

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

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

8.

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

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

11.

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

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

14.

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

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

17.

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

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

20.

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

22. // set, the automatic calibration routine is completed successfully. (PG. 495)

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

24. return (1);

25. }

26.

27. // ====== CALIBRATION FUNCTION (PG.495) =====

28.

29. // 1. Initialize or clear a 16-bit variable in RAM.

30. uint16\_t calibration\_var = 0x0000;

31.

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

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

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

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

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

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

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

39.

40. // 3. Divide the variable by two.

41. calibration\_var /= 2;

42.

43. // 4. Set the MSB of the variable.

44. calibration\_var |= 0x8000;

45.

46. // 5. Store the value in the plus-side gain calibration register PG.

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

48.

49. // 6. Repeat the procedure for the minus-side gain calibration value.

50. calibration\_var = 0x0000;

51.

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

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

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

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

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

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

58.

59. calibration\_var /= 2;

60.

61. calibration\_var |= 0x8000;

62.

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

64.

65. // Incheierea calibrarii

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

67.

68. return (0);

69. }

70.

Funcția *ADC0\_Read* citește valori de la senzorul de sunet analogic conform manualului:

1. uint8\_t ADC0\_Read(){

2.

3. // A conversion is initiated following a write to SC1A, with SC1n[ADCH] not all 1's (PG. 485)

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

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, Rn (PG. 486)

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 (PG. 486)

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

17. return (uint8\_t) ADC0->R[0];

18.

19. }

La fiecare întrerupere de ceas, funcția *ADC0\_IRQHandler* preia valorile de la senzorul de sunet din *registrul ADC0\_R* pe care le stochează în variabila *analog\_input* și se dezactivează întreruperile:

1. void ADC0\_IRQHandler(){

2.

3. analog\_input = (uint8\_t) ADC0->R[0];

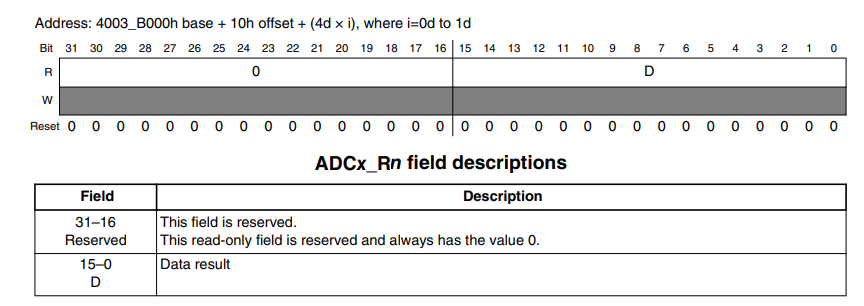
4.

5. flag=1;

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

7.

8. }



### Inițializarea modului TPM și generare PWM

Vom folosi modulul TPM2 (Timer/PWM Module) pentru generarea semnalului PWM.

Setarea canalului pentru servomotor:

1. #define SERVO\_PIN (1) // PORT A , PIN 1

1. void TPM2\_Init(){

2.

3. // Activarea semnalului de ceas pentru utilizarea LED-ului de culoare rosie

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

5.

6. // Utilizarea alternativei de functionare pentru perifericul TMP

7. // TMP2\_CH0

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

9.

10. PORTB->PCR[SCOPE\_PIN\_TPM2\_CH0] |= PORT\_PCR\_MUX(3);

11. // Selects the clock source for the TPM counter clock (MCGFLLCLK) - PG. 196

12. // MCGFLLCLK Freq. - 20 MHz

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

14.

15. // Activarea semnalului de ceas pentru modulul TPM

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

17.

18. // Divizor de frecventa pentru ceasul de intrare

19. // PWM CLK -> 48MHz / 128 = 48.000.000 / 128 [Hz] = 375.000 [Hz] = 375 kHz

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

21.

22. // LPTPM counter operates in up counting mode. - PG. 553

23. // Selects edge aligned PWM

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

25.

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

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

28.

29.

30. // LPTPM counter operates in up-down counting mode. - PG. 553

31. // Selects center aligned PWM

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

33.

34.

35. // Edge-Aligned Pulse Width Modulation

36. // TPM->SC[CPWMS] = 0

37.

38. // ===== Mode selection ====

39. // Configured for EPWM

40. // TPM->CnSC[MnSB] = 1

41. // TPM->CnSC[MnSB] = 0

42.

43. // ==== Edge selection ====

44. // Set output on counter overflow, clear output on match

45. // Counter overflow = LPTPM counter reaches the value configured in the MOD register, and then resets

46. // Match = LPTPM counter reaches the value configured in the CnV register

47.

48. // TPM->CnSC[ELSnB] = 1

49. // TPM->CnSC[ELSnA] = 0

50.

51. // Regiter associated to the control of channel 0

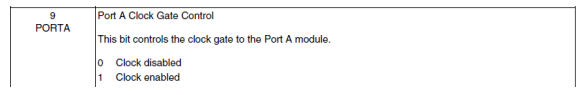
52. // Why channel 0?

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

54. }

55.

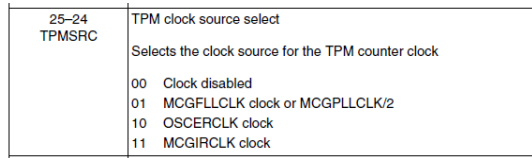
Vom porni ceasul portului A, unde se află servomotorul conectat prin setarea bitului al 9-lea al registrului SIM\_SCGC5 pe valoarea 1 cu ajutorul măștii SIM\_SCGC5\_PORTA\_MASK ce are valoarea 0x200:



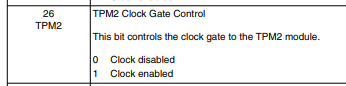
Setăm multiplexarea pinului 1 din portul A pe canalul 0 al modului 2 TPM:

*PORTA->PCR[SERVO\_PIN] |= PORT\_PCR\_MUX(3);*

Setăm FLL ca sursă de ceas prin setarea câmpului TPMSRC (biții 25-24, TPM clock source select) din registrul System Options Register 2 (SIM\_SOPT2) pe valoarea 0b01 (MCGFLLCLK).

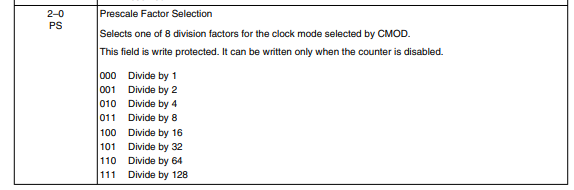
*SIM->SOPT2 |= SIM\_SOPT2\_TPMSRC(1);*

Activăm ceasul modului 2 TPM prin setarea câmpului TPM0 (bitul 24) din SIM\_SCGC6 pe 1.

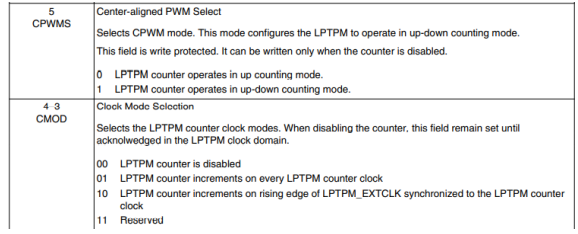
*SIM->SCGC6 |= SIM\_SCGC6\_TPM2(1);*

Prin setarea câmpului PS (biții 2-0, Prescale Factor Selection) pe valoarea 0b111, ceasul sistemului va fi divizat cu 128, deci ceasul PWM va avea frecvența de 375 KHz:

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



Setăm semnalul PWM generat ca fiind edge-aligned, setând counterul în mod up counting prin valoarea 0 în câmpul CPWMS (bitul 5, Center-aligned PWM Select) și valoarea 0b01 în câmpul CMOD (biții 4-3, Clock Mode Selection) pentru incrementarea numărătorului LPTPM.



Funcția *Signal\_Control* are rolul de a reseta valorea pentru counter-ul LPTPM (valoarea până la care trebuie să numere pentru a termina o perioadă) și de a seta pentru canalul 0 anumite valori specifice modului de rotire a elicei servomotorului, valori necesare pentru calcularea factorului de umplere(duty-cycle).

1. void Signal\_Control(uint8\_t position){

2.

3. // Resetarea valorii numaratorului asociat LPTPM Counter

4. TPM2->MOD = 7500; // 375,000 Hz / 7500 = 50 Hz (20 ms)

5.

6. // Setarea duty cycle-ului pentru 3 pozitii

7. switch (position) {

8. case 0:

9. TPM2->CONTROLS[0].CnV = 188; // 2.5% pentru 0°

10. break;

11. case 1:

12. TPM2->CONTROLS[0].CnV = 562; // 7.5% pentru 90°

13. break;

14. case 2:

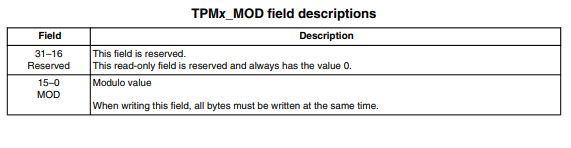
15. TPM2->CONTROLS[0].CnV = 937; // 12.5% pentru 180°

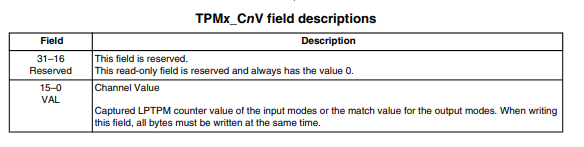
16. break;

17. }

18. }

19.





## *Configurarea ceasului*

Modulul MCG (Multipurpose Clock Generator) oferă semnale de ceas pentru unitatea centrală. Acesta conține un FLL (frequency-locked loop), controlabil de către un ceas de referință intern.

1. void SystemClock\_Configure(void) {

2.

3.

4. // MCGOUTCLOCK are ca si iesire, selectia FLL

5. MCG->C1 |= MCG\_C1\_CLKS(0);

6.

7. // Selectarea ceasului intern ca si referinta pentru sursa FLL

8. MCG->C1 |= MCG\_C1\_IREFS\_MASK;

9.

10. // Setarea frecventei ceasului digital la 48 Mhz

11. // MCG->C4[DRST\_DRS] = 1

12. // MCG->C4[DMX32] = 1

13. MCG->C4 |= MCG\_C4\_DRST\_DRS(1);

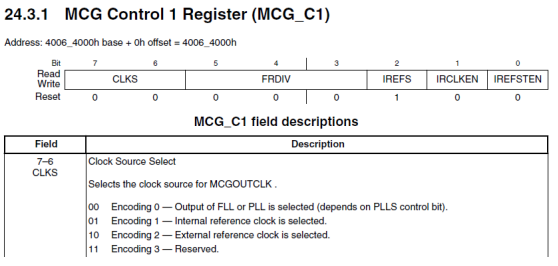
14. MCG->C4 |= MCG\_C4\_DMX32(1);

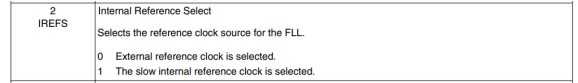
15.

16. }

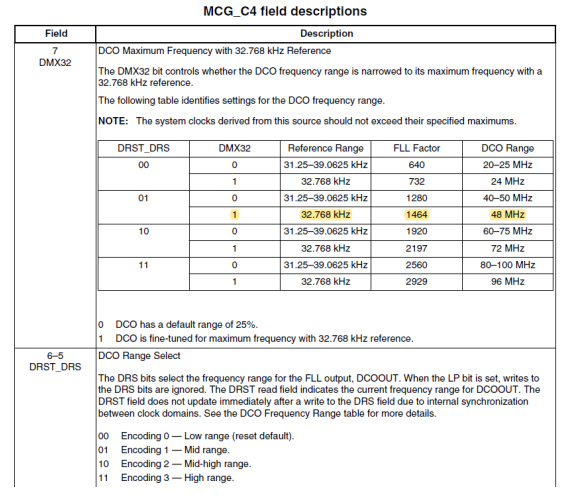
17.

În MCG\_C1 (MCG Control 1 Register) am setat câmpul CLKS (Clock Source Select, biții 7-6) pe valoarea 0b00 pentru selectarea FLL/PLL ca output.



Cel folosit va fi selectat prin câmpul IREFS (Internal Reference Select, bitul 2) pe valoarea 1 – se selectează ceasul intern de referință, așadar se folosește FLL (PLL – phase-locked loop este configurabil doar extern).

În MCG\_C4 (MCG Control 4 Register), setăm câmpul DRST DRS (biții 6-5) pe valoarea 0b01 – se selectează frecvența DCO (Digitally-controlled oscillator) ca fiind una medie (40 – 50 MHz).

În același registru setăm câmpul DMX32 (DCO Maximum Frequency, bitul 7) pe valoarea 1 – DCO va avea frecvența de 48 MHz.

Configurarea întreruperii de ceas:

1. void SystemClockTick\_Configure(void){

2.

3. // Deoarece imi doresc ca frecventa de generare a semnalului PWM sa tina cont de tick-uri de 1 ms,

4. // trebuie realizate urmatoarele configurari

5. // 48.000.000 pulsuri .......... 1 secunda

6. // x pulsuri .......... 10^(-3) secunde

7. //

8. SysTick->LOAD = (uint32\_t)(48000000UL / 1000 - 1UL);

9.

10. NVIC\_SetPriority(SysTick\_IRQn, (1UL << \_\_NVIC\_PRIO\_BITS) - 1UL);

11.

12. // Resetarea numaratorului digital

13. SysTick->VAL = 0UL;

14.

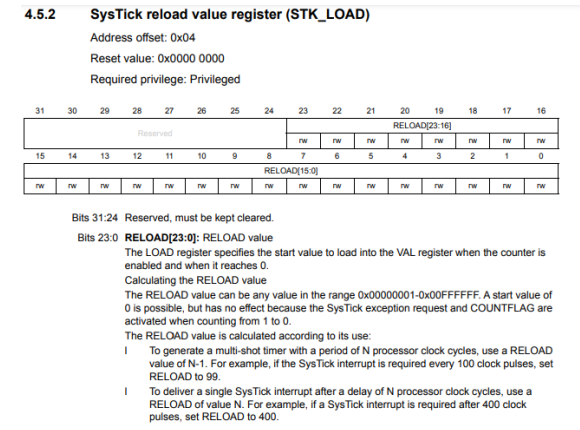
15. SysTick->CTRL |= (SysTick\_CTRL\_CLKSOURCE\_Msk | // Ceasul procesorului utilizat ca referinta (48Mhz)

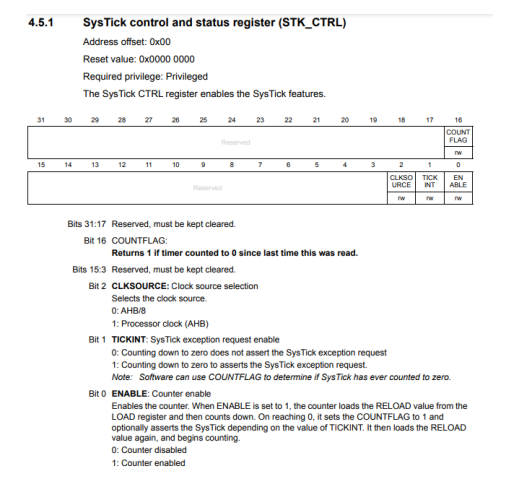
16. SysTick\_CTRL\_TICKINT\_Msk | // Atunci cand numaratorul ajunge la 0, va porni handler-ul de intrerupere

17. SysTick\_CTRL\_ENABLE\_Msk); // Incarca valoarea pusa in registrul RELOAD si incepe numaratoarea

18. }

19.

Setăm valoarea registrului de reload (STK\_LOAD) cu valoarea de resetare a timerului, raportul dintre cele două frecvențe, care specifică intervalul de numărare al counterului.

Setăm prioritatea întreruperii SysTick pe valoarea maximă și valoarea inițială pe 0. În registrul STK\_CTRL (registrul de control și stare SysTick) setăm biții 2 (CLKSOURCE – selectează ca sursă ceasul procesorului), 1 (TICKINT – activează întreruperea la valoarea 0) și 0 (ENABLE – pornește timerul).

## *Funcția main*

Funcția main include toate fișierele header în care sunt declarate funcțiile și variabilele ce au fost folosite în fiecare modul și are rolul de a inițializa toate modulele descrise anterior:

1. #include "ClockSettings.h"

2. #include "Pwm.h"

3. #include "Pit.h"

4. #include "adc.h"

5. #include "gpio.h"

6. #include "uart.h"

7.

1. int main() {

2.

3. SystemClock\_Configure();

4. SystemClockTick\_Configure();

5. OutputPIN\_Init();

6. TPM2\_Init();

7. UART0\_Init(115200);

8. ADC0\_Init();

9. PIT\_Init();

10.

11. for(;;){

12. if(flag){

13.

14. int i;

15. char v[8];

16. int count = 0;

17. int value;

18.

19. value = (int)analog\_input;

20.

21. // Afiseaza valoarea obtinuta

22. UART0\_Transmit('S');

23. UART0\_Transmit('N');

24. UART0\_Transmit('D');

25. UART0\_Transmit(':');

26.

27. for (i = 0; i < 8; i++) {

28. v[i] = '-';

29. }

30.

31. while (value != 0) {

32. v[count] = (char)(value % 10) + 0x30;

33. value = value / 10;

34. count = count + 1;

35. }

36.

37. for (i = 0; i < count; i++) {

38. UART0\_Transmit(v[count - i - 1]);

39. }

40.

41.

42. UART0\_Transmit(0x0A);

43. UART0\_Transmit(0x0D);

44.

45.

46. flag=0;

47. }

48.

49. if(flag\_1s){

50. if(analog\_input < 23){

51. Signal\_Control(0);

52. }

53. else if(analog\_input >= 23 & analog\_input < 46){

54. Signal\_Control(1);

55. }

56. else if(analog\_input > 46){

57. Signal\_Control(2);

58. }

59. flag\_1s = 0U;

60. }

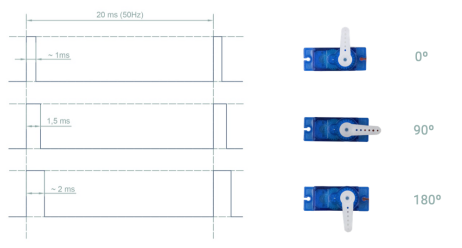
61. }

62. }

63.

Logica programului este următoarea:

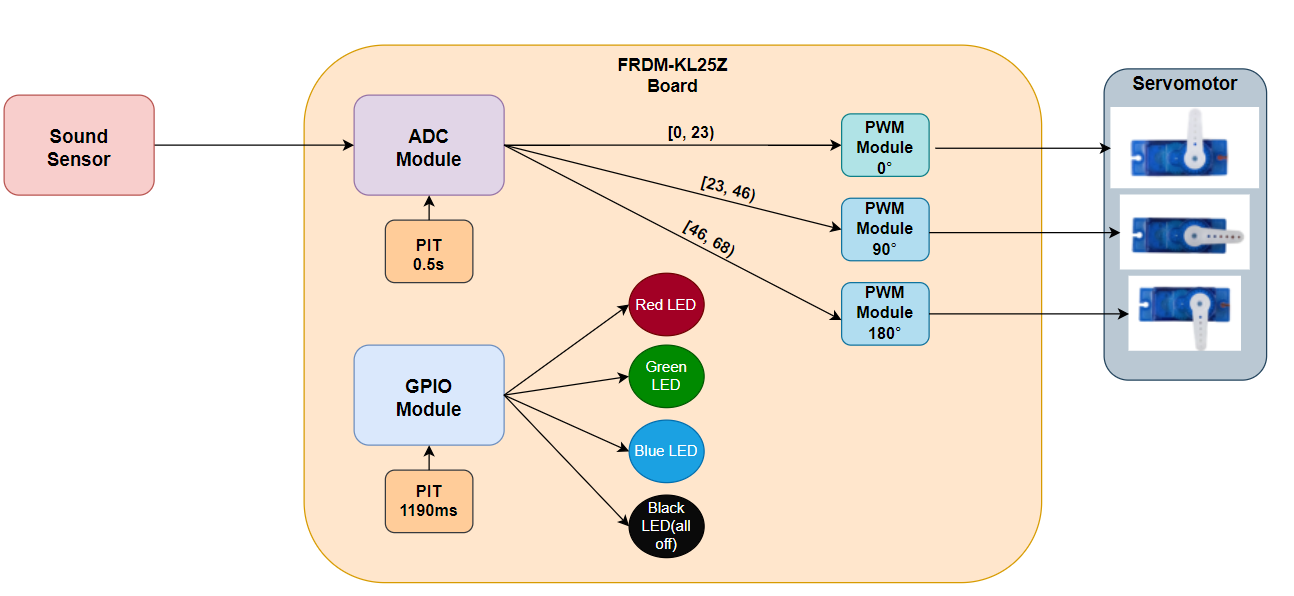
* Se apelează funcția de inițializare a timerelor și modulelor;
* În ciclul infinit se verifică dacă se primesc date de la senzorul de sunet. Dacă se primesc, acestea sunt transformate în numere întregi și stocate în variabila *value* care mai apoi sunt transformate într-un vector de caractere pentru a putea fi printabile în UART. La final are loc resetarea flag-ului la 0 pentru a indica faptul că acest bloc de cod s-a executat;
* După se verifică la fiecare trecere de 1s în ce interval se încadreaza valoarea primită de la senzorul de sunet astfel:
  + [0, 23) – setează factorul de umplere la 188 specific pentru 2.5% pentru 0° ce determină poziția elicei servomotorului în sus;
  + [23, 46) - setează factorul de umplere la 562 specific pentru 7.5% pentru 90° ce determină poziția elicei servomotorului în dreapta;
  + [46, 68) - setează factorul de umplere la 937 specific pentru 12.5% pentru 180° ce determină poziția elicei servomotorului în jos;

*OBS: valoare max pentru stabilirea intervalului a fost măsurat anterior de noi prin transmiterea mai multor sunete prin intermediul senzorului de sunet și am constat faptul că acesta are max=68; apoi a avut loc împărțirea intervalui în cele 3 subintervale specifice poziției elicei servomotorului.*

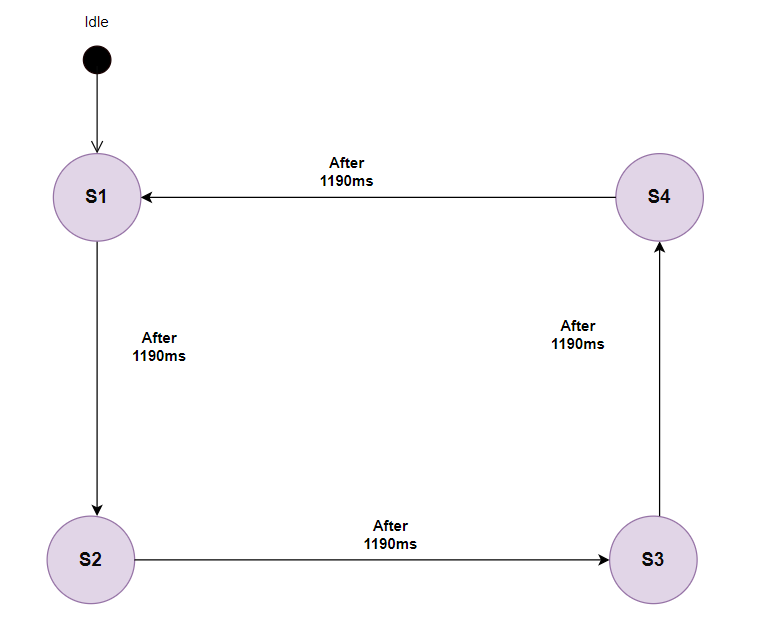
# **Set-up**

Poze cum am conectat totul si cum trb rulat etc pasii gen TO DO

# **Schemă bloc**



# **Diagrame de stări**



|  |  |  |
| --- | --- | --- |
| Stare | ledState | Observații |
| S1 | 0 | LED-ul roșu este aprins (**Red**) |
| S2 | 1 | LED-ul verde este aprins (**Green**) |
| S3 | 2 | LED-ul albastru este aprins (**Blue**) |
| S4 | default | Toate LED-urile sunt stinse (**Black**) |

# **Rezultate aplicație Python**

# **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.

# **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/>