Acordor Digital

Introducere

Despre chitară se spune că este un instrument fascinant, dar pe cât de interesantă este, pe atât de ușor se dezacordează. Puțini sunt cei care pot reacorda “după ureche”, motiv pentru care mulți apelează la un acordor. Inspirată de această nevoie, am venit cu ideea acestui proiect după ce mi-am achiziționat o chitară în speranța de a învăța să cânt. În acest proces, am descoperit și am fost impresionată de modul în care funcționează un tuner.

Descriere generală

Principiul de funcționare al acordorului meu este următorul:

1. Utilizatorul selectează coarda pe care dorește să o acordeze.
2. Utilizatorul ciupește respectiva coardă.
3. Microfonul preia sunetul și îl convertește în semnal electric.
4. Acest semnal este amplificat, pentru a putea fi procesat în detaliu, astfel determinând nota muzicală.
5. Pe ecranul LCD-ului se afișează indicații pentru a se acorda chitara perfect în jurul notei respective.

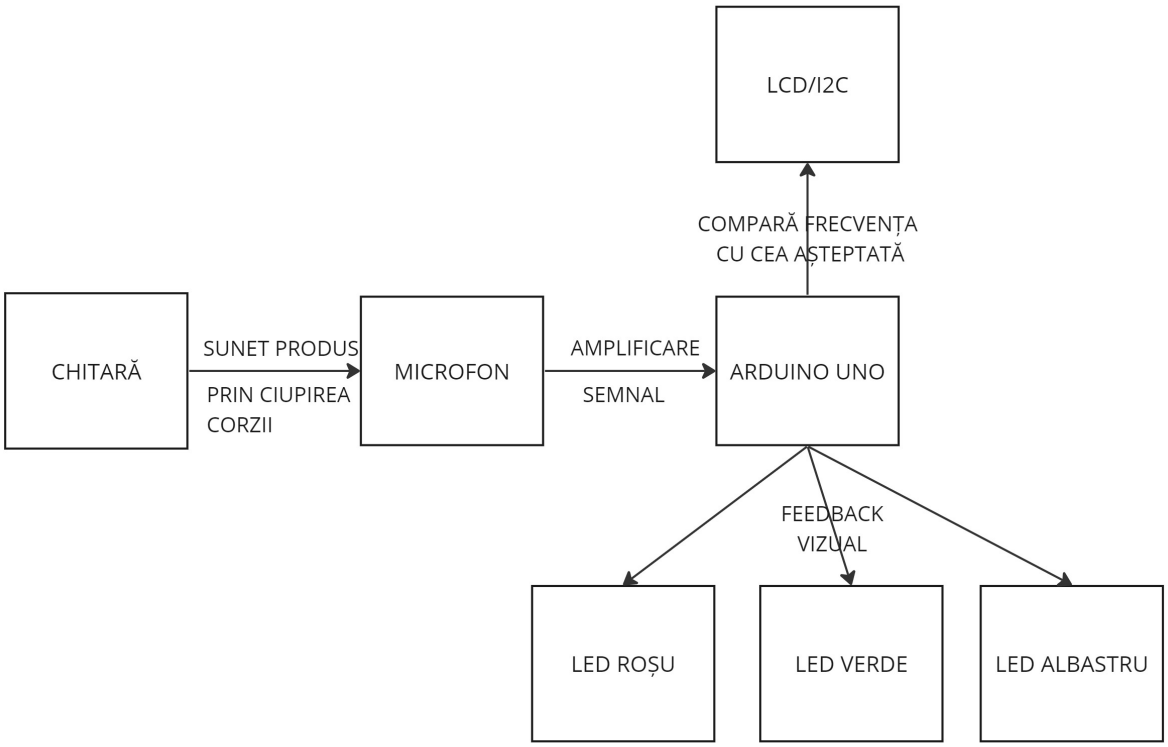
Când sunt acordate corect, corzile au următoarele frecvențe:

- 329.63 Hz (E)
- 246.94 Hz (B)
- 196.00 Hz (G)
- 146.83 Hz (D)
- 110.00 Hz (A)
- 82.41 Hz (E)

Pentru a oferi feedback vizual suplimentar, voi folosi leduri, pe care le voi aprinde astfel:

- **Roșu** - nota este joasă față de valoarea corectă;
- Primul led roșu din extremitate se aprinde când nota este cu **minim 20 Hz** mai joasă, al doilea red roșu când nota este cu **15-20 Hz** mai joasă, iar al treilea led când nota este cu **10-15** Hz mai joasă.
- **Verde** - coarda notei este bine acordată;
- Primul led verde (de lângă cele roșii) se aprinde când nota este cu minim **5-10 Hz** mai joasă, al doilea red roșu când nota este în **intervalul corect** (intervalul corect este considerat cel din jurul frecvenței dorite cu o **eroare de +-2.5 Hz**), iar al treilea când nota este cu **5-10 Hz** mai înaltă.
- **Albastru** - nota este înaltă față de valoarea corectă;
- Primul led albastru apropiat de cele verzi se aprinde când nota este cu minim **10-15 Hz** mai înaltă, al doilea când nota este cu **15-20 Hz** mai înaltă, iar ultimul când nota este cu **minim 20 Hz** mai înaltă.

Schema bloc:



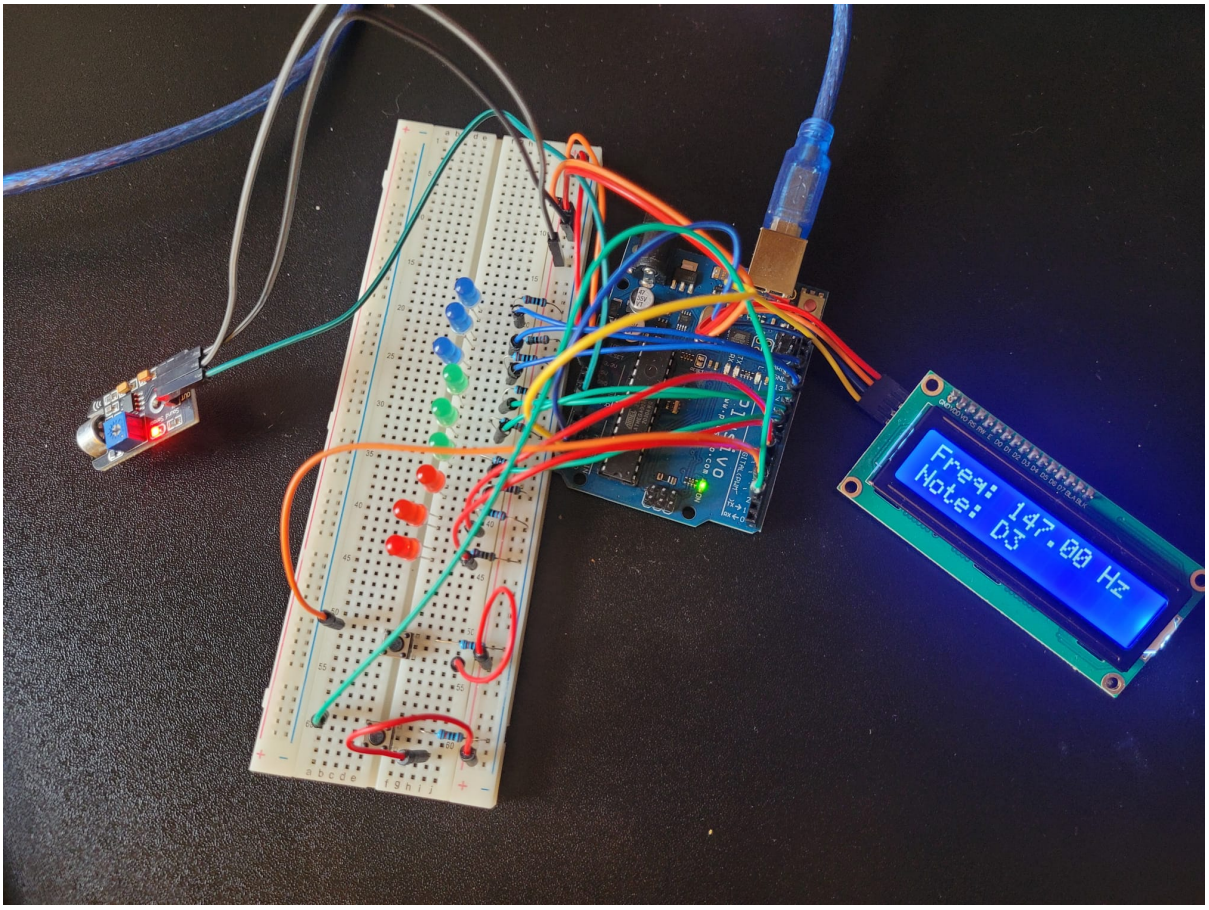
Hardware Design

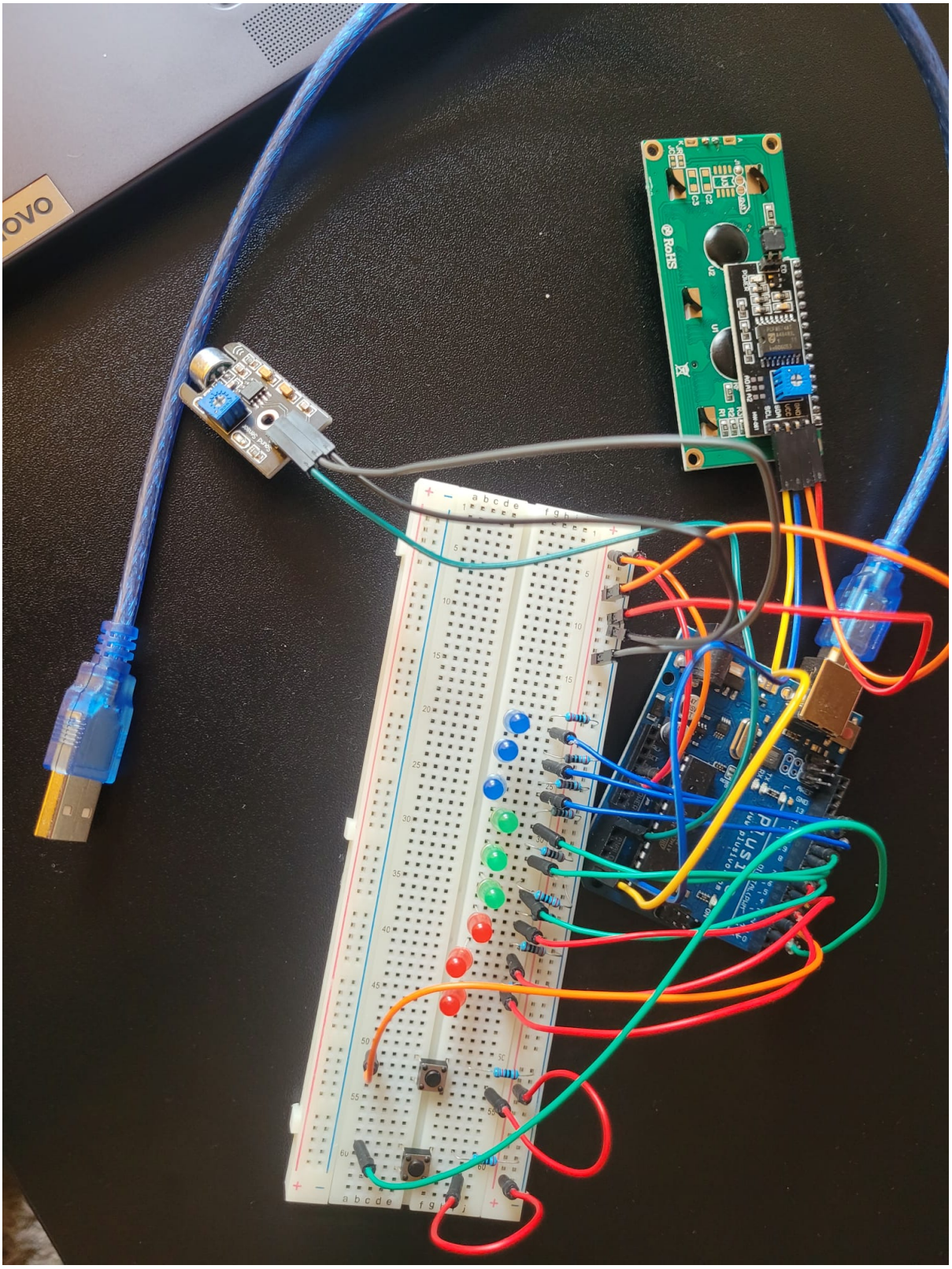
Lista pieselor:

- **Arduino Uno** - microcontroller-ul care va controla întreg sistemul;
- **Modulul I2C** - interfața de comunicare între Arduino si ecranul LCD;
- **Ecran LCD 1602** - pentru a oferi feedback vizual despre starea corzilor;
- **Breadboard** - pentru conectarea, precum și organizarea componentelor electronice;
- **Microfon + LM393** - pentru a prelua sunetul de la coarde, pentru a putea fi convertit + amplificator operațional;
- **LED-uri** - pentru a oferi feedback vizual despre starea corzilor;
- **Rezistori** - pentru a limita curentul prin LED-uri și pentru a proteja Arduino-ul;
- **Fire de legătură** - pentru realizarea conexiunilor între componente;
- **Butoane** - pentru selectarea notei dorite;

Simulare în ThinkerCad:

Hardware Design:





Software Design

Laboratoare folosite

Laboratorul 0: GPIO → folosit pentru aprinderea și stingerea ledurilor;

Laboratorul 2: Întreruperi → folosit pentru realizarea întreruperilor pe butoane;

Laboratorul 4: ADC → pentru citirea frecvenței transmise de microfon;

Laboratorul 6: I2C → pentru afișarea datelor pe ecranul LCD-ului;

Mediul de dezvoltare

Arduino IDE

Componente și Biblioteci folosite:

- **arduinoFFT.h** - Bibliotecă utilizată pentru a efectua FFT pe semnalele audio;
- **LiquidCrystal_I2C.h** - Bibliotecă utilizată pentru a interfața cu un ecran LCD I2C;
- **ADC** - Pentru a citi semnalul audio de la microfon;
- **Pini GPIO** - Utilizați pentru a citi intrările butoanelor și pentru a controla LED-urile;
- **Întreruperi și ISR** - Pentru a gestiona apăsările butoanelor pentru a comuta tunerul și a schimba frecvențele target;

```
#include "arduinoFFT.h"
#include "LiquidCrystal_I2C.h"

// Canalul ADC pentru citirea semnalelor audio de la microfon
#define CHANNEL A0
// SNUmarul de esantioane pentru FFT
const uint16_t samples = 128;
// Frecventa de esantinare
const double samplingFrequency = 2000;
// Perioada de esantionare in ms
unsigned int sampling_period_us;
// microsecunde
unsigned long microseconds;
// Threshold-ul pentru detectarea frecventei
unsigned int threshold = 200;
// Intervalul de deviatie acceptabil in jurul frecventei tinta
const float frequencyRange = 40.0;
// Dimensiunea pasului pentru luminarea LED-urilor
const float frequencyIncrement = 5.0;

// Partea reala pentru stocarea inputului FFT
double vReal[samples];
// Partea imaginara pentru stocarea inputului FFT
double vImag[samples];

// Interfata I2C pentru displayul LCD-ului
LiquidCrystal_I2C lcd(0x3F, 16, 2);
```

```
// Pinii GPIO pentru ON/OFF (Intrerupere)
const int onOffButtonPin = 2;
// Pinii GPIO pentru ciclu (Intrerupere)
const int cycleButtonPin = 3;

// Starea tunerului (pornit/oprit)
volatile bool isOn = false;
// Indexul pentru frecventa tinta curenta
volatile int targetFrequencyIndex = 0;
// Lista frecventelor tinta pentru fiecare nota in parte
int targetFrequencies[] = { 82, 110, 147, 196, 247, 330 };

// Obiectul FFT creat
ArduinoFFT<double> FFT = ArduinoFFT<double>(vReal, vImag, samples,
samplingFrequency);
```

Mai jos voi explica fiecare funcție în parte, precum și funcționalitățile acestora:

Funcția setup()

- Această funcție inițializează diverse component și configurări ale tunerului digital la pornirea programului.
- Prima dată calculează **perioada de eșantionare** în microsecunde pe baza frecvenței de eșantionare, pentru a ne asigura că **citirile ADC** se fac la intervale regulate, corespunzătoare frecvențelor de eșantionare dorite.
- Inițializez afișajul **LCD I2C**, activez iluminarea de fundal și curat ecranul.
- Configurez **pinii GPIO** pentru cele **2 butoane** (de on/off și ciclare printre notele disponibile) cu rezistențe de pull-up interne.
- Atașez **întreruperile** la cele 2 butoane pentru a detecta apăsările și pentru a executa cele 2 funcții ISR implementate de mine (onOffButtonISR și cycleButtonISR) pentru apariția **fronturilor crescătoare** (RISING - la ridicarea degetului).

```
void setup() {
  // Perioada de esantionare in microsecunde
  sampling_period_us = round(1000000 * (1.0 / samplingFrequency));

  // Comunicarea seriala penrtu debugging
  Serial.begin(115200);
  while (!Serial);
  Serial.println("Ready");

  lcd.begin(16, 2);
  lcd.init();
  lcd.backlight();
  lcd.clear();

  // Configurarea pinilor GPIO pentru butoane
```

```
  pinMode(onOffButtonPin, INPUT_PULLUP);
  pinMode(cycleButtonPin, INPUT_PULLUP);

  // Intreruperi pentru cele 2 butoane pe front crescator
  attachInterrupt(digitalPinToInterrupt(onOffButtonPin), onOffButtonISR,
RISING);
  attachInterrupt(digitalPinToInterrupt(cycleButtonPin), cycleButtonISR,
RISING);
}
```

Funcția loop():

- Această funcție rulează continuu și este împărțită în **2 cazuri** - tunerul este pornit sau oprit.
- Dacă tunerul este **oprit** se apelază funcția **clearLEDs()** care stinge toate ledurile conectate, curață ecranul LCD-ului, setează cursorul pe prima linie, poziția 0 și se afișează mesajul “Waiting OFF” pentru a indica faptul că tunerul este oprit și se introduce o întârziere de 50 milisecunde.
- Dacă tunerul este **pornit**, prin apelarea funcției **sample()** citim semnalele audio de la microfon pe care efectuăm FFT și procesăm astfel frecvențele detecate și se introduce o întârziere de 50 milisecunde.

```
void loop() {
  if (isOn) {
    // Tuner pornit
    // Citeste semnalele audio
    sample();
    delay(50);
  } else {
    // Stinge toate ledurile conectate
    clearLEDs();
    lcd.clear();
    lcd.setCursor(0, 0);
    // Afiseaza mesajul Waiting OFF pentru a semnala ca tunerul e pornit
    lcd.print("Waiting OFF");
    delay(50);
  }
}
```

Funcția clearLEDs():

- Iterează prin **toți pinii** de la 5 la 13 și îl setează pe fiecare la LOW, ceea ce îi va stinge.

```
void clearLEDs() {
  // Itereaza prin toti pinii de la 5 la 13
  for (int pin = 5; pin <= 13; pin++) {
    digitalWrite(pin, LOW);
  }
```

```
}
```

Funcția sample():

- Această funcție preia datele de la microfon, efectuează **transformarea Fourier** pentru a analiza frecvențele semnalului, **elimină zgomotul** și actualizează ledurile și LCD-ul în funcție de rezultatul obținut.
- Prima dată, pentru sincronizarea ADC-ului, salvez **timpul curent** în microsecunde.
- Iterez prin numărul de eșantioane specificate la început și stochez valoarea de la microfon în **parte reală** și **parte imaginară**.
- Aștept până când perioada de eșantionare este completă, apoi actualizez timpul pentru următoarea citire.
- Aplic o **fereastră Hamming** pentru a minimiza efectele de discontinuitate și efectuez transformarea Fourier, ale cărei valori le convertesc în magnitudini.
- Apoi filtrez frecvențele pentru a elimina zgomotul și elimin valorile ce sunt **sub pragul stabilit** de mine sau în afara intervalului de frevențe țintă.
- Determin **frecvența principală** din datele FFT și mă asigur să fie într-un interval valid. Daca e, actualizez LED-urile folosindu-mă de funcția **mapFrequencyToLEDs(x)**.

```
void sample() {
  // Initializeaza timpul de esantionare
  microseconds = micros();

  // Citirea esantioanelor audio
  for (int i = 0; i < samples; i++) {
    // Cititrrea valorii ADC la microfon
    vReal[i] = analogRead(CHANNEL);
    vImag[i] = 0;
    // Pana cand perioada de esantionare e completa
    while (micros() - microseconds < sampling_period_us) {}
    microseconds += sampling_period_us;
  }

  // Efectuarea FFT
  FFT.windowing(FFTWindow::Hamming, FFTDirection::Forward);
  FFT.compute(FFTDirection::Forward);

  // Convertirea valorilor complexe la magnitudine
  FFT.complexToMagnitude();

  // Filtrarea sunetului (frecvente neimportante)
  for (uint16_t i = 0; i < (samples >> 1); i++) {
    if (vReal[i] < threshold ||
    (vReal[i] > targetFrequencies[targetFrequencyIndex] - frequencyRange &&
vReal[i] < targetFrequencies[targetFrequencyIndex] + frequencyRange)) {
      vReal[i] = 0.0;
    }
  }
```

```
  // Frecventa principala
  double x = FFT.majorPeak();
  if (x > 20 && x < 980) {
    Serial.println(x, 6);
    // Actualizarea LED-urilor in functie de frecevnta
    mapFrequencyToLEDs(x);
    // Actualizarea LCD-ului dupa frecventa
    displayFrequency(targetFrequencies[targetFrequencyIndex]);
  } else {
    Serial.println("Waiting");
  }
}
```

Funcția mapFrequencyToLEDs():

- Această funcție **mapează frecvența** detectată la LED-urile corespunzătoare, fiind responsabilă pentru aprinderea lor.
- Primul pas este acela de a opri toate LED-urile.
- Calculez **deviația** față de frecvența țintă, ajustată cu intervalul de frecvență.
- În funcție de deviație, calculez indexul LED-ului și determin apoi **pinul GPIO** care controlează LED-ul corespunzător, pe care îl aprind.

```
void mapFrequencyToLEDs(double dominantFrequency) {
  // Opreste toate ledurile
  clearLEDs();

  // Deviatia frecventei in functie de target
  float deviation = dominantFrequency + frequencyRange - targetFrequencies[
targetFrequencyIndex];

  // Index led ce trebuie aprins
  int ledIndex = round(deviation / frequencyIncrement) - 4;

  Serial.println("Deviation: ");
  Serial.print(deviation);
  Serial.print(", ledIndex: ");
  Serial.print(ledIndex);

  ledIndex = constrain(ledIndex, 0, 8);
  Serial.print(", constrained ledIndex: ");
  Serial.print(ledIndex);
  Serial.print("\n\n");

  // Determinarea pinului GPIO corespunzator LED-ului
  int pin = ledIndex + 5;

  // Aprinderea ledului corespunzator
```

```
  digitalWrite(pin, HIGH);
}
```

Funcția displayFrequency():

- Afișează **frecvența țintă** pe ecranul LCD-ului împreună cu **nota muzicală**.

```
void displayFrequency(double frequency) {
  // Sterge afisajul de pe LCD
  lcd.clear();

  // Seteaza cursorul la pozitia (0, 0)
  lcd.setCursor(0, 0);
  lcd.print("Freq: ");
  // Frecventa cu 2 zecimale
  lcd.print(frequency, 2);
  lcd.print(" Hz");
  lcd.setCursor(0, 1);
  lcd.print("Note: ");

  // Nota muzicala corespunzatoare frecventei
  switch (static_cast<int>(frequency)) {
    case 82:
      lcd.print("E2");
      break;
    case 110:
      lcd.print("A2");
      break;
    case 147:
      lcd.print("D3");
      break;
    case 196:
      lcd.print("G3");
      break;
    case 247:
      lcd.print("B3");
      break;
    case 330:
      lcd.print("E4");
      break;
    default:
      lcd.print("Unknown");
      break;
  }
}
```

Funcția onOffButtonISR():

- Funția are ca scop tratarea evenimentului de **ON/OFF** declanșată de apăsarea butonului de către utilizator.
- Verific dacă a trecut suficient timp de la **ultima întrerupere** și dacă da, inversez starea variabilei isOn ce are ca scop controlarea funcționalității tunerului.

```
void onOffButtonISR() {
  static unsigned long lastInterruptTimeOnOff = 0;
  unsigned long interruptTime = millis();

  // Daca a trecut suficient timp de la ultima intrerupere
  if (interruptTime - lastInterruptTimeOnOff > 400) {
    // Inverseaza starea tunerului
    isOn = !isOn;
  }

  // Actualizeaza timpul ultimei intreruperi
  lastInterruptTimeOnOff = interruptTime;
}
```

Funcția cycleButtonISR():

- Este foarte asemănătoare cu funcția **onOffButtonISR()**, doar că gestionează întreruperile generate de butonul de ciclare.
- În loc să inverseze statusul tunerului, **ciclează** prin frecvențele target.

```
void cycleButtonISR() {
  static unsigned long lastInterruptTimeCycle = 0;
  unsigned long interruptTime = millis();

  // Daca a trecut suficient timp de la ultima intrerupere
  if (interruptTime - lastInterruptTimeCycle > 400) {
    // Ciclu prin frecventele tinta
    targetFrequencyIndex = (targetFrequencyIndex + 1) % 6;
  }

  // Actualizeaza timpul ultimei intreruperi
  lastInterruptTimeCycle = interruptTime;
}
```

Rezultate Obținute

În urma proiectului, am obținut rezultate bune în urma testării cu un generator de frecvențe de note muzicale (am generat unde sinusoidale corespunzătoare fiecărei note muzicale).

Mai jos, am realizat câteva videoclipuri pentru a demonstra funcționalitatea completă și corectă a acorodrului:

Trecerea prin notele muzicale - implementare cu întreruperi - Prima dată tunerul e în starea de OFF, iar primul buton apăsat îl trece în starea de ON. Cât timp este identificată o frecvență validă care nu este zgomot, utilizatorul poate trece prin notele disponibile prin apăsarea butonului de cilare.

330 Hz - E4

247 Hz - B3

196 Hz - G3

147 Hz - D3

110 Hz - A2

82 Hz - E2

Acorodul meu a identificat pentru fiecare caz în parte momentul când nota este corectă, iar indicațiile oferite de LED-uri au fost corecte și consistente.

Deși acorodul funcționează bine în general, am observat că pentru amplitudini foarte mici, sistemul are unele dificultăți în identificarea corectă a frecvenței.

Concluzii

Acordorul digital s-a dovedit a fi un succes, deomonstrând o funcționalitate corectă și completă. Utlizarea unui generator de frecvențe de note muzicale a confirmat capacitatea acordorului de a recunoaște corect frecvențele și de a oferi indicații precise pentru acordaj, ceea ce înseamnă că am realizat corect eliminarea zgomotului, precum și FFT-ul aplicat semnalului audio.

Deși există anumite limitări pentru amplitudini foarte joase, acest acordor poate fi utlizat cu încredere pentru acordarea diferitelor instrumente muzicale.

Mă bucur că am avut ocazia să realizez acest proect, deoarece am reușit să îmbin pasiunea mea pentru tehnologie cu cea pentru muzică. Procesul de proiectare și dezvoltare m-a făcut să înțeleg mai bine cum pot utiliza diferite componente electronice pentru a crea în final un produs util și eficient.

Download

Cod complet acordor

Jurnal

- **28.04.2024** → Realizarea paginii proiectului;
- **29.04.2024** → Realizarea introducerii și descrierea generală a proiectului;
- **03.04.2024** → Achiziționarea componentelor;
- **11-12.05.2024** → Realizarea designului hardware;
- **19-25.05.2024** → Software Design;
- **26.05.2024** → Realizarea videoclipurilor care atestă corectitudinea acordorului;

Bibliografie/Resurse

Bibliografie

https://forum.arduino.cc/t/i-want-to-create-a-guitar-tuner-steps-to-take-parts-to-use/686392

https://www.youtube.com/watch?v=CvqHkXeXN3M&list=PL2orsk2zoELnX3RqZVVvgOhDJaFO8XCjQ

https://www.youtube.com/watch?v=Sj142ZOLEhM&list=PL2orsk2zoELnX3RqZVVvgOhDJaFO8XCjQ&index=2

https://www.youtube.com/watch?v=0nmDqKCMh3Q&list=PL2orsk2zoELnX3RqZVVvgOhDJaFO8XCjQ&index=3

https://docs.arduino.cc/built-in-examples/basics/Blink/

https://docs.arduino.cc/built-in-examples/digital/Button/

https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

https://www.allaboutcircuits.com/technical-articles/using-interrupts-on-arduino/

https://www.arduino.cc/reference/en/libraries/arduinofft/

https://docs.arduino.cc/learn/electronics/lcd-displays/

https://www.circuitbasics.com/how-to-use-microphones-on-the-arduino/

Export to PDF