

**ROMÂNIA**  
**MINISTERUL APĂRĂRII NAȚIONALE**  
**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**  
**FACULTATEA DE SISTEME INFORMATICE ȘI**  
**SECURITATE CIBERNETICĂ**

**Specializarea: Calculatoare și sisteme informatice pentru apărare și  
securitate națională**



**Proiect Sisteme Specializate cu Microprocesoare**

Sd. Sg. Maj. DUMITRU Andreea-Ioana

Sd. Sg. Maj. MOCANU Răzvan

**București**

**2023**

# CUPRINS

<b>1. Scopul proiectului.....</b>	<b>3</b>
<b>2. Configurare senzori.....</b>	<b>3</b>
2.1. <i>Senzorul analog de sunet DFR0034.....</i>	<i>3</i>
2.2. <i>Servomotorul SG90 .....</i>	<i>4</i>
<b>3. Reprezentarea grafică a modului de conectare a senzorilor și a componentelor electronice .....</b>	<b>5</b>
<b>4. Descrierea programului.....</b>	<b>6</b>
4.1. <i>Inițializarea modulelor.....</i>	<i>6</i>
4.1.1. Modulul UART.....	6
4.1.2. Inițializarea modului GPIO .....	10
4.1.3. Inițializarea modului PIT .....	14
4.1.4. Inițializarea modului ADC și prelucrarea datelor înregistrate de către senzor .....	17
4.1.5. Inițializarea modului TPM și generare PWM .....	23
4.2. <i>Configurarea ceasului.....</i>	<i>26</i>
4.3. <i>Funcția main.....</i>	<i>29</i>
<b>5. Set-up .....</b>	<b>32</b>
<b>6. Schemă bloc.....</b>	<b>35</b>
<b>7. Diagrame de stări .....</b>	<b>35</b>
<b>8. Rezultate aplicație Python .....</b>	<b>36</b>
<b>9. Probleme întâmpinate.....</b>	<b>37</b>
<b>10. Referințe .....</b>	<b>37</b>
<b><u><a href="https://github.com/andreeaa23/proiect_micro_FRDM_KL25Z128">https://github.com/andreeaa23/proiect_micro_FRDM_KL25Z128</a></u></b>	

# 1. Scopul proiectului

Scopul proiectului este acela de a modifica unghiul de rotație al servomotorului utilizând un senzor de sunet printr-un semnal PWN generat și manipulat în funcție de plaja de valori a senzorului de sunet.

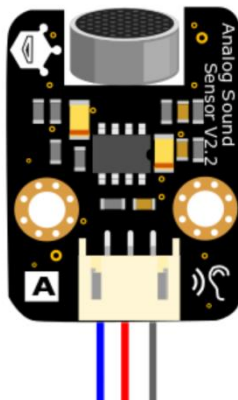
Valoarea digitală care va rezulta în urma conversiei din tensiune a datelor achiziționate de la senzorul de sunet se vor regăsi într-un interval care va fi apoi împărțit în trei subintervale care vor corespunde cu valori mici, medii și mari ale datelor achiziționate.

În funcție de subintervalul în care se regăsesc datele convertite din analogic în digital, servomotorul își va schimba poziția elicei ( $0^\circ$ ,  $90^\circ$  și  $180^\circ$ ).

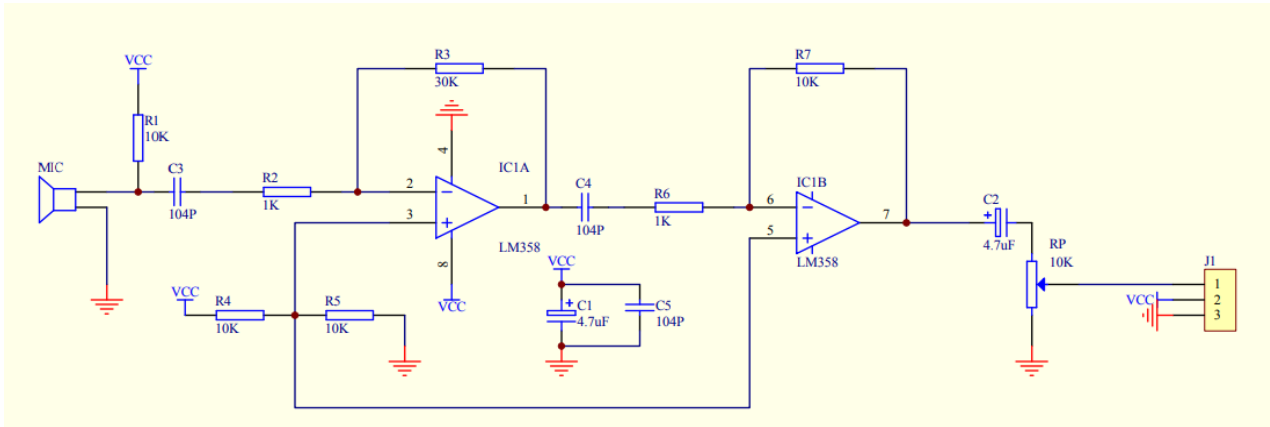
## 2. Configurare senzori

### 2.1. *Senzorul analog de sunet DFR0034*

Senzorul DFR0034 este un senzor analogic ce detectează sunetele din mediul în care este pus.



- Blue – Analog signal output
- Red - VCC
- Black – GND



Specificații:

- Tensiune de alimentare: 3.3V până la 5V;
- Detectează intensitatea sunetului;
- Mărime: 22x30 mm.

Senzorul de sunet a fost conectat pe plăcuță astfel:

- ✚ **Firul roșu (VCC)** se conectează la pinul de 3V (P3V3);
- ✚ **Firul negru (GND)** se conectează la pinul GND;
- ✚ **Firul albastru (output sensor value)** se conectează la pinul PTC2.

## 2.2. Servomotorul SG90

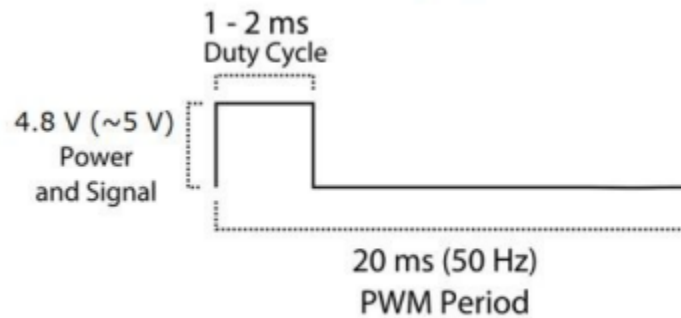
Servomotorul SG90 se poate roti până la aproximativ 180 grade(90 în fiecare direcție).



- Orange – PWN
- Red – VCC
- Brown – GND

Specificații:

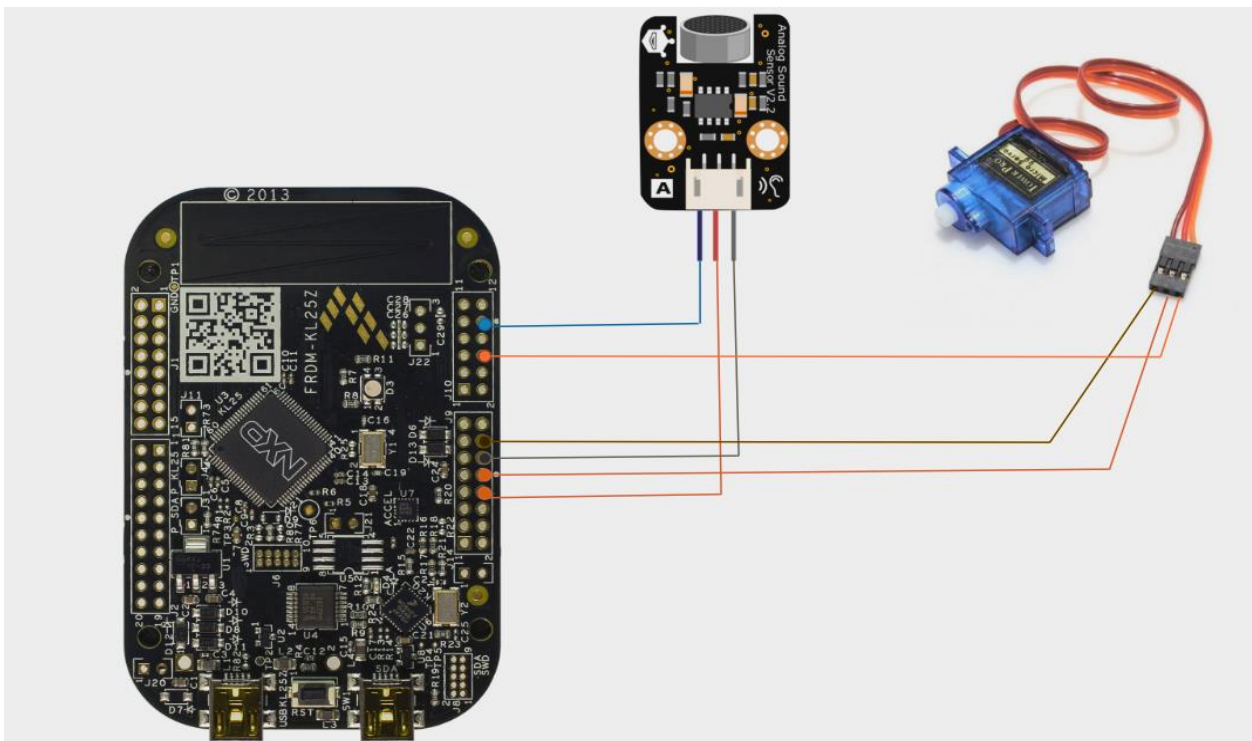
- Tensiune de alimentare: 4.8V până la 5V;
- Rotire elice;
- Mărime: 23x32 mm.



Servomotorul a fost conectat pe plăcuță astfel:

- ✚ **Firul maroniu (GND)** se conectează la pinul GND;
- ✚ **Firul roșu (VCC)** se conectează la pinul de 5V (P5V\_USB) ;
- ✚ **Firul portocaliu (PWN)** se conectează la PTB2.

### 3. Reprezentarea grafică a modului de conectare a senzorilor și a componentelor electronice



## 4. Descrierea programului

### 4.1. Inițializarea modulelor

#### 4.1.1. Modulul UART

Vom folosi modulul UART0 pentru comunicația serială cu PC prin cablul USB.

Prin funcția *UART0\_init* configurăm următoarele:

```
1. void UART0_Init(uint32_t baud_rate)
2. {
3.
4.     //Setarea sursei de ceas pentru modulul UART
5.     SIM->SOPT2 |= SIM_SOPT2_UART0SRC(01);
6.
7.     //Activarea semnalului de ceas pentru modulul UART
8.     SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;
9.
10.    //Activarea semnalului de ceas pentru portul A
11.    //intrucat dorim sa folosim pinii PTA1, respectiv PTA2 pentru comunicarea UART
12.    SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;
13.
14.    //Fiecare pin pune la dispozitie mai multe functionalitati
15.    //la care avem acces prin intermediul multiplexarii
16.    PORTA->PCR[1] = ~PORT_PCR_MUX_MASK;
17.    PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Configurare RX pentru UART0
18.    PORTA->PCR[2] = ~PORT_PCR_MUX_MASK;
19.    PORTA->PCR[2] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Configurare TX pentru UART0
20.
21.
22.    UART0->C2 &= ~(UART0_C2_RE_MASK | (UART0_C2_TE_MASK));
23.
24.    //Configurare Baud Rate
25.    uint32_t osr = 4; // Over-Sampling Rate (numarul de esantioane luate per bit-time)
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 0 0 0
0 0 0 0 0
30.    //          0 0 0 b13 b12 b11 b10 b09 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.
34.    uint32_t sbr = 4800000UL / ((osr*4) * baud_rate);
35.    uint8_t temp = UART0->BDH & ~(UART0_BDH_SBR(0x1F));
36.    UART0->BDH = temp | UART0_BDH_SBR(((sbr & 0x1F00)>> 8));
37.    UART0->BDL = (uint8_t)(sbr & UART0_BDL_SBR_MASK);
38.    UART0->C4 |= UART0_C4_OSR(osr);
39.
40.
41.    //Setare numarul de biti de date la 8 si fara bit de paritate
42.    UART0->C1 = 0;
43.
44.    //Dezactivare intreruperi la transmisie
45.    UART0->C2 |= UART0_C2_TIE(0);
46.    UART0->C2 |= UART0_C2_TCIE(0);
47.
```

```

48.      //Activare intreruperi la receptie
49.      UART0->C2 |= UART0_C2_RIE(1);
50.
51.      UART0->C2 |= ((UART_C2_RE_MASK) | (UART_C2_TE_MASK));
52.
53.      NVIC_EnableIRQ(UART0_IRQn);
54.
55. }
56.

```

În registrul SIM\_SOPT2(System Options Register 2) setăm pe 01 câmpul UART0SRC(biții 27-26) pentru selectarea ca sursă de ceas a modului MCGFLLCLK.

27-26 UART0SRC	<b>UART0 clock source select</b> Selects the clock source for the UART0 transmit and receive clock.
00	Clock disabled
01	MCGFLLCLK clock or MCGPLLCLK/2 clock
10	OSCECLK clock
11	MCGIRCLK clock

În registrul SIM\_SCGC4(System Clock Gating Control Register 4) setăm pe 1 câmpul UART0(bitul 10) pentru activarea ceasului pentru acest modul, folosind masca SIM\_SCGC4\_UART0\_MASK. Masca are valoarea 0x400(1024 în zecimal), adică are setat pe 1 al 10-lea bit(UART0).

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

10 UART0	<b>UART0 Clock Gate Control</b> This bit controls the clock gate to the UART0 module.
0	Clock disabled
1	Clock enabled

În registrul SIM\_SCGC5(System Clock Gating Control Register 5) setăm pe 1 câmpul PORTA(bitul 9) pentru activarea ceasului acestui port, folosind masca SIM\_SCGC5\_PORTA\_MASK. Masca are valoarea 0x200( 512 în zecimal), adică are setat pe 1 al 9-lea bit(PORTA).

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

9 PORTA	<b>Port A Clock Gate Control</b> This bit controls the clock gate to the Port A module.
0	Clock disabled
1	Clock enabled

În regiștrii de control ai pinilor 1 și 2 din portul A (PORTA\_PCR1/PORTA\_PCR2), setăm câmpul MUX(biții 10-8) pe valoarea 2, care înseamnă folosirea acestora în modulul de UART0 (RX/TX).

PTA1	27	TSI0_CH2	PTA1	UART0_RX
PTA2	28	TSI0_CH3	PTA2	UART0_TX

Address: Base address + 0h offset

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

**UARTx\_BDH field descriptions**

Address: Base address + 1h offset

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

**UARTx\_BDL field descriptions**

Având în vedere că baud rate-ul nostru este 115200 și over sampling rate-ul(osr) este 4, formula pentru calcularea câmpului SBR am modificat-o astfel încât să respectăm valorile SBR-ului, aferent baud rate-ului și setărilor Putty:

$$uint32\_t\ sbr = 48000000UL / ((osr * 4) * baud\_rate );$$

4-0 SBR	Baud Rate Modulo Divisor.  The 13 bits in SBR[12:0] are referred to collectively as BR, and 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).
------------	---

Portul serial este setat implicit cu 8 biți date, niciun bit de paritate și un bit de stop.

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.
0 PT	Parity Type  Provided parity is enabled (PE = 1), this bit selects even or odd parity. Odd parity means the total number of 1s in the data character, including the parity bit, is odd. Even parity means the total number of 1s in the data character, including the parity bit, is even.  0 Even parity. 1 Odd parity.
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.

Address: Base address + 2h offset

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



Activăm transmiterea prin UART prin setarea câmpului TE(Transmitter Enable) din UART0\_C2 pe valoarea 1 cu ajutorul măștii UART\_C2\_TE\_MASK și activăm recepția prin UART prin setarea câmpului RE (Receiver Enable) din UART0\_C2 pe valoarea 1 cu ajutorul măștii UART\_C2\_RE\_MASK.

7 TIE	Transmit Interrupt Enable for TDRE 0 Hardware interrupts from TDRE disabled; use polling. 1 Hardware interrupt requested when TDRE flag is 1.
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.

Funcția *UART0\_Trasmit* are rolul de a transmite serial caracterul transmis ca parametru.

```

1. void UART0_Transmit(uint8_t data)
2. {
3.     //Punem in asteptare pana cand registrul de transmisie a datelor nu este gol
4.     while(! (UART0->S1 & UART0_S1_TDRE_MASK));
5.         UART0->D = data;
6.
7. }
8.

```

### 39.2.5 UART Status Register 1 (UARTx\_S1)

Address: Base address + 4h offset

Bit	7	6	5	4	3	2	1	0
Read	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Write				w1c	w1c	w1c	w1c	w1c
Reset	1	1	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
Read	R7T7	R6T6	R5T5	R4T4	R3T3	R2T2	R1T1	R0T0
Write								
Reset	0	0	0	0	0	0	0	0

UARTx\_D field descriptions

Funcția *UART0\_receive* are rolul de a primi serial date.

```
1. uint8_t UART0_receive(void)
2. {
3.     //Punem in asteptare pana cand registrul de receptie nu este plin
4.     while(!(UART0->S1 & UART0_S1_RDRF_MASK));
5.     return UART0->D;
6.
7. }
8.
```

Funcția *UART0\_IRQHandler* are rolul de a transmite datele cu majuscule și de a seta flag-ul *flag\_switch* care schimbă secvența GPIO atunci când se primește orice caracter:

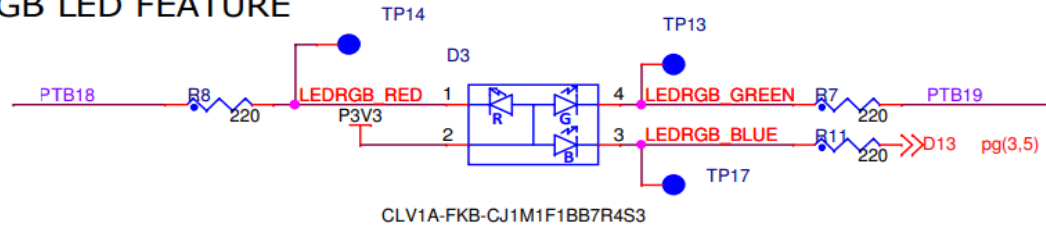
```
1. void UART0_IRQHandler(void) {
2.
3.     if(UART0->S1 & UART0_S1_RDRF_MASK) {
4.         c = UART0->D;
5.         if(flag_swich==0){
6.             flag_swich=1;
7.         }
8.         else{
9.             flag_swich=0;
10.        }
11.    }
12.
13.    if(c >= 'a' && c <= 'z') {
14.        UART0_Transmit(c - 'a' + 'A');
15.    }
16.    else if ( c >= 'A' && c <= 'Z') {
17.        UART0_Transmit(c - 'A' + 'a');
18.    }
19.    else if(c == 0X0D) {
20.        UART0_Transmit(0X0D);
21.        UART0_Transmit(0X0A);
22.    }
23. }
24.
```

#### 4.1.2. Inițializarea modului GPIO

Pentru aplicația noastră avem nevoie de 3 LED-uri: roșu (Red), verde (Green), albastru (Blue) și negru (Black) care semnifică faptul că toate becurile sunt stinse.

```
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.
```

## RGB LED FEATURE



Funcția *OutputPIN\_Init* are rolul de a inițializa modulul GPIO pentru a controla aprinderea/stingerea celor 3 LED-uri.

```

1. void OutputPIN_Init(void){
2.
3.     SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTD_MASK;
4.
5.     // --- RED LED ---
6.
7.     // Utilizare GPIO ca varianta de multiplexare
8.     PORTB->PCR[RED_LED_PIN] &= ~PORT_PCR_MUX_MASK;
9.     PORTB->PCR[RED_LED_PIN] |= PORT_PCR_MUX(1);
10.
11.    // Configurare pin pe post de output
12.    GPIOB_PDDR |= (1<<RED_LED_PIN);
13.
14.    // Stingerea LED-ului (punerea pe 0 logic)
15.    GPIOB_PSOR |= (1<<RED_LED_PIN);
16.
17.    // --- GREEN LED ---
18.
19.    // Utilizare GPIO ca varianta de multiplexare
20.    PORTB->PCR[GREEN_LED_PIN] &= ~PORT_PCR_MUX_MASK;
21.    PORTB->PCR[GREEN_LED_PIN] |= PORT_PCR_MUX(1);
22.
23.    // Configurare pin pe post de output
24.    GPIOB_PDDR |= (1<<GREEN_LED_PIN);
25.
26.    // Stingerea LED-ului (punerea pe 0 logic)
27.    GPIOB_PSOR |= (1<<GREEN_LED_PIN); //scriu 1 stinge ledul
28.
29.    // --- BLUE LED ---
30.
31.    // Utilizare GPIO ca varianta de multiplexare
32.    PORTD->PCR[BLUE_LED_PIN] &= ~PORT_PCR_MUX_MASK;
33.    PORTD->PCR[BLUE_LED_PIN] |= PORT_PCR_MUX(1);
34.
35.    // Configurare pin pe post de output
36.    GPIOD_PDDR |= (1<<BLUE_LED_PIN);
37.
38.    // Stingerea LED-ului (punerea pe 0 logic)
39.    GPIOD_PSOR |= (1<<BLUE_LED_PIN); //set e stins
40.
41. }

```

Se activează semnalul de ceas pentru pinii folosiți în cadrul fiecărui LED:

```
SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTD_MASK;
```

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

Utilizăm GPIO ca variantă de multiplexare pentru pinii corespunzători:

```
19. // Utilizare GPIO ca varianta de multiplexare
20. PORTB->PCR[GREEN_LED_PIN] &= ~PORT_PCR_MUX_MASK;
21. PORTB->PCR[GREEN_LED_PIN] |= PORT_PCR_MUX(1);
```

10–8 MUX	<b>Pin Mux Control</b> 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).
-------------	--

Configurăm pinii pe post de output:

```
35. // Configurare pin pe post de output
36. GPIOD_PDDR |= (1<<BLUE_LED_PIN);
```

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																																
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.

Stingerea ledului constă în punerea pe 0 logic a pinilor:

```
// Stingerea LED-ului (punerea pe 0 logic)
15. GPIOB_PSOR |= (1<<RED_LED_PIN);
```

Funcția *Control\_RGB\_LEDs* are rolul de a aprinde/stinge un LED. LED-urile sunt configurate să se stingă/aprindă după un interval de 1190 ms, perioadă care a fost inițializată folosind canalul 1 din PIT.

```
1. void Control_RGB_LEDs(uint8_t red, uint8_t green, uint8_t blue) {
```

```

2.
3. // Setare sau stinge LED-ul rosu
4. if (red) {
5.     GPIOB_PCOR |= (1 << RED_LED_PIN); // Seteaza pinul
6. }
7.     else {
8.         GPIOB_PSOR |= (1 << RED_LED_PIN); // Stinge pinul
9.     }
10.
11. // Setare sau stinge LED-ul verde
12. if (green) {
13.     GPIOB_PCOR |= (1 << GREEN_LED_PIN); // Seteaza pinul
14. }
15.     else {
16.         GPIOB_PSOR |= (1 << GREEN_LED_PIN); // Stinge pinul
17.     }
18.
19. // Setare sau stinge LED-ul albastru
20. if (blue) {
21.     GPIOD_PCOR |= (1 << BLUE_LED_PIN); // Seteaza pinul
22. }
23.     else {
24.         GPIOD_PSOR |= (1 << BLUE_LED_PIN); // Stinge pinul
25.     }
26.
27. }

```

### Controlul LED-ului **roșu**:

- Dacă valoarea variabilei **red** nu este zero (adevărat), se șterge (se setează la 0) bitul corespunzător în registrul GPIOB\_PCOR. Acest lucru activează LED-ul roșu.
- Dacă valoarea variabilei **red** este zero (fals), se setează (se setează la 1) bitul corespunzător în registrul GPIOB\_PSOR. Acest lucru dezactivează LED-ul roșu.

### Controlul LED-ului **verde**:

- Logica similară cu cea pentru LED-ul roșu, dar aplicată LED-ului verde pe GPIOB.

### Controlul LED-ului **albastru**:

- Dacă valoarea variabilei **blue** nu este zero (adevărat), se șterge (se setează la 0) bitul corespunzător în registrul GPIOD\_PCOR. Acest lucru activează LED-ul albastru.
- Dacă valoarea variabilei **blue** este zero (fals), se setează (se setează la 1) bitul corespunzător în registrul GPIOD\_PSOR. Acest lucru dezactivează LED-ul albastru.

### 4.1.3. Inițializarea modului PIT

Funcția *PIT\_Init* inițializează perifericul PIT, mai exact cele 2 canale ale sale:

```
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 decrementării valorilor numărătoarelor în modul debug
8.     PIT->MCR |= PIT_MCR_FRZ_MASK;
9.     // Setarea valorii numărătoarelor de pe canalul 0 la o perioadă de 1,119 secunde pt led-
uri
10.    PIT->CHANNEL[1].LDVAL = 56879999;
11.    //jumătate de secundă pentru adc
12.    PIT->CHANNEL[0].LDVAL = 23999999;
13.
14.    // Activarea întreruperilor pe canalul 0 și 1
15.    PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TIE_MASK;
16.    PIT->CHANNEL[1].TCTRL |= PIT_TCTRL_TIE_MASK;
17.
18.    // Activarea timerului de pe canalul 0 și 1
19.    PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TEN_MASK;
20.    PIT->CHANNEL[1].TCTRL |= PIT_TCTRL_TEN_MASK;
21.
22.    // Activarea întreruperii mascabile și setarea priorității
23.    NVIC_ClearPendingIRQ(PIT_IRQn);
24.    NVIC_SetPriority(PIT_IRQn,5);
25.    NVIC_EnableIRQ(PIT_IRQn);
26. }
27.
```

Se activează semnalul de ceas folosind masca *SIM\_SCGC6\_PIT\_MASK*:

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

S-a setat semnalul de ceas pentru tabloul de timere (*PIT\_MCR &= ~PIT\_MCR\_MDIS\_MASK;*) și s-a oprit decrementarea valorii numărătoarelor în modul debug (*PIT->MCR |= PIT\_MCR\_FRZ\_MASK;*):

**PIT\_MCR field descriptions**

Field	Description
0–28 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
29 Reserved	This field is reserved.
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.

S-au setat valorile celor 2 canale ce semnifică perioada de timp pentru întreruperile timer-ului, astfel:

- Canalul 1 a fost folosit pentru controlul LED-urilor și setat la 56879999(calculată folosind formula:  $LDVAL = Durată * Frecvență_{ceas} - 1$ ) care înseamnă 1119 ms;
- Canalul 0 a fost folosit pentru modulul ADC și este setat la 23999999(calculată folosind formula:  $LDVAL = Durată * Frecvență_{ceas} - 1$ ) care înseamnă 0,5 s.

Cele 2 timere generează întreruperi la perioadele de timp setate în registrul LDVAL. Ele își încarcă în acest registru valoarea, scade până la 0 și apoi își încarcă valoarea de start înapoi. Mereu când un timer ajunge la 0, acesta va genera un impuls de întrerupere și va seta flag-ul de întrerupere.

Apoi, am activat întreruperile(TCTRL\_TIE) și timer-ul(TCTRL\_TEN) pe ambele canale folosindu-ne de registrul TCTRL (Timer Control Register) care conține bitul de control pentru fiecare timer în parte.

30 TIE	<p>Timer Interrupt Enable</p> <p>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.</p> <p>0 Interrupt requests from Timer n are disabled. 1 Interrupt will be requested whenever TIF is set.</p>
31 TEN	<p>Timer Enable</p> <p>Enables or disables the timer.</p> <p>0 Timer n is disabled. 1 Timer n is enabled.</p>

Funcția *PIT\_IRQHandler* este apelată atunci când se declanșează întreruperea de către modulul PIT:

```

1. void PIT_IRQHandler(void) {
2.
3.     if(PIT->CHANNEL[0].TFLG & PIT_TFLG_TIF_MASK) {
4.         ADC0->SC1[0] |= ADC_SC1_AIEN_MASK;
5.         PIT->CHANNEL[0].TFLG &= PIT_TFLG_TIF_MASK;
6.     }
7.     if(PIT->CHANNEL[1].TFLG & PIT_TFLG_TIF_MASK){
8.         PIT->CHANNEL[1].TFLG &= PIT_TFLG_TIF_MASK;
9.     }
10.
11.
12.     if(flag_switch==0){
13.         switch (ledState) {
14.             case 0: // Red

```

```

15.                                     Control_RGB_LEDs(1, 0, 0);
16.                                     break;
17.                                     case 1: // Green
18.                                     Control_RGB_LEDs(0, 1, 0);
19.                                     break;
20.                                     case 2: // Blue
21.                                     Control_RGB_LEDs(0, 0, 1);
22.                                     break;
23.                                     default: // Turn off LEDs -- Black
24.                                     Control_RGB_LEDs(0, 0, 0);
25.                                     break;
26.     }
27. }
28. else{
29.     switch (ledState) {
30.         case 0: // blue
31.             Control_RGB_LEDs(0, 0, 1);
32.             break;
33.         case 1: // Green
34.             Control_RGB_LEDs(0, 1, 0);
35.             break;
36.         case 2: // red
37.             Control_RGB_LEDs(1, 0, 0);
38.             break;
39.         default: // Turn off LEDs -- Black
40.             Control_RGB_LEDs(0, 0, 0);
41.             break;
42.     }
43. }
44.
45. // Increment LED state
46. ledState++;
47.
48. // Check for overflow
49. if (ledState > 3) {
50.     ledState = 0;
51. }
52. }
53. }
54.

```

Se verifică dacă întreruperea a fost declanșată pentru canalul 0 al PIT, apoi se setează un bit pentru controlul ADC și se șterge flag-ul de întrerupere pentru canalul 0 al PIT-ului:

```

if(PIT->CHANNEL[0].TFLG & PIT_TFLG_TIF_MASK) {
5.     ADC0->SC1[0] |= ADC_SC1_AIEN_MASK;
6.     PIT->CHANNEL[0].TFLG &= PIT_TFLG_TIF_MASK;
7. }

```

După se verifică dacă întreruperea a fost declanșată pentru canalul 1 al PIT, apoi se șterge flag-ul de întrerupere pentru canalul 1 al PIT-ului și începe schimbarea culorii LED-urilor în funcție de flag-ul flag\_switch(1=RGB și 0=BGR) și în funcție de starea curentă a variabilei ledState.



### PIT\_TFLGn field descriptions

Field	Description
0–30 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
31 TIF	Timer Interrupt Flag  Sets to 1 at the end of the timer period. Writing 1 to this flag clears it. Writing 0 has no effect. If enabled, or, when TCTRLn[TIE] = 1, TIF causes an interrupt request.  0 Timeout has not yet occurred. 1 Timeout has occurred.

#### 4.1.4. Inițializarea modului ADC și prelucrarea datelor înregistrate de către senzor

Setarea canalului pentru senzorul de sunet și inițializarea unor variabile:

```
1. #define ADC_CHANNEL (11) // PORT C PIN 2
2. volatile uint8_t flag=0;
3. uint8_t analog_input=0;
```

J10_10	A4	PTC2	57	ADC0_SE11/TS10_CH15	PTC2	I2C1_SDA	FTM0_CH1
--------	----	------	----	---------------------	------	----------	----------

Inițializarea modului ADC:

```
1. void ADC0_Init() {
2.
3.     // Activarea semnalului de ceas pentru modulul periferic ADC
4.     SIM->SCGC6 |= SIM_SCGC6_ADC0_MASK;
5.
6.     // Functia de calibrare
7.     ADC0_Calibrate();
8.
9.     ADC0->CFG1 = 0x00;
10.
11.     // Selectarea modului de conversie pe 8 biti single-ended --> MODE
12.     // Selectarea sursei de ceas pentru generarea ceasului intern --> ADICLK
13.     // Selectarea ratei de divizare folosit de periferic pentru generarea ceasului intern -->
ADIV
14.     // Set ADC clock frequency fADCK less than or equal to 4 MHz (PG. 494)
15.     ADC0->CFG1 |= ADC_CFG1_MODE(0) |
16.                                     ADC_CFG1_ADICLK(0) |
17.                                     ADC_CFG1_ADIV(2);
18.
19.     // DIFF = 0 --> Conversii single-ended (PG. 464)
20.     ADC0->SC1[0] = 0x00;
21.     ADC0->SC3 = 0x00;
22.
23.     // Selectarea modului de conversii continue,
24.     // pentru a-l putea folosi in tandem cu mecanismul de intreruperi
25.     ADC0->SC3 |= ADC_SC3_ADC0_MASK;
26.
27.     // Activarea subsistemului de conversie prin aproximari succesive pe un anumit canal
(PG.464)
28.     ADC0->SC1[0] |= ADC_SC1_ADCH(ADC_CHANNEL);
29.     // Enables conversion complete interrupts
```

```

30.     ADC0->SC1[0] |= ADC_SC1_AIEN_MASK;
31.
32.     NVIC_ClearPendingIRQ(ADC0_IRQn);
33.     NVIC_EnableIRQ(ADC0_IRQn);
34.     //NVIC_DisableIRQ(ADC0_IRQn);
35. }

```

S-a setat modul pe 8 biți single-ended când DIFF=0:

*ADC\_CFG1\_MODE(0)*

<p>3-2 MODE</p>	<p>Conversion mode selection</p> <p>Selects the ADC resolution mode.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p>
<p>5 DIFF</p>	<p>Differential Mode Enable</p> <p>Configures the ADC to operate in differential mode. When enabled, this mode automatically selects from the differential channels, and changes the conversion algorithm and the number of cycles to complete a conversion.</p> <p>0 Single-ended conversions and input channels are selected.</p> <p>1 Differential conversions and input channels are selected.</p>

S-a setat ca sursă de ceas bus clock:

*ADC\_CFG1\_ADICLK(0)*

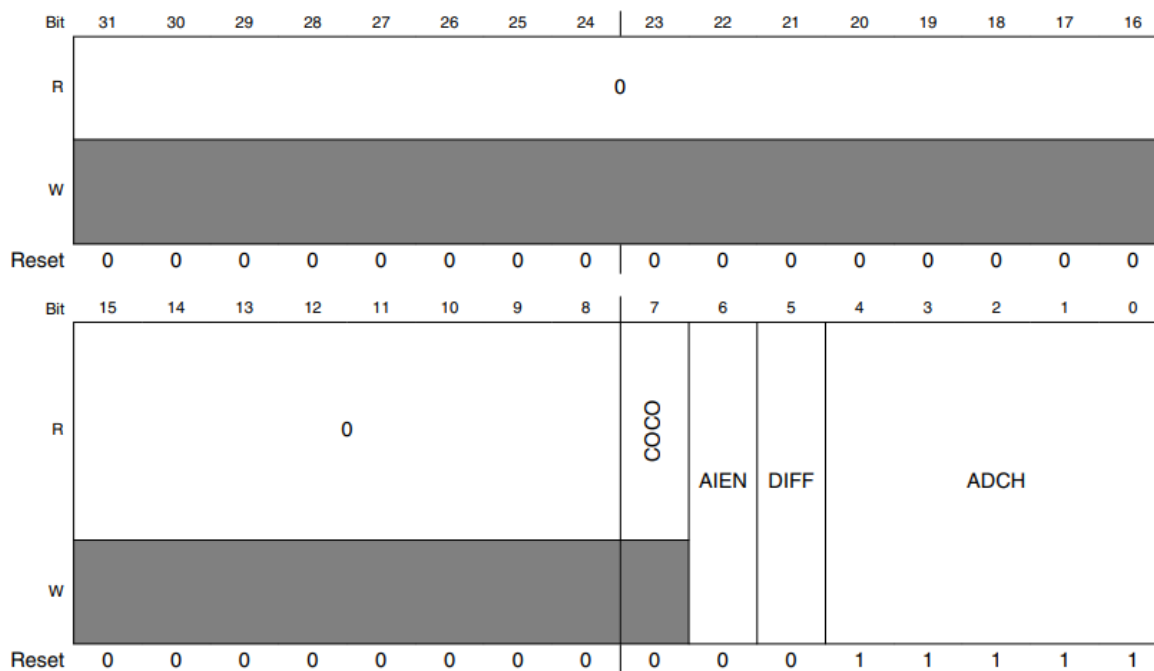
<p>1-0 ADICLK</p>	<p>Input Clock Select</p> <p>Selects the input clock source to generate the internal clock, ADCK. Note that when the ADACK clock source is selected, it is not required to be active prior to conversion start. When it is selected and it is not active prior to a conversion start, when 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.</p> <p>00 Bus clock</p> <p>01 (Bus clock)/2</p> <p>10 Alternate clock (ALTCLK)</p> <p>11 Asynchronous clock (ADACK)</p>
-----------------------	--

și ca rată de divizare:

*ADC\_CFG1\_ADIV(2)*

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

Activăm conversiile single-ended.



#### ADCx\_SC1n field descriptions

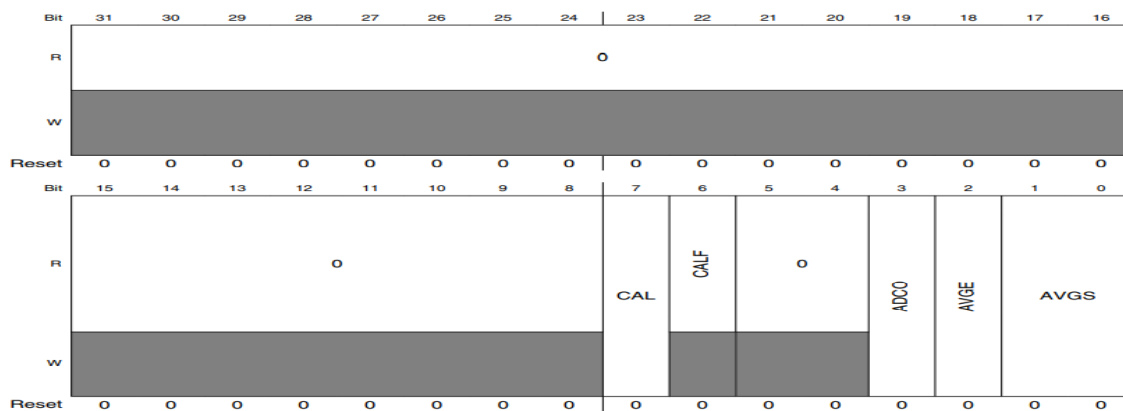
Activăm întreruperile la terminarea conversiilor:

$ADC0 \rightarrow SC1[0] \neq ADC\_SC1\_AIEN\_MASK;$

6 AIEN	<p>Interrupt Enable</p> <p>Enables conversion complete interrupts. When COCO becomes set while the respective AIEN is high, an interrupt is asserted.</p> <p>0 Conversion complete interrupt is disabled.</p> <p>1 Conversion complete interrupt is enabled.</p>
-----------	--

Ne asigurăm că registrul de status și control 3 este 0 înainte de a face calibrările necesare:

$ADC0 \rightarrow SC3 = 0x00;$



#### ADCx\_SC3 field descriptions

Activăm conversiile continue pentru a-l putea folosi cu mecanismul de întreruperi:

*ADC0->SC3 /= ADC\_SC3\_ADCO\_MASK;*

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

Selectăm canalul specific senzorului (11 pentru senzorul de sunet):

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

4-0 ADCH	<p>Input channel select</p> <p>Selects one of the input channels. The input channel decode depends on the value of DIFF. DAD0-DAD3 are associated with the input pin pairs DADPx and DADMx.</p> <p><b>NOTE:</b> Some of the input channel options in the bitfield-setting descriptions might not be available for your device. For the actual ADC channel assignments for your device, see the Chip Configuration details.</p> <p>The successive approximation converter subsystem is turned off when the channel select bits are all set, that is, ADCH = 11111. This feature allows explicit disabling of the ADC and isolation of the input channel from all sources. Terminating continuous conversions this way prevents an additional single conversion from being performed. It is not necessary to set ADCH to all 1s to place the ADC in a low-power state when continuous conversions are not enabled because the module automatically enters a low-power state when a conversion completes.</p> <p>00000 When DIFF=0, DADP0 is selected as input; when DIFF=1, DAD0 is selected as input.  00001 When DIFF=0, DADP1 is selected as input; when DIFF=1, DAD1 is selected as input.  00010 When DIFF=0, DADP2 is selected as input; when DIFF=1, DAD2 is selected as input.  00011 When DIFF=0, DADP3 is selected as input; when DIFF=1, DAD3 is selected as input.  00100 When DIFF=0, AD4 is selected as input; when DIFF=1, it is reserved.  00101 When DIFF=0, AD5 is selected as input; when DIFF=1, it is reserved.</p>
-------------	--

Funcția de calibrare pentru conversii ale modulului ADC, conform manualului de referință al platformei doar că s-a setat modul pe 8 biți single-ended:

*ADC\_CFG1\_MODE(0)*

```

1. int ADC0_Calibrate() {
2.
3.     // ===== For best calibration results =====
4.
5.     ADC0_CFG1 |= ADC_CFG1_MODE(0) |                               // 8 bits mode
6.                 ADC_CFG1_ADICLK(1) | // Input Bus Clock divided by 2
7.                 ADC_CFG1_ADIV(3);  // Clock divide by 8
8.
9.     // The calibration will automatically begin if the SC2[ADTRG] is 0. (PG. 495)
10.    ADC0->SC2 &= ~ADC_SC2_ADTRG_MASK;
11.}

```

```

12.    // Set hardware averaging to maximum, that is, SC3[AVGE]=1 and SC3[AVGS]=0x11 for an
average of 32 (PG. 494)
13.    ADC0->SC3 |= (ADC_SC3_AVGE_MASK | ADC_SC3_AVGS(3));
14.
15.    // To initiate calibration, the user sets SC3[CAL] (PG. 495)
16.    ADC0->SC3 |= ADC_SC3_CAL_MASK;
17.
18.    // At the end of a calibration sequence, SC1n[COCO] will be set (PG. 495)
19.    while(!(ADC0->SC1[0] & ADC_SC1_COCO_MASK));
20.
21.    // At the end of the calibration routine, if SC3[CALF] is not
22.    // set, the automatic calibration routine is completed successfully. (PG. 495)
23.    if(ADC0->SC3 & ADC_SC3_CALF_MASK){
24.        return (1);
25.    }
26.
27.    // ===== CALIBRATION FUNCTION (PG.495) =====
28.
29.    // 1. Initialize or clear a 16-bit variable in RAM.
30.    uint16_t calibration_var = 0x0000;
31.
32.    // 2. Add the plus-side calibration results CLP0, CLP1, CLP2, CLP3, CLP4, and CLPS to the
variable.
33.    calibration_var += ADC0->CLP0;
34.    calibration_var += ADC0->CLP1;
35.    calibration_var += ADC0->CLP2;
36.    calibration_var += ADC0->CLP3;
37.    calibration_var += ADC0->CLP4;
38.    calibration_var += ADC0->CLPS;
39.
40.    // 3. Divide the variable by two.
41.    calibration_var /= 2;
42.
43.    // 4. Set the MSB of the variable.
44.    calibration_var |= 0x8000;
45.
46.    // 5. Store the value in the plus-side gain calibration register PG.
47.    ADC0->PG = ADC_PG_PG(calibration_var);
48.
49.    // 6. Repeat the procedure for the minus-side gain calibration value.
50.    calibration_var = 0x0000;
51.
52.    calibration_var += ADC0->CLM0;
53.    calibration_var += ADC0->CLM1;
54.    calibration_var += ADC0->CLM2;
55.    calibration_var += ADC0->CLM3;
56.    calibration_var += ADC0->CLM4;
57.    calibration_var += ADC0->CLMS;
58.
59.    calibration_var /= 2;
60.
61.    calibration_var |= 0x8000;
62.
63.    ADC0->MG = ADC_MG_MG(calibration_var);
64.
65.    // Incheierea calibrarii
66.    ADC0->SC3 &= ~ADC_SC3_CAL_MASK;
67.
68.    return (0);
69. }
70.

```

Funcția *ADC0\_Read* citește valori de la senzorul de sunet analogic:

```

1. uint8_t ADC0_Read(){
2.
3.     // A conversion is initiated following a write to SC1A, with SC1n[ADCH] not all 1's (PG.
485)
4.     ADC0->SC1[0] |= ADC_SC1_ADCH(ADC_CHANNEL);
5.
6.     // ADACT is set when a conversion is initiated
7.     // and cleared when a conversion is completed or aborted.
8.     while(ADC0->SC2 & ADC_SC2_ADACT_MASK);
9.
10.    // A conversion is completed when the result of the conversion is transferred
11.    // into the data result registers, Rn (PG. 486)
12.
13.    // If the compare functions are disabled, this is indicated by setting of SC1n[COC0]
14.    // If hardware averaging is enabled, the respective SC1n[COC0] sets only if
15.    // the last of the selected number of conversions is completed (PG. 486)
16.    while(!(ADC0->SC1[0] & ADC_SC1_COC0_MASK));
17.    return (uint8_t) ADC0->R[0];
18.
19. }
```

La fiecare întrerupere de ceas, funcția *ADC0\_IRQHandler* preia valorile de la senzorul de sunet din *registrul ADC0\_R* pe care le stochează în variabila *analog\_input* și dezactivează întreruperile:

```

1. void ADC0_IRQHandler(){
2.
3.     analog_input = (uint8_t) ADC0->R[0];
4.
5.     flag=1;
6.     ADC0->SC1[0] &= ~ADC_SC1_AIEN_MASK;
7.
8. }
```

Address: 4003\_B000h base + 10h offset + (4d × i), where i=0d to 1d

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																D															
W																																
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	

**ADCx\_Rn field descriptions**

Field	Description
31–16 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
15–0 D	Data result

### 4.1.5. Inițializarea modului TPM și generare PWM

Vom folosi modulul TPM2 (Timer/PWM Module) pentru generarea semnalului PWM.

Setarea canalului pentru servomotor:

```
1. #define SERVO_PIN (2) // PORT B , PIN 2
2.
3. // Activarea semnalului de ceas pentru utilizarea LED-ului de culoare rosie
4. SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK;
5.
6. // Utilizarea alternativei de functionare pentru perifericul TMP
7. // TMP2_CH0
8. PORTB->PCR[SERVO_PIN] |= PORT_PCR_MUX(3);
9.
11. // Selects the clock source for the TPM counter clock (MCGFLLCLK) - PG. 196
12. // MCGFLLCLK Freq. - 20 MHz
13. SIM->SOPT2 |= SIM_SOPT2_TPMSRC(1);
14.
15. // Activarea semnalului de ceas pentru modulul TPM
16. SIM->SCGC6 |= SIM_SCGC6_TPM2(1);
17.
18. // Divizor de frecventa pentru ceasul de intrare
19. // PWM CLK -> 48MHz / 128 = 48.000.000 / 128 [Hz] = 375.000 [Hz] = 375 kHz
20. TPM2->SC |= TPM_SC_PS(7);
21.
22. // LPTPM counter operates in up counting mode. - PG. 553
23. // Selects edge aligned PWM
24. TPM2->SC |= TPM_SC_CPWMS(0);
25.
26. // LPTPM counter increments on every LPTPM counter clock
27. TPM2->SC |= TPM_SC_CMOD(1);
28.
29.
30. // LPTPM counter operates in up-down counting mode. - PG. 553
31. // Selects center aligned PWM
32. //TPM2->SC |= TPM_SC_CPWMS(1);
33.
34.
35. // Edge-Aligned Pulse Width Modulation
36. // TPM->SC[CPWMS] = 0
37.
38. // ===== Mode selection =====
39. // Configured for EPWM
40. // TPM->CnSC[MnSB] = 1
41. // TPM->CnSC[MnSB] = 0
42.
43. // ===== Edge selection =====
44. // Set output on counter overflow, clear output on match
45. // Counter overflow = LPTPM counter reaches the value configured in the MOD register, and
then resets
46. // Match = LPTPM counter reaches the value configured in the CnV register
47.
48. // TPM->CnSC[ELSnB] = 1
49. // TPM->CnSC[ELSnA] = 0
50.
51. // Regiter associated to the control of channel 0
52. // Why channel 0?
```

```

53.     TPM2->CONTROLS[0].CnSC |= (TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK);
54. }
55.

```

Vom porni ceasul portului B, unde se află servomotorul conectat prin setarea bitului al 9-lea al registrului SIM\_SCGC5 pe valoarea 1 cu ajutorul măștii SIM\_SCGC5\_PORTB\_MASK:

10 PORTB	Port B Clock Gate Control This bit controls the clock gate to the Port B module.
<i>Table continues on the next page...</i>	
0	Clock disabled
1	Clock enabled

Setăm multiplexarea pinului 2 din portul B pe canalul 0 al modului 2 TPM:

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

Setăm FLL ca sursă de ceas prin setarea câmpului TPMSRC (biții 25-24, TPM clock source select) din registrul System Options Register 2(SIM\_SOPT2) pe valoarea 1(MCGFLLCLK).

```
SIM->SOPT2 /= SIM_SOPT2_TPMSRC(1);
```

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

Activăm ceasul modului 2 TPM prin setarea câmpului TPM2 din SIM\_SCGC6 pe 1.

```
SIM->SCGC6 /= SIM_SCGC6_TPM2(1);
```

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



Prin setarea câmpului PS(Prescale Factor Selection) pe valoarea 0b111, ceasul sistemului va fi divizat cu 128, deci ceasul PWM va avea frecvența de 375 KHz:

$TPM2 \rightarrow SC \neq TPM\_SC\_PS(7);$

2-0 PS	<b>Prescale Factor Selection</b> 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
-----------	--

Setăm semnalul PWM generat ca fiind edge-aligned, setând counterul în mod up counting prin valoarea 0 în câmpul CPWMS(Center-aligned PWM Select) și valoarea 01 în câmpul CMOD(Clock Mode Selection) pentru incrementarea numărătorului LPTPM.

5 CPWMS	<b>Center-aligned PWM Select</b> 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	<b>Clock Mode Selection</b> Selects the LPTPM counter clock modes. When disabling the counter, this field remain set until acknowledged 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

Funcția *Signal\_Control* are rolul de a seta valoarea pentru counter-ul LPTPM (valoarea până la care trebuie să numere pentru a termina o perioadă de 20ms adică 7500) și de a seta pentru canalul 0 anumite valori specifice modului de rotire a elicei servomotorului. Valori necesare pentru calcularea factorului de umplere(duty-cycle):

- $TPM2 \rightarrow CONTROLS[0].CnV = 188;$  - a fost calculat pentru unghiul de  $0^\circ$  și reprezintă 2.5% din LPTPM Counter;

- TPM2->CONTROLS[0].CnV = 562; - a fost calculat pentru unghiul de 90° și reprezintă 7.5% din LPTPM Counter;
- TPM2->CONTROLS[0].CnV = 937; - a fost calculat pentru unghiul de 180° și reprezintă 12.5% din LPTPM Counter

```

1. void Signal_Control(uint8_t position){
2.
3.     // Resetarea valorii numaratorului asociat LPTPM Counter
4.     TPM2->MOD = 7500; // 375,000 Hz / 7500 = 50 Hz (20 ms)
5.
6.     // Setarea duty cycle-ului pentru 3 pozitii
7.     switch (position) {
8.         case 0:
9.             TPM2->CONTROLS[0].CnV = 188; // 2.5% pentru 0°
10.            break;
11.        case 1:
12.            TPM2->CONTROLS[0].CnV = 562; // 7.5% pentru 90°
13.            break;
14.        case 2:
15.            TPM2->CONTROLS[0].CnV = 937; // 12.5% pentru 180°
16.            break;
17.    }
18. }
19.

```

**TPMx\_MOD field descriptions**

Field	Description
31–16 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
15–0 MOD	Modulo value When writing this field, all bytes must be written at the same time.

**TPMx\_CnV field descriptions**

Field	Description
31–16 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
15–0 VAL	Channel Value Captured LPTPM counter value of the input modes or the match value for the output modes. When writing this field, all bytes must be written at the same time.

## 4.2. Configurarea ceasului

Modulul MCG(Multipurpose Clock Generator) oferă semnale de ceas pentru unitatea centrală. Acesta conține un FLL (frequency-locked loop), controlabil de către un ceas de referință intern.

```

1. void SystemClock_Configure(void) {
2.
3.
4.     // MCGOUTCLOCK are ca si iesire, selectia FLL

```

```

5.     MCG->C1 |= MCG_C1_CLKS(0);
6.
7.     // Selectarea ceasului intern ca si referinta pentru sursa FLL
8.     MCG->C1 |= MCG_C1_IREFS_MASK;
9.
10.    // Setarea frecventei ceasului digital la 48 Mhz
11.    // MCG->C4[DRST_DRS] = 1
12.    // MCG->C4[DMX32] = 1
13.    MCG->C4 |= MCG_C4_DRST_DRS(1);
14.    MCG->C4 |= MCG_C4_DMX32(1);
15.
16. }
17.

```

În MCG\_C1(MCG Control 1 Register) am setat câmpul CLKS(Clock Source Select) pe valoarea 0 pentru selectarea FLL/PLL ca output.

### 24.3.1 MCG Control 1 Register (MCG\_C1)

Address: 4006\_4000h base + 0h offset = 4006\_4000h

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

MCG\_C1 field descriptions

Field	Description
7-6 CLKS	<p>Clock Source Select</p> <p>Selects the clock source for MCGOUTCLK .</p> <p>00 Encoding 0 — Output of FLL or PLL is selected (depends on PLLS control bit).</p> <p>01 Encoding 1 — Internal reference clock is selected.</p> <p>10 Encoding 2 — External reference clock is selected.</p> <p>11 Encoding 3 — Reserved.</p>

Cel folosit va fi selectat prin câmpul IREFS(Internal Reference Select) pe valoarea 1 – se selectează ceasul intern de referință, așadar se folosește FLL (PLL – phase-locked loop este configurabil doar extern).

2 IREFS	<p>Internal Reference Select</p> <p>Selects the reference clock source for the FLL.</p> <p>0 External reference clock is selected.</p> <p>1 The slow internal reference clock is selected.</p>
------------	--

În MCG\_C4(MCG Control 4 Register), setăm câmpul DRST DRS pe valoarea 1 – se selectează frecvența DCO(Digitally-controlled oscillator) ca fiind una medie (40 – 50 MHz).

În același registru setăm câmpul DMX32(DCO Maximum Frequency) pe valoarea 1 – DCO va avea frecvența de 48 MHz.

### MCG\_C4 field descriptions

Field	Description																																									
7 DMX32	<p>DCO Maximum Frequency with 32.768 kHz Reference</p> <p>The DMX32 bit controls whether the DCO frequency range is narrowed to its maximum frequency with a 32.768 kHz reference.</p> <p>The following table identifies settings for the DCO frequency range.</p> <p><b>NOTE:</b> The system clocks derived from this source should not exceed their specified maximums.</p> <table><tr><th>DRST_DRS</th><th>DMX32</th><th>Reference Range</th><th>FLL Factor</th><th>DCO Range</th></tr><tr><td rowspan="2">00</td><td>0</td><td>31.25–39.0625 kHz</td><td>640</td><td>20–25 MHz</td></tr><tr><td>1</td><td>32.768 kHz</td><td>732</td><td>24 MHz</td></tr><tr><td rowspan="2">01</td><td>0</td><td>31.25–39.0625 kHz</td><td>1280</td><td>40–50 MHz</td></tr><tr><td>1</td><td>32.768 kHz</td><td>1464</td><td>48 MHz</td></tr><tr><td rowspan="2">10</td><td>0</td><td>31.25–39.0625 kHz</td><td>1920</td><td>60–75 MHz</td></tr><tr><td>1</td><td>32.768 kHz</td><td>2197</td><td>72 MHz</td></tr><tr><td rowspan="2">11</td><td>0</td><td>31.25–39.0625 kHz</td><td>2560</td><td>80–100 MHz</td></tr><tr><td>1</td><td>32.768 kHz</td><td>2929</td><td>96 MHz</td></tr></table> <p>0 DCO has a default range of 25%.</p> <p>1 DCO is fine-tuned for maximum frequency with 32.768 kHz reference.</p>	DRST_DRS	DMX32	Reference Range	FLL Factor	DCO Range	00	0	31.25–39.0625 kHz	640	20–25 MHz	1	32.768 kHz	732	24 MHz	01	0	31.25–39.0625 kHz	1280	40–50 MHz	1	32.768 kHz	1464	48 MHz	10	0	31.25–39.0625 kHz	1920	60–75 MHz	1	32.768 kHz	2197	72 MHz	11	0	31.25–39.0625 kHz	2560	80–100 MHz	1	32.768 kHz	2929	96 MHz
DRST_DRS	DMX32	Reference Range	FLL Factor	DCO Range																																						
00	0	31.25–39.0625 kHz	640	20–25 MHz																																						
	1	32.768 kHz	732	24 MHz																																						
01	0	31.25–39.0625 kHz	1280	40–50 MHz																																						
	1	32.768 kHz	1464	48 MHz																																						
10	0	31.25–39.0625 kHz	1920	60–75 MHz																																						
	1	32.768 kHz	2197	72 MHz																																						
11	0	31.25–39.0625 kHz	2560	80–100 MHz																																						
	1	32.768 kHz	2929	96 MHz																																						
6–5 DRST_DRS	<p>DCO Range Select</p> <p>The DRS bits select the frequency range for the FLL output, DCOOUT. When the LP bit is set, writes to the DRS bits are ignored. The DRST read field indicates the current frequency range for DCOOUT. The DRST field does not update immediately after a write to the DRS field due to internal synchronization between clock domains. See the DCO Frequency Range table for more details.</p> <p>00 Encoding 0 — Low range (reset default).</p> <p>01 Encoding 1 — Mid range.</p> <p>10 Encoding 2 — Mid-high range.</p> <p>11 Encoding 3 — High range.</p>																																									

### Configurarea întreruperii de ceas:

```

1. void SystemClockTick_Configure(void){
2.
3.     // Deoarece imi doresc ca frecventa de generare a semnalului PWM sa tina cont de tick-uri
    de 1 ms,
4.     // trebuie realizate urmatoarele configurari
5.     // 48.000.000 pulsuri ..... 1 secunda
6.     //           x           pulsuri ..... 10^(-3) secunde
7.     //
8.     SysTick->LOAD = (uint32_t)(48000000UL / 1000 - 1UL);
9.
10.    NVIC_SetPriority(SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL);
11.
12.    // Resetarea numaratorului digital
13.    SysTick->VAL = 0UL;
14.
15.    SysTick->CTRL |= (SysTick_CTRL_CLKSOURCE_Msk | // Ceasul procesorului utilizat ca
    referinta (48Mhz)
16.
17.    SysTick_CTRL_TICKINT_Msk | // Atunci cand numaratorul ajunge la 0, va porni handler-ul de
    intrerupere
18.    SysTick_CTRL_ENABLE_Msk); // Incarca valoarea pusa in registrul RELOAD si incepe
    numaratoarea
19. }

```

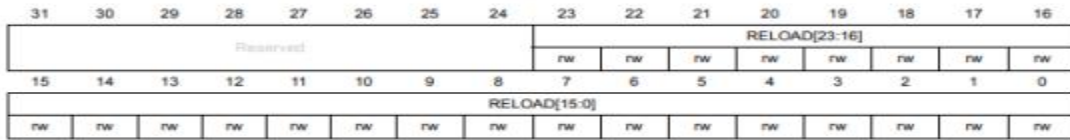
Setăm valoarea registrului de reload(STK\_LOAD) cu valoarea de resetare a timerului, raportul dintre cele două frecvențe, care specifică intervalul de numărare al counterului.

#### 4.5.2 SysTick reload value register (STK\_LOAD)

Address offset: 0x04

Reset value: 0x0000 0000

Required privilege: Privileged



Bits 31:24 Reserved, must be kept cleared.

Bits 23:0 **RELOAD[23:0]**: RELOAD value

The LOAD register specifies the start value to load into the VAL register when the counter is enabled and when it reaches 0.

Calculating the RELOAD value

The RELOAD value can be any value in the range 0x00000001-0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value is calculated according to its use:

- I To generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. For example, if the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.
- I To deliver a single SysTick interrupt after a delay of N processor clock cycles, use a RELOAD of value N. For example, if a SysTick interrupt is required after 400 clock pulses, set RELOAD to 400.

Setăm prioritatea întreruperii SysTick pe valoarea maximă și valoarea inițială pe 0. În registrul STK\_CTRL(registrul de control și stare SysTick) setăm biții 2 (CLKSOURCE – selectează ca sursă ceasul procesorului), 1(TICKINT – activează întreruperea la valoarea 0) și 0(ENABLE – pornește timerul).

#### 4.5.1 SysTick control and status register (STK\_CTRL)

Address offset: 0x00

Reset value: 0x0000 0000

Required privilege: Privileged

The SysTick CTRL register enables the SysTick features.



Bits 31:17 Reserved, must be kept cleared.

Bit 16 **COUNTFLAG**:

Returns 1 if timer counted to 0 since last time this was read.

Bits 15:3 Reserved, must be kept cleared.

Bit 2 **CLKSOURCE**: Clock source selection

Selects the clock source.

0: AHB/8

1: Processor clock (AHB)

Bit 1 **TICKINT**: SysTick exception request enable

0: Counting down to zero does not assert the SysTick exception request

1: Counting down to zero asserts the SysTick exception request.

Note: Software can use COUNTFLAG to determine if SysTick has ever counted to zero.

Bit 0 **ENABLE**: Counter enable

Enables the counter. When ENABLE is set to 1, the counter loads the RELOAD value from the LOAD register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.

0: Counter disabled

1: Counter enabled

### 4.3. Funcția main

Funcția main include toate fișierele header în care sunt declarate funcțiile și variabilele ce au fost folosite în fiecare modul și are rolul de a inițializa toate modulele descrise anterior:

```
1. #include "ClockSettings.h"
2. #include "Pwm.h"
3. #include "Pit.h"
4. #include "adc.h"
5. #include "gpio.h"
6. #include "uart.h"
```

```

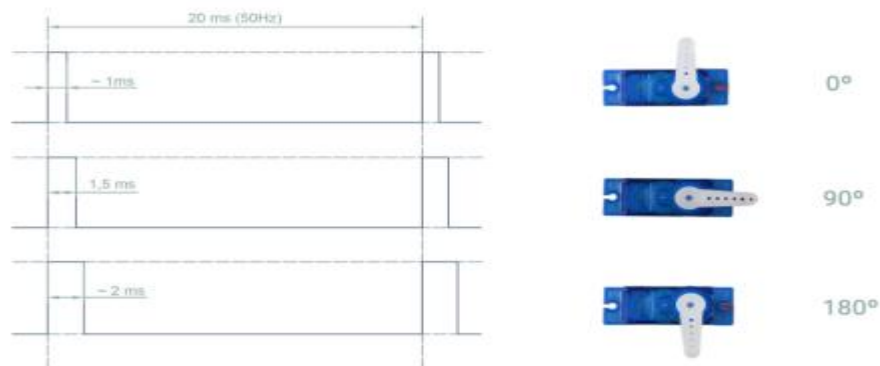
7.
8.
9. int main() {
10.
11.     SystemClock_Configure();
12.     SystemClockTick_Configure();
13.     OutputPIN_Init();
14.     TPM2_Init();
15.     UART0_Init(115200);
16.     ADC0_Init();
17.     PIT_Init();
18.
19.     for(;;){
20.         if(flag){
21.
22.             int i;
23.
24.             char v[8];
25.             int count = 0;
26.             int value;
27.
28.             value = (int)analog_input;
29.
30.             for (i = 0; i < 8; i++) {
31.                 v[i] = '-';
32.             }
33.
34.             while (value != 0) {
35.                 v[count] = (char)(value % 10) + 0x30;
36.                 value = value / 10;
37.                 count = count + 1;
38.             }
39.
40.             for (i = 0; i < count; i++) {
41.                 UART0_Transmit(v[count - i - 1]);
42.             }
43.
44.             UART0_Transmit(0x0A);
45.             UART0_Transmit(0x0D);
46.
47.
48.             flag=0;
49.         }
50.
51.         if(flag_500ms){
52.             if(analog_input < 23){
53.                 Signal_Control(0);
54.             }
55.             else if(analog_input >= 23 & analog_input < 46){
56.                 Signal_Control(1);
57.             }
58.             else if(analog_input > 46){
59.                 Signal_Control(2);
60.             }
61.             flag_500ms = 0U;
62.         }
63.     }
64. }
65.

```

Logica programului este următoarea:

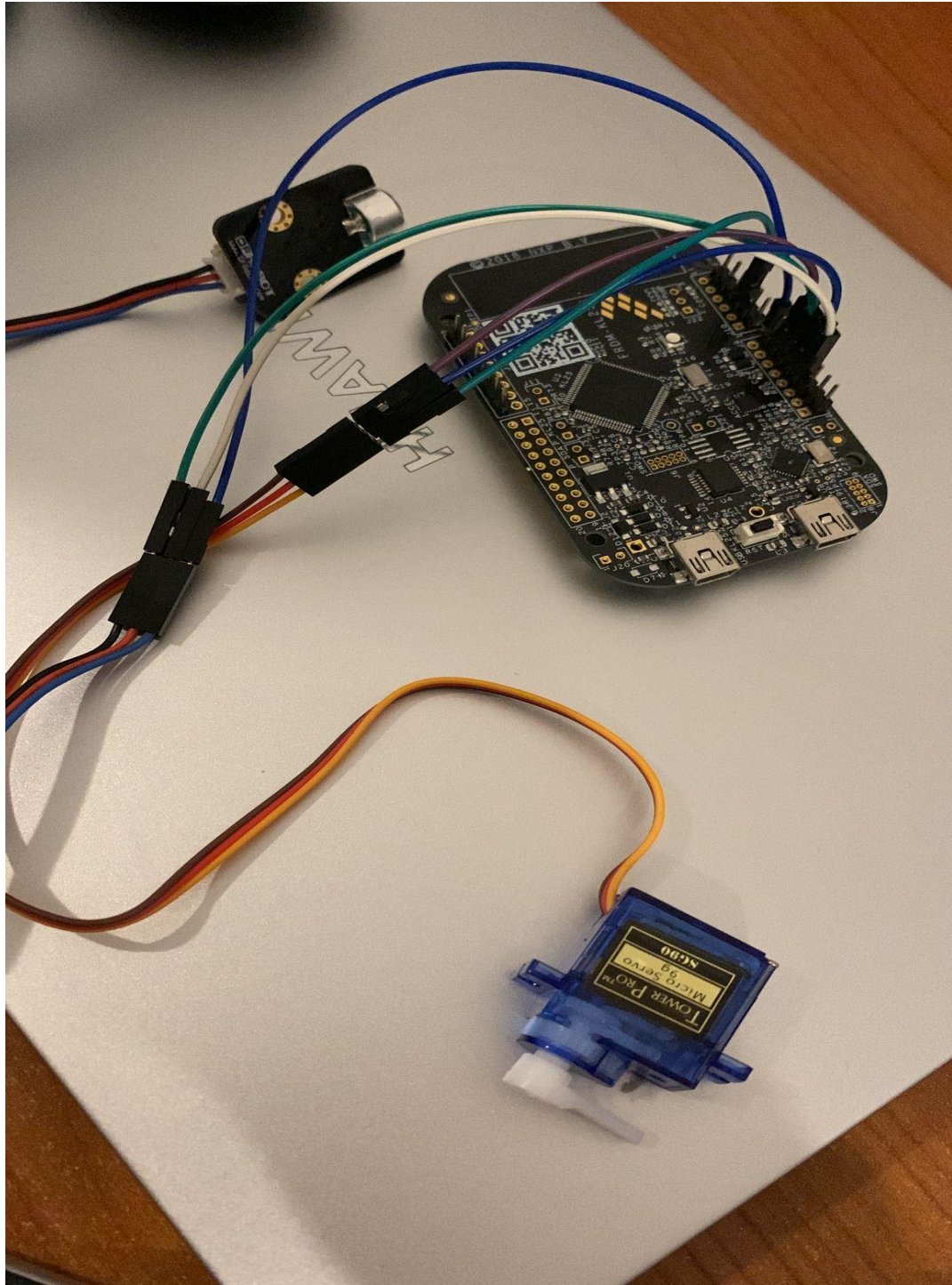
- Se apelează funcția de inițializare a timerelor și modulelor;
- În ciclul infinit se verifică dacă se primesc date de la senzorul de sunet. Dacă se primesc, acestea sunt transformate în numere întregi și stocate în variabila *value* care mai apoi sunt transformate într-un vector de caractere pentru a putea fi printabile în UART. La final are loc resetarea flag-ului la 0 pentru a indica faptul că acest bloc de cod s-a executat;
- După se verifică la fiecare trecere de 0.5s în ce interval se încadrează valoarea primită de la senzorul de sunet astfel:
  - [0, 23) – setează factorul de umplere la 188 specific pentru 2.5% pentru 0° ce determină poziția elicei servomotorului în sus;
  - [23, 46) - setează factorul de umplere la 562 specific pentru 7.5% pentru 90° ce determină poziția elicei servomotorului în dreapta;
  - [46, 68) - setează factorul de umplere la 937 specific pentru 12.5% pentru 180° ce determină poziția elicei servomotorului în jos;

*OBS: valoare max pentru stabilirea intervalului a fost măsurată anterior de noi prin transmiterea mai multor sunete prin intermediul senzorului de sunet și am constatat faptul că acesta are max=68; apoi a avut loc împărțirea intervalui în cele 3 subintervale specifice poziției elicei servomotorului.*








## 5. Set-up








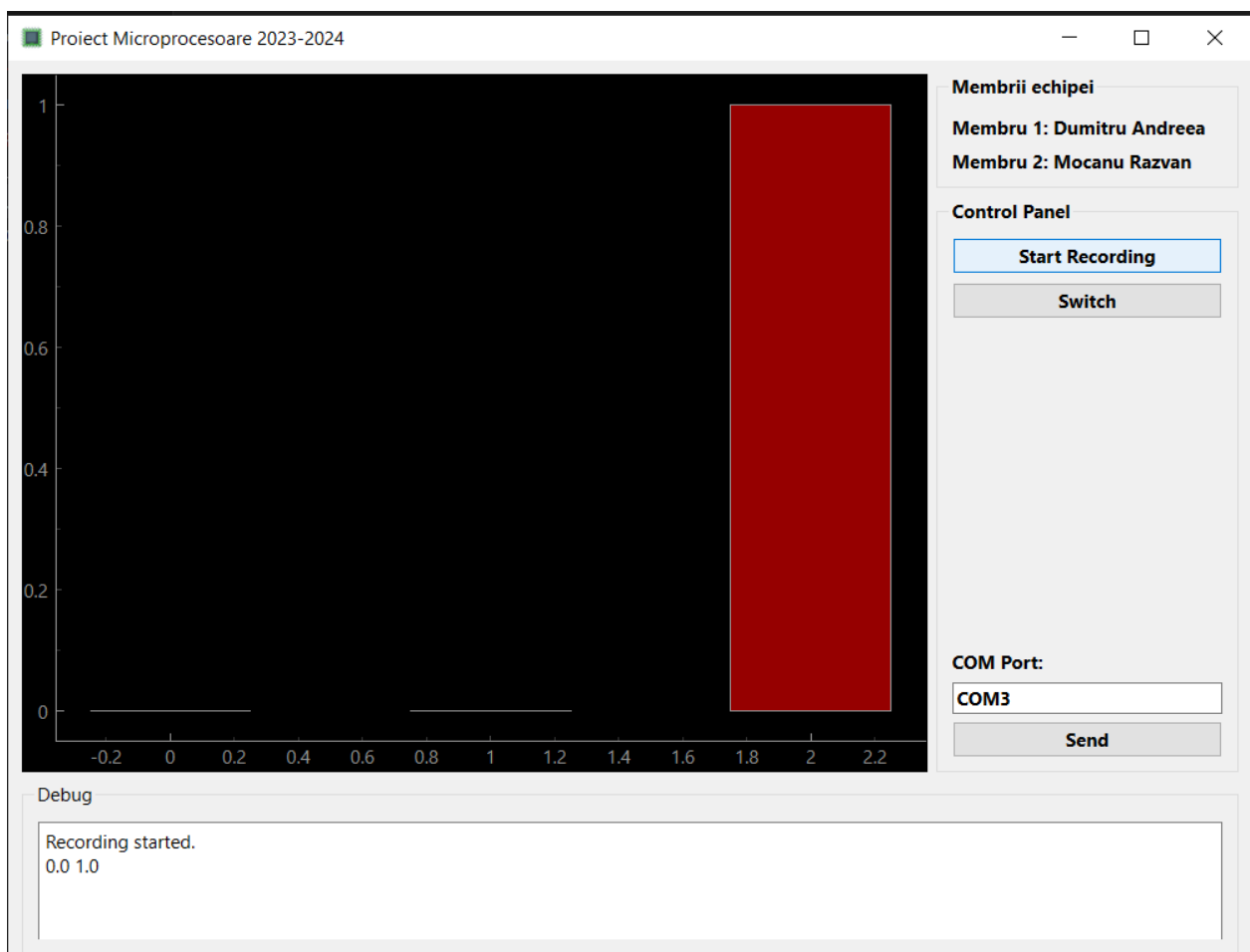
Senzorul de sunet a fost conectat pe plăcuță astfel:

-  **Firul roșu (VCC)** se conectează la pinul de 3V (P3V3);
-  **Firul negru (GND)** se conectează la pinul GND;
-  **Firul albastru (output sensor value)** se conectează la pinul PTC2.

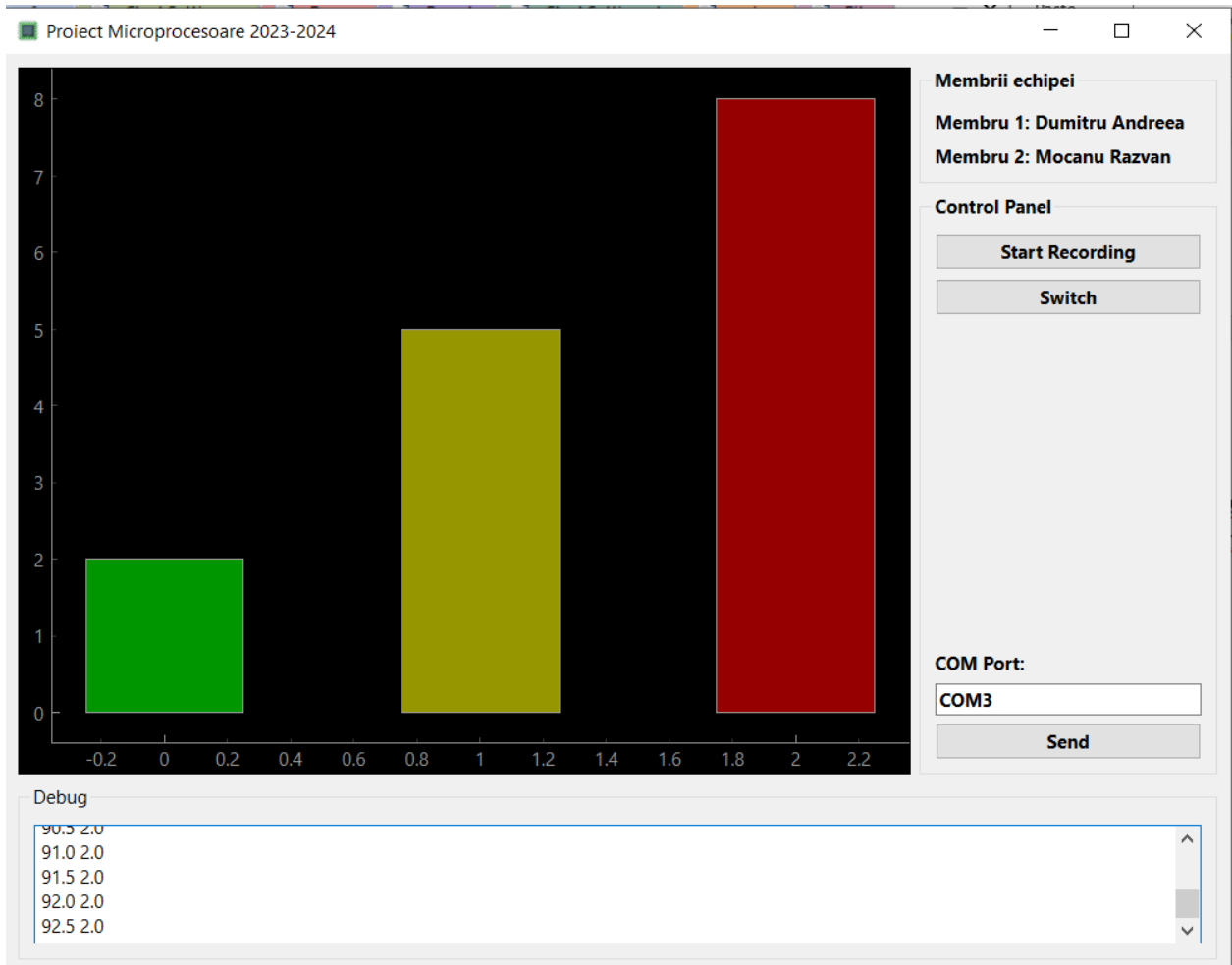
Servomotorul a fost conectat pe plăcuță astfel:

-  **Firul maroniu (GND)** se conectează la pinul GND;
-  **Firul roșu (VCC)** se conectează la pinul de 5V (P5V\_USB) ;
-  **Firul portocaliu (PWN)** se conectează la PTB2.

Se compilează proiectul din qVision făcând Build la cod urmat de Load pe plăcuță, iar apoi din Python se pornește interfața grafică unde se introduce portul pe care este conectat target-ul și se apasă butonul "Start Recording".

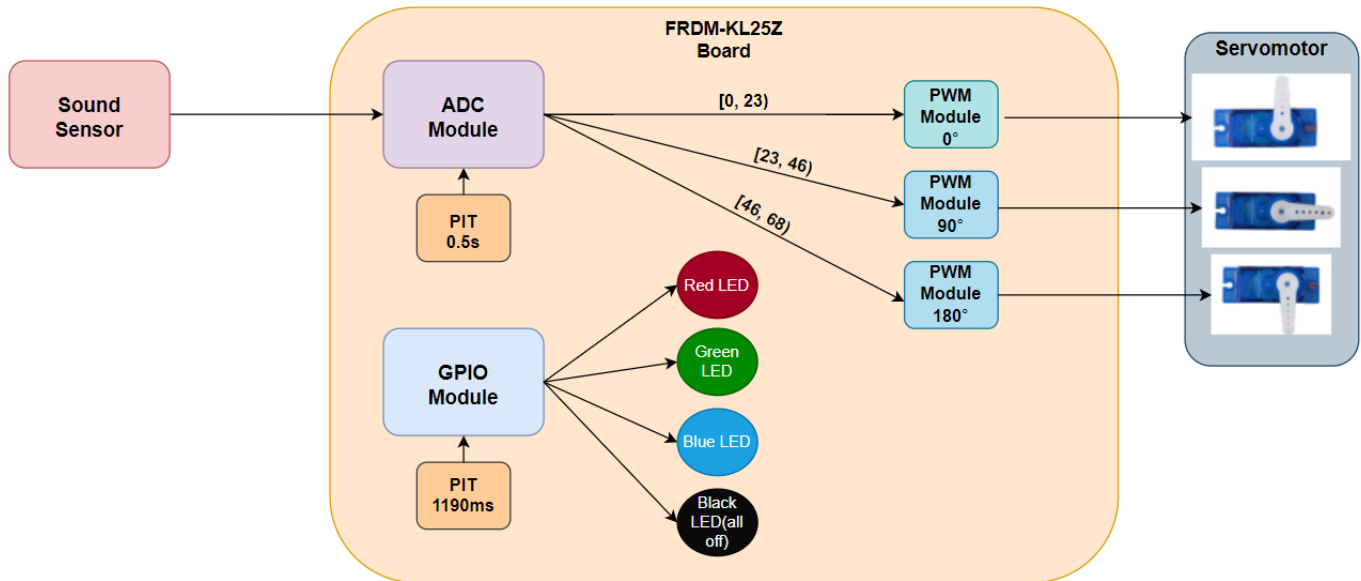


În grafic, se vor plota 3 barplot-uri ce reprezintă cele 3 intervale în care sunt valorile primite de la senzorul de sunet, ce sunt actualizate în mod dinamic.

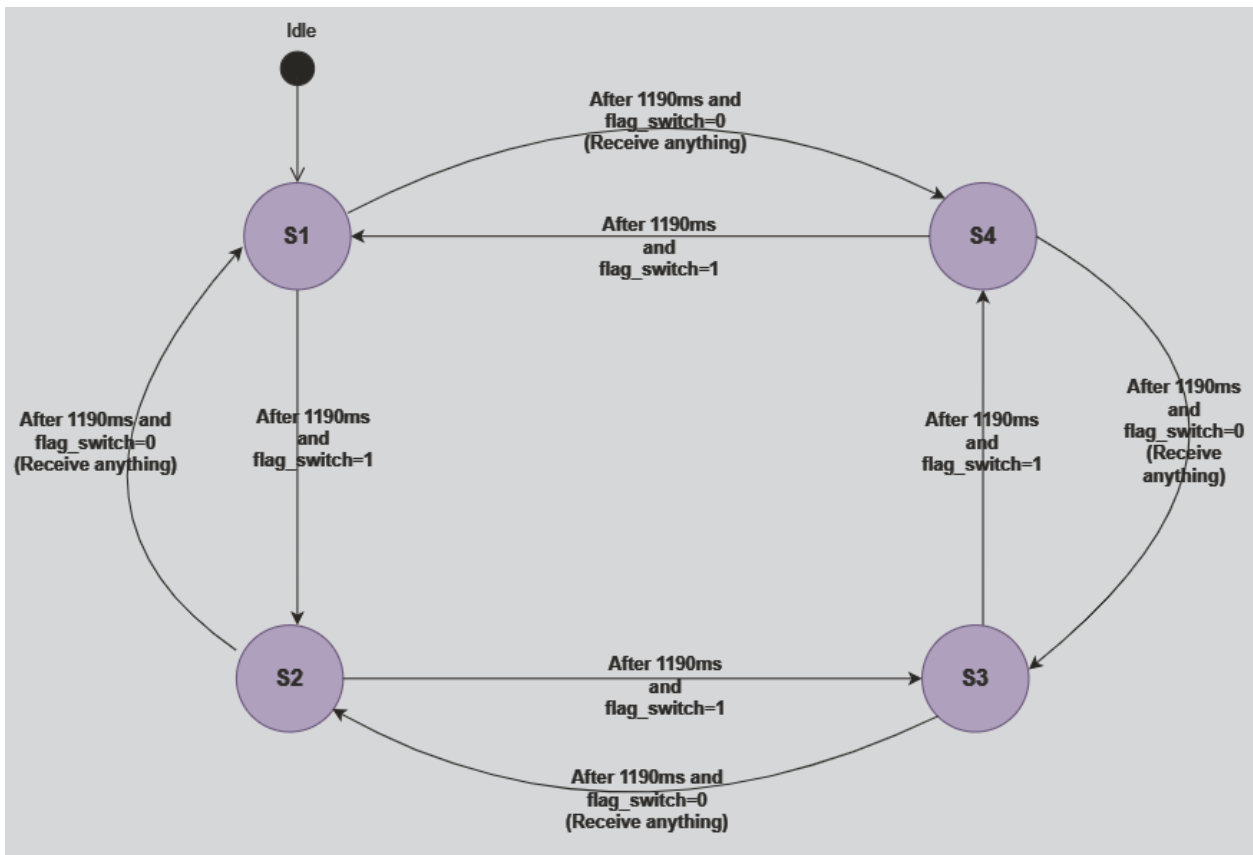


Pentru schimbarea secvenței de LED-uri, se va apăsa pe butonul "Switch" ceea ce va face inversare din RGB în BGR.

## 6. Schemă bloc

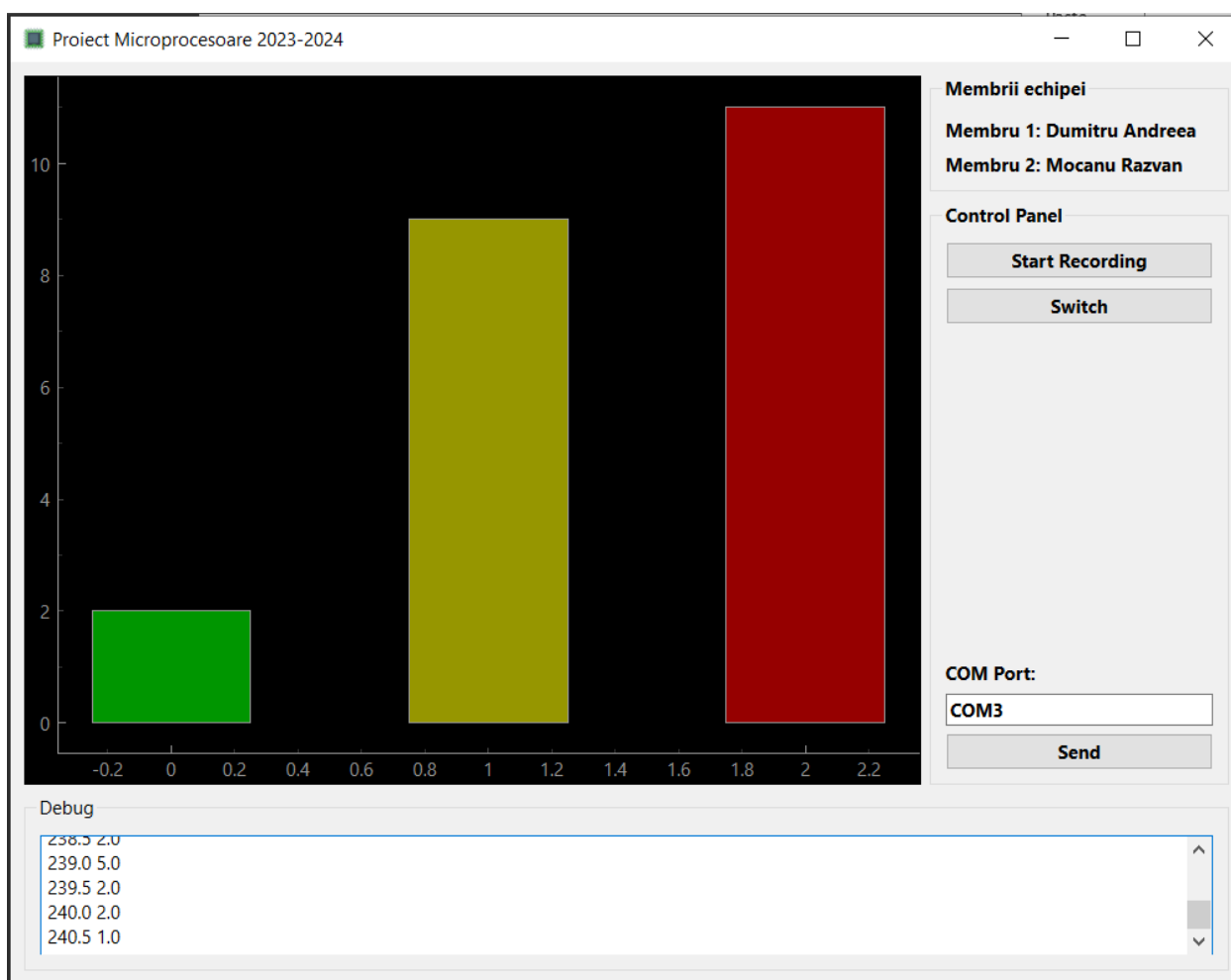


## 7. Diagrame de stări



Stare	ledState	Observații
S1	0	LED-ul roșu este aprins ( <b>Red</b> )
S2	1	LED-ul verde este aprins ( <b>Green</b> )
S3	2	LED-ul albastru este aprins ( <b>Blue</b> )
S4	default	Toate LED-urile sunt stinse ( <b>Black</b> )

## 8. Rezultate aplicație Python



În grafic, se vor plota 3 barplot-uri ce reprezintă cele 3 intervale în care sunt valorile primite de la senzorul de sunet, ce sunt actualizate în mod dinamic.

Pentru schimbarea secvenței de GPIO de LED-uri, se va apăsa pe butonul "Switch" ceea ce va face inversare din RGB în BGR.

## 9. Probleme întâmpinate

Printre probleme întâlnite se pot număra:

- + Sevomotorul primit nu era de 180 ° ci de 360 °;
- + Intervalele precizate în laborator nu erau corecte pentru generarea semnalului PWM;
- + DEFAULT\_SYSTEM\_CLOCK în laborator este 20,9MHz deși în comentariul din codul din laborator era precizat că este 48Mhz;
- + Platforma Tinkercad nu deține componentele necesare realizării reprezentării grafice a modului de conectare a senzorilor pe plăcuță.

## 10. Referințe

- + Freescale Semiconductor, Inc., KL25 Sub-Family Reference Manual, 2012
- + <https://datasheetspdf.com/pdf-file/791970/TowerPro/SG90/1>
- + [https://wiki.dfrobot.com/Analog\\_Sound\\_Sensor\\_SKU\\_DFR0034](https://wiki.dfrobot.com/Analog_Sound_Sensor_SKU_DFR0034)
- + <https://learningmicro.wordpress.com/serial-communication-interface-using-uart/>