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

Ibanez WH-10 wah-wah effekt modellezése iOS környezetben

Szakdolgozat

Készítette
Simon Tibor

Konzulens
dr. Bank Balázs

2012. december 11.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Simon Tibor*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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 autentikált 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.

Budapest, 2012. december 11.

Simon Tibor
hallgató

Kivonat

A technika fejlődésével egyre inkább megjelenik az igény a zenei iparban, hogy a hagyományos analóg effektezés helyett digitálisan oldjuk meg a felmerülő feladatokat, az analóg minőségből mit sem veszítve. Ennek oka, hogy az analóg eszközök jó minőségben csak nagyon drágán kaphatóak, esetenként igen nagy kivitelezésben. Ezzel szemben a digitális eszközök egy számítógépen futnak, viszonylag olcsóbbak, és általában több lehetőséget is nyújtanak, mint az analóg megfelelőik. Jobban kiépített paraméterkezelésükkel akár az analóg effektekkel lehetetlen, időalapú paraméterezési feladatok is megoldhatóak.

Jelen dolgozat során bemutatásra kerül egy analóg effekt digitalizálása részletekbe menő alapossággal, majd az elkészült digitális modellt iOS eszközön valósítjuk meg. Megismerkedünk az áramkör felépítésével, annak funkcióival és működésével. Az analóg rendszert áttranszformáljuk digitálissá, majd az iOS eszköz adta lehetőségeket kihasználva beprogramozzuk, és teszteljük az elkészült effektet. A munka során MATLAB segítségével oldjuk meg a számítási és tervezési feladatokat.

Abstract

In the developing technology there appeared a need in the music industry to simulate analog effects in a digitally way, but not loosing any quality of the analog sound. The reason is that the pro quality analog instruments could be very expensive and sometimes they could be rather large in size. Instead of the digital instruments run on a computer, they are relatively cheap and they can offer sometimes more than the analog ones. With their unique digital parameter handling systems, it is possible to achieve such a time based parameter handling that is impossible in an analog way.

This thesis will guide you through the process of transforming an analog effect into a digital one. The final digital model will be implemented in an iOS environment. We will get acquaintance with the structure of the circuit, it's behaviors. We will transform the analog system into a digital one, then, with the iOS resources, we will implement the digital system. During the work we use MATLAB to design the necessary elements.

Bevezető

A wah-wah pedál egy elektromos gitárokhoz használt elektromos effekt pedál. Hangja jellegzetes, azonnal felismerhető. A vele előállított hang hasonlít az emberi beszéd hangjaira. A wah-wah hatást először a fúvós zenében használták, ahol a hangszer tölcsérébe helyezett különböző alakú tompítókkal értek el az eredetitől eltérő hangszíneket, a lágy, mély tónusú hangoktól egészen az érces, éles, magas hangszínekig. A wah-wah pedál segítségével a zenész aktívan befolyásolhatja hangszere hangszínét, ezzel még színesebbé téve a játékát.

A wah-wah pedálok a hatvanas években kezdtek el megjelenni zenei piacon. Nevük felépítésükből adódik: egy pedál, amit előre hátra lehet dönteni, ezzel szabályozva az effektezett hangszer hangszínét. Előre döntve a pedál a magas hangokat emeli ki, hátra döntve pedig a mélyeket. Ebből következik, hogy a wah-wah effekt lényegében egy változtatható kiemelési frekvenciájú sávszűrő.

A zenei iparban először a Vox által kifejlesztett, tekercsen alapuló effektek terjedtek el, de sorra jelentek meg az újabb konstrukciók, a dupla T szűrőkonstrukciók, a többszörös visszacsatolású aktív, műveleti erősítős áramkörök és az állapotváltozókon alapuló effektek. Sokak szerint a hagyományos, tekercs alapú wah-wah pedálok rendelkeznek a legfényesebb hanggal, de mindegyik típusú effektnek megvan a maga zenei szerepe [19].

Az első fejezetben kívülről belülről megismerkedünk magával az analóg effekttel, és megnézzük, hogy milyen részegységekből áll.

A második fejezetben ezen részegységek példáján keresztül nézzük meg, hogy milyen lehetőségeink vannak a digitális átalakításra.

A harmadik fejezetben betekintést nyerünk az iOS operációs rendszer zenei moduljának, a Core Audio-nak a működésébe.

A negyedik fejezetben az addig megszerzett ismereteket összesítjük, és létrehozzuk a digitális rendszereket.

A dolgozat olvasásához fontos tudnivalók:

- Kapcsolási rajzra való hivatkozásnál, a rajz egyes szegmenseire is hivatkozhatok. Például az [F.3 CD:26] jelentése: az F.3 ábrán a rajz felosztásában a C-D és 2-6 oldalú téglalapban szereplő részlet.
- A képletekben a replusz művelet jelölése: $\otimes$.

- Tizedespont jelölésére pontot használok.
- A dolgozatban szereplő kapcsolási rajz részletek a CircuitLab [13] segítségével készültek.
- A közölt forráskódok a MATLAB programozási nyelvében íródtak, hogy mindenki számára érthetőek legyenek.

1. fejezet

Az analóg effekt

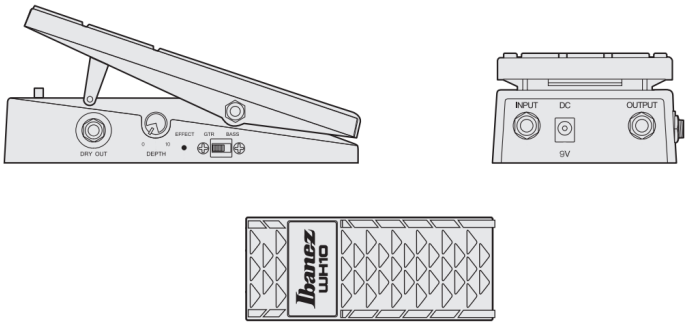
1.1. A WH-10 megjelenése

Az Ibanez 2009-ben úgy döntött, hogy újra kiadja WH-10 nevű, híres, műveleti erősítős wah-wah pedálját. A WH-10v2 a cég állítása szerint ugyanaz, mint az elődje, egyedül a pedál háza változott műanyagról öntöt fémre, ezzel sokkal ellenállóbbá téve a használat miatti esetleges sérülésektől.

A régi pedál ma már igen nehezen beszerezhető effektnek számít, csak használtan lehet hozzájutni. Ezzel az új kiadással azonban bárki számára hozzáférhetővé vált az Ibanez híres wah-wah hangzása.

Sok ismert zenész szereti a WH-10 hangját, de talán mindközül a leghíresebb John Frusciante, a Red Hot Chili Peppers volt gitárosa, akinek az egyedi gitárhangzásának az egyik alapvető építőköve az Ibanez wah-wah pedál.

1.2. Fizikai felépítés



1.1. ábra. *WH-10 pedál felépítése [18].*

A WH-10 szürkére porfestett, öntött fémházban kapható ma a boltokban. Felépítése a hagyományos wah-wah pedálokét tükrözi: nagy, előre hátra dönthető felület, ami segítségével a zenész a lábával szabályozhatja a wah-wah pedál hangzását. A felületet erősen előre nyomva az effekt ki/be kapcsolható. Négy darab csatlakozó került rá: egy külső tápegység

csatlakozó, egy bemeneti hangszer csatlakozó, és két kimeneti csatlakozó, egy a direkt, effektezés nélküli és egy az effektezett jelhez. Egyedi a wah-wah pedálok között, hogy a wah-wah hangzás erőssége változtatható egy potenciométer segítségével, ami a tiszta és a szűrt jelet keveri egymáshoz a beállított aránynak megfelelően. A WH-10 mind elektromos gitárhoz, mind basszusgitárhoz használható. A két hangszertípus eltérő frekvenciatartományához egy kapcsolóval tud igazodni az effekt, ezzel még szélesebbé téve a felhasználási körét.

1.3. Kapcsolási rajz, áramkör visszafejtés

Az interneten keresve legalább háromféle kapcsolási rajz található a pedálról. Ezek között szerepel az egy vagy több műveleti erősítővel megvalósított sávszűrő, egy vagy két potenciométerrel vezérelve, sima wah-wah, vagy torzítóval felszerelt wah-wah.

A mi esetünkben egy sima wah-wah pedálról van szó, amiben két potenciométer oldja meg a wah-wah hatást. A hozzá tartozó kapcsolás megtalálható Dirk Hendrik [17] weboldalán, a dolgozatban pedig a függelékeknél látható [F.3].

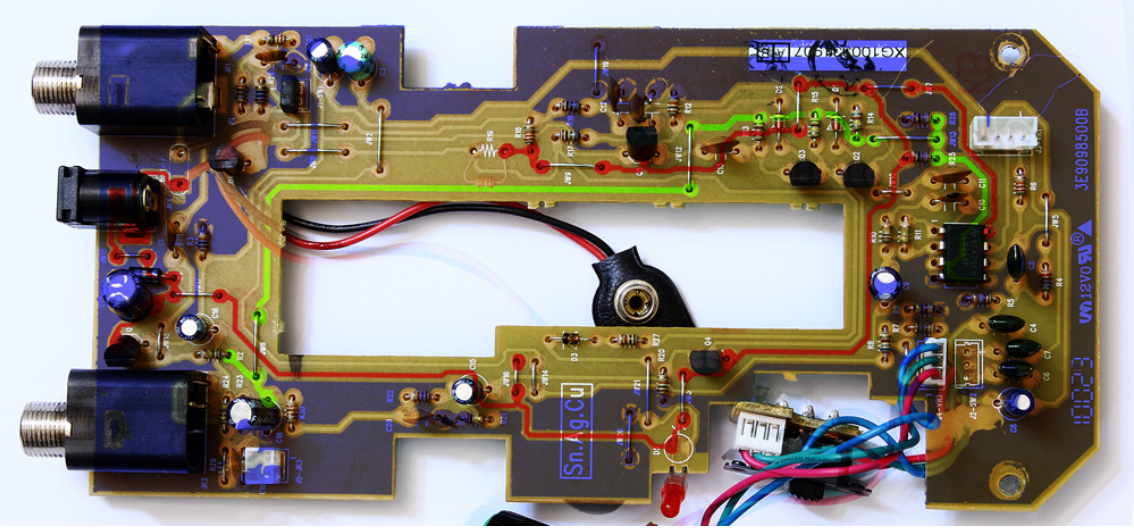
A sok kapcsolási változat miatt felmerül az emberben a kétség, hogy az ő pedáljában biztosan az az áramkör van-e megvalósítva, ami a kapcsolási rajzon szerepel. Emiatt, hogy teljesen biztosak legyünk abban, hogy a tényleges hardvert modellezzük, a pedál nyákja alapján visszakövetjük a kapcsolási rajzot.

A nyák furatszerelt technológiával készült, ezért az egyik oldalán csak alkatrészek, a másik oldalán csak a huzalozás van, ezért viszonylag nehézkes szabad szemmel, az áramkört kézben tartva követni a vezetősávokat. Valamilyen más megoldás után kell nézni.

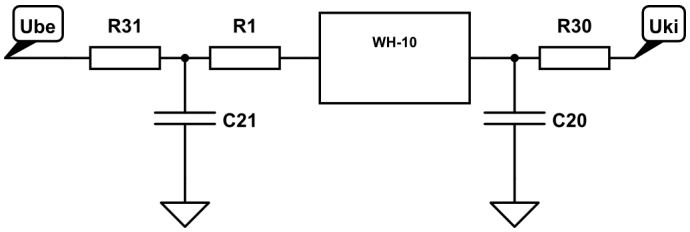
A vizsgálathoz egy vizuális módszert fogunk használni, ami során a nyákról két fényképet készítünk, egyet az alkatrész oldalról, egyet a huzalozási oldalról, majd a két réteget digitálisan utómunkával egymásra helyezzük úgy, hogy az alkatrész oldalra tükrözve kerüljön a huzalozási oldal. A felhelyezett réteg keverési módjának és áttetszőségének állításával olyan hatást érhetünk el, mintha átlátnánk az áttetszetlen nyáklemezen, ezzel mintegy felfedve az alatta futó vezetősávokat, így digitális röntgenképet kapunk (1.2. ábra). A pontos pozícionálás után egyszerűen lehet követni a jelek útját. Nincs más hátra, mint minden alkatrész bekötését és értékét ellenőrizni. Ha szerencsénk van, akkor a kapcsolásnak, és a tényleges áramkörnek meg kell egyeznie.

A jobb átláthatóság érdekében az áramkör három fix potenciálját ($VCC-piros, GND-kék, VBIAS-világoszöld$) jól elkülöníthetően kiemeltem a képen. Az egyes potenciálszintek funkciójára később visszatérünk.

Az ellenőrzés után kiderült, hogy az áramkör nagy része megegyezik a kapcsolási rajzzal, eltérés csak a bemeneti fokozat és az effektezett kimenet csatoló fokozata között van, valamint a wah effekt mélyégéért felelős potenciométer bekötése eltér a kapcsolási rajzon feltüntetettől. A bemeneti fokozatban R31 után a jelút és a föld közé C21, a jelúttal sorosan



1.2. ábra. *Digitális röntgenkép.*



1.3. ábra. *Kiegészítés az eredeti áramkörhöz.*

pedig R1 került be, a kimeneti fokozatban R22 után, a jelút és a föld közé C20, a jelúttal sorosan R30 került be (1.3. ábra).

Az áramkör pontos megismerése után nézzünk bele az áramkör működésébe!

1.4. Az áramkör működése

Működés szempontjából három jól elkülöníthető funkcionális egységre bonthatjuk az áramkört: a működéshez szükséges feszültségszintek előállításáért felelős, az audiojel vezetésére és kapcsolására szolgáló és az audiojel alakítására szolgáló funkcionális egységekre.

1.4.1. Tápellátás és feszültségszintek

A wah-wah pedál működhet elemről, vagy külső tápegységről. Mindkét esetben 9 V feszültségű forrásra van szüksége. A tápfeszültséget kezelő áramkör nagyon egyszerű. Valamelyik külső forrásról kapott feszültséget közvetlenül rávezeti az áramkör VCC sínjére, miután egy párhuzamos diódával (D4) védte a fordított polaritású bekötés ellen, és szűrte C17 kondenzátorral. Az elem két szinten is védve van a használaton kívüli lemerülés ellen. Egyidejű elemes és külső tápegységes táplálás esetén, a külső tápegység csatlakozója fizikailag leválasztja (J3) az elem pozitív kapcsát a többi áramköri résztől, rajta nem folyhat áram. Elemes táplálás esetén az elem negatív sarka csak akkor csatlakozik a GND sínhez, ha a bemeneti csatlakozóba a dugó be van helyezve [F.3 AB:78].

Az áramkör aszimmetrikus táplálása miatt a műveleti erősítők nem képesek negatív feszültséget kiadni a földpotenciálhoz képest, ezért az effekt tervezői kénytelenek voltak szimulálni a szimmetrikus táplálást oly módon, hogy az egyoldalas tápfeszültséget a feszültségszint felénél megcsapolták ($VBIAS$), és ezt használták földpotenciálnak a váltakozó jelek szempontjából. Az ezt megvalósító áramkör egy 1/2 arányú feszültségosztó (R25A és R26), aminek a kimeneti feszültsége szűrve van egy kondenzátorral [F.3 BC:78].

A bemeneti fokozat egyik első feladata a bemeneti jel DC szintjének felhúzása VBIAS potenciálra, R2 ellenálláson keresztül. A kimeneteken AC leválasztással szabadulunk meg az ofszettől [F.3 D:7 és CD:23].

1.4.2. Jelút vezetése

Miután a jel a bemeneti fokozatban a feldolgozáshoz megfelelő feszültségszintre került, a további szűréseket és módosításokat végrehajtó áramkörökhöz el kell juttatni azt.

Mint általában minden effektpedálnak, a WH-10-nek is két állapota van működés szempontjából: bekapcsolt (effektezett) és bypass mód.

Bekapcsolt módban a bemenőjelet a szűrést elvégző áramkörökhöz kapcsoljuk, majd a szűrt jelet vezetjük a kimenetre, bypass módban pedig a jel szűrés nélkül, közvetlenül a kimenetre jut.

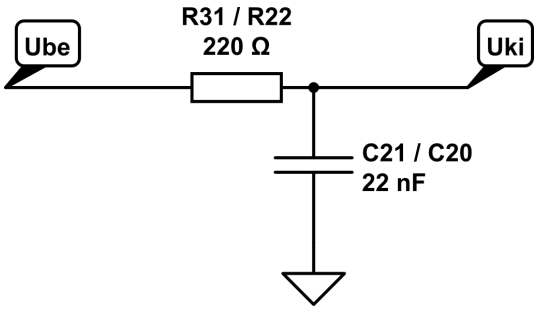
Ezt a jelkapcsolást a jeltovábbító hálózat feladata megoldani. Funkció szerint két részre bontható: tényleges kapcsolást végző, valamint a kapcsolást vezérlő jeleket előállító áramkör.

A vezérlőjelek előállítását egy egyszerű tranzisztoros (Q6) áramkör végzi. Az effekt főkapcsolója (a pedálba épített lábkapcsoló, ami a pedálra való erős rálépéskor vált bekapcsolt és kikapcsolt állapot között) által válthatunk a két üzemmód között. Az áramkör két analóg, magas aktív vezérlőjelet ad: $BYPASS$, EFF_ON.

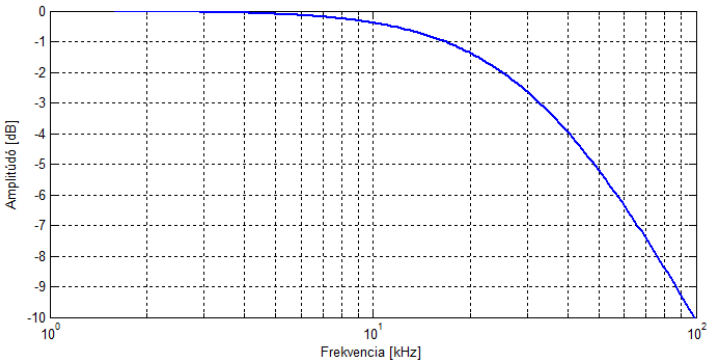
A kapcsoló zárásakor Q6 bázisa és a $BYPASS$ vezérlőjel földpotenciálra kerül, Q6 lezár, nem vezet, EFF_ON vezérlőjelet R18 ellenállás felhúzza VCC szintre, valamint az effekt működését jelző LED is bekapcsol.

A kapcsoló nyitásakor Q6 bázisának földelése megszűnik, a $BYPASS$ vezérlőjellel együtt VCC szintre kerül, áram fog bele folyni R28 ellenálláson keresztül, Q6 kinyit, EFF_ON vezérlőjel közel földpotenciálra kerül [F.3 AB:12].

A vezérlőjeleket a jelkapcsoló hálózat fogadja. A kapcsolást Q2 és Q3 N csatornás JFETek végzik. A két vezérlőjel állapota egymásnak mindig az ellentettje. $BYPASS$,$\overline{EFF\_ON}$ állapotban Q3 nyitott, Q2 zárt, ezért Q3 által vezetett szűretlen jel kerül a kimenetre. EFF_ON,$\overline{BYPASS}$ állapotban, pont fordítva, Q3 zárt, Q2 nyitott, Q2 által vezetett szűrt jel kerül a kimeneti fokozatra [F.3 CD:26].



1.4. ábra. *Bemeneti/kimeneti szűrők.*



1.5. ábra. *Bemeneti/kimeneti szűrők átvitele.*

1.4.3. Bemeneti és kimeneti szűrők

Az 1.3 ábrán már láthattuk, hogy az elméleti és a valós kapcsolás között csupán a bemeneti és a kimeneti szűrők mutattak eltérést. Az elméleti rajzon nem szerepelt két kondenzátor (C21,C20), az effektben azonban ezek bele vannak építve, ezzel mind a bemeneten, mind a kimeneten egy aluláteresztő szűrőt alkotnak (1.4. ábra).

Mindkét esetben egy egypólusú aluláteresztő szűrő szerepel, azonos paraméterekkel. Átvitelük a komplex frekvenciatartományban feszültségosztással kapható meg, ahol a kondenzátort helyettesítjük egy komplex impedanciával.

$$H(s) = \frac{1}{1 + sRC} \tag{1.1}$$

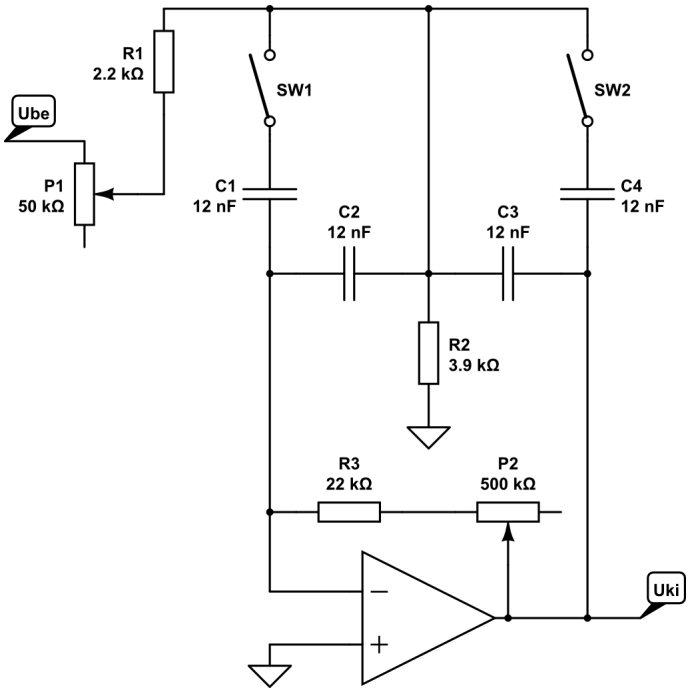
Ahol R és C helyébe az aktuális értékeket behelyettesítve az 1.5. ábrán látható átvitel valósul meg. A szűrő vágási frekvenciája audio szempontból nézve igen nagy, $f_c = 32992$ Hz, az ember számára nem érzékelhető a hatása, szerepük csupán a nagyfrekvenciás zavarok kiszűrésében van.

1.4.4. Sávszűrő

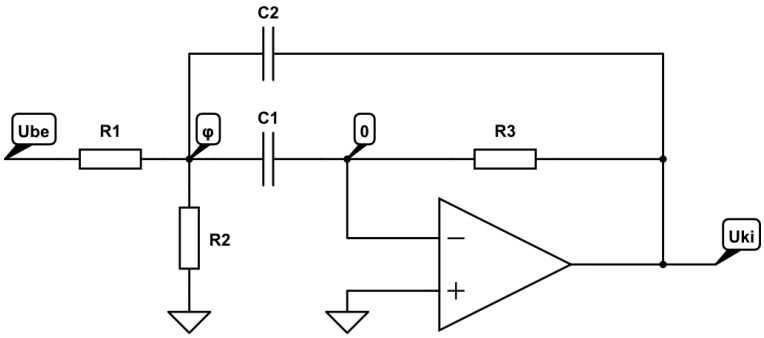
Minden wah-wah effektek központi eleme egy sávszűrő, és ezzel a WH-10 sincs másképp.

A kapcsolási rajz alapján a WH-10 esetén ezt egy két ágon visszacsatolt műveleti erősítős kapcsolás valósítja meg (1.6. ábra).

Az áramkörben négy állítási lehetőség van: két potenciométer (P1, P2) és két kapcsoló (SW1, SW2). A valóságban ez leegyszerűsödik két paraméterre: a kapcsolók fizikailag egy



1.6. ábra. *WH-10 wah-wah szűrő.*



1.7. ábra. *WH-10 egyszerűsített wah-wah szűrő.*

kétállású kapcsolóval vannak megoldva, a potenciométerek pedig közös tengelyen futnak.

A kapcsolónak csak az egyik fele van használatban, basszusgitár üzemmódban párhuzamosan rákapcsolja a két kapcsolható kondenzátort (C1, C4) a fix kondenzátorokra (C3, C4), gitár üzemben pedig leválasztja őket. Azaz a négy kondenzátor helyettesíthető két, a kapcsoló állásától függő értékű kondenzátorral.

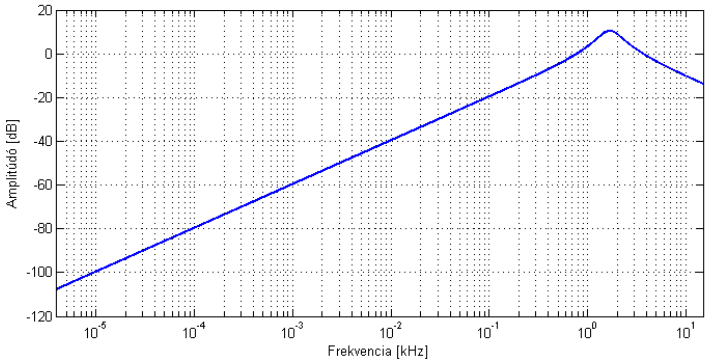
Mindkét potenciométer logaritmikus és egy tengelyen futnak. Tovább egyszerűsítődik a helyettesítő kapcsolás, ha a potenciométerek és a velük sorba kötött ellenállásokat összevonjuk egy ellenállássá.

Ezeket az egyszerűsítési lehetőségeket kihasználva, és az áramkört némiképp átrendezve sokkal átláthatóbb kapcsoláshoz jutunk (1.7. ábra). Az új kapcsolás elemei az előző kapcsolás értékeivel (felülvonással jelezve) kifejezve a 1.2. táblázatban láthatók.

A szűrő átvitele egyszerűen felírható a csomóponti potenciálok módszerével. Ehhez feltételezzük, hogy a rendszerünk lineáris, a műveleti erősítő ideális, ezért a negatív visszacsatolásnak köszönhetően, az erősítő két bemeneti lábának a potenciálja azonos. Mivel a

1.1. táblázat. *Kapcsolat a kapcsolási rajzok között.*

Egyszerűsített	Eredeti
R1	R1' + P1'
R2	R2'
R3	R3' + P2'
C1	C2' vagy C1' + C2'
C2	C3' vagy C3' + C4'



1.8. ábra. *Átvitel ábrázolása MATLAB-ban.*

nem invertáló láb váltakozó jelek szempontjából földelt, ezért az invertáló láb váltakozó jelek szempontjából virtuális földpotenciálra kerül. A megmaradt, név nélküli csomópont potenciálja legyen φ.

Ekkor a keletkező egyenletrendszer a komplex frekvenciatartományban:

$$0 = \frac{\phi - U_{be}}{R_1} + \frac{\phi - 0}{R_2} + \frac{\phi - 0}{1/sC_1} + \frac{\phi - U_{ki}}{1/sC_2} \tag{1.2}$$

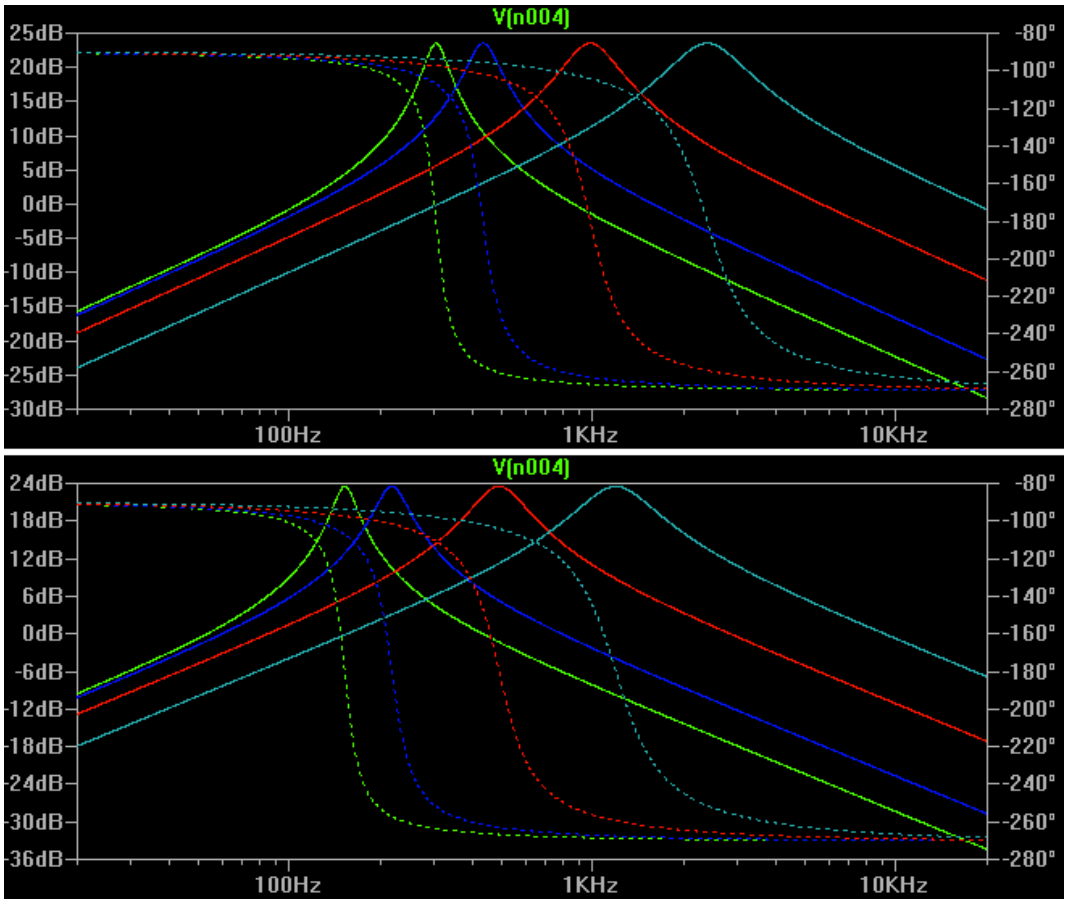
$$0 = \frac{0 - \phi}{1/sC_1} + \frac{0 - U_{ki}}{R3}$$

Az egyenletrendszert megoldva megkapjuk a szűrő átviteli függvényét (F.1):

$$H(s) = \frac{U_{ki}}{U_{be}} = \frac{s(R_2R_3C_1)}{s^2(-R_1R_2R_3C_1C_2) + s(-R_1R_2(C_1 + C_2)) + (-R_1 - R_2)} \tag{1.3}$$

A kapott átvitelt MATLAB-ban ábrázolva (1.8. ábra) egy véletlenszerűen kiválasztott pedállállásnál látható a wah-wah effektekre jellemző háromszög alakú átviteli karakterisztika. A frekvencia kiemelés a pedál mozgatásával csúsztatható jobbra-balra a frekvenciatengely mentén.

LTSpice-ban elemezve a szűrőt [F.2], megnézhetjük, hogy hogyan változik az átvitel a különböző paraméterek változtatásának hatására (1.9. ábra).



1.9. ábra. *Szűrő átvitele gitár és basszusgitár üzemmódban.*

A felső ábrán az effekt gitár, az alsón basszusgitár üzemmódba van állítva. Basszusgitár üzemben a sávszűrő átfogása 152 - 1180 Hz, az erősítése a maximális kiemelésen 23.4 dB. Gitár üzemben az átfogás szélesebb és magasabban is helyezkedik el a frekvenciaskálán. 303 - 2380 Hz, az erősítés a kiemelésnél 23.5 dB.

1.4.5. Keverő áramkör

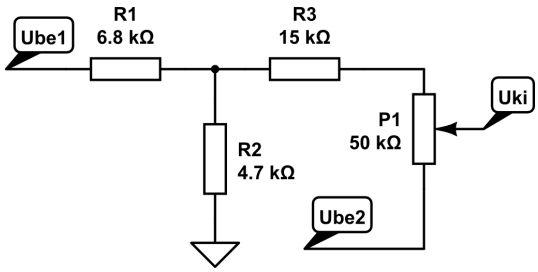
A sávszűrő bemeneti jele egyaránt bemeneti jele egy ellenállásokból felépített kétcsatornás keverő áramkörnek. A keverő másik bemeneti jele a sávszűrő által megszűrt jel. Azaz a keverő áramkör a szűretlen és a szűrt jel arányát módosítja egy potenciométer állását figyelembe véve.

A kapcsolási rajz alapján a keverést egy reziztív hálózat valósítja meg (1.10. ábra).

Egy kétbemenetű egykimenetű hálózatról van szó. A két bemeneten kívül még a potenciométer állása is befolyásolja a kimenet alakulását. E három paraméter egyszerű kezelhetősége érdekében – hasonlóan a sávszűrőhöz – itt is bevezetünk egy paramétert (`c`), ami a potenciométer állását hivatott modellezni. Azaz a kimenet lényegében egy háromváltozós függvénnyel írható le: $U_{ki} = f(U_{be1}, U_{be2}, c)$.

Vegyünk fel új értékeket az egyes ellenállásoknak, és alakítsuk át a modellünknek megfelelően az áramkört!

Az új paraméterekkel felrajzoljuk a kapcsolást a 1.11. ábrára.



1.10. ábra. *WH-10 keverő.*

1.2. táblázat. *Kapcsolat a keverő kapcsolási rajzai között.*

Egyszerűsített	Eredeti
R1	R1'
R2	R2'
R3	R3'
R4	P1'

Lineáris hálózatot feltételezve, a feladat a szuperpozíció segítségével megoldható. A megoldás két lépésben fog kiadódni: hol az egyik, hol a másik bemeneti forrást vesszük figyelembe, miközben a másikat dezaktivizáljuk (feszültségforrás miatt, rövidzárként helyettesítjük), mindkét esetre kiszámoljuk a kimenetet, majd az így kapott két részeredményt összeadjuk. A konkrét számolás egyszerű feszültségosztások alapján számolható:

$$U_{ki} = \frac{(1-c)R_4}{R_3+R_4}\frac{R_2 \otimes (R_3+R_4)}{R_2 \otimes (R_3+R_4)+R_1}U_{be1} + \frac{cR_4+R_3+R_1 \otimes R_2}{R_4+R_3+R_1 \otimes R_2}U_{be2} \tag{1.4}$$

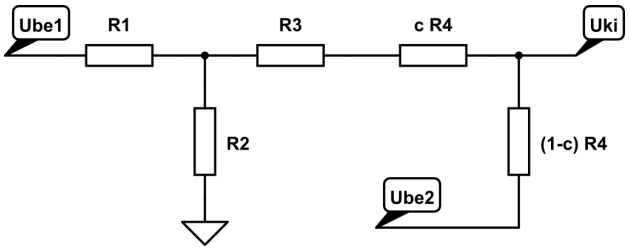
Feloldva a műveleteket, és rendezve az egyenletet:

$$\begin{aligned} U_{ki} = &\frac{(1-c)R_4}{R_3+R_4}\frac{R_2(R_3+R_4)}{R_2(R_3+R_4)+R_1(R_2+R_3+R_4)}U_{be1}+ \\ &+\frac{(R_1+R_2)(cR_4+R_3)+R_1R_2}{(R_1+R_2)(R_4+R_3)+R_1R_2}U_{be2} \end{aligned} \tag{1.5}$$

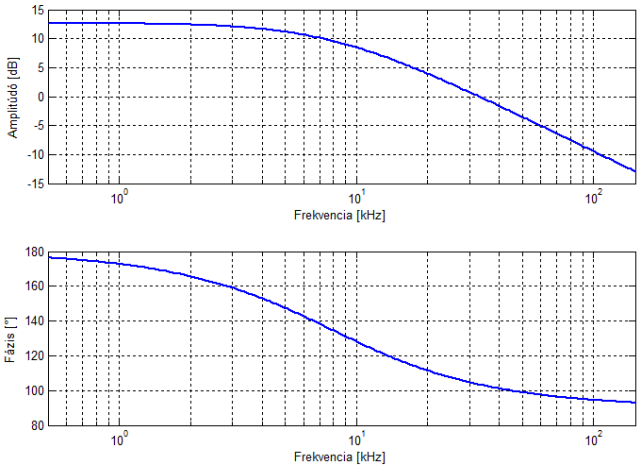
Ha tovább alakítjuk az egyenletet, akkor a következő egyszerűsített alakra jutunk:

$$U_{ki} = (K_1 + K_2(1-c))U_{be1} + (K_3 + K_4 c)U_{be2} \tag{1.6}$$

K_1 és K_3 miatt a kimeneten nem lesz teljesen lekeverve egyik bemenet sem, mindig a kettő valamilyen arány szerinti összege jelenik meg.



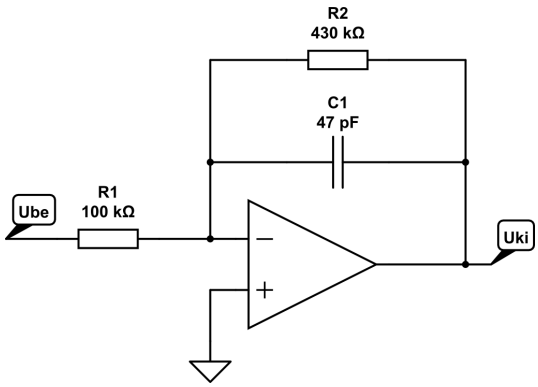
1.11. ábra. *WH-10 keverő egyszerűen kezelhető paraméterekkel.*



1.13. ábra. *Aluláteresztő szűrő átvitele.*

1.4.6. Aluláteresztő szűrő

A szűrési folyamat utolsó része a szűrt és kevert jel felerősítése. Ezért egy műveleti erősítővel megvalósított aluláteresztő szűrő a felelős.



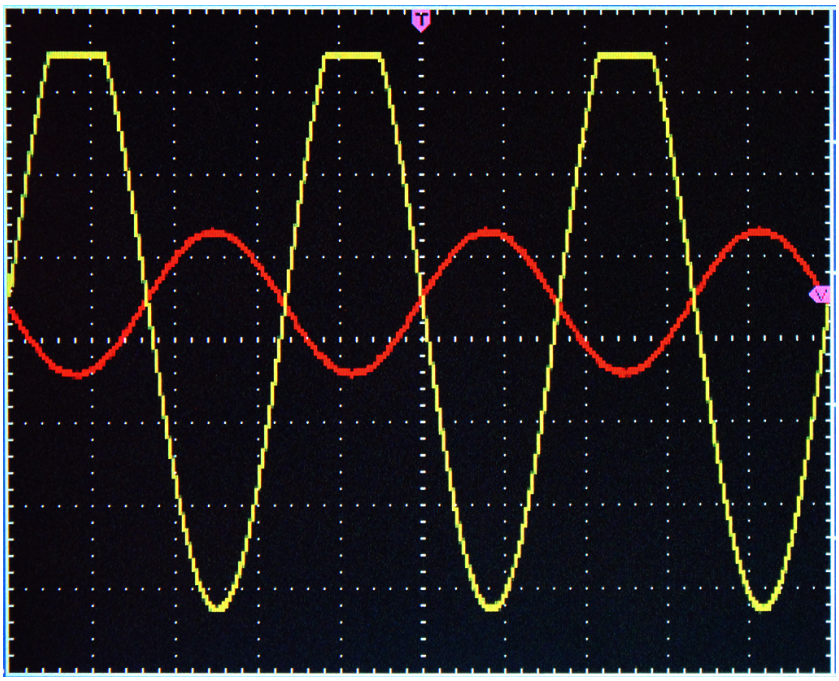
1.12. ábra. *Aluláteresztő szűrő.*

A műveleti erősítőt ideálisnak feltételezve, az átvitel könnyen kiszámolható, ha felírjuk az invertáló alapkapcsolásra vonatkozó alapegyenletet, a visszacsatoló ágban lévő ellenállást és kondenzátort pedig helyettesítjük egy komplex impedanciával:

$$H(s) = \frac{-R_2/R_1}{1 + R_2 C s} \tag{1.7}$$

Ábrázolva az átvitelt (1.13. ábra) látható, hogy a szűrő vágási frekvenciája 8 kHz, ami a felharmonikusokat jelentősebben befolyásolja, mint a be- és kimeneti szűrők, így szerepe nem hanyagolható el.

Az egyenletekből nem látszik, de az áramkör jelentős nemlinearitással bír. Az ok a műveleti erősítőben keresendő. Mint minden integrált áramkörnek, a műveleti erősítőnek is szüksége van tápellátásra a működéséhez. Ez vagy szimmetrikus, azaz pozitív és negatív feszültségszintű táp is rá van kötve, vagy aszimmetrikus, a negatív tápfeszültséget levált-



1.14. ábra. *Műveleti erősítő túlvezérlése.*

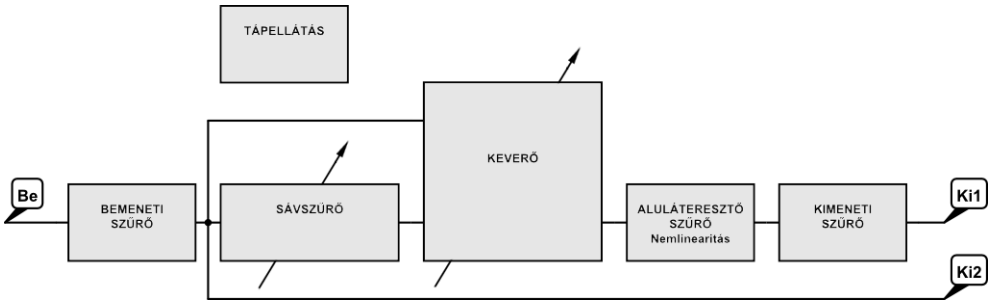
ja a földpotenciál. Ideális esetben az erősítő kimenetének maximuma nem lépheti át a tápforrás értékét. A valóságban azonban örülünk annak is, ha csak megközelíti. Léteznek speciális műveleti erősítők [15], melyek kimenete igen közel képesek kerülni a táphoz, de a mi esetünkben ez nem így van. Az Ibanez hagyományos Texas Instruments LM358 műveleti erősítőket használ a wah-wah pedáljában, melyek 9 V-os aszimmetrikus táplálás esetén a mérések alapján nullától maximum 7.52 V-ig képes a kimenetén feszültséget megjeleníteni. Ha a bemenet hatására a kimenet feljebb kéne menjen, az erősítő élesen levág.

A nemlineáris viselkedést oszcilloszkóppal vizsgáltuk. Az effektre szinuszos gerjesztést (1.78 V_{pp}, 600 Hz szinusz) adva figyeltük a kimenetet úgy, hogy a sávszűrő a gerjesztő jel frekvenciájára volt hangolva és a keverő ezt a sávszűrt jelet maximális amplitúdóval vitte át. Az 1.14. ábrán az oszcilloszkóp mindkét csatornája DC csatolással 1 V/osztásban volt, a piros a műveleti erősítős blokk bemeneti, a sárga a kimeneti jele. Megfigyelhető a 4.5 V-os eltolás a jelek középértékében, valamint a műveleti erősítő fázisfordítása.

Egy hagyományos passzív elektronikás elektromos gitár kimenetén a pengetés erősségétől függően maximum 400 mV amplitúdójú feszültség jelenhet meg. Ebből látható, hogy az effekt nemlineáris viselkedése csak akkor számít jelentősen, ha a WH-10 előtt a jel erősítésen esett át, például egy torzító pedál lett elékötve.

1.5. Áramkör blokkvázlata

A részegységek megismerése után felrajzolhatjuk az effekt működésének a blokkvázlatát (1.15. ábra).



1.15. ábra. *Analóg effekt blokkvázlata.*

1.6. Áramkör bemérése

Függvénygenerátor és oszcilloszkóp segítségével megmérjük az áramkör egyes pontjainak az átvitelét, hogy megbizonyosodjunk, tényleg a valós működést feltételeztük az előzőekben.

A mérési pontok pontos helye a függelékekben (F.4) található, a mérési pontok a kapcsolási rajzon is fel vannak tüntetve.

Három mérést fogunk elvégezni: egy húr erősebb pengetésének megfelelő jelszint (200 mV_{pp}), igen erős akkordpengetésnek (határhelyzet, mikor még éppen nem áll fenn a húrszakadás kockázata) megfelelő jelszint (400 mV_{pp}), túlvezérelt jelszint (640 mV_{pp}). Mindhárom mérés esetén az effekt gitár üzemmódban volt, a bemenetére 600 Hz-es jel került, a sávszűrő ezen a frekvencián emelt ki maximálisan, a keverő maximális effektezett jelátvitelre volt állítva.

A mérési eredményeket táblázatos formában közlöm. A értékeknél szereplő off a 4.5 V-os DC ofszetre utal.

1.3. táblázat. *Egyes mérési pontokon mért jel, a három gerjesztésnek megfelelően.*

	Egy húr	Akkord	Túlvezérelt
A	$200mV_{pp}$	$400mV_{pp}$	$640mV_{pp}$
B	$200mV_{pp} + off$	$400mV_{pp} + off$	$640mV_{pp} + off$
C	$200mV_{pp} + off$	$400mV_{pp} + off$	$640mV_{pp} + off$
D	$200mV_{pp} + off$	$400mV_{pp} + off$	$640mV_{pp} + off$
E	$3.2V_{pp} + off$	$5.36V_{pp} + off$	$7.68V_{pp} + off$
F	$3.2V_{pp} + off$	$5.36V_{pp} + off$	$7.68V_{pp} + off$
G	$3.1V_{pp}$	$5.32V_{pp}$	$7.61V_{pp}$
H	$3.1V_{pp}$	$5.32V_{pp}$	$7.61V_{pp}$
I	$41mV_{pp}$	$100mV_{pp}$	$130mV_{pp}$
J	$172mV_{pp}$	$264mV_{pp}$	$524mV_{pp}$
K	$120mV_{pp}$	$220mV_{pp}$	$328mV_{pp}$
L	$840mV_{pp} + off$	$1.36V_{pp} + off$	$2.64V_{pp} + off$
M	$840mV_{pp} + off$	$1.36V_{pp} + off$	$2.64V_{pp} + off$
N	$3.36V_{pp} + off$	$5.35V_{pp} + off$	$7.76V_{pp} + off$

A mérésekből kiderült, hogy alap használat esetén az effektnél nem érvényesül a nemlinearitás. Az fejezet során feltételezett működés helyesnek bizonyult.

2. fejezet

A digitális effekt

A digitális technika fejlődésével egyre gyorsabb eszközök állnak a rendelkezésünkre, hogy az analóg technológia hátrányait átlépve digitálisan hozzunk létre rendszereket. Egy digitálisan megvalósított rendszer jó közelítéssel időinvariáns, könnyen reprodukálható és olcsó. Azonban analóg rendszerek digitális modellezésénél felmerülhetnek nem várt problémák.

Ebben a fejezetben a már megismert analóg wah-wah pedál funkcionális blokkjait transzformáljuk át digitálisan megvalósítható formába.

2.1. Digitalizálási módszerek

Az analóg világból digitálisba való áttérésre több módszer létezik. Van amelyiket gyakran használják pusztán az egyszerűsége miatt, van ami egyszer használható, máskor nem. Nézzük végig, hogy mik a lehetőségeink!

2.1.1. Impulzus invariáns transzformáció [1]

A transzformáció a mintavételezés modellezésén alapul. Van egy analóg rendszerünk, aminek ismerjük az impulzusválaszát. Ezt a hagyományos mintavételi eljárással ellentétben sávkorlátozás nélkül mintavételezzük, megkapva egy diszkrét jelsorozatot, ami a digitális rendszer impulzusválasza lesz.

Hátránya, hogy a sávkorlátozás nélküli mintavételezés miatt a digitális rendszer frekvenciamenetének periódusossága miatt átlapolódás lép fel. Ez az átlapolódás aluláteresztő jellegű rendszereknél elhanyagolható mértékű, ezért az ilyen típusú rendszerek digitális modellezésére alkalmas a legjobban a módszer.

2.1.2. Előre és hátralépő Euler [16]

Ez a két áttérési eljárás akkor használható, ha az analóg rendszerünk állapotváltozós alakjából akarunk digitális rendszert létrehozni. Lényege, hogy az állapotváltozók differenciálegyenletét a digitális rendszer lépéseiként oldjuk meg közelítéssel.

Mindkét esetben az analóg rendszer válaszát lépésenként közelítjük digitálisan. Az így kapott átvitel nem lesz tökéletesen azonos az eredeti rendszer átvitelével, néha meg sem

közelíti azt. Ilyen esetben célszerű a számítás finomságán állítani, nagyobb mintavételi frekvenciával, kisebb lépésközökkel számolni a rendszer válaszát.

Stabilitás szempontjából az hátralépő Euler módszer stabil analóg rendszerből stabil digitális rendszert képez, azonban az előrelépő Euler nem. A stabilitás nem garantált digitális tartományban, viszont számítási igényt tekintve sokkal előnyösebb, mint a hátralépő Euler, mivel a lépések megoldása explicit módon azonnal kiadódik. Hátralépő Euler esetén minden lépésnél egy egyenletet kell megoldani valamilyen egyenlet megoldási módszerrel, például Newton-Rapson algoritmus vagy zérushelykereséses módszer.

Ezen két módszer előnye, hogy nemlineáris rendszerek esetén is használhatóak.

2.1.3. Bilineáris transzformáció [1]

A legelterjedtebben használt transzformációs eljárás. Az analóg rendszer átviteli függvényét transzformálja át egy számítási szempontból kedvező, z-tartománybeli alakra.

$$s = \frac{2}{T}\frac{z-1}{z+1} \tag{2.1}$$

A képletben T a mitavételi idő. A behelyettesítést használva s tartományból z tartományba kerülünk úgy, hogy az s tartománybeli bal félsíkot az egységkörön belülre, a jobb félsíkot az egységkörön kívülre, a képzetes tengelyt az egységkör körvonalára transzformálja. Ebből látható, hogy stabil s tartománybeli rendszert, stabil z tartománybeli rendszerré transzformál.

A módszer egyszerűségének azonban ára van. Frekvenciatartományban a folytonos időbeli [0,∞] tartományt [0,f_s/2] tartományba transzformálja, jelentősen torzítva a frekvenciamenetet. Ez aluláteresztő jellegű karakterisztikáknál a legsúlyosabb, hiszen az analóg rendszertől eltérően, aminek az átvitele $f = f_s/2$ frekvencián nem nulla, addig a bilineárisan digitalizált változatában az átvitel nullába zuhan.

Általánosságban elmondható, hogy ha bilineáris transzformáció segítségével akarunk digitális szűrőt tervezni, akkor a kívánt pontos frekvenciamenet eléréséhez az analóg modellünket a transzformáció torzítását figyelembe véve módosítani kell.

Ha azonban konkrét analóg szűrőáramköröket akarunk digitálisan megvalósítani, akkor más módszerek után kell nézni.

2.2. Analóg blokkok digitalizálása

Az átalakítások során haladjunk sorban az analóg blokkvázlatnak megfelelően! (1.15. ábra)

2.2.1. Bemeneti és kimeneti szűrő digitalizálása

Láthattuk, hogy a be- és kimeneti szűrők vágási frekvenciája jóval meghaladja az emberi hallásküszöböt. A digitális audiorendszerek (kivéve az 96 kHz-es mintavételen működő rendszerek) jó közelítéssel az emberi hallás tartományára vannak lekorlátozva olyan módon, hogy a mintavételi frekvenciájuk kétszerese az ember által elméletileg hallható maximális frekvenciájú hangoknak. E szint felett egyszerűen nem jelenítenek meg frek-

venciakomponenseket. Ebből kifolyólag a bemeneti és kimeneti szűrő elhagyásával nem követünk el hibát, hiszen azt a tartományt, ahol a hatásuk érvényesülne nem halljuk, és a digitális audiorendszerek sem törődnek már vele.

2.2.2. Sávszűrő digitalizálása

A 1.4.4. fejezetben kiszámoltuk a wah-wah effektben használt sávszűrő átvitelét. Az egyszerűbb átláthatóság érdekében nevezzük el az 1.3 egyenlet együtthatóit!

$$H(s) = \frac{b_1 s}{a_2 s^2 + a_1 s + a_0} \tag{2.2}$$

Alkalmazzuk a bilineáris transzformációt!

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

$$H(s) \Longrightarrow H(z) = \frac{b_1 \frac{2}{T}\frac{z-1}{z+1}}{a_2(\frac{2}{T}\frac{z-1}{z+1})^2 + a_1\frac{2}{T}\frac{z-1}{z+1} + a_0} \tag{2.3}$$

Némi rendezés után megkapjuk a z tartománybeli átvitelt:

$$H(z) = \frac{(\frac{2b_1}{T})z^2 + (-\frac{2b_1}{T})}{(\frac{4a_2}{T^2} + \frac{2a_1}{T} + a_0)z^2 + (-\frac{8a_2}{T^2} + 2a_0)z + (\frac{4a_2}{T^2} - \frac{2a_1}{T} + a0)} \tag{2.4}$$

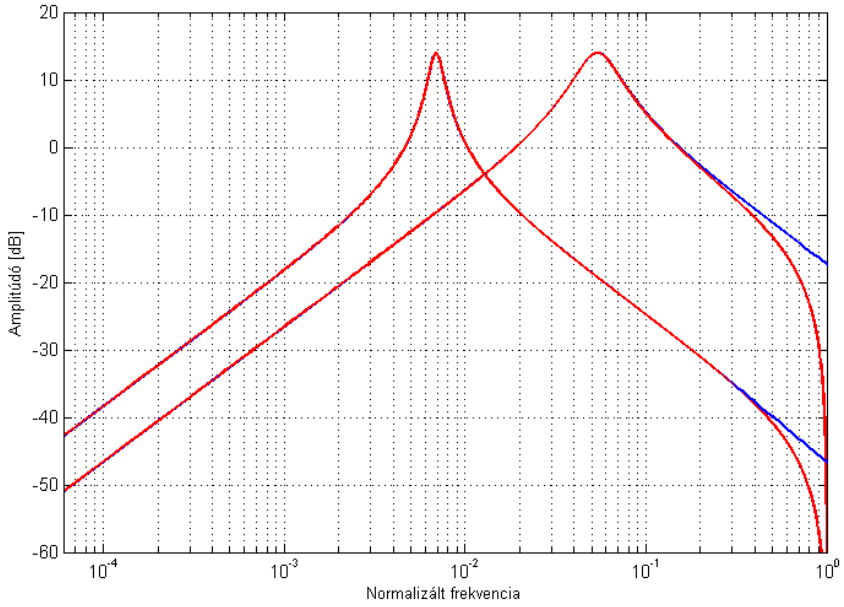
Áttekinthetőbb formát kapunk, ha az együtthatókat helyettesítjük:

$$H(z) = \frac{z^2 B_0 + B_2}{z^2 A_0 + z A_1 + A_2} = \frac{B_0 + B_2 z^{-2}}{A_0 + A_1 z^{-1} + A_2 z^{-2}} \tag{2.5}$$

A kapott rendszer egy nagyon könnyen és gyorsan megvalósítható IIR szűrő lett, melynek átvitelét egy paraméter (a pedál állása) képes szabályozni. Ez az egy paraméter az 1.3 képletben R_1 és R_3 értékeit változtatja, ami hatására az átvitel kiemelési frekvenciája változni fog.

Nézzük meg, használható-e ez a megoldás! Ábrázoljuk az analóg és a digitális rendszer átvitelét!

Az 2.1. ábrán az analóg rendszer átvitele kékkel, a digitális rendszer átvitele pirossal van ábrázolva. Az ábra feltételezi, hogy a digitális rendszer mintavételi frekvenciája $f_s = 44100$ Hz, azaz $f_s/2 = 22050$ Hz. Jól látható, hogy a bilineáris transzformáció milyen jól modellezi az analóg átvitelt kisfrekvenciákon, azonban ahogy a frekvencia egyre jobban megközelíti $f_s/2$-t úgy egyre jobban érvényesül a transzformáció torzítása. $f = f_s/2$ frekvencián a digitális rendszer átvitele nullába zuhan. Ez nem elfogadható. Az átvitel túl



2.1. ábra. *Analóg és digitális átvitel eltérése.*

torz ahhoz, hogy hitelesen modellezze az analóg szűrő viselkedését!

Keressünk megoldást a problémára!

A bilineáris transzformáció tulajdonságaiból kiindulva tudjuk, hogy a $[0,\infty]$ tartományt a $[0,f_s/2]$ tartományba sűríti. Triviális megoldásnak tűnik, hogy a mintavételi frekvenciát növeljük addig, amíg az átvitel nem lesz megfelelő a számunkra megkívánt tartományon. Jó ötletnek tűnik, de mi van akkor, ha az eszköz, amin megvalósítjuk a digitális rendszerünket nem tudja megváltoztatni a mintavételi frekvenciáját?

Ha a hardver nem képes változtat a mintavételi frekvencián, akkor szoftveresen kell a problémát megoldani. A mintavételi frekvenciát meg tudjuk többszörözni interpolációval, majd a kényes szűrési szakasz elvégeztével decimálással vissza tudunk térni az eredeti mintavételi frekvenciára. Ezt a módszert részletesebben a 4. fejezetben fogjuk tárgyalni.

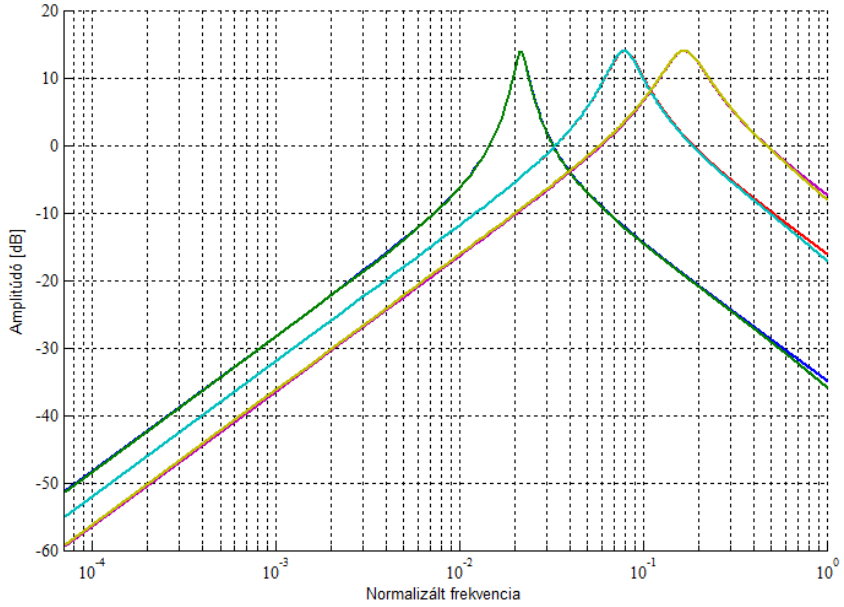
A módszer egyetlen problémája, hogy rendkívüli módon képes megnövelni a számítási szükségleteket, ezért csak ott érdemes használni, ahol már tényleg nincs más megoldásra lehetőség.

Szerencsére a bilineáris transzformáció problémájával már sokan szembekerültek, és elég jól dokumentált megoldáshalmazból válogathatunk.

Célunk az, hogy a lehető legjobb eredményt érjük el, optimális kivitelezésben. A legjobb lenne, ha csupán a szűrő paramétereinek a változtatásával érnénk el a kívánt javulást.

Sophocles J. Orfanidis cikkében [21] pont erre a problémára ad megoldást.

A módszere a digitális ekvalizereket hivatott hozzáigazítani az analóg megfelelőikhez,



2.2. ábra. *Analóg és digitális átvitel eltérése módosított digitális szűrő esetén.*

ugyanis a bilineáris transzformáció következtében a digitális szűrők túl gyorsan vágtak az analógokhoz képest a mintavételi frekvencia felénél. Ehhez a tervezéshez felhasznált paraméterek: $\{f_s, f_0, \Delta f, G_0, G_1, G, G_B\}$, ahol f_s a mintavételi frekvencia, f_0 az kiemelési vagy elnyomási frekvencia, Δf a kiemelés szélessége, G_0 az erősítés DC-n, G_1 az erősítés $f_s/2$-nél, G az erősítés f_0 helyen, G_B pedig a kiemelés sávszélességének az erősítése, ott ahol Δf-et mértük.

A cikk szerint, az eddigi módszerekben mindenhol feltételezték, hogy G_0 és G_1 azonos, ez azonban bilineáris transzformációt alkalmazva torz eredményhez vezet.

Ebben a módszerben megengedjük, hogy G_0 eltérjen G_1-től, pontosabban G_1-et pont akkorára választjuk, mint az analóg rendszer átvitele az $f_s/2$ pontban. Ez a feltétel együtt a többivel $(G_0, f_0, G, \Delta f, G_B)$ öt szabadsági fokra ad megkötést a digitális szűrő öt paraméterére. Az így megtervezett szűrő átvitelben annyira idomul az analóg szűrő átviteléhez, amennyire csak lehet.

A cikk egy MATLAB függvényt is megad, aminek a megfelelő paramétereket beadva, kiszámolja nekünk az átvitelben pontosan illeszkedő szűrőt [F.5].

A módszert használva az 2.2. ábrán látható átvitelt kapjuk meg. Látható, hogy az hasonlóság közel tökéletes!

Az új átviteli függvény továbbra is másodfokú marad, csupán a számlálójában megjelenik az elsőfokú tag is.

$$H^*(z) = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{A_0 + A_1 z^{-1} + A_2 z^{-2}} \tag{2.6}$$

Digitális feldolgozás szempontjából nincs nagy különbség, mintánként eggyel több szorzást és összeadást kell elvégezni, és mégis egy sokkal jobban teljesítő szűrőt kapunk, ami szinte tökéletesen modellezi az analóg megfelelőjét.

A paraméterezés megvalósításáról bővebben a függelékekben lehet olvasni [F.6].

2.2.3. Keverő áramkör digitalizálása

A blokkvázlaton a következő egységünk a keverő áramkör. A rezisztív hálózat átvitelét már felírtuk a 1.4.5. fejezetben, most nézzük meg, hogy ez hogyan valósítható meg digitálisan.

A kapott átvitel egy ehhez hasonló alakban írható fel:

$$U_{ki} = (K_1 + K_2(1-c))U_{be1} + (K_3 + K_4 c)U_{be2} \tag{2.7}$$

Itt egyszerű dolgunk van, hiszen a rezisztív hálózatot frekvenciafüggetlennek feltételezhetjük az audio tartományban, ezért az analóg áramkör egyenlete közvetlenül felhasználható a digitális modellben, így nem szükséges transzformáció.

2.2.4. Aluláteresztő szűrő digitalizálása

Blokkvázlatunk utolsó eleme az aluláteresztő szűrő. Ha nem lenne a nemlineáris tulajdonsága, akkor hasonlóan a sávszűrőhöz, egyszerű módosított bilineáris transzformációval megoldhatnánk a digitális modellezést.

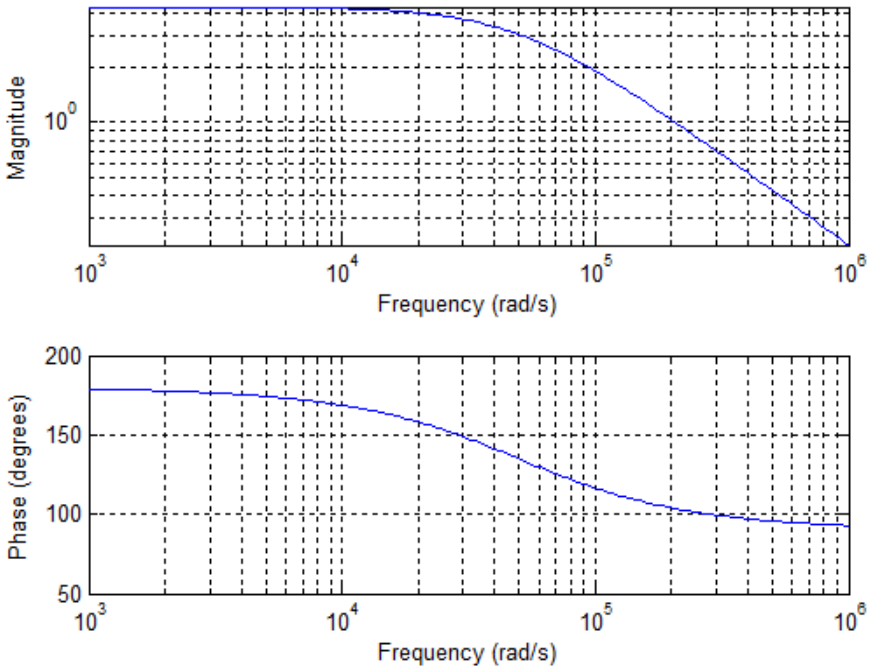
A nemlinearitás miatt plusz felharmonikusok keletkeznek a szűrés közben a műveleti erősítő levágása miatt. Megtehetnénk, hogy az egyszerűen megvalósított szűrő kimenetét egyszerűen korlátozzuk, egy határ után nem engedjük tovább nőni azt, de ezzel egy lehetséges hibaforrást vinnénk be a rendszerbe. Ha a keletkező felharmonikusok a frekvenciatengelyen átlógnának a mintavételi frekvencia felénél túlra, akkor az átlapolódás következne be, a túllógott komponensek belapolódnának, és elrontanák az effekt hangzását.

Ezt a problémát digitális rendszereknél a mintavételi frekvencia változtatásával szokták megoldani. Ahogy az előbb is írtam, a módszer pontos működését a 4. fejezetben fogjuk tárgyalni.

Ha növeljük a mintavételi frekvenciát, akkor úgymond nagyobb helyük marad a felharmonikusoknak, ahol még nem lapolódnak át. A gyakorlatban négyszeres és nyolcszoros közötti túlmintavételezést szoktak alkalmazni, ezzel már kielégítő eredményt kapva.

Egy nemlineáris áramkört készülünk modellezni, azaz nem használhatóak a jól megszokott lineáris áramkörökre használatos módszerek. Nem létezik átviteli függvény, nincs impulzusválasz. Egy olyan modellt kell keresni, ami képes modellezni a nemlinearitást.

Ehhez az állapotváltozós leírás használható. Ha ezt felírjuk a 1.12. ábrán látható kap-



2.3. ábra. *Aluláteresztő szűrő átvitele.*

csolásra, akkor a következő alakot kapjuk:

$$\begin{aligned} U_c' &= [\tfrac{-1}{R_2C}]U_c + [\tfrac{-1}{R_1C}]U_{be} \\ U_{ki} &= U_c \end{aligned} \tag{2.8}$$

Megkaptuk az állapotváltozós leírást, amiből a négy mátrix:

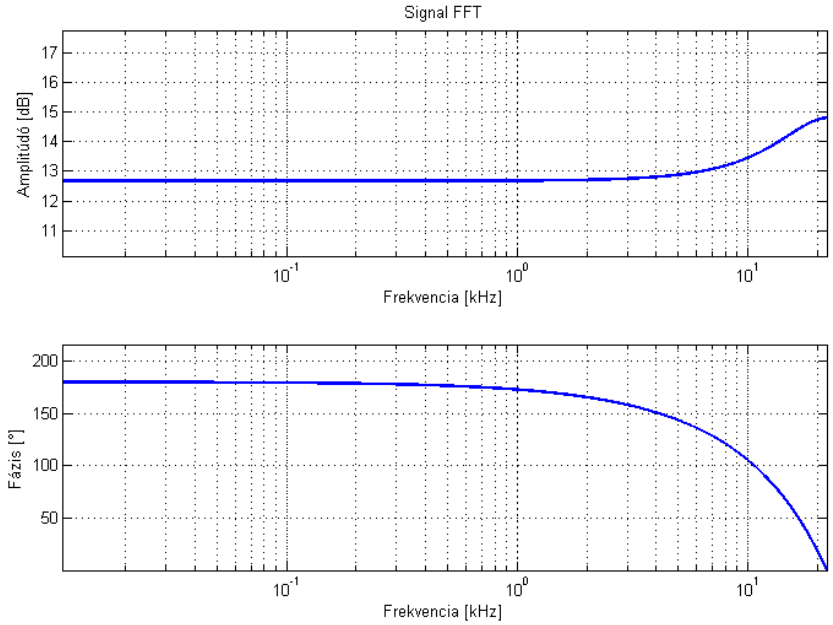
$$\begin{aligned} \mathbf{A} &= \tfrac{-1}{R_2C} \quad & \mathbf{B} &= \tfrac{-1}{R_1C} \\ \mathbf{C} &= 1 & \mathbf{D} &= 0 \end{aligned} \tag{2.9}$$

A nemlinearitás hozzáadása előtt bizonyosodjunk meg róla, hogy helyes-e a kiszámolt rendszer. Ábrázoljuk az átvitelét (2.3. ábra)!

A következő lépés, hogy az analóg módon leírt rendszert digitálisba áttranszformáljuk. Ehhez az előrelépő Euler módszert használjuk. Lineáris esetben hasonló a két módszer számításigénye, azonban az előrelépő Euler módszernek a fázismenete jóval inkább hasonlított az analóg aluláteresztő szűrőéhez.

Tudjuk, hogy a deriválás s tartományban s-sel való szorzásnak felel meg. Átírva az egyenletet:

$$sU_{ki} = (\frac{-1}{R_2C})U_{ki} + (\frac{-1}{R_1C})U_{be} \tag{2.10}$$



2.4. ábra. *Digitálisan megvalósított szűrő alap mintavételi frekvencián.*

Alkalmazva az előrelépő Euler módszernél használatos helyettesítést:

$$s = \frac{1 - z^{-1}}{Tz^{-1}} \tag{2.11}$$

$$\frac{1 - z^{-1}}{Tz^{-1}} U_{ki} = (\frac{-1}{R_2 C}) U_{ki} + (\frac{-1}{R_1 C}) U_{be} \tag{2.12}$$

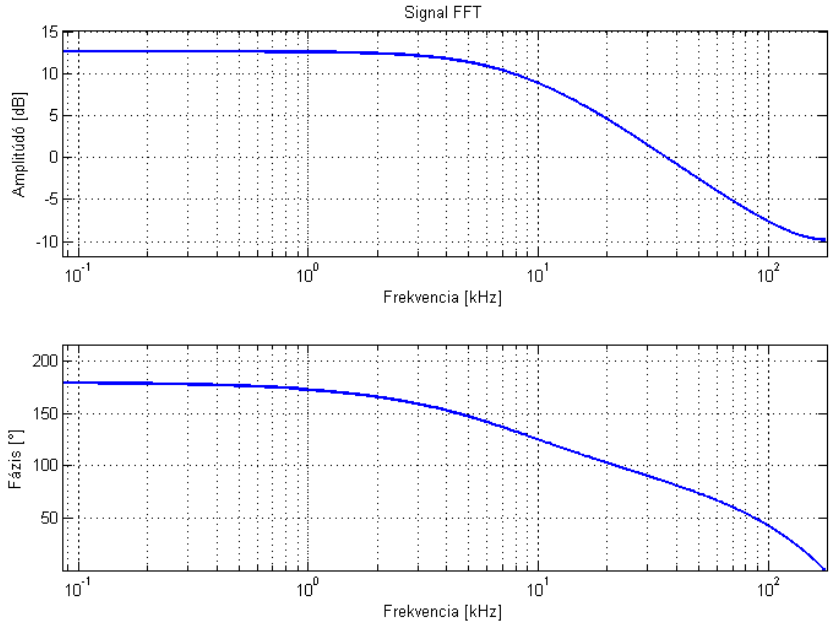
$$U_{ki} = (1 + \mathbf{A}T) U_{ki} z^{-1} + \mathbf{B} T U_{be} z^{-1} \tag{2.13}$$

$$U_{ki}[n] = (1 + \mathbf{A}T) U_{ki}[n-1] + \mathbf{B} T U_{be}[n-1] \tag{2.14}$$

Ezek után a rendszerünket már meg tudjuk valósítani programozottan!

```
% x a bemeneti vektor
% y a kimeneti vektor
% N a bemenet hossza
% A,B az állapotváltozós leírásban szereplő együtthatók
% T a lépésköz

% kezdeti érték beállítása
y(1) = 0;

% szűrőt megvalósító ciklus
for n = 2:N
   y(n) = (1+A*T)*y(n-1) + (B*T)*x(n-1);
end
```

A rendszer még lineáris, azért ha a bemenetére egy impulzust adunk, a választ Fourier transzformálva megkapjuk az átvitelét. Ellenőrizzük le, hogy az algoritmusunk a kívánt aluláteresztő szűrőt valósítja-e meg! Ábrázoljuk az átvitelt! (2.4. ábra)



2.5. ábra. *Digitálisan megvalósított szűrő nyolcszoros mintavételi frekvencián*

Az átvitel nem pont olyan, mint amilyet szeretnénk. Aluláteresztőnek kellene lennie, de felüláteresztő jellegű lett. Mit lehet ilyenkor tenni?

A lépésenkénti megoldási módszerek gyakran nem adják vissza a kívánt rendszertulajdonságokat, a valós kimenettől eltérőt produkálnak. Ilyenkor sűríteni kell a algoritmus lépéseit, nagyobb mintavételi frekvencián kell futtatni a szűrőt.

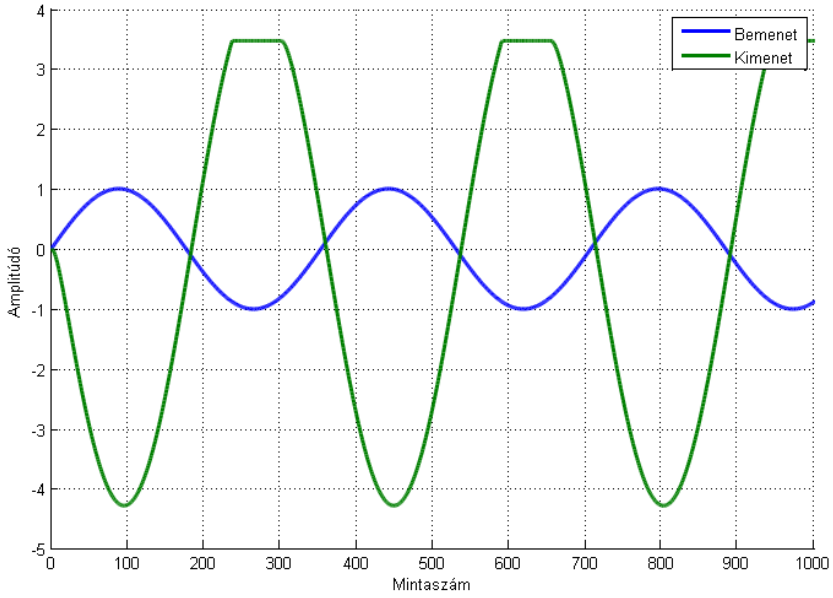
A nemlinearitás miatt úgy terveztünk, hogy valószínűleg szükség lesz túlmintavételezésre az átlapolódások elkerülése érdekében, de most a helyes szimuláció miatt ez már elkerülhetetlen. Mint azt korábban említettük, általánosságban négy és nyolcszoros túlmintavételezést szokás alkalmazni audio feldolgozásban. Első kísérletként próbáljuk meg lefuttatni az algoritmusunkat nyolcszoros mintavételi frekvencián!

Magában az algoritmusban semmi nem fog változni, csupán a T konstans értéke, ami nyolcadára csökken.

Lefuttatva az algoritmust és ábrázolva azt a 2.5. ábrán szereplő átvitelt kapjuk. A feltételezés helyes volt, a sűrűbb mintavétel valóban a várt viselkedést hozta. Az aluláteresztő szűrő vágási frekvenciája 8 kHz környékére esik, ezzel hűen modellezi a műveleti erősítős áramkör átvitelének amplitúdómenetét.

Magát a nemlinearitást a lehető legegyszerűbb módon fogjuk modellezni. Feltételezzük, hogy a műveleti erősítő kimenete nagyon hirtelen, szinte azonnali módon kerül telítésbe.

Az algoritmusban minden kiszámolt kimeneti mintára ellenőrizni kell, hogy nem lépte-e át a telítődési tartományt. Felül és alul is vizsgálódni kell. A felső határ a telítődési határ, alsó határ analóg esetben a földpotenciál, digitális esetben pedig az analóg földnek megfelelő számérték. Az analóg áramkörben a jelet feljebb csúsztatták egy DC szinttel, hogy ne kelljen szimmetrikus táplálást kiépíteni a műveleti erősítőkhöz, mi ezt digitálisan



2.6. ábra. *Nemlineárisan feldolgozott jel időfüggvénye. Bemenet és kimenet.*

nem a mintasorozat, hanem az alsó ellenőrzési határ lejjebbvitelével érjük el.

A vágási határok pontos beállításához mindenképpen összehasonlító mérések szükségesek, hiszen nem tudhatjuk pontosan, hogy az eszköz AD átalalakítója adott feszültségértéket mekkora numerikus értékké alakít át.

A mérésekkel beállított paraméterekkel megszűrt jel szinte teljesen azonos az előző fejezetben oszcilloszkóppal mért jellel (1.14. ábra). Az MATLAB-ban lefuttatott majd ábrázolt algoritmuson jól látható, hogy egy határon túl nem engedjük tovább nőni a jelet.

Ehhez az algoritmust csupán kis mértékben kell módosítani. Az új kimenet kiszámítása után meg kell vizsgálni, hogy az nem lépte-e túl a megszabott határt.

```
% x a bemeneti vektor
% y a kimeneti vektor
% N a bemenet hossza
% A és B az állapotváltozós leírásban szereplő együtthatók
% T a lépésköz
% MAX és MIN az eredeti effekthez hangolt értékek

% kezdeti érték beállítása
y(1) = 0;
% szűrőt megvalósító ciklus
for n = 2:N
   y(n) = (1+A*T)*y(n-1) + (B*T)*x(n-1);
   if y(n) > MAX
      y(n) = MAX;
   end
   if y(n) < MIN
      y(n) = MIN;
   end
end
```

2.2.5. Interpoláció

A nemlineáris feldolgozás előtt meg kell többszörözni a mintavételi frekvenciát az átlapolódás mértékének csökkentése érdekében. Ehhez interpolációt használunk.

Interpoláció esetén a célunk, hogy a mintavételi frekvenciát megnöveljük. Ehhez tudni kell, hogy a mintavételi frekvencia értelmezése szerint azt mondja meg, hogy egy jelből másodpercenként hányszor veszünk mintákat. Növelés esetén sűrűbben kell mintavételezni, azaz a mintasorozatunkat ki kell bővíteni további mintákkal.

Erre egy triviális megoldás, hogy ha az adott mintasorozatot N-szeresére akarjuk felinterpolálni, akkor N-1 darab nullát be kell szúrni a mintasorozat meglévő mintái közé. A jel spektrálisan nem változik, viszont egységnyi időtartam alatt N-szer több mintánk lesz, megnőtt a mintavételi frekvencia. f_s mintavételi frekvenciából $f'_s = 8f_s$ mintavételi frekvencia lett (2.7. ábrán $X(f)$ jelből $X_1(f)$ jel lett).

A jel spektrálisan nem változott, ezért a interpolált jelünk mostani állása szerint spektrálisan megegyezik a kiindulási jellel, ami a diszkrét jelekre jellemzően periodikus a mintavételi frekvenciára f_s-re. A mintavételi frekvencia növelésével szeretnénk, ha a jel periodikus lenne az új mintavételi frekvencia (f'_s) szerint. A köztes komponenseket ki kell szűrni.

Erre egy, az eredeti mintavételi frekvencián $f_s/2$ vágási frekvenciájú aluláteresztő szűrőt kell használni úgy, hogy az már ne engedjen át komponenseket $f_s/2$ után (2.7. ábrán $H_{int,8}(f)$ szűrő).

Fontos, hogy a szűrő áteresztő tartományában nem egységnyi az átvitel. Az alapjelnek és az interpolált jelnek meg kell egyezzen az energiatartalma, azonban mi minden alap minta közé egy nullát szúrtunk, csökkentettük az energiatartalmát. N-szeres interpolációnál M darab mintából N*M darab minta lesz, az energiatartalom N-ed részére csökken. A szűrő ezt képes visszaállítani, ha az áteresztő tartományában az átvitel N-szeres.

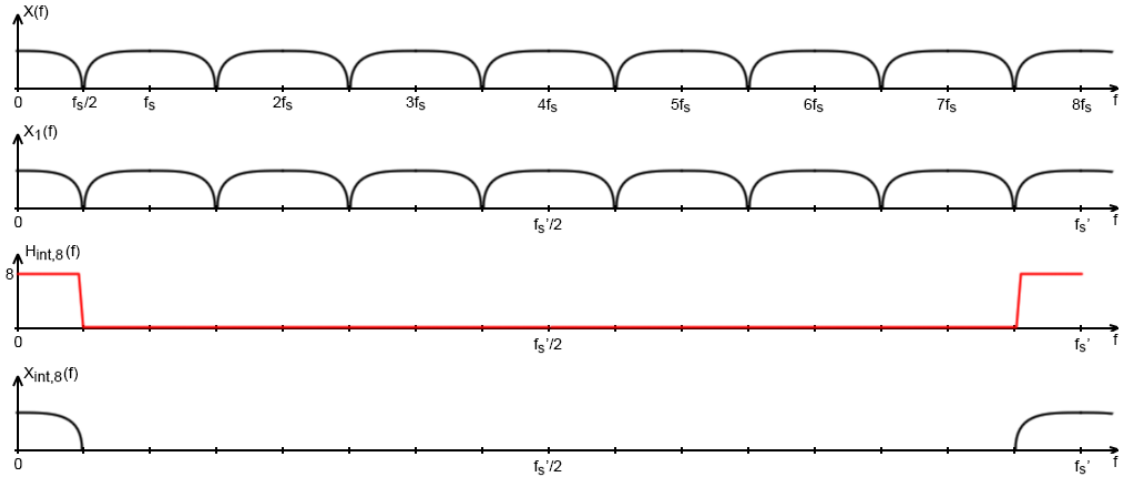
Ideális esetben a szűrés után megkapjuk az eredeti jelünk felinterpolált megfelelőjét, ami az új mintavételi frekvencia szerint periodikus, minden más tulajdonságában azonos maradt (2.7. ábrán $X_{int,8}(f)$ jel).

Ha megnézzük a felinterpolált jel időfüggvényét, akkor láthatjuk, hogy az interpoláló szűrő a beszúrt nulla mintákat kitöltötte, mintha az eredeti jelre ránagyítottunk volna és az új mintavételi frekvencia szerint mintavételeztük volna (2.8. ábra).

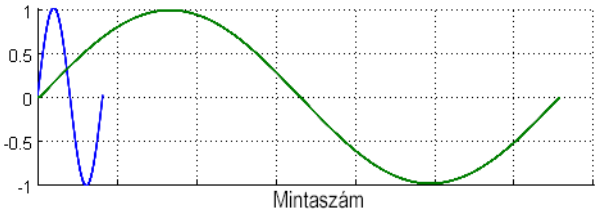
Kérdés, hogy hogyan valósítsuk meg ezt a gyakorlatban?

Követhetjük a triviális módszert, beszúrjuk a nullákat, aztán az új, kibővített mintasort megszűrjük az interpoláló szűrővel, de ez nem tartozik a leghatékonyabb implementálások közé.

Az interpoláló szűrőnk elég meredek levágással rendelkezik, és egy igen szűk tartományban ereszt csak át. Ha ezt egy FIR szűrővel valósítjuk meg, akkor az csakis egy nagy fokszámú szűrő lehet. Nagy fokszámú FIR szűrőt már az eredeti mintavételi frekvencián is problémás futtatni, felinterpolált mintaszámra pedig esetenként - hardvertől függően - lehetetlen. A szűrőt a nullákkal kibővített mintákra futtatjuk, ezért sok nullával való szor-



2.7. ábra. *Nyolcszoros interpoláció egy lépésben.*



2.8. ábra. *Eredeti és interpolált jel mintákban összehasonlítva.*

zással dolgoztatjuk feleslegesen a jelfeldolgozó egységet. Rajzoljunk fel egy ilyen módszerű szűrést N=4-szeres interpolációra!

Az 2.9. ábrán egy négyszeres interpolálást végző tipikus FIR szűrő megvalósítást látunk. Felül a téglalapok és a pontok jelentik a beérkező jelsorozatot. Az eredeti mintákat jelképező téglalapok közé három pontot szúrtunk be, amik a nullákat szimbolizálják. Jobb oldalon van a legrégebbi minta, bal oldalon a legfrissebb. Az interpoláló szűrő egyszerűbb bemutatása kedvéért a konvolúciót a szűrőegyütthatók folyamatos balra csúsztatásával végezzük el. A felesleges nullával való szorzásokat halványan jelöli az ábra. Ezeket a szűrőnek nem kell elvégeznie.

2.9. ábra. *Négyszeres interpoláció bemutatása.*

Az ábra alapján megfigyelhetünk bizonyos szabályosságokat:

- N=4-szeres interpolálást végzünk
- K=8 tap-es interpoláló szűrőt használunk
- minden bemeneti mintát négyszer (N) használunk fel a szűrések során
- minden kimeneti mintához két (K/N) szűrőegyüttható val szorzunk
- ezek a kettes szűrők négyesével (N) ismétlődnek
- a nulla mintákat egyáltalán nem használtuk fel, beszúrásuk felesleges

Összegezve: egy interpoláló szűrő felbontható N darab kisebb szűrőre úgy, hogy az egyes szűrők K/N együtthatóból állnak, első elemeik indexe 0-tól megy (N-1)-ig, a következő elemeik indexe N-es lépésekkel nő:

$$\underbrace{\begin{matrix} h_0 & h_{0+N} & h_{0+2N} & \dots \\ h_1 & h_{1+N} & h_{1+2N} & \dots \\ h_2 & h_{2+N} & h_{2+2N} & \dots \\ \vdots & \vdots & \vdots & \vdots \\ h_{N-1} & h_{N-1+N} & h_{N-1+2N} & \dots \end{matrix}}_{K/N} \tag{2.15}$$

A fenti mátrix minden sora megfelel egy kisebb szűrőnek. Egy új bemeneti minta cirkuláris bufferbe írásakor, mindegyik kisebb szűrővel el kell végezni a konvolúciókat a bufferen. Mindegyik szűrés eredménye egy újabb interpolált mintát ad ki.

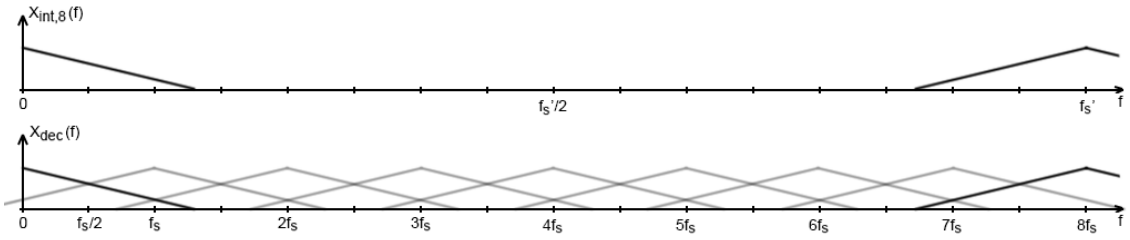
N-szeres interpolálás esetén a szűrő N darab kisebb szűrőre bomlik. Egy új bemeneti mintát mind az N darab szűrővel meg kell szűrni, azaz egy új bemeneti mintára keletkezik N darab kimeneti minta.

Az ilyen módon megvalósított interpoláló szűrőt *polifázisú szűrő*nek nevezzük [1].

Mit nyerünk ezzel a struktúrával? Először is a nullák beszúrása feleslegessé válik, nem kell vele foglalkozni és időt vesztegetni rá. Másrészt K/N hosszú szűrővel kell szűrni, ami szintén eléggé felgyorsítja a feldolgozást.

Ha K osztható N-nel, akkor a konstrukció optimálisnak mondható, a kialakuló kisebb szűrőkből álló mátrix minden elemét hasznosítjuk a szűrés alatt. Ha ez nem teljesül, akkor a szűrőt ki kell egészíteni nullákkal, amíg az interpoláció mértéke nem lesz osztható a szűrő együtthatóinak a számával. A kisebb szűrőkből álló mátrix nullákat is fog tartalmazni, amikre algoritmus szinten fel lehet készülni, hogy ne hajtsunk végre felesleges nullával való szorzásokat.

Ez a módszer csak FIR típusú szűrőkkel valósítható meg, IIR szűrőknél ezt nem tehetjük meg: be kell szúrni a nullákat, majd minden mintára végig kell futtatni a szűrőt. Előnye viszont, hogy az IIR szűrők fokszáma tipikusan sokkal kisebb, mint az azonos meredekségű



2.10. ábra. *Szűrés nélküli decimálás.*

FIR szűrőké.

Az interpolálás még tovább optimalizálható, ha több lépésben végezzük el [24]. Enyhébb specifikációjú és kisebb fokszámú szűrőket lehet alkalmazni. A legélesebb vágás az első lépésben szükséges, utána egyre lankásabb szűrőket lehet alkalmazni. Az optimális interpoláció akkor érhető el, ha FIR és IIR szűrőket is egyaránt felhasználunk a megvalósítás során: több lépés vagy FIR (polifázisú) vagy IIR szűrők alkalmazásával [1].

2.2.6. Decimálás

Nemlineáris szűrés után a felharmonikusokban gazdag, nagy mintavételi frekvenciás jelet vissza kell alakítani az alap mintavételi frekvenciára. Mint azt már az interpolálásnál láttuk, felfelé minták hozzáadásával mehetünk mintavételi frekvenciában, akkor lefelé ezek szerint minták elhagyásával mehetünk.

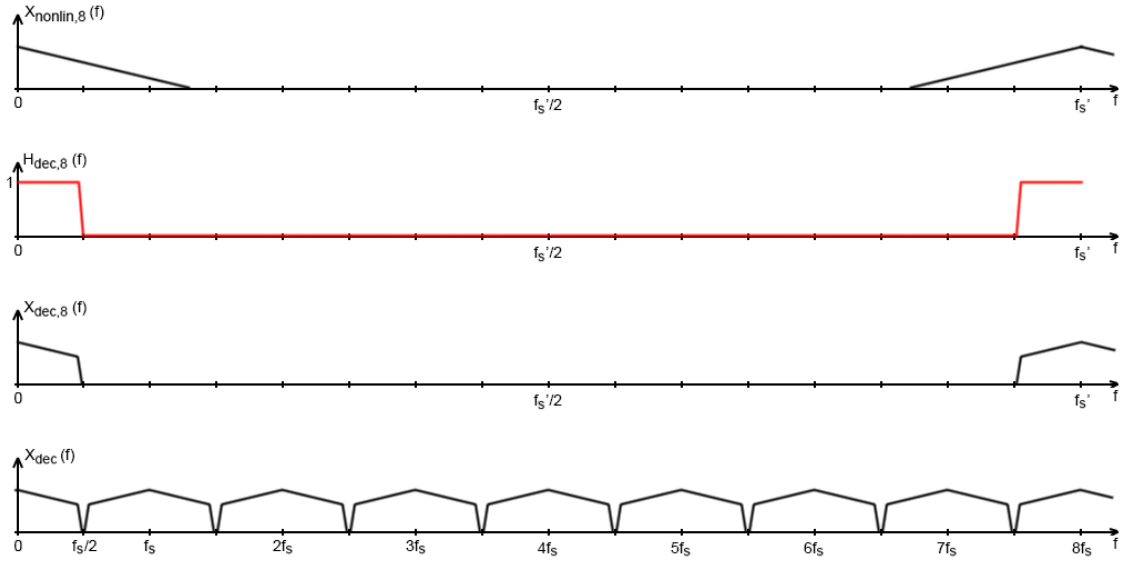
Nyolcszoros frekvenciára interpoláltuk fel a jelünket. Nézzük meg, hogy mi történne, ha egyszerűen minden nyolcadik mintát tartanánk meg, és így jutnánk vissza a kiindulási mintavételi frekvenciára!

A minták elhagyása után a jel spektrálisan nem változik, csak a mintavételi frekvencia csökken le nyolcadára. Ha a nemlineáris feldolgozás túl sok felharmonikus adott a jelhez - mint az a képen is látszik - átlapolódás következik be decimálás után, ami nemkívánt komponensek megjelenésével jár, a teljes spektrum az összes átlapolódás összegeként fog kiadódni (2.10. ábra).

Ha azonban a minták elhagyása előtt sávkorlátozzuk a jelet, az átlapolódás teljesen megszüntethető (2.11. ábra).

A szűrő, amivel szűrtünk, a cél mintavételi frekvencia felénél túl már nem szabad átengedjen, viszonylag éles szűrésre van szükség. Választhatunk IIR vagy FIR megvalósítások közül. Decimálásnál nem lehet kiépíteni polifázisú rendszert, azonban itt is van lehetőség a számításigény csökkentésére.

Ha N mértékben decimálunk, akkor csak minden N. mintát hagyjuk meg, a többit eldobjuk. Az ötlet az, hogy minek számoljuk ki az úgyis eldobandó mintákra a szűrőt, mi lenne, ha csak minden N. mintára tennénk ezt? Az elképzelés megoldható. Azonban ez nem jelenti azt, hogy minden N. mintát használunk csak fel a szűrő számításához. N



2.11. ábra. *Helyes decimálási eljárás.*

mintánként számolunk szűrőt.

A cirkuláris bufferünkbe N mintát beléptetünk, majd elvégezzük a konvolúciókat. Újra N mintát beléptetünk, újra konvolválunk, és így tovább. Az egyes konvolúciók kimenetei lesznek a decimált jelünk mintái. A folyamat során nem kell a mintákat programozottan elhagyni, az N mintánkénti szűrés automatikusan megoldja ezt.

(Az optimális cirkuláris buffer megvalósításról bővebben a F.7 fejezetben lehet olvasni)

3. fejezet

iOS és Core Audio

Ehhez és a következő fejezethez felhasznált forrásokról az irodalomjegyzékben hivatkozott dokumentumokból lehet többet megtudni, a gyors elérés érdekében itt csak felsorolom őket:

Learning Core Audio [12]

Core Audio Glossary [10], Multichannel Mixer [11], Audio Unit Properties [6], Audio Unit Hostnig [4], Audio Unit Processing Graph [5], Audio Mixer Host [3], Aurio [7], Core Audio Data Types [8], Core Audio Framework Refference [9], Converting Audio Samples On iOS [20].

3.1. Core Audio

A *Core Audio* egyike a legalacsonyabb szintű hangfeldolgozási API-nak, amit az Apple eszközein felhasználhatunk. Ha nem tartjuk elegendőnek, hogy egy hangfájlt csak pár kódsorral megnyissunk és lejátszunk, ha teljes irányítást szeretnénk a hangminták szintjén, akkor a *Core Audio* lesz az eszköz, amit használni tudunk.

Az API C nyelven íródott, ezért bármely másik nyelvben, ami képes C függvényeket hívni használható (C++, Objective C, Java). Nincsenek osztályok és objektumok, használata közben csak és kizárólag C függvényeket hívunk meg, amivel az egyes elemek paramétereit módosítjuk. Például, ha egy virtuális eszköz kimenetét össze szeretnénk kötni egy másik virtuális eszköz bemenetével, akkor egy olyan függvényt kell meghívni, aminek paramétereiben megadjuk a két eszközt (forrás és cél eszköz), valamint azt, hogy melyik kimenetet kívánjuk összekötni melyik bemenettel (egy eszköznek akár több be- és kimenete is lehet). A függvény ezután elvégez minden szükséges beállítást.

A *Core Audio* API működése közben virtuális eszközöket valósít meg (bemeneti, kimeneti, keverő, EQ, effekt), melyek kezelését, létrehozását paramétereik kezelését a meghívott C függvények látják el. Ennek a rendszernek két nagy hátránya van: rengeteg paraméter és valós idejű feldolgozás.

Az eszközöknek nagyon sok paramétere van (nagyon hosszú a nevük is), ezért használatuk kitanulása igen hosszadalmas.

Akár számítógépen, akár iOS eszközön használjuk az API-t, annak a jelfeldolgozása mindig egy elsődleges prioritású szálon fog futni (mindkét esetben multitaszkos rendszerekről beszélünk). Ennek előnye, hogy a feldolgozás jó közelítéssel valós időben, nagyon kicsi késleltetéssel történik. Előnye pont a hátránya is: egy sima általános programból történő paraméterváltoztatás nem biztos, hogy atomi módon (végig lefut, nem szakítja meg magasabb prioritású szál) képes lefutni. Elképzelhető, hogy a futását megszakítja a feldolgozó függvény, érvénytelen állapotban találva az struktúrát.

Megoldás lehet a multitaszkos rendszerekből ismert szinkronizáló és lezáró módszerek használata.

Nagyon hatékony jelfeldolgozási eszköz van a kezünkben, de ha a használatának a kitanulására, és egyáltalán a használata során felmerülő nem a témába vágó problémák megoldásra több időt kell fordítani, mint magára a feldolgozásra, akkor ez kevésbé előnyös.

Erre a problémára az Apple is gondolt, és kiadott egy programozásilag átlátszó befoglaló rendszert, aminek a segítségével nagyon egyszerűen, intuitívan és nem mellesleg thread safe módon kezelhetjük a jelfeldolgozási feladatainkat. Csak az érdemi rész megvalósítására kell koncentráljunk.

Ez a rendszer az *Audio Graph*.

3.2. Az Audio Graph használata

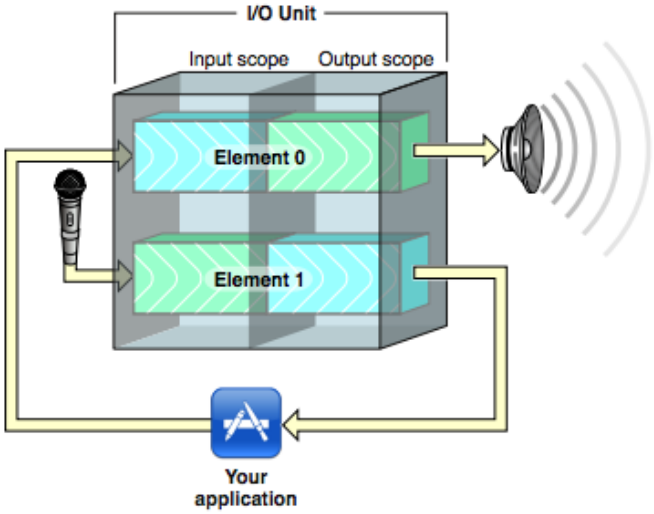
Az *Audio Graph* API segítségével összeköthetünk több *Audio Unit*-ot, valamint feldolgozó függvényt, ezeknek a topográfiáját tetszőlegesen változtathatjuk, akár futás közben is adhatunk hozzá vagy vehetünk el egységeket, ezzel szinte bármilyen audio feldolgozási feladat megoldhatóvá válik. Az egyes egységek reprezentálására egy új típust vezettek be, az *AUNode*-ot. Az API használata közben ezeket a reprezentációkat fogjuk a legtöbbet használni.

3.2.1. IOUnit

Az *IOUnit* az egyik legalapvetőbb jelfeldolgozó egység, ami a *Core Audio* API-ban szerepel. Minden alkalmazás használja, méghozzá pontosan egyszer. Egy *IOUnit* pontosan két részegységet tartalmaz: bemenet és kimenet. Rájuk az azonosítójukkal hivatkozhatunk. A bemenet (angolul input) azonosítója az 1, az I betű után, a kimenet azonosítója a 0, hasonló megfontolások alapján.

A 3.1. ábrán az *IOUnit* tipikus felhasználását látjuk. A jelfolyam bővíthető a bemeneti részegység kimenetét és a kimeneti egység bemenetét összekötő jelút mentén. Ide bármilyen másik jelfeldolgozó egység beköthető.

Az ábrából még látható, hogy mindkét részegységnek van bemenete és kimenete.



3.1. ábra. *IOUnit felépítése és tipikus bekötése. Az ábra forrása: [4].*

3.2.2. Az Audio Graph működése

Ha két feldolgozó egységet összekötünk és a rendszert futtatni kezdjünk, akkor a folyamatot mindig a legutolsó egység kezdi a sorban, ami minden esetben az *IOUnit* kimeneti részegysége. Tudja, hogy ki van rákapcsolva, ezért meghívja annak a feldolgozási függvényét. A meghívott egység vagy az *IOUnit* bemeneti egysége (a legegyszerűbb eset), ami rögtön lerendereli a függvényhíváskor kapott bufferbe a bemeneten várakozó mintákat, annyit, amennyit a hívás kért. Ha az eszköz nem tud mintákkal szolgálni, akkor tudja, hogy ki van előtte a sorban, ezért meghívja annak a feldolgozási függvényét, ahonnan vagy megkapja a szükséges adatokat, vagy a hívott fél is további hívást intéz a sorban következő eszköznek.

Látható, hogy minták igénylése a lánc végétől az elejéig terjed, maguk a minták pedig a lánc elejétől a vége felé adódnak át.

Tudjuk, hogy a legutolsó elem mindig az *IOUnit* kimeneti részegysége, viszont az legelső eleme nem feltétlenül kell az *IOUnit* bemeneti részegysége legyen. Lehet egy általunk megvalósított feldolgozási függvény, amiben mi generálunk programozottan mintákat. Hangszintézis esetén pont ez történik.

3.2.3. Egy feldolgozó függvény felépítése

Ha feldolgozó függvényt készítünk, akkor bárhogy elnevezhetjük azt, de a paramétereinek a megválasztására megkötések vonatkoznak. Nézzük végig, milyen paraméterekkel kell rendelkezzünk!

- *felhasználói adatra mutató mutató*: ezzel a paraméterrel válik teljesen rugalmassá a függvények kiépítése. Ide megadhatjuk a függvény által felhasznált adatok külső elérését (szűrőegyüttthatók, ideiglenes tárolók, cirkuláris bufferek).
- *opcionálisan felhasználható jelzőbiteket tartalmazó tömbre mutató mutató*

- *adott hívás pontos időpontja*: jelszintézisnél hasznos információ
- *jelút azonosítója, ahonnan a hívás érkezett*
- *beérkező minták száma*
- *mintákat tartalmazó tömbre mutató mutató*: sztereó jel esetén a tömb kétdimenziós

A függvény végrahajtása alatt az a cél, hogy a lehető minél gyorsabban feldolgozzuk a bejövő mintákat, majd feldolgozás után írjuk felül őket a feldolgozottakkal.

3.2.4. Adattípusok a feldolgozásban

Két egység között nemcsak az összekötést lehet definiálni, hanem az összekötésen keresztüláramló minták adattípusát is. Erre az API külön adattípus definiáló struktúrát használ. Egy összekötésnél elegendő egyszer definiálni az adattípust, a rendszer automatikusan továbbterjeszti azt. Ha egy beépített feldolgozó egység más adattípust kap a bemenetére, mint amit a feldolgozás során használna, akkor a feldolgozás előtt automatikus konverzió hajtódik végre.

3.2.5. Az Audio Session

iOS eszközöknél az audio feldolgozást külső események megszakíthatják (telefonhívás, értesítés, eszköz lezárása), ezért a *Core Audio* speciálisan az iOS eszközökre tartalmaz egy *AudioSession* nevű globális objektumot, ami nagyban befolyásolja a jelfeldolgozási egységünk viszonyát az egész iOS eszközhöz képest. Kérhet jogokat az operációs rendszertől hang felvételére, hang lejátszására, háttérben való futáshoz, beállíthatja a kapott mintasorozatok maximális hosszát, valamint megválaszthatja megszakítás esetén a megszakítás módját.

3.2.6. Audio Graph kiépítése

Ahhoz hogy be tudjuk indítani a jelfeldolgozási folyamatot, ki kell építeni azt. Ebben nagyban segítségünkre van az *Audio Graph* API.

1. Létre kell hozni egy *AudioSession*-t, azaz a globális objektumra készíteni kell egy lokális referenciát, majd megfelelően fel kell paraméterezni, ezzel tudatva az operációs rendszert a szándékainkról.
2. Meg kell határozni, hogy milyen *Audio Unit*-okat akarunk felhasználni a feldolgozás során. Erre az API egy paraméteres kereséshez hasonló szolgáltatást nyújt. Meg kell adni, hogy milyen tulajdonságú egységekre keressen rá, ő pedig visszatér egy tömbbel, benne a keresésnek megfelelő egységek típusával.
3. Létre kell hozni az *Audio Graph*-ot az előzőleg megalált típusokkal, ezzel létrehozva a konkrét egységeket.

4. A már létrehozott egységeket fel kell paraméterezni, hogy tisztában legyenek a szerepükkel.

5. Az egyes egységeket össze kell kötni.

6. Minden készen áll, inicializálhatjuk és elindíthatjuk a feldolgozást.

4. fejezet

Megvalósítás

4.1. iOS környezet kiépítése

4.1.1. iOS projektfájl hagyományos felépítése

Egy hagyományos iOS alkalmazás legegyszerűbb esetben két fő osztályból és egy kezelői felületet leíró fájlból áll.

Az *AppDelegate*-nek nevezett osztály tartja a kapcsolatot az operációs rendszerrel, reagálási lehetőséget biztosít a programnak a lehetséges események lekezelésére (hívás érkezett; kevés memória; az alkalmazás a háttérben fog futni; az alkalmazás a háttérben való futást megkezdte; az alkalmazás újra aktív lesz; az alkalmazás újra aktív lett; az alkalmazás kiléptetésre kerül; az alkalmazás befejezte a betöltést, képernyő megjelenítésére készen áll). Az osztály a megjelenítésért felelős objektummal rendelkezik. Az *AppDelegate* objektum a programkódból bárhonnan elérhető, az operációs rendszer biztosítja hozzá az elérést, ezért jól használható adatszerkezetek tárolására.

A képernyő kezeléséért a másik fő osztály, az úgynevezett *ViewController* a felelős. Ez az osztály tartalmazza a kezelői felület kezelésére használatos függvényeket, amik akkor hívódnak meg, ha a felhasználó valamilyen műveletet hajt végre az érintőképernyőn.

Az Apple fejlesztői környezete nagyon intuitív ad a kezelői felületek létrehozására. Egy, a környezet által értelmezett fájlban (úgynevezett *StoryBoard* fájl) mindennemű programozás nélkül hozzáadogathatjuk a kívánt elemeket (gombok, képek, csúszkák, kapcsolók, saját egyedi objektumok), majd a hozzájuk kapcsolódó eseményeket egyszerűen hozzákötheztjük a már megírt kiszolgáló függvényekhez a *ViewController* osztályunkban.

Az alkalmazás beindítása után, a operációs rendszer elkezdi kiépíteni a program futásához szükséges környezetet, majd meghívja az *AppDelegate* indítási műveletek elkészültét jelző függvényét. Itt elvégezhetjük a szükséges egyedi inicializálási feladatainkat, beállíthatunk konstansokat, adatszerkezeteket építhetünk ki. A függvény visszatérése után megkezdődik a *ViewController* inicializálása, a kezelői felületet leíró fájl alapján a program magától példányosítja a megfelelő osztályokat, ha szükséges, egy ezekre mutató referenciát átad a *ViewController*-nek, létrehozza a kijelzőn megjelenő úgynevezett *View*-k hierarhiáját, majd miután ez megtörtént, a *ViewController*-ben is meghívódik az előkészületek végét

jelző függvény. Hasonlóan az előző esethez, itt is elvégezhetjük a *ViewController* specifikus beállításokat. A függvényen belül már elérjük a kezelői felület minden elérni kívánt elemét (programozottan jelezhetjük, hogy mely elemeket szeretnénk a programunkban manuálisan felhasználni), a referenciák már kiépítésre kerültek. A függvény visszatérésekor megjelenik a képernyő, és elkezdődik az alkalmazás tényleges futása.

4.1.2. A Model-View-Controller [MVC] programozási szemlélet

Az objektumorientált programozás megjelenésével a programok egyre inkább átláthatóbbá, könnyen karbantarthatóbbakká, bővíthetőkké váltak. Ehhez jelentősen hozzájárult az a szemlélet, ami a programozási feladatokat jól elkülönítve három nagy csoportba osztja.

A ”model” tartalmazza az osztályokat, objektumokat, amik az adatszerkezetet valósítják meg, kommunikálnak adatbázisokkal, a program lényegi részét tartalmazzák.

A ”view” a megjelenésért felelős, a felhasználónak ez van a szeme előtt, itt jelenik meg az információ.

A ”controller” az előbbi két részegységet köti össze úgy, hogy ő reagál az eseményekre, ő dolgozza fel a felhasználó beavatkozásait, és ezek függvényében módosítja az másik két részegységet.

Ha a három feladatkör elkülönítését betartjuk, akkor egy nagyon rugalmas szerkezethez jutunk, amivel a későbbiek folyamán (bővítés, karbantartás) nem lesz különösebb problémánk.

4.1.3. A program felépítése

Mint azt az előző fejezetben láttuk, az Audio Graph felépítése és beindítása után nincs sok dolgunk, a rendszer magától dolgozza fel a hangmintákat, nekünk csak a paraméterváltozásokat kell lekezelni.

Nézzük meg, hogy milyen osztályokat kell megtervezni és elkészíteni a feladat megoldásához!

A programban szigorúan betartjuk az MVC szemléletet, és nem keverjük össze az egyes feladatokat az objektumokon belül. A controller és a view alapértelmezetten adva van a projekt indításakor, nekünk csak a szükséges módosításokat kell elvégezni rajtuk (kezelői felület kiépítése, view és controller eseményeinek összekötése). A jelenlegi implementációban nincs kihasználva az *AppDelegate* által nyújtott bárhonnan elérhető osztály előnye. (Megjegyzem: az ilyen viselkedés igen könnyen kiépíthető egyedi osztályoknál is. Részletek a függelékekben [F.8].)

A hangfeldolgozás elvégzése egy külön osztály feladata lesz, ami magán belül kiépíti az Audio Graph-ot, beindítja azt, majd az általa megvalósított jelfeldolgozó függvényekben elvégzi a tényleges jelfeldolgozási feladatokat.

Ez az osztály fogja tárolni és kezelni az jelfeldolgozó függvénynek átadott struktúrát,

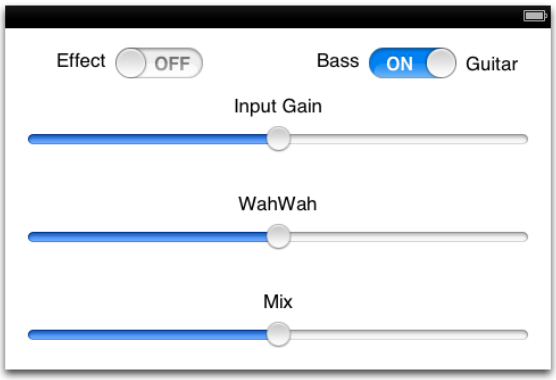
aminek jelentőségét később részletesebben tárgyalni fogjuk.

Az Audio Grap felépítését tekintve két *Audio Unit* szerepel benne: egy kimenetet és bemenet kezelő *RemoteIO*, a másik pedig egy *MultiChannelMixer*. A *RemoteIO* egységet az előző fejezetben már megismertük, a *MultiChannelMixer* pedig egy egyszerű keverést végez. A bemenetére kötött egységeket keveri össze a megadott arányoknak megfelelően (alapértelmezetten minden bemenete egységnyi átvitelű). A keverő a későbbi esetleges bővítéseket könnyíti meg, nem kell teljesen átírni az Audio Graph-ot, hanem csak egy új eszközt kell csatlakoztatni egy nem használt keverőbemenetre.

A keverőnek egy bemenetét használjuk, azt, amire a *RemoteIO* egység van rákötve. Az egész lánc teljesen automatikusan, programozott beavatkozás nélkül működik egészen a keverő kimenetéig, amihez hozzácsatoljuk a wah-wah effektet megvalósító jelfeldolgozó függvényt, amit a keverő után kötött kimeneti egység fog meghívni minden alkalommal, amikor szüksége van új mintákra. A függvény tartalmát, az általa megvalósított jelfeldolgozást a következő alfejezetekben részletekbemenőleg tárgyaljuk.

4.1.4. Felhasználói felület

A felhasználói felület kiépítésénél a funkcionalitás volt az elsődleges szempont, ezért elég egyszerű a kivitele. Csak az iOS alap építőköveiből építkeztünk.



4.1. ábra. *Felhasználói felület iPod-on.*

Vegyük sorba az egyes vezérlőelemek feladatát!

Az effekt be és kikapcsolható az Effect kapcsoló segítségével. Két üzemmód közül választhatunk: gitár vagy basszusgitár. Az effekt bemenetére jutó jel erőssége szabályozható, hogy tökéletesen az analóg effekthez lehessen hangolni (mindkettő azonos jelszinten kezdjen el torzítani). A pedál előre hátra döntését a Wah csúszka, míg az oldalán található potenciométert a Mix csúszka valósítja meg.

A csúszkák csak húzásra mozdulnak el, érintésre nem, ezzel megelőzhető a szűrő hirtelen paraméterváltása.

4.1.5. Kompatibilitás más iOS eszközökkel

Az feladat fő célja, hogy az wah-wah effekt modellezését egy harmadik generációs iPod-on valósítsuk meg, de az Apple kínálatában más iOS operációs rendszert futtató készülékek is vannak. Fog-e rajtuk futni az iPod-on futó program?

A válasz igen. Az Apple remekül kiépített rendszerrel rendelkezik, ezért az összes program, ami fut a hagyományos kis képernyős eszközökön (iPod, iPhone), az futni fog a nagyobb képernyővel rendelkező táblagépeken (iPad) is.

Ez visszafelé már nincs így, az iPad alkalmazások nem futnak a kisebb eszközökön.

4.2. Kiegészítő hardver [14]

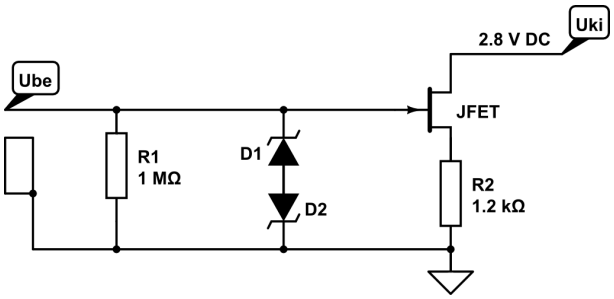
Az mai iOS eszközök mind rendelkeznek beépített mikrofonnal, amihez programozottan hozzá lehet férni, de ez alkalmatlan külső elektromos hangszerek, vagy más audio eszközök használatára. A 3. generációtól minden iPod is rendelkezik mikrofont meghajtani képes, 3.5 mm-es csatlakozóval ellátott audio ki- és bementettel, amihez a hagyományos 3.5 mm-es hárompólusú jack csatlakozótól eltérően egy négypólusú csatlakozó csatlakoztatható. A tartozék fejhallgató rendelkezik két hangszóróval, egy mikrofonnal és egy távirányítóval, amin három gomb található. Hogy kompatibilis legyen hagyományos csatlakozókkal is, melyek nem támogatják a mikrofont és a távirányítót, a jobb és bal csatorna a csatlakozón megszokott kiosztásban, a csúcstól befelé haladva: BAL, JOBB, FÖLD. A földelés ketté van osztva, és a csatlakozó tövénél kapott helyett a mikrofon és a kommunikáció csatlakozója.

A beépített mikrofon egy kondenzátormikrofon, ami működéséhez külső táplálás szükséges, amit az eszköz biztosít. A föld és a mikrofon csatlakozója között 2.8 V egyenfeszültség van.

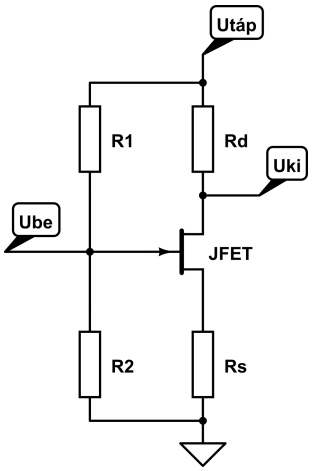
Ha ehhez a bemenethez egy elektromos gitárt direkt rákapcsolunk, akkor az eredmény nem lesz túl meggyőző. Az elektromos pengetős hangszerek általában nagy impedanciás ($R_{be} \approx 1\ M\Omega$), feszültségentes bemenetre számítanak, így lettek megtervezve, itt azonban egy fix egyenfeszültségű viszonylag kis impedanciás ($R_{be} \approx 1\ k\Omega$) bementettel találja szembe magát. Az átvitt hang torz lesz, a gitár kezelőszervei nem működnek.

Ahhoz, hogy elfogadható minőségű hangot kapjuk, egy illesztő áramkört kell betenni a hangszer és a eszköz bemenete közé, ami biztosítja mindkét irányba a megfelelő impedancia illesztést.

Erre a legmegfelelőbb egy jfet-es előerősítő áramkör:



4.3. ábra. *Illesztő áramkör kapcsolási rajza.*



4.2. ábra. *Hagyományos közös source-os jfet fokozat.*

Az illesztő áramkörnek a hangszer felé feszültségmentes, magas bemeneti impedanciát kell mutatni. A 4.2. ábrán látható áramkörből az eszközbe R_1 és R_d be van építve, ezeket nem tudjuk befolyásolni. Mivel pontos értékeiket nem tudjuk, ezért a megfelelő hangzáshoz kísérletezni kell az illesztő áramkör alkatrészeinek a megválasztásakor.

A 4.3. ábrán a megvalósított illesztő áramkör kapcsolása látható. A nagy bemeneti impedanciát R_1 biztosítja. A jfet-ek érzékenyek a nagy feszültségszintekre, ezért a gate elé egy zener diódás vágóáramkör került, aminek a letörési feszültsége meghaladja a maximálisan bemenő hasznos jelet, viszont a nagy feszültségcsúcsokat (elektrosztatikus kisülés) levezeti a földbe.

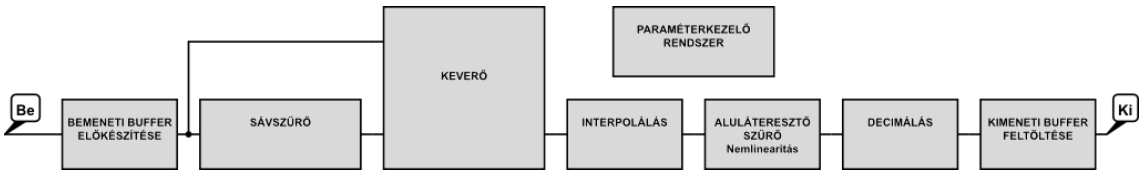
A mikrofon bemenetén kívül az áramkör többi része kivezeti a fejhallgató jobb és bal csatornáját egy hagyományos hárompólusú csatlakozóba, hogy a feldolgozott jelet meg is lehessen hallgatni.

4.3. Jelfeldolgozási algoritmus

Minden szükséges szoftveres és hardveres rész kiépítése után, nincs más hátra, mint a programozás lényegi részét, a tényleges jelfeldolgozást megcsinálni. Ehhez a már elkészített render callback függvényt használjuk.

4.3.1. Megvalósítandó digitális rendszer blokkvázlata

Az analóg blokkvázlat (1.15. ábra) összes részegységnek ismerjük a pontos működését, modellezésének a módját. Rajzoljuk fel a digitálisan megvalósítandó rendszer blokkvázlatát!



4.4. ábra. *Digitális rendszer blokkvázlata.*

Az eredeti effektnek egy bemenete és két kimenete van, amiből az egyik kimenet az effektezés nélküli jelet vezeti ki. Az iOS eszközökön ez nem valósítható meg, az audio interfésznek ugyanis csak egy bemenete és egy kimenete létezik. A tiszta kimenetet ezért nem építjük ki a programban.

Vegyük sorba az egyes elvégzendő feladatokat!

- *Bemeneti buffer előkészítése*: a függvény által paraméterként kapott buffer feldolgozáshoz kényelmes alakra hozása.
- *Sávszűrő*: IIR szűrőként megvalósítva.
- *Keverő*: a sávszűrő jelét és az tiszta jelet keveri egymáshoz a beállított paraméternek megfelelően.
- *Interpolálás*: nemlineáris feldolgozás előtt a mintavételi frekvenciát megnyolcszorozzuk.
- *Nemlineáris feldolgozás*: az aluláteresztő szűrőt a 2.2.4. fejezetben kidolgozott algoritmussal valósítjuk meg.
- *Decimálás*: visszatérünk alap mintavételi frekvenciára.
- *Kimeneti buffer feltöltése*: az effektünk utáni programrészek kétcsatornás mintasorozatra számítanak, ki kell szolgálni őket.
- *Paraméterkezelő rendszer*: a sávszűrő és a keverő egységeknek a változtatható paramétereit itt állítjuk be.

A rendszerünk az API-tól a *Core Audio* alapértelmezett formátumában kapja meg a mintákat, ami a 32 bites 8.24 felosztású, fixpontos számábrázolás. Minden kapott minta 1 és -1 közé van normálva, ezért az egészrész előjeltől függően csak nullákból vagy csak egyesekből áll. Fixpontos számokról van szó, amelyeket az integer aritmetika fog feldolgozni a processzorban, ami úgy veszi, mintha egész számokkal dolgozna, nem tud a kettedes

pontról. Ezért nekünk kell gondoskodni arról, hogy a számolásaink helyesek legyenek. Összeadásnál és kivonásnál nincs semmilyen változás az egész aritmetikához képest, azonban a szorzás és az osztás egy kiegészítő művelet beiktatásával jár. Szorzásnál a két 32 bites számból lesz egy 64 bites szám, ami az akkumulátorba kerül, ahonnan egy 24 bittel való jobbra shiftelés után vesszük ki az alsó 32 bitet. Osztásnál ugyan így járunk el, csak balra toljuk el az eredményt.

4.3.2. Bemeneti buffer előkészítése

A függvényünk a bemeneti buffert egy sztereó 256 mintát tartalmazó tömbként kapja meg, miután a bemeneti egység lerenderelte. Az effektünk egycsatornás, mono jelet dolgoz fel, ezért a bemeneti sztereó jelet egycsatornássá kell alakítani.

Az iOS eszközök audio bemenete - a mikrofon - egycsatornás, csak a jelet feldolgozó keretrendszer alakítja át kétcsatornássá, ezért biztosan kimondható, hogy a bemeneti buffer két csatornájának tartalma azonos. A további algoritmusoknak készítünk egy, a bemeneti buffer egyik csatornájára mutató mutatót, hogy a buffer hosszú elérési útvonala helyett egyszerűbben férjünk hozzá a buffer tartalmához.

4.3.3. Sávszűrő

A sávszűrő képletét a 2.2.2. fejezetben meghatároztuk. Az alakjából adódik, hogy IIR szűrőstruktúrával valósítható meg. Tudjuk, hogy az IIR szűrőket többféleképp lehet digitálisan megvalósítani:

Direkt formában [1], amikor a szűrőt rendszeregyenlet formában programozzuk le. Viszonylag egyszerű algoritmust kapunk, a lehető legminimálisabb számításigénnyel. Négy alváltozata létezik: Direkt forma 1 és 2, valamint mindkettő transzponált változata [DF1, DF1-T, DF2, DF2-T] [1]. Transzponáció alatt a *Tellengen tétel* által megfogalmazott jelfolyamgráf transzponációt értjük: minden jelirány megfordul, kimenetből bemenet, elosztóból összegző lesz.

A négy változat közül a DF1 és a DF2-T az elterjedten használt.

A DF1-et nagyon egyszerű leprogramozni, és numerikusan kedvező tulajdonsága, hogy az algoritmus során először a zérusok kerülnek felhasználásra, azaz nem kell félni a rejtett túlcsordulástól. Ha a pólusok kiemelését számolnánk először, és a jel túlcsordul, az utána következő zérusok leskálázhatják a jelet annyira, hogy a kimenetet alapján a szűrő eredményét helyesnek könyveljük el, miközben feldolgozás alatt túlcsordulás történt (telítődéses aritmetikát feltételezve). Hátránya a DF1-nek, hogy a felhasznált késleltetők száma nem kanonikus, pazarol az erőforrásokkal.

A DF2-T előnye, hogy minimális számú késleltetőt használ, ezzel memóriakezelés szempontjából nagyon optimális tud lenni.

A direkt formáknak egy nagy hátrányuk van, hogy ha nagy a paraméterérzékenységük. Ha egy erős pólus az egységkör széléhez közel, a számábrázolási kvantálás miatt véletlenül az egységkörön kívülre kerül, akkor a rendszer instabillá válik. Főleg nagyon éles levágású

szűrőknél lép fel ez a lehetőség.

A paraméterérzékenység csökkentése érdekében további IIR szűrőket megvalósító struktúrákat találtak ki az évek során. Létezik a rezonátoros struktúra, ami alkalmas IIR szűrők realizálására. Előnye, hogy strukturálisan stabil, azonban elég nagy számításigénnyel rendelkezik. A Lattice struktúra [22], vagy más néven rács struktúra a beszédfeldolgozásban elterjedt szűrőstruktúra. Jellemzően strukturálisan stabil, garantálja, hogy akármilyen szűrő ábrázolható vele, azonban a nagyon éles szűrőknél a struktúra paraméterei egymáshoz nagyon közel kerülhetnek, megfelelő számábrázolás szükséges hozzá.

Létezik még egy kevésbé elterjedt módszer, a hullámdigitális struktúra [22], amikor az analóg szűrőket alkatrészszinten modellezzük, majd a tényleges analóg topológiát visszük tovább a számításokhoz.

Ezen utóbbi három módszernek a szerepe nagyobb fokszámú IIR szűrők esetén jelentős, ahol a direkt formában való megvalósítás nagy valószínűséggel kudarcot vall.

Mi a direkt forma 2 transzponált verzióját használjuk, mivel mindössze egy másodfokú szűrőről van szó. Magasabb fokszámoknál meg lehet próbálni a direkt formáknál a soros vagy párhuzamos alakot, ha azonban ez nem ad kielégítő eredményt, akkor az összetettebb struktúrákat kell megvalósítani.

Nincs éles szűrőnk, nincs benne éles kiemelés, ezért feltételezhetően nem kell félni a számábrázolás végessége miatti instabilitás és hibás eredmény miatt.

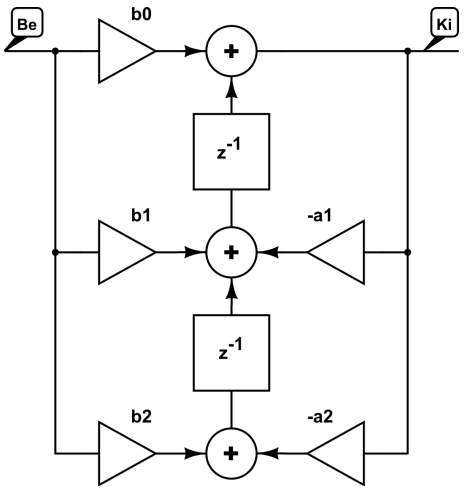
Az 4.5. ábra alapján felírható a szűrőt megvalósító algoritmus. Az alsó késleltetőt nevezzük DL_0-nak, a felsőt pedig DL_1-nek.

$$\begin{aligned} y &= DL_1 + b_0 x \\ DL_1 &= DL_0 + b_1 x - a_1 y \\ DL_0 &= b_2 x - a_2 y \end{aligned} \tag{4.1}$$

Az egyenleteket közvetlenül be lehet programozni. DL_0 és DL_1 értékét két mintablokk között el kell tárolni, hogy a szűrő ugrás nélkül tudja folytatni a működését. A paraméterek kezelését később tárgyaljuk.

4.3.4. Keverő

Ahogy az előző két fejezetben is láttuk, a keverő áramkör nem csinál mást, mint bizonyos arányoknak megfelelően összeadja a tiszta bemenetet a sávszűrő kimenetével, jó közelítéssel frekvencia független módon. Áramköri szinten ezt egy rezisztív hálózat teszi (1.10. ábra), digitálisan pedig egy összegző algoritmus, ami veszi a két jelet (tiszta és szűrt jel), mindkettőt megszorozza az adott beállításnak megfelelő konstanssal (részletesen később), majd összegzi őket. A programban egy az egyben a 2.7 képletet valósítjuk meg:



4.5. ábra. *IIR direkt forma 2 transzponált.*

```
% N a bemeneti blokk hossza
% Ktiszta és Kszurt a az aktuális beállításnak megfelelő paraméterek
% Utiszta a tiszta mintatömb
% Uszurt a sávszűrő kimenete
for i=1:N
        Ukev(i) = Ktiszta * Utiszta(i) + Kszurt * Uszurt(i);
end
```

Mivel a bemeneti tömböt tisztán és szűrten is fel kell használni a keverőben, azért el kell tárolni a sávszűrés előtt. Ehhez a programban egy előre allokált helyre a *memcpy()* c könyvtári függvénnyel lemásoljuk, majd a keverőben innen vesszük elő a szükséges értékeket.

4.3.5. Interpoláció

A 2.2.5 fejezetben megismert interpolálási módszereket használjuk az effektben. A még hatékonyabb algoritmus érdekében a polifázisú szűrőmegvalósításon kívül a nyolcszoros túlmintavételezést három lépésben valósítjuk meg.

Ehhez első lépésnek egy éles vágású IIR szűrőt tervezünk (4.6. ábra, (a)). A hetedfokú szűrőt hat darab másodfokú, tag sorbakötésével valósítjuk meg direkt kettes transzponált módon. Mivel IIR szűrőről van szó, ezért a visszacsatolás miatt nem lehet a teljes szűrőt polifázisúként futtatni, nem oldja meg automatikusan nullák beszúrását, azt nekünk kell megtenni a szűrés előtt. Három lépésben valósul meg a teljes interpoláció, minden lépésnél kétszerese lesz a mintavételi frekvencia. Minden minta közé be kell szúrni egy nullát, majd az így kapott mintasorozatot meg kell szűrni az IIR szűrővel.

A második lépésnél már elegendő a jóval lankásabb szűrő is [23]. Egyre több mintával kell foglalkoznunk, ezért célszerű FIR szűrőt alkalmazni, polifázisú megvalósításban (4.6. ábra, (b)). A második szűrő egy 32 tap-es FIR szűrő. Látható, hogy az átvitele jóval lan-

kásabb, mint az első szűrőé volt. Ez megengedhető, hiszen az interpolálás végén az összes szűrő átvitele együttesen fog érvényesülni. Polifázisú szűrőről van szó, ezért nem kell a nullák beszúrásával foglalkozni.

A harmadik lépést szintén egy FIR szűrővel szűrünk. 16 tap-es, polifázisú megvalósításban (4.6. ábra, (c)).

A teljes interpolációs szűrés átvitele végül a három szűrő átvitelének a szorzata lesz, mivel soros rendszerről beszélünk. A mintavételi frekvencia szűrés közbeni változása miatt a végső $[0,f_s'/2]$ tartományba az első szűrő ábrán ábrázolt átvitele négyszer fog beleférni (megfelelően tükrözve az második és negyedik ismétlésnél, lényegében, mintha ábrázolnánk a szűrő átvitelét az ő mintavételi frekvenciáján $4f_s$-ig), a második szűrő ábrán ábrázolt átvitele pedig kétszer. Az így kapott teljes átvitel az 4.7. ábrán látható.

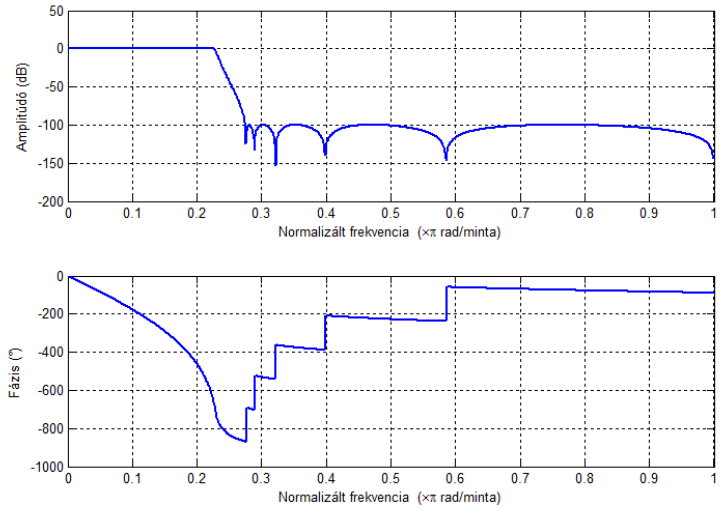
4.3.6. Nemlineáris aluláteresztő szűrő

A minták nyolcszoros felinterpolálása után a 256 mintából 2048 minta lett. Ezeket kell átfuttatni a nemlineáris viselkedésű aluláteresztő szűrőn. A szűrés elvégzése után a jelnek jelentős felharmonikus tartalma lesz, ami akár át is lóghat az alap mintavételi frekvencia felénél túlra. Átlapolódás azonban nem lesz jelentős, hiszen az interpolálás hatására a mintavételi frekvencia az eredeti nyolcszorosára nőtt.

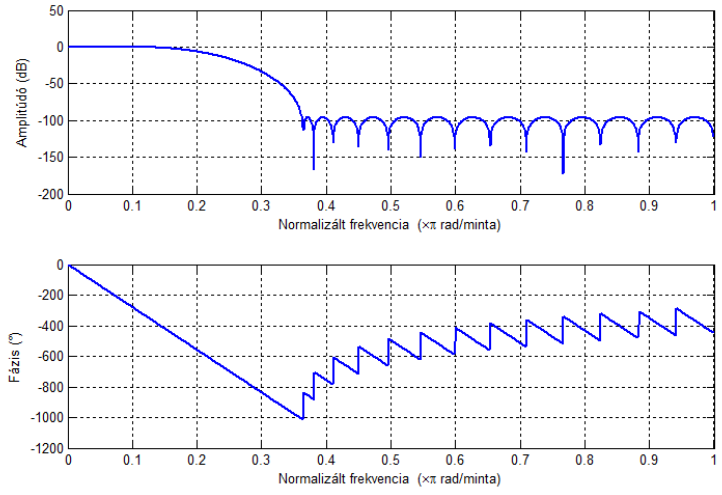
A szűrést a 2.2.4 fejezetben megtervezett algoritmus fogja ellátni. Az előrelépő Euler módszert használva az alábbi algoritmust tudjuk MATLAB-ba beprogramozni:

```
% x a bemeneti vektor
% y a kimeneti vektor
% N a bemenet hossza
% A és B az állapotváltozós leírásban szereplő együtthatók
% T a lépésköz
% MAX és MIN az eredeti effekthez hangolt értékek

% kezdeti érték beállítása
y(1) = 0;

% szűrőt megvalósító ciklus
for n = 2:N
   y(n) = (1+A*T)*y(n-1) + (B*T)*x(n-1);
   if y(n) > MAX
      y(n) = MAX;
   end
   if y(n) < MIN
      y(n) = MIN;
   end
end
```

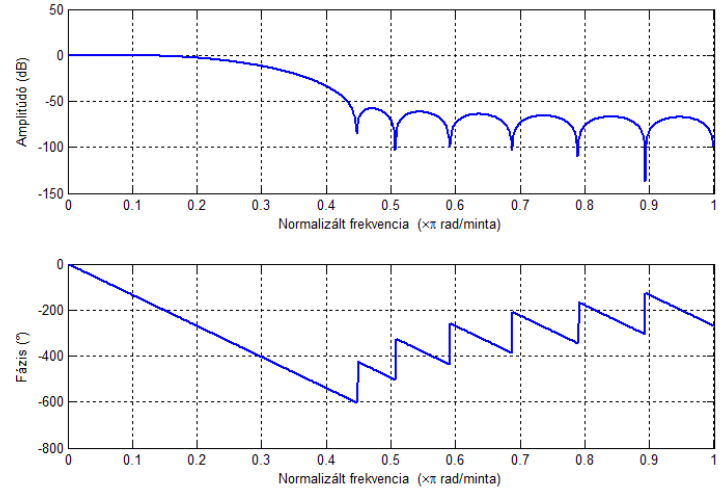
Ahogy már a 2. fejezetben láttuk, mindegyik kiszámolt kimeneti mintára lefuttatunk egy ellenőrzést, hogy az analóg rendszernek megfelelően képes lenne-e kiadni az adott értéket



(a)

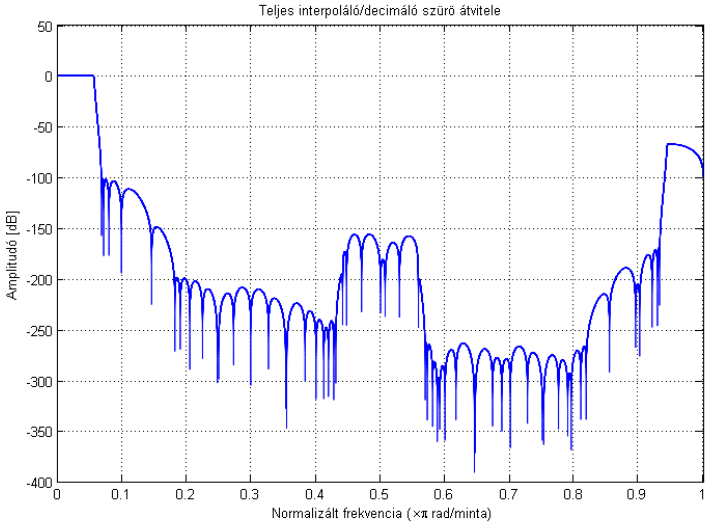


(b)



(c)

4.6. ábra. *Első, második és harmadik interpoláló szűrő átvitele.*



4.7. ábra. *A teljes interpolációs szűrés átvitele.*

a rendszer. Ha az érték kifutna a maximális vagy minimális határon, akkor lekorlátozzuk. Ekkor lényegében az állapotváltozó értékét írjuk felül, amit az algoritmus a következő kimenet számolásánál fel fog használni. A kimenet korlátozása hatással van a következő kiszámolt kimenet értékére.

4.3.7. Decimálás

Hasonlóan az interpoláláshoz, decimálásnál is három lépésben érjük el nyolcszor kisebb mintavételi frekvenciát. Ugyanazokat a szűrőket használjuk itt is, csak fordított sorrendben. Az utolsó lépés szűrése kell a legélesebb legyen.

Az első két lépésben a FIR szűrőket használjuk, ezért megtehető, hogy a 2.2.6 fejezetben közölt módon, hatékonyan szűrjünk, csak minden ténylegesen felhasznált mintára futtatjuk le a szűrési algoritmust.

4.3.8. Kimeneti buffer feltöltése

Az eddigi jelfeldolgozást egy csatornában, monoban tettük, a további programrészletek azonban kétcsatornás jelfolyamot várnak. Hogy igényüket kielégítsük, nem kell mást tenni, mint a kapott szűrt mintasorozatot a már megismert *memcpy()* függvénnyel az eddig fel nem használt bemeneti buffer helyére be kell másolni, és a kétcsatornás szűrt jelünk átadódik további iOS felügyelte feldolgozásra.

4.3.9. Paraméterkezelő rendszer

Az előző fejezetben megismerkedtünk a Core Audio rendszerével, és beláttuk, hogy az audio feldolgozó függvénynek átadott felhasználói paraméter a lehető leggyorsabban kezelhető kell legyen. Fontos, hogy ne használja az Objective C lassú üzenettovábbítási rendszerét, alapvető C elemekből kell építkezzen.

Erre a feladatra a struktúrákat találtam a legmegfelelőbbnek.

A hangfeldolgozásért felelős objektum egy olyan struktúrát tartalmaz, amin keresztül az egyes szűrők paraméterei adódnak át a feldolgozó függvénynek, valamint ebben a struktúrában helyezkednek el a két feldolgozandó blokk közötti adatok tárolására hivatott memóriaterületekre mutató mutatók, melyek segítségével a függvény hozzáférési lehetőséget kap. A jelfeldolgozó függvény egy a struktúrára mutató mutatót kap meg, amit felhasználva (C nyelvben a -> operátorral hozzáférve) eléri azt.

A Model-View-Controller elvhez hűen, ez a struktúra jelenti az összeköttetést a Model (jelfeldolgozó objektum) és a Controller (ViewController objektum) között. Az interfészen bekövetkező felhasználói paraméterváltoztatásról a Controller értesítést kap, megkapja az aktuális paramétert, amit a Model-nek továbbítva, az képes feldolgozni azt, és frissíteni a struktúrát, hogy a következő jelfeldolgozási hívás már a friss paraméterkészlettel futhasson le.

Vegyük sorba, hogy milyen változtatható paramétereket kell átadni a jelfeldolgozó függvénynek!

A digitális rendszerünkben két változtatható paraméterű egység van, valamint egy kényelmi funkciót ellátó bemeneti jelerősség szabályzó. Kezdjük a két legegyszerűbbel!

A keverő áramkör és bemeneti jelerősség szabályzó hasonló elven működik. Mindkettő a jelet (jeleket) beszorozza egy konstanssal. Ezt a konstanst számolja ki a Controller hívására a Model: a bemeneti jelerősség szabályzónál nem kell számolni, csak át kell konvertálni az interfésztől kapott lebegőpontos értéket fixpontosan ábrázolt számmá, és frissíteni a struktúrát. A keverő paraméterei kissé összetettebb módon számolhatók: a kapott lebegőpontos paramétert felhasználva, egy képletbe behelyettesítve, majd a kapott eredményt fixpontosra átkonvertálva megkapjuk a szükséges paramétereket.

Legnehezebb dolgunk egyértelműen a sávszűrőnél adódik. Mint azt a 2.2.2 fejezetben eredményül megkaptuk, az analóg szűrőt hűen modellező digitális szűrő paramétereihez egy kiegészítő lépés beiktatásával juthatunk hozzá. Ez a lépés egy függvényhívás [F.5], aminek a paraméterei sajnálatos módon nem adhatók meg gyorsan programozott úton. Ha analitikusan, zárt alakban szeretnénk számolni, akkor a paraméterek kiszámítása előtt ki kéne számolni az analóg szűrő frekvenciatartománybeli amplitúdó menetét, azon méréseket kéne végezni, hogy megkapjuk a szükséges paramétereket a F.5 függvényhez (átvitel erőssége DC-n G_0. kiemelési frekvencia f_0, kiemelési frekvencián az erősítés G, kiemelés szélessége Δf, a kiemelés szélességénél mért erősítés G_B), ami végül kiadja a pontosan illeszkedő szűrő együtthatóit.

Ez elég hosszadalmas folyamat, és bár a jelfeldolgozást sebességét nem befolyásolná (a jelfeldolgozási függvény egy elsődleges prioritású szálon fut), paraméterváltásra lassan reagálna a rendszer, ami egy wah-wah pedál esetén elfogadhatatlan.

Az online számolás helyett a paraméterezés megoldható offline módon is. Egy előre meghatározott sűrűséggel manuálisan kiszámoljuk a szükséges paramétereket, letároljuk, majd a megfelelő helyről elővesszük és felhasználjuk őket. Ekkor feltételezzük, hogy a felhasználó nem képes az interfészen bármilyen tetszőleges értéket megadni paraméterként, csak korlátozott felbontással módosíthatja őket. Ekkor nem kell a letárolt értékek közötti interpolációról gondoskodni, hanem a megfelelő helyről elő kell venni a paramétert, fixpontosra konvertálni (vagy időspórolás miatt már eleve fixpontosan is tárolhatjuk őket), majd frissíteni velük a struktúrát.

Ez már egy használható megoldás, de közel sem optimális. A függelékekben [F.6] bemutatok egy hatékonyabb módszert, ami végül ténylegesen bekerült a digitális effektbe.

4.4. A megvalósítási eszköz

Végül a fent megtervezett programot egy harmadik generációs iPod touch eszközön valósítjuk meg.

Ezt a modellt 2009 szeptemberében adták ki, az iOS harmadik generációjával együtt. Belül egy Samsung által gyártott, System on Chip technológiával készült, összetett rendszer található. Az integrált áramkörben egy ARM Cortex-A8 processzor mag kapott helyet, ami alapértelmezetten 833 MHz-en fut, de az iPod-ban le van korlátozva 600 MHz-re, az akkumulátor üzemidejének növelése érdekében. A grafikáért egy PowerVR SGX535 GPU mag felelős. Alapvető hangfeldolgozást egy Cirrus Logic CS4398 chip látja el. A második generációhoz képest megduplázták a készülék memóriáját, a harmadik generációban 256 MB DRAM kapott helyet. Az eszköz nem rendelkezik kamerával, vezeték nélküli hálózatokhoz Wi-Fi-n keresztül képes csatlakozni. Megjelenésében újdonságnak számított az iPod-ok körében a rendkívül vékony készülékház, az iPhone-okból átemelt hangvezérlés és a távirányítós fejhallgató. Tárkapacitást tekintve 32 vagy 64 GB-os eszközöket vásárolhattunk.

A specifikációból látható, hogy a maga nemében erős hardver dolgozik az iPod belsejében. Kérdés, hogy a tervezett audio jelfeldolgozási algoritmusainkat képes lesz-e kiszolgálni valós időben.

4.4.1. Kényszerű minőségbeli visszalépések

Sajnos a harmadik generációs iPod jóval gyengébben teljesített a vártnál, ezért a gyakorlati megvalósítás során jelentősen le kellett butítani a digitális rendszert, hogy egyáltalán képes legyen tartani az iramot a beérkező adatokkal.

Az eszközölt butítások egyértelműen a nemlineáris feldolgozás előtti interpolációt majd az utána következő decimálást érintették.

Első próbálkozásra egy lépésben próbáltam nyolcszoros mintavételi frekvenciára felinterpolálni a jelet egy viszonylag éles levágású szűrővel. A kísérlet kudarcba fulladt. Ezután

kiépítettem erre a szűrőre a polifázisú FIR szűrés algoritmusát, de ez sem segített rajta. Még csak az interpolálásról van szó, még nem volt nemlineáris szűrés sem decimálás. De így sem bírta el a rendszer.

Kisebb fokszámú (jóval kisebb 300 tap helyett 50-60 tap) szűrők esetén éppen belefért a teljes rendszer, de a gyenge szűrők miatt nem szerepelt valami fényesen.

Egylépéses megvalósítás helyett háromlépésesre váltottam. Az első fokozat egy éles, 11 tap-es IIR szűrő, kétszeresre interpolálva a jelet, utána két egyre kisebb tap-es FIR szűrő következett (32,4), amik mind 2-2-szeres interpolálást hajtottak végre, polifázisú üzemmódban. Jóval hatékonyabbnak bizonyult, mint az egylépéses polifázisú szűrő, de még így sem fért bele a rendelkezésre álló időbe.

Felmerült az ötlet, hogy mi lenne, ha a sok for-ciklussal megvalósított programrészletet helyettesítenénk begépelt kóddal, hogy ne a ciklusszervezéssel menjen el a CPU ideje. A kísérletezések alatt kiderült, hogy ezt sem szabad túlzásba vinni. Ha minden for-ciklust kibontunk, akkor 3,73-szor tovább tart ugyanannak a számítási sorozatnak az elvégzése, mintha csak for-ciklusokat használnánk erre a célra. Az okát nem tudtuk megmondani.

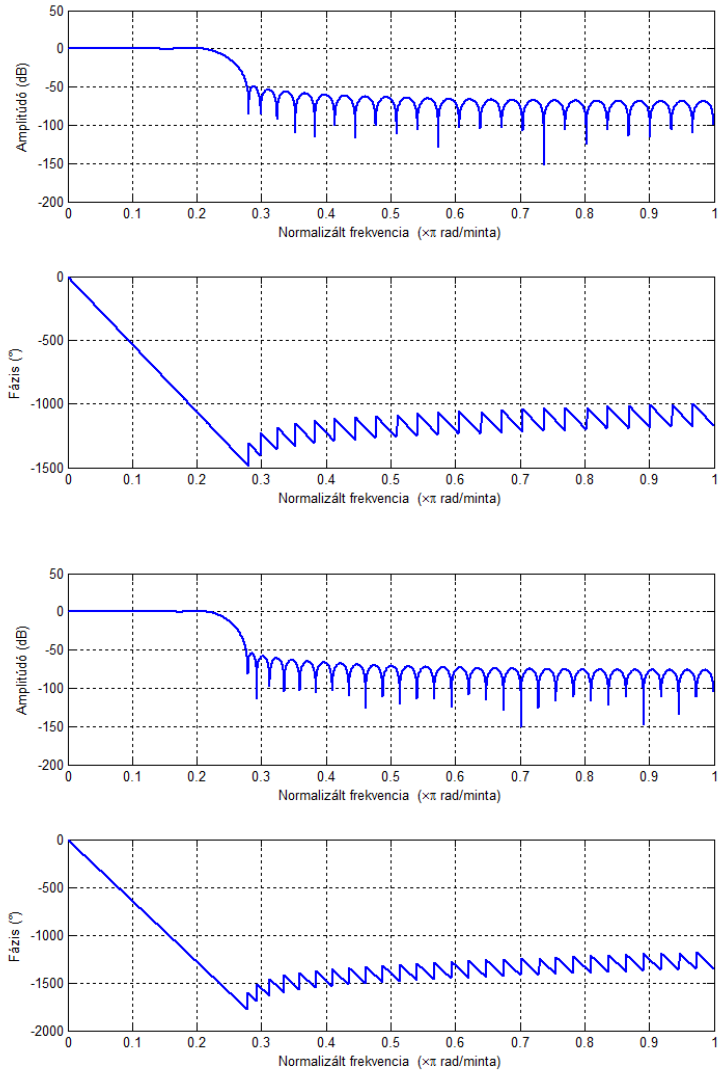
Ha azonban a for-ciklus ciklusai és a kézzel leírt sorok számának aránya 0.1 és 100 között van, akkor maximálisan 2.5-szeres sebességnövekedés érhető el.

A végső működőképes megvalósításnál feladtam az igényt a nyolcszoros túlmintavételezésre és csak kétszeres mintavételi frekvenciára mentem fel. Ekkor enyhébb specifikációjú szűrőket lehet használni, nem kell olyan keskeny áteresztő tartomány, mint az előző esetekben.

Megvalósítás tekintetében ugyanúgy polifázisú fir szűrők lettek használva. Interpoláláshoz 60 tap-es, a decimáláshoz 72 tap-es szűrőt alkalmaztam (4.8. ábra). Ez volt a maximum, ami belefért az feldolgozási időbe.

A kétszeres interpoláció miatt félő volt, hogy az előrelépő Euleres aluláteresztő szűrő nem fog aluláteresztő szűrőként viselkedni, ahogy alap mintavételi frekvencián nem tette azt, de kiderült, hogy kétszer kisebb lépéseket alkalmazva, már hűen előállítja a nyolcszoros mintavételi frekvencián számolt aluláteresztő jellegű karakterisztikát.

A dolgozat mellékletébe megtalálható mind a nyolcszoros, mind a kétszeres túlmintavételezéses modellhez egy hangminta, valamint egy-egy MATLAB függvény, ami pontosan ugyan azt a jelfeldolgozást végzi, mint a tényleges iOS eszköz. A függvények időben paraméterezhetőek. Tetszőleges pedálmozgás szimulálható velük tetszőleges hangmintán.



4.8. ábra. *iOS eszközben használt interpoláló és decimáló szűrő átvitele.*

5. fejezet

Eredmény és összegzés

5.1. Végső összehasonlítás

A kész algoritmust a MATLAB-ban és az iOS eszközön tesztelve igen jó eredményeket kaptunk. Igaz, az iPod-on nem a legjobb minőségű algoritmus futott, de mégis hűen visszaadta az eredeti WH-10 wah-wah pedál hangzását.

A helyes működésről bizonyosodjunk meg mérésekkel!

A függelékekben található módszert (F.9) használva, mérjük meg az eredeti analóg és a megvalósított digitális rendszer átvitelét. A gerjesztő jel legyen egy 0.1 és 10 *kHz* között logaritmikusan átsöprő *chirp* jel, aminek a hossza 250 *ms*. Ezt a jelet periodikusan ismételgetve rákötve a rendszerekre, oszcilloszkóppal vizsgáltuk a kialakuló kimeneti jelet.

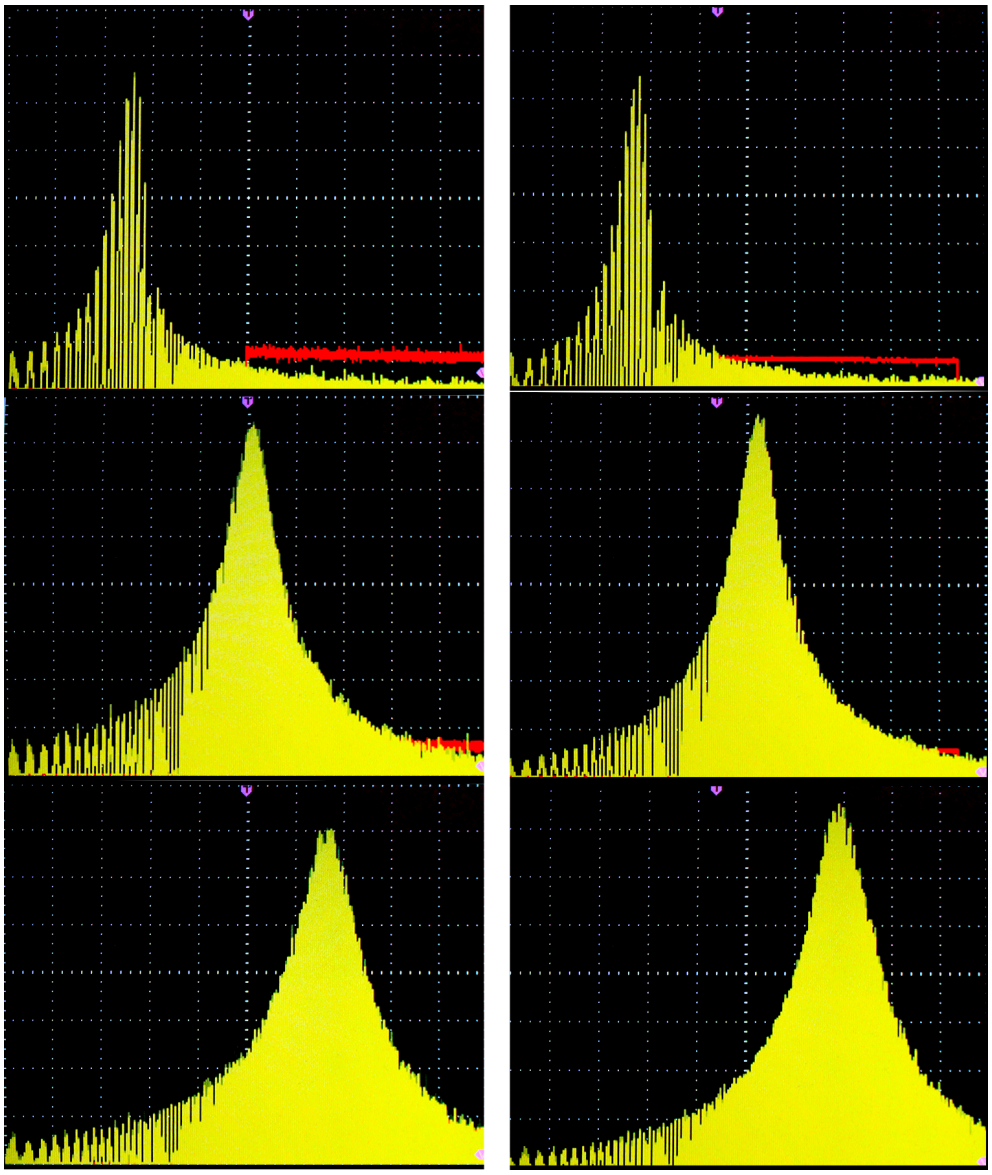
Az 5.1. ábrán az első oszlopban az analóg, a második oszlopban a digitális effekt átvitelét vizsgáljuk legkisebb, közepes és legnagyobb pedálállás esetén. A rendszer átvitelének a sárga jel burkolója felel meg, a piros jel, ami egyes képeken a sárga mögött látható a triggerelést megkönnyítő jel.

Az átvitelek 100 *Hz* és 10 *kHz* között vannak logaritmikus skálán ábrázolva. Megfigyelhető, hogy a jobb oldali digitális effekt esetén a triggerelési pont el van csúsztatva balra, hogy kompenzálja a digitális jelfeldolgozásból adódó késleltetést (ami jelen esetben 15.8 *ms*), és az rendszer átvitele a képernyő közepére kerüljön.

Ha összehasonlítjuk a jobb és bal oldalt, akkor elmondhatjuk, hogy a két rendszernek közel azonos az átvitele. Alaposabban szemügyre véve azonban feltűnnek apró eltérések:

A digitális effektnek kissé keskenyebb a karakterisztikája, élesebben emeli ki az adott frekvenciasávot, valamint nagyobb frekvenciáknál az erősítése valamivel nagyobb, mint az analóg pedálnak.

Ezek az apró eltérések a mellékelt hangminták meghallgatásánál is kivehetőek. Az analóg effekttel szűrt jel melegebb hangon szól, több mélyebb komponense van, lágyabb a hatása, mint a digitális esetben. Mindkét felvétel azonos beállítások mellett készült, azonos hangfelvételt szűrt meg a két effekt. Egyedül a pedálmozgás nem azonos, de igyekeztem

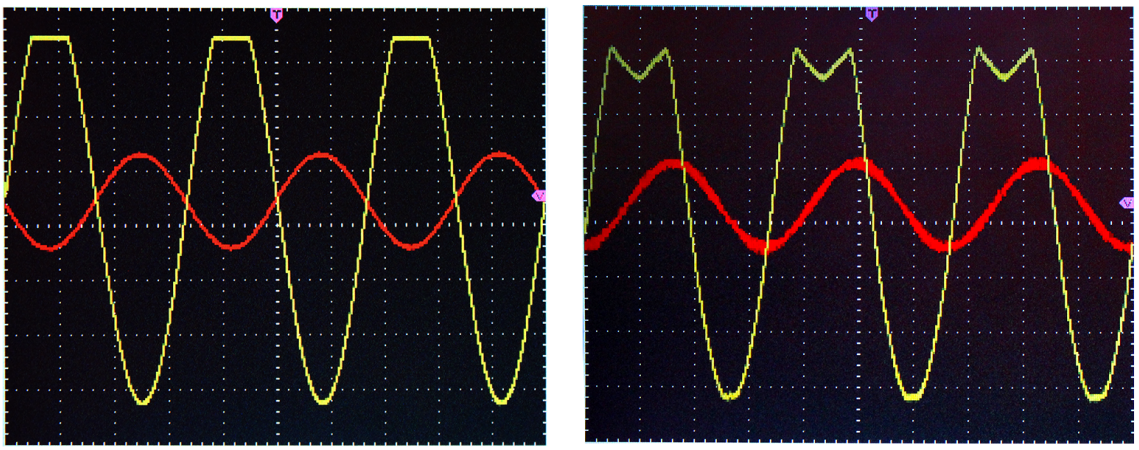


5.1. ábra. *Analóg és digitális rendszer átvitele legkisebb, közepes és legnagyobb pedálállás esetén.*

hasonlóan szabályozni őket. Az analóg felvételen a kattogást a kopott potenciométer okozta.

A sávszűrő összehasonlítása után nézzük meg, hogy hogyan szerepel a nemlineáris egység a megvalósított kétszeres interpolációval. Ugyan úgy, mint az 1.4.6 fejezetben, most is egy szinuszos 600 Hz-es 1.78 V_{pp} jelet adtunk mind az analóg, mind az iOS effektre. A mérések a 5.2. ábrán láthatóak.

Megfigyelhető, hogy digitális esetben a kétszeres túlmintavételezés miatt belapolódnak frekvenciakomponensek és elrontják a hullámformát. Ez nem nevezhető megfelelő megoldásnak, de az iPod sajnos csak erre képes.



5.2. ábra. *Nemlineáris viselkedés összehasonlítása (analóg, digitális).*

5.2. Zárás, összegzés, fejlesztési lehetőségek

Az első fejezetben megismertük az analóg effekt működését, kapcsolási rajzát és a valós áramkörét. Részegységekre bontottuk, felírtuk az egyes részegységek átvitelét majd ellenőrzésképpen bemértük az áramkört.

A második fejezetben áttértünk digitális tartományba. Az első fejezetben meghatározott blokkokat áttranszformáltuk digitálisan megvalósítható blokkokká. Megismertük a digitalizálásra használatos módszereket.

A harmadik fejezetben átnéztük, hogy a céleszköz milyen feldolgozási szolgáltatásokat képes nyújtani, majd a negyedik fejezetben ténylegesen megvalósítottuk az effektet az eszközön. A fejezetben leírt módszerek alapján került beprogramozásra az iPod, azonban nem várt hibába ütköztünk: nem volt elég erős a hardver a jó minőségű algoritmushoz, ezért a követelményekből vissza kellett venni.

Nem gondoltam volna, hogy ilyen rosszul fog szerepelni számítási kapacitásban az iPod, de ha a nemlineáris blokkot nem vesszük figyelembe, akkor még így is igen hűen képes modellezni az eredeti wah-wah pedál hangját. Összességében sikeresnek mondható a feladat megoldása, mert a kidolgozott algoritmusokat egy erősebb hardverbe átültetve remek eredmény érhető el.

Összeségében elmondható, hogy az elméleti modell alapján felépített digitális rendszer igen nagyfokú hasonlóságot mutat az eredeti analóg rendszerrel. Túlvezérlés nélküli használat esetén hűen visszaadja az WH-10-re jellemző karakterisztikus hangzást. Véleményem szerint, ha kellően erős jelfeldolgozó rendszerben lenne implementálva az effekt, akkor jó minőségű, az eredeti pedállal szinte egyenértékű digitális wah-wah effektet kapnánk.

Fejlesztési lehetőség a még pontosabb modellezés érdekében a nemlineáris feldolgozásban van. A mostani egyszerű megoldás ugyan kielégítő eredménnyel szolgál, azonban nem a legszofisztikáltabb megoldás. Ezen kívül a teljes digitális modell az kapcsolási rajz elemzése útján lett felépítve. Ha teljesen azonos digitális effektet szeretnénk készíteni, akkor a már meglévő rendszert további mérések útján az analóg rendszerhez kéne hangolni.

A szakdolgozathoz mellékelve megtalálható egy MATLAB program, ami egy bemeneti wav fájlon elvégzi a szűrést a megadott paramétereknek megfelelően. Ezzel bárki, akinek kedve tartja kipróbálhatja - igaz csak offline módon - az effektet működés közben, tetszőlegesen felparaméterezve azt.

Irodalomjegyzék

[1] Ronald W. Schafer Alan V. Oppenheim. *Discrete-Time Signal Processing, Second Edition*. Prentice-Hall, 1999.

[2] Vladimir P. Russakov Alexander L. Andreyev, Dmitry I. Balyakhov. The Technique of High-Level Optimization of DSP Algorithms Implementation. *The Proceedings of The 7th International Conference on Signal Processing Applications & Technology*, 1(1):826–830, October 1996.

[3] Apple. Audio Mixer (MixerHost), 2010 December. `https://developer.apple.com/library/ios/#samplecode/MixerHost/Introduction/Intro.html#//apple_ref/doc/uid/DTS40010210`.

[4] Apple. Audio Unit Hosting Guideline for iOS, 2009 September. `https://developer.apple.com/library/ios/#documentation/MusicAudio/Conceptual/AudioUnitHostingGuide_iOS/Introduction/Introduction.html#//apple_ref/doc/uid/TP40009492`.

[5] Apple. Audio Unit Processing Graph Service Reference, 2010 July. `https://developer.apple.com/library/ios/#documentation/AudioToolbox/Reference/AUGraphServicesReference/Reference/reference.html#//apple_ref/doc/uid/TP40007289`.

[6] Apple. Audio Unit Property Reference, 2010 December. `https://developer.apple.com/library/ios/#documentation/AudioUnit/Reference/AudioUnitPropertiesReference/Reference/reference.html#//apple_ref/doc/uid/TP40007288`.

[7] Apple. Aurio Touch, 2010 July. `https://developer.apple.com/library/ios/#samplecode/aurioTouch2/Introduction/Intro.html#//apple_ref/doc/uid/DTS40011369`.

[8] Apple. Core Audio Data Types Reference, 2011 October. `https://developer.apple.com/library/ios/#documentation/MusicAudio/Reference/CoreAudioDataTypesRef/Reference/reference.html#//apple_ref/doc/uid/TP40004488`.

[9] Apple. Core Audio Framework Reference, 2008 July. https://developer.apple.com/library/ios/#documentation/MusicAudio/Reference/CACoreAudioReference/_index.html#//apple_ref/doc/uid/TP40002090.

[10] Apple. Core Audio Glossary, 2010 August. https://developer.apple.com/library/ios/#documentation/MusicAudio/Reference/CoreAudioGlossary/Introduction/Introduction.html#//apple_ref/doc/uid/TP40004453.

[11] Apple. iPhone Multichannel Mixer Test, 2010 December. https://developer.apple.com/library/ios/#samplecode/iPhoneMultichannelMixerTest/Introduction/Intro.html#//apple_ref/doc/uid/DTS40009113.

[12] Kevin Avila Chris Adamson. *Learning Core Audio - A Hands-On Guide to Audio Programming for Mac and iOS*. Addison-Wesley, 2012.

[13] CircuitLab. CircuitLab, sketch, simulate, and share schematics. https://www.circuitlab.com.

[14] John Cooper. Simple JFET Preamp for an iDevice Guitar Interface. http://www.planetz.com/simple-jfet-preamp-for-an-idevice-guitar-interface.

[15] Analog Devices. Precision Rail-to-Rail Amplifiers. http://www.analog.com/en/precision-op-amps/precision-rail-to-rail-amplifiers/products/index.html.

[16] Fodor György. *Hálózatok és Rendszerek*. Műegyetemi Kiadó, 2006.

[17] Dirk Hendrik. Ibanez WH10v2 dualpot schematic. http://www.dirk-hendrik.com/wh10dualpot.pdf.

[18] Ibanez. Ibanez WH10v2 manual. http://www.ibanez.co.jp/world/manual/effects/WH10V2.pdf.

[19] R.G. Keen. The Technology of Wah Pedals. http://www.geofex.com/article_folders/wahpedl/wahped.htm.

[20] Kev. Converting 8.24 bit samples in CoreAudio on iOS. http://www.kevatron.co.uk/tag/core-audio/.

[21] Sophoncles J. Orfanidis. Digital parametric equalizer design with prescribed nyquist-frequency gain. *J. Audio Eng. Soc*, 45(6):444–455, 1997.

[22] C. S. Burrus t. W. Parks. *Digital Filter Design*. Prentice-Hall, 1999.

[23] Daniel B. Turek. Design of Effycient Digital Interpolation Filters for Integer Upsampling. http://www.rle.mit.edu/dspg/documents/main.pdf.

[24] Daniel B. Turek. *Design of Effcient Digital Interpolation Filters for Integer Upsampling*. MASSACHUSETTS INSTITUTE OF TECHNOLOGY, June 2004. URL: http://www.rle.mit.edu/dspg/documents/main.pdf.

Függelék

F.1. Sávszűrő átvitelének számítása komplex frekvencia tartományban

A kiindulási egyenletrendszer:

$$\frac{\phi - U_{be}}{R_1} + \frac{\phi - 0}{R_2} + \frac{\phi - 0}{1/sC_1} + \frac{\phi - U_{ki}}{1/sC_2} = 0 \tag{F.1.1}$$

$$\frac{0 - \phi}{1/sC_1} + \frac{0 - U_{ki}}{R3} = 0 \tag{F.1.2}$$

A második egyenletből φ kifejezve:

$$\phi = \frac{-U_{ki}}{sR_3C_1} \tag{F.1.3}$$

Visszahelyettesítve az első egyenletbe:

$$\frac{\frac{-U_{ki}}{sR_3C_1} - U_{be}}{R_1} + \frac{\frac{-U_{ki}}{sR_3C_1}}{R2} + \frac{\frac{-U_{ki}}{sR_3C_1}}{1/sC_1} + \frac{\frac{-U_{ki}}{sR_3C_1} - U_{ki}}{1/sC_2} = 0 \tag{F.1.4}$$

$$\frac{-U_{ki}}{sR_1R_3C_1} - \frac{U_{be}}{R1} - \frac{U_{ki}}{sR_2R_3C_1} - \frac{U_{ki}}{R_3} - \frac{U_{ki}C_2}{R_3C_1} - U_{ki}sC_2 = 0 \tag{F.1.5}$$

$$U_{ki}\left(\frac{-1}{sR_1R_2C_1} - \frac{1}{sR_2R_3C_1} - \frac{1}{R_3} - \frac{C_2}{R_3C_1} - sC_2\right) = U_{be}\left(\frac{1}{R_1}\right) \tag{F.1.6}$$

$$U_{ki}\left(\frac{-R_2 - R_1 - sR_1R_2C_1 - sR_1R_2C_2 - s^2R_1R_2R_3C_1C_2}{sR_1R_2R_3C_1}\right) = U_{be}\left(\frac{1}{R_1}\right) \tag{F.1.7}$$

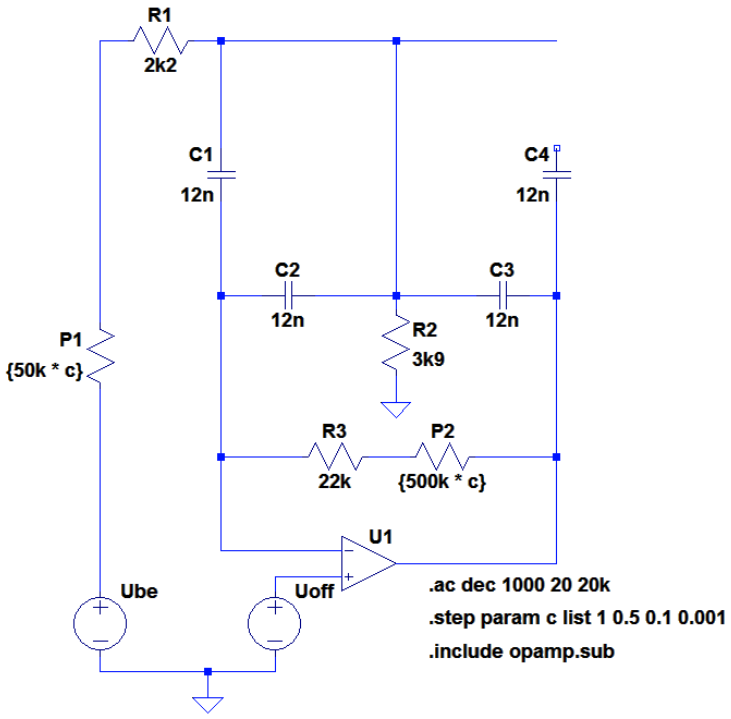
Átrendezve és egyszerűsítve:

$$H(s) = \frac{U_{ki}}{U_{be}} = \frac{s(R_2R_3C_1)}{s^2(-R_1R_2R_3C_1C_2) + s(-R_1R_2(C_1 + C_2)) + (-R_1 - R_2)} \tag{F.1.8}$$

F.2. LTSpice használata változó paraméterű számításokra

Az LTSpice remek eszköz áramkörök gyors elemzésére. Az alap funkciói eléggé kézenfekvők, a felhasználói interfészről könnyen elérhetők, viszont kevesen ismerik néhány nagyon hasznos, kezelőfelületről nem elérhető képességeit.

Ha egy elemzés során egy paramétert változtatni szeretnénk, és a különböző értékekre



F.2.1. ábra. *LTSpice konfigurációja az átvitel vizsgálatához*

vonatkozó átvitelt egy ábrán szeretnénk ábrázolni, akkor ehhez egy direktívát kell alkalmaznunk.

Ennek a leggyorsabb módja az 'S' billentyű leütése (SPICE Directive). A megjelenő ablakban beírjuk a parancsunkat:

```
.step param c list 1 0.5 0.1 0.001
```

A parancs jelentése: a `c` paraméter a vizsgálat közben vegye fel sorra az `1, 0.5, 0.1` és `0.001` értékeket. A `.step` parancs jelenti a léptetést. A változtatandó paramétert a `param` szó után kell megadni. A környezetben minden paraméter változtatható. Speciális esetként, ha a szimuláció hőmérsékletét akarjuk léptetni, akkor a `param` parancsot elhagyva, rögtön a `temp` kulcsszóval folytatva tehetjük meg. A `list` jelentése, hogy ezt követően fel fogjuk sorolni a kívánt értékeket.

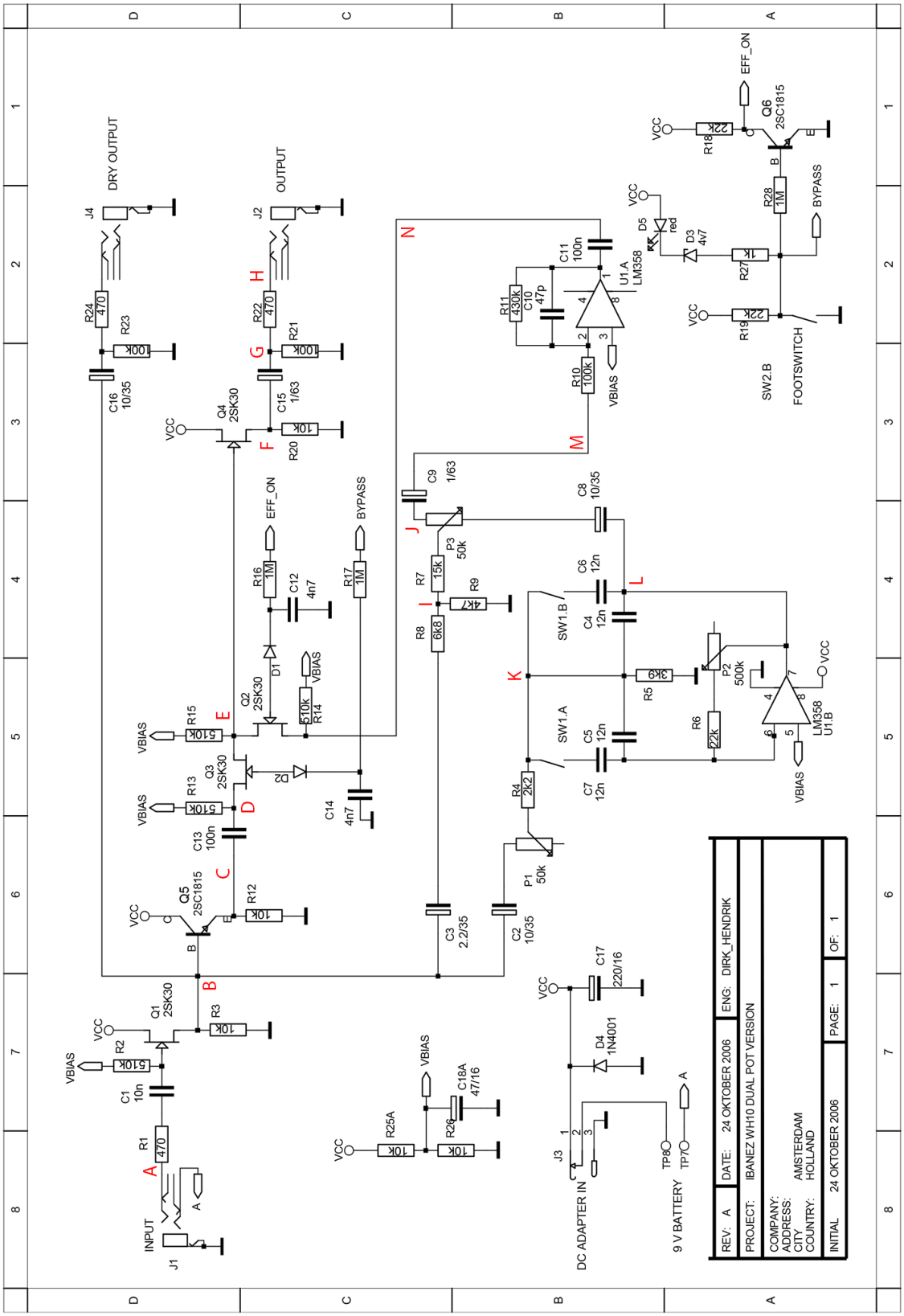
Ha egy paramétert változtatunk, akkor azt bármelyik alkatrész, bármelyik paraméterének értékül tudjuk adni. Ehhez kapcsos zárójeleket kell alkalmazni:

```
{50k*c}
```

Ha ez egy ellenállás értékének van megadva, akkor a szimuláció során a `c` paraméter helyére a Spice minden megadott értéket behelyettesít, és újraszámolja az áramkört.

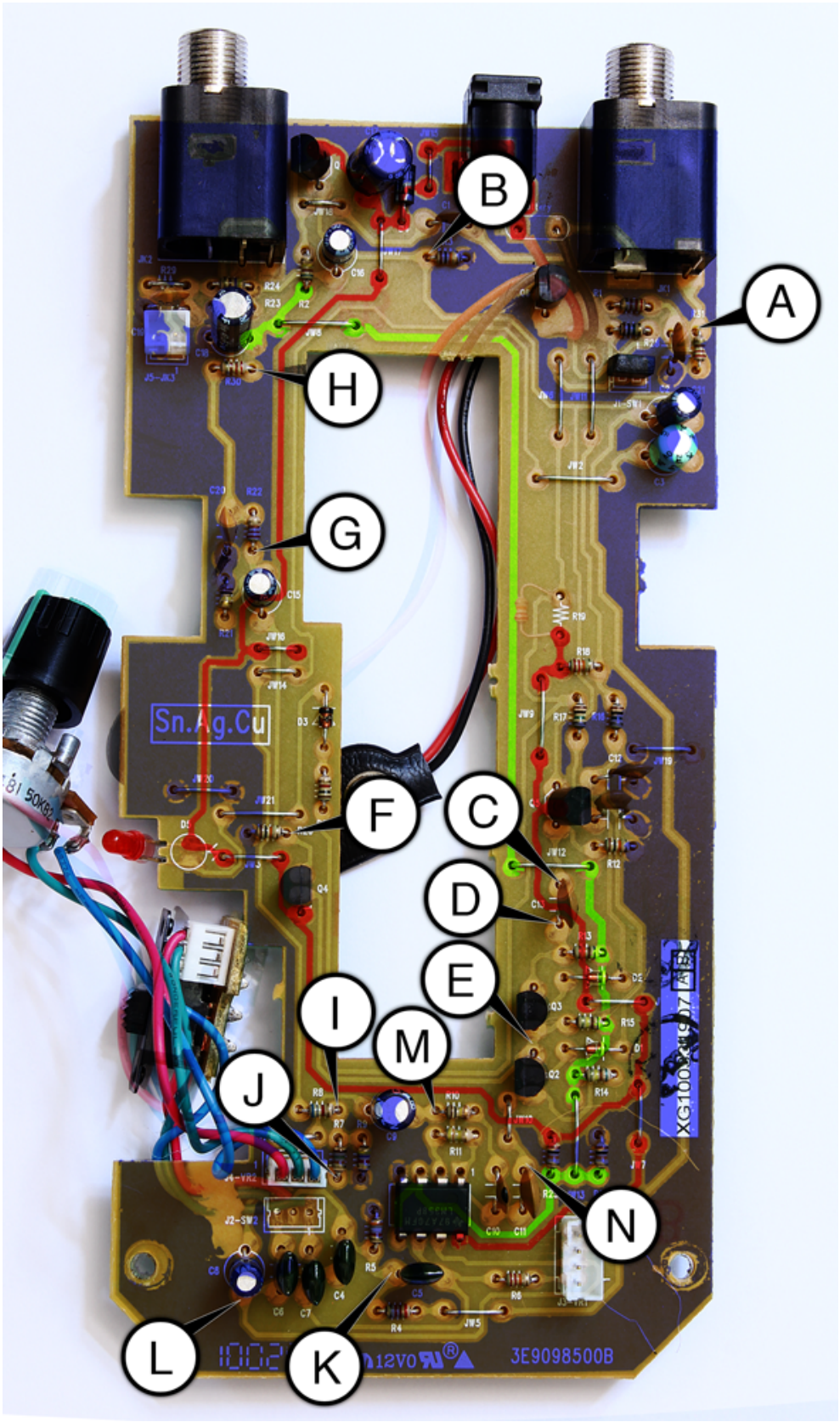
Ezzel a módszerrel határoztuk meg a sávszűrő átvitelét több potenciométer-állásra az 1.4.4 fejezetben.

F.3. WH-10 dupla potenciométeres kapcsolás kiegészítve a bemérési pontokkal



F.3.1. ábra. *WH-10 wah-wah kapcsolási rajz*

F.4. Bemérésre használt mérési pontok elhelyezkedése



F.4.1. ábra. *WH-10 wah-wah áramköri bemérési pontok*

F.5. Analóg szűrőt optimálisan modellező szűrőtervezési algoritmus

```
% peq.m - Parametric EQ with matching gain at Nyquist frequency
%
% Usage: [b, a, G1] = peq(G0, G, GB, w0, Dw)
%
% G0 = reference gain at DC
% G = boost/cut gain
% GB = bandwidth gain
%
% w0 = center frequency in rads/sample
% Dw = bandwidth in rads/sample
%
% b = [b0, b1, b2] = numerator coefficients
% a = [1, a1, a2] = denominator coefficients
% G1 = Nyquist-frequency gain
%
% Available from: www.ece.rutgers.edu/~orfanidi/intro2sp/mdir/peq.m
function [b, a, G1] = peq(G0, G, GB, w0, Dw)
F = abs(G^2 - GB^2);
G00 = abs(G^2 - G0^2);
F00 = abs(GB^2 - G0^2);
num = G0^2 * (w0^2 - pi^2)^2 + G^2 * F00 * pi^2 * Dw^2 / F;
den = (w0^2 - pi^2)^2 + F00 * pi^2 * Dw^2 / F;
G1 = sqrt(num/den);
G01 = abs(G^2 - G0*G1);
G11 = abs(G^2 - G1^2);
F01 = abs(GB^2 - G0*G1);
F11 = abs(GB^2 - G1^2);
W2 = sqrt(G11 / G00) * tan(w0/2)^2;
DW = (1 + sqrt(F00 / F11) * W2) * tan(Dw/2);
C = F11 * DW^2 - 2 * W2 * (F01 - sqrt(F00 * F11));
D = 2 * W2 * (G01 - sqrt(G00 * G11));
A = sqrt((C + D) / F);
B = sqrt((G^2 * C + GB^2 * D) / F);
b = [(G1 + G0*W2 + B), -2*(G1 - G0*W2), (G1 - B + G0*W2)] / (1 + W2 + A);
a = [1, [-2*(1 - W2), (1 + W2 - A)] / (1 + W2 + A)];
```

F.6. Analóg szűrőhöz jól közelítő digitális szűrő paraméterszámítása

A 2.2.2 fejezetben megismert szűrő valós idejű módosításához egy paraméterezési módszert kell készíteni, ugyanis a F.5 függvénynek más paramétereket kell megadni, mint amikkel az eddigi képletek alapján rendelkezünk.

A szükséges paraméterek: $\{G_0, G, G_B, w_0, D_w\}$, ahol G_0 a DC átvitel, G a kiemelési átvitel, G_B az átvitel, aminél a kiemelés szélességét mérjük, w_0 a kiemelési frekvencia, D_w pedig a kiemelés szélessége.

A módszer működése során úgy vesszük, hogy a DC erősítés mindig 0, a kiemelési átvitel gitárnál és basszusgitárnál is egyaránt 5, a kiemelés szélességét 3-as erősítésnél mértük. Ezzel az öt paraméterből csak kettő maradt, amiket kézzel ki kell mérni és letárolni bizonyos beállításokra.

Két lehetőségünk van:

Minden egyes pedálállásra leolvassuk ezeket a paramétereket, kiszámoljuk rájuk a konkrét szűrőparamétereket, majd az értékeket egy táblázatba letároljuk, és az effekt futása közben a szükséges értékeket elővesszük. Nagyon gyors módszer, de ha azt vesszük, hogy egy pedálálláshoz tartozó paraméterkészlet megkeresése (kézzel kell görbeillesztési feladatot végrehajtani, esetenként több lépéses iterációval) igen sok időbe kerül, és egy jól használható paraméterkészlethez sok (legalább 128 vagy 256 paraméterkészletet) kéne letárolni gitár és basszusgitár üzemmódban is, akkor kiderül, hogy lehet nem ez a legkézreállóbb megoldás a feladatra.

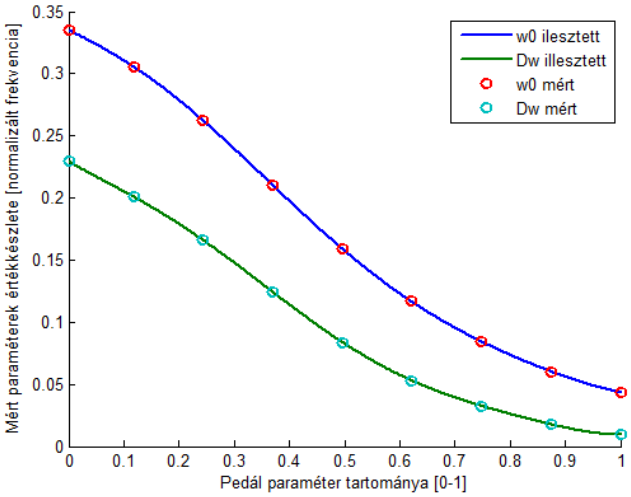
A másik lehetőségünk, hogy néhány pontban lemérjük a paraméterkészletet, a mért értékekre görbét illesztünk, így zárt formában tudjuk számolni az adott függvényhez tartozó paramétereket.

Mi a második megoldást választjuk. Tudjuk, hogy a pedál állását szimbolizáló paraméter 0 és 1 közötti értékeket vehet fel. Ebben a tartományban megvizsgáltunk 9 paraméterkészletet, egyenletesen elosztva az egyes mérési pontokat a tartományon, úgy, hogy MATLAB-ban ábrázoltuk az analóg effekt átvitelét s tartományban, majd az ábrán megkerestük a kiemelési frekvenciát. Ezt beírva az F.5 függvénybe, az analóg és a digitális átvitel kiemelése azonos frekvenciára került. Már csak a kiemelés szélességét kellett lemérni, ahol az átvitel metszi a 3-szoros erősítést jelző vízszintes egyenest, majd ezt a paramétert is beírtuk a függvénybe, és az analóg és a digitális átvitel jó közelítéssel egybeesett. Az így kapott más-más pedálálláshoz tartozó kiemelési frekvenciákat és kiemelési szélességeket ábrázoltuk egy ábrán, majd a kapott pontokra egy heted fokú görbét illesztettünk (a hatodfokú még nem közelítette eléggé a mérési pontokat, a nyolcad fokú pedig szakaszokon már hullámossá vált). Az illesztést a MATLAB *polyfit()* függvényével végeztük.

A mérés és illesztés végén kaptunk két-két polinomot, w_0 és D_w paraméterekre, gitár és basszusgitár üzemmódra. Ezek után nincs más dolgunk, mint az éppen aktuális pedállállást behelyettesíteni ezekbe a polinomokba, majd a kapott értékeket átadni a szűrőegyütthatókat számoló függvénynek, ami kiszámolja az éppen aktuális együtthatókat. Jóval elegánsabb megoldás, mint a teljes tartomány letárolása, és ráadásul folytonosan kapjuk meg a szükséges paramétereket, nem kell az egyes minták között interpolálni.

F.7. Optimális FIR-szűrő megvalósítás memóriatükrözéssel, és for ciklus helyettesítéssel

Olyan rendszereken, melyek nem rendelkeznek DSP képességekkel, a FIR-szűrők megvalósítása igen számításigényes feladatnak számít. Új minta betöltése a cirkuláris bufferbe, cirkuláris bufferen átfutva végigszorozgatni a szűrőegyütthatókkal, az eredményt letárolni, miközben végig figyelni kell, hogy nem léptük-e túl a cirkuláris buffer határát. Ez a



F.6.1. ábra. *Mérési pontokra illesztett görbék.*

folyamatos figyelés (if vizsgálódás) nagyon rossz hatással van a modern processzorok sebességéhez erősen hozzájáruló pipeline rendszer működésére. Minden feltételvizsgálatnál a pipeline-ba betöltött utasítások vagy jók lesznek, a program arra folytatja az útját, vagy rosszak lesznek, és az egész pipeline-t ki kell üríteni, és újra elkezdeni éleszteni. A processzor nem képes kihasználni a fő gyorsítási rendszerét.

Hasonló eseménysor játszódik le egy for ciklus esetében is. Minden ciklus lefutása után meg kell vizsgálni, hogy a futóváltozó átlépte-e a kiírt határt vagy nem. A pipeline vagy kiürül, vagy nem.

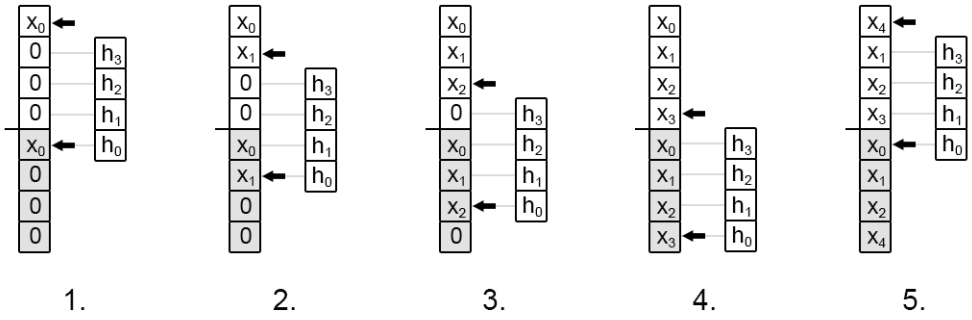
A következőkben bemutatom a memóriatükrözésnek nevezett eljárást, ami képes jelentősen felgyorsítani a cirkuláris bufferek kezelését.

Az ötlet onnan jött, hogy a FIR-szűrők megvalósításánál nem a memória, hanem a feldolgozási erőforrás a szűk keresztmetszet. Mi lenne ha beáldoznánk egy kis memóriát, konkrétan kétszer annyi memóriát használnánk a cirkuláris bufferünkhüz, hogy sebességbeli előnyhöz jussunk?

A módszer kétféleképpen is megvalósítható. Az egyikben előrefele, a másikban hátrafele lépkedünk a cirkuláris bufferben, ennek megfelelően a szűrő együtthatóit tároló tömbben növekvő vagy csökkenő sorrendben helyezkednek el a együtthatók. Mi az előrefele lépkedős módszert választjuk.

Van egy FIR-szűrőnk, amit szeretnénk egy rendszeren belül megvalósítani. Tudjuk, hogy a szűrő együtthatóinak a száma N. A rendszerben lefoglalunk egy 2N nagyságú memóriarészt. Ez lesz a cirkuláris bufferünk. A buffer címzésére és a beírásra egy darab mutatót használunk, ennek neve legyen p_w. Ennek a lehetséges értékei N és 2N-1 között változhatnak.

Az F.7.1. ábra segítségével nézzük meg a memóriatükrözéses cirkuláris buffer működését N=4 esetére!



F.7.1. ábra. *Memóriatükrözéses cirkuláris buffer működése*

Látható, hogy minden egyes lépésnél az újabb bemeneti minta egyszerre két helyre íródik be: p_w és $p_w - N$ memóriacímekre (a rendszer indulásakor p_w értéke N). A két egyforma memóriaterület megoldja, hogy a kettő közti határ átlépésekor a következő cím úgy viselkedik, mintha a cirkuláris buffer átfordult volna, holott csak továbbléptünk egy lépésnyit.

Minden bejövő mintára lefuttatjuk a szűrőt, azaz p_w mutatótól (beleértve p_w-t is) N lépésben kiszámoljuk a FIR szűrő kimenetét. Ezt megtehetjük for ciklussal is, de ha igazán hatékonyak akarunk lenni, akkor hagyományos (nem DSP) architektúrán nem, vagy csak ritkán használjunk feltételvizsgálást igénylő for ciklusokat.

Plusz memória befektetésével elég jelentősen fel lehet gyorsítani a feldolgozást [2], akár 10-20 százalékkal is.

Még tovább optimalizálható a kódunk magas szinten, ha alkalmazzuk a for-ciklusok kicsomagolását [2]. Ennek lényege, hogy ahelyett, hogy a processzorra bíznánk minden indexelési feladatot, mi oldjuk meg egy részüket. Ezzel lehetővé tesszük a compilernek, hogy jobban ki tudja optimalizálni a kódunkat. A módszer akár 60-70 százalékot is képes gyorsítani a programunkon [2].

A következőkben két C nyelven írt kódrészletben mutatom be a fenti módszerek megvalósítását:

```
// hagyományos, nem optimális for ciklus
for(int i=0; i<N; i++) {
        sum += x[i] * h[i];
}
// kétszeresen kicsomagolt for-ciklus, 10-15%-os gyorsulás
for(int i=0; i<N; i+=2) {
        sum += x[i] * h[i] + x[i+1] * h[i+1];
}
// kétszeresen kicsomagolt for-ciklus, 60-70%-os gyorsulás
for(int i=0; i<N; i+=8) {
        sum += x[i] * h[i] + x[i+1] * h[i+1] +
        x[i+2] * h[i+2] + x[i+3] * h[i+3] +
        x[i+4] * h[i+4] + x[i+5] * h[i+5] +
        x[i+6] * h[i+6] + x[i+7] * h[i+7];
}
```

F.8. Singleton osztályok és szerepük az objektumorientált programozásban

iOS környezetben, ha használni szeretnénk egy objektumot, akkor rendelkeznünk kell egy mutatóval, ami az adott objektumra mutat. Ha egy forráskódban több helyen is szeretnénk alkalmazni egy bizonyos objektumot, akkor erre két lehetőségünk van:

A program vezérlésének a futása alatt mindig magunkkal cipeljük az objektumra mutató mutatót, mindig átadva függvények, és objektumok között. Ezzel egy nagyon behálózott kódot hozunk létre, amit nagyon nehéz karbantartani vagy bővíteni, hiszen, ha az átadás csak egy helyen is megszakad, akkor az azt követő kódrészletek már nem fogják elérni az szükséges objektumot.

A másik lehetőség, hogy a szükséges objektumot singleton osztályként valósítjuk meg. Ennek a lényege, hogy csak egy objektum példányosodik az osztályból, amire mutató mutatóval maga az osztály tud szolgálni egy statikus tagfüggvényen keresztül.

Ezután ha bárhol a kódunkban használni szeretnénk az objektumot, akkor az osztályától el kell kérni a rá mutató mutatót.
Egyszerűbb és jól karbantartható kód kapható így.

F.9. Rendszer átvitelének mérése oszcilloszkóppal

Ha nem áll rendelkezésünkre más eszköz, mint egy oszcilloszkóp, és meg szeretnénk mérni egy rendszer frekvenciatartománybeli átvitelét, akkor azt ügyes gerjesztéssel megtehetjük.

Ehhez nem kell mást tenni, mint előállítani egy olyan szinuszos gerjesztő jelet, aminek a frekvenciája folyamatosan nő (úgynevezett *chirp* jel) egy kiindulási értéktől egy meghatározott frekvenciáig. Ez a két érték lesz az átvitel ábrázolásának a két határa. A jel hosszának akkora időtartamot kell beállítani, amekkora az oszcilloszkóp képernyőjén megjelenített időszelet (25 *ms/osztás* esetén, ha a képernyő 10 osztásból áll, ez 250 *ms*-nak felel meg).

Ezt a jelet ráadva a rendszerre, a kimeneti jel burkolója visszaadja az adott rendszer átvitelét.

Oszcilloszkóppal befogható a rendszer válasza, ha a gerjesztést periodikusan ismételve adjuk ki. A vizsgált csatornának a középértékét érdemes teljesen levinni a képernyő aljához, hogy a kimeneti jel felső felét lássuk csak.

A befogás minősége javítható, ha gerjesztő jellel együtt kiadunk egy trigger jelet is (négyszögjel, periódusideje a gerjesztő jel hossza), és erre triggerelünk az oszcilloszkóppal.

A módszer hátránya, hogy a függőleges tengely nem decibelben, hanem feszültségben adott, ezért a kapott ábra némileg eltér a megszokott ábrázolásmódtól.

A vízszintes tengely tetszőlegesen lehet lineáris vagy logaritmikus is, attól függően, hogy a gerjesztő jel lineárisan vagy logaritmikusan növeli a jel frekvenciáját.