Diplomaterv

Audio jelek zajcsökkentése szűrőbankok segítségével

Bodor József
2007

Nyilatkozat

Alulírott, Bodor József, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

……

Bodor József

Tartalomjegyzék

Kivonat 5

Abstract 6

1. Bevezetés 7

2. Régi hanglemezek off-line zajcsökkentése 9

2.1. A sercegések és a kattogások hatásának csökkentése 9

2.2. Az alapzaj hatásának csökkentése 11

2.3. Off-line zajszűrés szűrőbankok segítségével 13

2.4. On-line zajcsökkentés szűrőbankokkal 15

3. A szűrőbankok 17

3.1. Matematikai modell 19

3.2. A megvalósítandó szűrőbank 21

3.3. A prototípusszűrő és a megvalósított szűrőbank 22

3.4. A szűréshez szükséges műveletek számának csökkentése 25

4. A megvalósítandó karakterisztikák 28

4.1. A lépcsős karakterisztika 28

4.2. A hiszterézises karakterisztika 30

4.3. A lineáris elnyomású karakterisztika 31

4.4. A lineáris elnyomású karakterisztika, ofszettel 32

4.5. A négyzetes elnyomású karakterisztika 33

5. A hardver és szoftver eszközök bemutatása 34

5.1. A processzor 34

5.2. A processzormag 35

5.3. Számábrázolási módok 38

5.4. Az EZ-LITE kártya 39

5.5. A szoftver eszközök 39

6. Az algoritmust megvalósító szoftver 41

6.1. A program által megvalósított algoritmus 42

6.2. A szűrés implementálása 43

6.3. A teljesítményablak 43

6.4. A lépcsős karakterisztika megvalósítása **45**
6.5. A hiszterézises karakterisztika **46**
6.6. A lineáris karakterisztika ofszettel és ofszet nélkül **46**
6.7. A kimenet előállítása **48**
7. Mérési eredmények, kiértékelés **49**
7.1. Alapjelek tisztítása **49**
7.2. Régi hanglemezfelvételek zajának csökkentése **52**
7.3. Zajjal terhelt zenei jel tisztítása **53**
7.4. Beszédhang tisztítása **54**
7.5. Beszédhang tisztítása valós körülmények között **56**
7.6. A szűrőbank, mint hangolható szűrő **57**
8. Összefoglalás, kitekintés **59**
9. Irodalomjegyzék **60**
10. Függelék **61**

Kivonat

Audio alkalmazások esetén gyakran a hasznos jelet nagyteljesítményű zaj terheli. A diplomaterv témája ezeknek a zajoknak a csökkentése olyan helyzetekben, amikor a hasznos jelet terhelő zaj teljesítménye nagy sávszélességen oszlik el. A feldolgozható audio jel sávszélessége maximum 24 kHz lehet. A zajcsökkentő algoritmus digitális, és on-line, vagyis a feldolgozás a jel mintavételezésével egy időben játszódik le. A jelfeldolgozást az Analog Devices cég terméke, a BF-537-es jelfeldolgozó processzor végzi.

A diplomaterv elején megvizsgálok néhány olyan algoritmust, melyet általában off-line zajcsökkentésekhez használnak (az off-line ebben az esetben azt jelenti, hogy a feldolgozás a felvétel elkészülte után történik), majd az általam választott megoldás on-line megvalósításának kérdéseivel foglalkozom.

A zajcsökkentés FIR szűrőkből felépülő szűrőbank segítségével történik. A diplomatervben leírom a szűrőbankok legfontosabb tulajdonságait, és alkalmazásuknak okát. Ezek után a megvalósítani kívánt szűrőbank prototípusszűrőjének specifikációja, majd megvalósításának leírása következik.

A zajcsökkentő algoritmus működésének lényege, hogy sávonként vizsgálva a jel teljesítményét, a kisebb teljesítményű frekvenciakomponenseket valamilyen karakterisztika szerint elnyomjuk, azzal a feltételezéssel élve, hogy zaj. A szűrőbank bemutatása után e karakterisztikák ismertetése, és a köztük lévő különbségek bemutatása következik.

A továbbiakban a rendelkezésre álló hardver és szoftver eszközökkel, majd az algoritmust megvalósító program sajátosságaival foglalkozom.

Végül az általam végzett mérések kiértékelése következik, melyeket alapjelekre (háromszög, négyszög, szinusz), beszédjelekre, és zenei jelekre végeztem el.

Az algoritmus főleg azokban az esetekben működik jól, mikor a jel-zaj viszony 10 dB-nél nagyobb. Az algoritmus a hasznos jelet is károsítja, ezért meg kell találni azokat a beállításokat, melyek mellett a hasznos jel még érthető marad, de a zaj nagymértékben lecsökken.

Abstract

In case of audio applications, the audio signal is usually corrupted by high power noise. The subject of the thesis is the reduction of these noises, in such cases when the power of the noise that distorts the audio signal has a wideband frequency distribution. The audio signal which my realization processes can have a bandwith of 24 kHz. The noise reduction algorythm is digital, and works on-line, which means that the signal processing happens while sampling. The signal processing is done by the product of Analog Devices Inc., the BF-537 DSP processor .

In the beginning of the thesis, I examine some algorithms, that are usaually used in off-line noise reduction (off-line means in this case, that the processing is done after recording the signal). Afther that, I'm dealing with the aspects of the solution I've chosen.

The noise reduction is done with the use of a FIR type filter bank. In the thesis, I introduce the main aspects of filter banks, and the reason of their application. After this comes the specification of the prototype filter of the filter bank, and the description of the realization.

The matter of the noise reduction algorithm is that the frequency components of the signal is being observed, and those components, that have a lower power, are suppressed with the use of a specific function, by using the assumption that those components belong to the noise. After introducing the filter banks, I'm dealing with the functions I've used, and the differences between them.

Next I introduce the hardware and software tools I've used, and the aspects of the program that realizes the algorithm.

Finally, I summarize the evaluation of the measurements I've made. The measurements were done for basic signals (triangle wave, sine wave, square wave), for speech audio signal, and for music audio signal.

The algorythm worked well primarily in such cases, when the signal-to-noise ratio was higher than 10 dB. The algorithm damages not only the noise but the audio signal, so the settings where the audio signal is still understandable and the noise is supressed must be found with care.

1. Bevezetés

Audio alkalmazások esetében gyakran előfordul, hogy a továbbítandó hasznos jelet nagyteljesítményű, szélessávú zaj terheli. Ennek két fő okozója van. Az egyik, hogy a hang mechanikai tulajdonságait elektromos jellé alakító rendszerbe már eleve mechanikai zajjal szennyezett jel jut. Ez mikrofon esetén lehet a háttérben egy zajforrás, míg például bakelitlemez lejátszása esetében lehet a lemezen lévő fizikai sérülések összessége. A másik zajkeltő tényező a már átalakított jel elektronikus jelútjához kapcsolódik. Ezt főleg a továbbításra használt kábelek, és erősítők zaja okozza. Ezek miatt felmerülhet egy zajcsökkentő rendszer behelyezése az audio jel jelfolyamába.

A digitális úton történő zajcsökkentés nem újkeletű fogalom, már az 1970-es években alkalmazták, főleg zenei jelekre. Azonban a bonyolultabb zajcsökkentő algoritmusok nagy számításigénye miatt ekkor még csak off-line módon lehetett igazán jó eredményeket elérni.

Az algoritmusok on-line megvalósítása a gyors processzorok megjelenésével vált lehetővé. Ma már az olcsó, gyors jelfeldolgozó processzoroknak köszönhetően az on-line zajcsökkentő rendszereket nemcsak a zeneiparban használják, hanem más, olcsóbb alkalmazások esetében is (egy ilyen lehet a beszédjel továbbítása).

A zaj jellegétől, teljesítményétől, és az alkalmazás típusától függően sokféle zajcsökkentő eljárás létezik. Néhány ilyen off-line eljárást mutatok be a 2. fejezetben, majd az on-line megvalósítás sajátosságaival is foglalkozom. Szélessávú zaj esetén az egyik járható út a szűrőbankok alkalmazása. A szűrőbank sávokra bontja a jelet, és a sávokban bizonyos feltételezésekkel élve különféle zajcsökkentő algoritmusokat lehet megvalósítani. A sávokra bontást indokolja, hogy audio jelek esetén az emberi fül nem minden frekvenciakomponenst hall ugyanolyan erősen, vannak fontosabb, és vannak kevésbé fontos komponensek. A szűrőbankok tulajdonságaival a 3. fejezetben, míg a megvalósított algoritmusokkal a 4. fejezetben foglalkozom részletesebben.

A fejlesztést egy ADSP EZ-LITE fejlesztőkártyán végeztem, melynek központi eleme egy BF-537 jelfeldolgozó processzor (a továbbiakban DSP). A kártya és a processzor is az Analog Devices cég terméke, csakúgy, mint a szoftveres fejlesztői környezet, a VisualDSP++. Az 5. fejezetben ezekkel, a feladat megoldásához használt hardver és szoftver eszközökkel foglalkozom részletesebben.

A szoftveres megvalósítás sajátosságait a 6. fejezetben ismertetem, a zajcsökkentő algoritmusokat megvalósító program szerkezetével együtt. A magyarázó szöveggel ellátott forráskód a diplomatervhez mellékelt CD-ROM-on található meg.

A teszteléshez használt mérési elrendezések leírását, és a mérési eredmények értékelését a 7. fejezet tartalmazza. A mérési eredményeket szemléltető hangminták a CD-ROM mellékleten találhatóak meg, .wav formátumban. A függelékben az algoritmus futási eredményei tekinthetők meg, alapjelek (szinusz, háromszög, négyszög) esetére.

2. Régi hanglemezek off-line zajcsökkentése

A régi hanglemezek minőségének javítása meglehetősen összetett, és számításigényes feladat. Az előbbi oka, hogy a zenei jel torzulása, és a mellette megjelenő zaj sok olyan komponensből áll, melyek jellege nagyon eltérő, ezért hatásuk csökkentése, vagy éppen eltávolításuk másfajta metódusokat tesz szükségessé. A régi hanglemezeken megjelenő zajoknak három főbb fajtája van: kattogás, sercegés és alapzaj.

2.1. A sercegések és a kattogások hatásának csökkentése [1][2]

A kattogások impulzusszerű, nagy energiájú zavarok, és az időtartományban vizsgálva viszonylag ritkán fordulnak elő. A lemezen lévő súlyosabb fizikai sérülések okozzák (mélyebb karcolások), hosszuk körülbelül 1 ms. A sercegés sokkal sűrűbben jelenik meg a jelben, és kisebb az energiája, de így is kellőképpen kiemelkedik a jelből, és ropogó hangja van.

Mivel a hosszúságuk összességében sokkal kisebb, mint az egész felvételé, ezért olyan algoritmussal kell csökkenteni a hatásukat, mely csak abban az időintervallumban változtat a jelen, ahol a kattogás, illetve a sercegés megjelenik. Mivel a jel nagy részében nincsenek jelen, ezért felesleges is ezeken a részeken egy olyan algoritmust alkalmazni, ami a sercegésektől mentes jelet leronthatja.

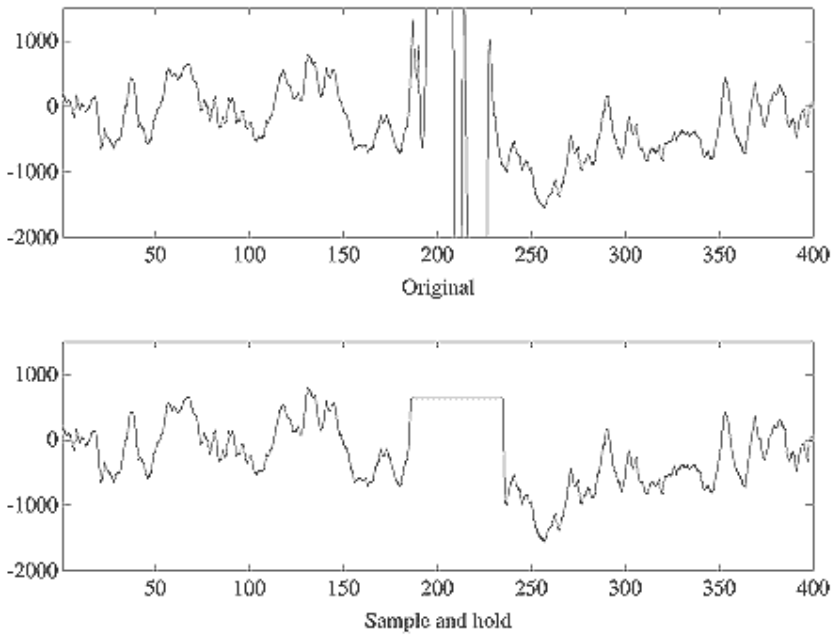
Éppen ezért ezeknek a típusú zavaroknak a csökkentése két fontos részre osztható: az egyik a sercegések, kattogások elejének és végének megkeresése az időtartományban, a másik pedig ezeken a területeken egy zajcsökkentő eljárás alkalmazása.

A zavarjelek határait felülateresztő szűrővel szokás detektálni. Ennek oka, hogy ezeknek az impulzusszerű zajoknak a nagyfrekvenciájú komponensei is elég nagy energiájúak, hiszen a spektrumuk lassan lecsengő, az ugrások miatt. Az algoritmusokat az impulzus időtartománybeli határain kicsit túlnyúlva érdemes végezni.

A detektálás után alkalmazható algoritmusok közül a legegyszerűbb a teljes elnémítás. Ezzel az a baj, hogy nincs esély arra, hogy az eredeti jelet visszaállítsuk ezeken a helyeken. A másik probléma az, hogy itt is ugrás következik be az

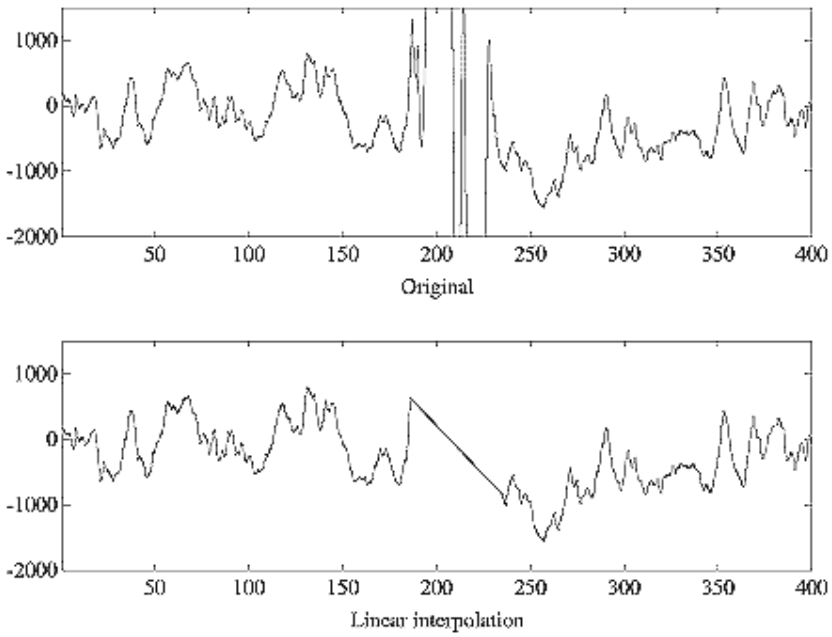
időfüggvényben, ami megint csak pattogó zörejt eredményez. Csak akkor érdemes alkalmazni, hogyha a kattanás energiája sokkal nagyobb, mint a hasznos jel energiája.

A fenti módszert főleg analóg rendszerekben alkalmazzák, de megvalósítható digitálisan is. Ha már digitális környezetben dolgozunk, érdemesebb azonban egy másik metódust megemlíteni, melynek lényege, hogy a teljes elnémítás helyett a határok közt nulladrendű tartót alkalmazunk. Itt a kattanó, sercegő hangok helyett tompább puffanások lesznek hallhatók, melyek energiája lényegesen kisebb, mint az eredeti zavarjelé. Az eredeti jelből több információ marad, mint az előző esetben – igaz, ez a többletinformáció csak egyetlen egy minta (az intervallum bal szélén). Az ábrán [2] ennek az algoritmusnak a hatása figyelhető meg (felül az eredeti, alul a módosított jelalak látható):



2.1. ábra. Nulladrendű interpoláció [2]

Lényegesen jobb eredmény érhető el lineáris interpolációval. Az intervallum két szélén lévő mintákat egy egyenessel kötik össze, ezáltal az eredeti jelből egy mintával több információt őrzünk meg, mint a nulladrendű tartó esetében. Az így kapott hang ráadásul nem annyira kellemetlen, mint az eddigi esetekben. A jel sávszélessége azonban ebben az intervallumban jelentősen lecsökken. Az alábbi ábrán látható az interpolálás hatása a jelalakra; felül az eredeti, alul az interpolált jel látható:



2.2. ábra. Lineáris interpoláció [2]

Ezek az alapvető algoritmusok. Az ezeknél bonyolultabb, de sokkal jobb eredményre vezető eljárások magasabb rendű interpolációt alkalmaznak úgy, hogy felhasználnak a jelről olyan információkat, melyeket a jel egy vagy több olyan szakaszából nyertek, melyek nem tartalmaztak sercegést, vagy kattanást.

A kattogások, sercegések eltávolításához lehet használni mediánszűrést is. Ilyenkor azonban a nagyfrekvenciás komponensek a szűrés helyén kiesnek. Ezeket a komponenseket lehet becsülni, ehhez a szűrt intervallum előtti jelet érdemes felhasználni. Az általam megvalósított algoritmust régi hanglemezfelvételeken is kipróbáltam, melyeket az algoritmus alkalmazása előtt a teljes időtartományban off-line mediánszűrésnek vetettem alá.

2.2. Az alapzaj hatásának csökkentése [1]

A régi hanglemezfelvételeken megjelenő alapzaj szélessávú, sztochasztikus jellegű, és végig jelen van az időtartományban. Az előzőekkel ellentétben tehát a felvétel egészére kell az algoritmust alkalmazni.

Az egyik ilyen algoritmus a frekvenciatartományban működik. A lényege, hogy ha a zajos jel teljesítménysűrűség-spektrumából kivonjuk a zaj teljesítménysűrűség-spektrumát, akkor a kapott jel már az alapzajtól mentes lesz. Természetesen a zaj teljesítménysűrűség-spektrumát nem ismerjük, de fel tudunk rá állítani egy becslőt. Ez a becslő a periodogram:

$$\hat{S}[k] = X[k] \cdot X[k]^* = \left| X^2[k] \right| \tag{2.1}$$

ha

$$X[k] = DFT\{x[n]\} \tag{2.2}$$

vagyis az időtartományban vett mintaregisztrátum diszkrét Fourier-transzformáltja.

A mintaregisztrátumot egy ”csendes” részből kell venni, vagyis szüneteknél, ahol nincs zene, ugyanis az alapzaj a teljes időtartományon egyforma intenzitással van jelen.

Az alapzaj spektruma közel fehér, vagyis 100%-os a varianciája, ezért nem használható közvetlenül a periodogram, mint a zaj teljesítménysűrűség-spektrumának a becslője. A Welch-módszerrel azonban csökkenthető a becslő varianciája; N darab független mintaregisztrátum periodogramjának átlagolása esetén a variancia N-ed részére csökken. Az új becslő:

$$\hat{S}'_{zaj}[k, L] = \frac{1}{N} \cdot \sum_{i=1}^{N} \hat{S}_{i\ zaj}[k, L] \tag{2.3}$$

ahol L a regisztrátum hossza. Ezt ki kell vonni az eredeti jel periodogramjából. Ekkor a szűrt jel teljesítménysűrűség-becslője:

$$\hat{S}_{sz}[k, L] = S_{jel}[k, L] - \hat{S}'_{zaj}[k, L] \tag{2.4}$$

Természetesen az eredeti jel spektrumbecslőjének, az $S_{jel}[k, L]$-nek, még mindig hatalmas a varianciája, ezért az $\hat{S}_{sz}[k, L]$ becslő nem teljesen pontos. Néhány helyen a

spektrumvonalak negatív értéket vesznek fel; ezeken a helyeken 0-val teszik egyenlővé a spektrumvonalakat. Ennek az algoritmusnak soft-elbow a neve. Az így visszamaradt zaj teljesítménye kisebb, mint az alapzaj, de más jellege lesz, ami a hallgató számára zavaróbb lehet, mint az eredeti, fehér jellegű zaj.

A hard-elbow algoritmus a fentihez hasonló, azzal a különbséggel, hogy az eredményspektrumban csak azokat a spektrumvonalakat teszik nullává, ahol a zajos spektrum értéke kisebb, mint az $\hat{S}'_{zaj}[k,L]$, azaz az alapzaj becslője.

2.3. Off-line zajszűrés szűrőbankok segítségével

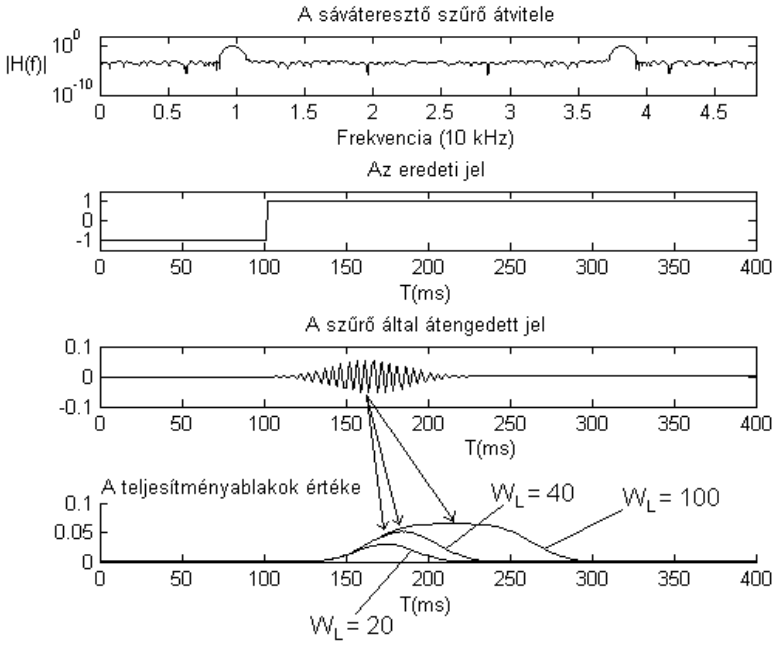
A következőkben ismertetett eljárás a fentebb említett zajtípusok közül az alapzajra alkalmazható eredményesen, ezért az alapzaj szűrésére már fentebb ismertetett algoritmushoz hasonló eredményre vezet. A teljes körű hanganyagjavításhoz tehát ez előtt is érdemes elvégezni a kattogások, sercegések eltávolítására ismertetett algoritmusok valamelyikét.

A szűrőbankokkal történő zajszűrés lényege, hogy a jelet sávokra bontjuk, és sávonként vizsgáljuk a zajos jel teljesítményét. A teljesítmény mérését a minták teljesítményének mozgóablakos átlagolásával végezzük. Tulajdonképpen az összegzett teljesítményt nem osztjuk el a minták számával, a folyamat mégis átlagolásnak tekinthető.

Ha az adott szűrőkimeneten a teljesítmény nem ér el egy bizonyos küszöbszintet, akkor feltételezzük, hogy zaj, és a választott algoritmus és a teljesítményszint függvényében a szűrőkimeneten valamilyen transzformációt hajtunk végre. A szűrőbank és az algoritmusok implementálása után a legfontosabb feladat ezeknek a küszöbszinteknek, és az ablakszélességnek a hangolása. Az algoritmus egyik hátránya, hogy a zajnak az a része, amelyik nagy teljesítményű, frekvenciában hozzá közel lévő (tehát ugyanabban a sávban van) hasznos jelen ül, csillapítás nélkül át fog jutni, ez ellen nem sokat tehetünk. Ebből következik, hogy akkor alkalmazható eredményesen, ha a zaj teljesítménye jóval a jel teljesítménye alatt van.

A teljesítményablak bizonyos mértékű késleltetést okoz, ezért nem szabad a hosszát túl nagyra állítani, mert nem kapunk valós információt a jel aktuális teljesítményéről: olyan minták teljesítménye is ”beleszólhat” az aktuális minta

értékének szabályozásába, amelyik időben sokkal régebben keletkezett, és a mostani mintával esetleg nincs korrelációban. Egy nagy teljesítményű impulzus az adott frekvenciasávban az utána következő minták teljesítményablak-értékét is megnöveli, így azokat az időtartománybeli mintákat is kiadjuk – legyenek azok akármik is. Túl kicsire sem érdemes állítani, mert ekkor a pillanatnyi teljesítménymméréshez közelítünk. A teljesítményablak és a jelalak változása követhető nyomon az alábbi MATLAB szimuláció által létrehozott ábrán, különböző ablakszélesség értékek mellett. A jel egy olyan lassan változó négyszögjel (frekvenciája 10 Hz), amely már átment egy sávátereszto szűrőn. A szűrő FIR típusú, átvitelének középfrekvenciája 9750 Hz, a teljesítményablak szélességét a W_L paraméter jelöli:



2.3. ábra. A teljesítményablak késleltetése az ablakszélesség függvényében

A szűrő által átengedett jel az eredeti jelhez képest a FIR szűrő együtthatóinak felével késik. A teljesítményablak csúcsa pedig ebben az esetben a szűrt jelhez képest a teljesítményablak felével.

A küszöbteljesítmény beállításánál több dolgot is figyelembe kell venni. Először is, a zenei felvételeknek a teljesítménye nem egyenletes eloszlású a frekvenciatartományban. A teljesítmény nagy része általában az alsó 8 kHz-ben van, ott

is csökkenő tendenciát mutat. Az emberi fül is ebben a tartományban hall a legjobban, ezért off-line feldolgozás esetén nem feltétlenül kell a sávokat azonos szélességűre beállítani: ebben a tartományban (beszédsávnak is nevezik) lehet több, vékonyabb szűrőt helyezni, míg az e fölötti frekvenciákon szélesebb, kevesebb szűrőt. Az ilyen beállítás esetén finomabban hangolhatóak a küszöbteljesítmények. Másodszor figyelembe kell azt is venni, hogy ha a hasznos jel a küszöbteljesítmény alatt van, akkor az sem jut át, vagyis a küszöböt úgy kell beállítani, hogy a hasznos jelet minél kisebb mértékben csonkítsa meg az algoritmus. Ilyenkor ezen kívül még a zaj jellege is megváltozik, bár természetesen a teljesítménye csökkenni fog. Meg kell találni tehát az ideális egyensúlyt, a zajcsökkentés és a zene jellegének megőrzése között.

Off-line szűrés esetén a zajról további információkat szerezhetünk. Ha a már fentebb is említett módon, olyan regisztrátumokat veszünk, ahol nincs zene, akkor elkészíthetjük a zaj periodogramját, amelyből kiderül, hogy milyen a frekvenciatartományban a teljesítményének az eloszlása, ezáltal meghatározhatjuk, hogy a sávokban beállított küszöbteljesítmények körülbelül mekkorák kell, hogy legyenek. Megvizsgálhatjuk a zajos jel periodogramját is, mert azokban a sávokban, ahol a jel teljesítménye nagyobb, magasabb küszöbteljesítményszinteket engedhetünk meg, a hasznos jel jellegének nagymértékű megváltozása nélkül.

2.4. On-line zajcsökkentés szűrőbankokkal

On-line zajcsökkentés esetén az előző pontban megismert alapelveket kell szem előtt tartanunk. Az algoritmus célja itt is a szélessávú alapzaj csökkentése lehet, és ha a feladat a régi hanglemezek hangjához hasonló zajos jel on-line megtisztítása, akkor a jelfolyamba egy kattogás-csökkentő rendszer is beépíthető, a szűrőbankos feldolgozás elé.

Az egyik különbség az off-line és az on-line zajcsökkentés között, hogy a zaj jellegéről nincs előzetes információnk – ha mégis van, az nyilván pontatlanabb, mint amit az off-line módszer esetén nyerhetünk. Ez utóbbi egyébként akkor lehetséges, ha ismerjük azt a zajkörnyezetet, aminek jelfolyamába a rendszert be kell illeszteni.

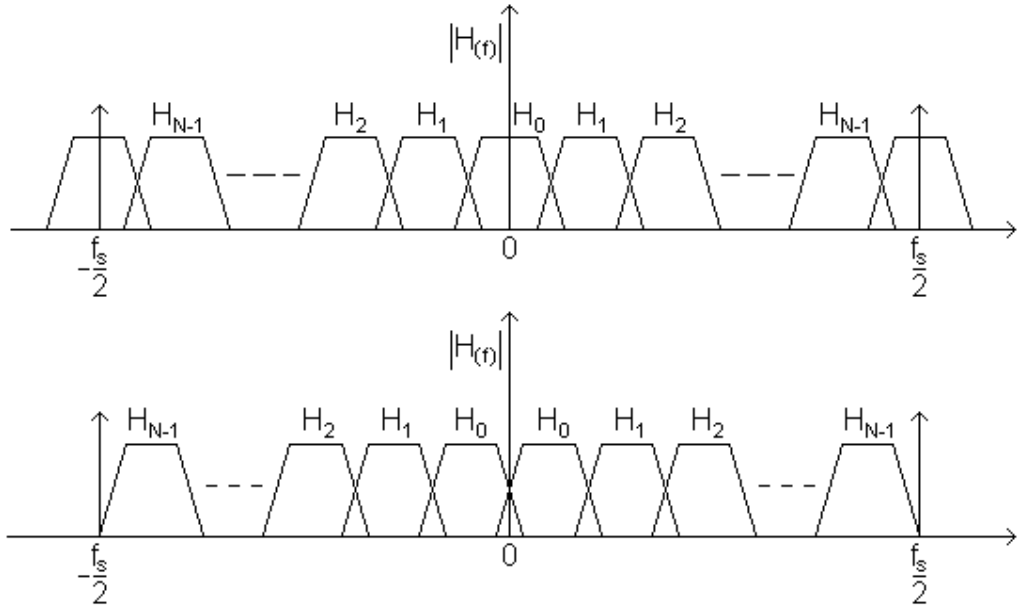
Teljesen ismeretlen környezet esetén jó közelítés lehet, ha fehér zajt feltételezünk. Egy kicsit ennél is jobb, ha a magasabb frekvenciákon nagyobb

küszöbteljesítményeket állítunk be – a lényeg úgyis az, hogy a 0-8 kHZ-es tartományban lévő jel minél kevesebb károsodással jusson át. A zajos jel szintjét viszont valamilyen szinten mindenképp ismerni kell, hogy a küszöbszinteket a zaj ismeretének hiányában legalább ehhez tudjuk igazítani.

Egy másik lényeges korlát az algoritmus jóságára az on-line megvalósítás miatt fellépő real–time követelmény. Két mintavétel között le kell futnia a teljes algoritmusnak, ezért nem használhatunk tetszőlegesen nagyszámú és együtthatójú szűrőket. Ezáltal a szűrők átviteli tartománya nem lehet tetszőlegesen keskeny sem, vagyis az olyannyira fontos 0-8 kHz-es sávot sem darabolhatjuk fel tetszőlegesen kis részekre, tehát a sávonként beállított teljesítményküszöbök nagyobb frekvenciaintervallumban lesznek érvényesek.

3. A szűrőbankok

A szűrőbank sávátereszt szűrők összessége. A szűrőbankot felépítő szűrők átviteli tartományai általában ugyanolyan szélesek, és kis mértékben átlapolják egymást. A szűrőbankok egyik legfontosabb tulajdonsága, hogy a sávátereszt szűrők eredő átvitele konstans. Az alábbi ábrán látható a szűrőbankot felépítő sávátereszt szűrők átviteli függvényének sémája [4]:



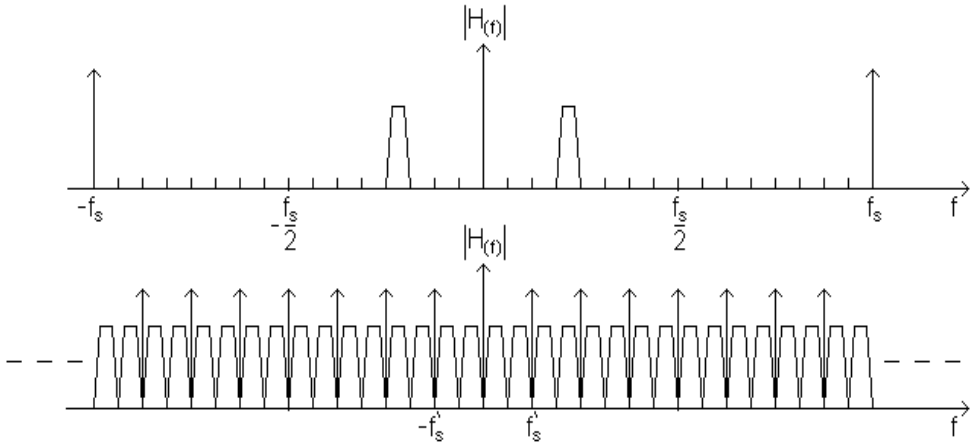
3.1. ábra A szűrőbankokat felépítő szűrők szokásos átvitele

Az alapsávi szűrő sávjának középpontja lehet a frekvenciatengelyen, ilyenkor ez egy aluláteresztő szűrő (az általam megvalósított szűrőbank ezt a sémát valósítja meg), vagy lehet maga is egy sávszűrő, az ábrának megfelelően. Általában audio alkalmazásokban használják, és célja, hogy az ember által hallható tartományt (20Hz-20kHz) sávokra bontsa. Tanulmányok igazolják, hogy az emberi fül nem egyforma erősen hallja a különböző frekvenciájú hangokat, még akkor sem, ha ezek teljesítménye megegyezik [3]. A 20 Hz-8 kHz sávban a fül sokkal jobban szét tudja választani az egymáshoz frekvenciában közel lévő komponenseket, míg az afölöttiben kevésbé. Ez azt jelenti, hogy audio jelek kódolása és átvitele esetén ezekben a sávokban finom felbontású kvantálás szükséges, míg vannak olyan sávok, melyekben durvább felbontású kvantálást alkalmazva a visszaállított jelen nem figyelhető meg nagy mértékű minőségromlás.

Az audio alkalmazásokban általában két szűrőbankot használnak. Az első, amelyen a jel átmegy, az ún. analízis szűrőbank. Ez elvégzi a bemenő jel sávokra bontását, majd minden sáv kimeneti jele különböző bitszámú kvantáláson megy át – a bitszám a beszédsávhoz tartozó szűrők esetén értelemszerűen nagyobb, míg az afeletti sávoknál kisebb.

A kvantálás előtt decimálást, utána interpolálást szokás végezni. Ennek oka, hogy nem kell a szűrőkimenetek összes mintáit feldolgozni, ugyanis azok felesleges információt tartalmaznak. N sávos szűrőbank esetén a decimálás maximális mértéke, amely nem jár információvesztéssel: N – ha a szűrők átvitelét sávkorlátoltnak tekintjük, sávszélességüket pedig $\frac{f_s}{2 \cdot N}$-nek.

A decimálásnak a jel spektrumára gyakorolt hatása figyelhető meg az alábbi ábrán:



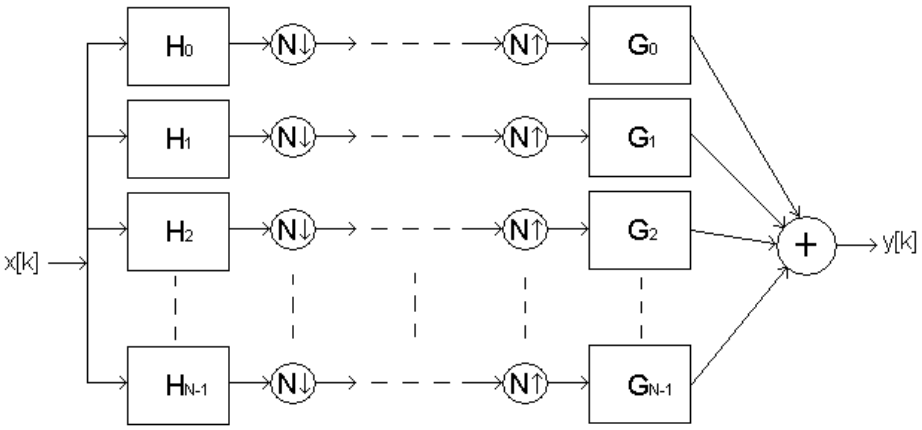
3.2. ábra. A decimálás hatása a sávszűrő átvitelére

Az ábrán egy nyolcsávos szűrőbank harmadik sávjának átvitele figyelhető meg, decimálás előtt, és decimálás után. Az alapsávi szűrő sávszűrő.

A decimálás ugyan fáziseltolást okoz, de azt az interpolálás helyreállítja. A jelfolyamban decimálás és az interpolálás közé szokták betenni a jelfeldolgozó algoritmusokat, tömörítés esetén a kvantálást. Mivel csak minden N-edik mintára van szükségünk, az alkalmazott jelfeldolgozó algoritmus hívásának száma N-ed részére csökken.

Az interpolálás miatt kell egy másik szűrőbankot is a jelfolyam végére betenni, amelynek neve szintézis szűrőbank. A szintézis szűrőbank szűrőinek kimeneteit összegezve megkapjuk végül a tömörített, de minőségileg némileg rosszabb kimenetet.

Az alábbi ábra a szűrőbankokat használó jelfeldolgozási algoritmusok szokásos blokkvázlatát mutatja:



3.3. ábra. A szűrőbankokat tartalmazó rendszerek szokásos felépítése

3.1. Matematikai modell

Legyen az alapsávi szűrő átvitele H_0, impulzusválasza h_0. Az impulzusválaszt diszkrét Fourier-transzformáltját (DFT) véve kapjuk az átviteli függvényt:

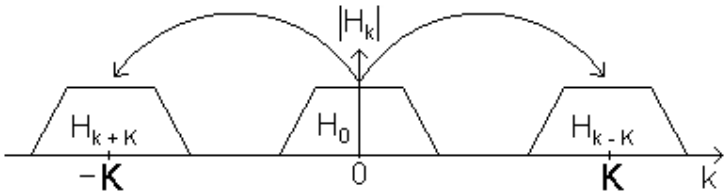
$$H_0[k] = \sum_{n=0}^{N-1} h_0[n] e^{-\frac{j2\pi}{N}kn}, \quad k = 0,...,N-1 \tag{3.1}$$

ahol N a minták száma, n a diszkrét időváltozó, k pedig a diszkrét frekvenciaváltozó.

A sávszűrők impulzusválaszát úgy kaphatjuk, hogy a frekvenciatartományban eltoljuk az alapsávi szűrő átvitelét, majd a kapott spektrumon inverz DFT-t hajtunk végre. Ahhoz, hogy a kapott impulzusválasz valós legyen, az eltolt spektrumnak páros függvénynek kell lennie, vagyis az eltolást negatív irányban is el kell végezni. Legyen az eltolás mértéke K, akkor az eltolt spektrum:

$$H'[k] = H_0[k-K] + H_0[k+K] \tag{3.2}$$

Az eltolt spektrum látható az alábbi ábrán:



3.4. ábra. Az eltolt spektrum

A frekvenciatartománybeli eltolás, majd az IDFT alkalmazása ekvivalens azzal, ha az impulzusválaszra alkalmazzuk a DFT-re vonatkozó eltolási tételt:

$$X[k-K] = DFT\left\{x[n]\cdot e^{\frac{j2\pi}{N}Kn}\right\}, \quad X[k+K] = DFT\left\{x[n]e^{-\frac{j2\pi}{N}Kn}\right\} \tag{3.3}$$

A DFT és az IDFT lineáris operátorok, vagyis:

$$IDFT\{A_k + B_k\} = IDFT\{A_k\} + IDFT\{B_k\} \tag{3.4}$$

Ezért igaz az alábbi:

$$h'[n] = IDFT\{H_0[k-K] + H_0[k+K]\} = IDFT\{H_0[k-K]\} + IDFT\{H_0[k+K]\} = \tag{3.5}$$
$$= h_0[n]e^{\frac{j2\pi}{N}Kn} + h_0[n]e^{-\frac{j2\pi}{N}Kn} = h_0[n]\left(e^{\frac{j2\pi}{N}Kn} + e^{-\frac{j2\pi}{N}Kn}\right)$$

Vegyük most az Euler formulát:

$$e^{jx} = \cos(x) + j\sin(x), \quad e^{-jx} = \cos(x) - j\sin(x) \longrightarrow 2\cos(x) = e^{jx} + e^{-jx} \tag{3.6}$$

Ebből következik, hogy az eltolt szűrő impulzusválasza:

$$h_n'[n] = 2 \cdot h_0[n] \cdot \cos\left(\frac{2\pi \cdot K \cdot n}{N}\right), \quad n = 0,...,N-1 \qquad (3.7)$$

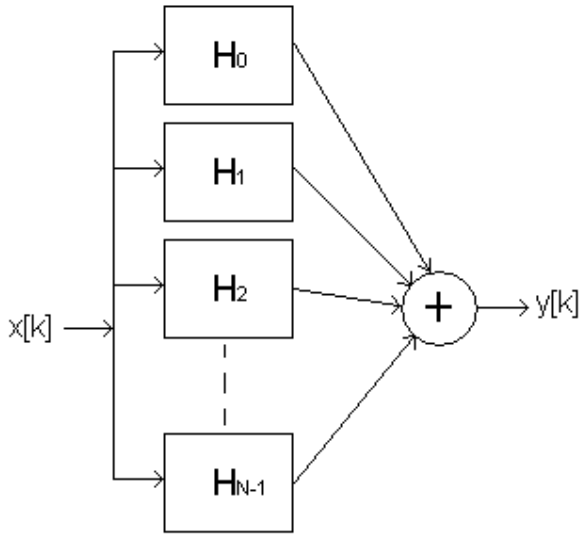
Az eltolás mértékét, és a szűrők átviteli tartományának szélességét úgy kell meghatározni, hogy az N sávos szűrőbank sávjai kitöltsék a [0,fs) frekvenciatartományt (normált frekvencia esetén a [0,π) tartományt). A K értéke nem muszáj, hogy egész szám legyen, de az általam megvalósított szűrőbanknál ez az érték 2-re adódik (32 sáv, 128 együttható). Ha a szűrők átlapolják egymást, akkor az egész tartományon érvényesülnie kell a konstans átvitelnek.

3.2. A megvalósítandó szűrőbank

A hardver lehetővé teszi, és a feladat megköveteli, hogy a sávok száma legalább 32 legyen. Ez utóbbinak az oka, hogy minél több sávra bontjuk fel a jelet, a zajcsökkentő algoritmusok annál keskenyebb sávban lesznek érvényesek, ami finom hangolhatóságot tesz lehetővé. 32 sáv esetén minden szűrő sávszélessége 750 Hz lesz, ami már elegendőnek tekinthető. A Bark féle skálán [3], ami a hallásra vonatkozóan 24 kritikus sávot határoz meg, a 0-8 kHz-es tartományba 21 kritikus sáv esik. 32 sávos szűrőbank esetén ebbe a tartományba 11 sáv tartozik, ami már megfelelően finom beállításokat tesz lehetővé.

Egyébként ez a szám viszonylag általánosnak mondható a szűrőbank alkalmazások esetében, az MPEG szabványban alkalmazott szűrőbankban is ennyi a sávok száma. Online szűrésről lévén szó, a sávok száma sajnos korlátokba ütközik. 64 sávos szűrőbank esetén például nem kétszer annyi a szűréshez szükséges műveletek száma, mint a 32 sávosnál, hanem négyszer annyi: ha kellően sávszelektív szűrőbankot akarunk megvalósítani – meredek levágást, és keskeny átviteli sávot-, akkor egy 64 sávos rendszerbe való sávszűrő együtthatószáma nagyjából kétszer akkora, és kétszer annyi a szűrők száma is – innen a négyes szorzó. Maradtam tehát a 32 sávos modellnél, a prototípusszűrő pedig egy aluláteresztő szűrő.

A fejezet elején lévő, általánosan alkalmazott szűrőbank modelltől eltértem. Az általam megvalósított modellben nincs szintézisszűrő, mivel nem alkalmazok sávonként eltérő kvantálást, interpolálást és decimálást. A megvalósítandó szűrőbank:



3.5. ábra. A megvalósítandó szűrőbank blokkvázlata

3.3. A prototípusszűrő és a megvalósított szűrőbank

A prototípusszűrő egy aluláteresztő szűrő, melynek határfrekvenciáját (a -3dB-es feszültségerősítés frekvenciája) N sávos szűrőbank esetén:

$$f_c = \frac{f_s}{2} \cdot \frac{1}{N} \cdot \frac{1}{2} \tag{3.8}$$

Ennél nagyobb is lehet a határfrekvencia, az eredő átvitel akkor is konstans marad, de érdemes a lehető legvékonyabbra tervezni, a sávszelektivitás miatt. A prototípusszűrő szimmetrikus a 0 Hz-es pontra, átvitelének csak fele esik a pozitív frekvencia tartományra, innen az utolsó, ½-es szorzó. Az f_s = 48 kHz, illetve N = 32 adatok ismeretében az f_c = 375 Hz értéket kapjuk. Az átviteli tartomány ingadozása persze egyenletes kell, hogy legyen, és a lehető legkisebbre kell csökkenteni, hisz a szűrőbank eredő átvitelének konstans 1-nek kell lennie az egész [0, f_s/2) tartományon.

A sávok szelektivitásának növelése érdekében továbbá biztosítani kell, hogy az egyes szűrők közötti átlapolódás minél kisebb mértékű legyen. Ezt meredek levágású prototípusszűrő alkalmazásával lehet elérni. Azonban minél meredekebb amplitudó karaketerisztikát szeretnénk megvalósítani, annál több szűrőegyüttható̋ra van szükség, ezért az együtthatók számát addig kell növelni, míg el nem érjük a számítási kapacitás határát – persze úgy, hogy jut elég erőforrás az algoritmus többi részének lefuttatására is. 128 együttható esetén a szűrés 42%-ban használja a processzort, és ezzel az együtthatószámmal a fent említett megkötéseket elég jól teljesítő prototípusszűrő tervezhető, tehát ennél az értéknél maradtam.

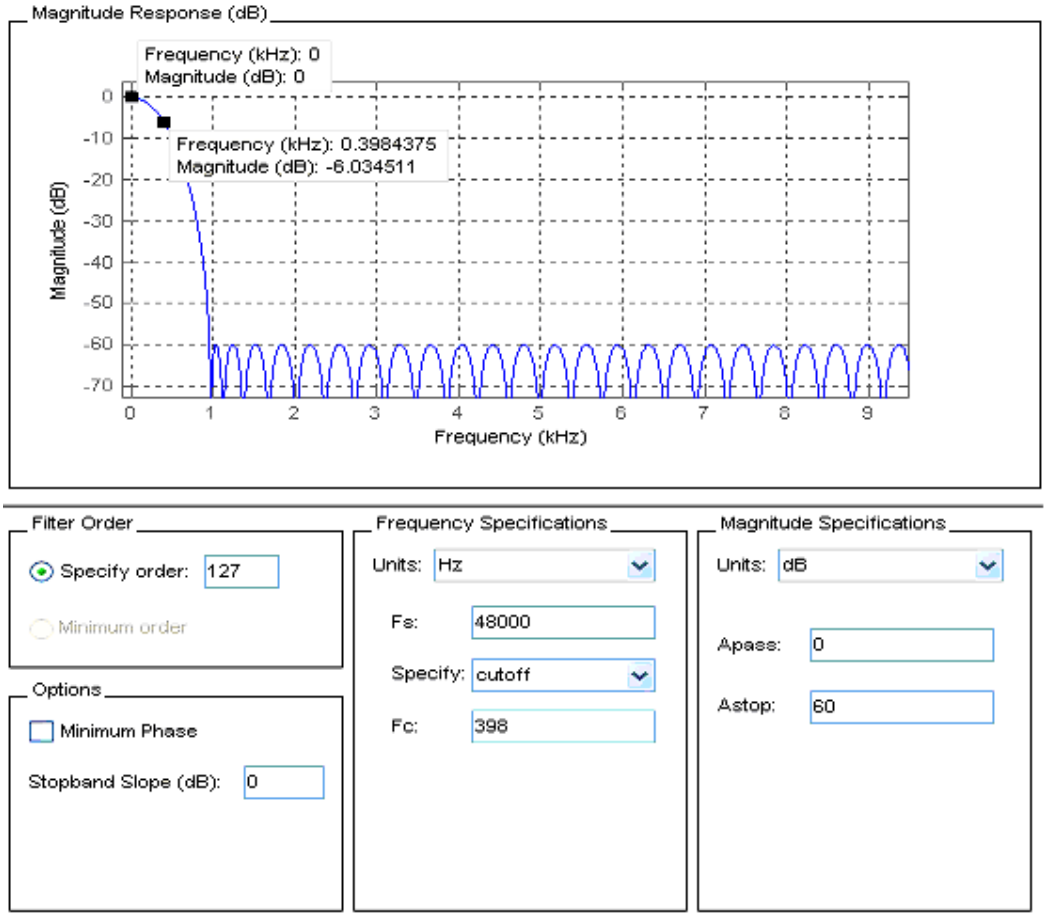
Magát a szűrőt a MATLAB fdatool nevű eszközével terveztem, mely sokféle szűrőtervezési algoritmus megvalósítására képes. A különböző algoritmusok közül a ”Constrained Equiripple” nevűt választottam, mivel a fentebb megfogalmazott elvárásokhoz ez idomul a legjobban. Itt a legfontosabb beállítható paraméterek: zárótartománybeli, és áteresztő tartománybeli erősítés, rendszám (az együtthatók száma ennél eggyel nagyobb lesz), mintavételi frekvencia. Ezen kívül három paraméter van, melyek közül egyszerre csak az egyikre alkalmazhatunk megkötést: törésponti frekvencia, áteresztő tartomány vége, záró tartomány kezdete.

A prototípusszűrő átviteli tartománya meglehetősen keskeny a [0, f_s] sávhoz képest. A szűrőtervezési algoritmus a fentebb említett f_c =375 Hz mellett 128 együttható esetén nem tudta a 0 Hz-es átvitelt 0 dB-ig kiemelni, ezért egy kicsit szélesebb áteresztő tartományt engedtem meg. Az új törésponti frekvencia 398 Hz lett.

A zárótartománybeli elnyomást 60 dB-esre terveztem. Nagyobb elnyomás mellett a szűrőtervező algoritmus nem tudta kiemelni a prototípusszűrő átvitelét 0 dB-ig, a már említett törésponti frekvencia mellett. A 60 dB-es zárótartománybeli elnyomás azonban megfelelőnek tekinthető. A sávszűrők egymásra gyakorolt hatásának csökkentése érdekében a zárótartománybeli ingadozás legyen egyenletes.

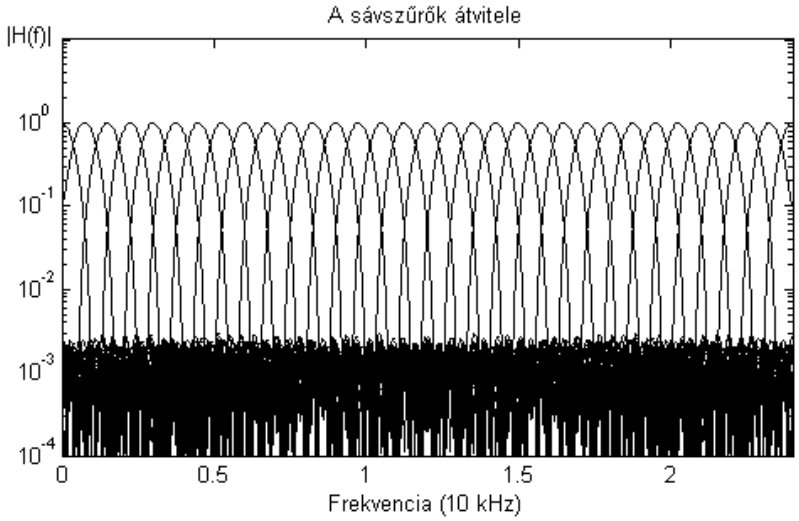
A mintavételi frekvencia értékére 48 kHz-et állítottam be, mert ennyi a kártyán lévő audio interfész mintavételi frekvenciája.

Az alábbi ábrán a végleges szűrő átvitele, valamint a beállított paraméterek láthatók (az algoritmus törésponti frekvenciaként a szokásos -3 dB helyett a -6 dB-hez tartozó frekvenciaértéket tekinti):



3.6. ábra. A prototípusszűrő átvitele

A prototípusszűrőből a 3.1. fejezet alapján létrehozott szűrőbank sávszűrőinek átvitele látható a következő ábrán:



3.7. ábra. A sávszűrők átvitele

A szűrőbank eredő átvitele az egész tartományban konstans.

3.4. A szűréshez szükséges műveletek számának csökkentése

Ebben a fejezetben egy olyan módszert mutatok be, mely segítségével a szűréshez szükséges műveletek száma csökkenthető, és amelyet a végső programból kihagytam, mert főleg nagyobb együtthatószámú szűrőbankok esetén hatásos.

Egy 32 sávos szűrőbank esetén, ha a szűrőegyüttható k száma szűrőnként 256, a szűréshez szükséges MAC műveletek száma: 256 * 32 = 8192. A MAC művelet két operandus összeszorzását és az eddigi főösszeghez való hozzáadását jelenti, erről később még lesz szó. Ez elsőre meglehetősen soknak tűnik, különösen, hogy az impulzusválaszok a prototípusszűrő koszinusz vektorral való szorzásával, eltolásával keletkeztek, amely periodikus függvény. Van azonban mód a műveletek számának csökkentésére, mely a koszinusz függvény pontosan ezen tulajdonságát használja ki.

Az i-edik szűrő impulzusválaszát így kapjuk (N az együtthatók száma, K_1 az egyes sorszámú szűrő létrehozásához szükséges eltolás, és n az együttható sorszáma):

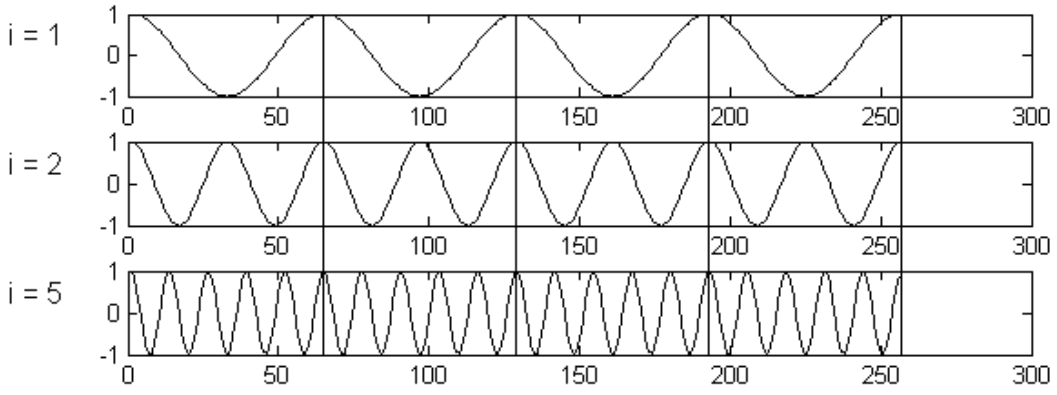
$$h_i[n] = h_0[n] \cdot \cos\left(\frac{i \cdot K_1 \cdot n \cdot 2 \cdot \pi}{N}\right), \quad n = 0 - 255 \tag{3.9}$$

32 sáv és 256 együttható esetén $K_1 = 4$ adódik:

$$\begin{aligned} h_i[n] &= h_0[n] \cdot \cos\left(\frac{i \cdot 4 \cdot n \cdot 2 \cdot \pi}{256}\right) = h_0[n] \cdot \cos\left(\frac{i \cdot n \cdot 2 \cdot \pi}{64}\right) = \\ &= h_0[n] \cdot C_i[n], \quad n = 0 - 255 \end{aligned} \tag{3.10}$$

Jelölje a továbbiakban $C_i[n]$ az i. szűrőhöz tartozó koszinusz vektort.

A koszinuszos tag i=1 – re a koszinusz négy periódusa. Minden további, magasabb fokszámú szűrő koszinuszos tagja periódusainak száma ennek egész számú többszöröse. Mindegyik szűrő koszinuszos tagjának van tehát négy szuperperiódusa:



3.8. ábra. A szuperperiódusok szemléltetése

Ez azt jelenti, hogy:

$$C_i[k] = C_i[k + j \cdot 64], \quad j = 1-3, \quad k = 0-63 \tag{3.11}$$

Az i. szűrő kimeneti értékét az impulzusválasza és a bemenet konvolúciójaként kapjuk:

$$y_i[k] = \sum_{n=0}^{N-1} h_i[n] \cdot x[k-n] \tag{3.12}$$

ahol k a diszkrét időváltozó. Ha a bemenetet egy bufferben tároljuk, mely shift-regiszter szerűen működik, vagyis a legkorábbi minta a tömb első, a legkésőbbi minta pedig a tömb utolsó eleme, és a buffer elemszáma megegyezik a szűrőegyütthatók számával, akkor az éppen aktuális kimenetet így kapjuk:

$$y_i = \sum_{n=0}^{N-1} h_i[n] \cdot x[n] = \sum_{n=0}^{N-1} C_i[n] \cdot h_0[n] \cdot x[n] \tag{3.13}$$

A $\sum_{n=0}^{N-1} h_0[n] \cdot x[n]$ összegzést minden ütemben el kell végeznünk. Az ebből nyert információt viszont a többi szűrőhöz szükséges számítások nagy részének elkerülése céljából felhasználhatjuk. Mentsük el az összegzés közben a $h_0[n] \cdot x[n]$ értékeket egy n hosszúságú tömbbe, és nevezzük az így kapott elemeket $hx_0[n]$ -nek. Ekkor igaz az alábbi:

$$y_i = \sum_{n=0}^{N-1} C_i[n] \cdot hx_0[n] \tag{3.14}$$

Felhasználva a (3.11) azonosságot adódik, hogy

$$y_i = \sum_{n=0}^{63} C_i[n] \cdot \sum_{j=0}^{3} hx_0[j \cdot 64 + n] \tag{3.15}$$

A $\sum_{j=0}^{3} hx_0[j \cdot 64 + n]$ tagot egyszer kell kiszámolni, majd el kell menteni egy 64 elemű tömbbe. A C_i vektornak csak az első 64 elemére van szükség, és sávonként a szükséges MAC műveletek száma 256 helyett 64 lesz (leszámítva a prototípusszűrőt, ott mind a 256 MAC műveletet el kell végezni). Követve a fenti lépéseket, a szűrés költsége a következőképpen alakul: 256 MAC művelet + 256 mentés + 64*4 MAC művelet + 64 mentés + 31*64 MAC művelet = 2816 művelet, 8192 MAC művelet helyett. 512 együtthatós szűrésre még jobb arány adódik: 512 MAC művelet + 512 mentés + 64*8 MAC művelet + 64 mentés + 31*64 MAC művelet = 3584 művelet, 16384 MAC művelet helyett. Minél nagyobb a szűrőegyütthatók száma, annál nagyobb mértékű javulás érhető el. Az általam megvalósított, 128 együtthatós prototípussűrőjű szűrőbankra az algoritmus korántsem működik ilyen jól, az eredmény: 128 MAC művelet + 128 mentés + 64*2 MAC művelet + 64 mentés + 31*64 MAC művelet = 2432 művelet 4096 MAC művelet helyett. Hozzá kell azonban ehhez venni, hogy a DSP processzor egy órajel ciklus alatt két MAC műveletet képes elvégezni, ráadásul a 2*64 MAC művelet az nem 128 órajel ciklus, hanem a kezdeti betöltések miatt több (az n darab MAC művelethez szükséges órajel ciklusok akkor tekinthetőek n darabnak, ha az egyszerre elvégzendő MAC műveletek száma nagy). A programszervezés pedig eléggé bonyolulttá válik, kis együtthatószám esetében előfordulhat, hogy rosszabbul működik a szűrőalgoritmus, mint az eredeti. Ezért, bár kipróbáltam az algoritmus működését, mégsem került bele a végső programba.

4. A megvalósítandó karakterisztikák

A továbbiakban karakterisztika alatt a szűrőkön átjutott jelek erősítését értem, az átlagteljesítmény függvényében. Ez lényegében átviteli karakterisztika jellegű görbe, csak a szokásos használattal szemben az átvitel nem a frekvenciától, hanem az átlagteljesítménytől függ.

A zajcsökkentő algoritmus bemenő paraméterei a szűrők aktuális kimeneteiből, és sávonként a W_L hosszúságú teljesítményablakok összegeiből állnak. Általánosan elmondható, hogy a szűrők kimeneti értékei egy bizonyos átlagteljesítmény-határt elérve változatlanok maradnak. A különböző karakterisztikákkal megvalósított algoritmusok közti eltérést főleg az adja, hogy mi történik az adott szűrőkimenettel, ha az adott sávon az addigi átlagteljesítmény a határ alatt van. Ez alapján ötféle karakterisztikát valósítottam meg:

- lépcsős
- hiszterézises
- lineáris elnyomású
- lineáris elnyomású, ofszettel
- négyzetes elnyomású.

Ezek a karakterisztikák természetesen nem lineárisak, hiszen a kimenet nem a bemenet lineáris függvénye, hanem a legutolsó W_L darab érték négyzetösszegének (átlagteljesítményének) mértékétől függően változik.

4.1. A lépcsős karakterisztika

A lépcsős karakterisztika megvalósítása esetén minden sávra definiálni kell egy K_i küszöb értéket, mely az adott sáv teljesítményére vonatkozik. Feltételezzük, hogy a zaj viszonylag rövid időre kiszámolt átlagteljesítménye kisebb, mint a jel átlagteljesítménye. A karakterisztika lépcsős voltát az adja, hogy ha az aktuális mintavételig számolt átlagteljesítmény nem éri el az i. szűrőre definiált K_i küszöbértéket, akkor a szűrő kimenete zérus lesz, ellenkező esetben változatlan marad:

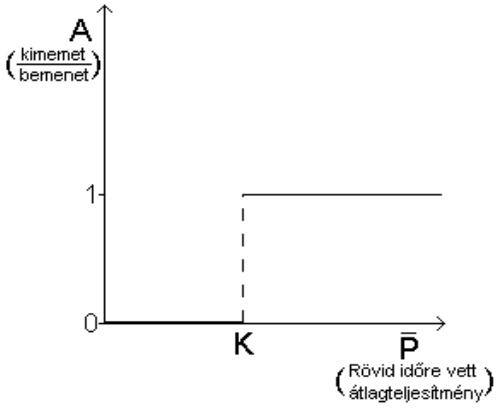
$$X_i = \begin{cases} 0, \ ha \ \sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 < K_i \\ X_i, \ ha \ \sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 \geq K_i \end{cases} \tag{4.1}$$

ahol X_i az i. szűrő kimenete az aktuális mintavételi ütemben, K_i az i. szűrőhöz tartozó küszöbérték, az összegzés pedig a legutolsó W_L ütemben a szűrőn átengedett jelteljesítményekre vonatkozik.

A karakterisztika lényege, hogy a K_i paraméternél kisebb teljesítményű jelet úgyis elnyomja az ott lévő zaj, ezért felesleges azt átengedni. A probléma ezzel az, hogy ha a zaj teljesítménye nagyobb, mint ez a küszöbérték, akkor a zajos jel teljes egészében átjut, illetve ha a K_i szintjét túl nagyra állítjuk, akkor a hasznos jelből is sokat veszíthetünk. Továbbá, mivel a zaj nem minden ütemben jut át, az eddigi egyenletes sistergés (közel fehér zajt feltételezve) helyett egy kisebb teljesítményű, de más jellegű (éles, pittyegő) zajt kapunk, mely esetleg zavaróbb lehet, mint az eredeti ”konstans” zúgás. Meg kell jegyeznünk ezen kívül, hogy teljesen zajmentes jelre futtatva az algoritmust, az eredetinél rosszabb minőségű jelet is kaphatunk, ha K_i értéke túl nagy. Megfelelőre választva ezt a paramétert, azért biztató eredmény érhető el.

A karakterisztika működési elve hasonló, mint a 2.2. fejezetben említett hard-elbow algoritmusé. Az általam megvalósított karakterisztikák között nem szerepel olyan, ami a szintén abban a fejezetben megemlített soft-elbow algoritmushoz hasonlóan működne.

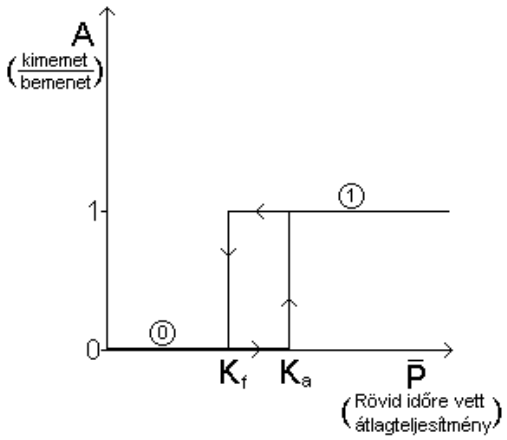
A lépcsős karakterisztika grafikusan ábrázolva:



4.1. ábra. A lépcsős karakterisztika

4.2. A hiszterézises karakterisztika

A hiszterézises karakterisztika a fentihez hasonló, itt is zérus lesz a kimenő jel, amennyiben a kívánt teljesítményátlagot az utolsó W_L minta négyzetösszege nem éri el. A különbség, a névből adódóan, hogy kétféle úton mozoghatunk a karakterisztika görbéjén. Minden szűrőre két paramétert definiálunk, egy alsó és egy felső határt. Az, hogy melyiket hasonlítjuk az aktuális átlagteljesítményhez, egy állapotváltozótól függ, mely azt mutatja meg, hogy az előző ütemben a karakterisztika mely részén mozgott az átlagteljesítmény. Az alábbi ábrán a hiszterézises karakterisztika látható:



4.2. ábra. A hiszterézises karakterisztika

Ha az adott szűrő a 0 állapotban van, akkor az aktuális átlagteljesítményt a K_a-hoz hasonlítjuk, ha az 1 állpotban van, akkor pedig a K_f-hez. Amennyiben $K_a = K_f$, akkor természetesen ez megegyezik a lépcsős karakterisztikával.

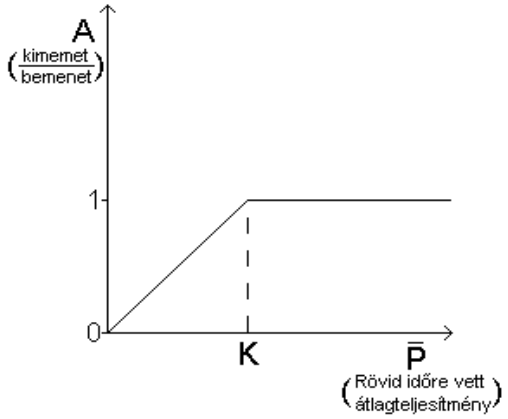
A karakterisztika alkalmazását a lépcsős metódusnál említett negatív hatások (főleg a pittyegés) csökkentése indokolja. Az alapgondolat az, hogy a hiszterézis miatt kevesebb ugrás lesz az átengedett zaj teljesítményében. Az átengedett zaj teljesítménye $K_a = K$ esetén (ahol K a lépcsős karakterisztika paramétere) nagyobb lesz ugyan, ugyanakkor a jellege nem lesz annyival másabb, az eredeti zaj teljesítményéhez képest pedig csökkenés érhető el.

4.3. A lineáris elnyomású karakterisztika

Ez a karakterisztika is a már fent ismertetett lépcsős karakterisztika továbbfejlesztése, azonban némileg kifinomultabb annál. A ”lineáris” szó kicsit megtévesztő lehet, mivel ez a karakterisztika is nemlineáris. A kifejezés arra utal, hogy ha a szűrőkimenet átlagteljesítménye nem éri el a küszöbparamétert, akkor a jel az aktuális, és a küszöb átlagteljesítmény arányában lesz elnyomva:

$$X_i = \begin{cases} X_i \cdot \dfrac{\sum_{k=1}^{W_L} (X_{i(j-k+1)})^2}{K_i}, & ha \ \sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 < K_i \\ X_i, ha \ \sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 \geq K_i \end{cases} \tag{4.2}$$

Ennek a karakterisztikának az a hátránya, hogy lényegében valamilyen formában minden komponens, tehát a zaj is átjut, csak legfeljebb csillapítva. Előnye, hogy az eddig vázolt két metódussal szemben nem lesznek az átengedett zaj teljesítményében meredek ugrások, ezért meg fog szűnni a zavaró pittyegés. A lineáris elnyomású karakterisztika ábrája:



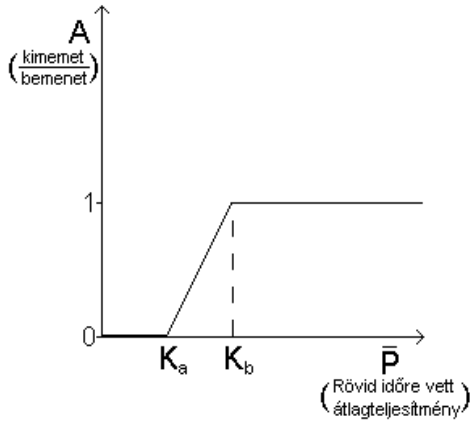
4.3. ábra. A lineáris elnyomású karakterisztika

4.4. A lineáris elnyomású karakterisztika, ofszettel

Ezt a lineáris, és a lépcsős karakterisztikákat vegyítve kapjuk meg. Két küszöbparamétert definiálunk, egy kisebbet (K_a), és egy nagyobbat (K_b). Ha az adott átlagteljesítmény K_a alatt van, akkor a kimenet zérus lesz, ha K_a és K_b között van, akkor lineárisan nyomjuk el a lineáris karakterisztikánál megismert módon. Ha nagyobb, mint K_b, akkor teljes egészében átengedjük:

$$X_i = \begin{cases} 0, ha \ \sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 < K_a \\ X_i \cdot \dfrac{\sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 - K_a}{K_b - K_a}, ha \ K_a \le \sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 < K_b \\ X_i, ha \ \sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 \ge K_b \end{cases} \tag{4.3}$$

A lineárishoz képesti előnye, hogy nem enged át minden komponenst, a nagyon kicsiket kiszűri. Ugyanakkor nem lesznek az átengedett teljesítményben akkora ugrások sem, hisz az átmenet a két erősítési szint között a karakterisztikán nem ugrásszerű, hanem folytonos. A lineáris, ofszetes karakterisztika:



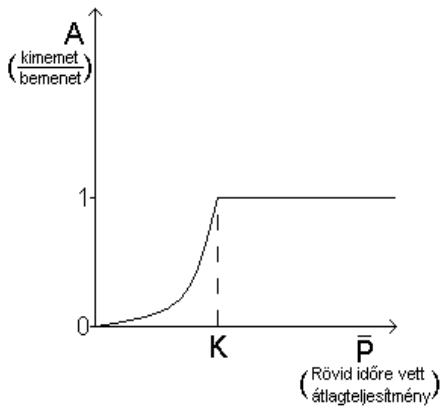
4.4. ábra. A lineáris elnyomású karakterisztika ofszettel

4.5. A négyzetes elnyomású karakterisztika

A négyzetes elnyomású karakterisztika úgy tekinthető, mintha a lineáris, és az ofszetes lineáris karakterisztika között félúton lenne. A karakterisztika-görbe formája a küszöbteljesítmény alatti átlagteljesítmény értékek esetében olyan alakú, mint az $f(x) = x^2$ függvény a [0,1] intervallumon. A karakterisztikában nincs zérus szakasz, ugyanakkor a kis teljesítményű zajokat jóval nagyobb mértékben nyomja el, mint a lineáris karakterisztika. A négyzetes karakterisztika képlete:

$$X_i = \begin{cases} X_i \cdot \left(\dfrac{\sum_{k=1}^{W_L} (X_{i(j-k+1)})^2}{K_i} \right)^2, & ha \ \sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 < K_i \\ X_i, & ha \ \sum_{k=1}^{W_L} (X_{i(j-k+1)})^2 \geq K_i \end{cases} \tag{4.4}$$

A négyzetes elnyomású karakterisztika:



4.5. ábra. A négyzetes elnyomású karakterisztika

5. A hardver és szoftver eszközök bemutatása

Ebben a fejezetben azoknak a hardver és szoftver eszközöknek a sajátosságaival foglalkozom, melyekkel a fejlesztést végeztem.

5.1. A processzor

Az algoritmust az Analog Devices cég ADSP-BF537-es processzorára implementáltam. A cég számos egyéb DSP terméke (ADSP-21xx, SHARC, TigerSHARC) közül a Blackfin család processzorainak főbb jellemzője: 16/32 bites Harvard architektúra, 600 MHz-es órajel, és viszonylag nagy számítási kapacitás mellett is alacsony fogyasztás, mely ideálissá teszi a beágyazott rendszerekben való alkalmazásra.

A Harvard architektúrát főleg DSP-kben és mikrokontrollerekben szokták alkalmazni. Ami megkülönbözteti a sokkal elterjedtebb Neumann architektúrától, hogy az utasítások és az adatok külön memóriaterületen helyezkednek el. Ennek két hatalmas előnye van. Először is, az adat és a programutasítás által megtett jelút nem esik egybe, ezért akár ugyanabban az órajel ciklusban is le lehet fetch-elni az adatmemóriából egy adatszót és egy a programmemóriából egy programszót. Másodszor, az adatmemória és a programmemória nem kell, hogy feltétlenül ugyanolyan szerkezettel rendelkezzen, általában a programmemória szóhosszúsága nagyobb. Ez utóbbi tulajdonságot főleg a RISC processzorok használják, ott az utasítások többsége úgy van megtervezve, hogy elférjen egy programszóban, ezáltal egy utasítás fetch-elése egy órajel ciklusig tartson. (A modern, nem DSP processzorok egyébként vegyes architektúrát használnak: a cache (gyorsítótár) elérése Harvard architektúrára épül, míg a külső memória elérése továbbra is Neumann alapú.)

A processzor belső memóriája három blokkból áll, és mindhármat maximális sebességgel képes elérni, ezek:

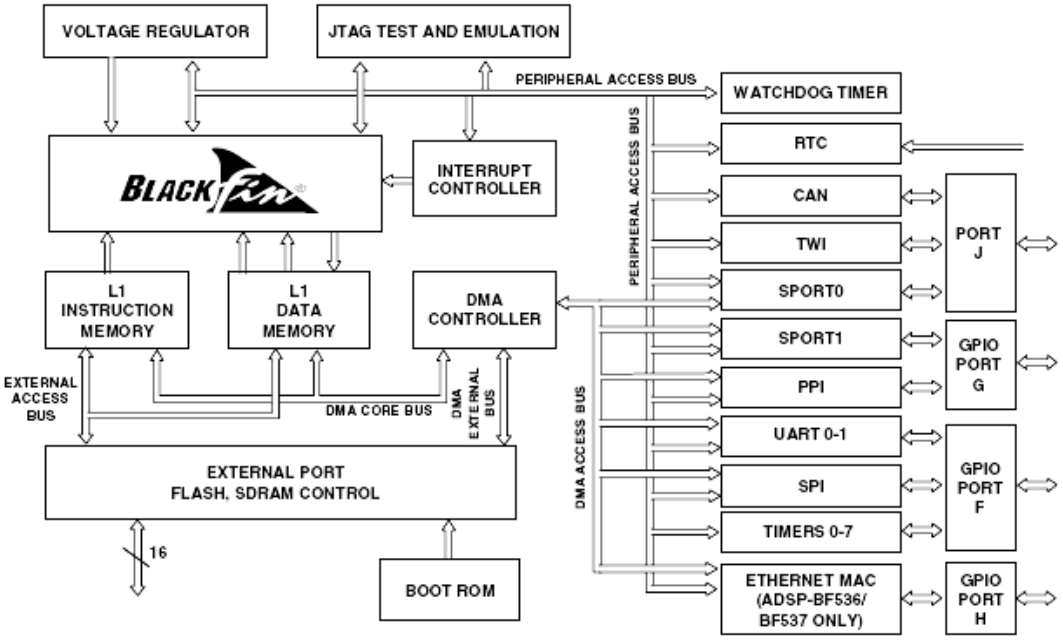
- elsőszintű programmemória, mely egy 64 kB-os SRAM, ebből 16 kB konfigurálható cache-nek, a processzor maximális sebességgel éri el
- elsőszintű adatmemória, ez két 32 kB-os SRAM blokkból áll, mindkét blokk beállítható cach-nek,

- 4 kB-os, csak SRAM-ként elérhető memória.

A BF537 nem rendelkezik másodszintű SRAM-mal, mint a Blackfin család többi processzora közül néhány. A 16 bites EBIU-n (external bus interface unit – külső busz interfész egység) keresztül a processzor külső memóriát is el tud érni: SDRAM-ot, flash memóriát és SRAM-ot is. Az így megcímezhető külső memória mérete maximum 516 Mb.

A processzor ezeken kívül rendelkezik a mikrokontrollereknél általánosnak tekinthető perifériákkal is, ezek: CAN interfész, DMA vezérlők, UART, SPI, JTAG interfész.

Az alábbi ábrán látható a processzor blokkvázlata:



5.1. ábra. A BF537 processzor blokkvázlata

5.2. A processzormag

A BF537 magja két nagyobb részből áll: az adatmanipuláló egységből (data arithmetic unit), és a címaritmetikát elvégző egységből (address arithmetic unit).

Az adatmanipuláló egység a számítási műveletek elvégzéséhez tartalmaz két darab 16 bites szorzót, két darab 40 bites akkumulátort, ugyancsak kettő 40 bites ALU-t,

négy darab nyolcbites video ALU-t, és egy 40 bites shiftelő egységet. Nyolc darab 32 bites adatregisztere van, melyek mindegyike viselkedhet két 16 bites adatregiszterként is. A számításokhoz szükséges operandusok vagy közvetlenül a programmemóriából, vagy az adatregiszterekből töltődnek be az aritmetikus egységekbe. A processzormag tartalmaz még egy státuszregisztert is (ASTAT), mely a legutóbbi számításokkal kapcsolatos szokásos flageket tartalmazza (Z, N, stb.).

Az akkumlátorok közvetlenül a szorzók kimeneti adatait kapják meg. Egy akkumulátor és egy szorzó együttesen egy speciális, a DSP processzorra jellemző egységet alkot, melynek neve MAC (multiply and accumulate – szorzás, majd inkrementálás). A MAC művelet két operandus összeszorzásából és annak az akkumulátorban tárolt részösszeghez való hozzáadásából áll. Ezt, és a két következő operandus fetch-elését a processzor egyetlen órajelciklus alatt végrehajtja. Ráadásul a két MAC egység egymással párhuzamosan is képes dolgozni, bár ekkor mindkettő MAC csak két 16 bites számot szorozhat össze egymással, és a négy operandust két 32 bites regiszterben kell tárolni, pakolt struktúrában. Ha például A és B két 32 bites regiszter, akkor legális utasítás az

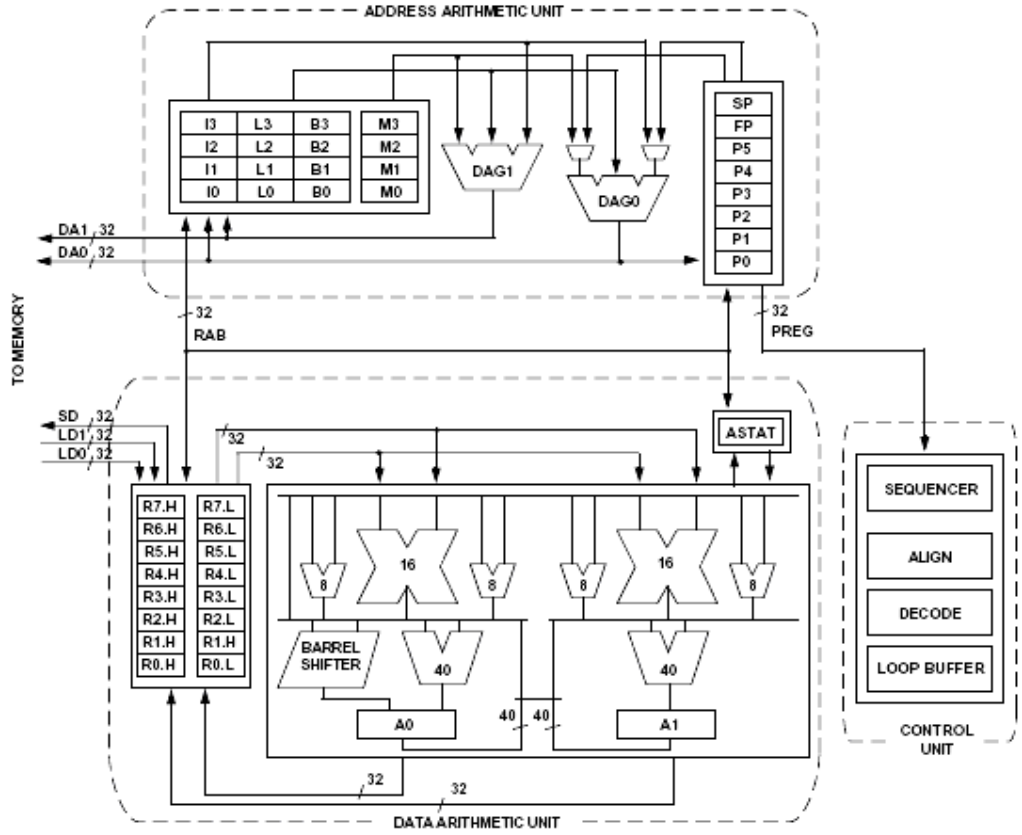
A(alsó 16 bit)*B(felső 16 bit) ; A(felső 16 bit)*B(alsó 16 bit)

műveletek ugyanabban az órajel ciklusban való elvégzése, de nem lehet ugyanabban az órajel ciklusban elvégezni az

A(alsó 16 bit) * B(felső 16 bit) ; A(felső 16 bit) * C(alsó 16 bit)

műveleteket. Ennek azért nagy a jelentősége, mert a FIR szűrés MAC műveletekből áll, ezért ezt a lehetőséget a szűrőbankot megvalósító program intenzíven használni fogja.

A processzormag felépítése látható az alábbi ábrán:



5.2. ábra. A processzormag felépítése

A fenti ábrán megfigyelhető a címaritmetikát elvégző egység blokkvázlata is. Ennek az egységnek két regisztercsoportja van. A P0-P5 (általános célú pointer regiszterek), FP (frame pointer), és SP (stack pointer) alkotják az első csoportot: ezek 32 bites regiszterek, és a címzésen kívül korlátozott mértékben aritmetikai műveletek is végrehajthatók tartalmukon. A másik regisztercsoport neve DAG (data address generator). Ennek a csoportnak négy fajta regisztere van: I (index regiszter), B (bázisregiszter), M (módosító regiszter), és L (hosszt tároló regiszter); mindegyikből négy áll rendelkezésre. Az I és a B regiszterekkel indirekt címzést valósíthatunk meg. Ha az M és az L regisztert is felhasználjuk, akár cirkuláris bufferekkel is dolgozhatunk. Ez egy olyan speciális hardver szervezésű tömb, mely esetén ha a tömbindex túllép a tömb maximális méretén, akkor az index visszaáll a tömb elejére (pontosabban az elejétől számítva annyiadik elemre, amennyivel túllépte az index a maximális méretet). A B regiszter tárolja a cirkuláris buffer kezdőcímét, az L a hosszúságát (byte-ban, ezért vigyázni kell az L értékének beállításakor, ha a bufferban nem egybyte-os adattípusokat fogunk tárolni), az I regiszterben van az aktuális cím (jelenleg hol tartunk a bufferben),

az M pedig azt, hogy az I által mutatott érték lefetchelése után az I értéke mennyivel változzon. A soron következő cím kiszámítása az alábbi képlettel történik:

$$I' = ((I + M - B) \bmod L) + B, \quad M < L$$

5.3. Számábrázolási módok

Az ADSP-BF537 hardveresen támogatja az aritmetikai műveleteket a 8, 16, 32, és 40 bites fixpontos számokon, de a többi szabványos C típushoz is van szoftveres támogatás. A leghatékonyabban a processzor a 16 bites számokkal tud dolgozni, a legtöbb művelet erre van optimalizálva. A leggyakrabban használt fixpontos adattípusok: előjel nélküli egész (az értéke 0 és 2^{15}-1 lehet), előjeles egész (-2^{15} … 2^{15}-1), előjel nélküli tört (0 … $1-2^{-16}$), és előjeles tört (-1 … $1-2^{-15}$); az előjeles számokat természetesen a processzor kettes komplemensű kódban ábrázolja. Ezek közül a szűrés miatt érdemes az egyik előjeles típust használni.

Ezt a típust – bár a szabványos C-nek nem eleme – nem csak az assembly kódban lehet használni, az Analog Devices ugyanis definiál két, az előjeles törtekre illeszkedő C típust: a fract16-ot (16 bites előjeles egész) és a fract32-t (32 bites előjeles egész). A kettő közül az előbbit választottam, ugyanis az A/D átalakító 24 bites, tehát 32 bites formátum használatakor is csak 24 bites pontosságú mérési eredményeink lennének. A feladathoz elegendő a 16 bites pontosság, ráadásul a processzor egy órajel ciklus alatt csak a 16 bites számok szorzásából képes kettőt megcsinálni.

Mivel a fract16 nem szabványos C típus, ezért nem is lehet C nyelvű programban a szokásos alapműveleteket végrehajtani ilyen típusú változókon. Az Analog Devices azonban a fejlesztői környezetbe integrált olyan függvényeket, melyekkel ezt meg lehet tenni; igaz, inkább assemblyben érdemes az ilyen műveleteket végrehajtani (a függvényhívást óriási erőforrás pocsékolás ilyesmire használni, mert számos regiszterművelettel jár).

5.4. Az EZ-LITE kártya

A fejlesztőkártya szintén az Analog Devies cég gyártmánya. A processzoron kívül számos periféria kapott rajta helyet, a feladat szempontjából azonban csak az USB programletöltő interfész, a nyomógomb interfész és az audio interfész volt fontos. Ez utóbbi egyébként egy codec, mely egy AD1871-es szigma-delta A/D konverterből, és egy AD1854-es szigma-delta D/A konverterből áll. A mintavételi frekvencia 48 kHz, és mintavételkor a codec megszakítást küld a processzornak. A jelfeldolgozó algoritmus ebben a megszakításrutinban fut le.

5.5. A szoftver eszközök

Az algoritmust megvalósító program fejlesztését a VisualDSP++ nevű fejlesztői környezettel végeztem. A környezet fő része egy projektszerkesztő, mellyel a forráskódokat tartalmazó file-okat szerkeszthetjük. A fejlesztés nyelve C++ és a kártya saját assembly-je. A fejlesztői környezet természetesen tartalmaz egy compilert is, és azokat a speciális könyvtári függvényeket is belefordíthatjuk a kódba, amit a cég a DSP processzorok számára fejlesztett ki. A C compiler mindig úgy próbálja optimalizálni a lefordított kódot, hogy az kihasználja a hardver által nyújtott speciális lehetőségeket. Ennek ellenére az időkritikus műveleteket – tipikusan ilyenek az online jeleldolgozási algoritmusok, melyek sokszor futnak le – érdemes assemblyben megírni. Ha csak ezeket a függvényeket írjuk meg assemblyben, de minden mást C-ben (olyan műveletek, melyek kevésszer futnak le, pl. inicalizálás), akkor az így kapott program nemcsak hogy gyors, de letisztult, áttekinthető szerkezetű is lesz.

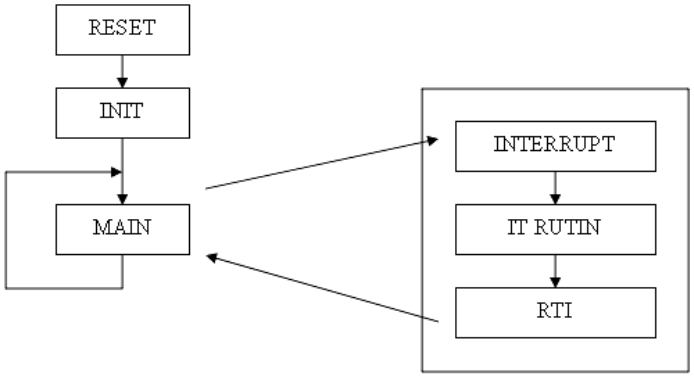
A lefordított programot a fejlesztői környezetbe integrált töltő program az USB interfészen keresztül fel is tölti a processzorra, és szoftveresen el is indítható annak futása. Futás előtt a programba töréspontokat is beállíthatunk, és mikor a program megáll egy ilyennél, megvizsgálhatjuk a teljes memóriaterületet és a processzormag összes regiszterének tartalmát. A memóriaterületeket ki is menthetjük, és van ”disassembly” lehetőség is: ilyenkor a memóriában tárolt értékekhez a processzor megpróbálja hozzátársítani azt az utasításnevet, ami hozzátartozik. Lehetőség van továbbá lépésről-lépésre való futtatásra is ebben az állapotban, ami nagymértékben

megkönnyíti a hibakeresést. Egy másik hasznos funkció a hisztogram, mely segítségével azt vizsgálhatjuk, hogy az egyes függvények a processzorkapacitás hány százalékát használják. Az online jelfeldolgozás valós idejű követelményei miatt ez roppant fontos, hiszen a jelfeldolgozó algoritmusok általában a teljes processzorkapacitást kihasználják, és ennek a funkciónak a segítségével könnyen kiválaszthatjuk azokat a függvényeket, melyek további optimalizálásra szorulnak.

A kártya hiányában a fejlesztőkörnyezet egy másik integrált modulját, a szimulátort is használhatjuk fejlesztésre. Ennek egyik nagy hátránya, hogy nem lehet valós idejű processzorterhelést mérni, valamint az eseményvezérelt programrészek (megszakításrutinok) tesztelése meglehetősen körülményes. A memória és a regiszterek tartalmának megtekintése, valamint a lépésről lépésre történő futtatás természetesen ebben az üzemmódban is engedélyezett.

6. Az algoritmust megvalósító szoftver

A programot a már bemutatott fejlesztői környezetben valósítottam meg, az EZ-LITE kártyán. A kártyán lévő codec mintavételi frekvenciája 48 kHz, és minden mintavételkor megszakítást küld a főprogramnak, mely egyébként végtelen ciklusban fut. Az alábbi ábrán látható a program (és általában a jelfeldolgozó programok) logikai felépítése:



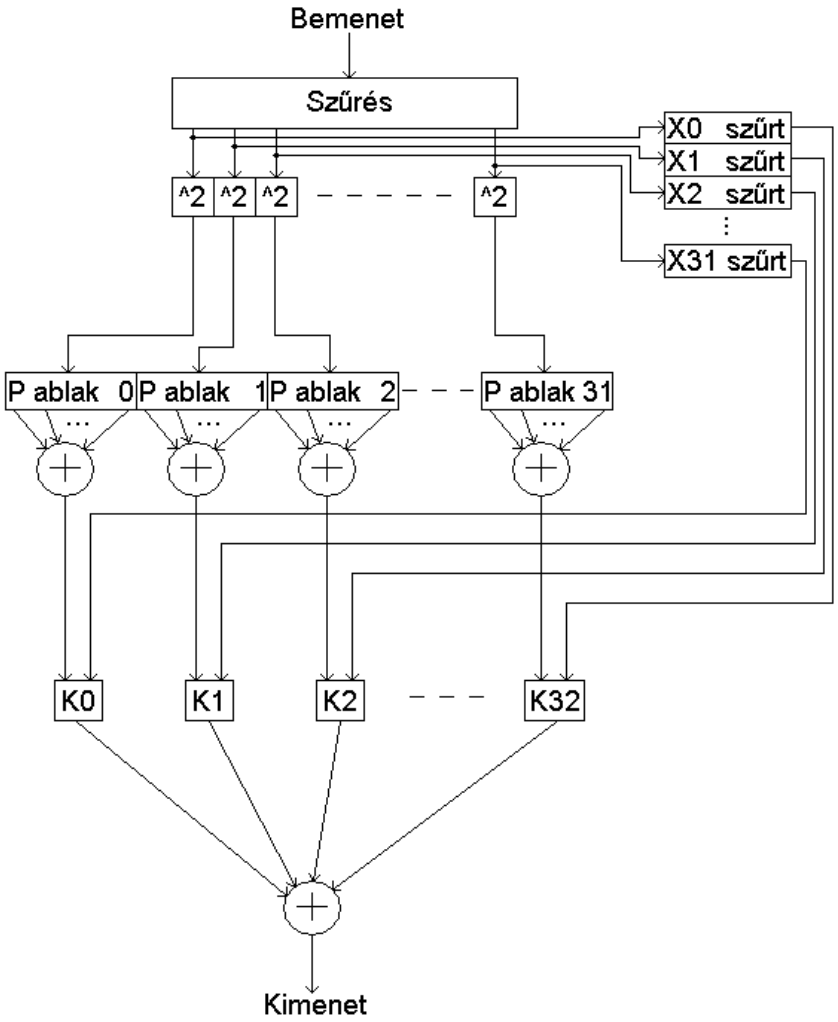
6.1. ábra. A program logikai felépítése

A mintavételi frekvenciát közvetlenül nem lehet kisebbre állítani, bár decimálással lényegében csökkenthető. Viszont zenei jelfeldolgozás esetén ezzel a 48 kHz-es mintavételi frekvenciával kell dolgoznunk. A jelfeldolgozó algoritmusnak minden két mintavétel között le kell futnia, másodpercenként majdnem ötvenezerszer. Mivel a processzor sebessége 600 MIPS, nagyjából 600MHz/50kHz = 12000 műveletből gazdálkodhat az algoritmus – elvileg, ugyanis a gyakorlatban a processzor csak MAC műveletből képes ennyit végrehajtani egy másodperc alatt (bár a gyártó mindenhol ezt a 600 MHz-es sebességet hirdeti), rendes műveletből csak 6000-et. Ha hozzávesszük, hogy maga a szűrés 32*128 = 4096 (kb 2000 órajelciklus) művelet, akkor az algoritmus többi részének végrehajtására kevesebb, mint 4000 művelet áll rendelkezésünkre, ami már nem is olyan sok.

A fejlesztés korai fázisában a programot C-ben írtam, de aztán át kellett térni az assemblyre, mert így lehetőségem volt kihasználni a hardveres ciklusszervezés által nyújtott lehetőségeket.

6.1. A program által megvalósított algoritmus

Az algoritmus működése az alábbi ábrán követhető nyomon:



6.2. ábra. Az algoritmus működése

A bemenet a szűrőbankba jut, ahol mindegyik sávon végbemegy a FIR szűrés. Ezután következik a négyzetre emelés, vagyis a pillanatnyi teljesítmény számítása. Az eredményt eltároljuk egy mozgó teljesítményablakba, ebből minden szűrőnek van egy. A teljesítményablakok aktuális elemeinek összegzése következik (átlagolás), majd az ablakösszegek aktuális értékeit az éppen kiválasztott karakterisztikától függően kényszerítjük a szűrőkimenetekre. Ezután következik az összegzés, és végül előáll a kimenet.

6.2. A szűrés implementálása

A szűrést egy assembly függvény hajtja végre.
A szűrést elvégző függvény szignatúrája:

```
extern int conv_asm(fract32 *, fract16 *, fract16 **);
```

Assembly függvény hívásakor három paramétert adhatunk át a hívott függvénynek, melyek sorrendben az R0, R1 és R2 regiszterbe kerülnek. Ha viszont több változót szeretnénk átadni, erre is van lehetőség: a szignatúrában lévő fract16** egy mutató, mely egy olyan tömbre mutat, melyben a függvénynek átadandó paraméterek mutatói vannak tárolva.

A másik fontos dolog a fract32* paraméter, amely a szűrőegyütthatókat tartalmazó tömb mutatója. Igaz, hogy az együtthatók 16 bitesek, de 32 bites számként vannak tárolva, így egy fetch művelettel két együtthatót is le tudunk tölteni a memóriából, tehát kihasználhatjuk a BF537 azon képességét, hogy egy órajel ciklus alatt két MAC műveletet is el tud végezni.

Megjegyzendő, hogy az utolsó, 32. sávszűrővel (amennyiben nulladiknak tekintjük a prototípusszűrőt) együtt 33 lenne a szűrők száma, és a fentebb leírt megvalósítás páros számú szűrők esetén optimális. Ez még nem lenne akkora baj, de az ott átengedett jelet (23675Hz-24000Hz) már az emberi fül úgysem képes meghallani. Ezért azt a szűrőt nem valósítottam meg.

Egy cirkuláris bufferben mindig a bemenet utolsó 128 értékét tároljuk. A szűrést végző program a FIR szűrők impulzusválasza és a bemenet konvolúcióját végzi el, a (3.12) szerint.

6.3. A teljesítményablak

Szűrés után a szűrők kimenetén megjelenő értéket a program kimenti a teljesítményablakokba, melyek szervezése cirkuláris buffer jellegű, és mindig a legutolsó W_L darab teljesítményértéket tartalmazza. A teljesítmények elmentését a

fract16 p_sav(fract16 *, fract16 *, fract16 *);

assembly függvény valósítja meg. Az elmentett teljesítményeket ablakonként összegezni kell, amit az

int powr_sum(fract16 *, fract16 *, fract16);

függvény hajt végre. Ez nem kevés művelet, 10 minta szélességű ablaknál például csak szorzásból 32*10=320 darab kell. Mindenesetre mindegyik általam megvalósított karakterisztikára belefér a processzoridőbe a 15-20 szélességű teljesítményablak, ami már elegendő.

Elvileg lehetne olyan algoritmust is készíteni, amely ablakszélesség-független műveletszámmal végzi el a teljesítményablakok elemeinek összegzését. Szűrőnként definiálhatunk egy változót, mely az éppen aktuális teljesítményösszeget tárolja. A teljesítményablakból mindig tudjuk, melyik a legrégibb, és melyik a legújabb teljesítményérték. Elég a teljesítményösszegből kivonni a legrégibb értéket, majd hozzáadni a legújabbat. Itt azonban komoly akadályokba ütközünk.

Tekintsük az alábbi szélsőséges, de szemléletes példát. Az értékeket decimálisan jelölve, az egyik szűrőn a teljesítményértékek sorban a következők:

0,9999 ; 0,9999 ; 0,9999 ; 0,9999 ; 0,9999 ; 0 ; 0 ; 0 ; 0

Ekkor az alábbi táblázatot kapjuk:

Diszkrét idő	p[0]	p[1]	p[2]	p[3]	P[4]	Tényleges összeg	Ábrázolható összeg	Régi összeg	Új összeg	Stabilizált összeg
1	0,9999	0	0	0	0	0,9999	0,9999	0	0,9999	0,9999
2	0,9999	0,9999	0	0	0	1,9998	0,9999	0,9999	0,9999	0,9999
3	0,9999	0,9999	0,9999	0	0	2,9997	0,9999	0,9999	0,9999	0,9999
4	0,9999	0,9999	0,9999	0,9999	0	3,9996	0,9999	0,9999	0,9999	0,9999
5	0,9999	0,9999	0,9999	0,9999	0,9999	4,9995	0,9999	0,9999	0	0
6	0	0,9999	0,9999	0,9999	0,9999	3,9996	0,9999	0	-0,9999	0
7	0	0	0,9999	0,9999	0,9999	2,9997	0,9999	-0,9999	-0,9999	0
8	0	0	0	0,9999	0,9999	1,9998	0,9999	-0,9999	-0,9999	0
9	0	0	0	0	0,9999	0,9999	0,9999	-0,9999	-0,9999	0

A ’Tényleges összeg’ oszlop a tömb elemeinek összegét tartalmazza. Az ’Ábrázolható összeg’ mező azt az értéket mutatja, amit akkor kapunk, ha minden órajelciklusban az összes tömbelemet összeadjuk. A többi három oszlop a fent leírt ablakszélesség-független összegzésre vonatkozik. A ’Régi összeg’ mező az új érték hozzáadása előtti teljesítményösszeget mutatja, az ’Új összeg’ mező pedig a hozzáadás utáni összeget. A tömb kezdetben 0-kal van feltöltve.

Mivel mindegyik fajta algoritmusnál van egy legfelső határa az átlagteljesítménynek, mely fölött a jelet nem manipuláljuk, ezért a hagyományos, műveletigényes összegzéssel, az ’Ábrázolható összeg’ oszlop adatait felhasználva, az algoritmus helyesen fog működni. Látható, hogy az ’Új összeg’ oszlopban (ablakszélesség-független összegzés) viszont negatív értékek is megjelennek, holott a tényleges ablakösszeg végig a maximálisan ábrázolható érték, vagy afelett van. Negatív nem lehet az átlagteljesítmény, ezért bevezethetünk egy stabilizáló mechanizmust: amennyiben a teljesítményösszeg negatív, 0-val tesszük egyenlővé. Ezt szemlélteti a ’Stabilizált összeg’ oszlop. Az algoritmusok azonban így sem fognak jól működni. A lépcsős karakterisztikánál például ha a határteljesítmény értéke 20, akkor elvileg minden ütemben ki kellene adni a szűrő kimenetén lévő jelet, változatlanul. Az újfajta összegzés alkalmazásakor azonban az utolsó öt ütemben a kimenet értéke 0 lesz, mivel a teljesítményösszeg a 20-as határ alatt van. Maradtam tehát a számításigényes, de biztosan jó eredményt adó, hagyományos teljesítményösszegzésnél.

Meg lehet oldani a feladatot egyébként úgy is, hogy 32 bites teljesítményösszeget használunk. A probléma ismertetését mégis fontosnak tartottam, mert a fejlesztési időből viszonylag sokat vett el a probléma forrásának felderítése.

6.4. A lépcsős karakterisztika megvalósítása

A lépcsős karakterisztika lényege – mint az előzőekben már leírtam – az, hogy ha az átlagteljesítmény nem ér el egy bizonyos küszöböt, akkor az aktuális sáv kimenete zérus lesz. A határteljesítményeket egy külön file-ban lehet beállítani, fract16 formátumban. A karakterisztikát a

fract16 comp_fcn(fract16 *, fract16 *, fract16 *)

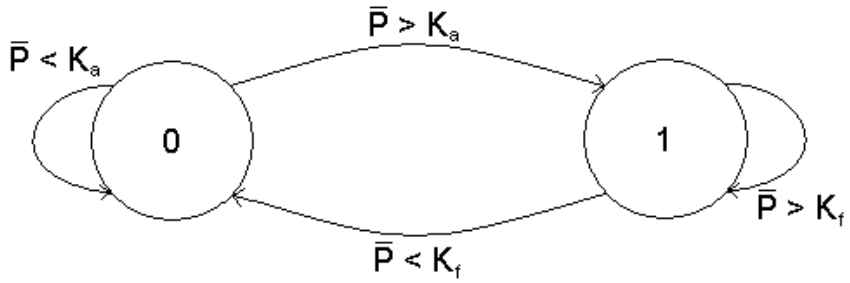
függvény kényszeríti a szűrőkimenetekre. Implementációs szempontból ezt a karakterisztikát a legkönnyebb megvalósítani.

6.5. A hiszterézises karakterisztika

Ezt a karakterisztikát a

fract16 hist_fcn(fract16 *, fract16 *, fract16 **)

függvény valósítja meg. Minden sávra definiálni kell a 4.2. ábrán lévő K_a és K_f paramétereket. A fordítóprogram ezt egy külön file-ból olvassa ki. A függvénynek a következő egyszerű állapotgép működése szerint kell eljárnia:



6.3. ábra. A hiszterézises karakterisztika állapotgépe

6.6. A négyzetes és a lineáris karakterisztika ofszettel és ofszet nélkül

Ezt a három karakterisztikát ugyanaz a függvény, a

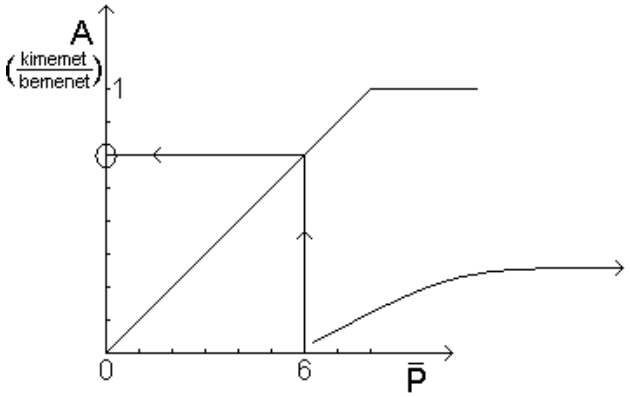
lin_fcn(fract16 *, fract16 *, fract16 *)

valósítja meg. A határteljesítmény alatt lévő tartományban 32 pont felbontásban minden értékhez tartozik egy szorzószám, ami a szűrőkimenethez tartozik, vagyis egy ilyen táblázatot kell generálni, és ebből kell kikeresni az aktuális teljesítményhez tartozó megfelelő szorzószámot.

Mivel ezt mind a 32 sávra el kell végezni minden ütemben, ezért a hagyományos keresések itt nem jöhetnek szóba. Lineáris keresés esetén az összehasonlítások számának várható értéke 32 érték mellett 16, minden sávban – ráadásul két szám nagyságának összehasonlítása nem egy processzorműveletbe kerül. Elvileg kevesebb lépésben lehet keresni, ha az értékeket fában tároljuk, azonban ehhez rekurzív függvényhívások kellenek, és ezért sokkal rosszabbul járunk, mint a lineáris keresés esetén.

A megoldás, ha az átlagteljesítményérték felhasználásával hajtunk végre címaritmetikát, melynek működését egy példán illusztrálom.

Legyen a felbontás 8 pont, és a teljesítményhatár is ennyi. Ha most az egyik szűrő kimenetén az átlagteljesítmény 6, akkor a karakterisztikáját tároló táblázat kezdőcíméhez 6 fract16 méretet hozzáadva (ez 2 byte, vagyis a kezdőcímhez 12-t adunk hozzá) megkapjuk azt a címet, ahonnan ki kell olvasni a megfelelő szorzószámot:



B, báziscím	0
B + 1 * méret(fract16)	0,125
B + 2 * méret(fract16)	0,25
B + 3 * méret(fract16)	0,375
B + 4 * méret(fract16)	0,5
B + 5 * méret(fract16)	0,625
B + 6 * méret(fract16)	0,75
B + 7 * méret(fract16)	0,875

6.4. ábra. A lineáris elnyomású karakterisztika programjának szemléltetése

A táblázatból a 0.75-ös értéket olvastuk ki, tehát ennyivel lesz megszorozva a szűrő kimenetén megjelenő érték. Alapesetben lényegében a teljesítményérték utolsó bitjét ki kell maszkolni, és a kapott értéket hozzáadni a táblázat kezdőcíméhez.

Az algoritmus egyik legnagyobb előnye, hogy bármilyen formájú karakterisztikát meg lehet valósítani, csak a táblázatot kell hozzá manipulálni. Az ofszetes és az ofszet nélküli karakterisztika megvalósítása csak abban különbözik egymástól, hogy az ofszetes esetében a táblázat első néhány eleme 0 (az alsó határteljesítményig). Érdemes a táblázat hosszát minél nagyobbra változtatni, nem baj, ha sokkal hosszabb, mint a határteljesítmény értéke – legfeljebb a felső részét 1-esekkel

töltjük fel. Ha minden sávnak külön karakterisztikát kívánunk beállítani, akkor viszont az általam megvalósított program esetében a kettő hatvány hosszúságú karakterisztika táblák maximális hossza 128 fract16 méretű, azaz 256 byte lehet. Nagyobb komparálási szintek beállításához ezért a teljesítményablakból több alsó bitet is ki lehet maszkolni, ilyenkor persze durvul a felbontás. Erre azonban nem volt szükségem, az átlagteljesítmény maximális komparálási szintjének megadható 128-as érték elegendőnek bizonyult.

6.7. A kimenet előállítása

A kimenet egyszerűen úgy áll elő, hogy az aktuális karakterisztikának már alávetett szűrőkimeneteket összegezzük. Az összegzést a

sum_res(fract16 , fract16 *, fract16)

assembly függvény végzi. Annak, hogy egy ilyen egyszerű műveletet is assembly függvény végez, az oka, hogy a fract16 típusú számot egy másik fract16-al C ben csak függvényhívás segítségével szorozhatunk össze – nyilván nem túl kifizetődő 32 függvényhívást elvégezni.

Az összegzett kimenetet 16 biten kapjuk meg, és mivel a D/A átalakító 24 bites, 8 bittel balra shift-eljük, mielőtt ténylegesen kiadjuk.

7. Mérési eredmények, kiértékelés

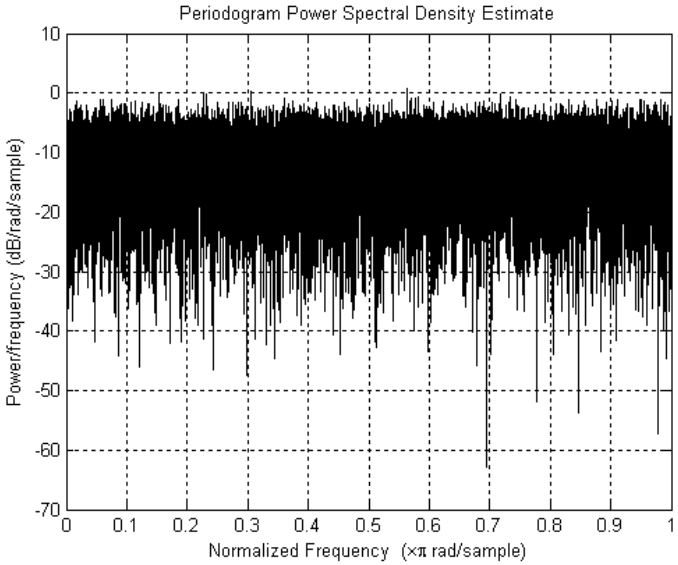
Az algoritmusok működését különféle zajos jelekre végeztem el. A tesztelés a fejlesztéssel párhuzamosan zajlott, majd a program elkészülte után következett a beállítások (teljesítményablak-hosszúság, komparálási szintek, karakterisztikák) finomhangolása. Megpróbáltam zenei jeleket, beszédhangot, és alapjeleket szűrni. Mivel a feladat lényege a szélessávú alapzaj eltávolítása a jelből úgy, hogy a kapott jel az emberi fül számára jobban érthető legyen, ezért a paraméterek beállítása meghallgatás útján történt. A célom az volt, hogy olyan beállításokat találjak, melyek sokféle jelre adnak jó eredményt.

A teljesítményablak mérete általában 20 minta volt. A processzorkihasználtság mindegyik tesztelésre használt program esetében meghaladta a 99%-ot.

7.1. Alapjelek tisztítása

Háromféle alapjelen próbáltam ki az algoritmust: 200 Hz-es négyszögjelen, 1 kHz-es háromszögjelen, és 1 kHz-es szinuszjelen, melyekhez zajt adtam hozzá. A jeleket MATLAB-ban állítottam elő, és wav file-ba írtam ki a wavwrite() függvénnyel. A jelek frekvenciájának pontos beállításához figyelembe kellett venni, hogy az általam a laboratóriumban használt MATLAB verzió nem tud olyan wav file-t készíteni, aminek a mintavételi frekvenciája 48 kHz. A maximális mintavételi frekvencia 44.1 kHz lehet. A hangkártya által kiadott jelalakokat és a frekvenciát oszcilloszkóppal ellenőriztem. A jelek amplitúdóját a hangerő-szabályozóval állítottam be a kívánt értékre.

A jelhez hozzáadott zajt a MATLAB random() függvényével állítottam elő. Ezzel a függvénnyel egy véletlen számokból álló vektort lehet előállítani, és be lehet állítani a használni kívánt eloszlást, és az eloszlásra jellemző paramétereket is. Az egyenletes eloszlással szélessávú, fehér zajt lehet generálni, ezért ennél maradtam. Az egyenletes eloszlású zajvektor periodogramjából látszik, hogy a zaj fehér jellegű:

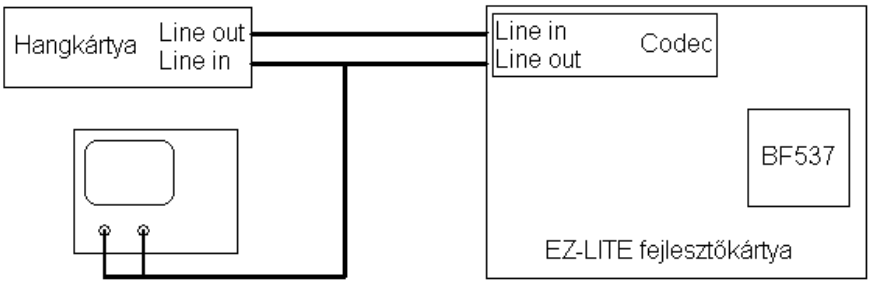


7.1. ábra. A méréshez használt zaj periodogramja

A zajvektor elemei a [-0.1 , 0.1] intervallumban vették fel az értéküket, míg a hasznos jel által felvehető két szélsőérték a -0.9 és a 0.9 volt, mindhárom jel esetében. Ebből kiszámolható a jel-zaj viszony is:

$$SNR = 10 \cdot \log\left(\frac{\sum_{i=1}^{N}(x_{jel})^2}{\sum_{i=1}^{N}(x_{zaj})^2}\right) \tag{7.1}$$

Négyszögjelre a jel tíz periódusát véve ez az érték 23 dB, háromszögjelnél 19 dB, szinusz jel esetében pedig 21 dB. A hasznos jel amplitudóját a szinusz és a háromszögjel esetében 2V-ra, míg a négyszögjelnél 1V-ra állítottam be. A mérési elrendezés blokkvázlata látható az alábbi ábrán:



7.2. ábra. A mérési elrendezés

A szűrőbank kimenetén megjelenő jel egyenlő kell, hogy legyen a bemenetére adott jellel, de a kimenet a bemenethez képest késleltetve van, méghozzá a FIR szűrő impulzusválasza hosszának felével. További, bár jóval kisebb mértékű késleltetést okoz a teljesítményablak (a késleltetés mértéke körülbelül az ablakszélesség fele, de a teljesítmény változásának gyorsaságától függ a pontos érték). Ezért a fejlesztőkártya audio interfész kimenetének egyik csatornájára a szűrőbank kimenetén megjelenő jelet, másik csatornájára az aktuális algoritmusnak alávetett szűrőbank kimenetet vezettem ki, így a két jel közötti különbség teljes egészében az algoritmus hatását mutatja, tehát a két jel azonos fázisban van, ami megkönnyíti a vizuális összehasonlítást. A hiszterézises karakterisztikát megvalósító programnál ez nem így van, ott a szűrőbank bemenete hasonlítható össze az algoritmussal szűrt kimenettel – ezzel a szűrőbank késleltetését próbáltam demonstrálni.

Oszcilloszkóp segítségével beállítható a hasznos jel értéke, és nyomon lehet követni a komparálási szintek beállításának jóságát. A könnyebb utólagos dokumentálhatóság, feldolgozhatóság miatt a hangkártya line-in bemenetét használva hangfelvételeket készítettem, melyeket MATLAB-bal dolgoztam fel. Az egyes karakterisztikák közötti különbségeket úgy vizsgáltam, hogy a küszöbteljesítmények mindegyik esetében ugyanakkorák voltak.

A háromszög és a szinusz jelre mind az öt algoritmus jól működött. A szinusz jel esetében, mivel csak egy fő komponense van, és arról tudjuk, melyik szűrőbank-csatornába esik, a többi sávban nagyra állítva a komparálási szinteket, sokkal jobb zajelnyomást lehet elérni, mint a többi jel esetében. Azonban, mivel a fehér zajra és bármely jelre általánosan jó beállításokat kerestem, mindhárom jelnél ugyanazokat a szinteket használtam.

A háromszögjel esetén a négyzetes és a lineáris karakterisztika simább jelet eredményez, ugyanakkor a háromszög csúcsa is tompább lett. A másik három karakterisztika esetében a háromszögjel lejtője zajosabb, de a csúcsok információtartalma jobban megmaradt.

A szinusz jel esetében már a lépcsős karakterisztika is jó eredményre vezet, számottevő különbség nem fedezhető fel a karakterisztikák között, mindegyik elég jól működik. A karakterisztikák szinusz és háromszögjelre kifejtett hatása megtalálható a mellékletben.

A négyszögjel spektruma 1/f jelleggel, lassan cseng le a frekvenciatartományban. Minden sávban megjelennek a komponensei, melyek

teljesítménye a lassú lecsengés miatt minden sávban elég nagy. Az időtartományban vizsgálva, az ugrásoknál mindegyik szűrőbank kimenetén nagy teljesítményű hasznos jel jelenik meg. Éppen ezért az ugrások helyén a zaj a hasznos jelre ráülve ”átszökik” a komparálási szinteken, minden sávban. A hangkártya véges sávszélessége miatt egyébként a hasznos jel is eltorzul, az ugrásoknál oszcilláció lép fel az aluláteresztő szűrés miatt.

A mellékletben látható a karakterisztikák zajos négyszögjelre kifejtett hatása. Megfigyelhető, hogy a négyszögjel közepén lévő zajt mindegyik karakterisztika erősen lecsillapította, ugyanakkor az ugrásoknál lévő nagyenergiájú hasznos jel komponensek miatt az ott lévő zaj mindegyik esetben nagymértékben átjutott. A legsimább jelalakot a négyzetes karakterisztika, és a lineáris, ofszetes karakterisztika esetében lehet megfigyelni.

A karakterisztikák viselkedése közötti különbségek egyébként legjobban akkor figyelhetőek meg, ha közvetlenül a zajvektorra alkalmazzuk azokat, a mellékletben ezek a mérések is megtalálhatók. Ebben az esetben a jel jellege (véletlen zaj) miatt az ábrák alapján a zajelnyomás mértékére, jóságára vonatkozóan nem érdemes messzemenő következtetéseket levonni, mindenesetre látszanak az egyes karakterisztikákra jellemző fontosabb vonások:

- a lineáris és a négyzetes karakterisztika esetében a jelben nincsenek olyan szakaszok, ahol a zaj értéke tartósan zérus
- a lineáris, ofszetes karakterisztika esetében vannak ilyen szakaszok, de nincsenek a megszűrt zajban olyan agresszív ugrások, mint a lépcsős karakterisztika esetében.

7.2. Régi hanglemezfelvételek zajának csökkentése

Három régi hanglemezfelvételen próbáltam ki az algoritmusokat, melyek meglehetősen rossz minőségűek. Mindhárom jellegzetes zajforma, a kattogás, sercegés, és alapzaj is nagy intenzitással van jelen a felvételeken. A sercegések és kattogások hatásának csökkentésére offline mediánszűrést használtam, és a különböző ablakszélességek közül a 20-as értéknél maradtam. Ez már eléggé széles ablak, és a nagyfrekvenciás komponensek észrevehetően gyengültek, de a sercegések hatása olyan

nagymértékben lecsökkent, hogy ez elfogadható kompromisszum. A mérési elrendezés ugyanaz volt, mint az előző esetben. A szűrt jelek megtalálhatóak a CD mellékleten.

Az alapzaj mértéke a hasznos jelhez képest olyan nagy mértékű, hogy az algoritmus nem tudta kellő hatékonysággal csökkenteni azt. Minden esetben megfigyelhető egyfajta bugyborékoló hang, mely némely esetben talán még zavaróbb, mint az eredeti zaj. Meglepő módon egyébként a legjobb eredményt a legegyszerűbb, lépcsős karakterisztika hozta. Ennek az lehet az oka, hogy a zaj mértéke olyan nagy, hogy többet ér, ha egyszerűen csonkoljuk az időtartományban a zajos jelet. A zajteljesítmény összességében lecsökken, igaz, a jelé is, de az emberi fül egy ilyen szakadozott jelet meg tud fejteni.

A többi karakterisztikával az a baj ebben az esetben, hogy mivel a zaj széles sávon rendelkezik nagy teljesítménnyel, minden sávban csak elnyomva azt, újfajta zajt kapunk, melynek teljesítménye jóval kisebb, mint az eredeti zajé, de mint már említettem, sokkal zavaróbb, bugyborékoló jellegű. Magasabbra állítva a komparálási szinteket, az eredeti jel amplitúdója is eltorzul, mely hullámzáshoz vezet. A legrosszabb eredményt a zajelnyomó karakterisztikák adták, a lineáris, és a négyzetes, vagyis amelyeknek görbéjében nincsen olyan szakasz, melyhez zérus erősítés tartozik. A hiszterézises és az ofszetes lineáris karakterisztikák némileg jobb eredményre vezettek.

7.3. Zajjal terhelt zenei jel tisztítása

Ebben az esetben ismét a MATLAB random() függvényét használtam, és az így kapott szélessávú zajt adtam hozzá egy zenei jelhez. A zenei jel egy Beethoven zongorakoncert volt. A hasznos jel, és a szűrt jel megtalálható a CD-n.

Kétféle zajjal próbáltam ki az algoritmus működését, melyek teljesítményben különböztek egymástól. Az egyik esetben a jel-zaj viszony 13dB, a másik esetben 27dB volt, a felvétel egészét tekintve. Figyelembe kell azonban venni, hogy a hasznos jel az időtartományban nem egyenletes – azoknál a részeknél, mikor az egész zenekar játszik, nyilván ennél nagyobb a jel-zaj viszony, amikor pedig csak az egyik hangszer szól, kisebb.

A mindkét zaj esetében a hiszterézises, és a lépcsős karakterisztika több zajt engedett át, és a hasznos jelet is jobban eltorzította. Ennél jobb eredményre vezetett az

ofszetes karakterisztika. A legjobban a lineáris és a négyzetes karakterisztika működött, nagy zaj esetében egyformán jól, kis zaj esetében a négyzetes egy kicsit jobban.

Kis zaj esetében az eredeti, zajjal terhelt jel némileg élvezhetőbb – az emberi fület a fehér zaj nem annyira zavarja. Az algoritmus a nagyobb zaj esetén adott az eredeti, zajos jelnél élvezhetőbb eredményt.

Minden algoritmusra jellemző, hogy mikor egyszerre több hangszer szól, és a komparálási szintek túl nagyra vannak állítva, az algoritmusnak alávetett jel esetében a a különböző hangszerek hangjai némileg összemosódnak, és a fül nem tudja annyira szétválasztani a hangokat, mint az eredeti, fehér zajjal terhelt jel esetében. Ebből a szempontból az algoritmusok akkor működtek a legjobban, mikor egyedül a zongora szólt.

7.4. Beszédhang tisztítása

Ennél a mérésnél a beszédhang Petőfi Sándor Szeptember végén című versének hangfelvétele volt, Hegedűs D. Géza előadásában. A mérési elrendezés ugyanaz volt, mint az eddigi esetekben. A hangminták megtalálhatóak a CD-mellékleten.

Beszédhang esetében könnyebb a komparálási szinteket beállítani, mint zene esetében. A hasznos jelben nagyon sok a szünet, amikor csak az alapzaj hallható. Első megközelítésként olyan nagyokra kell állítani a komparálási szinteket, hogy a háttérzaj teljes egészében eltűnjön. Ekkor a hasznos jelből is sok elveszhet, a jel-zaj viszonytól függően. Ezek után addig kell csökkenteni a szinteket, míg a beszédjelből minden hang felismerhető. Tapasztalataim szerint egyébként főleg a magánhangzók tűnnek el túl nagy komparálási szintek esetében, különösen az ’e’ hangra jellemző ez – míg például a kemény hangzású mássalhangzók (’t’,’p’) és a zöngés mássalhangzók nagyobb teljesítményűek.

Így kialakul egyfajta optimum, ahol a beszédjel hangszíne megváltozik ugyan, de felismerhető; ugyanakkor a zaj nagymértékben lecsökken. A hasznos jel torzulása beszéd esetében egyébként korántsem annyira zavaró a hallgató számára, mint zene esetében – a lényeg, hogy a szöveg tartalma érthető legyen.

Megtehetjük továbbá, hogy 8-10 kHz felett nagy komparálási szinteket állítunk be, mert az emberi beszéd sávszélessége még ennél is jóval kisebb, ezért a nagyobb frekvenciákon biztos, hogy csak zaj van, amit felesleges átengedni.

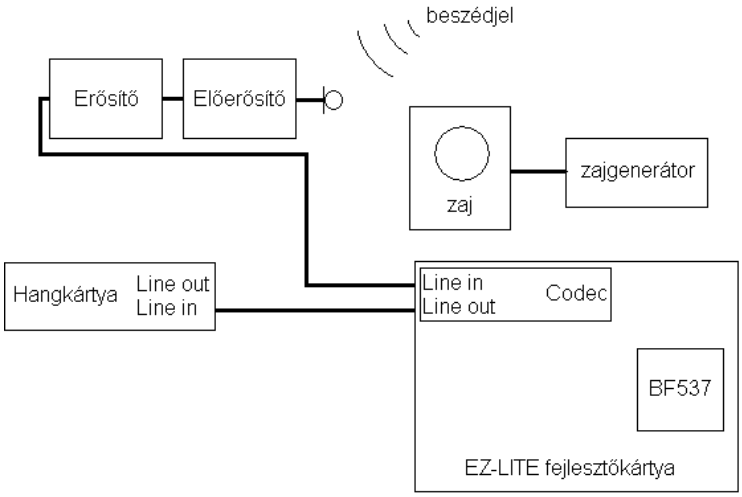
Ennél a mérésnél is egy kisebb, és egy nagyobb teljesítményű fehér zajt adtam hozzá a jelhez. A kisebb zajnál a jel-zaj viszony 31 dB, a nagyobbnál 17 dB volt, olyan jelrészletre nézve, amelyben nincsenek szünetek.

Kis zajjal terhelt jel esetén látszanak a karakterisztikák közötti különbségek a legjobban. A szüneteknél az ofszetes, hiszterézises, és lépcsős karakterisztikák esetében majdnem teljes csend van. Látszik, hogy a hasznos jelre ráülő zaj a hiszterézises és a lépcsős karakterisztikánál nagyobb mértékű, mint az ofszetesnél. A legtisztább a beszédjel a négyzetes és a lineáris karakterisztikákkal szűrt jelek esetében, azonban itt a háttérben a szüneteknél lévő zaj nem tűnt el teljesen, más a jellege, de jóval kisebb lett. A komparálási szintek ezen a zajszinten mindegyik karakterisztika esetén ugyanakkorák voltak, hogy a karakterisztikák működése között lévő különbségek jobban érzékelhetőek legyenek.

Nagy zaj esetén nem általános beállításokat kerestem, hanem az adott hangfelvétel zajának minél nagyobb mértékű csökkentése volt a célom. Itt a legjobb eredményt a lineáris, és a négyzetes karakterisztikák adták. Ezeknél minden sávban ugyanakkora volt a komparálási szint. Ugyancsak jó eredményt adott egy olyan karakterisztika, melynél a komparálási szintek sávonként különböztek, és lineárisan nőttek (a legkisebb szint a 0. sávban, a legnagyobb szint a 31. sávban volt); a sávokon belül pedig négyzetes elnyomás volt. A lépcsős karakterisztika esetében ismét teljes csend állt elő a szüneteknél, de a beszéd viszonylag zajos maradt. A legtisztább a jel a lineáris karakterisztika esetében lett, de itt a háttérzaj halk sistergéssé alakult át, ami a többi karakterisztika esetében nem volt megfigyelhető.

7.5. Beszédhang tisztítása valós körülmények között

A mérési elrendezés ebben az esetben más volt, mint az eddigiekben; az elrendezés az alábbi ábrán látható:



7.3. ábra. A mérési elrendezés

Az előerősítő és a hangolható erősítő által felerősített mikrofonjelet a kártya line-in bemenetébe vezettem, az algoritmus által megszűrt jelet pedig felvettem a hangkártyával. A zajforrás jelét vagy a zajgenerátor, vagy a hangkártya kimenete szolgáltatta. A jelforrás pedig az általam kiadott beszédjel volt.

A mikrofonjel már önmagában is eléggé zajos volt, ezért olyan méréseket is végeztem, ahol csak ezt a zajt próbáltam csökkenteni. Ennek oka egyébként az, hogy a mikrofont az előerősítővel kábel köti össze, mely így felerősíti a kábel zaját. Hozzáadódik ehhez az előerősítő zaja, majd pedig az előerősítőt a hangolható erősítővel összekötő kábel zaja; a hangolható erősítőbe már ez a zajos jel jut, amely további erősítésen megy keresztül. A mérés során a hangolható erősítőn 30dB-es feszültségerősítést állítottam be. A jel-zaj viszony közepes hangerejű beszéd esetén így hozzávetőlegesen 20-30 dB lehetett. Ez csak közelítés, a pontosabb számításokhoz nem állt rendelkezésre külön a hasznos jel, csak a zaj (olyan helyek az időtartományban, ahol nincs beszéd), és a zajjal terhelt jel.

Ugyanazokat a komparálási szinteket használtam, amit az előző mérésnél a kis zaj esetében. A jel zaj viszony itt nem volt annyira jó, a zajteljesítmény nagyobb volt, ezért a csendes részeknél is átjutott a zaj.

A legjobb eredményt a lineáris, és a négyzetes karakterisztikák adták, a beszédhang ezeknél volt a legtisztább, bár a háttérben itt megjelent az ezekre a karakterisztikákra jellemző bugyborékoló hang, mely akkor jelentkezik, ha az alapzaj túl nagy a küszöbteljesítményhez képest. A másik három karakterisztika esetében zaj jellege nem változik annyira meg az eredetihez képest, de az egyenletes zúgás helyett halkabb, ropogó zaj tűnik fel, a beszédhang pedig zajosabb. A hiszterézises karakterisztika esetében a háttérzaj némileg egyenletesebb, mint a lépcsős karakterisztikánál.

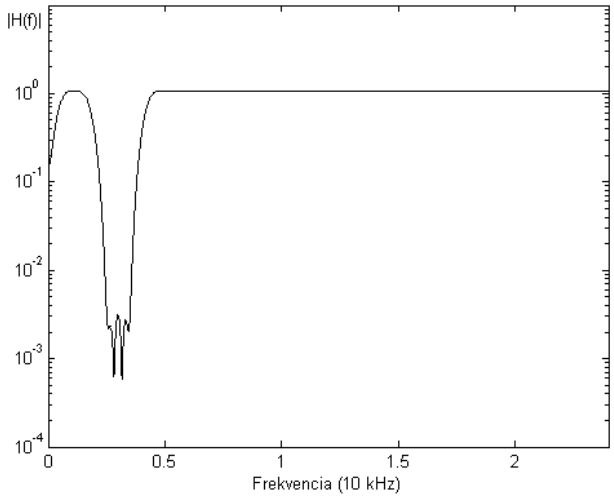
A zajgenerátorral többféle, főleg alacsonyfrekvenciás zajt generáltam. Ezeknek a teljesítménye jóval meghaladta a komparálási szinteket, ezért az algoritmusok viszonylag kis mértékben csökkentették a zajt. A hasznos jel a négyzetes elnyomású, és a lineáris elnyomású karakterisztikák esetében maradt meg a legmegfelelőbb mértékben.

7.6. A szűrőbank mint hangolható szűrő

Valós alkalmazásoknál előfordulhat, hogy a háttérben egy állandó, viszonylag keskenysávú, de zavaró hangú zajforrás szól. Ilyen esetben a lépcsős karakterisztikával érhetünk el jó eredményt. A zajforrás spektrumát megmérve megkapjuk, hogy a teljesítményének nagy része melyik sávba esik. Ezekben a sávokban olyan magasra kell állítani a komparálási szinteket, hogy a zajforrás átlagteljesítménye azt soha ne érje el. Ily módon tulajdonképpen egy lyukszűrőt kapunk. Az sem okoz nagy problémát, ha a zavarjel a beszédsávba esik, mert ha ki is maszkolunk néhány szűrőt ebben a sávban, a többi szűrőn átjut annyi beszédjel, hogy a beszéd érthető legyen. A többi sávban pedig, ha nem zérusra állítjuk a szinteket, még az alapzajt is csökkenthetjük.

Ennél a mérésnél egy szélessávú zajos jelet hoztam létre, majd hozzáadtam egy 35 Hz-es, egy 3000 Hz-es, és egy 3500 Hz-es szinusz jelet, melyek amplitudója a hangkártya által kiadható jel maximumához közel volt. A sávok közül a 0., 4., 5., és 6. sávokban a komparálási szintet a maximumra állítottam. Ezeknek a sávoknak a középfrekvenciája rendre: 0 Hz, 2250 Hz, 3000 Hz, 3750 Hz. A szűrők átlapolják egymást, és bár a 3000 Hz és a 3750 Hz középfrekvenciájú szűrőt kimaszkolva a zavarjel teljesítménye nagymértékben lecsökken, még mindig hallható. Ezért vannak

kimaszkolva környező szűrők is. Ha a többi sávban zérusra állítjuk a komparálási szinteket, akkor a szűrőbank eredő átvitele megváltozik. Ez látható az alábbi ábrán:



7.4. ábra. A módosított átvitel

A zajos jelre alkalmazva a lépcsős karakterisztikát, a zavaró nagyfrekvenciás komponensek eltűntek. A kimaszkolt sávok beleesnek a beszéd spektrumába, ezért abból is elvesznek a komponensek, de a beszéd ettől még tisztán érthető – igaz, némileg halkabb.

Erre a típusú zajra jó eredményt adott még a négyzetes elnyomású karakterisztika, ha a zajszint jóval kisebb volt.

8. Összefoglalás, kitekintés

Diplomatervem célja egy olyan zajcsökkentő eljárás kidolgozása és implementálása volt, mely nagy hatékonysággal csökkenti a szélessávú zajokat. A feladat megoldásához a szűrőbankokat választottam, mert az ily módon felbontott jel fontosabb sávjaiban más feldolgozási eljárás alkalmazható, mint a kevésbé fontos sávokban. A szűrőbank tervezésekor az ”Önálló Laboratórium” című tantárgy során az ezzel kapcsolatban megszerzett tapasztalataimat is felhasználtam. Ezek után következett a megvalósítani kívánt karakterisztikák megválasztása.

A szűrőbank és a karakterisztikák megtervezése után meg kellett ismerkednem a hardver és szoftver eszközök képességeivel, korlátaival. A legnagyobb kihívást itt a programszervezés jelentette, ugyanis a szűrés és a karakterisztikák alkalmazása rendkívül számításigényes feladat. A korábban már megtervezett szűrőbank specifikációt a számítást végző erőforrás korlátaihoz, és a valós idejű követelményekhez kellett igazítani.

A fejlesztés során folyamatosan kellett a program működését tesztelni, ezért menet közben a komparálási beállítások már nagyjából kirajzolódtak. A fejlesztés befejezése után következett a beállítások pontosítása, majd tesztelése audio jelekre. A pontos beállítások megtalálása az algoritmus és a feladat jellege miatt rendkívül időigényes folyamat volt, hiszen minden beállítás teszteléséhez újra kellett fordítani a programot, és le kellett tölteni a kártyára.

A rendszer továbbfejlesztésének módja attól függ, mire szeretnénk azt használni. Zenei jelek tisztításához mindenképpen érdemes lenne egy olyan interfészt készíteni, melyen on-line módon állíthatóak lennének a komparálási szintek. Régi hanglemezfelvételek on-line tisztításához ezen kívül a jelfolyamba lehetne helyezni a rendszer elé egy kattogás detektáló rendszert, amely a kattogás detektálása után az ott eltorzult jelet valamilyen interpolációs eljárással csökkentené. Zenei jelek feldolgozására a mérések alapján a lineáris és a négyzetes karakterisztika tűnik a legalkalmasabbnak. Beszédhang on-line feljavítására alkalmas a lépcsős karakterisztika ipari környezetben, ha állandó, keskeny sávú zavaró háttérzaj van jelen; ilyenkor a megszűrt jel sokat veszít a teljesítményéből, de ez egy erősítő segítségével megoldható, és a visszamaradt beszédjel tisztább, érthetőbb lesz. A többi sávban pedig még az alapzaj teljesítménye is csökkenthető.

9. Irodalomjegyzék

[1] Bakó Tamás Béla – Zenei anyagok zajcsökkentése (A cikk elektronikus formátumban megtalálható a következő webcímen: http://home.mit.bme.hu/~bako/ZAOZ/zaz.htm)

[2] www.cedar-audio.com (Introduction to Audio Restoration)

[3] The Bark Frequency Scale - http://ccrma.stanford.edu/~jos/bbt/Bark_Frequency_Scale.html

[4] Helmut Bölcskei, Franz Hlawatsch – Oversampled Cosine Modulated Filter Banks with Perfect Reconstruction (IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing, Vol. 45, No. 8, August 1998, p. 1058)

[5] Davis Pan – A Tutorial on MPEG/Audio Compression (IEEE Multimedia Journal, Summer 1995 issue)

[6] VisualDSP++ 4.5 C/C++ Compiler and Library Manual for Blackfin Processors (Revision 4.0, April 2006)

[7] ADSP-BF53x/BF56x Blackfin Processor Programming Reference (Revision 1.1, February 2006)

10. Függelék

A CD-ROM tartalma:

Hangminták:
a 7.2, 7.3, 7.4, 7.5, 7.6 fejezetekhez.

Programok forráskódjai:

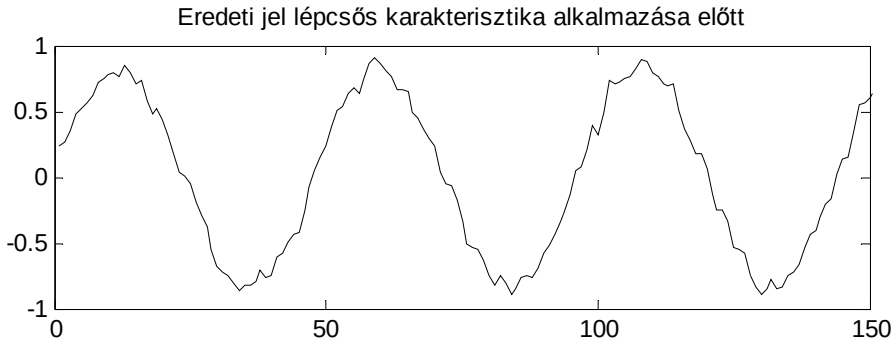
”Lepcsos”: a lépcsős karakterisztikát megvalósító program, tetszőlegesen nagyra állítható komparálási szintekkel.

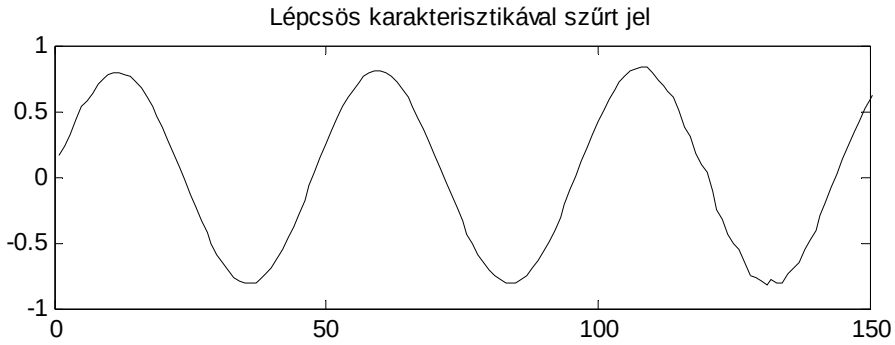
”Hiszterezises”: a hiszterézises karakterisztikát megvalósító program, tetszőlegesen nagyra állítható komparálási szintekkel.

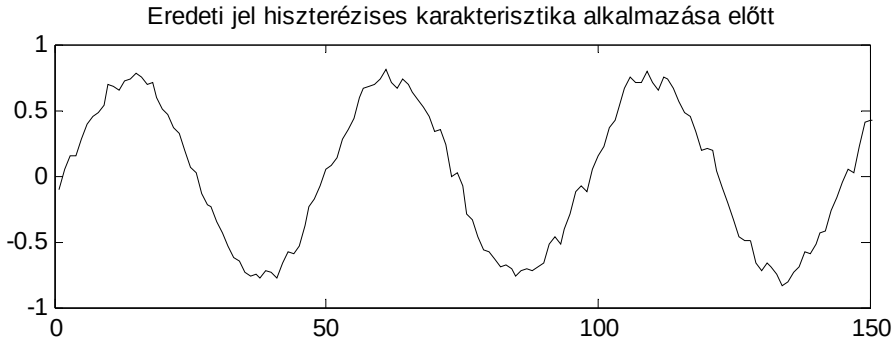
”Linearis_01”: a négyzetes, ofszetes, lineáris, és lépcsős karakterisztika valósítható meg vele. A ”k_<karakterisztika neve>_32” elnevezés konvenciójú file-okban konkrét beállítások is megtekinthetők. A szűrőknek egy közös karakterisztika táblája van. Tesztelésre alkalmas program, mert gyorsan állíthatóak a karakterisztika értékek. A maximálisan beállítható komparálási szintet a maszkolás és a táblázat hossza határozza meg.

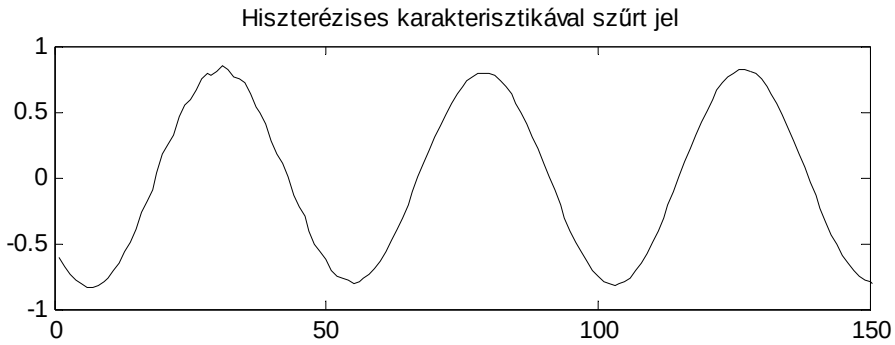
”Linearis_02”: a négyzetes, ofszetes, lineáris, és lépcsős karakterisztika valósítható meg vele. Minden szűrőnek saját, file-ból beolvasható karakterisztika táblája van, ezért finombeállításra alkalmas. A maximálisan beállítható komparálási szintet a maszkolás és a táblázat hossza határozza meg.

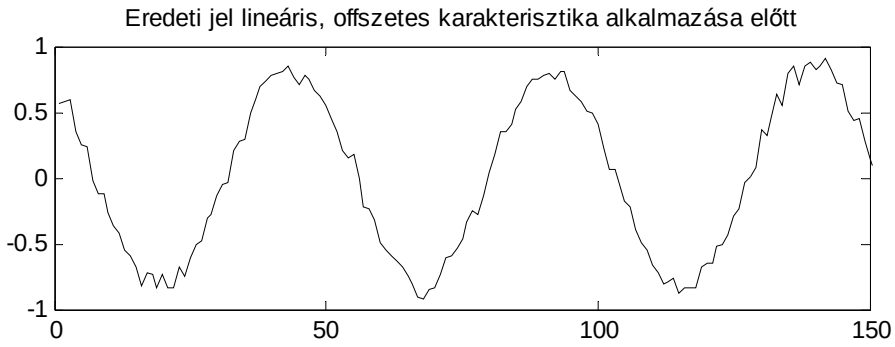
Alapjelek tisztítása: szinusz jel

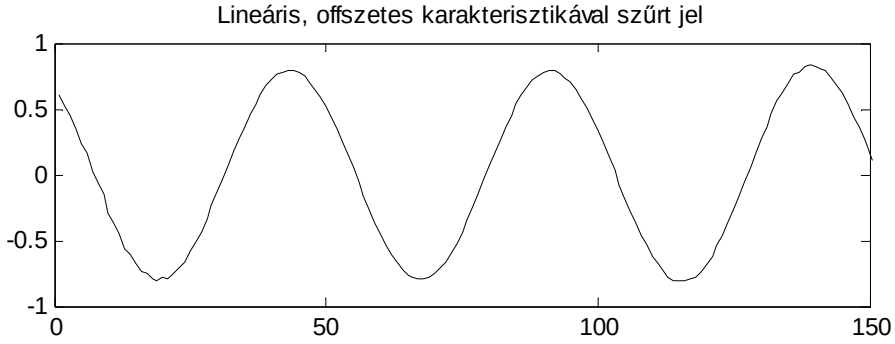


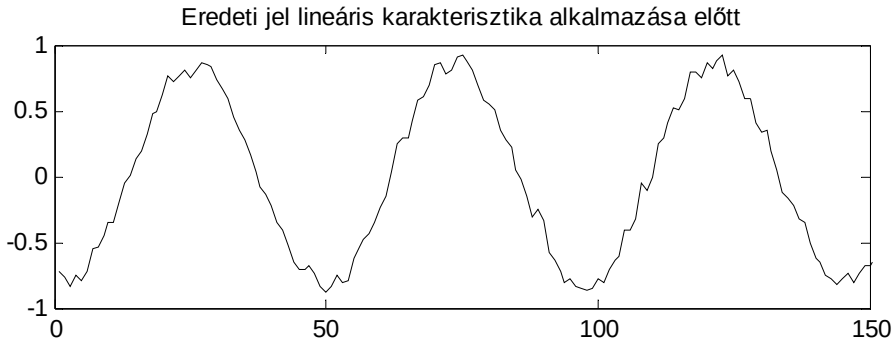


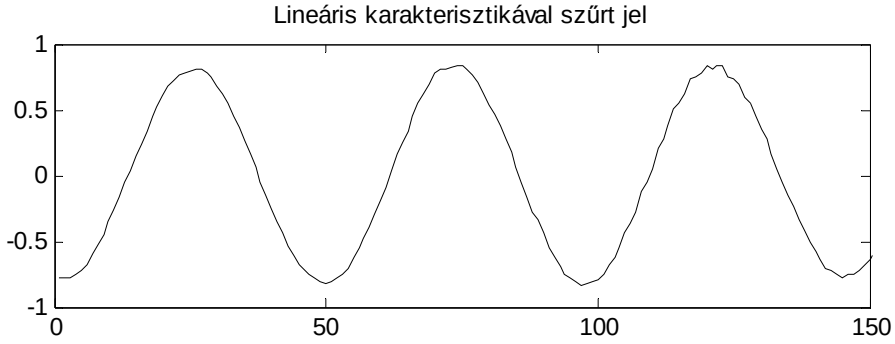




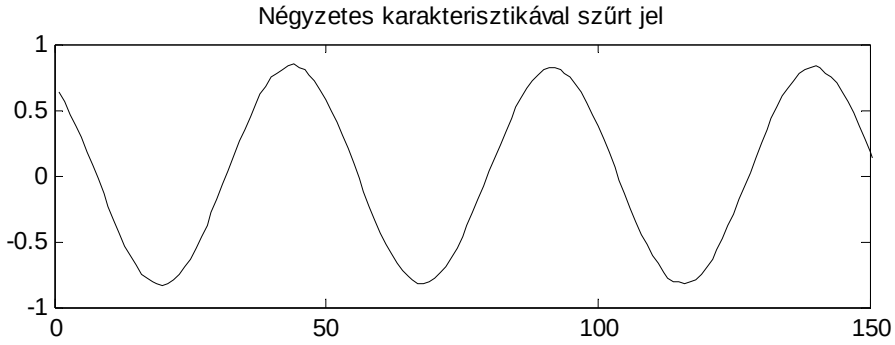

Eredeti jel lineáris, offszetes karakterisztika alkalmazása előtt
1
0.5
0
-0.5
-1
0
50
100
150


Lineáris, offszetes karakterisztikával szűrt jel
1
0.5
0
-0.5
-1
0
50
100
150

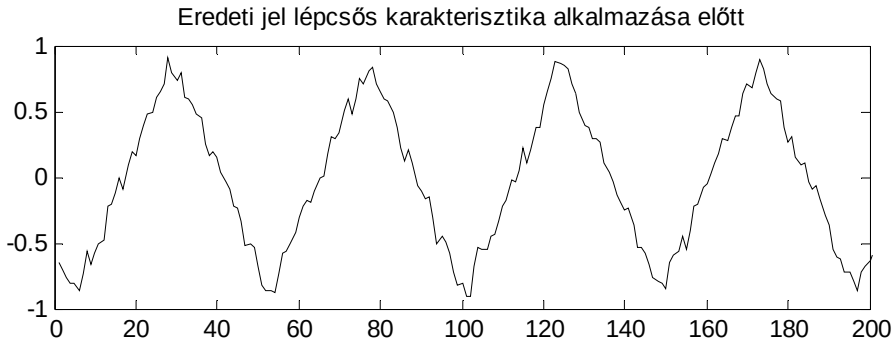

Eredeti jel lineáris karakterisztika alkalmazása előtt
1
0.5
0
-0.5
-1
0
50
100
150

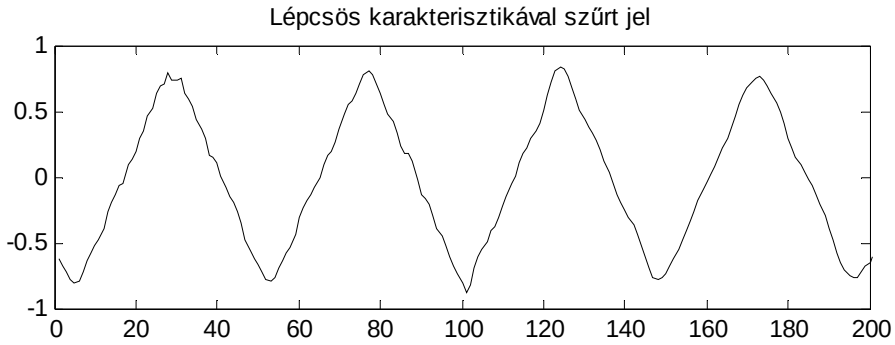

Lineáris karakterisztikával szűrt jel
1
0.5
0
-0.5
-1
0
50
100
150

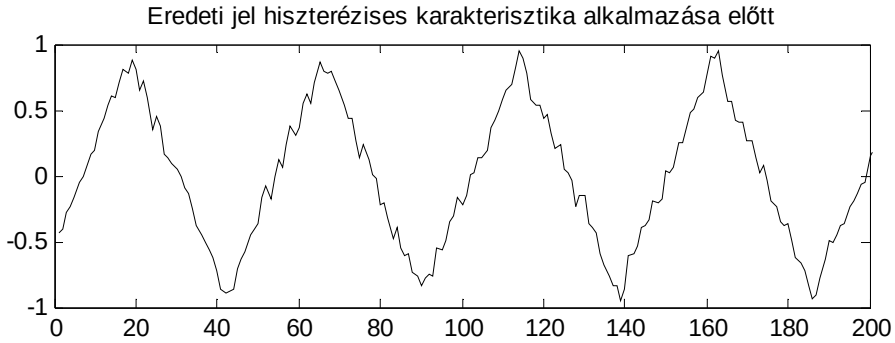

Eredeti jel négyzetes karakterisztika alkalmazása előtt
1
0.5
0
-0.5
-1
0
50
100
150

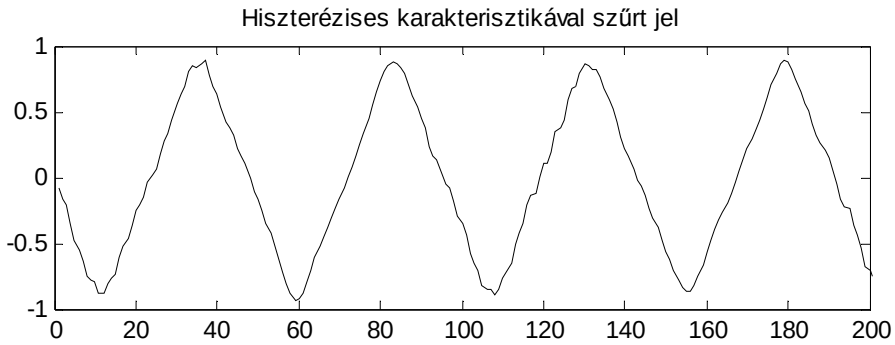

Négyzetes karakterisztikával szűrt jel
1
0.5
0
-0.5
-1
0
50
100
150

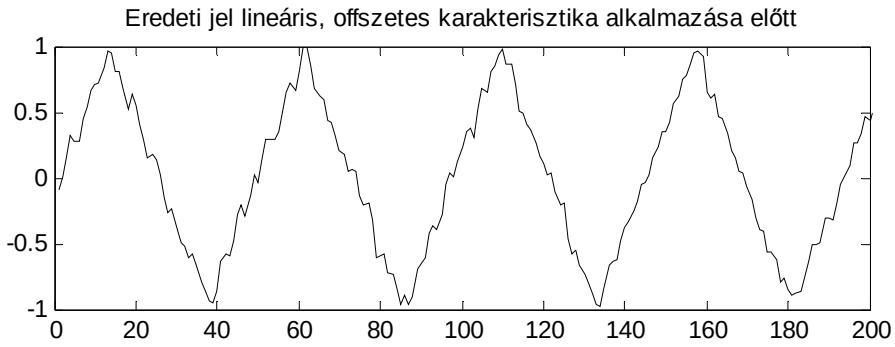
Alapjelek tisztítása: háromszögjel

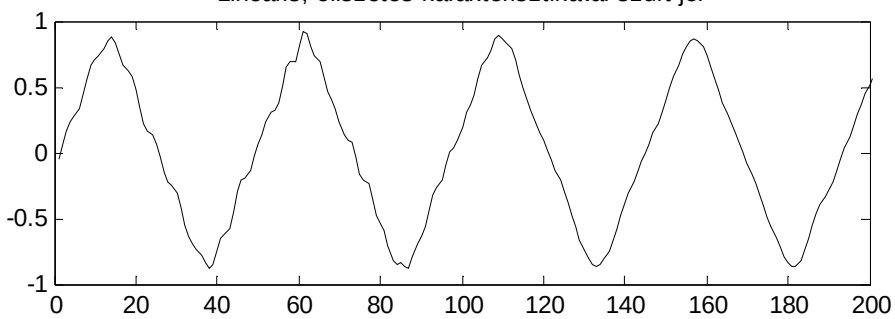


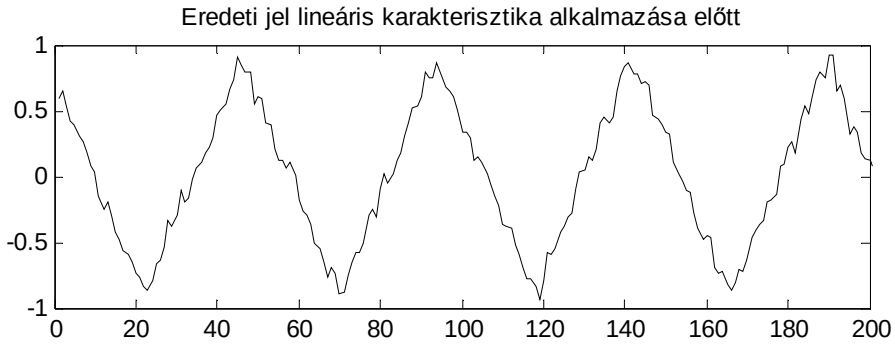


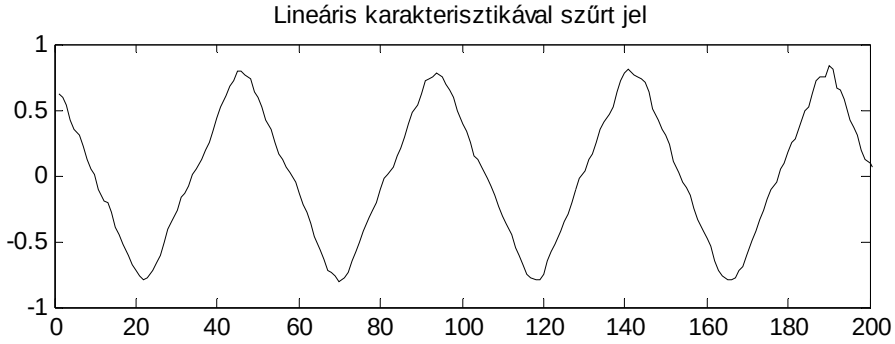


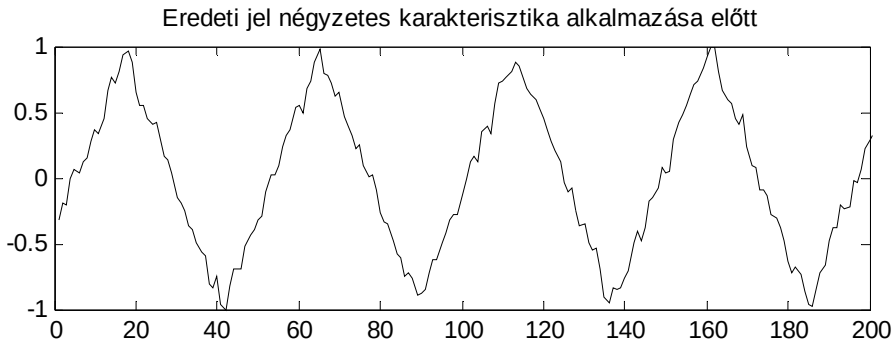


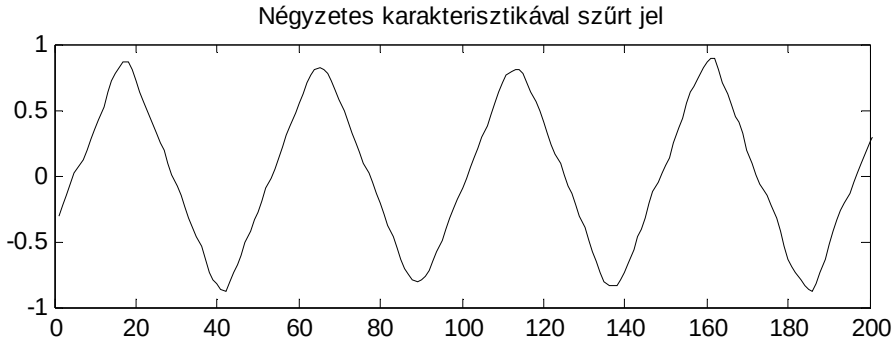

Eredeti jel lineáris, offszetes karakterisztika alkalmazása előtt
1
0.5
0
-0.5
-1
0
20
40
60
80
100
120
140
160
180
200


Lineáris, offszetes karakterisztikával szűrt jel
1
0.5
0
-0.5
-1
0
20
40
60
80
100
120
140
160
180
200

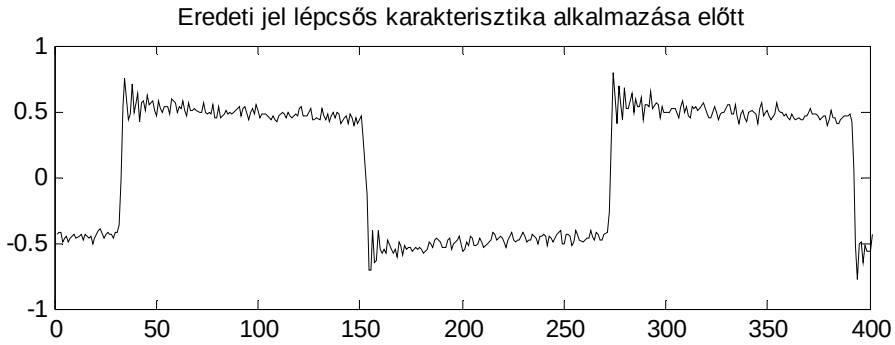

Eredeti jel lineáris karakterisztika alkalmazása előtt
1
0.5
0
-0.5
-1
0
20
40
60
80
100
120
140
160
180
200

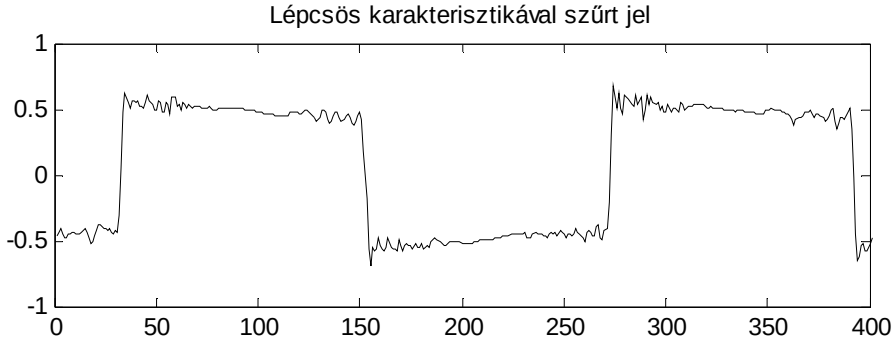

Lineáris karakterisztikával szűrt jel
1
0.5
0
-0.5
-1
0
20
40
60
80
100
120
140
160
180
200

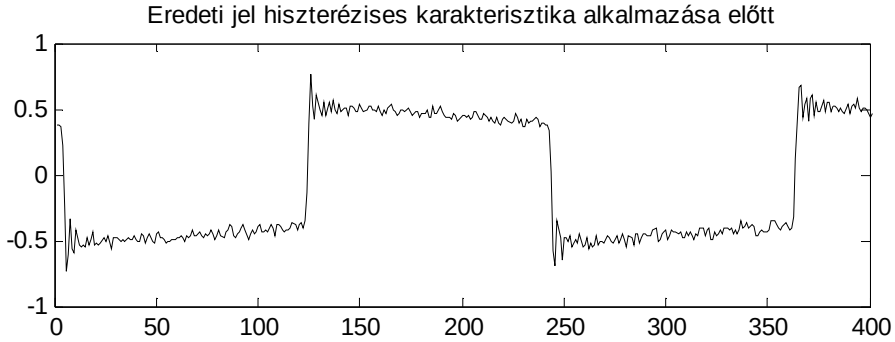

Eredeti jel négyzetes karakterisztika alkalmazása előtt
1
0.5
0
-0.5
-1
0
20
40
60
80
100
120
140
160
180
200

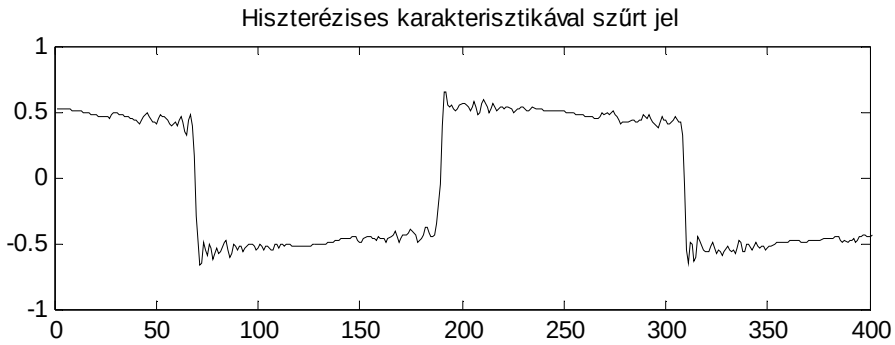

Négyzetes karakterisztikával szűrt jel
1
0.5
0
-0.5
-1
0
20
40
60
80
100
120
140
160
180
200

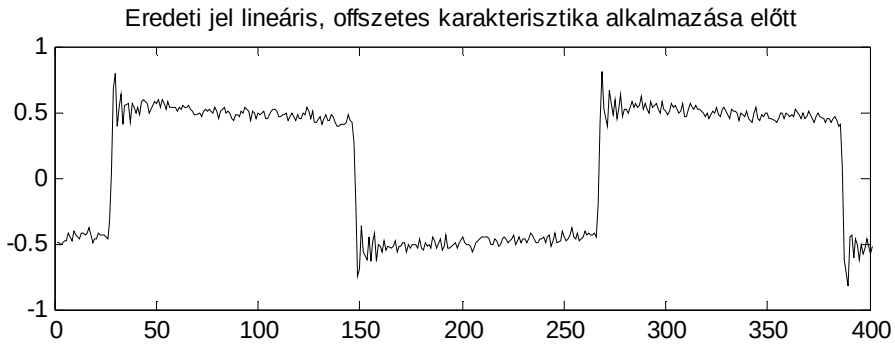
Alapjelek tisztítása: négyszögjel

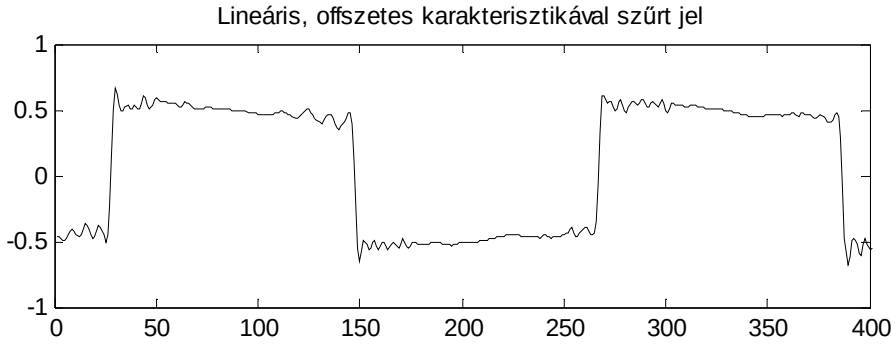


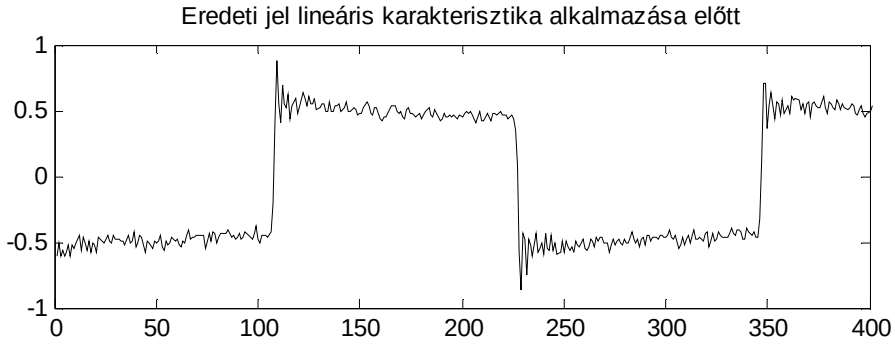


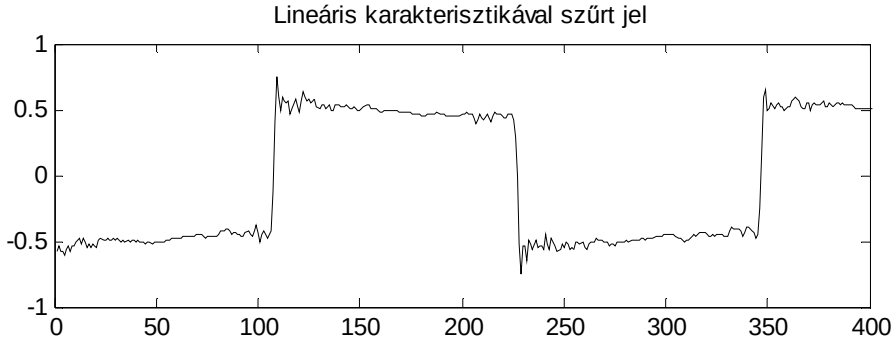


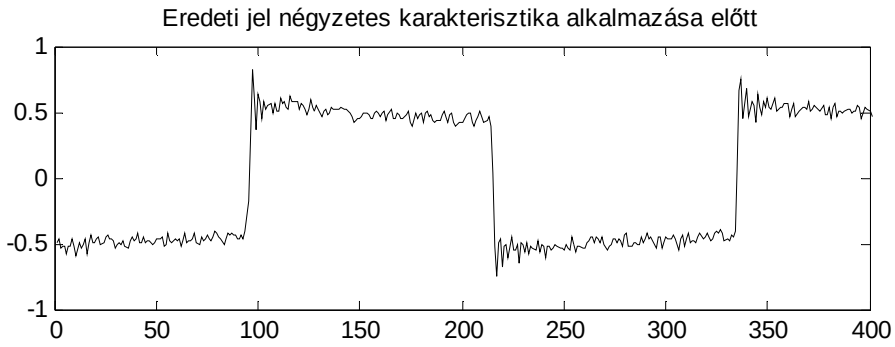


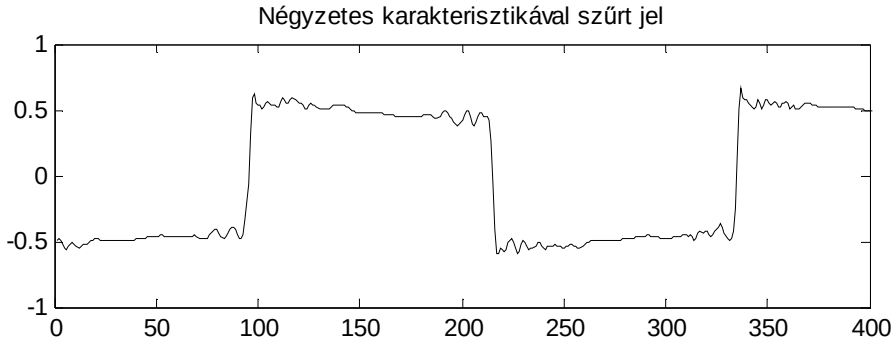

Eredeti jel lineáris, offszetes karakterisztika alkalmazása előtt
1
0.5
0
-0.5
-1
0
50
100
150
200
250
300
350
400


Lineáris, offszetes karakterisztikával szűrt jel
1
0.5
0
-0.5
-1
0
50
100
150
200
250
300
350
400

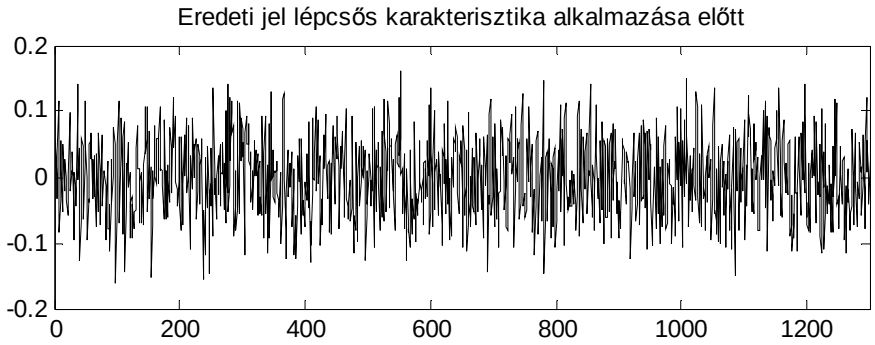

Eredeti jel lineáris karakterisztika alkalmazása előtt
1
0.5
0
-0.5
-1
0
50
100
150
200
250
300
350
400

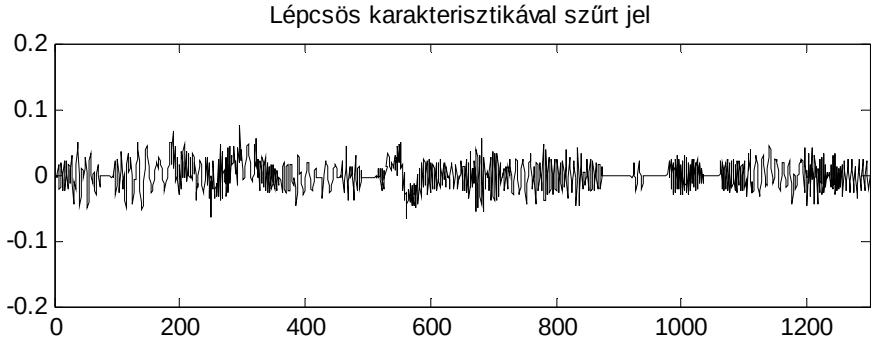

Lineáris karakterisztikával szűrt jel
1
0.5
0
-0.5
-1
0
50
100
150
200
250
300
350
400

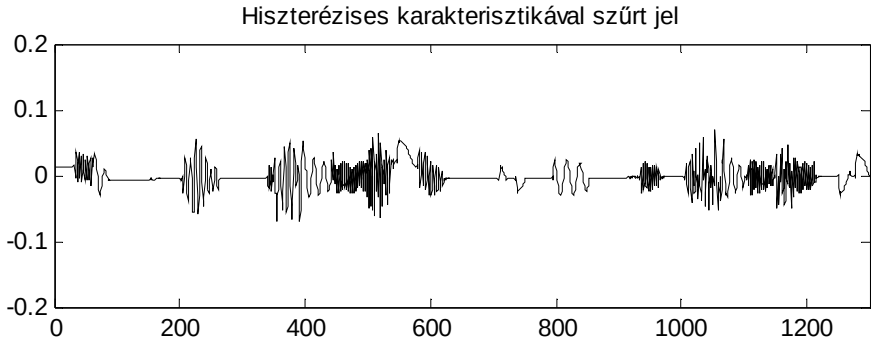

Eredeti jel négyzetes karakterisztika alkalmazása előtt
1
0.5
0
-0.5
-1
0
50
100
150
200
250
300
350
400

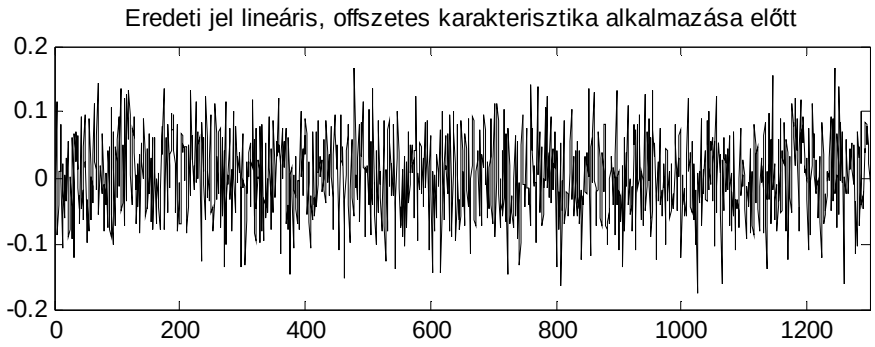

Négyzetes karakterisztikával szűrt jel
1
0.5
0
-0.5
-1
0
50
100
150
200
250
300
350
400

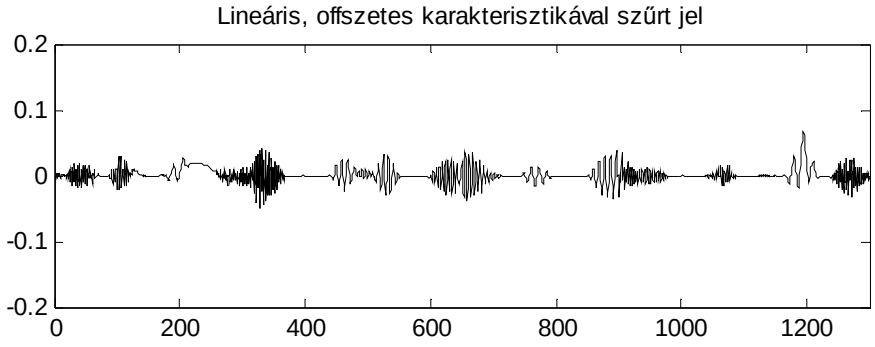
Alapjelek tisztítása: az előző jeleket terhelő zaj

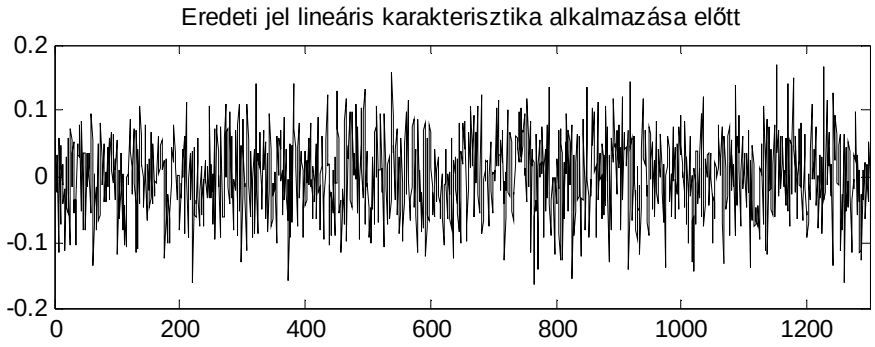


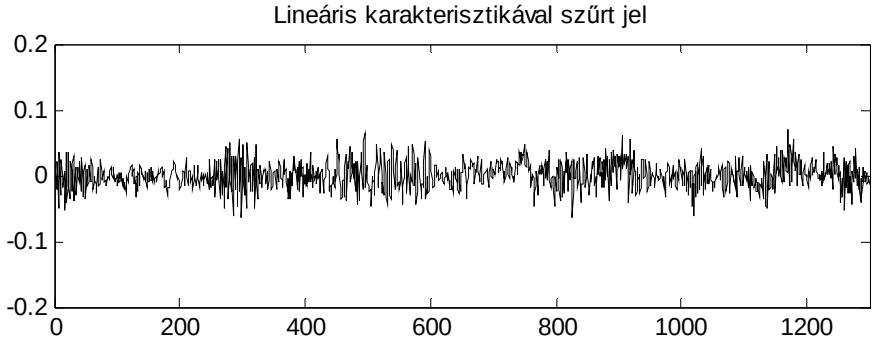


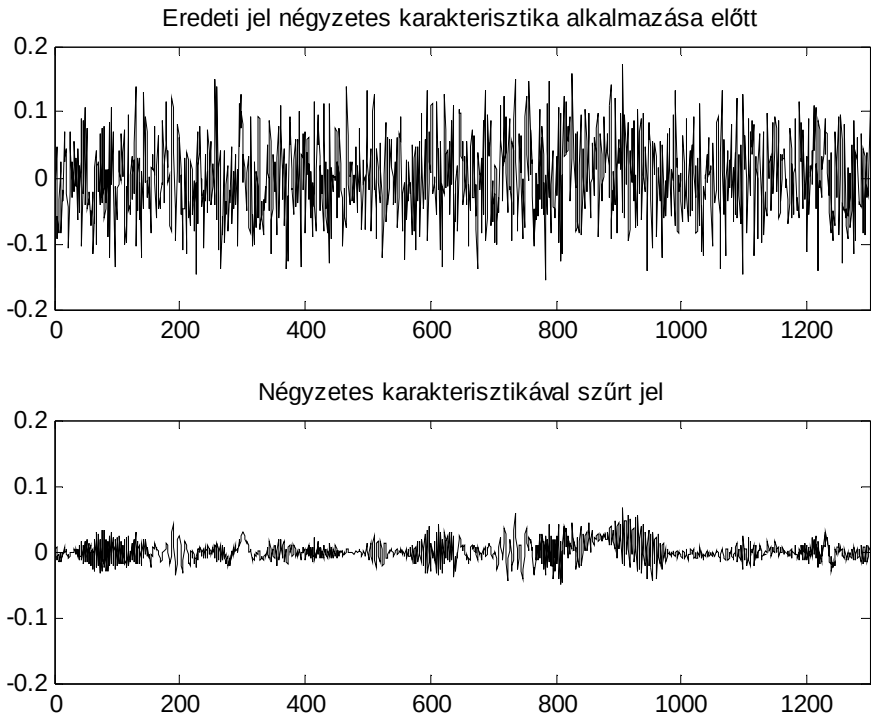



Eredeti jel lineáris, offszetes karakterisztika alkalmazása előtt
0.2
0.1
0
-0.1
-0.2
0
200
400
600
800
1000
1200


Lineáris, offszetes karakterisztikával szűrt jel
0.2
0.1
0
-0.1
-0.2
0
200
400
600
800
1000
1200


Eredeti jel lineáris karakterisztika alkalmazása előtt
0.2
0.1
0
-0.1
-0.2
0
200
400
600
800
1000
1200


Lineáris karakterisztikával szűrt jel
0.2
0.1
0
-0.1
-0.2
0
200
400
600
800
1000
1200


Eredeti jel négyzetes karakterisztika alkalmazása előtt
0.2
0.1
0
-0.1
-0.2
0
200
400
600
800
1000
1200
Négyzetes karakterisztikával szűrt jel
0.2
0.1
0
-0.1
-0.2
0
200
400
600
800
1000
1200