ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZUBTRAKTÍV SZINTETIZÁTOR MEGVALÓSÍTÁSA VST KÖRNYEZETBEN

OE-NIK Hallgató neve: **Balásfalvi Ádám**
2022 Hallgató törzskönyvi száma: **T/005033/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Balásfalvi Ádám**
Törzskönyvi száma: T/005033/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Szubtraktív szintetizátor megvalósítása VST környezetben
Developing a subtractive synthesizer VST

Intézményi konzulens: Lovas István
Külső konzulens:

Beadási határidő: 2021. december 15.

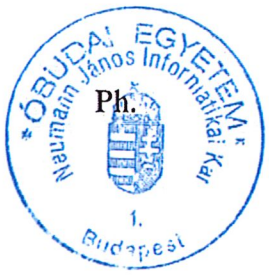
A záróvizsga tárgyai: Számítógép architektúrák
Beágyazott rendszerek és Mobil informatika specializáció

A feladat

Tervezzen és valósítson meg egy szubtraktív VST szintetizátort. A dolgozatban ismertesse a hangszintetizálás alapjait és a szubtraktív szintetizátorok működését. Mutassa be a meglévő irodalom alapján hogy milyen komponensekből áll egy szintetizátor és milyen módokon lehet az egyes komponenseket megvalósítani. A bemutatáskor vegye figyelembe az egyes megoldások előnyeit és hátrányait. Ismertesse a rendszer megvalósításához szükséges technológiákat. A szintetizátor kezeléséhez írjon egy kezelőfelületet, és tegye lehetővé, hogy a szintetizátort MIDI jelek segítségével lehessen vezérelni. Részletesen dokumentálja a megvalósítás lépéseit.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- a meglévő rendszerek vizsgálatának elemzését,
- az alkalmazni kívánt technológiák elemzését,
- a rendszertervet, döntések indoklásait,
- a megvalósítás leírását,
- a tesztelési tervet és a tesztek eredményeit,
- az elkészült rendszer továbbfejlesztése lehetőségeit.



.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

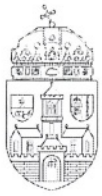
HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. május 14.

.....

hallgató aláírása



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Balásfalvi Ádám

Neptun kód: [redacted]

Tagozat: Nappali

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka címe magyarul:

Szubtraktív szintetizátor megvalósítása VST környezetben

Szakdolgozat / Diplomamunka címe angolul:

Developing a subtractive synthesizer VST

Intézményi konzulens: Lovas István

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.16	Specifikáció áttekintése	
2.	2022.03.09	Rendszerterv áttekintése	
3.	2022.04.06	Fejlesztés, tesztelés ellenőrzés	
4.	2022.05.04	Tartalmi, formai ellenőrzés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 tantárgy követelményét teljesítette, beszámolóra / védésre bocsátható.

Budapest, 2022. 05. 11.

.....
Intézményi konzulens

1 TARTALOMJEGYZÉK

2 BEVEZETÉS

2.1 A szintetizátorok rövid története [5][7][13]

A szintetizátorok születése a 20. század elejére nyúlik vissza. Bár a hangok elektromos jelekké átalakítása már a 19. században megtörtént a telefon segítségével, itt még nem történt elektromos jelgenerálás. A következő évtizedekben számtalan elektronikus hangszer látott napvilágot: Telharmonium (1896), theremin (1920). Ezek mindegyike kikövezte a modern szintetizátorok alapjait.

Az egyik legelső mai értelemben vett szintetizátor az RCA Mark II Sound Synthesizer (1957) volt, ez már képes volt szubtraktív szintézisre is. Az oszcillátorai és szűrői elektroncsövekből épültek fel, emiatt mérete a korabeli számítógépekéhez volt mérhető. Bár mérete és nehézkes használata miatt hamar elavult, de az itt alkalmazott felépítés jelentette a jövőt.

A következő években a technológia fejlődésével mind a szintetizátorok ára, mind a mérete csökkent, ezzel egyidőben a megbízhatóságuk nőtt. Ekkor jelentek meg az első Robert Moog és Don Buchla által épített szintetizátorok [5][7]. Ezek úgynevezett moduláris szintetizátorok voltak, ami azt jelentette, hogy a különböző egységeket (oszcillátorok, szűrők, burkológörbék) patch kábelekkel lehetett összekapcsolni. Itt jelent meg a Robert Moog által kifejlesztett feszültség vezérelt oszcillátor is, amivel nagy pontossággal lehetett a kívánt hangmagasságot beállítani. Az itt megjelenő irányelvek (moduláris felépítés, feszültség vezérelt elektronika, burkológörbék használata) a mai napig meghatározóak.

Az áttörést az 1970-ben megjelent Minimoog jelentette, ahol bár az egyes egységek előre össze voltak kapcsolva, de emiatt egyben sokkal hordozhatóbbá és felhasználó-barátabbá is vált, és ezzel megnyitotta az utat a szintetizátorok széleskörű felhasználásához a zenében.

Napjainkban a digitális és hibrid hangszintézist használunk döntő többségben. Ez gyakorlatban azt jelenti, hogy analóg szintetizátorokat szimulálunk digitális jelfeldolgozási eszközökkel és algoritmusokkal. Dolgozatomban szeretném bemutatni egy digitális megvalósítását a hang szintézisnek. Ennek folyamán bemutatom a hang előállítását szubtraktív szintézissel, az ehhez szükséges egységeket, az egységek felépítését és működését, és ezek digitális megvalósítását.

2.2 A szubtraktív szintézisről röviden

A szubtraktív szintézisnek az az alapötlete, hogy az oszcillátorok egy felharmonikusokban gazdag hullámformát állítanak elő, majd a különböző szűrők addig alakítják ezt a hullámformát, amíg a kívánt hang lesz az eredmény.

Ezt alapvetően három egység egymáshoz kapcsolásával érhető el:

1. Az első egységben egy vagy több oszcillátor állítja elő a meghatározott frekvenciájú, harmonikusokban gazdag hullámformát. Alakja lehet szinusz, négyszög, fűrészjel. A továbbiakban ez a jel lesz tovább alakítva.
2. A következő szakaszban az oszcillátorokkal előállított jelek különböző szűrőkön haladnak át. Itt szintetizátortól függően számtalan fajta szűrő jöhet szóba, de általában megtalálható egy aluláteresztő, felüláteresztő és egy sáváteresztő szűrő. Itt található még a törésponti frekvencia és rezonancia állításának a lehetősége is.

3. Az utolsó egység gondoskodik az előállított jel erősítéséről. Ennek segítségével lehet a kimeneti hangerőt szabályozni.

Ezen kívül mindegyik egységhez lehetőség van még további egységeket kapcsolni:

- Lehetséges egy alacsony frekvenciájú oszcillátor (LFO – low frequency oscillator) kapcsolása. Ez az oszcillátor a kívánt paramétert modulálja az LFO frekvenciájával. Például, ha a hullámformát elállító oszcillátor hangmagasság (pitch) paraméterére kapcsolódik az LFO, akkor egy időben (az LFO frekvenciájától függő) változó hangmagasságú jel lesz az eredmény.
- Hozzákapcsolható az erősítőegységhez egy burkológörbe (envelope). A burkológörbével az előállított hang dinamikai jellemzőit lehet állítani.

A továbbiakban bemutatom az egyes egységek működését és az ezeket előállító algoritmusokat.

2.3 **Oszcillátor**

A hang-előállítási lánc első eleme az oszcillátor. Oszcillátoron egy olyan áramkört értünk, amely egy periodikus jelet állít elő. A dolgozatban ennek az áramkörnek a modellezése lesz megvalósítva. Szubtraktív szintézis esetén elsősorban felharmonikusokban gazdag jel a cél (pl. háromszög vagy fűrészfog jel), de természetesen lehetővé kell tenni egyszerű szinusz jel megalkotását is.

Az oszcillátor által generált jelet alapvetően két paraméter szerint lehet állítani:

1. Frekvencia (hang magassága)
2. A jel hullámformája

A múltban ez elsősorban feszültség-vezérelt oszcillátorokkal történt (VCO), itt diszkrét áramköri elemek segítségével analóg jelet állítottak elő. Hátrányuk volt az érzékenységük a külső fizikai behatásokra és az állandó elhangolódás. Előnyük, hogy analóg működésükből adódóan tiszta jelet állítanak elő.

A 80-as években megjelentek az első digitális oszcillátorok. Itt a jelet már diszkrét időközönként generálják, jellemzően logikai áramkörök segítségével. Emiatt jóval kezelhetőbbé és stabilabbá váltak az oszcillátorok. Hátrányuk viszont, hogy a megjelent a mintavételezési frekvencia, mint egy újabb paraméter. Ha ez túl alacsony, akkor limitálja a létrehozandó jel frekvenciáját és minőségét. Szerencsére ma az integrált áramkörök világában ez ritkán jelent problémát.

2.4 **Szűrő**

A lánc második eleme a szűrő. Alapvető feladata az oszcillátor által generált jel nem kívánt tartományainak eltávolítása. Az oszcillátorokhoz hasonlóan kezdetben analóg módon történt a szűrés, majd a technológia fejlődésével lehetővé vált a digitális megvalósítása is. Az analóg szűrők költséghatékonyak; hátrányuk viszont, hogy viszonylag pontatlanok. A digitális szűrők ezzel szemben igen pontosak, de megvalósításuk igencsak erőforrásigényes tud lenni.

Dolgozatomban csak az alapvető szűrőket fogom megvalósítani:

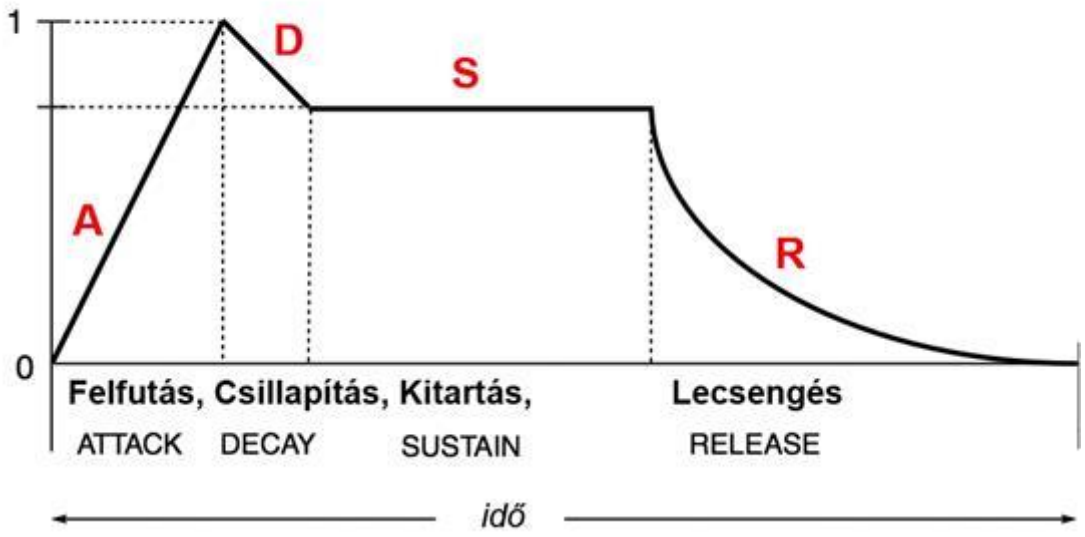
- Aluláteresztő szűrő

- Felültáteresztő szűrő
- Sávátereszt szűrő

2.5 Burkológörbe

A következő elem a burkológörbe megvalósítása. A burkológörbe a beérkező jel valamely paraméterének időbeli lefolyásába avatkozik bele. Jelen esetben a jel amplitúdója a beavatkozás célpontja. Itt a bemenet a már készre formázott jel és ennek a jelnek a hangszíne és hangmagassága már nem változik. Az alábbi paraméterek alapján lehet a jel amplitúdóját változtatni:

1. Felfutás (Attack): Az az idő, ami a csendtől a jel maximumáig telik el, miután a billentyűt leütöttük.
2. Csillapítás (Decay): A jel maximumától a kitartás szintjéig eltelt idő.
3. Kitartás (Sustain): Az a hangerősség, amit egészen addig tartunk, amíg a billentyűt lenyomva tartjuk.
4. Lecsengés (Release): Az az idő, ami a billentyű elengedésétől számítva a jel teljes lecsengéséig telik el.



2.1 ábra: ADSR burkológörbe. (Forrás: [9])

2.6 Alacsony frekvenciájú oszcillátor (LFO)

Némileg kilóg a sorból az LFO: bár egy oszcillátor, de nem feladata hang előállítása. Helyette más elemek paramétereinek modulációja a célja. Épp ezért a frekvencia tartománya szűk és alacsony, 0-tól 20 Hz-ig terjed. Ez nagyban leegyszerűsíti a megvalósítását is, hiszen ilyen alacsony frekvencia tartományt egyszerűbb algoritmusok segítségével is létre lehet hozni.

2.7 Virtual Studio Technology (VST)

A VST egy audio plugin interfész, amely lehetővé teszi különböző szoftveres szintetizátorok és audio effektek csatlakoztását különböző digitális audio

munkaállomásokhoz. A Steinberg audio szoftver és hardver cég fejlesztette ki, és megjelenése óta a zenei iparban szinte sztenderd lett. Az első verziója 1996-ban, a 2.0-ás verzió 1999-ben, a 3.0-ás pedig 2008-ban jelent meg.

3 IRODALOMKUTATÁS

A következő fejezetekben az irodalomkutatás során fellelt anyagok, tanulmányok és szakdolgozatok röviden bemutatásra kerülnek. Az itt áttekintett anyagok képezik a megvalósítás alapját, segítségükkel megismertem a szintetizátor elkészítéséhez szükséges elméletet és betekintést nyertem a digitális jelfeldolgozás témakörébe. Továbbá segítséget nyújtottak, hogy a megvalósítás során milyen algoritmusokat, programnyelveket és keretrendszereket lehetséges használni és milyen szempontok alapján érdemes mérlegelni a kiválasztásuk során.

3.1 Analóg szintetizátor modellezése különböző oszcillátor algoritmusokkal [1]

Ambrits Dániel dolgozata először egy történeti áttekintést ad az analóg szintetizátorok kialakulásáról, majd ezek után nagy vonalakban leírja az analóg szintetizátorok részeit és azok működését. Utána kidolgozza az egyes részek megvalósításának lehetőségeit.

A szakdolgozat nagy hangsúlyt fektet az erőforrás-hatékony, átlapolódás-mentes oszcillátor algoritmusok kutatására és megvalósítására. A második fejezetben bemutatja a kikutatott irodalom alapján, hogy milyen algoritmusok segítségével tudunk feszültség-vezérelt oszcillátorok által generált, felharmonikusokban gazdag periodikus jeleket (fűrészjel, háromszögjel, négyszögjel) szimulálni, amik a szubtraktív szintézis során felhasználhatóak. Ezeket a megvalósításokat az alábbi módon csoportosítja:

1. Triviális előállítási mód
2. Sávkorlátozott megoldás
    a. Additív szintézis
3. Kvázi-sávkorlátozott megoldások
    a. Bandlimited Impulse Trains
    b. Bandlimited Step Sequences
    c. Polynomial Bandlimited Step Function
4. Spektrum meredekségét növelő algoritmusok
    a. Differentiated Parabolic Waveform (DPW)
    b. Polynomial Transition Regions (PTR)

Mindegyik előállítási módnál bemutatja a módszerek matematikai leírását, szimulációk segítségével megmutatja az algoritmusok erőforrás-igényét és átlapolódását. Ezután átalakítja a DPW és PTR algoritmusokat oly módon, hogy PWM jel segítségével modulálhatóak legyenek. A PWM jel kitöltési tényezőjét később egy LFO oszcillátor segítéségével modulálja.

A következő részben összehasonlítja a DPW és a PTR algoritmusokat számítási-igény szempontjából, és ennek alapján a DPW algoritmust választja a VST szintetizátor oszcillátor algoritmusának.

A dolgozat 3. fejezetében a szűrő megvalósítására tér rá. Itt egy feszültség-vezérelt analóg Moog-szűrő digitális létrehozását írja le. Ez a szűrő 4 sorba kapcsolt elsőfokú aluláteresztő szűrőt tartalmazott, a rezonancia mértéke szabályozható volt. Emellett képes volt a sajátrezgésre is, így oszcillátorként is használható. A digitális megvalósítás esetén

ugyanezeket a paramétereket modellezi. Az irodalom alapján bemutatja a szűrő digitális modelljét és a különböző paraméterek megvalósítását. Hasonlóan az oszcillátor algoritmusokhoz, itt is különböző szimulációk segítségével ismerteti a szűrő modelljének viselkedését és átvitelét különböző frekvencia-tartományokban. Ezután megvizsgálja a töresponti frekvencia modulációjának a lehetőségét egy LFO segítségével.

A 4. fejezetben az ADSR burkológörbe és az LFO megvalósítását írja le. Az ADSR esetén az irodalom alapján arra a következtetésre jut, hogy a burkológörbét egy 4 állapotú állapotgéppel lehet reprezentálni. LFO esetén arra jut, hogy elegendő a jelet triviális módon generálni. Továbbá felhívja a figyelmet, hogy ne hívjuk meg a sin() függvényt minden egyes minta esetén, ez felesleges számításigényt generál.

Ezek után a kikutatott algoritmusok segítségével megvalósítja a szintetizátor modelljét VST plugin formájában. A fejlesztéshez a Steinberg VST SDK-et használta.

Az utolsó részben összefoglalja a dolgozatot és javaslatot tesz a további fejlesztési lehetőségekre.

3.2 **Hammond orgona-szintetizátor megvalósítása [2]**

Budai Benjámin dolgozata kísérletet tesz egy Hammond orgona szoftveres megvalósítására. A szakdolgozat 1. és 2. fejezetében leírja az orgona történetét, működési elvét, majd ismerteti a hangkeltés módját. Ebben leírja, hogy a hangkeltés alapjául egy mágneses hangszedő szolgál, amelynek a közelében egy fém lekerített fogaskerék forog, az úgynevezett tonewheel. Ezek a kerekek szinuszos váltakozó áramot indukálnak, amelyek összeségében egy additív jelgenerálást valósítanak meg. Egy hanghoz több fogaskerék is tartozik, ezek generálják a felhangokat. A mechanikus működés miatt ez a jelgenerálás közel sem tökéletes, de ez adja az orgona jellegzetes hangját.

A 3. fejezetben megalkotja az orgona modelljét MATLAB környezetben. Itt sorra veszi a különböző jelgenerálási módszereket:

- Additív szintézis
- Szubtraktív szintézis
- Hullámtáblás szintézis
- Fizikai alapú szintézis

A Hammond orgona modell megalkotásához az additív szintézist használja, hiszen a maga az orgona is mechanikus additív szintézist használ a hang megalkotásához. Az orgona jellegzetes hangjához szükséges Leslie-hangszóró modellezéséhez fizikai alapú szintézist alkalmaz. A másik szükséges hanghatást, a Percussion hangzás megvalósításához pedig a második és a harmadik felharmonikust egy speciális amplitúdó burkológörbével kell hozzáadni a generált hanghoz.

A 4. fejezetben a modell alapján megalkotja az orgonát VST plugin formájában. Itt ismerteti a szükséges technológiákat (VST, MIDI) és a fejlesztői környezetet. A fejlesztés során a Steinberg-féle VST SDK-át használja, itt egy meglévő plugin vázát felhasználva implementálja az orgona modelljét.

Az 5. fejezetben modellezi a Leslie-hangszórót, és a létrehozott modell alapján akusztikai végeselem módszerrel szimulálja a hanghatást. Itt két megoldást tesztel, egy terelővel és terelő nélkül felszerelt hangszórót. Ezután kiértékeli a két megoldás szimulációjának az

eredményeit, a szimulált hangteret és az amplitúdó- és fázis-iránykarakterisztikákat. Ezután a beépíti a szimulációt a meglévő VST pluginba.

A 6. fejezetben összefoglalja a dolgozat eredményeit és számba veszi a továbbfejlesztési lehetőségeket.

3.3 Numerikusan vezérelt oszcillátorok (Numerically Controlled Oscillators) [3]

Az első részben ismerteti magát a módszert; a numerikusan vezérelt oszcillátor szinusz és egyéb periodikus jelek előállítására szolgál. Ez a fajta megvalósítás gyakori megoldás a hardveres és szoftveres oszcillátoroknál. Amellett, hogy hatékony, lehetővé teszi az azonnali frekvencia módosítást.

A második részben leírja az eljárást matematikailag: egy periódust egy tömb segítségével reprezentálunk, amelynek egy adott elemét az alábbi egyenlettel lehet megvalósítani:

$$y[n] = \cos(2\pi f n + \theta_0) \quad (1)$$

Ebben az esetben a frekvencia nem változik, hanem fixnek tekinthető.

Ha a frekvencia vagy a fázis változik a jel során, akkor az alábbi rekurzív egyenlet használható:

$$\Phi_1[n] = \Phi_1[n-1] + 2\pi f[n] \quad (2)$$

$$\Phi[n] = \Phi_1[n] + \theta_0[n] \quad (3)$$

ahol Φ[n] az adott értékhez tartozó fázist jelöli. A kapott értékeket végül egy lookup-table segítségével konvertáljuk a megfelelő értékre.

Végül javaslatot tesz a megvalósítandó részek különböző paramétereire, mint például a használt változók típusára és méretére, ismerteti a módszer pontosságát és ötleteket ad az algoritmus hatékonyságának növelésére.

3.4 Szintetizátor tervezése és implementálása (Design and Implementation of a Sound Synthesizer) [4]

Ez a diplomamunka egy szoftveres szintetizátor teljes megvalósítását bemutatja. Az elején áttekinti a hangszintézis egyes részeit és bemutatja az ott felhasznált technikákat:

- Additív szintézis
- Szubtraktív szintézis
- Modulációs eljárások
- Hullámformázás (Waveshaping)
- Karplus-Strong algoritmus
- Hullámtábla szintézis
- Az előző technikák kombinációja

A következő részben sorra veszi az effektek megvalósítását:

- Idő-alapú környezeti effektek
- Modulációs effektek
- Szűrők
- Dinamika és erősítés
- Sztereó és panoráma

A következő fejezet a különböző vezérlőjeleket tekinti át:

- Amplitúdó-burkológörbe
- LFO-k
- és ezek keverése mátrix eljárással

Ezután rátér a szintetizátor szoftveres és hardveres implementáció problémáira és azok megoldásaira. Ezek lehetnek:

- Az audio és kontroll jelek keverése
- Az oszcillátorok számának optimális megválasztása
- A szintetizátor implementációjának formájának megfelelő kiválasztása
- A különböző pluginok belső felépítése
- MIDI működése az egyes pluginokban

Ezek alapján javaslatot tesz egy példa szintetizátor paramétereire.

A következő részben elkészíti a szoftver egyes részegységeinek specifikációját az Európai Űrügynökség által használt ESA PSS-05-0 szoftverfejlesztési folyamat alapján. Miután a szoftver követelményeit rögzíti, a követelményekre támaszkodva elkészíti a szoftver architektúrális modelljét. Itt UML diagramok segítségével felvázolja a rendszer felépítését és sorra veszi a program egyes fázisaiban előálló problémákat és azok megoldásait:

- Inicializáció
- Termináció
- Működésképtelenség

Ezekután elkészíti az alábbi részegységek architektúrális modelljét:

- Központi osztályok (a szintetizátor inicializásáért és terminálásáért felelős osztályok)
- Eseménykezelés
- Jelfeldolgozás
- Paraméterek kezelése
- Grafikai felület
- Fájlok kezelése
- Beállítások kezelése

A modell elkészülte után végül implementálja a szoftvert a specifikáció alapján VST plugin formájában. A kész szintetizátort teszteli az alábbi paraméterek alapján:

- Hang analízis
- Teljesítmény analízis
- Forráskód analízis

3.5 **FM szintetizátor megvalósítása VST környezetben [6]**

Makovecz Ádám dolgozatának célja egy frekvencia-moduláción alapuló szintetizátor megírása és a hozzátartozó absztrakt módszerek ismertetése. Ehhez először bemutatja a három fő absztrakt módszert:

1. Karplus-Strong módszer

2. Waveshaping
3. FM szintézis

A továbbiakban az OPL3 chip modellezését tűzi ki célul, ami egy FM szintézisen alapuló hangkártya chip. Ennek érdekében bemutatja a chip egyes részeit, amelyen hasonlóak a szubtraktív szintetizátornál használt egységekhez.

A hangkártya 4 oszcillátort tartalmaz, mindegyik oszcillátor 8 féle hullámformát tud generálni. Az oszcillátoroknak 6 összeköttetési módja van, két összeköttetési mód 2 oszcillátort, a maradék négy pedig mind a 4 oszcillátort használja. A kimeneti jel egy burkológörbe-generátoron halad át. A fejezetben leírja a burkológörbe generátort elvi működését is.

A következő részben sorra veszi az egyes kapcsolásokat, amelyek különféle hullámformákat hoznak így létre. Ezek után a kapcsolásokat MATLAB-ban szimulálja, modellezve az egyes jelalakok lefutását amplitúdó és hullámforma tekintetéből.

A dolgozat következő részében bemutatja plugin megírásához felhasznált technológiákat. Először ismerteti a VST technológiát és annak történetét. Utána bemutatásra kerül a WDL-OL technológia, amely a beépített könyvtáraival megkönnyíti a pluginek írását. A potenciométerek megvalósításához a KnobMan nevű programot használja.

Végül az ismertetett technológiák segítségével megalkotja a szintetizátort és annak kezelőfelületét C++ nyelven és dokumentálja az egyes lépéseket. Az elkészült programot teszteli és összehasonlítja az interneten található többi FM szintetizátorral.

3.6 The Scientist and Engineer's Guide to Digital Signal Processing [8] (Mérnökök és kutatók kalauza a digitális jelfeldolgozáshoz)

A könyv arra a nem könnyű feladatra vállalkozik, hogy viszonylag röviden áttekintést adjon a digitális jelfeldolgozás alapvető témaköreiről és azok felhasználásairól. Ennek érdekében témakörönként ismerteti az általános koncepciókat és a hozzátartozó elméletet. Az alábbi témaköröket taglalja:

- Általános bevezetés a jelfeldolgozásba
- A zaj és statisztika kapcsolata
- Analóg-digitális és digitális-analóg jelátalakítók
- Jelfeldolgozó szoftverek
- Lineáris rendszerek
- Konvolúció és tulajdonságai
- Diszkrét Fourier-transzformáció
- Gyors Fourier-transzformáció
- Folytonos jelfeldolgozás
- Digitális szűrők
- Rekurzív szűrők
- Chebyshev szűrők
- Hangfeldolgozás
- Képfeldolgozás
- Jelfeldolgozás és neurális hálók
- Tömörítés

- Jelfeldolgozó processzorok

A könyv segítségével áttekintést kaptam a szűrők működéséről, csoportosításukról, matematikai leírásukról. Segítséget nyújtott még a konvolúció megértésében is.

3.7 **Hangdizájn, hangszintézis és hangátalakítás [9]**

A könyv bemutatja a hangszín létrehozásának különböző módjait, illetve ismerteti az ehhez kapcsolódó szintézis és jelfeldolgozási technikákat.

Az 1. fejezetben definiálja a hang és a hangszín fogalmát, bemutatja a legelterjedtebb módszereit a hang ábrázolásának: a frekvencia és amplitúdó spektrumot. Továbbá bemutatja a hang alapvető fizikai paramétereit is.

A 2.-tól a 7. fejezetig bemutatja, hogy milyen módszerekkel lehet hangot szintetizálni. Ezek a módszerek az additív, szubtraktív, granuláris és formáns szintézis, valamint az adott hangszer fizikai modellezése. Az említett módszereket bemutatja, majd példákkal illusztrálja.

A 8. és 9. fejezet a különböző modulációs technikákat mutatja be, az amplitúdó és a frekvencia-modulációt. Ide tartozik a szintetizátorban használt LFO is. Ezek egyaránt tekinthetőek szintézis és jelfeldolgozási technikának is. A 11. és 12. fejezetek olyan technikákat mutatnak be, amelyek a hang analízisén és az ebből kinyert hang-rekonstrukción alapulnak

A 12.-14. fejezet bemutatja a leggyakoribb jelfeldolgozási módszereket és azok felhasználását különböző effektekben:

- Késleltetésen alapuló effektek
- Torzítók
- Kompresszálás és zengetés

Az utolsó, 15. fejezetben bemutatja, hogy a hangszínt milyen paraméterek alapján lehet leírni és ezek alapján hogyan lehet a megfelelő szintézis technikát kiválasztani.

3.8 **Virtuális analóg szűrők tervezésének művészete [10] (The art of VA filter design)**

Ez a könyv bevezeti az olvasót a népszerű szintetizátorokban használt szűrőknek a digitális megvalósításába és ismerteti az olvasóval az ehhez szükséges matematikai és digitális jelfeldolgozási ismereteket is. A könyv alapvetően analóg áramkörök digitális szimulálásával foglalkozik és ebből a szempontból taglalja az ehhez szükséges jelfeldolgozási ismerteket. A következőkben ismertetem a könyv témaköreit a teljesség igénye nélkül:

Az első fejezet a matematikai és jelfeldolgozási elméletről szól:

- Periodikus jelek felírása komplex formában
- Fourier-transzformáció
- Fourier-integrálás
- Dirac-delta függvény
- Laplace-transzformáció

A második fejezet az analóg egypólusú szűrőket taglalja:

- Egyszerű RC áramkör felírása differenciálegyenlettel
- Áramkör felírása blokkdiagram segítségével
- Komplex impedancia
- Amplitúdó és fáziskarakterisztikák bevezetése
- Aluláteresztő szűrés megvalósítása
- Pólusok és zérusok fogalmának bevezetése
- Felüláteresztő szűrő megvalósítása
- A többi típusú szűrő megvalósítása

A harmadik fejezetben a bevezetett analóg szűrés digitális implementálását írja le:

- Diszkrét idejű jelek
- Naiv integrálás
- Naiv aluláteresztő szűrő
- Trapéz integrálás
- Bilineáris transzformáció
- Vágási frekvencia átalakítása

A negyedik fejezetben egy népszerű szűrő típus, a state-variable filter digitális megvalósítását ismerteti:

- Analóg modell leírása
- Rezonancia fogalmának bevezetése
- Digitális modell leírása
- Normalizált sáváteresztő szűrő megvalósítása
- Felüláteresztő és sáváteresztő szűrés megvalósítása aluláteresztő szűréssel
- További szűrő típusok
- Tranziens jelenségek

A következő fejezetekben további szűrő típusokkal foglalkozik:

- Létraszűrő
- Nemlineáris szűrők
- Magasabb rendű szűrők
- Klasszikus jelfeldolgozó szűrők

3.9 Átlapolódásmentes oszcillátorok szubtraktív szintézisben [11] (Antialiasing Oscillators in Subtractice Synthesis)

A korai digitális szintetizátoroknál az 1970-es és 1980-as években probléma volt a szubtraktív szintézis implementálása, mert hiányoztak a megfelelő hatékony algoritmusok. A cikk bemutatja, hogy hogyan állíthatunk elő hatékonyan digitálisan jeleket a szubtraktív szintézishez.

A digitális oszcillátorok megvalósításánál az egyik legnagyobb nehézség, hogy hatékony, kis számításigényű, átlapolódásmentes algoritmusokat találjunk. Ennek szemléltetéséül egy triviális módon előállított fűrészfog jelet mutat be a szerző: ennek amplitúdó és frekvencia spektrumán nagy átlapolódások látszanak. Az átlapolódások számtalan módon csökkentik az előállított jel minőségét: inharmonikus frekvenciák, pulzálás és amplitúdó-torzítás jelenik meg. A következőkben ezek kiküszöbölését tekinti célnak a szerző.

A tanulmány 3 csoportra bontja az átlapolódás-mentes algoritmusokat:

1. Sávlimitált algoritmusok, melyek fix számú harmonikusokat állítanak elő
2. Kvázi-sávlimitált algoritmusok, melyek egy folytonos függvény jelét vezetik át egy aluláteresztő szűrőn
3. A jel meredekségének szabályozásán alapuló algoritmusok

Az 1. csoportba tartozik az additív szintézis, ami gyakorlatilag a Fourier-transzformáció megfordítása. Ezzel a módszerrel egy ideális zajmentes jelet, ez esetben fűrész jelet lehet generálni. Sajnos ennek hátránya a nagy erőforrás igény; a harmonikusok számával lineárisan növekszik a számítási költség. A szinusz jelek explicit összeadását elkerülendő, az alábbi alternatívák lehetségesek:

- Diszkrét addíció
- Zárt alakú addíció
- Hullámtábla szintézis
- Csoport-additív szintézis
- Multiráta-additív szintézis
- Gyors Fourier-transzformáción alapuló szintézis

Mindegyik módszer rendelkezik előnnyel és hátránnyal: általánosságban kijelenthető, hogy a számítási igény csökkentésével nő a memória-igény.

A 2. csoportba azok az algoritmusok tartoznak, ahol az átlapolódás szabályozható egy szűrő segítségével. Ebben a csoportban található a BLIT algoritmus (bandlimited impulse train), a BLEP (bandlimited step sequences) és a MinBLEP algoritmus (minimum bandlimited sequence). Ezen algoritmusok közös tulajdonsága, hogy mindegyik a Dirac-fésű integrálásán alapszik. Az integrálás módja az adott algoritmustól függ, BLIT algoritmus esetén egy aluláteresztő szűrővel történik, MinBLEP esetén egy minimum-fázis szűrővel.

A 3. csoportba tartozó algoritmusok az átlapolódást a jel meredekségének változtatásával csökkentik. Emiatt egy exponenciális lecsengést applikálnak a felharmonikusokra; miután ez megtörtént, egy digitális szűrővel visszaállítják a jel eredeti meredekségét. Ebbe a csoportba tartozik a DPW algoritmus (differentiated parabolic waveform). Ennek legegyszerűbb esete a fűrészfog jel előállítása. Ebben az esetben egy triviálisan megalkotott fűrészjel minden ciklusban differenciálásra kerül majd skálázódik egy konstanssal. Ennek fejlesztése, amikor a jelet a mintavételezési frekvenciánál többször kerül generálásra majd szűrésre: ekkor átlapolódás-mentesebb jelet lehet eredményül kapni.

A cikk a továbbiakban összehasonlítja a leírt módszereket NMR alapján (noise-to-mask ratio), és összesíti ezeket a módszereket számítási igény, memória felhasználás és az előállított jel minősége alapján.

3.10 Oszcillátor és szűrő algoritmusok virtuális analóg szintetizáláshoz [12] (Oscillator and Filter algorithms for Virtual Analog Synthesis)

A tanulmány a virtuális analóg szintetizátoroknál használt oszcillátor algoritmusokat mutatja be. Ezek a technikák a 60-as és 70-es évek analóg szintetizátorok hangzását próbálják utánozni. Ez gyakorlatban a szubtraktív szintetizátor jelenti.

A digitális oszcillátorok megvalósításánál alapvető probléma, hogy a geometrikus jelek (fűrészfog és négyszög) sarkai átlapolódást okoznak. Ennek kiküszöbölésére a tanulmány 3 megoldást sorol fel:

1. Sávlimitált megvalósítások
2. Kvázi-sávlimitált megoldások
3. Átlapolódást elnyomó megvalósítások

Az 1. megoldás a Nyquist-frekvencia alatt generál harmonikusokat. Ilyen például az additív szintézis. Ezek hátránya az óriási számításigény.

A 2. megoldásnál az átlapolódás viszonylag kicsi, és az átlapolódás szintje változtatható azért, hogy számítási erőforrásokat takarítsunk meg.

A 3. megoldásnál bár előfordulhat átlapolódás, de csillapítva van. A cikk ezzel a megközelítéssel foglalkozik. Ezt a DPW (differentiated parabolic waveform) algoritmus segítségével éri el. A továbbiakban különböző hullámformákon keresztül mutatja be a DPW alkalmazását.

Fűrészfogjel esetén először az aktuális fázis kerül kiszámításra, amelyet egy modulo számláló segítségével valósít meg. Ez a számláló 0-tól 1-ig számlál. Ebből létrehozásra kerül egy bipoláris modulo számláló, amely -1-től 1-ig számlál. Minden ciklusban a négyzetére emelődik a bipoláris modulo számláló értéke, majd ez az érték kivonásra kerül az előző ciklus bipoláris modulo számláló érték négyzetéből, tehát a különbségüket vesszük. Ez az érték eltárolásra kerül a következő ciklus számára. Majd végül a kapott értéket egy konstanssal kell megszorozni, azaz:

$$c = \frac{fs}{\left[4f\left(1 - \frac{f}{fs}\right)\right]} \tag{4}$$

ahol fs a mintavételezési frekvenciát, f pedig a hang frekvenciáját jelöli.

A cikk ezután a változtatható kitöltési tényezőjű impulzusjel megvalósítását mutatja be DPW algoritmussal. Ennek egy speciális esete a szintetizátorban használt négyszögjel, ahol a kitöltöttségi tényező 50%-os. Ebben az esetben két fűrészfogjel generátort érdemes használni, amelyek egymástól 180°-os késésben vannak.

A következő jel a sorban a háromszögjel. Ebben az esetben az átlapolódás nem olyan súlyos probléma, akár egyszerű triviális megoldással is jó eredményt lehet kapni. DPW megvalósítás esetén először egy triviális módon létrehozott fűrészfog jelet kell létrehozni. Következő lépésként a jel értékeit a négyzetére kell emelni. Ezután a kapott jel szorzásra kerül egy négyszögjellel. Majd a kapott érték differenciálásra kerül és végül a már ismert

$$c = \frac{fs}{\left[4f\left(1 - \frac{f}{fs}\right)\right]} \tag{5}$$

egyenlettel megadott c konstanssal szorzásra kerül.

Az utolsó részben a tanulmány egy rezonancia-szűrő digitális megvalósításával foglalkozik. Ez a fajta szűrő 3 módban különbözik egy általános végtelen impulzus válasz szűrőtől:

1. A paraméterek gyorsan változnak
2. A szűrő fokszáma előre meghatározott
3. Egy változtatható rezonancia pont található a vágási frekvencia mellett

Ezekután felsorolja egy optimális digitális rezonancia-szűrő kritériumait:

- Az együtthatók frissítése gyorsan történjen.
- A szűrő vágási frekvenciája és rezonanciája egymástól függetlenek kell, hogy legyenek.
- A szűrőnek stabilnak kell maradnia mindaddig, amíg a paramérek a meghatározott tartományon belül mozognak.
- A szűrőnek hasonlóan kell viselkednie egy analóg szűrőhöz.
- A szűrőnek képesnek kell lennie sajátrezgésre.

Ezek alapján a szerző leírja a Moog szintetizátor (1965) létra-szűrőjének egy digitális megvalósítását.

4 KONKLÚZIÓ

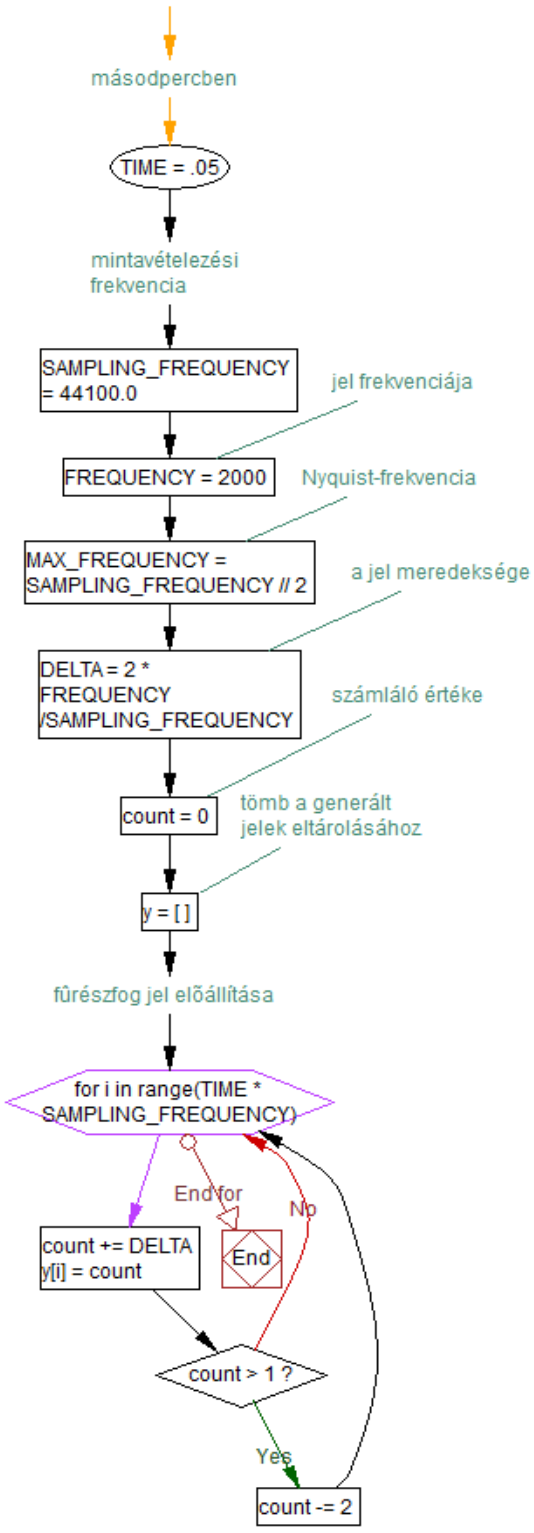
4.1 Oszcillátor

4.1.1 A megfelelő algoritmus kiválasztása

Ahogy az irodalomkutatásnál már említettem [11], a digitális oszcillátorok megvalósításánál a keletkező zaj okozza a legnagyobb problémát. Ez a probléma csak a felharmonikusokban gazdag jeleknél okoz problémát, egyszerű szinusz és koszinusz jeleknél nem. Egy fűrészjel előállításának példáján keresztül bemutatom a létrejövő zaj problémáját. A mintavételezési frekvencia 44,1 kHz.

4.1.2 Fűrészfogjel

A triviális megoldás [11] esetén egy számláló értékei adódnak össze minden mintavételezési periódusban. A számláló minden mintavételezési intervallumban $2f/fs$ értékkel inkrementálódik, ahol az f generálni kívánt frekvenciát, az fs pedig a mintavételezési frekvenciát jelenti. Ha a számláló értéke elérné az 1-et, akkor 2 levonásra kerül belőle.

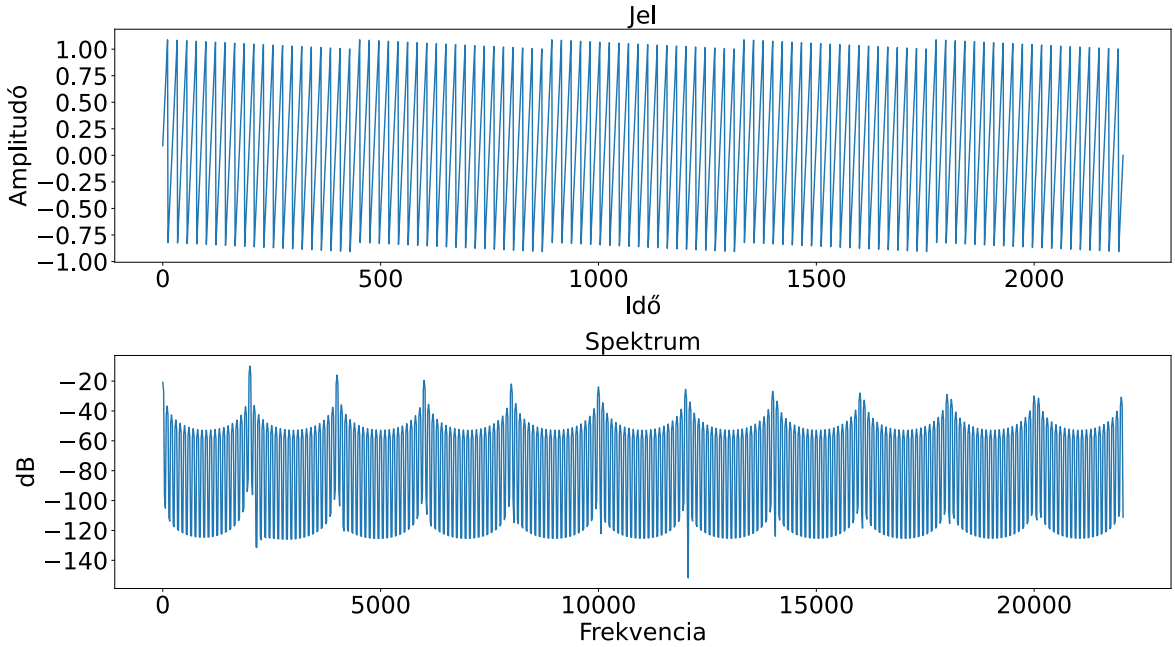


4.1 ábra: Triviálisan előállított fűrészjel folyamatábrája

Ez az előállítás bár algoritmikai és számításigény szempontjából hatékony, mégse használható teljeskörűen. Ennek az az oka, hogy csak szűk tartomány esetén lehet

viszonylag zajmentes jelet generálni. (Az LFO esetében ez a módszer használható, mert alacsony frekvencia esetén az átlapolódás mértéke csekély.)

Az 4.1-es ábrán bemutatásra kerül a generált jel és annak spektruma. Látható, hogy triviális módszerrel előállított fűrészjel spektrumában nagy átlapolódások vannak, emiatt nagy zaj keletkezik. Ideális esetben a csak az alaphang és annak felharmonikusai képződnének (a 4.1-es ábrán ezek a csúcsok a spektrumban).



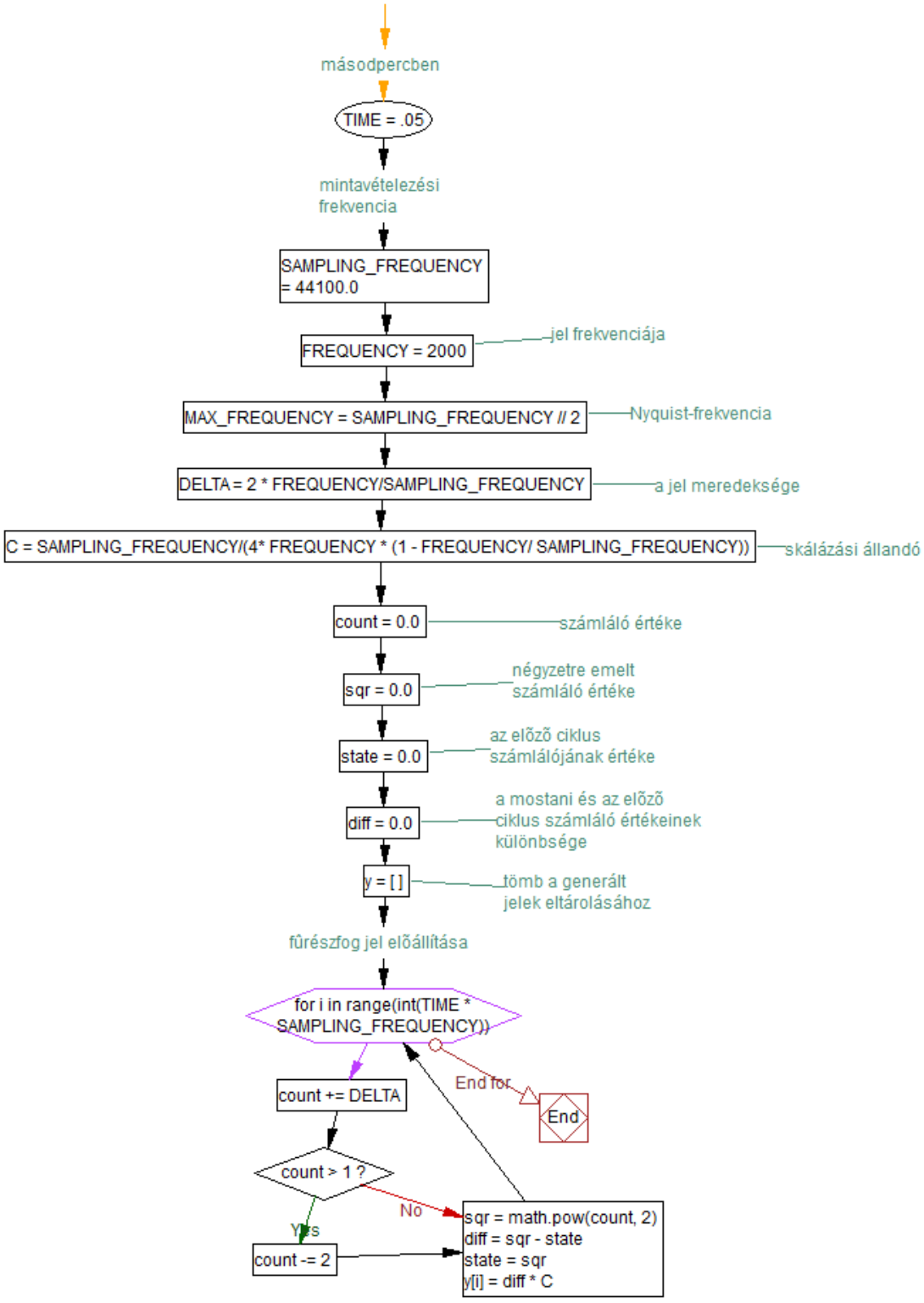
4.2 ábra: Triviálisan generált fűrészjel és spektruma.

A DPW algoritmus jóval erőforrás hatékonyabb jelgenerálást tesz lehetővé mint az előbb bemutatott triviális megoldás [11]. Hasonlóan a triviális megoldáshoz, először létrehozásra kerül egy számláló és ez a számláló minden mintavételezési ciklusban $2f/fs$ értékkel inkrementálódik. Ha eléri az 1-et, akkor 2-őt levonásra kerül belőle és újra kezdődik a ciklus.

A különbség abban rejlik, hogy minden ciklusban a számláló értéke a négyzetére emelődik, majd ez az érték levonásra kerül az előző ciklus értékből. Az így kapott értéket végül szorzódik egy előre meghatározott állandóval, amelyet a

$$c = \frac{fs}{4f\left(1 - \frac{f}{fs}\right)} \tag{6}$$

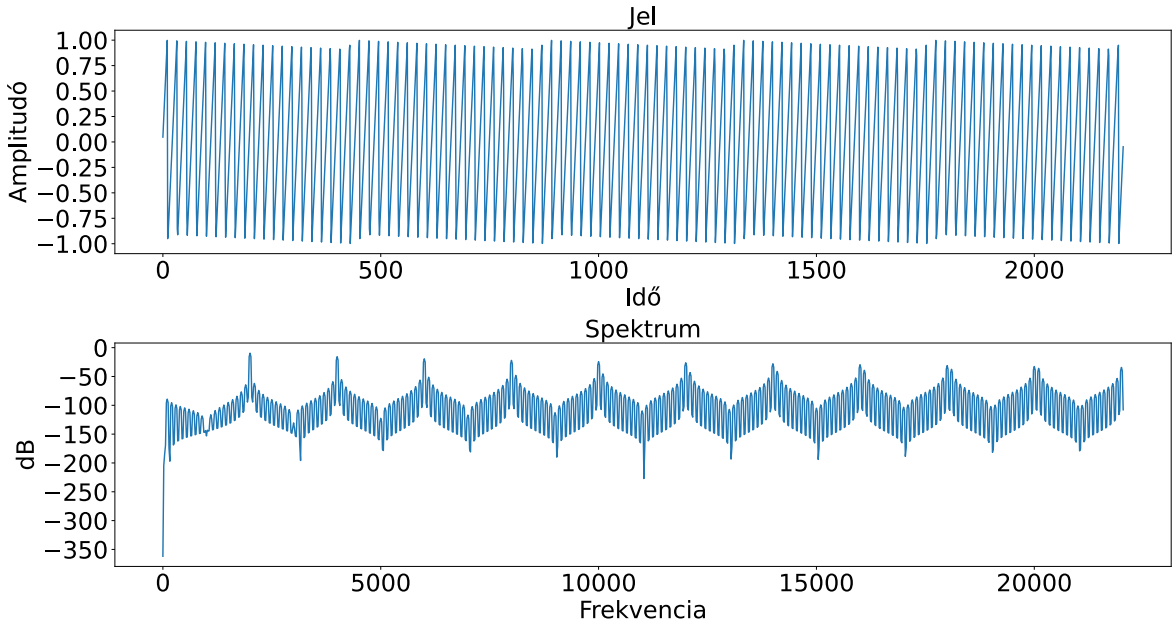
egyenlet írja le [11][12].



4.3 ábra: DPW algoritmussal generált fűrészjel folyamatábrája

A 4.4-es ábrán látható, hogy a generált jel jóval kevesebb zajt tartalmaz, mint a triviális algoritmussal generált esetben. Erőforrás-felhasználás tekintetében lényegében

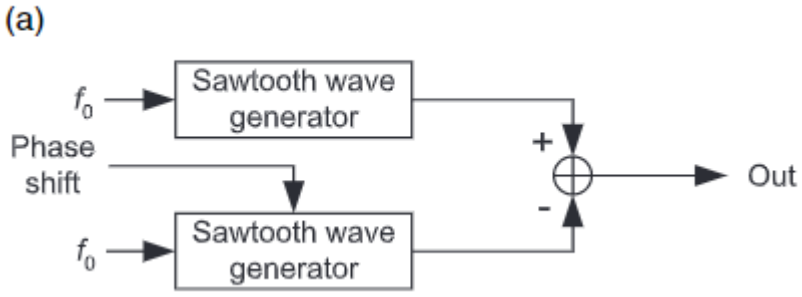
megegyezik a triviális megoldás algoritmusával. Ezért a szintetizátor megvalósítása során a DPW algoritmus segítségével generált jeleket fogom használni.



4.4 ábra: DPW algoritmussal generált fűrészjel és spektruma.

4.1.3 Négyszögjel

Egy négyszög jelet két háromszögjel segítségével lehet előállítani. Ehhez két fűrészfogjel-oszcillátort szükséges, amelyből az egyik oszcillátor jele 180°-os fáziskéséssel érkezik, majd a két jel kivonásra kerül egymásból. Ennek erőforrás-igénye így két fűrészfogjel generátoréval egyezik meg. Előnyös tulajdonságai miatt itt is érdemes DPW algoritmust használni [1][12].



4.5 ábra: Négyszögjel generálás két háromszögjel-oszcillátor segítségével. (Forrás: [12])

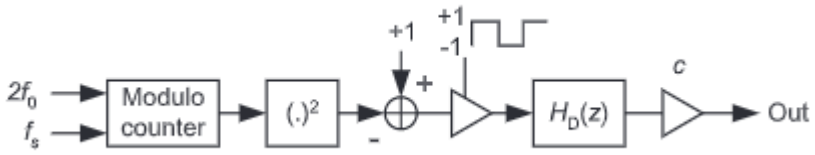
4.1.4 Háromszögjel

A háromszögjel generálásánál is használható a DPW algoritmus egy módosított változata. Ebben az esetben első lépésben generálásra kerül egy triviális fűrészfogjel. Ennek a jelnek az értéke a következő lépésben a négyzete lesz: ezért az így kapott jel csak pozitív értékeket vesz fel. Következő lépésben modulálásra kerül a jel egy négyszögjellel. Ekkor a négyszögjel pozitív periódusában a jel változatlan, a negatív periódusban pedig értéke

az ellentétjére változik. Fontos, hogy a négyszögjel és a fűrészfogjel oszcillátor frekvenciájának meg kell egyeznie. Ezek után a már megismert módon az előző ciklus és a mostani ciklus értékei kivonásra kerülnek, majd a

$$c = \frac{fs}{4f\left(1 - \frac{f}{fs}\right)} \tag{7}$$

állandóval skálázódik [1][12].



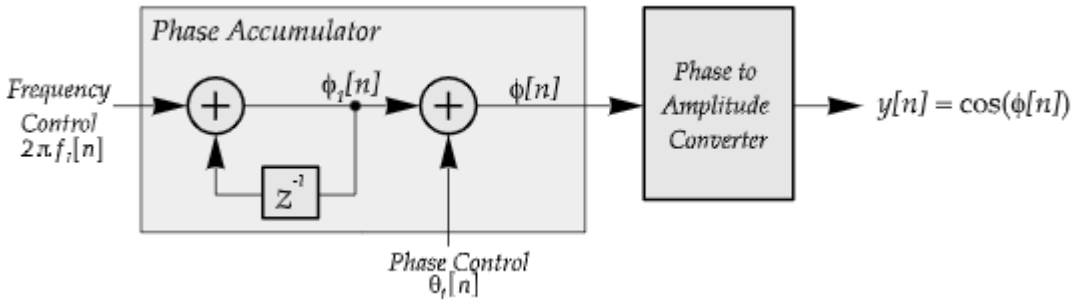
4.6 ábra: A háromszögjel generálásának blokkvázlata. (Forrás: [12])

4.1.5 Szinuszjel

A szinuszjel generálásához egy numerikusan vezérelt oszcillátort (NCO) érdemes használni; ez a módszer hatékony és elterjedt módja a periodikus jelek előállításának [3].

Az oszcillátor alapvetően két részből áll:

- Fázis akkumulátor
- Fázis-amplitúdó konverter



4.7 ábra: NCO oszcillátor blokkvázlata. (Forrás: [3])

A fázis akkumulátorban történik a beállított frekvencia szerinti fázis kiszámítása. Ez alapvetően a harmonikus rezgőmozgás egyenlete azzal a különbséggel, hogy idő helyett diszkrét értékeket használ:

$$y[n] = \sin(2\pi fn + \theta_0) \tag{8}$$

Az aktuális fázis (tehát a szinusz argumentuma) rekurzívan az előző fázisból számítódik ki és hozzáadásra kerül az aktuális frekvenciából kiszámított fázis:

$$\Phi_1[n] = \Phi_1[n-1] + 2\pi f[n] \tag{9}$$

Ezután hozzáadásra kerül az eltolási érték (ha van):

$$\Phi[n] = \Phi_1[n] + \theta_0[n] \tag{10}$$

Az utolsó lépésben a fázis-amplitúdó konverterben a kiszámított fázis egy táblázat segítségével (lookup-table) lehet átváltani a kívánt függvény értékére.

4.2 Szűrő

4.2.1 Digitális szűrők

Alapvetően kétfajta digitális szűrő létezik:

- Véges impulzus válasz szűrőt (FIR)
- Végtelen impulzus válasz szűrőt (IIR)

FIR szűrő esetén a bemeneti jel sorozatán egy csúszó ablak segítségével konvolúcióval történik a szűrés. Az alul-, felül- vagy sáváteresztést a konvolúciós függvény alakja határozza meg, amit az egyes értékek együtthatóival tudjuk reprezentálni. Az együtthatókat egy táblázat segítségével (lookup-table) lehet a megfelelő vágási frekvenciához társítani. Mivel a FIR szűrők véges mennyiségű értékekkel működnek, ezért lecsengésük is véges, és alapvetően stabilabbak és könnyebben tervezhetőek mint az IIR szűrők. Egy FIR szűrő működését a

$$y[n] = \sum_{i}^{N} a_i x[n-i] \tag{11}$$

egyenlet írja le, ahol $y[n]$ a kimeneti értékek tömbjét, a_i az i-edik bemeneti érték együtthatóját, $x[n]$ pedig a bemeneti értékek tömbjét jelenti.

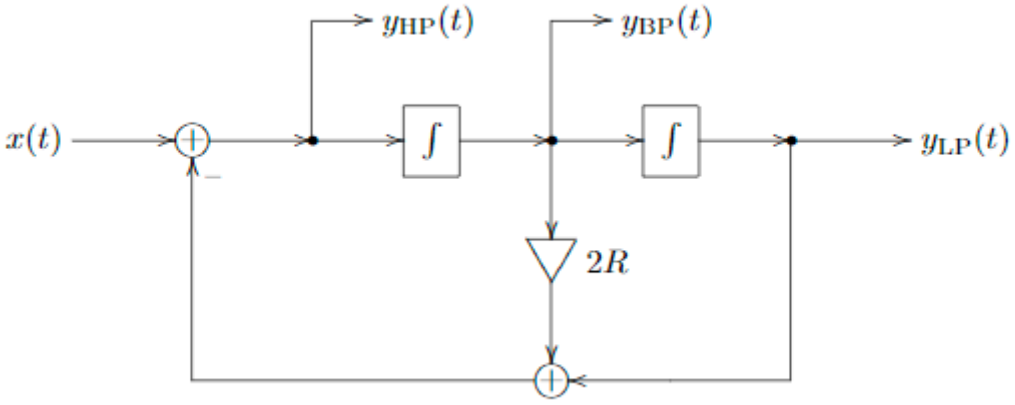
IIR szűrőnél nem a bemeneti jel értékei kerülnek felhasználásra, hanem a már kiszámított kimeneti jel értékeiből kerül kiszámításra rekurzívan az éppen aktuális érték. Ez a típusú szűrő szimulálja legjobban az analóg szűrőket. Előnyük, hogy számítási szempontból hatékonyabbak és kevesebb memóriát igényelnek, mint a FIR szűrők. Hátrányuk viszont, hogy nehezebb őket tervezni és jellegüknél fogva instabilabbak, mint a FIR szűrők. Matematikailag a

$$y[n] = x[n] + ay[n-1] \tag{12}$$

egyenlet írja le egy IIR szűrő működését, ahol $y[n]$ a kimeneti, $x[n]$ a bemeneti értékek tömbje, a pedig a szűrő együtthatóját jelenti.

4.2.2 Állapotváltozós szűrő (State-variable filter)

Ez a szűrő az egyik legelterjedtebb típus, amit a szintetizátorokban használnak, nem ok nélkül: működéséből adódóan egyszerre jelenik meg a bemenő jel alul-, felül- és sáváteresztett jele. Ezenkívül vágási frekvencia és rezonancia paraméterek azonnal, nagyobb számításigény nélkül változtathatóak. Digitális megvalósítása az analóg változat topológiáját megtartva készül, ezért ez a szűrő TPT (topologic preserved transform) típusú. (Ezt a metodológiát használják az áramkör szimulációs programok is.)



4.8 ábra: Analóg állapotváltozós szűrő blokkdiagramja. (Forrás: [10])

Az analóg modell blokkdiagramjából kiindulva lehet felírni a szükséges differenciálegyenleteket:

$$y_{HP} = x - 2Ry_{BP} - y_{LP} \tag{13}$$

$$\dot{y}_{BP} = \omega_c \cdot y_{HP} \tag{14}$$

$$\dot{y}_{LP} = \omega_c \cdot y_{BP} \tag{15}$$

Itt az y_{HP} a felüláteresztett kimenetet, a y_{BP} a sáváteresztett kimenetet, a y_{LP} az aluláteresztett kimenetet, ω_c a vágási körfrekvenciát, az R pedig a csillapítási tényezőt (rezonancia) jelenti.

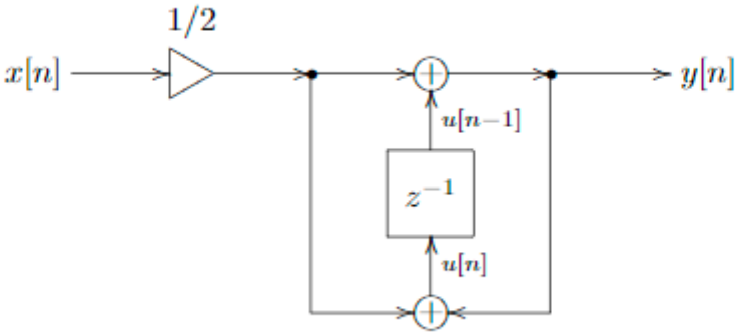
A differenciálegyenleteket átalakítva, majd rajtuk Laplace-transzformációt végezve az alábbi átviteli függvények lesznek az eredmények:

$$H_{HP}(s) = \frac{s^2}{s^2 + 2Rs + 1} \tag{16}$$

$$H_{BP}(s) = \frac{s}{s^2 + 2Rs + 1} \tag{17}$$

$$H_{LP}(s) = \frac{1}{s^2 + 2Rs + 1} \tag{18}$$

Az analóg modellről a diszkrét modellre történő áttérésnél z-transzformáció van alkalmazva. Ekkor az integráló tagok esetén többféle integrálási módszer létezik, ebből a trapézformula jó közelítő eredményt ad, kis számításigény mellett. A trapézformula az alábbi blokkdiagrammal szemléltethető:



4.9 ábra: Trapézformula integrátor blokkdiagramja. (Forrás: [10])

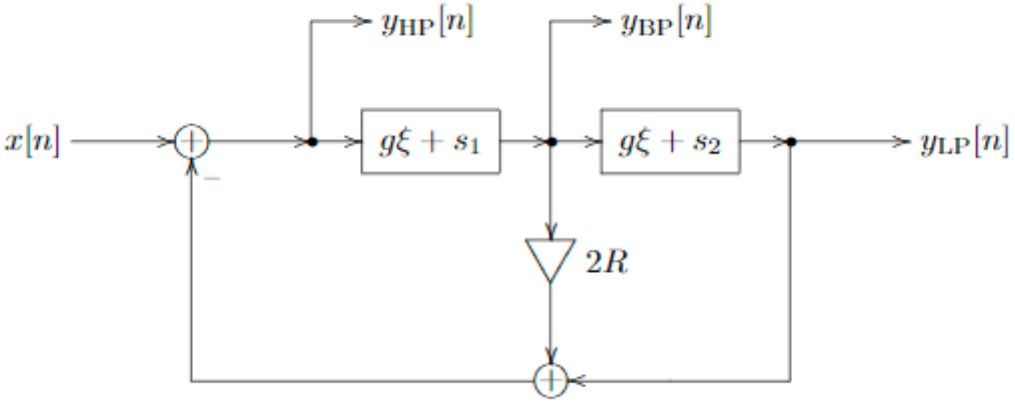
Itt az $x[n]$ a bemenet n-edik mintáját, $y[n]$ a kimenet n-edik mintáját, z^{-1} pedig egy mintányi késleltetést jelent. Az $\frac{1}{2}$ -el való szorzás a trapéz területének kiszámításából adódik. Ez a tag felírható a

$$y[n] = gx[n] + s \tag{19}$$

lineáris egyenlettel, ahol az s a $x[n-1]$-ik bemeneti mintát jelenti, a g tényezőben pedig egyesítésre kerül a trapézformula integrálásánál használt $\frac{1}{2}$ -es szorzó a vágási körfrekvenciával (ω_c) és a mintavételezési idővel (T), tehát:

$$g = \frac{\omega_c T}{2} \tag{20}$$

Ezt az integráló tagot visszahelyezve az analóg modell blokkdiagramjába az alábbi eredmény adódik:



4.10 ábra: Állapotváltozós szűrő blokkdiagramja. (Forrás: [10])

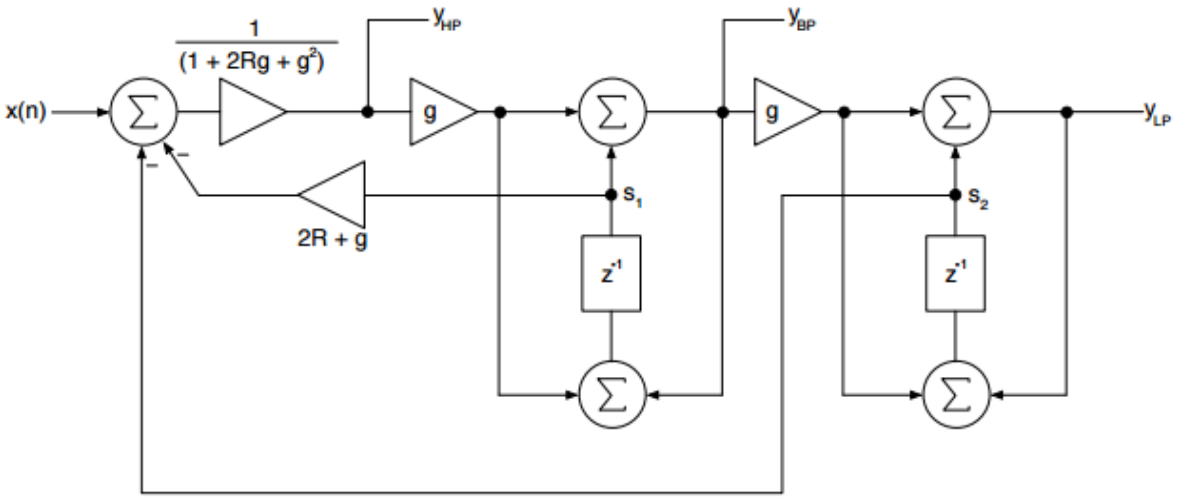
Itt a ξ az $x[n]$-et jelöli, tehát az n-edik bemeneti mintát. Ezeket felhasználva felírható a jelek egyenlete:

$$y_{HP} = \frac{x - (2R + g)s_1 - s_2}{1 + 2Rg + g^2} \tag{21}$$

$$y_{BP} = g \cdot y_{HP} + s_1 \tag{22}$$

$$y_{LP} = g \cdot y_{BP} + s_2 \tag{23}$$

A kapott egyenletek segítségével felírható a szűrő teljes blokkdiagramja:



4.11 ábra: Diszkrét állapotváltozós szűrő blokkdiagramja. (Forrás: [14])

4.3 Burkológörbe

A burkológörbét egy állapotgéppel lehet modellezni, ahol a burkológörbe 4 szakasza egy-egy állapotot reprezentál. Mivel a növekedés és a csökkenés lineáris az összes szakasz esetén, ezért két minta közti különbséget az

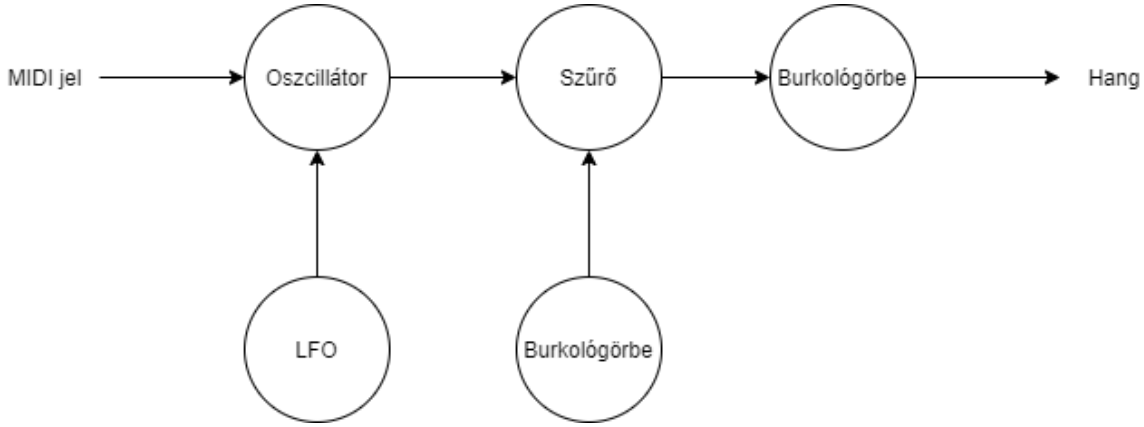
$$(f_s \cdot A_{max} \cdot A)^{-1} \tag{13}$$

egyenlettel adható meg, ahol A_{max} a maximális amplitúdót jelöli, A pedig a maximális amplitúdó bizonyos százalékát jelenti 0-tól 1-ig. Az f_s a mintavételezési frekvenciát jelöli [1].

4.4 LFO

Mivel az LFO-nak a már generált hangok modulálása a célja és frekvencia-tartománya 0-20 Hz-ig terjed, ezért nincs akkora követelmény támasztva felé, mint egy normál oszcillátor felé. De a már említett oszcillátor algoritmusok itt is ugyanúgy használhatóak.

5 RENDSZERTERV



5.1 ábra: Szubtraktív szintetizátor modellje

A szintetizátor a 5.1-es ábra szerint lesz megvalósítva, tehát az oszcillátorok, a szűrők, a burkológörbe és az LFO modellje külön osztályokban lesz implementálva. Az egyes osztályok láncszerűen fogják a generált jelet feldolgozni, tehát az oszcillátor komponens kimenete a szűrő komponens bemenete lesz és így tovább. Az LFO osztály feladata az oszcillátor osztályban generált jel modulálása lesz. A szűrőbe csatlakozó burkológörbe a szűrő paramétereinek időbeli lefolyását fogja tudni módosítani. A lánc végén található burkológörbe pedig a generált hang dinamikai jellemzőit képes változtatni. A fejlesztés során a C++ programozási nyelvet fogom használni, maga a VST API is ebben van megírva.

6 MEGVALÓSÍTÁS

A megvalósítás első lépéseként sorra vettem, hogy milyen lehetőségek állnak a rendelkezésemre a plugin megírásához. Két lehetőség állt fenn:

- Közvetlenül a VST API használata
- Egy keretrendszer használata

Először a VST API-t próbáltam használni a fejlesztés során, de rövid időn belül kiderült, hogy két kulcsfontosságú dolog hiányzik: a megfelelő mennyiségű és minőségű dokumentáció és irodalom. Emiatt hamar rájöttem, hogy a megfelelő keretrendszer kiválasztása és használata lesz a plugin sikeres megírásának a záloga. A továbbiakban sorra vettem, hogy milyen keretrendszerek vannak, és milyen szintű dokumentáció van hozzájuk. Végül két keretrendszert találtam, amelyek elég kiforrottak voltak, hogy a fejlesztést biztonságosan meg lehessen kezdeni velük:

- WDL-OL (IPlug) és IPlug2
- JUCE

Az IPlug egy nyílt forráskódú keretrendszer, amely nagy sikernek örvend a pluginok fejlesztésének területén. A dokumentációja elég részletes és sok plugin is állt rendelkezésre, amelyből a fejlesztést el lehet indítani. Végül azonban nem mellette döntöttem az alábbi okok miatt:

- Bár az első verziója kiforrott volt, de a kezelőfelületet beépítetten nem támogatta, ahhoz egy külön keretrendszert kellett volna használni. Továbbá már nem volt fejlesztés alatt évek óta, már csak a második verziót fejlesztették.
- A második verziója már támogatta beépítetten a kezelőfelület megírását, de még béta verzióban működött, és támogatása is hiányos volt.

Ezen okok miatt a JUCE keretrendszert választottam.

6.1 A JUCE keretrendszer bemutatása

A JUCE keretrendszer magja a Tracktion digitális audio munkaállomás kezelőfelületének és hangkezelésének megírásakor született meg. 2004-ben adták ki az első önálló verzióját, ekkor még nem lehetett vele plugin-okat írni. A későbbiekben fokozatosan adták hozzá a különböző funkciókat, a mostani verzió (6.0.7) már egy kiépített könyvtárral rendelkezik, amely nemcsak VST, hanem más plugin formátum (AU, RTAS, AAX) és önálló audio applikációk megírását is lehetővé teszi. A könyvtárában megtalálható az audio fejlesztéshez szükséges eszközök teljes vertikuma:

- Audio input és output kezelés
- MIDI kezelés
- Digitális jelfeldolgozáshoz szükséges osztályok és függvények
- Gyakran használt adatstruktúrák audio kezeléshez igazított változatai
- Grafikus felület megírásához és kezeléséhez szükséges osztályok és struktúrák
- OpenGL támogatás
- Projekt menedzsment

A dokumentációja és támogatása kiemelkedő, a rendelkezésre álló oktató cikkek alapján viszonylag hamar meg lehetett ismerni a felépítését és elsajátítani a használatát. Bár maga a keretrendszer fizetős, de saját projekt esetén ettől eltekintenek.

6.2 A projekt létrehozása

A JUCE keretrendszer egy saját projektmenedzserrel rendelkezik: ez a Projucer. Mivel a JUCE nemcsak a VST plugin formátumot, hanem más plugin formátumokat és önálló audio és grafikus programok megírását is támogatja, ezért a projekt kezdeti létrehozásakor itt lehetséges a különböző projekt típusok közül választani. A projekt létrehozása során kiválasztásra kerül az applikáció formátuma, a használni kívánt fejlesztői környezet és a fejlesztéshez szükséges JUCE könyvtárak.

Miután a projekt generálása megtörtént, ugyanitt a Projucer applikációban lehetséges a plugin típusát, nevét és adminisztratív adatait, valamint a kívánt C++ fordítót beállítani. Emellett a fejlesztés során itt történik a header és cpp állományok hozzáadása a projekthez. Ezen állományok hozzáadása a fejlesztői környezeten keresztül nem ajánlott, mert a változások nem regisztrálódnak a Projucer-ben, és a projekt mentése során felülíródnak a generált állományokkal.

6.3 A keretrendszer vázlatos felépítése

6.3.1 PluginProcessor

A PluginProcessor felel a plugin jelfeldolgozási és logikai részéért, emellett a grafikus interfészből érkező adatokat is kiértékeli és feldolgozza. A konstruktora inicializálja a plugint a megfelelő paraméterekkel (inputok és outputok száma, MIDI jel fogadás és

generálás engedélyezése), és tartalmaz olyan függvényeket, melyek az adott plugin beállításait mentéséért és a már mentett adatok beolvasásért felelnek. Emellett megtalálhatóak az egyes konfigurációk getter és setter függvényei is.

A `prepareToPlay()` függvényben lehet a hang lejátszása előtti szükséges paramétereket beállítani. A legfontosabb mintavételezési frekvencia beállítása, ez tipikusan a plugin elemeinek (oszcillátor, szűrő, burkológörbe) saját `prepareToPlay()` függvényeinek meghívását jelenti a megadott mintavételezési frekvenciával. Opcionálisan be lehet állítani a buffer méretét is. A `releaseResources()` függvényben erőforrásokat lehet felszabadítani, miután egy elem vagy a plugin bezáródott.

6.3.2 PluginEditor

A grafikus megjelenítésért a PluginEditor osztály felel. Itt a konstruktorban lehet hozzáadni a kezelőfelülethez az egyes elemeket, ezen elemeket beállítani, megjeleníteni az ablakot, és beállítani a szükséges paramétereket (alapértelmezett magasság, szélesség). Az elemek pozícióját a `resized()` függvényben lehet megadni, a színezésért a `paint()` metódus felel.

A kezelőfelület elemei és a hozzájuk tartozó csatlakozó elemek, mint tagváltozók vannak felvéve az osztályban. A csatlakozó elemek lehetővé teszik, hogy a kezelőfelületen bekövetkezett változások azonnal regisztrálódjanak a PluginProcessor osztályban.

6.3.3 SynthSound és SynthVoice

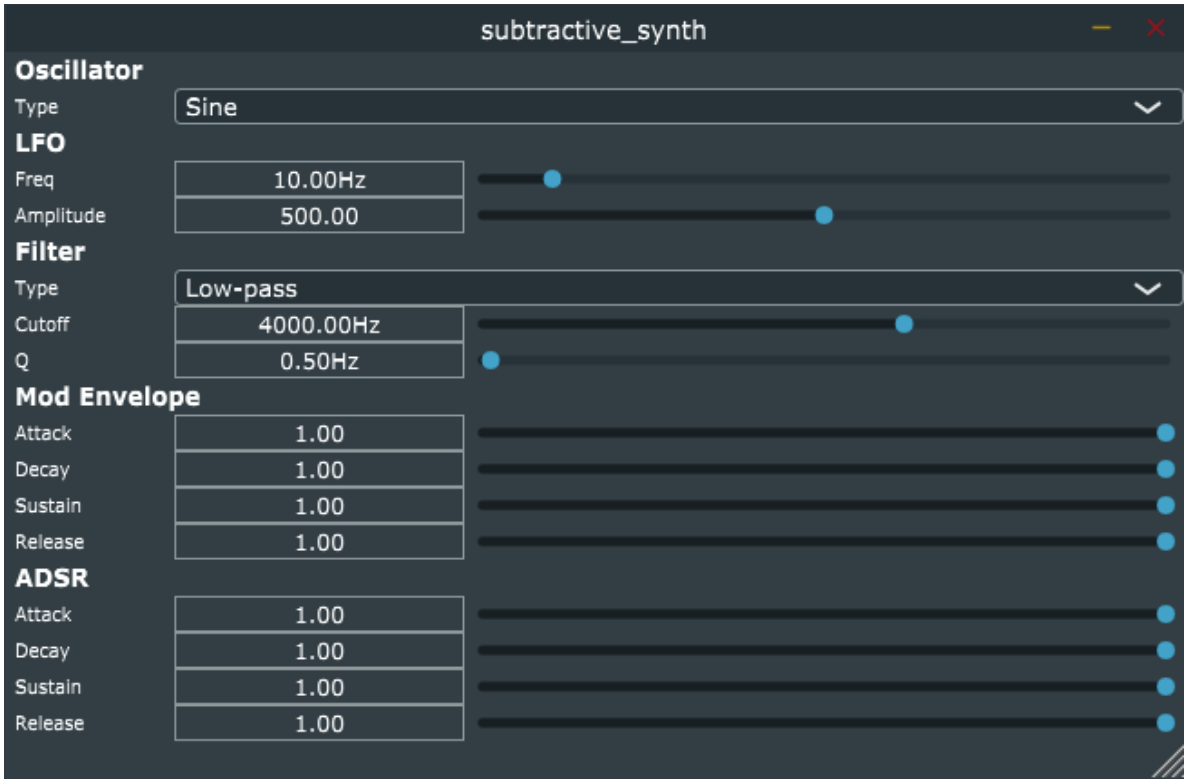
Ezen osztályok felelősek egy hang és annak szólamainak a keretbefoglalásáért. A SynthSound osztály egy hangot képvisel, a SynthVoice pedig egy szólamot. A SynthSound osztály a keretrendszer SynthesiserSound osztályából származik le, a SynthVoice pedig a SynthesiserVoice osztályából öröklődik. Mind a kettő egy Synthetiser objektum része, mely a PluginProcessor osztályban inicializálódik. Az inicializálás során annyi SynthVoice osztály jön létre, ahány szólammal rendelkezik a szintetizátor.

A SynthVoice osztály egyben a plugin kontrollere is. Itt kezelődnek le a beérkező MIDI jelek, és itt folytatódik tovább a PluginProcessor-ban meghívott `prepareToPlay()` függvény. Ezenkívül itt kapcsolódik össze a jelgenerálási lánc: a `renderNextBlock()` függvényben az egyes elemek jelfeldolgozó függvényei itt hívódnak meg és adják át egymásnak a generált jelet. A SynthSound osztály tagváltozói tehát az 5.1-es ábrán látható folyamat elemei:

- `ADSR adsr` //hang burkológörbéje
- `ADSR modEnv` //szűrő burkológörbéje
- `Oscillator oscillator` //oszcillátor
- `SVFilter filter` //szűrő

Az egyes elemek paramétereinek frissítését az update függvények végzik, ezek a függvények meghívják az adott elem paraméter beállításáért felelős függvényét az új paraméterekkel. A `startNote()` és `stopNote()` függvények a billentyű lenyomásakor és elengedésekor hívódnak meg. Ezenkívül még található két getter az oszcillátor és a szűrő objektumához, mert ezek frissítése nem itt, hanem a PluginProcessor osztályában történik.

6.4 Kezelőfelület



6.1 ábra: A szintetizátor kezelőfelülete

A szintetizátor kezelőfelületének a kialakítását a már taglalt PluginEditor osztályban lehet megvalósítani. A kezelőfelület megírásánál rácsos szerkezetet használtam: a jelgenerálás egyes elemei egymás alatt találhatóak, és a rácsokban elemtől függően annyi sort és oszlopot hoztam létre, amennyi az adott elem paramétereinek vezérléséhez szükséges.

A rácsos elrendezés előnye, hogy az ablak méretétől függően helyezi el a benne lévő elemeket, tehát nem szükséges az egyes elemek méretét és pozícióját pontosan megadni, hanem elég csak a rácson belül a sorok és oszlopok számát megadni.

6.5 MIDI jelek kezelése

A JUCE rendszer alapértelmezetten képes a MIDI jelek kezelésére; a projekt generálása után engedélyezhető a MIDI jelek fogadása és generálása is. A MIDI jelek kezelése emellett megtörténhet a PluginProcessor osztályban is: az `acceptsMidi()` és `producesMidi()` függvények a beállított makrók szerint true vagy false visszatérési értékkel térnek vissza.

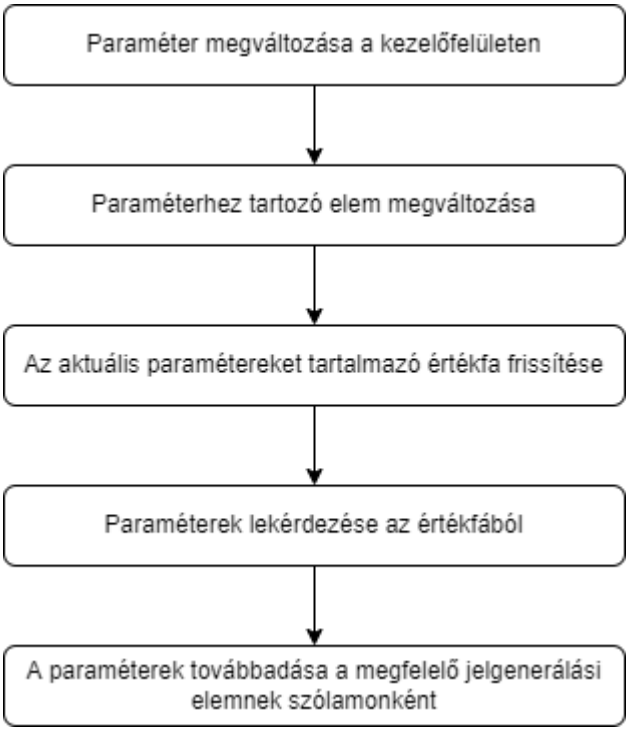
A MIDI jelek fogadásának engedélyezése után a keretrendszer osztályai már képesek kezelni a beérkező MIDI jeleket, mint függvények és metódusok paraméterei. Ez a szintetizátor megvalósítása során a SynthVoice osztály `startNote()` metódusában történik: az itt bemenő paraméterként szereplő midiNoteNumber a beérkező MIDI jel kódját jelenti, és ez kerül átadásra az oszcillátornak.

6.6 A jelgenerálási paraméterek lekérdezése és frissítése

6.6.1 ValueTreeState osztály

Ahhoz, hogy a kezelőfelületen bekövetkezett változások, tehát a PluginEditor osztály valamely elemének megváltozása regisztrálódjon a jelfeldolgozásért felelős osztályban (PluginProcessor), szükséges egy struktúra, amely mindig az aktuális paramétereket tartalmazza jelgenerálási elemekre lebontva, és összeköti a kezelőfelületet a jelfeldolgozással. Ezt a szerepet a ValueTreeState osztály tölti be, amely egy fa szerkezetű adatstruktúra, és rendelkezik az adatok felviteléhez, módosításához, törléséhez szükséges metódusokkal. Ezenkívül rendelkezik olyan elemekkel, amelyek csatlakozva a kezelőfelület elemeihez képessé teszik a paraméterek azonnal frissítését az adatstruktúrán belül. Az egyes paraméterekhez egyedi azonosító szükséges, ennek segítségével lehet az adott paramétert lekérdezni. Ez az azonosító string formátumban adható az adott elemnek.

6.6.2 A paraméterátadás folyamata



6.2 ábra: A paraméterátadás folyamata

A paraméterek módosulását a felhasználó indítja el. Ekkor a kezelőfelületen az adott paraméterhez tartozó elem megváltozik és egyben a paraméterhez tartozó csatlakozó elem is frissül. Ez a csatlakozó elem már az értékfa része, tehát a változás azonnal jelentkezik. Következő lépésben a jelfeldolgozó osztály (PluginProcessor) jelfeldolgozó függvénye (`processBlock()`) lekérdezi az aktuális adatokat az értékfából az egyedi azonosítók segítségével, majd ezen adatokat átadja az egyes szólamokon belül megtalálható elemek paramétereinek frissítéséért felelős függvényeinek.

Az egyes szólamokat a SynthVoice osztály írja le és itt található az elemek paraméter frissítési metódusai:

- `void updateADSR();` //hang burkológörbéjének frissítése
- `void updateModEnv();` //szűrő burkológörbéjének frissítése
- `void updateFilter();` //szűrő frissítése

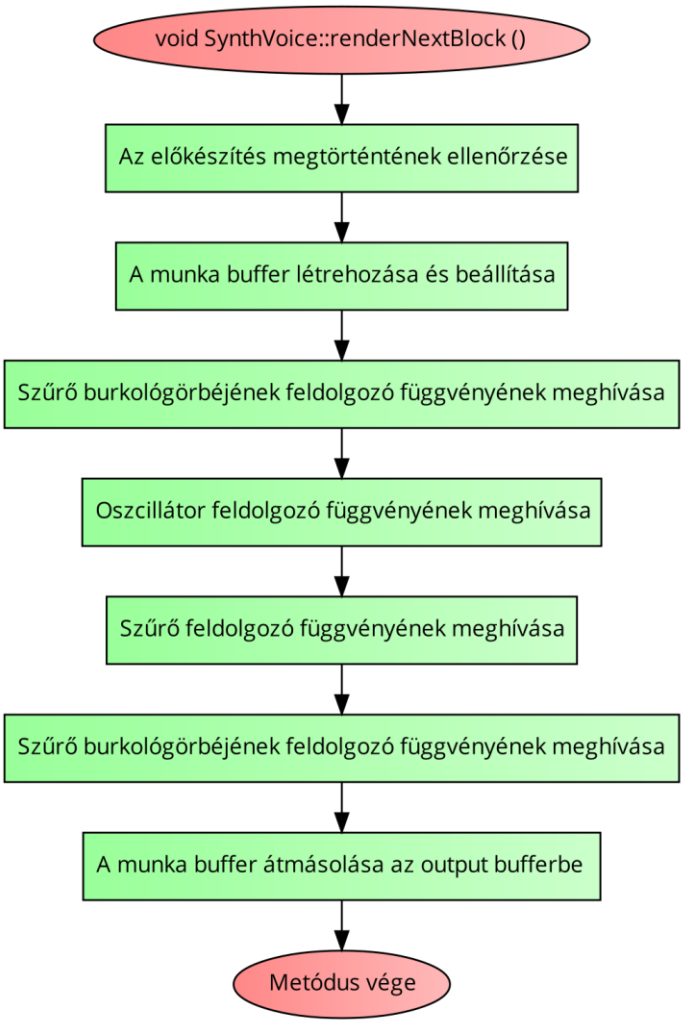
Ezen függvények mindegyike meghívja az adott elem paramétereinek beállításáért felelős függvényt (pl. a burkológörbe esetén az ADSR osztályhoz tartozó `setParameters()` függvényt). Kivétel ez alól az oszcillátor és a szűrő paramétereinek frissítése, ezeknél közvetlenül hívódnak meg a paraméter beállításért felelős metódusok, tehát nincs külön frissítési függvényük a SynthVoice osztályban.

Ezzel a folyamat véget ér, és a kezelőfelületen bekövetkezett változás már eljutott a megfelelő jelfeldolgozó elemhez.

6.7 A jelgenerálási folyamat

A jelgenerálás először a PluginProcessor osztályban található `prepareToPlay()` metódus meghívásával kezdődik, ez a plugin megnyitásakor azonnal lefut. Ez a metódus paraméterként megkapja a beállított mintavételezési frekvenciát és buffer méretet, majd meghívja a szintetizátor szólamainak saját `prepareToPlay()` metódusait. (Ez a folyamat hasonló a paraméterátadásnál látottakhoz.) Itt az elemek saját inicializáló függvényei hívódnak meg a mintavételezési frekvencia és buffer méretének paramétereivel. Miután ez megtörtént, az isPrepared logikai változó igazra vált, tehát megtörtént az előkészület a hang generálására.

Az előkészület után minden programiterációban meghívódik a `processBlock()` metódus a PluginProcessor osztályban. Ez a metódus paraméterként megkapja az output buffert és a beérkező MIDI jeleket, majd a függvényen belül meghívódik a szintetizátor objektum `renderNextBlock()` metódusa. Ez a függvény meghívja az egyes szólamok `renderNextBlock()` metódusát.



6.3 ábra: A renderNextBlock() metódus folyamatábrája

Az egyes szólamok `renderNextBlock()` metódusaiban ellenőrzésre kerül az isPrepared változó, majd miután ez megtörtént, létrejön egy új buffer. Ez azért szükséges, hogy a jelgenerálás ne közvetlenül az output bufferbe jöjjön létre, hanem egy üres munka bufferbe, és majd csak a jelgenerálási lánc végén lesz átmásolva a munka buffer tartalma az output bufferbe.

Miután létrejött a buffer, ezt a buffert referencia szerint átadva meghívodnak az egyes elemek feldolgozó metódusai:

- `oscillator.processBlock()`
- `filter.process()`
- `adsr.applyToBuffer()`

Ezen metódusok hívásának sorrendje megegyezik az 5.1-es és a 6.4-es ábrán található jelgenerálási lánc sorrendjével. Ezután a munka buffer tartalma átmásolásra kerül az output bufferbe.

A hang megszólaltatásához szükséges még a billentyű lenyomása is. A lenyomáskor meghívódik a startNote() függvény, amely a keletkezett MIDI jelet paraméterként

megkapja, majd ennek segítségével meghívja az oszcillátor frekvenciájának beállításáért felelős függvényt (setWaveFrequency()), és elindítja a két burkológörbét (`noteOn()`). A billentyű elengedésekor meghívódik a stopNote() függvény, amelyben a két burkológörbe leállításáért felelős függvény (`noteOff()`) indul el.

Tehát a háttérben minden programiterációban generálódik jel, de amíg egy billentyű nem kerül lenyomásra, addig nem érkezik MIDI jel az oszcillátorhoz, a beállított frekvencia 0 lesz, tehát „üres" hang generálódik. Ezenkívül, mivel a hang burkológörbéje sem indul el, ezért a hanghoz tartozó amplitúdó érték is 0 lesz.

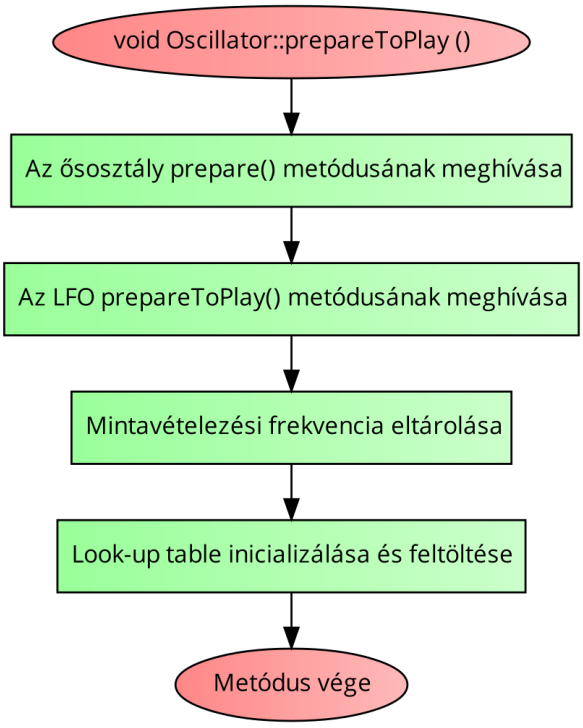
A továbbiakban ismertetésre kerülnek az egyes jelgenerálási és jelfeldolgozási elemek és az elemek működéséhez szükséges függvények és metódusok.

6.8 A jelgenerálás és jelfeldolgozás elemei

6.8.1 Oszcillátor

Előkészítés és inicializálás

A jelfeldolgozó elemek esetén először elő kell készíteni és inicializálni kell az elemet, majd miután ez megtörtént, minden mintavételezési időpillanatban futtatni kell feldolgozásért felelős függvényt. Ebben nem kivétel az oszcillátor sem. Előkészítés a `prepareToPlay()` függvénnyel történik, ez a már korábban ismertettet módon a plugin betöltésekor lefut. Mivel az oszcillátor megvalósításánál felhasználtam a JUCE beépített Oscillator osztályát és a saját Oscillator osztályt ebből az osztályból származtattam le, ezért előkészítéskor a beépített `prepare()` függvény fut le, ennek paraméterként van átadva a generáláshoz szükséges mintvételezési frekvencia és buffer méret. Ezután meghívódik az LFO saját előkészítő metódusa, majd létrejön a szinuszjel generáláshoz szükséges look-up table.



6.4 ábra: Az oszcillátor inicializálásának folyamatábrája

Paraméterek beállítása

Az oszcillátorban öt paramétert lehet beállítani:

- Az oszcillátor jelalakját
- Az oszcillátor frekvenciáját
- Az LFO frekvenciáját
- Az LFO amplitúdóját
- Az LFO modulálásának mértékét

Az oszcillátor jelalakját a `setWaveType()` metódussal lehet beállítani, itt a bemenő integer típusú paraméter alapján egy switch-case elágazásban lefut az ősosztály inicializáló metódusa. Ez a metódus paraméterként egy lambda függvényt fogad, amely átadva jelgenerátorként működik az ősosztályban.

Mivel az LFO az Oscillator osztály egy tagváltozójaként él, ezért a konfigurálásához szükséges paramétereket is az oszcillátoron keresztül tudjuk átadni. Ez azért is előnyös, mert így a modulálás mértékét közvetlenül le lehet kérdezni az LFO változón keresztül, nem kell a szólamon keresztül átadni ezt az értéket. Tehát az LFO frekvenciáját és amplitúdóját a `setLFO()` metódusban lehet beállítani, ahol az LFO még megkapja az utoljára leütött billentyű MIDI kódját is.

A modulálás mértéke a `setModulation()` metódusban állítható be, ez beállítja a modulation tagváltozó értékét, amely a frekvencia kiszámításánál lesz szükséges.

A frekvenciát a `setWaveFrequency()` metódussal lehet konfigurálni, itt a paraméterként megkapott MIDI jel számából könnyen ki lehet számítani a szükséges frekvenciát. Ebben a metódusban a már kiszámolt moduláció hozzáadódik a beállított frekvenciához, így megvalósul a LFO effektje. Végezetül ebben a függvényben számítódik ki a DPW algoritmusokhoz szükséges c állandó értéke is, mert ennek kiszámításához szükséges az aktuálisan beállított frekvencia.

Jelgenerálás

A megkapott paraméterekkel már megkezdődhet a jel generálása. Ez a `processBlock()` metódusban valósul meg, ahol először kiszámítódik az LFO moduláció mértéke, majd következő lépésben beállításra kerül az aktuális frekvencia, majd végezetül meghívásra kerül az ősosztály jelfeldolgozó metódusa, amely paraméterként megkapja a munka buffert.

Az ősosztály beépítetten tartalmaz egy fázisszámlálót, amely $-\pi$-től $+\pi$-ig számol a beállított frekvenciától függő sebességgel. A jelgeneráló függvények paraméterként ennek a fázisszámlálónak az értékét kapják meg, és ennek alapján számítják ki a generált jel aktuális értékét.

Mivel az ősosztály paraméterként kapja meg a különböző jelalakok generálásához szükséges metódusokat, ezért a továbbiakban ezen metódusokat ismertetem. **Sajnos a kikutatott DPW algoritmusok nem működtek kifogástalanul, a generált hangok felismerhetőek, de zajosak. Ezért az oszcillátorban a triviális módon működő algoritmusok is megírásra kerültek és alapbeállításként ezek futnak**. A dolgozat írása során teszteltem a triviális és a DPW módszerrel generált jeleket, és minimális eltéréssel

ugyanazt az eredményt kaptam. Ezek közül kivétel a szinuszjel előállítása, hiszen ez nem DPW módszerrel generálódik, és felharmonikusoktól, ezáltal átlapolódásoktól is mentes.

Szinuszjel generálása

A szinuszjel előállítása egy look-up táblázatból történik, ennek létrehozása és feltöltése az oszcillátor inicializálásánál történik. Maga a táblázat a JUCE keretrendszer beépített típusa, amely hasonlóan az oszcillátor jelgeneráláshoz, paraméterként egy generátor függvényt fogad, és ennek alapján tölti fel a táblázatot. Szükséges még megadni a tábla felbontását, tehát hogy a tábla hány pontból álljon. A szintetizátorban használt look-up table generátor függvénye a standard könyvtárban található szinuszfüggvény, felbontása pedig 256 pont. A jel előállításánál paraméterként átadjuk az aktuális fázist és ennek alapján visszatérésre kerül a fázishoz tartozó szinusz érték.

Fűrészjel generálása

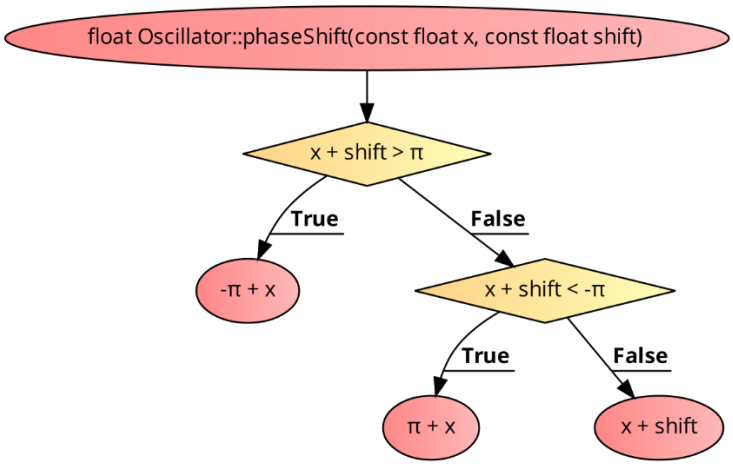
Triviális módszerrel fűrészjelet egyszerű előállítani: a fázisszámláló értéke elosztásra kerül π-vel, és mivel a fázisszámláló $-\pi$-től $+\pi$-ig számol, ezért az eredmény egy -1-től +1-ig számoló számláló. Érdekes módon ez a módszer nem eredményez jelentős mértékben hallható átlapolódást, csak a magas frekvenciák előállításakor hallható kis mértékben zaj.

A DPW módszer a 4.1.3-as fejezetben ismertet módon zajlik; az egyetlen különbség, hogy a jelt generáló metódus paraméterként megkapja a fázisszámláló értékét. Majd létrehozásra kerül egy triviális módon működő számláló, majd ennek a számlálónak az eredménye négyzetre emelésre kerül. Majd az előző iteráció és a mostani iteráció értékének a különbsége pedig szorzásra kerül a c állandóval. Az előző iteráció értékének eltárolásához egy statikus változó szükséges, különben a függvény újrahívása során elveszne az eltárolt állapotváltozó.

Négyszögjel generálása

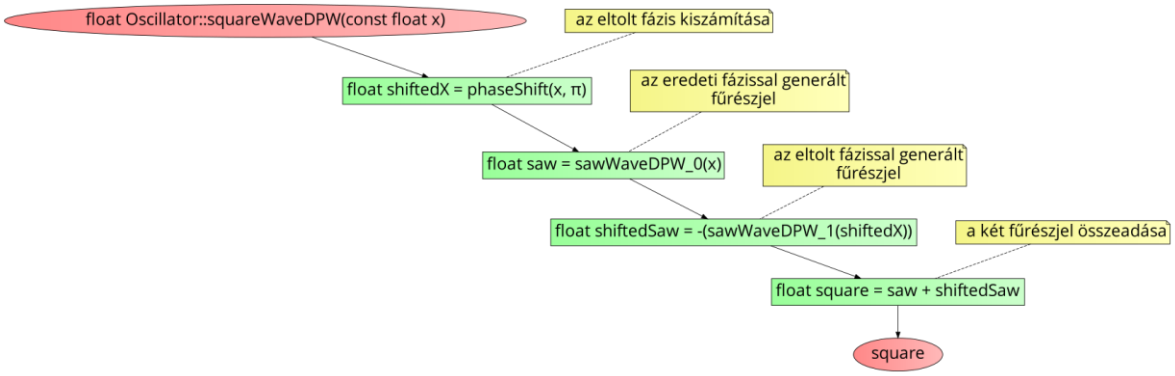
Triviális módszerrel a négyszögjel előállítása egy feltételes elágazással oldható meg: ha a fázisszámláló értéke kisebb, mint 0, akkor értéke legyen -1, különben legyen 1.

A DPW módszerhez szükséges a fázis eltolása, mivel két fűrészjel kerül összeadásra egymástól π fázis távolsággal. Ehhez szükséges volt egy fázistoló függvény megírása: ez a függvény paraméterként a fázisszámláló értékét kapja meg, eredménye pedig az eltolt fázis. Működésének folyamatábrája a 6.6-os képen látható.



6.5 ábra: A fázistoló függvény folyamatábrája

Az algoritmus implementálása az eltolt fázis eltárolásával kezdődik, majd két fűrészjel kerül létrehozásra: az első az eredeti fázissal, a második pedig az eltolt fázissal. Fontos, hogy az eltolt fázissal létrehozott fűrészjel invertálva van, tehát értéke -1-el szorzásra kerül. Miután ez megtörtént, a két jel összeadásra kerül, és az összeadás eredménye lesz a függvény kimenete.

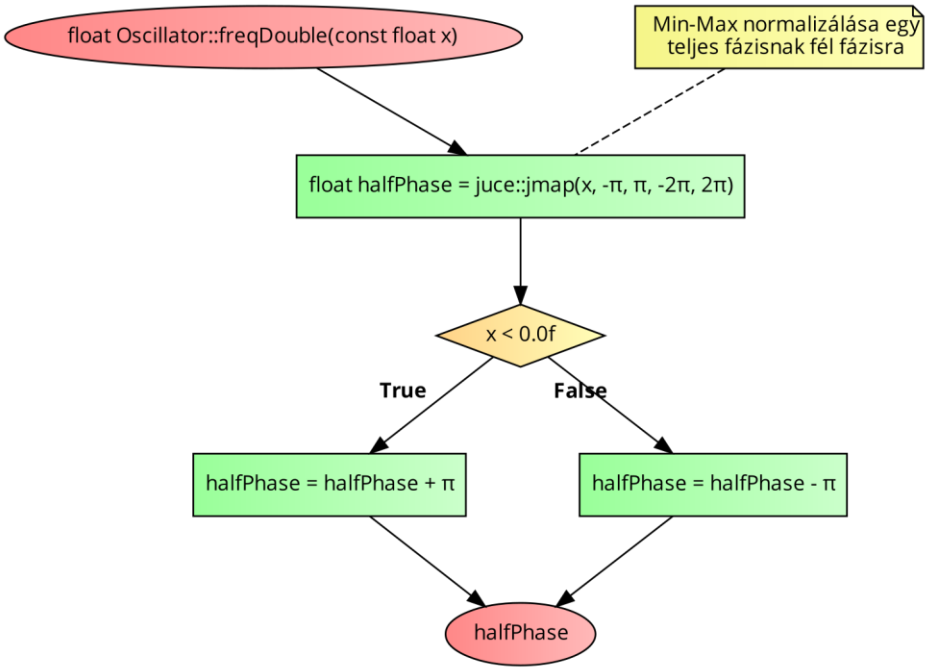


6.6 ábra: A DPW módszerrel generált négyszögjel folyamatábrája

Háromszögjel generálása

A triviális módszerrel megalkotott háromszögjelhez először egy triviális módon generált fűrészjelre van szükség. Ez a jel invertálásra, majd eltolásra kerül +0,5 értékkel. Végezetül az így kapott jel szorzásra kerül 2-vel, és a kapott jel lesz a kívánt háromszögjel.

A DPW módszerhez szükséges egy újabb segédfüggvény használata, amely a fázisszámláló értékét megduplázza. Erre azért van szükség, mert a jel előállításához szükséges fűrészjelet a kívánt frekvencia kétszeresén kell generálni, de a JUCE keretrendszer fázisszámlálójához közvetlenül nem lehet hozzáférni.



6.7 ábra: A fázisszámláló duplázó függvényének a folyamatábrája

A függvény egy teljes periódust ($-\pi$-től $+\pi$-ig) normalizál kettő periódusra (-2π-től $+2\pi$-ig). Ahhoz, hogy a jelgeneráló függvény ne kapjon egy periódusban megtalálható nagyobb értéket (tehát ne lépjen ki a $-\pi$-től $+\pi$-ig terjedő tartományból), szükséges a kapott érték korrigálása egy feltételes elágazással. A függvény működését a 6.8-as ábra szemlélteti.

A segédfüggvény segítségével már megvalósítható az algoritmus. Első lépésben a duplázott fázisértékkel egy számláló kerül létrehozásra (amely egy triviális fűrészjel generátornak tekinthető). Következő lépésben a számláló négyzetre emelésre kerül, majd invertálódik, tehát a jel értéke kivonásra kerül 1-ből. Ezután létrejön egy négyszögjel generátor, amely a kívánt frekvencián fut. A következő lépésben a négyzetre emelt majd invertált fűrészjel modulálásra, tehát szorzásra kerül a négyszögjellel. Az eredményül kapott jel kivonásra kerül az előző iterációban eltárolt értékkel, majd a mostani iteráció eredménye eltárolódik. Végezetül az eredményül kapott jel a DPW módszerben használatos c állandó felével szorzásra kerül.

6.8.2 LFO

Az LFO működése nagyban követi a már taglalt oszcillátorét, hiszen az LFO egy speciális, alacsony frekvencián működő oszcillátornak tekinthető, amelynek működési tartománya 0 Hz-től 20 Hz-ig terjed. Emiatt nem szükséges a DPW módszer használata, elég a triviális módszer. Az LFO osztálya, hasonlóan az Oscillator osztályhoz, a JUCE keretrendszer Oscillator osztályából származik le, tehát az előkészítésnél, inicializálásnál és jelfeldolgozásnál a keretrendszer megfelelő függvénye kerül meghívásra.

Első lépésben elő kell készíteni az oszcillátort, tehát be kell állítani a megfelelő mintavételezési frekvenciát, majd inicializálni szükséges, tehát az ősosztálynak át kell adni a generátorfüggvényt. A generátorfüggvényt paraméterként kell átadni az ősosztály inicializáló függvényének, amely jelen esetben a standard könyvtár szinuszfüggvénye.

Az LFO esetén nem szükséges look-up táblázatot használni a jelgeneráláshoz, hiszen az oszcillátor alacsony frekvenciatartományban működik, átlapolódás nem hallható.

Az LFO frekvenciája és amplitúdója a `setWave()` metódusban konfigurálható, és a beállított értékekkel a `calculateModulation()` függvényben lehet az aktuális értéket kiolvasni. A kiolvasott érték szorzásra kerül a megadott amplítúdóval, és ezt az értéket kapja meg a normál oszcillátor mint modulációs érték.

6.8.3 Szűrő

Előkészítés és inicializálás

A szűrő előkészítése hasonlóan történik az oszcillátorhoz; paraméterként itt is átadásra kerül a használt mintavételezési frekvencia és beállításra kerül, mint tagváltozó. Az oszcillátortól eltérően a szűrő osztálya nem származik le keretrendszeri osztályból, ezért ez előkészítés során nem kerül meghívásra más metódus. A szűrő aluláteresztő módban kerül inicializálásra.

Paraméterek beállítása

A szűrő négy paraméterrel vezérelhető:

- A szűrő módja
- Vágási frekvencia
- Rezonancia mértéke
- A vágási frekvencia modulációjának mértéke

A szűrő módját a `setFilterType()` metódus segítségével lehet konfigurálni, itt egy switch-case elágazással a paraméterként megkapott integer alapján lehet a kívánt módot beállítani. A vágási frekvenciát és a rezonancia mértékét pedig a `setFilter()` metódusban lehet beállítani. Ebben a metódusban az előbb említett paramétereken kívül még a moduláció mértékét is megkapja a metódus, ennek értékével a beállított vágási frekvencia szorzásra kerül, így valósítva meg a szűrő modulációját. A modulált vágási frekvencia kiszámítása során ellenőrzésre kerül a moduláció értéke, ha ez 0, akkor a moduláció értéke nem kerül figyelembe és moduláció nélkül kerül a vágási frekvencia beállításra. Ezenkívül, ha a modulált vágási frekvencia kisebb, mint 20 Hz vagy nagyobb mint 20000 Hz, akkor a modulált frekvencia korlátozásra kerül, 20 Hz-nél alacsonyabb frekvencia esetén 20 Hz-re, 20000 Hz-nél magasabb frekvencia esetén 20000 Hz-re.

A szűrő paramétereinek kiszámítása

Még a `setFilter()` metóduson belül meghívásra kerül a `calculateParams()` metódus, amely az állapotváltozós szűrő működéséhez szükséges paramétereket számolja ki. A metódusban a 4.2.2-es fejezetben ismertet modellhez szükséges paraméterek kerülnek kiszámításra a mintavételezési frekvencia és az aktuálisan beállított vágási frekvencia alapján. Az alábbi paraméterek kerülnek kiszámításra:

- T - mintavételezési periódus
- ω_a - analóg tartományban megadott vágási körfrekvencia
- ω_d - digitális tartományban megadott vágási körfrekvencia
- g - erősítési tényező
- R - rezonancia mértéke

Jelfeldolgozás

A jelfeldolgozást a szűrő `process()` metódusa valósítja meg, amely bemeneti paraméterként a már előzőlegesen az oszcillátor által feltöltött munka buffer kapja meg referencia szerint. A metódusban csatornánként és mintánként végighaladva valósul meg a szűrés. Az analóg modellhez hasonlóan itt is mindhárom szűrés kiszámításra kerül, az egyes szűrési típusok a felüláteresztett jelből kerülnek kiszámításra. Az aktuálisan kiválasztott szűrési mód szerint egy switch-case elágazásban kerül a szűrt jel átmásolásra a munka bufferbe.

6.8.4 Hang és modulációs burkológörbe

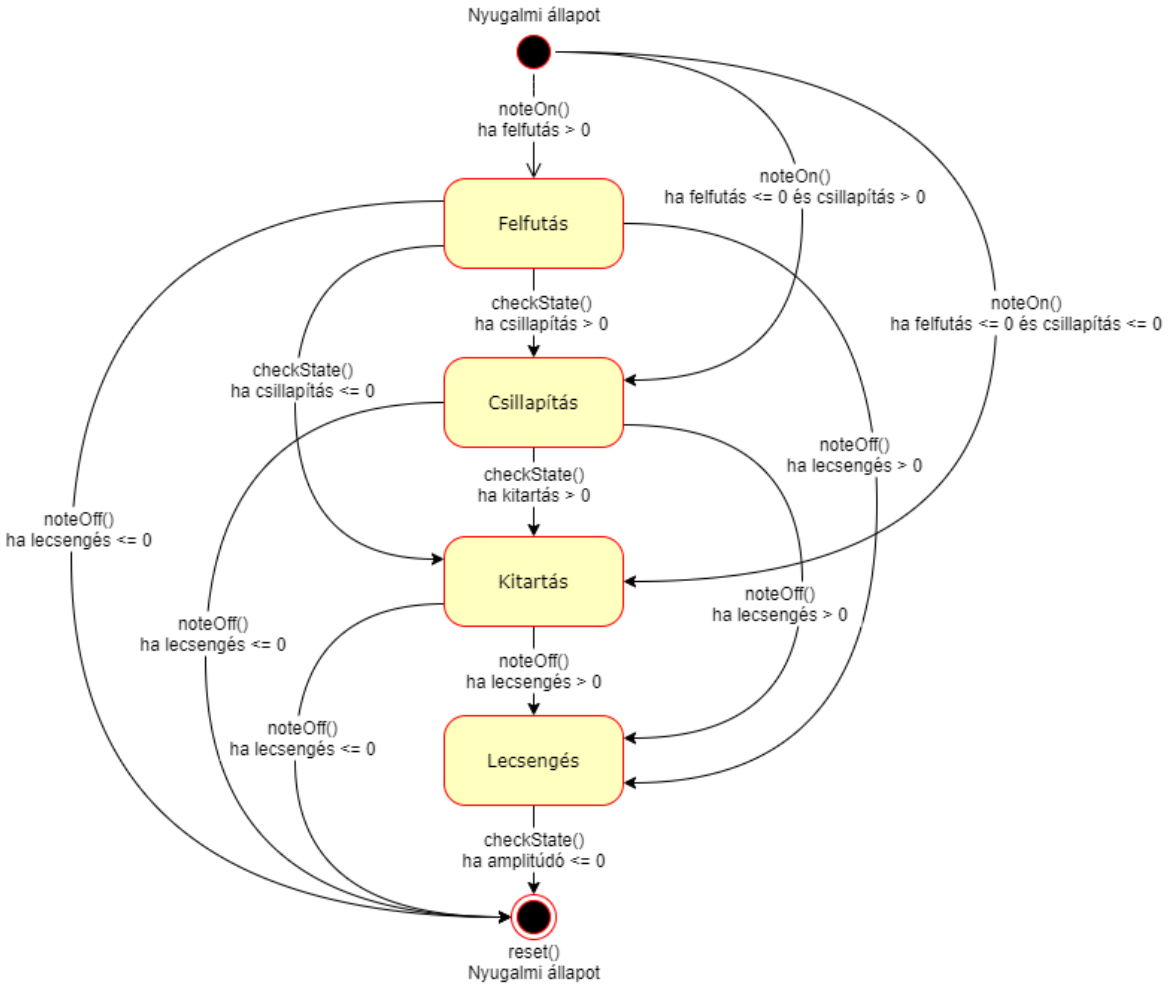
A burkológörbe működése, hasonlóan a többi jelfeldolgozó egységhez, az egység előkészítésével kezdődik, ahol a mintavételezési frekvencia beállítása történik. A kezdő paraméterek (attack, decay, sustain, release), az amplitúdó, a maximális amplitúdó és a nyugalmi állapot beállítása a header fájlban történik, itt a változók a deklarálás során megkapják az alapértelmezett értékeiket.

A burkológörbe működése a 4.3-as fejezetben elmondottak alapján történik: egy állapotgép modellezi a burkológörbe működését, amely állapotgépnek öt állapota van. Nyugalmi állapotban a burkológörbe nem tesz semmit, és amíg nem kerül billentyű lenyomásra, addig nem lép be a következő állapotba. Ha egy billentyű lenyomásra kerül, akkor a beállított paraméterektől függően három állapotba léphet:

- Ha a felfutás nagyobb mint 0, akkor a felfutás állapotba lép.
- Ha a felfutás értéke 0, és a lecsengés értéke nagyobb mint 0, akkor a lecsengés állapotba lép. Ebben az esetben az amplitúdó értéke azonnal 1 lesz.
- Ha a mind a felfutás, mind a lecsengés értéke 0, akkor egyből a kitartás állapotába kerül a burkológörbe.

Miután a burkológörbe a megfelelő induló állapotba került, a `process()` függvény minden időpillanatban kiszámítja az aktuális amplitúdó értéket az aktuális állapottól függően. Az amplitúdó kiszámítása a 4.3-as fejezetben ismertet módon történik. A metódus végén meghívásra kerül a `checkState()` metódus, amely az aktuális állapottól és amplitúdó értéktől függően a jelenlegi állapotot megtartja, vagy egy következő állapotba lépteti.

A billentyű felengedésekor meghívásra kerül a `noteOff()` metódus, amely a beállított paraméterektől függően a burkológörbét a kitartás állapotba váltja, vagy a meghívja a `reset()` metódust, amely a burkológörbét a nyugalmi állapotba teszi. A teljes állapotdiagramot a 6.9-es ábra szemlélteti.



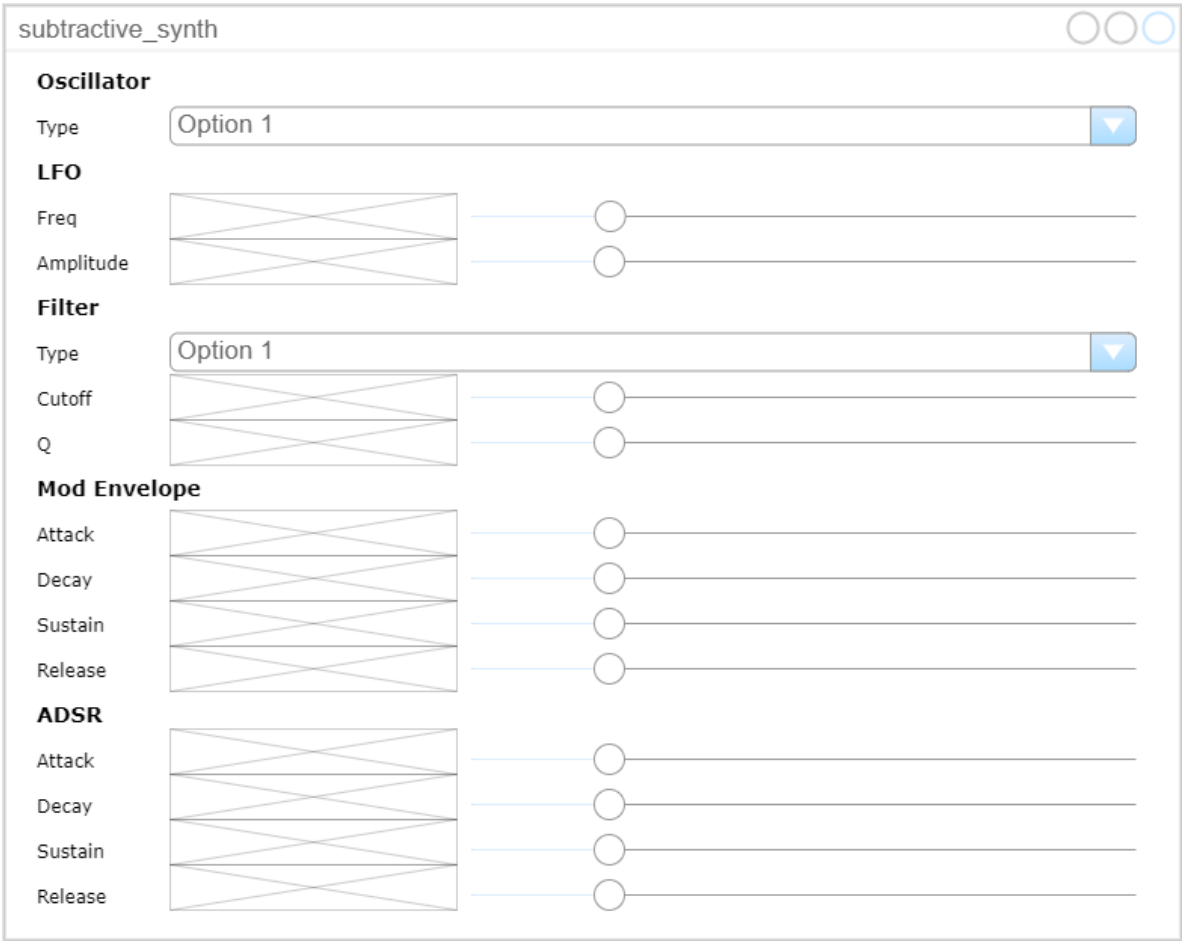
6.8 ábra: A burkológörbe állapotdiagramja

7 A MEGVALÓSÍTÁS LÉPÉSEI

A szintetizátor megírásának minden lépését a kikutatott irodalom és az aktuális lépéshez hozzátartozó JUCE segédlet, valamint különböző audio szoftverekkel foglalkozó fórumok átolvasása előzte meg.

A szintetizátor fejlesztése a JUCE projekt generálásával kezdődött. Ebben segítségre volt a JUCE keretrendszerhez mellékelt Projucer applikáció, amellyel a megfelelő paramétereket kiválasztva legenerálta a kiválasztott fejlesztőkörnyezet projektjét, jelen esetben a Visual Studio solution-jét. A fejlesztés során szükséges forrásállományokat is itt adtam hozzá a projekthez.

A szintetizátor megírását a kezelőfelülettel kezdtem. Figyelembe véve a szintetizátor elemeit és azok paramétereit, létrehozásra került a kezelőfelület wireframe-je. Ezt alapul véve megírásra került a kezelőfelület kódja a PluginEditor osztályában. A tervezésnél törekedtem az átláthatóságra és az egyes kezelőfelületi elemek újrafelhasználhatóságára. Emellett arra is igyekeztem ügyelni, hogy az elemek elrendezése kövesse a jelgenerálás útját fentről lefelé.



7.1 ábra: A kezelőfelület wireframe-je

A következő lépés célja egy MIDI jellel vezérelhető egyszerű szinuszhang generálása volt, amellyel egyidőben létrehozásra kerül a szintetizátor logikai váza. Ennek érdekében hozzáadásra kerültek az egyes jelfeldolgozó egységek header és cpp állományai a Projucer applikáció segítségével. Ennek a fázisnak a végére a szintetizátor képes volt egy MIDI jellel vezérelhető triviálisan generált szinuszhang lejátszására.

A következő mérföldkő az ADSR burkológörbe megírása volt. Itt nagy segítségemre volt a JUCE saját burkológörbéjének az implementációja, ezt a kikutatott irodalom alapján módosítottam: a görbe lefutását a 4.3-as fejezetben ismertet módon implementáltam. Ez az elem volt ez első, amely összekötésre került a kezelőfelülettel. A kód megírása során ügyeltem a tesztelésre, több próbálkozásba is került, mire a megfelelő módon működött a burkológörbe.

A burkológörbe megírásával párhuzamosan nekiláttam az LFO megírásának. Ennek implementálásakor segítségemre volt a már megírt oszcillátor osztály, hiszen az LFO egy egyszerű oszcillátornak tekinthető. Itt több próbálkozásba került, mire az LFO osztálya jelgenerálási lánc megfelelő helyére került: először a SynthVoice osztályba volt felvéve mint tagváltozó, de így az LFO moduláció paraméterének átadása az oszcillátornak nehézségekbe ütközött. Ezt végül az LFO osztály átrakása oldotta meg: az LFO áthelyezésre került a SynthVoice osztályból az oszcillátor osztályba, és itt mint tagváltozó

lett felvéve. Az osztály megírása után az LFO összekötésre került a kezelőfelülettel és így vezérelhetővé vált.

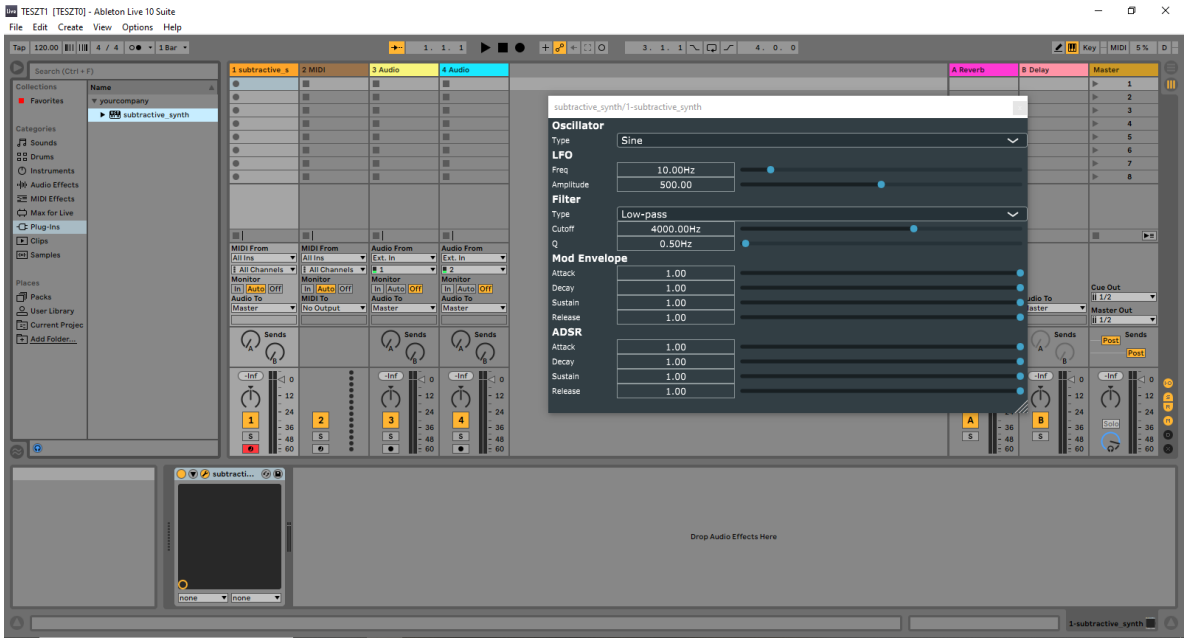
Az ADSR burkológörbe és az LFO megírása után a szűrő megvalósítása következett. Ezt a meglévő irodalomkutatás kibővítése előzte meg, mert a szűrő első verziójának megírása során kiderült, hogy a kikutatott digitális FIR szűrő együtthatóinak kiszámítása túlzottan erőforrásigényes. A kikutatott állapotváltozós szűrő ezt a problémát sikeresen kiküszöbölte.

A szűrő megírása során több szintetizátor implementációval találkoztam. Ezeket tanulmányozva arra jutottam, hogy az eredeti tervvel ellentétben érdemesebb a szűrőt egy burkológörbével modulálni az LFO helyett, mert ez a megoldás művészi szempontból jobb eredményt ad. Az implementálás bonyolultsága ezzel nem változott, hiszen mind az LFO, mind a burkológörbe osztálya már megírásra került ezen a ponton.

Így a következő lépés a szűrő modulációs burkológörbéjének megírása volt, amely a már meglévő ADSR burkológörbe osztályának a módosítását és annak a szűrő osztályába kapcsolását jelentette. Ez logikailag megegyezett az LFO implementálása során követett metodikával, tehát a modulációt generáló elem mint tagváltozó létezik a modulációt fogadó elem osztályában. Ennek a fázisnak a végére a szintetizátor jelfeldolgozási lánca már teljeskörűen működött. A lépés végén, a többi elemhez hasonlóan, a szűrő is összekötésre került a kezelőfelülettel.

Az utolsó mérföldkő az oszcillátor jelgeneráló algoritmusainak megírása volt. Ennek a mérföldkőnek az első lépése a többi triviálisan generált jelek megírása volt, tehát a fűrészfog, négyszög és háromszögjel implementálása, majd az ezeket megvalósító függvények hozzákapcsolása a kezelőfelülethez és a paramétereket számontartó értékfához. A következő lépésben a triviálisan generált jelek DPW verziójának megvalósítása került sorra. Itt már csak a jelgenerálási algoritmusok helyes implementációjára kellett koncentrálni, hiszen a bekötésük a kezelőfelülethez már megtörtént a triviális verziók megírásánál. Sajnos minden próbálkozásom és tesztelés ellenére a DPW algoritmusok nem működtek megfelelően, ezért visszaállítottam a triviális verziókat és a DPW jelgenerálási függvényeket kikommenteztem.

A megvalósítás minden fázisánál legeneráltam a projektet, és a JUCE-hoz mellékelt VST teszter applikációba betöltöttem tesztelés céljából. A nagyobb mérföldköveknél (az egyes jelfeldolgozó egységek megírása után) a legenerált VST plugint betöltöttem az általam használt digitális audio munkaállomásba, az Ableton programba.



7.2 ábra: A plugin betöltve az Ableton digitális audio munkaállomásban

8 TESZTELÉS

A szintetizátor megvalósítását annak tesztelése követte. A szintetizátor tesztelése során unit teszteket használtam, ezt a tesztelési módszert a JUCE keretrendszer beépítetten támogatja. Tesztelésre kerültek a jelfeldolgozó lánc elemei, tehát az oszcillátor, az LFO, a szűrő és a burkológörbe. Az egyes elemekhez tartozó tesztek külön osztályokban kerültek megírásra, amely osztályok a JUCE UnitTest osztályának leszármazottjai. A tesztelés csak Debug mód esetén fut le, ekkor a megírt tesztek összegyűjtésre és futtatásra kerülnek a PluginProcessor osztályban. A következőkben bemutatásra kerülnek az egyes elemek tesztesetei és azok eredményei.

Oszcillátor:

1. Az oszcillátor előkészítése
2. Oszcillátor frekvenciájának beállítása
3. Negatív frekvencia beállítása
4. A DPW algoritmushoz szükséges c konstans kiszámítása
5. Negatív moduláció beállítása
6. A fázistoló függvény működése
7. A fázisszámláló duplázó működése
8. A triviális módszerrel generált fűrészjel előállítása
9. A DPW módszerrel generált fűrészjel előállítása
10. A triviális módszerrel generált négyszögjel előállítása
11. A DPW módszerrel generált négyszögjel előállítása
12. A triviális módszerrel generált háromszögjel előállítása
13. A DPW módszerrel generált háromszögjel előállítása

Az oszcillátor tesztelésekor először létrehozásra került egy üres oszcillátor változó. Következő lépésben külön változókban definiálásra került az oszcillátorban használt audio buffer mérete, a mintavételezési frekvencia és a csatornák száma. Ezután

létrehozásra került egy integer típusú változóban egy 72-es MIDI kód (amely a c5 hangnak felel meg), majd egy float típusú változóban a kódhoz tartozó frekvencia, amely jelen esetben 523,25 Hz.

Az első teszteset az oszcillátor előkészítése volt. Ebben a tesztben ellenőrzésre került, hogy az oszcillátor előkészítése során megfelelően átadásra kerülnek az oszcillátor működéséhez szükséges paraméterek, legfőbbképpen a mintavételezési frekvencia. Miután a létrehozott oszcillátor példánynak átadásra került a mintavételezési frekvencia, az átadott mintavételezési frekvencia összehasonlításra került az eredeti mintavételezési frekvenciával. Ez a teszt minden hiba nélkül lefutott.

A második teszteset az oszcillátor frekvenciájának beállítását tesztelte. Ebben a tesztben az oszcillátor először alapállapotba lett állítva, tehát a meghívásra került az oszcillátor `reset()` metódusa. A következő lépésben átadásra került a már definiált MIDI kód az oszcillátor frekvencia beállításáért felelős metódusának, a `setWaveFrequency()`-nek. Ezután összehasonlításra került a beállított frekvencia az eredetileg definiált frekvenciával a `getFreq()` és `getFrequency()` getter függvények segítségével. Ez a teszt is hiba nélkül lefutott.

A harmadik teszteset a negatív frekvencia beállítását tesztelte. Mivel ezt MIDI kóddal nem lehetett elérni, ezért az oszcillátor modulációs értékének a c5-ös hang frekvenciája lett beállítva és ebből kivonásra került 1 Hz. Mivel az aktuális moduláció értéke kivonásra kerül a beállított frekvenciából, ezért a beállított frekvencia várhatóan -1 Hz lett. Ezután az előző tesztesethez hasonlóan átadásra került a c5-ös hang MIDI kódja. Az oszcillátor működése során elvárt volt, hogy a negatív frekvenciákat invertálja, hiszen az oszcillátor LFO modulációja során könnyen létrejöhet negatív frekvencia. Emiatt a beállított -1 Hz esetén a teszt +1 Hz-et várt eredményként. Ennek a tesztnek a felállítása időigényes volt, de a teszt minden hiba nélkül lefutott.

A negyedik teszt a DPW algoritmushoz szükséges c konstans kiszámítását tesztelte negatív frekvencia esetén. A teszt inicializálása az előző tesztekhez hasonlóan történt, tehát az oszcillátor először alapállapotba lett állítva, majd az előző teszthez hasonlóan lett beállítva a negatív frekvencia. Ezután a teszten belül egy külön változóban kiszámításra került a c konstans a beállított negatív frekvencia értékének ellentétjével, tehát +1 Hz-el. A teszt végén összehasonlításra került az oszcillátorban és a tesztben kiszámított c konstans értéke. A teszt hiba nélkül lefutott.

Az ötödik teszteset a negatív moduláció beállításának tesztelése volt. Az oszcillátor alapállapotba állítása után az oszcillátor modulációs értéke a c5-ös hang frekvenciájának ellentétjére lett állítva. Mivel a moduláció értéke lehet negatív, ezért a teszt végén a negatív frekvencia lett összehasonlítva. Ez a teszt is hiba nélkül lefutott.

A hatodik teszt a DPW algoritmusokhoz szükséges fázistoló függvényt tesztelte. Az oszcillátor alapállapotba állítása után a függvény az alábbi paraméterekkel lett tesztelve:

- -π kezdeti érték +π-vel eltolva
- 0 kezdeti érték +π-vel eltolva
- +π kezdeti érték +π-vel eltolva
- -π kezdeti érték -π-vel eltolva
- 0 kezdeti érték -π-vel eltolva
- +π kezdeti érték -π-vel eltolva

A tesztesetek mindegyike sikeresen lefutott.

A hetedik teszteset a fázisszámláló duplázó működését tesztelte. Miután az oszcillátor alapállapotba lett állítva, egy változóban kiszámításra kerültek a +π, 0 és -π fázisok duplázásának értékei a segédfüggvény használatával. Mindegyik fázis összehasonlításra került az elvárt értékekkel, és mindegyik teszt sikeresen lefutott.

A nyolcadik teszteset a triviális fűrészjel generálását tesztelte. Az oszcillátor alapállapotba lett állítva, majd a -π, 0, és +π fázisok esetén kapott értékek tesztelésre kerültek. -π esetén -1, 0 esetén 0, +π esetén pedig 1 értékkel kerültek összehasonlításra az egyes fázisokban kapott értékek. A tesztek mindegyike sikeresen lefutott.

A kilencedik tesztben a DPW algoritmussal generált fűrészjel került tesztelésre. Első lépésként egy változóban létrehozásra került egy inkrementum, amellyel egy mintavételezési időpillanatot lehet modellezni. Egy másik változóban pedig kiszámításra került az egy egységnyi idő alatt bekövetkező inkrementumok száma. Ezen segédváltozók felvétele után külön változókban definiálásra került egy triviális és egy DPW algoritmussal működő fűrészjel generátor. A két generátor értéke egy for ciklusban minden lépésben összehasonlításra került 0.1 tűréshatárral. A teszt minden hiba nélkül lefutott, ezért nem érthető a DPW algoritmus zaja.

A tizedik, tizenegyedik, tizenkettedik és tizenharmadik tesztek esetén tesztelésre kerültek a négyszög és a háromszögjelek triviális és DPW algoritmussal generált jelei. Ezek tesztelését külön nem ismertetem, a tesztelésük ugyanúgy történt, mint a fűrészjel esetén. A tesztek mindegyike sikeresen lefutott.

LFO:

1. Az LFO előkészítése
2. Az LFO frekvenciájának és amplitúdójának beállítása

Mivel az LFO egy egyszerűsített oszcillátornak tekinthető, ezért tesztelése is egyszerűbb. Az első teszteset az LFO előkészítését tesztelte. Ennek tesztelése az oszcillátor előkészítésének teszteléséhez volt hasonló: először létrehozásra került egy változóban egy LFO példány, majd külön változókban az előkészítéshez szükséges adatok: a buffer méret, a mintavételezési frekvencia és a csatornák száma. Ezen adatok egy struktúra részeként átadásra kerültek az LFO `prepareToPlay()` metódusának. Miután ez megtörtént, az átadott mintavételezési frekvencia összehasonlításra került az eredeti mintavételezési frekvenciával. A teszt sikeresen lefutott.

A második tesztben az LFO frekvenciájának és amplitúdójának beállításáért felelős `setWave()` metódus került tesztelésre. A tesztben egy 1000 Hz-es 5 amplitúdó értékű jel kerül beállításra a metódus segítségével. Miután ez megtörtént, a `getFreq()`, `getFrequency()` és `getAmplitude()` getter függvények segítségével lekérdezésre kerülnek az LFO beállított értékei, majd ezután összehasonlításra kerülnek az eredeti értékekkel. A tesztek sikeresen lefutottak.

Szűrő:

1. A szűrő előkészítése
2. A vágási frekvenciájának beállítása moduláció nélkül
3. A szűrő jelfeldolgozó függvényének működése
4. A szűrő alapállapotba váltása

5. A vágási frekvenciájának beállítása modulációval
6. A szűrő típusának beállítása

A szűrő tesztelésének előkészítése során először létrehozásra került a szűrő egy példánya, majd külön változóban definiálásra került egy 2 csatornás, 256 minta nagyságú buffer, amely inicializálva lett 1-es értékekkel. Ezután létrehozásra került egy előre definiált vágási frekvencia 1500 Hz-es értékkel, egy Q változó 1-es értékkel, egy kis értékű modulációs változó 0,01-es értékkel, egy közepes értékű modulációs változó 10-es értékkel és egy nagy értékű modulációs változó 100-as értékkel.

Az első teszteset a szűrő előkészítését tesztelte. Ennek során az oszcillátor előkészítésének teszteléséhez hasonlóan átadásra került egy előre definiált mintavételezési frekvencia, jelen esetben 44100 Hz. Ezután ez az érték kiolvasásra került a `getSampleRate()` getter függvény segítségével. Ezenkívül még kiolvasásra került az alapbeállított vágási frekvencia és szűrő típus a `getCutoff()` és `getFilterTypeString()` getter metódusok segítségével. A teszt végén a kiolvasott értékek összehasonlításra kerültek az elvárt értékekkel, amely a mintavételezési frekvencia esetén a beállított 44100 Hz, a vágási frekvencia esetén az alapbeállított 0 Hz, és a szűrő típusa esetén pedig az alapbeállított aluláteresztő mód. A teszt sikeresen lefutott.

A második teszteset a szűrő vágási frekvenciájának beállítását tesztelte moduláció nélkül. A tesztben az előre definiált vágási frekvencia és Q értékek átadásra kerültek a szűrő példányának. Miután ez megtörtént, az átadott paraméterek kiolvasásra kerültek a megfelelő getter metódusokkal. A kapott értékek a teszt végén összehasonlításra kerültek az elvárt eredményekkel. A teszt sikeresen lefutott.

A harmadik teszteset a szűrő működését tesztelte. Ebben a tesztben a már előző tesztekben megadott vágási frekvencia és Q paraméterekkel a szűrő `process()` metódusának átadásra kerül a létrehozott buffer. Miután a metódus feldolgozta a bufferben lévő tartalmat, a szűrő állapotváltozói tesztelésre kerülnek, amelynek során az az elvárt eredmény, hogy az állapotváltozók értékei ne legyenek 0-val egyenlőek. A teszt sikeresen lefutott.

A negyedik teszteset a szűrő `reset()` metódusát tesztelte, amely a szűrő alapállapotba állításáért felel. Ennek során meghívásra kerül a metódus, majd a szűrő állapotváltozói ellenőrzésre kerülnek. Az elvárt eredmény mindegyik állapotváltozó esetén 0. A teszt sikeresen lefutott.

Az ötödik teszteset a szűrő vágási frekvenciájának beállítását tesztelte beállított modulációval. Első lépésként a szűrő alapállapotba lett állítva a már tesztelt `reset()` metódus segítségével. Az első esetben a szűrő beállításra került a már definiált vágási frekvenciával, Q értékkel és közepes modulációs értékkel. Az elvárt eredmény a beállított vágási frekvencia és a modulációs érték szorzata volt. A kiszámított modulált vágási frekvencia a `getCutoff()` getter függvénnyel lett lekérdezve. Ez a teszt sikeresen lefutott. A következő tesztekben a moduláció a kis és nagy modulációs értékekkel került kiszámításra. Itt először nem futott le a teszt, mert a szűrő működésénél elvárt volt, hogy 20 Hz-nél kisebb frekvencia esetén 20 Hz-re, 20000 Hz-nél nagyobb frekvencia esetén pedig 20000 Hz-re állítása a vágási frekvenciát. Emiatt módosítani kellett a szűrő

`setFilter()` metódusát, ahol a standard könyvtár fmax és fmin függvényeivel lehetett megoldani a problémát. A módosítás után a teszt sikeresen lefutott.

Burkológörbe:

1. A burkológörbe előkészítése
2. A burkológörbe alapállapotba váltása
3. Billentyű lenyomása különböző paraméterek esetén
4. Billentyű felengedése
5. A burkológörbe jelfeldolgozó függvényének működése különböző paraméterek esetén

A burkológörbe tesztelésének előkészítése egy burkológörbe példány létrehozásával kezdődött. Miután ez megtörtént, az első teszteset a burkológörbe előkészítését tesztelte. Hasonlóan az oszcillátor és szűrő előkészítéséhez, itt is átadásra került egy 44100 Hz-es mintavételezési frekvencia. A teszt során ellenőrzésre került az átadott mintavételezési frekvencia, és a burkológörbe alapértelmezett nyugalmi állapota. Mindkét teszt sikeresen lefutott.

A második teszteset során ellenőrzésre került a burkológörbe `reset()` metódusa. Ennek során először meghívásra került a metódus, majd ellenőrzésre került a burkológörbe aktuális amplitúdó értéke a `getA()` getter függvény segítségével és az alapértelmezett nyugalmi állapot a `getStateString()` getter függvénnyel. Mindkét teszt sikeresen lefutott.

A harmadik teszt során ellenőrzésre került a burkológörbe elindulása, miután a szintetizátorba egy MIDI jel érkezett. Ennek során először a burkológörbe alapértelemezett állapotba lett állítva a `reset()` metódus segítségével. Ezután a burkológörbe paraméterei az alábbi módon lettek beállítva:

- Felfutás – 0,5
- Csillapítás – 0
- Kitartás – 0
- Lecsengés – 0

A paraméterek beállítása után meghívásra került a burkológörbe `noteOn()` metódusa. A beállított paramétereknek megfelelően az elvárt eredmény a burkológörbe Felfutás állapotba való lépése jelentette. Ez ellenőrzésre került, és a teszt sikeresen lefutott.

A negyedik teszt során tesztelésre került a burkológörbe Csillapítás állapota. Ennek érdekében a burkológörbe paraméterei az alábbi módon lettek beállítva:

- Felfutás – 0
- Csillapítás – 0,5
- Kitartás – 0
- Lecsengés – 0

A paraméterek beállítása után megint meghívásra került a `noteOn()` metódus. Miután ez lefutott, ellenőrzésre került a burkológörbe aktuális állapota és amplitúdója. Az elvárt eredmény az állapot esetén a Csillapítás állapot, amplitúdó esetén pedig 1-es érték volt. Mindkét teszt sikeresen lefutott.

Az ötödik teszt a burkológörbe Lecsengés állapotát ellenőrizte. Ennek során először a burkológörbe alapállapotba lett állítva a `reset()` metódus segítségével. Miután ez megtörtént, az alábbi paraméterek lettek beállítva:

- Felfutás – 0
- Csillapítás – 0,5
- Kitartás – 0
- Lecsengés – 0

A paraméterek beállítása után meghívásra került a burkológörbe `noteOff()` metódusa, amely a billentyű elengedése után hívódik meg. A metódus meghívása után ellenőrzésre került a burkológörbe aktuális állapota, amelynek elvárt eredménye a Lecsengés állapota. Ez ellenőrzésre került és a teszt sikeresen lefutott.

A hatodik teszteset a burkológörbe jelfeldolgozó `process()` metódusát tesztelte különböző állapotok esetén. Először a megszokott módon a burkológörbe alapértelmezett állapotba lett állítva a `reset()` metódus segítségével. Miután ez megtörtént, az alábbi paraméterek lettek beállítva a burkológörbének:

- Felfutás – 0,5
- Csillapítás – 0,5
- Kitartás – 0,5
- Lecsengés – 0,5
- Állapot – Nyugalmi állapot
- Amplitúdó – 1

A paraméterek beállítása után meghívásra került a `process()` metódus. Az elvárt eredmény az amplitúdó értékének 0-ra változása jelentette, hiszen nyugalmi állapotban az amplitúdó értéke csak 0 lehet. Ez ellenőrzésre került, és a teszt sikeresen lefutott.

A következő teszt esetén az alábbi paraméterek lettek beállítva a `reset()` metódus meghívása után:

- Felfutás – 0,5
- Csillapítás – 0,5
- Kitartás – 0,5
- Lecsengés – 0,5
- Állapot – Felfutás állapot
- Amplitúdó – 1

A paraméterek beállítása után megint meghívásra került a `process()` metódus, amelynek lefutása után az elvárt eredmény az amplitúdó értékének változatlanul 1-es értéken való maradása. Ez ellenőrzésre került, és a teszt sikeresen lefutott.

A következő esetben a Kitartás állapotában lett tesztelve a `process()` metódus. Az alábbi paraméterek lettek beállítva a `reset()` metódus meghívását követően:

- Felfutás – 0,5
- Csillapítás – 0,5
- Kitartás – 0,5
- Lecsengés – 0,5

- Állapot – Kitartás állapot
- Amplitúdó – 1

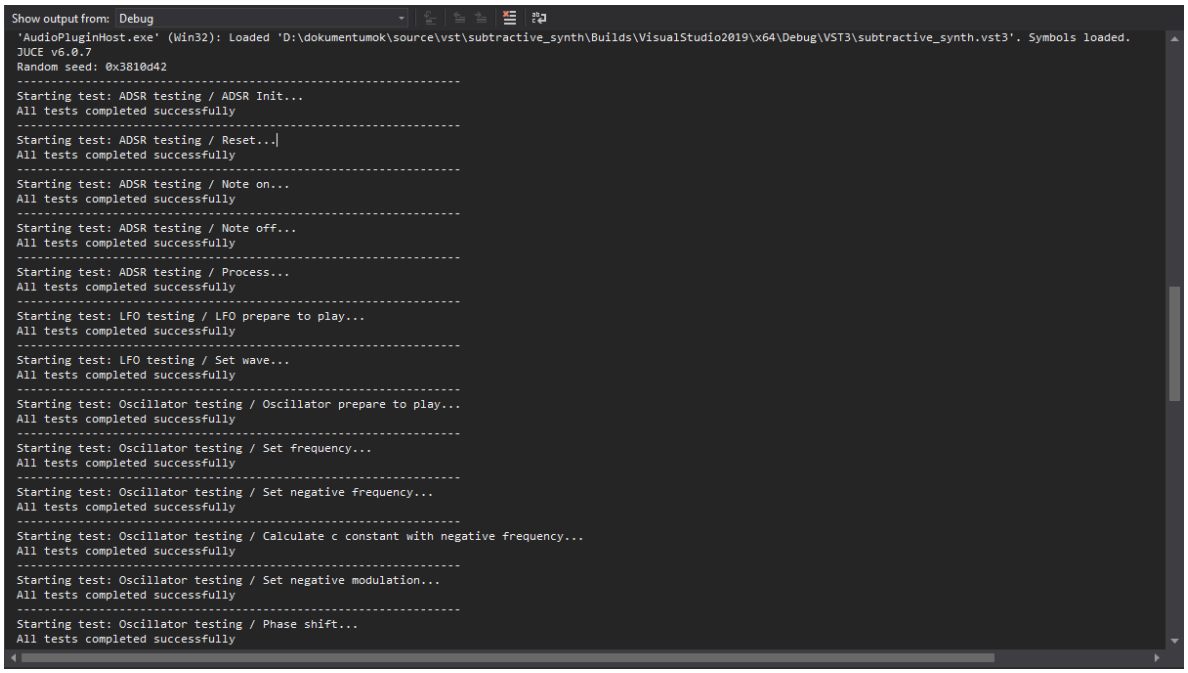
A paraméterek konfigurálása után meghívásra került a `process()` metódus. Ennek lefutása után az elvárt eredmény az amplitúdó értékének 0,5-re változása, amely a következő lépésben ellenőrzésre került. Az ellenőrzés sikeresen lefutott.

Az utolsó esetben a Lecsengés állapotában került tesztelésre a `process()` metódus. Első lépésben a burkológörbe alapértelmezett állapotba lett állítva a `reset()` metódus meghívásával. A metódus lefutása után az alábbi paraméterek lettek beállítva a burkológörbe számára:

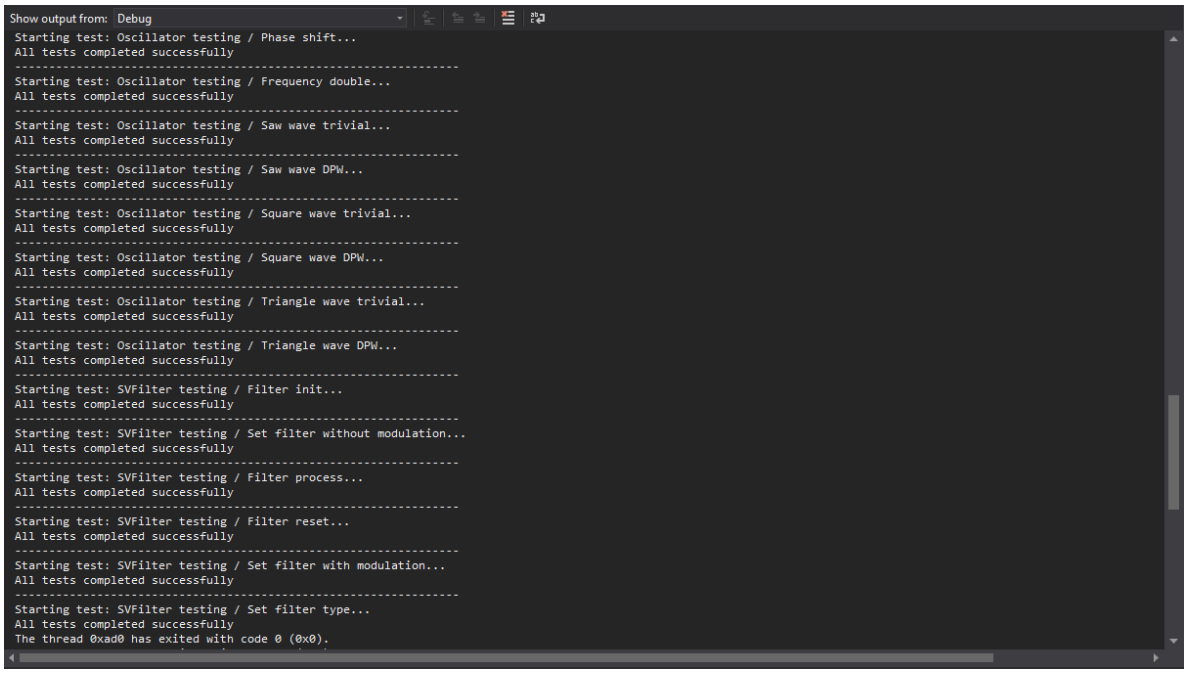
- Felfutás – 0,5
- Csillapítás – 0,5
- Kitartás – 0,5
- Lecsengés – 0,5
- Állapot – Lecsengés állapot
- Amplitúdó – 0

A paraméterek beállítása után lefutott a `process()` metódus. A lefutás után az elvárt eredmény az amplitúdó érték változatlansága, tehát a 0 értéken való maradása. Ez az elvárás a `process()` lefutása után ellenőrzésre került, és a teszt sikeresen lefutott.

A tesztek eredménye a következő ábrákon látható:



8.1 ábra: A tesztek eredménye 1. rész



8.2 ábra: A tesztek eredménye 2. rész

9 TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A dolgozatban megvalósított szintetizátor a szubtrakciós szintézis alapvető elemeit valósítja meg: a jelgeneráláshoz szükséges oszcillátort, a generált jelből a felharmonikusok kivonásához szükséges szűrőt, és a hang dinamikai lefutását szabályzó burkológörbét. Modulálási lehetőségként pedig az alacsony frekvenciájú oszcillátor és a szűrőbe csatlakoztatott burkológörbe áll. Továbbfejlesztési lehetőségként ezen elemek fejlesztése az egyik irány; az oszcillátor esetén a használt DPW jelgenerálási algoritmusok nem működtek megfelelően, ezek kicserélhetőek egy másik típusú jelgenerálási algoritmusra, mint például a PolyBLEP algoritmus. Egyszerűen megoldható fejlesztési lehetőség még több oszcillátor használata; így több jelalak összeadásával új jelalakok állíthatóak elő.

Az alacsony frekvenciájú oszcillátor esetén fejlesztési lehetőség többféle jelalak használata, és a moduláció kiterjesztése több paraméterre, nemcsak az oszcillátor frekvenciájára. Az egyes paraméterek összekapcsolása a moduláló elemmel egy mátrix kapcsolótábla implementálását is igényli.

A szűrő esetén továbbfejlesztési lehetőség többféle szűrő alkalmazása, így például a Moog szintetizátorokban használt létra szűrő használata, vagy véges és végtelen impulzus válaszú szűrők használata.

Új szintézis módszer használata is fejlesztési lehetőség; használható az átlapolódást alapvetően nélkülöző additív szintézis, a frekvenciamoduláción alapuló szintézis és a generált hangokat egy look-up táblából kinyerő hullámtábla szintézis.

10 A SZAKDOLGOZAT ÖSSZEFOGLALÁSA

A dolgozat egy egyszerű, de minden alapvető funkciót tartalmazó szubtraktív szintetizátor megvalósítását tűzte ki célul, és ennek sikeresen eleget tett. Az

irodalomkutatás és konklúzió során megismerésre és feldolgozásra került a szintetizátor megírásához szükséges jelfeldolgozási elmélet: ennek során megismerésre kerültek egy numerikusan vezérelt oszcillátor digitális megvalósításához szükséges elemek, a hatékony és átlapolódásmentes jelgeneráláshoz szükséges algoritmusok és egy fűrészjel példáján keresztül az átlapolódás problémájának a szemléltetése. A szűrő esetén feltérképezésre került a digitális implementáláshoz szükséges jelfeldolgozási elmélet, a digitális szűrők típusai és egy analóg állapotváltozós szűrő digitális megvalósításának módszere. A burkológörbe esetén pedig megismerésre került, hogy az egyes fázisok egy állapotgéppel modellezhetőek, ahol az egyes állapotok az egyes fázisoknak feleltethetőek meg.

A megvalósítás során ismertetésre kerültek a plugin megvalósításának lehetőségei, az egyes keretrendszerek előnyei és hátrányai. Ennek alapján kiválasztásra került a JUCE keretrendszer, majd áttekintésre került a keretrendszer vázlatos felépítése és működése. Ezután részletesen dokumentálásra került a kezelőfelület kialakítása, a jelgenerálási paraméterek lekérdezése és frissítése és a jelgenerálásban résztvevő elemek kialakítása és megvalósítása a JUCE keretrendszer segítségével. Végezetül a szintetizátor egyes elemei tesztelésre kerültek többféle unit teszt segítségével.

A dolgozat megírása során megismertem a hangszintézis egyes módszereit, a szubtraktív hangszintézishez szükséges elemek működését, és átfogó képet kaptam a digitális jelfeldolgozásról. A szintetizátor megvalósítása során gyakorlatot szereztem a C++ programozási nyelv használatában és megismertem a JUCE keretrendszert is.

11 SUMMARY

The goal of the thesis was the realization of a simple subtractive synthesizer with all basic functions needed for sound synthesis, and this was achieved successfully. During the literature search and conclusion, the digital signal processing theory required to write the synth was introduced and processed: the elements required for the digital implementation of a numerically controlled oscillator, the algorithms needed for an efficient and antialiasing signal generation and the problem of aliasing in signal generation through a saw wave example. In the filter section, the signal processing theory required for the digital implementation, the types of digital filters and the method of digital implementation of an analog state variable filter were explored. In the envelope section it was learned that the individual stages can be modeled with a state machine, where the stages correspond to the states.

During the implementation, the possibilities of implementing the plugin, the advantages and disadvantages of each framework were described. Based on this information, the JUCE framework was selected, and the basic structure and operation of the framework were reviewed. The design of the interface, the query and update of the signal generation parameters, the design and implementation of the elements involved in the signal generation using the JUCE framework were the documented in detail. Finally, each element of the synthesizer was tested using several unit tests.

During the writing of the thesis, I get acquainted with several methods of sound synthesis, the elements and operations necessary for subtractive synthesis, and I got a comprehensive picture of digital signal processing. During the implementation of the

synthesizer, I gained experience in using the C++ programming language and got acquainted with the JUCE framework.

12 IRODALOMJEGYZÉK

[1] Ambrits Dániel*:* Analóg szintetizátor modellezése különböző oszcillátor algoritmusokkal, Budapest, Hungary, 2012.

[2] Budai Benjámin Marcell*:* Hammond orgona-szintetizátor megvalósítása, Budapest, Hungary, 2015.

[3] Donald Hummels: Numerically Controlled Oscillators http://web.eece.maine.edu/~hummels/classes/ece486/docs/NCO_tutorial.pdf, accessed 1-30-2021.

[4] Jari Kleimola*:* Design and Implementation of a Software Sound Synthesizer, Espoo, Finnland, 2005.

[5] Jeff Harder: Early History of Synthesizers https://electronics.howstuffworks.com/gadgets/audio-music/synthesizer3.htm, accessed 3-17-2021.

[6] Makovecz Ádám*:* FM szintetizátor megvalósítása VST környezetben, Budapest, Hungary, 2014.

[7] Ross H: A Brief History Of Synthesizers https://www.hi5electronics.co.uk/a-brief-history-of-synthesizers/#:~:text=The%20term%20synthesizer%20was%20first,forks%20that%20were%20stimulated%20electromagnetically., accessed 3-17-2021.

[8] Smith, S.W.*:* The scientist and engineer's guide to digital signal processing. California Technical, San Diego, Calif., 1997.

[9] Szigetvári Andrea, S.Á.*:* Hangdizájn, hangszintézis és hangátalakítás. Typotex Kiadó, 2014.

[10] Vadim Zavalishin*:* The art of VA filter design, 2nd edn., 2018.

[11] Valimaki, V. and A. Huovilainen*:* Antialiasing Oscillators in Subtractive Synthesis, IEEE Signal Processing Magazine, 24(2), 2007, pp. 116–125.

[12] Välimäki, V. and A. Huovilainen*:* Oscillator and Filter Algorithms for Virtual Analog Synthesis, Computer Music Journal, 30(2), 2006, pp. 19–31.

[13] Vesa Välimäki, J.P.: The Brief History of Virtual Analog Synthesis https://www.researchgate.net/profile/Vesa-Vaelimaeki/publication/242020149_The_Brief_History_of_Virtual_Analog_Synthesis/links/00b4952e6420eacc57000000/The-Brief-History-of-Virtual-Analog-Synthesis.pdf?origin=publication_detail, accessed 3-17-2021.

[14] Will Pirkle: Virtual Analog (VA) Filter Implementation and Comparisons v2.0 https://www.willpirkle.com/Downloads/AN-4VirtualAnalogFilters.pdf, accessed 10-28-2021.

13 ÁBRAJEGYZÉK