ROMÂNIA
MINISTERUL APĂRĂRII NAȚIONALE
ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I” – BUCUREȘTI

FACULTATEA DE SISTEME INFORMATICE ȘI SECURITATE CIBERNETICĂ

CIRCUIT ELECTRONIC DE ACȚIONARE A BECURILOR LED PRIN INTERMEDIUL UNUI SENZOR DE ROTAȚIE UTILIZÂND PLATFORMA DE DEZVOLTARE FRDM-KL25Z

Realizat de:
Sd. Sg. Maj. IONEL Ana-Maria
Sd. Sg. Maj. GRIGORE Maria-Emilia
Grupa C114-C

Profesor îndrumător:
Cpt. Dr. Ing. COCA Mihai

București
2024

Cuprins

1.Scopul proiectului

Scopul proiectului constă în acționarea unor becuri LED prin intermediul unui senzor de rotație (potențiometru), în funcție de poziția cursorului acestuia, astfel:

- 0 – 1.1 V = **verde**, galben, roșu;
- 1.1 – 2.2 V = **verde**, **galben**, roșu;
- 2.2 – 3.3 V = **verde**, **galben**, **roșu**.

Printre alte funcționalități pe care le prezintă aplicația se numără modificarea culorilor becului LED integrat în microcontroller în secvența: alb, verde, turcoaz, galben, fiecare culoare schimbându-se la un interval de 461 ms.

O aplicabilitate practică a proiectului ar putea fi în cadrul unui sistem de iluminare ambientală a unei camere, în funcție de preferințele unei persoane. Un alt scenariu în care acest proiect s-ar putea dovedi util ar fi în cadrul unui joc de societate, în care fiecare combinație de culori are o anumită semnificație sau impune o anumită regulă.

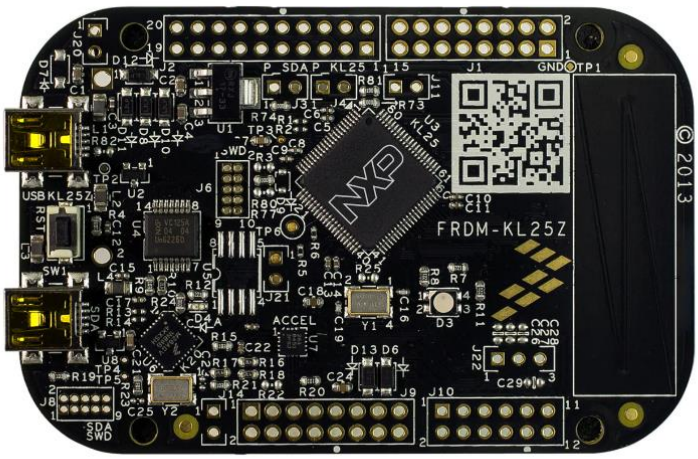
2.Configurare

2.1. Componente hardware

Componentele hardware cu ajutorul cărora a fost realizat proiectul sunt:

- **Placă de dezvoltare FRDM-KL25Z**

  FRDM-KL25Z este o platformă de dezvoltare hardware bazată pe un microcontroler de la NXP (Freescale), modelul KL25Z. Acesta este utilizat pentru dezvoltare de aplicații încorporate, oferind acces la numeroase periferice pentru a crea și testa aplicații embedded.

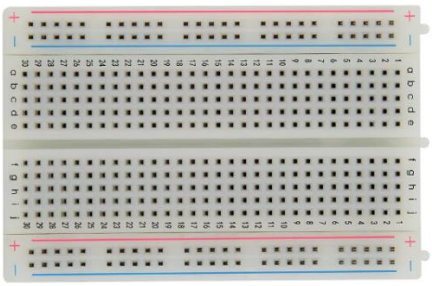


- **Potențiometru**

Un potențiometru este un dispozitiv electronic ajustabil cu rezistență variabilă, care permite controlul manual al nivelului de tensiune sau curent într-un circuit.

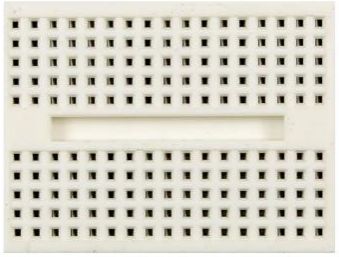
- **Breadboard**

Un breadboard este o placă de testare utilizată în electronică pentru conexiunea temporară a componentelor electronice, fără a necesita sudură, permițând testarea și experimentarea rapidă cu circuite electronice fără a le fixa permanent.



- **Mini breadboard**

Un mini breadboard este o versiune compactă a unei breadboard half-sized; l-am folosit pentru a fixa potențiometrul, evitând astfel conexiunile instabile ce ar fi putut apărea în cazul utilizării firelor directe.



- 3 LED-uri: roșu, galben, verde

LED-urile (diodă emițătoare de lumină) sunt dispozitive electronice care convertesc curentul electric în lumină.

- Fire de conexiune

Firele de conexiune sunt utilizate pentru a stabili și a asigura transmiterea semnalului electric între componentele proiectului.

- 3 rezistențe 0.1

Rezistențele sunt componente electronice utilizate pentru a limita curentul și a preveni deteriorarea (degradarea) becurilor LED.

2.2. Periferice

Perifericele reprezintă module cu ajutorul cărora se gestionează fiecare componentă a proiectului la nivel software.

- **UART** (Universal Asynchronous Receiver/Transmitter) - În contextul comunicării seriale asincrone între GUI și server, UART facilitează schimbul de date între acestea. Prin transmiterea și recepționarea informațiilor în format serial, acest protocol asigură o comunicare eficientă și sincronizată între componentele sistemului.

- **GPIO** (General Purpose Input/Output) - Prin configurarea piniilor GPIO, se poate controla activarea și dezactivarea LED-urilor atât de pe breadboard, cât și de pe placa de dezvoltare.

- **PIT** (Periodic Interrupt Timer) - Pentru gestionarea evenimentelor la intervale precise, PIT furnizează întreruperi periodice la intervale de timp definite, în cazul de față pentru a schimba o secvență de 4 culori pe LED-ul de pe placa de dezvoltare la un interval specific de 461 ms.

- **ADC** (Analog-to-Digital Converter) - Pentru preluarea semnalului analogic de la senzorul de rotație și transformarea acestuia într-un semnal digital, ADC joacă un rol esențial. Convertorul analog-digital convertește semnalele analogice provenite de la potențiometru în valori digitale, prelucrate ulterior pentru acționarea becurilor LED de pe breadboard.

2.3. Implementare software

2.3.1. Server

Partea de server cuprinde un fișier main prin care este gestionat workflow-ul aplicației și câte 2 fișiere asociate fiecărui modul, source (*.c) și library (*.h).

În main există o funcție de interpretare a semnalelor primite pe canalul de comunicație UART, sub forma unor flag-uri. Pentru apelul în mod corespunzător al funcțiilor modulelor, am conceput un protocol care schimbă valorile flag-urilor în funcție de un caracter uint8_t sugestiv:

```
 1.   void   interpret_flags(uint8_t*   receivedChar,   uint8_t*   change_sequence,   uint8_t*
individual_color, uint8_t* duration_flag, uint8_t* stop_led)
 2. {
 3.       if (is_UART_DataAvailable()) {
 4.                (*receivedChar) = UART_Receive();
 5.       }
 6.       if((*receivedChar) == 'M')
 7.       {
 8.                (*change_sequence) = '1';
 9.                (*individual_color) = 'X';
10.       }
11.       else
12.       {
```

```
13.                 (*change_sequence) = '0';
14.                 (*individual_color) = 'X';
15.         }
16.
17.         if((*receivedChar == 'N') && (*change_sequence) == 1)
18.         {
19.                 (*change_sequence) = '0';
20.                 (*individual_color) = 'X';
21.                 (*stop_led) = 'q';
22.         }
23.
24.         if((*receivedChar) == 'W')
25.                 (*individual_color) = 'W';
26.         if((*receivedChar) == 'G')
27.                 (*individual_color) = 'G';
28.         if((*receivedChar) == 'T')
29.                 (*individual_color) = 'T';
30.         if((*receivedChar) == 'Y')
31.                 (*individual_color) = 'Y';
32.
33.         if((*receivedChar) == '1')
34.                 (*duration_flag) = '1';
35.    if((*receivedChar) == '2')
36.                 (*duration_flag) = '2';
37.         if((*receivedChar) == '3')
38.                 (*duration_flag) = '3';
39.         if((*receivedChar) == '4')
40.                 (*duration_flag) = '4';
41.
42.         if((*receivedChar) == 'w')
43.                 (*stop_led) = 'w';
44.         if((*receivedChar) == 'g')
45.                 (*stop_led) = 'g';
46.         if((*receivedChar) == 't')
47.                 (*stop_led) = 't';
48.         if((*receivedChar) == 'y')
49.                 (*stop_led) = 'y';
50.
51. }
52.
```

Funcția main apelează pe rând funcțiile de inițializare a fiecărui modul în parte și pornește o buclă infinită în care sunt apelate funcțiile de tratare a întreruperilor pentru fiecare modul, cât și funcția de interpretare a flag-urilor. Totodată, se realizează transmiterea unui mesaj care indică tensiunea indicată de cusrosul potențiometrului prin intermediul canalului UART0.

```
 1. int main(void) {
 2.
 3.         uint8_t receivedChar = 'X';
 4.
 5.         UART0_Init(19200);
 6.         RGBLed_Init();
 7.         ADC0_Init();
 8.         PIT_Init();
 9.
10.         while(1) {
11.         interpret_flags(&receivedChar, &change_sequence, &individual_color, &duration_flag,
&stop_led);
12.         PIT_IRQHandler();
13.         ADC0_IRQHandler();
14.         UART0_IRQHandler();
15.
16.                 if(flag)
17.                 {
18.                         uint8_t parte_zecimala = (uint8_t) measured_voltage;
19.                         uint8_t parte_fractionara1 = ((uint8_t)(measured_voltage * 10)) %
10;
```

```
                        uint8_t parte_fractionara2 = ((uint8_t)(measured_voltage * 100)) %
10;


                        UART0_Transmit('V');
                        UART0_Transmit(' ');
                        UART0_Transmit('=');
                        UART0_Transmit(' ');
                        UART0_Transmit(parte_zecimala + 0x30);
                        UART0_Transmit('.');
                        UART0_Transmit(parte_fractionara1 + 0x30);
                        UART0_Transmit(parte_fractionara2 + 0x30);
                        UART0_Transmit('V');
                        UART0_Transmit(0x0A);
                        UART0_Transmit(0x0D);
                        /* UART0_Transmit(receivedChar); */
                        flag = 0;
       }
      }

}

```

Perifericele utilizate în cadrul proiectului și implementarea lor:

- **UART**

Modulul UART0 se folosește pentru comunicația serială între potențiometru, clientul GUI și server.

Se realizează inițializarea modulului UART0:

```
1. void UART0_Init(uint32_t baud_rate)
2. {
3.   uint32_t osr = 15; /* Over-Sampling Rate (numarul de esantioane luate per bit-time) */
4.
5.       /* Setarea sursei de ceas pentru modulul UART */
6.       SIM->SOPT2 |= SIM_SOPT2_UART0SRC(01);
7.
8.       /* Activarea semnalului de ceas pentru modulul UART */
9.       SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;
10.
11.      /* Activarea semnalului de ceas pentru portul A
12.        intrucat dorim sa folosim pinii PTA1, respectiv PTA2 pentru comunicarea UART */
13.      SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;
14.
15.      /* Fiecare pin pune la dispozitie mai multe functionalitati
16.        la care avem acces prin intermediul multiplexarii */
17.      PORTA->PCR[1] = ~PORT_PCR_MUX_MASK;
18.      PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); /* Configurare RX pentru UART0 */
19.      PORTA->PCR[2] = ~PORT_PCR_MUX_MASK;
20.      PORTA->PCR[2] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); /* Configurare TX pentru UART0 */
21.
22.
23.      UART0->C2 &= ~((UART0_C2_RE_MASK) | (UART0_C2_TE_MASK));
24.
25.      /* Configurare Baud Rate */
26.
27.      /* SBR - vom retine valoarea baud rate-ului calculat pe baza frecventei ceasului de
sistem
28.                SBR  -            b16 b15 b14 [b13 b12 b11 b10 b09                b08   b07
b06 b05 b04 b03 b02 b01] &
29.          0x1F00 -                0                  0   0    1   1   1   1   1      0   0
0   0   0   0   0   0
30.                    0   0   0    b13 b12 b11 b10 b09    0   0   0   0   0   0   0   0 >> 8
31.           BDH  -   0   0   0    b13 b12 b11 b10 b09
32.           BDL  -   b08 b07 b06  b05 b04 b03 b02 b01 */
33.      uint16_t sbr = (uint16_t)((DEFAULT_SYSTEM_CLOCK)/(baud_rate * (osr+1)));
```

```
34.       uint8_t temp = UART0->BDH & ~(UART0_BDH_SBR(0x1F));
35.       UART0->BDH = temp | UART0_BDH_SBR(((sbr & 0x1F00)>> 8));
36.       UART0->BDL = (uint8_t)(sbr & UART_BDL_SBR_MASK);
37.       UART0->C4 |= UART0_C4_OSR(osr);
38.
39.
40.       /* Setare numarul de biti de date la 8 si fara bit de paritate */
41.       UART0->C1 = 0;
42.
43.       /* Dezactivare intreruperi la transmisie */
44.       UART0->C2 |= UART0_C2_TIE(0);
45.       UART0->C2 |= UART0_C2_TCIE(0);
46.
47.       /* Activare intreruperi la receptie */
48.       UART0->C2 |= UART0_C2_RIE(1);
49.
50.       UART0->C2 |= ((UART_C2_RE_MASK) | (UART_C2_TE_MASK));
51.
52.       UART0->C2 &= ~((UART0_C2_RE_MASK) | (UART0_C2_TE_MASK));
53. }
54.
```

- **PIT**

Acest modul este necesar configurării frecvenței de timp la care se schimbă culoarea LED-ului integrat pe platformă. Astfel, am inițializat modulul:

```
 1. void PIT_Init(void) {
 2.
 3.       /* Activarea semnalului de ceas pentru perifericul PIT */
 4.       SIM->SCGC6 |= SIM_SCGC6_PIT_MASK;
 5.       /* Utilizarea semnalului de ceas pentru tabloul de timere */
 6.       PIT_MCR &= ~PIT_MCR_MDIS_MASK;
 7.       /* Oprirea decrementarii valorilor numaratoarelor in modul debug */
 8.       PIT->MCR |= PIT_MCR_FRZ_MASK;
 9.       /* Setarea valorii numaratorului de pe canalul 0 la o perioada de 1 secunda */
10.
11.       /*461ms:*/
12.       PIT->CHANNEL[0].LDVAL = 0xAB917D ;
13.
14.    /* Activarea intreruperilor pe canalul 0 */
15.       PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TIE_MASK;
16.       /* Activarea timerului de pe canalul 0 */
17.       PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TEN_MASK;
18.
19.       /* Activarea intreruperii mascabile si setarea prioritatii */
20.       NVIC_ClearPendingIRQ(PIT_IRQn);
21.       NVIC_SetPriority(PIT_IRQn,5);
22.       NVIC_EnableIRQ(PIT_IRQn);
23.
24.       led_state=0;
25. }
26.
```

În funcția de tratare a întreruperii de PIT am implementat multiple use-case-uri, accesibile din interfața grafică, astfel:

- Frecvență default: culorile LED-ului de modifică la un interval de 461 ms

Pentru a atinge acest scop, observăm în Reference Manual, în capitolul 32 – Periodic Interrupt Timer, subcapitolul 32.3.4 - Timer Load Value Register (PIT_LDVALn), următorul aspect:

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

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

Registrul care trebuie populat în vederea modificării frecvenței timer-ului este LDVAL[0], (0 deoarece folosim primul timer). Urmărind explicațiile din laborator și secțiunea 32.5 a aceluiași capitol, am calculat valoarea hexazecimală cu care trebuie populat registrul LDVAL.

32.5 Initialization and application information

In the example configuration:

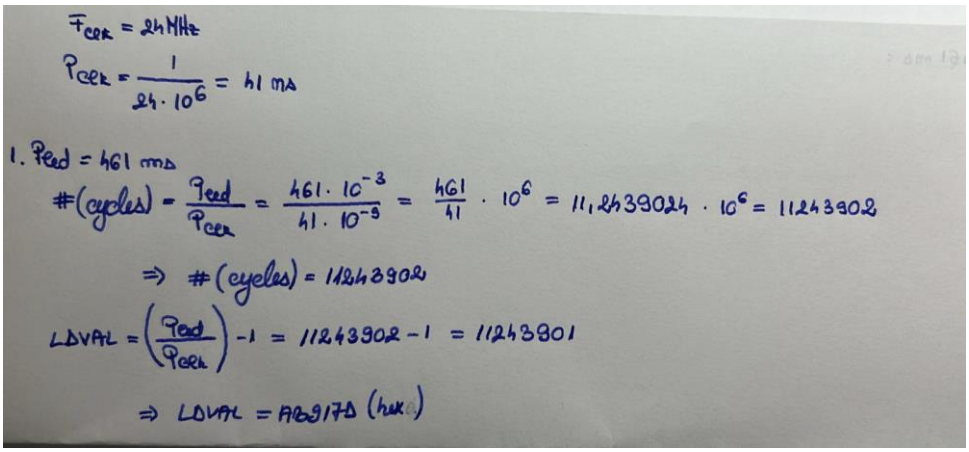
- The PIT clock has a frequency of 50 MHz.

- Timer 1 creates an interrupt every 5.12 ms.
- Timer 3 creates a trigger event every 30 ms.

The PIT module must be activated by writing a 0 to MCR[MDIS].

The 50 MHz clock frequency equates to a clock period of 20 ns. Timer 1 needs to trigger every 5.12 ms/20 ns = 256,000 cycles and Timer 3 every 30 ms/20 ns = 1,500,000 cycles. The value for the LDVAL register trigger is calculated as:

LDVAL trigger = (period / clock period) -1

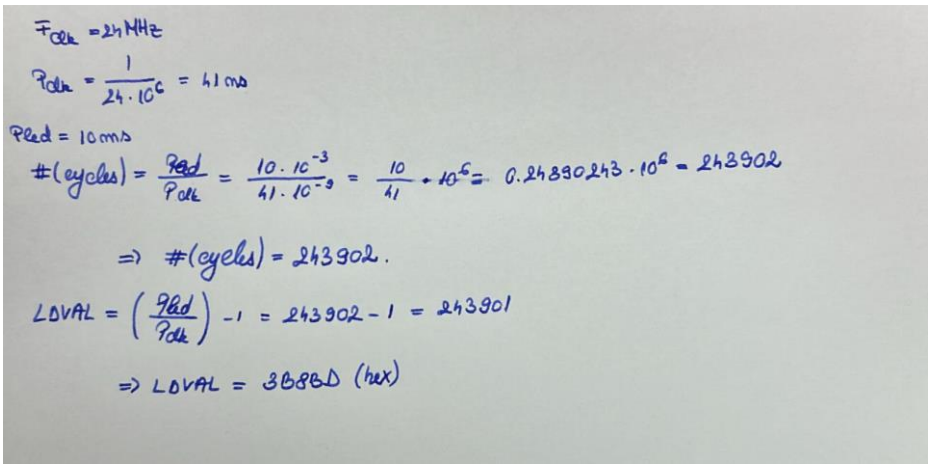


Pentru gestionarea acestei funcționalități, se utilizează un flag, *duration_flag*, care ia valori diferite în funcție de semnalul primit de la GUI prin canalul de recepție UART:

```
1. void PIT_IRQHandler(void) {
2.
3.        /*461ms*/
4.        if(duration_flag == '4')
5.                  PIT->CHANNEL[0].LDVAL = 0xAB917D ;
6.
```

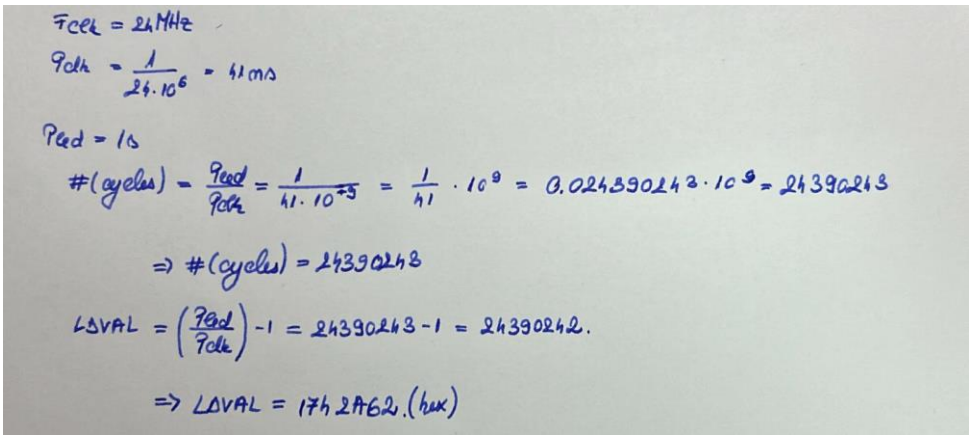
Am urmărit aceiași pași și pentru următoarele frecvențe de modificare a culorilor LED-ului:

- Frecvență de schimbare a culorilor de 10 ms



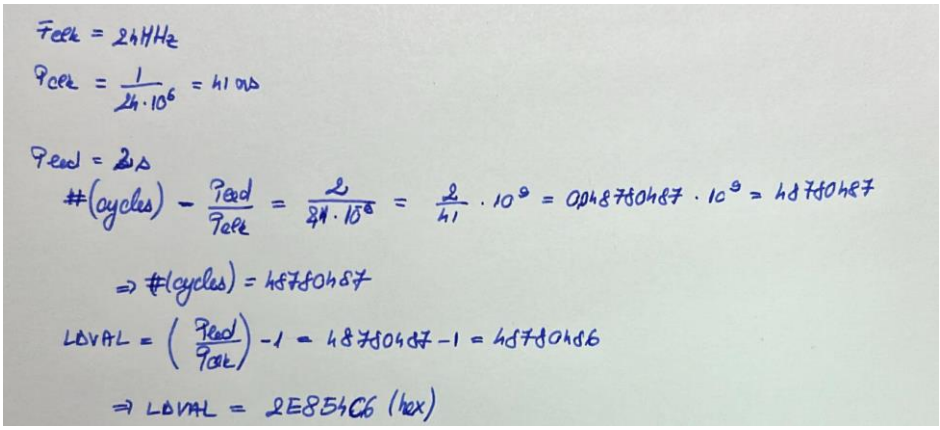

```
1.        /*10 ms*/
2.        if(duration_flag == '1')
3.                  PIT->CHANNEL[0].LDVAL = 0x25376F;
4.
```

- Frecvență de schimbare a culorilor de 1 s




```
1. /*1 s*/
2.        if(duration_flag == '2')
3.                  PIT->CHANNEL[0].LDVAL = 0x1742A62 ;
4.
```

- Frecvență de schimbare a culorilor de 2 s




```
1. /*2 s*/
2.         if(duration_flag == '3')
3.                 PIT->CHANNEL[0].LDVAL = 0x2E854C9 ;
4.
```

- Ordine default: LED-ul are pe rând culorile: alb, verde, turcoaz, galben

Am definit pinii la care sunt conectate cele 3 LED-uri RGB ce compun LED-ul de pe microcontroller:

```
1. #define RED_LED_PIN (18) /* PORT B */
2. #define GREEN_LED_PIN (19) /* PORT B */
3. #define BLUE_LED_PIN (1) /* PORT D */
4.
```

Am manipulat, în funcție de valoarea flagului *led_state*, ce LED-uri se vor aprinde și în ce ordine, ținând cont că obținem culoarea alb prin activarea tuturor celor 3 LED-uri, galben prin activarea LED-urilor verde și roșu, turcoaz prin LED-urile verde și albastru. Având o schemă de conectare a LED-urilor și a rezistențelor de tip Pull-Up, GPIOx_PCOR (clear) va determina aprinderea LED-ului, iar GPIOx_PSOR (set), stingerea acestuia.

```
 1.                 if(led_state == 0)
 2.                 {
 3.                         /* WHITE */
 4.                         GPIOB_PSOR |= (1<<GREEN_LED_PIN);
 5.                         GPIOB_PSOR |= (1<<RED_LED_PIN);
 6.
 7.                         if(stop_led == 'w')
 8.                         {
 9.                                 GPIOB_PSOR |= (1<<RED_LED_PIN);
10.                                 GPIOD_PSOR |= (1<<BLUE_LED_PIN);
11.                                 GPIOB_PSOR |= (1<<GREEN_LED_PIN);
12.                         }
13.                         else
14.                         {
15.                                 GPIOB_PCOR |= (1<<RED_LED_PIN);
16.                                 GPIOD_PCOR |= (1<<BLUE_LED_PIN);
17.                                 GPIOB_PCOR |= (1<<GREEN_LED_PIN);
18.                                 stop_led = 'x';
19.                         }
20.                         led_state = 1;
21.                 }
```

```
22.                 else if (led_state == 1)
23.                 {
24.                           /* GREEN */
25.                           GPIOB_PSOR |= (1<<RED_LED_PIN);
26.                           GPIOD_PSOR |= (1<<BLUE_LED_PIN);
27.                           GPIOB_PSOR |= (1<<GREEN_LED_PIN);
28.
29.                           if(stop_led == 'g')
30.                           {
31.                                     GPIOB_PSOR |= (1<<RED_LED_PIN);
32.                                     GPIOD_PSOR |= (1<<BLUE_LED_PIN);
33.                                     GPIOB_PSOR |= (1<<GREEN_LED_PIN);
34.
35.                           }
36.                           else
37.                           {
38.                                     GPIOB_PCOR |= (1<<GREEN_LED_PIN);
39.                                     stop_led = 'x';
40.                           }
41.                           led_state = 2;
42.                 }
43.                 else if (led_state == 2)
44.                 {
45.                           /* TURQUOISE */
46.                           GPIOB_PSOR |= (1<<GREEN_LED_PIN);
47.
48.                           if(stop_led == 't')
49.                           {
50.                                     GPIOB_PSOR |= (1<<RED_LED_PIN);
51.                                     GPIOD_PSOR |= (1<<BLUE_LED_PIN);
52.                                     GPIOB_PSOR |= (1<<GREEN_LED_PIN);
53.                           }
54.                           else
55.                           {
56.                                     GPIOB_PCOR |= (1<<GREEN_LED_PIN);
57.                                     GPIOD_PCOR |= (1<<BLUE_LED_PIN);
58.                                     stop_led = 'x';
59.                           }
60.                           led_state = 3;
61.                 }
62.                 else if (led_state == 3)
63.                 {
64.                           /* YELLOW */
65.                           GPIOB_PSOR |= (1<<GREEN_LED_PIN);
66.                           GPIOD_PSOR |= (1<<BLUE_LED_PIN);
67.                           if(stop_led == 'y')
68.                           {
69.                                     GPIOB_PSOR |= (1<<RED_LED_PIN);
70.                             GPIOD_PSOR |= (1<<BLUE_LED_PIN);
71.                             GPIOB_PSOR |= (1<<GREEN_LED_PIN);
72.                           }
73.                           else
74.                           {
75.                                     GPIOB_PCOR |= (1<<GREEN_LED_PIN);
76.                                     GPIOB_PCOR |= (1<<RED_LED_PIN);
77.                                     stop_led = 'x';
78.                           }
79.                           led_state = 0;
80.                 }
81.
```

- Ordine inversă: LED-ul are pe rând culorile: galben, turcoaz, verde, alb

Se aplică același principiu ca la ordinea default:

```
1. if(change_sequence == '1')
2.   {
```

```
 3.                 if(led_state == 0)
 4.                 {
 5.                           /* YELLOW */
 6.                           GPIOB_PSOR |= (1<<RED_LED_PIN);
 7.                           GPIOD_PSOR |= (1<<BLUE_LED_PIN);
 8.                           GPIOB_PSOR |= (1<<GREEN_LED_PIN);
 9.
10.                           GPIOB_PCOR |= (1<<GREEN_LED_PIN);
11.                           GPIOB_PCOR |= (1<<RED_LED_PIN);
12.
13.                           led_state = 1;
14.                 }
15.                 else if (led_state == 1)
16.                 {
17.                           /* TURQUOISE */
18.                           GPIOB_PSOR |= (1<<GREEN_LED_PIN);
19.                           GPIOB_PSOR |= (1<<RED_LED_PIN);
20.
21.                           GPIOB_PCOR |= (1<<GREEN_LED_PIN);
22.                           GPIOD_PCOR |= (1<<BLUE_LED_PIN);
23.                           led_state = 2;
24.                 }
25.                 else if (led_state == 2)
26.                 {
27.                           /* GREEN */
28.                           GPIOB_PSOR |= (1<<GREEN_LED_PIN);
29.                           GPIOD_PSOR |= (1<<BLUE_LED_PIN);
30.
31.                           GPIOB_PCOR |= (1<<GREEN_LED_PIN);
32.                           led_state = 3;
33.                 }
34.                 else if (led_state == 3)
35.                 {
36.                           /* WHITE */
37.                           GPIOB_PSOR |= (1<<GREEN_LED_PIN);
38.
39.                           GPIOB_PCOR |= (1<<RED_LED_PIN);
40.                           GPIOD_PCOR |= (1<<BLUE_LED_PIN);
41.                           GPIOB_PCOR |= (1<<GREEN_LED_PIN);
42.                           led_state = 0;
43.
44.                 }
45.       }
46.
```

- Afișare individuală a unei singure culori

Pentru a realiza această funcționalitate, am gestionat valorile flag-ului individual_color:

```
 1. /* show individual colors */
 2.
 3.       if(individual_color == 'W')
 4.       {
 5.                 GPIOB_PCOR |= (1<<RED_LED_PIN);
 6.                 GPIOD_PCOR |= (1<<BLUE_LED_PIN);
 7.                 GPIOB_PCOR |= (1<<GREEN_LED_PIN);
 8.       }
 9.
10.       if(individual_color == 'G')
11.       {
12.                 GPIOB_PSOR |= (1<<GREEN_LED_PIN);
13.                 GPIOB_PSOR |= (1<<RED_LED_PIN);
14.                 GPIOD_PSOR |= (1<<BLUE_LED_PIN);
15.
16.                 GPIOB_PCOR |= (1<<GREEN_LED_PIN);
17.       }
18.
```

```
19.       if(individual_color == 'T')
20.       {
21.                 GPIOB_PSOR |= (1<<GREEN_LED_PIN);
22.                 GPIOB_PSOR |= (1<<RED_LED_PIN);
23.                 GPIOD_PSOR |= (1<<BLUE_LED_PIN);
24.
25.                 GPIOB_PCOR |= (1<<GREEN_LED_PIN);
26.                 GPIOD_PCOR |= (1<<BLUE_LED_PIN);
27.       }
28.
29.       if(individual_color == 'Y')
30.       {
31.                 GPIOB_PSOR |= (1<<GREEN_LED_PIN);
32.                 GPIOB_PSOR |= (1<<RED_LED_PIN);
33.                 GPIOD_PSOR |= (1<<BLUE_LED_PIN);
34.
35.                 GPIOB_PCOR |= (1<<GREEN_LED_PIN);
36.                 GPIOB_PCOR |= (1<<RED_LED_PIN);
37.       }
38.
```

- Eliminarea unei culori din secvență

Acest feature este integrat în tratarea cazului de ordine default a secvenței de culori, prin flag-ul *stop_led* (codul de la bullet-ul *Ordine default: LED-ul are pe rând culorile: alb, verde, turcoaz, galben*).

- **ADC**

Modulul ADC servește la citirea valorilor analogice primite de la senzorul de rotație (potențiometru) și transformarea acestora în valori digitale, așa fel încât să poată fi folosite în mod util în cod: schimbarea stării LED-urilor integrate pe breadboard în funcție de tensiunea indicată de cursorul potențiometrului.

Inspectând Reference Manual, capitolul 28 – Analog to Digital Convertor (ADC), subcapitolul 28.3.4 – ADC Data Result Register, se observă faptul că registrul ADC0→R[0] menține valoarea analogică primită de la senzor.

28.3.4 ADC Data Result Register (ADCx_Rn)

The data result registers (Rn) contain the result of an ADC conversion of the channel selected by the corresponding status and channel control register (SC1A:SC1n). For every status and channel control register, there is a corresponding data result register.

Astfel, se populează variabila *analog_input* cu valoarea registrului ADC0->R[0], efectuându-se ulterior o conversie a acesteia într-o valoare digitală, în variabila *measured_voltage*.

```
1. void ADC0_IRQHandler(void) {
2.   uint16_t analog_input = (uint16_t) ADC0->R[0];
3.       measured_voltage = (analog_input * 3.3f) / 65535;
4.
```

Logica acestei funcționalități constă în împărțirea intervalului de valori pe care le poate lua senzorul analogic, [0 – 3.3 V], în 3 subintervale egale și acționarea LED-urilor, astfel:

- 0 – 1.1 V → se aprinde LED-ul verde;
- 1.1 – 2.2 V → LED-ul verde stă aprins și se aprinde și LED-ul galben;
- 2.2 – 3.3 V → LED-urile verde și galben stau aprinse și se aprinde LED-ul roșu.

```
 1. if(measured_voltage >= 0 && (double)measured_voltage < 1.1)
 2.       {
 3.                GPIOA->PSOR |= (1<<GREEN_LED_PIN); /* pornesc ledul verde */
 4.                GPIOA->PCOR |= (1<<YELLOW_LED_PIN); /* sting ledul galben */
 5.                GPIOA->PCOR |= (1<<RED_LED_PIN); /* sting ledul rosu */
 6.       }
 7.
 8.       else if((double)measured_voltage >= 1.1 && (double)measured_voltage < 2.2)
 9.       {
10.                GPIOA->PCOR |= (1<<RED_LED_PIN); /* sting ledul rosu */
11.                GPIOA->PSOR |= (1<<YELLOW_LED_PIN); /* pornesc ledul galben */
12.                GPIOA->PSOR |= (1<<GREEN_LED_PIN); /* pornesc ledul verde */
13.       }
14.
15.       else if((double)measured_voltage >= 2.2 && (double)measured_voltage <= 3.3)
16.       {
17.                GPIOA->PSOR |= (1<<GREEN_LED_PIN); /* pornesc ledul verde */
18.                GPIOA->PSOR |= (1<<RED_LED_PIN); /* pornesc ledul rosu */
19.                GPIOA->PSOR |= (1<<YELLOW_LED_PIN); /* pornesc ledul galben */
20.       }
21.       else
22.       {
23.                GPIOA->PCOR |= (1<<GREEN_LED_PIN); /* pornesc ledul verde */
24.                GPIOA->PCOR |= (1<<RED_LED_PIN); /* pornesc ledul rosu */
25.                GPIOA->PCOR |= (1<<YELLOW_LED_PIN); /* pornesc ledul galben */
26.       }
27.
28.
```

- **GPIO**

Am definit macro-uri pentru fiecare port la care sunt legate becurile LED, atât de pe breadboard, cât și cele integrate în platforma de dezvoltare:

```
1. #define RED_LED_PIN (18) /* PORT B */
2. #define GREEN_LED_PIN (19) /* PORT B */
3. #define BLUE_LED_PIN (1) /* PORT D */
4.
5. #define LED_PIN12 (12) /* PORT A12*/
6. #define LED_PIN4 (4) /* PORT A4 */
7. #define LED_PIN5 (5) /* PORT A5 */
8.
```

Pentru utilizarea becurilor LED, este necesară activarea semnalului lor de ceas. Astfel, am setat pe 1 falg-urile aferente semnalului de ceas pentru pinii LED-urilor plăcii:

```
1.        SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTD_MASK;
2.
```

SIM_SCGC5 field descriptions

Field	Description
31–20 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
19 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
18–14 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
13 PORTE	Port E Clock Gate Control This bit controls the clock gate to the Port E module. 0 Clock disabled 1 Clock enabled
12 PORTD	Port D Clock Gate Control This bit controls the clock gate to the Port D module. 0 Clock disabled 1 Clock enabled
11 PORTC	Port C Clock Gate Control This bit controls the clock gate to the Port C 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.

Am procedat la fel și pentru becurile LED de pe breadboard, toți conectați la PTA. Consultând Reference Manual, observăm că putem activa alternativa de GPIO pe un pin prin registrul PCR:

PORTx_PCR*n* field descriptions (continued)

Field	Description
23–20 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.

...

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).

Pentru a activa pinul 12, aplicăm operația AND cu masca de multiplexare definită în biblioteca MKL25Z4.h:

```
#define PORT_PCR_MUX_MASK                        0x700u
```

În binar, această valoare se reprezintă ca 0111 0000 000. Negată, această valoare devine 1000 1111 1111. Având în vedere că registrul PCR arată în felul următor:

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											0					
Reset	0	0	0	0	0	x*	x*	x*	0	x*	0	x*	0	x*	x*	x*

* Notes:
- x = Undefined at reset.

, operația de AND cu inversul măștii va determina resetarea stării celor 3 biți asignați MUX-ului la starea lor inițială. Apoi, pentru activarea alternativei GPIO, se setează biții MUX-ului în 001, prin operația de OR cu macro-ul PORT_PCR_MUX cu indexul 1.

```
#define PORT_PCR_MUX(x)                          (((uint32_t)(((uint32_t)(x))<<PORT_PCR_MUX_SHIFT))&PORT_PCR_MUX_MASK)
```

Apoi, pentru configurarea pinilor așa fel încât aceștia să redea un output (aprinderea LED-urilor), se modifică regiștrii PDDR și PCOR, PDDR cu valoarea 1 prin operația OR, pentru configurarea acestuia ca fiind de ieșire.

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.

41.3.3 Port Clear Output Register (FGPIOx_PCOR)

This register configures whether to clear the fields of PDOR.

Address: Base address + 8h 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	0																															
W	PTCO																															
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

FGPIOx_PCOR field descriptions

Field	Description
31–0 PTCO	Port Clear Output Writing to this register will update the contents of the corresponding bit in the Port Data Output Register (PDOR) as follows: 0 Corresponding bit in PDORn does not change. 1 Corresponding bit in PDORn is cleared to logic 0.

```
1.      PORTA->PCR[LED_PIN12] &= ~PORT_PCR_MUX_MASK;
2.      PORTA->PCR[LED_PIN12] |= PORT_PCR_MUX(1);
3.
4.      GPIOA->PDDR |= (1<<LED_PIN12);
5.      GPIOA->PCOR |= (1<<LED_PIN12);
6.
7.      /*----*/
8.
9.      PORTA->PCR[LED_PIN4] &= ~PORT_PCR_MUX_MASK;
10.     PORTA->PCR[LED_PIN4] |= PORT_PCR_MUX(1);
11.
12.     GPIOA->PDDR |= (1<<LED_PIN4);
13.     GPIOA->PCOR |= (1<<LED_PIN4);
14.
15.     /*----*/
16.
17.     PORTA->PCR[LED_PIN5] &= ~PORT_PCR_MUX_MASK;
18.     PORTA->PCR[LED_PIN5] |= PORT_PCR_MUX(1);
19.
20.     GPIOA->PDDR |= (1<<LED_PIN5);
21.     GPIOA->PCOR |= (1<<LED_PIN5);
22.
23.     /* --- RED LED --- */
24.
25.     /* Utilizare GPIO ca varianta de multiplexare */
26.     PORTB->PCR[RED_LED_PIN] &= ~PORT_PCR_MUX_MASK;
27.     PORTB->PCR[RED_LED_PIN] |= PORT_PCR_MUX(1);
28.
29.     /* Configurare pin pe post de output */
30.     GPIOB_PDDR |= (1<<RED_LED_PIN);
31.
32.     /* Stingerea LED-ului (punerea pe 0 logic) */
33.     GPIOB_PSOR |= (1<<RED_LED_PIN);
34.
35.     /* --- GREEN LED --- */
36.
37.     /* Utilizare GPIO ca varianta de multiplexare */
38.     PORTB->PCR[GREEN_LED_PIN] &= ~PORT_PCR_MUX_MASK;
39.     PORTB->PCR[GREEN_LED_PIN] |= PORT_PCR_MUX(1);
40.
41.     /* Configurare pin pe post de output */
42.     GPIOB_PDDR |= (1<<GREEN_LED_PIN);
43.
44.     /* Stingerea LED-ului (punerea pe 0 logic) */
45.     GPIOB_PSOR |= (1<<GREEN_LED_PIN);
46.
```

```
47.       /* --- BLUE LED --- */
48.
49.       /* Utilizare GPIO ca varianta de multiplexare */
50.       PORTD->PCR[BLUE_LED_PIN] &= ~PORT_PCR_MUX_MASK;
51.       PORTD->PCR[BLUE_LED_PIN] |= PORT_PCR_MUX(1);
52.
53.       /* Configurare pin pe post de output */
54.       GPIOD_PDDR |= (1<<BLUE_LED_PIN);
55.
56.       /* Stingerea LED-ului (punerea pe 0 logic) */
57.       GPIOD_PSOR |= (1<<BLUE_LED_PIN);
58.
```

2.3.2. Client – GUI

Pentru realizarea interfeței am urmat pașii din template-ul de pe repository-ul de GIT, personalizându-l cu următoarele funcționalități:

- Implementarea funcționalităților prin transmiterea de semnale cu ajutorul butoanelor de tip push regăsite la nivelul Control Panel:

```
 1.          control_panel_box = QGroupBox("Control Panel")
 2.          control_panel_box.setFont(bold_font)
 3.
 4.          button1 = QPushButton("Reverse led sequence", self)
 5.          button1.clicked.connect(lambda: self.send_signal(b'M'))
 6.          button2 = QPushButton("Normal led sequence", self)
 7.          button2.clicked.connect(lambda: self.send_signal(b'N'))
 8.
 9.          button3 = QPushButton("Stop on white led ")
10.          button3.clicked.connect(lambda: self.send_signal(b'W'))
11.
12.          button4 = QPushButton("Stop on green led ")
13.          button4.clicked.connect(lambda: self.send_signal(b'G'))
14.
15.          button5 = QPushButton("Stop on turquoise led ")
16.          button5.clicked.connect(lambda: self.send_signal(b'T'))
17.
18.          button6 = QPushButton("Stop on yellow led ")
19.          button6.clicked.connect(lambda: self.send_signal(b'Y'))
20.
21.
22.          button7 = QPushButton("Stop white led")
23.          button7.clicked.connect(lambda: self.send_signal(b'w'))
24.
25.          button8 = QPushButton("Stop green led")
26.          button8.clicked.connect(lambda: self.send_signal(b'g'))
27.
28.          button9 = QPushButton("Stop turquoise led")
29.          button9.clicked.connect(lambda: self.send_signal(b't'))
30.
31.          button10 = QPushButton("Stop yellow led")
32.          button10.clicked.connect(lambda: self.send_signal(b'y'))
33.
34.          button11 = QPushButton("LED Sequence Duration - 10ms")
35.          button11.clicked.connect(lambda: self.send_signal(b'1'))
36.
37.          button12 = QPushButton("LED Sequence Duration - 1s")
38.          button12.clicked.connect(lambda: self.send_signal(b'2'))
39.
40.          button13 = QPushButton("LED Sequence Duration - 3s")
41.          button13.clicked.connect(lambda: self.send_signal(b'3'))
42.
```

```
        button14 = QPushButton("Reset Duration to 461ms")
        button14.clicked.connect(lambda: self.send_signal(b'4'))

        control_panel_box_layout = QVBoxLayout()

        control_panel_box_layout.setSpacing(5)
        control_panel_box_layout.addWidget(button1, 1)
        control_panel_box_layout.addWidget(button2, 1)
        control_panel_box_layout.addWidget(button3, 1)
        control_panel_box_layout.addWidget(button4, 1)
        control_panel_box_layout.addWidget(button5, 1)
        control_panel_box_layout.addWidget(button6, 1)
        control_panel_box_layout.addWidget(button7, 1)
        control_panel_box_layout.addWidget(button8, 1)
        control_panel_box_layout.addWidget(button9, 1)
        control_panel_box_layout.addWidget(button10, 1)
        control_panel_box_layout.addWidget(button11, 1)
        control_panel_box_layout.addWidget(button12, 1)
        control_panel_box_layout.addWidget(button13, 1)
        control_panel_box_layout.addWidget(button14, 1)


        control_panel_box.setLayout(control_panel_box_layout)

        tertiary_layout.addWidget(team_box, 1)
        tertiary_layout.addWidget(control_panel_box, 5)

```

- Reprezentarea intervalelor de tensiune pe culori:

```
plot_widget = pg.PlotWidget()
         self.plot_curve = plot_widget.plot(pen={'color': 'k', 'width': 2})  # 'k' for black color, 'width': 2 for a thicker line

        zone1_start, zone1_stop, zone1_color = 0, 1, (0, 204, 102)
        zone2_start, zone2_stop, zone2_color = 1, 1.8, (255, 255, 102)
        zone3_start, zone3_stop, zone3_color = 1.8, 3.3, (255, 0, 0)

                           plot_widget.addItem(pg.LinearRegionItem([zone1_start,   zone1_stop], orientation='horizontal', brush=pg.mkBrush(zone1_color), movable=False))
                           plot_widget.addItem(pg.LinearRegionItem([zone2_start,   zone2_stop], orientation='horizontal', brush=pg.mkBrush(zone2_color), movable=False))
                           plot_widget.addItem(pg.LinearRegionItem([zone3_start,   zone3_stop], orientation='horizontal', brush=pg.mkBrush(zone3_color), movable=False))

        secondary_layout.addWidget(plot_widget, 3)
        secondary_layout.addLayout(tertiary_layout, 1)

        primary_layout.addLayout(secondary_layout, 4)
        self.text_edit = QTextEdit()
        self.text_edit.setReadOnly(True)

```

- Realizarea comunicației seriale cu platforma de dezvoltare:

```
  def serial_communication(self):
         self.serial_port = serial.Serial(SERIAL_COM, BAUD_RATE, timeout=1)  # Replace "COMx" with your actual COM port

        self.sensor_timer = QTimer(self)
        self.sensor_timer.timeout.connect(self.update_plot)
        self.sensor_timer.start(0.10)
        self.start_time = datetime.now()
        self.previous_duration = 0
        self.previous_sensor_value = 0
```

- Transmiterea semnalelor spre platforma de dezvoltare:

```
def send_signal(self, character):
        self.serial_port.write(character)
```

- Citirea și afișarea tensiunii:

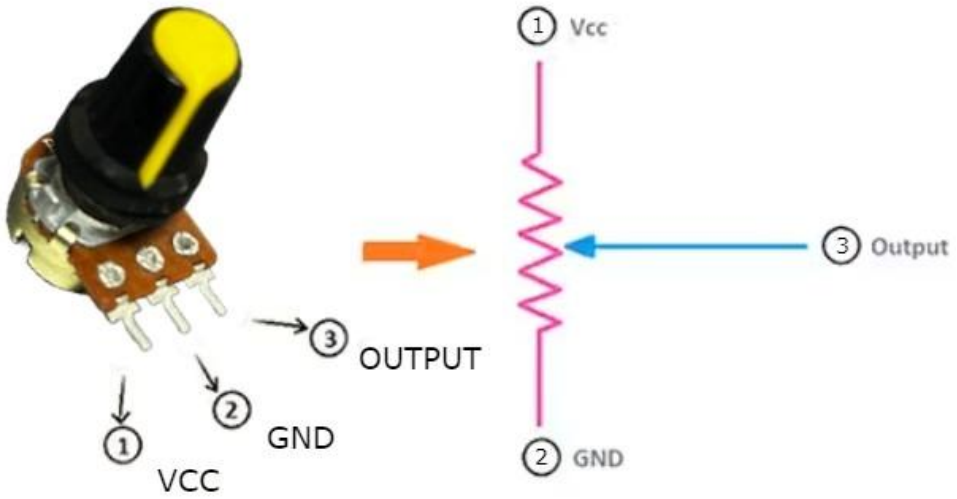
```
def update_plot(self):
        try:
            data = self.serial_port.readline().decode('ascii', errors='ignore').strip()

            print("Raw data:", data)

            match = re.search(r'[-+]?\d*\.\d+|\d+', data)
            if match:
                sensor_value = float(match.group())
                duration = (datetime.now() - self.start_time).total_seconds()
                    self.text_edit.insertPlainText(f"Sensor Value: {sensor_value}, Duration:
{duration} seconds\n")
                sensor_value = max(0, min(sensor_value, 3.3))
                x_values = np.linspace(0, duration, 1000)
                y_values = np.full_like(x_values, sensor_value)
                self.plot_curve.setData(x_values, y_values)
                self.plot_curve.setZValue(10)

            else:
                print("Invalid data format: ", data)

        except Exception as e:
            print(f"Error: {e}")
```

3. Setup

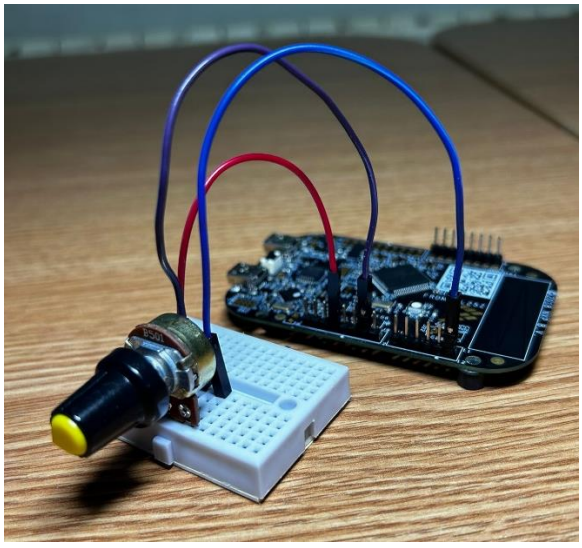
3.1. Configurare hardware

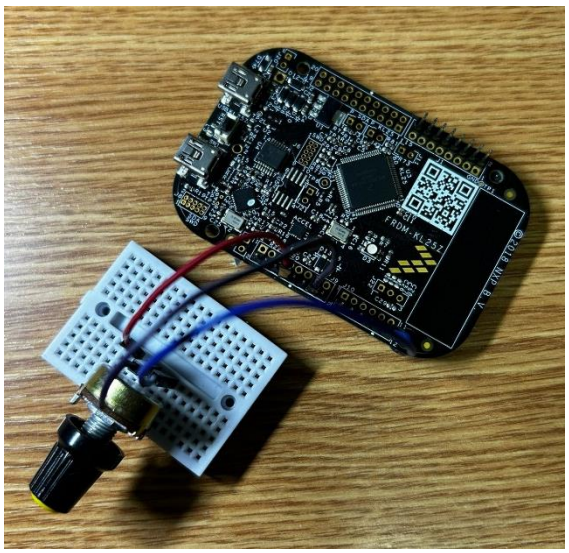
3.1.1. Configurare senzor

1. Se fixează potențiometrul pe mini breadboard



2. Se conectează firele la potențiometru în următorul fel:
   - Firul roșu se conectează pe pinul 1;
   - Firul mov închis pe pinul 2;
   - Firul albastru pe pinul 3.
3. Se conecteaza aceleași fire pe microcontroller astfel:
   - Firul roșu pe pinul 3.3V de pe J9;
   - Firul mov pe unul din pinii GND de pe J9;
   - Firul albastru pe pinul C1 din J10.

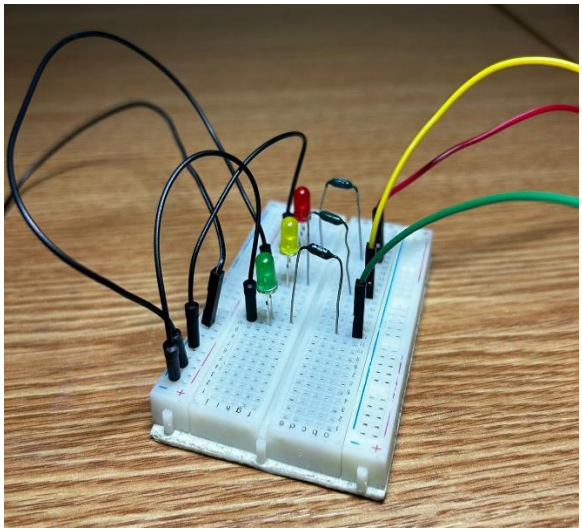


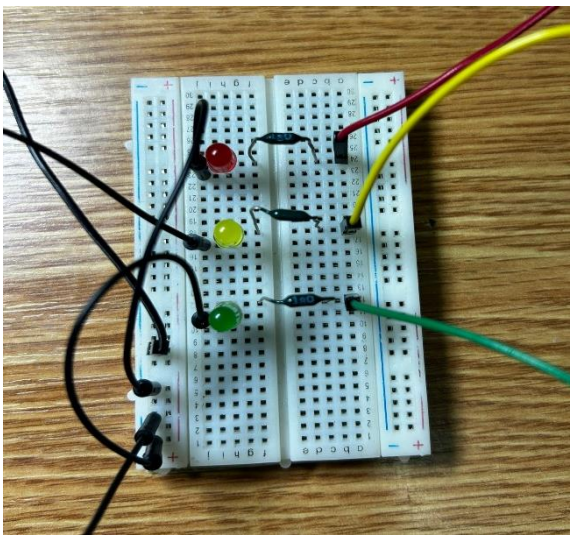


3.1.2. Configurare LED-urilor pe breadboard

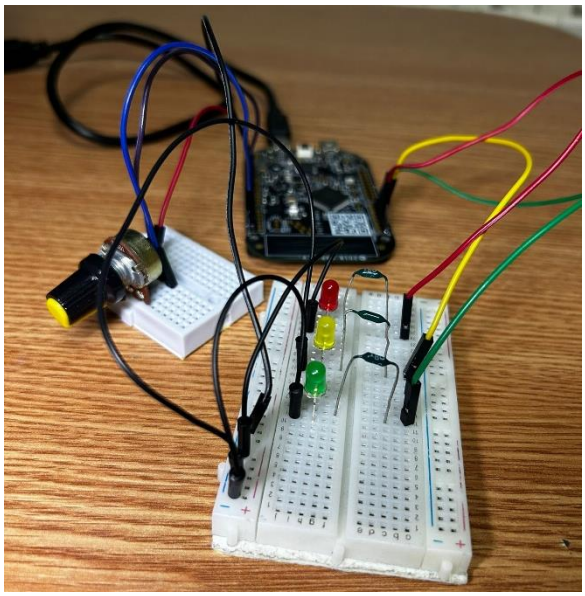
Fiecare LED se configurează în următorul mod:

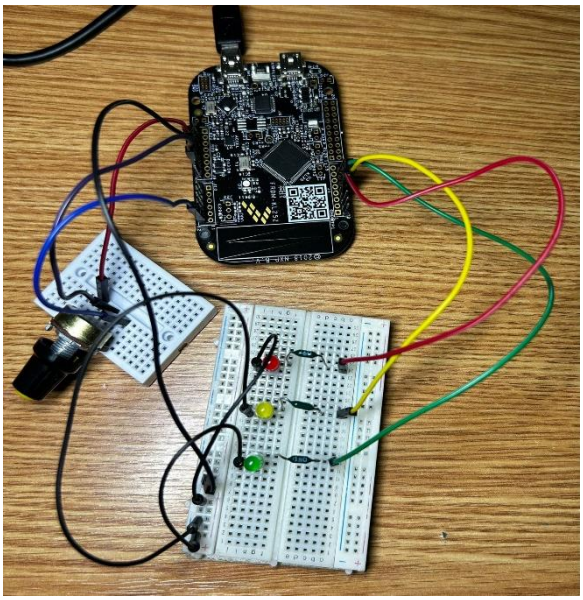
- Se fixează pe breadboard;
- Se adaugă o rezistență în serie cu anodul ledului;
- Se fixează catodul la ground(-) printr-un fir;
- În serie cu rezistențele se conectează câte un fir conectat la pinii de pe placă, astfel:
    - Firul roșu pe pinul A12 de pe J1;
    - Firul galben pe pinul A4 de pe J1;
    - Firul verde pe pinul A5 de pe J1.



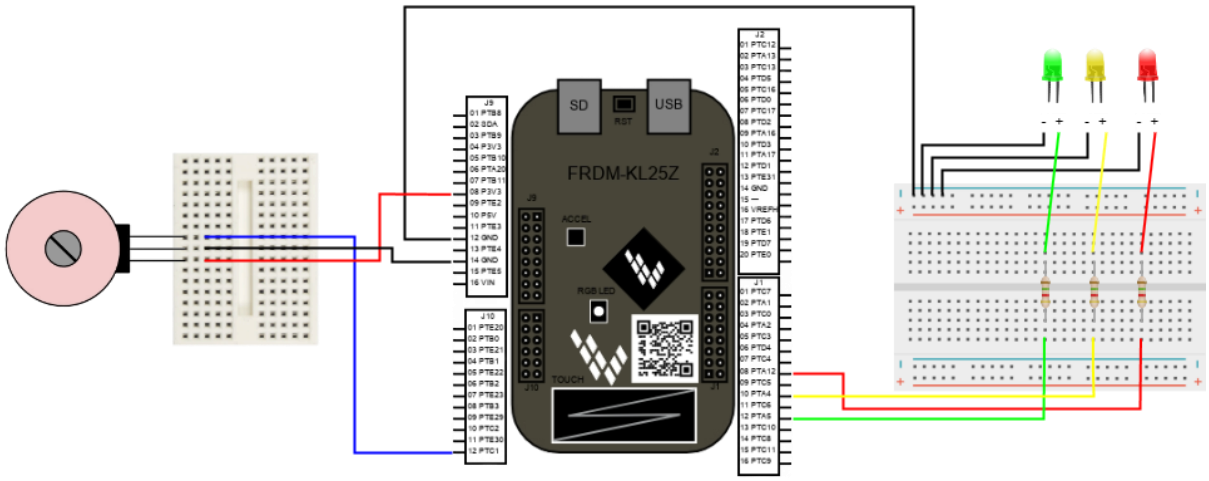


În final, configurația va arăta în următorul fel:





Schema circuitului electronic:



3.2. Configurare software

1. Se instalează Keil uVision și o versiune de python3;

2. Se accesează repository-ul de git pentru a descărca proiectul: https://github.com/MerryWex/ProiectMicroprocesoare.git

3. Pentru configurarea environment-ului necesar interfeței grafice se realizează următorii pași în folder-ul care conține interfața:

   3.1. Se creează fișierul requirements.txt și se adaugă următoarele linii:
   PySide6==6.6.0
   pyserial==3.5
   pyqtgraph==0.13.3

   3.2. Se rulează următoarele comenzi:
   rm -rf venv-up-all
   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

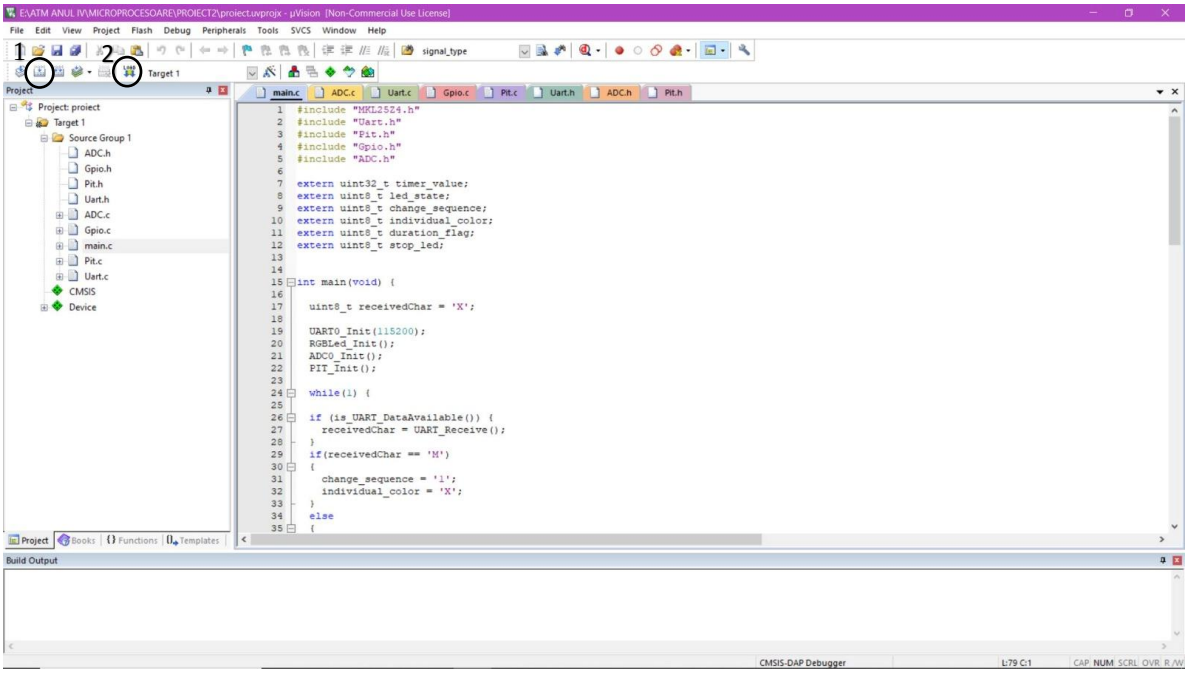
   3.3. Se instalează următoarele biblioteci, în cazul în care acestea nu există:
   pip install numpy
   pip install pyserial

4. Se deschide partea de backend folosind Keil uVision se compilează proiectul și se încarcă pe platforma de dezvoltare;

5. Se deschide partea de GUI folosind Visual Studio Code și se rulează programul din terminal folosind comanda py main.py.

3.3. Testarea

În urma parcurgerii acestor pași, obținem rezultatele:



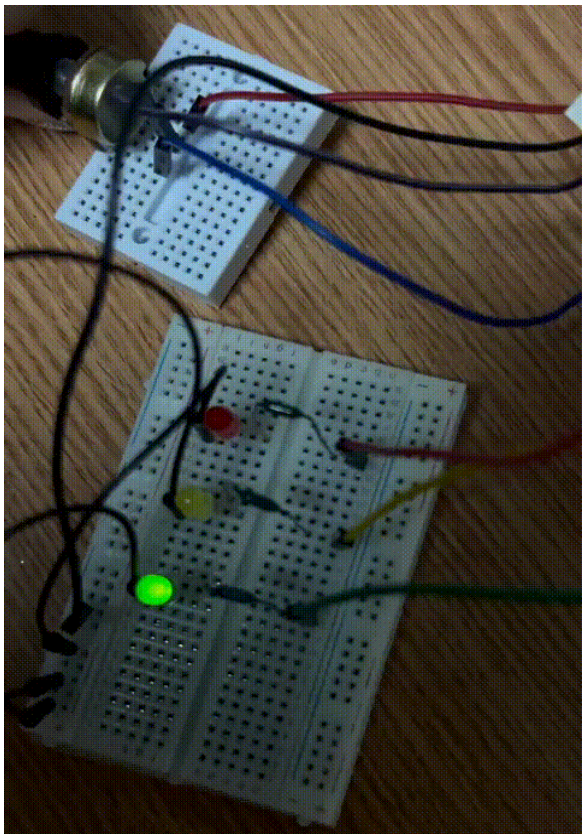
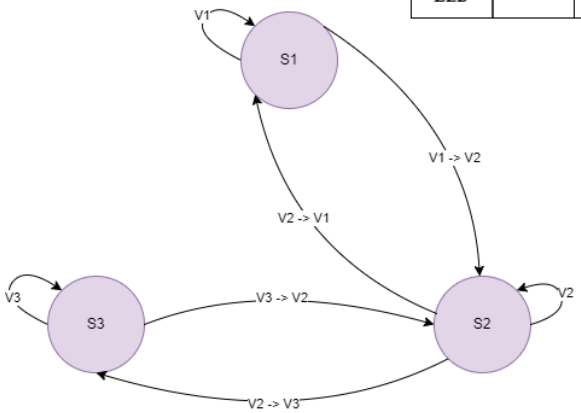


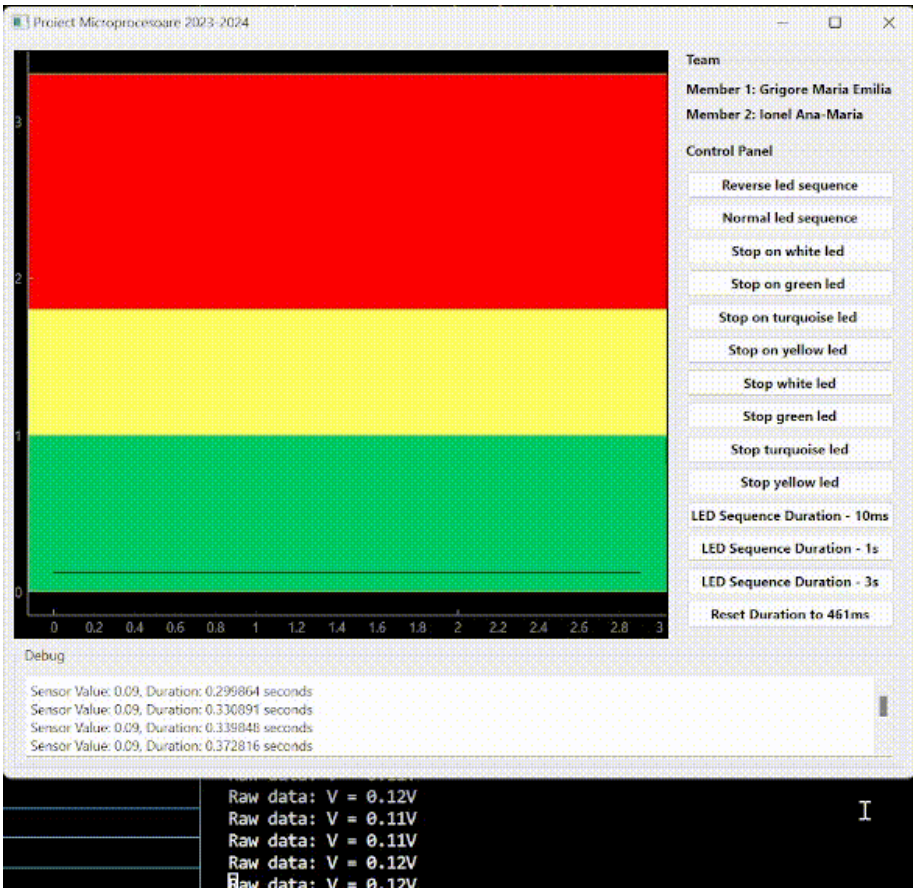
Diagrama de stări a becurilor LED de pe breadboard:

S1	S2	S3
0 - 1.1 V	1.1 - 2.2 V	2.2 - 3.3 V
V1	V2	V3

	S1	S2	S3
GREEN LED	ON	ON	ON
YELLOW LED	OFF	ON	ON
RED LED	OFF	OFF	ON



Interfața grafică:



4. Probleme întâmpinate

Pe parcursul implementării proiectului am întâmpinat o serie de probleme, precum:

- Environment-ul nu e programmer friendly: lipsa identării reprezintă un dezavantaj, deoarece devine complicată respectarea standardelor de scriere a codului. De asemenea, apar warning-uri multiple care, deși nu afectează rularea programului, provoacă impresia unui cod neglijent.
- În Tinkercad nu există platforma de dezvoltare FRDM-KL25Z, ducând astfel la o proiectare digitală mai complicată a schemei, utilizând o altă aplicație (draw.io).
- Imposibilitatea testării interfeței grafice neavând placa la dispoziție, ceea ce a condus la o serie de conflicte în ceea ce privește transmiterea semnalului spre server și interpretarea acestuia. Rezolvarea a fost realizată prin debugging, observând formatul datelor transmise vs formatul pe care îl poate interpreta modulul UART (b'X' vs uint8_t).
- Imposibilitatea reprezentării grafice sub formă continuă în timp, înlocuita prin reprezentarea unei linii de lungime constantă care își modifică poziția în funcție de tensiunea indicată de cursorul potențiometrului. Poziția graficului coincide cu porțiunea de culoare aferentă culorii LED-ului care se activează în intervalul indicat de senzor.
- Setback-ul pe care l-a produs senzorul de rotație defect, acesta reușind să poziționeze cursorul doar într-un interval de 3.23 – 3.3 V, ducând astfel la timp mort în încercarea rezolvării problemei presupuse a fi din cod. Problema s-a rezolvat înlocuind senzorul cu un potențiometru comercial.

5. Bibliografie

- https://www.digikey.ro/ro/articles/the-fundamentals-of-digital-potentiometers
- https://www.nxp.com/design/design-center/development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-kl14-kl15-kl24-kl25-mcus:FRDM-KL25Z
- https://app.diagrams.net/
- https://www.riverbankcomputing.com/static/Docs/PyQt6/
- https://doc.qt.io/qtforpython-6/quickstart.html#quick-start
- FRDM-KL25Z_ReferenceManual
- FRDM-KL25Z_Schematics
- FRDM-KL25Z_Datasheet