



**SZÉCHENYI
EGYETEM**
UNIVERSITY OF GYŐR
GÉPÉSZMÉRNÖKI, INFORMATIKAI
ÉS VILLAMOSMÉRNÖKI KAR

TMDK dolgozat

Hangoló pedál készítése frekvencia detektáló algoritmus alapján

Konzulens: Somogyi Miklós

Buni Roland István, Szabadi Tibor

Villamosmérnöki BSc

3. évfolyam

Kivonat

Hangoló pedál készítése frekvencia detektáló algoritmus alapján

Célunk egy működésében hangoló pedálra emlékeztető hangtechnikában használatos elektronikai eszköz elkészítése. Egy hangoló pedál, olyan eszköz, amely az elektromos gitár kimeneti jele alapján meghatározza, hogy az melyik hangon, illetve milyen frekvencián szólal meg. Ez nagyon hasznos sok zenésznek, hisz koncert és gyakorlás előtt kényelmesen hangolni tudja hangszerét, mivel az az idő haladtával elhangolódik (megfeszül a húr stb). Kíváncsiak voltunk/vagyunk mégis mekkora összegből tudunk egy ilyen eszközt összehozni, mennyit spórolunk meg azzal, hogy egy kicsit is értünk az elektronikához, illetve programozáshoz.

Egy hangoló pedál felépítését tekintve kettő 6,3 mm-es jack bemenetből és kimenetből, illetve egy kijelzőből áll, ami mutatja az éppen megszólaltatott húr hangját, továbbá ennek a hangtól való pozitív, valamint negatív irányú eltérését. Használat közben ez az eszköz a gitár és a gitárerősítő között helyezkedik el.

Ennek a hangolónak a megvalósításához fel kell használni, mind az elektronikában, mind a mikrovezérlő programozásban tanultakat. Terveink szerint ehhez el kell készítenünk egy előerősítő áramkört, amivel a gitár kimeneti jelét hozzá lehet illeszteni a mikrovezérlőnk bemenetére, és írni egy frekvencia felismerő programot C++ nyelven, ami alkalmas feldolgozni a gitár analóg jelét. Ezt a legegyszerűbb módon egy Arduino UNO REV 3-as boarddal tudjuk megvalósítani, mivel könnyen elérhető. Az adatok kiírása egy LCD kijelzőre fog történni. Ezen LCD kijelzőn minden SPN táblázatban foglalt hangot terveink szerint meg fogunk tudni jeleníteni, továbbá a pontos frekvenciájukat is.

Tartalomjegyzék

1. Bevezetés	2.
2. Elektromos gitár működése	3.
3. Projekt felépítése	4.
4. Arduino	5.
4.1 ATmega328P	6.
4.2 LCD kijelző	7.
4.3 LCD kijelző bekötése	8.
5. Áramkör	10.
5.1 Erősítő	11.
5.2 TL 072CP/DIP	11.
5.3 DC Offset	11.
5.4 Nyomtatott Áramkör	13.
6. Frekvencia felismerő program	14.
6.1 main.ino	16.
6.2 for_display.h	20.
7. Mérési eredmények	25.
8. Összefoglalás	27.
9. Irodalomjegyzék	28.

1. Bevezetés

Dolgozatunk egy hangoló pedál (1. ábra) megalkotásáról szól. Egy ilyen eszköz arra képes, hogy egy elektromos gitár, illetve az ahhoz tartozó erősítő közé kötve, mutatja valamilyen formában (LED vagy kijelző), hogy éppen melyik hangon szólal meg a hangszer, továbbá jelezze azt is, hogy a legközelebb eső, SPN táblázatban foglalt hangtól pozitív vagy negatív irányban tartózkodik éppen a megszólaltatott húr frekvenciája.

A hangoló pedál gyakorlati haszna gyakorlás és koncert előtt jelenik meg. Minden játék előtt illik újrhangolni a hangszerünket, hiszen az a mechanikai hatások miatt elhangolódik, ha csak egy kicsit is. Egy ilyen pedál meglehetősen drága, dolgozatunk végén megnézzük mégis mennyivel olcsóbban úsztuk meg egy kis elektronikai, illetve programozási tudás felhasználásával.

Megalkotása ennek az eszköznek az előbb is említett elektronikai és programozási tudásunk felhasználásával történt. Kipróbálhattuk az érintett tárgyakból tanultakat éles helyzetben is, és kiegészíthettük azokat új ismeretekkel. Dolgozatunkban szeretnénk biztosítani, hogy annak elolvasása után az Olvasó teljes mértékben megértse a projekt felépítését és abban jelenlevő minden alkotóelemének szerepét. Végigkísérjük a gitár jelének létrejöttétől, a szükséges hardveren át a programig, ami majd a végén az LCD kijelzőnkön megmutatja az aktuális hangot.



1. ábra: Egy Donner márkájú pedál.

2. Elektromos gitár működése

Mielőtt bárminek is neki tudtunk állni először meg kell értenünk egy elektromos gitár működését. Mi alapján tudjuk kinyerni a nekünk szükséges információt? Mi az a jel? Nevéből adódóan is következik, hogy valamelyik villamos jelenség fogja közölni velünk ezt. Ez pedig a gitár kimeneti feszültsége lesz.

Egy elektromos gitár alapvetően annyiban különbözik akusztikus társától, hogy nincs neki „dobja”, ami a hangot felerősíti, helyette vannak ezek az úgynevezett hangszedők (2. ábra). Egy gitáron egy, de akár három hangszedő is elhelyezkedhet és egy kapcsoló állításával lehet közölni választani, hogy éppen melyik „működjön”. A működjön, azért van idézőjelben, mivel ezek (javarészt) passzív eszközök. Ezek a hangszedők nem csinálnak mást, mint a gitár húrjainak rezgési frekvenciáját átalakítják azzal arányos feszültséggé.



2. ábra: Egy humbucker pick up.

Nézzük meg ez, hogy történik. Felépítését tekintve a hangszedők legegyszerűbb formája, hogy mágneseket helyeznek el minden húr alatt és ezeket körbetekercselik. Ez azt eredményezi, hogy a húrok mágneses mezőbe kerülnek és, ha azokat megpengetjük Faraday törvénye értelmében mozgásuk miatt a tekercsben feszültség fog indukálódni. Rengeteg féle pick up létezik, az előbb említett felépítés a klasszikus rockba kedvelt, mivel kicsit zajos kimeneti jelünk lesz, de léteznek több tekercses (zajcsökkentett) humbucker hangszedők, illetve aktív hangszedők is, amik már más elven működnek. A lényeg, hogy mindegy milyen hangszedőnk van a kimeneti jelünk az feszültség lesz, aminek frekvenciája a húr hangjának frekvenciájával lesz azonos, illetve amplitúdója 20mV –tól

400mV –ig változhat (ritka kiugrások lehetnek 2V –ig, de ez nem fog problémát okozni). Az előbb említett „zajosság”-ból az ember azt gondolhatja, hogy valamiféle szűrést, esetleg FFT-t kell majd alkalmazni, de ezekre semmi szükség, mivel a felharmonikusok a gitár jelére „ülnek” rá és elhanyagolhatók. Csakis a gitár „hangszínére” lesznek hatással.

3. Projekt felépítése

Így, hogy már tudjuk mi az az információ hordozó, amit fel kell dolgoznunk és mi is az az eszköz, aminek mintájára akarunk egy hasonlót készíteni, letudjuk fektetni a mi eszközünkkel támasztott feltételeket, ötleteket.

A mi hangolóknak tehát szüksége lesz egy 6,3 mm-es jack bemenetre, illetve kimenetre. Ki és bekapcsolhatónak kell lennie, de kikapcsolt állapotban is tovább kell küldenie a jelet a gitár erősítő felé. Nem okozhat zajt a gitárerősítő által felerősített jelben. Fel kell tudnia dolgozni a gitár jelét oly módon, hogy az nagyon kicsi tűrésen belül a tényleges SPN táblázatban (3. ábra) foglalt hangok egyikét ábrázolja, és elemnek vagy elemeknek kell lenniük a táp forrásoknak.

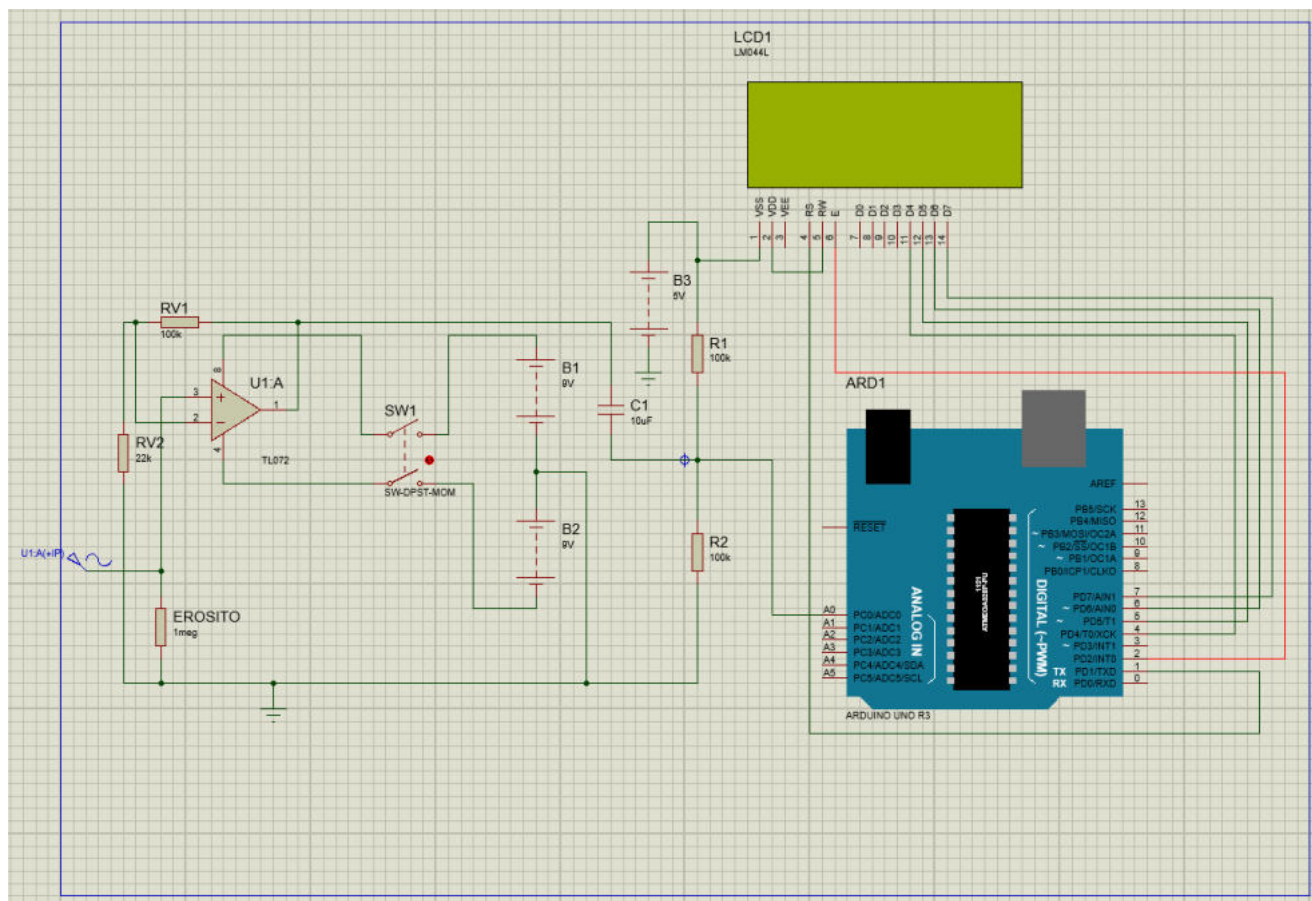
Note	0	1	2	3	4	5	6	7	8	9
C	16.352	32.703	65.406	130.81	261.63	523.25	1046.5	2093	4186	8372
D	18.354	36.708	73.416	146.83	293.66	587.33	1174.7	2349.3	4698.6	9397.3
E	20.602	41.203	82.407	164.81	329.63	659.26	1318.5	2637	5274	10548
F	21.827	43.654	87.307	174.61	349.23	698.46	1396.9	2793.8	5587.7	11175
G	24.5	48.999	97.999	196	392	783.99	1568	3136	6271.9	12544
A	27.5	55	110	220	440	880	1760	3520	7040	14080
B	30.868	61.735	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1	15804

Each scientific pitch notation corresponds to a certain frequency.
doi:10.1371/journal.pone.0058710.t002

3. ábra: Scientific pitch notation.

Szükségünk lesz egy vezérlőre, ami képes a gitár jelét feldolgozni, digitális jellé alakítani, illetve egy LCD kijelzőt is tud kezelni. Kézen fekvő megoldás egy mikrovezérlő. Sok lehetőség van a Microchip termékeitől kezdve az Intelig, de a legkézenfekvőbb mégis valamelyik Arduino board lenne.

Nem minden mikrovezérlőnek azonos az analóg/digitális feldolgozó egysége. Vannak határok, hogy milyen tulajdonságú jeleket tudnak értelmezni, ezért a mi gitár jelünket is alakítani kellett kicsit. Ennek köszönhetően egy előerősítő áramkört kellett terveznünk, ami ezt meg tudja nekünk oldani. A teljes áramkör a 4. ábrán található.



4. ábra: A hangoló kapcsolási rajza.

4. Arduino

Több ok miatt döntöttünk az Uno Rev 3-as board mellett. Az első az volt, hogy ingyenes, egyszerű fejlesztő környezete van, rengeteg különböző fórumokon található segédanyaggal, illetve könyvtárral. A másik, ami inkább jelentősebb ok volt, az az, hogy nem egy önmagában egyedülálló mikrokontrollerről beszélünk, mint például egy PIC 16-os esetében, hanem egy Atmel chip köré van tervezve egy NYÁK lap, amin I/O portok, táp stb csatlakozók kivezetésre kerültek, így ez rengeteg szabadságot nyújtott nekünk tesztelesek során.

4.1 ATmega328P

Az Uno Rev 3 –ban egy Atmega328P mikrovezérlő kapott helyett. A gitár feszültségének feldolgozásához ennek a mikrokontrollernek az A/D konverterét kell használnunk. Ez a következő specifikációkkal rendelkezik:

Mintavételezési frekvenciája:

$$f_m = 9600[Hz] \quad (1)$$

Ez nekünk bőven elegendő, hiszen a „legmagasabb” hang, amit a hangolónknak ismernie kell az az E4, ami pontosan 329,63 Hz. Shannon tételének értelmében, bőven a szükséges minimális mintavételezési frekvencia felett teljesít a mikrovezérlőnk. Jogosan merülhet fel a kérdés miért E4 a legmagasabb hang, pedig az SPN táblázatban ettől jóval magasabb frekvenciájú hangok is szerepelnek. Ez azért van, mert a gitár hangolásoknál az E standard a legmagasabb hangolás, nagyon ritka az, hogy úgynevezett „up tuningot” használnának. Ennek az E standart hangolásnak a legmagasabb hangja az E4, ettől csak mélyebb hangokat használunk. Nézzünk további adatokat. Az A/D konverter felbontása

$$Q = 2^{10} = 1024 \quad (2)$$

A legkisebb helyiértékű bit megváltoztatásához szükséges feszültség érték

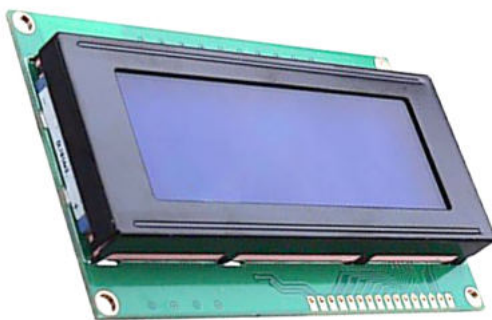
$$U_{LSB} = \frac{U_{FS}}{Q - 1} = \frac{5[V]}{1023} = 4.89[mV] \quad (3)$$

ahol U_{FS} az a feszültség tartomány, amelyet az arduino A_0 -ás bemenete kezelni tud.

Tehát $dU = 4,89[mV]$ feszültség változás kell ahhoz, hogy a legkisebb helyiértékű bit megváltozzon. Már most látszik, hogy önmagában a gitár kimeneti feszültségét erre a pinre rákötve nem hozna eredményt, mivel van olyan hang, aminek amplitudója csupán maximum 20mV. Ahhoz, hogy megfelelően használni tudjuk az A/D konvertert egy erősítő áramkör és DC offset alakít a jelünkön. Ezt egy későbbi fejezetben taglaljuk.

4.2 LCD kijelző

Megjelenítendő frekvenciákat, és az ahhoz tartozó SPN hangokat egy LCD kijelzőre (5. ábra) jelenítjük meg, melyet az Arduinoval tudunk elérni. Kiválasztott kijelző a 20x4 I2C Character LCD Display. Választásunk azért erre a kijelzőre esett, mivel kellően nagy, ezáltal bármilyen hangot és frekvenciát egyszerre is kitudunk írni, anélkül, hogy helyhiányban szűkösködnénk.



5. ábra: 20x4 I2C Arduino LCD Display.

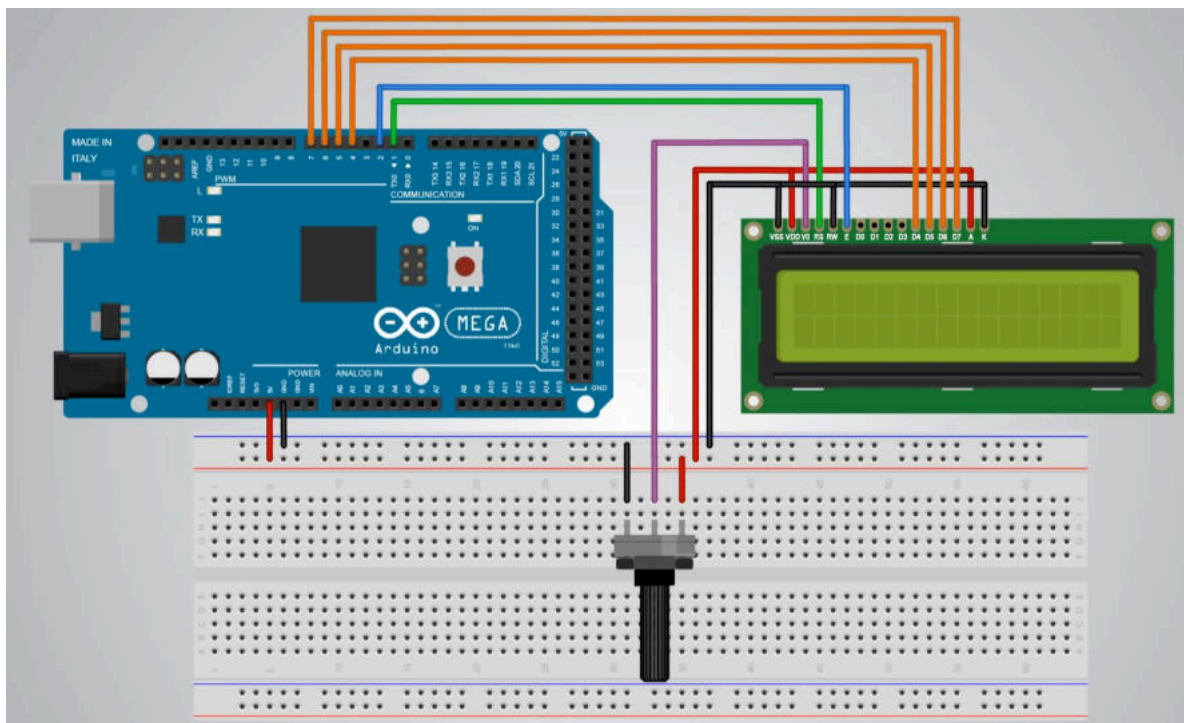
Az adatokat olyan elrendezésben jelenítjük meg, hogy mind a frekvencia és a hang is leolvasható legyen. A frekvenciát a kijelző LCD bal alsó sarkába, a hangot pedig középre tettük. Ez látható a 6. ábrán.



6. ábra: Valós példa a kijelzőről.

4.3 LCD kijelző bekötése

Az LCD kijelzőn összesen 16 pin található. Az első 6 pin (V_{SS} , V_{DD} , V_0 , R_S , R_W , E) tápfeszültségért, kontrasztért, regiszterekért felelős, további 8 pin (D_0 , D_1 , D_2 , D_3 , D_04 , D_5 , D_6 , D_7) adathordozó biteket tartalmaz, az utolsó 2 pin (A , K) pedig a háttérvilágításért felelős.



7. ábra: LCD kijelző teljes kapcsolási rajza Arduinoval.

V_{SS} és V_{DD} lábak adnak tápfeszültséget a kijelzőnek, V_{SS} -t földre, V_{DD} -t pedig 5[V] feszültségre kötjük. V_0 lábat egy 10k Ω -os potencióméterrel vezéreljük, ez a láb felelős a kijelző kontrasztjáért.

R_S láb egy H/L regiszter, a jel kiválasztásáért felelős, ezt a lábat az Arduino PWM (Pulse Width Modulation) 1. lábába kell kötni. R_W szintén egy H/L regiszter, mellyel beállíthatjuk, hogy írunk, vagy olvasunk a memóriába. Földelni kell (tehát alacsony logikába tenni), hogy a kijelzőt Write módba használjuk, azaz Arduinótól kapott adatokat megjelenítse. Az E láb Enable-t (engedélyt) jelent, ezt az Arduino PWM 2. lábába kötöttük.

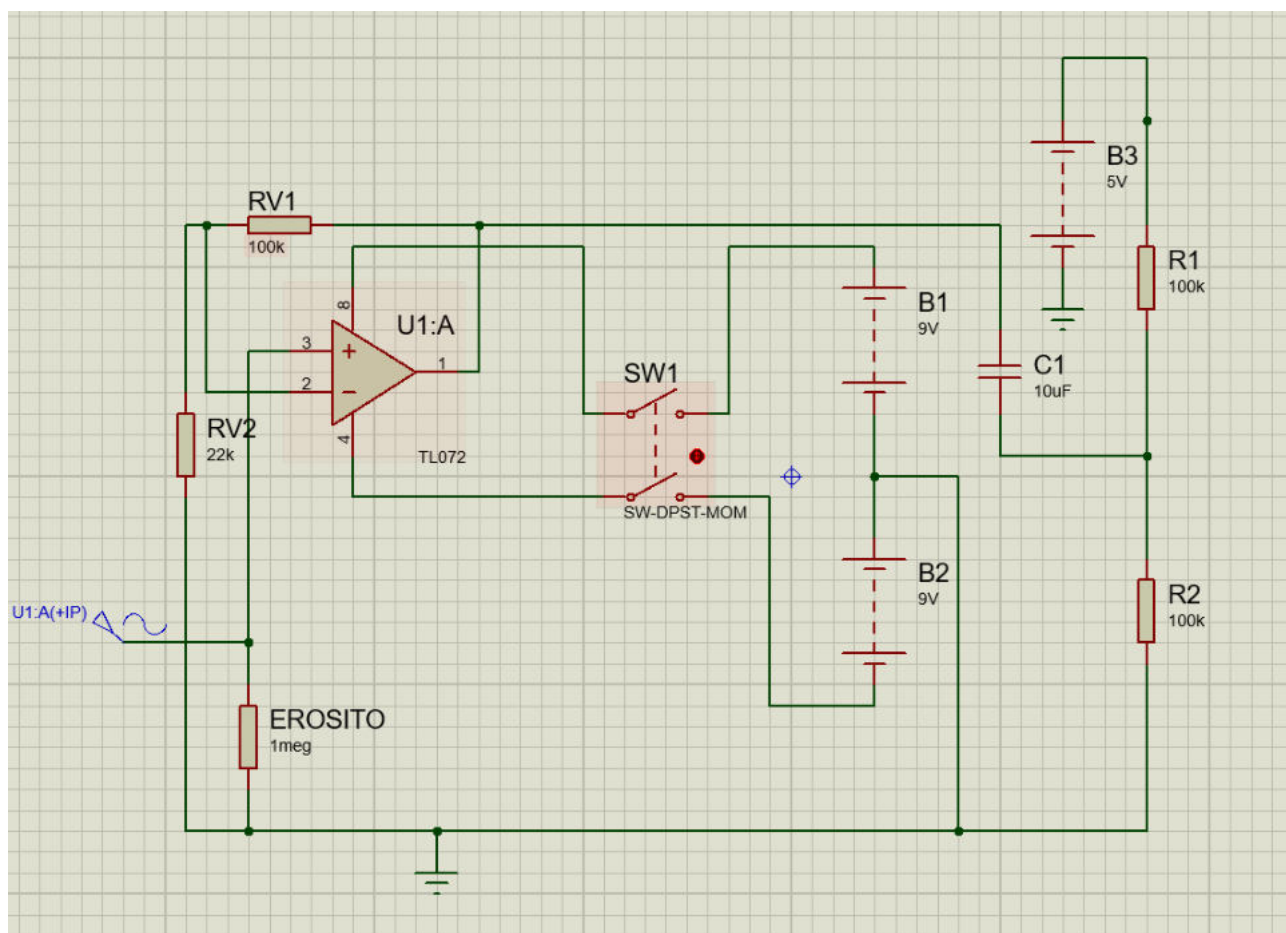
Az adat lábakat óvatosan kellett bekötni, mivel nem mindegy, hogy melyik biteket használjuk. Mi az utolsó 4 adat lábakat (D_4 , D_5 , D_6 , D_7) használtuk fel, hogy az Arduinótól kapott adatokat megtudjuk jeleníteni. Ezeket a lábakat az Arduino PWM 4-7 lábaihozhoz kellett csatlakoztatni.

Az utolsó 2 pin a kijelző háttérvilágításáért felelős, A (anód) lábat 5[V]-ra, K (katód) lábat pedig földre kell lehúzni.

```
1 #include <LiquidCrystal.h>
2 #define SOR 4
3 #define OSZLOP 20
4
5 LiquidCrystal lcd(1,2,4,5,6,7); //RS,E,D4,D5,D6,D7
```

8. ábra: C++-ban megadott kód. "lcd" objektum paraméterei az Arduino PWM lábak sorrendjei.

5. Áramkör



9. ábra: Előerősítő és DC Offset áramkör.

Az áramkörünk 3 részegységre bontható. Előerősítő, DC Offset (9. ábra), és Arduino részre. Ebben a fejezetben végigmegyünk, és részletesen leírjuk, hogy mi miért felelős. Gyors bevezető gyanánt, az erősítő egy nem invertáló alapkapsolás, mely a beérkező gitár jelünket felerősíti, hogy az Arduino felismerje, és tudjon vele számolásokat végezni. A DC Offset eltolja a jelünket, hogy az Arduino felismerje a negatív fél periódusokat, az Arduino pedig a frekvencia számolásért és az adatok kiírására az LCD képernyőnkre felelős.

5.1 Erősítő

Mint ahogy az 9. ábrán is látható egy nem invertáló alapkapsolás lesz az erősítő áramkörünk. A tápellátásunkról 2 db 9V-os elem gondoskodik. R_{V1} és R_{V2} értékéből következik

$$A_u = 1 + \frac{R_{V1}}{R_{V2}} = 1 + \frac{100[k\Omega]}{22[k\Omega]} = 5,54 \quad (4)$$

ezzel az erősítés értékkel a legkisebb és legnagyobb U_{cs-cs} érték, ami az Uno Rev 3 A_0 -ás pinjére fog érkezni.

$$U_{cs-csmin} = 222[mV] \quad (5)$$

$$U_{cs-csmax} = 4,44[V] \quad (6)$$

Ahogy kapott értékekből látható az $U_{FS} = 5[V]$ feszültség tartományunkat eléggé jól kihasználjuk.

5.2 TL 072CP/DIP

Több szempontot néztünk mikor műveleti erősítőt választottunk. Mivel hangtechnikai eszközbe szeretnénk implementálni, ezért a lehető legkisebb zajjal rendelkezzen, ne torzítsa. Nagy slew rattal rendelkezzen, hiszen a gitárhúr pengetése után, amint lejár a „tranzienst” idő szeretnénk, hogy mihamarabb az „állandósult” feszültség alapján számoljon frekvenciát a programunk és, ami a legfontosabb legyen elérhető, olcsó. A TL 072 minden feltételnek megfelel.

Slew rateja

$$SW = 13 \left[\frac{V}{\mu s} \right] \quad (7)$$

Maximum harmónikus torzítás 0,003%

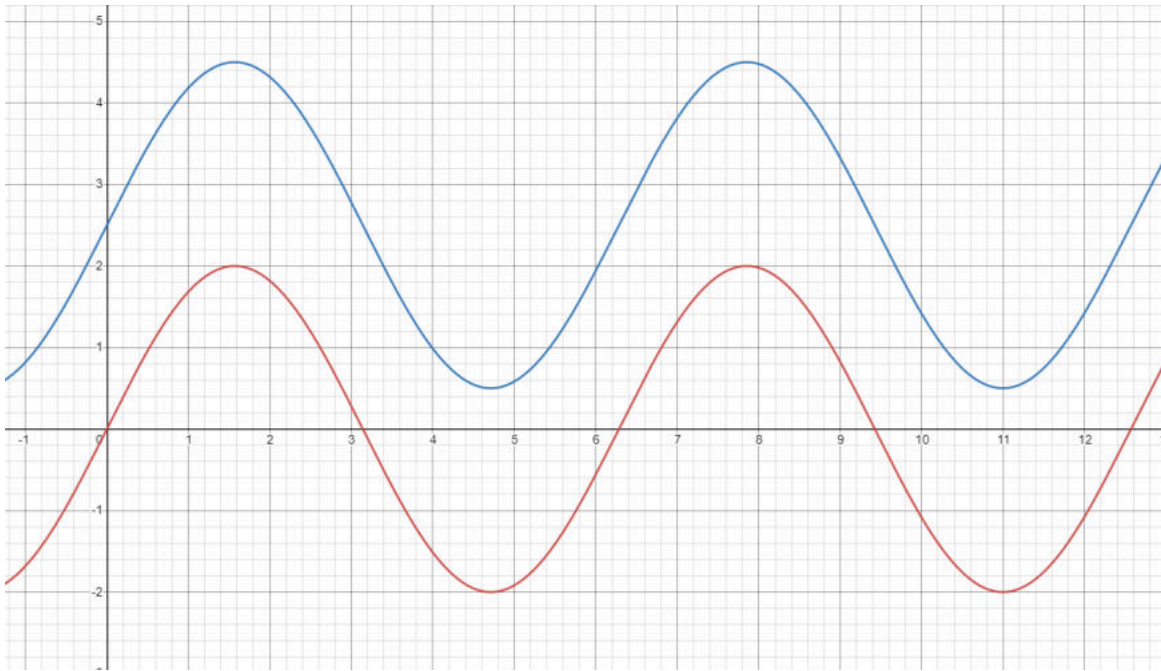
5.3 DC Offset

Egy problémába ütközünk, pontosabban hogy, a feszültségünk jelenleg 0V potenciál körül oszcillál, viszont az A_0 -ás pin csak 0-5 [V] -ig érzékel, vagyis a negatív félperiódusokat levágná. Emiatt az R_2 , R_1 , és C_1 elemekből a műveleti erősítő kimenetére van csatlakoztatva egy egyenfeszültségű eltolás, ami az oszcillációt 2,5 [V] köré teszi számunkra (10. ábra). A szükséges 5 [V]-os tápfeszültség

séget az Arduino szolgáltatja hozzá. Egyből feltűnik, hogy ezek az elemek egy felüláteresztő szűrőt képeznek, de az alkatrész értékek úgy vannak megválasztva, hogy a vágási körfrekvencia

$$\omega = 0,3[Hz] \quad (8)$$

vagyis elhanyagolható szűrő jelleg van.

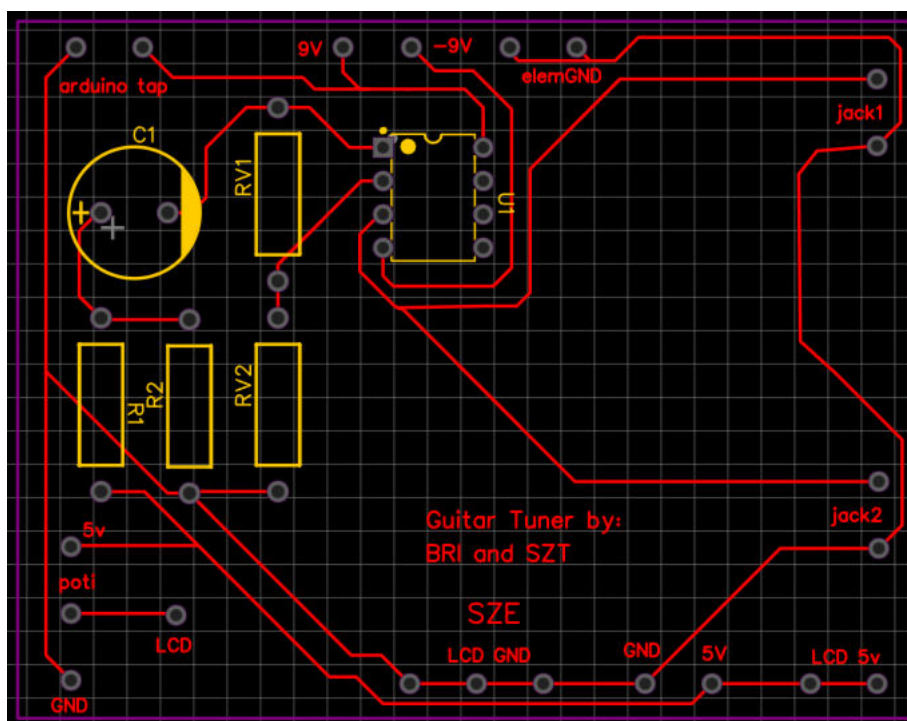


10. ábra: DC Offset. Jól látható, ha eltoljuk a jelünket 2,5V-al akkor a negatív félperiódus megmarad.

R_1 és R_2 értékének megválasztásánál a feszültségerősítésén kívül az játszott döntő tényezőt, hogy az ofszetet és driftet minél jobban csökkentsük. Ennek feltétele, hogy a műveleti erősítő mindkét lábán azonos generátor ellenállások legyenek. Mivel a generátor ellenállásunk jelenleg a gitár pickupjából, a rajtalévő tónust és hangerőt szabályzó potméterekből tevődik össze, ennek nagyságrendje pár $10k\Omega$.

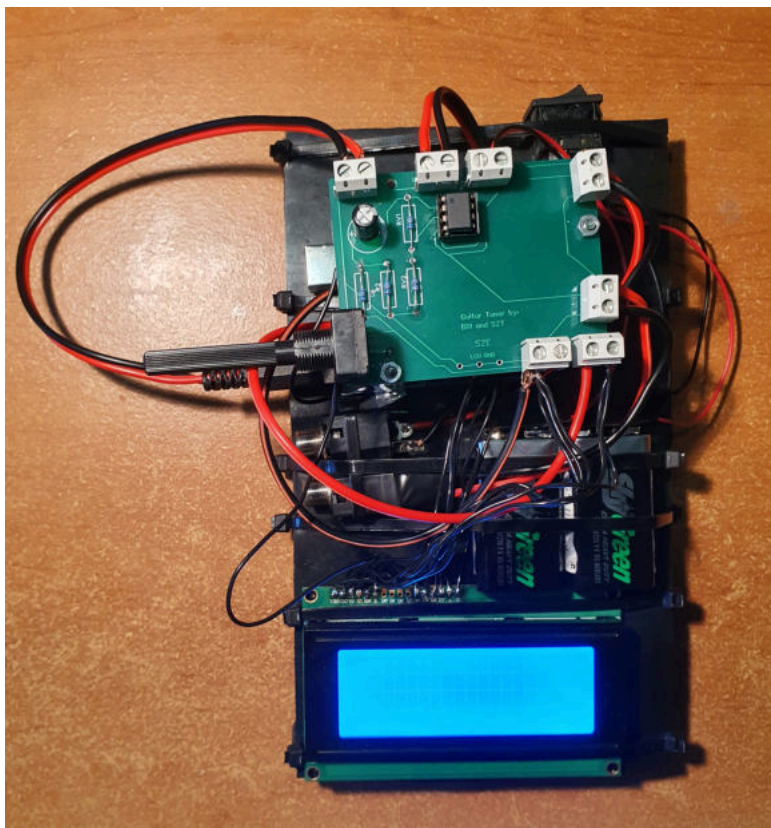
5.4 Nyomtatott Áramkör

Az eszközünket a rendelkezésünkre álló lehetőségekhez mérten minél kompaktabbra szeretnénk volna elkészíteni, ezért az előerősítő áramkörünket, ami a gitár jelét az Arduino A₀-ás pinjéhez illeszti egy külön NYÁK tervet kapott. A 11. ábrán láthatjuk.



11. ábra: NYÁK terv

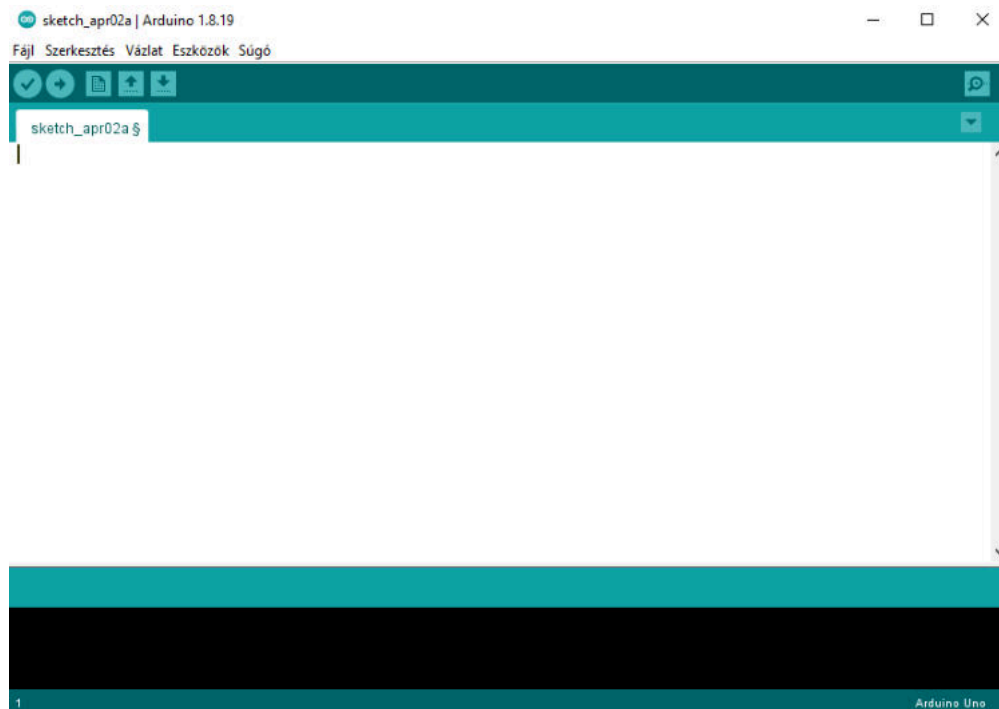
A tervezés alatt azt vettük figyelembe, hogy maga az Arduino board mekkora, ezzel megegyező méretűre készítettük, hisz így egymásra tudtuk két csavarral illeszteni őket. Ez a méret rengeteg szabad helyet is biztosított nekünk a NYÁK lapon, ami nagyban megkönnyítette a tervezést. A NYÁK -on szereplő potenciál értékeket az elemek és az Arduino pinjei szolgáltatják. Az elkészült gitár hangolót az alábbi ábra mutatja.



12. ábra: Megépített hangoló

6. Frekvencia felismerő program

Ahhoz, hogy a jelünkkel matematikai műveleteket tudjunk végezni szoftveres szinten egy mikrovezérlőt kell használnunk. Jelen esetünkben Arduinot választottunk, melynek egyedi programozási környezete van.



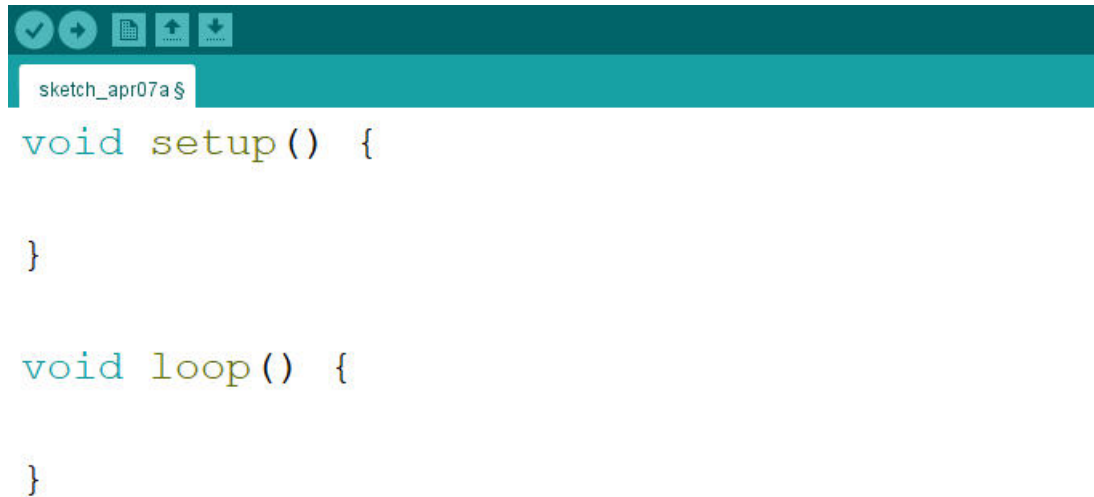
13. ábra: Arduino programozási felülete, az Arduino IDE.

Az Arduino programozási nyelve a C++. A C++ egy nagyon népszerű nyelv a teljesítménykritikus alkalmazásokhoz, amelyek gyorsaságon és hatékony memóriakezelésen alapulnak. Számos iparágban használják, beleértve a szoftver- és játékfejlesztést, a VR-t, a robotikát és a tudományos számítástechnikát. 1985-ben adták ki az első verzióját. Elődjéhez képest, a C-hez a változás az, hogy már támogatja az osztályokat és az objektumokat, ezáltal OOP (Objektum Orientált Programozási) nyelv. Ezt a funkciót ki is használtuk.

Programunk két fájlból tevődik össze. Az egyik a *main.ino*, melyben a frekvencia kiszámolása és az Arduino kezdeti beállításai vannak, másik fájl pedig egy header fájl, *for_display.h*. A header fájlok a C nyelv találmányai, lehetővé teszik számunkra, hogy a deklarációkat egy helyre helyezzük, majd bárhová importáljuk őket, ahol szükségünk van rájuk. Ezzel sok gépelést megspórolhatunk a többfájlos programokban. Jelen esetünkben a header fájlunkba frekvencia kiválasztását, és az egyedi betűk, számok kijelzését tettük, és az LCD beállításait.

Minden Arduinon történő kódnak három része van (14. ábra). A "setup" nevezetű scopeban azt a programkódot írjuk, amelyiket csak akkor akarjuk lefutatni mikor az Arduino elindul, és ez csak egyszer fut le. A "loop"-ba azt írjuk, amit szeretnénk, hogy folyamatosan lefusson, jelen esetünkben a frekvencia kiszámolást. Ezeken kívül pedig publikus változók létrehozását, vagy header fileok

beolvasását szokás csinálni a "whitespace"-ben.



14. ábra: Arduino 3 része, *setup*, *loop*, és a *whitespace*.

6.1 main.ino

A programunk fő fájljában az Arduino kezdeti beállításai és a frekvencia kiszámolása kapott szerepet. Ebben a fejezetben kódblokkok halmazával írjuk le a program működését.

```
1 //Importing header files
2 #include <LiquidCrystal.h>
3 #include "for_display.h"
```

Legelőször beimportáljuk a kötelező könyvtárakat, header fájlokat. Jelen esetünkben kettő van, amelyek nélkül nem működne a programunk. Legelső a *LiquidCrystal* nevezetű könyvtár, mely a kommunikációért felel az Arduino és az LCD kijelző között. Következő header fájl amit importálunk, a *for_display.h*, általunk létrehozott header fájl.

Kezdjük először a *LiquidCrystal* könyvtárral, ez a könyvtár lehetővé teszi az Arduino számára az LCD kijelzők vezérlését a Hitachi HD44780 (vagy egy kompatibilis) chipsettel, amely a legtöbb szöveges LCD-n megtalálható. A könyvtár 4 vagy 8 bites módban működik (azaz 4 vagy 8 adatsor használatával az R_S , Enable és opcionálisan az R_W vezérlővonalak mellett). Mi csak 4 bites üzemmódban dolgozunk, mivel 4 bit elég, hogy ASCII táblázat alapján megjelenítse számunkra fontos

információkat.

A *for_display.h* header fájlt egy későbbi fejezetben bővebben bemutatjuk.

```
1 //Creating variables
2 #define SOR 4
3 #define OSZLOP 20
4 #define PIN A0
5 int bekapcs,kikapcs;
6 float frek,periodus;
7
8 //Display instantiation
9 Display d;
```

Következő sorokban változókat hozunk létre és példányosítunk, mely a header fájl miatt szükséges. Három fajta változó típussal dolgozunk a programban, define, int és float. A define parancssal olyat változókat hozunk létre, melyeknek értéke a program futási ideje alatt nem változik, konstans marad. Jelen esetünkben ilyen a *SOR*, *OSZLOP* és *PIN* változók. Az első kettő az LCD méreteit definiálja, míg a *PIN* változó azt a lábat adja meg, amelyen az Arduinó a gitár jelét kapja. Ezek a konstansok soha nem változnak.

Következő négy változó már olyan értékeket képviselnek, amelyek folyamatosan változnak. Az int és a float közötti különbség, hogy az int, mint integer, egész számokat tud csak felvenni, míg a float akár törtszám értéket is tud értelmezni. Ez a négy változó mögöttes tartalmába később megyünk bele, mikor elérünk a kódrészletekhez.

Példányosításra azért van szükségünk, hogy a *for_display.h* header fileből megtudjuk hívni azokat a függvényeket amelyeket abban a fájlban hoztunk létre.

```
1 //Initial settings
2 void setup()
3 {
4   pinMode(PIN, INPUT); //A0 pin to input
5   lcd.begin(OSZLOP, SOR); //Display's size
6 }
```

A *setup* scope az Arduinonak az a része, amely csak egyszer fut le, mikor elindul. A *void* arra utal, hogy a *setup* függvénynek nincs semmilyen return értéke, azaz nem ad semmi visszatérési értéket. Két dolgot állítunk be, hogy az A_0 -s láb legyen a bemenet, és hogy mekkora a kijelzőnk mérete.

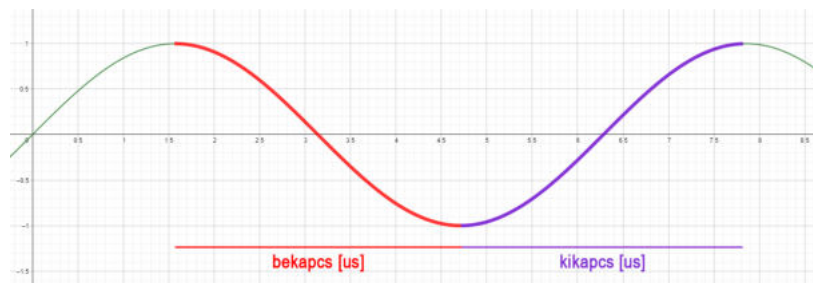
A **pinMode** egy beépített függvény, ami azt jelenti, hogy az Arduino fejlesztői környezete felismeri alapból. Feladata, hogy a megadott lábat úgy konfigurálja, hogy bemenetként vagy kimenetként viselkedjen. Két paramétere van, amelyet meg kell adni. Értелеmszerűen a lábat, amellyel dolgozni szeretnénk, és a viselkedési módot. Jelen esetünkben az A_0 -s lábat használjuk, ezért megadjuk a *PIN* konstanst, mivel értéke A_0 , és bemenetként szeretnénk vele dolgozni, mivel innen kapjuk a gitár jelét, ezért INPUT értéket adunk meg a második paraméter helyére.

Az **lcd.begin** paranccsal adjuk meg, hogy egy LCD-t használunk, és hogy mekkora a mérete. Jelen esetünkben egy 20x4-es kijelzőt használunk, ezért megadjuk az *OSZLOP* és *SOR* konstanst.

```
1 void loop()
2 {
3     bekapcs = pulseIn(PIN,HIGH); //In microseconds, we get back the time until
        it goes from high to low
4     kikapcs = pulseIn(PIN,LOW); //In microseconds, we get back the time until
        it goes from low to high
5     periodus = bekapcs+kikapcs; //Period calculation
6     frek = 1000000.0/periodus; //Frequency calculation
7
8     if(periodus > 0){ //Real data
9         d.frequency_selector(frek); //SPN selector and display
10        lcd.setCursor(0,3); //Cursor to lower left corner
11        lcd.print(frek); //Exact frequency display
12    }
13
14    delay(200); //Delay
15    lcd.clear(); //Clear display
16 }
```

Következő és egyben a *main.ino* utolsó részletdarabja, a *loop scope*. Ez az a része a kódnak, mely folyamatosan fut és számolja a frekvenciát.

A frekvencia kiszámolásához a *bekapcs*, *kikapcs*, *periodus*, és *frek* változókat, és a beépített **pulseIn** függvényt használtuk fel. A **pulseIn** függvény egy időértéket ad vissza. Két paramétert fogad el, a lábat (jelen esetben nálunk az A_0), és hogy HIGH vagy LOW. Például, ha az érték HIGH, a **pulseIn()** megvárja, hogy a láb LOW-ról HIGH-ra váltson, elindítja az időzítést, majd megvárja, amíg a láb LOW-ra megy, és leállítja az időzítést. Visszaadja az impulzus hosszát mikroszekundumban, vagy feladja és 0-t ad vissza, ha az időkorláton belül nem érkezett teljes impulzus. Legegyszerűbben ezt a 15. ábra segítségével tudjuk szemléltetni.



15. ábra: pulseIn függvény szemléltetése.

A *bekapcs* és *kikapcs* változók mentik el ezeket az értékeket. A *bekapcs* változó menti el azt az időszakot, míg a jelünk magasról alacsonyra megy, míg a *kikapcs* alacsonyról magasra való időt. Mindezek nem másodpercben, hanem mikromásodpercben vannak elmentve. Ez később fontos, mivel a frekvencia kiszámolásához ezt figyelembe kell venni!

Jól látható, ha a frekvencia minél magasabb, annál gyorsabban zajlott le valami. Kérdés már csak az, hogy jelen esetünkben mi az a "valami". Nálunk a T , a periódusidő ami a frekvenciát adni fogja. Periódusidő alatt értjük azt, hogy a jel visszatér ugyanazon állapotába. 15. ábrát megtekintve nem nehéz rájönni, hogy a periódusidőnk a *bekapcs* és *kikapcs* összege. Ennek tudatában már tudunk frekvenciát számolni, csak venni kell a *periodus* változónk reciprokát. Nem egészen! Gondoljunk vissza, *bekapcs* és *kikapcs* változóink értéke mikroszekundumban, azaz egy milliomod másodpercben vannak megadva, összegük is mikroszekundum lesz. Ha a frekvenciát Hz -ben akarjuk megkapni, akkor ezt a reciprokot még meg kell szorozni 1 millióval.

Jogos feltevés lehet az, hogy az ábrán mutatott jelünk egy teljesen szinuszos jel, míg a valóság nem így működik. Okai rengeteg, pl. gitár megpengetett húrja már alaptól nem teljesen szinuszos

jelet ad ki, erősítő nem csak a hasznos jelet erősíti, hanem a zajt is ráülteti a jelünkre, stb... Ezek az állítások teljesen igazak, de megmértük az eltérést, minél magasabb frekvencián egyre nagyobb ez a tévedés, viszont a legmagasabb frekvencia eltérés 8Hz volt 2000Hz-nél.

Frekvenciát kiszámolva már kiírhatjuk a kijelzőnkre, de először nézzük meg, hogy egyáltalán pengetjük-e a gitárt, ezt pedig egy if-es szerkezettel tudjuk megvalósítani. A feltételünk az, hogy a frekvenciánk magasabb-e mint 0, tehát játszunk a gitárral. A **frequency_selector** függvény már nem beépített, hanem saját magunk által írt találmány, *for_display.h* fájlban található meg. Későbbi fejezetben részletesen leírjuk, hogyan működik, egyelőre azt érdemes tudni, hogy frekvencia alapján kiírja a potenciális húr hangot, pl. C₄. Következőleg nem csak a megpengetett hangot, hanem a pontos frekvenciát is ki akarjuk írni. Ezt úgy érjük el, hogy a kurzort letesszük a baloldali sarokba, ahol kiírjuk a frekvenciát. Kurzort a **setCursor** beépített függvénnyel tudjuk elérni, melynek paraméterei oszlop és sor. Jelen esetben a kijelző bal alsó sarkába szeretnénk tenni, tehát az 1. oszlop 4. sorába. Mivel C++-ban írjuk a programot a számozás 0-tól kezdődik, ezért a 0. oszlop és a 3. sort adjuk meg paraméternek.

Amint kilépünk az if szerkezetből már csak 2 dolgunk van hátra. Azt akarjuk, hogy az értékek a kijelzőről leolvashatóak legyenek, ezért késleltetjük a programot a **delay** függvénnyel, mely paramétere milimásodpercben adandó meg. Majd a legvégén letöröljük a kijelzőt, hogy az új adatok hiba nélkül jelenjenek meg.

6.2 *for_display.h*

Miért volt szükségünk egy külön fájlra? Ez igazából betudható annak, hogy mennyire olvashatónak és érthetőnek szántuk a frekvencia kiszámolást. A *main.ino* kevesebb, mint 45 sornyi kódból áll. Ránézésre érthetőnek és egyszerűnek tűnik, ami a célunk volt. A *for_display.h* header fájlunk több mint 1600 sorból áll. Ha ennyi kódot egy fájlba tömörítettünk volna, olvashatatlan, taszító hatást keltett volna az olvasóban. Ezért 2 külön fájlt hoztunk létre.

A header fájlba tettük a kijelzőn lévő karakterek megjelenítését és a frekvenciakiválasztó függvényünket, a **frequency_selector**-t. Azt szerettük volna, hogy a megpengetett húr hangját írassuk ki és legyen a legfeltűnőbb adat, mivel az átlag embert az jobban érdekli, mint hogy mekkora frekvenciával szólalt meg a hang. Ezt az által tudjuk elérni, hogy a kijelző közepére tesszük a hangot. Ez

probléma, mivel a kijelzőnk közepe külön sorok és külön oszlopok, névlegesen 10. és 11. oszlop 2. és 3. sor. A probléma az, hogy ezekre külön-külön lehet rajzoltatni karaktereket, de egy, nagy egységeset nem. Ezért azt terveztük meg, hogy külön-külön hozzunk létre saját magunk által készített karakter darabokat, melyeket ha a korrekt helyre (10., 11. oszlop, 2., 3. sor) teszünk akkor a megfelelő betűket és számokat adják ki, ez a 16. ábrán látható.



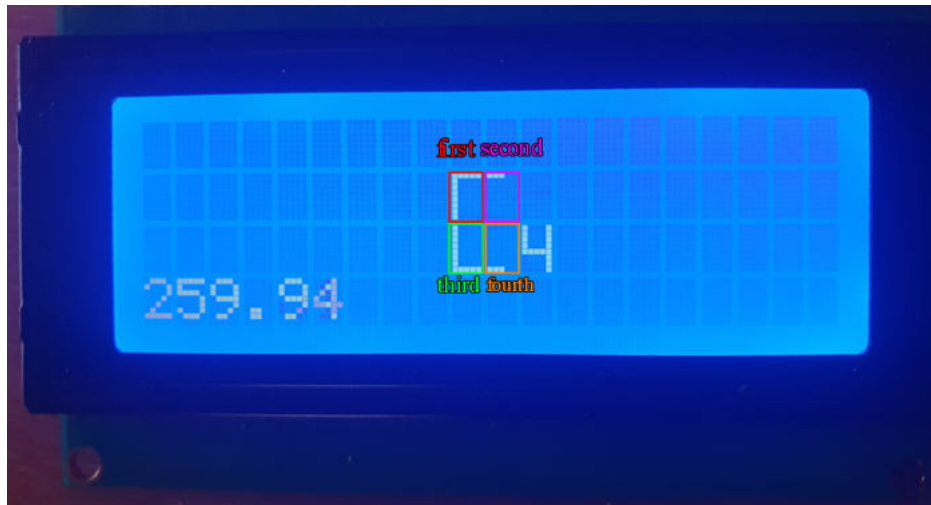
16. ábra: C betű a kijelző közepén, több oszlopot és sort igényel.

```
1 void letter_C(){
2
3     byte first[] = {
4         B11111,
5         B10000,
6         B10000,
7         B10000,
8         B10000,
9         B10000,
10        B10000,
11        B10000
12    };
```

Példaként maradjunk a C betű kiíratásnál. Kód értelmezéséhez kicsit nyúljunk vissza az LCD felépítéséhez. Már ismertettük, hogy a kijelzőnk 20 oszlopból, és 4 sorból áll, azaz összesen 80 rublikánk, vagy pixeltáblánk van. Egy ilyen pixeltábla, vagy pixelmátrix, egy $5 \cdot 8$ pixelek halmaza,

és minden pixelnek 2 állapota lehet, világít vagy sem. Ezeket az állapotokat definiálni lehet, ezáltal egyéni karaktereket lehet kiírni.

Összesen A, B, C, D, E, F, G betűket hoztunk így létre. Nem mutatjuk be az összeset, mivel több száz kód sor lenne, és a folyamat azonos mindegyiknél. Maradjunk a C betűnél. Létrehozunk egy **letter_C** nevű függvényt, melynek nincs visszatérési értéke. Mindegyik betű függvényben definiálunk 4 tömböt, melyek értéke változó minden betűnél, mivel ezekkel a tömb adatokkal adjuk meg a felvillágított pixeleket. A tömbök típusa byte, adatai bitek. A 4 tömb first, second, third, és fourth névvel vannak ellátva, és mindig ugyanazon pixeltáblázatokat töltik ki. A first mindig a 10. oszlop 2. sora, second 11. oszlop 2. sora, stb... 17. ábrán ábráztuk, hogy melyik tömb melyik pixeltáblázathoz tartozik.



17. ábra: Byte tömbök pozíciója a kijelzőn.

A tömb adat típusai bitek, melyeknek 2 állapotuk lehet, 0 vagy 1. Jelen esetünkben ezek adják meg, hogy felvillantjuk az adott pixeleket vagy sem. 1-nél világítanak, 0-nál sötétek. Nem megyünk végig mindegyik 4 tömbön, bemutatjuk a C betű első tömbjét. Jól látható, hogy a byte tömbünk 8 adatot tartalmaz, melyek hossza 5. Pontosan annyi mint a pixeltáblázatban a pixelek száma. Soronként lehet megadni, hogy melyik pixelek világítsanak. Egy sorban 5 pixel van, menjünk végig az első soron. Mindegyik pixel felvilágítjuk, mivel ez lesz a C betűnk felső része, ilyenkor **BIIIII** lesz a sorunk értéke. Folytatva a sorokat látjuk, hogy az értékek ugyanazok, mivel a C betű bal része azonos, mindig csak az első pixel felvilágítjuk meg, többbit lekapcsoljuk.

```

1  lcd.createChar(1, first);
2  lcd.createChar(2, second);
3  lcd.createChar(3, third);
4  lcd.createChar(4, fourth);
5
6  lcd.setCursor(9,1);
7  lcd.write(1);
8
9  lcd.setCursor(10,1);
10 lcd.write(2);
11
12
13 lcd.setCursor(9,2);
14 lcd.write(3);
15
16
17 lcd.setCursor(10,2);
18 lcd.write(4);
19
20 }

```

A `letter_C` függvényünk maradéka pedig, hogy létrehozzuk ezeket a karaktereket, ezt a **createChar** függvénnyel lehet megtenni, majd rendes helyükre tegyük ezeket a karaktereket. Ezeket a jól ismert **setCursor** függvénnyel tesszük meg (0-ról kezdődik a számozás!), majd a **write** függvénnyel kiírjuk.

Betűkön kívül még számokat íratunk ki ilyen módszerrel, különbség az, hogy elég csak 1 tömböt alkalmazni, mivel csak egy pixeltáblázatba tesszük bele, pont a betű utáni jobb alsó sarokba (azaz a 12. oszlop 3. sorába). Számokon kívül még egy + és - szimbólumokat hozunk létre, melyek a számok feletti pixelmátrixba helyezünk (azaz 12. oszlop 2. sorába).

A **frequency_selector** függvényünket a 3. ábrán látható táblázatból építettük fel. Legmagasabb hang ameddig elmentünk az a B_6 , vagyis körülbelül a 2000Hz. Azért "csak" eddig, mivel már ez a frekvencia is valószínűtlen, hogy egy gitár húrjával megpengetjük. Általában a legmagasabb frekvencia amit gitárral szokás játszani az a kb. 350Hz, E_4 -s hang, jól látható, hogy elegendő frekvenciát vettünk fel.

```

1 void frequency_selector(float frek){
2
3 lcd.clear();
4
5 //C0
6 if(frek > 14 && frek < 17.75){
7
8
9     if(frek < 16){
10         letter_C();
11         number_zero();
12         signal_minus();
13     }else if(frek > 17){
14         letter_C();
15         number_zero();
16         signal_plus();
17     }else{
18         letter_C();
19         number_zero();
20     }
21
22 }

```

A függvényünk void, azaz nincs visszatérési értéke, paramétere amit megadunk az a frekvencia, amelyet a *main.ino* fájlban számolunk ki. C_0 -tól B_6 -ig, azaz 48 hangot tud a gitár hangolónk érzékelni. Példaként a C_0 -s hangot mutatjuk be, a folyamat azonos mindegyik hangnál.

A **frequency_selector** függvényünk if-es szerkezetek halmaza. Megnézzük, hogy a frekvencia egy adott tartományon belül van, példánkban a C_0 -s hangot keressük, ezért a táblázatból kinézve ezeket az adatokat adjuk meg a frekvenciatartománynak. Példánkban ez most a 14 és 17,75 Hz. Ezután belépünk még egy if-es szerkezetbe, amellyel azt tudjuk megadni, hogy a kijelölt hang frekvenciájától magasabbra, vagy alacsonyabbra vagyunk. Nézzük végig a példában lévő kódot. Megnézzük, hogy a frekvencia 16 Hz alatt van, ha igen, akkor írjuk ki, hogy C, 0, majd hogy -, azaz a kijelölt, pontos C_0 -s hangtól kicsit alacsonyabban vagyunk. Ilyenkor a kijelzőn ez az adat fog megjelenni: C_0^- . Ha ez nem igaz, akkor nézzük meg, hogy a frekvenciánk nagyobb-e, mint 17,

ha igen akkor C, 0, és + jelet írjuk ki, azaz C_0^+ . Ha ez két állítás közül semelyik sem igaz (tehát 16 és 17 Hz között vagyunk), akkor mondhatjuk, hogy a kijelölt, pontos C_0 -s hanghoz közel állunk, tehát nem kell - vagy + jel, ilyenkor a kijelzőnkön egyszerűen C_0 szerepel.

Példánkban a C_0 -s hangot mutattuk be, de a folyamat mindegyik hangnál azonos. Mindig megnézzük, hogy egy frekvenciatartományon belül vagyunk-e, utána pedig megállapítjuk milyen közel vagyunk a pontos frekvenciától. Innen jön ki az 1600 sor, mely ebben a header fájlban van. Nem logika, nem számolás, hanem a megjelenítési karakterek, és kiírása, kiválasztása a hangnak. Bemutatva ezt a részét a projektünknek remélem érthető, miért szedtük szét 2 fájlra a kódunkat.

A kódot, és az áramkört, LCD bekötést *githubra* feltettük, hogy bárki megtudja építeni és működtetni Arduino segítségével ezt a gitár hangolót. Irodalomjegyzékben a githubos link elérhető.

7. Mérési eredmények

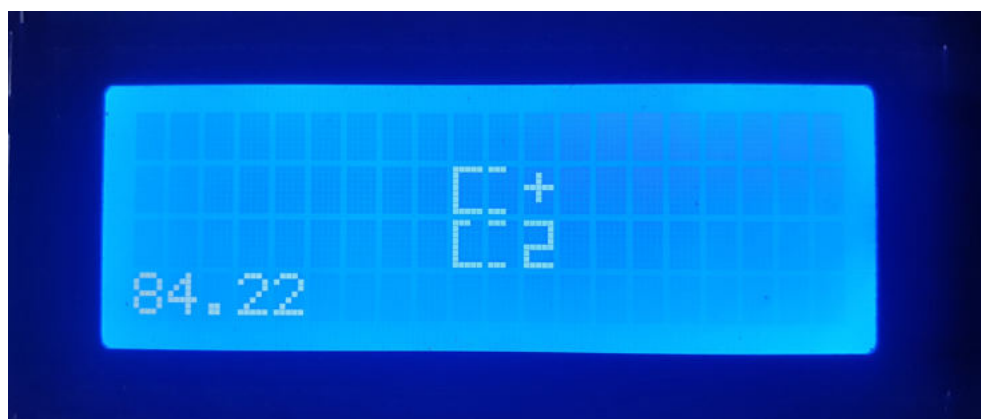
Eszközünk minőségét végül az adja meg, hogy mennyire lett pontos. Ennek megállapításához oszcilloszkópot és egy külső hangkártyát használtunk. Az oszcilloszkóp mintavételezési frekvenciája $2GSa/s$, a hangkártyáé pedig $172kHz$ volt. A mérés az alábbi módon zajlott. A gitárt behangoltuk E standard hangolásra az Amplitube 5 szoftver és a hangkártya használatával. Az így behangolt gitár húrjainak frekvenciáját összehasonlítottuk az oszcilloszkópon kapott értékekkel és a mi gitár hangolónk által adott értékekkel. Következő ábrákon szemléltetjük a mérési eredményeket.



18. ábra: E_2 -s hang az Amplitube 5 szoftverrel.



19. ábra: E_2 -s hang oszcilloszkópon mérve.



20. ábra: E_2 -s hang gitár hangolón mérve.

Jól látható, hogy mély hangoknál a mért érték a gitár hangolón közel megegyezik az oszcilloszkópon és az szoftveren mért értékekkel. Összefoglaltuk egy táblázatba E standard hangolásban lévő hangokat, ebben a hangolásban mind a magas és mély hangok szerepelnek. Következtetést tudunk vonni, hogy mennyire pontos a mi hangolónk az oszcilloszkópon és a hangkártyán mért értékekhez képest.

Hang	SPN érték	Hangkártya (referencia)	Oscilloszkóp	Gitár hangoló
E_2	82,407 Hz	82,31 Hz	82,515 Hz	84, 22 Hz
A_2	110 Hz	110,03 Hz	110,40 Hz	111,93 Hz
D_3	146,83 Hz	146,73 Hz	146,97 Hz	148,94 Hz
G_3	196 Hz	196,17 Hz	194,21 Hz	197,67 Hz
B_3	246,94 Hz	246,81 Hz	247,22 Hz	245,64 Hz
E_4	329,63 Hz	329,68 Hz	325,2 Hz	333,56 Hz

Az alábbi következtetéseket vonhatjuk le: a hangkártya amely zenei körökben elterjedt és elfogadott műszer is téved, amelyet az oscilloszkóp mérési adataiból láthatunk. Magas hangokhoz közeledve a tévedési arány egyre magasabb. Mivel, hogy nem minden zenésznek áll rendelkezésre egy oscilloszkóp kimondhatjuk azt, hogy a hangkártyás hangoláshoz képest a mi hangolónk tökéletesen kielégíti azt a szerepet, hogy otthoni gyakorlásra egy gitárt behangoljunk. Természetesen ezt fellépésekre nem ajánljuk, de ez betudható az Arduino mintavételezésének, más mikrovezérlőt használva, amelynek pontosabb a mintavételezési frekvenciája, pontosabb eszközt tudtunk volna készíteni.

8. Összefoglalás

A dolgozatunk célja a hangoló pedál megépítése volt. Ehhez az elektronikai és programozási tudásunkat használtuk fel. Kipróbáltuk éles helyzetben az egyetemen tanultakat, és sikerült megalkotnunk egy olyan eszközt amelyet otthoni használatra bátran merünk ajánlani és használatba is fogjuk állítani.

A piacon elérhető eszközökhöz képest sikerült olcsóbban egy ilyen hangolót megvalósítanunk, összesen 8600 Ft-ból. A konkurens hangolók ára 12-64000 Ft-ig terjed.

Aki ezt a hangolót megszeretné építené és üzembe helyezni, esetleg tovább fejleszteni, annak ajánljuk a githubos felületünket, és ezt a dokumentumot amelyben minden hasznos információ fellelhető.

9. Irodalomjegyzék

[1] Borbély Gábor: *Elektronika II. A műveleti erősítő és kapcsolástechnikája*
Széchenyi István Egyetem, 2006.

[2] Szabadi Tibor: *Arduino frekvencia felismerő elektromos gitárokhoz*
Github, 2021

[3] Arduino: *Arduino Documentation*

[4] Texas Instruments: *TL072 datasheet*