UNIVERSITATEA „SAPIENTIA” DIN CLUJ-NAPOCA

FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,

TÎRGU-MUREȘ

SPECIALIZAREA CALCULATOARE

PRELUCRAREA DIGITALĂ A SEMNALELOR CU PLĂCI DE DEZVOLTARE NUCLEO

PROIECT DE DIPLOMĂ

Coordonator științific:

Ș.l. dr.ing. Fehér Áron

Absolvent:

Horváth Levente

2023

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Specializarea: **Calculatoare**

Viza facultății:

LUCRARE DE DIPLOMĂ

Coordonator științific:	Candidat: **Horváth Levente**
Ș.l. dr. ing. Fehér Áron	Anul absolvirii: **2023**

a) Tema lucrării de licență:
PRELUCRAREA DIGITALĂ A SEMNALELOR CU PLĂCI DE DEZVOLTARE NUCLEO

b) Problemele principale tratate:
- Studiul efectelor digitale clasice de audio
- Modelarea efectelor digitale folosite in procesoarele de audio pentru chitare
- Simularea efectelor în Python
- Proiectarea unui circuit imprimat pentru eșantionare și generare semnal audio
- Testarea efectelor cu ajutorul circuitului pe placă de dezvoltare NUCLEO-G491

c) Desene obligatorii:
- Schema circuitului imprimat
- Layout-ul circuitului imprimat
- Diagramele în domeniul timp și frecvență a efectelor

d) Softuri obligatorii:
-Script Python pentru efectele tip distortion, noise gate, equalizer
-Firmware C/C++ pentru efectele tip distortion, noise gate, equalizer

e) Bibliografia recomandată:
- John G. Proakis, Dimitris G. Manolakis. “Digital Signal Processing: Principles, Algorithms, and applications” 3rd edition, 1995.
- Alan V. Oppenheim, Ronald Schafer – Discrete-Time Signal Processing, 3rd edition, Pearson Education, 2011

f) Termene obligatorii de consultații: săptămânal
g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Primit tema la data de: 31.03.2022
Termen de predare: 27.06.2023

Semnătura Director Departament

Semnătura coordonatorului

Semnătura responsabilului programului de studiu

Semnătura candidatului
Horváth

Declarație

Subsemnata/ul Horváth Levente, absolvent(ă) al/a specializării Calculatoare, promoția 2023 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, Târgu - Mureș

Data: 28.06.2023.

Absolvent

Semnătura Horváth

Prelucrarea digitală a semnalelor cu plăci de dezvoltare NUCLEO

Extras

Scopul acestei lucrări este de a implementa o unitate de efecte sonore digitale cu costuri reduse și ușor de accesat. O unitate digitală de efecte sonore este un dispozitiv care poate modifica sunetul unui semnal de intrare, cum ar fi o chitară sau un microfon, prin aplicarea diferitelor efecte, ca de exemplu distorsiunea, întârzierea, reverberația etc. Aceste efecte pot spori expresia muzicală și creativitatea interpretului, precum și calitatea și diversitatea sunetului. Cu toate acestea, majoritatea unităților de efecte sonore digitale sunt fie costisitoare, fie complexe, fie necesită echipamente suplimentare. Prin urmare, este nevoie de o unitate de efecte sonore digitale simplă, accesibilă și de sine stătătoare.

Proiectul presupune studierea teoriei procesării digitale a semnalelor, proiectarea și dezvoltarea de algoritmi și implementarea hardware. Teoria prelucrării semnalelor digitale reprezintă fundamentul matematic și științific pentru înțelegerea modului în care semnalele sonore pot fi manipulate de sistemele digitale. Proiectarea și dezvoltarea algoritmilor reprezintă procesul de creare și testare a metodelor de implementare a diferitelor efecte sonore cu ajutorul codului software. Implementarea hardware este procesul de construire și integrare a componentelor fizice necesare pentru a rula codul software și pentru a realiza interfața cu dispozitivele de intrare și ieșire.

În prima fază, efectele studiate au fost testate sub Python cu fișiere de sunet de chitară brute preeșantionate. Python este un limbaj de programare popular și versatil pentru prelucrarea digitală a semnalelor. Fișierele de sunet de chitară brute preeșantionate sunt înregistrări audio ale sunetelor de chitară fără efecte. Acestea sunt utilizate ca semnale de intrare pentru testarea algoritmilor.

În cea de-a doua fază, algoritmii au fost testați pe un sistem multimedia dezvoltat în mod personal cu un microcontroler STM32G491, folosind semnale generate artificial. Un sistem multimedia este un sistem care poate gestiona diferite tipuri de medii, cum ar fi audio, video etc. Un microcontroler STM32G491 este un cip de calculator mic și puternic care poate procesa semnale digitale, poate controla dispozitive de intrare și ieșire, poate comunica cu alte dispozitive etc. Semnalele generate artificial sunt semnale sonore care sunt create prin formule matematice

sau programe software. Acestea sunt utilizate ca semnale de intrare pentru testarea sistemului hardware. Calitatea efectelor a fost testată folosind atât metode obiective, cât și subiective. Metodele obiective se bazează pe măsurători cantitative și pe analiza semnalelor de ieșire. Metodele subiective se bazează pe feedback-ul și evaluarea calitativă din partea ascultătorilor umani.

Cuvinte cheie: procesarea digitală a semnalelor, efecte sonore, programare ARM, simulare Python

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM

MAROSVÁSÁRHELYI KAR

SZÁMÍTÁSTECHNIKA SZAK

DIGITÁLIS JELFELDOLGOZÁS NUCLEO LAPOKKAL

DIPLOMADOLGOZAT

Témavezető:

Dr. Fehér Áron,

egyetemi adjunktus

Végzős hallgató:

Horváth Levente

2023

Kivonat

A dolgozat célja egy alacsony költségű, könnyen hozzáférhető digitális hangeffektus egység megvalósítása. A projekt magába foglalja a digitális jelfeldolgozási elmélet tanulmányozását, algoritmusok tervezését és fejlesztését, valamint hardveres megvalósítását. Első fázisban a tanulmányozott effektusokat Python alatt teszteltük előre mintavételezett nyers gitár hangállományokkal. Második fázisban STM32G491 mikrovezérlőt tartalmazó saját fejlesztésű multimédia rendszeren teszteltük az algoritmusokat mesterségesen előállított és hangszeren szintetizált jelek segítségével. Az effektusok minőségét mind objektív, mind szubjektív módszerekkel teszteltük.

Kulcsszavak: digitális jelfeldolgozás, hang effektusok, ARM programozás, Python szimuláció

Abstract

The aim of this work is to implement a low cost, easily accessible digital sound effect unit. The project involves the study of digital signal processing theory, algorithm design and development, and hardware implementation. In the first phase, the studied effects were tested under Python with pre-sampled raw guitar sound files. In the second phase, the algorithms were tested on a self-developed multimedia system with an STM32G491 microcontroller using artificially generated and synthesized signals. The quality of the effects was tested using both objective and subjective methods.

Keywords: digital signal processing, sound effects, ARM programming, Python simulation

Tartalomjegyzék

Ábrák jegyzéke

1. Bevezető

1.1. A digitális jelfeldolgozás alapjai

A digitális jelfeldolgozás(Digital Signal Processing/DSP) egy olyan technika, amelyet diszkrét jelek, például hang, kép és adat feldolgozására és elemzésére használnak. Az így feldolgozott digitális jelek számsorozatok, amelyek egy folytonos változó mintáit reprezentálják egy olyan tartományban, mint az idő, a tér vagy a frekvencia. A DSP alkalmazásai közé tartozik többek között a hang- és beszédfeldolgozás, képfeldolgozás, adattömörítés, videó- és hangkódolás, biológiai jelek feldolgozása és a szeizmológia is [1]. Mindezen lehetőségek közül én a hangfeldolgozás irányában indultam el.

1.2. A digitális jelfeldolgozás történelme

A digitális jelfeldolgozás történelme az 1960-as és 1970-es évekre vezethető vissza, amikor a digitális számítógépek megjelenése lehetővé tette, hogy a digitális jeleken bonyolult matematikai műveleteket hajtsanak végre. A DSP egyik legkorábbi alkalmazása a digitális szűrés volt, amely a digitális jelek digitális szűrőkkel történő feldolgozását jelentette a nem kívánt zaj és torzítás eltávolítása érdekében. Ezt a technikát először a távközlési iparban alkalmazták, ahol a digitális hangátvitel minőségének javítására használták [2],[3].

Az 1980-as években a DSP-t egyre szélesebb körben kezdték alkalmazni, többek között a zeneiparban is. A zenét mindig analóg formában rögzítették és továbbították egészen a ’80-as évekig, amikor a CD-lejátszó általánossá tette a zene digitális rögzítését. Amikor a stúdióban CD-t készítenek, a zenét először egy mikrofon elektromos analóg jellé alakítja át, majd az elektromos jelet mintavételezéssel (a hang intenzitásának mérése meghatározott időpontokban, másodpercenként sok ezer alkalommal) és kvantálással (minden egyes intenzitásnak egy véges számú intenzitási szinthez való hozzárendelése) nullák és egyesek sorozatává alakítják át. Ez a nullák és egyesek sorozata az, ami a CD spirálsávjába van vésve [4]. Ezekben az években jelentek meg a digitális gitárpedálok is, amelyek digitális jelfeldolgozó (DSP) chipeket használtak az effektusok széles skálájának előállításához. Ezek a pedálok gyakran sokoldalúbbak voltak, mint analóg társaik, és lehetővé tették a paraméterek széles körének pontos szabályozását [5].

1.3. Effektusok és történelmük

Az első elektromos gitárok az 1930-as évek elején jelentek meg, ekkor még a swing korszak uralkodott, ahol a kürtösök voltak a sztárok, a gitárosok pedig szerettek volna szintén reflektorfény elé kerülni, azonban a korai erősített gitárok természetes hangzása vékony, rekedtes és elég antiklimatikus volt. A figyelem felkeltése érdekében gyorsan elkezdték felturbózni az unalmas hangzást, a ’30-as években például a legelső gitáreffektet magába a gitárba építette be a Rickenbacker, mely motoros csigákkal rázta a gitárnyakat, ezzel vibráló effektust létrehozva. Az ’50-es évekre már több erősítő is rendelkezett beépített tremolo, vibrato, visszhang és reverb effektekkel. A korai gitáreffektusok vákumcsövekkel működtek, ezek terjedelmesek, drágák, törékenyek voltak, tehát nem túl praktikusak élő játékhoz. A ’60-as években végül megjelent a tranzisztor, a mérnökök pedig képesek voltak megfizethető, standalone effekteket létrehozni és a ’70-es és ’80-as évektől kezdve már a DSP is kezdett beleszólni a dolgokba, ezáltal is növelve az effektusok népszerűségét [6].

Ma már a gitárpedálok széles skálája áll rendelkezésre, beleértve a torzítást, az overdriveot, a késleltetést, a visszhangot, a kórust és sok mást. Ezek a pedálok két fő kategóriába sorolhatók: kemény (hard) és lágy (soft) effektusok.

A kemény effektusok olyan pedálok, amelyek egy adott hatást hoznak létre, amely a pedál áramkörébe van bekötve. A kemény effektusok közé tartozik például a torzítás, a fuzz és a wah, ezért voltak annyira elterjedtek ezek a ’60-as években, mert nem volt feltétlenül szükség még DSP-re, sokszor elég volt az elektroncsöves erősítőkön felhúzni a hangerőt a torzított hangzáshoz.

A lágy effektusok ezzel szemben olyan pedálok, amelyek a DSP technológiát használják, hogy a felhasználó által programozható és testre szabható hangzás széles skáláját hozzák létre [7].

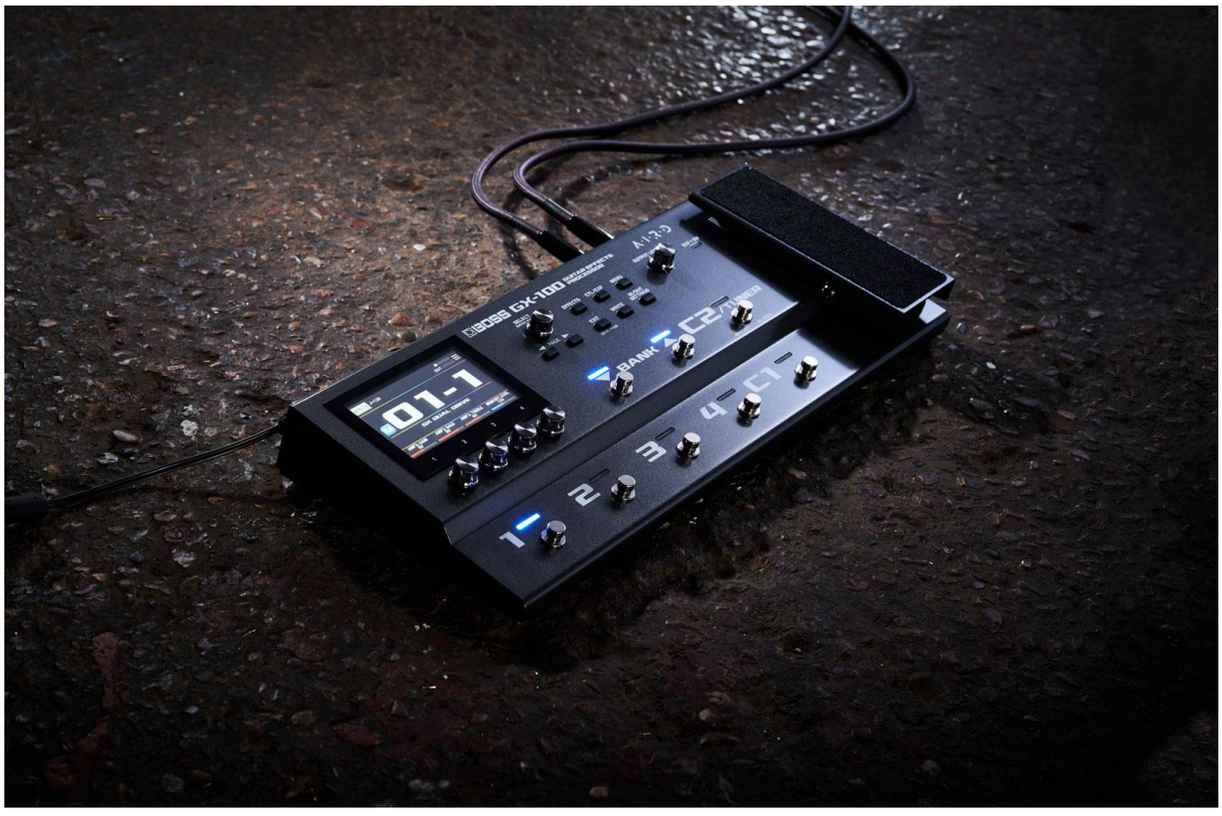
1.4. Célkitűzés

Azért indultam el a hangfeldolgozás irányában, mert én magam is zenélek és produceri munkával is foglalkozom hobbi szinten, ezért az a célom, hogy egy olyan egységet hozzak létre, amely valós időben működik, jó minőségű, és lehetővé teszi a zeneszerzők, zenészek és hangmérnökök számára, hogy kreatívabb módon dolgozzanak a hangzásukkal, anélkül, hogy nagy költségeket kellene ráfordítaniuk professzionális hangfeldolgozási felszerelésekre.

Munkám során az alábbi effektusok megvalósításával és tanulmányozásával foglalkoztam:

- EQ
- Torzítás
- Overdrive
- Késleltetés és konvolúciós téreffektus
- Noisegate

Természetesen ezek az effektusok már tökéletesítve vannak nagy cégek által, mint például a Boss, Digitech, Vox, Zoom és egyebek, viszont a nagy hírnév nagy árat is jelent. Ezzel szemben én szeretnék olcsón megvalósítható effektusokat létrehozni, amiket akár testre is szabhatok. Az alábbi pár példában bemutatok 4 multieffekt pedált, melyek az én megvalósításomon kívül még számos effektust tartalmaznak:



1. ábra: Boss GX-100

A Boss GX-100 fontosabb jellemzői:

-23 erősítőtípussal rendelkezik

-Több, mint 150 Boss effektust valósít meg, ezek nagyon sokszínű hangzást képesek biztosítani

-Expression pedál (valós idejű effektusok modulálása, pl. hangerő, wah effektus, hangmagasság)

-A hangokat akár szerkeszteni is lehet a BOSS Tone Studio programmal számítógépről.

-Akár két lábpedált is hozzá lehet kötni a bővített vezérléshez.

Piaci ára Romániában 2700 RON körül mozog.

2. ábra: Digitech RP360XP

Digitech RP360XP:

-Több, mint 160 effektussal rendelkezik

-40 másodperces looper

-Beépített dobgép

-Expression pedál

-Csatlakoztatható számítógéphez, könnyen lehet rögzíteni hangfájlokat egy USB-n keresztül

Piaci ára Romániában 1000-1200 RON között mozog.

3. ábra: VOX Stomplab 2G

VOX Stomplab 2G:

-104 modellező effektus, mely magába foglal erősítő modelleket is és cabinet modelleket (ezt nem tudom hogyan fordítsam le magyarra), emellé 20 felhasználói mentés, hogy eltárolhassuk az egyedi hangzást

-100 preset program, vagyis előre beállított hangzások

-Expression pedál

-Piaci ára Romániában körülbelül 400 RON

4. ábra: Zoom G1 Four

Zoom G1 Four:

-60 beépített effektus

-13 erősítő típus

-Egyszerre 5 effektust lehet sorba kötni

-Looper, ami 30 másodpercet képes felvenni

-Beépített dobgép, mely összeköthető a looperrel

-Csatlakoztatni lehet számítógéphez és az ezen futtatható Zoom Guitar Labhoz

-Piaci ára Romániában körülbelül 470 RON

Ezen multieffekt pedáloknak gyártói rengeteg tapasztalattal rendelkeznek és garantáltan nagy minőségű termékeket adnak ki, azonban nem mindenki engedheti meg magának, hogy ennyi pénzt költsön rájuk. Ezzel szemben az én megvalósításom nem von magába több fizikai komponenst, mint a mikrovezérlőt, a rajta levő nyákot, illetve egy számítógépet egy micro-USB kábellel.

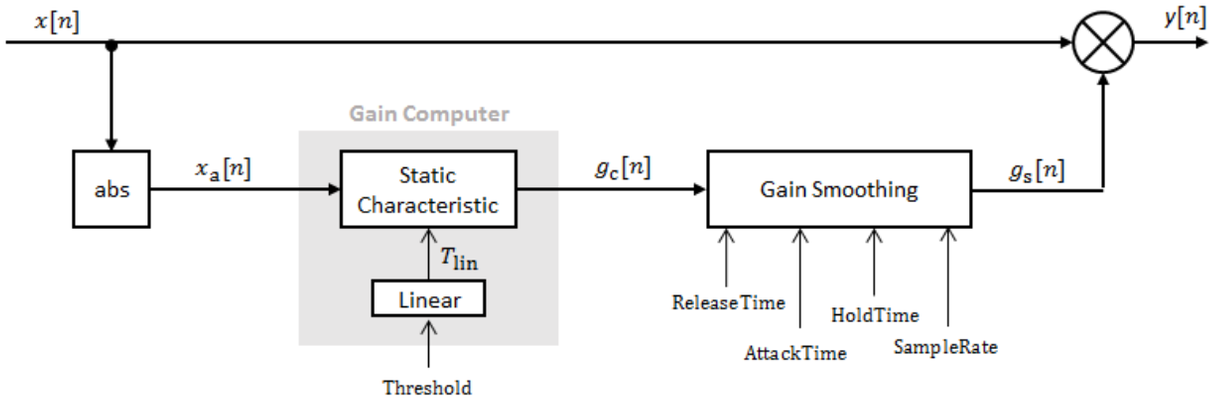
2. Hangeffektusok elmélete

2.1. Noisegate

A noisegate egy dinamikus hangsáv szűrőt valósít meg. A hangforrásból érkező jelet összehasonlítjuk egy küszöbértékkel, amelynek alapján eldöntjük, hogy a jel zaj-e, vagy hasznos tartalommal bír. A zaj szűréséhez a jel erősítését szabályozzuk, míg a hasznos tartalom áthalad a szűrőn anélkül, hogy a jelt erősítenénk vagy gyengítenénk.

Az alábbi ábrán láthatóak a megvalósítás lépései, az első lépés a bemeneti x[n] jel nagyságrendű jellé való alakítása, ez után következik az a fázis, mely a jelet csak egy bizonyos *threshold* küszöbérték felett engedi át a jelet, itt a noisegate aktiválási függvénye (mely az ábrán a Gain computer) a következőképpen fog értéket kapni:

$$Gc[n] = \begin{cases} 1, ha\ |x[n]| \geq threshold \\ 0, ha\ |x[n]| < threshold \end{cases} \quad (1)$$



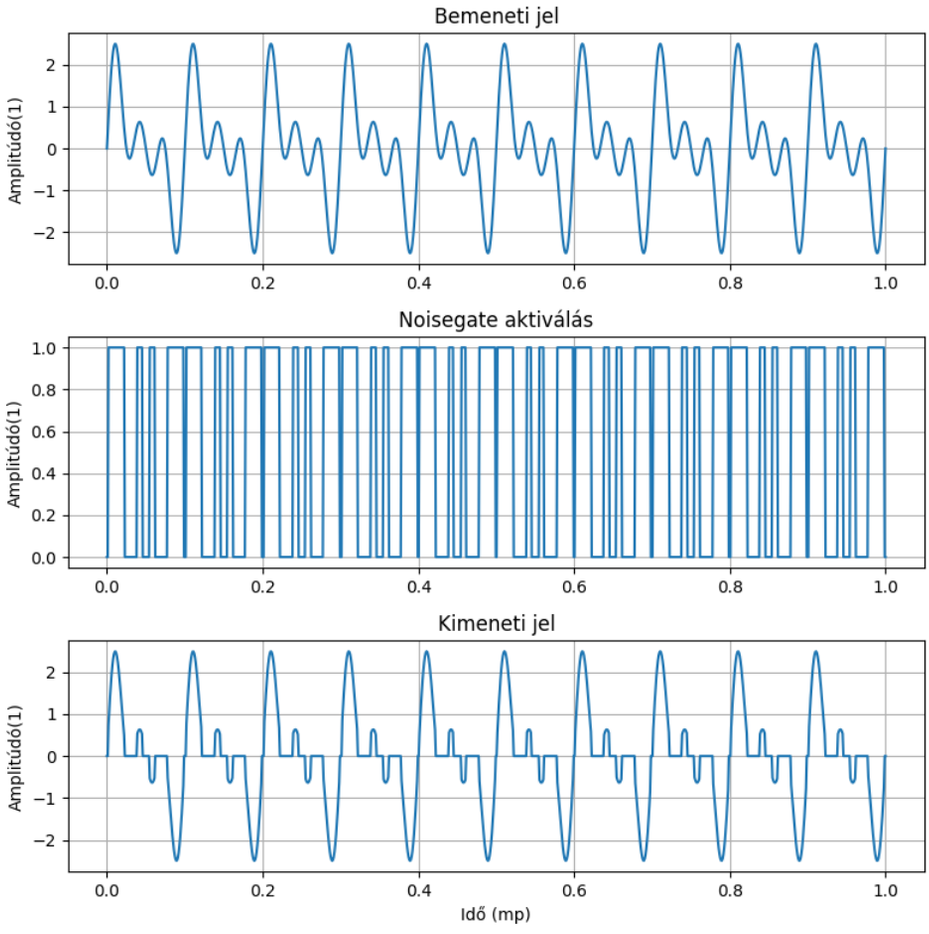
5. ábra: noisegate tömbvázlata

A kiszámított erősítést az attack, és release (melyeket nyitási és zárási paramétereknek fordítottam le), illetve a hold (tartási) fázisokban simítjuk el. Mindhárom paramétert miliszekundumban adják meg a legtöbbször.

Az *attack* azt határozza meg, hogy a kapu mennyi idő alatt vált a zárt állapotból teljesen nyitott állapotba.

A *release* azt határozza meg, hogy a kapu mennyi idő alatt változik nyitott állapotból teljesen zárt állapotba. Egy gyors zárás hirtelen elvágja a hangot, míg a lassabb kioldás egyenletesen csillapítja a jelet a nyitástól a zárásig, mely sokkal kellemesebb hangzást biztosít.

A *hold* azt határozza meg, hogy a kapu mennyi ideig maradjon teljesen nyitva, miután a jel a küszöbérték alá esik, és mielőtt a zárás megkezdődik. A holdot gyakran azért állítják be, hogy a kapu ne záródjon olyan rövid szünetek alatt, amikor épp nincs gitárjáték.



6. ábra: noisegate aktiválási függvény szinusz jelre

1. A 6. ábra első ábráján az bemeneti jel látható, amely három különböző szinusz jel összetételéből áll.
2. A második ábrán a noisegate aktiválási pontjai láthatóak, amelynek értéke 0, ha a bemeneti jel abszolút értéke a küszöbérték alatt van, 1 pedig, ha a küszöbérték felett van.
3. Az utolsó ábrán pedig a kimeneti jel látható, amelyet a bemeneti jel és a noisegate függvény szorzata ad ki. Azok a részek, ahol a bemeneti jel abszolút értéke a küszöbérték alatt maradt nullázódtak, míg azok, amelyek a küszöbérték felett voltak, az eredeti jel alakját őrzik.

Ha a n idő nagyobb, mint a küszöbérték, és az előző időpillanatban Gc értéke kisebb volt, akkor az állapotátmenet az attack fázisban történik:

$$z[n] = \alpha_a \cdot z[n-1] + (1 - \alpha_a) \cdot Gc[n] \text{ (2)}$$

ahol z[n] a jelenlegi állapot, és αa az attack időállandója.

Ha a n idő kisebb, vagy egyenlő, mint a küszöbérték, akkor az állapotátmenet az álló fázisban történik:

$$z[n] = z[n-1] \text{ (3)}$$

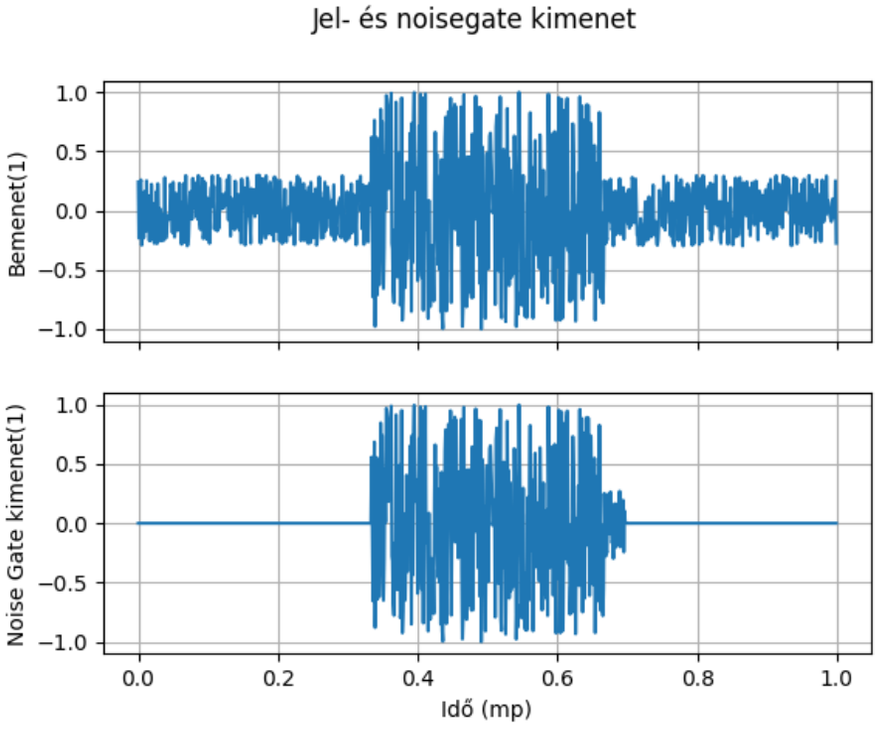
Ha Gc értéke nagyobb, mint az előző időpillanatban, akkor az állapotátmenet a release fázisban történik:

$$z[n] = \alpha_r \cdot z[n-1] + (1 - \alpha_r) \cdot Gc[n] \text{ (4)}$$

ahol z[n] a jelenlegi állapot, és αr az release időállandója.

A kimeneti jel y[n] a bemeneti jel és az állapotszámolt érték szorzata:

$$y[n] = x[n] \cdot z[n] \text{ (5)}$$



7. ábra: noisegate alkalmazása 44.1kHz-es mintavételezési frekvenciával

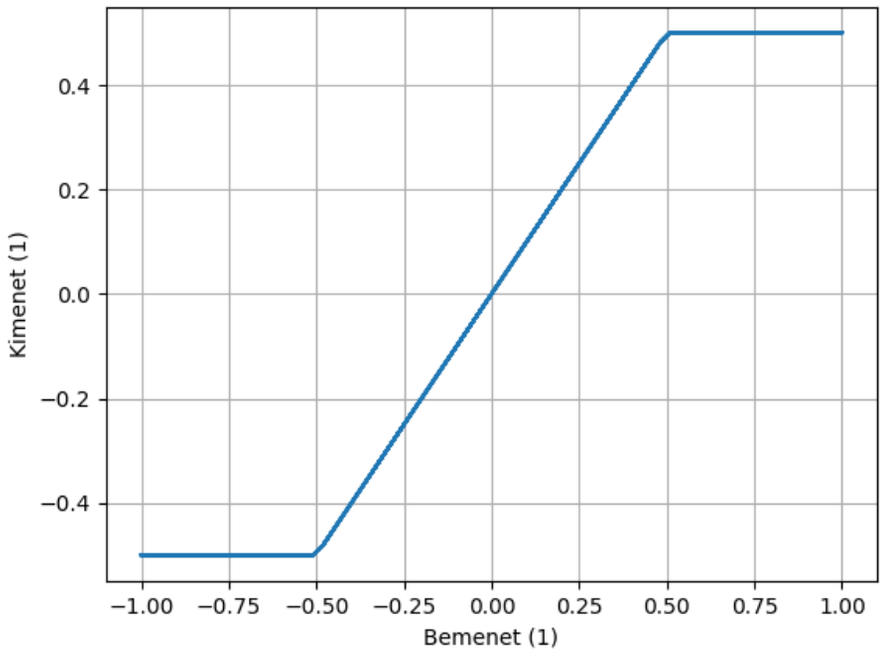
2.2. Overdrive

Az overdrive egy torzított hangjel hatást kelt, viszont kontrolláltabb, mint a torzítás, mint effektus, erről lennebb beszélek. Egy overdrive effektus esetében lehetőségünk van arra, hogy a torzítási szintet a saját igényeinkre szabjuk, és így a torzult hangzást még mindig kellemesnek és zenének tűnőnek halljuk [8].

3 fajta overdriveot valósítok meg: hard clippingből egyet és soft clippingből kettőt.

- Hard clipping
    - Az overdrive ezen típusa a kimeneti szint korlátozásával működik, amihez a bemeneti jelet egy erősítési tényezővel szorozzuk, majd normalizáljuk -1 és 1 között. A normalizálásra azért van szükség, hogy a véges felbontással rendelkező egységek könnyen tudják kezelni az értékeket.

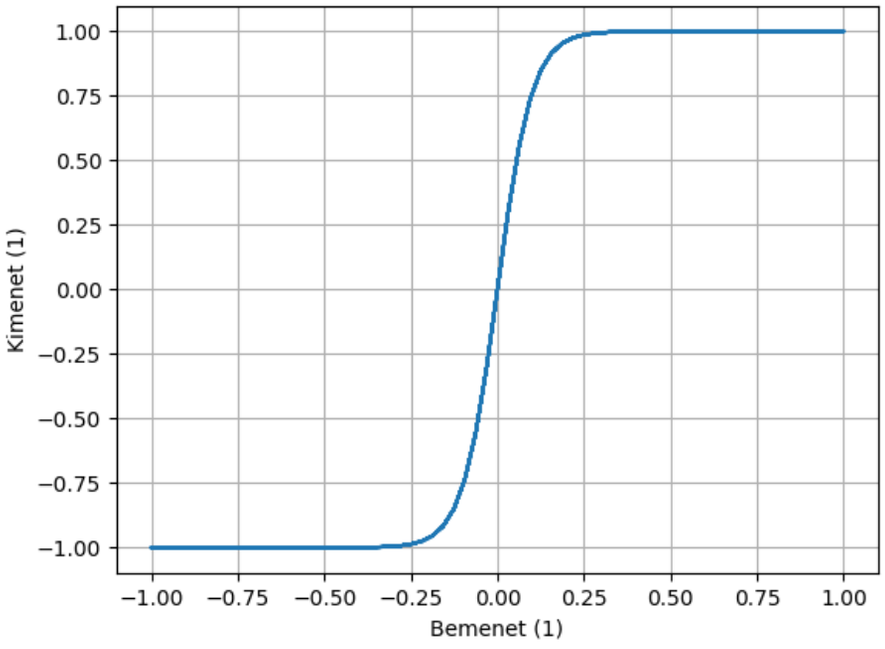
$$y[n] = \begin{cases} g \cdot x[n], ha\ |g \cdot x[n]| < threshold \\ sign(x[n]), másképpen \end{cases} \quad (6)$$



8. ábra: overdrive hard clipping be-/kimeneti függvény

- Soft clipping (1)
    - Ezen típusú overdrive esetében a bemeneti jelre egy nem lineáris függvényt alkalmazunk, amely "puhítja" az erősített és levágott jelek közötti átmenetet. Az egyenletben alkalmazott függvény a hiperbolikus tangens (tanh), amelynek sima és fokozatos az átmenete a lineáris és a nem lineáris tartomány között.

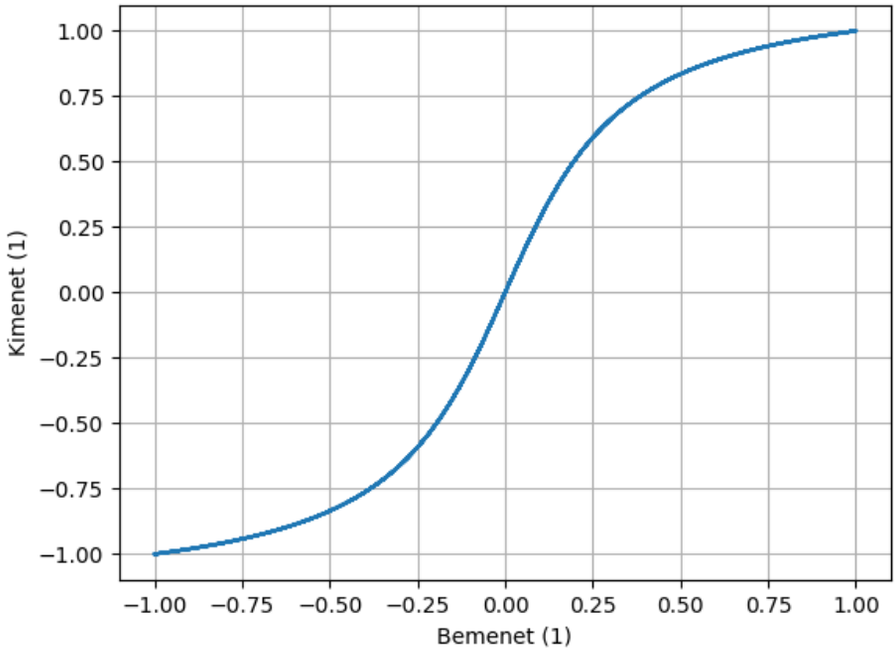
$$y[n] = \tanh(gain \cdot x[n]) \quad (7)$$



9. ábra: overdrive soft clipping (1 - tanh) be-/kimeneti függvény

- Soft clipping (2)
    - Ez az overdrive hasonló az előző soft clipping típushoz, de egy másik nem lineáris függvényt használ a kimeneti jel formálásához. Az egyenletben alkalmazott függvény az arkusz tangens (arctan). Az arctan hasonlóan sima az átmenetben a lineáris és nem lineáris tartományok között, de a görbe alakja eltér a tanh-tól.

$$y[n] = \arctan(gain \cdot x[n]) \quad (8)$$



10. ábra: overdrive soft clipping (2 - arctan) be-/kimeneti függvény

Összefoglalva tehát, az overdrive egy olyan effektus, amely lehetővé teszi a torzítási szint kezelését, míg a torzítás általában nagyobb torzítási szinteket eredményez, amely már az átvitt információban is megváltoztatja a jelet.

2.3. Torzítás

A torzítás amit használok, egy egyszerű hatást valósít meg: gondoljunk arra, hogy egy hangfalnak adunk jelet. Ha az erősítés mértéke alacsony, akkor a hangfal szépen és tisztán fogja megszólaltatni az adott jelet. Azonban ha az erősítés túl magasra van állítva, akkor az erősítő nem tudja megfelelően kezelni a jelet, és a hangfalba küldött jel torzulni fog.

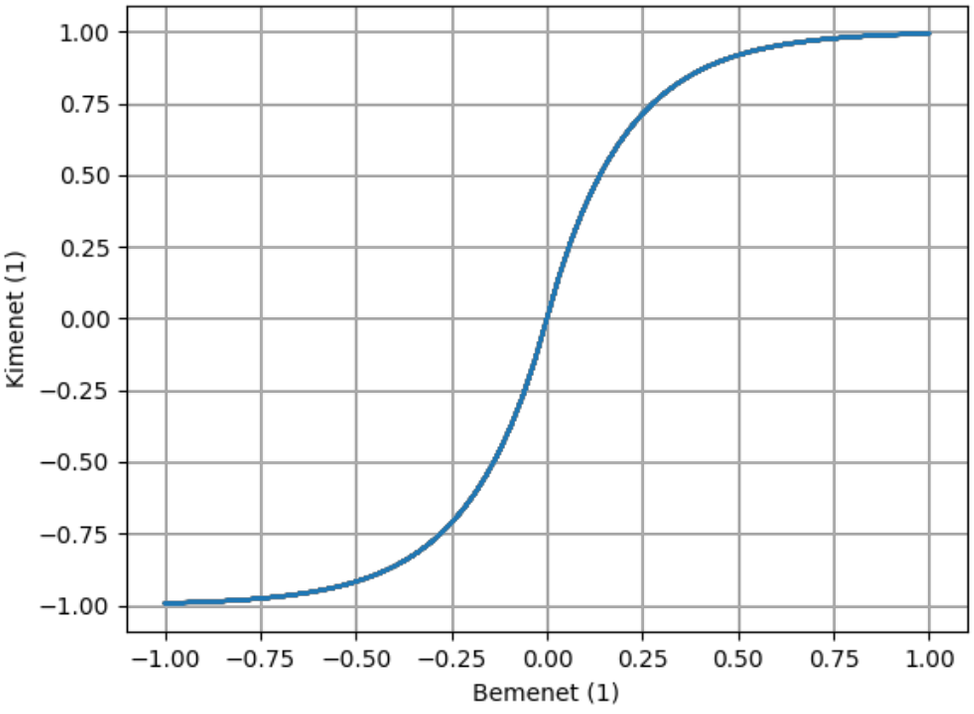
A képlet lényegében egy új jelet hoz létre, amely a bemeneti x[n] jelre alkalmazott nemlineáris transzformáció eredménye. Ekkor a kimeneti y[n] jel a következőképpen áll elő:

$$y[n] = sign(x[n]) \cdot \left(1 - e^{-|x|}\right) \quad (9)$$

ahol sign(x) a bemeneti jel előjele, vagyis $sign(x) = \begin{cases} -1, ha\ x < 0 \\ 0, ha\ x = 0; \\ 1, ha\ x > 0 \end{cases}$

az így kapott kifejezés a bemeneti jel torzított változatát adja vissza. Egy gain paraméter segítségével továbbfejleszthetjük a torzítást, melynek segítségével szabályozhatjuk a nemlinearitás mértékét.

Az alábbi grafikon egy sima, nem lineáris átmenetet mutat a bemeneti és a kimeneti jelek között, ahol a görbe alakját a bemeneti jelre alkalmazott torzítás mértéke határozza meg.



11. ábra: torzítás be-/kimeneti függvény

2.4. EQ (hangszín szabályozó)

Az EQ (Equalizer) egy olyan eszköz vagy funkció, amely a hangjelen belüli különböző frekvenciasávok hangerejének beállításáért felel. Az EQ tervezésének egyik legfontosabb lépése hogy nekünk megfelelő szűrőtípust válasszunk. Ebben az esetben egy 3 sávos Butterworth EQ-ra gondoltam. Ez maximálisan lapos frekvenciaválasszal rendelkezik az átviteli sávban, és amelyet a hangtechnikában gyakran használnak. Az IIR-szűrők más típusai, például a Chebyshev- és az elliptikus szűrők szintén gyakran használatosak EQ-khoz, és ezeknek is megvannak a maguk előnyei és hátrányai.

A Butterworth EQ-nak amit választottam az a lényege, hogy 3 különböző helyen tudja szűrni a bemeneti jelet. Van egy alul-, közép- és felüláteresztő szűrője, mindegyiknek külön meg lehet adni a fokszámát és a frekvenciát, ahol vágni szeretnénk. Ha a jel 3 pontján megtörtént a vágás, akkor ezt a 3 komponenst össze kell adni, és megkapjuk az EQ által szűrt jelet. Azt is megadhatjuk, hogy a bizonyos komponensek milyen erősítéssel rendelkezzenek, így kiválasztva, hogy a mély, közepes vagy magas hangokat szeretnénk jobban hallani.

A tervezés során tehát az első lépés, hogy kiválasszuk a megfelelő *középfrekvenciát (fc)* és *jósági tényezőt (Q)*. A *középfrekvencia* a hangspektrumban található frekvencia tartomány középpontja, melynek megnövelésével a hang közelebbinek és erősebbnek hangzik, csökkentésével pedig távolabbinak, lazábbnak, visszafogottabbnak. A *jósági tényező (Q)* meghatározza egy szűrő sáv szélességét vagy meredekségét. Ennek értéke befolyásolja, hogy a szűrő mennyire keskeny vagy széles sávot szabályoz. A magas Q-érték keskenyebb sávot eredményez, ezért kevesebb frekvenciát érint, mely egy specifikus, hangsúlyos hatást hoz létre. Az alacsony Q-érték szélesebb sávot jelent, mely több frekvenciát foglal magába [9].

Tegyük fel például, hogy egy aluláteresztő szűrőt szeretnénk tervezni egy 200Hz-es fc-vel és 0.0701-es Q-val, egy sáváteresztő szűrőt 1000Hz-es fc-vel és 1.4142-es Q-val, valamint egy felüláteresztő szűrőt 5000Hz-es fc-vel és 0.7071-es Q-val.

Ezek után meg kell határozzuk az analóg szűrő fokszámát. Tegyük fel, hogy 60dB-es csillapítást szeretnénk a zártsávban minden szűrő esetén. A Butterworth-szűrő esetében az n fokszám a következőképpen számítható ki:

$$n = \frac{log\left(\frac{10^{0.1\cdot As}-1}{10^{0.1\cdot Ap}-1}\right)}{2\cdot log\left(\frac{wp}{ws}\right)} \quad (10)$$

ahol As a kívánt zártsáv-csillapítás dB-ben, Ap az áteresztő sáv hullámossága dB-ben (normalizált Butterworth-szűrőknél 3 dB 1rad/s pontnál), wp az áteresztő sáv szélső frekvenciája és ws a zártsáv szélső frekvenciája. Tegyük fel, hogy minden szűrő esetében $\frac{wp}{ws} = 2$. Ekkor minden szűrő esetében $n = \frac{log\left(\frac{10^{0.1*60}-1}{10^{0.1*3}-1}\right)}{2\cdot log(2)}$, tehát n ≈ 5.

Ezután meghatározzuk az aktuális közép frekvenciát a pre-warping képlettel:

$$wc = \frac{2}{T} \cdot tan(\frac{wdT}{2}) \quad (11)$$

ahol wc az pre-warpolt középfrekvencia, wd a kívánt középfrekvencia és T a mintavételi idő. Tegyük fel, hogy a mintavételi frekvenciánk 44100 Hz, tehát T $= \frac{1}{44100}$ s. Ekkor az aluláteresztő

szűrőnk esetében wc $= \frac{2}{\mathrm{T}} \cdot \tan(\frac{200\mathrm{T}}{2}) \approx$ 200,6 Hz; a sáváteresztő szűrőnk esetében wc $= \frac{2}{\mathrm{T}} \cdot \tan(\frac{1000\mathrm{T}}{2}) \approx$ 1003 Hz; és a felüláteresztő szűrőnk esetében wc $= \frac{2}{\mathrm{T}} \cdot \tan(\frac{5000\mathrm{T}}{2}) \approx$ 5035 Hz.

Az aluláteresztő szűrőnk esetében az s-t $\frac{s}{wc}$-vel helyettesítjük a

$$H(s) = \frac{wc^n}{s^n + b[n-1]\cdot s^{n-1}\cdot wc + \ldots + b[1]\cdot s\cdot wc^{n-1} + wc^n} \textit{(12)}$$

átviteli függvényben, ahol b[i] a szűrő n foka és Q foka által meghatározott együtthatói.

Hasonlóan járunk el felüláteresztő és sáváteresztő szűrők esetén is, ahol a helyettesítés $\frac{wc}{s}$-sel illetve $Q\frac{s^2+w_0^2}{w_0 s}$-sel történik.

A bilineáris transzformáció alapgondolata az, hogy a H(s) átviteli függvényben az s-t a z-transzformációt reprezentáló komplex változó, a z függvényével helyettesítjük. Ez a transzformáció a teljes s-síkot a z-sík egységkörére képezi le, megőrizve az analóg szűrő frekvenciaválaszát. Az így kapott digitális szűrő H(z) átviteli függvénye a z függvényben kifejezhető.

Ha adott egy $H_a(s)$ átviteli függvény az s-tartományban, megkapjuk a $H_d(z)$ diszkrét átviteli függvényt a z-tartományban a következő helyettesítéssel:

$$s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \textit{(13)}$$

Ahol $T = \frac{1}{fs}$ a diszkrét rendszer mintavételi intervalluma, amiben fs a mintavételi sebesség. Ezért:

$$H_d(z) = H_a\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\right) \textit{(14)}$$

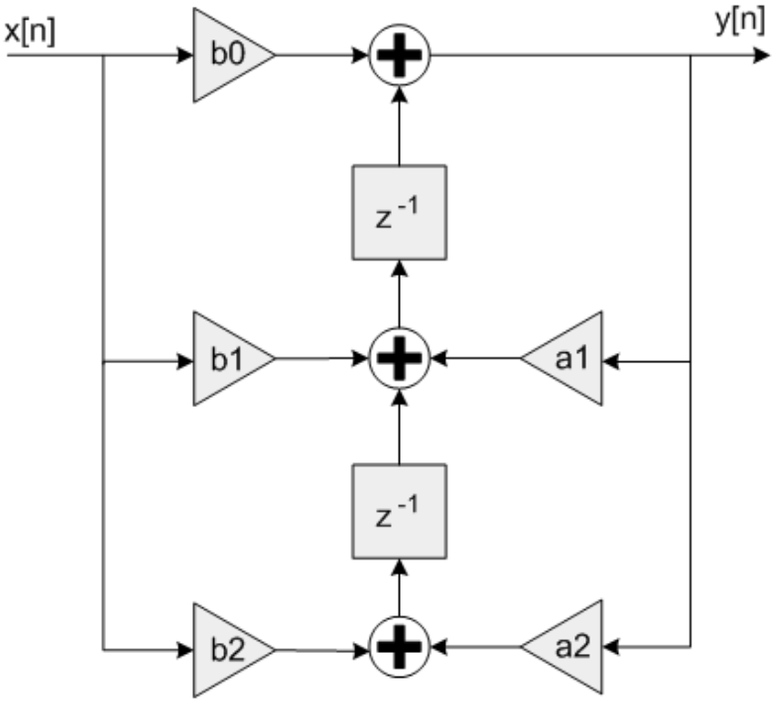
azaz

$$H(z) = wc^n \cdot \frac{(z+1)^n}{(z+1)^n + b[n-1]\cdot(z+1)^{n-1}\cdot(z-1) + \ldots + b[1]\cdot(z+1)\cdot(z-1)^{n-1} + (z-1)^n} \textit{(15)}$$

Az így kapott differenciálegyenletet standard alakba hozzuk

$$H(z) = \frac{\sum_{i=0}^{N-1} a[i]z^i}{1-\sum_{i=1}^{N-1} b[i]z^i} \textit{(16)}$$

amit a II transzponált direkt alak segítségével (DTF-2) programozhatunk le, mivel ez az alak a legrobusztusabb stabilitás és túlcsordulás szempontjából.

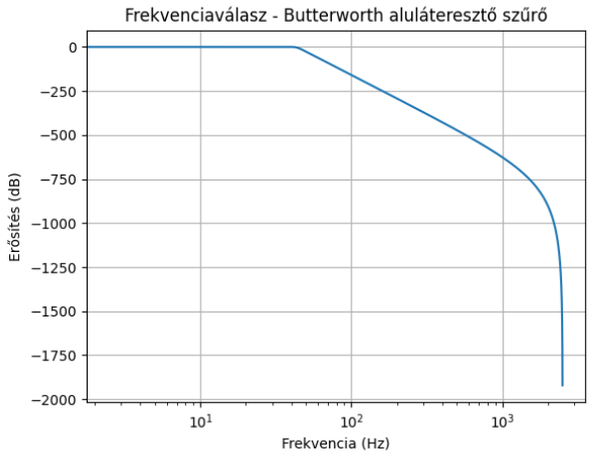


12. ábra: transzponált direkt alak (DTF-2) diagramja

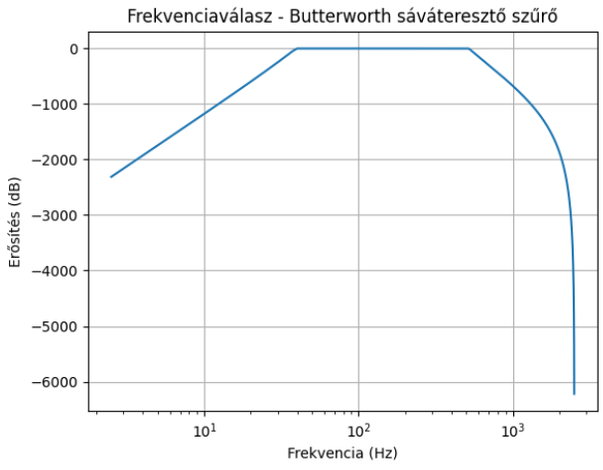
Természetesen másfajta EQ-kat is lehet használni, egy 5 sávos Butterworth EQ középfrekvenciája lehet 100 Hz, 500 Hz, 1 kHz, 5 kHz és 10 kHz, és minden sáv más-más szűrőtípust (aluláteresztő, sáváteresztő, felüláteresztő) használ a hangzás alakítására. A pontos szűrőtípusok és frekvenciák az adott alkalmazástól és a kívánt hatástól függően változhatnak.

Általában az aluláteresztő szűrőket a jel magas frekvenciájú tartalmának csillapítására, a felüláteresztő szűrőket az alacsony frekvenciájú tartalmak csillapítására, a sávszűrőket pedig a magas és az alacsony frekvenciájú tartalmak csillapítására használják, miközben középen egy frekvenciasávot engednek át.

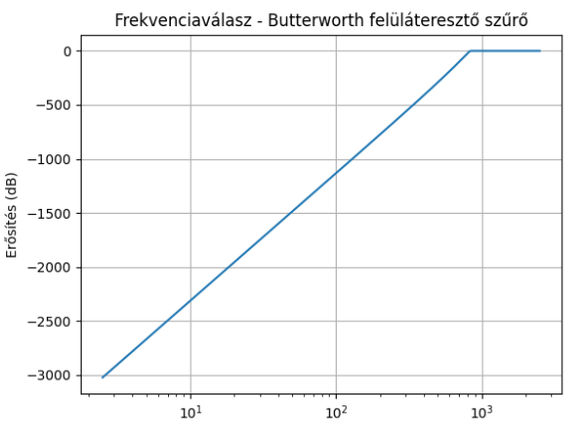
Érdemes megjegyezni, hogy a szűrők tervezésekor nem csak a középfrekvenciát, hanem a Q-tényezőt is figyelembe kell venni, amely a sávszűrő szélességét szabályozza. A magasabb Q-tényező keskenyebb sávszűrőt, míg az alacsonyabb Q-tényező szélesebb sávszűrőt eredményez. A Q-tényező megválasztása a kívánt hatástól és az alkalmazástól függ.



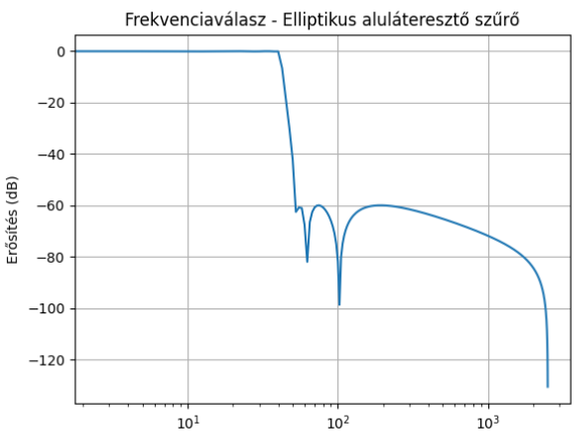
14. ábra: diszkrét aluláteresztő N=22-ed rendű szűrő



13. ábra: diszkrét sáváteresztő N=22-ed rendű szűrő



15. ábra: diszkrét felüláteresztő N=22-ed rendű szűrő

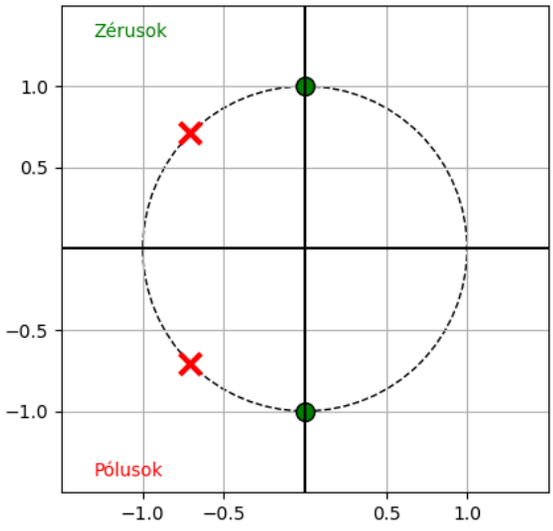


16. ábra: diszkrét aluláteresztő N=22-ed rendű szűrő (elliptikus)

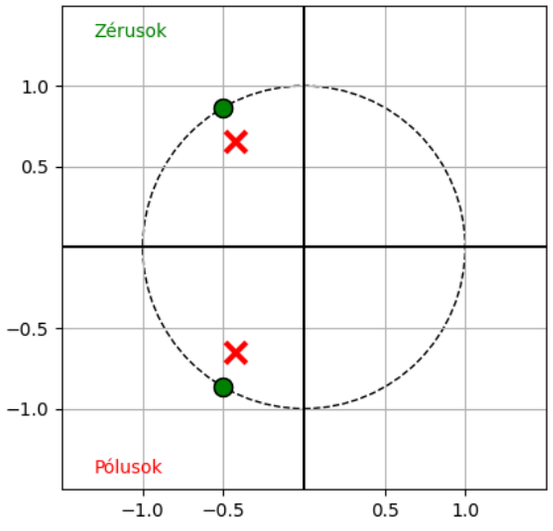
A fenti ábrákon 13-tól 15-ig a Butterworth alul-, sáv- és felüláteresztő szűrőinek frekvenciaválasza látható, míg a 16-oson az elliptukus aluláteresztő szűrő frekvenciaválasza. Mivel a sáv- és felüláteresztő szűrőket is az aluláteresztőből képezzük, a következőkben összehasonlítom a Butterworth és elliptikus aluláteresztő szűrőket.

A Butterworth szűrő egy olyan szűrő, amely a lehető legsimább frekvenciaátvitelt biztosítja az áteresztő sávon, az átviteli függvénye egyenletes, nincsenek éles átmenetek. Az elliptikus szűrő a Butterworthnél élesebb frekvenciaátvitelt és jobb átviteli csillapítást nyújt a stop-sávban, az átviteli

karakterisztika pedig az áteresztő sávon is gyorsan csökken. Az elliptikus szűrők összetettebbek és nehezebben kivitelezhetőek a pólusok helyzete miatt.



17. ábra: pólusok és zérusok Butterworth szűrő esetén

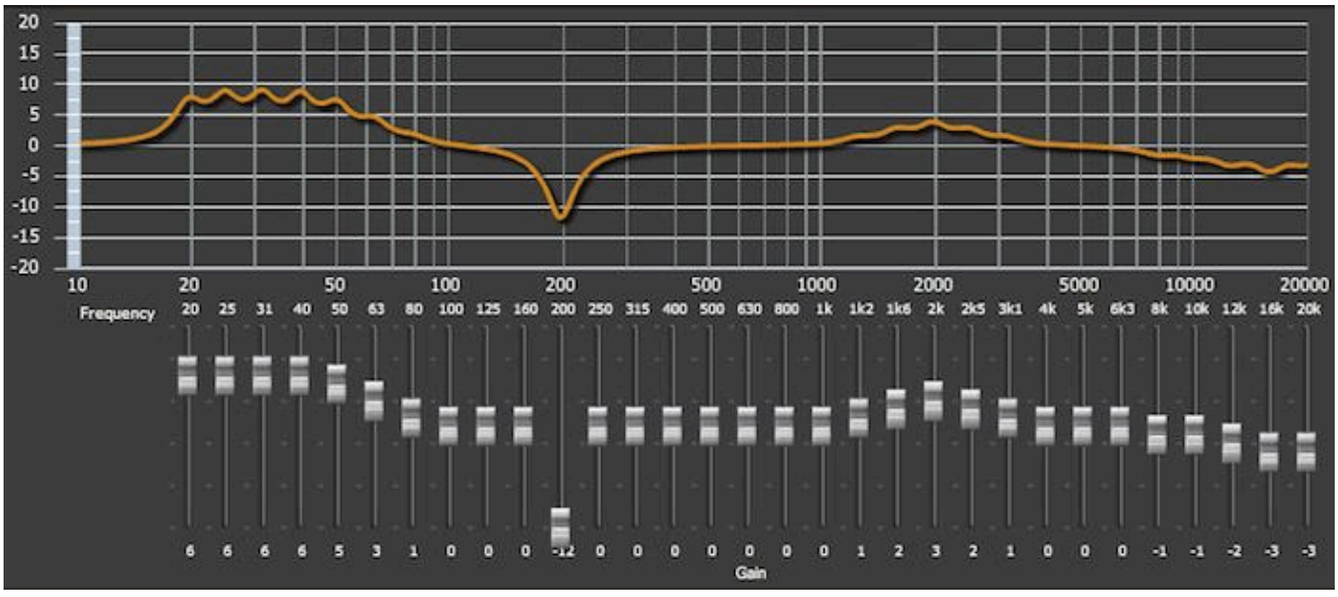


18. ábra: pólusok és zérusok elliptikus szűrő esetén

Ha a sima frekvenciaátvitel elegendő és nem akarjuk a bonyolultabb tervezési folyamatot, akkor a Butterworth szűrő is egy jó választás [10].

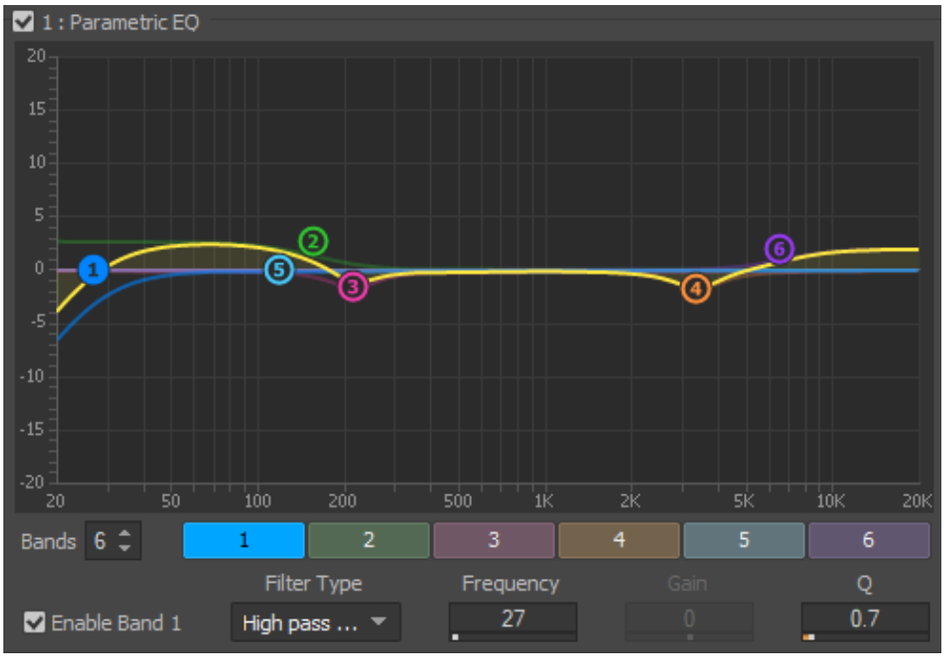
Két további példa a hangtechnikában gyakran használt EQ-kra lehet a grafikus EQ-k és a parametrikus EQ-k.

A grafikus EQ-kat gyakran használják élő hangalkalmazásokban, és egy sor rögzített frekvenciasávot tartalmaznak, általában 20 Hz és 20 kHz között, amelyeket csúszkák segítségével egyenként lehet bizonyos mértékben növelni vagy csökkenteni. Az egyes sávok szélessége rögzített, ami azt jelenti, hogy az egyes sávok által érintett frekvenciák mennyisége előre meghatározott. A grafikus EQ-kat gyakran használják általános hangszínformálásra, és hasznosak lehetnek egy keverés vagy élő előadás hangzásának gyors beállításához.



19. ábra: grafikus EQ

A parametrikus EQ-k az EQ-k precízebb típusai, amelyek nagyobb ellenőrzést tesznek lehetővé a beállítandó frekvenciák felett. Általában három fő paramétert tartalmaznak minden sávhoz: középfrekvencia, sávszélesség és erősítés. A középfrekvencia határozza meg, hogy melyik frekvencia kerül beállításra, a sávszélesség határozza meg, hogy a középfrekvencia körüli frekvenciák milyen tartományát érinti, az erősítés pedig azt, hogy az érintett frekvenciák mennyire erősödnek vagy gyengülnek. A parametrikus EQ-val a felhasználó pontosan megcélozhat és beállíthat bizonyos problémás frekvenciákat, vagy a frekvenciaspektrum bizonyos területeit erősítheti, így hatékony eszközzé válik a keverés alakításához vagy a felvétel problémáinak kijavításához [11].



20. ábra: parametrikus EQ

Összehasonlításképpen a grafikus EQ-k általában egyszerűbb kezelést biztosítanak és a hang általános hangolására használhatók, míg a parametrikus EQ-k nagyobb rugalmasságot kínálnak a hangfrekvenciák célzott beállításában. A választás a konkrét alkalmazástól és a felhasználó preferenciáitól függ, hogy melyik típusú EQ a legmegfelelőbb a kívánt hangbeállítások eléréséhez.

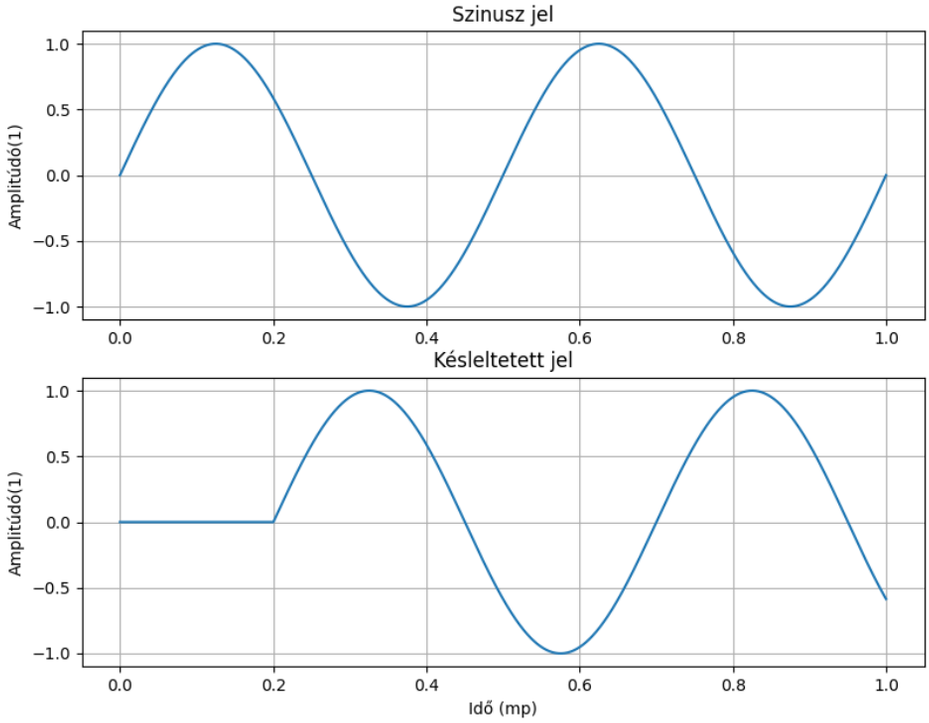
2.5. Késleltetés

A késleltetés egy olyan effektus, amely lehetővé teszi a hangjel időben történő eltolását. Ez azt jelenti, hogy a jel egy kis idővel később érkezik meg a kimeneten, mint amikor beérkezik a bemenetre. A késleltetés lehetővé teszi a hangzás módosítását és különböző effektusok létrehozását, például az "echo" hatást.

Az 17. egyenletben látható késleltetési módszer a hagyományos delay, amely a bemeneti jel elejéhez hozzáad egy nullákból álló szakaszt, amely a kívánt időtartamú késleltetést jelenti, majd ezt a késleltetett jelet hozzáadja az eredeti jelhez egy meghatározott amplitúdóval.

$$y[n] = x[n] + a \cdot x[n - d] \quad (17)$$

ahol "x[n]" az eredeti jel, "a" az erősítési tényező, "d" pedig a késleltetési idő.



21. ábra: késleltetés szemléltetése szinusz jelen

2.6. Konvolúciós téreffektus

A konvolúció egy olyan művelet, mely során két függvényt kombinálunk úgy, hogy az egyiküket eltoljuk és a másikkal szorozzuk össze. A gyakorlatban a konvolúció alkalmazásakor a diszkrét jeleket általában vektorokként kezeljük. Tehát x és h vektorok, amelyek elemei az adott időpillanatokban mintavételezett értékek. Az eredmény is egy vektor lesz, amely az x és h vektorok konvolúciójának eredményét tartalmazza. A konvolúciós téreffektusnál egy rögzített hangot (pl. egy szobában rögzített visszaverődést) alkalmaznak egy adott jelre. Ehhez használnak egy előre felvett impulzusválaszt (impulse response), amely leírja hogy a késleltetett jel hogyan viselkedik az idő függvényében. Ez az impulzusválasz kerül alkalmazásra a konvolúciós formulán keresztül, hogy a jelre alkalmazzák a visszaverődött hang hatását. Ezt követően a két jel összeszorzódik, és a kimeneti jel a konvolúció eredménye lesz.

A csillag (*) művelet jelenti magát a konvolúciót, mely tulajdonképpen az alábbi módon írható fel diszkrét rendszer esetén:

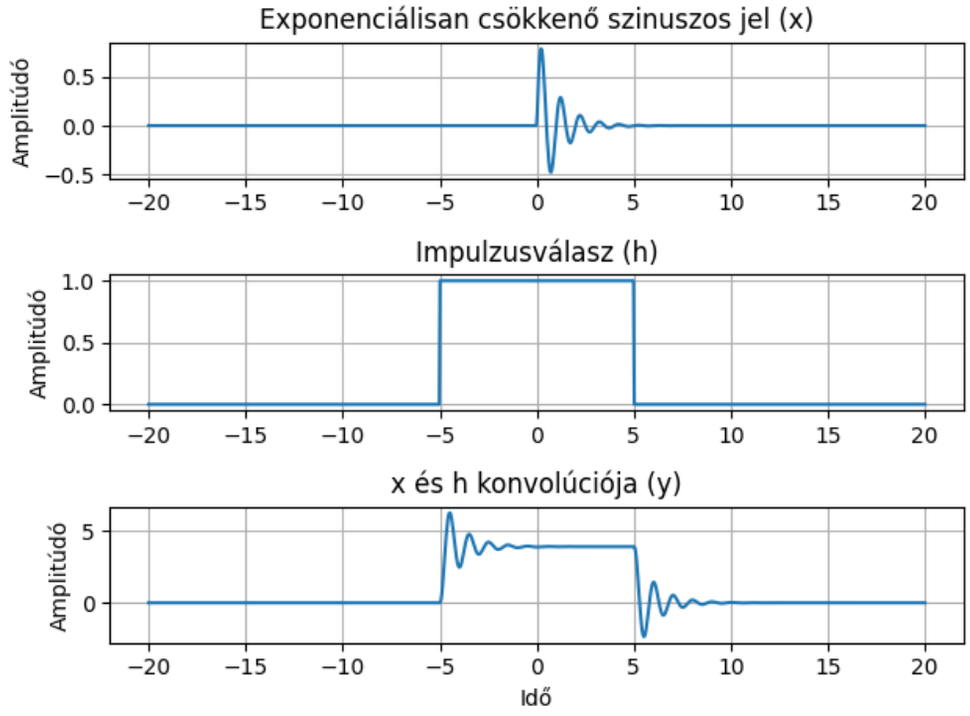
$$(x * h)[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \quad (18)$$

Tehát a kimeneti y[n] jel a következőképpen áll elő:

$$y[n] = (x[n] * h[n]) \cdot 2^{15} \quad (19)$$

ahol x[n] az eredeti jel, h[n] az impulzusválasz és 2^{15} a normalizáló konstans.

Az alábbi ábrán egy diszkrét rendszer ábrázolása látható egy bemeneti exponenciálisan csökkenő szinusz jelre (x[n]), alatta pedig egy h[n] négyszög jel, melyeket konvolválva megkapjuk a kimeneti y[n] jelet.



22. ábra: konvolúció vizuális szemléltetése

3. Megvalósítás

3.1. Python szimulációk

A pythonban megírt programok a következő szerkezetet tartják be:

1. Betöltjük a bemeneti hangfájlt.

```
fajlnev = "gitar.wav"
fs, signal_int16 = wavfile.read(fajlnev)
```

2. Konvertáljuk a mintaszámokat a lebegőpontos számokra.

```
signal_float64 = signal_int16.astype(float) / 2**15
```

3. Végrehajtjuk a különböző effektusokat a hangon, például a torzítást, az overdrive hard vagy soft clippingjét, alkalmazzuk az hangszínszabályozót vagy a noisegatet.

3.1.1. EQ (hangszín szabályozó):

```
def butterworth_eq(signal, fs, low_cutoff, high_cutoff, low_gain, mid_gain, high_gain):
    low_b, low_a = butter(4, low_cutoff / fs, 'low')
    high_b, high_a = butter(4, high_cutoff / fs, 'high')
    mid_b, mid_a = butter(4, [low_cutoff / fs, high_cutoff / fs], 'bandpass')

    low_signal = lfilter(low_b, low_a, signal)
    mid_signal = lfilter(mid_b, mid_a, signal)
    high_signal = lfilter(high_b, high_a, signal)

    filtered_signal = low_gain * low_signal + mid_gain * mid_signal + high_gain * high_signal
```

3.1.2. Torzítás:

```
def distortion(data_in: float, gain: float = 1.0) -> float:
        temp = np.sign(data_in) * (1.0 - np.exp(gain*np.sign(data_in)*data_in))
        temp = np.clip(temp, -1.0, 1.0)
        return temp
y = np.zeros_like(signal_float64)
for (i, x_i) in enumerate(signal_float64):
    y[i] = distortion(x_i, 5) * 2**15
wavfile.write("distortion.wav", fs , y.astype(np.int16))
```

3.1.3. Overdriveok:

```
def overdrive_hard_clipping(data_in: float, amplitude: float = 1.0, gain: float = 1.0) -> float:
    temp = gain*data_in
    if temp > 1.0:
        return 1.0
    if temp < -1.0:
        return -1.0
    return temp

def overdrive_soft_clipping_1(data_in: float, amplitude: float = 1.0, gain: float = 1.0) -> float:
    temp = amplitude * np.tanh(gain * data_in)
    if temp > 1.0:
        return 1.0
    if temp < -1.0:
        return -1.0
    return temp

def overdrive_soft_clipping_2(data_in: float, amplitude: float = 1.0, gain: float = 1.0) -> float:
    temp = amplitude * np.arctan(gain * data_in)
    if temp > 1.0:
        return 1.0
    if temp < -1.0:
        return -1.0
    return temp
y = np.zeros_like(signal_float64)
for (i, x_i) in enumerate(signal_float64):
    y[i] = overdrive_soft_clipping_1(x_i, 5) * 2**15
wavfile.write("distortion.wav", fs , y.astype(np.int16))
```

Itt bármelyik fenti függvényt le lehet hívni a wavfile.write-on belül.

3.1.4. Késleltetés:

```
echo_duration_sec = 0.2
delay_amplitude = 0.7

delay_len_samples = round(echo_duration_sec * fs)
zero_padding_signal = np.zeros(delay_len_samples)
delayed_sig = np.concatenate((zero_padding_signal, signal_float64))
signal_float64 = np.concatenate((signal_float64, zero_padding_signal))

summed_sig = (signal_float64 + delay_amplitude * delayed_sig * 2**15).astype(np.int16)
wavfile.write("basic_delay.wav", fs , summed_sig)
```

3.1.5. Konvolúciós térhatás:

```
impulse_response = np.zeros(delay_len_samples)
impulse_response[0] = 1
impulse_response[-1] = delay_amplitude
output_sig = (np.convolve(signal_float64, impulse_response)*2**15).astype(np.int16)
wavfile.write("convolutional_delay.wav", fs , output_sig)
```

3.1.6. Noisegate:

```
def noisegate(data_in: float, threshhold_attack: float = 0.2, threshhold_release: float = 0.05) -> float:
    temp = data_in
    global time
    time += 1 / fs
    if abs(temp) >= threshhold_attack:
        Gc = 1
    else:
        Gc = 0
    if time > threshhold_release and Gc <= z[1]:
        z[0] = atk_alpha*z[1] + (1 - atk_alpha)*Gc
    else:
        if time <= threshhold_release:
            z[0] = z[1]
        else:
            if Gc > z[1]:
                z[0] = rel_alpha*z[1] + (1-rel_alpha)*Gc
                time = 0
    z[1] = z[0]
    return temp*z[0]


y = np.zeros_like(signal_float64)


for (i, x_i) in enumerate(signal_float64):
    y[i] = noisegate(x_i, 0.05) * 2**15 * 20
```

4. A kapott hangmintákat visszaalakítjuk egész számokká egyidőben a módosított hangfájlok mentésével:

```
wavfile.write("effektus_neve.wav", fs , filtered_signal.astype(np.int16))
```

3.2. Hardver

A digitális jelfeldolgozó programunk futtatásához egy NUCLEO G491RE lapot használunk, amely egy STM32G491 mikrovezérlőt tartalmaz. Ez egy Arm Cortex-M4, 32 bites RISC magon alapul, ami 170 MHz-es frekvencián működik és rendelkezik single-precision lebegőpontos egységgel (floating-point unit – FPU), amely támogatja az összes Arm egypontos adatfeldolgozási utasítást és összes adattípust. Rendelkezik analóg-digitális átalakítókkal (ADC-k) és a digitális-analóg átalakítókkal (DAC-ok), melyek nagy pontossággal dolgoznak, és lehetővé teszik a jó minőségű jelfeldolgozást. Emellett az STM32G491 mikrovezérlő kiváló valós idejű jelfeldolgozásra, mivel rendelkezik DMA (Direct Memory Access) vezérlővel, amely lehetővé teszi a memóriához történő közvetlen hozzáférést anélkül, hogy a CPU-nak be kellene avatkoznia és I2S (Inter-IC Sound) interfésszel, ami biztosítja a digitális audio jelek közvetlen átvitelét a nyákon keresztül, az USB interfész segítségével pedig lehetőség van ezen jelek számítógépre küldésére is. [12]

Néhány másik DSP ami közül választhattam volna, a Texas Instrumentstől származik. Ennek a cégnek egyik legnépszerűbb DSP-je a TMS320C6000 család, amelyet különféle ipari, automotív és audioalkalmazásokban használnak. A család tagjai közé tartozik a TMS320C6748, ez egyaránt lehet fix- vagy lebegőpontos, attól függően hogy 375MHz-en vagy 456MHz-en fut. A mikrovezérlő egy alacsony fogyasztású C674x DSP magon alapszik, jelentősen kevesebb energiát felhaszálva, mint a család többi DSP-je. Szintén kiváló valós idejű jelfeldolgozásra, EDMA (Enchanced Direct Memory Access) vezérlővel rendelkezik, mely a DMA-nak egy fejlettebb változata. Sok perifériával rendelkezik, például SATA, USB 2.0 OTG, EMAC, McASP, McSPI és UART3. [13].

Végül azért döntöttem az STM32G491 mellett, mert gazdaságosabb megoldást kínál, mint a Texas Instruments legtöbb DSP-je, a beépített műveleti erősítők és komparátorok segítségével analóg feldolgozási képességekkel is rendelkezik, képes lebegőpontos számokkal dolgozni, akár szoftveres, akár hardveres gyorsítással, audiojelek feldolgozására tökéletesen megfelel, lévén hogy ezek nem igényelnek nagy pontosságot vagy dinamikatartományt. Könnyen integrálható, rugalmas, felhasználóbarát, könnyen programozható a saját fejlesztői környezetében amely nyílt forráskódú ARM-GCC kompilátorral rendelkezik, tehát bárki számára elérhető. A TMS320C6748 speciálisabb eszközöket és szoftvereket igényel a fejlesztéshez.

Az Altium Designer nevű tervezőprogram segítségével terveztünk egy saját nyákot, melynek lábait a mikrovezérlőn található kimenetekre szabtuk.

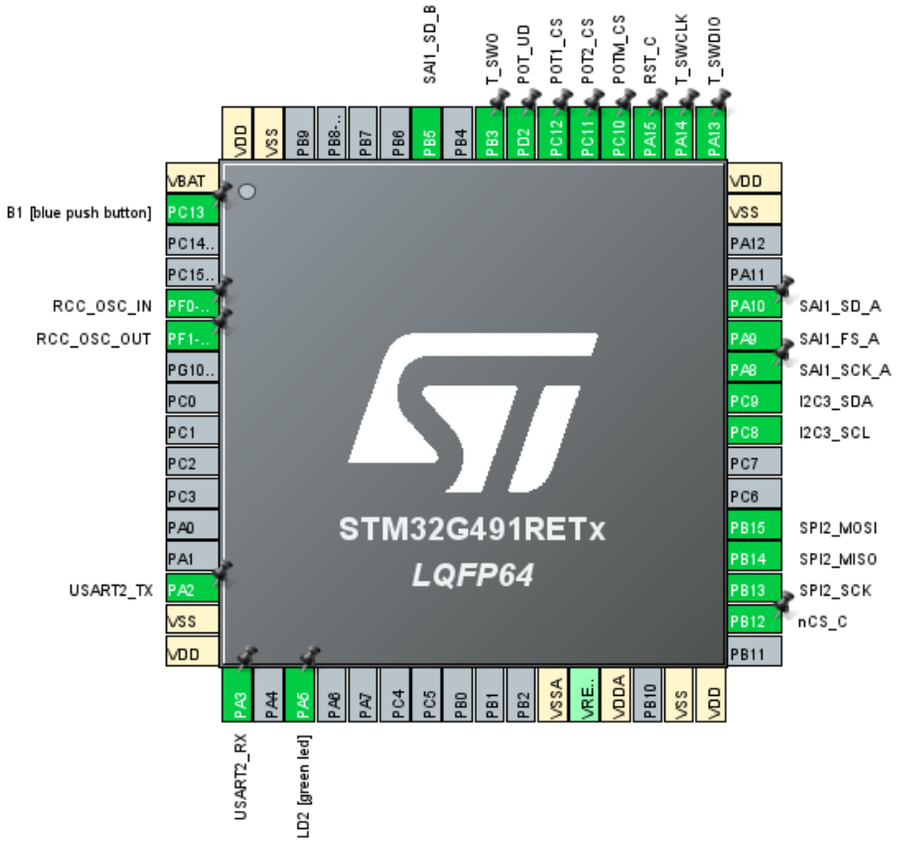
Miután elkészült a nyákterv, azt legyártattuk és a mikrovezérlőre helyeztük. Az elkészült nyákon 2 darab mono bemenet és 2 darab mono kimenet található, amelyek segítségével az audiojeleket vesszük és továbbítjuk. Emellett egy sztereó kimenet is rendelkezésre áll. Ennek struktúrája látható a függelékben.

Analóg bemenet: Az itt felvázolt rendszerben az MCP6002-es műveleti erősítő által erősített és előkészített analóg jelek, amelyeket az MCP4011-es digitális potenciométerek állítanak be, továbbítódnak az AD1938-as codec felé. Az anti-aliasing szűrő a bemeneti jelek frekvencia tartományának feléig vágja le a jeleket, hogy elkerülje az alulmintavételezésből adódó torzításokat. Ez a szűrő csökkenti a magasabb frekvenciájú zajt és nem kívánt komponenseket, amelyek az ADC mintavételi frekvenciájánál magasabb frekvencián helyezkednek el.

Analóg kimenet: Az itt felvázolt rendszerben az MCP6002-es műveleti erősítők negatív visszacsatolású erősítőként működnek, amelyben a kimenet visszakapcsolódik a negatív bemenetre. Ennek eredményeként az erősítő erősítése függ az erősítő két bemenetének különbségétől. Az erősítő negatív visszacsatolásának célja a pontos és stabil erősítés biztosítása. Az anti-imaging szűrő célja az, hogy kiszűrje vagy csökkentse a magas frekvenciás komponenseket, amelyek az analóg jelben jelentkezhetnek a digitális jelek átalakítása során. Az anti-imaging szűrőt a digitális-analóg átalakító (DAC) közelében helyezik el, mivel az analóg jel kondicionálása és az aliasing hatásának minimalizálása ebben a pontban a legfontosabb.

CODEC: Amint már fennebb is említettem, egy AD1938-as codecet használok, ezáltal is biztosítom, hogy az analóg jel ne keveredjen a digitális jellel, mert ez befolyásolhatja az átalakítási folyamatot és torzításokhoz vezethet. Az analóg jel és a digitális jel átvitele két különböző rendszeren keresztül történik, és ezeket a rendszereket el kell különíteni egymástól annak érdekében, hogy biztosítsuk a jelminőség megfelelő szintjét és az adatok helyes átvitelét. A codec választásakor szempont volt, hogy könnyen integrálható legyen Nucleo platformba, melynek köszönhetően közvetlenül rá lehet helyezni a lapra, így egyszerűsítve a rendszer megtervezését és relatív gyors tesztelését.

Mindezek után megkezdtük a digitális jelfeldolgozó programunk fejlesztését az STM32CubeIDE nevű fejlesztői környezetetben és átkonvertáltuk a Python kódot C kódra. Végül ezt a C kódot töltöttük fel a mikrovezérlőre, hogy elindíthassuk a digitális jelfeldolgozást a saját tervezésű hardverünkön.



23. ábra: a tervezett egység .ioc konfigurációs állománya

3.3. Firmver

Firmverünk az előbb említett STM32CubeIDE-ben készült fejlesztői környezetben íródott. Ez egy olyan szoftveralkalmazás, amelyet beágyazott rendszerekhez terveztek. Általában alacsony szintű, közvetlenül az eszközhöz és a hardverhez kapcsolódó műveleteket hajt végre, például adatbevitelt és -kimenetet, illetve az adatok feldolgozását. A lenti kód a HAL (Hardware Abstraction Layer) könyvtárakat használja.

A HAL és LL könyvtárakról: A **HAL** egy magasabb szintű réteg, amely absztrakciót biztosít a hardveres perifériák használata során, könnyen használható, magasabb szintű API-t kínál, ezzel lehetővé teszi a gyorsabb fejlesztést, azonos módon kezeli a különböző STM32 mikrovezérlőket,

ezért könnyebb a portolás az eltérő modellek között. Az alkalmazási réteg és a hardver között helyezkedik el, ezért a felhasználónak nem kell mélyen megértenie a hardver működését. Ezzel szemben az **LL (Low-Level)** egy alacsonyabb szintű réteg, mely közelebb áll a hardverhez és kevesebb absztrakciót biztosít. Közvetlen hozzáférést nyújt a hardver perifériáihoz, így a fejlesztő részletesen irányíthatja ezek működését, kisebb méretű kódot eredményezhet és gyorsabb végrehajtást, illetve az adott mikrovezérlő specifikus funkcióira fókuszál, ezért ha áttérünk egy másik mikrovezérlőre, újra kell írni a kódot [14].

A **HAL_SAI_RxCpltCallback** függvény egy megszakítási rutin, amelyet a SAI (Serial Audio Interface) fogadó megszakítása okoz. A függvény másolja a fogadott adatokat a **receiveBuffer** tömbbe, majd másolja a **transmitBuffer** tömb tartalmát az előzőleg fogadott adatokra. Végül a függvény átkapcsol a **DMA** (Direct Memory Access) átviteli módba, amely lehetővé teszi a **transmitBuffer** tartalmának továbbítását a **SAI**-on keresztül, majd újra elindítja a fogadási folyamatot a **HAL_SAI_Receive_IT** függvénnyel. A szemléltetett kódszegmens effektus nélkül visszatéríti a mintavételezett jelet.

```
void HAL_SAI_RxCpltCallback(SAI_HandleTypeDef *hsai){
   transmitBuffer[0] = receiveBuffer[0];
   transmitBuffer[1] = receiveBuffer[1];

   HAL_SAI_Transmit_DMA(&hsai_BlockB1, transmitBuffer, 8);
   sai_rx_flag = 1;
   HAL_SAI_Receive_IT(&hsai_BlockA1, receiveBuffer, 8);
}
```

A **distortion_apply** függvény egy egyszerű torzító algoritmus, amely a kapott bemeneti adatot megváltoztatja egy torzítási faktorral, ez a Distortion struktúrán belül gain néven van deklarálva, mely egy lebegőpontos számot tárol.

```
float distortion_apply(Distortion *dist, float data_in) {
    float temp = (data_in < 0) ? -1 : 1;
    temp *= (1 - exp(dist->gain * temp * data_in));
    return fmaxf(-1, fminf(temp, 1));
}
```

Az **overdrive_apply_hard_clipping**, **overdrive_apply_soft_clipping_1** és **overdrive_apply_soft_clipping_2** függvények különböző módszereket alkalmaznak az eredményezett torzítás módosítására, mindez az Overdrive struktúrán belül történik, amely 2 lebegőpontos adattagot tartalmaz, amplitude és gain.

```
float overdrive_clip(Overdrive *od, float value) {
    return fminf(fmaxf(value, -1.0f), 1.0f);
}

float overdrive_apply_hard_clipping(Overdrive *od, float data_in) {
    float temp = od->gain * data_in;
    return overdrive_clip(od, temp);
}

float overdrive_apply_soft_clipping_1(Overdrive *od, float data_in) {
    float temp = od->amplitude * tanhf(od->gain * data_in);
    return overdrive_clip(od, temp);
}

float overdrive_apply_soft_clipping_2(Overdrive *od, float data_in) {
    float temp = od->amplitude * atanf(od->gain * data_in);
    return overdrive_clip(od, temp);
}
```

A **convolve** függvény egy konvolúciós algoritmust valósít meg, amely lehetővé teszi két adatsor összeszorzását, majd a szorzatok összeadását. Ez az algoritmus hasznos lehet például az akusztikus visszacsatolás (feedback) csökkentéséhez.

```
void convolve(const float* input_signal, uint32_t input_len, const float* impulse_response, uint32_t impulse_len, float* output_signal) {
    for (int i = 0; i < input_len + impulse_len - 1; i++) {
        output_signal[i] = 0.0;
        int start = (i >= impulse_len - 1) ? i - (impulse_len - 1) : 0;
        int end = (i < input_len - 1) ? i : input_len - 1;
        for (int j = start; j <= end; j++) {
            output_signal[i] += input_signal[j] * impulse_response[i - j];
        }
    }
}
```

Az **apply_basic_delay** és **apply_convolutional_delay** függvények két különböző késleltető algoritmust valósítanak meg. Az előbbi egyszerűen csak eltárolja a bemeneti adatokat egy bufferekben, majd azokat késlelteti és összeadja a késleltetett adatokkal.

```
void apply_basic_delay(Delay* delay, int16_t* signal, uint32_t signal_len) {
    if (delay->delay_buffer == NULL) {
        delay->delay_buffer = (int16_t*) calloc(delay->delay_len_samples, sizeof(int16_t));
    }

    int16_t* temp_buffer = (int16_t*) malloc(delay->delay_len_samples * sizeof(int16_t));
    memcpy(temp_buffer, delay->delay_buffer, delay->delay_len_samples * sizeof(int16_t));

    memcpy(delay->delay_buffer, signal, signal_len * sizeof(int16_t));

    for (int i = 0; i < signal_len; i++) {
        signal[i] += (int16_t) (delay->amplitude * temp_buffer[i]);
    }

    free(temp_buffer);
}
```

Az konvolúciós téreffektus használja a **convolve** függvényt a késleltető effektus létrehozására. Mindkét függvény a Delay struktúra segítségével és ennek 4 adattagjával határozza meg a késleltetés karakterisztikáját.

```
void apply_convolutional_delay(Delay* delay, int16_t* signal, uint32_t signal_len) {

   float* impulse_response = (float*) calloc(delay->delay_len_samples, sizeof(float));
   impulse_response[0] = 1.0;
   impulse_response[delay->delay_len_samples - 1] = delay->amplitude;

   float* input_signal = (float*) malloc(signal_len * sizeof(float));
   for (int i = 0; i < signal_len; i++) {
      input_signal[i] = (float) signal[i] / 32768.0f;
   }
   float* output_signal = (float*) calloc(signal_len + delay->delay_len_samples, sizeof(float));
   convolve(input_signal, signal_len, impulse_response, delay->delay_len_samples, output_signal);

   for (int i = 0; i < signal_len; i++) {
      signal[i] = (int16_t) (output_signal[i + delay->delay_len_samples - 1] * 32767.0f);
   }

   free(impulse_response);
   free(input_signal);
   free(output_signal);
}
```

A főfüggvényben, vagyis a **main**ben inicializáljuk a perifériákat:

```
SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C3_Init();
  MX_SPI2_Init();
  MX_DMA_Init();
  MX_SAI1_Init();
  MX_USART2_UART_Init();
  MX_TIM6_Init();
  AD1938_init();
  HAL_SAI_Init(&hsai_BlockB1);
  HAL_SAI_Init(&hsai_BlockA1);

  HAL_SAI_Receive_IT(&hsai_BlockA1, receiveBuffer, 8);
  //HAL_SAI_Transmit_IT(&hsai_BlockB1, transmitBuffer, 8);
  HAL_SAI_Transmit_DMA(&hsai_BlockB1, transmitBuffer, 8);
  //HAL_TIM_Base_Start_IT(&htim6);

  __asm("NOP");
```

Ezek után meghatározzuk, hogy melyik effektusokat akarjuk alkalmazni az alábbi struktúrák segítségével:

```
typedef struct Distortion {
    float gain;
} Distortion;
typedef struct Overdrive{
    float amplitude;
    float gain;
} Overdrive;
typedef struct Delay {
    float duration_sec;
    float amplitude;
    uint32_t delay_len_samples;
    int16_t* delay_buffer;
} Delay;
```

Az alábbi módon:

```
Distortion dist = {1.0};
Overdrive overdrive = {0.5, 1.0};
Delay delay = {0.5, 0.8, 44100};
```

Az ezt követő kód végtelen ciklusban fut, és a beérkező hangmintákat folyamatosan feldolgozza az effektusok alkalmazása és a visszaküldés során.

```
int main(void)
{
  Distortion dist = {1.0};
  Overdrive overdrive = {0.5, 1.0};
  Delay delay = {0.5, 0.8, 44100};

  while (1)
  {
     if (sai_rx_flag == 1) {
        // samples
        float sample_left = (float)receiveBuffer[0]/normalizalas;
        float sample_right = receiveBuffer[1]/normalizalas;

        //distortion on samples
        sample_left = distortion_apply(&dist, sample_left);
        sample_right = distortion_apply(&dist, sample_right);

        //overdrive on samples
        sample_left = overdrive_apply_hard_clipping(&overdrive, sample_left);
        sample_right = overdrive_apply_hard_clipping(&overdrive, sample_right);

        //delay on samples
        int16_t samples[2] = { (int16_t) (sample_left * 32767.0f), (int16_t) (sample_right * 32767.0f) };
        apply_basic_delay(&delay, samples, 2);
        apply_convolutional_delay(&delay, samples, 2);

        //out
        transmitBuffer[0] = sample_left;
        transmitBuffer[1] = sample_right;

        // transmit the data
        HAL_SAI_Transmit_DMA(&hsai_BlockB1, transmitBuffer, 8);
        sai_rx_flag = 0;
     }
  }
}
```

4. Következtetések és továbbfejlesztés

A dolgozat célja egy alacsony költségű, könnyen hozzáférhető digitális hang effektus egység megvalósítása volt. A projekt során a digitális jelfeldolgozási elméletet tanulmányoztuk, algoritmusokat terveztünk és fejlesztettünk, valamint a hardveres megvalósítást is elkészítettük. Az effektusok minőségét objektív és szubjektív módszerekkel is teszteltük.

Az eredmények alapján megállapíthatjuk, hogy sikerült egy alacsony költségű, könnyen hozzáférhető digitális hang effektus egységet megvalósítani, amely képes a kívánt effektusok létrehozására, valós időben működik és jó minőségű hangot szolgáltat.

Az effektusok továbbfejlesztése számos lehetőséget kínál. Az egyik lehetőség a meglévő effektusok finomhangolása, hogy még jobb minőségű hangzást érjünk el. Ezen kívül, új effektusok fejlesztése is lehetséges, hogy további kreatív lehetőségeket nyújtsunk a felhasználók számára.

A hardveres megvalósítás terén is van továbbfejlesztési lehetőség. Például, a jelenlegi hardver korlátai miatt az effektusok száma és bonyolultsága limitált. Azonban, további hardveres fejlesztésekkel a processzor sebességét és memóriáját növelve, lehetővé válhat az effektusok számának és bonyolultságának növelése is.

5. Irodalomjegyzék

[1] Oppenheim, Alan V. and Cram. “Discrete-time signal processing” 3rd edition. (2011).

[2] The IEEE Signal Processing Society. “Fifty years of Signal Processing” (1998).

[3] John G. Proakis, Dimitris G. Manolakis. “Digital Signal Processing: Principles, Algorithms, and applications” 3rd edition. (1995).

[4] https://ethw.org/Digital_Signal_Processing

[5] https://www.roland.com/uk/blog/the-history-of-boss-compact-pedals/

[6] https://electronics.howstuffworks.com/gadgets/audio-music/guitar-pedal1.htm

[7] Dave Hunter. “Guitar Effects Pedals: The Practical Handbook”. (2013).

[8] https://jetpedals.com/blogs/news/clipping-diodes-and-how-they-affect-your-tone

[9] Timothy A. Dittmar. “Audio Engineering 101: A Beginner's Guide to Music Production” (2012).

[10] M.E. Van Valkenburg “Analog Filter Design” (1982).

[11] Bobby Owsinski. “The Mixing Engineer’s Handbook” 3rd edition. (2013).

[12] https://www.st.com/en/microcontrollers-microprocessors/stm32g491re.html

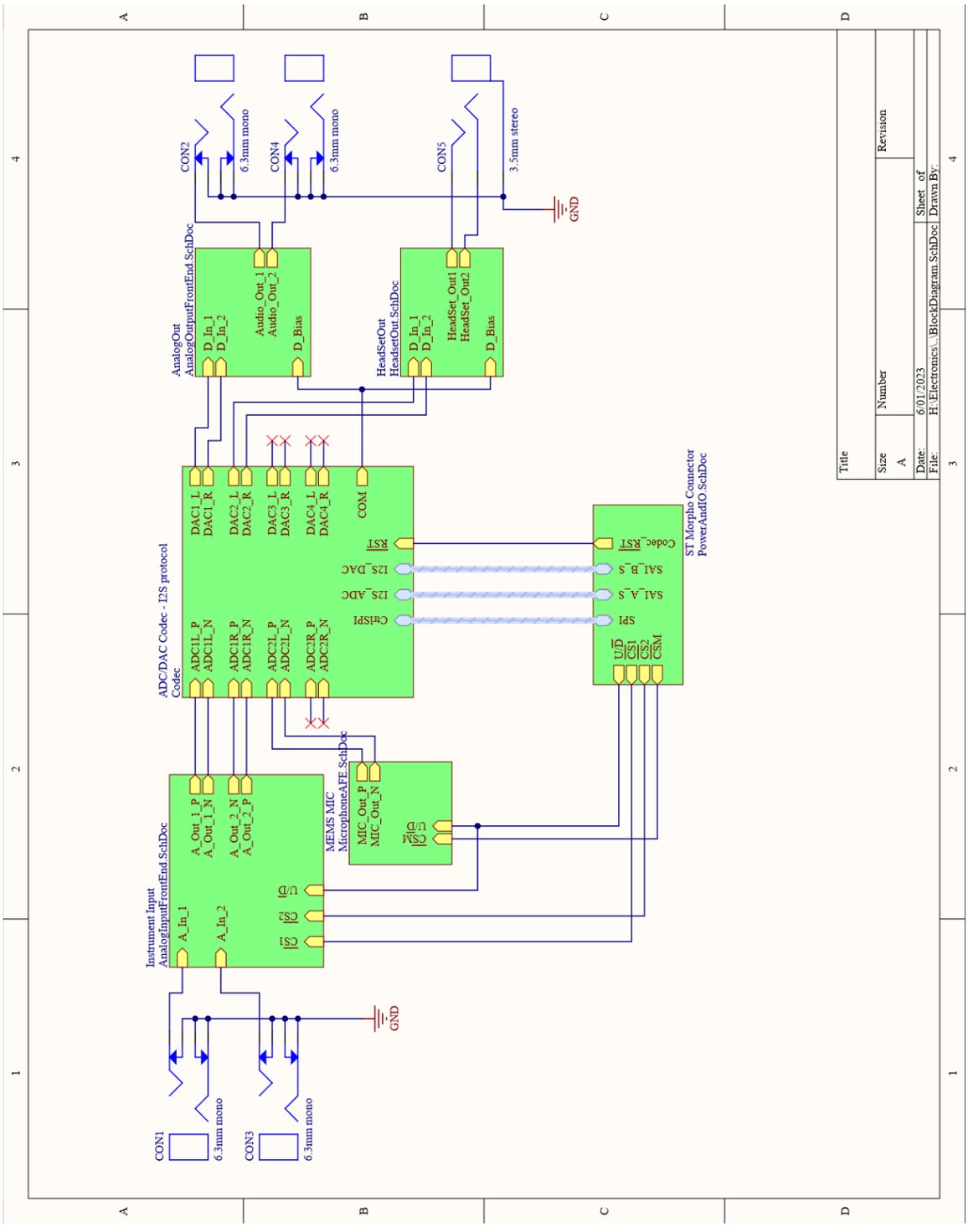
[13] https://www.ti.com/product/TMS320C6748

[14] https://community.st.com/s/question/0D50X00009XkhQSSAZ/hal-vs-ll

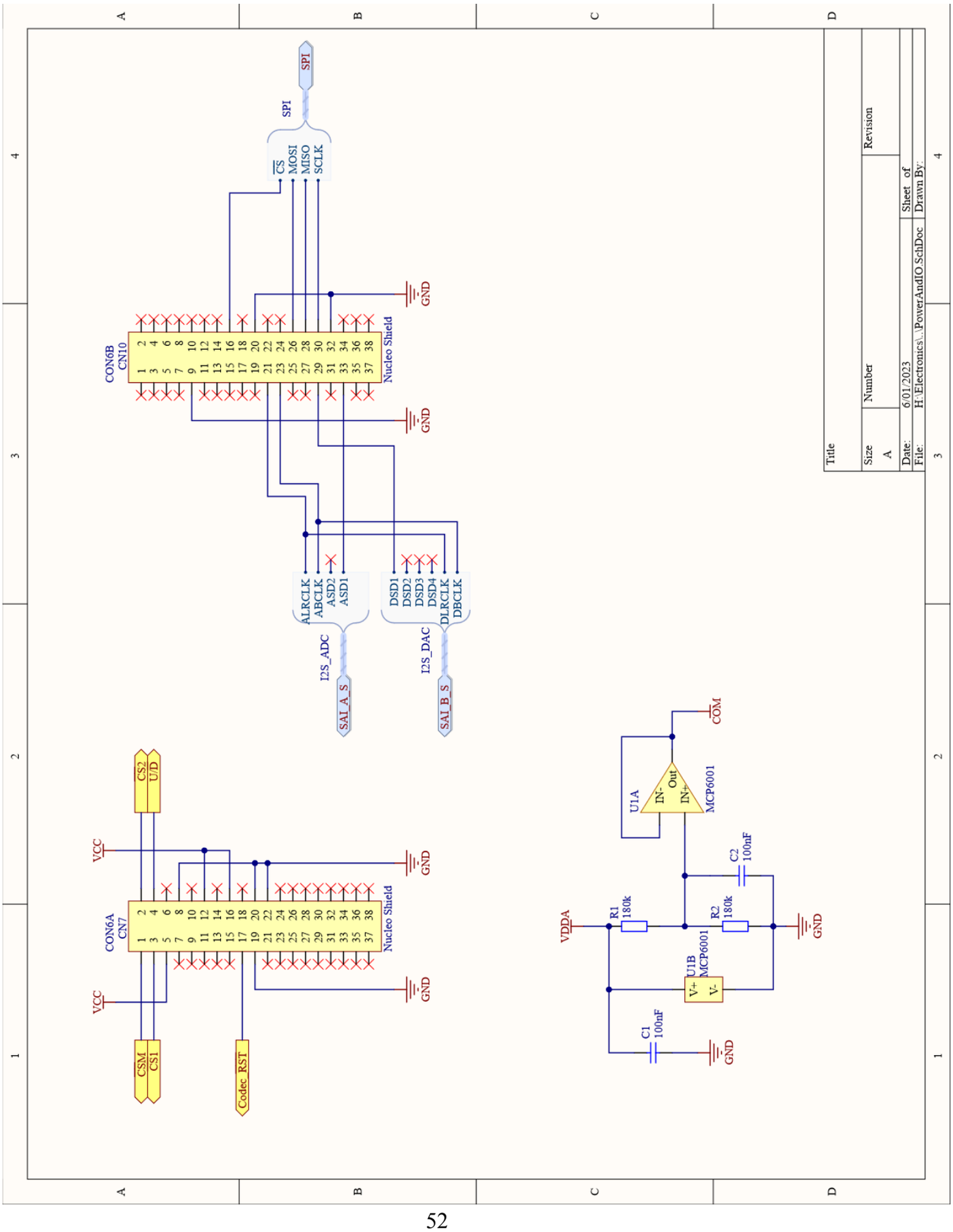
6. Függelék

6.1. Hardver struktúrája

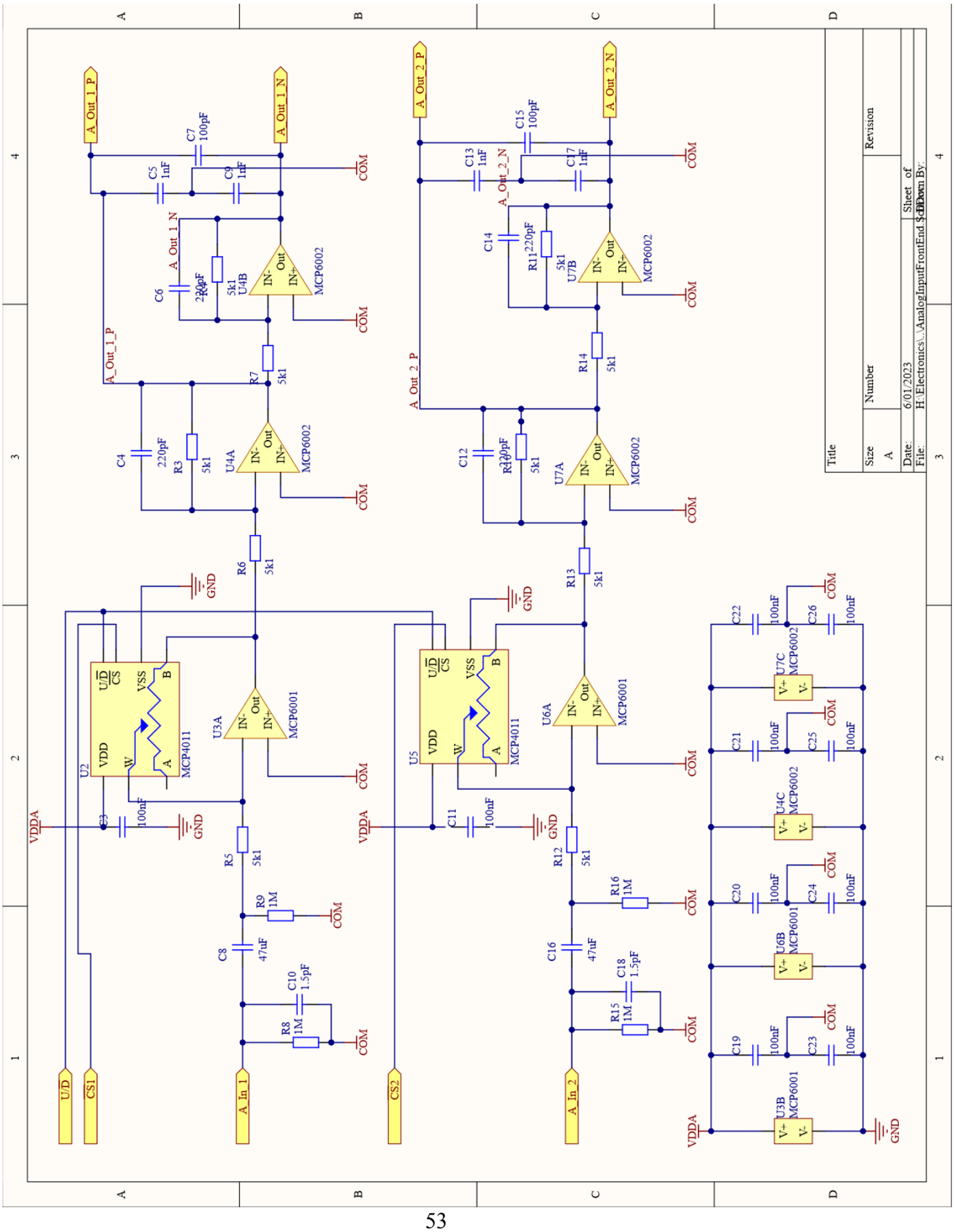
6.1.1. Blokkdiagram



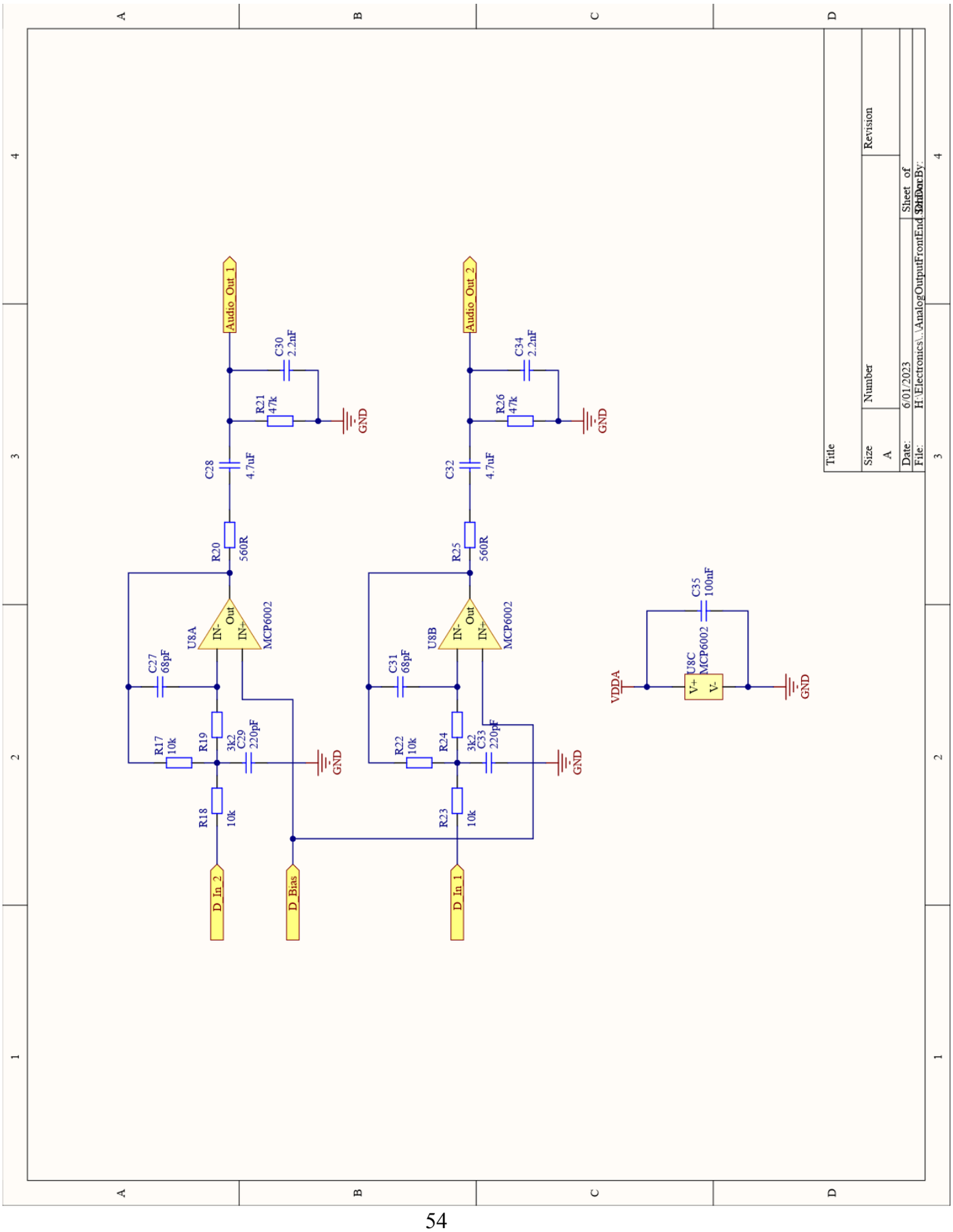
6.1.2. Táp és IO portok



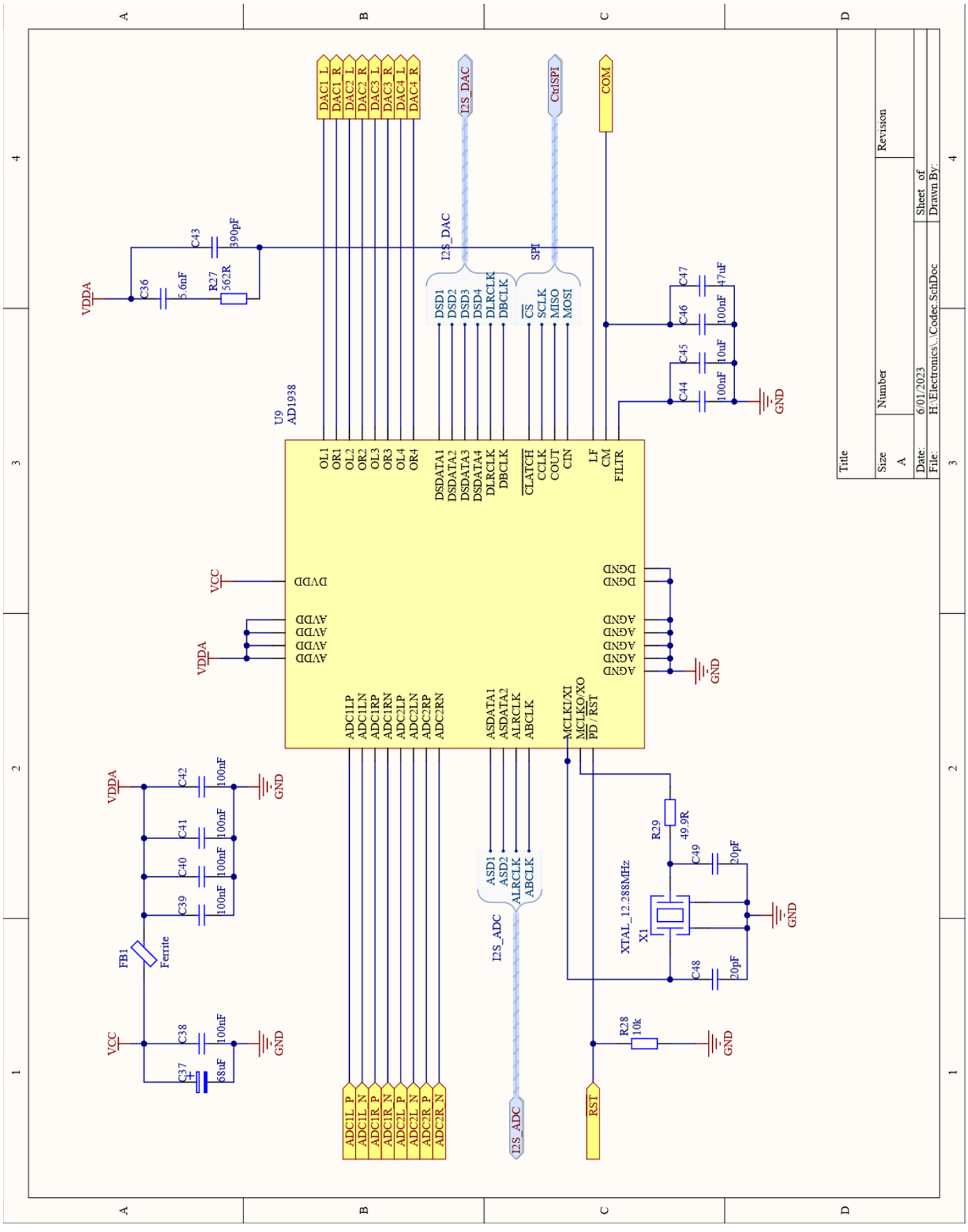
6.1.3. Analóg bemenet



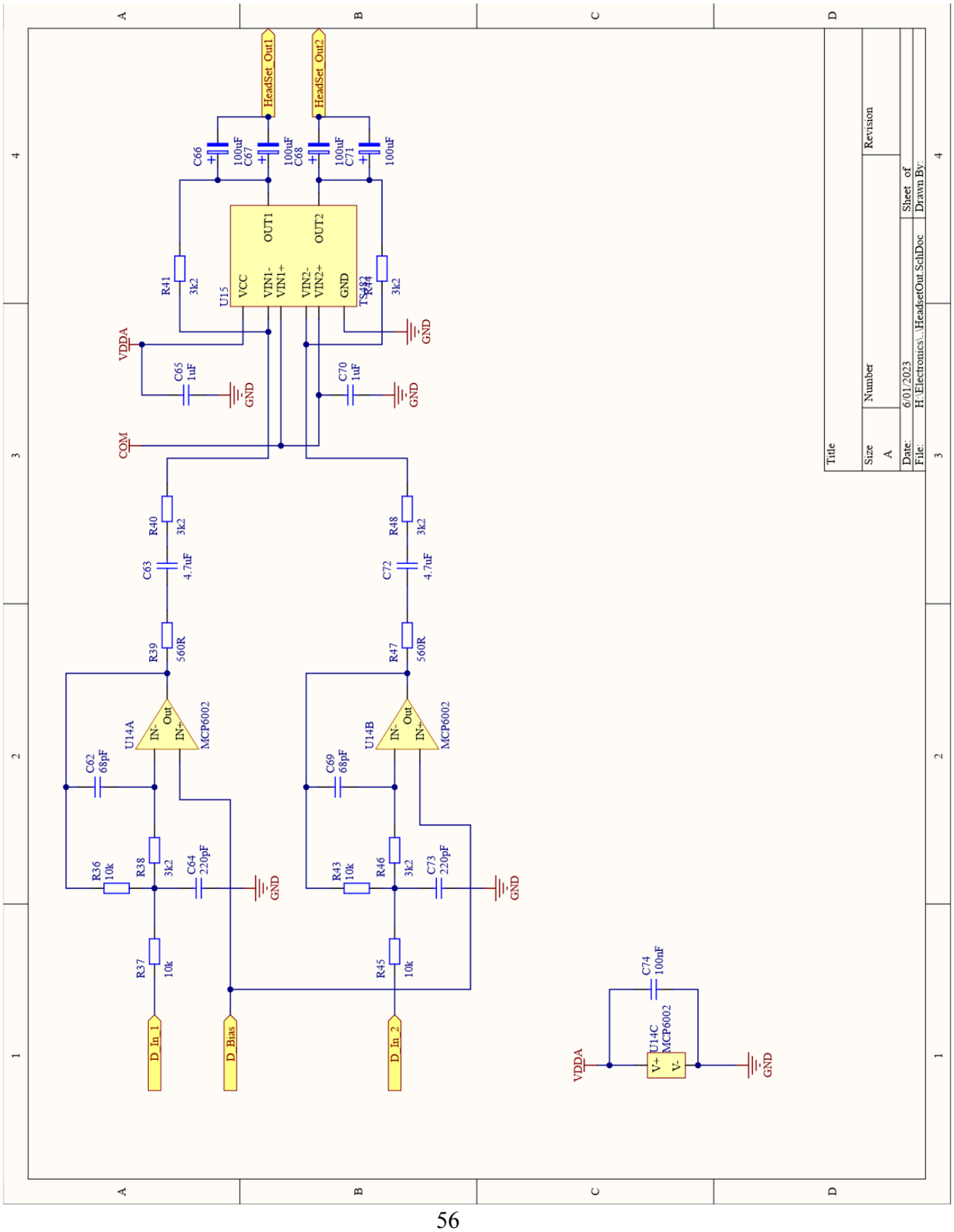
6.1.4. Analóg kimenet



6.1.5. Codec


U9
AD1938
DAC1_L
DAC1_R
DAC2_L
DAC2_R
DAC3_L
DAC3_R
DAC4_L
DAC4_R
I2S_DAC
CtrlSPI
COM
OL1
OR1
OL2
OR2
OL3
OR3
OL4
OR4
DSDATA1
DSDATA2
DSDATA3
DSDATA4
DLRCLK
DBCLK
CLATCH
CCLK
COUT
CIN
LF
CM
FILTR
DVDD
AVDD
DGND
AGND
ADC1LP
ADC1LN
ADC1RP
ADC1RN
ADC2LP
ADC2LN
ADC2RP
ADC2RN
ASDATA1
ASDATA2
ALRCLK
ABCLK
MCLKI/XI
MCLKO/XO
PD / RST
DSD1
DSD2
DSD3
DSD4
DLRCLK
DBCLK
SPI
CS
SCLK
MISO
MOSI
VDDA
VCC
GND
C43
390pF
C36
5.6nF
R27
562R
C47
47uF
C46
100nF
C45
10uF
C44
100nF
C42
100nF
C41
100nF
C40
100nF
C39
100nF
FB1
Ferrite
C38
100nF
C37
68uF
R29
49.9R
C49
20pF
C48
20pF
X1
XTAL_12.288MHz
R28
10k
ASD1
ASD2
ALRCLK
ABCLK
I2S_ADC
RST
Title
Size
A
Number
Revision
Date:
6/01/2023
Sheet of
File:
H:\Electronics\...\Codec.SchDoc
Drawn By:

6.1.6. Sztereó kimenet



UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA

FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ

SPECIALIZAREA CALCULATOARE

Vizat decan

Conf. dr. ing. Domokos József

Vizat director departament

Ș.l. dr. ing Szabó László Zsolt