**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**Facultatea de Sisteme Informatice și Securitate Cibernetică**

**Departamentul de Calculatoare și Securitate Cibernetică**



***Utilizare senzor ANALOGIC SUNET***

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

Std. sg. maj. Bianca-Daniela IONASCU

Std. sg. maj. Radu-Marian Toader

Grupa C114D-C114C

**București**

**2023**

Cuprins

[1. Prezentarea componentelor 3](#_Toc155114575)

[1.1. DFR0034 – Sound sensor 3](#_Toc155114576)

[1.2. DFR0032 – Buzzer 4](#_Toc155114577)

[1.3. Senzor lumina 5](#_Toc155114578)

[2. Scop proiect 6](#_Toc155114579)

[3. Conectare senzori – placă de dezvoltare 7](#_Toc155114580)

[4. Descriere program 8](#_Toc155114581)

[4.1. Funcția main 8](#_Toc155114582)

[4.2. Inițializarea modulelor 8](#_Toc155114583)

[4.2.1. Configurarea ceasului 9](#_Toc155114584)

[4.2.2. Configurarea întreruperii de ceas 10](#_Toc155114585)

[4.2.3. Inițializarea modulului UART 12](#_Toc155114586)

[4.2.4. Inițializarea modulului TPM 15](#_Toc155114587)

[4.2.5. Inițializarea modulului GPIO 17](#_Toc155114588)

[4.3. Generare PWM 17](#_Toc155114589)

[4.4. Transmitere date prin UART 19](#_Toc155114590)

[5. Rezultate 19](#_Toc155114591)

[6. Dificultăți întâmpinate 20](#_Toc155114592)

[7. Referinte 21](#_Toc155114593)

# Prezentarea componentelor

## DFR0034 – Sound sensor

Senzorul DFR0034 este un senzor analog pentru detectia sunetului cu output active LOW (valoarea 1 înseamnă că nu este detectat sunet, valoarea 0 apare doar la detectia sunetului).



Figura 1. Senzorul DFR0034 [1]

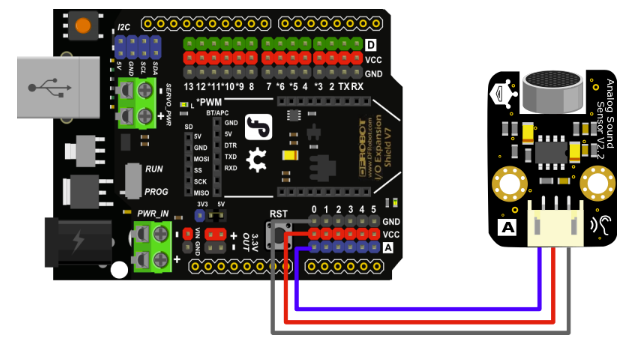


Figura 2. Exemplu conectare senzor DFR0034 [1]

Specificatii:

* Tensiune suportata: 3.3-5V
* Interfata analogica

## DFR0032 – Buzzer

DFR0032 este un dispozitiv electronic simplu, destinat sa produca sunete atunci cand este activat printr-un semnal digital.

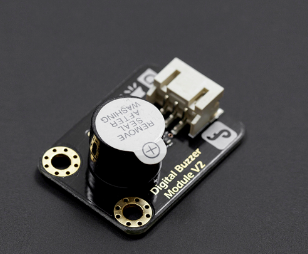


Figura 3. Senzorul DFR0032 [2]

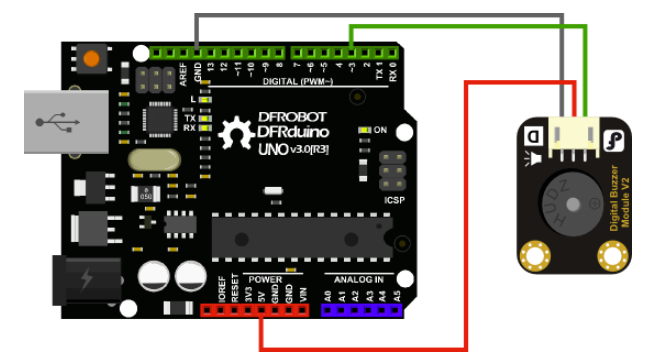


Figura 4. Exemplu conectare senzor DFR0032 [2]

Specificatii:

* Tensiune suportata: 5V (in urma testelor, functioneaza si la 3.3)
* Interfata digitala

## Senzor lumina

Acest modul este un senzor digital activ pe 0, ce poate detecta intensitatea luminoasa in functie de sensibilitate. Sensibilitatea acestuia este reglata printr-un procedeu mecanic (se strange/slabeste surubul fizic atasat pe placa).



Figura 5. Senzor lumina [3]

Conectarea senzorului se face prin cei 3 pini asociati, acestia fiind marcati direct pe modul:

* Pin 1: VCC (sursa de curent)
* Pin 2: GND (ground-ul)
* Pin 3: DO (data)

Specificatii:

* Tensiune suportata: 3.5-5V (din teste, rezulta ca lucreaza si la 3.3V)
* Interfata digitala

# Scop proiect

Scopul acestui proiect este de a oferi o experienta de invatare si practica in dezvoltarea de software la nivelul hardware-ului, ceea ce inseamna programarea directa a unei placi, fara utilizarea unui sistem de operare. Acest lucru presupune interactiunea directa cu componentele hardware, cum ar fi senzori sau module de comunicare.

Ne propunem sa atingem cateva obiective:

* Intelegerea conceptelor hardware;
* Scrierea de drivere;
* Imbunatatirea abilitatilor de depanare;
* Crearea de aplicatii personalizate.

Pentru atingerea obiectivelor anterior mentionate, exista si o serie de cerinte:

1. Aprinderea led-ului integrat pe placa, in diferite culori (mentionate in documentul cu cerinte), care se va schimba la un anumit interval de timp.
2. Posibilitatea schimbarii directiei pentru culori, printr-un buton din interfata grafica.
3. Citirea si conversia detelor primite de la senzorul de sunet, acesta fiind un senzor analogic se va folosi un convertor ADC.
4. Citirea datelor de la un senzor digital (senzorul de lumina, prezentat anterior).
5. Transmiterea de date catre un buzzer, atunci cand datele senzorului de sunet depasesc un interval, cu scopul de a semnaliza schimbarea. Aceste sunete vor fi de scurta durata si se vor repeta in functie de intervalul in care se trece.
6. Transmiterea datelor citite de la senzori pe interfata seriala UART catre calculator.
7. Calculatorul o sa expuna o interfata grafica in care se afiseaza datele primite de la senzorul de sunet, intr-un mod usor de interpretat.
8. Calculatorul o sa expuna cateva butoane cu rolul de a controla anumita functionalitati alea software-ului incarcat pe placa (schimbarea ordinii culorilor, pornirea melodiei, citirea datelor de la senzorul de lumina).

Aceasta aplicatie ar putea fi folosită în mai multe contexte practice:

* **Monitorizarea zgomotului ambiental**: Aplicația poate fi folosită pentru a măsura nivelurile de zgomot în diferite medii, cum ar fi locuri de muncă, școli sau zone rezidențiale, pentru a asigura respectarea normelor de zgomot.
* **Sistem de alertă pentru persoane cu deficiențe auditive**: Pentru persoanele cu dificultăți de auz, aplicația poate transforma sunetele detectate în semnale vizuale sau tactile (prin bipuri ale buzzerului), ajutându-le să recunoască prezența unor sunete importante în mediul înconjurător.
* **Monitorizarea siguranței la domiciliu sau la locul de muncă**: Aplicația poate fi utilizată ca un sistem de alarmă care detectează sunete neobișnuite sau suspecte, cum ar fi spargerea unei ferestre sau un strigăt de ajutor, activând buzzerul pentru a alerta proprietarii sau personalul de securitate.

# Conectare senzori – placă de dezvoltare

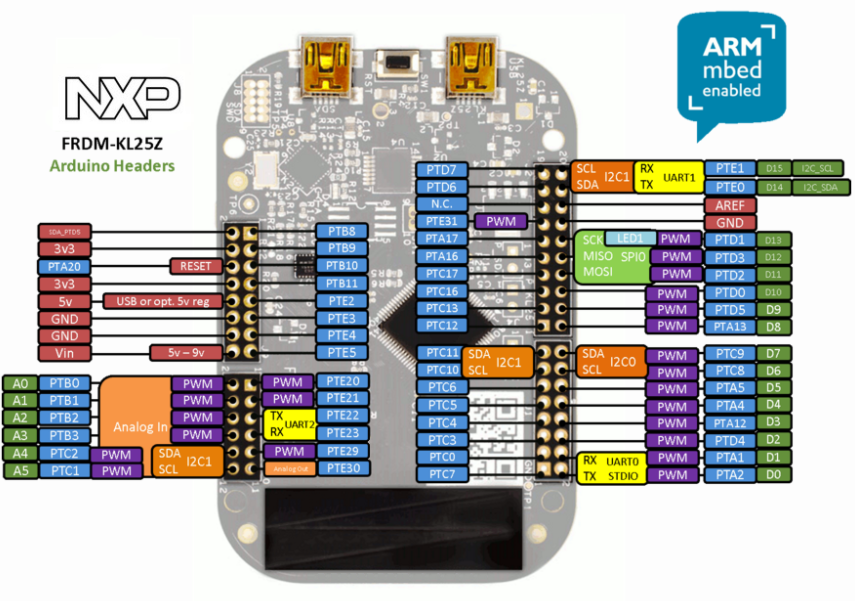


Figura 6. Pini placă dezvoltare [4]

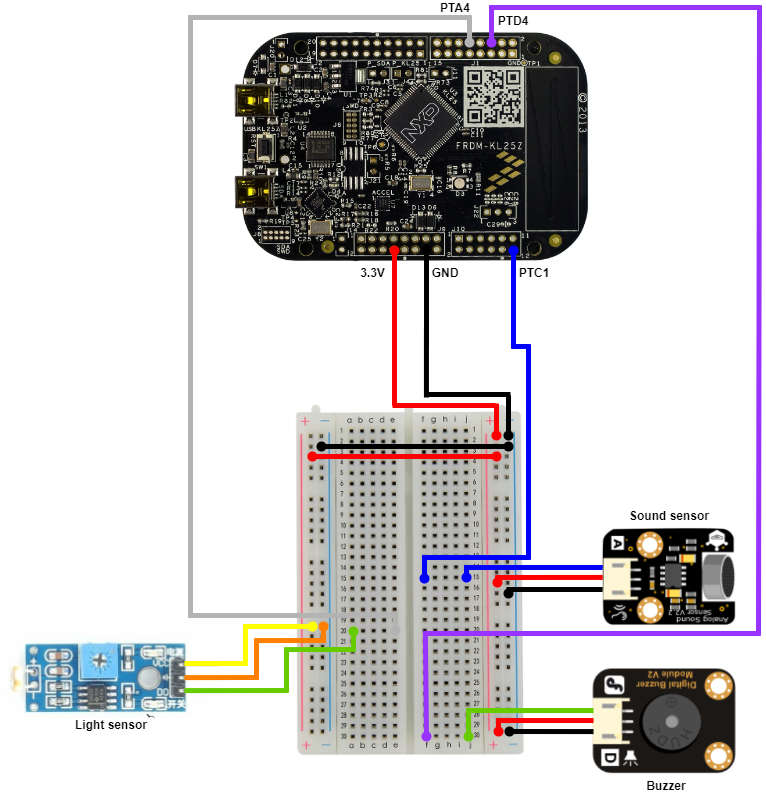


Figura 7. Mod de conectare

# Descriere program

## Funcția main

În fișierul main.c am inclus fișierele header în care sunt declarate funcții și variabile ce urmează a fi folosite:

1. #include "Pit.h"

2. #include "Gpio.h"

3. #include "Uart.h"

4. #include "SysTick.h"

5. #include "Adc.h"

6.

7. int main(void)

8. {

9. UART0\_Init(115200);

10. Buzzer\_Init();

11. ADC0\_Init();

12.

13. PIT\_Init();

14. Init\_SysTick();

15. RGBLed\_Init();

16.

17. Light\_Init();

18.

19. float prev = 0;

20.

21. for (;;)

22. {

23. if (play == 1)

24. {

25. Buzzer\_Sing();

26. play = 0;

27. }

28.

29. if(flagADC)

30. {

31. float measured\_voltage = 3.3f - (analog\_input \* 3.3f) / 1023;

32.

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

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

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

36.

37. uint8\_t lightData = Light\_Read() + 0x30;

38.

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

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

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

42. UART0\_Transmit(lightData);

43.

44. if(prev < 0.2 && measured\_voltage > 0.4)

45. Buzzer\_Sound(2);

46. else if(prev < 0.2 && measured\_voltage > 0.2)

47. Buzzer\_Sound(1);

48.

49. flagADC=0;

50. }

51. }

52. }

* *Pit.h*

1. /\* variabila de control pentru ordinea secventei led-urilor \*/

2. static uint8\_t order;

3.

4. /\* variabila pentru memorarea starii curente a led-ului \*/

5. static uint8\_t state;

6.

7. /\* functie de initializare a modulului periferic (PIT[0]) \*/

8. void PIT\_Init(void);

9.

10. /\* functia de tratare a intreruperilor pentru perifericul PIT \*/

11. void PIT\_IRQHandler(void);

12.

13. /\* functia care schimba ordinea de aprindere a culorilor(schimbare valoare variabila order)\*/

14. void PIT\_LED\_Change\_Order(void);

* *Gpio.h*

1. #define RED\_LED\_PIN 18 /\* PORT B PIN 18 \*/

2. #define GREEN\_LED\_PIN 19 /\* PORT B PIN 19 \*/

3. #define BLUE\_LED\_PIN 1 /\* PORT D PIN 1 \*/

4.

5. #define LIGHT\_PIN 4 /\* PORT A PIN 4 \*/

6.

7. #define BUZZER\_PIN (4) /\* PORT D PIN 4 \*/

8.

9. /\* variabila pentru declansarea buzzerului cu o melodie \*/

10. extern volatile uint8\_t play;

11.

12. /\* initializarea led-urilor \*/

13. void RGBLed\_Init(void);

14.

15. /\* initializarea buzzerului \*/

16. void Buzzer\_Init(void);

17.

18. /\* initializarea senzorului de lumina \*/

19. void Light\_Init(void);

20.

21. /\* functie pentru citirea datelor de la senzor de lumina \*/

22. uint8\_t Light\_Read(void);

23.

24. /\* activarea buzzerului pentru declansarea semnalelor sonore pentru cele 3 intervale \*/

25. void Buzzer\_Sound(uint8\_t noBip);

26.

27. /\* pornirea melodiei \*/

28. void Buzzer\_Sing(void);

* *Uart.h*

1. /\* functie folostia pentru a trimite un octet pe interfata UART \*/

2. void UART0\_Transmit(uint8\_t data);

3.

4. /\* functie ce citeste un octet de pe interfata UART \*/

5. uint8\_t UART0\_Receive(void);

6.

7. /\* initializare a modulului UART pentru trasmiterea datelor, ce primeste ca parametru baud rate-ul \*/

8. void UART0\_Init(uint32\_t baud\_rate);

9.

10. /\* functie de tratare a intreruperii pentru perifericul UART (receptie) \*/

11. void UART0\_IRQHandler(void);

* *SysTick.h*

1. /\* flag pentru semnalizarea resetarii contorului \*/

2. extern volatile uint8\_t flagSysTick;

3.

4. /\* functie de initializare pentru SysTick \*/

5. void Init\_SysTick(void);

6.

* *Adc.h*

1. /\* PORT C PIN 1 => canalul 15\*/

2. #define ADC\_CHANNEL (15)

3.

4. /\* semnalizare ca s-a citit valoarea si se poate trasmite \*/

5. extern volatile uint8\_t flagADC;

6.

7. /\* valoarea citita de la senzor \*/

8. extern uint16\_t analog\_input;

9.

10. /\* initializarea ADC-ului \*/

11. void ADC0\_Init(void);

12.

13. /\* functie de calibrare de la ADC \*/

14. int ADC0\_Calibrate(void);

15.

16. /\* tratarea intreruperilor pentru perifericul ADC \*/

17. void ADC0\_IRQHandler(void);

18.

* *MKL25Z4.h* = fișierul header generat de către mediul de dezvoltare Keil, specific plăcii de dezvoltare

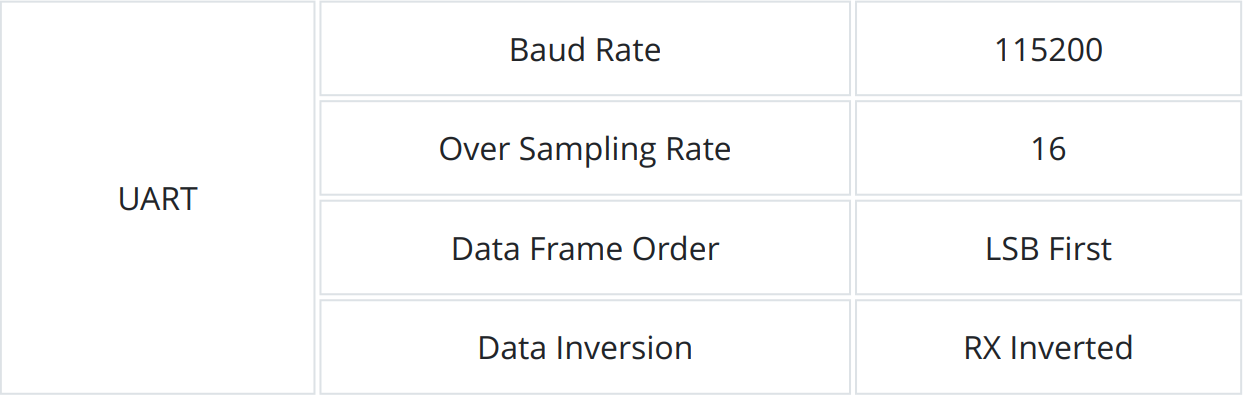
Logica principală a programului este următoarea: întâi se apelează funcțiile de inițializare a modulelor, apoi, într-un ciclu infinit, se verifică daca butonul de play a fost apasat, atunci va porni muzica cu ajutorul buzzerului si totodata, se face si verificarea daca flagADC este activ (1) pentru a transmite date catre UI, acesta va fi activ cand se va apela intreruperea de la sysTick, adica la intervale de 100 ms. Daca valoarea citita de la senzor trece intr-un alt interval, buzzerul va fi actionat in conformitate.

## Inițializarea modulelor

Prin apelul functiilor de initializare, se configureaza anumite flag-uri si semnale necesare pentru constructia anumitor module. Avem urmatoarele periferice initializate:

1. *UART*
2. *PIT*
3. *SYSTICK*
4. *GPIO*
5. *ADC*

### Inițializarea modulului UART



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

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

2. {

3. /\* over-Sampling Rate (numarul de esantioane luate per bit-time) \*/

4. uint32\_t osr = 16;

5.

6. /\* activarea semnalului de ceas pentru modulul UART0 \*/

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

8.

9. /\* activarea semnalului de ceas pentru portul A \*/

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

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

12.

13. /\* ne asiguram ca transmitter-ul si receiver-ul sunt dezactivate inainte de initializare \*/

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

15.

16. /\* setarea sursei de ceas pentru modulul UART \*/

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

18.

19. /\* fiecare pin pune la dispozitie mai multe functionalitati \*/

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

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); /\* Configurare TX pentru UART0 \*/

25.

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

30. BDH - 0 0 0 b13 b12 b11 b10 b09

31. BDL - b08 b07 b06 b05 b04 b03 b02 b01 \*/

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

// seteaza pe 0 cei mai nesemnificativi 5 biti

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

34. // setarea celor 5 biti asociati lui SBR din BDH(cei mai semnificativi) calculati(sbr- >13 biti)

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

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

37.

38. // salvam valoarea osr-ului

39. UART0->C4 |= UART0\_C4\_OSR(osr - 1);

40.

41. /\* setare numarul de biti de date la 8 si fara bit de paritate \*/

42. /\* UART0->C1 = 0; \*/

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

44.

45. /\* transmisie incepand cu LSB \*/

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

47.

48. /\* dezactivare intreruperi la transmisie \*/

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

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

51.

52. /\* activare intreruperi la receptie \*/

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

54.

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

56.

57. /\* receive data inversion \*/

58. /\* UART0->S2 |= UART0\_S2\_RXINV(1); \*/

59.

60. /\* activam intreruperea \*/

61. NVIC\_ClearPendingIRQ(UART0\_IRQn);

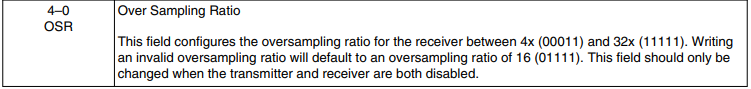
62. NVIC\_EnableIRQ(UART0\_IRQn);

63.

64. \_\_enable\_irq();

65. }

**OSR (Over Sampling Ratio)** este un parametru de configurare care determina de cate ori semnalul de intrare este esantionat pe durata unui bit in transmisia de date. Valoarea implicita a OSR este 0b01111, ceea ce reprezinta valoarea binara pentru numarul 15 => pentru fiecare bit de date primit, receptorul va face 15 esantionari. In cerinta am avut de setat OSR la valoarea 16 => o rata de esantionare mai mare poate imbunatati acuratetea detectarii bitului prin reducerea erorilor cauzate de zgomot sau de alte variatii ale semnalului.

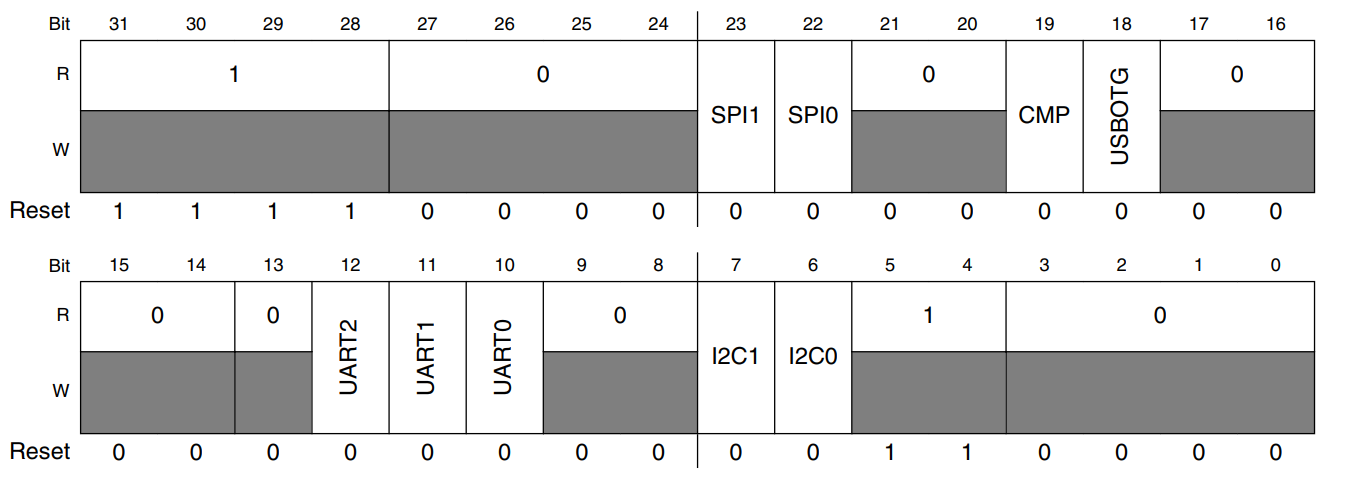


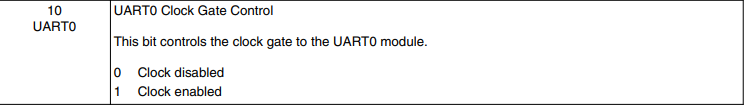
**Data Sampling Tehnique:** (informatie preluata din reference manual [5])

Pentru o decodare corecta a datelor prin esantionare trebuie sa tinem cont de cateva aspecte:

* ***Oversampling:*** Pentru fiecare bit de date, receptorul verifică semnalul de mai multe ori (între 4 și 32 de ori) pentru a fi sigur de ceea ce primește.
* ***Căutarea unui semnal de inceput:*** Receptorul caută un anumit tip de semnal (o scădere bruscă) care indică începutul unui nou set de date.
* ***Confirmarea semnalului de inceput:*** După ce găsește semnalul de început, receptorul verifică încă de câteva ori pentru a se asigura că nu este doar zgomot aleator.
* ***Citirea datelor:*** Pentru fiecare bit, receptorul decide dacă este un 1 sau un 0 bazându-se pe majoritatea rezultatelor din verificările repetate.
* ***Detectarea erorilor:*** Dacă receptorul găsește discrepanțe în semnal, el notează că a fost zgomot în date.
* ***Eșantionare pe două părți:*** Receptorul poate citi semnalul de două ori mai des dacă este setat așa, ceea ce ajută la o citire și mai precisă.
* ***Reînceperea citirii***: Dacă receptorul se pierde sau găsește semnale neașteptate, el începe să citească din nou de la început pentru a se sincroniza bine cu datele care vin.
* ***Reacția la erori:*** Dacă sunt probleme în citirea datelor, receptorul se resetează rapid pentru a fi gata să înceapă citirea unui nou set de date.

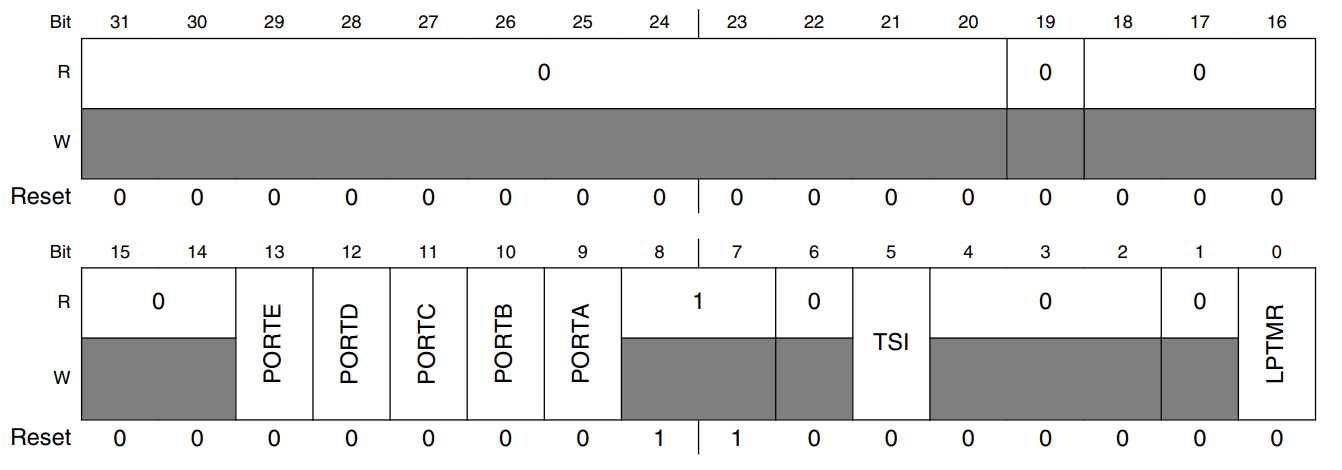
În registrul **SIM\_SCGC4** (System Clock Gating Control Register 4) setăm pe 1 câmpul UART0 (bitul 10) pentru activarea ceasului pentru acest modul. [5]

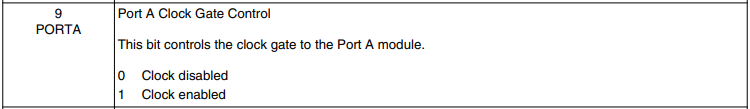




În registrul **SIM\_SCGC5** (System Clock Gating Control Register 5) setăm pe 1 câmpul PORTA (bitul 9) pentru activarea ceasului acestui port, deoarece dorim sa folosim pinii PTA1, respectiv PTA2 pentru comunicarea UART. [5]

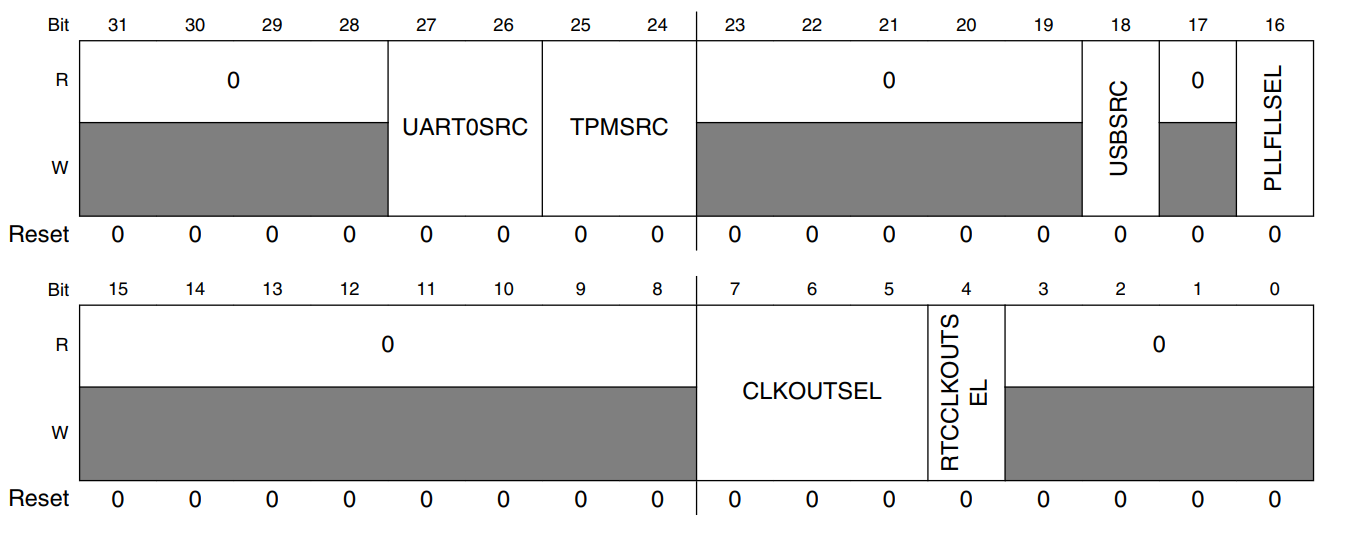




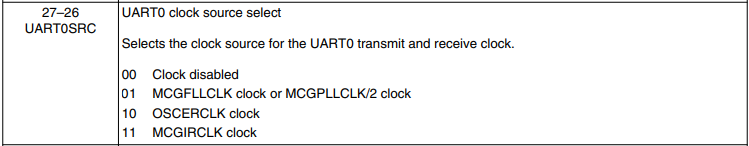


Transmitter-ul si receiver-ul pot fi activati sau dezactivati cu ajutorul campurilor **TE si RE**. Aceste campuri trebuie sa fie dezactivate pentru a putea accesa anumiti registrii de control. [6]

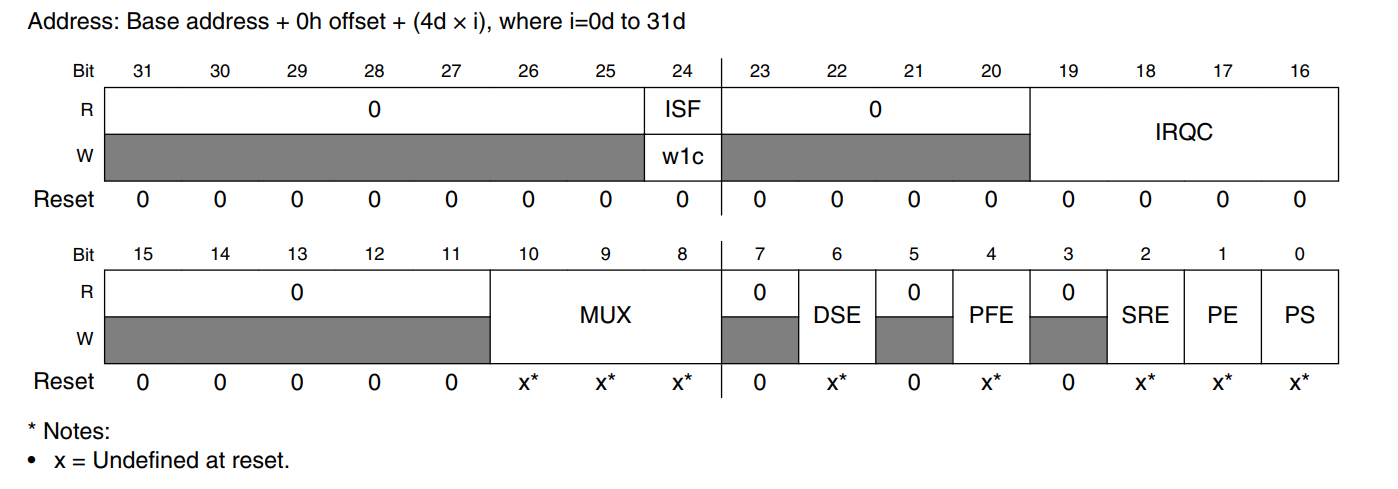
SIM\_SOPT2 (System Options Register 2) contine controalele pentru selectarea unei surse de ceas. [5]



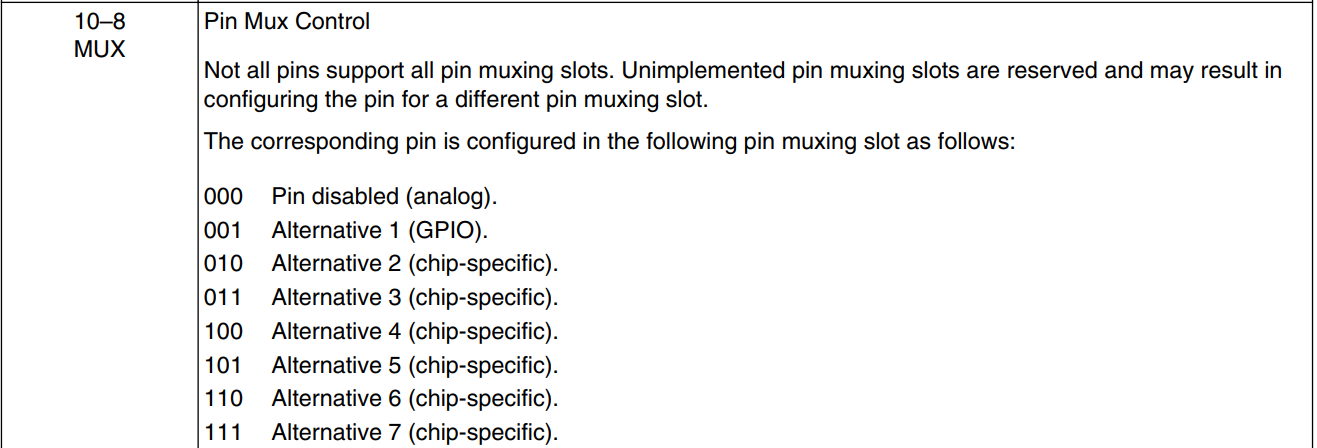
Setăm registrul pe 0b01 câmpul UART0SRC (biții 27-26) pentru selectarea ca ceas al modulului MCGFLLCLK (Multipurpose Clock Generator Filtered Low-Frequency Lock).



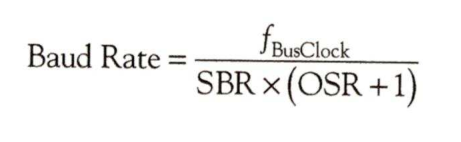
In registrul PCR exista un camp MUX care seteaza alternativa pentru un anumit pin. In functie de alternativa, un pin poate fi folosit cu o anumita functionalitate. In cazul meu o sa setez pentru pinii 1 si 2 de pe portul A (PTA1 si PTA2) campul MUX la alternativa 2, care inseamna ca ii folosesc in modulul UART0 (RX/TX).







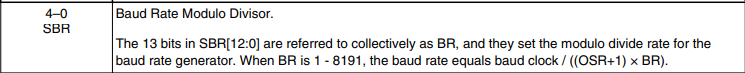
Setam **SBR** la valoarea data de formula: (DEFAULT\_SYSTEM\_CLOCK) / (baud\_rate \* osr), care inlocuita cu valorile corespunzatoare este: 48 MHz/(115200 \* 16), care este aproximativ 26. [6]

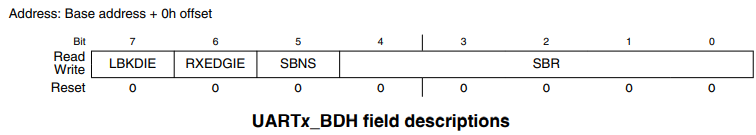


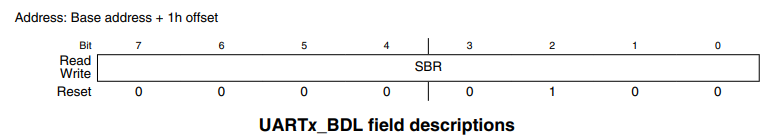
In formula de mai sus SBR este calculat cu OSR + 1 dar eu am folosit OSR = 16 si nu am folosit OSR + 1 ca nu a mai fost nevoie.

Valoarea calculata a lui SBR va fi pusa astfel:

* Ignorand primii 3 biti ai variabilei pe 16 biti SBR (deoarece cele 2 campuri adunate din BDH si BDL ne dau un total de 13 biti) punem urmatorii 5 biti (cei mai semnificativi ai SBR-ului) in campul SBR din registrul BDH si cei mai nesemnificativi 8 biti in regsitrul BDL. [5]

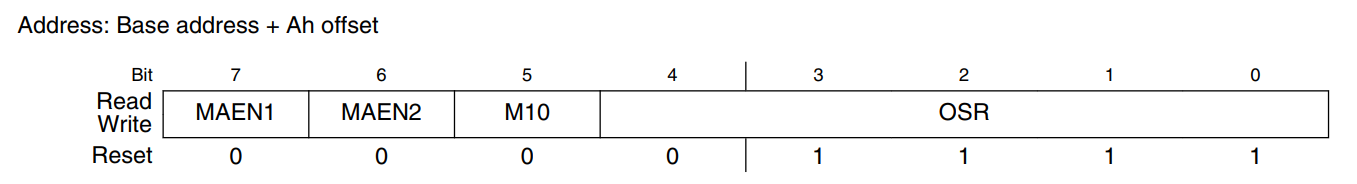


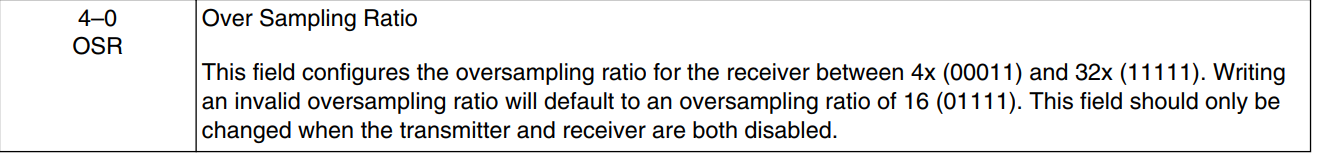




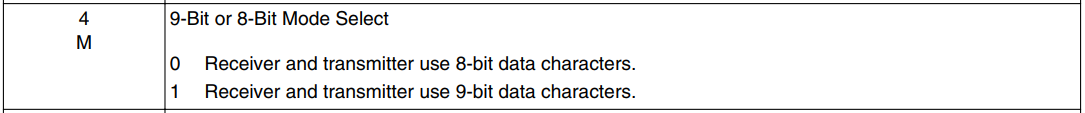
Astfel, valoarea SBR-ului nostru va fi egala cu 26 => 0x1A, deci vom pune in BDH (campul SBR) valoarea 0b00000 si in BDL vom pune valoarea 0x1A.

In registrul **UART0\_C4** se salveaza valoarea OSR, dar indexarea se face de la 0 si va fi OSR – 1 ( din cate se observa si din explicatia campului OSR valoarea 16 este reprezentata in binar: 0b1111).

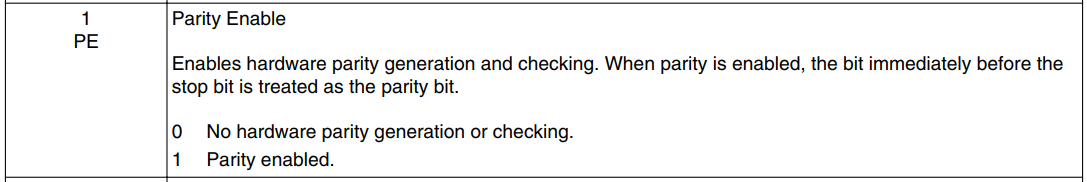




**UART0\_C1** este un registru de control in care putem configura mai multe proprietati. Implicit, acesta are toti bitii pusi pe 0, dar pentru o verificare suplimentara si pentru o mai buna intelegere, o sa setam conform Reference Manual bitii corespunzatori pentru configuratia noastra.



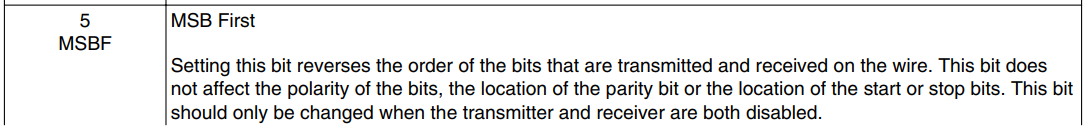
Selectam pentru campul ‚M’ valoarea 0, deoarece vrem sa transmitem un caracter cu lungimea de 8 biti.

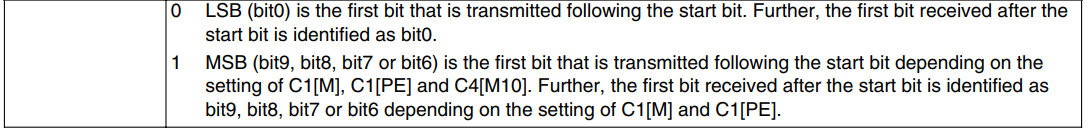


Nu vrem sa avem un bit de paritate, deci o sa selectam optiunea 0 pentru campul ‚PE’:

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

In registrul **S2 al UART0** putem configura si modul de transmisie a datelor, noi o sa alegem LSB (conform cerintei).

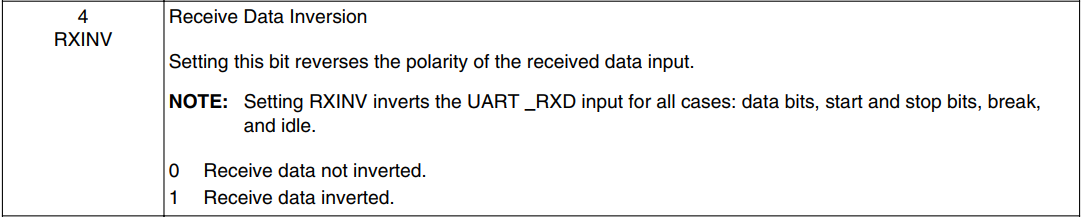




1. UART0->S2 |= UART0\_S2\_MSBF(0);

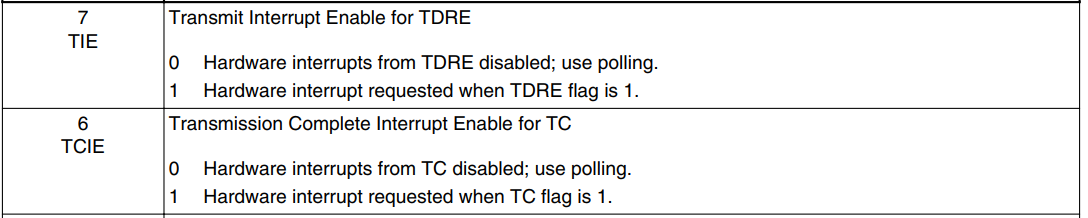
Pentru inversarea semnalului, avem campul **RXINV**, dar deoarece nu am gasit in biblioteca pyserial o modalitate de a lucra cu un semnal inversat, am comentat linia corespunzatoare acestei configuratii.

1. /\* UART0->S2 |= UART0\_S2\_RXINV(1); \*/



*\*Problema intampinata de noi in receptia datelor inversate, este legata de idle time cand se transmite continuu valoarea ‚1’ deoarece acest idle time nu poate fi configurat software din biblioteca folosita, singura varianta fiind una hardware (un cablu cu invertor de semnal).*

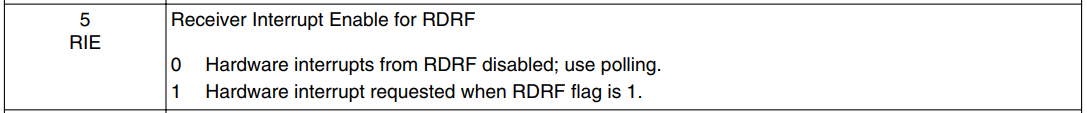
Deoarece datele pe care le transmitem pe interfata UART catre calculator vor fi transmise la un interval de timp configurat de noi, aceasta transmisie facandu-se prin pooling, o sa dezactivam intreruperea pentru transmisie asociata modului UART.



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

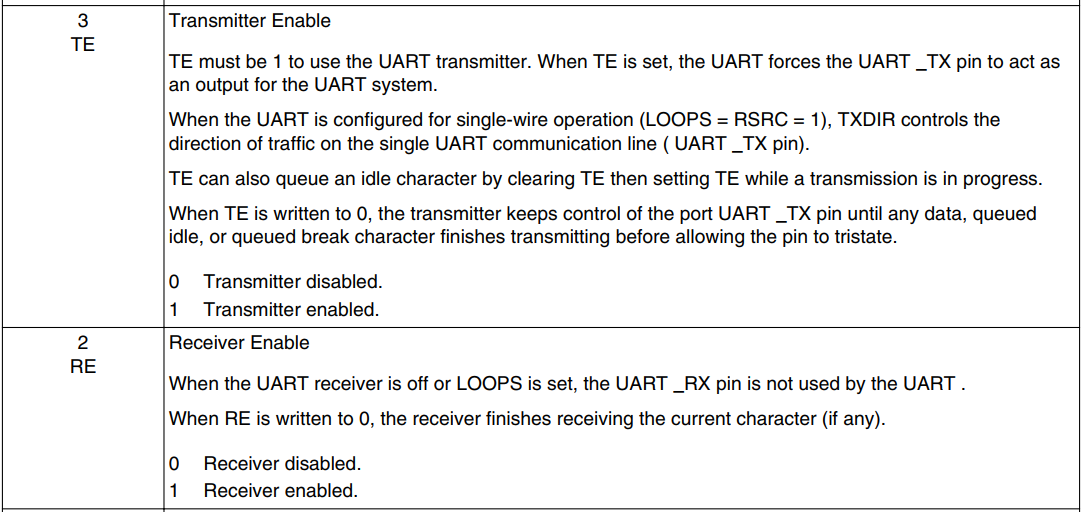
2. UART0->C2 |= UART0\_C2\_TCIE(0);

Insa, o sa folosim intreruperea pentru receptia caracterelor (comenzilor) de la calculator. Daca punem 1 in campul ‚RIE’ o sa primim o intrerupere cand se primesc noi date, registrul RDRF din UART0\_S1 o sa fie setat pe 1 in acest caz (la primirea datelor).



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

Acum ne mai ramane sa activam pinii pentru transmisie/receptie.



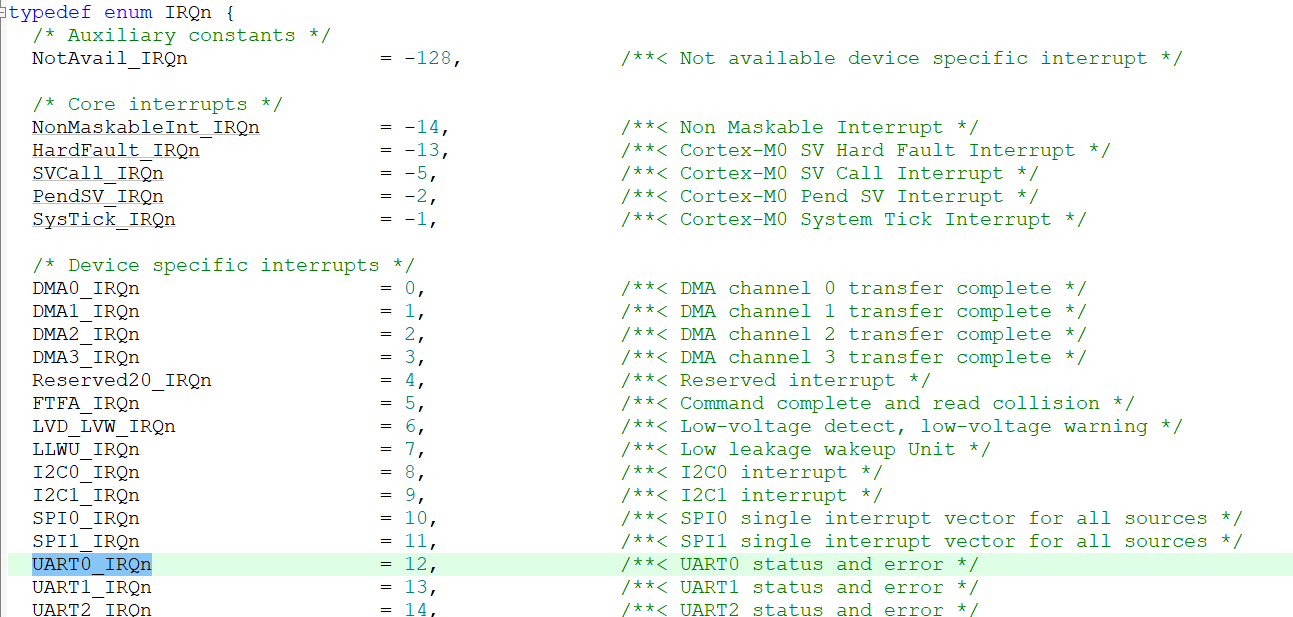
1. UART0->C2 |= ((UART\_C2\_RE\_MASK) | (UART\_C2\_TE\_MASK));

Pentru activarea din NVIC a intreruperii pentru UART0, ne putem uita in reference manual:





Aceasta intrerupere se poate gasii si in lista de intreruperi din header-ele asociate target-ului.



### Inițializarea modulului PIT



Folosit pentru a genera intreruperi la intervalul de timp specificat in cerinta pentru schimbarea culorii led-ului.

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 valorii numaratorului de pe canalul 0 la o perioada de 1677 milisecunde \*/

10. PIT->CHANNEL[0].LDVAL = 0x10C51EA;

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. /\* activarea intreruperii mascabile si setarea prioritatii \*/

18. NVIC\_ClearPendingIRQ(PIT\_IRQn);

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

20. NVIC\_EnableIRQ(PIT\_IRQn);

21.

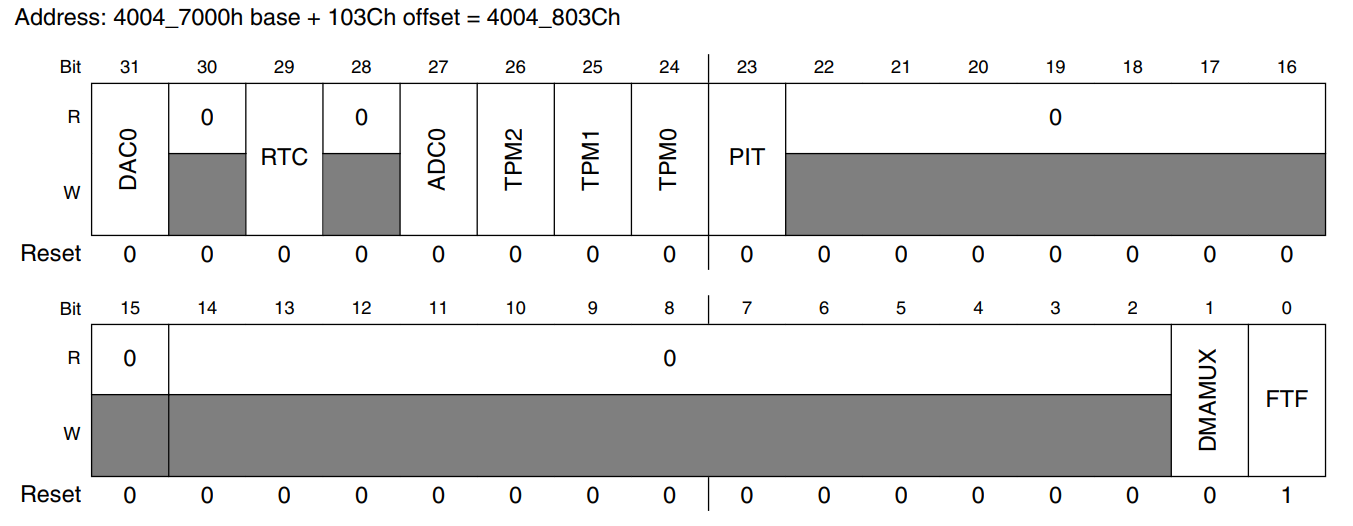
22. order = 0;

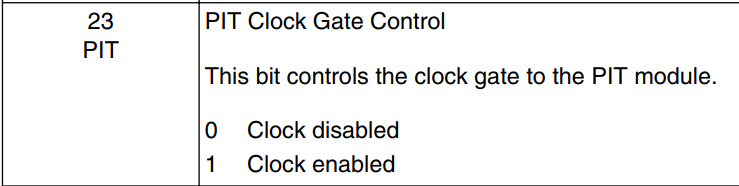
23. state = 0;

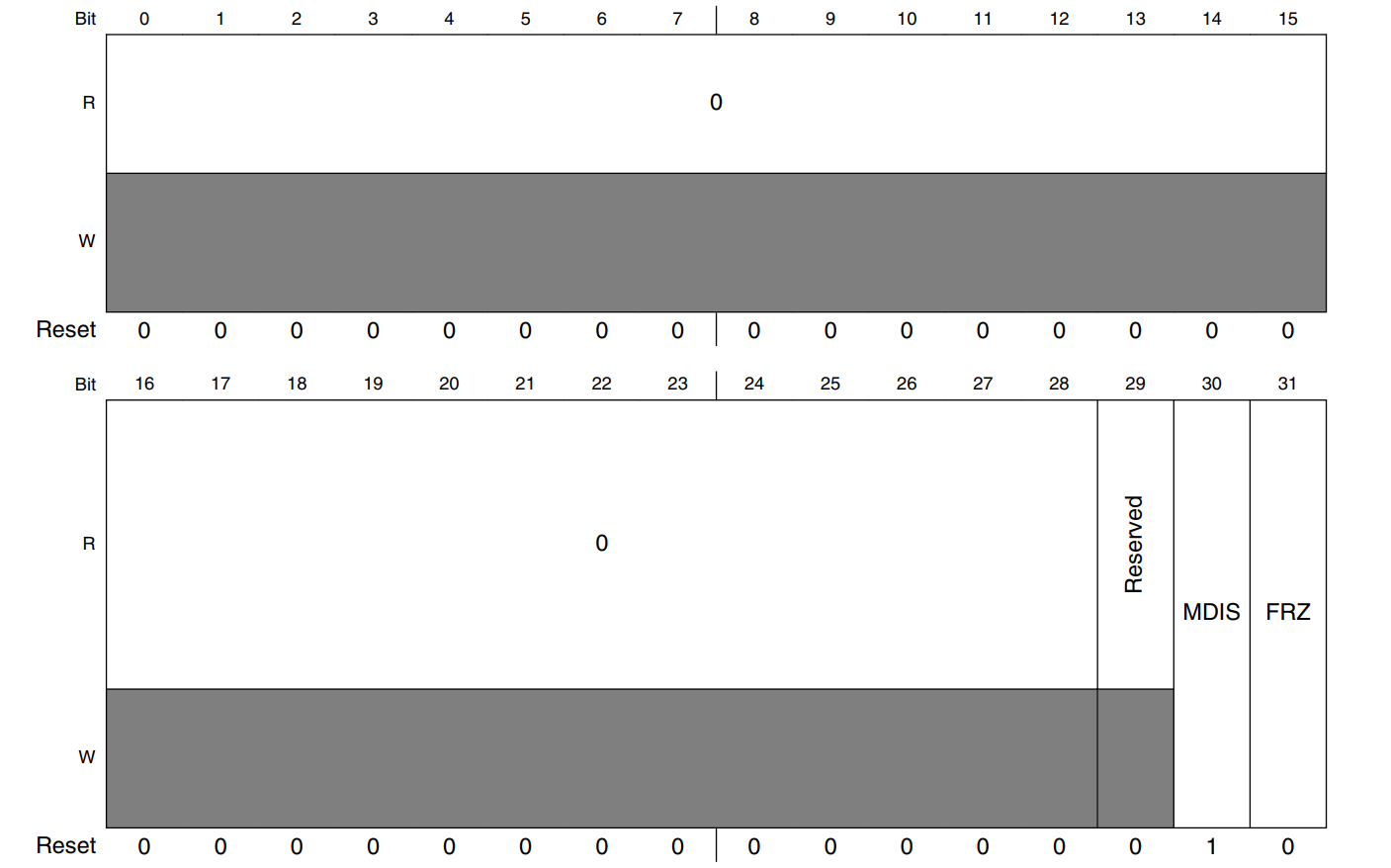
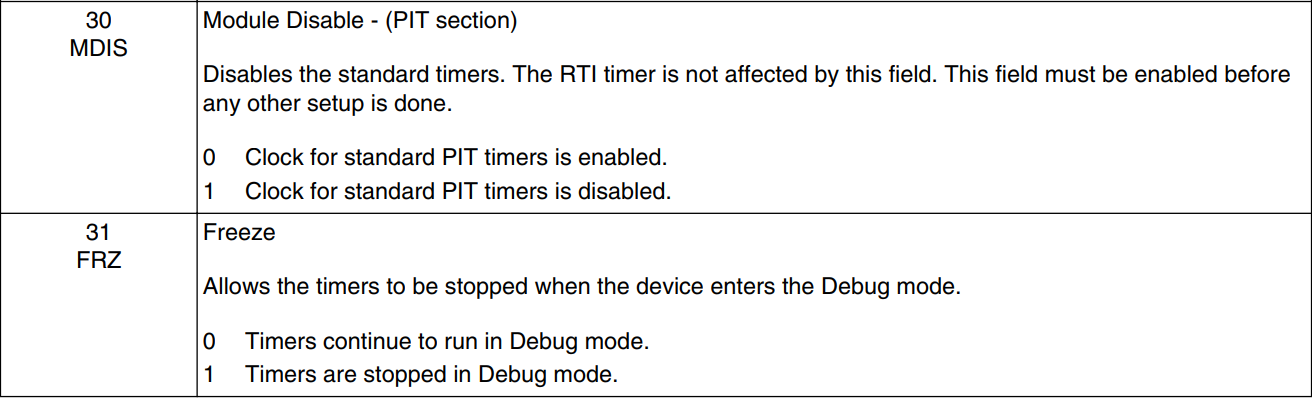
24. }

Modul de functionare al acestuia:

* Numaratorul va avea incarcata o valoare pe care o va decrementa pana va ajunge la valoare 0.
* Cand ajunge la valoarea 0 se va activa intreruperea si se va reseta numaratorul cu valoarea initiala regasita in PIT\_LDVAL.
* Verificarea daca numaratorul a ajuns la valoarea 0 se face prin registrul PIT\_CVAL, care reprezinta valoarea curenta a numaratorului. La resetare in acest registru se va incarca valoarea din PIT\_LDVAL.

În registrul **SIM\_SCGC6** (System Clock Gating Control Register 6) setăm pe 1 câmpul PIT (bitul 23) pentru activarea ceasului pentru acest modul. [5]



Registrul **PIT\_MCR** => utilizarea semnalului de ceas pentru timerele puse la dispozitie de PIT (MDIS) si configurarea comportamentului timerelor la debug (FRZ). [5]

PIT dispune de doua canale folosite pentru timere: Channel 0 si Channel 1. Avem nevoie de un singur numarator pentru valoarea de 1677 ms = 1.677 s deci ne vom folosi doar de canalul 0 al timerului PIT. Vom seta valoarea numaratorului de pe acest canal cu ajutorul sintaxei:

1. PIT->CHANNEL[0].LDVAL = 0x10C51EA;

**PIT\_LDVAL** - configurarea valorii numărătorului se face pe baza formulei

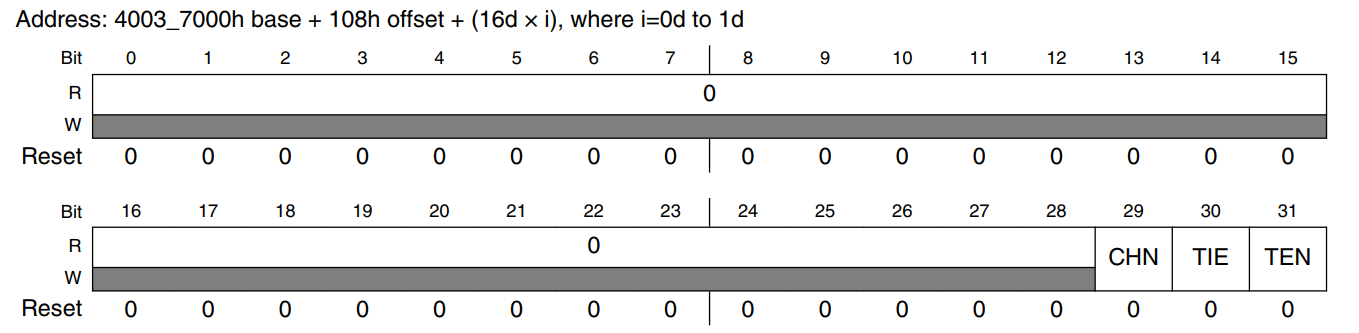
Load Value = Nr. sec. \* BUS CLOCK Freq. – 1

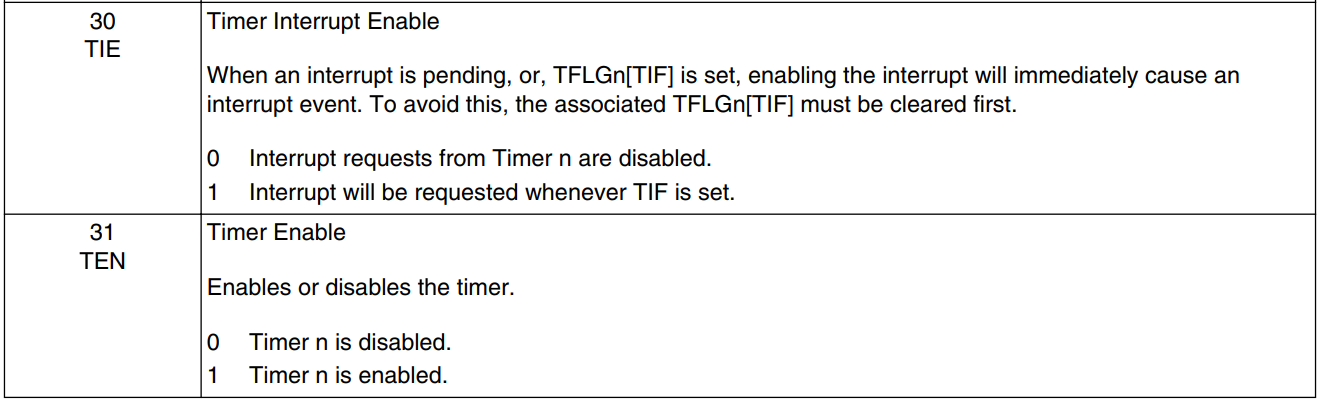


***Calcularea lui LDVAL:***

Val = 1.677 \* 10 485 760 – 1 = 17 584 619.52 – 1 = 17 584 618.52 ~ 17 584 618= **10C 51EA**

Registrul **PIT\_TCTRL** pentru canalul 0 contine bitii de control pentru fiecare timer. [5]





Prin TIE activam inintreruperile, iar prin TEN activam timerul de pe canalul 0.

Activam intreruperile cu ajutorul NVIC si ii setam o prioritate de 5 ceea ce inseamna ca este mai putin prioritar decat altele (intreruperile au default prioritatea 0, cu cat crestem prioritatea cu atat este mai putin prioritar). Facem acest lucru pentru ca atunci cand vor exista doua intreruperi, de exemplu: una de la PIT si alta de la UART sa se execute mai intai cea de la UART pentru ca ne intereseaza mai mult.

### Inițializarea modulului SysTick

O buna practica pentru citirea datelor de la un senzor analogic este sa o faci la un interval de timp, deoarece conversia datelor poate dura (in functie de ceea ce ne dorim single sau differential). Din acest considerent, am ales sa folosim SysTick Timer care ne poate oferi o intrerupere la un interval de timp (asemanator PIT). Acest SysTick Timer este gandit pentru a oferi un semnal periodic kernel-ului.

1. void Init\_SysTick(void)

2. {

3.     /\* setarea valorii de inceput/reset pentru numarator \*/

4.     SysTick->LOAD = (4800000L); /\* 100ms \*/

5.     /\* activarea intreruperii si setarea prioritatii in NVIC \*/

6.     NVIC\_SetPriority(SysTick\_IRQn, 2);

7.     /\* setarea valorii curente a numaratorului (reset) \*/

8.     SysTick->VAL = 0;

9.     /\* modificam registrul de control \*/

10.     SysTick->CTRL = SysTick\_CTRL\_CLKSOURCE\_Msk /\* selectam clock-ul procesorului \*/

11.                     | SysTick\_CTRL\_TICKINT\_Msk /\* activam intreruperea \*/

12.                     | SysTick\_CTRL\_ENABLE\_Msk; /\* activam numaratorul \*/

13.     /\* CLOKSOURCE = 0 (Sursa de ceas alternativa) \*/

14.

15.     flagSysTick = 0;

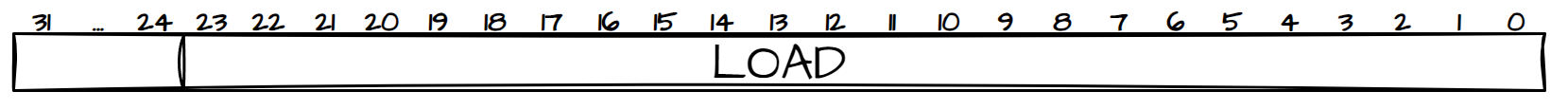
16. }

Numaratorul are o lungime de 24 de biti si este decrementat la fiecare semnal (tick).

Cand se ajunge la 0, se poate genera o exceptie SysTick, daca este activat.

**Registre:** [6]

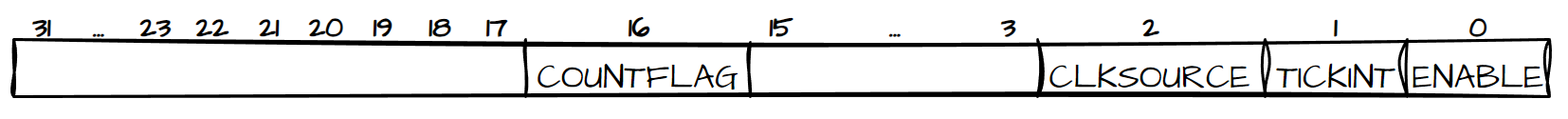
* LOAD: registru de unde se incarca valoarea initiala sau valoarea la reset



* VAL: valoarea curenta a numaratorului



* CTRL: registru de control



* CALIB: registru pentru calibrare (necesar doar daca avem selectata o sursa de ceas externa)

Pentru sursa de ceas (CLKSOURCE) avem 2 variante de selectie:

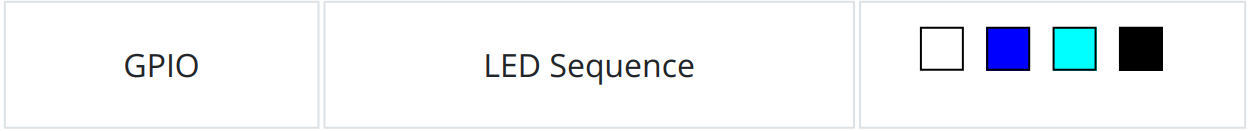
* O sursa de ceas externa (valoarea 0): ceasul procesorului impartit la 16
* Ceasul procesorului (valoarea 1)

 [6]

Dorim sa facem o intrerupere de 10 ori pe secunda, deoarece frecventa procesorului este de 48MHz (ceasul procesorului ticaie de 48 de milioane de ori pe secunde) inseamna ca trebuie sa impartim 48MHz la 10Hz -> 4.8MHz = 4.800.000 (decimal).

Vom scrie in registrul LOAD valoarea 4.800.000 (fiind pe 24 de biti 2^24-1 = 16.777.215 deci incape).

### Inițializarea modulului GPIO



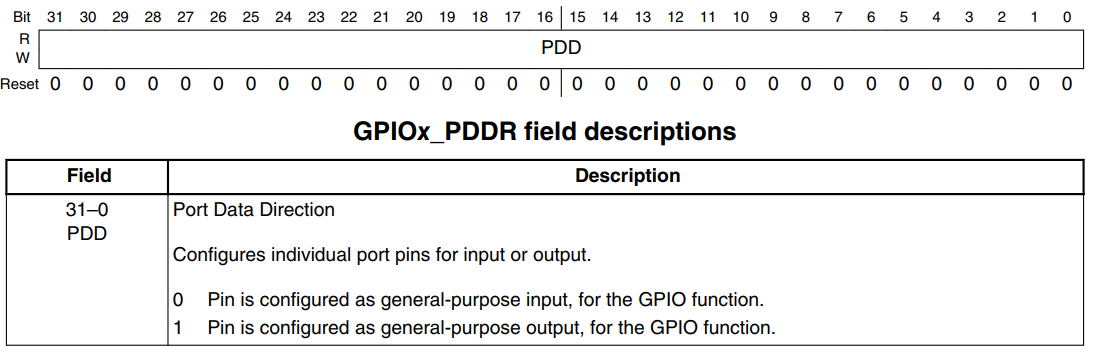
Conform reference manual [5], GPIO ne ofera o serie de facilitate, precum:

* pin pentru input
* pin pentru output cu posibilitatea de set/clear/toggle
* setarea directiei

Modulul GPIO foloseste ceasul sistemului, fiind nevoie de activarea acestuia pentru functionare.

Orice pin asociat unui port poate fi folosit pentru GPIO, cat timp acesta este expus si folosirea acestuia nu se suprapune cu o alta functionalitate.

Pentru activarea unui pin pentru input sau output, trebuie sa setam campul corespunzator din GPIOx\_PDDR:



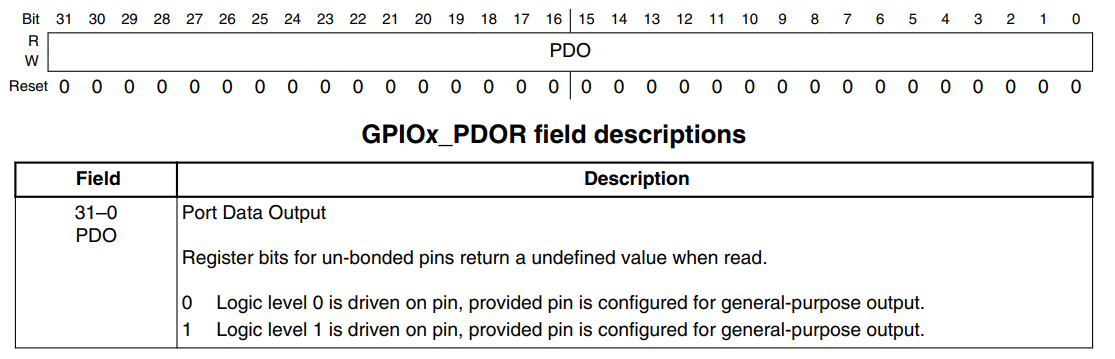
Dupa cum putem vedea, pentru input trebuie sa punem valoarea 0 in campul afferent, iar pentru output sa punem valoarea 1. Valoarea lui ‚x’ o sa fie inlocuita cu litera port-ului pe care se afla pinul (ex. pentru PORT D PIN 20, o sa avem GPIOD\_PDDR[20] setat).

Pentru pinul ales, trebuie sa setam si alternativa dorita conform cu tabelul Pinouts [5] incat sa selectam PTxy (ex. PTA4).





Pentru transmiterea de date prin interfata GPIO, atunci cand un pin este configurat pentru output se foloseste campul GPIOx\_PDOR. Acesta se poate seta prin intermediul celor 3 registre PCOR/PSOR/PTOR (urmeaza si un exemplu practic).



Campurile asociate pinilor configurati vor fi activati, celelalte campuri sunt implicit dezactivate pentru a nu consuma curent.

Pentru led-urile integrate pe placa avem urmatoarea initializare:

1. void RGBLed\_Init(void)

2. {

3. /\* activarea sursei de ceas pentru portul B \*/

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

5. /\* activarea sursei de ceas pentru portul D \*/

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

7.

8. /\* resetam valoarea pentru alternativa selectata pentru pini \*/

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

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

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

12.

13. /\* selectarea alternativei pentru pinii folositi pentru led-uri \*/

14. PORTB->PCR[RED\_LED\_PIN] |= PORT\_PCR\_MUX(1); /\* PTB18 \*/

15. PORTB->PCR[GREEN\_LED\_PIN] |= PORT\_PCR\_MUX(1); /\* PTB19 \*/

16. PORTD->PCR[BLUE\_LED\_PIN] |= PORT\_PCR\_MUX(1); /\* PTD1 \*/

17.

18. /\* setam pinii ca fiind pentru output \*/

19. GPIOB->PDDR |= (1 << RED\_LED\_PIN);

20. GPIOB->PDDR |= (1 << GREEN\_LED\_PIN);

21. GPIOD->PDDR |= (1 << BLUE\_LED\_PIN);

22.

23. /\* resetam valoarea transmisa pe pini (punem valoarea 0) pentru a stinge led-urile \*/

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

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

26. GPIOB->PCOR |= (1 << GREEN\_LED\_PIN);

27. }

Pentru initializarea senzorului de lumina:

1. void Light\_Init(void)

2. {

3. /\* activam sursa de ceas (chiar daca este deja activa, poate folosim separat de restul acest modul)\*/

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

5.

6. /\* selectarea alternativei pentru pinul pe care am conectat senzorul de lumina \*/

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

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

9.

10. /\* Setam pentru input pinu-ul desi el default este pe input dar sa fim siguri \*/

11. GPIOA->PDDR |= (0 << LIGHT\_PIN);

12. }

Pentru initializarea buzzer-ului:

1. void Buzzer\_Init(void)

2. {

3. /\* activarea semnalului de ceas pentru portul D (conexiune pe PTD4) \*/

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

5.

6. /\* utilizare GPIO ca varianta de multiplexare \*/

7. PORTD->PCR[BUZZER\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

8. PORTD->PCR[BUZZER\_PIN] |= PORT\_PCR\_MUX(1); /\* GPIO \*/

9.

10. /\* configurare pin pe post de output \*/

11. GPIOD\_PDDR |= (1 << BUZZER\_PIN);

12.

13. /\* stingerea buzzer-ului / punerea pe 0 logic \*/

14. GPIOD\_PCOR |= (1 << BUZZER\_PIN);

15.

16. play = 0;

17. }

Pentru a citi date de la un senzor prin interfata GPIO, putem citi direct din registrul PDIR, de pe pozitia asociata pinului (numarul pinului in cadrul portului).

Ex: citirea datelor de la senzorul digital de lumina

1. uint8\_t Light\_Read(void)

2. {

3. /\* activ pe 0 \*/

4. if (GPIOA->PDIR & (1 << LIGHT\_PIN))

5. {

6. return 1;

7. }

8. else

9. {

10. return 0;

11. }

12. }

Verificam daca in PDIR pe pozitia asociata pinului este pusa valoarea 1, returnam 1, altfel returnam 0.

Pentru scrierea datelor avem 3 registre, ca exemplu o sa discutam functia de playTone a buzzer-ului digital:

1. static void playTone(int tone, int duration)

2. {

3. long i;

4. for (i = 0; i < duration \* 1000; i += tone \* 2)

5. {

6. GPIOD\_PSOR |= (1 << BUZZER\_PIN);

7. delay(tone / 4);

8. /\* delay(tone / 2); \*/

9. GPIOD\_PCOR |= (1 << BUZZER\_PIN);

10. delay(tone \* 2.5);

11. /\* delay(tone \* 2); \*/

12. }

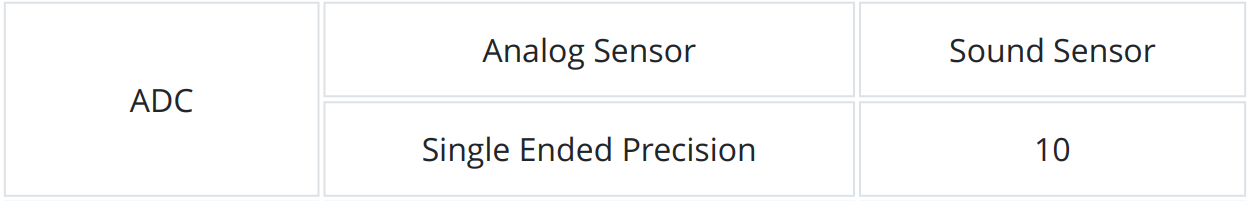
13. GPIOD\_PCOR |= (1 << BUZZER\_PIN);

14. }

Dupa cum vedem, la linia 6 se foloseste GPIOD\_PSOR |= (1 << BUZZER\_PIN); pentru a transmite o valoare de 1 catre buzzer, iar in continuare la linia 9 avem functia care schimba campul din registru PCOR astfel incat sa se transmita valoarea 0: GPIOD\_PCOR |= (1 << BUZZER\_PIN);

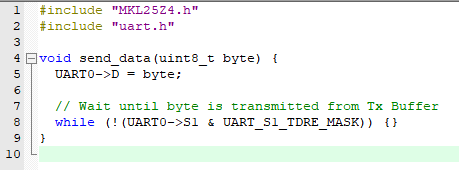
Al treilea registru ce se poate folosi pentru transmitere, este GPIOx\_PTOR care face o schimbare a valorii existente din 0 in 1 si invers (putem spune ca inversam valoarea transmisa).

### Inițializarea modulului ADC

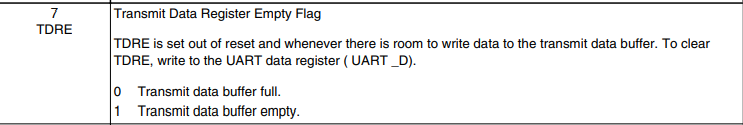


## Transmitere date prin UART

O dată la 10 ms, când flagul corespunzător devine, este apelată funcția send\_data ce primește ca parametru valoarea primită de la senzor.



Valoarea octetului dat ca parametru este pus în registrul UART0\_D (UART Data Register). Nu se părăsește funcția până când câmpul TDRE (Transmit Data Register Empty Flag) din registrul UART0\_S1 (UART Status Register 1) este 1, adică bufferul de trimitere s-a golit.



# Rezultate

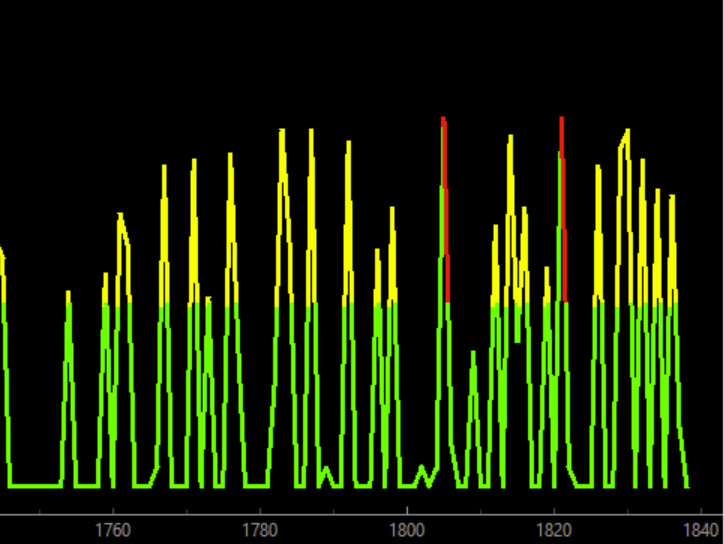


Figura . Exemplu grafic

# Dificultăți întâmpinate

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

* Intelegerea modului de functionare a buzzer-ului digital, pentru asta am accesat un videoclip de pe youtube (<https://www.youtube.com/watch?v=77h1JhD9Syw>) si am citit un articol (<https://www.cuidevices.com/blog/buzzer-basics-technologies-tones-and-driving-circuits>).
* Distribuirea operatiilor pe thread-uri pentru afisarea datelor provenite de la senzor, in python si colorarea segmentelor graficului in diferite culori pentru anumite intervale de valori.

# Referinte

|  |  |
| --- | --- |
| [1] | DFROBOT, "WIKI DFROBOT34," [Online]. Available: https://wiki.dfrobot.com/Analog\_Sound\_Sensor\_SKU\_\_DFR0034. |
| [2] | DFROBOT, "WIKI DFROBOT32," [Online]. Available: https://wiki.dfrobot.com/Digital\_Buzzer\_Module\_\_SKU\_\_DFR0032\_. |
| [3] | CLESTE, "Modul senzor lumina," [Online]. Available: https://cleste.ro/modul-senzor-lumina-intensitate-luminoasa.html. |
| [4] | NXP, "arm MBED," [Online]. |
| [5] | NXP, "KL25 Sub-Family Reference Manual," [Online]. Available: https://spivey.oriel.ox.ac.uk/dswiki/images-digisys/5/56/KL25-refman.pdf. |
| [6] | [Online]. |