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 LUMINĂ AMBIENTALĂ

PLATFORMA DE DEZVOLTARE FRDM-KL25Z

Std. sg. maj. Simona Mihaela CĂTĂNOIU
Std. sg. maj. Andreea Loredana DUMITRAȘCU
Grupa C114D

București

2023-2024

Cuprins

1 Prezentarea senzorului PT550

Senzorul DFR0026 este un senzor analog ce transmite informații despre luminozitate către controllerul printr-un semnal de tensiune analogică, având o gamă extinsă de la 1 Lux la 6000 Lux.

Figura 1. Senzorul PT550

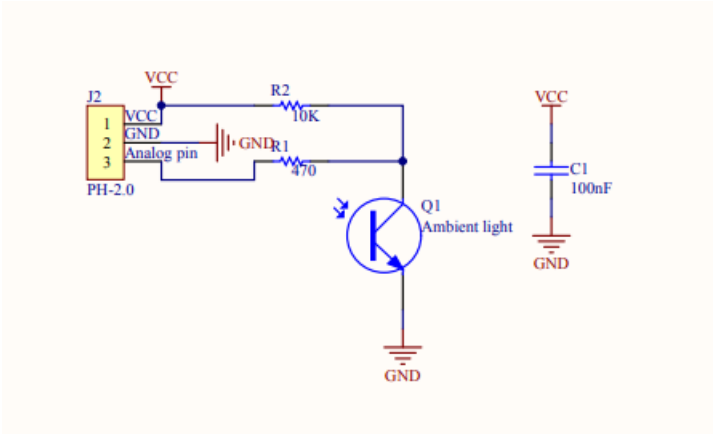


Figura 2. Circuitul senzorului PT550

Senzorul include un condensator de 100 nF, un tranzistor și două rezistențe (una de 10KΩ pe pinul de alimentare și una de 470Ω pe pinul de semnal analogic). Acesta se conectează la placa de dezvoltare prin trei conectori: unul negru (GND), unul roșu (3.3V-5V) și unul albastru (OUTPUT).

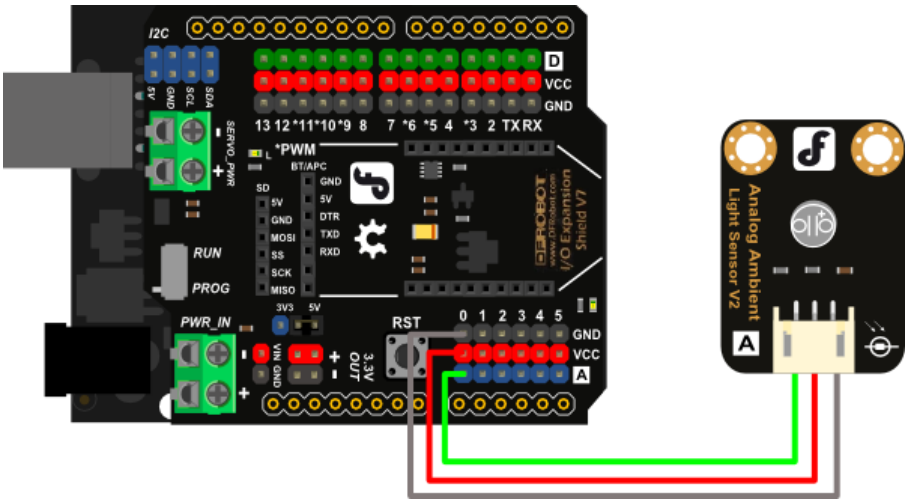


Figura 3. Exemplu conectare senzor

2 Prezentarea servomotorului SG90 9g

Servomotorul SG90 9g este un servomotor compact și ușor, dar cu o putere de ieșire ridicată. Servomotorul este un tip special de motor care poate fi comandat să se rotească într-un interval de aproximativ 180 de grade, realizând pozitionarea precisă a elicei la un anumit unghi și menținută până la primirea unui nou semnal. El este constituit dintr-un motor DC, diferite angrenaje pentru a crește cuplul (și a reduce viteza) și un sistem de servomotor.

Roțile sunt din nylon. Tensiunea de alimentare este în intevalul 4.8 si 6V, iar viteza nominală de rotație este de 0.1s/60˚. Motorul are un angrenaj care reduce viteza de rotație a motorului prin scăderea turațiilor pe minut (RPM) și creșterea cuplului.

Poziția motorului este controlată ajustând lățimea impulsului semnalului de intrare, de obicei prin utilizarea unui semnal PWM. Acest motor necesită un controller pentru a regla impulsul, cum ar fi Arduino sau FRDM-KL25z. Semnalul PWM produs ar trebui să aibă o frecvență de 50 Hz, adică perioada PWM să fie de 20 ms. Din aceasta, timpul activ (duty cycle-ul) poate varia de la 1 ms la 2 ms. Astfel, atunci când timpul activ este de 1 ms, motorul va fi la 0°, iar când este de 1,5 ms, motorul va fi la 90°; similar, când este de 2 ms, motorul va fi la 180°. Prin varierea timpului activ de la 1 ms la 2 ms, motorul poate fi controlat de la 0° la 180°. Direcțiile motorului servo sunt transmise de microcontroler sub forma unor impulsuri PWM.

Servomotorul se conectează la placa de dezvoltare prin trei conectori: unul negru (GND), unul roșu (5V) și unul galben (control).

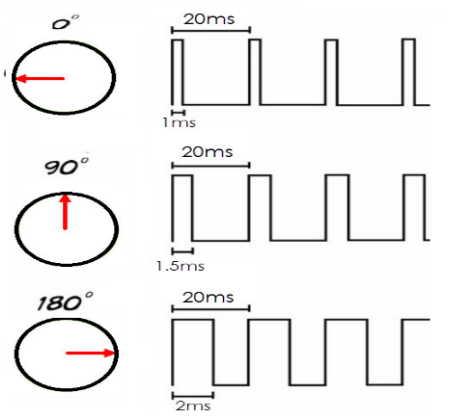
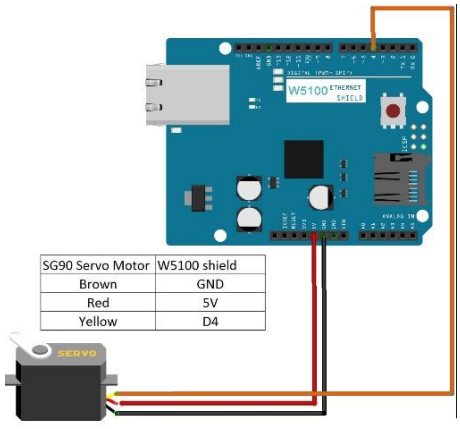


Figura 4. Mod de rotație SG90

Figura 5. Servomotorul SG90



SG90 Servo Motor	W5100 shield
Brown	GND
Red	5V
Yellow	D4

Figura 6. Exemplu conectare servomotor

3 Scop proiect

Scopul acestui proiect constă în ajustarea vitezei de rotație a elicei servomotorului, având la bază valorile obținute de la senzorul de lumină ambientală conectat la placa de dezvoltare. Prin intermediul unui semnal PWM generat în funcție de nivelul de luminozitate detectat de senzor, se realizează controlul motorului. Programul va permite și utilizarea LED-urilor de pe plăcuță, implementând o secvență de culori care se va schimba la un interval de 275ms.

Programul dezvoltat va efectua verificări ale ieșirii senzorului conectat la fiecare milisecundă, cu o frecvență de 1 kHz. Pentru fiecare ieșire, se va genera un semnal PWM. Factorul de umplere al acestui semnal va fi direct proporțional cu valoarea detectată de senzor, împărțită pe niveluri de lumină. Semnalul PWM rezultat va fi transmis către servomotor, ajustând astfel unghiul de rotație al elicei în concordanță cu factorul de umplere.

De asemenea, se va transmite prin protocolul UART către PC valoarea ieșirii senzorului de lumină ambientală. Această informație va fi utilizată pentru actualizarea unui GUI (interfață grafică utilizator) dezvoltat în Python. GUI-ul va oferi o reprezentare vizuală a nivelului de lumină detectat și va permite monitorizarea în timp real a ajustărilor vitezei servomotorului prin intermediul unei palete de culori.

În plus, interfața va dispune de un buton care să permită modificarea secvenței de afișare a culorilor LED-ului, prin intermediul modulului UART. Pentru a putea observa realizarea cu succes a acestei modificări, un mesaj sugestiv este transmis de către microcontroller și apoi afișat în interfață. Pentru schimbarea culorilor la o anumită perioadă de timp, se va folosi perifericul PIT.

4 Conectare senzor-placă de dezvoltare

Vom realiza conexiunile senzorului în următorul mod:

- Firul albastru-portocaliu va fi conectat de la senzor la pinul PTB3.
- Firul roșu-rosu va fi conectat de la senzor la pinul de alimentare 3.3V.
- Firul negru-negru va fi conectat de la senzor la pinul de masă (GND).

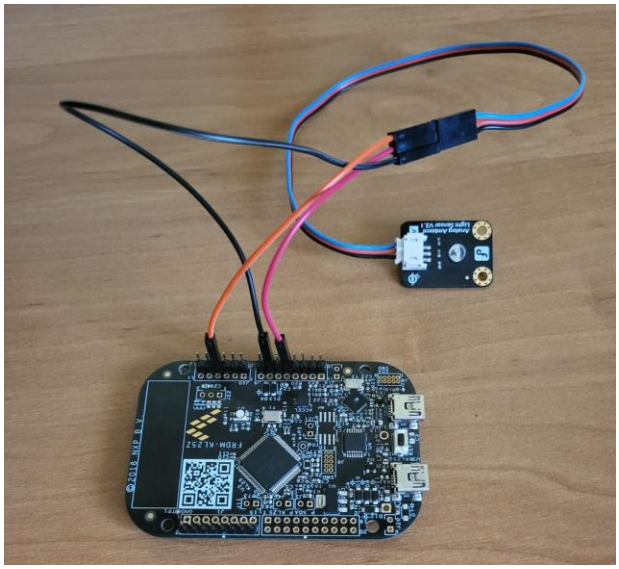


Figura 7. Conectare senzor

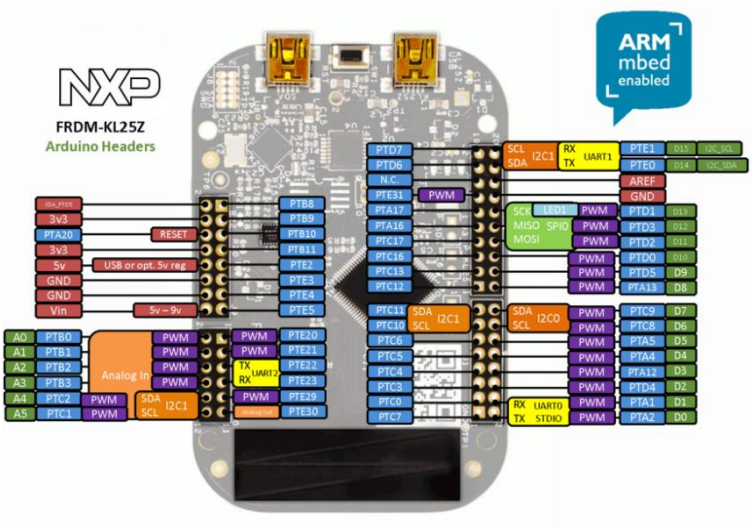


Figura 8. Pini placă dezvoltare

Conectare la placa de dezvoltare se realizează astfel:

- Fir maro - maro se conectează la pinul GND de pe placă
- Fir roșu - portocaliu se conectează la pinul 5V de pe placă
- Fir galben - galben se conectează la ieșirea digitală de pe placă (în cazul nostru PTB2)

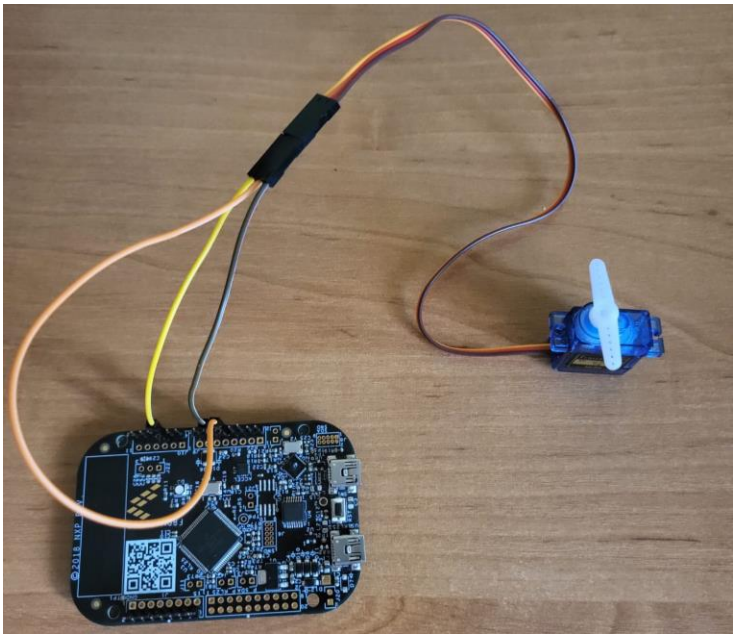


Figura 9. Conectare servomotor-placă de dezvoltare

Pentru a exemplifica mai bine modul de conectare, am realizat următoarea schemă în Tinkercad, cu precizarea că am folosit un senzor de temperatură pentru că pe cel de lumină nu

l-am găsit, iar ca plăcuță am folosit Arduino, dar am respectat cablarea de pe plăcuța noastră (A2 și A3 sunt corespunzători B2 și B3):

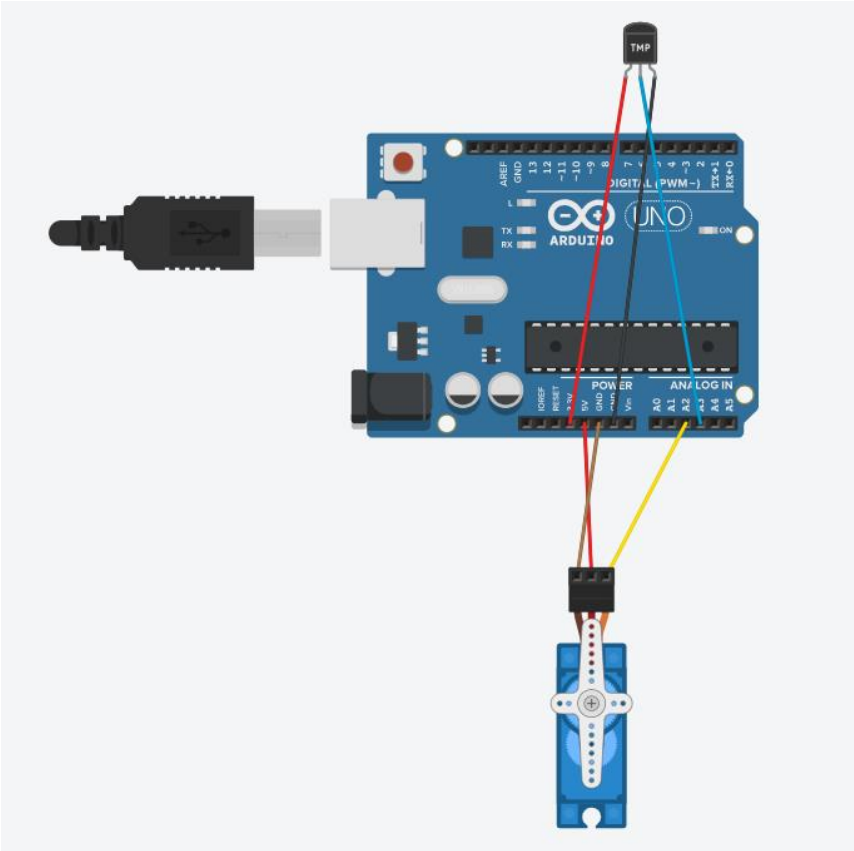


Figura 10. Schema tinkercad de conectare a perifericelor la plăcuță

5 Descriere program

5.1 Schema bloc a programului

În diagrama de mai jos, am realizat o reprezentare bloc a programului nostru, evidențiind două fluxuri principale. Aceste fluxuri ilustrează modul în care care modulele interacționează între ele pentru a asigura funcționarea armonioasă a întregului sistem.

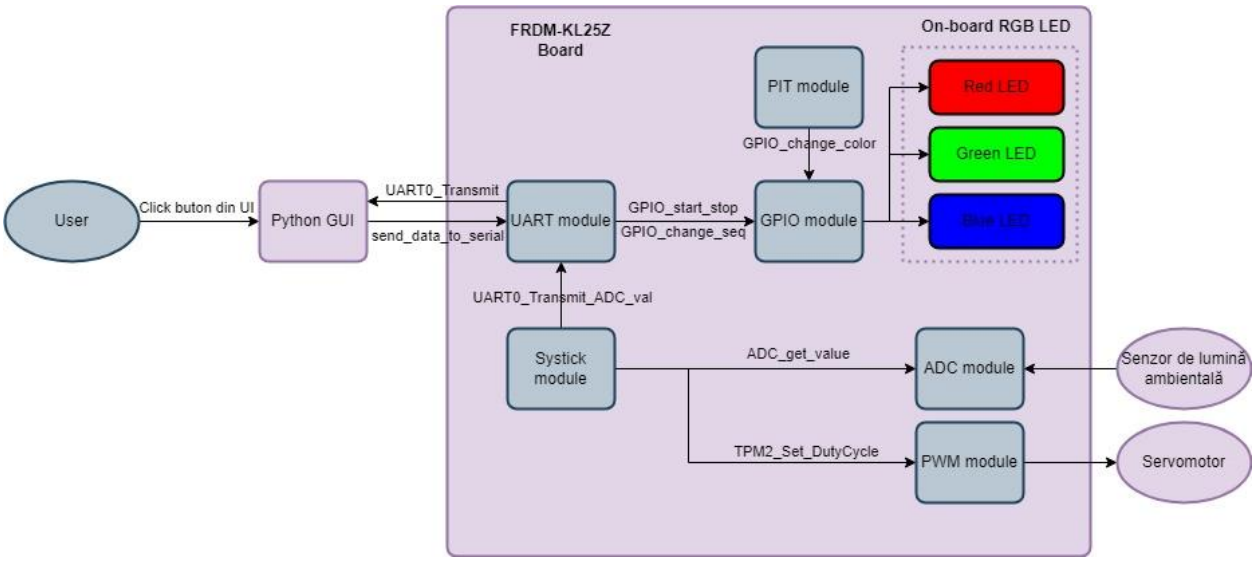


Figura 11. Schema bloc a proiectului

Primul flux este asociat modulului GPIO pentru activarea LED-ului RGB și schimbarea culorii la o anumită perioadă de timp gestionată de modulul PIT prin apelarea funcției GPIO_change_color, care aduce modificări în culoarea luminii emise de LED..

Pe de altă parte, al doilea flux este legat de modulul systick, care interacționează ulterior cu modulele ADC și PWM. Aici, systick acționează ca un cronometru, coordonând interacțiunea dintre modulul ADC (convertor analog-digital) și modulul PWM (modulație de lățime a impulsului). Datele obținute din conversia ADC sunt utilizate pentru a controla parametrii modulației PWM, influențând astfel unghiul de rotație al elicei servomotorului.

Ambele fluxuri comunică cu modulul UART, canalul de comunicație prin care primul flux primește comenzi de la utilizator, acționate de butoanele din interfața grafică, iar al doilea flux trimite date către interfața grafică pentru a fi afișate într-un grafic. Această interconectare între modulele și fluxurile programului asigură o funcționare integrată și sincronizată a întregului sistem.

5.2 Funcția main

În cadrul fișierului main.c, am inclus fișierele header care conțin declarațiile funcțiilor ce urmează să fie utilizate. Acestea includ: *uart.h* (cu funcțiile UART0_Initialize, UART0_Transmit și UART0_Transmit_ADC_val), *gpio.h* (cu funcția GPIO_Init, GPIO_change_color, GPIO_change_seq), *pit.h* (cu funcția PIT_Init), *Adc.h* (cu funcțiile ADC0_Init și ADC_get_value), *ClockSettings.h* (cu funcția SystemClockTick_Configure și variabila flag_1s), *Pwm.h* (cu funcțiile TPM2_Init și TPM2_Set_DutyCycle) și fișierul header generat de mediul de dezvoltare Keil, MKL25Z4.h, specific plăcii de dezvoltare.

```
#include "uart.h"
#include "gpio.h"
#include "pit.h"
#include "Adc.h"
#include "ClockSettings.h"
#include "Pwm.h"
#define BAUD_RATE 9600
```

```
int main(void) {
          GPIO_Init();
          PIT_Init();
          SystemClockTick_Configure();
          UART0_Initialize(BAUD_RATE);
          ADC0_Init();
          TPM2_Init();

          for(;;){
            if(flag_1s){
              uint16_t val = ADC_get_value();
              UART0_Transmit_ADC_val(val);
              TPM2_Set_DutyCycle(val);
              flag_1s = 0U;
            }
          }
}
```

Figura 12. Funcția main

Logica centrală a programului este structurată după cum urmează: se efectuează pe rând inițializările necesare pentru fiecare modul (GPIO_Init pentru GPIO, PIT_Init pentru PIT, SystemClockTick_Configure pentru SysTick, UART0_Initialize pentru UART, ADC0_Init pentru ADC și TMP2_Init pentru PWM). Ulterior, într-un ciclu infinit, programul verifică dacă variabila flag_1s a fost modificată de la valoarea 0 (indicând că au avut loc suficiente întreruperi SysTick). În caz afirmativ, se preia valoarea de la ADC, se transmite mai departe prin UART, se actualizează configurarea PWM conform noii valori citite, iar apoi se reinițializează variabila flag_1s la 0. Acest lucru este reprezentat de următoarea diagramă de stări:

5.2.1 Diagramă de stări funcția main

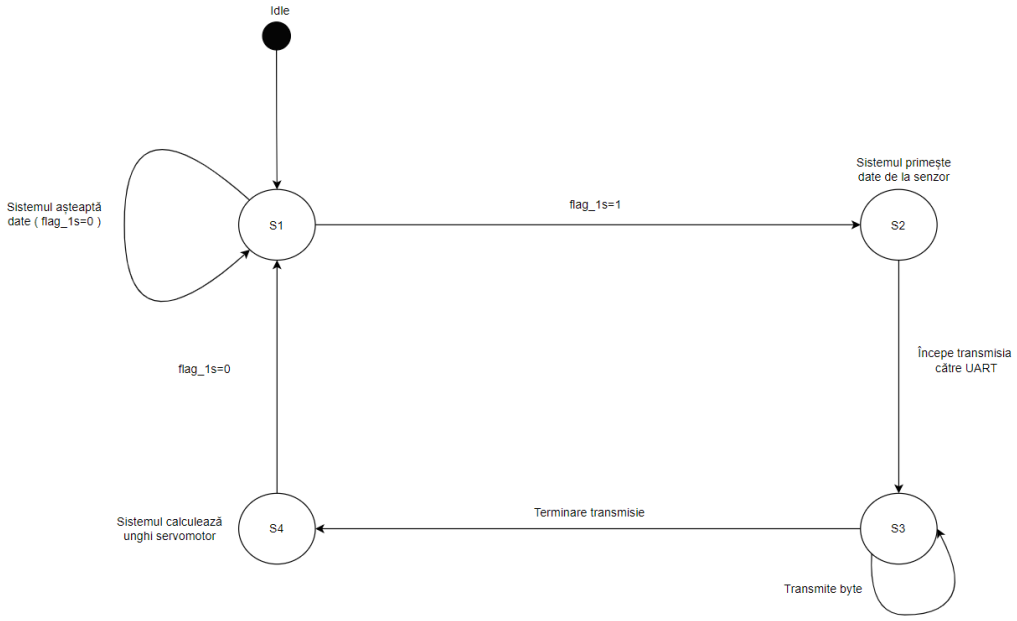


Figura 23 Diagramă de stări program program principal

5.2.2 Inițializarea modulului GPIO

```
void GPIO_Init(void)
{
          SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK;
          SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK;

          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);

          GPIOB->PDDR|= (1<<RED_LED_PIN);
          GPIOB->PDDR|= (1<<GREEN_LED_PIN);
          GPIOD->PDDR|= (1<<BLUE_LED_PIN);

          GPIOB->PSOR|=(1<<RED_LED_PIN);
          GPIOB->PSOR|=(1<<GREEN_LED_PIN);
          GPIOD->PSOR|=(1<<BLUE_LED_PIN);

          state=1;
}
```

În cadrul registrului SIM_SCGC5 (System Clock Gating Control Register 5), am configurat setarea biților corespunzători porturilor B și D (deoarece ledurile roșu și verde sunt aferente protului B, iar cel albastru portului D), folosind operația de „sau”.

Prin această operație, am stabilit valorile biților relevanți la 1. Această acțiune are ca rezultat activarea ceasului (clock) pentru porturile B și D în cadrul microcontroller-ului. Activarea acestor ceasuri este esențială pentru a permite utilizarea funcționalităților oferite de aceste porturi în cadrul programului.

12.2.9 System Clock Gating Control Register 5 (SIM_SCGC5)

Address: 4004_7000h base + 1038h offset = 4004_8038h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0												0	0		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0		PORTE	PORTD	PORTC	PORTB	PORTA		1	0	TSI		0		0	LPTMR
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

12 PORTD	Port D Clock Gate Control This bit controls the clock gate to the Port D module. 0 Clock disabled 1 Clock enabled

10 PORTB	Port B Clock Gate Control This bit controls the clock gate to the Port B module.
	0 Clock disabled 1 Clock enabled

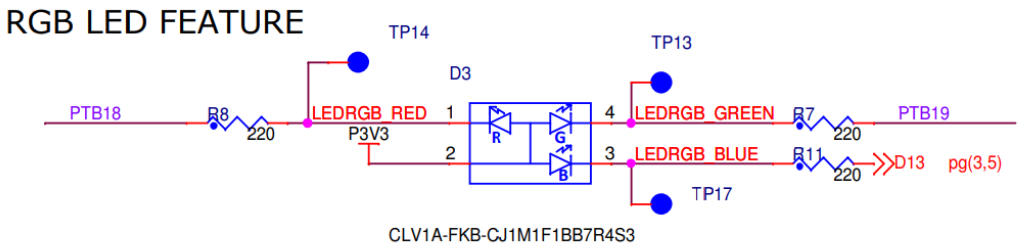
Macro-urile RED_LED_PIN, GREEN_LED_PIN și BLUE_LED_PIN sunt definite în headerul gpio.h.

```
#define RED_LED_PIN (18) //PORT B PIN 18
#define GREEN_LED_PIN (19) //PORT B PIN 19
#define BLUE_LED_PIN (1) //PORT D PIN 1
```

Și au fost alese conform tabelei de pinouts a plăcuței

On-board Usage	I/O Header & Pin Num	Arduino™ R3 Pin Name	FRDM-KL25Z Pin Name
Red LED	–	–	PTB18
Green LED	–	–	PTB19
Blue LED	J2 12	D13	PTD1

Circuit FRDM-KL25z pentru LED-urile RGB:



Urmează configurarea pinilor pentru LED-uri, pe rând, pentru fiecare canal de culoare. Astfel pentru led-ul roșu (RED_LED_PIN) se realizează următoarele:

În regiștrii de control ai ledului roșu (PORTB->PCR[RED_LED_PIN]) setăm cu 0 biții corespunzători multiplexorului pentru pinul RED_LED_PIN, deoarece ei nu au o valoare de resetare stabilă („undefined at reset”). Setăm apoi multiplexorul pentru pinul RED_LED_PIN la valoarea 1, stabilind alternativa de funcționare specifică modului GPIO.

Address: Base address + 0h offset + (4d × i), where i=0d to 31d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0					MUX			0	DSE	0	PFE	0	SRE	PE	PS
W																
Reset	0	0	0	0	0	x*	x*	x*	0	x*	0	x*	0	x*	x*	x*

* Notes:
- x = Undefined at reset.

Aceeași logică este aplicată și pentru LED-ul verde (GREEN_LED_PIN) și LED-ul albastru (BLUE_LED_PIN).

10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific). 101 Alternative 5 (chip-specific). 110 Alternative 6 (chip-specific). 111 Alternative 7 (chip-specific).

Apoi prin punerea de biți de 1 în regiștrii GPIOB_PDDR și GPIOD_PDDR pe pozițiile aferente ledurilor noastre(și anume 18 pentru roșu, 19 pentru verde și 1 pentru albastru), setăm direcția pinilor RED_LED_PIN, GREEN_LED_PIN și BLUE_LED_PIN ca fiind de ieșire (output), permițând astfel controlul lor.

41.2.6 Port Data Direction Register (GPIOx_PDDR)

The PDDR configures the individual port pins for input or output.

Address: Base address + 14h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PDD																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIOx_PDDR field descriptions

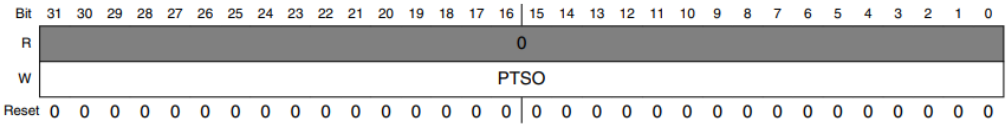
Field	Description
31–0 PDD	Port Data Direction Configures individual port pins for input or output. 0 Pin is configured as general-purpose input, for the GPIO function. 1 Pin is configured as general-purpose output, for the GPIO function.

Prin setarea biților aferenți ledurilor noastre din regiștrii GPIOB_PSOR și GPIOD_PSOR cu 1, practic facem ca ledurile noastre să se afle inițial într-o stare de oprire. (conform circuitului prezentat mai devreme, led-urile de pe plăcuța FRDMKL25z se opresc pentru valoarea 1 logic deoarece sunt conectate la 3.3V);

41.2.2 Port Set Output Register (GPIOx_PSOR)

This register configures whether to set the fields of the PDOR.

Address: Base address + 4h offset



GPIOx_PSOR field descriptions

Field	Description
31–0 PTSO	Port Set Output Writing to this register will update the contents of the corresponding bit in the PDOR as follows: 0 Corresponding bit in PDORn does not change. 1 Corresponding bit in PDORn is set to logic 1.

5.2.3 Inițializarea modulului PIT

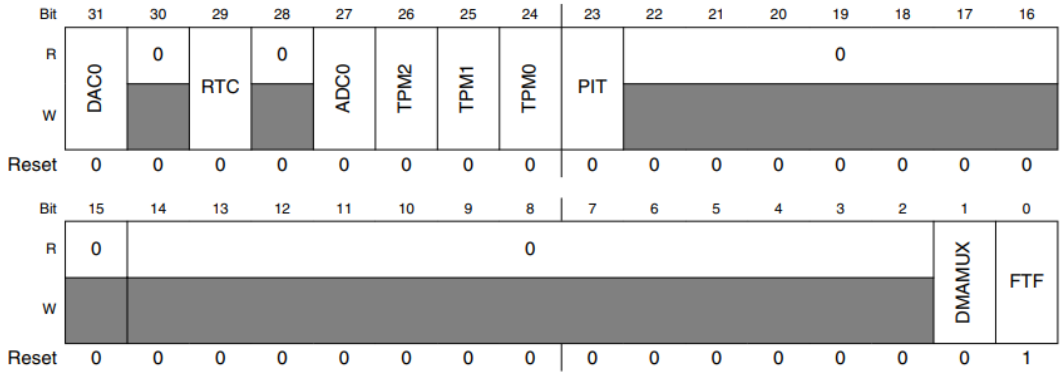
Vom folosi modulul PIT pentru a genera întreruperi periodice pe canalul 0 și pregătim astfel sistemul pentru a gestiona evenimentele asociate acestor întreruperi.

```
void PIT_Init(void) {

          SIM->SCGC6 |= SIM_SCGC6_PIT_MASK;
          PIT_MCR &= ~PIT_MCR_MDIS_MASK;
          PIT->MCR |= PIT_MCR_FRZ_MASK;

          // Setarea valoarea numaratorului de pe canalul 0 la o perioada de 275 ms
          // 0.275*10485760 -1 = 2883583 (2B FFFF)
          PIT->CHANNEL[0].LDVAL = 0x2BFFFF;
          PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TIE_MASK;
          PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TEN_MASK;

          NVIC_ClearPendingIRQ(PIT_IRQn);
          NVIC_SetPriority(PIT_IRQn,1);
          NVIC_EnableIRQ(PIT_IRQn);
}
```

Inițializarea se realizează prin setarea bitului corespunzător pentru modulul PIT în registrul SIM_SCGC6. Această acțiune este necesară pentru a permite utilizarea funcționalităților oferite de modulul PIT.

12.2.10 System Clock Gating Control Register 6 (SIM_SCGC6)

Address: 4004_7000h base + 103Ch offset = 4004_803Ch



Următoarele modificări sunt aduse la nivelul registrului PIT_MCR (PIT Module Control Register). Aici setăm câmpul MDIS cu 0 pentru a ne asigura că modulul PIT este activat și pregătit să primească și să gestioneze întreruperi, iar câmpul FRZ primește valoarea 1 pentru a ne asigura funcționarea corectă a modulului PIT chiar și în modul de depanare.

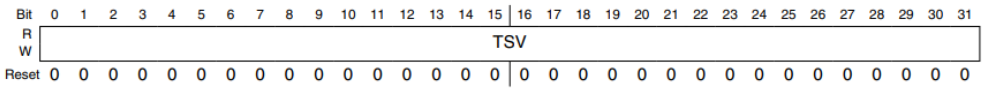
30 MDIS	Module Disable - (PIT section) Disables the standard timers. The RTI timer is not affected by this field. This field must be enabled before any other setup is done. 0 Clock for standard PIT timers is enabled. 1 Clock for standard PIT timers is disabled.
31 FRZ	Freeze Allows the timers to be stopped when the device enters the Debug mode. 0 Timers continue to run in Debug mode. 1 Timers are stopped in Debug mode.

Prin modificarea registrului LDVAL se setează practic valoarea de încărcare a numărătorului de pe canalul 0 la o perioadă de 275 ms. Valoarea este calculată în funcție de frecvența de ceas a modulului și perioada dorită. În acest caz, valoarea dorită este 0x2BFFFF.

32.3.4 Timer Load Value Register (PIT_LDVAL*n*)

These registers select the timeout period for the timer interrupts.

Address: 4003_7000h base + 100h offset + (16d × i), where i=0d to 1d



PIT_LDVAL*n* field descriptions

Field	Description
0–31 TSV	Timer Start Value Sets the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer; instead the value will be loaded after the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again.

Dorim să configurăm ultimii doi biți ai registrului TCTRL (Timer Control Register) cu valoarea 1. Prin setarea câmpului TIE la 1, se activează întreruperea pentru canalul 0. Acest lucru permite generarea unei întreruperi periodice atunci când numărătorul atinge valoarea de încărcare. Prin setarea câmpului TEN la 1, se activează efectiv numărătorul, inițiind procesul de numărare în

32.3.6 Timer Control Register (PIT_TCTRL*n*)

These register contain the control bits for each timer.

Address: 4003_7000h base + 108h offset + (16d × i), where i=0d to 1d

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0													CHN	TIE	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

jos de la valoarea de încărcare până la 0.

30 TIE	Timer Interrupt Enable When an interrupt is pending, or, TFLGn[TIF] is set, enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TFLGn[TIF] must be cleared first. 0 Interrupt requests from Timer n are disabled. 1 Interrupt will be requested whenever TIF is set.
31 TEN	Timer Enable Enables or disables the timer. 0 Timer n is disabled. 1 Timer n is enabled.

Ultimele trei configurări au loc la nivelul registrului NVIC (Nested Vector Interrupt Controller). Acest controler gestionat de hardware facilitează administrarea și controlul întreruperilor în sistemului nostru.

„NVIC_ClearPendingIRQ(PIT_IRQn);” asigură că starea de așteptare pentru întreruperea asociată modulului PIT este curățată în cadrul NVIC pentru a evita orice întârziere nedorită în activarea întreruperilor și pentru a garanta un răspuns corespunzător la evenimentele care generează aceste întreruperi.

„NVIC_SetPriority(PIT_IRQn,1);”, constă în stabilirea priorității pentru întreruperea asociată modulului PIT în cadrul NVIC cu 1. Acest aspect devine semnificativ în situațiile în care mai multe întreruperi pot concura pentru execuție, permițând sistemului să prioritizeze și să gestioneze aceste evenimente în funcție de importanța lor relativă.

„NVIC_EnableIRQ(PIT_IRQn);” activează întreruperea asociată modulului PIT în cadrul NVIC, conferind astfel permisiunea procesorului de a răspunde la aceste întreruperi. Această activare este crucială pentru ca procesorul să poată executa rutina de tratare a întreruperii atunci

când evenimentele asociate modulului PIT apar, asigurând o sincronizare eficientă între hardware și software în cadrul sistemului.

5.2.4 Inițializarea modulului SysTick

În această funcție vom configura și inițializa modulul SysTick pentru a genera semnale periodice care vor semnala trecerea unei secunde prin intermediul varibilei externe flag_1s.

```
void SystemClockTick_Configure(void){

          SysTick->LOAD = (uint32_t)(48000000UL / 1000 - 1UL);

          NVIC_SetPriority(SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL);

          SysTick->VAL = 0UL;

          SysTick->CTRL |= (SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk |

          SysTick_CTRL_ENABLE_Msk);
}
```

Prin „SysTick->LOAD = (uint32_t)(48000000UL / 1000 - 1UL);” setăm valoarea de încărcare a numărătorului SysTick. În cazul nostru, am ales valoarea astfel încât să se genereze un tick la fiecare 1 ms. Valoarea încărcată este calculată pe baza frecvenței de ceas a sistemului (48 MHz).

Prin NVIC_SetPriority stabilim prioritatea pentru întreruperea asociată numărătorului SysTick. Prioritatea este setată la valoarea maximă posibilă, adică la (1UL << __NVIC_PRIO_BITS) - 1UL. Această acțiune asigură că întreruperea SysTick are cea mai înaltă prioritate în sistem.

„SysTick->VAL = 0UL;” resetează numărătorul digital al numărătorului SysTick la 0. Acest pas este important pentru a asigura o valoare inițială corectă a numărătorului nostru.

În final, configurăm registrul de control al numărătorului SysTick (SysTick->CTRL) prin aplicarea operațiilor de "sau" și setarea la 1 a unor biți specifici conform măștilor definite pentru:

- SysTick_CTRL_CLKSOURCE_Msk: Selectează ceasul procesorului ca sursă pentru numărătorul SysTick.
- SysTick_CTRL_TICKINT_Msk: Activează întreruperea SysTick atunci când numărătorul atinge valoarea 0.
- SysTick_CTRL_ENABLE_Msk: Activează numărătorul SysTick, încărcând valoarea din registrul LOAD și începând numărătoarea propriu-zisă. Această etapă finală completează configurarea modulului SysTick, pregătindu-l pentru a genera temporizări precise și pentru a răspunde la evenimentele temporale definite în sistemul încorporat.

[2]	CLKSOURCE	Selects the SysTick timer clock source: 0 = external reference clock. 1 = processor clock.
[1]	TICKINT	Enables SysTick exception request: 0 = counting down to zero does not assert the SysTick exception request. 1 = counting down to zero asserts the SysTick exception request.
[0]	ENABLE	Enables the counter: 0 = counter disabled. 1 = counter enabled.

Referință: https://developer.arm.com/documentation/dui0662/b/Cortex-M0--Peripherals/System-timer--SysTick/SysTick-Control-and-Status-Register

5.2.5 Inițializarea modulului UART

Vom utiliza modulul UART0 pentru a realiza comunicarea serială cu un PC prin intermediul cablului USB.

```
void UART0_Initialize(uint32_t baud_rate) {

          uint16_t sbr;

          SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;
          SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;

          UART0->C2 &= ~((UART0_C2_RE_MASK) | (UART0_C2_TE_MASK));
          SIM->SOPT2 |= SIM_SOPT2_UART0SRC(01);

          PORTA->PCR[1] = ~PORT_PCR_MUX_MASK;
          PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2);
          PORTA->PCR[2] = ~PORT_PCR_MUX_MASK;
          PORTA->PCR[2] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2);

          sbr = (uint16_t)((DEFAULT_SYSTEM_CLOCK)/(baud_rate * (osr)));
          UART0->BDH &= UART0_BDH_SBR_MASK;
          UART0->BDH |= UART0_BDH_SBR(sbr>>8);
          UART0->BDL = UART_BDL_SBR(sbr);
          UART0->C4 |= UART0_C4_OSR(osr - 1);

          UART0->C1 = UART0_C1_M(0) | UART0_C1_PE(0);

          //MSB first
          UART0->S2 = UART0_S2_MSBF(1);

          UART0->C2 |= UART0_C2_RIE_MASK;

          UART0->C2 |= ((UART_C2_RE_MASK) | (UART_C2_TE_MASK));

          NVIC_ClearPendingIRQ(UART0_IRQn);
          NVIC_SetPriority(UART0_IRQn,2);
          NVIC_EnableIRQ(UART0_IRQn);
          __enable_irq();
}
```

Activăm ceasul pentru modulul UART0 prin setarea bitului corespunzător în registrul SIM_SCGC4.

12.2.8 System Clock Gating Control Register 4 (SIM_SCGC4)

Address: 4004_7000h base + 1034h offset = 4004_8034h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1				0				SPI1	SPI0	0		CMP	USBOTG	0	
W																
Reset	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0		0	UART2	UART1	UART0	0		I2C1	I2C0	1		0			
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0

levanți comun

On-board Usage										Reset State/Function
–	J1 02	D0	PTA1	27	TSI0_CH2	PTA1	UART0_RX	FTM2_CH0		DISABLED
–	J1 04	D1	PTA2	28	TSI0_CH3	PTA2	UART0_TX	FTM2_CH1		DISABLED

Oprim receptorul și transmițătorul pentru a configura modulul UART înainte de a începe operațiile efective, prin punerea de 0 în registrul UART_C2 în câmpurile RE și TE.

39.2.4 UART Control Register 2 (UARTx_C2)

This register can be read or written at any time.

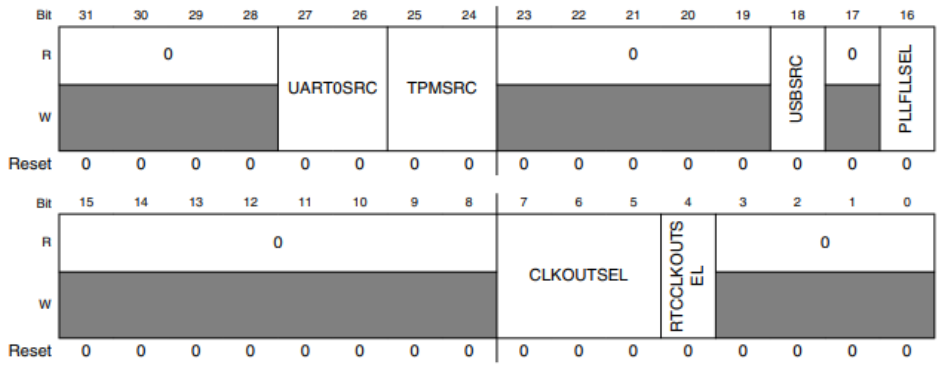
Address: Base address + 3h offset

Bit	7	6	5	4	3	2	1	0
Read Write	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

3 TE	Transmitter Enable TE must be 1 to use the UART transmitter. When TE is set, the UART forces the UART _TX pin to act as an output for the UART system. When the UART is configured for single-wire operation (LOOPS = RSRC = 1), TXDIR controls the direction of traffic on the single UART communication line (UART _TX pin). TE can also queue an idle character by clearing TE then setting TE while a transmission is in progress. When TE is written to 0, the transmitter keeps control of the port UART _TX pin until any data, queued idle, or queued break character finishes transmitting before allowing the pin to tristate. 0 Transmitter disabled. 1 Transmitter enabled.
2 RE	Receiver Enable When the UART receiver is off or LOOPS is set, the UART _RX pin is not used by the UART . When RE is written to 0, the receiver finishes receiving the current character (if any). 0 Receiver disabled. 1 Receiver enabled.

Prin intermediul SOPT2 (System Options Register 2) stabilim sursa de ceas pentru modulul UART0, la 48 MHz.

Address: 4004_7000h base + 1004h offset = 4004_8004h

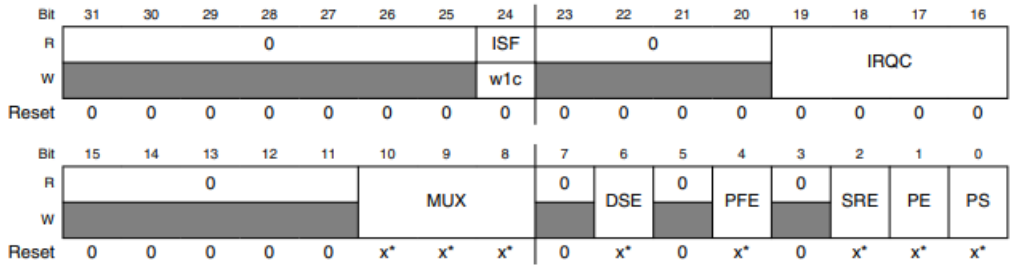


27–26 UART0SRC	UART0 clock source select Selects the clock source for the UART0 transmit and receive clock. 00 Clock disabled 01 MCGFLLCLK clock or MCGPLLCLK/2 clock 10 OSCERCLK clock 11 MCGIRCLK clock

On L-series devices the MCGFLLCLK frequency is limited to 48 MHz max.

Urmează o serie de configurări ale Pin Control Register n (PORTx_PCRn). În primul rând, instrucțiunea `PORTA->PCR[1] = ~PORT_PCR_MUX_MASK;` elimină configurația anterioară a câmpului MUX (Multiplexor) pentru pinul PTA1 (RX), asigurându-se că acesta este pregătit pentru o nouă configurare. Apoi, `PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2);` setează MUX la 2, indicând utilizarea alternativei asociate modulului UART pentru acel pin, și activează mecanismul de semnalizare a evenimentelor (ISF), pregătind astfel pinul pentru a fi utilizat în mod corespunzător într-un sistem de comunicații seriale. Similar, cele două instrucțiuni pentru pinul PTA2 (TX) urmează aceeași logică, asigurându-se că pinul este configurat corespunzător pentru transmiterea datelor. Aceste setări sunt esențiale pentru a stabili o corectă interfață de comunicație între dispozitiv și un alt echipament, cum ar fi un PC, prin intermediul porturilor de comunicație seriale.

Address: Base address + 0h offset + (4d × i), where i=0d to 31d



* Notes:
- x = Undefined at reset.

24 ISF	Interrupt Status Flag This bit is read only for pins that do not support interrupt generation. The pin interrupt configuration is valid in all digital pin muxing modes. 0 Configured interrupt is not detected. 1 Configured interrupt is detected. If the pin is configured to generate a DMA request, then the corresponding flag will be cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to the flag. If the pin is configured for a level sensitive interrupt and the pin remains asserted, then the flag is set again immediately after it is cleared.

10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific). 101 Alternative 5 (chip-specific). 110 Alternative 6 (chip-specific). 111 Alternative 7 (chip-specific).

Se observă apoi expresia sbr = (uint16_t)((DEFAULT_SYSTEM_CLOCK)/(baud_rate * (osr))); , care este responsabilă pentru calcularea valorii prescaler-ului (sbr) în funcție de rata de baud dorită și factorul de suprasarcină (osr). Rata de baud reprezintă numărul de simboluri transmise pe secundă într-o comunicație serială, iar factorul de suprasarcină indică de câte ori se suprasamplează semnalul pentru a asigura o detecție precisă. În proiectul nostru, am folosit osr=32 și baud rate = 9600.

Instrucțiunile următoare sunt responsabile pentru configurarea corectă a parametrilor de comunicație serială. În primul rând, `UART0->BDH &= UART0_BDH_SBR_MASK;` elimină orice setări anterioare ale biților corespunzători valorii prescaler-ului din registrul superior al ratei de baud (UART0_BDH), asigurându-se că acesta este pregătit pentru a primi noile informații. Urmează să punem valoarea sbr calculată de noi în UART Baud Rate Register care este împărțit pe doi regiștri high și low. Astfel primii biți sunt puși în UART_BDH, iar ultimii în UART_BDL. Împreună, aceste acțiuni asigură o corectă alcătuire a întregii valori a prescaler-ului pentru a atinge rata de baud dorită.

39.2.1 UART Baud Rate Register High (UARTx_BDH)

This register, along with UART _BDL, controls the prescale divisor for UART baud rate generation. The 13-bit baud rate setting [SBR12:SBR0] should only be updated when the transmitter and receiver are both disabled.

Address: Base address + 0h offset

Bit	7	6	5	4	3	2	1	0
Read Write	LBKDIE	RXEDGIE	SBNS	SBR				
Reset	0	0	0	0	0	0	0	0

39.2.2 UART Baud Rate Register Low (UARTx_BDL)

This register, along with UART _BDH, control the prescale divisor for UART baud rate generation. The 13-bit baud rate setting [SBR12:SBR0] can only be updated when the transmitter and receiver are both disabled.

UART _BDL is reset to a non-zero value, so after reset the baud rate generator remains disabled until the first time the receiver or transmitter is enabled; that is, UART _C2[RE] or UART _C2[TE] bits are written to 1.

Address: Base address + 1h offset

Bit	7	6	5	4	3	2	1	0
Read / Write	SBR							
Reset	0	0	0	0	0	1	0	0

UARTx_BDL field descriptions

Field	Description
7–0 SBR	Baud Rate Modulo Divisor These 13 bits in SBR[12:0] are referred to collectively as BR. They set the modulo divide rate for the baud rate generator. When BR is 1 - 8191, the baud rate equals baud clock/((OSR+1) × BR).

`UART0->C4 |= UART0_C4_OSR(osr - 1);` configurează factorul de suprasarcină (OSR) pentru modulul UART0. Ajustarea acestui factor este crucială pentru suprasamplearea semnalului și asigurarea unei detecții precise în cadrul comunicației seriale, oferind flexibilitate în adaptarea configurației la specificațiile cerute de protocolul de comunicare.

Prin instrucțiunea "UART0->C1 = UART0_C1_M(0) | UART0_C1_PE(0)", modulul UART0 este configurat pentru a efectua transmiterea datelor utilizând un format de 8 biți și fără utilizarea bitului de paritate.

39.2.3 UART Control Register 1 (UARTx_C1)

This read/write register controls various optional features of the UART system. This register should only be altered when the transmitter and receiver are both disabled.

Address: Base address + 2h offset

Bit	7	6	5	4	3	2	1	0
Read / Write	LOOPS	DOZEEN	RSRC	M	WAKE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

4 M	9-Bit or 8-Bit Mode Select 0 Receiver and transmitter use 8-bit data characters. 1 Receiver and transmitter use 9-bit data characters.
1 PE	Parity Enable Enables hardware parity generation and checking. When parity is enabled, the bit immediately before the stop bit is treated as the parity bit. 0 No hardware parity generation or checking. 1 Parity enabled.

Prin intermediul lui UART0_S2 (UART Status Register 2) vom seta câmpul MSBF cu valoarea 1, configurând modul de transmitere pentru a începe cu cel mai semnificativ bit (MSB). Acest aspect este crucial pentru asigurarea unei sincronizări corespunzătoare între dispozitivul de transmitere și cel de recepție, stabilind ordinea corectă a biților în cadrul fiecărui caracter transmis.

Address: Base address + 5h offset

Bit	7	6	5	4	3	2	1	0
Read	LBKDIF	RXEDGIF	MSBF	RXINV	RWUID	BRK13	LBKDE	RAF
Write								
Reset	0	0	0	0	0	0	0	0

5 MSBF	MSB First Setting this bit reverses the order of the bits that are transmitted and received on the wire. This bit does not affect the polarity of the bits, the location of the parity bit or the location of the start or stop bits. This bit should only be changed when the transmitter and receiver are both disabled.

	0 LSB (bit0) is the first bit that is transmitted following the start bit. Further, the first bit received after the start bit is identified as bit0. 1 MSB (bit9, bit8, bit7 or bit6) is the first bit that is transmitted following the start bit depending on the setting of C1[M], C1[PE] and C4[M10]. Further, the first bit received after the start bit is identified as bit9, bit8, bit7 or bit6 depending on the setting of C1[M] and C1[PE].

Prin instrucțiunea "UART0->C2 |= UART0_C2_RIE_MASK;", se activează întreruperile asociate cu recepționarea datelor în cadrul modulului UART0. Această setare permite modulului să semnalizeze procesorului atunci când există date noi disponibile pentru citire, facilitând astfel gestionarea eficientă a fluxului de informații și permitând procesorului să reacționeze imediat la evenimentele de recepție, garantând astfel un răspuns prompt la datele primite.

În continuare, instrucțiunea "UART0->C2 |= ((UART_C2_RE_MASK) | (UART_C2_TE_MASK));" activează atât receptorul (RE - Receiver Enable), cât și transmițătorul (TE - Transmitter Enable) în cadrul modulului UART0. Această acțiune pornește inițierea comunicării seriale bidirecționale, permițând dispozitivului să transmită date către alte componente ale sistemului prin portul UART0 și, în același timp, să primească informații de la acestea. Prin aceste configurații, modulul UART0 este complet pregătit să faciliteze schimbul bidirecțional de date în cadrul sistemului, contribuind la funcționarea corespunzătoare a dispozitivului și la interoperabilitatea acestuia cu alte componente ale sistemului.

Ne pregătim să pornim întreruperile pentru UART la nivelul NVIC (Nested Vector Interrupt Controller). Întâi vom șterge eventualele cereri în așteptare pentru întreruperile asociate modulului UART0 prin apelarea NVIC_ClearPendingIRQ(UART0_IRQn). Setăm apoi prioritatea pentru întreruperea asociată modulului UART0 cu valoarea 2, prin NVIC_SetPriority(UART0_IRQn, 2). Prin NVIC_EnableIRQ(UART0_IRQn) activăm apoi întreruperile asociate modulului UART0 în NVIC. Iar în final prin “__enable_irq()” activăm global întreruperile.

5.2.6 Inițializarea modulului ADC

Vom folosi modulul ADC pentru a citi datele furnizate de senzorul ambiental conectat de noi la pinul 3 al portului B (ADC_CHANNEL 13).

```
#include "Adc.h"
#include "Uart.h"

#define ADC_CHANNEL (13) // PORT B PIN 3 -> senzor ambiental

void ADC0_Init() {
          SIM->SCGC6 |= SIM_SCGC6_ADC0_MASK;

          ADC0_Calibrate();

          ADC0->CFG1 = 0x00;

          ADC0->CFG1 |= ADC_CFG1_MODE(3) | ADC_CFG1_ADICLK(0) | ADC_CFG1_ADIV(2);

          ADC0->SC1[0] = 0x00;
          ADC0->SC3 = 0x00;

          ADC0->SC3 |= ADC_SC3_ADCO_MASK;

          ADC0->SC1[0] |= ADC_SC1_ADCH(ADC_CHANNEL);
}
```

Prima dată activăm ceasul (clock) pentru modulul ADC0 prin setarea bitului corespunzător în registrul SIM_SCGC6 (System Clock Gating Control Register 6).

27 ADC0	ADC0 Clock Gate Control This bit controls the clock gate to the ADC0 module. 0 Clock disabled 1 Clock enabled

Conform manualului de utilizare (Reference Manual, pag. 482), pentru ca modulul ADC să poată îndeplini specificațiile de precizie, el trebuie calibrat folosind funcția de calibrare încorporată. Funcția ADC0_Calibrate(), realizează acest lucru, urmărind următorii pași (Reference Manual pag. 495):

1. Initialize or clear a 16-bit variable in RAM.
2. Add the plus-side calibration results CLP0, CLP1, CLP2, CLP3, CLP4, and CLPS to the variable.
3. Divide the variable by two.
4. Set the MSB of the variable.
5. The previous two steps can be achieved by setting the carry bit, rotating to the right through the carry bit on the high byte and again on the low byte.
6. Store the value in the plus-side gain calibration register PG.
7. Repeat the procedure for the minus-side gain calibration value.

Urmează apoi configurarea propriu-zisă a modulului prin modificarea valorilor din registrul ADC0->CFG1 (ADC Configuration Register 1) astfel:

- Inițial setăm toți biții pe 0
- Setăm ADC_CFG1_MODE cu 3 –> vom lucra în modul single-ended pe 16 biți
- Setăm ADC_CFG1_ADICLK cu 0 –> sursa de ceas pentru ADC va fi Bus clock
- Setăm ADC_CFG1_ADIV cu 2 –> setăm ca ceasul ADC-ului nostru să fie (Bus clock)/4

Address: 4003_B000h base + 8h offset = 4003_B008h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0								ADLPC	ADIV		ADLSMP	MODE		ADICLK	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3–2 MODE	Conversion mode selection Selects the ADC resolution mode. 00 When DIFF=0:It is single-ended 8-bit conversion; when DIFF=1, it is differential 9-bit conversion with 2's complement output. 01 When DIFF=0:It is single-ended 12-bit conversion ; when DIFF=1, it is differential 13-bit conversion with 2's complement output. 10 When DIFF=0:It is single-ended 10-bit conversion ; when DIFF=1, it is differential 11-bit conversion with 2's complement output. 11 When DIFF=0:It is single-ended 16-bit conversion; when DIFF=1, it is differential 16-bit conversion with 2's complement output.
1–0 ADICLK	Input Clock Select Selects the input clock source to generate the internal clock, ADCK. Note that when the ADACK clock source is selected, it is not required to be active prior to conversion start. When it is selected and it is not active prior to a conversion start, when CFG2[ADACKEN]=0, the asynchronous clock is activated at the start of a conversion and deactivated when conversions are terminated. In this case, there is an associated clock startup delay each time the clock source is re-activated. 00 Bus clock 01 (Bus clock)/2 10 Alternate clock (ALTCLK) 11 Asynchronous clock (ADACK)

6–5 ADIV	Clock Divide Select ADIV selects the divide ratio used by the ADC to generate the internal clock ADCK. 00 The divide ratio is 1 and the clock rate is input clock. 01 The divide ratio is 2 and the clock rate is (input clock)/2. 10 The divide ratio is 4 and the clock rate is (input clock)/4. 11 The divide ratio is 8 and the clock rate is (input clock)/8.

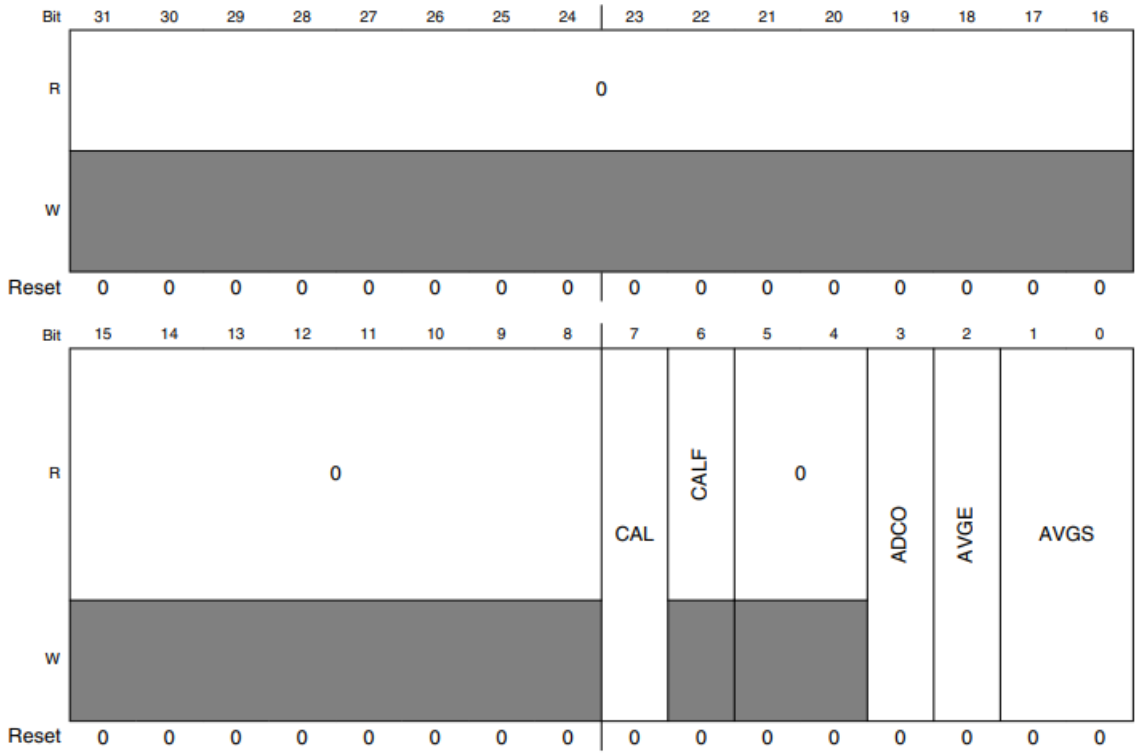
Încep prin a reseta toți biții din registrele ADC0->SC1[0] și ADC0->SC3 la valoarea 0, urmând să activez ulterior doar acei biți esențiali pentru funcționarea modulului.

Prin "ADC0->SC3 |= ADC_SC3_ADCO_MASK;", se activează modul de operare continuu al modulului ADC0, setând bitul ADCO în registrul SC3 (Status and Control Register 3). Aceasta permite modulului să efectueze conversii în mod continuu în locul operațiilor discrete.

28.3.7 Status and Control Register 3 (ADCx_SC3)

The Status and Control Register 3 (SC3) controls the calibration, continuous convert, and hardware averaging functions of the ADC module.

Address: 4003_B000h base + 24h offset = 4003_B024h



3 ADCO	Continuous Conversion Enable Enables continuous conversions. 0 One conversion or one set of conversions if the hardware average function is enabled, that is, AVGE=1, after initiating a conversion. 1 Continuous conversions or sets of conversions if the hardware average function is enabled, that is, AVGE=1, after initiating a conversion.

În final, "ADC0->SC1[0] |= ADC_SC1_ADCH(ADC_CHANNEL)" specifică canalul pentru conversie, în acest caz, canalul stabilit de noi prin constanta ADC_CHANNEL (13). Această setare determină modulul ADC să efectueze conversii pe acest canal, pentru citirea datelor de la senzorul ambiental conectat.

5.2.7 Inițializarea modulului TMP

Vom utiliza modulul TPM2 (Timer/PWM Module) de pe microcontroller pentru a genera semnalul PWM. Pentru inițializarea modului, am folosit următoarele:

```
 #define SERVO_PIN (2) // PORT B, PIN 2

void TPM2_Init(){
          SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK;

          PORTB->PCR[SERVO_PIN] |= PORT_PCR_MUX(3);

          SIM->SOPT2 |= SIM_SOPT2_TPMSRC(1);
          SIM->SCGC6 |= SIM_SCGC6_TPM2(1);

          TPM2->SC |= TPM_SC_PS(7);
          TPM2->SC |= TPM_SC_CPWMS(0);
          TPM2->SC |= TPM_SC_CMOD(1);

          TPM2->CONTROLS[0].CnSC |= (TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK);
}
```

SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK: activează ceasul pentru portul B, deoarece vom folosi pinul 2 de pe acest port pentru transmiterea impulsurilor către servomotor.

PORTB->PCR[SERVO_PIN] |= PORT_PCR_MUX(3): configurăm multiplexorul pentru pinul specificat pe valoarea 3, acest lucru indicând alternative de funcționare a sa, și anume cea de TPM2, canalul 0.

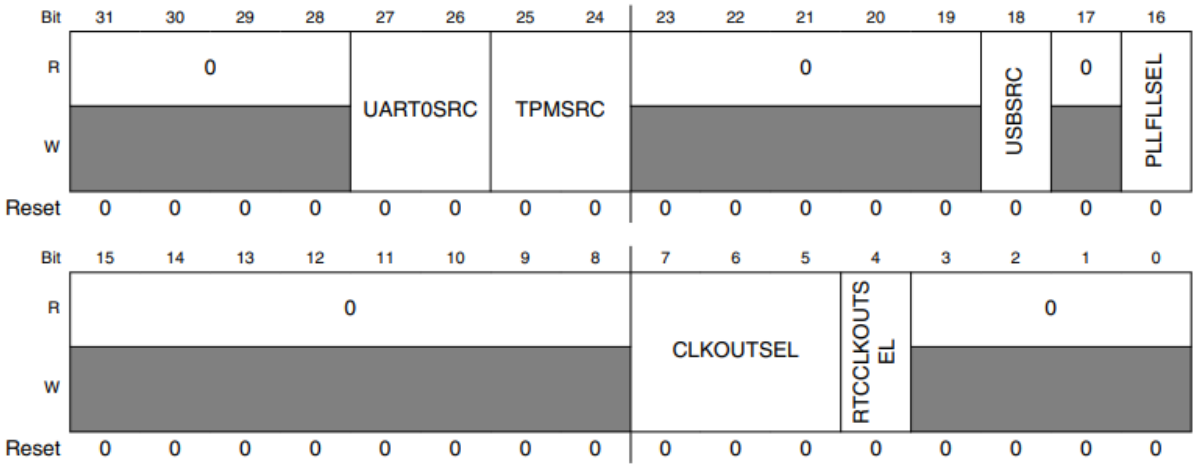
În continuare, vom configura sursa de ceas pentru modulul TPM, selectând MCGFLLCLK, care are o frecvență de funcționare de 48MHz. Pentru a putea folosi modulul, va trebui și să activăm semanlul de ceas pentru modulul TPM. Acest lucru este realizat prin intermediul regiștrilor SOPT2 și SCGC6 din SIM (configureaza ceasurile ssitemului, UART, TPM etc).

25–24 TPMSRC	TPM clock source select Selects the clock source for the TPM counter clock 00 Clock disabled 01 MCGFLLCLK clock or MCGPLLCLK/2 10 OSCERCLK clock 11 MCGIRCLK clock

12.2.3 System Options Register 2 (SIM_SOPT2)

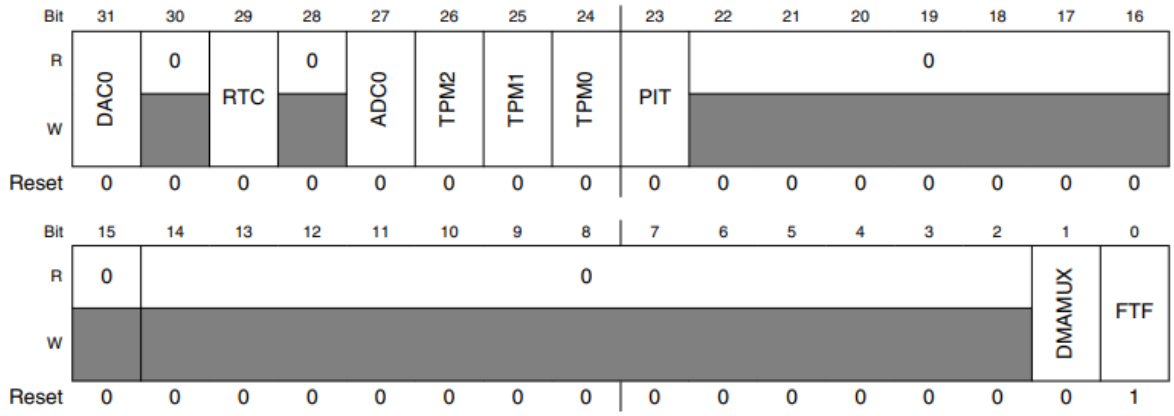
SOPT2 contains the controls for selecting many of the module clock source options on this device. See the Clock Distribution chapter for more information including clocking diagrams and definitions of device clocks.

Address: 4004_7000h base + 1004h offset = 4004_8004h



12.2.10 System Clock Gating Control Register 6 (SIM_SCGC6)

Address: 4004_7000h base + 103Ch offset = 4004_803Ch



26 TPM2	TPM2 Clock Gate Control This bit controls the clock gate to the TPM2 module. 0 Clock disabled 1 Clock enabled

Vom folosi un divisor de frecvență care să împartă frecvența de bază a modului TPM (48 MHz) pentru a obține o valoare mai mica pentru a putea încetini procesul de numărare al modului TPM. Am ales un dizivor de frecvență de 2^7 = 128 (TPM_SC_PS(7) în registrul de status și control), deci frecvența semnalui de intrare PWM va fi acum de 375kHz.

Tot în registrul de status și control, voi seta direcția de numărare pe up (CPWMS) și voi activa număratoarea (CMOD).

31.3.1 Status and Control (TPMx_SC)

SC contains the overflow status flag and control bits used to configure the interrupt enable, module configuration and prescaler factor. These controls relate to all channels within this module.

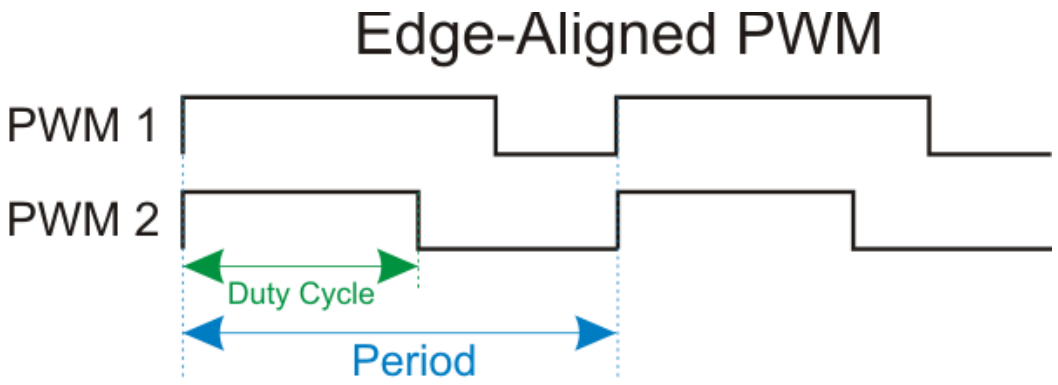
Address: Base address + 0h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0							DMA	TOF	TOIE	CPWMS	CMOD		PS		
W									w1c							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 CPWMS	Center-aligned PWM Select Selects CPWM mode. This mode configures the LPTPM to operate in up-down counting mode. This field is write protected. It can be written only when the counter is disabled. 0 LPTPM counter operates in up counting mode. 1 LPTPM counter operates in up-down counting mode.
4–3 CMOD	Clock Mode Selection Selects the LPTPM counter clock modes. When disabling the counter, this field remain set until acknolwedged in the LPTPM clock domain. 00 LPTPM counter is disabled 01 LPTPM counter increments on every LPTPM counter clock 10 LPTPM counter increments on rising edge of LPTPM_EXTCLK synchronized to the LPTPM counter clock 11 Reserved
2–0 PS	Prescale Factor Selection Selects one of 8 division factors for the clock mode selected by CMOD. This field is write protected. It can be written only when the counter is disabled. 000 Divide by 1 001 Divide by 2 010 Divide by 4 011 Divide by 8 100 Divide by 16 101 Divide by 32 110 Divide by 64 111 Divide by 128

TPM2->CONTROLS[0].CnSC |= (TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK) va configure canalul 0 al modulului TPM2 pentru a genera un semnal PWM. Se setează bitul MSB și ELSB pentru a putea să fie folosit în modul edge-aligned:



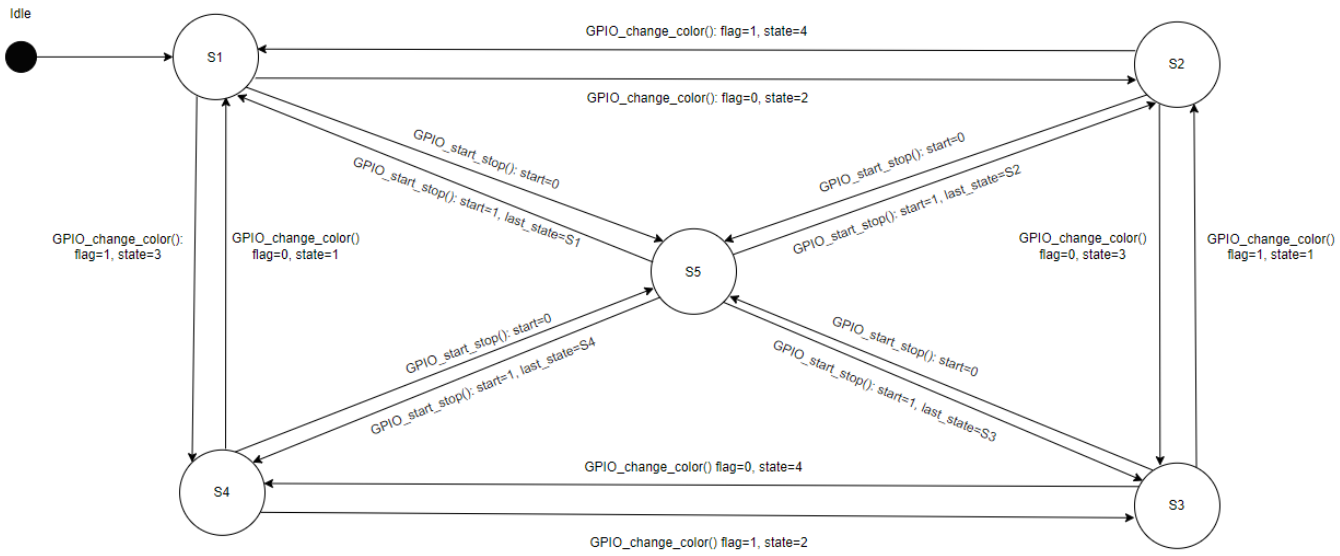
5.3 Controlul LED-urilor

Modulul GPIO este cel responsabil să controleze starea și culoarea ledurilor RGB de pe plăcuță prin intermediul pinilor de GPIO. Schimbarea culorilor are loc în urma declanșării unui semnal de întrerupere de către PIT, care are loc la fiecare 275ms.

```
void PIT_IRQHandler(void) {
          if(PIT->CHANNEL[0].TFLG & PIT_TFLG_TIF_MASK) {
                    GPIO_change_color();
                    PIT->CHANNEL[0].TFLG &= PIT_TFLG_TIF_MASK;
          }
}
```

Pentru logica de schimbare a culorilor, s-au folosit 2 flag-uri: state și flag. State controlează ce culoare urmează să se afișeze, iar flag definește care este ordinea de afișare. Dacă flag este 0, atunci ordinea de afișare va fi alb, verde, albastru, mov, iar pentru 1, ordinea de afișare este mov, alb, albastru, verde. Trecerea de la o stare la cealaltă se face conform secvenței 1,2,3,4 respectiv 4,3,2,1, fiecare număr reprezentând o culoare dorită. Acest lucru este reprezentat în următoarea diagramă de stări:

5.3.1 Diagrama de stări pentru LED-uri



State	Led	Observații
S1	Alb	Toate ledurie sunt aprinse
S2	Verde	Ledul Verde este aprins
S3	Albastru	Ledul Albastru este aprins
S4	Mov	Ledurile Albastru și Roșu sunt aprinse
S5	Stinse	Toate LED-urile sunt aprins

```
uint8_t start=1;
uint8_t state=0;
uint8_t flag=0;

void GPIO_change_seq(void)
{
          if(flag==0)
          {
                    flag=1;
                    state=3;
                    return;
          }
          flag=0;
          state=1;
}

void GPIO_start_stop(void)
{
          if(start==0)
                    start=1;
          else
          {
                    start=0;
                    GPIOB->PSOR|=(1<<RED_LED_PIN);
                    GPIOB->PSOR|=(1<<GREEN_LED_PIN);
                    GPIOD->PSOR|=(1<<BLUE_LED_PIN);
          }
}

void GPIO_change_color(void)
{
          if(start==0)
          {
                    return;
          }
          //  state=1-> white on   state=2 -> green on    state=3  -> blue on  state=4 -> purple on
          //flag =1 -> reverse state
          if((state ==1 && flag==0)||(state==4 && flag==1))
          {
                    GPIOB->PCOR|=(1<<RED_LED_PIN);
                    GPIOB->PCOR|=(1<<GREEN_LED_PIN);
            GPIOD->PCOR|=(1<<BLUE_LED_PIN);
          }
          if((state==2 && flag==0)||(state==1 && flag==1))
          {
            GPIOB->PSOR|=(1<<RED_LED_PIN);
            GPIOB->PCOR|=(1<<GREEN_LED_PIN);
            GPIOD->PSOR|=(1<<BLUE_LED_PIN);
          }
          if((state ==3 && flag==0)||(state==2 && flag==1))
          {
            GPIOB->PSOR|=(1<<RED_LED_PIN);
            GPIOB->PSOR|=(1<<GREEN_LED_PIN);
            GPIOD->PCOR|=(1<<BLUE_LED_PIN);
          }
          if((state ==4 && flag==0)||(state==3 && flag==1))
          {
            GPIOB->PCOR|=(1<<RED_LED_PIN);
            GPIOB->PSOR|=(1<<GREEN_LED_PIN);
            GPIOD->PCOR|=(1<<BLUE_LED_PIN);
          }
          if(flag==0)
          {
                    state=(state%4)+1;
          }
```

```
          else
          {
                    if(state!=1)
                      state=state-1;
                    else
                      state=4;
          }
}
```

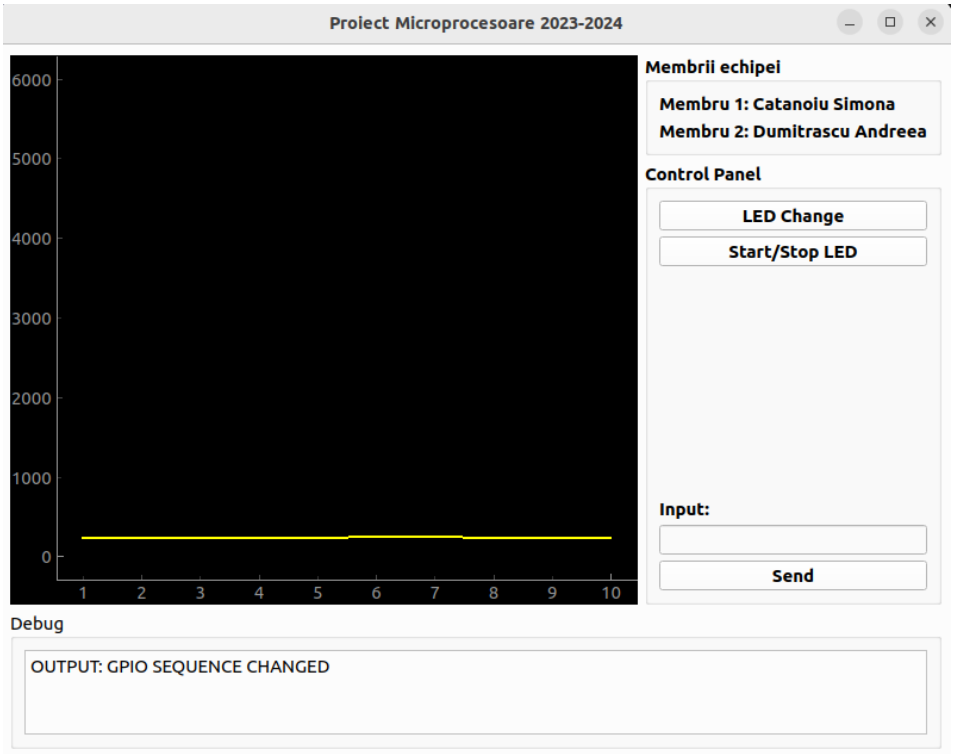
Semnalul de schimbare al culorii și de aprindere și stingere se realizează din interfață, prin apăsarea butonului de „LED Change”și ”Start/Stop LED” din interfață, prin transmiterea unor octeți speciali către UART (se vor trimite octeții 15 și 17 care sunt non-ASCII pentru a nu fi confundați cu INPUT-ul dat de utilizator):

```
COD PYTHON:
def led_change_event(self):
        send_ledTrigger_to_serial(self.serial_conn,17)
        print(f"Sented GPIO change led sequence flag")

    def button2_action(self):
        send_ledTrigger_to_serial(self.serial_conn,15)
        print(f"Sented GPIO start/stop flag")

def send_ledTrigger_to_serial(serial_conn,int_data):
    try:
        if serial_conn is not None and serial_conn.is_open:
            serial_conn.write(int_data.to_bytes(1,byteorder='little'))
    except serial.SerialException as e:
        print(f"Error writing to serial port: {e}")
```

În logica de recepție a datelor de la portul serial, în funcția de UART0_IRQHandler, avem următoarele:

```
#define GPIO_CHANGE_FLAG 17 -> uart.h

//GPIO FLAG HANDLER
          if(reverse_data==GPIO_CHANGE_FLAG)
          {
                    GPIO_change_seq();
                    gpio_message_flag=1;
                    UART0_Transmit(0); //transmit 0 ca sa intre in UART0_Transmit, unde am un while
                                       //ce va trimite un mesaj interfata pentru a semnaliza
                                       //schimbarea cu succes a culorilor
                    return;
          }
          //GPIO START/STOP LEDs
          if(reverse_data==15)
          {
                    GPIO_start_stop();
                    return;
          }
```

Rezultate obținute din intefață:



5.4 Controlul recepției și transmisiei

La nivelul interfeței, am creat un thread separat care să asculte pe portul serial pentru a detecta când vin date de la microcontroller. În plus, detectarea portului pe care să asculte este făcut în mod automat prin parcurgerea tuturor porturilor disponibile în sistem.

COD PYTHON:

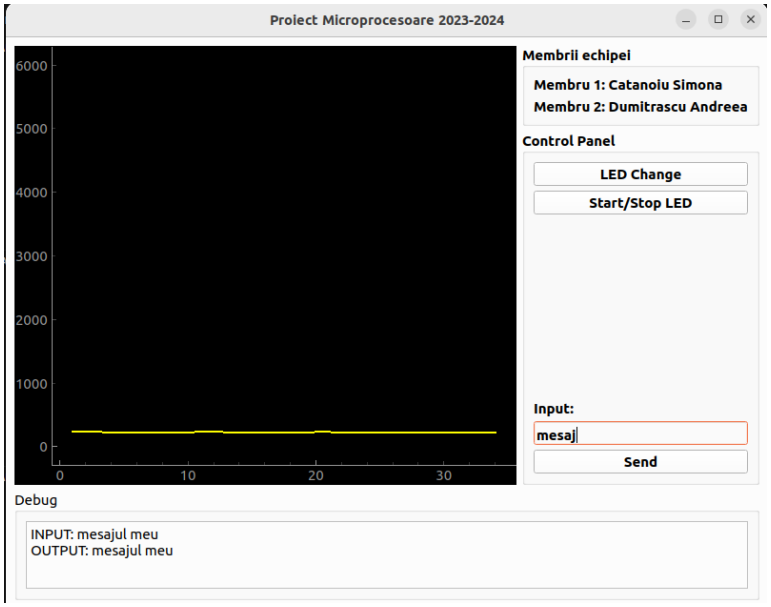
```python
class SerialReceiverThread(QThread):
    received_signal = Signal(str)

    def __init__(self, serial_conn):
        super().__init__()
        self.serial_conn = serial_conn

    def run(self):
        while True:
            output = read_data_from_serial(self.serial_conn)
            self.received_signal.emit(output)
```

COD PYTHON:

```python
def find_open_serial_port():
    #parcurge toate porturile seriale (/dev/ttyS*) dev/ttyS5<=>COM5
    my_ports = serial.tools.list_ports.comports()
    for port in my_ports:
        try:
            serial_conn = serial.Serial(port.device, baud_rate,bytesize=8, timeout=1)
            serial_conn.close()
            if(port.description!="ttyS0"):
                return port.device
        except serial.SerialException as e:
            print(f"ERROR:{e}")
```

```
            pass
    return None
```

Datele vor fi transmise sub formă de bytes, little-endian. Utilizatorul va putea să trimită output către microcontroller, care apoi va detecta aceste valori și le va trimite din nou în interfață sub formă de output în caseta din partea de jos a interfeței:



Am implementat acest lucru într-o fază inițială a proiectului pentru a ne asigura că am configurat bine componentele și mașina virtuală. Atât la nivelul transmisiei, cât și a recepției există niște valori speciale care sunt folosite fie pentru a identifica valorile ce urmează să fie reprezentate grafic (delimitate de byte-ul cu valoarea 19), fie pentru executarea unor operații speciale (schimbarea secvenței led-urilor – byte-ul 17, sau oprirea/aprinderea led-urilor – byte-ul 15). Au fost alese aceste valori deoarece ele sunt non-ASCII și să nu fie confundate cu input-ul utilizatorului.

La nivelul microcontroller-ului, recepția se face asincron prin intermediul întreruperilor, iar transmisia se face folosind polling prin 2 funcții: UART0_Transmit și UART0_Transmit_ADC_val. De asemenea, deoarece UART folosește MSB iar datele sunt primite din interfață conform modelului little-endian, am creat o funcție în utils.c care să ma ajute să fac conversia la recepție și transmisie: msb_to_lsb.

Cod:

```
void UART0_Transmit(uint8_t data) {
          while(!(UART0->S1 & UART_S1_TDRE_MASK)) {}
          if(!gpio_message_flag)
                    UART0->D = lsb_to_msb(data);
          if(gpio_message_flag)
          {
                    uint8_t gpio_message_index=0;
                    while(gpio_message_index<gpio_message_size)
                    {
                              while(!(UART0->S1 & UART_S1_TDRE_MASK)) {}
                              UART0->D = lsb_to_msb(gpio_message[gpio_message_index]);
                              gpio_message_index++;
```

```
                    }
                    gpio_message_flag=0;
          }
}

void UART0_Transmit_ADC_val(uint16_t data)
{
            uint8_t letters[5];
                    int index=0;
                    while(data)
                    {
                              letters[index] = data%10+0x30;
                              data/=10;
                              index++;
                    }
                    index--;

                    UART0_Transmit(ADC_START_END_TRANSMISSION); //ADC_START_END_TRANSMISSION=19
                    while(index>=0)
                    {
                              UART0_Transmit(letters[index--]);
                    }

                     UART0_Transmit(ADC_START_END_TRANSMISSION);
}
void UART0_IRQHandler(void) {
                    if(UART0->S1 & UART0_S1_RDRF_MASK)
                    {
                              uint8_t data=UART0->D;
                              //Reverse LSB to MSB
                              uint8_t reverse_data=msb_to_lsb(data);

                              //ENTER HANDLER
                              if(reverse_data==0x0D)
                              {
                                        UART0_Transmit(0x0D);
                                        UART0_Transmit(0x0A);
                              }

                              //GPIO FLAG HANDLER
                              if(reverse_data==GPIO_CHANGE_FLAG)
                              {
                                        GPIO_change_seq();
                                        gpio_message_flag=1;
                                        UART0_Transmit(0);
                                        return;
                              }
                              //GPIO START/STOP LEDs
                              if(reverse_data==15)
                              {
                                        GPIO_start_stop();
                                        return;
                              }
                              else
                              {
                                        UART0_Transmit(reverse_data);
                              }
                    }
}
utils.c
uint8_t msb_to_lsb(uint8_t data)
{
          uint8_t reverse_data=0;
          uint8_t i;
          for(i=0;i<8;i++)
```

```
        {
                reverse_data<<=1;
                reverse_data|=(data&1);
                data>>=1;
        }
        return reverse_data;
}
```

5.5 Controlul servomotorului

Informația este trimisă către servomotor prin modularea lățimii impulsurilor primite de la modulul TPM. În esență, semnalul constă în impulsuri cu durate variabile, iar modularea constă în ajustarea lățimii acestor impulsuri pentru a transmite informația dorită. Un semnal PWM constă într-un ciclu format dintr-o perioadă și unul sau mai multe impulsuri. Lățimea unui impuls este măsurată în unități de timp și se numește duty cycle (raportul dintre proporția în care sistemul este pe nivel înalt și timpul total al perioadei). Pentru servomotorul nostru, factorul de umplere va determina poziția în care va fi plasată elicea. Acest lucru se face prin intermediul registrelor MOD(setează perioada la 20ms) și Cnv (setează duty cycle pentru primul canal) corespunzătoare modulului TPM2.

Elicea se va roti în funcție de valoarea recepționată de la senzorul de lumină. Acesta transmite valori cuprinse între 0 si 6000, dar din observații practice, pentru condiții de lumină normale, valorile se află undeva în intervalul 1000-3000. De aceea, vom considera pragul de lumină puternică valorile de peste 3000, pe cele de lumină normală între 1000 și 3000, iar sub 1000 pe cele de lumină scăzută. În funcție de aceste valori, calculăm prin registrul CnV ca elicea să se afle la 0°- 100, 90°-250 și 180°-400. Aceste valori au fost calculate să fie echivalentul a unui factor de umplere de 5, 7.5 respectiv 10% și sunt asociate cu impulurile ce vor realiza controlul servomotorului SG90 la unghiurile dorite.

```
void TPM2_Set_DutyCycle(uint16_t val)
{
  uint8_t duty_cycle = 0;
  if (val > 3000) {
      duty_cycle = 2;
  } else if (val > 1000) {
     duty_cycle = 1;
  } else {
    duty_cycle = 0;
  }
  TPM2->CNT = 0x0000;
  TPM2->MOD = 375 * 20;
  TPM2->CONTROLS[0].CnV =100+duty_cycle*150;  // 100 pentru 0, 250 pentru 90 si 400 pentru 180
}
```

5.6 Manipularea datelor de la senzorul analogic

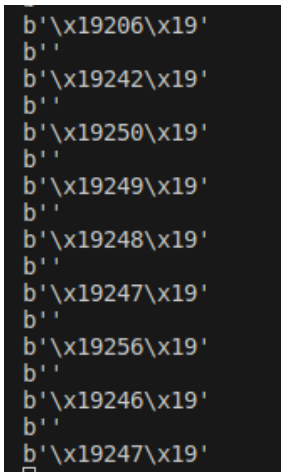
În cadrul funcției main, datele provenite de la senzor sunt solicitate o dată la fiecare secundă prin apelul funcției ADC_get_value(). Această funcție utilizează în interiorul său funcția ADC0_Read() pentru a obține valoarea actuală de la convertorul analog-digital (ADC). Valoarea citită este ulterior convertită într-o măsură a iluminanței, având la bază o formulă specifică, deoarece valorile ADC se situează în intervalul de la 0 la 65535, iar iluminanța este estimată între 1 și 6000 (conform specificațiilor senzorului).

Rezultatul transformării este returnat sub forma unui întreg de 16 biți (uint16_t). Această ilustrare a nivelului de iluminare este apoi transmisă prin intermediul modulului UART și folosită pentru a regla unghiul de rotație al elicei servomotorului.

În cadrul funcției ADC0_Read(), se inițiază procesul propriu-zis de convertire analog-digitală pentru canalul ADC_CHANNEL. Programul așteaptă apoi finalizarea procesului, monitorizând starea conversiei prin intermediul indicelui ADC_SC1_COCO_MASK. Odată ce conversia este finalizată, valoarea convertită este extrasă din registrul corespunzător și returnată sub forma unui întreg de 16 biți (uint16_t). Această metodă de citire sincronizată a datelor asigură obținerea și utilizarea corespunzătoare a informațiilor provenite de la senzorul ADC în cadrul sistemului.

```
uint16_t ADC0_Read(){
          ADC0->SC1[0] = ADC_SC1_ADCH(ADC_CHANNEL);

          while(!(ADC0->SC1[0] & ADC_SC1_COCO_MASK));

          return (uint16_t) ADC0->R[0];

}

uint16_t ADC_get_value()
{
          uint16_t analog_input = (uint16_t) ADC0_Read();
          float measured_illuminance = analog_input / (float)(65535) * (6000.0f - 1.0f);  // (Max
Lux - Min Lux) / Max ADC
          uint16_t val=(uint16_t)measured_illuminance;
          return val;
}

În main.c:
for(;;){
                    if(flag_1s){
                              uint16_t val = ADC_get_value();
                              UART0_Transmit_ADC_val(val);
                              TPM2_Set_DutyCycle(val);
                              flag_1s = 0U;
                    }
          }
```

În ceea ce privește interfața, pentru primirea datelor de la plăcuță, s-a folosit un thread separat care la fiecare interval de 1 secundă va lua valoarea salvată în variabila globală plot_value și o va transmite către interfață pentru a o putea manipula în grafic. S-a ales 1 secundă deoarece acesta este timpul în care sunt primite datele de la ADC. Valoarea este salvată de funcția ce se ocupă de primirea datelor de la UART (read_data_from_serial), care primește datele sub această formă (\x19 este byte-ul setat pentru a delimita valorile ce reprezintă datele recepționate de senzor):



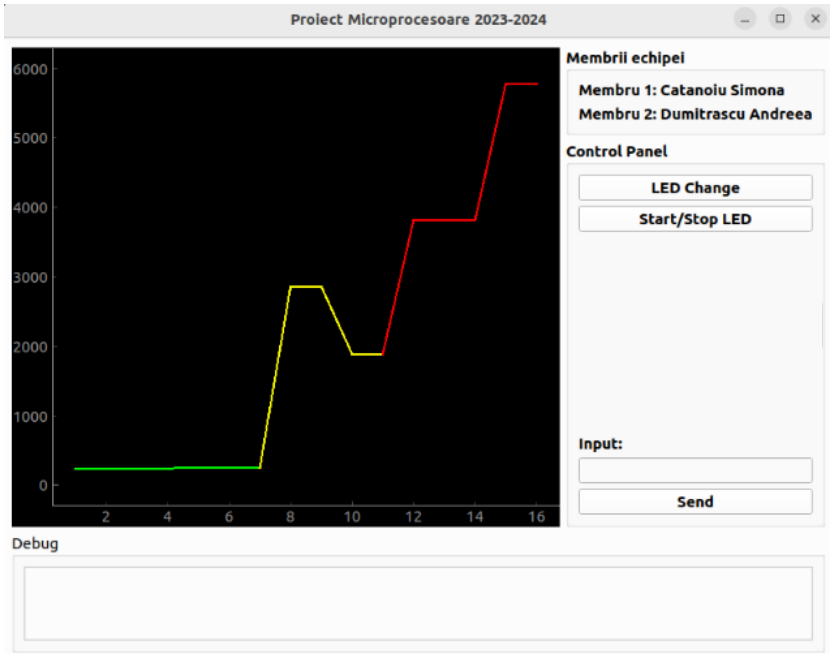

```
COD PYTHON:
class ADCThread(QThread):
    adc_value_received = Signal(int)
    def __init__(self, serial_conn):
        super().__init__()
        self.serial_conn = serial_conn
    def run(self):
        while True:
            adc_value = read_adc_value()
            self.adc_value_received.emit(adc_value)
            time.sleep(1)
În serial_controller.py:
def read_adc_value():
    global plot_value
    ret_value = plot_value
    return ret_value
```

Codul folosit pentru constuirea reprezentării graficului folosește 2 variabile globale ce vor salva secundele și valoarile asociată primite de la controller pe perioada unui minut. După un minut, graficul se va reseta și se va începe construirea unui grafic nou. Colorarea graficului se va face folosind culoarea verde pentru valori mici, galben pentru cele medii și roșu pentru cele ridicate. Așa cum am precizat și la PWM, aceste praguri sunt setate în funcție de valorile practice observate la utilizarea senzorului. În fiecare secundă, nu se va recolora tot graficul, ci doar partea corespunzătoare acelui interval.

```
COD PYTHON:
seconds = []
light_value = []
def update_plot(self, adc_value):
        plot_value = adc_value
        if plot_value != -1:
            self.light_value.append(plot_value)
            self.seconds.append(len(self.light_value))

            if plot_value > 3000:
                color = (255, 0, 0)
            elif plot_value > 1000:
                color = (255, 255, 0)
            else:
                color = (0, 255, 0)

            if len(self.seconds) > 1:
```

```
                x_start, x_end = self.seconds[-2], self.seconds[-1]
                y_start, y_end = self.light_value[-2], self.light_value[-1]
                line_segment = pg.PlotCurveItem(x=[x_start, x_end], y=[y_start, y_end],
pen=pg.mkPen(color=color, width=2))
                self.plot_widget.addItem(line_segment)

            self.plot_widget.getPlotItem().getViewBox().setYRange(0, 6000)

            if len(self.seconds) > 60:
                self.light_value.clear()
                self.seconds.clear()
                self.plot_widget.clear()
```

Rezultate obținute în interfață:

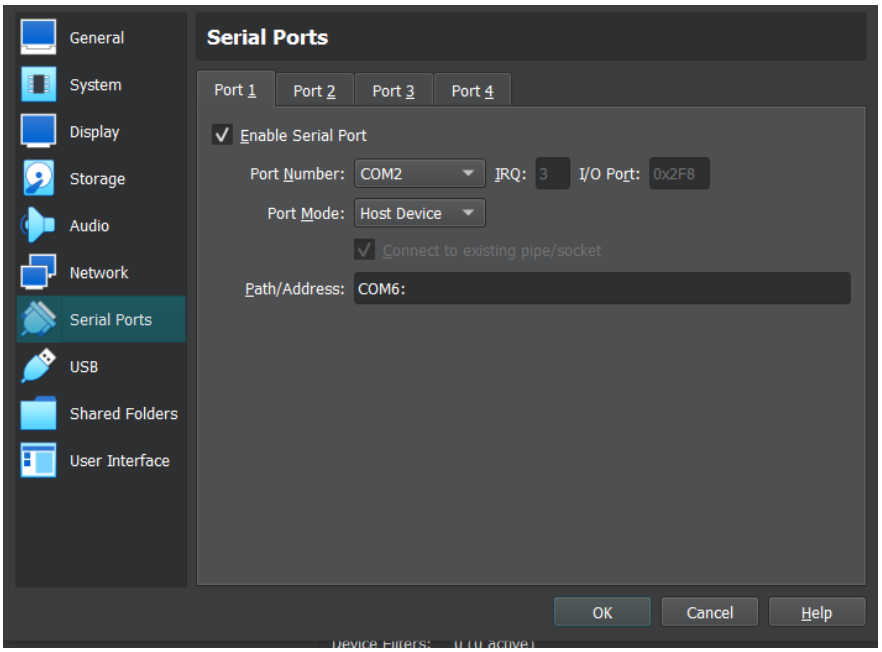


6 Setup proiect

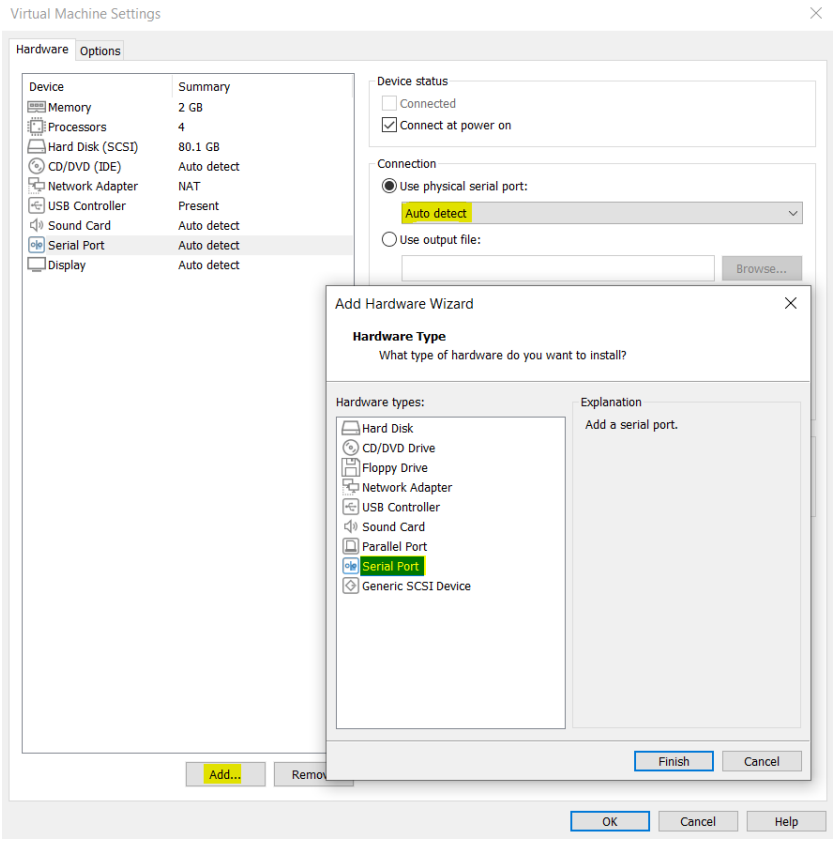
Acest proiect a fost dezolvat folosind Ubuntu ca mediu de dezvoltare, folosind o versiune de python 3.11, cu următoarele plugin-uri: pySide 6.6.0 , pyserial 3.5 și pyqtgraph==0.13.3. Pentru instalarea dependințelor necesare, se vor putea urmări pașii descriși și în proiectul de pe Github:

```
python3.11 -m venv venv-up-all

source venv-up-all/bin/activate
python -m pip install --upgrade pip
pip install --force-reinstall -r requirements.txt
```

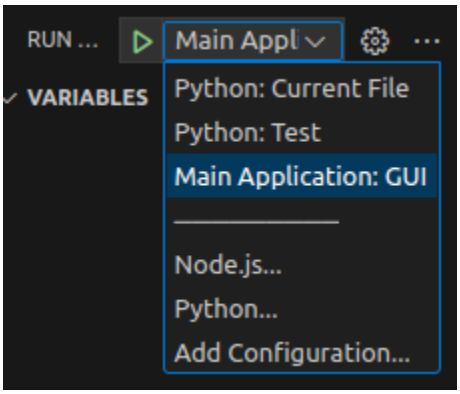
De asemenea, pentru a putea permite mașinii virtuale comunicarea cu portul serial al host-ului, va fi nevoie ca acesta să fie configurat din setările software-ului de virtualizare. Pentru VirtualBox, avem următorul exemplu care demonstrează conectarea portului COM6 al host-ului la /dev/ttyS2 din Ubuntu:



În cazul folosirii VmWare, conectarea se realizează astfel:



Pentru a putea rula interfața, se deschide codul de VSCode, și se rulează “Main Application:GUI”.



7 Dificultăți întâmpinate

- Nu am reușit să rulăm interfața din Windows sau din WSL. A trebuit să o configurăm în mașina virtuală și să dăm drumul la portul serial.
- Inițial transmisia de la UART era făcută cu întreruperi, dar afecta funcționalitatea ADC-ului, așa că am schimbat cu logica de polling.
- Am primit un servomotor de 360 de grade, în loc de 180. Pentru el puteam să modific doar turația la rotire. Am cumpărat altul corespunzător cu cerințele proiectului, însă fie modul de funcționare nu corespunde cu datasheet-ul, fie e un conflict între faptul ca folosesc același ceas și pentru UART și pentru PWM. (a trebuit să reduc valorile calculate la jumătate pentru duty cycle ca să se rotească corespunzător)

8 Link-uri proiect

- Proiect Github: https://github.com/SimonaCatanoiu/Microprocesoare
- Prezentare Video: https://1drv.ms/v/s!ArAt-GPK5642iwyM5RiS01CmISQX?e=mfmfNx

9 Bibliografie

- https://github.com/undacmic/MCULabs/blob/main/Resurse/FRDM-KL25Z_ReferenceManual.pdf
- https://github.com/undacmic/MCULabs/blob/main/Resurse/FRDM-KL25Z_Datasheet.pdf
- https://github.com/undacmic/MCULabs/blob/main/Resurse/FRDM-KL25Z_Pinouts.pdf
- https://datasheetspdf.com/pdf/791970/TowerPro/SG90/1
- https://wiki.dfrobot.com/DFRobot_Ambient_Light_Sensor_SKU_DFR0026