**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**Facultatea de Sisteme Informatice și Securitate Cibernetică**

**Departamentul de Calculatoare și Securitate Cibernetică**



***Utilizare senzor Analog de flacără***

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

Std. sg. maj. Tudor MIRON-ONCIUL

Std. sg. maj. Toraș-Mihnea JIPIANU

Grupa C114A și Grupa C114B

**București**

**2024**

Curpins

[1. Prezentarea Senzorului DFR0076 3](#_Toc155992668)

[1.1 Specificații 3](#_Toc155992669)

[1.2 Aplicații 3](#_Toc155992670)

[1.3 Conectare 4](#_Toc155992671)

[2. Scop Proiect 4](#_Toc155992672)

[3. Arhitectură 5](#_Toc155992673)

[3.1 Conectare Senzor 6](#_Toc155992674)

[3.2 Conectare Plăcuță cu Breadboard 7](#_Toc155992675)

[4. Descriere Program 8](#_Toc155992676)

[4.1 Fișierul main.c 8](#_Toc155992677)

[4.2 UART 10](#_Toc155992678)

[4.2.1 Inițializare 10](#_Toc155992679)

[4.3 Modulul GPIO 14](#_Toc155992680)

[4.3.1 Inițializare 14](#_Toc155992681)

[4.3.2 Leduri Plăcuță 20](#_Toc155992682)

[4.3.3 Leduri Breadboard 22](#_Toc155992683)

[4.4 Modulul ADC 24](#_Toc155992684)

[4.4.1 Inițializare 24](#_Toc155992685)

[4.4.2 Descrierea Flow-ului 28](#_Toc155992686)

[4.5 Modulul PIT 29](#_Toc155992687)

[4.5.1 Inițializare 29](#_Toc155992688)

[5. Rezultate Python 33](#_Toc155992689)

[6. Demo 33](#_Toc155992690)

[7. Dificultăți întâmpinate 33](#_Toc155992691)

[8. Recomandări 34](#_Toc155992692)

[9. Referințe 34](#_Toc155992693)

# Prezentarea Senzorului DFR0076

Senzorul de flacără poate fi folosit pentru a detecta focul sau alte lungimi de undă cuprinse între 760 nm și 1100 nm.



Figure 1. Senzorul DFR0076

Unghiul de detectare al senzorului de flacără este de 60 de grade, având o sensibilitate specială la spectrul flăcării. Dispune de două găuri de montare M3 pentru a stabiliza modulul și a preveni rotirea acestuia.

Temperatura de funcționare a senzorului de flacără este cuprinsă între -25 de grade Celsius și 85 de grade Celsius. În timpul detecției flăcării, este important să se mențină o distanță corespunzătoare pentru a evita deteriorarea senzorului.

## Specificații

* Tensiune de alimentare: 3,3V până la 5V
* Distanța de detectare: 20cm (4,8V) până la 100cm (1V)
* Lungimi de undă: 760nm to 1100nm
* Timp de răspuns: 15 microsecunde
* Interfață: Analogică
* Dimensiuni: 22x30mm

## Aplicații

* Alarmă incendiu
* Robot specializat în detectarea focului

## Conectare

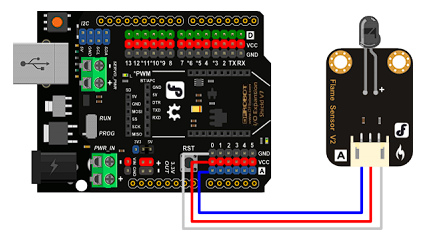


Figure -Exemplu de conectare senzor

Definire cabluri:

* Cablu albastru – analog
* Cablu Roșu – VCC
* Cablu Gri – Ground

# Scop Proiect

Scopul proiectului constă în iluminarea a 3 leduri dispuse pe o placă experimentală (breadboard), în funcție de valorile furnizate de senzorul de flacără. Valorile detectate de senzor vor fi categorisite în 3 intervale distincte. În funcție de intervalul în care se încadrează valoarea detectată de senzor, se vor activa 1, 2 sau 3 leduri.

De asemenea, prin intermediul interfeței UART, valorile detectate de senzorul de flacără vor fi transmise către o aplicație dezvoltată în Python. Această aplicație are rolul de a citi o dată la 1 secundă valoarea transmisă de senzor și de a afișa un grafic care reflectă aceste valori. Graficul va prezenta ultimele 25 de valori, actualizându-se la fiecare jumatate secundă cu noua valoare recepționată. Fiecare bară din barchart va fi colorată în funcție de intervalul în care se încadrează valoarea respectivă, respectiv:

* 0% - 33% - verde
* 33% - 66% - galben
* 66% - 100% - roșu

Pe placa de dezvoltare, ledurile vor fi activate urmărind secvența: alb, verde, galben, stins. Interfața dezvoltată în Python va conține un buton care va permite inversarea acestei secvențe.

# Arhitectură

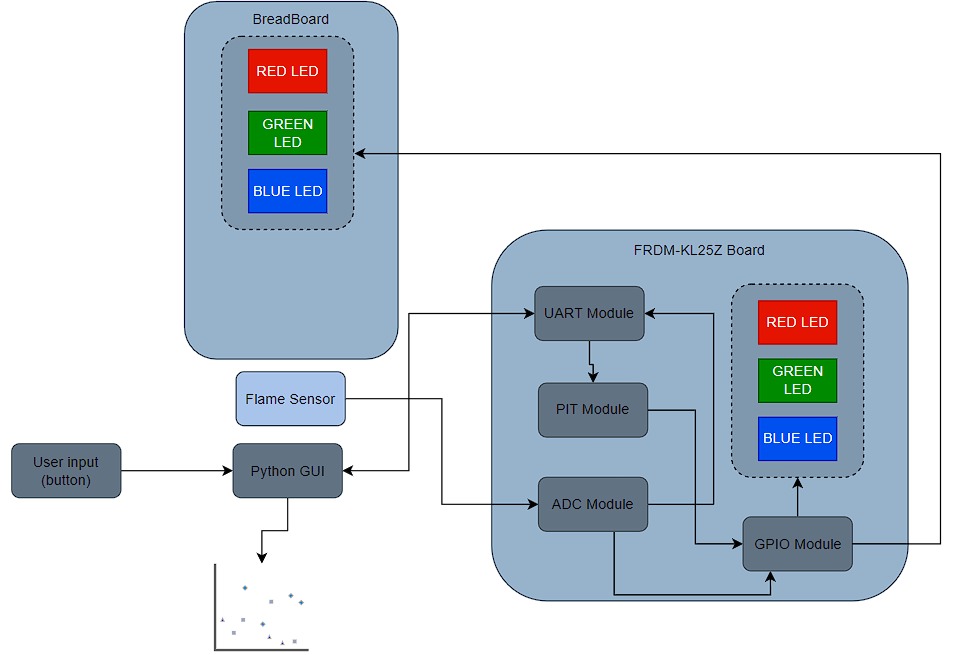


Figure 3

## Conectare Senzor

Senzorul este conectat la placuță astfel:

* Firul albastru – gri conectează senzorul la PTC2
* Firul roșu – roșu conectează senzorul la 5V
* Firul negru – negru conectează senzorul la GND



Figure - Conectare senzor

## Conectare Plăcuță cu Breadboard

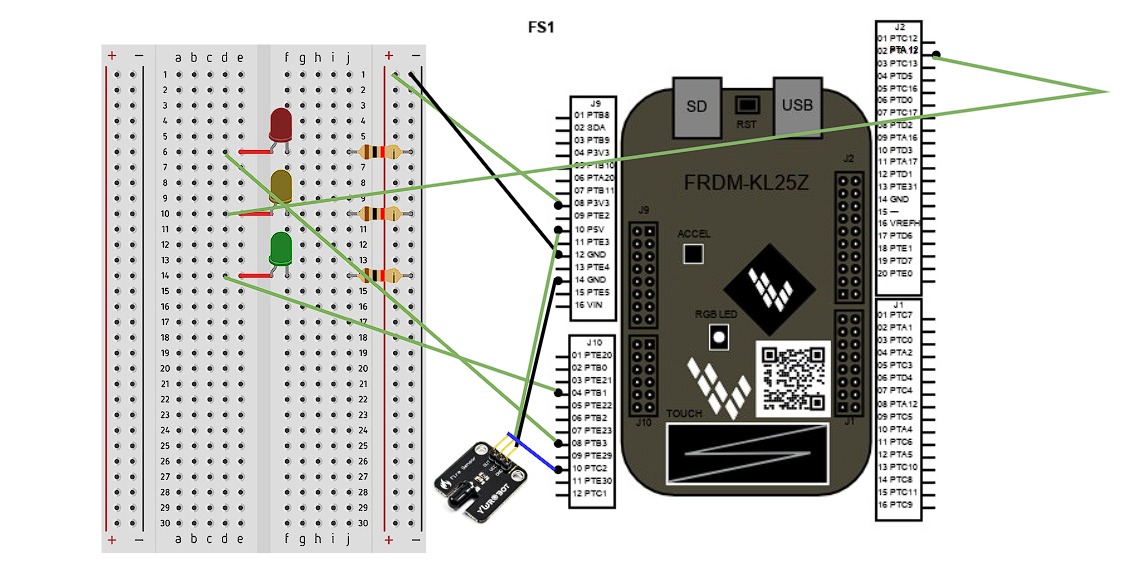


Figure 5

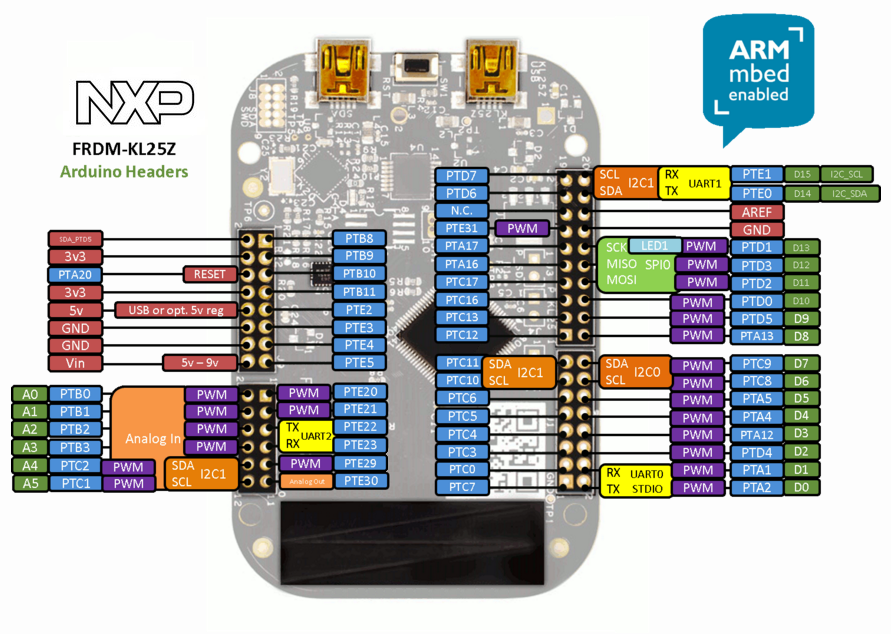


Figure -Pini placă de dezvoltare

# Descriere Program

## Fișierul main.c

1. #include "Gpio.h"

2. #include "Uart.h"

3. #include "Adc.h"

4. #include "Pit.h"

5.

6. uint8\_t flag = 0;

7.

8. void changeLedSeq(void)

9. {

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

11. {

12. char c = UART0->D;

13. if(flag == 0)

14. flag = 1;

15. else

16. flag = 0;

17. ledDirection = flag;

18. }

19. else

20. return;

21. }

22.

23. int main(void)

24. {

25. float measured\_voltage;

26. uint8\_t parte\_zecimala;

27. uint8\_t parte\_fractionara1;

28.

29. UART0\_Init(38400);

30. RGBLed\_Init();

31. ADC0\_Init();

32. PIT\_Init();

33.

34. for(;;)

35. {

36. changeLedSeq();

37.

38. measured\_voltage = (analog\_input \* 5.0f) / 65535;

39.

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

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

42.

43. ChangeColorFromFlame(parte\_zecimala+(parte\_fractionara1)/10);

44.

45. UART0\_Transmit('V');

46. UART0\_Transmit('a');

47. UART0\_Transmit('l');

48. UART0\_Transmit('u');

49. UART0\_Transmit('e');

50. UART0\_Transmit(' ');

51. UART0\_Transmit('=');

52. UART0\_Transmit(' ');

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

54. UART0\_Transmit('.');

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

56. UART0\_Transmit('V');

57. UART0\_Transmit(0x0A);

58. UART0\_Transmit(0x0D);

59.

60. readyToTransmit = 0;

61. }

62. }

63.

În fișierul „*main.c*” sunt incluse fișierele header în care sunt declarate funcțiile și variabilele folosite:

* „Gpio.h” conține funcțiile și variabilele pentru interacțiunea cu ledurile (atât de pe placă cât și de pe breadboard)
* „Uart.h” conține funcțiile pentru a interacșiona cu perifericul UART care este responsabil pentru trimiterea și primirea datelor din aplicația Python.
* „Adc.h” conține funcțiile și variabilele responsabile pentru interacțiunea cu senzorul de flacără
* „Pit.h” conține funcțiile pentru a interacționa cu cronometrul responsabil pentru sincronizarea ledurilor de pe plăcuță

Fluxul programului este următorul:

* În prima parte a programului sunt apelate funcțiile de inițializare ale prifericelor:

* + UART0\_Init(38400) – inițializare periferic UART (pentru transmitere și recepție date de la aplicația Python)
  + RGBLed\_Init() – inițializare perriferic GPIO (pentru becurile de pe plăcuță și breadboard)
  + ADC0\_Init() – inițializare convertor analogic-digital pentru citirea valorilor recepționate de senzorul de flacără
  + PIT\_Init() – inițializare periferic PIT, pentru setarea cronometrelului
* În a doua parte se inițializeaza o buclă infinită în care se vor aștepta întreruperi de la periferice:
  + Funcția *changeLedSeq()* verifică dacă perifericul UART a recepționat date de la aplicația Python pentru a schimba sensul secvenței de aprindere a becurilor LED de pe plăcuță
  + În continuare se preia valoarea pe 16 biți recepționată de senzor și se scaleaza la o valoare *float* cuprinsă între 0 si 6. Acesta este intervalul în care vom include valorile recepționate de senzor.
  + Funcția *ChangeColorFromFlame(parte\_zecimala+(parte\_fractionara1)/10)* este responsabilă pentru aprinderea becurilor de pe breadboard. Parametrul este valoarea în *float.*
  + În continuare se trimite, prin intermediul perifericului UART, valoarea măsurată însoțită de cuvântul *Value.*

## UART

### Inițializare

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

2. {

3. uint32\_t osr;

4. uint16\_t sbr;

5. uint8\_t temp;

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

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

8.

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

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

11.

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

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

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

15.

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

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

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

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

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

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

22.

23.

24.

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

26.

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

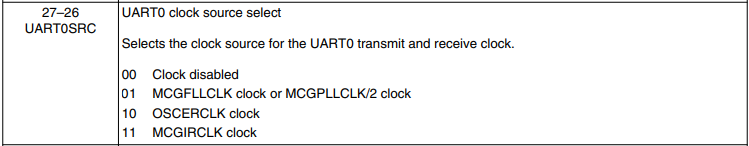


Figure 7

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

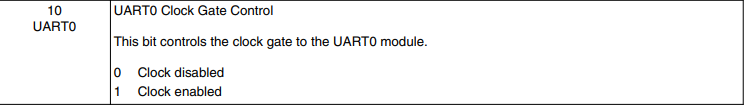


Figure 8

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

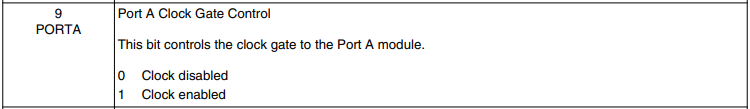


Figure 9

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



Figure 10

1. /\* Configurare Baud Rate\*/

2. osr = 15; /\* Over-Sampling Rate (numarul de esantioane luate per bit-time)\*/

3.

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

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

6. /\* 0x1F00 - 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0\*/

7. /\* 0 0 0 b13 b12 b11 b10 b09 0 0 0 0 0 0 0 0 >> 8\*/

8. /\* BDH - 0 0 0 b13 b12 b11 b10 b09\*/

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

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

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

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

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

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

15.

16.

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

18. UART0->C1 = 0;

19.

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

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

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

23.

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

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

26.

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

28.

29.

30.

31. }

32.

Configuram baud rate în funcție de osr=15.

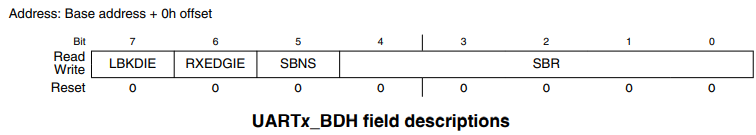


Figure 11

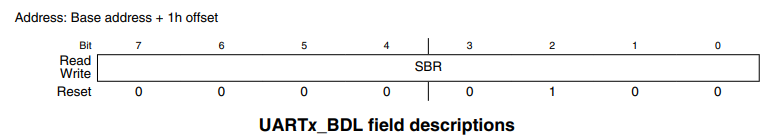


Figure 12

Formula după care se calculează baud rate este: DEFAULT\_SYSTEM\_CLOCK / ((OSR+1) × BR) (pentru SBR între 1 – 8191).

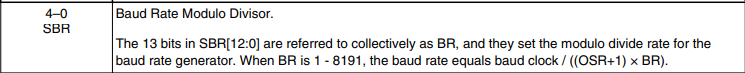


Figure 13

OSR (Over Sampling Ratio) are valoarea implicită 0b01111 adică 15, iar baud clock a fost setat la cel de 48 MHz.

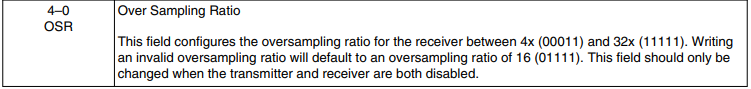


Figure 14

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

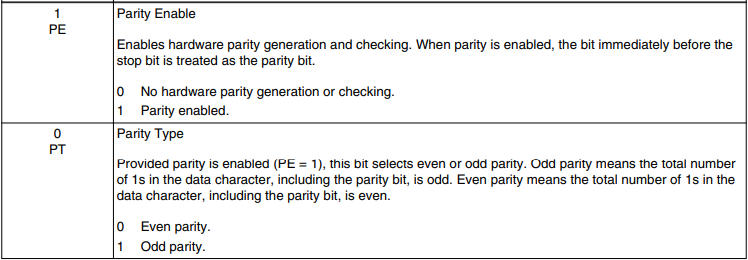


Figure 15

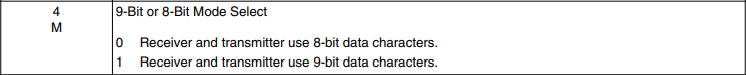


Figure 16

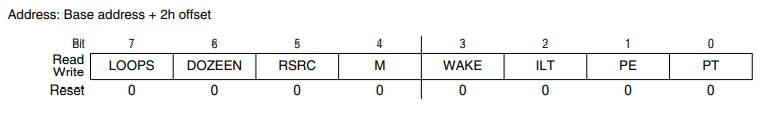


Figure 17

Registrul UARTx\_C1 ( UART Control Register 1 ) se setează pe 0.

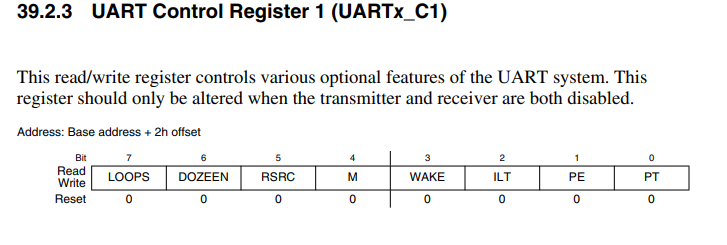


Figure 18

Registrul UARTx\_C2

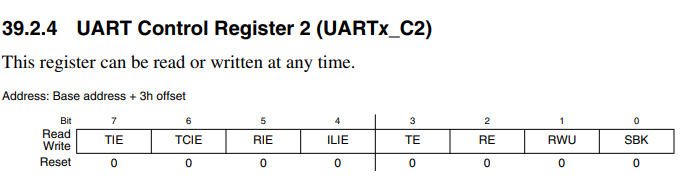


Figure 19

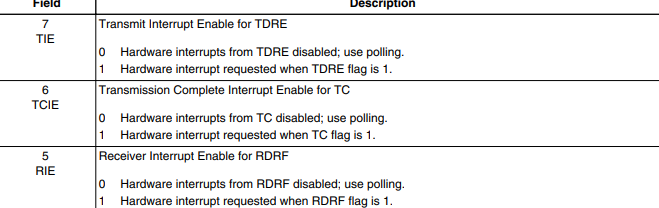


Figure 20

## Modulul GPIO

Modulul GPIO (General-Purpose Input/Output) asigură capacitatea de a personaliza funcționarea unui pin specific al unui circuit integrat în timpul execuției. Mai exact, este posibil să se regleze direcția fluxului de curent electric (intrare sau ieșire) și starea logică (nivelul de tensiune) asociată pinului.

Un pin al modulului GPIO poate fi configurat fie ca pin de intrare, pentru a recepționa semnale de la alte dispozitive, fie ca pin de ieșire, pentru a trimite semnale către alte componente.

Exemple de interfețare cu dispozitive externe includ:

* Pentru intrare: un comutator (switch), care permite detectarea schimbărilor de stare ale utilizatorului;
* Pentru ieșire: un LED, care poate indica stări sau evenimente prin aprindere/stingere sau prin modificarea intensității luminoase.

Următoarea secțiune va detalia cum este integrat acest modul în cadrul proiectului nostru.

### Inițializare

1. void RGBLed\_Init(void){

2.

3. /\* Activarea semnalului de ceas pentru pinii folositi in cadrul led-ului RGB \*/

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

5.

6. /\* --- RED LED --- \*/

7.

8. /\* Utilizare GPIO ca varianta de multiplexare \*/

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

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

11.

12. /\* Configurare pin pe post de output \*/

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

14.

15. /\* Stingerea LED-ului (punerea pe 0 logic) \*/

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

17.

18.

19. /\* --- GREEN LED --- \*/

20.

21. /\* Utilizare GPIO ca varianta de multiplexare \*/

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

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

24.

25. /\* Configurare pin pe post de output \*/

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

27.

28. /\* Stingerea LED-ului (punerea pe 0 logic) \*/

29. GPIOB\_PCOR |= (1<<GREEN\_LED\_PIN);

30.

31. /\* ----YELLOW LED ---- \*/

32.

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

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

35.

36. /\* Configurare pin pe post de output \*/

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

38.

39. /\* Stingerea LED-ului (punerea pe 0 logic) \*/

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

41.

42.

43. /\* LED-ul de pe placuta: \*/

44. /\* ----- RED ---- \*/

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

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

47.

48. /\* Configurare pin pe post de output \*/

49. GPIOB\_PDDR |= (1<<PLACA\_RED\_LED\_PIN);

50.

51. /\* Stingerea LED-ului (punerea pe 1 logic) \*/

52. GPIOB\_PSOR |= (1<<PLACA\_RED\_LED\_PIN);

53.

54. /\* ----- GREEN --- \*/

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

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

57.

58. /\* Configurare pin pe post de output \*/

59. GPIOB\_PDDR |= (1<<PLACA\_GREEN\_LED\_PIN);

60.

61. /\* Stingerea LED-ului (punerea pe 1 logic) \*/

62. GPIOB\_PSOR |= (1<<PLACA\_GREEN\_LED\_PIN);

63.

64. /\* ---- BLUE ---- \*/

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

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

67.

68. /\* Configurare pin pe post de output \*/

69. GPIOD\_PDDR |= (1<<PLACA\_BLUE\_LED\_PIN);

70.

71. /\* Stingerea LED-ului (punerea pe 1 logic) \*/

72. GPIOD\_PSOR |= (1<<PLACA\_BLUE\_LED\_PIN);

73. }

74.

În registrul SIM\_SCGC5 (System Clock Gatting Control Register) – setăm pe 1 bitul porturilor D, B si A folosind operația de “sau” logică cu măștile aferente (0x1000u , 0x400u, 0x200u).

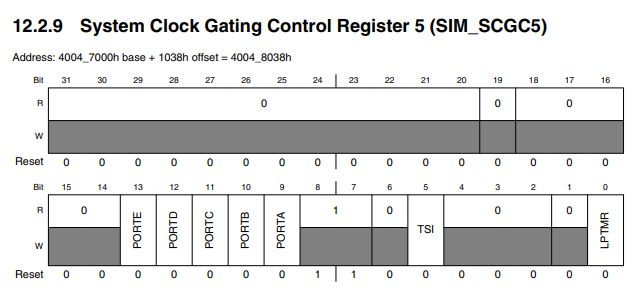


Figure 21

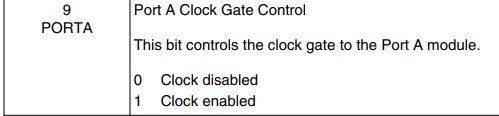


Figure 22



Figure 23

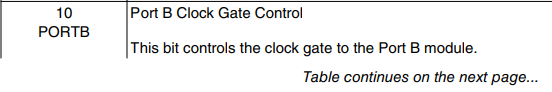


Figure 24



Figure 25



Figure 26

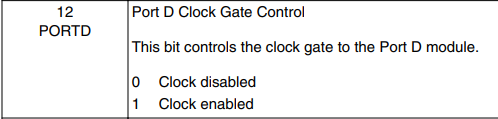
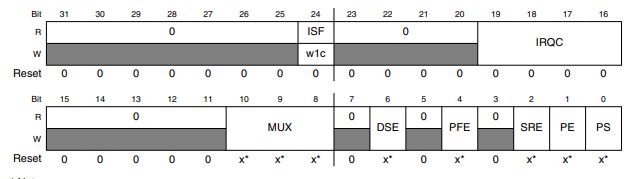


Figure 27

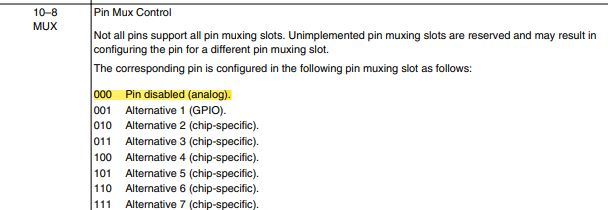


Figure 28

În regiștrii PORTx\_PCRn (Pin Control Register n) – pentru porturile B(pinii 1-GREEN\_LED\_PIN,3-RED\_LED\_PIN,18-PLACA\_RED\_LED\_PIN,19-PLACA\_GREEN\_LED\_PIN), D(pinul 1-PLACA\_BLUE\_LED\_PIN) și A(pinul 12- YELLOW\_LED\_PIN) – dezactivăm biții pentru MUX.



Figure



Figure



Figure

Setăm bitul pentru alternativa 1 în MUX pentru a folosi GPIO.



Figure

Registrul GPIOx\_PDDR (Port Data Direction Register) – este setat bitul x (asocial cu pinul x) cu valoarea 1 pentru output.

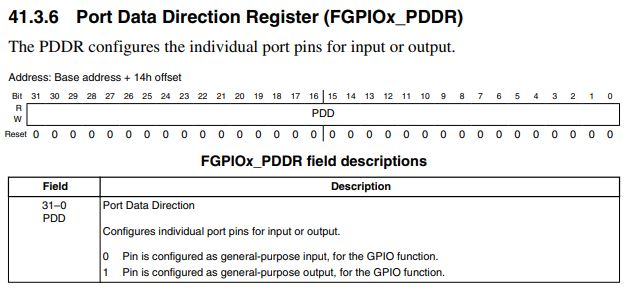


Figure 33

Registrul GPIOx\_PSOR (Port Set Output Register) – setează bitul x asociat cu pinul x pe 1 pentru a seta output-ul pinului.

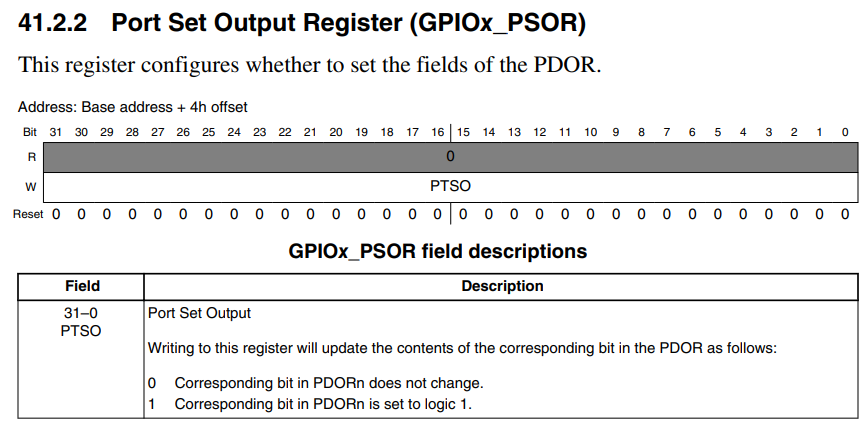


Figure 34

Registrul GPIOx\_PCOR (Port Clear Output Register) – setează bitul x asociat cu pinul x pe 0 pentru a da „clear” output-ului pinului.

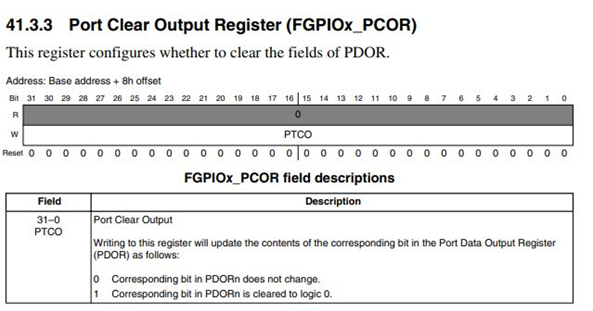


Figure 35

### Leduri Plăcuță

Pentru a controla culorile ledurilor de pe plăcuță, avem următoarele funcții:

1. void black(void)

2. {

3. GPIOB\_PSOR |= (1<<PLACA\_GREEN\_LED\_PIN);

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

5. GPIOD\_PSOR |= (1<<PLACA\_BLUE\_LED\_PIN);

6. }

7.

Funcția black() este responsabilă pentru stingerea celor 3 leduri, aceasta este apelată în rumătoare funcție:

1. void ChangePlacaColor(void)

2. {

3. if(state == 0)

4. {

5. black();

6. GPIOB\_PTOR |= (1<<PLACA\_GREEN\_LED\_PIN);

7. GPIOB\_PTOR |= (1<<PLACA\_RED\_LED\_PIN);

8. GPIOD\_PTOR |= (1<<PLACA\_BLUE\_LED\_PIN);

9.

10. if(ledDirection == 0)

11. state = 1;

12. else

13. state = 3;

14.

15. }

16. else if(state == 1)

17. {

18. black();

19. GPIOB\_PTOR |= (1<<PLACA\_GREEN\_LED\_PIN);

20. if(ledDirection == 0)

21. state = 2;

22. else

23.

24. state = 0;

25. }

26. else if (state == 2)

27. {

28. black();

29. GPIOB\_PTOR |= (1<<PLACA\_RED\_LED\_PIN);

30. GPIOB\_PTOR |= (1<<PLACA\_GREEN\_LED\_PIN);

31. if(ledDirection == 0)

32. state = 3;

33. else

34. state = 1;

35. }

36. else

37. {

38. black();

39. if(ledDirection == 0)

40. state = 0;

41. else

42. state = 2;

43. }

44. }

45.

Ledurile trebuie să se aprindă într-o anumită ordine, aceasta este: alb, verde, galben, negru. De asemenea, în cazul în care se primește un anumit input, această ordine este inversată: negru, galben, verde, alb.

Avem 4 stări în care ne putem afla, fiecare stare rerprezintă culoarea ledului. Înainte de a activa un anumit led, le stingem pe toate pentru a fi siguri că nu vom obține o altă culoare decât cea dorită.

În funcție de direcția, pe care utilizatorul o alege din interfață, se alege starea următoare.

Diagrama de stări este următoarea:

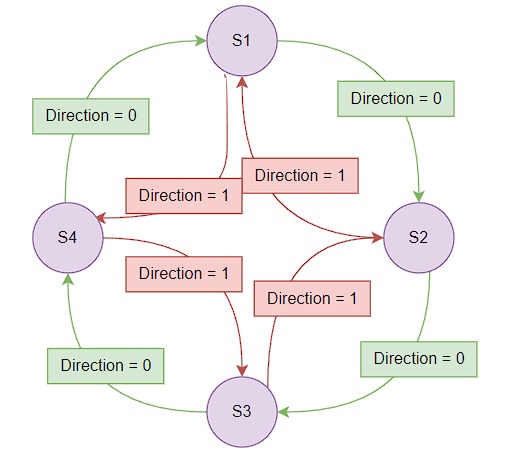


Figure 36

### Leduri Breadboard

Pentru a controla ledurile de pe plăcuță avem următoarea funcție:

1. void ChangeColorFromFlame(float measured\_voltage)

2. {

3. if(measured\_voltage < 1.66f)

4. {

5. if(flag\_green == 0)

6. {

7. GPIOB\_PTOR |= (1<<GREEN\_LED\_PIN);

8. flag\_green = 1;

9. }

10. if(flag\_red == 1)

11. {

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

13. }

14. if(flag\_yellow == 1)

15. {

16. GPIOA\_PTOR |= (1<<YELLOW\_LED\_PIN);

17. }

18. flag\_red = 0;

19. flag\_yellow = 0;

20.

21. }

22. else

23. if(measured\_voltage < 3.32f)

24. {

25. if(flag\_yellow == 0)

26. {

27. GPIOA\_PTOR |= (1<<YELLOW\_LED\_PIN);

28. flag\_yellow = 1;

29. }

30. if(flag\_red == 1)

31. {

32. GPIOB\_PTOR |= (1<<RED\_LED\_PIN);

33. }

34. if(flag\_green == 0)

35. {

36. GPIOB\_PTOR |= (1<<GREEN\_LED\_PIN);

37. }

38. flag\_red = 0;

39. flag\_green = 1;

40.

41. }

42. else

43. {

44. if(flag\_red == 0)

45. {

46. GPIOB\_PTOR |= (1<<RED\_LED\_PIN);

47. flag\_red = 1;

48. }

49. if(flag\_green == 0)

50. {

51. GPIOB\_PTOR |= (1<<GREEN\_LED\_PIN);

52. }

53. if(flag\_yellow == 0)

54. {

55. GPIOA\_PTOR |= (1<<YELLOW\_LED\_PIN);

56. }

57. flag\_green = 1;

58. flag\_yellow = 1;

59.

60. }

61. }

62.

Intervalul în care se poate afla valoarea măsurată de senzor este [0.0, 5.0]. Acest interval este la rândul său împărțit în 3 intervale:

* [0.0, 1.66) – ledul verde este aprins
* [1.66, 3.32) – ledurile verde și galben sunt aprinse
* [0.0, 5.0] – ledurile verde, galben si roșu sunt aprinse

Inițial toate ledurile sunt stinse. La fiecare recepționare a valorii, se verifică intervalul din care aceasta face parte. În funcție de intervalul în care se află valoarea, se aprind ledurile respective și se sting cele care nu sunt responsabile pentru acel interval. Există câte un flag pentru fiecare led:

* flag\_green
* flag\_yellow
* flag\_red

Dacă flag-ul este setat pe 1 atunci led-ul este aprins de la starea anterioară, dacă este nevoie îl stingem, dacă nu, atunci îl lasăm aprins.

Diagrama de stări este următoarea:

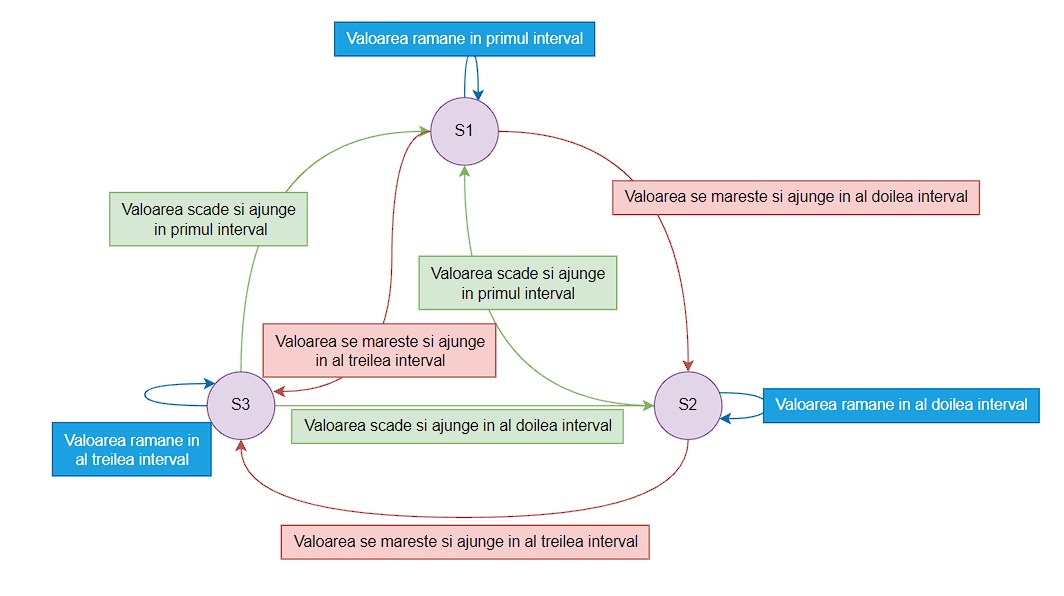
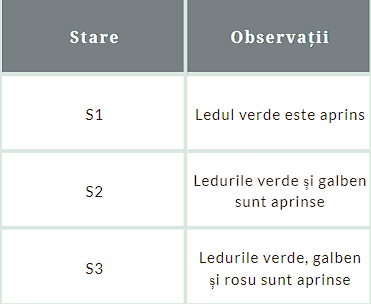


Figure 37



## Modulul ADC

### Inițializare

1. void ADC0\_Init(void) {

2.

3. /\* Activarea semnalului de ceas pentru modulul periferic ADC\*/

4. SIM->SCGC6 |= SIM\_SCGC6\_ADC0\_MASK;

5.

6. /\* Functia de calibrare\*/

7. ADC0\_Calibrate();

8.

9. ADC0->CFG1 = 0x00;

10.

11. /\* Selectarea modului de conversie pe 16 biti single-ended --> MODE\*/

12. /\* Selectarea sursei de ceas pentru generarea ceasului intern --> ADICLK\*/

13. /\* Selectarea ratei de divizare folosit de periferic pentru generarea ceasului intern --> ADIV\*/

14. /\* Set ADC clock frequency fADCK less than or equal to 4 MHz (PG. 494)\*/

15. ADC0->CFG1 |= ADC\_CFG1\_MODE(3) |

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(FLAME\_CHANNEL);

29. /\* Enables conversion complete interrupts\*/

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

31.

32. NVIC\_ClearPendingIRQ(ADC0\_IRQn);

33.

34. NVIC\_EnableIRQ(ADC0\_IRQn);

35. }

36.

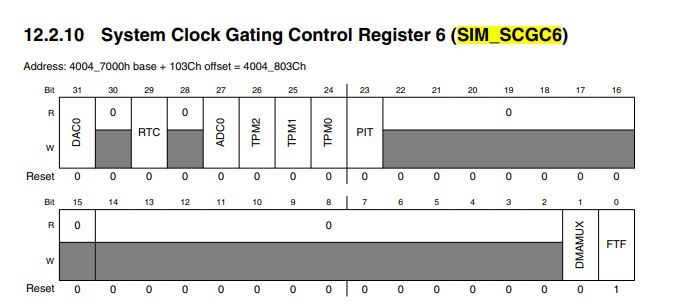


Figure 38

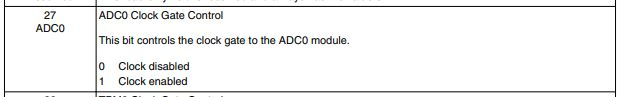


Figure 39

Inserting image...

Figure 40

Următorul pas este apelarea funcției de tip ADC0\_Calibrate().Registrul ADCx\_CFG1(Configuration Register 1) va avea toti biții setați pe 0. Apoi vom seta “MODE”, “ADIV” si “ADICLK” cu valoarea 1 pe bitul asociat.

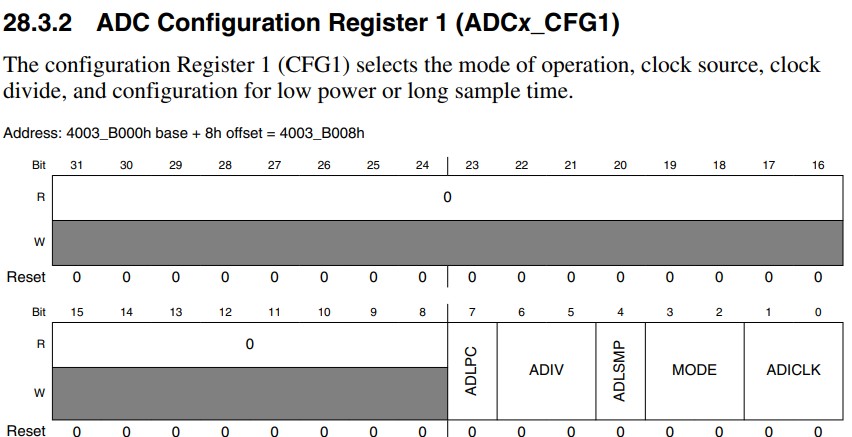


Figure 41



Figure 42

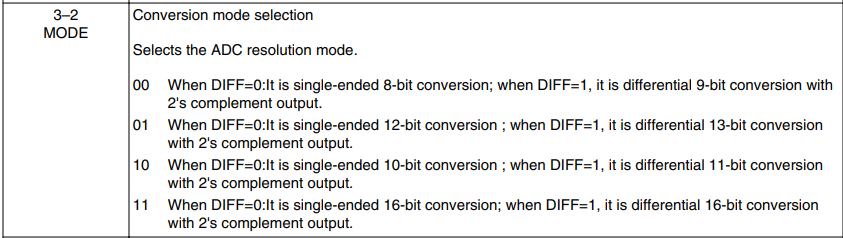


Figure 43



Figure 44

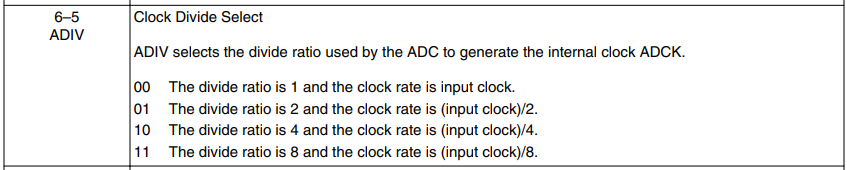


Figure 45



Figure 46

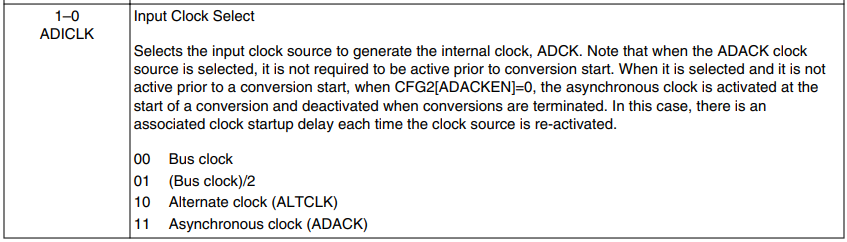


Figure 47



Figure 48

Registrul ADCx\_SC1n(Status and Control Register) va avea inițila valoarea 0.

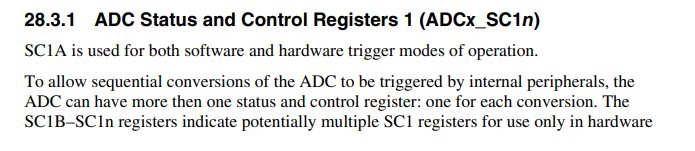


Figure 49

Ne folosim de modulul ADCH pentru a selecta canalul de comunicație pentru senzorul de flacără(este selectat canalul 11).



Figure 50

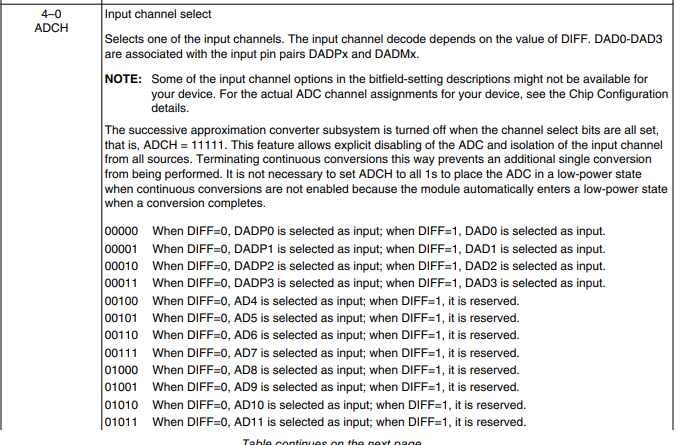


Figure 51

### Descrierea Flow-ului

Funcția de întrerupere ADC0\_IRQHandler este utilizată pentru captarea datelor emise de un senzor de flacără. Valorile detectate sunt înregistrate în variabila externă volatilă denumită analog\_input. În momentul în care datele de la senzor sunt colectate, variabila readyToTransmit este actualizată cu valoarea 1. Aceasta servește ca semnal pentru modulul UART că datele sunt gata și pot fi acum transmise.

1. void ADC0\_IRQHandler(void)

2. {

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

4. readyToTransmit = 1;

5. }

6.

1. volatile extern uint8\_t readyToTransmit;

2. volatile extern uint16\_t analog\_input;

3.

## Modulul PIT

### Inițializare

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 secunda\*/

10. PIT->CHANNEL[0].LDVAL = 0x3F3332; /\* in decimal: 4 141 874, pt 395 ms \*/

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 prioritatiis \*/

18.

19. NVIC\_ClearPendingIRQ(PIT\_IRQn);

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

21. NVIC\_EnableIRQ(PIT\_IRQn);

22. }

23.

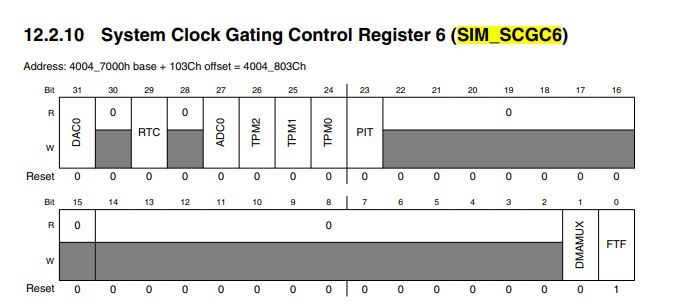


Figure 52

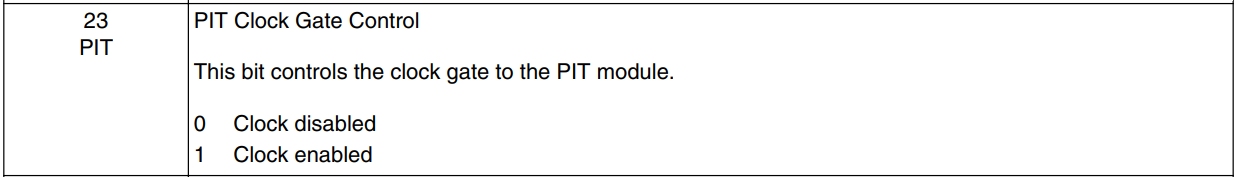


Figure 53



Figure 54

Registrul PIT\_MCR(PIT Module Control Register) - este folosit pentru activarea semnalelor de ceas pentru modulul PIT.

Bitul 1 din registrul PIT\_MCR(câmpul MDIS) este setat pe 0.

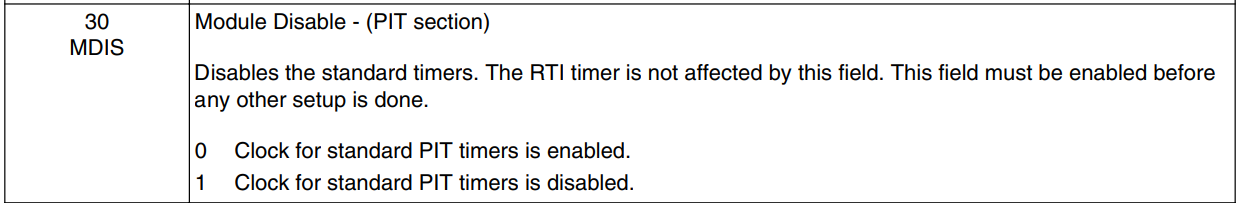


Figure 55



Figure 56

Bitul 0 din registrul PIT\_MCR(câmpul FRZ-Freeze) este setat pe 1 pentru a oprii decrementarea lui LDVAL in modul debug.

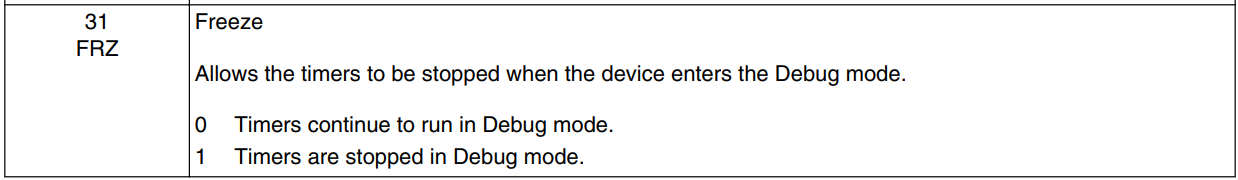


Figure 57



Figure 58

Urmatoarea etapă este calcularea valorii ce trebuie introdusa in registrul PIT\_LDVALn pentru a stabili perioada dorită, mențtionată in cerința proiectului.

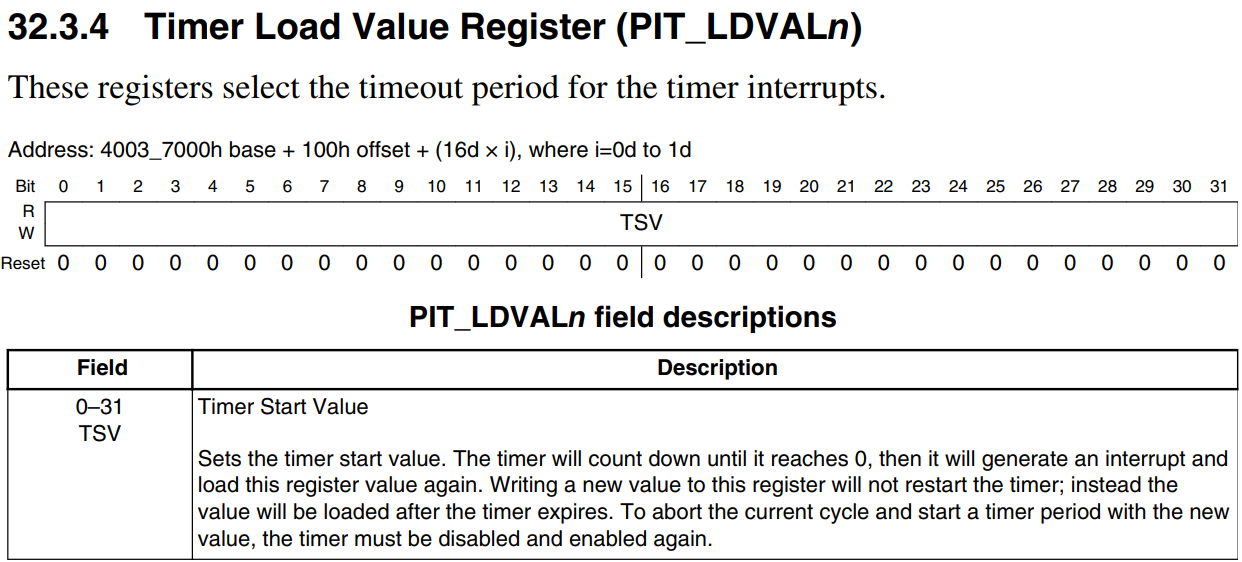


Figure 59

După efectuarea calculelor, s-a determinat că valoarea aproximativă pentru un interval de timp de 395 milisecunde este 0x3F3332. Această valoare se încadrează într-un spațiu de 32 de biți, permițându-ne astfel să utilizăm exclusiv canalul 0 al modulului PIT pentru această temporizare.

Registrul PIT\_TCTRLn(PIT Timer Control Register) - conține biți de control pentru fiecare canal al modului PIT.

Activăm întreruperile si timer-ul pe canalul 0 prin intermediul bițtilor 0 si 1 din registrul PIT\_TCTRLn(câmpurile TIE și TEN), pe care îi setăm pe 1.

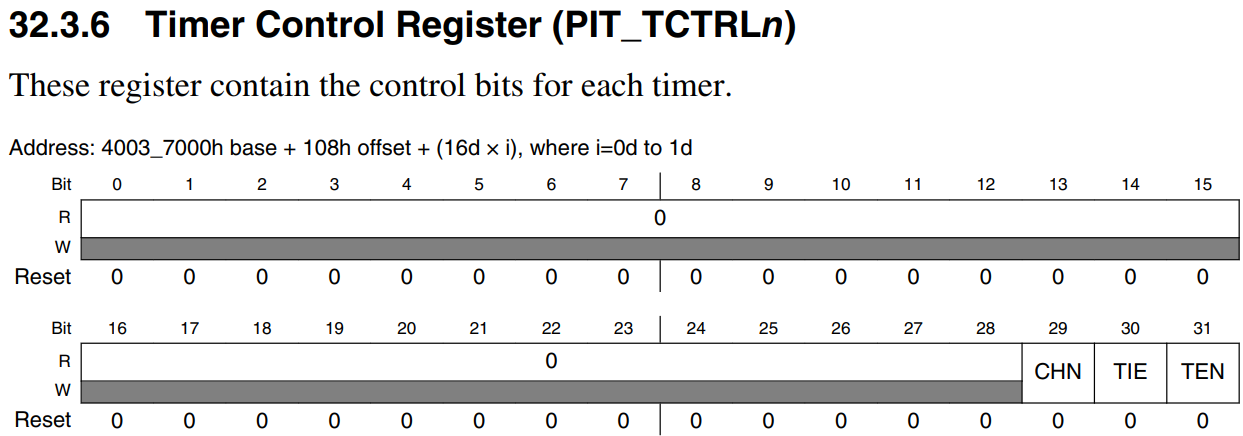


Figure 60



Figure 61



Figure 62

Vom configura NVIC(Nested Vector Interrupt Timer) pentru a activa intreruperea mascabilă. Setăm prioritatea 5 acestei întreruperi.

# Rezultate Python

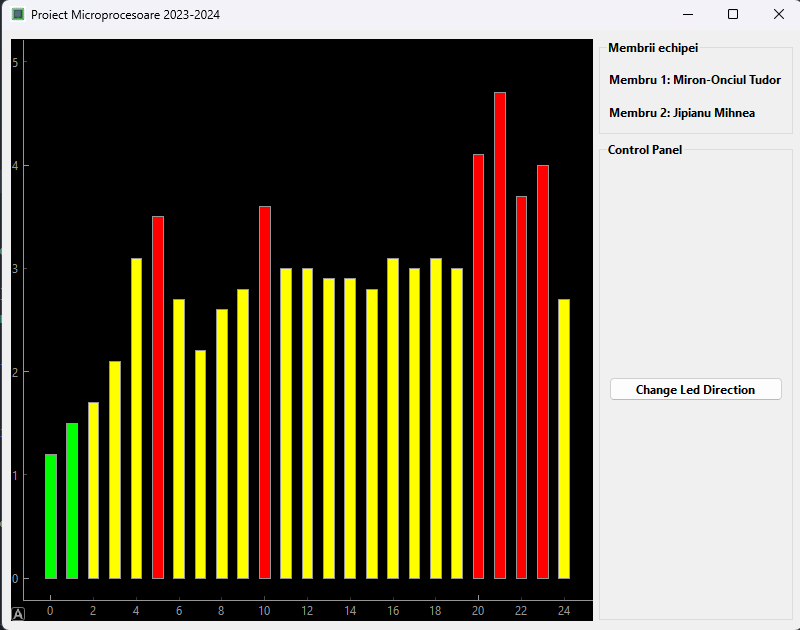


Figure 63 – Grafic Python

# Demo

# Dificultăți întâmpinate

* Din lipsa cunoștiințelor în electronică, am întâmpinat probleme la conectarea ledurilor și rezistențelor de pe plăcuță.
* Probleme la citirea datelor transmise prin UART din aplicația Python. Intenția a fost de a citi câte o linie la un anumit moment de timp, dar aplicația citea jumătate de linie, nu toată linia. Pentru a rezolva problema, am urmat pașii:
  + Am deschis conexiunea serială
  + Citeam 2 linii, a doua fiind cea completă
  + Am închis conexiunea
  + Reptam acești pași la fiecare 250 ms

# Recomandări

* O mică prezentare a conceptelor de bază în electronică.

# Referințe

* <https://www.designcap.com/> - cele 2 tabele pentru diagramele de stări
* <https://wiki.dfrobot.com/Flame_sensor_SKU__DFR0076> - documentație senzor
* <https://www.drawio.com/> - diagrama arhitecturii
* <https://www.tinkercad.com/> - pentru figura 5
* KL25 Sub-Family Reference Manual