Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Makovecz Ádám

FM SZINTETIZÁTOR MEGVALÓSÍTÁSA VST KÖRNYEZETBEN

KONZULENS

Dr. Bank Balázs

BUDAPEST, 2014

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Makovecz Ádám**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2014. 12. 18.

...……

Makovecz Ádám

Kivonat

Az első szintetizátorok analóg áramkörökkel működtek, a kezdetleges technológia miatt a lehetőségek korlátozottak voltak. Az 1970-es években a digitális technika fejlődésével elterjedtek az absztrakt algoritmusokon alapuló szintetizátorok. Az analóg módszerekhez képest olcsóbb eszközök álltak rendelkezésre, a processzorok számításaival megbízhatóbb és több különböző hangzás volt elérhető. Egyik ilyen absztrakt módszer az FM hangszintézis, ami az egyszerű algoritmusával jellegzetes hangzást ért el. Az 1990-es évek hangkártyáiban megjelentek az FM hangszintézist használó chipek, egyik ilyen technológián alapuló hardver az OPL3-as volt.

A technológia további fejlődésével már stúdiószoftverekkel dolgoznak a zeneszerzők. A régi technológiákat szívesen alkalmazzák az ilyen programokban is, a szakdolgozatom témája is egy FM hangszintézist előállító szoftver.

A dolgozat első fejezet az absztrakt algoritmusokkal foglalkozik, a Karplus-Strong és waveshaping módszer mellett az FM hangszintézissel. A második fejezet az OPL3 tulajdonságait írja le, a kapcsolások és jelalakok ismertetésével. A harmadik fejezet az OPL3 MATLAB programban megírt szimulációját mutatja be. A negyedik fejezet a VST plugin megvalósításához használt programok leírása található. Az ötödik fejezetben a program C++ megvalósítását és az elkészült program elemeit mutatja be. Az utolsó fejezetben a megírt programot hasonló FM pluginokkal hasonlítom össze.

Abstract

The first synthesizers worked with analog circuits, and because of the high price of the technology, the options were few. In 1970s, by the development of digital technology, synthesizers based on abstract algorithms have become popular. Compared to analog techniques, these devices were cheaper, and with the calculations of the built in processor, the possibilites of sound synthesis methods increased and they were more reliable. FM sound synthesis was one of the most popular abstract method for sound synthesis, with its simple algorithm and characteristic sound. In the 1990s, the FM synthesis based chips were implemented in many audio cards used in computers. One of these chips was the OPL3.

Nowadays, with the further progress in technology, composers work with studio software. There is still a need to emulate older technologies in their software, and the topic of this thesis is about the implementation of FM synthesis as a software plugin.

The first chapter of my thesis describes the abstract algorithms, the Karplus-Strong, waveshaping and the FM sound synthesis methods. The next chapter describes the properties of the OPL3 chip, with its waveforms and oscillator connections. The third chapter shows the MATLAB simulation of the OPL3 sound chip. In the fourth chapter, all the used programs and software libraries are outlined that are used for the development of VST plugin. The fifth chapter describes the C++ software classes and compares the C++ implementation with the MATLAB simulations. The sixth chapter compares other FM based plugins with the plugin developed in this work.

1 Bevezetés

A hangszintézis módszerek az első szintetizátorok megjelenése óta rengeteget fejlődtek. Kezdetben analóg módszereket használtak a zeneszerzők, melyek csak egyszerű jelalakokat generáltak. A műszaki lehetőségek korlátozottak voltak, így egészen a 70-es évekig várni kellet az új technológiák megjelenésére. A digitális áramkörök elterjedésével az analóg módszerek háttérbe szorultak, az absztrakt algoritmusok felváltották őket. Az FM hangszintézis a 80-as évek elektronikus alapú zenéinek leggyakrabban használt technológiája volt. Jellegzetes hangzásával, olcsóbb és megbízhatóbb kivitelezésével az előzőleg alkalmazott analóg áramköröket teljesen felváltotta. A szintetizátorokon kívül a számítógép hangkártyáiban is megjelentek az FM hangszintézist használó chipek. A PC hardvereiben gyakran megtalálhatóak voltak a SoundBlaster hangkártyák, amik az OPL chipek változataival működtek. Az OPL3-as chip is FM hangszintézist használt.

Manapság a hangok előállítását szoftveresen végzik el. A modern programok leggyakrabban az akusztikus hangszerek felvett hangmintái segítségével szintetizálja a hangokat. Az internet elterjedésével több millió alkalmazást találhatunk. A felvett minták felhasználásán kívül a felhasználók újra elkezdték alkalmazni a régi technológiákat. Leggyakrabban az analóg áramköröket szimulálják a szoftverek, viszont számos FM technológián alapuló programot is lehet találni.

A feladatom az OPL3 chip működéséhez hasonló szoftver implementálása volt VST környezetben. Az interneten található programokhoz képest letisztultabb grafikus felületű, sokszínű program megírása volt a célom.

A szakdolgozatom első fejezetében az absztrakt algoritmusokon alapuló hangszintézis módszereket tekintem át. A második fejezetben megismerhetjük az OPL3 jellegzetességeit. A chip szimulációját először MATLAB környezetben valósítottam meg, ahol jelalakokat és kapcsolásokat ismertetem. A C++ alkalmazás megírásához szükséges programokat a következő fejezetben mutatom be. A hatodik fejezetben a működést létrehozó programrészletekkel ismerkedhetünk meg. Végül programomat összehasonlítottam több hasonló elven alapuló alkalmazással.

2 Absztrakt algoritmusok

2.1 Bevezetés

A különböző szintézismódszereket [1] alapján foglalom össze. A hangszintézis módszereket J. O. Smith [2] 1991-ben négy csoportra osztotta fel: absztrakt módszerek, felvett hangok feldolgozása, fizikai és spektrum modell alapú módszerek. Az absztrakt módszerek különböző algoritmusok segítségével, már létező hangok és jelalakok tulajdonságainak megváltoztatásával új hangokat képesek előállítani. Az első ilyen kísérleteket 1920-ban, különböző lejátszási sebességű fonográfokkal végezték el. 1950-ben megalapították a Studio de Musique Concréte-t Párizsban, ahol a zeneszerzők felvett hang alkotóelemeivel dolgoztak.

Az absztrakt algoritmusok segítségével működő hangszintetizátorok és eljárások főleg az 1970-es évektől az 1990-es évek közepéig voltak népszerűek. Az elkövetkezendő években a technológia folyamatos fejlődésének köszönhetően felváltották őket az újabb módszereken alapuló eszközök.

Ebben a fejezetben az absztrakt módszerek három fő típusával foglalkozom, a waveshaping-gel, a Karplus-Strong módszerrel és az FM szintézissel.

2.2 Karplus-Strong módszer

Kevin Karplus és Alex Strong egy egyszerű, mégis jó minőségű hangszintézis módszert fedezett fel 1983-ban, mely kiváló húr- és dobhangokat képes létrehozni [1]. Beépített hullámtábláját az idő során folyamatosan változtatja, ezért Smith az absztrakt algoritmusok közé sorolja. Későbbiekben a fizikai modell alapján működő szintézisek egy speciális eseteként lett elkönyvelve.

Egy gitár húrjának pendítésekor először az harmonikus komponensekben gazdag hangot bocsájt ki. Az idő folyamán ez a hang folyamatosan veszít a nagyfrekvenciás komponenseiből és elhalkul. A Karplus-Strong módszer ezt a jelenséget modellezi.

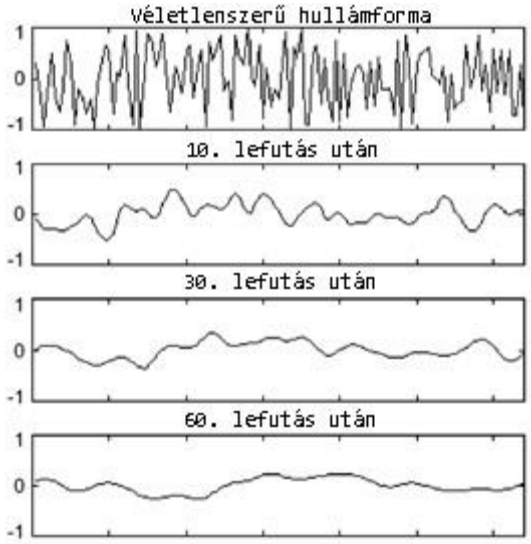
A módszer az egyszerű hullámforma szintézistől eltérően nem felvett hangszerek jelalakjait tárolja, hanem a memória véletlenszerű számokkal van feltöltve. A hang generálása során, ha csak a tárolt adatokat olvasná ki, felharmonikusokban gazdag periodikus hangot eredményezne, melynek periodikussága az adott tároló

méretétől függ. A valóságban az algoritmus minden egyes olvasás során megváltoztatja az olvasott adat értékét a memóriában, átlagolja az előző értékkel. Az átlagolás az alábbi módon írható le:

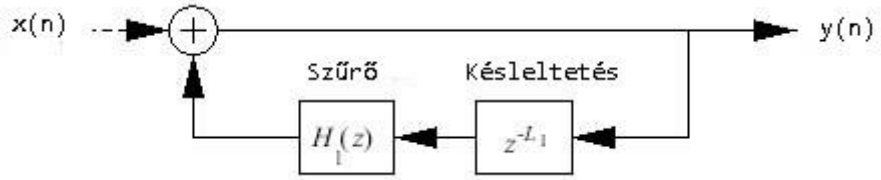
$$Y_t = \frac{1}{2}(Y_{t-p} + Y_{t-p-1}) \qquad (1)$$

ahol Y_t a t időpont szerinti mintája, p a késleltetés száma.

Az átlagolás miatt minden olvasás után az adatok közötti különbség kisebb lesz, a jelalak ugrásai mérséklődnek. A nagyfrekvenciás komponensek az idő elteltével folyamatosan elhalkulnak. A módszer egy aluláteresztő szűrővel, egy pozitív visszacsatolással és egy késleltetéssel valósítható meg.



1. ábra: A kezdeti és az idő múlásával létrejövő hullámforma [3]



2. ábra: Karplus-Strong algoritmus blokkvázlata [3]

2.3 Waveshaping

A waveshaping, más néven a nemlineáris torzítás egy egyszerű jelből különböző, felharmonikusokban gazdag hangot képes előállítani egy statikus karakterisztika függvény segítségével. Az első kísérleteket 1969-ben Risset végezte el, majd függetlenül egymástól Arfib és Le Brun 1979-ben írta le az eljárás matematikai

alapjait [1]. A waveshaping felhasználásával befolyásolni lehet a kimeneti jelalak spektrumát a bemenő jel torzításával.

A waveshaping módszer egyszerűen leírható egy torzító függvény alkalmazásával, általános képlete [4,5]:

$$y = f(x) \tag{2}$$

ahol x a bemenetjel, y a kimenet. A függvény felhasználótól függően akármilyen lehet, viszont általánosan a függvény nulla bemeneti értékhez nulla kimeneti értéket rendel. Az algoritmus egy hullámtábla segítségével tárolja a függvény értékeit. Ennek paraméterei:

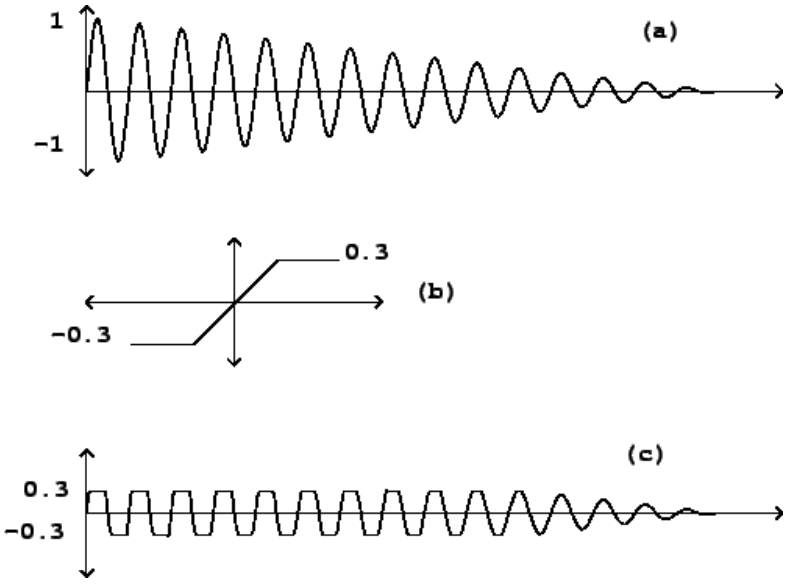
- a kimenet intervalluma
- a bemeneti függvény értékei
- interpoláció módja

Általában lineáris interpolációt használnak abban az esetben, mikor egy bemeneti mintához a hullámtábla nem rendel hozzá kimeneti mintát. Ekkor a program felveszi a két szomszédos minta értékét, és azokat súlyozottan átlagolja, így adva egy új eredményt. Arfib és Le Brun felfedezte, hogy a felharmonikus komponensek arányai közvetlenül és pontosan állíthatóak, amennyiben Chebyshev polinomokat használunk a torzításhoz. Ha egy *n*-ed rendű polinomot használunk torzításhoz, és a bemeneti függvény szinuszos, frekvenciája *f*, akkor a kimeneti jelalak is szinuszos, frekvenciája *n*f* lesz. A polinomok általános formája:

$$T_{k+1}(x) \; = \; 2xT_k(x) - \; T_{k-1}(x) \tag{3}$$

ahol $T_0(x) = 1$ és $T_1(x) = x$.

Az alábbi képen látható egy példa, ahol az (a) a bemeneti jelalak, (b) a torzító függvény és (c) a kimenet.



3. ábra: Egyszerű példa a waveshaping szintézisre [6]

2.4 FM szintézis

2.4.1 A frekvencia- és fázismoduláció

A frekvenciamodulációt (FM) elsősorban a rádió elektromágneses jeleinek átvitelére használják. A rádióállomáson rögzített hangokat az FM eljárás segítségével kódolják. A felhasználó az otthoni készülékén a megfelelő paraméterek beállításával dekódolja az adást, és képes váltani a csatornák között [9].

A frekvenciamoduláció két jel segítségével állítható elő, a vivőjellel (x_c) és a moduláló jellel (x_m). A végeredmény az alábbi képlet szerint írható le:

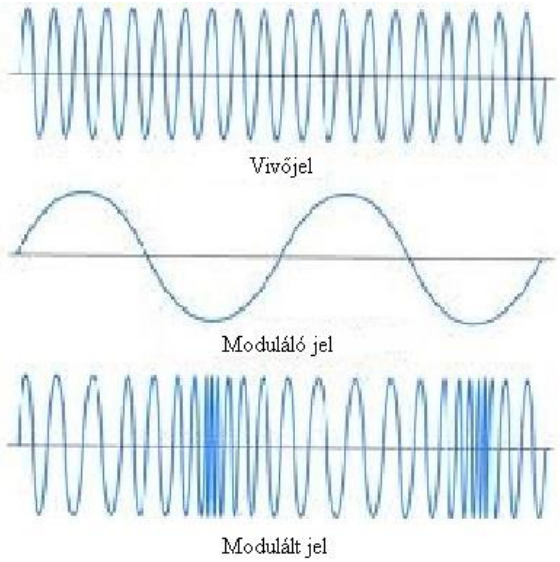
$$y(t) = I_c cos\left(2\pi f_c t + \int_0^t x_m(\tau) d\tau\right)$$

$$= I_c cos\left(2\pi f_c t + \frac{I_m}{2\pi f_m} \cos(2\pi f_c t)\right) \quad (4)$$

ahol a vivőjel $I_c cos(2\pi f_c t)$, a moduláló jel $I_m \cos(2\pi f_c t)$. Látható, hogy a pillanatnyi frekvencia a moduláló jel integráljának pillanatnyi értékétől függően változik. A frekvencia moduláció egy fontos jellemzője a modulációs index. Megmutatja, hogy mekkora a modulált jelnek a vivőjel frekvenciájához képest a maximális eltérése.

A fázis moduláció hasonló elven működik, de a moduláló jelet nem integráljuk, képlete:

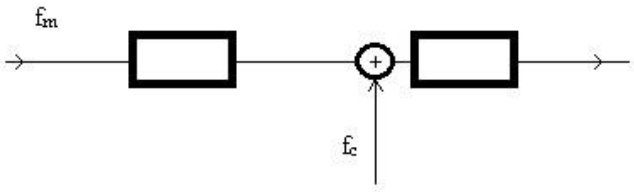
$$y(t) = Icos(2\pi f_c t + x_m(t)) \quad (5)$$



4. ábra: Példa egy frekvencia modulációra [7]

2.4.2 FM hangszintézis

A frekvenciamodulációnak a hangszintézisre történő alkalmazhatóságát csak a 1960-as évek végén kezdték el tanulmányozni. John Chowning, a Stanford Egyetem professzora megfigyelte, hogy két oszcillátor segítségével egy harmonikusokban gazdag spektrum állítható elő. 1967-ben megalkotta az FM hangszintézist [1]. A digitális megfelelőjét 1975-ben szabadalmaztatta, majd a japán Yamaha felfedezte a lehetőséget az eljárásban, és Chowninggal együttműködve létrehozták az első FM hangszintézisen alapuló szintetizátort, a DX7-et. A nevével ellentétben, a Yamaha által alkalmazott FM hangszintézis fázismodulációval állítja elő a hangokat. A frekvencia moduláció alkalmazásakor a moduláló jel egyenkomponensét integrálva egy vivőjel frekvencia viselkedése megváltozik, a fázismoduláció stabilabb jelalakot eredményez.



5. ábra: A frekvencia moduláció blokkvázlata

Amennyiben a modulálójel frekvenciáját 20 Hz alatti értékre állítjuk, ami a hallható hang tartományán kívül esik, a zenei vibrato hatás érhető el. A frekvencia változásának gyorsaságát a modulálófrekvencia, a maximális frekvenciaváltozást a modulálójel amplitúdója határozza meg.

Amikor a modulálójel frekvenciája már a hallható hang frekvenciatartományába állítjuk, érdekes jelenséget tapasztalunk. A vivőjel frekvenciája körül megjelenő további frekvenciakomponenseket felharmonikusokként érzékeljük. Az így kialakult komponensek frekvenciáját az alábbi képlettel lehet felírni:

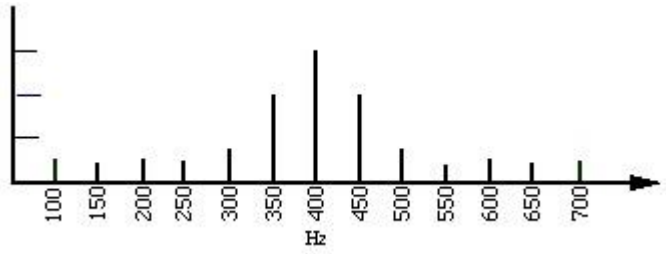
$$f_c \pm nf_m \tag{6}$$

f_c: vivőfrekvencia

n: 1,2,3,…

f_m: moduláló frekvencia

Egy 400Hz-es vivőfrekvenciájú és 50 Hz-es modulálófrekvenciájú jel spektruma:



6. ábra: Fázis moduláció spektrum [10]

Amikor a modulálófrekvencia egész számú többszöröse nagyobb a vivőfrekvenciánál, a fenti képlet alapján negatív komponenseket is kapunk. Ezeket a komponenseket abszolút értékükkel kell figyelembe venni, csak a fázisuk lesz különböző a pozitív oldalsávhoz képest, pontosan 180 fokkal eltolva. Lehetséges, hogy az oldalsávok a 20 Hz-20000 Hz-es határon kívül esnek, így azok nem lesznek hallhatóak. Az FM hangszintézissel képesek vagyunk harmonikus és nem harmonikus spektrumot előállítani. Amennyiben a vivőfrekvencia és a moduláló frekvencia aránya egész szám, harmonikus spektrumot állíthatunk elő, ellenkező esetben pedig nem lesz harmonikus a spektrum.

3 OPL3

Munkám során az OPL3 chip szoftveres modellezését tűztem ki célul, így ebben a fejezetben az OPL3 főbb tulajdonságait tárgyalom.

3.1 Bevezetés

A Yamaha YMF262, más néven az OPL3 egy FM szintézisen alapuló hangkártya chip, az OPL2 továbbfejlesztett verziója. A chipet a Soundblaster 16, Soundblaster Pro 2.0 és az AWE hangkártyáiban használták. Ezen hangkártyáik igen közkedveltek voltak. A korábbi kártyákkal ellentétben az új hardverek már 16 bites adatokkal dolgoztak, minőségük kiválónak számított.

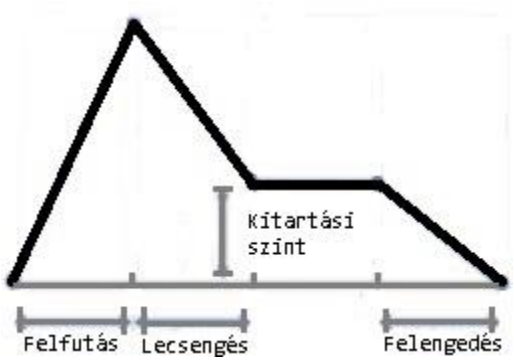
Az OPL3-nak négy beépített oszcillátora van. Minden egyes oszcillátor nyolc hullámformát képes generálni. A négy oszcillátornak hat összeköttetési módja van, a hang generálásához kettő kapcsolás két oszcillátort, a további négy pedig négy oszcillátort használ. Az összeköttetések alapja az FM hangszintézis és a kimeneti jelalakok összeadása. Az oszcillátorok kimenetei egy burkológörbével vezérelt erősítőre vannak kötve [11].

3.2 Burkológörbe-generátor

A burkológörbe-generátor a hang paramétereinek időbeli változását képes befolyásolni. A paraméter lehet a hang amplitúdója, frekvenciája és az alkalmazott hanghatás is. A legelterjedtebb burkológörbe az ADSR görbe, ami négy paraméter segítségével állítja a hang tulajdonságait időbeli lefutását [12]. A paraméterek [13]:

- Felfutási idő (Attack): A hang adott paraméterének nulláról a teljes érték eléréséig tartó idő.
- Lecsengési idő (Decay): A maximális értéktől a kitartási szint értékéig eltelt időt adja meg.
- Kitartási szint (Sustain): A hang lejátszásakor kívánt paraméterszint.
- Felengedési idő (Release): A hang lejátszása után a nulla szint elérésének ideje.

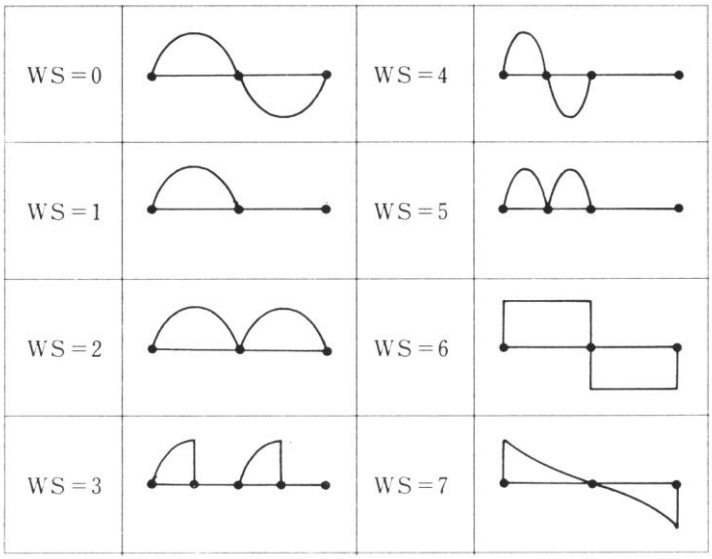
Megjegyezzük, hogy az OPL3 chip esetében a burkológörbe-generátor csak a hang hangerejének időbeli lefutását befolyásolja.



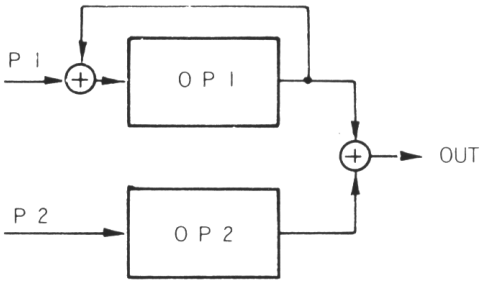
7. ábra: Burkológörbe [12]

3.3 Kapcsolások

Az OPL3 oszcillátorai több érdekes hangzást képesek elérni a különböző kapcsolási módok alkalmazásával. Ebben az alfejezetben a kapcsolások magyarázata szerepel, a kimenet *y(t)*, a négy oszcillátor pedig rendre *a(t)*, *b(t)*, *c(t) és d(t)*. Az oszcillátor bemenete a szintetizátor által kapott frekvencia. A kimenet képletének felírásakor itt szinuszos hangokat feltételezünk, viszont az OPL3 további 7 hullámforma generálására is képes, azokat a MATLAB szimuláció során mutatom be. A képleteket az egyszerűség kedvéért szinuszos oszcillátor-jelekre adom meg. A leírt képletekben I_x az x oszcillátor amplitúdója, f_x az x oszcillátor frekvenciája.



8. ábra: OPL3 hullámformái [14]



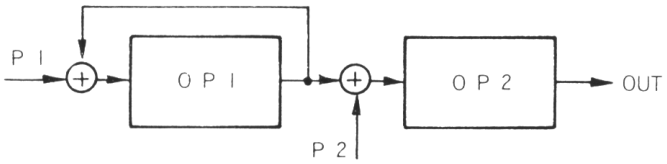
9. ábra: Első kapcsolás blokkvázlata [14]

Az első kapcsolás egy szimpla hangösszegzés, két oszcillátor kimenetének összeadása.

$$y(t) = a(t) + b(t) = I_a \sin(2\pi f_a t) + I_b \sin(2\pi f_b t) \qquad (7)$$

Azonos frekvencia esetén:

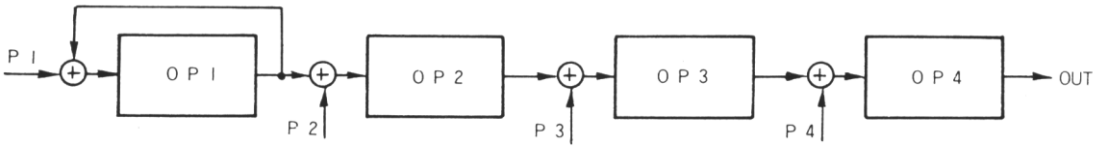
$$y(t) = (I_a + I_b) \sin(2\pi f t) \qquad (8)$$



10. ábra: Második kapcsolás blokkvázlata [14]

A második kapcsolás FM szintézist valósít meg, az első oszcillátor kimenete hozzáadódik a második oszcillátor bemenetéhez.

$$y(t) = b[t + a(t)] = I_b sin[2\pi f_b t + I_a sin(2\pi f_a t)] \qquad (9)$$

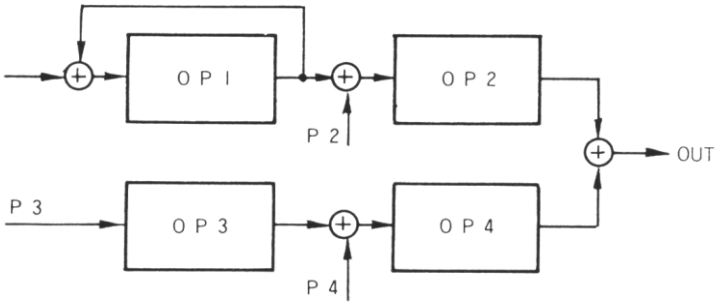


11. ábra: Harmadik kapcsolás blokkvázlata [14]

A harmadik kapcsolás már négy oszcillátort használ, a kimenet előállításához többszörös modulációt végez el a hardver. Az első oszcillátor kimenete hozzáadódik a második oszcillátor bemenetéhez, a második oszcillátor kimenete hozzáadódik a harmadik oszcillátor bemenetéhez és a harmadik oszcillátor kimenete hozzáadódik a negyedik oszcillátor bemenetéhez.

$$y(t) = d\langle t + c\{t + b[t + a(t)]\}\rangle$$

$$= I_d sin\langle 2\pi f_d t + I_c sin\{2\pi f_c t + I_b sin[2\pi f_b t + I_a sin(2\pi f_a t)]\}\rangle \qquad (10)$$

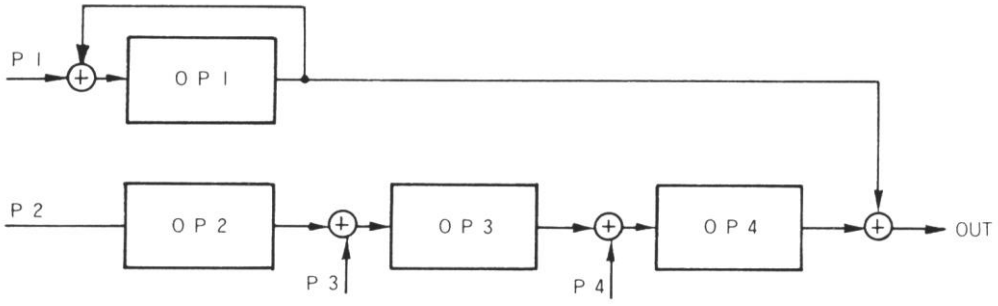


12. ábra: Negyedik kapcsolás blokkvázlata [14]

A negyedik kapcsolás két modulált hang összegzését eredményezi. Az első oszcillátor kimenete hozzáadódik a második oszcillátor bemenetéhez, a harmadik kimenet pedig a negyedik bemenethez, utána a második és a negyedik oszcillátor kimenetét összeadjuk.

$$y(t) = b[t + a(t)] + d[t + c(t)]$$

$$= I_b sin[2\pi f_b t + I_a sin(2\pi f_a t)] + I_d sin[2\pi f_d t + I_c sin(2\pi f_c t)] \qquad (11)$$

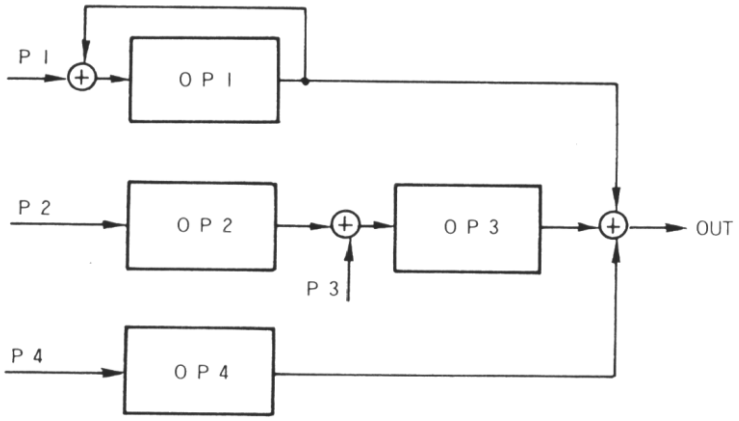


13. ábra: Ötödik kapcsolás blokkvázlata [14]

Az ötödik kapcsolás három oszcillátor kétszeres modulációja és egy negyedik oszcillátor összekapcsolásával állítja elő a megfelelő hangzást. Az előzőekhez hasonlóan a modulációnál két oszcillátor kimenete megváltoztatja a többi oszcillátor bemenetét.

$$y(t) = a(t) + d\{t + c[t + b(t)]\}$$

$$= I_a sin(2\pi f_a t) + I_d sin\{2\pi f_d t + I_c sin[2\pi f_c t + I_b sin(2\pi f_b t)]\} \quad (12)$$



14. ábra: Hatodik kapcsolás blokkvázlata [14]

A hatodik kapcsolásnál egy moduláció és két oszcillátor kimenetének összegét kapjuk.

$$y(t) = a(t) + c[t + b(t)] + d(t)$$

$$= I_a sin(2\pi f_a t) + I_c sin[2\pi f_c t + I_b sin(2\pi f_b t)] + I_d sin(2\pi f_d t) \quad (13)$$

4 MATLAB szimuláció

Az OPL3-at először nem valós időben, a MATLAB program alkalmazásával szimuláltam. A MATLAB felhasználásával numerikus számításokat, algoritmusok implementálást, függvények, illetve adatok ábrázolását végezhetjük el. A munkám során a jelalakok pontos ábrázolását végeztem el, spektrumait elemeztem és a kapcsolások kimenetét vizsgáltam meg.

4.1 Jelanalízis

A hullámformák spektrumainak előállításával könnyen megérthetjük azok hangzásának okait, feltárhatóak a különböző frekvenciakomponensek és értékeik. A Fourier-transzformáció kiváló matematikai módszer a jelalak spektrumának vizsgálatára. A folytonos függvények esetén az alábbi képlet szerint írható le [15]:

$$X(f) = \int_{-\infty}^{\infty} x(t) * e^{-j2\pi ft} dt \qquad (14)$$

ahol $X(f)$ a függvény Fourier transzformáltja, $x(t)$ a jel időfüggvénye, t az idő és f a frekvencia.

Az előállított hullámformák diszkrét jelalakok, a mintavételes rendszernél az alábbi képletet kell használni:

$$X(s) = T_s \sum_{n=-\infty}^{\infty} x(n) * e^{-j2\pi fn} \qquad (15)$$

ahol $X(s)$ a függvény Fourier transzformáltja, $x(n)$ az $x(t)$ jel n-edik időpontban vett értéke.

Ebben az esetben f relatív frekvencia, amit a mintavételi frekvenciához képest számolunk. Az előállított spektrum a mintavételi frekvencia szerint ismétlődik. A formula szerint végtelen mennyiségű adat lenne szükséges. Periodikus jelre a beépített DFT használható, ami N darab minta alapján számolja ki a spektrumot:

$$X(k) = \sum_{n=0}^{N-1} x(n) * e^{-j\frac{2\pi}{N}kn}, \quad \text{k=0…N-1} \qquad (16)$$

A DFT-t a MATLAB-on belül az fft() függvénnyel lehet számítani. Az véges mintahossz miatt a spektrumban a frekvenciacsúcsok értéke nem egyszerű Dirac-deltákkal írható le, és a csúcsok mellett nemkívánatos oldalsávok is megjelennek. Ezen

hatások csökkentésére a jelanalízis során Hanning ablakot használtam, azaz a mintákat a 0,5[1-cos($2\pi\frac{n}{N}$)] függvénnyel súlyoztam. A jelanalízishez függvényt írtam, ami egy hullámforma bemeneti időfüggvényét várja, a kimenet egy decibel skálán ábrázolt spektrum.

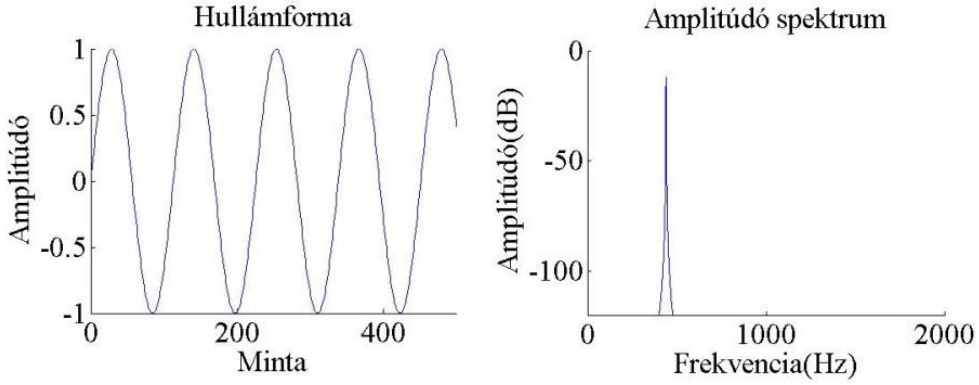
A jelalakokat az *A* hangmagasságának megfelelően generáltam, ennek a frekvenciája 440 Hz. Minden egyes jelalak kimenetét és spektrumát ábrázoltam és elemeztem.

4.2 Jelalakok megvalósítása

Az OPL3 nyolc féle jelalak generálására képes. Egy függvény használatával egy oszcillátort valósítottam meg. A függvény bemenetei a kezdőfázis, a frekvencia, az amplitúdó és a jelalak típusa. A függvény kimenete az adott fázis alapján generált minta értéke és kimeneti fázis.

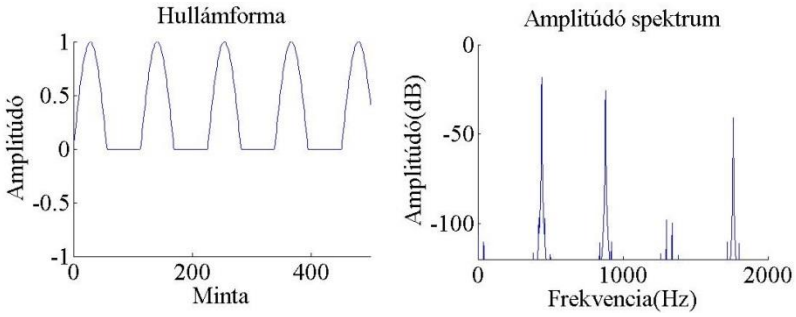
```
function [Y, fiki] = op1(fibe,f,I,waveform)
```

A függvényen belül felvettem egy FI változót, mely a mintavételi frekvencia alapján egy időegység elteltével új fázist generál a bemeneti fázisból. A kimeneti minta értékét ebből a változóból számolom ki. A jelalakok típusa szerinti értékadást switch-case szerkezettel oldom meg. Minden egyes hullámforma generálásakor ellenőrzöm, hogy a fázis meghaladja-e a 2π értéket, és ennek megfelelően adok új értéket neki. A kimeneti fázis pedig az ellenőrzés utáni új fázis lesz.



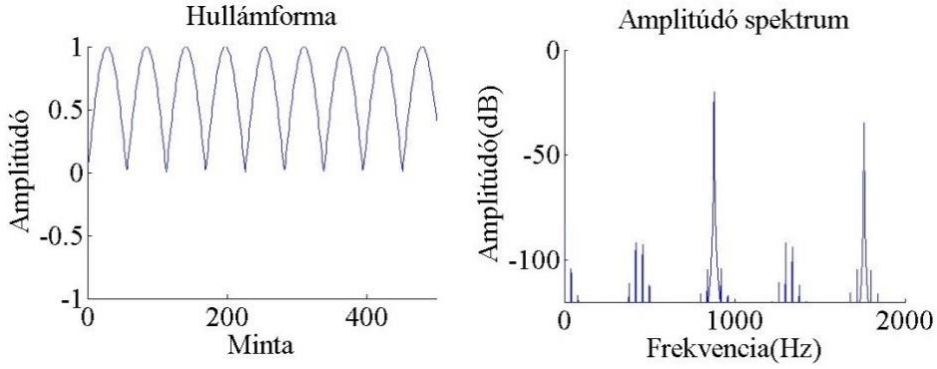
15. ábra: első jelalak és spektruma

Az első jelalak a szinuszjel. A MATLAB beépített szinuszgeneráló függvényével és a belső fázis változóval előállítom a mintát. A spektrumán csak egy csúcs található az elvártak megfelelően.



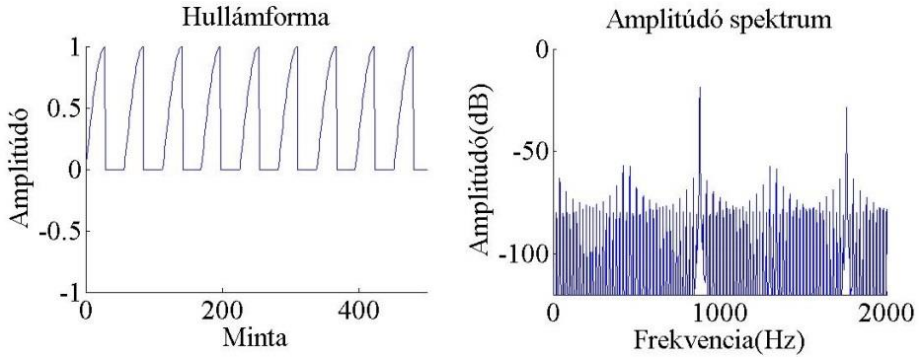
16. ábra: második jelalak és spektruma

A második jelalak a szinuszjel módosításával elérhető hullámforma. Amennyiben a fázis a [0;π] tartományon belül mozog, úgy a függvény a szinusz értékeit veszi fel, viszont a]π;2π[tartományban az értéke nullával lesz egyenlő. A]π;2π[tartományban a szinuszfüggvény negatív értéket vesz fel, így csak a kimenetet kellett vizsgálnom, hogy a nulla értéknél kisebb-e. A spektrumábrán több frekvenciakomponens is található, a csúcsok az alapfrekvenciánál és annak többszörösénél találhatóak. A megjelenő kisebb csúcsok a spektrum átlapolódásából erednek.



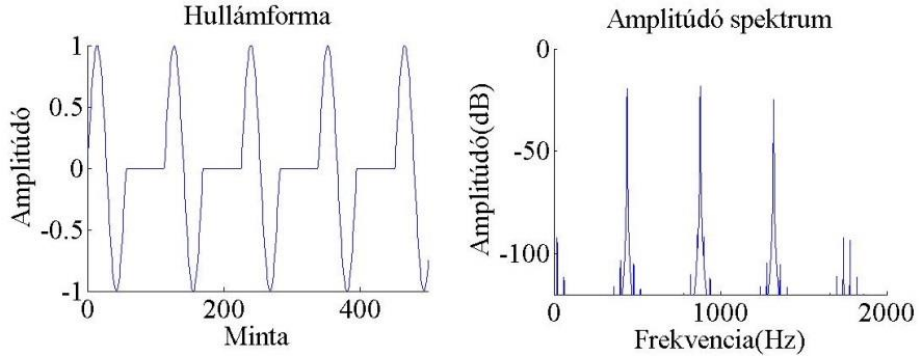
17. ábra: harmadik jelalak és spektruma

A harmadik jelalak a szinuszfüggvény abszolút értékének hullámformája. A mintát a MATLAB beépített abszolútérték-függvényével generáltam. A jelalak az alapfrekvencia kétszeresénél veszi fel az első, egyenkomponenstől eltérő csúcsát. A megadott frekvenciához képest a függvény π szerint periodikus.



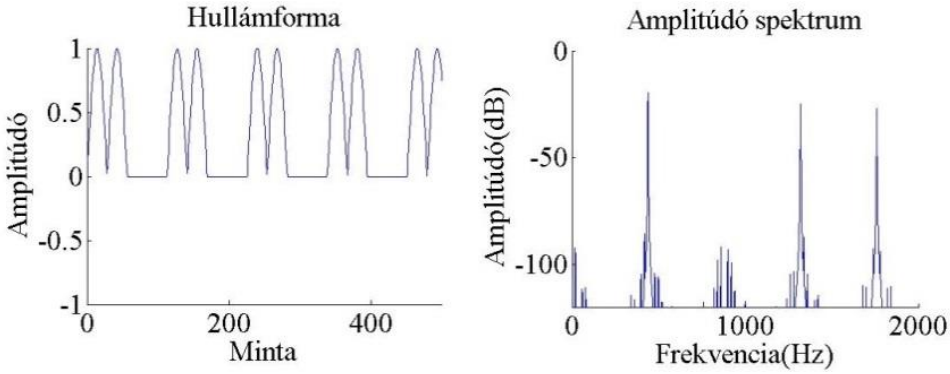
18. ábra: negyedik jelalak és spektruma

A negyedik jelalak is a szinuszfüggvény módosításával érhető el. Kinézetre a hullámforma egy abszolútérték-szinuszfüggvénynek feleltethető meg, amelynek a $[0;\frac{\pi}{2}]$ és $[\pi; 3\frac{\pi}{2}]$ intervallumban értéke nulla. Megoldásként a 2π-t ellenőrző függvényemet módosítottam, a fázis maximálisan a π értéket veheti fel, a $[\frac{\pi}{2};\pi]$ tartományban értéke nulla. A spektrumábrán számtalan frekvenciakomponenst lehet találni, a nagy értékű csúcsok az előző jelalakhoz hasonlóan az alapfrekvencia kétszeresénél találhatóak, az előző jelalakhoz hasonlóan. Mivel a jel spektruma különösen gazdag felharmonikusokban, az átlapolódott komponensek nagyon jelentősek (lásd a -50 dB alatti csúcsokat).



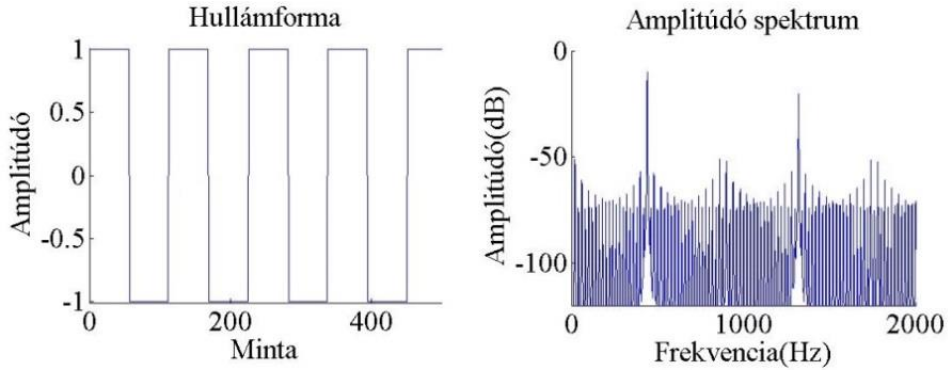
19. ábra: ötödik jelalak és spektruma

Az ötödik jelalak kétszeres frekvenciájú szinusznak feleltethető meg, aminek minden második periódusa konstans nulla. A minta generálásánál a fázis értékének dupláját vettem a [0;π] határokon belül, ettől különböző fázis esetén a kimenet nulla.



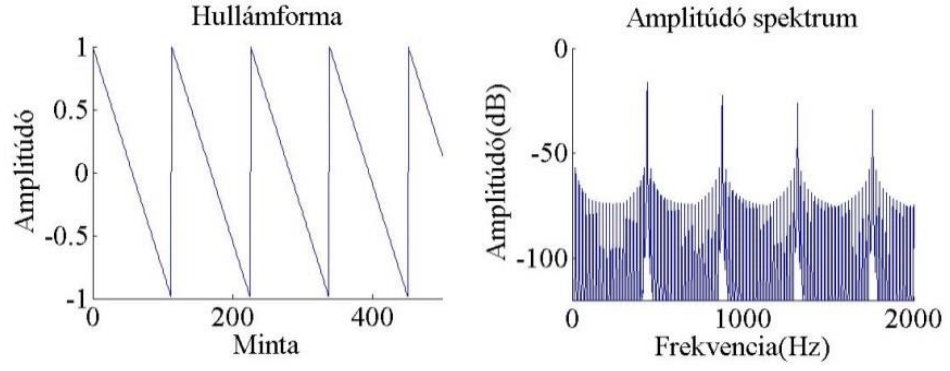
20. ábra: hatodik jelalak és spektruma

A hatodik jelalak az ötödik hullámforma módosításával érhető el. A függvény 0 és π közötti tartományban a szinusz abszolút értékét veszi fel. A hullámforma ugyanúgy az alapfrekvenciánál és többszörösénél veszi fel az értékeket, a komponensek szintje a frekvencia növekedésével csökken.



21. ábra: hetedik jelalak és spektruma

A hetedik jelalak egy egyszerű négyszögjel. A kimenet értéke [0;π] határokon belül a bemeneti amplitúdó,]π;2π[intervallumban a bemeneti amplitúdó negatív értékét. A négyszögjel spektrumábráján csak páratlan felharmonikusnak kellene megjelenniük, de az átlapolódás miatt más komponensek is láthatóak.



22. ábra: nyolcadik jelalak és spektruma

A nyolcadik jelalakot egy egyszerű fűrészjelként generáltam. A nulla fázis esetén a kimenet értéke 1, 2π-hez -1-et. A függvény meredeksége 1/π. A spektrumábrán látható, hogy a függvény a frekvenciakomponenseit az alapfrekvenciánál és annak többszöröseinél veszi fel, illetve itt is megjelennek az átlapolódott komponensek.

4.3 Kapcsolások megvalósítása és kimeneteik vizsgálata

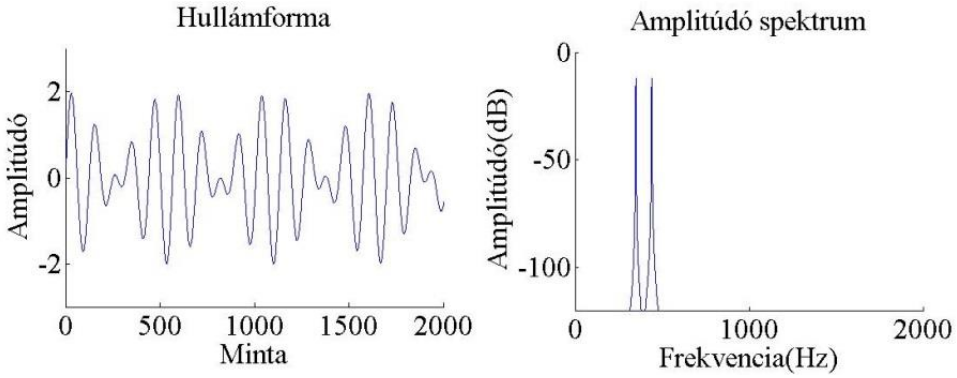
A MATLAB szimuláció során minden egyes kapcsoláshoz külön-külön függvényt rendeltem. A függvények argumentuma az oszcillátorok paramétereit tároló vektor. A függvény az oszcillátorok három paraméterével számol, ami az amplitúdó, a hang MIDI kódja és a jelalak formája. A kapcsolás módjától függően a bemenet kettő vagy négy oszcillátor paramétereiből áll.

Az OPL3-as hangchip nem frekvenciát alkalmaz a számoláshoz, ehelyett MIDI kódot használ. A MIDI hangoknak 128 különböző hangmagassága van. A kívánt frekvenciát a MIDI kód bemenet alapján képlet segítségével számolja ki, ami az *A* hanghoz viszonyítva adja meg a kimenetet. Az *A* hang frekvenciája 440 Hz.

$$f = \left(\frac{440}{32}\right) * 2^{\left(\frac{n-9}{12}\right)} \tag{17}$$

A képletben *f* a kimeneti frekvencia, *n* a MIDI kód száma.

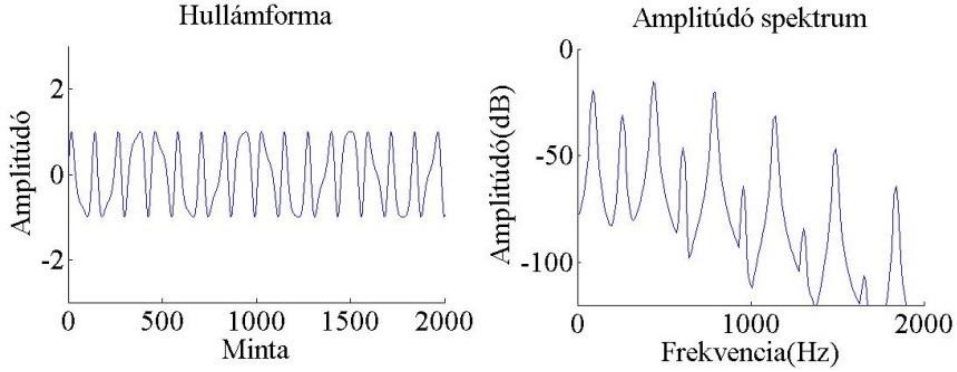
Minden kapcsolást megvalósító függvény először feldolgozza a bemeneti adatokat. A vektorok adatait külön változókba menti. A MIDI kód bemenetet a (17) képlet segítségével frekvenciává alakítja át. A teljes jelalak generálását egy *for* ciklussal oldottam meg, minden egyes ciklusban egy mintát számol ki, és menti egy vektor elemeként. A *for* ciklus megkezdése előtt a függvény egy-egy mintát generál. A szimulációban az oszcillátorok amplitúdója 1, a jelalakok szinuszosak. Az ábrázoláshoz használt MIDI kódok: 65, 68, 66, 69. A frekvenciájuk: 349.2 Hz, 415.3 Hz, 369.9 Hz és 440 Hz. Az szemléltetés kedvéért a grafikonokat megfelelően skáláztam, az ábrázolt minták számát pedig megnöveltem az előző ábrákhoz képest.



23. ábra: első kapcsolás kimenetének időfüggvénye és spektruma

```
function AM = AM(ope1, ope2 )
```

Az első kapcsolás egy egyszerű összeadással megoldható. A függvény a *for* cikluson belül mindkettő oszcillátor mintáit létrehozza, a minták elemeit összeadja, és a kimeneti vektorban tárolja. A spektrumábrán látható a két jel frekvenciakomponense.



24. ábra: második kapcsolás kimenetének időfüggvénye és spektruma

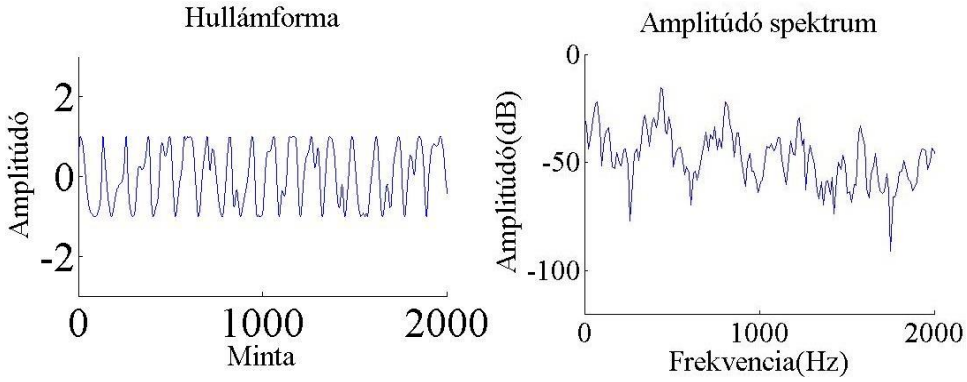
```
function [FMY] = FM(ope1, ope2 )
```

A második kapcsolás alapja a fázis moduláció. A függvény a *for* cikluson belül az *ope1* oszcillátor mintáját létrehozza. Az *ope2* bemeneti fázisa az alábbi kód szerint írható fel:

```
(Y(i)-Y(i-1)+FMfi).
```

Y(i) az *ope1* `i`-edik lefutás utáni kimeneti mintája, az *FMfi* az *ope2* kimeneti fázisa. Az *FMfi* kimeneti fázisként tartalmazza az *ope2* modulálatlan fázisát és az *ope1* kimeneti értékét. Amennyiben a *for* ciklus `i`-edik lefutáskor ezt fel szeretnénk használni bemeneti fázisként, *FMfi* változóból ki kell vonni az *ope1* előző kimeneti értékét, hogy a megfelelő modulációt kapjuk, különben az előző lefutások kimeneti értékeit folyamatosan hozzáadnánk a bemeneti fázishoz. A kimeneti hullámformán jól látszódik

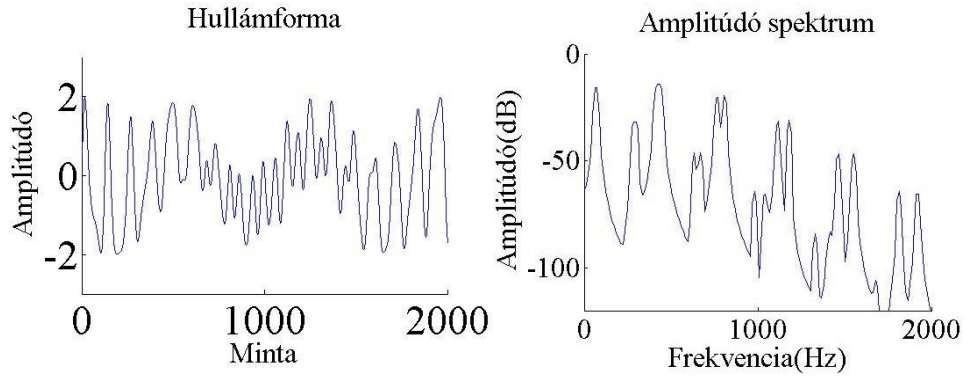
a moduláció. A spektrumábrán láthatóak az alapfrekvenciához képest a moduláló jel frekvenciájával eltolt komponensek.



25. ábra: a harmadik kapcsolás kimenetének időfüggvénye és spektruma

```
function [FMFM] = FMFM(ope1, ope2, ope3, ope4 )
```

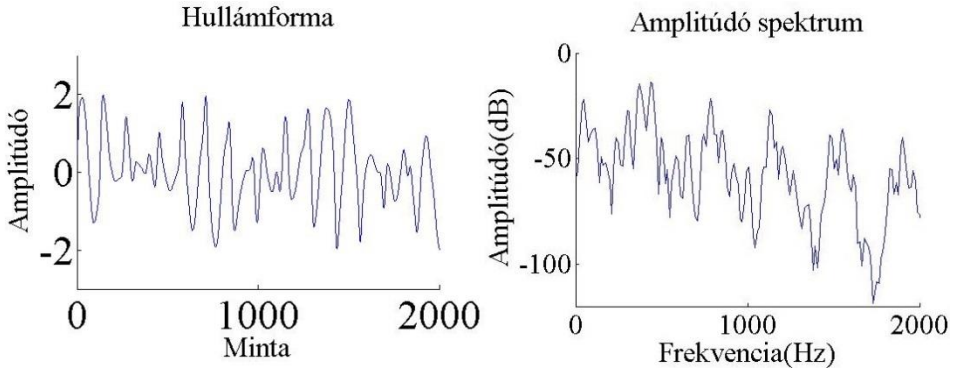
A harmadik kapcsolás már négy oszcillátorból áll. A függvény bemenetei a négy oszcillátor paramétereit tartalmazó vektorok. A *for* ciklus egy lefutásán sorrendben minden oszcillátor egy-egy mintáját létrehozza. Az *ope1*, *ope2*, *ope3* oszcillátor kimeneti mintája az *ope2*, *ope3*, *ope4* bemeneti fázisában szerepel, és a második kapcsoláshoz hasonlóan az előző lefutás értékét kivonjuk. A fázisok folyamatos változását láthatjuk a hullámformán, a jelalak a többszörös moduláció miatt a szinuszfüggvénytől jelentősen eltér. A spektrumábrán az oldalsávok száma jelentősen megnőtt az előző kapcsoláshoz képest.



26. ábra: a negyedik kapcsolás kimenetének időfüggvénye és spektruma

```
function [FMAM] = FMAM(ope1, ope2, ope3, ope4 )
```

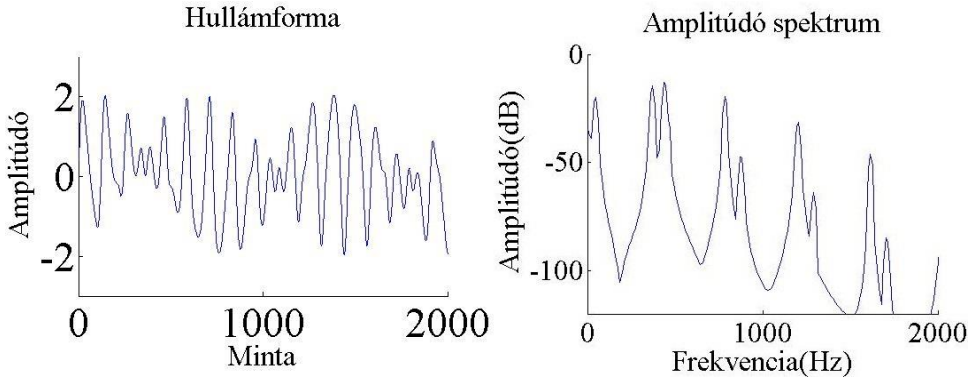
A negyedik kapcsolás két fázismodulált jel összeadását eredményezi. A *for* cikluson belül már öt vektort vettem fel, az első négy az oszcillátorok kimenete, az ötödik az *ope2* és *ope4* oszcillátor összege. Az fázismoduláció az *ope1*, *ope2* és az *ope3*, *ope4* oszcillátorokból jön létre. A spektrumábrán láthatjuk az *ope2*, *ope4* frekvenciakomponenseit és a fázismoduláció miatt kialakult oldalsávokat.



27. ábra: az ötödik kapcsolás kimenete és spektruma

```
function [AMFM] = AMFM(ope1, ope2, ope3, ope4 )
```

Az ötödik kapcsolás kimenete az *ope2*, *ope3*, *ope4* segítségével előállított fázismoduláció és az *ope1* összege. Az *ope2* kimeneti mintája az *ope3*, az *ope3* kimeneti mintája pedig az *ope4* bemeneti fázisában szerepel. A függvény a *for* cikluson belül az előzőhöz hasonlóan öt vektort generál, az ötödik vektor a fázismodulált jel és az *ope1* összege. A hullámforma ábráján a jelalak fázisváltozása látható. A spektrumábrán az átlapolódás miatti frekvenciakomponensek mellett a két jel frekvenciakomponensei és az oldalsávok láthatóak.



28. ábra: a hatodik kapcsolás kimenete és spektruma

```
function [AMAM] = AMAM(ope1, ope2, ope3, ope4 )
```

A hatodik kapcsolás kimenete egy fázismodulált jel és két egyszerű jel összege. A függvény a *for* cikluson belül öt vektort hoz létre. Az *ope3* bemeneti fázisában az *ope2* kimeneti mintája szerepel. A függvény kimenete az *ope1*, *ope3* és *ope4* segítségével generált minták összege. A hullámformaábráján a három jel összeadása miatt a fázismoduláció nehezen észrevehető. A spektrumábrán a frekvenciakomponensek jól elkülönülnek.

5 Felhasznált programok

5.1 VST

A Virtual Studio Technology egy szoftver felület, mely képes a program specifikációjától függően különféle hangszerek, hangzások szimulálására és a hangokat befolyásoló effektek implementálására [16]. A VST programokat általában C++ nyelven írják. Az így megírt programokat, más néven pluginokat hangszerkesztő programokon belül használhatjuk. A VST-t a Steinberg cég alkotta meg 1996-ban. Az első pluginok csak hangzásokat változtató effekteket voltak képesek megvalósítani. Ilyen programok voltak az Especial, ami egy zengetőt tartalmazott, a Choirus, egy chorus hanghatást létrehozó plugin, az Stereo Echo és az Autopanner. Az első hangstúdió szoftver, ami a VST formátumot támogatta, a CuBase 2.0 volt. 1999-ben a pluginok már MIDI adatokkal is dolgoztak, létrejöttek az első szoftver alapú hangszerek is. A legelterjedtebb hangstúdió programok, munkaállomások az Ableton Live, a Cubase, az FL studio és a Logic.

A munkaállomások egyszerű kezelőfelületet nyújtanak a zeneszerzők számára. Grafikus felületen keresztül különböző MIDI hangsávokat lehet programozni. Minden egyes hangsávra egy előre felvett hangot vagy egy VST hangszert lehet illeszteni. Az ilyen hangszerek felismerik a hangstúdió küldött adatait, így a megfelelő beállítások mellet a hangsáv szerinti hangmagasságon a kívánt hangot generálja a program. Az ilyen munkaállomások hardverigénye meglehetősen nagy, korszerű zene komponálásához nagyméretű memória és gyors processzor szükséges.

A VST programok három kategóriába sorolhatóak. A VST hangszer programok általában szintetizátorok és samplerek. A szoftver az analóg szintetizátorokat és hangszereket, például a régi DX7-es szintetizátorát, vagy akusztikus gitárok és zongorák hangját szimulálja. A VST effekteket használó pluginek bejövő adatokat dolgoznak fel. A beérkező hangokat a kiválasztott hanghatásokkal befolyásolja. Ilyen hangeffektek a zengetők, szűrők és késleltetők. Egy grafikus felület segítségével könnyen állíthatóak a hang paraméterei, például potméterek és kapcsolók segítségével. A harmadik típus a VST Midi effektusokat létrehozó programok csoportja. A bemenő MIDI adatokat feldolgozzák és megváltoztatják a felhasználó beállításainak megfelelően.

5.2 WDL, WDL-OL

A WDL a Cockos Incorporated cég ingyenes, nyílt forráskódú és általános felhasználásra alkalmas C++ könyvtára. A könyvtár a beépített osztályai és függvényei segítségével megkönnyíti bonyolult programunk írását [17].

A WDL nagy segítséget nyújt az alapműveletek megírásához. A programozás során gyakran használt memória, tömbök és listák törlése és allokációja könnyen megoldható. A fájlműveletek függvényei lehetőséget nyújtanak aszinkron olvasásra és írásra. Egyik jelentős probléma a nagyméretű fájlok felhasználása, a műveletek lassúak, a WDL beépített függvénye viszont optimalizálja a program sebességét. Adatműveletek gyors alkalmazását is támogatja a könyvtár. Emellett a véletlenszám generátor, a Fourier transzformáció és a tömb elemeinek rendezésére használt Merge sorting is megtalálható a függvények között.

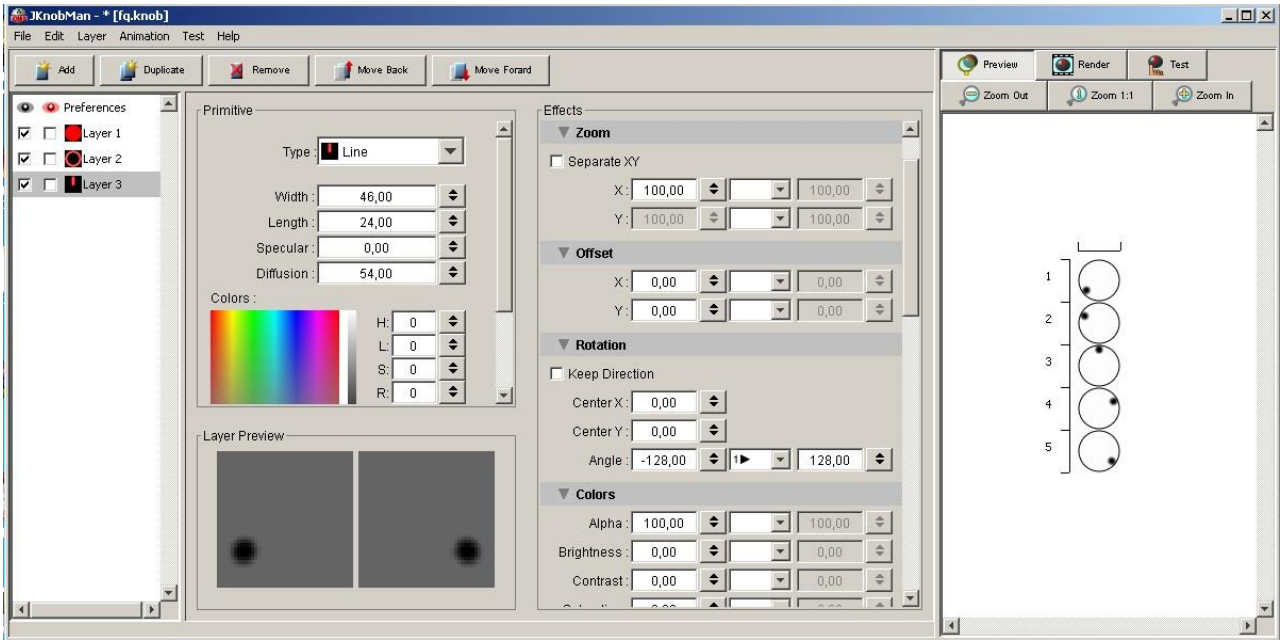
A WDL az alapműveletek mellett több specifikus felhasználásra alkalmas függvényeket és osztályokat is tartalmaz. A hangot feldolgozó egysége tartalmaz egy valós idejű konvolúciót számoló függvényt, egy gyors szinuszfüggvény generátort és egyszerű hanghatásokat alkalmazó osztályokat. Lehetőségünk van képfájlok feldolgozására is. A könyvtár támogatja a legelterjedtebb képfájl kiterjesztéseket, és rajzoló függvényeivel egyszerű formákat generálhatunk. Ezek mellett a WDL magába foglalja a 3D modellezés alapvető műveleteit, az alkalmazások menürendszerét létrehozó függvényeket és a hálózaton keresztül való kommunikálás eszközeit.

A WDL könyvtárnak a régebbi verzióiban az IPlug is szerepelt. John Schwartz 2007-ben hozta létre az első verziót. Újabban a WDL-OL és a WDL-Tale könyvtárakban található. Az IPlug egy audió feldolgozásra használt grafikus fejlesztői környezet. A Windows és OSx operációs rendszereket egyaránt támogatja. A képeket az eredeti WDL LICE könyvtár függvényeivel jeleníti meg. Az IPlug a programokat egyszerre öt protokoll szerint valósítja meg: aax, app, rtas, vst2 és vst3. A programomat VST2 környezetben írtam.

5.3 KnobMan

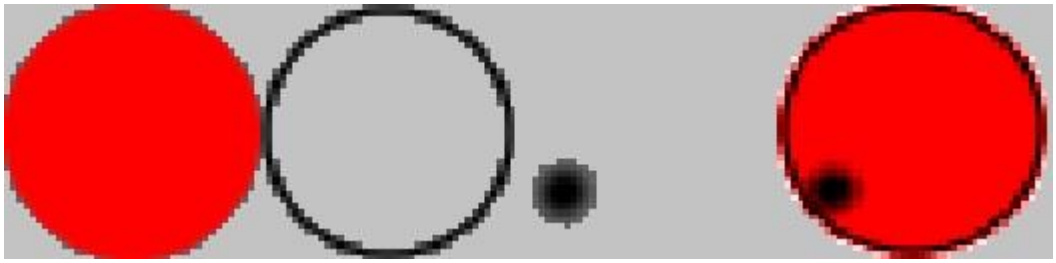
A zeneszerkesztő programok számos szabályzóval működnek, a legelterjedtebb a potenciométer. A KnobMan kiváló lehetőséget nyújt a potméter képeinek megvalósításában. A C++ alkalmazáson belül a kép nem forog, a potenciométer változtatásakor a program a potméter állásoknak megfelelő képkockát jeleníti meg. Ezeket a különböző potméterállásnak megfelelő képeket generálja le a KnobMan program.

Az opciók közül csak a legfontosabbakat nézzük át.



29. ábra: KnobMan grafikus felülete

A program alapvetően három rétegből álló potméter ábráját hozza létre. Minden egyes réteg számos beállítással rendelkezik. A rétegek mind kör alakúak, a Primitive fülön belül kiválaszthatjuk képet, és a négy opció segítségével formázhatjuk azt. A rétegeket a potméterek árnyékolása miatt használjuk.



30. ábra: A három réteg, és a végső kép

Az effects fülön belül további opciót találhatunk. A Zoom opcióban a réteg nagyítását az x és y koordináta beállításaival lehet változtatni. Az offset segítségével a kör középpontjához képest a beillesztett képet eltolhatjuk két irányba. A Rotation fülben a kép forgásának módját állíthatjuk be. A Center x,y koordinátákkal a kör forgáspontját lehet beállítani, az Angle beállítással a 0°-hoz képest beállíthatjuk a forgás kezdő és végpontját.

A Preferences ablakban a végleges kép adatait lehet megadni. A magasság és szélesség mellett a képkockák számát is megadhatjuk.

5.4 Az Iplug osztályai és a példaprogram

A git segítségével másolatot csináltam a példakódról, elneveztem *example*-nek. Az alapprogram egy kiválasztott mikrofonbemenet hangerejét egy állítható potméterrel változtatja [18]. A programnak három fontos fájlja van a programozása szempontjából, a *resource.h,* az *example.h* és az *example.cpp*. A *resource* fájl különböző azonosítók és forrásfájlok elérését tárolja. A programablak és a megjelenítés paramétereit is be lehet állítani, a képazonosító és a kép elérési útjának deklarálásával a grafikus felületet állíthatjuk be.

Az *example.h* tárolja a program grafikus felhasználói felület paramétereit és függvényeit [19]. A példaprogram public része tartalmazza a konstruktor, destruktor és három funkció függvényeit. A mintavételi frekvencia megváltozásakor a *Reset* függvény hívódik meg, a függvény felülírja az előző mentett értéket. Amennyiben a példaprogramban megváltoztatjuk a potméter állását, a program meghívja az *OnParamChange* függvényt, és az új értékre állítja be a paramétert. A *ProcessDoubleReplacing* a bejövő hangot dolgozza fel.

Az *example.cpp EParams*-ban tárolja a program grafikus felületén változtatható objektumok paramétereit, ezeken keresztül a program a változók értékét állítja át. A példaprogramban csak a hangerőért felelős *kGain* és a *kNumParams* szerepel. Az *enum* lista végén fog szerepelni a *kNumParams*, így a program tudni fogja a paraméterek számát. Az *ELayout enum* a példaprogram grafikus felület paramétereit tárolja, a megjelenő ablak szélességét és magasságát, és a beillesztett potméter x,y koordinátáit és a kép fájl képkockáinak számát.

A konstruktor függvény felelős a grafikai elemek beillesztéséért. Az iPlugon belül az *IGraphics* nevű osztály segítségével lehet a grafikus felületet létrehozni. A programban a *pGraphics* nevű példányosított osztály tartalmazza a felhasználói interfész elemeit. Beépített függvényei felhasználásával lehet a grafikus felületet változtatni. Az *AttachPanelBackground* függvénnyel a program háttérszínét lehet állítani. A *kGain* változó típusát és értékeit a *GetParam* függvény inicializálja. A paraméterek függvényei:

- *InitBool*: Egy *bool* típusú változót hoz létre. A függvény argumentuma a változó neve, az alapértelmezett érték. Kétállású kapcsolóknál használják.
- *InitEnum*: Egy *enum* típusú változót hoz létre. A függvény argumentuma a változó neve, az alapértelmezett érték és a lehetséges értékek száma. Többállású kapcsolókhoz használják fel.
- *InitInt*: Egy *int* típusú változót hoz létre. A függvény argumentuma a változó neve, az alapértelmezett, minimális és maximális érték. Nagy lépésközű változókra használják.
- *InitDouble*: Egy *double* típusú változót hoz létre. A függvény argumentuma a változó neve, az alapértelmezett, minimális és maximális érték, és az értékek közötti lépésköz.

A példaprogram esetében a hangerő egy *double* típusú változó lesz. A változókat lehetőségünk van több grafikai elem segítségével változtatni. A felhasznált képeket egy *IBitmap* osztályban kell tárolni. Az *IGraphics* osztály *LoadIBitmap* függvényével az *IBitmap* osztály előállítható. Az osztály konstruktorának argumentumai a *resource.h*-ban inicializált képazonosító és elérési útja, valamint a hivatkozott kép elemeinek száma. Minden egyes kép több kisebb képből áll elő, a program képkockánként vált a képek között. A paramétereket irányító grafikai elemek ősosztálya az *IControl*. Az elemeknek több típusa létezik, ezek a következők:

- Az *IBitmapControl* osztály egy beszúrt kép változtatását engedélyezi. Amennyiben a képre kattintunk, az osztály a soron következő képet jeleníti meg, és az annak megfelelő funkciót alkalmazza. Leggyakrabban a program hullámformáinak változtatására használják. A konstruktor függvény argumentumai: *IPlugBase* osztály, amiben létrehozzuk a

grafikai elemet, kép x,y koordinátája, az *EParams* listából kiválasztott paraméter és egy *IBitmap* osztály.

- Az *IFaderControl* egy csúszkát hoz létre a grafikai felületen. Leggyakrabban a pan hanghatás változtatására használják, ahol a sztereo hangzást a jobb és bal oldali csatorna egymáshoz képesti hangerejével lehet állítani. A konstruktor függvény argumentumai: *IPlugBase* osztály, hivatkozott kép x,y koordinátája, csúszka hossza, *EParams* paraméter, *IBitmap* osztály, a csúszka elmozdulásának iránya és a csúszka elmozdításának módja. Ha a csúszka elmozdításának módja 1, akkor a csúszkát elmozdítani csak rákattintással lehet.
- A fájlműveleteket az *IFileSelectorControl* alkalmazásával lehet elérni. Leggyakrabban a különböző wav és mp3 fájlok beolvasására és mentésére használják. A konstruktor függvény argumentumai: az *IPlugBase* osztály, az *IRECT* struktúra, ami az előugró ablak tulajdonságait tárolja, az *EParams* paraméter, a hivatkozott kép *IBitmap* osztálya, a fájlművelet típusa, az elérési út karakter tömbje és a fájlkiterjesztések típusai.
- Az *IInvisibleSwitchControl* egy kétállású kapcsoló, ami a grafikus felület egy területére való kattintásával változtatható. Nevéből adódóan az adott terület kinézete nem változik meg a kattintás után, a kapcsoló láthatatlan. A konstruktor függvény argumentuma: *IPlugBase* osztály, a terület tulajdonságait tároló *IRECT* struktúra és az *EParams* paraméter.
- A grafikus felületen belső zongorabillentyűzetet is használhatunk az *IKeyboardControl* osztály alkalmazásával. A függvény argumentumai: *IPlugBase* osztály, zongorabillentyűzet bal felső sarkának x,y koordinátája, legkisebb MIDI kód szám, lejátszható oktávok száma, a fehér és fekete billentyűk *IBitmap* struktúrája és a billentyűk koordinátái.
- Lehetőségünk van potméterek beillesztésére is az *IKnobControl* osztállyal. A potmétereket a hangerő és a hanghatások értékeinek állítására használják. A szabályzó *IBitmap* képkockái között vált, és nem

a kép forog. A függvény argumentumai: *IPlugBase* osztály, a szabályzó x,y koordinátája, az *EParams* paraméter és az *IBitmap* struktúra.

- Az *IPanelControl* a grafikus felületen egy kiválasztott területet a kiválasztott színre színezi ki. A függvény argumentumai: *IPlugBase* osztály, a terület tulajdonságait tartalmazó *IRECT* struktúra és a szín.
- Az *IRadioButtonsControl* kétállású kapcsolókat hoz létre. A hanghatások ki-be kapcsolását lehet irányítani. A függvény argumentumai: *IPlugbase* osztály, a kapcsolók területét jellemző IRECT struktúra, EParams paraméter, a kapcsolók száma, *IBitmap* struktúra, a kapcsolók elmozdításának iránya és a reverse változó, ami lehetőséget ad a kapcsolók fordított működésére.
- Az *ITextControl* szöveget képes kiíratni a képernyőre. Leggyakrabban a hanghatások értékeinek kiíratására használják, például a hangerő értékét írják ki vele. A függvény argumentuma: *IPlugbase* osztály, *IRECT* struktúra a szöveg megjelenítéséhez használt területre, egy *IText* struktúra, ami a szöveg tulajdonságait tartalmazza és a kiíratott szöveg.
- Weboldalak megnyitására az *IURLControl* osztályt használja az IPlug. Egy meghatározott területre rákattintva egy weboldal fog megnyílni. Lehetőség van egy tartalék weblap megnyitására is, amennyiben ez a weboldal se jelenik meg, egy általunk megadott hibaüzenetet is kiírathatunk. A függvény argumentuma: *IPlugbase* osztály, kijelölt területet jellemző *IRECT* struktúra, a megjelenítendő honlap címe, a másodlagos honlap címe és a hibaüzenet.

A program a *ProcessDoubleReplacing* függvényben dolgozza fel a bejövő adatokat. A függvény argumentumai a bemeneti és kimeneti amplitúdók tömbjei, és az adatok száma. A bemenő adatokat *for* ciklussal dolgozza fel a függvény, a felhasználó beállításai szerint egyenként feltölti a kimenet tömbjét. A példaprogramban mintánként megszorozzuk a bemenetet az *mGain* értékével, az eredmény a kimenet lesz.

A példaprogramon belül csak egy hangerőszabályzó szerepel, az *OnParamChange* függvény segítségével változtatjuk meg a kimenet értékét. Az *EParams* listában szereplő bármelyik paraméter változásakor a program meghívja a függvényt. Egy *switch-case* kiválasztással a paraméter típusától függően a program

megváltoztatja a változó értékét. A példaprogramban a *kGain* paraméter megváltozásakor a *GetParam->Value* függvény segítségével lekérdezi a változó értékét, és ez alapján a *ProcessDoubleReplacing* függvényben szereplő *mGain* változó értékét átírja.

6 C++ megvalósítás

Ebben a fejezetben a példakódhoz képest felépített programom tulajdonságait mutatom be. Ötleteimet a [18] és [20] forrásokból merítettem. A program fő célja az FM szintézis és az OPL3 megvalósítása. Programozás során három osztályt használtam fel a [18] forrásból, segítségükkel az OPL3-hoz hasonló jelalakvariánsokat, a MIDI adatok feldolgozását és a burkológörbe tulajdonságait könnyen kezelhetővé tettem. Az elkészült plugin egy VST2-es alkalmazás, a fordítóban a projectfájlon belül a vst2 kiterjesztésű alkalmazást kell elindítani. A programozáshoz használt fordítóprogram a Microsoft Visual Studio 2010-es verziója volt.

6.1 A jelalakgenerálás

A jelalakokat egy oszcillátort megvalósító osztály segítségével implementáltam. Minden egyes hang lejátszásakor a fő osztályunk *ProcessDoubleReplacing* függvénye meghívja az oszcillátor osztály függvényeit, a számolt értékeket pedig a kimeneti bal és jobb hangsáv tárolóiba menti.

Az osztály *header* fájlja elején megtalálhatjuk a kimeneti jelalakok típusait. A jelalakok sorrendben a szinuszfüggvény (15. ábra), fűrészjel (22. ábra), négyszögjel (21. ábra), a háromszögjel, és a szinuszfüggvény variánsai (17., 16., 18. és 13. ábra). Az osztályon belül tároljuk az kimeneti jelalakot.

Az oszcillátornak *private* változói és függvényei is vannak, amik biztosítják, hogy a program külső használatakor a felhasználó nem tudja megváltoztatni az értéküket. Ezek a következőek. Konstans értéknek vettem fel a π-t és 2π-t, a nevük *mPI* és *twoPI*. Az oszcillátor frekvenciáját *mFrequency*, fázisát *mPhase,* mintavételi frekvenciáját *mSampleRate* és két minta közötti fázisváltozását *mPhaseIncrement* nevű változók tárolják.. Az *updateIncrement* függvény meghívásakor a fázisváltozás értékét változtatja. Az osztály *.cpp* fájljában a *public* függvények leírása található. A *public* függvények között az osztály konstruktora is megtalálható.

Az *osztály set* függvényei a változók értékét állítják be. Az *updateIncrement* függvény a mintavételi frekvencia változásakor hívódik meg, ilyenkor a fázisváltozás értéke:

$$\Delta\tau = 2\pi \frac{f}{f_s} \qquad (18)$$

ahol $\Delta\tau$ a fázisváltozás, f a frekvencia és f_s a mintavételi frekvencia. A következő függvény a *nextSample*, ami az oszcillátor egy mintájával tér vissza. Amennyiben az oszcillátor némítva van, a visszatérési érték 0. A beállított jelalak függvényében különböző módon állnak elő a kimeneti minta értékei. A kiértékelés után a függvény a fázisváltozás értékét hozzáadja a fázishoz, és ellenőrzi, hogy az érték meghaladja-e a 2π-t. A *math.h* könyvtár *sin* függvényével a szinuszfüggvényt lehet létrehozni. A fűrészjelet egy egyszerű képlettel lehet előállítani, a minta értéke az alábbi képlet szerint írható fel:

$$n = 1 - \frac{2\tau}{2\pi} \qquad (19)$$

ahol n a minta értéke, τ a fázis. A négyszögjel mintája egyszerűen számolható egy *if* feltétellel. Amennyiben a fázis értéke kisebb, mint π, a kimeneti minta értéke 1, a többi fázisra a visszatérési érték -1. A háromszögjel kimenetét is egy feltételhez kötöttem. Amennyiben a fázis nem haladja meg a π értéket, a visszatérési érték (20) képlettel számolható ki.

$$n = -1 + (2\frac{\tau}{\pi}) \qquad (20)$$

Ha a fázis meghaladja a π értéket, a kimenet a (21) szerint írható fel.

$$n = 1 - (2\frac{\tau}{\pi}) \qquad (21)$$

A következő jelalak a 11. ábrán látható. A MATLAB szimulációhoz képest a frekvenciát itt a felére vettem, az alapfrekvencia ilyenkor megegyezik az oszcillátor frekvenciájával. A beépített *sin* függvény a fázis felével számolva adja meg a kimenetet. Az ötödik jelalak megegyezik a MATLAB szimuláció kimeneti jelalakjával (10. ábra). A [0; π[határokon belül a *sin* visszatérési értéke lesz a kimenet, a határon kívül a kimenet 0. A hatodik jelalakot egy összetett feltétel szerint lehet felírni. Amennyiben a fázis [0; $\frac{\pi}{2}$[vagy]$\frac{\pi}{2}$;$3\frac{\pi}{2}$[határokon belül mozog, a kimenet a szinuszfüggvény abszolút értékét veszi fel, a többi esetben mintahívó függvény visszatérési értéke 0. Az utolsó jelalak a 13. ábrán látható. A kimenet a feltétel szerint [0; π[határokon belül egy kétszeres frekvenciájú szinuszfüggvény, nagyobb fázis esetén a 0.

Az FM szintézishez egy *sample* függvényt is írtam. A függvény egy külső fázisértékből számolja ki a *nextSample*-höz hasonlóan a mintát. A bemeneti fázis

meghaladhatja a 2π értéket, így a függvény elején és végén is megfelelően állítom be a számoláshoz szükséges értéket.

Az FM szintézishez szükséges az oszcillátor fázisát lekérdezni, az *mPhase private* változó, ezért a *getPhase* függvény segítségével megkapjuk a kívánt értéket.

6.2 MIDI feldolgozó

Amikor a plugint betöltjük a munkaállomásba, megkapja a rendelkezésre álló eszközök MIDI adatait, amiket a *MidiReceiver* osztály dolgoz fel. A MIDI adatok feldolgozását a *ProcessMidiMsg* függvény végzi el. A *ProcessMidiMsg* egy *IMidiMsg* struktúra átadásával kommunikál a programmal. Az *IMidiMsg* változói között megtalálható a *NoteNumber* és a *Velocity*, ami lejátszott hang hangmagassága és hangereje. A *MidiReceiver* tárolja a MIDI üzeneteket, amíg a főosztály *ProcessDoubleReplacing* függvénye az előző adatokból kimenetet generál. Az osztály működéséhez az *IMidiQueue* osztály is szükséges, ami a beérkező MIDI adatokat egy várakozási sorba rendezi [18].

Az osztályon belül számos változó szükséges a megfelelő működéshez. A *keyCount* a hangmagasságok számát, a *nNumkeys* a lejátszott hangok számát, az *mLastNoteNumber, mLastFrequency* és az *mLastVelocity* az utolsó lejátszott hang hangmagasságát, frekvenciáját és hangerejét tárolja. Az *mOffset* megmutatja, hogy a tárolón belül az adott hang hol található. A *noteNumberToFrequency* kiszámolja a MIDI hang kódjából a frekvenciát. Az *mKeyStatus* tömb megmutatja, hogy a lehetséges bemeneti hangmagasságokból mely MIDI kódú hangokat játssza le.

Az *onMessageReceived* a MIDI üzenetek állapotát olvassa. Amennyiben az üzenet szerint egy hangot lejátszanak, vagy megállítanak, az üzenetet az *IMidiQueue* várakozási sorba továbbítja a függvény.

Az advance függvényt a beérkező adat feldolgozásához használjuk. Amikor a várakozási sor nem üres, a függvény minden kiértékelés előtt feltölti azt MIDI üzenetekkel. Ezek után egy *if* feltétellel megnézi a beérkező üzenet prioritását. A MIDI üzenetek *status* függvényével a beérkező hang adatait kérdezzük le, amit a MIDI feldolgozáshoz használunk.

```
if (status == IMidiMsg::kNoteOn && velocity) {
            if(mKeyStatus[noteNumber] == false) {
                mKeyStatus[noteNumber] = true;
                mNumKeys += 1;
            }
            if (noteNumber != mLastNoteNumber) {
                    mLastNoteNumber = noteNumber;
                mLastFrequency=noteNumberToFrequency(mLastNoteNumber);
                    mLastVelocity = velocity;
                    noteOn(noteNumber, velocity);
                }

else {
         if(mKeyStatus[noteNumber] == true) {
             mKeyStatus[noteNumber] = false;
             mNumKeys -= 1;
         }
         if (noteNumber == mLastNoteNumber) {
                        mLastNoteNumber = -1;
                        noteOff(noteNumber, mLastVelocity);
                  }
     }
```

A fenti kódrészlet bemutatja a feltétel szerinti adatfeldolgozást. Az előbb mentett *status* változó alapján eldöntjük, hogy a beérkezett hangot lejátszzuk-e éppen, és a hangereje nagyobb-e nullánál. Amennyiben a feltétel teljesül, a lejátszott MIDI kódú hang szerint beállítjuk az *mKeyStatus* értékét, megnöveljük a lejátszott hangok számát, és ha a beérkező hang nem az utolsó lejátszott hang volt, akkor változók átírásával azzá tesszük. Amennyiben a feltétel nem teljesül, akkor az adott hangot lejátszottuk már, ennek megfelelően állítjuk át az *mKeyStatus* tömb megfelelő elemét, a lejátszott hangok számát csökkentjük. Az adatok feldolgozása után az üzetenet töröljük az *IMidiQueue* várakozási sorból.

A *get* kezdetű függvények változók értékeit kérdezik le, a sorrend felállításához szükségesek. A *Flush* és *Resize* függvényekkel a várakozási listát törölni és átméretezni lehet.

6.3 Burkológörbe-generátor osztálya

Az *EnvelopeGenerator* osztály egy burkológörbe-generátort valósít meg, a lejátszott hang mintáit egy 0 és 1 közötti értékkel szorozza meg. A minta sorszámából az *EnvelopeGenerator* az állapotok értéket adja vissza, amit a fő osztály *ProcessDoubleReplacing* függvényében felhasználunk a kimeneti minta kiszámolásához.

Az osztály *header* fájljában egy *enum*ban tárolja a burkológörbe öt állapotát. Az öt állapot: nem játszunk le hangot (*ENVELOPE_STAGE_OFF*), a felfutási időben tart a hang (*ENVELOPE_STAGE_ATTACK*), a maximális értéktől lecseng a hang (*ENVELOPE_STAGE_DECAY*), a hang kitartását jellemző állapot (*ENVELOPE_STAGE_SUSTAIN*) és a hang felengedése utáni állapot (*ENVELOPE_STAGE_RELEASE*).

A legtöbb funkcióhoz a *calculateMultiplier* függvény szükséges. A függvény két kimeneti szint közötti lefutást valósít meg, például a hanglejátszás kezdetétől a maximális kimeneti érték elérését. A két szint értékéből és az időközben lejátszott minták számából egy szorzóértéket számol.

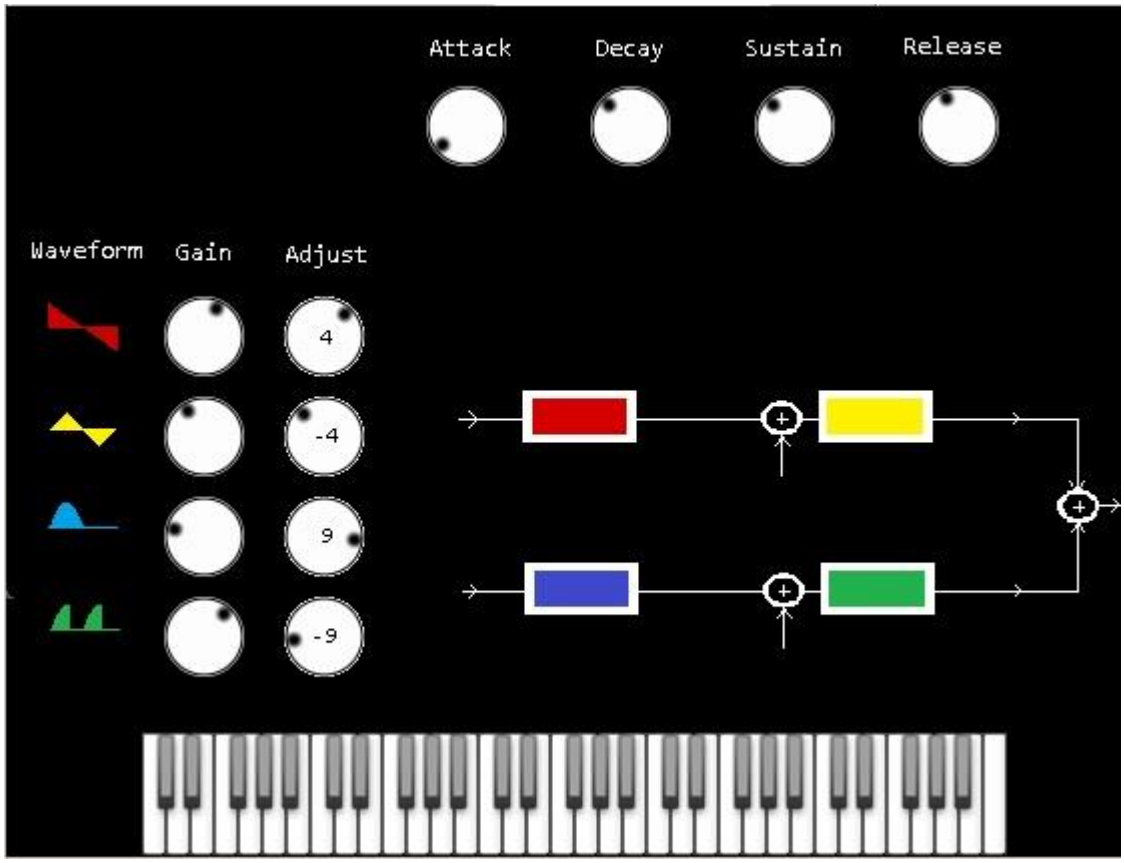
Az *enterStage* függvény az állapotváltások során hívódik meg, a függvény argumentuma a következő állapot. Amennyiben az előző állapot a felfutás, lecsengés vagy a felengedés, az osztály automatikusan állapotot vált a beállított idő után. A kitartási és ”off” állapotot külső parancs hatására hagyja el. Minden egyes új hang lejátszásakor egy ciklust indít el a burkológörbe-generátor, ami az állapotokon sorrendje szerint az oszcillátorok kimenetét megváltoztatja. Az új állapot megkezdésekor a belső számláló értékét nullára állítja. Ez a számláló a lejátszott minták számát tárolja, az osztály ezzel valósítja meg a kimenet időbeli változását. A függvény az állapotok szerinti szorzó és kimeneti értéket a switch-case segítségével számolja ki.

A *setStageValue* a burkológörbe-generátor potmétereinek megváltoztatásával hívódik meg. A függvény a paraméter értéke és állapota szerint átállítja az állapotokhoz rendelt változókat. Amennyiben ez a megváltoztatott érték az épp lejátszott hangra befolyással van, a függvény kiszámolja a pillanatnyi kimeneti érték és a következő állapotszint közötti értékváltoztatás módját.

A *nextSample* függvény a szorzó és az oszcillátor kimenete alapján egy új kimenetet generál.

6.4 Felhasználói felület

Az alkalmazásom grafikus felülete az alábbi képen látható.



31. ábra: GUI

A program elnevezésével megegyező *header* fájlban inicializáltam az alkalmazás függvényeit és változóit. A példaprogramban látott *Reset, OnParamChange* és *ProcessDoubleReplacing* mellet a beépített szintetizátor és a MIDI feldolgozó függvényei is megtalálhatóak. A *GetNumKeys* és a *GetNumStatus* függvény a lenyomott billentyű hangmagasságát és állapotát kérdezi le. Egy konstans változóban tároljuk a megjelenített zongorabillentyűzet legkisebb lejátszható hang MIDI kódját. A *private* részben a számoláshoz használt szabályzók változói és az oszcillátorok, MIDI feldolgozó és burkológörbe példányai szerepelnek.

A függvényeket implementáló *cpp* fájl egy *EParams enum*-mal kezdődik. A változtatható paraméterek az oszcillátorok jelalakja (*Waveformone, Waveformtwo, Waveformthree, Waveformfour*), hangereje (*Gainone, Gaintwo, Gainthree, Gainfour*) elhangolása (*fqone, fqtwo, fqthree, fqfour*) és kapcsolás módja (*mOscMode*). A burkológörbe paraméterei az *mAttack, mDecay, mSustain* és az *mRelease*. Az

*ELayout*ban tárolom a grafikus felület magasságát és szélességét. Az *FmMode* az összeköttetések módját tárolja.

A programot megvalósító osztály konstruktorában hozom létre a paramétereket változtató eszközöket és a beépített szintetizátort. Az eszközökön kívül egy háttérképet is használok a grafikus felülethez, ezt az *AttachBackground* függvénnyel teszem meg, az argumentuma a csatolt háttérkép azonosítója és elérési útja, amit a *resource.h* fájlban definiáltam. A szintetizátorbillentyűzet a háttérképen szerepel, a lenyomott billentyűkhöz tartozó képeket viszont külön kell beilleszteni. A fekete billentyűnek egy lehetséges képe van, a fehérnek hat. Az 5. fejezetben leírtak alapján a szintetizátort az *IKeyboardControl* osztály alkalmazásával lehet létrehozni. A lejátszható oktávok száma öt, a beillesztett kép x,y koordinátáját és a billentyűk elhelyezkedését tároló tömböt definiáltam. A jelalakokat egy *ISwitchControl* osztállyal lehet változtatni, aminek az *IBitmapControl* az ősosztálya. A jelalakokat *enum* típusú változóban tároltam, az *InitEnum* függvénnyel létrehoztam hullámforma kapcsolóját. Minden oszcillátor nyolc féle hullámformát tud létrehozni, a kép beillesztését is eszerint kell megvalósítani. A burkológörbe paramétereit potenciométerrel lehet változtatni, értéküket *double* változóban tároltam, az *InitDouble* függvény segítségével az értékhatárokat és a lépésközt beállítottam. A potmétert az *IKnobMultiControl* osztály felhasználásával lehet létrehozni. A hangerőt is egy potméterrel lehet állítani. Az alapértelmezett érték százalékban kifejezve 50, a határai 0 és 100, a lépésköz pedig 0.01, viszont a program csak a képkockák számának megfelelően tud váltani értékek között. Az elhangolást már egy *int* típusú változóban tárolom, a bemenetet a billentyűhöz tartozó hangmagassághoz képest maximálisan egy oktávval lehet elhangolni. A paraméter minimális és maximális értéke -12 és 12, ami a félhangok számát jelöli.

A paraméterek megváltozásakor az *OnParamChange* függvény hívódik meg. Hullámformát az oszcillátor osztály *setMode* függvényével lehet átállítani, a *GetParam->Int* a jelalak számát kérdezi le. A burkológörbe paramétereit a *setStageValue* függvény segítségével megváltoztatjuk. Az oszcillátorok kapcsolását az *FM_mode* érték szerint változtatom. A hangerő és az elhangolás változóit (*mGain, mfq*) a *ProcessDoubleReplacing* függvényben használom, a *GetParam->Value* függvénnyel állítom át a megadott értékre,.

A *ProcessDoubleReplacing* függvény adja meg a kimeneti minták értékét. A bal és jobboldali sáv tárolóját deklaráltam a függvény elején, az utána következő *for*

ciklusban feltöltöttem a megfelelő értékekkel. A két sáv adatai megegyeznek. A *for* ciklus minden egyes lefutásakor a *MidiReceiver advance* függvényével a MIDI üzeneteket lekérdezem. A *velocity* a hangerő értékét tárolja. A MIDI feldolgozóból kapott frekvenciát az elhangolásnak megfelelően átállítom. A bemeneti hanghoz képest a frekvenciát az alábbi képlet szerint lehet felírni:

$$f_{new} = (\sqrt[12]{2})^n f \tag{22}$$

ahol f_{new} az elhangolt frekvencia, f a MIDI hang frekvenciája és n a két hang közötti félhangok száma. Az oszcillátorok frekvenciáját (22) szerint beállítottam. A plugin hét kapcsolását egy *switch-case* szerkezettel oldottam meg, az *FM_mode* értéke alapján. Az összeköttetések képleteiben az $O_x()$ az x oszcillátor mintája, EG a burkológörbe állapot szerinti mintája, τ_x az x oszcillátor fázisa, G_x az x oszcillátor hangerőszabályzó értéke és N a kimeneti minta. Az oszcillátorok fázisát a *getPhase* függvénnyel kapom meg.

Az első kapcsolás csak egy oszcillátort használ, a burkológörbe és oszcillátor *nextSample* függvényeinek szorzatát az *mGainone* változóval a megfelelő hangerő értékre állítom.

$$N = G_1 O_1(\tau_1) EG \tag{23}$$

A második kapcsolás két minta összegét adja meg, az oszcillátorok mintáit egyenként megszorzom a hangerőszabályzó értékével, az így kapott eredményeket összeadom és a burkológörbe értékével megszorzom.

$$N = [G_2 O_2(\tau_2) + G_1 O_1(\tau_1)] EG \tag{24}$$

A harmadik kapcsolás FM hangszintézist valósít meg. Az oszcillátor *getPhase* és *sample* függvényével valósítom meg a fázis modulációt. Az első oszcillátor kimenetét a második *sample* függvény argumentumában használom fel.

$$N = \{G_2 O_2[\tau_2 + G_1 O_1(\tau_1)]\} EG \tag{25}$$

A negyedik kapcsolás a négy oszcillátor mintái felhasználásával jön létre. Az első oszcillátor kimenetét megszorzom a hangerő változóval. A második oszcillátor mintájának értékét a *getPhase* függvény szerinti fázis és az első oszcillátor értékével számolom ki.

$$N = \langle G_4 O_4\{\tau_4 + G_3 O_3[\tau_3 + G_2 O_2(\tau_2 + G_1 O_1(\tau_1))]\}\rangle EG \tag{26}$$

Az ötödik kapcsolás kimenete két fázismodulált jel összege. Az *Oscillatortwo* és *Oscillatorfour* hullámformáit az *Oscillatorone* és *Oscillatorthree* kimeneti értékeivel moduláltam. A két minta összege lesz a végeredmény.

$$N = \{G_4O_4[\tau_4 + G_3O_3(\tau_3)] + G_2O_2[\tau_2 + G_1O_1(\tau_1)]\}EG \qquad (27)$$

A hatodik kapcsolásban a negyedik oszcillátort a harmadik oszcillátor mintájával modulálom, amit már moduláltam a második oszcillátor értékével. Az így modulált értéket összeadom az első oszcillátor mintájával.

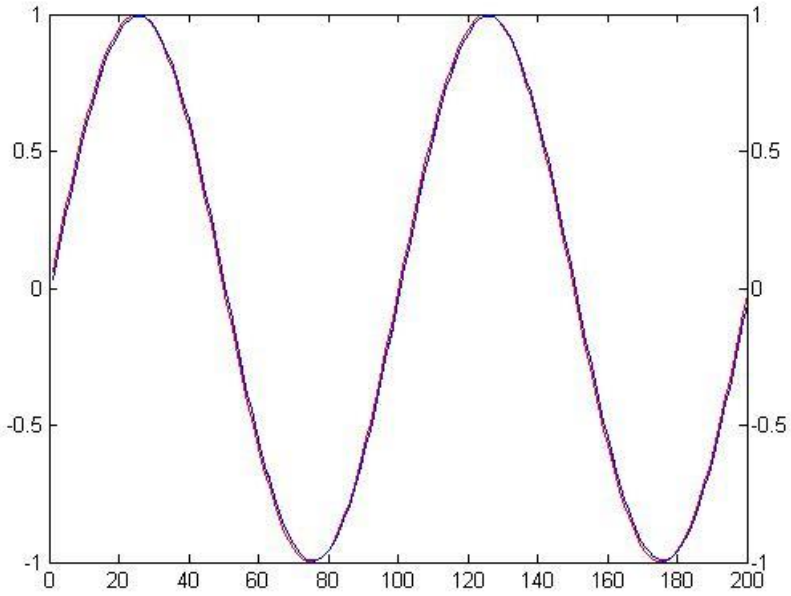
$$N = \langle G_4O_4\{\tau_4 + G_3O_3[G_2O_2(\tau_2)]\} + G_1O_1(\tau_1)\rangle EG \qquad (28)$$

A hetedik kapcsolás a (24) és (25) összegével leírható. Az FM hangszintézist az *Oscillatorthree és Oscillatortwo* valósítja meg. A kimenetek összegét a burkológörbe mintájával megszorzom.

$$N = \{G_4O_4(\tau_4) + G_3O_3[\tau_3 + G_2O_2(\tau_2)] + G_1O_1(\tau_1)\}EG \qquad (29)$$

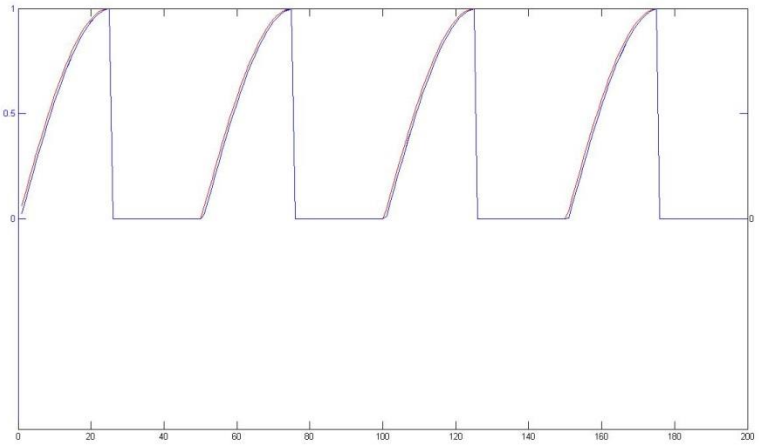
6.5 A C++ megvalósítás tesztelése

A C++ plugin-t a SAVIhost nevű programmal indítottam el. Az MATLAB és C++ összehasonlítását az első kapcsolás segítségével oldottam meg. A hoszt programban lehetőség van a kimeneti jelalakokat wav fájlba menteni. A burkológörbe felfutási, lecsengési és felengedési idejét a minimális értékre állítottam, a kitartási szintet maximumra. A hangfájlt az Audacity programmal megvágtam, és ábrázoltam. A MATLAB *wavread* függvényével a fájl értékeit egy kétdimenziós tömbbe mentettem, és a *plotyy* függvénnyel ábrázoltattam. Az összehasonlításhoz a MATLAB *op1* függvényt megváltoztattam, hogy azonos mennyiségű mintát lehessen összehasonlítani, a kapcsolások függvényeinél a kimenetet kétdimenziós tömbbe mentettem. A piros jelalak a MATLAB szimuláció, a kék a C++ program kimenete. Az előzőleg 49700 Hz értékű mintavételi frekvenciát megváltoztattam a C++ programban használt értéke.



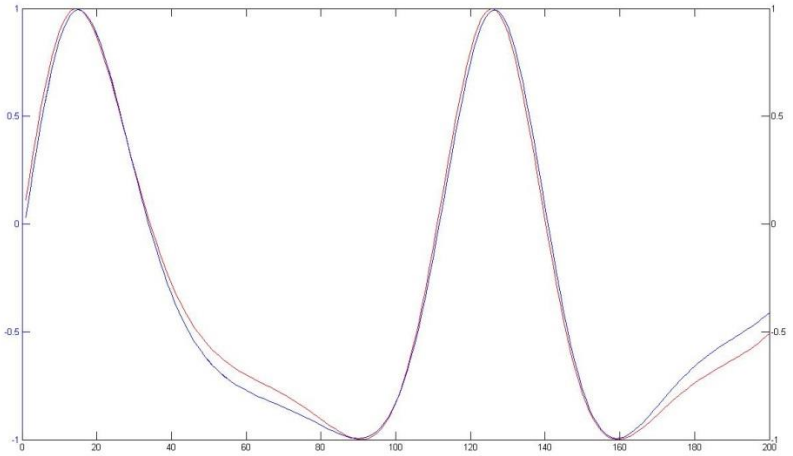
32. ábra: első hullámforma összehasonlítása

Az ábrán a szinuszfüggvény hullámformája látható. A kék és piros vonalak közötti eltérés minimális, a MATLAB függvény egy mintával előbb kezdi el a számolást és rajzolást.



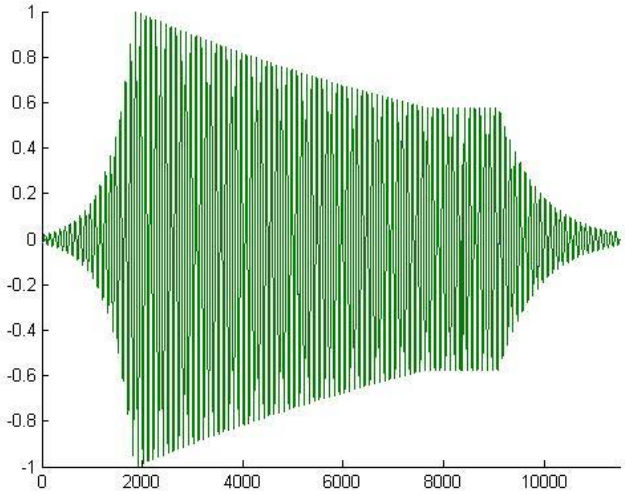
33. ábra: második hullámformák összehasonlítása

A második tesztelt jelalak a C++ program *OSCILLATOR_MODE_THREE* kimenete. A két hullámforma jellege megegyezik.



34. ábra: FM hangszintézis összehasonlítása

A harmadik tesztelt hullámformán az FM hangszintézis összehasonlítása látható. A két hullámforma között már jelentősebb különbség látható, a C++ program frekvencia közötti váltásnál egy közelítő értéket használ $\sqrt[12]{2}$ értékre, így a frekvenciák nem ugyanazok a két szimuláció felhasználásánál.



35. ábra: Burkológörbe szemléltetése

Az ábrán egy tetszőleges beállítású burkológörbe-generátor által vezérelt szinuszjelet láthatjuk. Jól látható a felfutási idő, lecsengési idő és a felengedési idő változtatásával elért változások, és megfigyelhető a 8000-9000 minták között a beállt szint. Az ábrák alapján elmondható, hogy a C++ program megfelelően hajtja végre az OPL3 szimulálását.

7 Összehasonlítás

Az elkészült plugint összehasonlítottam az interneten található ingyenes VST programokkal, a fő eltéréseket kétpélda program segítségével szemléltetem [21]. A legtöbb FM szintetizátor általános megoldást alkalmaz, és grafikus felülete zsúfolva van a szabályzókkal. A chipeket szimuláló programok általában az OPL2 alapján működnek.

7.1 Brzoza

36. ábra: Brzoza [19]

Az első program egy általános FM hangszintetizátor. A [19] honlapon az FM szintetizátorok közül a legjobb értékelést kapta. Két oszcillátor és hat modulátor segítségével állítja elő a hangot. Minden egyes oszcillátor négy féle jelalak előállítására képes, a jelalakokat a bementi MIDI üzenethez képest -12 és 12 félhanggal lehet elhangolni. A két félhang között egy másik potméter segítségével finomabb elhangolásra is lehetőség van. A fázismodulációt a beérkező hanghoz képest lehet beállítani egy szabályzó segítségével. A hangerő értékét egy potenciométerrel és egy burkológörbével szabályozhatjuk. A programban még megtalálható szűrő is, ami az eredeti FM hangszintézisben nem volt megtalálható.

A Brzoza programhoz képest letisztultabb grafikus felületű programot készítettem el. Bár az állítási lehetőségek száma nagy, az oszcillátorokat külön moduláljuk, és nem egymást modulálják. A programom az oszcillátorok nagyobb számú hullámformáival,

az egymásra való hatásaival és a kapcsolások nagyobb számával a kimenetet megannyi módon lehet befolyásolni, valamint a felhasználói felülete is átláthatóbb.

7.2 FM-Four

37. ábra: FM-Four OPL2 szintetizátor

Az ingyenes FM szintetizátorok közül talán az FM-Four hasonlít legjobban az elkészített programomhoz. Az FM-Four négy oszcillátort tartalmaz, amelynek 8 lehetséges jelalakja van. Az oszcillátorok szabályzóival az elhangolás és a kimeneti hang szintjét lehet állítani. A program tartalmaz beépített szűrőt, burkológörbé-generátort, és a kapcsolásokat megvalósító algoritmusokat. Az FM-Four a programomhoz hasonlóan állítja elő a hangot, az oszcillátorok összeköttetését a kép megváltoztatásával lehet beállítani.

A programomhoz képest az FM-Four több beépített szabályzóval változtatja kapcsolások kimenetét. A fő különbség az oszcillátorok kapcsolásának módja, az FM-Four az OPL2 algoritmusait alkalmazza, és a felhasználható jelalakok is különböznek.

8 Összefoglalás

A szakdolgozatom az FM hangszintézis megvalósításával foglalkozott. Olvasása során megismerkedtünk az absztrakt algoritmusok működéseinek módjaival, különös tekintettel az FM hangszintézisre. Az OPL3 működését a következő fejezetben írtam le, kitértem a chip kapcsolási módjaira és a burkológörbe-generátor bemutatására. A negyedik fejezet a MATLAB szimulációt tárgyalja, amiben a jelalakokat és kapcsolásokat először megvalósítottam és analizáltam. A következő fejezetben a VST, WDL-OL/IPlug környezet leírása található, emellett kitérek a grafikus felület megjelenítéséhez használt KnobMan program működésére is. A megvalósított program felépítése és működésének leírása a hetedik fejezetben található. Megismerkedhetünk a felhasznált osztályok függvényeivel, a program felületével és a MATLAB, C++ hangszintézis összehasonlításával. Az utolsó fejezetben két népszerű FM szintetizátort megvalósító plugin segítségével leírom a programom jellegzetességét.

Bár a feladatkiírásban nem szerepel, a polifonikus hangzások előállítását is szerettem volna elvégezni, amit az idő hiányában nem tehettem meg. A plugint más irányban is szeretném továbbfejleszteni. A hang szűrése, késleltetése és zengetése további lehetőségeket kínál a kívánt hangzás elérésében. A hangok generálását további programkód írásával gyorsabbá lehet tenni.

A feladatkiírásban szereplő célokat sikerült megvalósítani. Az IPlug lehetőségeit megfelelően alkalmaztam a programom megvalósítása során, a hang lejátszása az elvártaknak megfelelően működik.

Irodalomjegyzék

[1] Tero Tolonen, Vesa Välimäki, Matti Karjalainen – Evaluation of Modern Sound Sythesis Methods – Technical Report - 1998. március –

letöltve: 2014. december 9.
http://legacy.spa.aalto.fi/publications/reports/sound_synth_report.pdf

[2] Julius O. Smith - Taxonomy of Digital synthesis techniques- Viewpoints on the History of Digital Synthesis

letöltve: 2014. december 9.
https://ccrma.stanford.edu/~jos/kna/Taxonomy_Digital_Synthesis_Techniques.html

[3] Burk, Polansky, Repetto, Roberts, Rockmore –

Music and Computers jegyzet- The synthesis of Sound by Computer fejezet – Karplus-Strong alfejezet

letöltve: 2014. december 9.

http://music.columbia.edu/cmc/musicandcomputers/chapter4/04_09.php

[4] Tamara Smith – Waveshaping synthesis előadás jegyzet – 2012

letöltve: 2014. december 9.

http://www.cs.sfu.ca/~tamaras/waveshapeSynth/waveshapeSynth.html

[5] Burk, Polansky, Repetto, Roberts, Rockmore –

Music and Computers- The synthesis of Sound by Computer fejezet – Waveshaping alfejezet - letöltve: 2014. december 9.
http://music.columbia.edu/cmc/musicandcomputers/chapter4/04_06.php

[6] Miller Puckette - Theory and Techniques of Electronic Music online könyv – Waveshaping – letöltve: 2014. december 9.

http://msp.ucsd.edu/techniques/v0.08/book-html/node74.html

[7] Phil Scott, Joel Sing, Ka Ching Chan – Data Communications online jegyzet – lecture 7 –

letöltve: 2014. december 9.

http://ironbark.xtelco.com.au/subjects/DC/lectures/

[8] Wikipedia - Frequency modulation - letöltve: 2014. december 9.

http://en.wikipedia.org/wiki/Frequency_modulation

[9] Wikipedia - Frequency modulation synthesis - letöltve: 2014. december 9.

http://en.wikipedia.org/wiki/Frequency_modulation_synthesis

[10] Jeffrey Hass - Principles of Audio-Rate Frequency Modulation jegyzet – Indiana University School of Music -2001-

letöltve: 2014. december 9.

http://www.indiana.edu/~emusic/fm/fm.htm

[11] Wikipedia - Yamaha_YMF262

letöltve: 2014. december 9.

http://en.wikipedia.org/wiki/Yamaha_YMF262

[12] Wikipedia - ADSR Envelope

letöltve: 2014. december 9.

http://en.wikipedia.org/wiki/Synthesizer#ADSR_envelope

[13] HWSW – Hangszintézis cikk - Wavetable szintézis – folytatás

letöltve: 2014. december 9.

http://www.hwsw.hu/hirek/40447/amit-a-hang-kartyak-rol-tudni-erdemes.html?oldal=7

[14] Yamaha- OPL3 adatlap - letöltve: 2014. december 9.

http://web.archive.org/web/20030520163334/http://www.fit.vutbr.cz/~arnost/opl/lost+found/ymf262.pdf

[15] Orosz György, Dr. Sujbert László –

Elosztott rendszerek és szenzorhálozatok 1. jegyzet – 2008. március - letöltve: 2014. december 9. http://www.mit.bme.hu/system/files/oktatas/targyak/8605/9meres_0.pdf

[16] Wikipedia - Virtual Studio Technology – letöltve: 2014. december 9. http://en.wikipedia.org/wiki/Virtual_Studio_Technology

[17] Cockos incorporated – WDL – letöltve: 2014. december 9. http://www.cockos.com/wdl/

[18] Martin Finke- Making Audio Plugins – letöltve: 2014. december 9.- http://martin-finke.de/blog/tags/making_audio_plugins.html

[19] Oli Larkin – Iplug Workshop slides & examples – letöltve: 2014. december 9. - http://olilarkin.blogspot.de/2012/01/m4u-convention-iplug-workshop-slides.html

[20] Oli Larkin – IPlug-OL – letöltve: 2014. december 9. – http://www.olilarkin.co.uk/html/annotated.html

[21] VST4free – letöltve: 2014. december 9. – http://www.vst4free.com/