**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

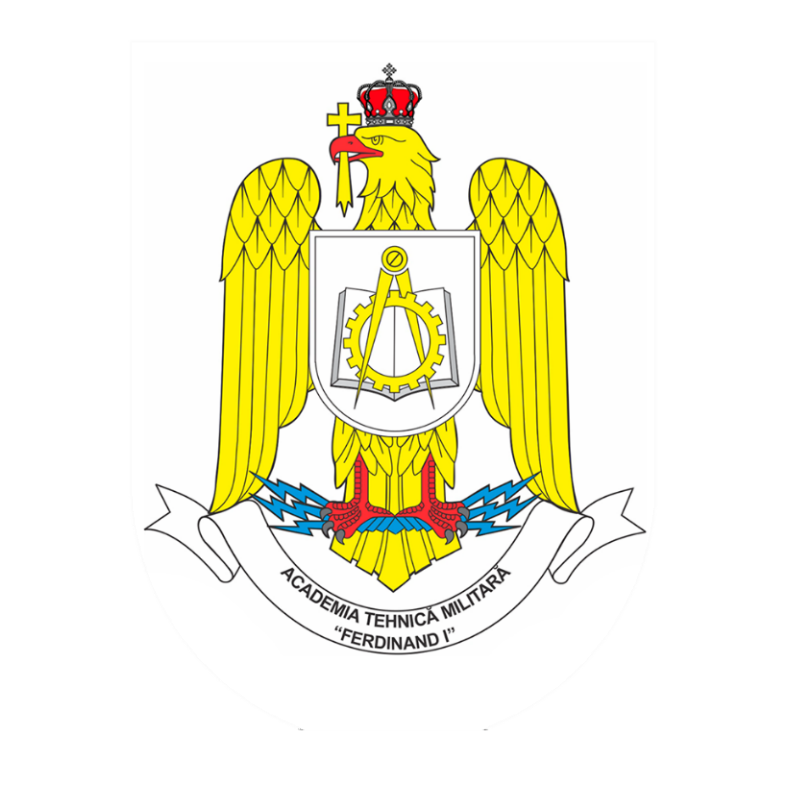
**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**FACULTATEA DE SISTEME INFORMATICE ŞI SECURITATE**

**CIBERNETICĂ**

**Specializarea: Calculatoare şi Sisteme Informatice pentru apărare şi**

**Securitate Naţională**



**Echipa nr. 30**

Std. sg. maj. Cucoș Nicoleta-Larisa

Std. sg. maj. Zugravu Paul

Grupa C114-B

**Cuprins**

[**1.** **Prezentare componente** 3](#_Toc189528111)

[**2.** **Scopul proiectului** 3](#_Toc189528112)

[**3.** **Conectare senzor – placă de dezvoltare** 4](#_Toc189528113)

[**4. Descriere program** 4](#_Toc189528114)

[**4.1. Funcția main** 4](#_Toc189528115)

[**4.2. Inițializarea modulelor** 4](#_Toc189528116)

[**4.2.1. Inițializarea modulului UART** 4](#_Toc189528117)

[**4.2.2. Inițializarea modulului GPIO** 4](#_Toc189528118)

[**4.2.3. Inițializarea modulului PIT** 7](#_Toc189528119)

[**4.2.4. Inițializarea modulului I2C** 9](#_Toc189528120)

[**4.3. Transmitere date prin UART** 9](#_Toc189528121)

[**5. Rezultate interfața Python** 10](#_Toc189528122)

[**6. Setup** 10](#_Toc189528123)

[**7. Dificultăți întâmpinate** 13](#_Toc189528124)

[**8. Referințe** 13](#_Toc189528125)

# **Prezentare componente**

1.1 !!! aici sa pui prezetarea senzorului

1.2. Prezentare LED integrat pe placă:

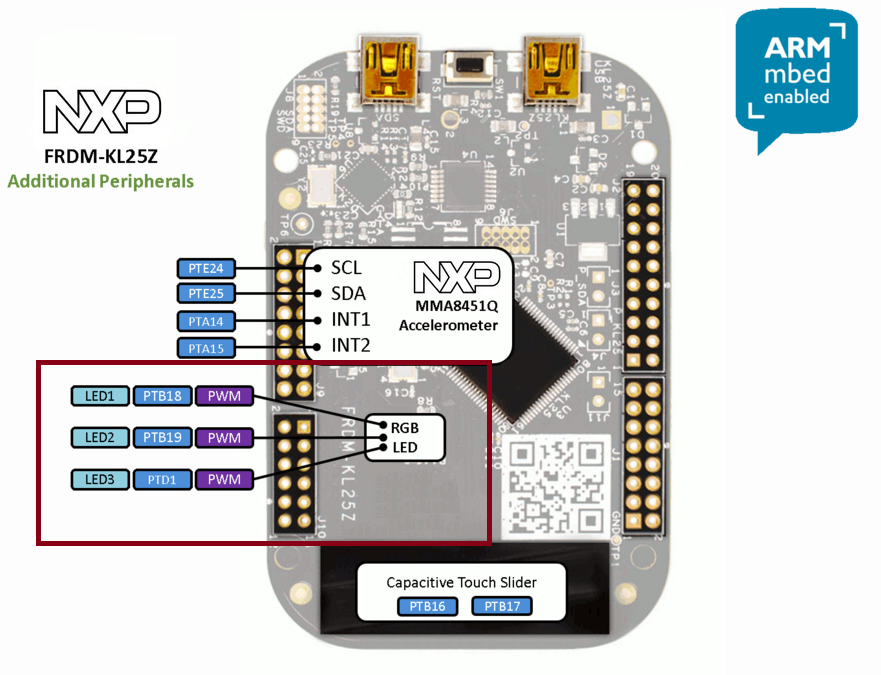


Figure 1. LED RGB

Acest LED este controlat independent de senzor, gestionând secvențe de iluminare conform unei configurări prestabilite.

LED-ul RGB de pe placa FRDM-KL25Z este un LED tricolor integrat, care poate afișa diferite culori prin combinarea celor trei componente: roșu, verde și albastru. Fiecare culoare este controlată separat prin pinii GPIO conectați(roșu-pin PTB18, verde-pin PTB19, albastru-pin PTD1)

Prin controlul intensității fiecărei culori LED-ul poate genera o gamă de culori.

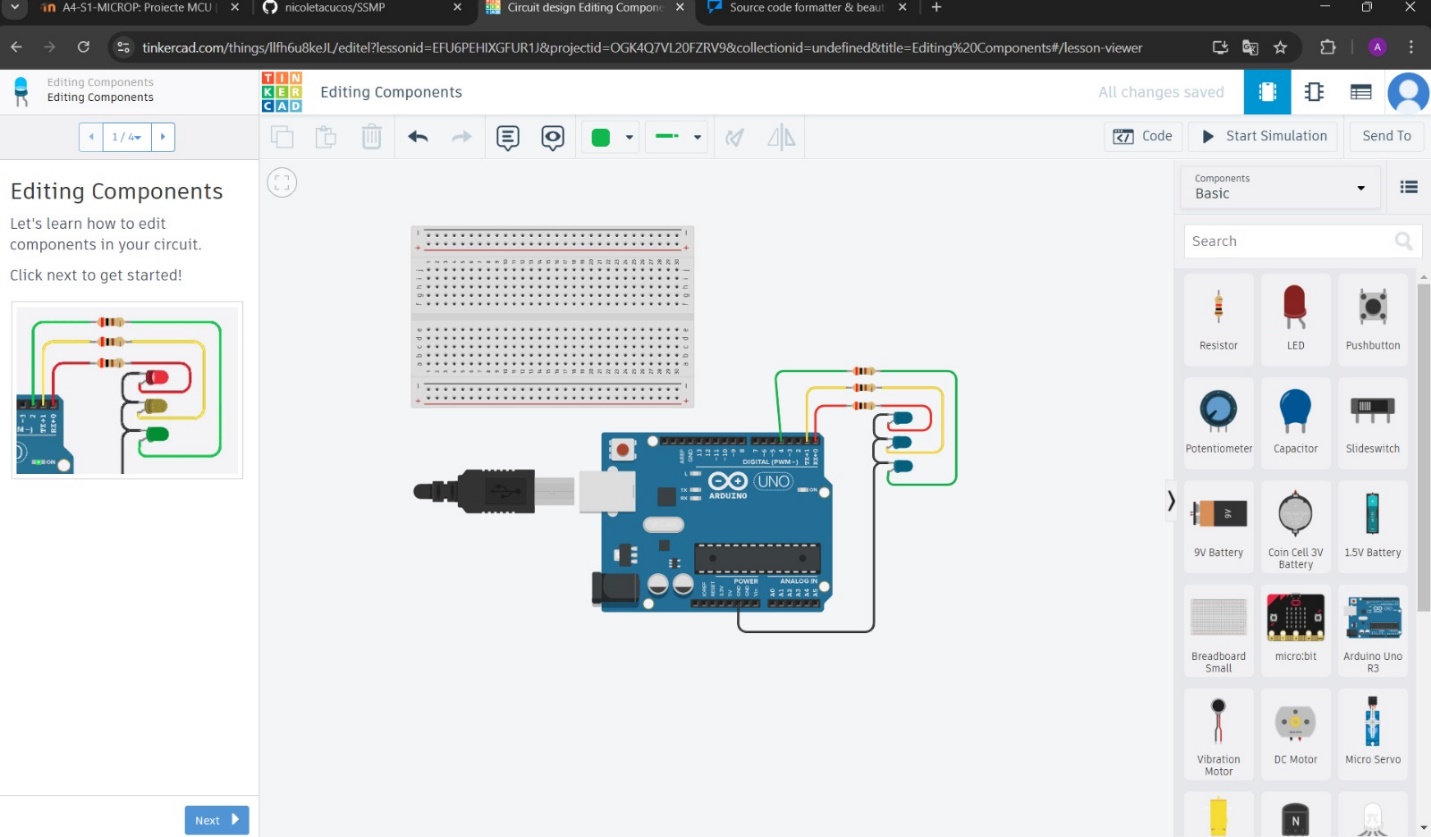
# **Scopul proiectului**

Scopul acestui proiect este dezvoltarea unui sistem integrat care să controleze iluminarea prin două componente principale: o secvență predefinită de culori pentru LED-ul RGB integrat pe placa de dezvoltare și controlul a două LED-uri separate conectate pe un breadboard, utilizând datele unui senzor inerțial pe trei axe integrat pe placă. Sistemul va răspunde în timp real la mișcările detectate pe axa X de senzor, activând LED-urile corespunzătoare (roșu pentru mișcare înainte și verde pentru mișcare înapoi), în timp ce pe o interfață Python se vor afișa statistici detaliate privind durata de funcționare a fiecărui LED. În plus, un buton dedicat va permite utilizatorului să schimbe ordinea secvenței de culori a LED-ului RGB integrat.

# **Conectare senzor – placă de dezvoltare**

**!! aici sa spui despre cum ai legat breadbordul, cu ledurile de placa**

**!!!poza de pe platforma asta**



# **4. Descriere program**

## **4.1. Funcția main**

!!! ramane la final ca sigur mai modifici pentru sensor  
pus doar codul si explicat ce se apeleaza

## **4.2. Inițializarea modulelor**

### **4.2.1. Inițializarea modulului UART**

!!!dupa ce termini partea ta de terminat aici

### **4.2.2. Inițializarea modulului GPIO**

Alegerea porturilor și a valorilor de shift pentru LED-uri a fost făcută în funcție de configurația hardware a microcontrolerului și de disponibilitatea pinilor. Fiecare LED este conectat la un pin specific, iar valoarea de shift indică poziția bitului corespunzător în registrul de control al portului.

#define RED\_LED\_SHIFT 18 // Pinul pentru LED-ul rosu pe portul B

#define GREEN\_LED\_SHIFT 19 // Pinul pentru LED-ul verde pe portul B

#define BLUE\_LED\_SHIFT 1 // Pinul pentru LED-ul albastru pe portul D

#define MASK(x) (1 << x);

Funcția **OutputPIN\_Init()** are rolul de a inițializa pinii microcontrolerului pentru a putea controla LED-urile roșu, verde și albastru, acționând ca niște întrerupătoare mici care pot fi aprinse sau stinse pentru a controla lumina LED-urilor.

Prima parte a codului se ocupă de activarea ceasului pentru porturile B și D.

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

Următoarea parte configurează fiecare pin (pentru LED-ul roșu, verde și albastru) ca ieșire. Aceasta înseamnă că microcontrolerul poate trimite semnale către acești pini pentru a aprinde sau stinge LED-urile.

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

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

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

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

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

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

Ultima parte setează direcția pinilor ca fiind ieșiri. Acești pini vor fi folosiți pentru a trimite semnale, nu pentru a primi.

PTB->PDDR |= MASK(RED\_LED\_SHIFT) | MASK(GREEN\_LED\_SHIFT);

PTD->PDDR |= MASK(BLUE\_LED\_SHIFT);

Culorile primare (roșu, verde și albastru) sunt controlate individual prin setarea sau resetarea biților corespunzători în registrele de ieșire.

Culori secundare sunt realizate prin combinarea a două culori primare, astfel se obțin culori precum magenta.



Figure 2. Secvență culori LED

Această funcție **getWhite()** aprinde toate LED-urile, rezultând culoarea albă.

void *getWhite()* {

PTB->PCOR |=MASK(RED\_LED\_SHIFT) | MASK(GREEN\_LED\_SHIFT);

PTD->PCOR |=MASK(BLUE\_LED\_SHIFT);

}

Funcția **getGreen()** aprinde doar LED-ul verde, folosindu-se de stingerea celorlalte 2 canale de cunlori.

void *getGreen()* {

PTB->PSOR=MASK(RED\_LED\_SHIFT);

PTD->PSOR=MASK(BLUE\_LED\_SHIFT);

}

Funcția următoare aprinde doar LED-ul albastru, prin stingerea LED-urilor roșu și verde.

void *getBlue()* {

PTB->PSOR=MASK(GREEN\_LED\_SHIFT);

PTB->PSOR=MASK(RED\_LED\_SHIFT);

PTD->PCOR |=MASK(BLUE\_LED\_SHIFT);

}

Ultima funcția aprinde LED-urile roșu și albastru, rezultând culoarea magenta prin stingerea LED-ului verde.

void *getMagenta()* {

PTB->PCOR |=MASK(RED\_LED\_SHIFT) | MASK(GREEN\_LED\_SHIFT);

PTD->PCOR |=MASK(BLUE\_LED\_SHIFT);

PTB->PSOR=MASK(GREEN\_LED\_SHIFT);

}

Secvența de culori poate fi fie normală, fie inversată, iar utilizatorul poate schimba între aceste două moduri printr-o comandă UART.

În secvența normală, culorile sunt schimbate într-o ordine ciclică:

**alb → verde → albastru → magenta**

În secvența inversată, culorile sunt schimbate în ordine inversă:

**alb → magenta → albastru → verde.**

Flag-ul reverse este schimbat atunci când în handler-ul de întrerupere UART (UART0\_IRQHandler) este recepționată comanda "i" din partea utilizatorului. Aceasta este trimisă printr-un buton din interfața utilizatorului a cărui funcție asociată trimite caracterul 'i'. Când acest caracter este primit, secvența de culori este inversată prin schimbarea valorii flag-ului reverse (de la 0 la 1 sau invers).

**extern uint8\_t reverse; //**0 initial

void UART0\_IRQHandler(void) {

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

c = UART0 - > D;

}

if (c == 'i') {

UART0\_TransmitString("Invert Sequence Command.\n");

**reverse = !reverse;**

}

//UART0\_Transmit(c);

}

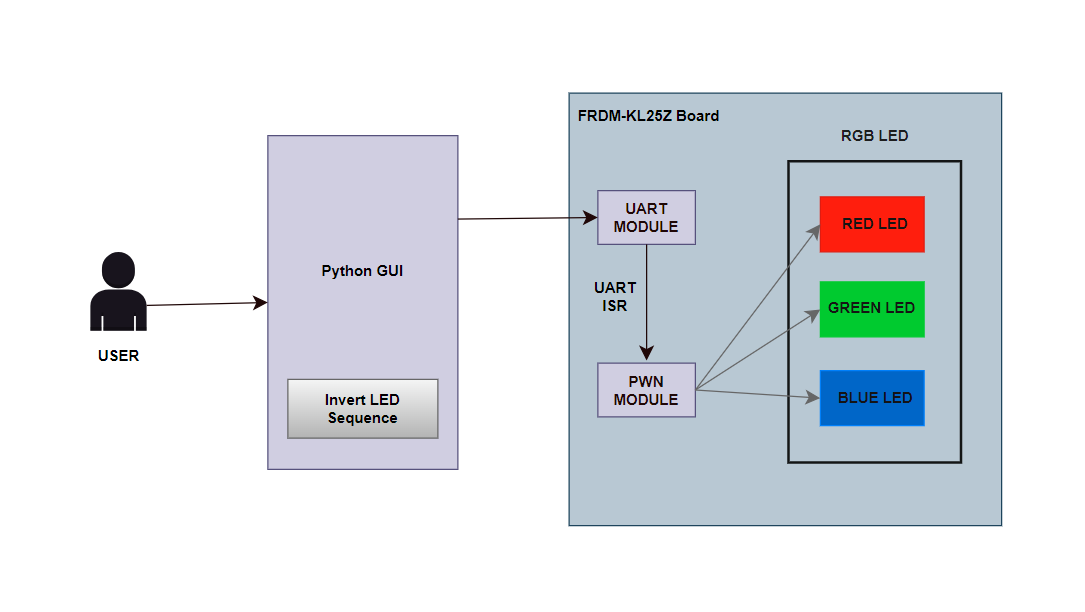


Figure 3. Schema bloc a sistemului

### **4.2.3. Inițializarea modulului PIT**

PIT (Periodic Interrupt Timer) este un periferic al microcontrolerului folosit pentru a genera întreruperi la intervale de timp precise. Este esențial pentru aplicații care necesită temporizări precise, cum ar fi controlul LED-urilor într-un anumit ritm.

PIT nu are pini externi.

Modulul PIT funcționează prin configurarea valorii inițiale în registrul "Load Value", inițierea numărării conform semnalului de ceas și, opțional, generarea unei întreruperi la expirarea timpului, moment în care microcontrolerul execută rutina ISR (Interrupt Service Routine) pentru gestionarea evenimentului, reluând automat procesul în mod periodic.

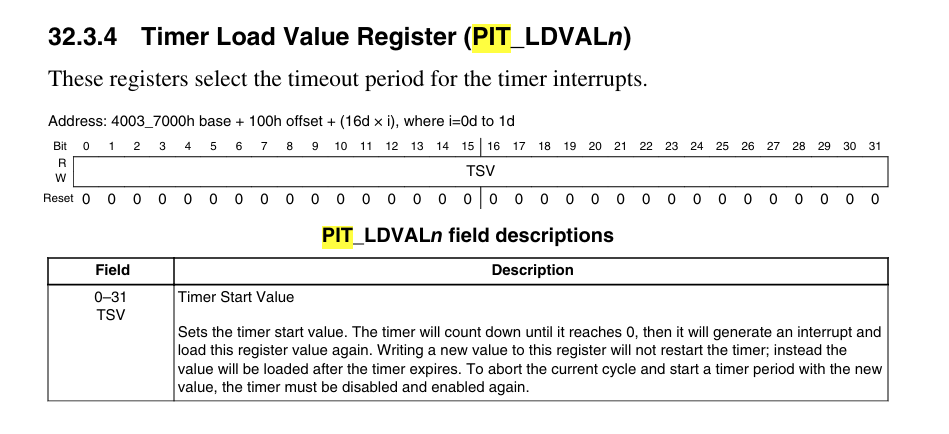


Figure 4. Setare valoare LDVAL

Diagrama modulului PIT ilustrează fluxul de funcționare al timer-ului, evidențiind cum registrul PIT\_LDVALn stabilește valoarea inițială a numărătorului, care scade până la zero, moment în care se generează o întrerupere și valoarea este reîncărcată automat pentru a relua ciclul, permițând astfel execuția de sarcini periodice în sistem.

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

Pentru a calcula valoarea LDVAL pentru o perioadă de 293 de milisecunde, cu o frecvență de ceas de 48MHz, se folosește formula: LDVAL = (0.293s \* 48.000.000 Hz) - 1, rezultând valoarea 0x27FFFF în hexazecimal. Această valoare indică numărul de cicluri de ceas necesare pentru a obține întreruperea după 293 de milisecunde.

Funcția PIT\_Init configurează și activează perifericul PIT (Periodic Interrupt Timer) al microcontrolerului pentru a genera întreruperi periodice la fiecare 293 milisecunde. Aceasta setează valoarea numărătorului PIT și activează întreruperea asociată canalului 0.

void PIT\_Init(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 0.293 secunde(293ms)

//Load Value = 0.293 sec \* 48.000.000 Hz - 1

// = 14064000 - 1

// = 0x27FFFF

PIT->CHANNEL[0].LDVAL=0x27FFFF;

// Activarea întreruperilor 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;

// Activarea întreruperii mascabile si setarea prioritatiis

NVIC\_ClearPendingIRQ(PIT\_IRQn);

NVIC\_SetPriority(PIT\_IRQn, 5);

NVIC\_EnableIRQ(PIT\_IRQn);

}

Funcția PIT\_IRQHandler este întreruperea asociată, care este apelată atunci când PIT-ul generează o întrerupere. În această funcție, se verifică dacă întreruperea a fost cauzată de un timeout și, dacă da, se schimbă starea curentă a LED-ului (alternând între culorile WHITE, GREEN, BLUE și MAGENTA) și se actualizează culoarea LED-ului corespunzător.

Primul canal (canalul 0) al PIT controlează secvența de culori a LED-urilor.

Linia if(PIT->CHANNEL[0].TFLG & PIT\_TFLG\_TIF\_MASK) verifică dacă a avut loc un timeout pe canalul 0 al PIT. Dacă da, PIT->CHANNEL[0].TFLG &= PIT\_TFLG\_TIF\_MASK; resetează flag-ul de întrerupere (TIF - Timeout Interrupt Flag), semnalizând că întreruperea a fost procesată.

Dacă un timeout a avut loc pe canalul 0, funcția controlează secvența culorilor LED-urilor, schimbându-le într-o ordine ciclică (alb → verde → albastru → magenta) sau inversată (alb → magenta → albastru → verde), în funcție de valoarea flag-ului reverse, actualizând starea curentă a LED-urilor la fiecare timeout pentru a reflecta culoarea următoare din secvență.

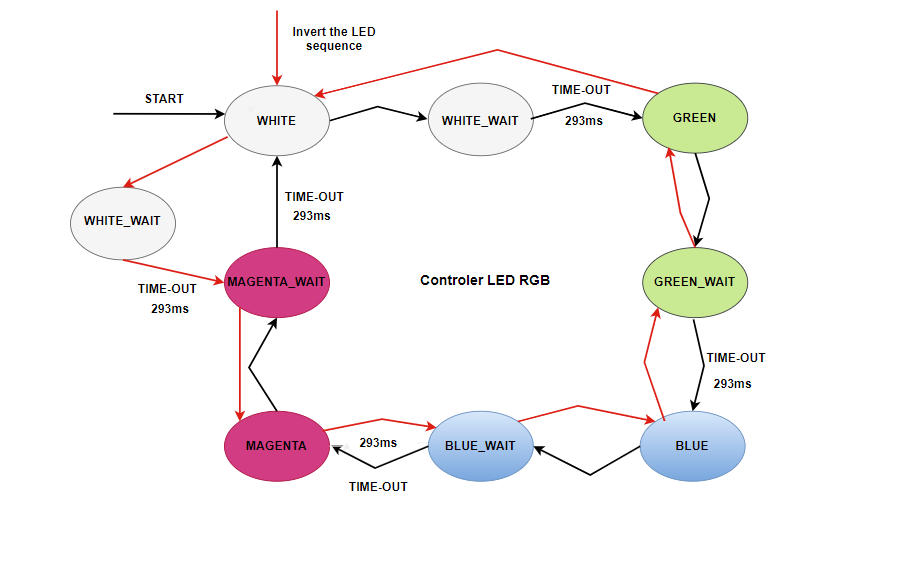


Figure 5. Schema bloc a secvenței de culori

!!! aici sa spui de al doilea canal PIT

### **4.2.4. Inițializarea modulului I2C**

!!! aici partea ta

## **4.3. Transmitere date prin UART**

!!cum trimiti coordonatele

# **5. Rezultate interfața Python**

Comunicarea serială între aplicația Python și microcontroler (placa conectată prin portul COM) se face prin trimiterea și recepționarea datelor cu biblioteca pyserial.

def \_\_init\_\_(self, port='COM5', baudrate= 14400):

super().\_\_init\_\_()

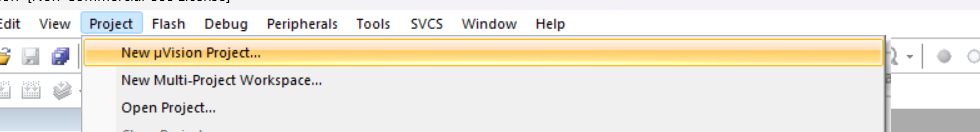
În cadrul aplicației, există un buton intitulat "Invert LED Sequence", care, atunci când este apăsat, trimite un caracter (în cazul de față 'i') către microcontroler. Această acțiune provoacă schimbarea secvenței de culori ale LED-urilor, conform funcționalității implementate pe placa de dezvoltare. Interfața grafică permite utilizatorului să vizualizeze datelor primite de la senzorul inertial și să controleze comportamentul sistemului.

!! aici o poza cu interfata si un ss la droidcam

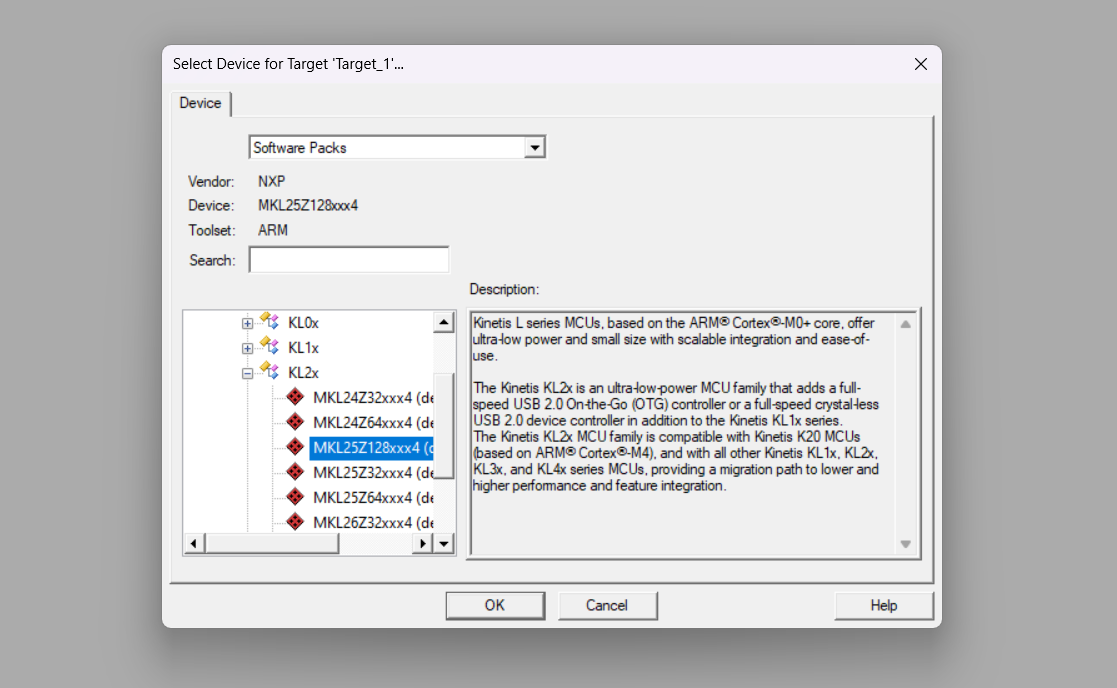
# **6. Setup**

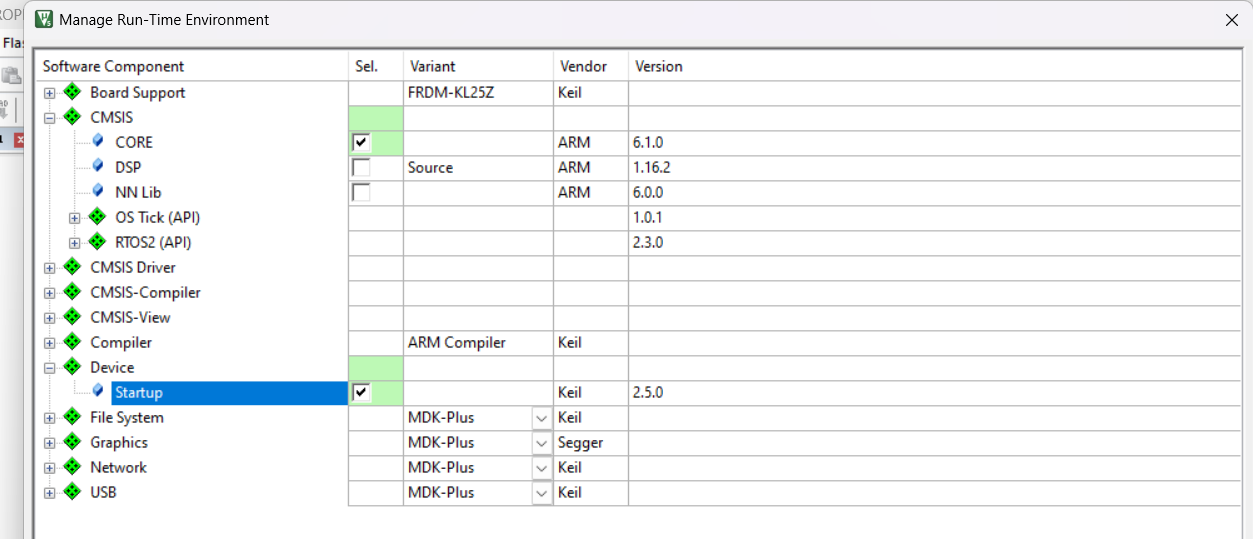
6.1. Se deschide aplicatia Keil uVision.

6.2. Se creeaza un nou proiect

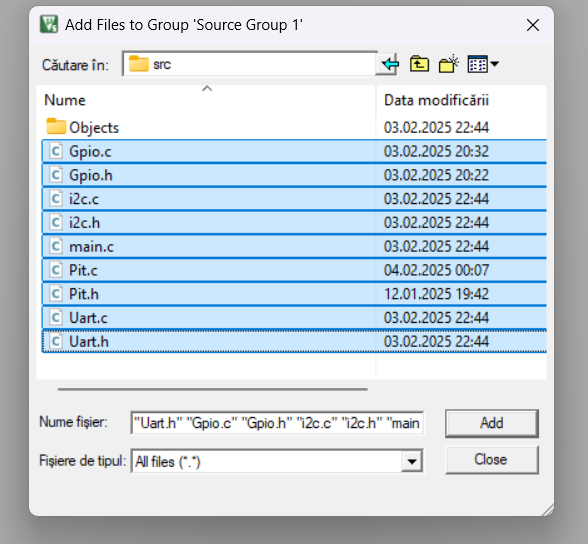


6.3. Se configurează setările microcontrolerului

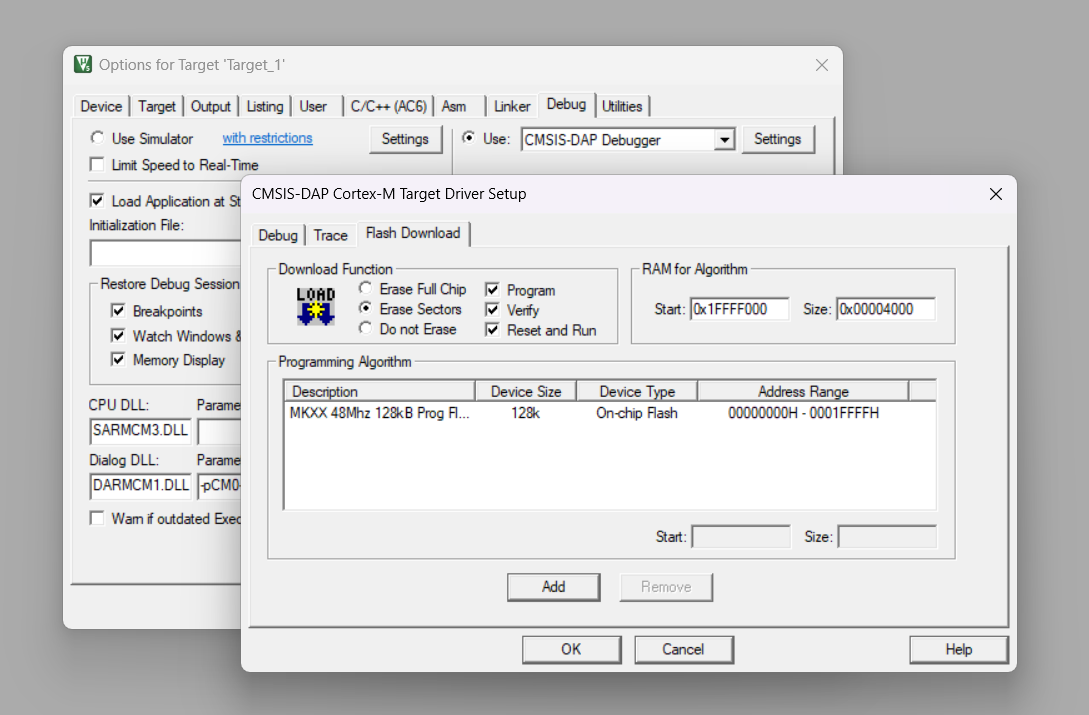




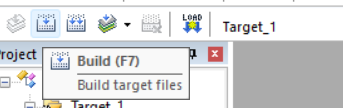
6.4. Se adauga sursele in proiect.



6.5. Se seteaza fin Flash Tools tipul de Debugger si urmatoarele casute.



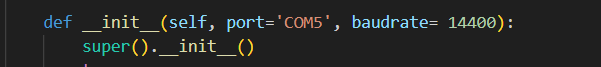
6.6. Se compilează și se încarcă proiectul în Keil.



6.7. Se conectează placa la computer prin portul USB.

6.8. Se identifică portul COM în managerul de dispozitive.

6.9. Se identifică portul COM în managerul de dispozitive.



6.10. Se rulează aplicația Python prin comanda urmatoare.

D:\Facultate\ANUL IV\MICROPROCESOARE\MICRO\SSMP\gui> **python.exe .\main.py**

6.11. Se testează comunicarea între Python și microcontroller.

# **7. Dificultăți întâmpinate**

Pentru a corecta comportamentul necorespunzător al schimbării culorilor între albastru și verde, am implementat o funcție separată care forțează aprinderea corectă a LED-ului verde în secvența inversată. În cadrul funcției getGreenInv(), se garantează că doar LED-ul verde este aprins, în timp ce celelalte LED-uri sunt stinse.

Acest proces elimină orice posibile erori de sincronizare și garantează că LED-ul verde este activat în mod corespunzător atunci când secvența este inversată.  
 void getGreenInv() {

//UART0\_TransmitString("GREEN Inv\r\n");

PTB - > PSOR = MASK(RED\_LED\_SHIFT); //stinge ledul rosu

PTD - > PSOR = MASK(BLUE\_LED\_SHIFT); //stinge ledul albastru

PTB - > PCOR |= MASK(GREEN\_LED\_SHIFT);//aprinde ledul verde

}

!!! dificultatile tale

# **8. Referințe**

1. Embedded Systems Fundamentals with ARM Cortex-M based Microcontrollers – Alexander G. Dean
2. <https://wiki.mta.ro/c/4/ssmp/lab/lab5>
3. KL25 Sub-Family Reference Manual
4. https://github.com/undacmic/MCULabs/blob/main/Resurse/FRDM-KL25Z\_Pinouts.pdf