ROMÂNIA
MINISTERUL APĂRĂRII NAȚIONALE
ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”

Facultatea de Sisteme Informatice și Securitate Cibernetică
Departamentul de Calculatoare și Securitate Cibernetică

UTILIZARE SENZOR DIGITAL BUZZER
PLATFORMA DE DEZVOLTARE FRDM-KL25Z

Std. plt. maj. Andrei RAICU
Std. sg. maj. Laurentiu BURGUI
Std. sg. maj. Alexandru GALEA
Grupa C114D

București

2023

Cuprins

I. Scopul proiectului

Scopul acestui proiect este de de acționa buzzerul pe perioade stabilite prin intermediul senzorului de rotație. Butonul de push va fi cel care va activă buzzer-ul. Vom realiza un grafic pentru valorile senzorului de rotație și un buton grafic pentru acționarea buzzer-ului.

Nr. crt	Senzor	Tip
1	Buzzer	Digital
2	Rotation	Analogic
3	Push	Digital

Figura 1 - Senzori necesari

II. Prezentarea senzorilor din proiect

Senzorul DFR0032 este un senzor digital ce funcționeaza atât cu impulsuri înalte cât și cu impulsuri joase, variază în funcție de ce sunet dorim să se audă. Poate fi integrat în diferite aplicații practice, fiind util datorită semnalului sonor pe care îl oferă.

Figura 2 - Senzorul DFR0032

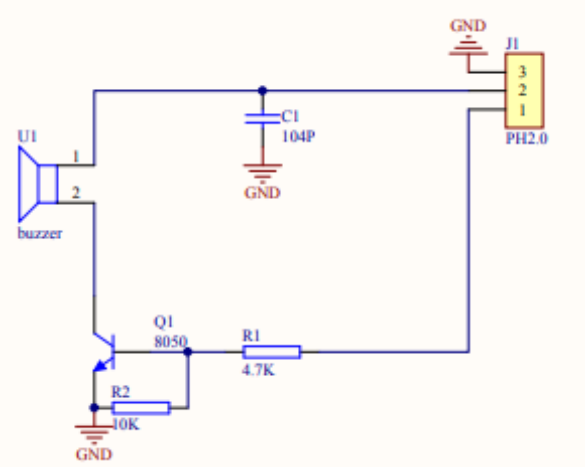

Figura 3 - Circuitul senzorului DFR0032

Acesta este format dintr-un condensator de 104 pF și două rezistențe, una de 10KΩ ,respectiv 4.7kΩ , conform schemei circuitului. Se conectează la placa de dezvoltare prin trei conectori: unul negru (GND), unul roșu (+5V) și unul verde (OUTPUT).

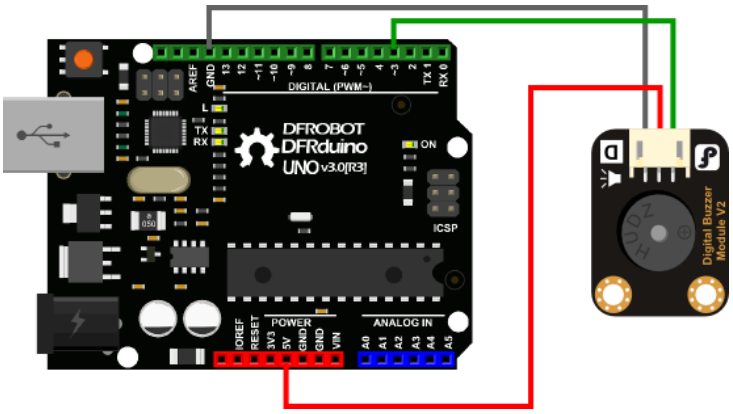

Figura 4- Exemplu de conectare senzor

Senzorul de rotație este folosit pentru a modifica tensiunea care va fi aplicată circuitului, Este un senzor analogic, de aceea, vom avea nevoie de modulul de conversie analog-digital (ADC), pentru a putea fi utilizat corespunzător.

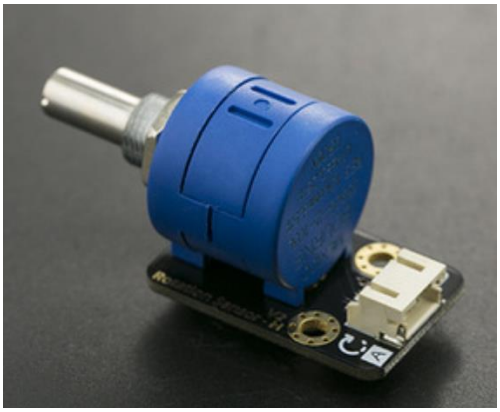

Figura 5 - Senzorul de rotație

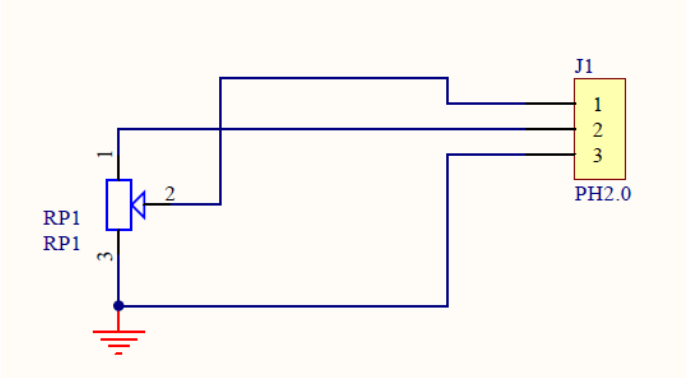

Figura 6 - Circuitul senzorului de rotație

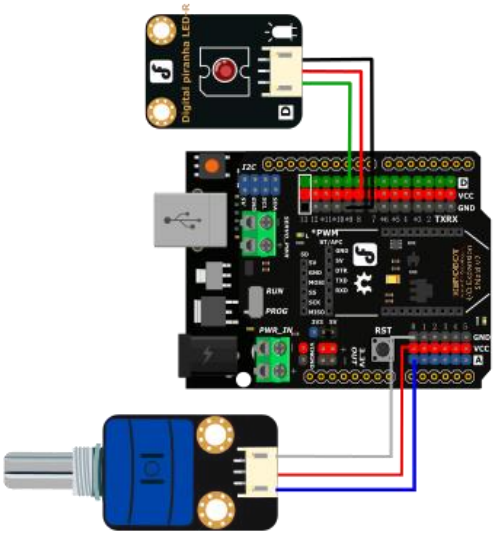

Figura 7- Exemplu de conectare senzor

Push-button-ul reprezintă un simplu buton, care la apăsarea lui, va acționa o întrerupere pentru a trimite un semnal senzorului DFR0032, să scoată sunete de diferite frecvențe, în funcție de tensiunea interpretată de la senzorul de rotație.

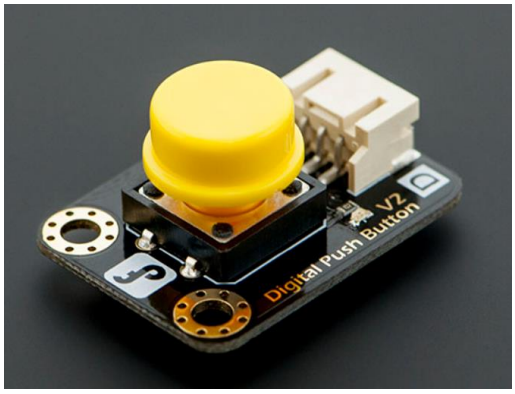

Figura 8 - Push-button-ul

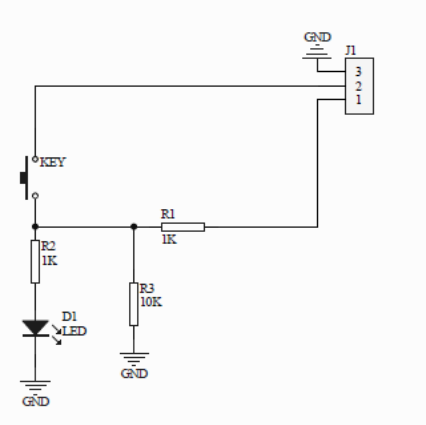

Figura 9 - Circuitul butonului

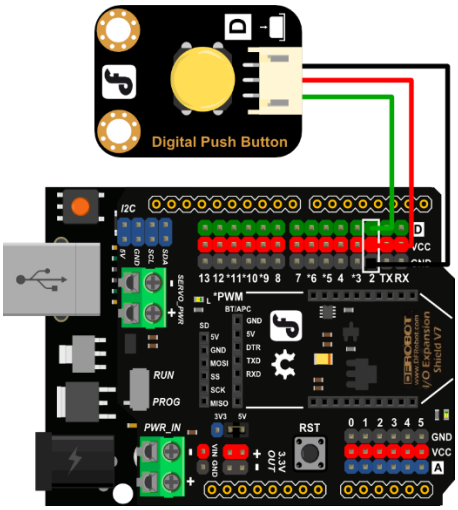

Figura 10 - Exemplu de conectare

III. Conectarea senzorilor la placa de dezvoltare

Vom conecta senzorul astfel:

- Buzzerul e conectat la PTD4
- Potențiometrul la PTB0
- Butonul de push la PTA12

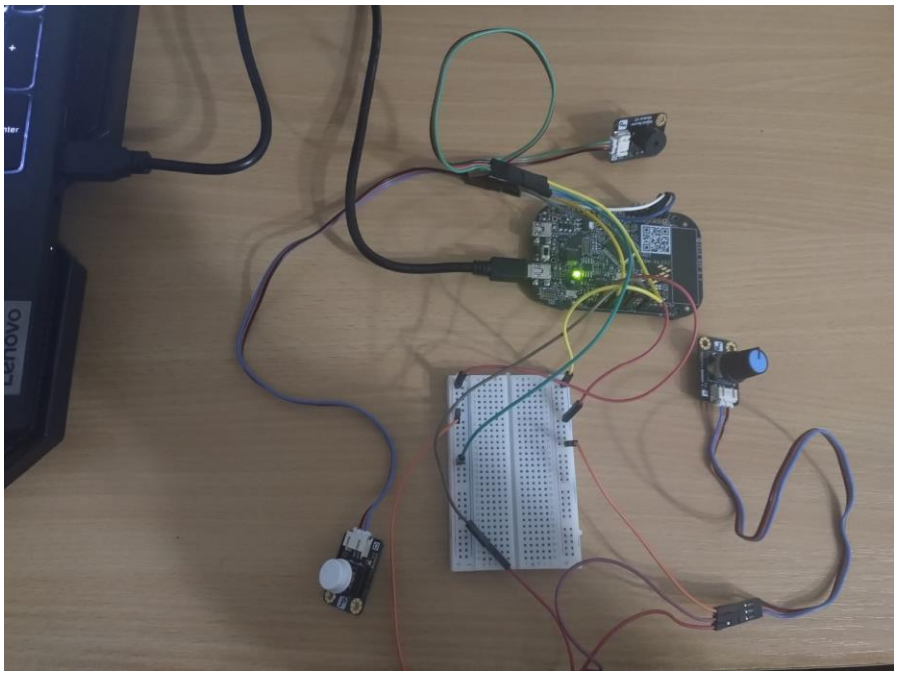

Figura 11 - Conectare senzori

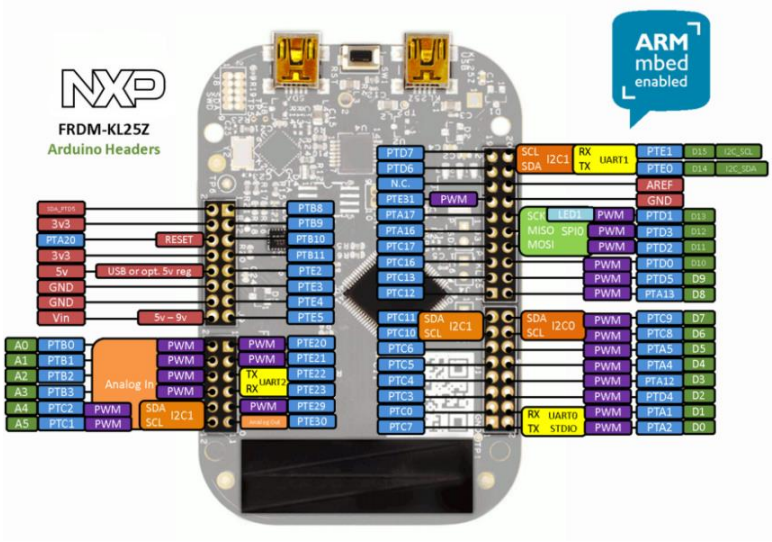

Figura 12 - Pini placa dezvoltare.

În figura de mai jos, se poate observa schema din Tinkercad a proiectului, cu mențiunea că în cadrul mediului de proiectare Tinkercad, nu am găsit plăcuța NXP, așa că am utilizat o plăcuță Arduino UNO, și am încercat să urmăm configurația reală a proiectului, deși am folosit modulul Arduino:

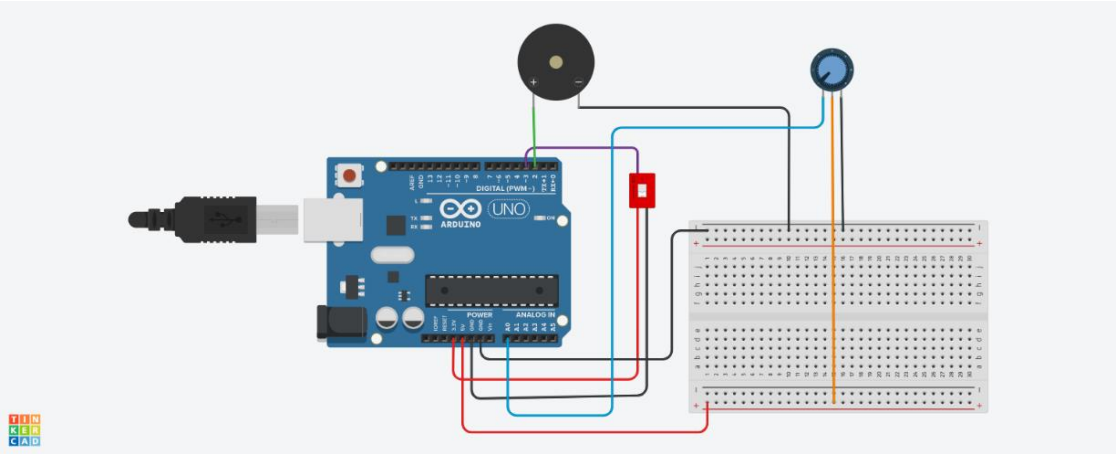

Figura 13 – Schema Tinkercad a proiectului

IV. Descriere program

a) Funcția main

În fișierul main.c am inclus fișierele header în care sunt declarate funcțiile și variabilele ce urmează să fie utilizate: fișierul header generat de către mediul de dezvoltare Keil, *MKL25Z4.h*, specific plăcii de dezvoltare, Buzzer.h (funcțiile de *init_buzzer*, *init_sw*, *play_buzzer*), Adc.h (funcțiile de *adcInit*, *adcSelect*, *adcRead*, *printVoltage*), Uart.h (funcția *UART0_Init*).

Adc.c | Adc.h | Buzzer.c | Buzzer.h | main.c | Uart.c | Uart.h

```
1  #include "MKL25Z4.h"
2  #include "Buzzer.h"
3  #include "Uart.h"
4  #include "Adc.h"
5  extern int val;
6
7  int main(void){
8
9    UART0_Init(115200);
10   adcInit(0);
11   init_sw();
12   init_buzzer();
13
14
15   while(1){ // infinate loop
16     adcRead(adcSelect());
17     printVoltage(ADC0->R[0]);
18
19     if(val == 1)
20       play_buzzer();
21
22   }
23 }
```

Figura 14 – Funcția main

Logica principală a programului este următoarea:

- întâi, se realizează inițializarile: inițializarea modulul de Uart0, setarea flagului corespunzător portului A (pentru componentă analogică de switch), setarea detaliilor despre întreruperea aferentă (prioritate 2, înlăturare flag pentru IRQn și start la întrerupere), respectiv activare port D (pentru buzzer);
- apoi, într-o buclă infinită se apelează funcția de conversie din alog în digital și apoi se preia inputul, se prelucrează și se transmite la consola. Se verifică dacă variabilă externă *val* (pentru apelarea din interfață) este setată, caz în care se apelează funcția de *play_buzzer* în cadrul căreia considerăm preliminar mai multe intervale pentru tensiunea transmisă de către potențiometru în funcție de care modificăm durata de timp în care buzzerul va funcționa. Apoi pornim buzzer-ul în aceste condiții cu un delay exprimat în funcție de frecvența stabilită.

b) Inițializarea modulelor

i. Modulul UART:

```
void UART0_Init(uint32_t baud_rate)
{

  //Setarea sursei de ceas pentru modulul UART
  SIM->SOPT2 |= SIM_SOPT2_UART0SRC(01);

  //Activarea semnalului de ceas pentru modulul UART
  SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;

  //Activarea semnalului de ceas pentru portul A
  //intrucat dorim sa folosim pinii PTA1, respectiv PTA2 pentru comunicarea UART
  SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;

  //Fiecare pin pune la dispozitie mai multe functionalitati
  //la care avem acces prin intermediul multiplexarii
  PORTA->PCR[1] = ~PORT_PCR_MUX_MASK;
  PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Configurare RX pentru UART0
  PORTA->PCR[2] = ~PORT_PCR_MUX_MASK;
  PORTA->PCR[2] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Configurare TX pentru UART0



  UART0->C2 &= ~((UART0_C2_RE_MASK) | (UART0_C2_TE_MASK));

  //Configurare Baud Rate
  uint32_t osr = 15; // Over-Sampling Rate (numarul de esantioane luate per bit-time)

  //SBR - vom retine valoarea baud rate-ului calculat pe baza frecventei ceasului de sistem
  //   SBR  -   b16 b15 b14 [b13 b12 b11 b10 b09    b08 b07 b06 b05 b04 b03 b02 b01] &
  // 0x1F00 -   0   0   0    1   1   1   1   1      0   0   0   0   0   0   0   0
  //            0   0   0   b13 b12 b11 b10 b09    0   0   0   0   0   0   0   0 >> 8
  //   BDH  -   0   0   0   b13 b12 b11 b10 b09
  //   BDL  -   b08 b07 b06  b05 b04 b03 b02 b01
  uint16_t sbr = (uint16_t)((DEFAULT_SYSTEM_CLOCK)/(baud_rate * (osr+1)));
  uint8_t temp = UART0->BDH & ~(UART0_BDH_SBR(0x1F));
  UART0->BDH = temp | UART0_BDH_SBR(((sbr & 0x1F00)>> 8));
  UART0->BDL = (uint8_t)(sbr & UART_BDL_SBR_MASK);
  UART0->C4 |= UART0_C4_OSR(osr);


  //Setare numarul de biti de date la 8 si fara bit de paritate
  UART0->C1 = 0;

  //Dezactivare intreruperi la transmisie
  UART0->C2 |= UART0_C2_TIE(0);
  UART0->C2 |= UART0_C2_TCIE(0);

  //Activare intreruperi la receptie
  UART0->C2 |= UART0_C2_RIE(1);

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

  NVIC_EnableIRQ(UART0_IRQn);

}
```

Figura 15 – Funcția UART0_Init

Funcția de UART0_Init, care conține rutine de activare și setare a semnalului de ceas, și de setare a pinilor PTA pentru comunicare. De asemenea, conține și rutine de dezactivare de întreruperi la transmisie și de activarea acestora la recepție. Tocmai de aceea, modulul UART se ocupă de modificarea variabilei val, care va asigura funcționarea buzzerului prin intermediul interfeței grafice.

Vom seta valoarea ‘01’ în registrul SIM_SOPT2, pentru a selecta modulul MCGFLLCLK ca și ceas.

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

În registrul SIM_SCG4, vom activa semnalul de ceas pentru UART:

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

În registrul SIM_SCG4 activăm semnalul de ceas pentru portul A, deoarece se vor folosi porturile acestuia PTA1 și PTA2:

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

În registrul PORTx_PCRn, setăm biții MUX pe 0, și apoi îi configurăm pentru porturile PTA alese, adică ALT2 pentru PTA1 și UART0_RX pentru PTA2:

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

În ultimă instanță, vom dezactiva registrul UART0_C2 pentru a seta baud rate-ul folosit și apoi îl vom reactiva la final:

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.

```
void UART0_IRQHandler() {
  if(UART0->S1 & UART0_S1_RDRF_MASK) {
    val = UART0->D;
    if(val != 0) {
      val = 0;
    }
    else
      val = 1;
  }
}
```

Figura 16 – Tratarea întreruperii

Prin funcțiile de transmitere și primire, se verifică registrele de recepție și de transmisie, și fie se așează în cadrul UART0->D valoarea ce se va transmite, fie se va returna, în cazul funcției de recepție.

```
void UART0_Transmit(uint8_t data)
{
  //Punem in asteptare pana cand registrul de transmisie a datelor nu este gol
  while(!(UART0->S1 & UART0_S1_TDRE_MASK));
        UART0->D = data;

}
```

Figura 17 – Funcția UART0_Transmit

```
uint8_t UART0_Receive(void)
{
  //Punem in asteptare pana cand registrul de receptie nu este plin

  while(!(UART0->S1 & UART0_S1_RDRF_MASK));
    return UART0->D;

}
```

Figura 18 – Funcția UART0_Receive

ii. Modulul Buzzer:

Funcția de inițializare a buzzerului, din imaginea de mai jos, arată faptul că acesta este folosit ca și un modul GPIO, și este utilizat pe portul D.

```
void init_buzzer(void){

    SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK; // activez portul D
    PORTD->PCR[buzzer] &= ~PORT_PCR_MUX_MASK;
    PORTD->PCR[buzzer] |= PORT_PCR_MUX(1); // GPIO
    PTD->PDDR |= MASK(buzzer); // buzzer ca output

}
```

Figura 19 – Funcția init_buzzer

Vom active semnalul de ceas pentru a folosi pinul 4 al portului D:

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

Funcția care va ajuta buzzerul să funcționeze este funcția play_buzzer, de mai jos, care setează o frecvență standard, de valoare 5000 ms, și în funcție de tensiunea dată de potențiometru, acestei frecvențe i se va adăuga un delay, după cum urmează: între 0 și 200 de mV, durata va fi de 6000 ms, între 200 și 400 mV, durata va fi 5000 ms, între 400 și 600 mV, durata va fi 4000 ms, între 600 și 800 mV, durata va fi 3000 ms, între 800 și 1000 mV, durata va fi de 2000 ms, și peste 1000 mV, se va întârzia cu 1000 ms. De asemenea, variabila val va fi resetată, pentru ca la următoarea apăsare a butonului din interfață, buzzer-ul să poată fi utilizat din nou.

```
void play_buzzer(void){

  uint32_t frequency = 5000;
  int duration = 0; // valoare ce va contoriza cat timp buzzerul va functiona
  if(ADC0->R[0] > 0 && ADC0->R[0] < 200) { // in functie de tensiunea transmisa de catre potentiometru, se modifica durata
    duration = 6000;
  }
  else if(ADC0->R[0] > 200 && ADC0->R[0] < 400) {
    duration = 5000;
  }
  else if(ADC0->R[0] > 400 && ADC0->R[0] < 600) {
    duration = 4000;
  }
  else if(ADC0->R[0] > 600 && ADC0->R[0] < 800) {
    duration = 3000;
  }
  else if(ADC0->R[0] > 800 && ADC0->R[0] < 1000){
    duration = 2000;
  }
  else {
    duration = 1000;
  }
  int cnt = 0;
  while(cnt != duration){
    PTD->PTOR = MASK(buzzer); // buzzerul va incepe sa bazaie
    delay(frequency); // delay-ul stabilit mai sus
    cnt++;
  }
  val = 0;
  return;
}
```

Figura 20 – Funcția play_buzzer

iii. Modulul ADC:

Funcția de inițializare a modului ADC, va activa semnalul de ceas pentru modulul periferic, va selecta modul de conversie pe 16 biți și divide ratio pe ½ .

```
void adcInit(int power){

    SIM->SCGC6 |= SIM_SCGC6_ADC0_MASK; // activez CLK ADC0

    if(power==1)
        ADC0->CFG1 = ADC_CFG1_ADLPC_MASK;

    ADC0->CFG1 |= ADC_CFG1_ADLSMP_MASK;
    ADC0->CFG2 = ADC_CFG2_ADLSTS(0x12 & 3);


    ADC0->CFG1 |=ADC_CFG1_ADIV(1)     // Clock Divide Select
               | ADC_CFG1_MODE(2)
               | ADC_CFG1_ADICLK(0);     // Input Clock: (Bus Clock=24MHz)

    ADC0->CFG2 = (uint32_t)((ADC0->CFG2 & (uint32_t)~(uint32_t)(
                ADC_CFG2_ADHSC_MASK |
                ADC_CFG2_ADLSTS(0x03)
               )) | (uint32_t)(
                ADC_CFG2_ADACKEN_MASK
               ));


    ADC0->SC3 = ADC_SC3_AVGE_MASK
               | ADC_SC3_AVGS(0x10 & 3);
}
```

Figura 21 – Funcția adcInit

Vom activa semnalului de ceas pe ADC0:

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

Vom selecta modul de conversie pentru ceasul intern al modulului ADC:

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

Vom selecta apoi configurația high-speed, outputul de clock asincron și selectarea pentru sample-ul de timp:

3 ADACKEN	Asynchronous Clock Output Enable Enables the asynchronous clock source and the clock source output regardless of the conversion and status of CFG1[ADICLK]. Based on MCU configuration, the asynchronous clock may be used by other modules. See chip configuration information. Setting this field allows the clock to be used even while the ADC is idle or operating from a different clock source. Also, latency of initiating a single or first-continuous conversion with the asynchronous clock selected is reduced because the ADACK clock is already operational. 0 Asynchronous clock output disabled; Asynchronous clock is enabled only if selected by ADICLK and a conversion is active. 1 Asynchronous clock and clock output is enabled regardless of the state of the ADC.
2 ADHSC	High-Speed Configuration

Table continues on the next page...

ADCx_CFG2 field descriptions (continued)

Field	Description
	Configures the ADC for very high-speed operation. The conversion sequence is altered with 2 ADCK cycles added to the conversion time to allow higher speed conversion clocks. 0 Normal conversion sequence selected. 1 High-speed conversion sequence selected with 2 additional ADCK cycles to total conversion time.

1–0 ADLSTS	Long Sample Time Select Selects between the extended sample times when long sample time is selected, that is, when CFG1[ADLSMP]=1. This allows higher impedance inputs to be accurately sampled or to maximize conversion speed for lower impedance inputs. Longer sample times can also be used to lower overall power consumption when continuous conversions are enabled if high conversion rates are not required. 00 Default longest sample time; 20 extra ADCK cycles; 24 ADCK cycles total. 01 12 extra ADCK cycles; 16 ADCK cycles total sample time. 10 6 extra ADCK cycles; 10 ADCK cycles total sample time. 11 2 extra ADCK cycles; 6 ADCK cycles total sample time.

Vom selecta apoi și registrul de status și control ADC0->SC3, și îl vom activa pentru hardware average și vom selecta 4 sample-uri:

2 AVGE	Hardware Average Enable Enables the hardware average function of the ADC. 0 Hardware average function disabled. 1 Hardware average function enabled.
1–0 AVGS	Hardware Average Select Determines how many ADC conversions will be averaged to create the ADC average result. 00 4 samples averaged. 01 8 samples averaged. 10 16 samples averaged. 11 32 samples averaged.

```
int adcSelect(){

    channel_pin=-1;
    SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK;// activez clock ul
    PORTB->PCR[PIN_Rotation] &= ~PORT_PCR_MUX_MASK;// analog input
    channel_pin=8;
    return channel_pin;
}
```

Figura 22 – Funcția adcSelect

```
void adcRead(int channel) {
    // conversia de la analog la digital
    if(channel>=0){
    ADC0->SC1[0] = ADC_SC1_ADCH(channel & ~(1 << 5));
    }
}
```

Figura 23 – Funcția adcRead

Funcțiile de adcSelect și adcRead, vor pregăti conversia de la analog la digital, și vor fi apelate în funcția main, în care valoarea citită va fi printată de funcția printVoltage.

```
void printVoltage(uint16_t value){
    uint8_t nr1=value /1000;
    uint8_t nr2=value %1000 /100;
    uint8_t nr3=value %100 /10;
    uint8_t nr4=value %10;
    UART0_Transmit(nr1+48);
    UART0_Transmit(nr2+48);
    UART0_Transmit(nr3+48);
    UART0_Transmit(nr4+48);
    UART0_Transmit(0x0D);
    UART0_Transmit(0x0A);

}
```

Figura 24 – Funcția printVoltage

c) Interfața grafică

Interfața grafică a fost realizată în PyQt vesiunea 5, care ne ajută să realizăm reprezentarea folosind limbajul python și sa o legam de fluxul principal de C.

- În primă instanță, am inclus bibliotecile necesare pentru banda serială, push button, desenarea graficului:

```
from PyQt5 import QtWidgets, QtCore
from PyQt5.QtWidgets import QPushButton
import pyqtgraph as pg
import sys
import serial
```

Figura 25 – Bibliotecile de python

- Am stabilit regulile pentru comunicația serială (Serial line-ul și baud rate-ul) astfel încât să corespundă realității aplicației, cu mențiunea că primul argument va fi diferit dacă introducem plăcuța în dispozitive diferite, deoarece vom avea porturi seriale diferite:

```
ser_com = serial.Serial('COM10', 115200)
```

Figura 26 – Setarea comunicației seriale

- Am construit fereastra principală ce va conține un grafic și un buton:

```
def __init__(self, *args, **kwargs):
    super(MainWindow, self).__init__(*args, **kwargs)
    self.setWindowTitle("Buzzer Window")

    self.graphWidget = pg.PlotWidget()
    self.setCentralWidget(self.graphWidget)
    self.graphWidget.setBackground((245, 245, 245))
    self.graphWidget.setTitle("\nData\n", color="black", size="25px")

    styles = {'color': 'black', 'font-size': '15px'}
    self.graphWidget.setLabel('left', 'Data', **styles)
    self.graphWidget.setLabel('bottom', 'Time', **styles)
    self.graphWidget.showGrid(x=True, y=True)
    self.graphWidget.setYRange(0, 2000, padding=0)
    self.graphWidget.setXRange(0, 800, padding=0)
    self.graphWidget.setAspectLocked(lock=True, ratio=1)


    self.x = list(range(100))
    self.y = list(range(100))

    pen = pg.mkPen(color=(0, 128, 128), width=5)
    self.data_line = self.graphWidget.plot(self.x, self.y, pen=pen)
```

Figura 27 – Inițializarea elementelor grafice

- Am setat un timer care la fiecare 10 ms va apela funcția update_plot_data:

```
self.timer = QtCore.QTimer()
self.timer.setInterval(10)
self.timer.timeout.connect(self.update_plot_data)
self.timer.start()
self.initUI()
```

Figura 28 – Inițializarea timerului

- Funcția care face update la date:

```
def update_plot_data(self):
    data = ser_com.readline().decode('utf-8')
    try:
        value = int(data)
        print(value)
        self.x = self.x[1:]
        self.x.append(self.x[-1] + 1)
        self.y = self.y[1:]
        self.y.append(value)
        self.data_line.setData(self.x, self.y)
    except:
        return
```

Figura 29 – Funcția de update

- Funcția care conține butonul și care apelează funcția signalBuzzer când este apăsat:

```
def initUI(self):

    self.setGeometry(100, 100, 1600, 900)
    self.buzzerButton = QPushButton(self)
    self.buzzerButton.setText("BUZZER")
    self.buzzerButton.clicked.connect(self.signalBuzzer)
    self.buzzerButton.move(0, 0)
```

Figura 30 – Inițializarea butonului

- Funcția care se apelează la apăsarea butonului din interfață, și care va transmite 1 pe magistrala serială, pentru a putea fi primit prin modulul UART și să declanșeze buzzerul:

```
def signalBuzzer(self):
    ser_com.write(1)
```

Figura 31 – Funcția ce se apelează la apăsarea butonului

- Funcția main, care va afișa fereastra principală și va executa interfața grafică:

```
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    SensorWindow = MainWindow()
    SensorWindow.show()
    sys.exit(app.exec_())
```

Figura 32 – Funcția main

V. Testarea aplicației:

Vom avea în vedere trei scenarii pentru testarea aplicației:

- Când potențiometrul este la minim:

  Putty:

Figura 33

Interfața:

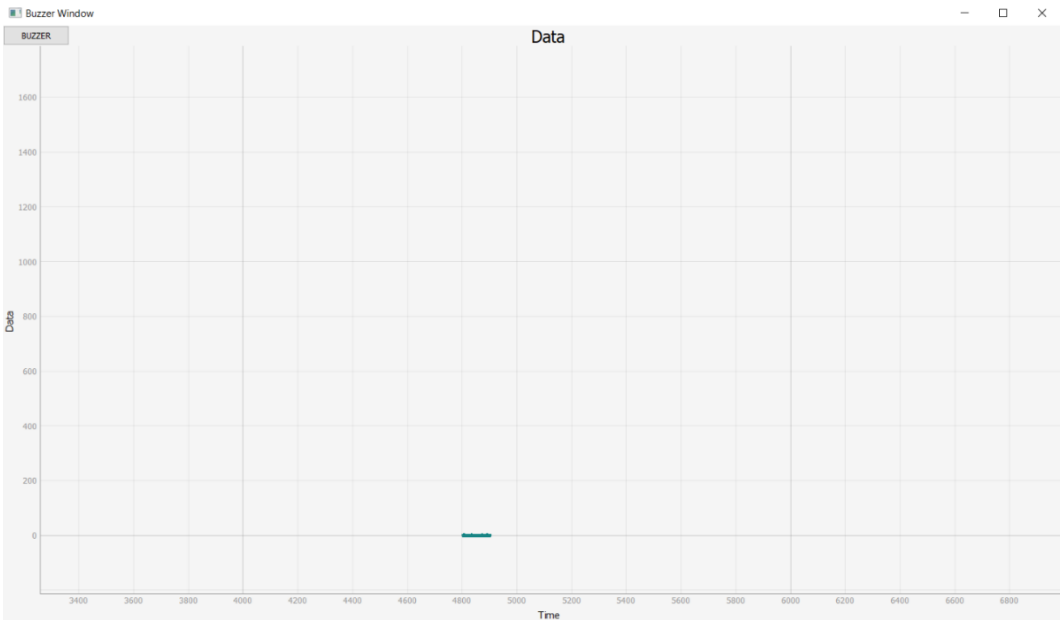

Figura 34

- Când potențiometrul este la jumătate:

  Putty:

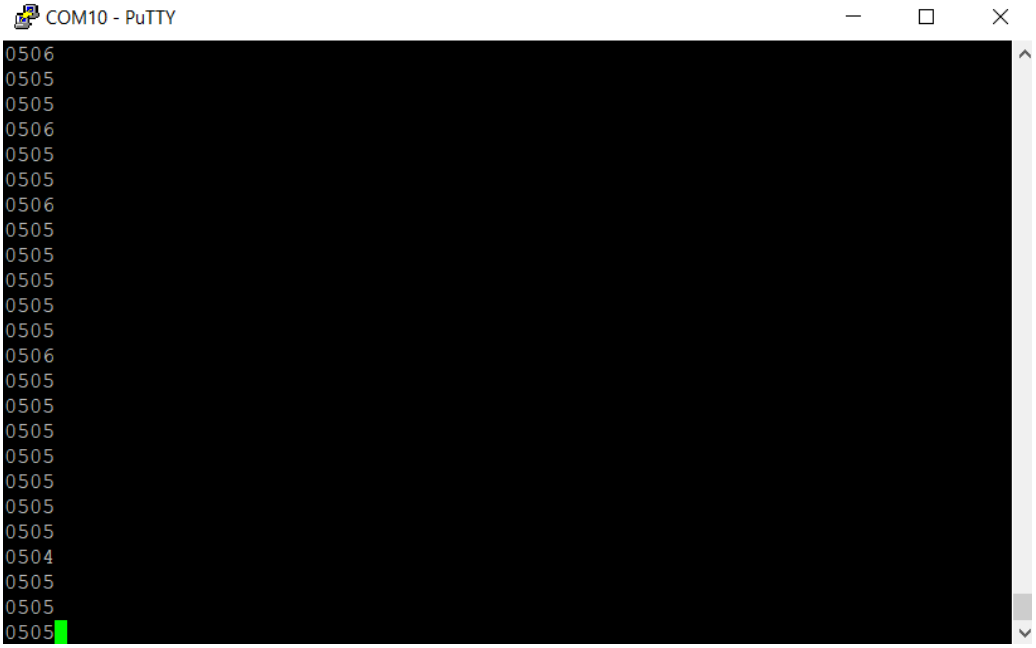

Figura 35

Interfața:

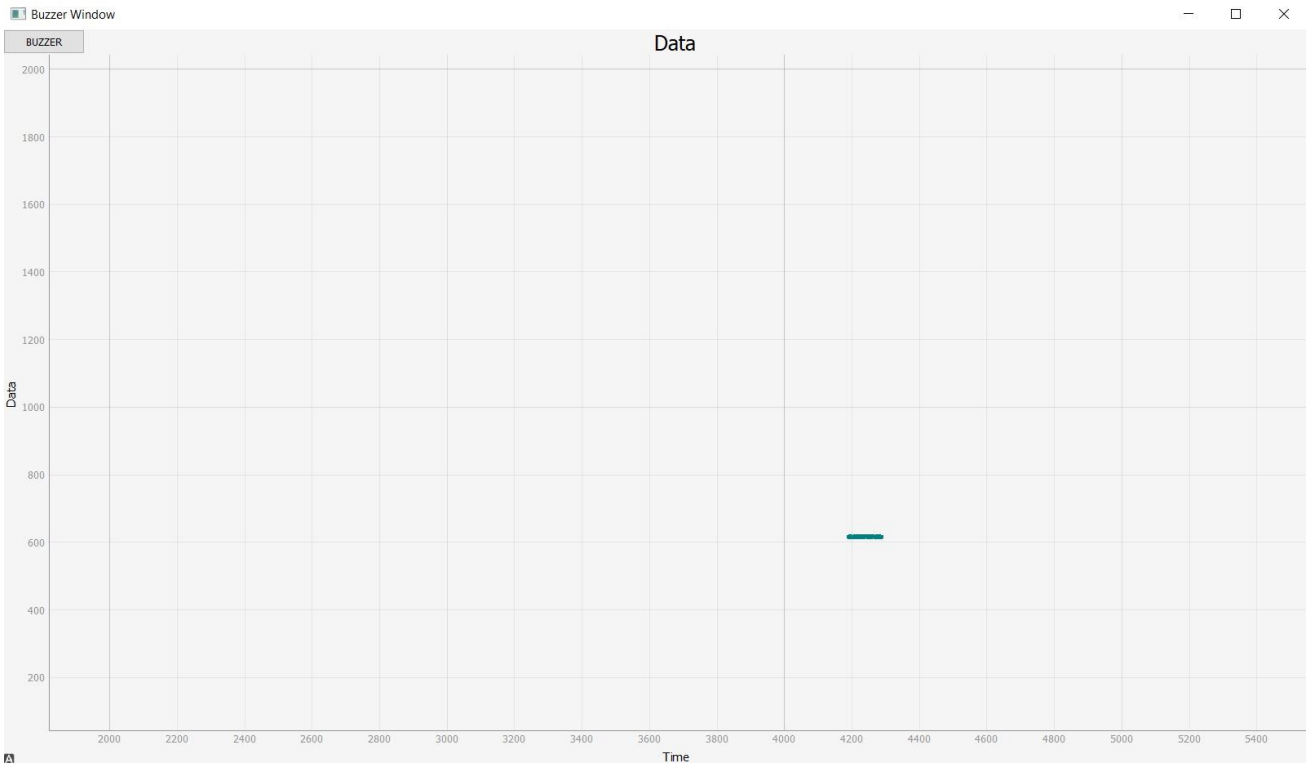

Figura 36

- Când potențiometrul este la maxim:

  Putty:

Figura 37

Interfața:

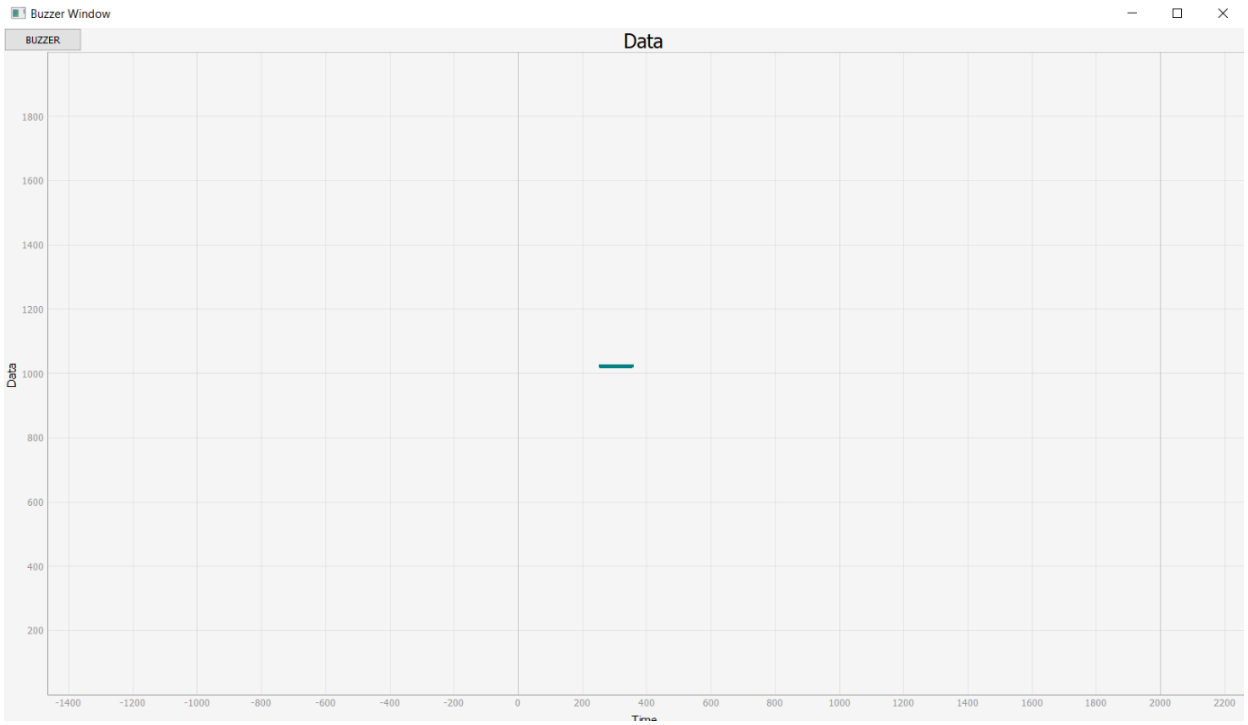

Figura 38

VI. Dificultăți întâmpinate

În timpul realizării acestui proiect, am întâmpinat anumite dificultăți, precum:

- Graficul nu a ieșit atât de responsive pe cât se dorea: apare un anumit delay pe grafic;
- Apare o lipsa de continuitate pe grafic; afișează doar momentul respectiv și imediata lui vecinatate.

VII. Bibliografie

https://wiki.dfrobot.com/Analog_Rotation_Sensor_V2__SKU__DFR0058_

https://wiki.dfrobot.com/DFRobot_Digital_Push_Button_SKU_DFR0029

https://wiki.dfrobot.com/Digital_Buzzer_Module__SKU__DFR0032_

https://wiki.mta.ro/c/4/ssmp/start

https://github.com/undacmic/MCULabs/blob/main/Resurse/FRDM-KL25Z_ReferenceManual.pdf