Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Virtuális analóg szintetizátor megvalósítása VST környezetben

Önálló laboratórium 1. zárójegyzőkönyv
2022/23. II. félév

Csikós Attila (LBD3WL)

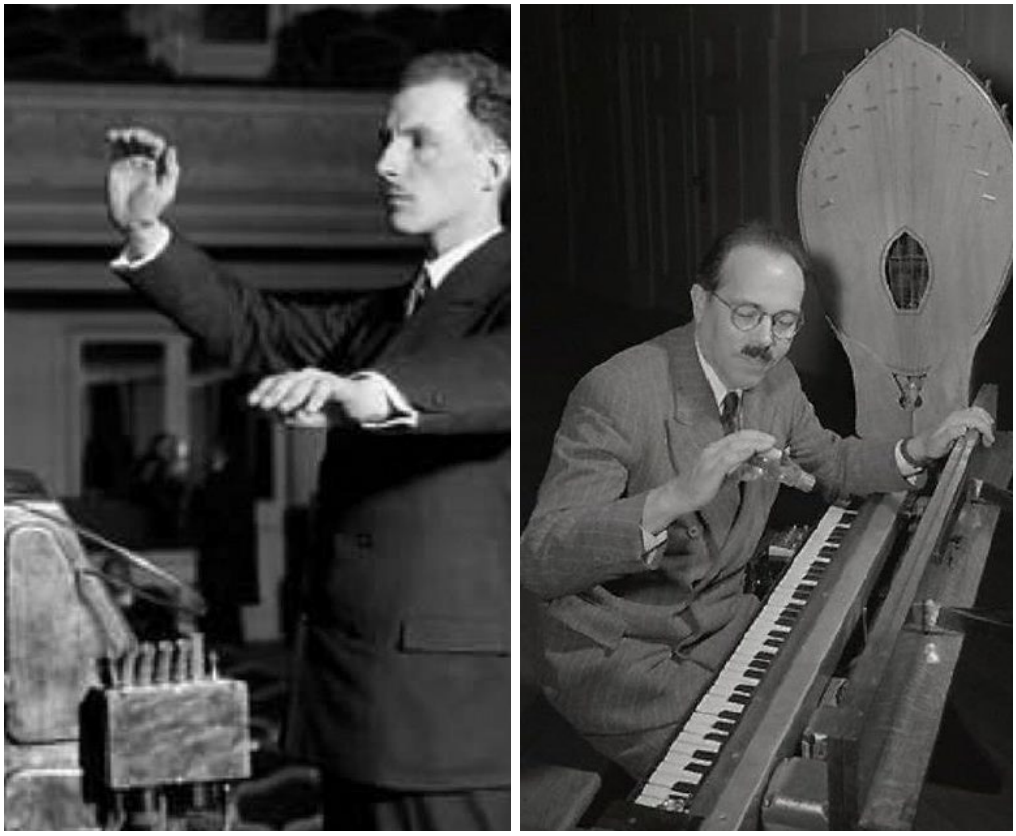
III. évf., villamosmérnök szakos hallgató
BSc, Beágyazott információs rendszerek ágazat

Konzulens: Dr. Bank Balázs docens
Méréstechnika és Információs Rendszerek Tanszék

Tartalom

1. Bevezetés a szintetizátorok világába és történelmébe

A szintetizátorok történelme a 20. század elejétől indul, olyan találmányokkal, mint például a Theremin és az Ondes Martenot *[Szigetvári Andrea – Az elektronikus zene története]*. A viszonylag hosszú múlt ellenére csak az 1960-as és 1970-es években kezdtek felszínre törni a populárisabb zenék világában.



1. és 2. Kép
A Theremin és az Ondes Martenot
[Szigetvári Andrea – Az elektronikus zene története]

Az első szintetizátorok hatalmasok és drágák voltak, illetve főleg akadémiai és kísérleti helyzetekben használták őket. Analóg áramkörökből épültek fel, ezekkel hangot generáltak és manipuláltak. Beállításukat potenciométerekkel és kapcsolókkal végezték. Az egyik ilyen leghíresebb korai hangszer a Moog szintetizátor volt, melyet Robert Moog fejlesztett ki 1964-ben.

Ahogy az lenni szokott, minél inkább elterjedtek, annál olcsóbbak, kisebbek és könnyebben használhatóak lettek. Újabbnál újabb funkciókkal és elemekkel látták el a fejlődő eszközöket, ezzel elérve, hogy a hallgatóknak addig ismeretlen eredetű és típusú hangokat hozzanak létre. A szintetizátorok így lettek elengedhetetlen segédei az elektronikus és pop zenei produceréknek, olyan modellekkel, mint például a Yamaha DX7 és a Roland TB-303.

Az idő elteltével és a modern technika fejlődésével a készítők rájöttek, hogy az analóg hangszintézisnek megvannak a saját problémái és határai. Ezek közé tartozott a méret, helyigény vagy a javítás költségessége. A virtuális analóg hangszintézis alkalmazása nagyban leegyszerűsítette a felhasználók dolgát. Emulálja az analóg hangok melegségét és karakterét, mindezt az új, digitális technikák flexibilitásával és kényelmességével, illetve megoldásokat kínál az analóg jelek mintavételezése során okozott az átlapolódás miatti zavarok és kellemetlen hangok javítására.

2. A feladat pontosítása

A félév során kitűzött cél egy olyan program megírása, mely képes modellezni a régi analóg szintetizátorok hangalkotási módszereit. A projekt elkészítéséhez pontosítanom kell a célkitűzésemet, ezért a hozzám legközelebb álló Roland TB-303 analóg szintetizátort választottam modellezésem céljául.

3. Kép
Az analóg Roland TB-303 szintetizátor

Az 1980-as években kiadott modell nem nyerte el a tervezett vevőréteg tetszését, mivel eredeti céljában, a basszusgitár helyettesítésében nem volt sikeres. Később viszont az elektronikus zene korai alkotóinak kezébe kerülve egy, a mai napig meghatározó hangzást teremtett. Innen erednek az Acid-House, Acid-Techno és Psy-Trance jellegzetes hangjai.

Az alapos modellezéshez meg kell ismernünk a szintetizátoron belüli elemeket és azoknak jellemzőit. Összesen egy darab oszcillátor található benne, ellenben két opció is rendelkezésre áll, még pedig, hogy fűrész- vagy négyszög jelet generáljon, majd használjon fel a hangalkotáshoz. A kiválasztott jelalak ezután egy 24 dB/oktáv-os aluláteresztő szűrőn halad át [*Ov Valle - The Fall and Rise of the TB-303*], melynek állítható a játék közbeni törésponti frekvenciája és rezonanciája. A jelút a következőkben egy amplitúdó burkológörbe generátoron halad keresztül, melynek csak a Decay-paramétere állítható.

A szintetizátor fontos tulajdonsága még, hogy billentyűket használó élő játék helyett előre be kell programozni a lejátszandó hangjegyeket. A megadott maximális tizenhat hangból álló dallam visszajátszása közben a felhasználó szabadon állíthatja a már felsorolt paramétereket, így elérve a folyamatosan változó, modellezendő hangzást.

3. A hangszintézis módszere

A program tervezésénél elsődleges szempont volt, hogy az elkészített verzió megőrizze az analóg hangszer jellegzetességét. Mindezt úgy, hogy minél inkább átlapolódás- és zajmentes legyen a végeredmény.

Az általam választott szintetizátor hangalkotási módszere a szubtraktív szintézis, melynek lényege, hogy egy harmonikusokban gazdag spektrumú jel generálását követően azt algoritmusokkal, szűrőkkel módosítjuk, így az én programom is ezen a módszeren alapult. Az ezzel kapcsolatos problémákat és megoldásokat később fogom kifejteni.

4. VST-re való áttérés

A program könnyedebb használatának és újabb funkciók implementálásának érdekében a már elkészült MATLAB kódot átalakítottam, és VST környezetben valósítottam meg.

A VST (Virtuális Stúdió Technológia) plugin-ek a szoftverek azon csoportjába tartozik, melyeket zeneszerkesztő programokban (DAW) alkalmaznak az egyedi effektek elérésének érdekében. Ilyenek lehetnek az egyszerűbb szűrők vagy EQ-k, de akár bonyolultabb hangszert imitáló szintetizátorok is.

Az általam készített szintetizátor plugint a nyitott forráskódú objektum orientált JUCE keretrendszer segítségével hoztam létre. Ez egy C++ alapú függvénykönyvtár, melyet 2001-ben hoztak létre audio- és zeneszerkesztő

szoftverek fejlesztésének céljából. Előnye, hogy a legtöbb platformmal kompatibilis, mint például Windows, macOS és Android. Továbbá megoldásokat kínál arra is, hogy egyedi és komplex felhasználói felületek legyenek létrehozhatók. A következő fejezetekben az ebben a rendszerben elkészült megoldásomat fogom bővebben kifejteni.

5. A szoftver alapjai és főbb változói

A program alapjai a PluginProcessor és PluginEditor forrás fájlok. Az előbbi felelős a jelfeldolgozásért és a számításokért, míg a második a grafikai felületet valósítja meg.

A PluginProcessor három fő függvényből épül fel. Az első a *preparetoPlay*, amely helyes működés esetén csak egyszer fut le, a jelgenerálás előtt. Feladata, hogy szinkronizálja a jelfeldolgozásban kritikus adatokat az összes jelet befolyásoló objektumban. Ilyen adatok a mintavételi frekvencia, a csatornák száma és a maximum minták száma, amelyek teljesen kitöltenek egy audiobuffert. Fejlesztés közben figyelni kellett arra, hogy az összes jelet alakító osztálynak legyen ilyen prepare típusú függvénye, amelyet egyenes ágon vagy más elemeken keresztül meghívhat a PluginProcessor.

A második építőelem a *createParams* függvény, amely szintén csak egyszer fut le a működés során. Feladata, hogy létrehozza a grafikai felületen változtatható paramétereket, inicializálja ezen elemek határértekeit, kezdeti értékeit és az identifikációért felelős string típusú változókat. A paramétereket a JUCE által ajánlott és megvalósított AudioProcessorValueTreeState vector-ba helyezi, hogy a PluginProcessor és a PluginEditor hozzáférhessen.

Változtatható paraméterek:

- oszcillátor által generált hullámforma – szinusz, négyszög, fűrészfog
- burkológörbe generátorok paraméterei – attack, decay, sustain, release
- szűrő típusa – alul-, felül-, sáv-áteresztő
- töréspontí frekvencia
- rezonancia

Az utolsó nagyobb függvény a *processBlock*, ami minden alkalommal lefut, amikor a szoftver egy újabb üres audiobuffer feltöltésére készül. Első lépésben a PluginEditor által frissített tulajdonságú paramétereket kiolvassa az AudioProcessorValueTreeState-ből, hogy a helyes beállításokkal kezdjen el jelet generálni, majd pedig meghívja a szintetizátor objektum process típusú függvényét, amely ténylegesen kitölti az új audiobuffert.

6. Synth objektum és SynthVoice osztály leírása

A PluginProcessor header fájlában példányosítottam egy synth objektumot, amelyet a JUCE által definiált Synthesizer osztály ír le, amely még két másikra bontható. Az egyik a SynthesizerSound, a másik pedig a SynthesizerVoice, melyek közül csak az utóbbira volt szükségem. Ebből örökítve létrejött az általam megírt SynthVoice osztály, melyben privát elemként vannak példányosítva a jelút lépcsőfokait megvalósító elemek, mint például: audio buffer (synthbuffer), oszcillátor (my_oscillator), hangszintszabályzó (gain), burkológörbe generátorok (adsr, modAdsr) és a szűrő (filter).

A legfontosabb függvény a *renderNextBlock*, amelyet a PluginProcessor hív meg a processBlock végén, ami egy feltöltött audioblock-kal tér vissza. Az így képzett bufferek egymásutánja valósítja meg a kimeneti hangot.

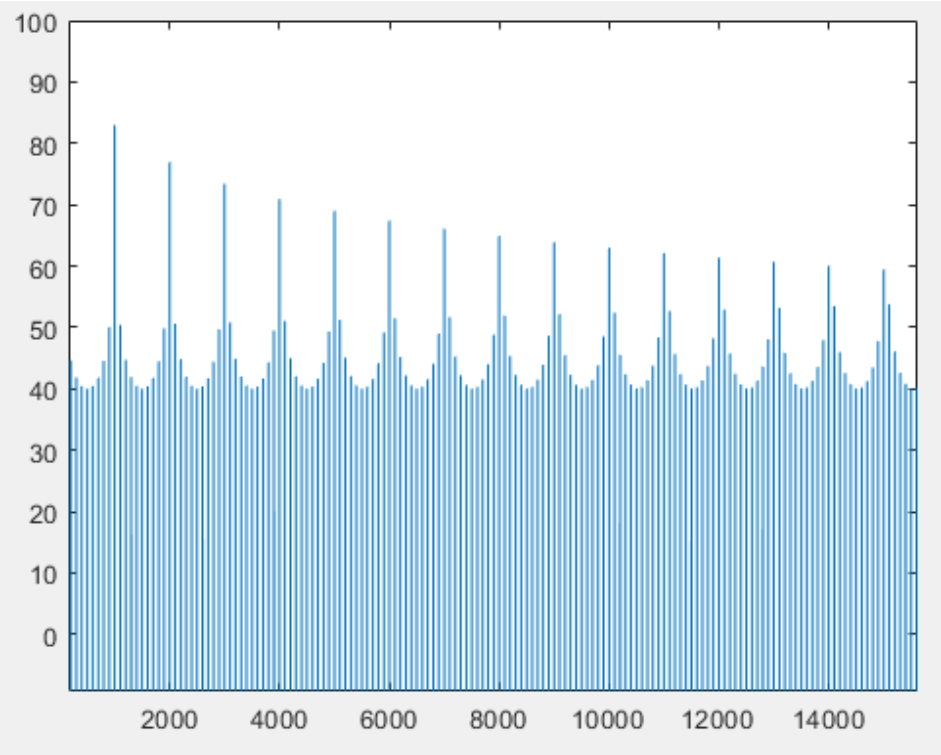
Mindenekelőtt kitöröljük a synthbuffer eddigi tartalmát, a hamis információk elkerülése végett. Az oszcillátor process függvénye feltölti a buffert, majd a hangszint szabályzó gain objektum újból végigmegy a generált jeleken, hogy a SynthVoice *preparetoPlay* függvényben megadott 0 és 1 közé eső érték szerint csökkenthesse a hangerejüket. Innentől a burkológörbe generátoron a sor, hogy módosítsa a hang időbeli jellemzőit a paraméterek alapján. Végül a szűrőobjektum moduláló hatására elkészül az audioblock.

Fontos még kiemelni a SynthVoice osztály másik két függvényét, amely a *startNote* és *stopNote*. Mindkettő a PluginProcessortól kapott MIDI információk alapján jelez az ADSR-eknek, hogy a bufferben pontosan hol fog kezdődni és végződni a számunkra fontos jel. Továbbá a *startNote* függvényben a MIDI információ alapján meg tudjuk adni az oszcillátornak, hogy milyen frekvenciájú hangot generáljon.

7. Oszcillátor osztály leírása

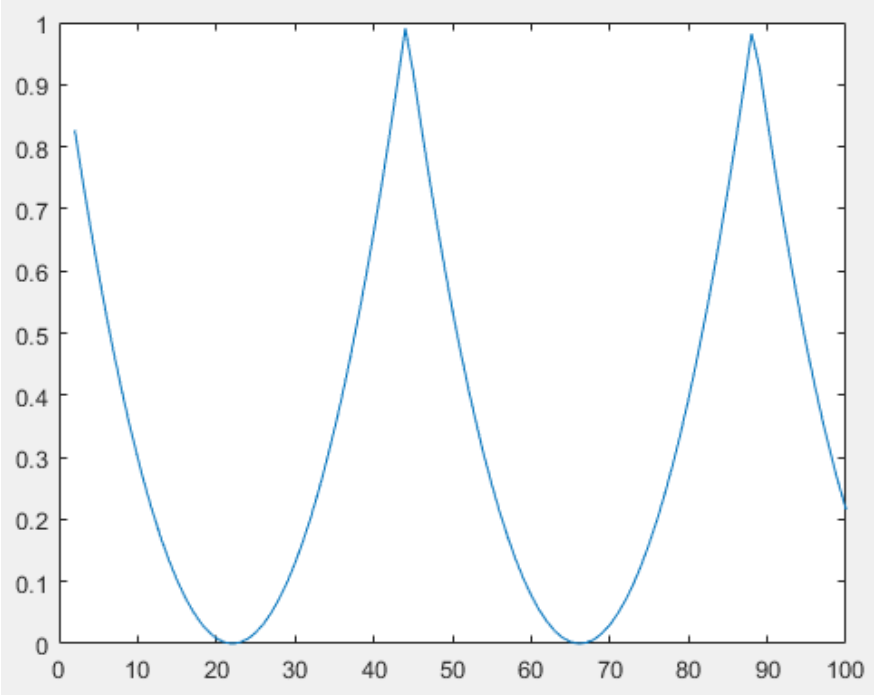
A plugin alapját adó oszcillátor három hullámforma közül egy kiválasztására ad lehetőséget egy grafikus felületen keresztül. A legördülő listából kiválaszthatjuk az általunk generálni kívánt, szinusz, négyszög vagy fűrészfog hullámformájú jelet. Bővebben a fűrészjel megvalósításával fogunk foglalkozni, mivel a másik kettő a JUCE által felkínált algoritmusokon keresztül történik.

A triviális megoldás az analóg jel mintavételezése, viszont az így kapott spektrum periodikus és végtelen széles lenne, ami miatt az átlapolódott komponensek összeadódnának és kellemetlen hangzást okoznának.

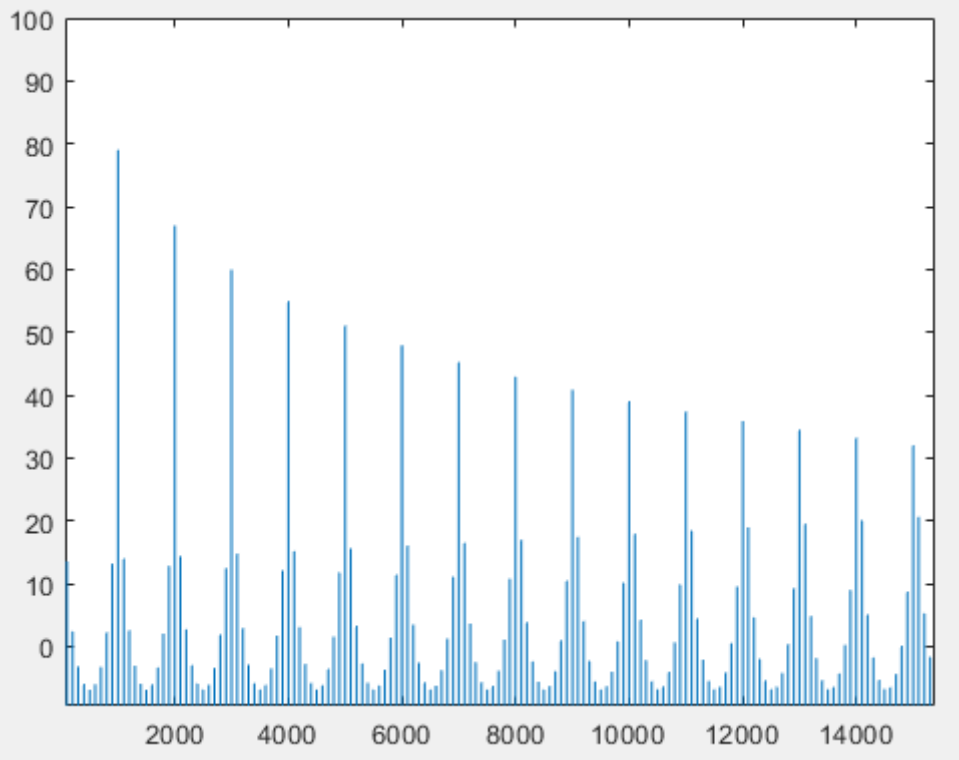


1. Ábra
Számlálóval és modulo függvénnyel generált fűrész jel spektruma

Ezen probléma megoldására az itt megjelölt [*Välimäki and Huovilainen - Oscillator and filter algorithms for virtual analog synthesis*] forrásban említett DPW (Differentiated Parabolic Waveform) algoritmust használtam. Az eljárás lényege, hogy az analóg jel integráltját mintavételezem, ezzel megkétszerezve a spektrum dekádonkénti csökkenését, így minimalizálom az átlapolódást és megszabadulok a kellemetlen hangzást okozó harmonikusoktól. Az algoritmus alapja egy egyszerű növekvő ciklikus számláló, amely -1 és +1 között generál számokat a jel frekvenciája és a mintavételi frekvencia által meghatározott lépésközönként. A kapott számadatok integrálásával érjük el a parabolikus hullámformát, majd diszkért időben differenciáljuk, ami megfelel egy $H_D(z) = 1 - z^{-1}$ átviteli karakterisztikájú szűrővel való módosításnak.

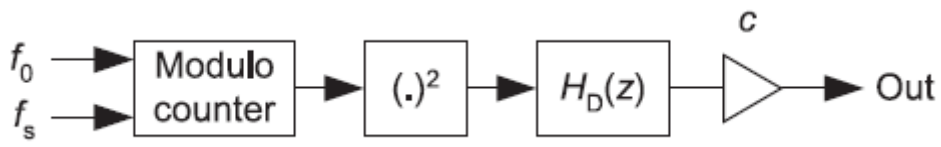


2. Ábra
Parabolikus hullámforma

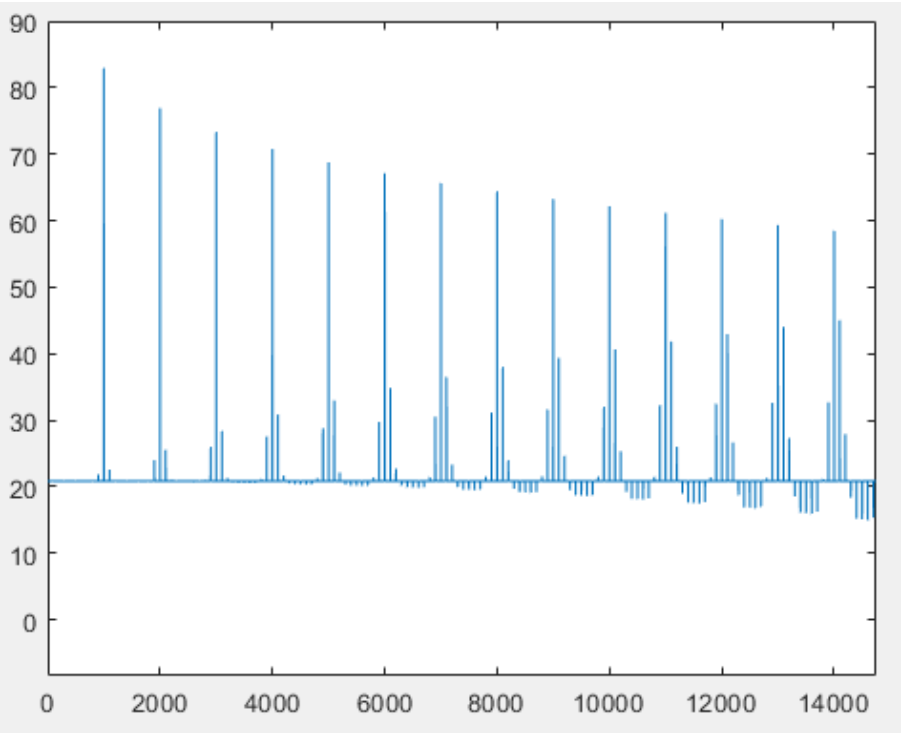


3. Ábra
Parabolikus jel spektruma

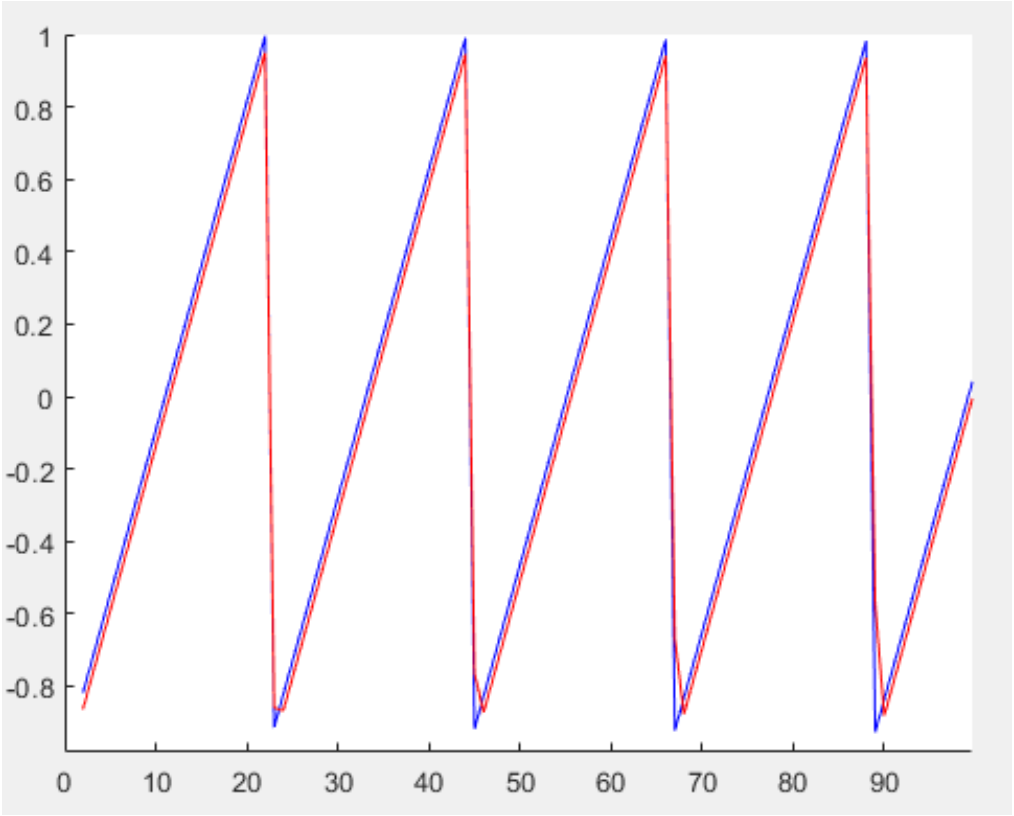
Végül az eredeti hullámforma eléréséhez visszaskálázom az értékeket egy $c = \frac{f_s}{4*f_0}$ együtthatóval, ahol f_0 a jel frekvenciája és f_s a mintavételi frekvencia.



4. Ábra
DPW algoritmus blokkvázlata
[*Välimäki and Huovilainen - Oscillator and filter algorithms for virtual analog synthesis*]

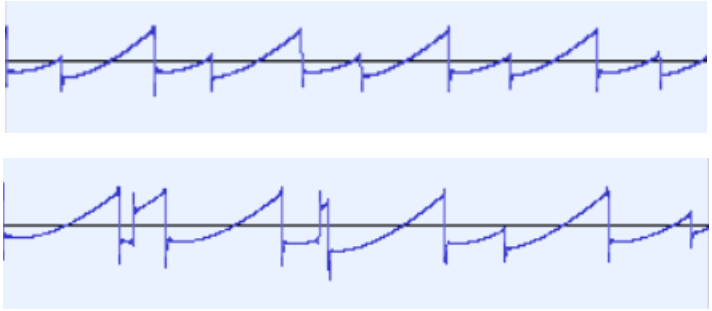


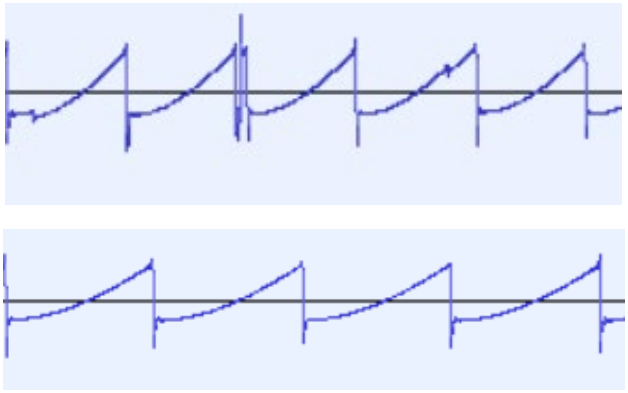
5. Ábra
DPW algoritmussal kapott jel spektruma



6. Ábra
Kék színnel – Modulo függvény által generált hullámforma
Piros színnel – DPW algoritmus végére kapott hullámforma
Látható, hogy az algoritmus végére a hullámforma +1 -1 átmenetébe bekerül egy törés, mellyel a meredekség, azzal pedig az átlapolódás is csökken.

Az algoritmust először Matlab-ban teszteltem, ahol egy általam lejátszani kívánt frekvenciához rendeltem egy mintaszámot, így megvalósítottam egy specifikus hangot és annak hosszát. A mi esetünkben viszont valós időben történik a hang generálása és a lejátszási idő felhasználónként változhat. Ennek eléréséhez oly módon kellett az algoritmust átalakítani, hogy az oszcillátor mindig csak a következő elemet adja ki és addig, amíg azt a MIDI információ azt kimondja. A fejlesztés ezen pontján hibába ütköztem. Frekvenciától és mintavételi frekvenciától függően a generált hang, hol hibátlan, hol hibás volt. Az utóbbi esetekben a jel, mely a kimenetre jut repetitíven zavart tartalmaz.





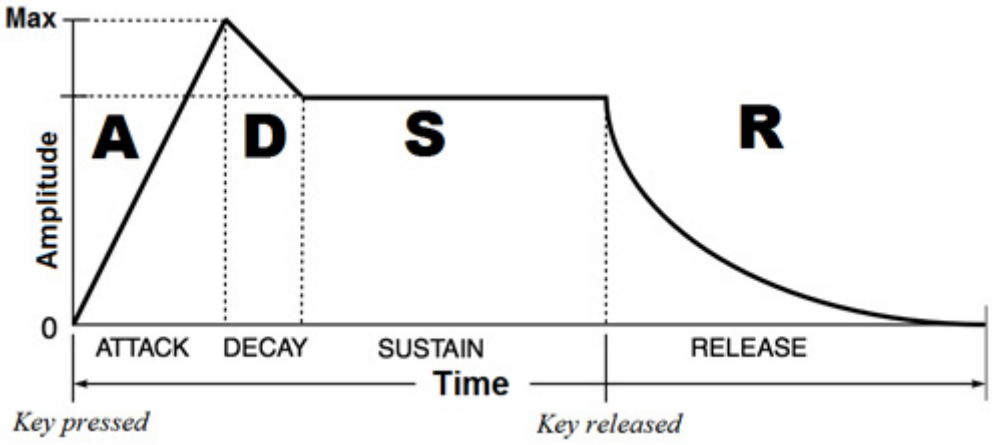
7. Ábra
Audacity-vel felvett hullámformák
Frekvenciák: 233.082 Hz, 246.942 Hz, 293.665 Hz, 311.127 Hz
Látható, hogy csak az utolsó frekvenciához tartozó hullám hibátlan

A tesztek során kiderült, hogy az oszcillátor által generált számadatok hibátlanok, hiszen megegyeznek a Matlab modell értékeivel. Eleinte arra gondoltam, hogy nem megfelelően szinkronizálom a mintavételi frekvenciát, de ezt kiküszöbölve is megjelent a probléma. A program jelen állapotában a hiba forrása még ismeretlen, de arra gyanakszom, hogy az elkészített audioblock-ok egymásután illesztése közben generálódik, ami okozza a nem kívánt zavarokat. Ez magyarázhatná a zavar „csak” bizonyos időközönkénti előbukkanását, de még ez a teória sem ad választ arra, hogy miért csak bizonyos frekvenciákon lép fel. Ezt szeretném orvosolni a jövőben.

8. Amplitúdó burkológörbe generátor

Az oszcillátor által feltöltött buffert átadjuk az amplitúdó burkológörbe generátornak, ami a hangerő változtatásért felelős az időben. Ennek megértéséhez ismernünk kell a paramétereket, és tulajdonságaikat.

Az Attack paraméter határozza meg, milyen gyorsan érje el a hang a kezdeti 0 hangerőről a maximális jelszintet a kiváltó billentyű lenyomása után. A maximális szint elérése után, a Decay paraméter által meghatározott szakaszba ér, ahol csökkenni kezd a Sustain paraméter által meghatározott értékig. A Decay paraméter rövidre választásával elérhetjük, hogy a hang sokkal ütésszerűbb, élesebb legyen. Ennek befejeztével jut el a Sustain szintre, ahol már nem modulál, csak stagnálva tartja a szintet, egészen a billentyű elengedéséig. A Release folyamat következik. Az ehhez tartozó paraméterrel a lecsengés gyorsasága határozható meg.



9. ADSR programon belüli implementációja

Minden egyes szakaszban a modulálandó minta egy, a burkológörbét leíró változóval szorzódik meg, ami csak 0 és 1 között vehet fel értéket. A maximális jelszint eléréséhez eggyel, míg némításkor nullával való szorzásra van szükség.

A SUSTAIN szakasz modulációjának megvalósítása némileg eltér a paraméter felhasználás tekintetében a többitől, így azt szeretném kifejteni. Ameddig a jel ebben a szakaszban van az összes mintát megszorozzuk a SUSTAIN paraméterrel, tehát ez egyszerűen megegyezik a burkológörbe pillanatnyi értékével.

A további esetekben a felhasználó által beállított szám egy másodpercben értendő értéket ad meg, ami alapján a program kiszámolja, mekkora legyen a lépésköz, amellyel növelni vagy csökkenteni kell a burkológörbe értékét, hogy adott idő alatt eljussunk adott szintre. Például, ha az ATTACK paraméter egy olyan időt határoz meg, amibe összesen 5 db minta fér, akkor a program az első mintát 0,2-vel a másodikat 0,4-el szorozza meg és így halad tovább, azért, hogy az ötödik mintához hozzárendelt burkológörbe érték (szorzó) már 1 legyen, tehát a lehetséges maximum.

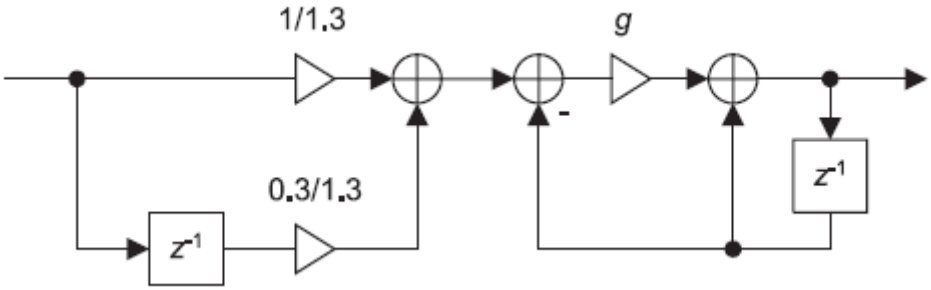
10. Szűrőt moduláló ADSR

A szűrőt moduláló ADSR ugyanazon az elven képez a mintákhoz egy, a burkológörbét leíró változót, mint az amplitúdó ADSR. A különbség a kettő között, hogy itt nem a mintát szorozzuk meg ezzel az értékkel, hanem a szűrő törésponti frekvenciáját. Ezzel érjük el a hanghatás időbeni változását.

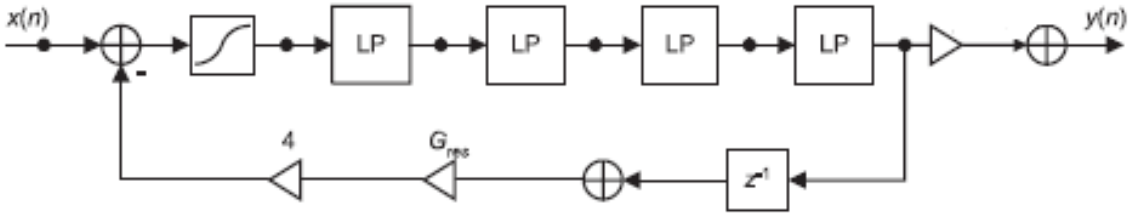
11. Szűrő kód leírása

Eleinte a digitális rezonáns mini Moog szűrőt szerettem volna implementálni, amely egy négypólusú rezonáns szűrő állítható a töréspont frekvenciával és

rezonanciával. Alappillére egy egypólusú szűrő melyből négy sorba kötésével megkapható a négypólusú aluláteresztő szűrő. A jelutat egy nemlineáris elemmel is kiegészítem, hogy hűbben tükrözze az eredeti analóg szűrő hangját, míg a rezonanciát a $Gres$ változó függvényében, a visszacsatoló ágban érem el.



8. Ábra
Egypólusú aluláteresztő szűrő blokkvázlata
[*Välimäki and Huovilainen - Oscillator and filter algorithms for virtual analog synthesis*]



9. Ábra
Négypólusó aluláteresztő szűrő blokkvázlata
[*Välimäki and Huovilainen - Oscillator and filter algorithms for virtual analog synthesis*]

$w_c = \frac{2\pi f_c}{f_s}$, ahol f_c törésponti frekvencia

$$g = 0{,}9892w_c - 0{,}4342{w_c}^2 + 0{,}1381{w_c}^3 - 0{,}0202{w_c}^4$$

$0 < C_{res} < 1$ rezonancia paraméter

$$G_{res} = C_{res}\ (1.0029\ +\ 0.0526w_c\ -\ 0.0926{w_c}^2\ +\ 0.0218{w_c}^3)$$

Ezzel eleget teszek annak a kitételnek, hogy a szűrő törésponti frekvenciája és rezonancia paramétere változtatható legyen. Matlab-os implementációban ez tökéltesen működött, VST környezetben azonban ugyanolyan problémába ütköztem, mint az oszcillátor esetében, tehát bizonyos időközönként zavar jelenik meg a hangban, kellemetlen zúgással és kattogással kísérve.

A szoftver minőségének romlása elkerüléséhez végül egy a JUCE által felajánlott szűrőt használtam a kész programomban, aminek felépítése egyszerűbb, de még így is megfelel a szűrőtől elvárt követelményeimnek. Emellett a grafikus felületen egyszerűen választható volt, hogy a szűrő alul-, felül-, vagy sáv-áteresztő legyen.

12. Felhasználói felület

A már említett PluginEditor felelős a grafikai felület létrehozásáért és működtetésért. Nagy segítséget nyújtottak JUCE által felkínált és előre megírt függvények, amelyek a háttér színezését, a változtatható paraméterek folyamatos frissítését, a csúszkák, tekerők, combobox-ok megvalósításáért feleltek. Én hoztam létre a különböző komponensekért felelős osztályokat, amelyekben előírtam, hogy milyen, kapcsolók vagy tekerők kellenek a paraméterek megváltoztatásához. Ezek mellett beállítottam a plugin ablakán hol helyezkedjenek el ezek az elemek, a komponensen belül. Komponensnek nevezem a felhasználói felület azon részeit, amelyek egy specifikus jelfeldolgozó objektum kinézetéért felelnek. A megvalósított elemek: Oszcillátor és az ahhoz tartozó legördülő lista, a burkológörbe generátorok és a hozzátartozó csuszkák, illetve a szűrő a combobox-szal és a két tekerővel. Ezek után a Plugineditorban csak az volt a feladatom, hogy példányosítsam ezeket és láthatóvá tegyem őket, ízlésesen felosszam az ablak felületén őket.

13. Összefoglaló és értékelés

A félév során képes voltam a témalaborban elkészített modellem újraimplementálására egy a számomra eddig ismeretlen környezetben. A modell nem hibátlan, viszont sokkal többet tud elődjéhez képest. Megtapasztalhattam egy keretrendszernek az előnyeit és hátrányait fejlesztés közben, illetve elmélyíthettem C++ tudásomat is. A munkám jelenlegi állapotában még javításra szorul, illetve tovább is fejleszthető. Például megvalósításra vár egy komolyabb, az eredeti szintetizátor használatát jobban visszaadó grafikai felület.

14. Felhasznált irodalom

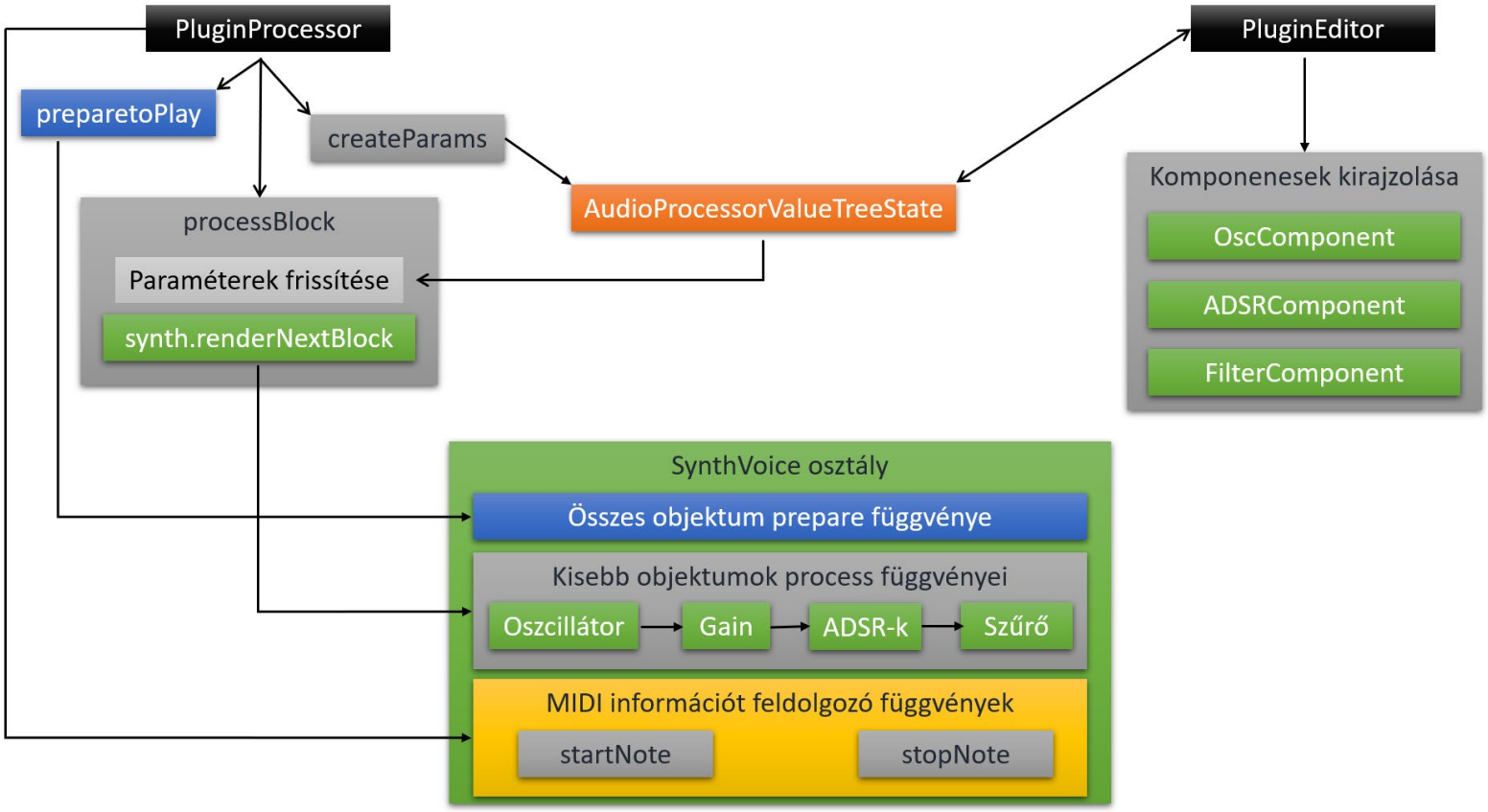
Szigetvári Andrea – Az elektronikus zene története – 2013

Ov Valle[Roland US] - The Fall and Rise of the TB-303 – 2013. március 28.

http://www.rolandus.com/blog/2013/03/28/tb-303-acid-flashback/

Vesa Välimäki and Antti Huovilainen - Oscillator and filter algorithms for virtual analog synthesis - Computer Music Journal, 30(2):19 31 – 2006 június

15. Melléklet



Blokkdiagram az elkészült programról