**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

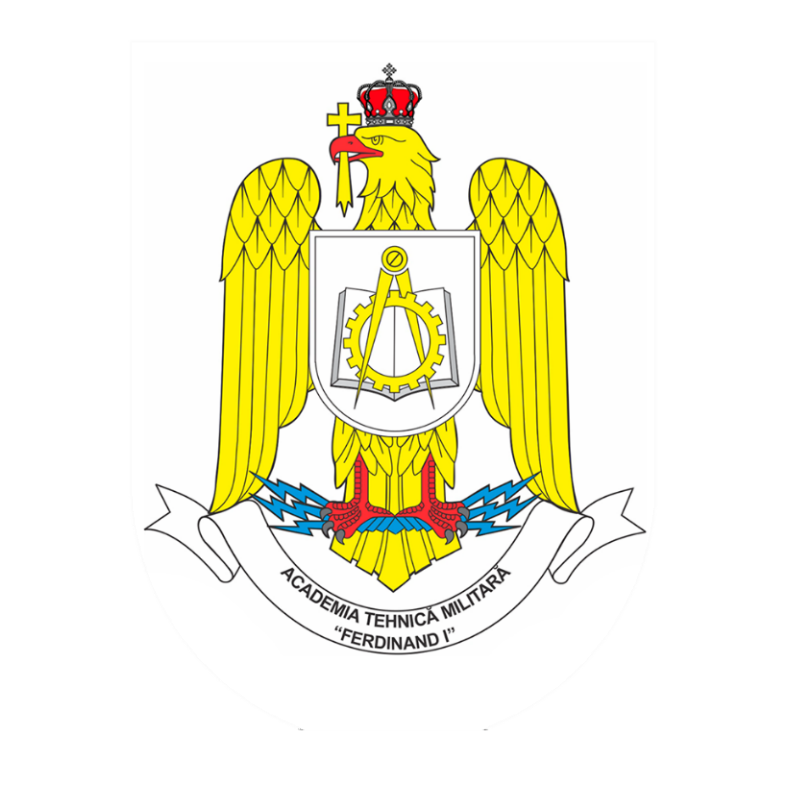
**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**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 C114B

**Cuprins**

[**1.** **Prezentare componente** 3](#_Toc188281752)

[**2.** **Scopul proiectului** 3](#_Toc188281753)

[**3.** **Conectare senzor – placă de dezvoltare** 4](#_Toc188281754)

[**4. Descriere program** 4](#_Toc188281755)

[**4.1. Funcția main** 4](#_Toc188281756)

[**4.2. Inițializarea modulelor** 4](#_Toc188281757)

[**4.2.1. Inițializarea modulului UART** 4](#_Toc188281758)

[**4.2.2. Inițializarea modulului GPIO** 4](#_Toc188281759)

[**4.2.3. Inițializarea modulului PIT** 5](#_Toc188281760)

[**4.3. Transmitere date prin UART** 8](#_Toc188281761)

[**5. Rezultate interfața Python** 8](#_Toc188281762)

[**6. Dificultăți întâmpinate** 8](#_Toc188281763)

[**7. Referințe** 8](#_Toc188281764)

# **Prezentare componente**

!!! aici sa pui prezetarea senzorului

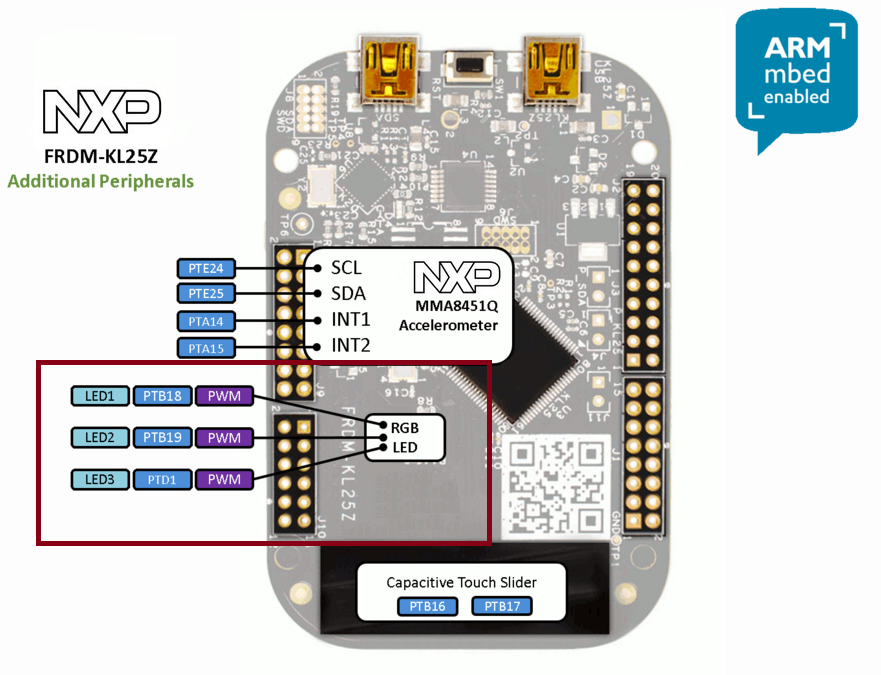


Figure 1. LED RGB

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 largă 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. 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, datele senzorului vor fi filtrate pentru a elimina zgomotul și pentru a asigura o funcționare precisă.

# **Conectare senzor – placă de dezvoltare**

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

# **4. Descriere program**

## **4.1. Funcția main**

! ramane la final

## **4.2. Inițializarea modulelor**

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

!dupa ce termini partea ta

### **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. Aceasta este o confirmare suplimentară că 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 secundare precum magenta.



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

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);

}

### **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.

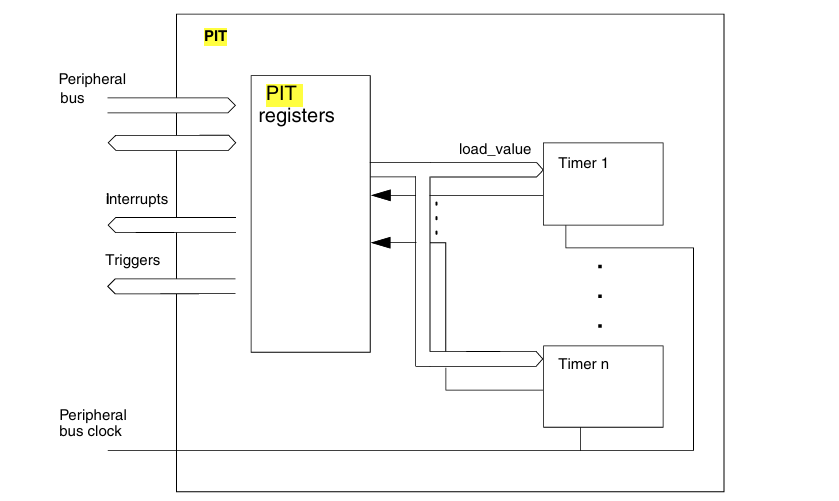


Figure 2. Diagrama bloc pentru PIT

Diagrama arată cum un modul PIT poate fi folosit pentru a crea mai multe temporizatoare care pot fi setați pentru a număra până la zero și a genera un semnal atunci când ajung la zero.

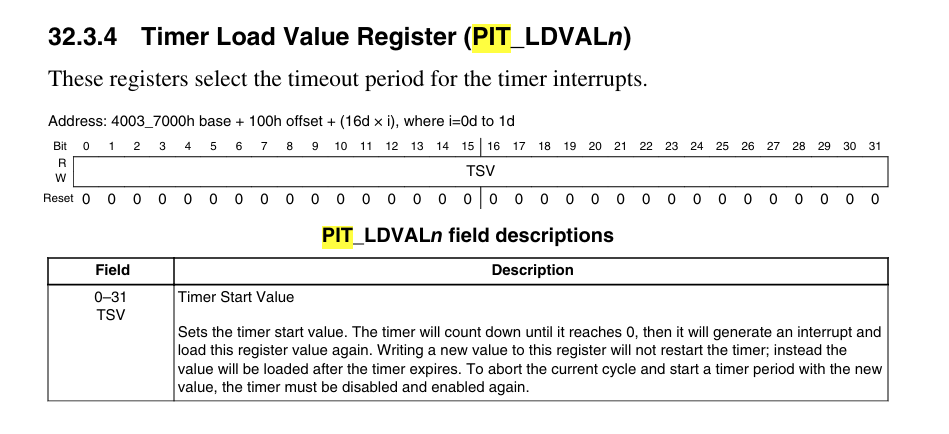


Figure 3. Setare valoare LDVAL

Prin modificarea valorii LDVAL, se poate controla cât de des se generează întreruperea.

**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.

void PIT\_IRQHandler(void) {

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

//timeout occured

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

switch (current\_state) {

case WHITE:

getWhite();

current\_state=GREEN;

break;

case GREEN:

getGreen();

current\_state=BLUE;

break;

case BLUE:

getBlue();

current\_state=MAGENTA;

break;

case MAGENTA:

getMagenta();

current\_state=WHITE;

break;

}

}

}

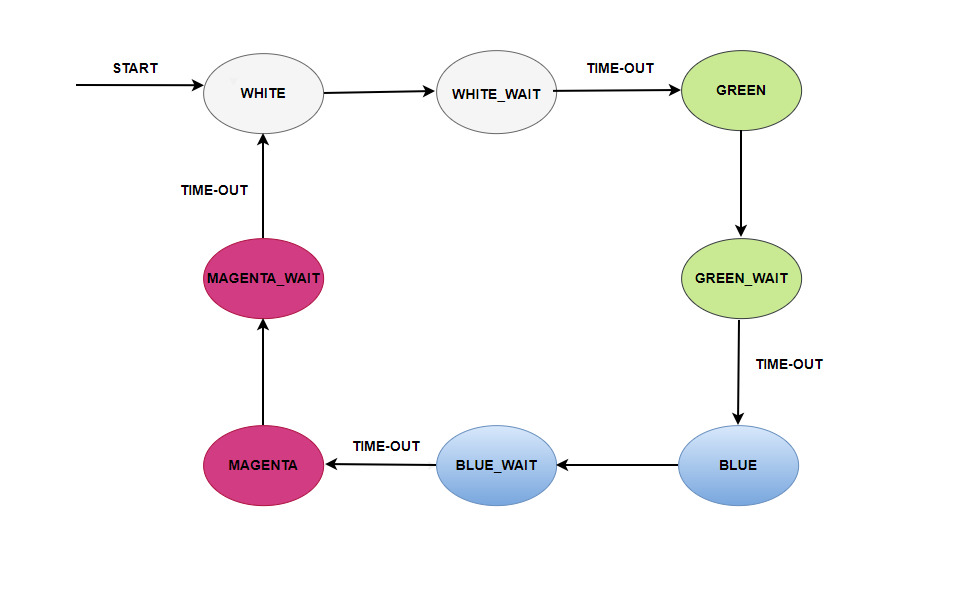


Figure 4. Automat finit pentru stările culorilor și stările de așteptare adăugate

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

# **5. Rezultate interfața Python**

Comunicarea serială între aplicația Python și microcontroler (placa conectată prin portul COM) se face prin trimiterea și recepționarea datelor via biblioteca pyserial. Datele sunt trimise sub formă de șiruri de caractere și sunt citite periodic din portul serial. Interfața grafică permite utilizatorului să vizualizeze aceste date.

# **6. Dificultăți întâmpinate**

# **7. 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