**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 flacără**

**Platforma de dezvoltare frdm-kl25z**

Std. sg. maj. Răzvan Gabriel Avrămescu

Std. sg. maj. Paul Cioloca

Grupa C114A

**București**

**2024**

**Cuprins**

1. Scopul proiectului .............................................................................................................. **3**
2. Prezentare platformă de dezvoltare FRDM-KL25Z .......................................................... **4**
3. Prezentare senzori .............................................................................................................. **5**

3.1. Senzor DFR0076 ......................................................................................................... **5**

* + 1. Conectare senzor-microcontroller .................................................................... **5**

3.2. Servomotor Tower Pro Micro Servo 9g SG90 ........................................................... **6**

3.2.1. Conectare servomotor-microcontroller ........................................................ **7**

4. Logica aplicației ................................................................................................................... **8**

4.1. Schema bloc a proiectului ........................................................................................... **8**

4.2. Diagrame ..................................................................................................................... **9**

5. Descriere program .............................................................................................................. **10**

5.1. Funcția main .............................................................................................................. **10**

5.2. Modulul GPIO .......................................................................................................... **12**

5.3. Modulul PIT .............................................................................................................. **16**

5.4. Modulul ADC ........................................................................................................... **23**

5.5. Modulul UART ......................................................................................................... **29**

5.6. Modulul PWM .......................................................................................................... **37**

5.7. Modulul TSI ............................................................................................................. **40**

6. Prezentare interfață grafică ................................................................................................ **43**

7. Probleme întâmpinate ........................................................................................................ **44**

8. Referințe ............................................................................................................................. **44**

9. DEMO Link ....................................................................................................................... **45**

10. GitHub Link ..................................................................................................................... **45**

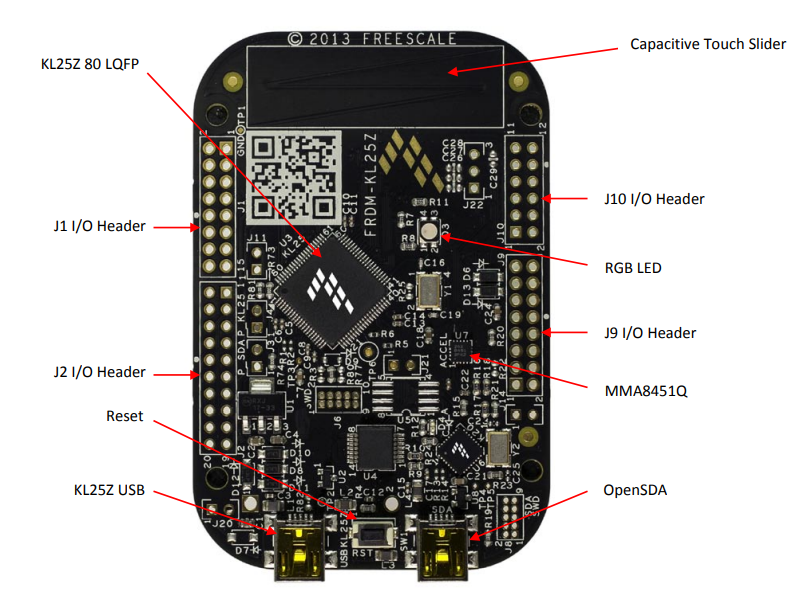
**Scopul proiectului**

Scopul acestui proiect este să exploreze și să demonstreze modul în care putem controla direcția de rotație a unui motor servo utilizând semnale de modulație a lățimii impulsurilor (PWM), pe baza datelor furnizate de un senzor de flacără.

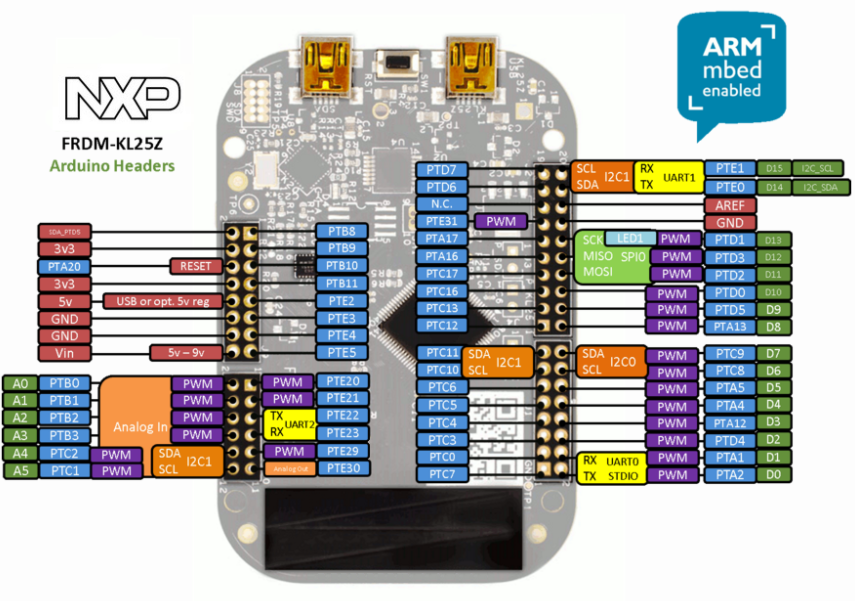
Proiectul se concentrează pe dezvoltarea unui sistem capabil să interpreteze semnalele analogice provenite de la senzorul de flacără pentru a genera semnale PWM corespunzatoare pentru controlarea poziției motorului servo. Un aspect vital al proiectului este capacitatea sa de a transmite datele din senzor către un calculator prin intermediul interfeţei UART (Universal Asynchronous Receiver-Transmitter), permiţând astfel vizualizarea datelor în timp real printr-o interfață grafică. Această funcționalitate nu doar îmbunătățește interactivitatea sistemului, dar oferă și o platformă pentru analiza și monitorizarea datelor

**Bonus:** Am extins proiectul prin includerea unei funcționalități suplimentare care permite controlul poziției servomotorului utilizând senzorul TSI (Touch Sensing Input) integrat pe microcontroler.

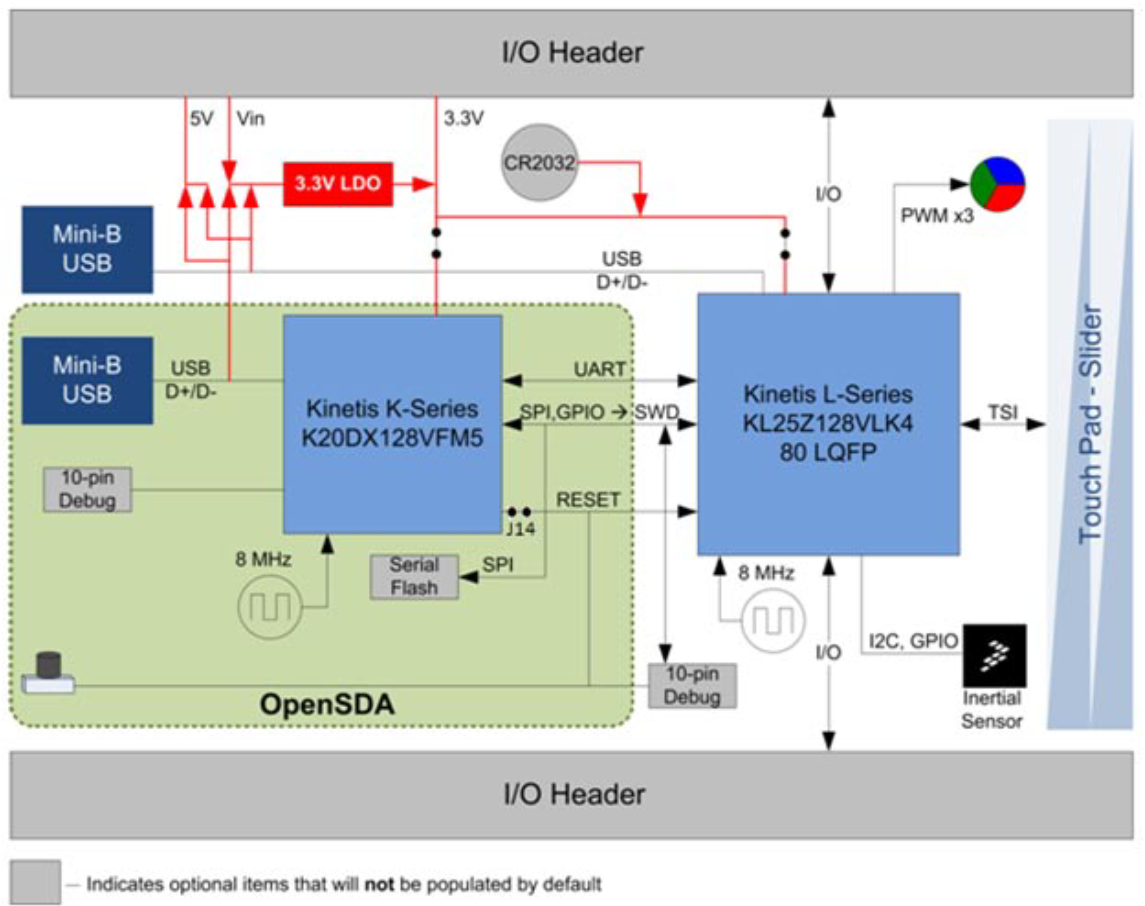
1. **Platforma de dezvoltare FRDM-KL25Z**



*[[1]](#footnote-1)Figura 1. Microncontroller FRDM-KL25Z*



*[[2]](#footnote-2)Figura 2. Pini placă dezvoltare*

**

*[[3]](#footnote-3)Figura 3. Diagramă bloc FRDM-KL25Z*

1. **Prezentare senzori**

**3.1. Senzor DFR0076**

Senzorul DFR0027 este un senzor analog care detecteaza flacăra, avand un range de detecție între 20cm și 100cm.Acesta poate lucra cu temperaturi cuprinse între -25 grade Celsius si 85 grade Celsius.



*Figura 4. Senzorul DFR0076*

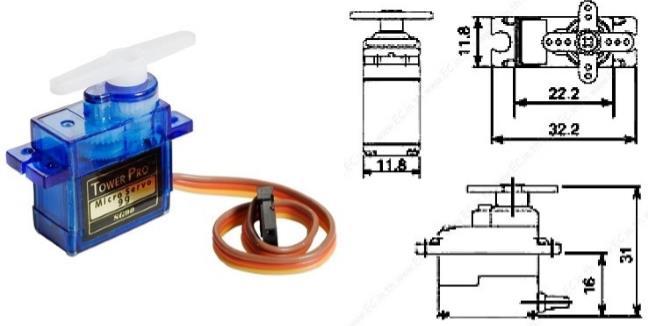
Senzorul se conectează la placa de dezvoltare prin trei conectori:unul negru(GND), unul roșu(care reprezintă alimentarea: 3.3V) și unul verde(care reprezinta OUTPUT-UL).

* + 1. **Conectare senzor-microcontroller**

Vom conecta senzorul astfel:

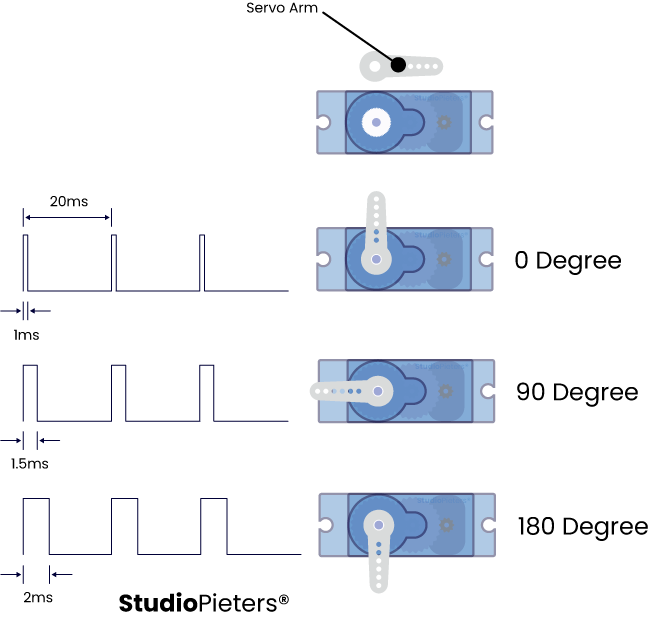
* Firul **verde** senzorul la **PTC2**
* Firul **roșu** conectează senzorul la **3.3V**
* Firul **negru** conectează senzorul la **GND**

**3.2. Servomotor Tower Pro Micro Servo 9g SG90**



*Figura 5. Servomotor Tower Pro Microservo 9g SG90*

Tower Pro Micro Servo 9g SG90 este un servomotor compact și ușor, ideal pentru proiecte de robotică și modele controlate prin radio. Oferă un unghi de rotație de 180 de grade, funcționează la 4.8-6V și este popular datorită prețului său accesibil și fiabilității.

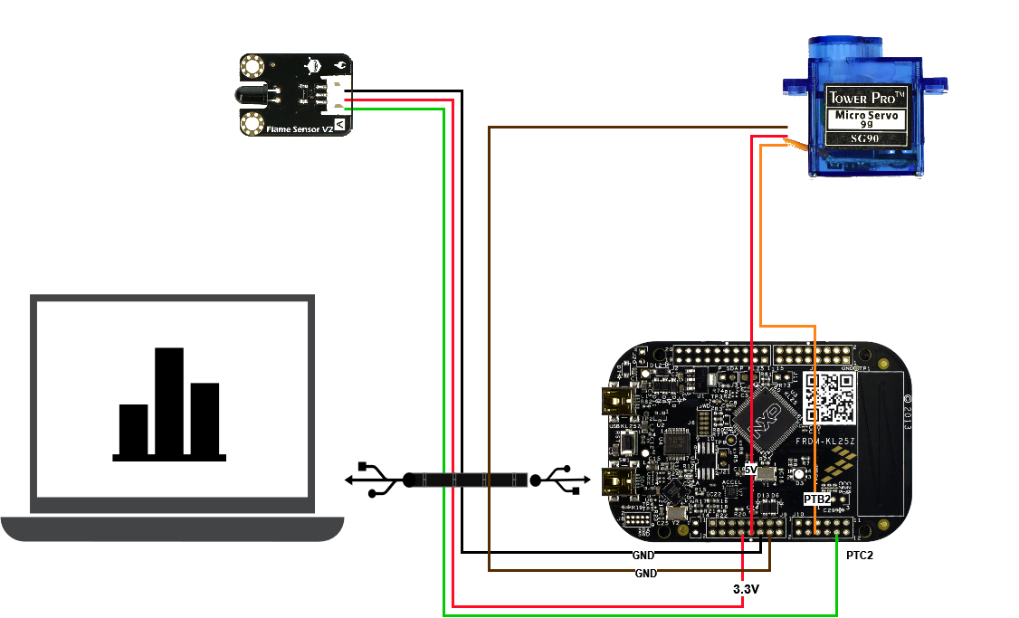


*Figura 6. Mapare perioadă semnal-poziție braț de rotație*

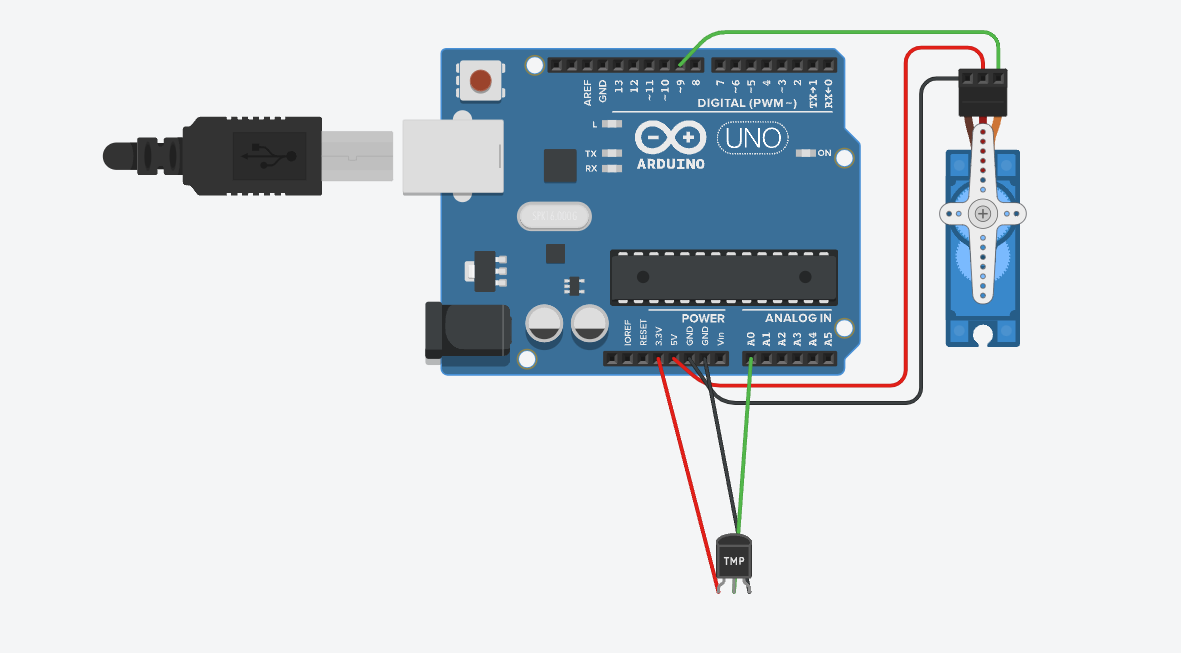
**3.2.1. Conectare servomotor-microcontroller**

Vom conecta servo motorul astfel:

* Firul **maro** servomotorul la **GND**
* Firul **roșu** conectează servomotorul la **5V**
* Firul **portocaliu** conectează servomotorul la **PTB2**



*Figura 6. Schema complete de conexiuni a proiectului*

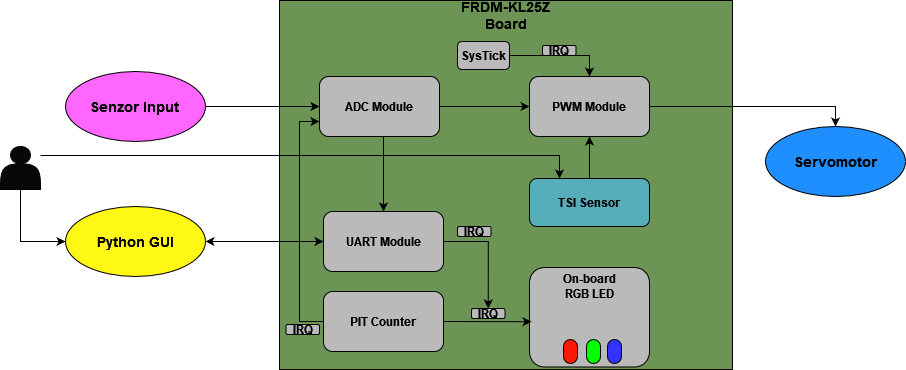


*Figura 7. Schema Tinkercad a proiectului*

În figura de mai sus se poate observa schema din Tinkercad a proiectului cu următoarele mențiuni:în cadrul mediului de proiectare Tinkercad nu am găsit plăcuța NXP, așa ca am uitilizat o plăcuță Arduino, și nici un senzor de flacără așa că am folosit un senzor de temperatură.

1. **Logica aplicației**

**4.1. Schema bloc a proiectului**

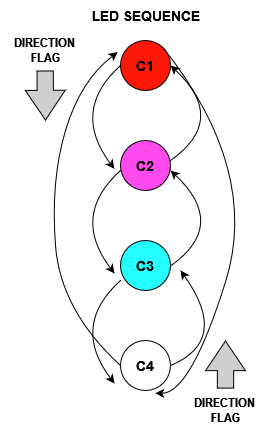


*Figura 8. Schema bloc a proiectului*

* **Senzor Input**: Un senzor extern trimite date la placa FRDM-KL25Z, prin intermediul modulului ADC (Analog-to-Digital Converter), pentru a fi procesate digital.
* **SysTick**: O unitate de timp sistem (System Tick Timer) este folosită pentru a genera întreruperi de timp sau pentru a măsura intervaluri de timp.
* **PWM Module**: Un modul de modulare a lățimii de impuls (Pulse Width Modulation) este utilizat pentru a controla servomotorul.
* **TSI Sensor**: Senzorul de Interfață Tactilă Sensibilă (Touch Sensing Input) este utilizat pentru input bazat pe atingere.
* **UART Module**: Modulul Universal Asynchronous Receiver/Transmitter este folosit pentru comunicarea serială.
* **PIT Counter**: Un temporizator de interval de programare (Programmable Interval Timer) care poate genera întreruperi la intervale regulate.
* **IRQ**: O cerere de întrerupere (Interrupt Request) este utilizată pentru a semnala o condiție care necesită atenția procesorului.
* **On-board RGB LED**: LED-uri RGB integrat pe placa FRDM-KL25Z
* **Servo motor**: ajustare unghi braț rotativ.
* **Python GUI**: O interfață grafică de utilizator Python pentru afișarea sau controlul datelor și operațiunilor sistemului.
  1. **Diagrame**



*Figura 9. Diagrama de stări*



*Figura 10. Diagrama de stări a led-urilor*

1. **Descriere program**
   1. **Funcția main**

În fisierul main.c am inclus fișierele header in care sunt declarate funcții si variabile ce urmeaza a fi folosite: *”gpio.h”*(funcția ledInit), *”Pit.h”*(funcția pitInit), *”Adc.h”*(funcția ADC0\_Init), *”Uart.h”*(funcția UART0\_Init)

#include "Gpio.h"

#include "Pit.h"

#include "Adc.h"

#include "Uart.h"

#include "Pwm.h"

#include "ClockSettings.h"

#include "tsi.h"

*uint16\_t* avg = 0;

*uint16\_t* TSI\_Readings = 0;

*uint8\_t* c = 'y';

*int* main(*void*)

{

    SystemClockTick\_Configure();

    TSI\_init();

    UART0\_Init(9600);

    ADC0\_Init();

    ledInit();

    pitInit();

    TPM2\_Init();

*uint8\_t* parte\_fractionara1 = 0;

*uint8\_t* parte\_fractionara2 = 0;

*uint8\_t* parte\_zecimala = 0;

    START\_flag = 1;

    while (1)

    {

        if (flag\_50ms)

        {

            Signal\_Control(parte\_zecimala);

            flag\_50ms = 0U;

        }

   if (START\_flag)

      {

if (flag)

        {

*float* measured\_voltage = (analog\_input \* 3.3f) / 4095;

                parte\_zecimala = (*uint8\_t*)measured\_voltage;

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

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

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

                UART0\_Transmit('.');

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

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

                UART0\_Transmit('V');

                UART0\_Transmit(0x0A);

                UART0\_Transmit(0x0D);

                flag = 0;

            }

        }

    }

}

Se apeleaza funcțiile de inițializare ale fiecarui modul în parte, se declara și se inițializează variabilele necesare, după care, într-o buclă infinită se verifică dacă aplicația a primit semnalul de START (trimit din interfață caracterul ‘S’). În cazul în care caracterul a fost trimis se trimite prin UART interfeței grafice valorile achiziționate de senzorul de flacără. În cazul în care se declanșează o întrerupere, este parcursă logica aplicației. De asemenea pentru a controla servomotorul, am folosit system clock: o data la 50ms este apelată funcția Signal\_Control() cu parametrul partea întreagă a valorii primite de la senzor. Pentru a opri aplicația (starea PAUSED), apăs butonul Stop din interfață (se trimite caracterul ‘s’ prin UART).

* 1. **Modulul GPIO**

#include "Gpio.h"

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

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

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

*void* ledInit(*void*)

{

*/\* Activarea semnalului de ceas pentru pinii folositi In cadrul led-ului RGB\*/*

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

*/\* --- RED LED ---\*/*

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

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

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

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

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

*/\* Aprinderea led-ului (punerea pe 0 logic)\*/*

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

*/\* --- GREEN LED ---\*/*

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

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

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

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

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

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

    GPIOB\_PSOR |= (1 << GREEN\_LED\_PIN);

*/\* --- BLUE LED ---\*/*

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

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

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

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

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

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

    GPIOD\_PSOR |= (1 << BLUE\_LED\_PIN);

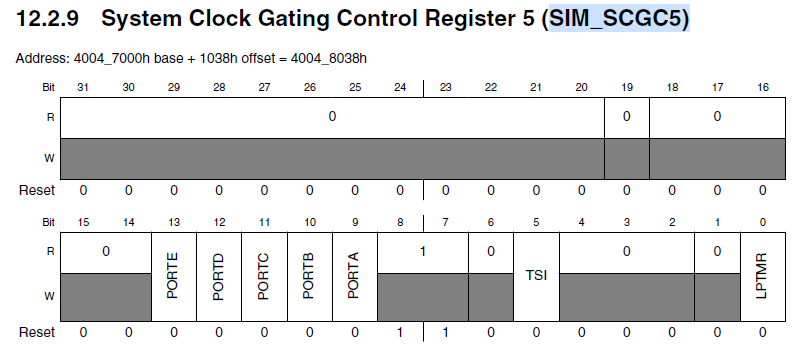
}

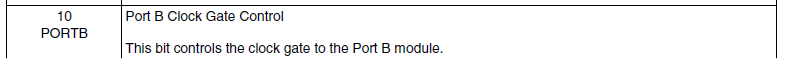
În cadrul fișierului Gpio.c sunt create mai multe macrodefiniții:

* RED\_LED\_PIN -canalul specific pentru aprinderea led-ului de culoare roșie de pe placă care se afla pe port-ul B
* GREEN\_LED\_PIN-canalul specific pentru aprinderea led-ului de culoare verde de placă care se afla pe port-ul B
* BLUE\_LED\_PIN-canalul specific pentru aprinderea led-ului de culoare albastră de pe placă care se afla pe port-ul D

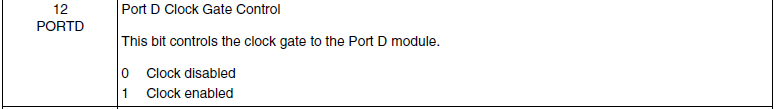
Primul pas în inițializarea modulului GPIO îl reprezintă activarea semnalului de ceas pentru led-urile pe care dorim să le folosim, în cazul de față trebuie sa activăm semnalul de ceas pentru port-ul B si D.

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









Apoi configurăm GPIO ca variantă de multiplexare pentru pinii corespunzători

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

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

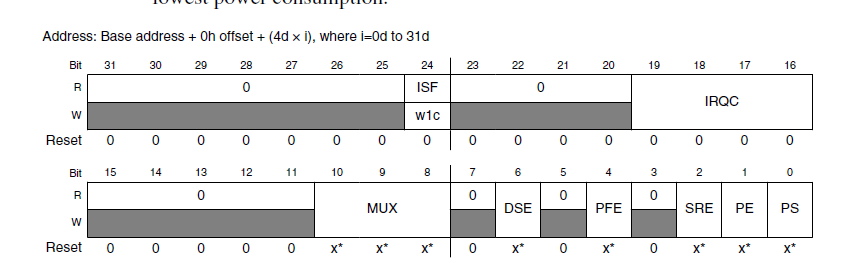
PORTB->PCR[GREEN\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

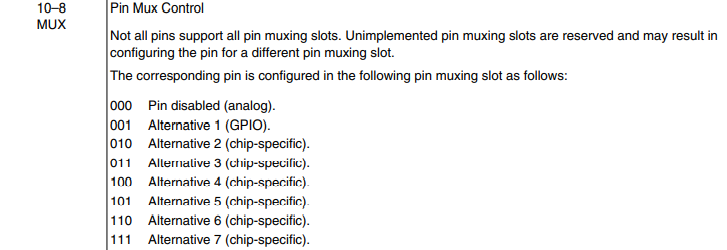
PORTB->PCR[GREEN\_LED\_PIN] |= PORT\_PCR\_MUX(1);

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

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





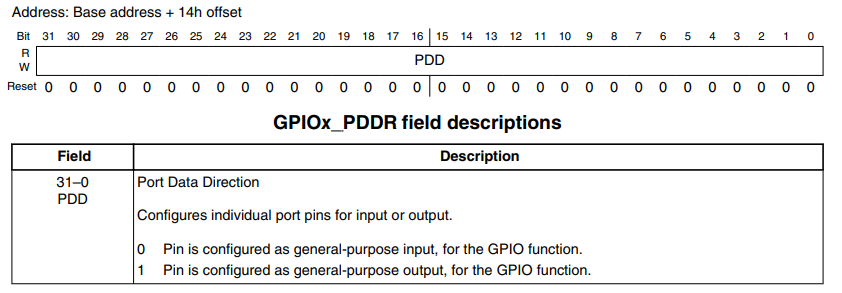


Configurăm pinii pe post de output.

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

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

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



Stingem(punem pe 1 logic) led-ul albastru și led-ul verde și aprindem(punere pe 0 logic) led-ul roșu deoarece dorim ca secventa noastra sa inceapa cu roșu.

* 1. **Modulul PIT**

#include "Pit.h"

#include "Uart.h"

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

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

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

*static* *uint8\_t* state;

*void* pitInit(*void*)

{

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

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

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

    PIT\_MCR &= ~PIT\_MCR\_MDIS\_MASK;

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

    PIT->MCR |= PIT\_MCR\_FRZ\_MASK;

*/\* Setarea valoarea numaratorului de pe canalul 0 la o perioada de 1 secunda\*/*

    PIT->CHANNEL[0].LDVAL = 0x911EB7;

*/\* Activarea intreruperilor pe canalul 0\*/*

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

*/\* Activarea timerului de pe canalul 0\*/*

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

*/\* Setarea valoarea numaratorului de pe canalul 1 la o perioada de 10 secunde\*/*

    PIT->CHANNEL[1].LDVAL = 0x9FFFFF;

*/\* Activara intreruperilor pe canalul 1\*/*

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

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

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

    state = 0;

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

    NVIC\_ClearPendingIRQ(PIT\_IRQn);

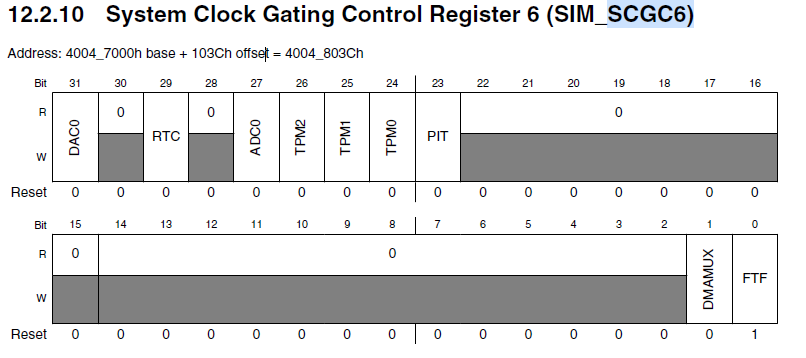
    NVIC\_SetPriority(PIT\_IRQn, 5);

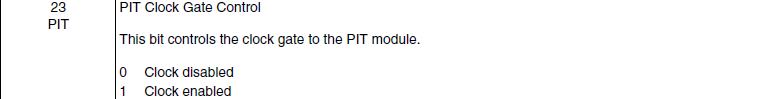
    NVIC\_EnableIRQ(PIT\_IRQn);

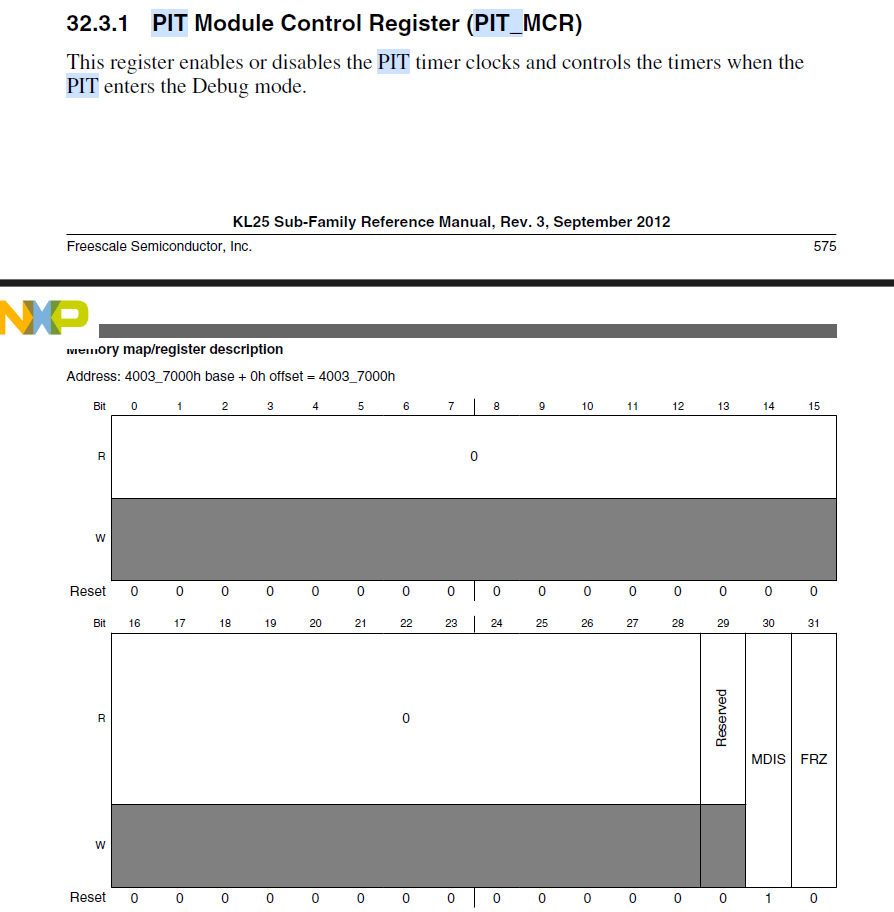
}

Primul pas este activarea semnalului de ceas pentru perifericul PIT.

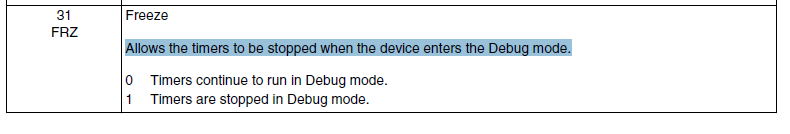
SIM->SCGC6 |= SIM\_SCGC6\_PIT\_MASK;











Setăm perioadele de timeout pentru cele 2 canale.

PIT->CHANNEL[0].LDVAL = 0x911EB7;

*/\* Activarea intreruperilor pe canalul 0\*/*

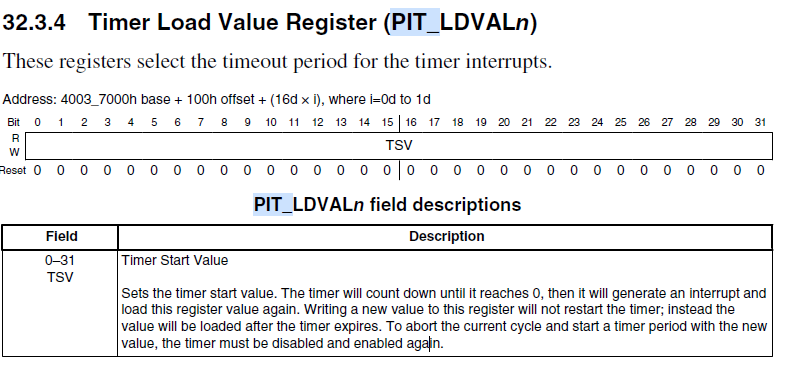
    PIT->CHANNEL[0].TCTRL |= PIT\_TCTRL\_TIE\_MASK;

*/\* Activarea timerului de pe canalul 0\*/*

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

*/\* Setarea valoarea numaratorului de pe canalul 1 la o perioada de 10 secunde\*/*

    PIT->CHANNEL[1].LDVAL = 0x9FFFFF;



Activăm întreruperile si timer-ului pentru fiecare canal.

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

*/\* Activarea timerului de pe canalul 0\*/*

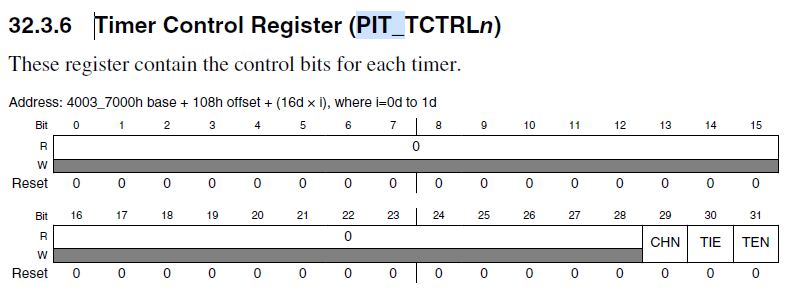
    PIT->CHANNEL[0].TCTRL |= PIT\_TCTRL\_TEN\_MASK;

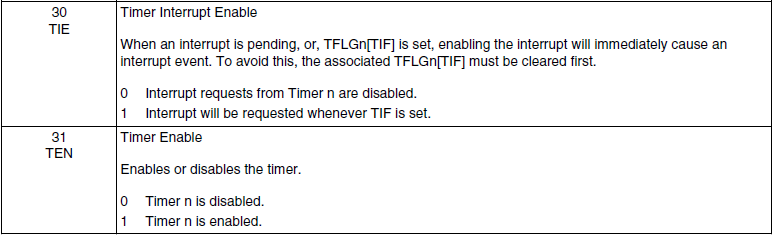
*/\* Activara intreruperilor pe canalul 1\*/*

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

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

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





Setăm statusul led-ului de pe placă astfel incât la fiecare întrerupere de pe canalul 0 să putem ști ce culoare este aprinsă pe led-ul de pe placă și să știm ce culoare ar trebui să aprindem.

state=0;

Activăm întreruperile pentru modulul pit și setăm prioritatea acestora.

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

    NVIC\_ClearPendingIRQ(PIT\_IRQn);

    NVIC\_SetPriority(PIT\_IRQn, 5);

    NVIC\_EnableIRQ(PIT\_IRQn);

În cadrul funcției care gestionează întreruperea se verifică dacă întreruperea aparută e pe canalul 0 (canalul care se ocupă de secvența de culori de pe placă) sau pe canalul 1 (canalul care se ocupă de activarea întreruperilor de pe modulul ADC, astfel încât să primim valori de la senzor la un interval de 1 secundă).

Dacă întreruperea este specifcă canalului 0 atunci se verifica stadiul in care se afla led-ul și se aprind led-urile astfel încât să se continue secvența de culori conform cerinței. Flagul state indică starea curentă a ledurilor, iar DIRECTION\_flag se setează în funcție de poziția butonului de tip switch din interfață (Normal/Rverse). La apăsarea acestui buton de către utilizator, se trimie prin intermediul UART caracterul ‘N’ - Normal sau ‘R’- Rerversed, iar în funcție de acesta este setat DIRECTION\_flag. Acest flag schimbă logica întreruperii(ordinea secvenței de leduri).

Dacă întreruperea este specifică canalului 1 atunci se activează întreruperile pentru modulul ADC.

După ce sunt gestionate întreruperile de pe cele 2 canale, acestea sunt marcate ca și gestionate, astfel încât întreruperea specifică canalului respectiv să poată fi gestionată din nou după perioada specifică.

*void* PIT\_IRQHandler(*void*)

{

    if (START\_flag)

    {

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

        GPIOB\_PSOR |= (1 << GREEN\_LED\_PIN);

        GPIOD\_PSOR |= (1 << BLUE\_LED\_PIN);

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

        {

            if (state == 0)

            {

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

            }

            else if (state == 1)

            {

                GPIOD\_PTOR |= (1 << BLUE\_LED\_PIN);

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

            }

            else if (state == 2)

            {

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

                GPIOD\_PTOR |= (1 << BLUE\_LED\_PIN);

            }

            else if (state == 3)

            {

*/\*stins\*/*

            }

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

            if (DIRECTION\_flag == 1)

            {

                state++;

            }

            else

            {

                state--;

            }

            state = state % 4;

        }

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

        {

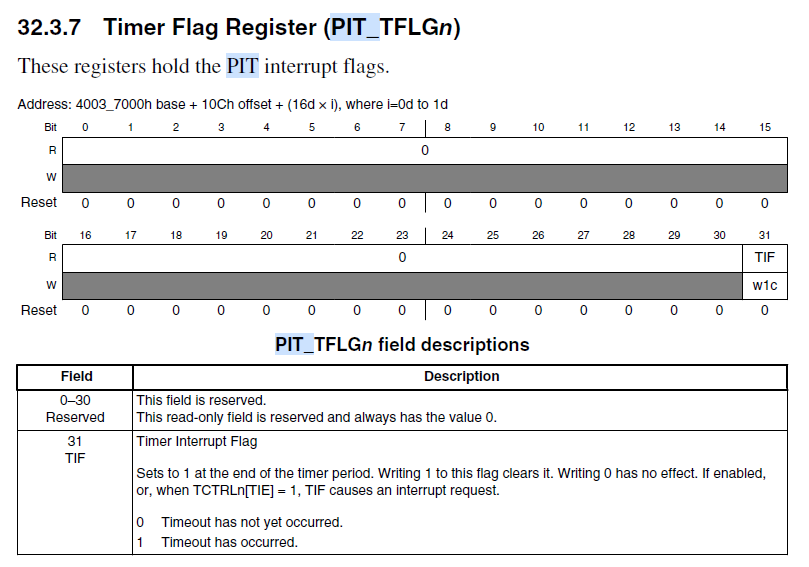
            ADC0->SC1[0] |= ADC\_SC1\_AIEN\_MASK;

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

        }

    }

}



* 1. **Modulul ADC**

#include "Adc.h"

#include "Uart.h"

#define ADC\_CHANNEL (11) */\* PORT C PIN 1\*/*

*volatile* *uint8\_t* flag = 0;

*uint16\_t* analog\_input;

*void* ADC0\_Init(*void*)

{

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

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

*/\* Functia de calibrare\*/*

    ADC0\_Calibrate();

    ADC0->CFG1 = 0x00;

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

*Selectarea sursei de ceas pentru generarea ceasului intern --> ADICLK*

*Selectarea ratei de divizare folosit de periferic pentru generarea ceasului intern --> ADIV*

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

    ADC0->CFG1 |= ADC\_CFG1\_MODE(1) |

                  ADC\_CFG1\_ADICLK(0) |

                  ADC\_CFG1\_ADIV(2);

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

    ADC0->SC1[0] = 0x00;

    ADC0->SC3 = 0x00;

*/\* Selectarea modului de conversii continue,*

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

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

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

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

*/\* Enables conversion complete interrupts\*/*

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

    NVIC\_ClearPendingIRQ(ADC0\_IRQn);

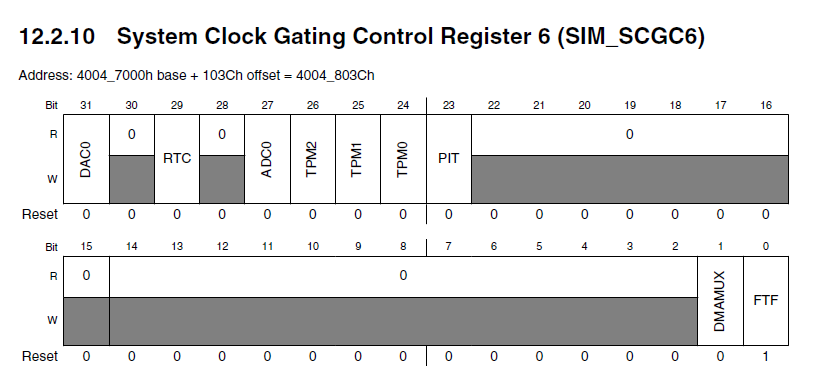
    NVIC\_EnableIRQ(ADC0\_IRQn);

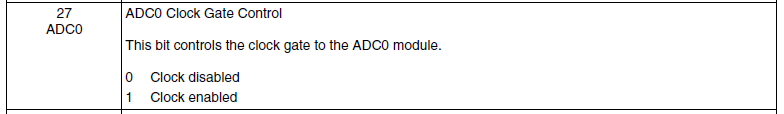
}

Este activat semnalul de ceas pentru modulul ADC.

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

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





Apoi apelăm funcția ADC0\_Calibrate, care este conformă manualului de referință al platformei.

*/\* Functia de calibrare\*/*

    ADC0\_Calibrate();

Resetez registrul CFG pe care urmează să îl configurăm conform cerinței.

ADC0->CFG1 = 0x00;

Setăm conversia pe 12 biți, selectăm input clock-ul BUS CLOCK, și selectăm raport-ul de împărțire pentru generarea ceasului intern.

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

*Selectarea sursei de ceas pentru generarea ceasului intern --> ADICLK*

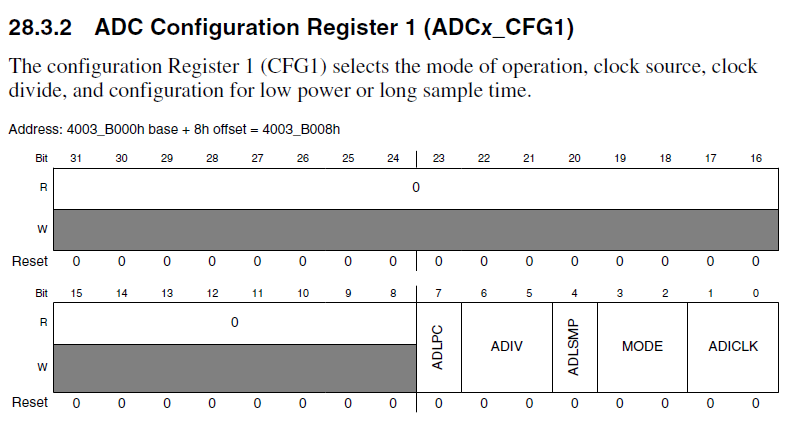
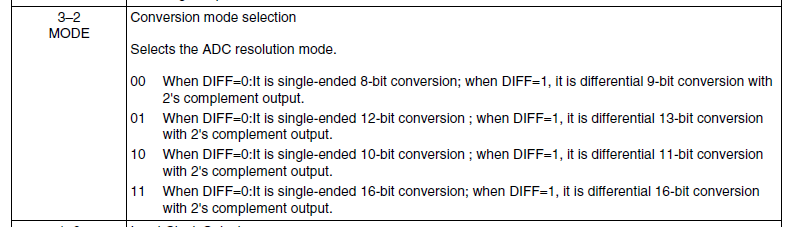
*Selectarea ratei de divizare folosit de periferic pentru generarea ceasului intern --> ADIV*

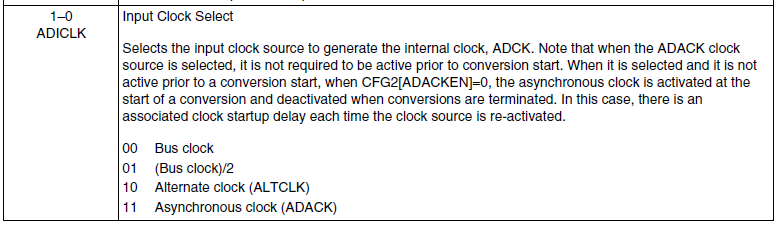
*Set ADC clock frequency fADCK less than or equal to 4 MHz (PG. 494)\*/*

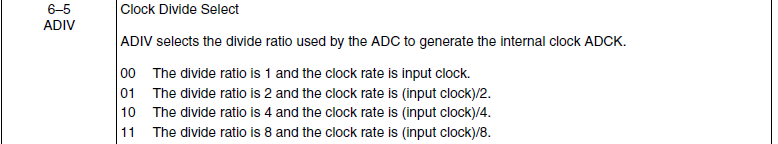
    ADC0->CFG1 |= ADC\_CFG1\_MODE(1) |

                  ADC\_CFG1\_ADICLK(0) |

                  ADC\_CFG1\_ADIV(2);

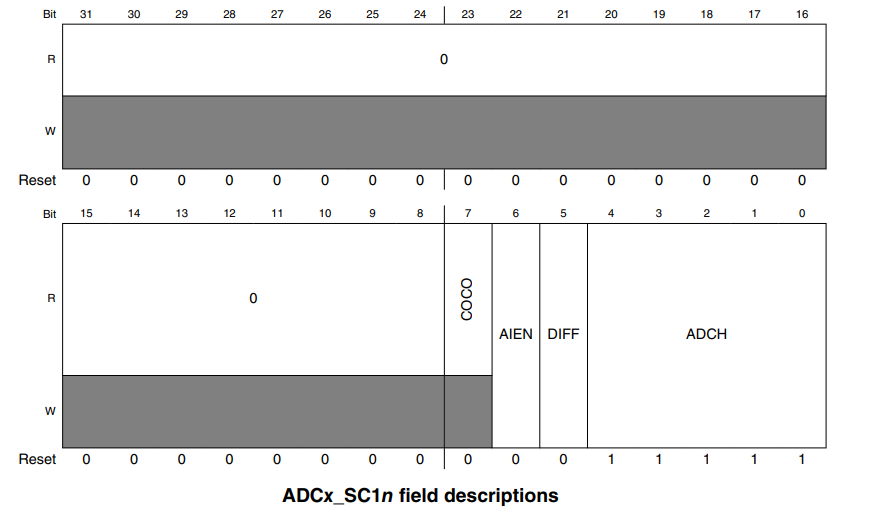


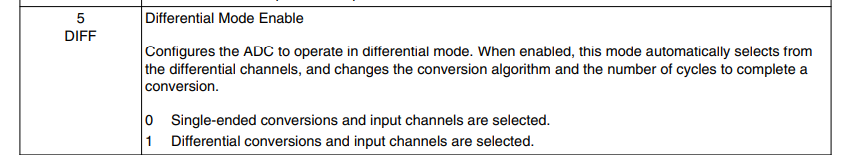


Activăm conversiile single-ended.

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

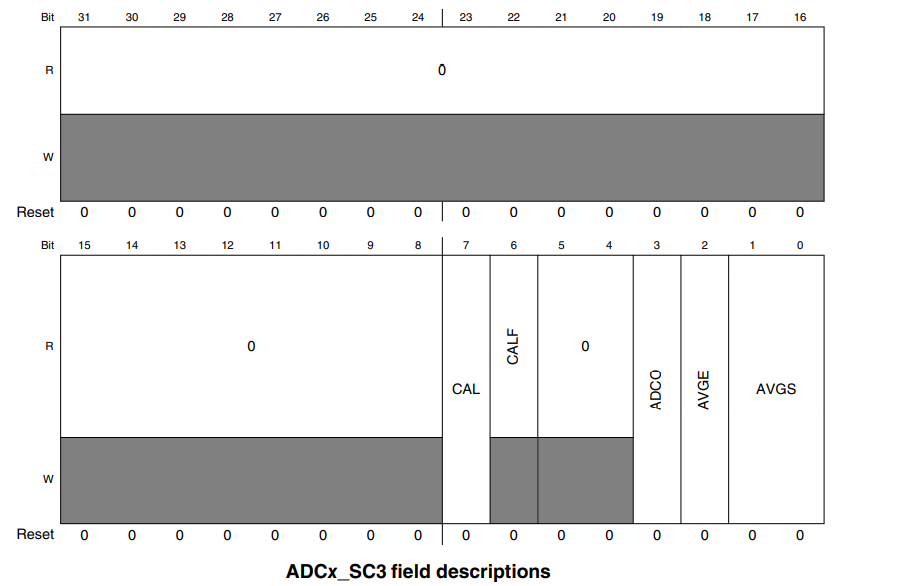
    ADC0->SC1[0] = 0x00;





Resetăm registrul SC3 înainte de a-l configura.

ADC0->SC3 = 0x00;

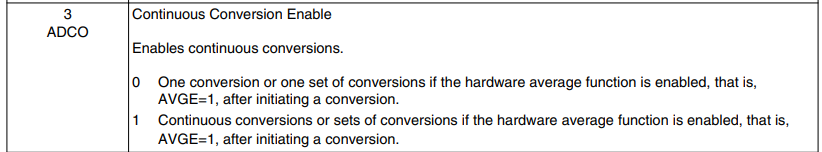


Activăm conversiile continue.

*/\* Selectarea modului de conversii continue,*

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

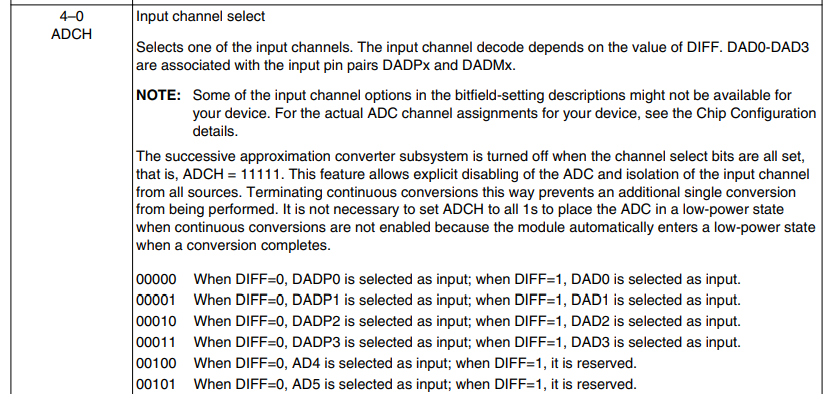
    ADC0->SC3 |= ADC\_SC3\_ADCO\_MASK;



Selectăm canalul specific pentru senzor definit la începutul fișierului.

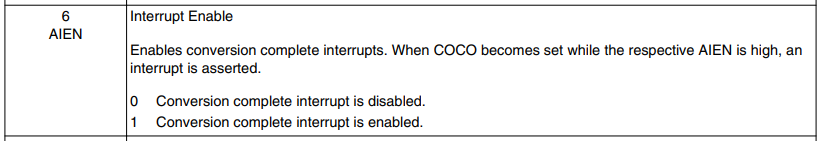
*/\* Activarea subsistemului de conversie prin aproximari succesive pe un anumit canal (PG.464)\*/*

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



Activăm întreruperile la terminarea conversiilor.

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



Activăm întreruperile pentru modulul ADC.

NVIC\_ClearPendingIRQ(ADC0\_IRQn);

NVIC\_EnableIRQ(ADC0\_IRQn);

În cadrul funcției de gestionare a întreruperii specifice acestui modul, salvăm in variabila externa analog\_input (definită în adc.h) valoarea de la senzor rezultată în urma conversiei, setăm valoarea variabilei externe volatile flag (definită in adc.h) la 1 (marcăm faptul că s-a realizat o conversie) și dezactivăm întreruperile specifice acestui modul.

*void* ADC0\_IRQHandler(*void*)

{

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

    flag = 1;

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

}

* 1. **Modulul UART**

*void* UART0\_Init(*uint32\_t* *baud\_rate*)

{

*uint32\_t* osr;

*uint16\_t* sbr;

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

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

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

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

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

*Intrucat dorim sa folosim pinii PTA1, respectiv PTA2 pentru comunicarea UART\*/*

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

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

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

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

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

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

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

*/\*Dezactivare receptor si emitator pentru a configura\*/*

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

*/\*Configurare Baud Rate\*/*

    osr = 4; */\* Over-Sampling Rate (numarul de esantioane luate per bit-time)\*/*

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

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

*0x1F00 -       0       0   0    1   1   1   1   1      0   0   0   0   0   0   0   0*

*0   0   0    b13 b12 b11 b10 b09    0   0   0   0   0   0   0   0 >> 8*

*BDH  -   0   0   0    b13 b12 b11 b10 b09*

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

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

    UART0->BDH = UART0\_BDH\_SBR((*uint8\_t*)((sbr & 0xFF00) >> 8));

    UART0->BDL = (*uint8\_t*)(sbr & 0xFF);

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

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

    UART0->C1 = 0;

*/\*Setare LSB first\*/*

    UART0->S2 = UART0\_S2\_MSBF(0);

*/\*Dezactivare intreruperi la transmisie\*/*

    UART0->C2 |= UART0\_C2\_TIE(0);

    UART0->C2 |= UART0\_C2\_TCIE(0);

*/\*Activare intreruperi la receptie\*/*

    UART0->C2 |= UART0\_C2\_RIE(1);

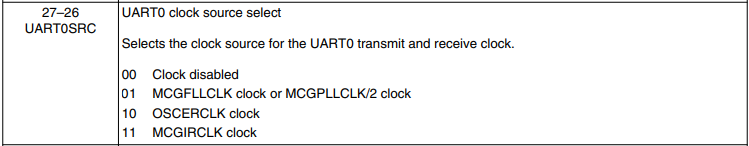
    UART0->C2 |= ((UART\_C2\_RE\_MASK) | (UART\_C2\_TE\_MASK));

    NVIC\_EnableIRQ(UART0\_IRQn);

}

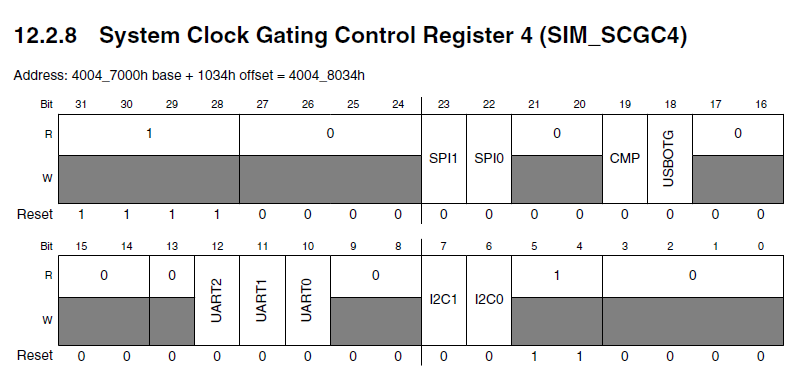
Selectam sursa de ceas pentru modulul UART0.

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



Activam semnalul de ceas pentru modulul UART0.

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

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

Configurăm pinul 2 din portul A pe modulul UART0\_TX si UART0\_RX prin selectarea alternativei 2.

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

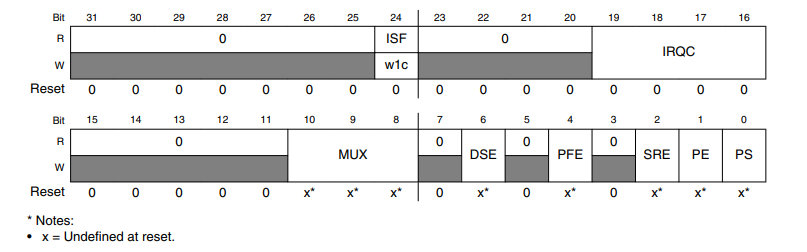
*/\*la care avem acces prin intermediul multiplexarii\*/*

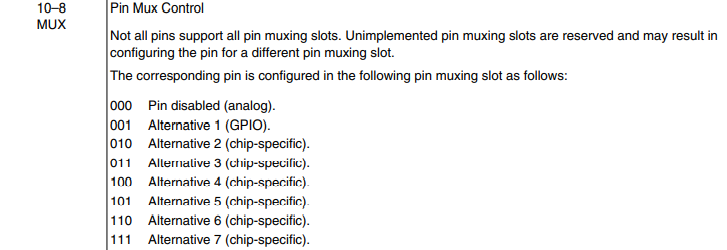
    PORTA->PCR[1] = ~PORT\_PCR\_MUX\_MASK;

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

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

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





Dezactivam transmisia si recepția pentru a putea configure mai departe

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

Pentru baud rate-ul de 9600 configurăm SBR de la regiștrii UART0\_BDH și UART0\_BDL.

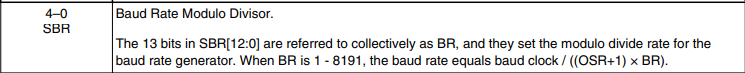
osr = 4;

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

UART0->BDH = UART0\_BDH\_SBR((*uint8\_t*)((sbr & 0xFF00) >> 8));

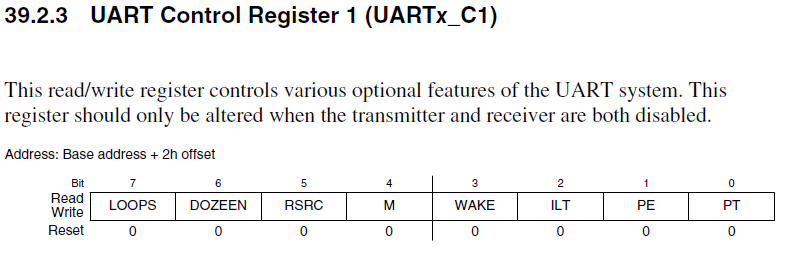
UART0->BDL = (*uint8\_t*)(sbr & 0xFF);

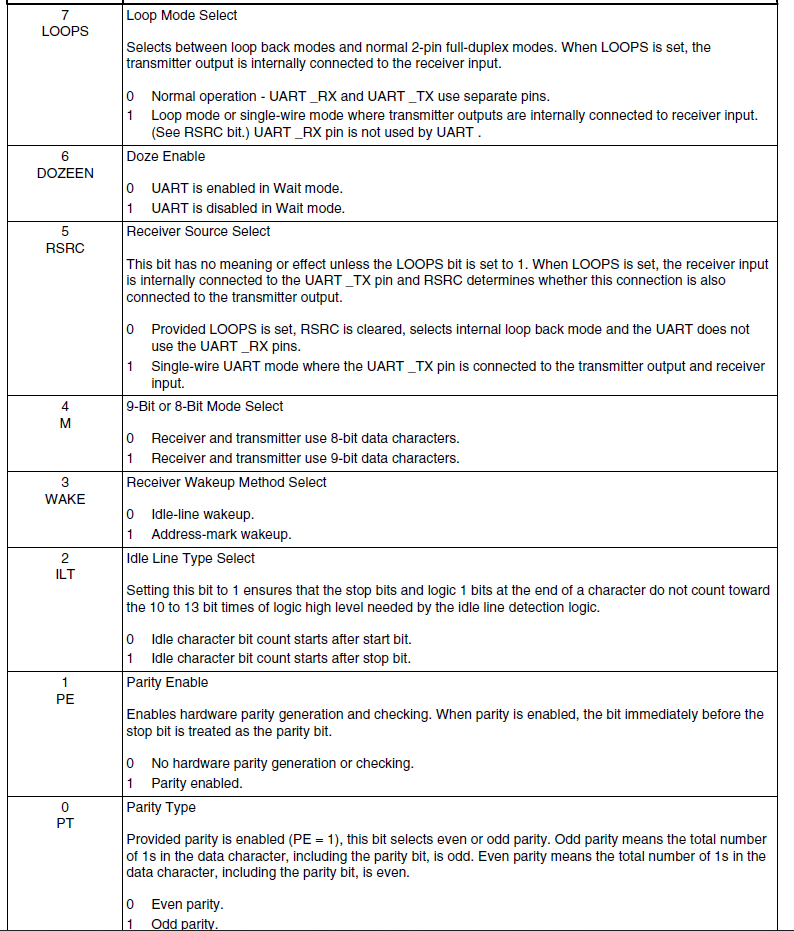
UART0->C4 = (UART0->C4 & ~UART0\_C4\_OSR\_MASK) | UART0\_C4\_OSR(osr);â



Setare numar biți la 8 si fara bit de paritate

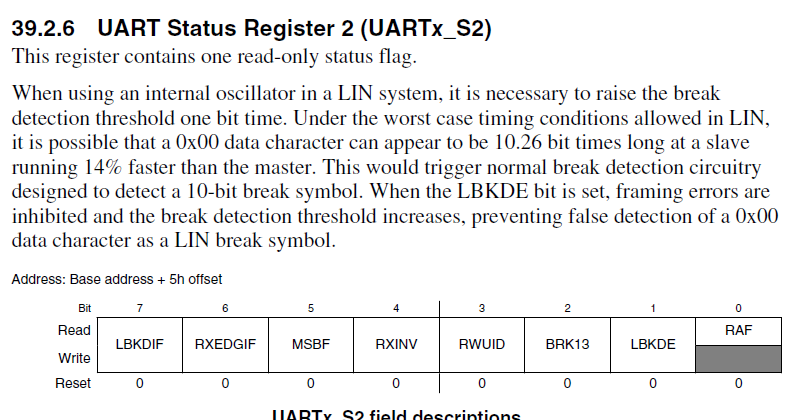
UART0->C1 = 0;

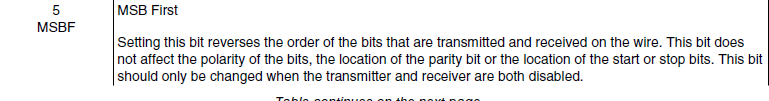


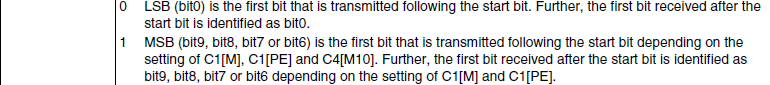


Setare LSB first.

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



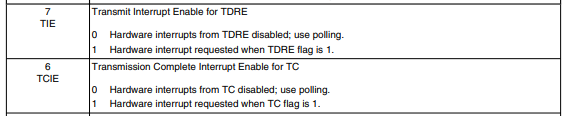




Dezactivăm întreruperile la transmisie.

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

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



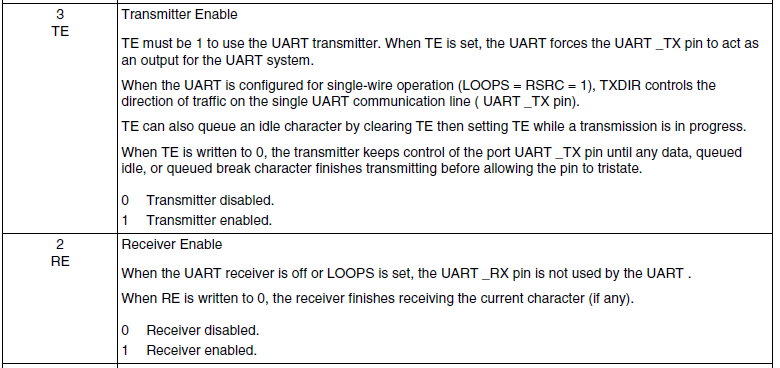
Activăm înapoi transmisia și recepția.

*/\*Activare intreruperi la receptie\*/*

    UART0->C2 |= UART0\_C2\_RIE(1);

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

    NVIC\_EnableIRQ(UART0\_IRQn);



Funcția UART0\_Trasmit() are rolul de a transmite serial caracterul transmis ca parametru, iar UART0\_Receive() primește flagurile de oprire/pornire aplicație și start/stop aplicație

*void* UART0\_Transmit(*uint8\_t* *data*)

{

*/\*Punem in asteptare pana cand registrul de transmisie a datelor nu este gol\*/*

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

        ;

    UART0->D = *data*;

}

*uint8\_t* UART0\_Receive(*void*)

{

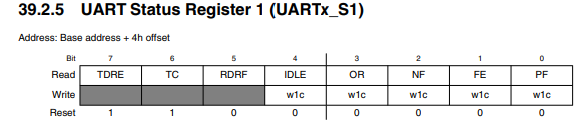
*/\*Punem in asteptare pana cand registrul de receptie nu este plin\*/*

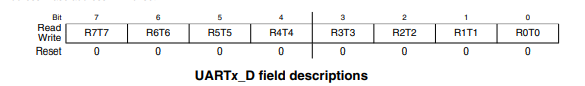
    while (!(UART0->S1 & UART0\_S1\_RDRF\_MASK))

        ;

    return UART0->D;

}





La activarea unei întreruperi pentru UART, se seteaza flagurile necesare funcționării aplicației.

*void* UART0\_IRQHandler(*void*)

{

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

    {

        c = UART0->D;

        if (c == 'S')

        {

            START\_flag = 1;

        }

        else if (c == 's')

        {

            START\_flag = 0;

        }

        else if (c == 'N')

        {

            DIRECTION\_flag = 1;

        }

        else if (c == 'R')

        {

            DIRECTION\_flag = 0;

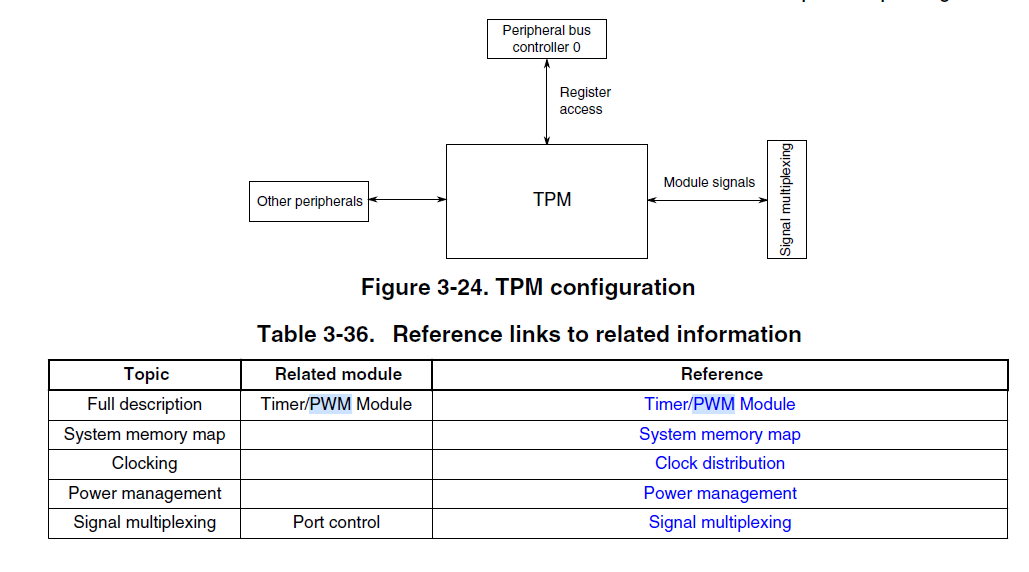
        }

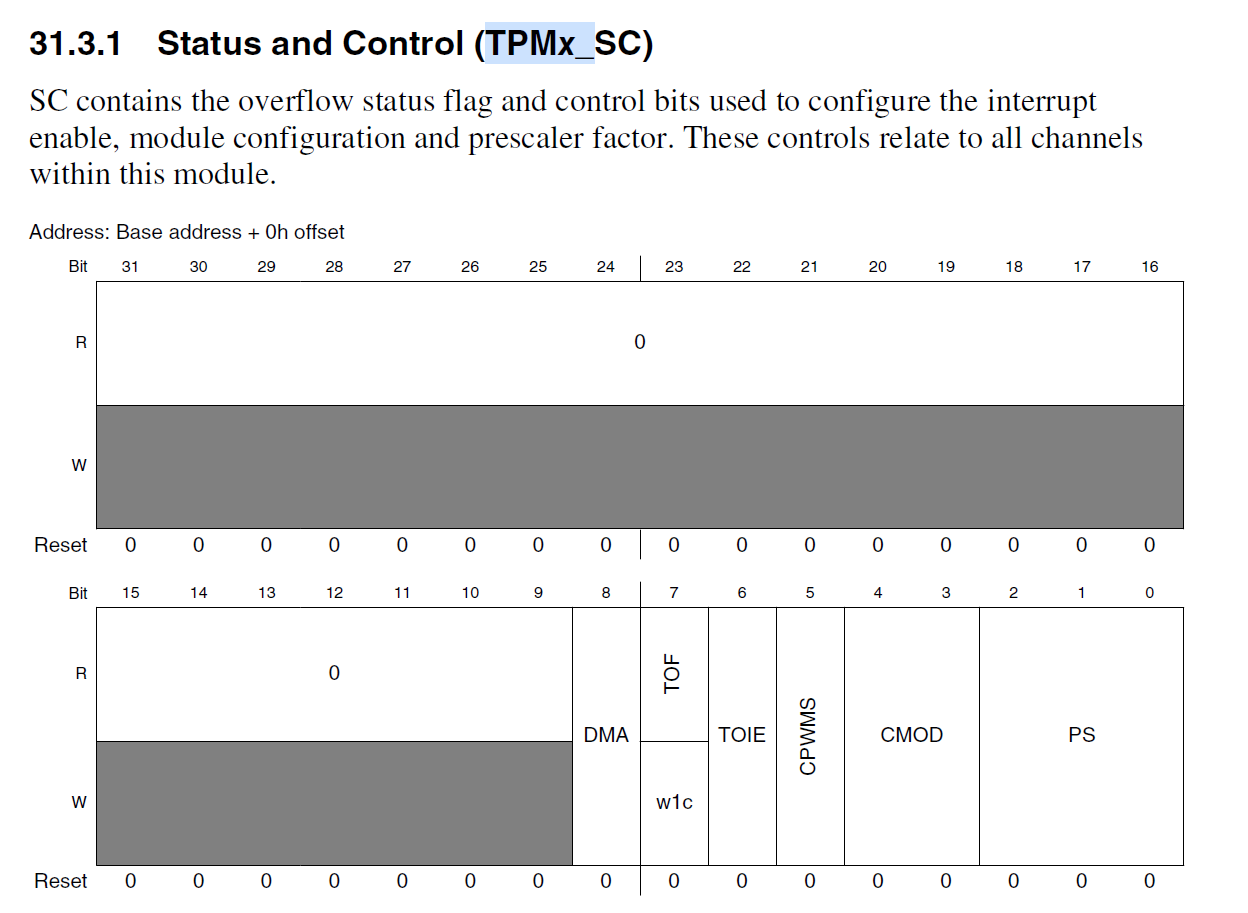
    }

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

}

* 1. **Modulul PWM**

****

****

*void* TPM2\_Init(*void*)

{

*/\* Activarea semnalului de ceas pentru utilizarea LED-ului de culoare rosie\*/*

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

*/\* Utilizarea alternativei de functionare pentru perifericul TMP  -> TMP2\_CH0\*/*

    PORTB->PCR[OSCILLOSCOPE\_PIN] |= PORT\_PCR\_MUX(3);

*/\* Selects the clock source for the TPM counter clock (MCGFLLCLK)  MCGFLLCLK Freq. - 48 MHz - PG. 196\*/*

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

*/\*Activarea semnalului de ceas pentru modulul TPM\*/*

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

*/\* Divizor de frecventa pentru ceasul de intrare\*/*

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

    TPM2->SC |= TPM\_SC\_PS(7);

*/\* Selects edge aligned PWM\*/*

    TPM2->SC |= TPM\_SC\_CPWMS(0);

*/\* LPTPM counter increments on every LPTPM counter clock\*/*

    TPM2->SC |= TPM\_SC\_CMOD(1);

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

}

Funcția **Signal\_Control()** controleaza unghiul de rotație în funcție de limite prestabilite raportate la valorile achiziționate de la senzorul de flacără.

*void* Signal\_Control(*uint8\_t* *value*)

{

*static* *uint8\_t* duty\_cycle = 0;

*static* *uint8\_t* increasing = 1; */\* 1 for increasing, 0 for decreasing\*/*

*/\* Reset the counter\*/*

    TPM2->CNT = 0x0000;

*// Set the PWM period (20 ms)*

    TPM2->MOD = 375 \* 20;

*/\* Extended the pulse width range slightly for full sweep\*/*

*uint32\_t* extended\_min\_pulse = 375 \* 0.2;

*uint32\_t* extended\_max\_pulse = 375 \* 2.0;

*uint32\_t* pulse\_range = extended\_max\_pulse - extended\_min\_pulse;

*uint8\_t* aux = (*uint8\_t*)avg; /\*value from TSI Sensor from the board \*/

    aux += 0x30;

*/\* Interpret value parameter to set the direction or stop\*/*

    switch (*value*)

    {

    case 0: */\* Slide the servo with TSI Sensor \*/*

        if (aux < 75)

            increasing = 1;

        else

            increasing = 0;

        break;

    case 1:

        increasing = 1;

        break;

    case 2:

        increasing = 0;

        break;

    case 3: */\*Toggle direction\*/*

        increasing = !increasing;

        break;

    }

*/\* Set the duty cycle for the servo\*/*

    TPM2->CONTROLS[0].CnV = extended\_min\_pulse + (pulse\_range \* duty\_cycle / 40);

*/\* Update duty\_cycle based on the direction\*/*

    if (increasing)

    {

        if (duty\_cycle < 20)

            duty\_cycle++;

    }

    else

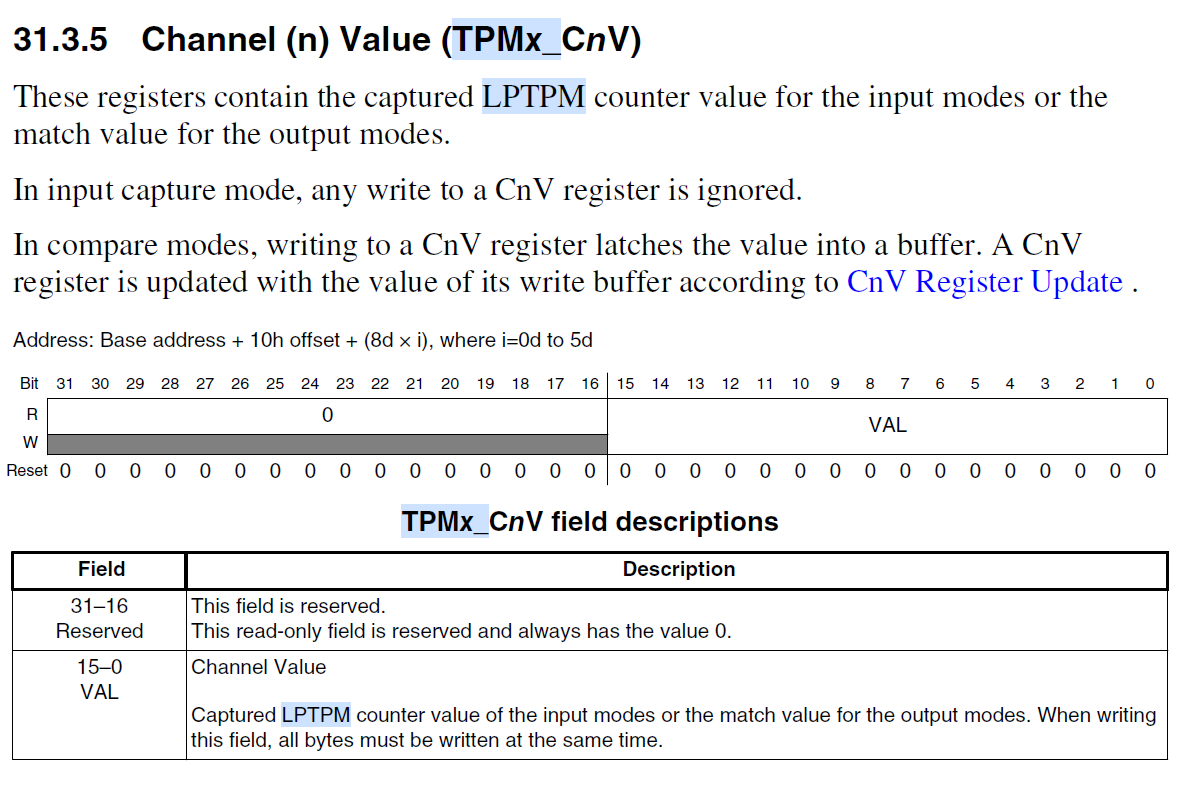
    {

        if (duty\_cycle > 0)

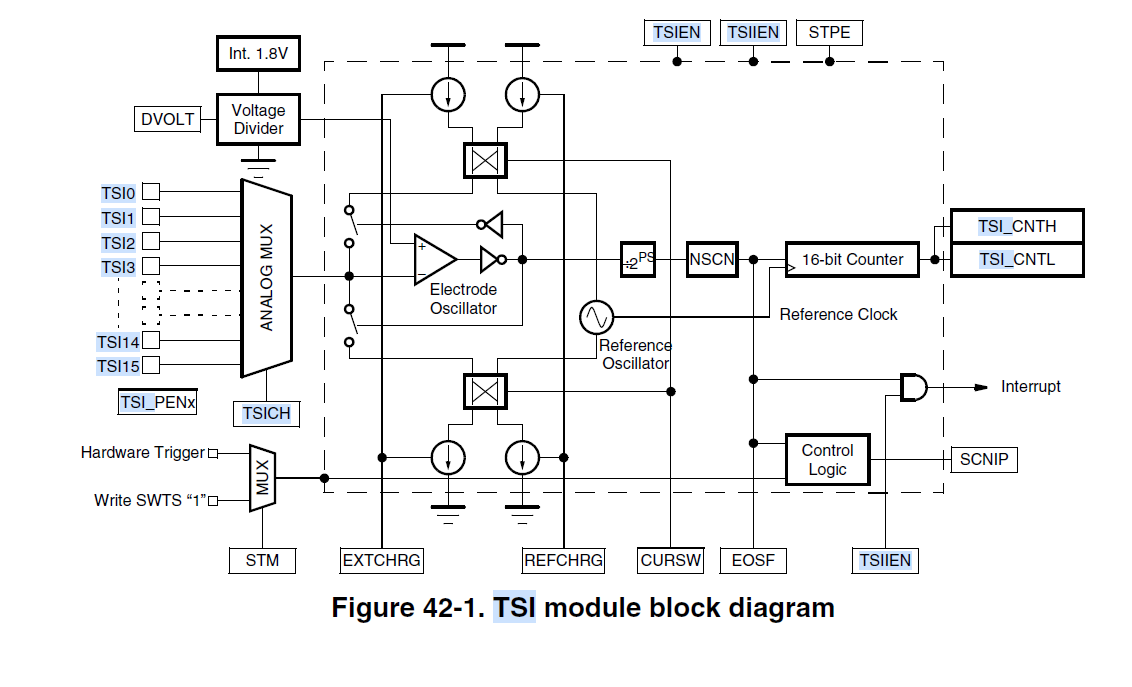
            duty\_cycle--;

    }

}

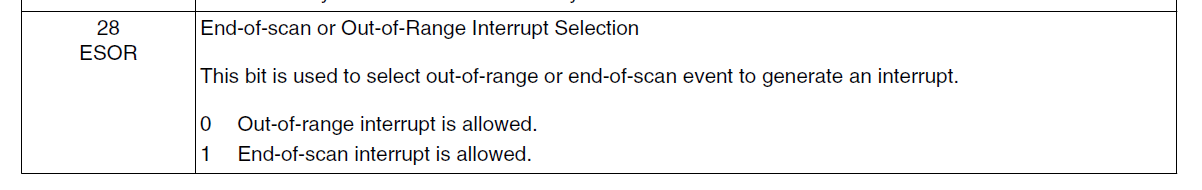


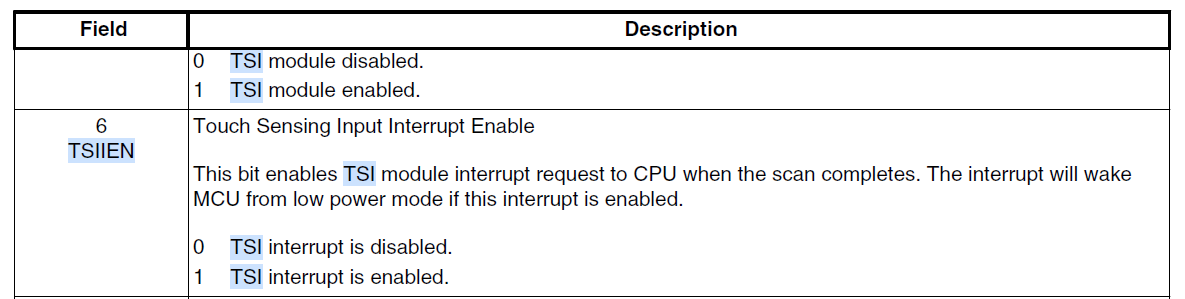
* 1. **Modulul TSI**

****

* **TSI\_init()**: Inițializează modulul TSI, activează ceasul pentru TSI și PortB, configurează modulul pentru a funcționa în modul de întrerupere la sfârșitul scanării și nu la depășirea unui interval, activează întreruperile TSI și începe prima scanare.
* **selectChannel(uint8\_t channel):** Selectează canalul TSI pentru scanare.
* **scan():** Pregătește modulul TSI pentru o nouă scanare și începe scanarea.
* **TSI\_update():** Verifică dacă o scanare este în progres și dacă nu, inițiază una nouă.
* **TSI0\_IRQHandler():** Rutina de întrerupere pentru TSI care este chemată când o scanare este completă. Aceasta citește valoarea scanată, resetează indicatorul de sfârșit de scanare, actualizează suma citirilor pentru a calcula o medie și reinițiază o nouă scanare după un mic întârziat.
* Variabilele globale **sum** și **count** sunt folosite pentru a calcula media valorilor citite de TSI la un număr specificat de eșantioane (SAMPLING\_RATE). Odată ce count atinge SAMPLING\_RATE, media este calculată și variabilele sunt resetate pentru următoarea serie de eșantioane.
* *void* TSI\_init()
* {
* */\* Enable TSI from SIM\*/*
* SIM->SCGC5 |= SIM\_SCGC5\_TSI\_MASK | *// Enable TSI*
* SIM\_SCGC5\_PORTB\_MASK; *// Enable PortB*
* */\*Config TSI\*/*
* TSI0->GENCS |= TSI\_GENCS\_TSIEN\_MASK | */\* Enable TSI\*/*
* TSI\_GENCS\_STPE\_MASK | */\* Enable TSI low power functioning ( just in case )\*/*
* TSI\_GENCS\_ESOR\_MASK; */\* End of scan interrupt ( not end of range )\*/*
* TSI0->GENCS |= TSI\_GENCS\_TSIIEN\_MASK; */\* enable tsi interrupts to enable DMA\*/*
* TSI0->GENCS &= ~TSI\_GENCS\_STM\_MASK; */\* Enable software interrupt mode\*/*
* NVIC\_EnableIRQ(TSI0\_IRQn);
* NVIC\_SetPriority(TSI0\_IRQn, 0); */\* Set priority to 0 ( highest )\*/*
* scan();
* }
* *void* TSI0\_IRQHandler()
* {
* */\*\* Check if the End of Scan flag is set \*/*
* if (((TSI0->GENCS & TSI\_GENCS\_EOSF\_MASK) >> TSI\_GENCS\_EOSF\_SHIFT) == 1)
* {
* */\*\* Read the touch sensor count from the TSI data register \*/*
* TSI\_Readings = (TSI0->DATA & TSI\_DATA\_TSICNT\_MASK) >> TSI\_DATA\_TSICNT\_SHIFT;
* */\*\* Clear the End of Scan flag \*/*
* TSI0->GENCS |= TSI\_GENCS\_EOSF\_MASK;
* */\*\* Add the current reading to the sum of readings \*/*
* sum += TSI\_Readings;
* */\*\* If we have reached the sampling rate, calculate the average \*/*
* if (count >= SAMPLING\_RATE)
* {
* */\*\* Calculate the average reading \*/*
* avg = sum / SAMPLING\_RATE;
* */\*\* Reset sum and count for the next round of sampling \*/*
* sum = 0;
* count = 0;
* }
* */\*\* Increment the count of readings taken \*/*
* count++;
* */\*\* Small delay loop \*/*
* *uint16\_t* i =0;
* for (i=0; i < 2500; i++)
* ;
* */\*\* Start a new scan \*/*
* scan();
* }
* }

Transmiterea prin DMA este posibilă doar când setarea TSI\_DATA[DMAEN] este activă și se inițiază atunci când anumite condiții din TSI\_GENCS sunt îndeplinite. Controlerul DMA de pe cip detectează această cerere și efectuează transferul de date între spațiul de memorie și spațiul de registru TSI, curățând automat TSI\_GENCS[EOSF] după transfer. Această funcție facilitează obținerea rezultatelor de la TSI și pregătirea pentru următoarea scanare.





1. **Prezentarea interfeței grafice**

****

*Figura 11. Interfața grafică realizată în python*

* Membrii echipei
* Start button – pornește aplicația
* Normal/Reverse button – inversează ordinea secvenței de led-uri
* Stop button – pune aplicația pe pauză
* Sensor value received – valorile achiziționate de la senzorul de flacără
* Environment state message – mesaj în funcție de valori
* Point plot
* Bar plot

1. **Probleme întâmpinate**

* Valorile prezentate in specificațiile servomotorului au fost ajustate experimental (prin încercări) pentru a ajunge la rezultatele dorite.
* Inițial, servomotorul era configurat pe pinul PTA1, iar ulterior am avut nevoie de activarea recepției in cadrul modulului UART, pentru a comunica cu interfața grafică, iar din neatenție am configurat recepția pe același pin. Ne-am confruntat cu unele dificultăți din cauza acestei situații, fiindcă a fost destul de complicat să determinăm sursa problemei.

Rezolvare: am configurat sermotorul pe pinul PTB2

Per total, proiectul s-a desfășurat cu succes, depășind provocările inițiale și atingând obiectivele stabilite eficient.

1. Referințe

* Secțiuni preluate din **KL25 Sub-Family Reference Manual, Rev. 3, September 2012, Freescale Semiconductor, Inc.**

1. DEMO Link: <https://drive.google.com/drive/folders/1WWKEkDLvVvDiV6fwfB6bgmav6WKV5YXh?usp=sharing> aici se regăsește și environment-ul de python, în arhiva venv-up-all.zip
2. GitHub Repo: https://github.com/PabL073/Proiect-Microprocesoare/tree/main

1. FRDM-KL25Z User's Manual Freescale Semiconductor, Inc. [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)
3. FRDM-KL25Z User's Manual Freescale Semiconductor, Inc. [↑](#footnote-ref-3)