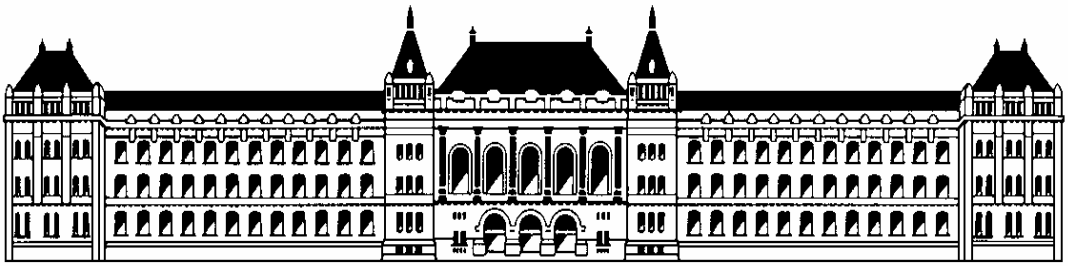


Diplomaterv

Tóth László

2001

Diplomaterv feladatkiírás

Nyilatkozat

Alulírott ***Tóth László***, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és 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.

.....

Tóth László

Tartalomjegyzék

Tartalmi összefoglaló

Dolgozatomban a beszédhangot terhelő zaj szűrésével foglalkozom. A feladatom egy olyan on-line algoritmus megvalósítása digitális jelfeldolgozó processzoron, amely megfelelően képes a beszédhangon lévő szélessávú zajt valós időben csökkenteni.

Feladatom során először mindig Matlab szimulációt végeztem az adott algoritmus tulajdonságainak megismerésére és helyes működésének megállapítására, majd mikor az algoritmus helyesen működött, akkor írtam meg a jelfeldolgozó processzorra a programot.

A zajszűrést a jel frekvenciasávokra bontásával és a frekvenciasávokon belüli feldolgozásával oldom meg. A jel sávokra bontása szűrőbank alkalmazásával oldható meg. A szűrőbankok között speciális helyet foglal el az ún. rezonátoros megfigyelő struktúra, amely paramétereinek megfelelő beállításával rekurzív Fourier-transzformációt valósít meg, így jól alkalmazható on-line feldolgozásokra.

Első lépésként a rezonátoros megfigyelő struktúrával és annak tulajdonságaival kellett megismerkednem, majd Matlabbal szimulálnom a működését.

Második lépésként a soft- és hard-elbow zajszűrő algoritmusokkal ismerkedtem meg. Ezen algoritmusok és azok változatainak működését és összehasonlítását szintén Matlabbal végeztem. Az Matlabban megírt algoritmusok tesztelését egy versrészlet felhasználásával oldottam meg. Majd a kiválasztott módosított soft- és hard-elbow algoritmust jelfeldolgozó processzorra is beprogramoztam.

Ezután a processzorra megírt programok működésének tesztelését végeztem el. Ebben az esetben a teszteléshez egy rádiót használtam. A tesztelések során fellépett problémák megoldása után a „tökéletesített” programot teszteltem, és ennek állapítottam meg a tulajdonságait.

Az elkészült program használható olyan helyeken, ahol beszédet nagymértékű zaj terheli, de a jel/zaj viszony még pozitív.

Abstract

I deal with filtering the noise on speech voice in my thesis. My task is the realisation of an on-line algorithm on digital signal processor, which is able to reduce the white noise on speech sound in real time properly.

During my task first I always carried out Matlab simulation to know the features of a given algorithm and its proper running. When the algorithm run correctly I wrote the program for the digital signal processor.

I work out noise filtering with dissolving the signal to frequency bands and processing it within the frequency bands. The signal can be dissolved to bands with a filterbank. The resonator band observer structure is a special filterbank which realise recursive Fourier-transform with proper parameters so it can be applied well to on-line processing.

First I had to learn the resonator based observer structure and its features and then I had to simulate its running with Matlab.

Secondly I learnt the soft- and hard-elbow noise-filtering algorithm. I also carried out the comparising and running of these algorithms and their variations with Matlab. I tested the algorithms written in Matlab with a part of a poem. Then I programmed the chosen modified soft- and hard-elbow algorithms on a signal processor, too.

After this I tested the running of the programs written on the processor. In this case I used a radio for testing. When I solved the problems appearing during the tests, I tested the improved program and found out its features.

This program can be used in such cases where a noise is on speech sound, but the signal/noise ratio is still positive.

1 Bevezetés

Gyakorlatban sokszor előfordul, hogy az elektronikusan továbbított hangjelet (beszédhangot) nagyteljesítményű zaj terheli. Ez a zaj adódhat az akusztikai rendszerből (mikrofonba a háttérzaj is bekerülhet, és be is kerül, ennek teljesítménye számottevő lehet, spektruma pedig elég változatos), vagy az elektronikus jelútból is származhat. A beszédhangot terhelő zajteljesítmény függvényében felmerülhet a zajszűrés alkalmazásának igénye.

A jelfeldolgozó processzorok megjelenése előtt a zajszűrést általában csak stúdiókban alkalmaztak, mivel elég körülményes volt megfelelő eredményeket elérni, hiszen csak szűrőket alkalmaztak a zaj csökkentésére. Ebben az időben elképzelhetetlen volt a bonyolult algoritmusokkal való zajszűrés, mint napjainkban. Ma már egy megfelelően gyors jelfeldolgozó processzor alkalmazásával bonyolultabb algoritmusok is megoldhatók. Ezért az ilyen irányú próbálkozások egyre több és jobb eredménnyel kecsegtetnek. Mivel a jelfeldolgozó processzorok ára nem túl magas, ezért ma már egyre több helyen találkozhatunk alkalmazásaikkal.

Gyakorlatban sok módszert alkalmaznak a zaj csökkentésére, ezen módszerek hatékonysága, számításigénye és jósága különböző. Munkám során hard- és soft-elbow algoritmusokkal és azok változataival foglalkoztam. Azért ezeket az algoritmusokat alkalmaztam a diplomatervemben, mert ezek az algoritmusok nem túl bonyolultak, tehát a számításigényük sem „túl” nagy. Ezen algoritmusok megfelelő nagyságú zajszűrést képesek megvalósítani, azonban, mint minden zajszűrő algoritmus, ezek sem képesek tökéletesen visszaállítani az eredeti jelet. Ezen algoritmusok részletes ismertetése és összehasonlítása a 4. fejezetben található.

A soft- és hard-elbow algoritmusok a frekvenciatartományban végzik a szűrést, ezért a jelet fel kell bontani frekvencia sávokra. A jel sávokra bontásának egyik módja szűrőbank alkalmazása. A szűrőbankok között speciális helyet foglal el az ún. rezonátoros megfigyelő struktúra, amely paramétereinek megfelelő beállításával rekurzív Fourier-transzformációt (RDFT) valósít meg. A rezonátoros megfigyelő struktúráról és a rekurzív Fourier-transzformációról a 3. fejezetben lesz szó. Másik módszer a jel frekvencia sávokra való bontására az *Fast Fourier Transform* (FFT). Mindkét módszernek megvan a maga előnye és hátránya.

Első lépésként a rezonátoros megfigyelő-elmélettel ismerkedtem meg, és **ezt** valósítottam meg Matlabban, majd a zajszűrő algoritmusokkal ismerkedtem meg jobban és vizsgáltam meg őket zajszűrés szempontjából Matlabban. A Matlabban való tesztelések alapját egy versrészlet adta, amihez zajt kevertem. Az eredmények vizsgálata meghallgatással történt. Miután Matlabbal megfelelő eredményeket értem el, és kiválasztottam a jelfeldolgozó processzoron megvalósítandó algoritmust, elkezdtem a processzorral ismerkedni. Első lépésként a már megírt integrátoros struktúrát megvalósító programmal ismerkedtem, majd ebbe beleágyaztam a hard- és soft-elbow algoritmusokat, és ezeket teszteltem. A tesztelés során meg kellett oldani az esetleges problémákat, és fel kellett deríteni a hibák okait.

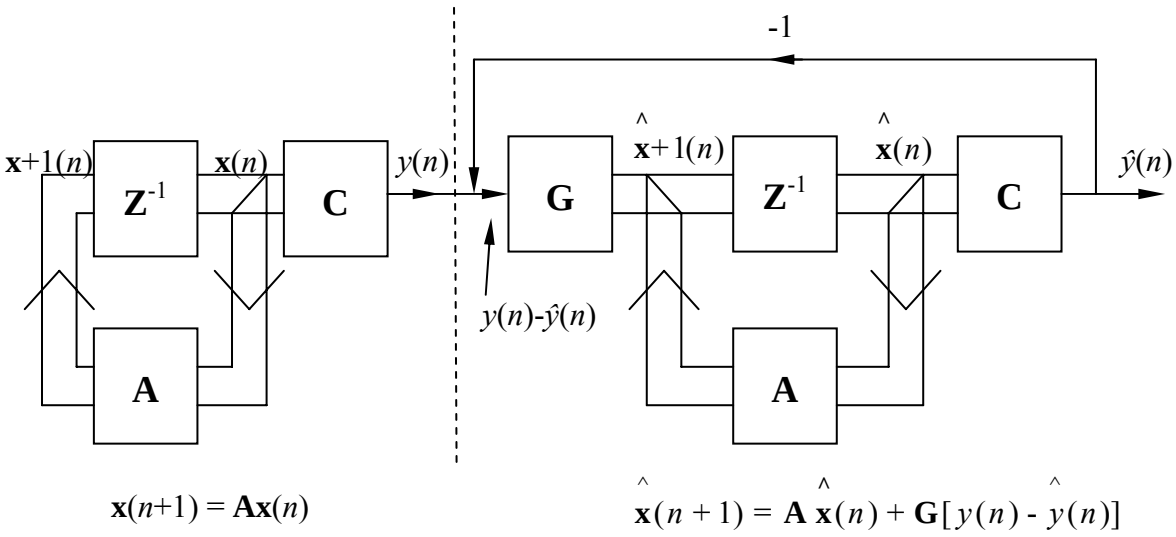
A teszteléseket nehezítette, hogy sem a zaj jellegét, sem a beszédjel amplitúdóját nem ismerjük előre, és becsülni sem tudjuk. A zaj jellege nagyon változatos lehet, ezért a legcélszerűbb a zajt mint fehérzajt feltételezni. A jel/zaj viszony is nagyon eltérő lehet egyik pillanatról a másikra. Tehát az algoritmusnak egy nagyon széles sávban kell megfelelően működnie. Tetszőleges zajra is jól kell működnie, és lehetőleg minél nagyobb beszédjel dinamikát engedjen meg. Tovább nehezíti a feladatot, hogy egy általános szűrési mértéket kell megadni, ami jól működik „szinte” minden beszédjelre.

2 Rezonátoros megfigyelő struktúra

Mivel a vizsgálandó zajszűrő algoritmusok a jelet a frekvenciatartományban változtatják meg, ezért gondoskodni kell a jel spektrális felbontásáról is. Ez megoldható FFT-vel vagy rezonátoros megfigyelő struktúrával is. Mivel a feladatkiírásban szűrőbank alkalmazásával kell megoldani a feladatott és a RDFT ennek egy speciális esete, ezért ezzel történik a jel spektrális felbontása. Ebben a fejezetben az RDFT kerül ismertetésre.

A megfigyelők olyan rendszerek, amelyek egy másik rendszer állapotváltozóinak vagy azokból származtatott mennyiségek maghatározására szolgálnak, ezáltal tulajdonképpen mérési eljárást valósítanak meg.

2.1 Megfigyelő elmélet alapjai



2.1. ábra Lineáris, mérendő- és megfigyelő rendszer

A 2.1-es ábrán látható egy mérendő és egy megfigyelő rendszer. A szaggatott vonaltól balra az autonóm rendszer modellje látható, jobbra pedig ennek a rendszernek a kópiája. (**A**: állapotátmenet mátrix, **C**: kicsatoló mátrix, **G**: becsatoló mátrix)

Az autonóm rendszer állapotváltozós leírása:

$$\mathbf{x}(n+1) = \mathbf{A}\mathbf{x}(n) \tag{2.1}$$

$$y(n) = \mathbf{C}\mathbf{x}(n) \tag{2.2}$$

Megfigyelő rendszert az autonóm rendszerből úgy kaphatunk, ha kiegészítjük egy bemenettel. Erre a bemenetre az autonóm rendszer kimenetének és a megfigyelő kimenetének a különbsége kerül, vagyis a megfigyelő a **G** becsatoló mátrixon keresztül a hibajelet kapja meg.

Így a megfigyelő rendszer állapotváltozós leírása:

$$\hat{\mathbf{x}}(n+1) = \mathbf{A}\hat{\mathbf{x}}(n) + \mathbf{G}[y(n) - \hat{y}(n)] \tag{2.3}$$

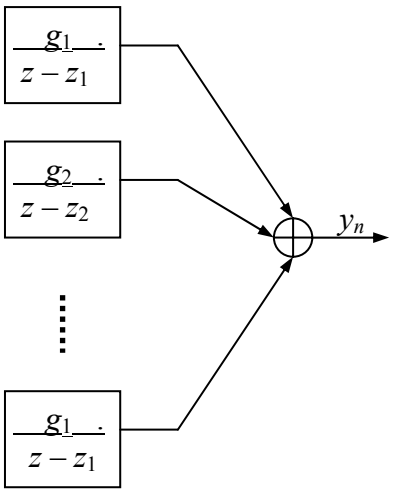
$$\hat{y}(n) = \mathbf{C}\hat{\mathbf{x}}(n) \tag{2.4}$$

Mivel az a cél, hogy az autonóm és a megfigyelő rendszer állapotváltozói megegyezzenek, ezért az állapotváltozók különbségére felírt hibarendszert lehet megadni tervezési feltételnek.

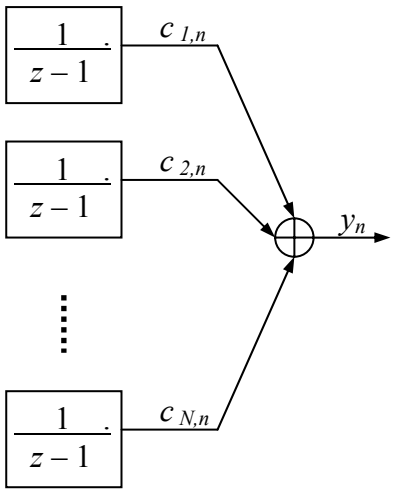
Hibarendszer:

$$\begin{aligned} &\mathbf{x}(n+1) - \hat{\mathbf{x}}(n+1) = (\mathbf{A} - \mathbf{GC}) * (\mathbf{x}(n) - \hat{\mathbf{x}}(n)) = \\ &= (\mathbf{A} - \mathbf{GC})^{n+1} * (\mathbf{x}(0) - \hat{\mathbf{x}}(0)) \end{aligned} \tag{2.5}$$

Az állapotváltozók akkor adnak helyes eredményt az N. lépés után, ha az $(\mathbf{A}\text{-}\mathbf{GC})^N=0$ teljesül, azaz az **A-GC** mátrix nemderogatórius nilpotens mátrix (ami annyit jelent, hogy az N. hatványa nulla mátrixot ad.). A **G** mátrix helyes megválasztásával tetszőleges beállási sebességű megfigyelő tervezhető, ha a mérendő rendszer teljesen megfigyelhető [1].

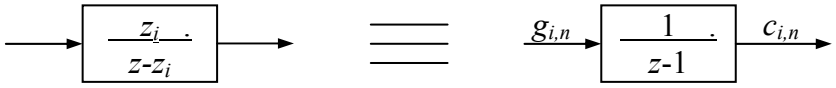


2.2 ábra A koncepcionális jelmodell rezonátorokkal

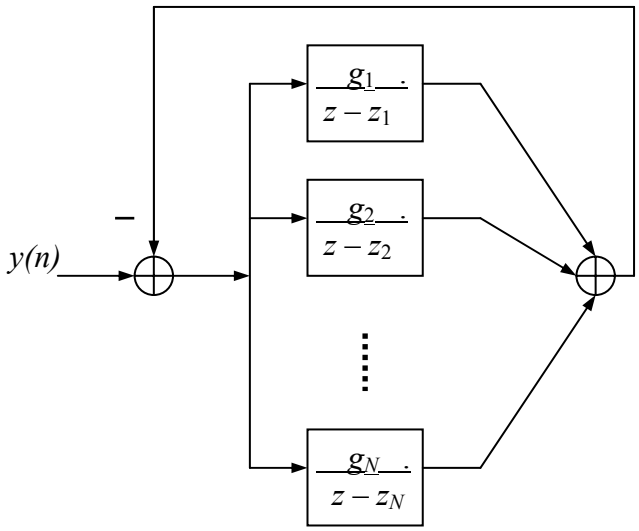


2.3 ábra A koncepcionális jelmodell integrátorokkal

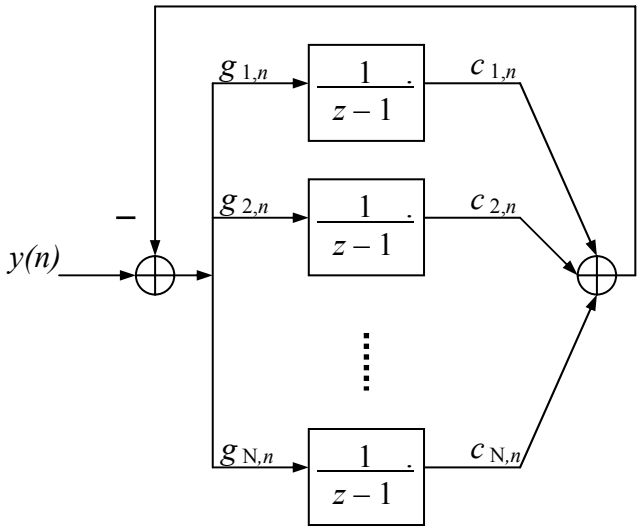
Munkám során a rezonátoros és integrátoros jelmodellhez tartozó megfigyelőkkel foglalkoztam, ezeket valósítottam meg Matlabban. A két megfigyelő esetében a kiindulási alap (amihez lényegében a megfigyelőt kell tervezni) a koncepcionális jelmodell más. Első esetben a jelet a modell a rezonátorok kimeneti jeleinek összegeként állítja elő (2.2. ábra). Második esetben a jelet a gerjesztetlen integrátorok kimenőjeleinek modulálása utáni összegzésével kapjuk meg, amely szorzótényezők a Fourier-sorfejtés bázisfüggvényei (2.3. ábra). A két jelmodell egy-egy csatornája ekvivalens, egymásnak úgy feleltethetőek meg, hogy a 2.3. ábrán szereplő integrátorok bemeneteit a megfelelő $c_{m,n}$ érték konjugáltjával beszorozzuk és egyidejűleg a 2.2. ábrán lévő rezonátorok számlálójában a megfelelő z_i-kel szorzunk. Ezt szemlélteti a 2.4. ábra, ahol $g_{i,n}$ nem más, mint $c_{i,n}$ konjugáltja.



2.4. ábra A rezonátoros és integrátoros jelmodell megfeleltetése



2.5. ábra A rezonátoros jelmodellhez tartozó megfigyelő



2.6. ábra A integrátoros jelmodellhez tartozó megfigyelő

A 2.5. ábrán a rezonátoros, míg a 2.6. ábrán pedig az integrátoros jelmodellhez tartozó megfigyelő felépítése látható.

Megfigyelők állapotegyenletei:

A rezonátoros struktúra:

$$\begin{aligned}\hat{\mathbf{x}}(n+1) &= \langle z_i \rangle \hat{\mathbf{x}}(n) + \mathbf{g}\{y_n - [1,1,...,1]\hat{\mathbf{x}}(n)\} \\ \hat{y}(n) &= [1,1,...,1]\hat{\mathbf{x}}\end{aligned} \tag{2.6}$$

Az integrátoros struktúra:

$$\begin{aligned}\hat{\mathbf{x}}(n+1) &= \hat{\mathbf{x}}(n) + \mathbf{g}_n\{y(n) - \mathbf{c}_n^T \hat{\mathbf{x}}(n)\} \\ \hat{y}(n) &= \mathbf{c}_n^T \hat{\mathbf{x}}(n)\end{aligned} \tag{2.7}$$

ahol $\mathbf{c}_n=[c_{i,n}]$ és $\mathbf{g}_n=[g_{i,n}]$

A véges beállás feltétele, ha a rezonátorok pólusai az egységkörön helyezkednek el, a rezonátoros struktúra esetén:

$$g_i = \frac{1}{N} z_i \tag{2.8}$$

A véges beállás feltétele, ha a rezonátorok pólusai az egységkörön helyezkednek el az integrátoros struktúra esetén:

$$g_{i,n} = \frac{1}{N}\overline{c}_{i,n} \tag{2.9}$$

A rezonátoros struktúra átviteli függvénye a következőképpen adódik:
Egy rezonátor átviteli függvénye:

$$H_i(z)=\frac{r_i z_i}{z-z_i} \tag{2.10}$$

A zárt hurok átviteli függvénye abban az esetben, ha:

$$\begin{aligned} z_i &= \sqrt[N]{1} \\ r_i &= 1/N \end{aligned} \tag{2.11}$$

akkor

$$P(z)=z^{-N} \tag{2.12}$$

lesz, tehát a struktúrának egy meghatározott késleltetése van, és nem változtatja meg a jel alakját.
A hibajelre vonatkozó átviteli függvény 1-$P(z)$ alakú lesz.

2.2 Rekurzív Fourier-transzformáció

Abban az esetben, ha az integrátoros megfigyelőben a 2.9-es egyenletben megadott feltétel teljesül, akkor rekurzív Fourier-transzformációt valósít meg. Az állapotváltozók ebben az esetben a Fourier-transzformált értékeket adják meg.

Ilyenkor a 2.7. egyenletben szereplő $c_{i,n}$-k és $g_{i,n}$-ek időben változnak, míg a rezonátoros struktúrában (2.6. egyenletben) szereplő z_i és **g**-k időinvariánsak.

A $c_{i,n}$-ket következőképpen lehet meghatározni (ami lényegében a $g_{i,n}$-ket is meghatározza a 2.9-es egyenlet alapján) [2]:

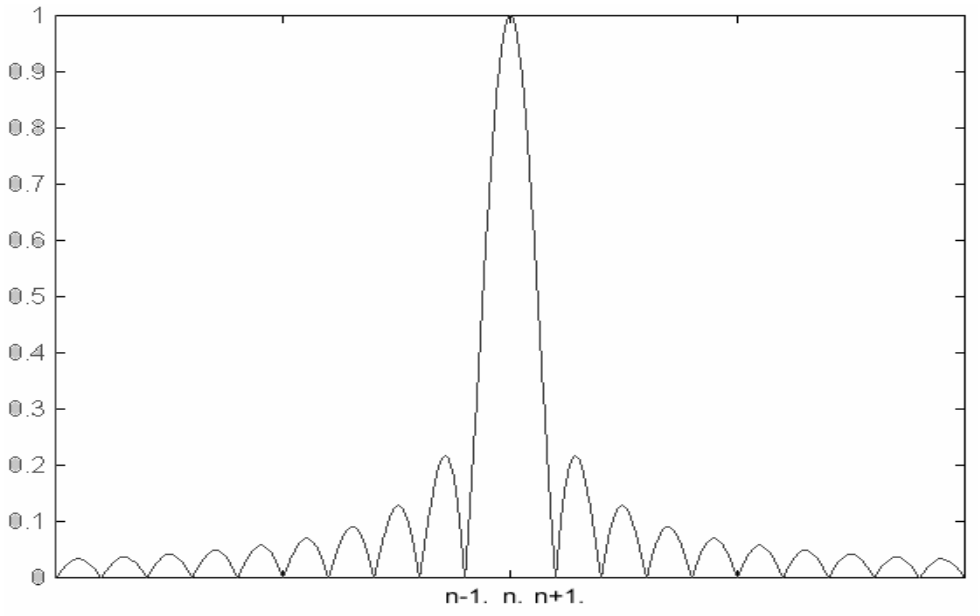
$$c_{k,n} = e^{j\frac{2\pi}{N}kn}; \quad k = -L...L; \quad N = 2L+1; \tag{2.13}$$

Ez esetben páratlan számú rezonátor van (egy DC, N darab komplex konjugált pár AC rezonátor).

DFT esetén az egy csatornára vonatkozó átviteli függvény a következőképpen alakul:

$$\left|U_i(f)\right| = \left|\frac{\sin \pi N(f - f_i)}{N \sin \pi (f - f_i)}\right| \tag{2.14}$$

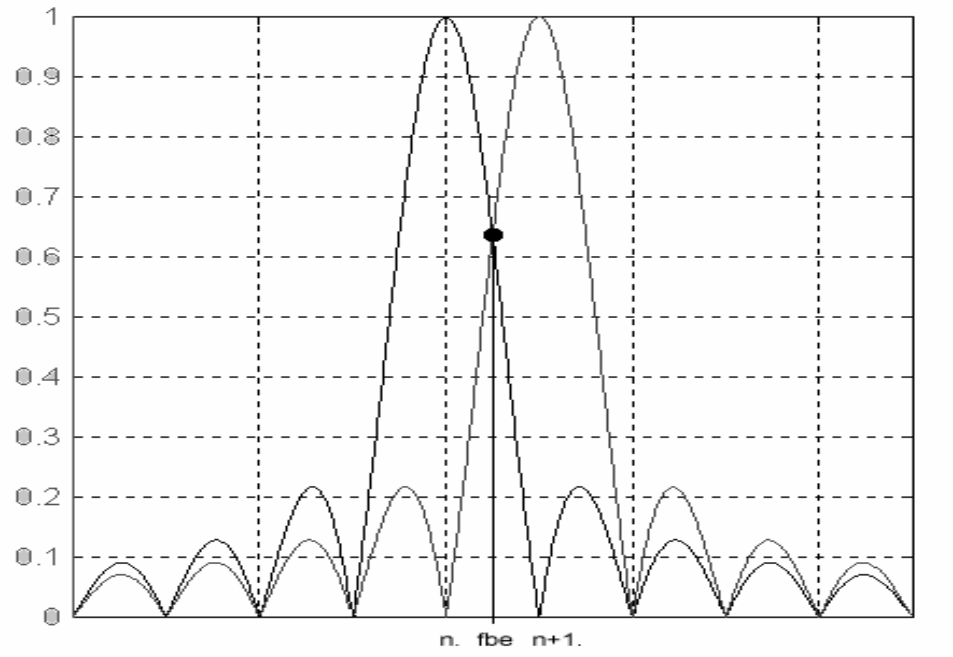
ahol f_i: a rezonátor frekvenciát jelöli, N a rezonátorszámot.



2.7. ábra Egy rezonátor csatorna átvitele

A 2.7 ábrán egy rezonátor átvitele látható (2.14 képlet alapján). Ha a bemenő jel frekvenciája megegyezik az *n.* rezonátor frekvenciájával, akkor ezen a rezonátoron egységnyi az átvitel, a többi rezonátor frekvenciákon pedig nulla. Ebből látható, hogy ilyenkor csak egy rezonátor fog részt venni a kimeneti jel előállításában, a többi rezonátor értéke nulla lesz. Abban az esetben, ha a bemenő frekvencia nem rezonátor

frekvenciára esik, akkor nem csak egy rezonátor vesz részt a kimeneti jel előállításában, hanem a többi is (megfelelő szorzótényezőkkel, lényegében spektrumszivárgás lép fel).



2.8.ábra Két rezonátor által felvett értékek bemutatása rezonátorközi frekvencia esetén

Ezt mutatja a 2.8. ábra (amin két szomszédos rezonátor átvitele van felrajzolva), ahol a legrosszabb eset lett feltüntetve, a bemenő frekvencia két rezonátorfrekvencia közé esik pont (a két rezonátorfrekvencia átlaga). Az ábráról látható, hogy ilyenkor a bemenőjel frekvenciájához legközelebb lévő két rezonátor azonos súllyal szerepel, de nem egységnyi átvitellel (*picket-fence* jelenség), és a többi rezonátornak is vannak értékei. Ponttal van jelölve a két szomszédos rezonátor által felvett érték (amik megegyeznek). A távolabbi rezonátorok is vesznek fel értékeket, mégpedig ezen értékek az oldalhullámok maximumai lesznek, csak mindegyik rezonátornak másik oldalhulláma kerül a bemeneti frekvencia értékére. Akkor, ha nem pont két rezonátor frekvencia közé esik a bemeneti frekvencia, mint az előbbi esetben, hanem az egyikhez közelebbi frekvenciájú, akkor az a rezonátor fog a legnagyobb súllyal szerepelni, amelyikhez a legközelebb van a bemeneti jel frekvenciája. A többi rezonátor az előbbi esetnél kisebb súllyal szerepel a kimeneti érték kiszámolásában. Minél közelebb van a bemenő jel frekvenciája a rezonátorfrekvenciához, annál kisebb súllyal szerepel a többi rezonátor a kimeneti jel előállításában (a jelenséget spektrumszivárgásnak hívják, a spektrumszivárgás mellett amplitúdó csökkenés is fellép, amit *picket-fence*-nek hívnak).

Ezen jelenség kiküszöbölésére ablakfüggvény alkalmazása lehet célravezető. A legtöbbször szívesen alkalmazott ablakfüggvény a *Hanning-ablak* és *Hamming-ablak*. Ezeknél az ablakoknál azonban a főhullám szélessége kétszer akkora, mint a *sinc (sinx/x)* esetében, emiatt rezonátorpozícióba eső bemeneti jelek esetén is egy helyett három rezonátornak lesz értéke.

3 Zajszűrő algoritmusok ismertetése

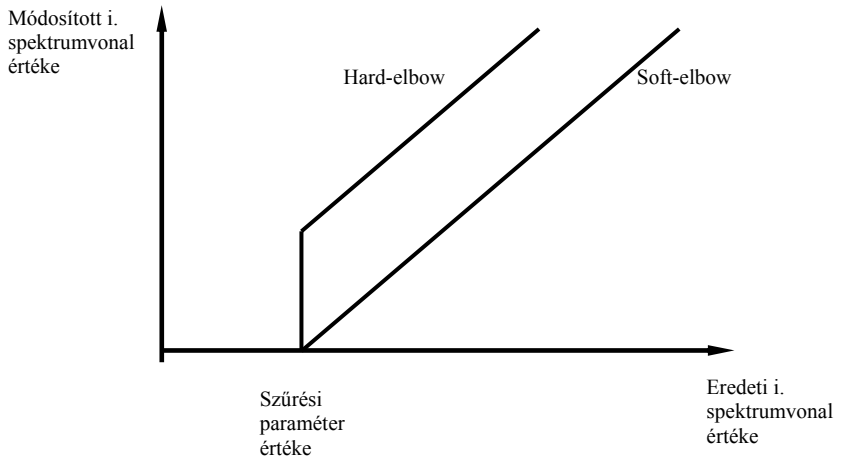
Az előző fejezetben a jel frekvenciasávokra való felbontásához alkalmazott rekurzív Fourier-transzformációról volt szó. Erre azért volt szükség, mivel a most bemutatásra kerülő zajszűrő algoritmusok a szűrést a frekvenciatartományban végzik, tehát a szűrendő jel spektrumát változtatják meg. Ebben a fejezetben az algoritmusok részletes ismertetése és az azok összehasonlításából származó eredmények bemutatása történik.

Megvizsgált algoritmusok:

- hard-elbow
- soft-elbow
- hiszterézises
- módosított hard-elbow
- módosított soft-elbow

Ezek az algoritmusok nemlineáris algoritmusok, mint a későbbiekben kiderül.

3.1 Algoritmusok leírása



3.1. ábra A soft- és hard-elbow karakterisztika [3]

A 3.1. ábrán látható a soft- és hard-elbow algoritmusok karakterisztikája.

3.1.1 Soft-elbow algoritmus

Az algoritmus a következő képlettel írható le:

$$\hat{X}(f_i) = \begin{cases} |X(f_i)| - K & ha\ |X(f_i)| > K \\ 0\ ha\ |X(f_i)| < K \end{cases} \tag{3.1}$$

ahol $\hat{X}(f_i)$ a szűrt jel i. frekvencia komponensének értéke, $X(f_i)$ az eredeti jel i. frekvencia komponensének értéke, *K* pedig a szűrési paraméter.

Az algoritmus szerint, ha *K* nagyobb, mint a jel spektrumának adott frekvenciájú komponense, akkor a szűrt jel értékét 0-vá teszi az algoritmus, hiszen ekkor a *K* paraméterrel megadott feltételezett zaj úgyis elnyomja a jelet az adott frekvencia komponensen. Ha fordított a helyzet, akkor a szűrt jel egyenlő lesz az eredeti jel adott frekvenciájú komponensének abszolút értékéből kivont *K* paraméter különbségével (3.1. ábra).

Egyik problémája az algoritmusnak az, hogy ha a zajos jelben a zaj által keltett frekvenciakomponens nagyobb, mint a *K* paraméter, akkor visszamarad a zajkomponens egy része. Ilyenkor a szűrt jelben a zajkomponens jóval kisebb, mint az eredeti jelben, de megváltozik a zaj jellege. Az addig egyenletes sziszegés helyett bugyborékoló, „kuruttyoló" hang jelenik meg. Ez halkabb, mint az alapzaj, ugyanakkor zavaróbb lehet, mint az eredeti [3].

3.1.2 Hard-elbow algoritmus

Az algoritmus matematikai leírása:

$$\hat{X}(f_i) = \begin{cases} 0\ ha\ |X(f_i)| < K \\ X(f_i)\ ha\ |X(f_i)| > K \end{cases} \tag{3.2}$$

A képlet jelölései megegyeznek a 3.1 képlet jelöléseivel.

Az előbbi algoritmushoz képest azonosság az, hogy ha *K* nagyobb, mint a jel spektrumának adott frekvenciájú komponense, akkor a becsült jel értékét 0-vá teszi az

algoritmus, hiszen ekkor a *K* paraméterrel megadott feltételezett zaj úgyis elnyomja a jelet az adott frekvencia komponensen. Ha fordított a helyzet, akkor a becsült jel egyenlő lesz az eredeti jellel (3.1. ábra). Ebben az esetben csak a *K* paraméternél kisebb komponensekkel foglalkozunk. Tehát az algoritmus csak a szűrési paraméternél nagyobb frekvenciájú komponenseket hagyja meg a jelből.

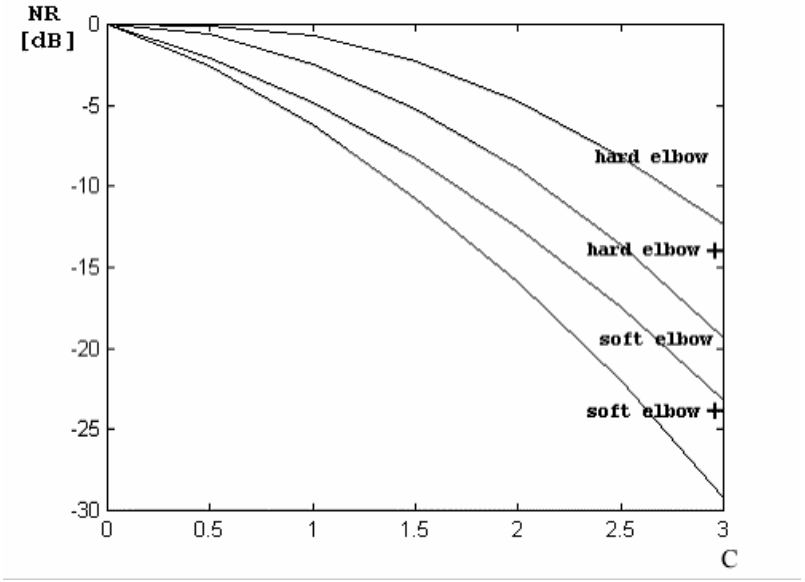
3.1.3 Hiszterézises soft- és hard-elbow algoritmus

Ezeknél az algoritmusoknál az alap algoritmus a soft- és hard-elbow algoritmus. Azoktól annyiban különbözik, hogyha egy rezonátor értéke a *K* küszöbszint alatti, akkor azt kinullázzuk, mint az eredeti algoritmusokban. Ezt a rezonátort azonban csak akkor fogjuk újra belevenni a kimeneti érték számolásba, ha egy *K*-nál nagyobb szintet elér a rezonátor értéke. Majd, ha újra *K* alatti értéket vesz fel, akkor kinullázzuk. Külön a soft- és a hard-elbow algoritmusra is meg lett valósítva és tesztelve.

3.1.4 Módosított soft- és hard-elbow algoritmus

Ezeknél az algoritmusoknál is az eredeti soft- és hard-elbow algoritmusokból indultunk ki. Azonban, ha a *K* paraméternél kisebb a rezonátor értéke, akkor nem nullázzuk a rezonátor értékét, hanem egy egynél kisebb számmal szorozzuk meg. A szorzó paraméter (maradékzaj paraméter) tipikusan 0.1..0.3, esetleg lehet több is, de ekkor a zajcsökkenés kisebb mértékű, de ennek vannak előnyei, amit a következő fejezetben mutatok be.

3.2 Algoritmusok összehasonlítása



3.2. ábra Algoritmusok összehasonlítása [4]

A 3.2. ábra a soft- és hard-elbow és azok módosított változataik zajszűrésének mértékét mutatják a „szűrési” paraméter függvényében. Az ábrán látható C paraméter az az érték, amellyel az előre becsült zajspektrum meg van szorozva. Itt a zajszűréshez nem egy küszöbszint (szűrési paraméter) van felhasználva, hanem egy a felvétel zajából előre becsült zajspektrum. Ebben az esetben zenei felvételek zajszűrése volt megoldva, és minden zenei felvétel elején van egy hanganyagot nem tartalmazó rész, ahol a zajbecslést el lehet végezni a Welch-módszerrel. Ez a zajbecslő van felhasználva a szűréshez, mégpedig úgy, hogy a zajbecslő adott frekvenciakomponensének értéke lesz az adott frekvenciára a szűrési paraméter. A C értékkel lényegében a szűrés nagyságát lehet állítani.

Az NR átlagos zajteljesítmény arány érték a következő képlettel lett meghatározva és értéke decibelben értendő:

$$NR = 10 \cdot \log \left(\frac{\sum_{i=1}^{N} y_i^2}{\sum_{i=1}^{N} x_i^2} \right) \tag{3,3}$$

ahol y_i a szűrt jel i. mintája, x_i pedig az eredeti jel egy mintája [4].

A „+” jellel ellátott karakterisztikák a módosított algoritmusokhoz tartoznak. Az ábráról látszik, hogy a módosított algoritmusok mindkét esetben jobb eredményeket adnak. A hard- és soft-elbow algoritmusok közül, pedig a soft-elbow adja a jobb eredményt, tehát a teszteléseim során azt az eredményt kell kapnom, hogy a legjobb algoritmus a soft-elbow.

Matlab szimulációval hasonlítottam össze a vizsgált algoritmusokat. A vizsgálat tárgyát egy versrészlet adta, amihez fehérzajt kevertem. Az eredmények összehasonlítása meghallgatással történt, így az algoritmusok megítélése elég szubjektív dolog, bár csak abban az esetben, ha kis eltéréseket mutatnak az algoritmusok.

Kezdetben csak a soft- és hard-elbow algoritmust vizsgáltam. A szimulációk során a soft-elbow algoritmus eredményezett jobb zajszűrést.

Probléma volt mindegyik algoritmussal, hogy, ha a szűrési paraméter kisebb volt, mint a fehérzaj spektrumának nagysága, azaz a szűrés után maradt a jelen maradvány zaj, akkor ugyan a zaj csökkent (a zajcsökkenés nagysága a szűrési paraméter függvényében változott), de a maradvány zaj egy kellemetlen “bugyborékolást” eredményezett. Ha a szűrési paraméter úgy lett beállítva, hogy nagyobb volt, mint a fehérzaj spektrumának nagysága, akkor a zaj megszűnt a vizsgált mintáról, de a beszéd torzított lett. Főleg kis jel szinteknél jelentős torzítás lépett fel, sokszor a beszéd sem volt érthető.

Következő feladat e jelenség csökkentése volt. Ezért lett megvizsgálva a hiszterézises és a módosított soft-, hard-elbow algoritmus.

A hiszterézises megoldás eredményei nem tértek el az előző két algoritmuséitól, így ez nem jelentett jelentős előrelépést, amellett a hiszterézises algoritmus tesztelése bonyolultabb is, hiszen rendelkezik egy plusz paraméterrel.

A módosított algoritmusok adták a legjobb eredményeket, megfelelő szorzó paraméter esetén, ugyan maradt a szűrt jelen zaj, de torzítás nem volt észlelhető. Ennek nem az az oka, hogy nincs is torzítás, csak a megmaradt zaj elfedi azt, és így nem halljuk.

3.1. Táblázat Algoritmusok összehasonlítása (65 rezonátor esetén)

Paraméter / Algoritmus	Hard-elbow		Soft-elbow	Módosított SE	
Zajszint	0,05	0,08	0.05	0,05	0,08
K	-	-	-	0,1	0,2
"Zaj megszűnése"	0,003	0,0055	0.003	0,003	0,004
Bugyborékolás kezdete	0,002	0,003	0.001	0,002	0,002
Torzítás kezdete	0,004	0,0055	0.003	0,007	0,008
Bugyborékolás nincs	0,006	0,008	0.005	0,004	0,005

A 3.1. táblázatban található értékek a következők szerint értendőek. Mivel wave file-lal dolgoztam, ezért azt Matlabban be kellett tölteni egy változóba, ez a *waveread* függvénnyel tehető meg, a betöltéskor a betöltött változó értékei 1 és -1 között vehet fel értékeket. A *rand* függvénnyel előállított zaj értéke 0 és 1 között vehet fel értékeket. Ezt a nullára szimmetrikussá teszem és beszorzom a zajszint paraméterrel, ami a táblázat szerint vagy 0,05, vagy 0,08. A *K* paraméter esetében az az érték van megadva, amellyel az állapotváltozó meg van szorozva, ha kisebb, mint a szűrési paraméter. A többi paraméter esetében az érték a szűrési paramétert adja meg, ami a 3.1. ábra szerinti töréspontot adja meg, tehát hard-elbow algoritmus esetében, ha az állapotváltozó értéke ennél nagyobb, akkor nem változtatja az értékét, ha kisebb, akkor kinullázza azt. A megadott értékek azt a szűrési paramétert adják meg, amelyeknél az adott jelenség fellépett (pl.: bugyborékolás nincs, …).

Az 3.1. táblázatból látható, hogy ugyanolyan zajszint mellett a "zaj megszűnése" ugyanakkor következik be, és a bugyborékolás is ugyanakkora szűrési paraméternél lép fel, tehát az algoritmusok e paraméterek alapján ugyanúgy viselkednek. A módosított algoritmusoknál azonban a torzítás lényegesen később lép fel és a bugyborékolás is előbb szűnik meg. Ennélfogva a legjobb algoritmusnak a módosított soft-elbow algoritmus látszik.

4 Matlab szimuláció

Szimulációra azért volt szükség, mert a Matlab programot egyszerűbb írni, mint a jelfeldolgozó processzort programozni, hiszen C szintaktikával rendelkezik és szimulációval az algoritmusok tulajdonságai megállapíthatóak.

Munkám során az elméleti bevezető után a rezonátoros és integrátoros struktúra megvalósítása következett Matlab környezetben. Majd az integrátoros struktúrába a zajszűrési algoritmusok beültetése és azok tesztelése következett. A tesztelések során el kellett dönteni, hogy melyik algoritmus a legmegfelelőbb a feladat elvégzésére és melyik algoritmus implementálható jelfeldolgozó processzorra.

A szimulációt Matlab 5.3-as verziójával végeztem.

A Matlab program editorával szerkesztettem a megírt M-filet, ami lényegében egy függvény volt. A függvény írása során kerülni kellett a ciklusokat, feltételes utasításokat, mert ezek lassították a függvény lefutását. Mivel beszédjelet akartam feldolgozni (aminek mintavételi frekvenciája 11kHz volt) és ha tesztelni akartam a szűrési eredményeket, akkor n*10 másodperces mintaregisztrátumot érdemes feldolgozni. Ez sok adat feldolgozását jelentette és - mivel az algoritmus N^2-es lépésszámú - ezért törekedi kellett a minél gyorsabb függvény megírására.

4.1 Teszteléshez használt minta

A vizsgálandó minta egy versrészlet (Ady Endre: Góg és Magóg fia vagyok... kezdetű verse) volt, ennek egy perces része állt rendelkezésre. A felvétel meg volt szűrve egy 3 kHz-es aluláteresztő szűrővel és 44.1 kHz-es mintavételi frekvenciával rendelkezett. Matlab segítségével megvizsgáltam, hogy vannak-e 3 kHz feletti komponensei, ezek mértéke elhanyagolható volt. Ezután minden negyedik mintát megtartva a felvételből wave formátumba írtam a disk-re, így előállítottam egy 11,025 kHz mintavételi frekvenciájú beszédjelet. A mintavételi frekvencia csökkentése több szempontból indokolt: kevesebb mintát tartalmaz (ugyanannyi idő alatt több másodpercnyi beszédjel dolgozható fel). A jelfeldolgozó processzorra már meg volt írva egy integrátoros struktúrájú rutin, ami 8 kHz mintavételi frekvencián dolgozott és az

emberi beszéd sávszélessége sem haladja meg jelentősen a 4 kHz-t. A zaj szimulálása matlab *rand* függvényével előállított zajjal történt, amit a beszédjelhez adtam hozzá.

4.2 A struktúrák megvalósítása, tesztelés módszerei

Először a rezonátoros struktúrát valósítottam meg, majd mikor ez sikerült, az integrátoros struktúra került sorra. Miután a két kiindulási alap struktúra jól működött Matlab alatt, azután az integrátoros struktúrába beépítettem a zajszűrő algoritmusokat és azokat teszteltem. Az alapstruktúrák tesztelése a hibajel vizsgálatával történt, mivel mindkét algoritmus N lépésben áll be. Ez azt jelenti, hogy a kimenet a bemeneti jelet N lépés után állítja vissza helyesen, csak egy késleltetéssel tér el a kettő egymástól. Mivel ablakfüggvény nem volt használva, vagy pontosabban megfogalmazza négyszögablak volt alkalmazva, ezért egy rezonátor átvitele *sinc* alakú (2.6 ábra). Ha a bemeneti jel nem rezonátorfrekvenciába esik, akkor a hibajel nem lesz nulla, hanem valamekkora szinuszos jel, szinuszos bemeneti jel esetén. A legrosszabb esetben, ha a két rezonátor frekvencia átlaga a bemenő jel frekvenciája (amit a 2.7 ábra is ábrázol), akkor a hibajel kétszeres amplitúdójú szinusz, csak a két jel között 180 fok a fáziseltérés. Az eredeti jel visszaállítása tetszőleges bemeneti frekvenciákra helyesen működik, csak fázistolás lesz a bemeneti és a kimeneti jel között.

4.3 A megfelelő rezonátorszám kiválasztása

A vizsgálat tárgyát képezte még az, hogy hány darab rezonátor kell a megfelelő működéshez. Vizsgált rezonátorszámok: 65, 129, 257. A meghallgatásos vizsgálatok azt az eredményt adták, hogy 65 rezonátor esetén is megfelelően működik az algoritmus. Az eredményeket a 4.1-es táblázat mutatja különböző rezonátorszámok esetén és hard-elbow algoritmust felhasználva.

4.1. Táblázat Rezonátorszámok összehasonlítása

Jellemzők / Rezonátorszám	65	129	257
Zajszint	0,08	0.08	0,08
Zaj megszűnése	0,004	0.004	0.003
Bugyborékolás kezdete	0,003	0.003	0.002
Torzítás kezdete	0.00575	0.0035	0.004
Bugyborékolás nincs	0.008	0.00575	0.0045

Nagyobb rezonátorszámok esetén előbb elkezd torzítani az algoritmus (ennek oka lehet, hogy nagyobb rezonátorszám esetén a felbontás nagyobb lesz, így egy spektrumsáv többre esik szét, és ennek következtében a több spektrumsáv értéke kisebb lehet az eredeti sáv értékénél. Ennélfogva kisebb értéket vehet fel, ami azonban már kisebb lehet a szűrési paraméter értékénél.), de a "bugyborékolás" előbb szűnik meg. A többi paraméterben nagyjából megegyeznek. Ebből az következik, hogy érdemesebb minél nagyobb rezonátorszámot használni. Azonban a rezonátorszám növelésének van egy hátrányos tulajdonsága, ami a Matlabban nem, de a jelfeldolgozó processzoron észlelhető. Mind a 2.6-os és 2.7-es képletben szereplő g illetve g_n paraméterben szerepel az N-el való osztás (N = rezonátorszám). Ez 257 rezonátor esetén 8 bit veszteséget jelent, míg 65 esetén csak 6 bit veszteséget. A jelfeldolgozó processzoron az eredeti program 512 rezonátort tartalmazott, itt már 9 bit a veszteség, a jelfeldolgozó processzor viszont 16 bites. 65 rezonátor esetén még nem, de 256, 512 esetén már jelentős mértékű (hallható) lett a kvantálási zaj. E jelenség miatt a 65 rezonátoros megoldást választottam és a jelfeldolgozó processzoron már csak ennyi rezonátor lett megvalósítva, míg Matlabban is az algoritmusok összehasonlítása ezzel a rezonátorszámmal történt.

4.4 A Matlab programok tesztelése

Miután az integrátoros alapstruktúra és a különböző zajszűrési algoritmusok is helyesen működtek, megkezdődhetett az algoritmusok összehasonlítása. Az összehasonlítás eredményeit a 3.3-as pontban található 3.1 táblázat tartalmazza. Az algoritmusok összehasonlítása meghallgatással történt, ami azt jelenti, hogy egy szűrt minta megítélése szubjektív dolog. Ebből következik az is, hogy minden ember számára

más lehet a zavaró, jelen esetben a “bugyborékolás” fellépte vagy valamekkora zaj visszamaradása (módosított algoritmusok esetében) vagy esetleg kisebb mértékű torzítás nem zavaró. Mivel beszédfeldolgozásról van szó, ez megengedhető, hiszen kis torzítások mellet a beszéd még érthető maradhat. Elsősorban nem a tökéletesen tiszta hang megőrzése a cél, hanem a minél nagyobb mértékű zajszűrés, de feltétel az, hogy a beszéd érthető maradjon. Az eredmények kiértékelésében az is szerepet játszhat, hogy minden ember hallása más, és milyen körülmények között hallgatja meg a felvételt. Ezért a 3.1. és 4.1. táblázatban található paraméterek értékeinek meghatározása szubjektív dolog. A táblázatban megadott értékek az én megítélésem szerintiek.

Az algoritmusok különböző jel/zaj viszonyokra lettek tesztelve. A jel/zaj viszony megadása elég nehéz ebben az esetben, hiszen pillanatról pillanatra változhat, attól függően, hogy mekkora a jel nagysága, mivel egy „állandó” zajt keverünk hozzá.

Első lépésként a hard-elbow algoritmus lett megvizsgálva. Alapvető baja az algoritmusoknak és a soft-elbow-nak is, hogy zajszint alatti szűrési paraméter esetén a zaj csökken, de nem szűnik meg és “bugyborékolás” lép fel. Ha zajszint feletti paramétert alkalmazunk, akkor a zaj és a “bugyborékolás” is megszűnik, de torzított lesz a beszéd. Másik probléma, hogy a torzítás előbb fellép, mint hogy a zaj és a “bugyborékolás” teljesen megszűnne, és így nem lehet egy határt megadni, ahova be lehetne állítani a szűrési paramétert (ez a paraméter erősen függ a zaj nagyságától, ezért a paramétert legjobb lenne a jelből magából meghatározni), hogy még ne torzítson, de zaj se legyen már. Ezeknél az algoritmusoknál egy optimumot lehet megadni, ahol még nem torzít igazán, de zaj is nagymértékben lecsökkent.

A hiszterézises algoritmus nem adott jobb eredményeket, mint a soft- és hard-elbow algoritmusok.

Az előbb említett problémák pedig a torzítás megjelenése és a „bugyborékolás” jelenléte. Ezt kiküszöbölni nem lehet, csak azt lehet tenni, mint a módosított algoritmusokban. A szűrési paraméternél kisebb értékeket nem kinullázzuk, hanem beszorozzuk egy 1-nél kisebb számmal (tipikusan 0.1..0.3). Ebben az esetben a torzítás szintén fellép, csak a megmaradt zaj helyes paraméter-beállítás esetén elfedi a torzítást. Ez a megoldás megfelelőnek bizonyult. El lehetett érni olyan eredményeket, hogy a zaj csak kis mértékben maradt a jelen, de bugyborékolás és torzítás nem volt hallható.

Az algoritmusok nagyon érzékenyek a kis szintű jelekre, abban az értelemben, hogy a torzítás ezeknél lép fel a legelőször, mivel ezek a jelszintek közel lehetnek és vannak a szűrési paraméterhez.

4.5 A programok ismertetése

Az elkövetkezőkben a megvalósított Matlab programokról lesz szó. Mind az integrátoros, mind rezonátoros alapstruktúra meg lett valósítva. A két program csak az alapstruktúra miatti különbségekben tér el.

Az algoritmusokban következő lépések hajtódnak végre (Időrendi sorrendben, az inicializációs résztől eltekintve, ahol a bufferek kezdőértékeinek beállítása történik):

Az integrátoros struktúra esetén:

1. Kimeneti érték kiszámolása
2. Állapotváltozók abszolút értékeinek és szögeinek előállítása
3. A szűrési paraméternél kisebb *(ix)* és nagyobb *(ix2)* abszolút értékű állapotváltozók indexeinek meghatározása.
4. *ix* és *ix2* segítségével a módosított soft-elbow algoritmus végrehajtása
5. Hibajelek előállítása
6. *g*[y(m)-c*x.']* különbségi jel előállítsa
7. DC rezonátor kiszámolása a következő ciklushoz
8. AC rezonátorok kiszámolása a következő ciklushoz
9. Állapotváltozók átállítása a következő ciklusra
10. Forgatóvektorok kiszámolása a következő ciklushoz

A rezonátoros struktúra esetén:

Megegyezik az előzővel, azzal a különbséggel, hogy ebben az esetben a 10. lépés nincsen, mert a forgatóvektorok a rezonátoros struktúrában időinvariánsak.

Minden tömb végén van egy plusz elem, ami azért kell, mert a 3. lépés szerinti indexek meghatározásánál a program kezdetén (első ciklusban, amikor még minden állapotváltozó nulla értékű) az *ix2* vektor üres lenne, és üres vektorral megcímezve egy

tömböt nem lehet szorozni. Ezért az állapotváltozók plusz elemének értéke egy, hiszen ilyenkor mindig lesz értéke az *ix2* vektornak, a forgatóvektorok plusz elemeinek értéke nulla. Ezért a kimeneti eredményben, ahol a két vektor szorozva van egymással nem játszik szerepet. Az *ix* vektor akkor lehet üres vektor, ha minden állapotváltozó nagyobb a szűrési paraméternél. Ez nem fordult elő a tesztelések során, ezért ezzel a problémával nem foglalkoztam. Azonban ezt egy újabb plusz elem bevezetésével meg lehet oldani, csak itt az állapotváltozókban lévő plusz elem, értékének nullának kell lennie, a forgatóvektorok új plusz eleme akármilyen értéket felvehet.

Az algoritmus az összes állapotváltozót kiszámolja, azonban mivel az állapotváltozók páronként egymásnak komplex konjugáltjai, ezért elég lenne csak a felét kiszámolni, és ezzel is lehetne gyorsítani a programot.

4.6 Fellépett hibák okainak keresése és azok kiküszöbölése

Eddig ismertetett programok hibái:

- a rezonátorközi frekvenciákra nem állították alakhelyesen vissza a jelet
- nagyon csillapították a jel nagyságát

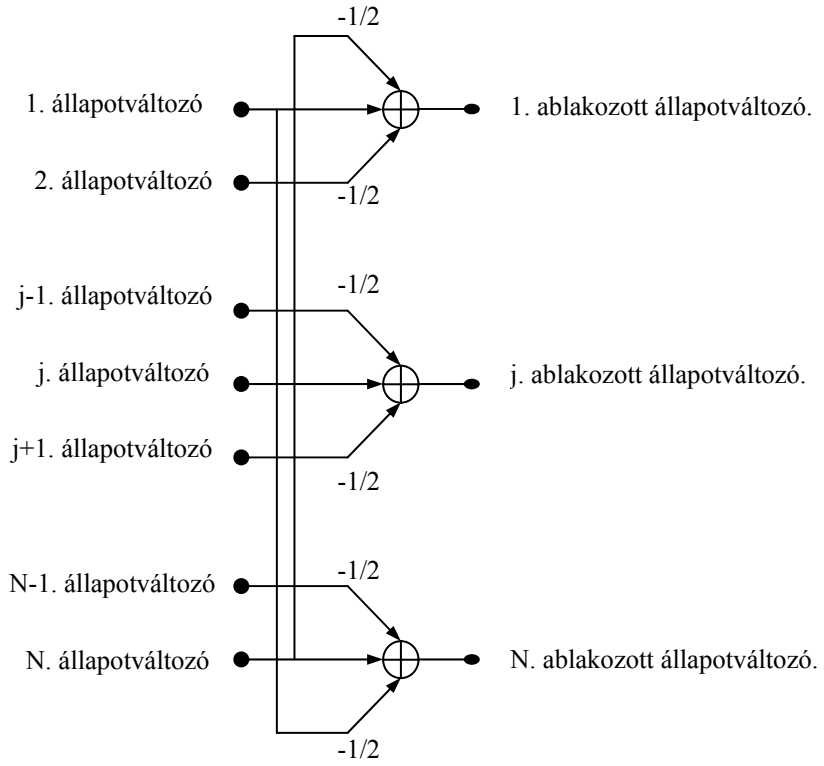
Ebben az alfejezetben ezen hibák vizsgálatával foglalkozom és kiküszöbölésére tett megoldást fogom ismertetni.

4.6.1 Jelalak hiba kiküszöbölése

A jelalak hiba kiküszöbölésére a programban az állapotváltozókon egy ablakozást végzünk el, és ezzel az ablakozott állapotváltozókkal van megcímezve a szűrési táblázat (ami az 5.2.1. alfejezetben lesz ismertetve). A kimeneti jel visszaállítása az eredeti állapotváltozókon elvégzett szűrés utáni értékek összegzésével történt.

Ezzel a módszerrel a jelalak hiba kiküszöbölhető. A programban *Hann* ablak lett alkalmazva, mivel ez az ablakfüggvény nagymértékben csökkenti a spektrumszivárgást

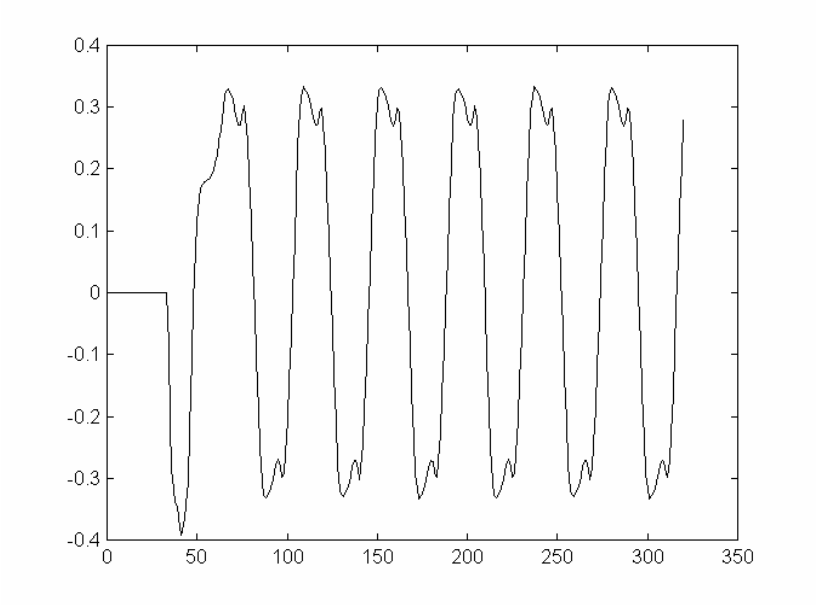
és a *picket-fence* jelenséget is. Mivel az algoritmus rekurzív, ezért az ablakozást a frekvenciatartományban kell elvégezni. Ez egy cirkuláris konvolúcióval megoldható.



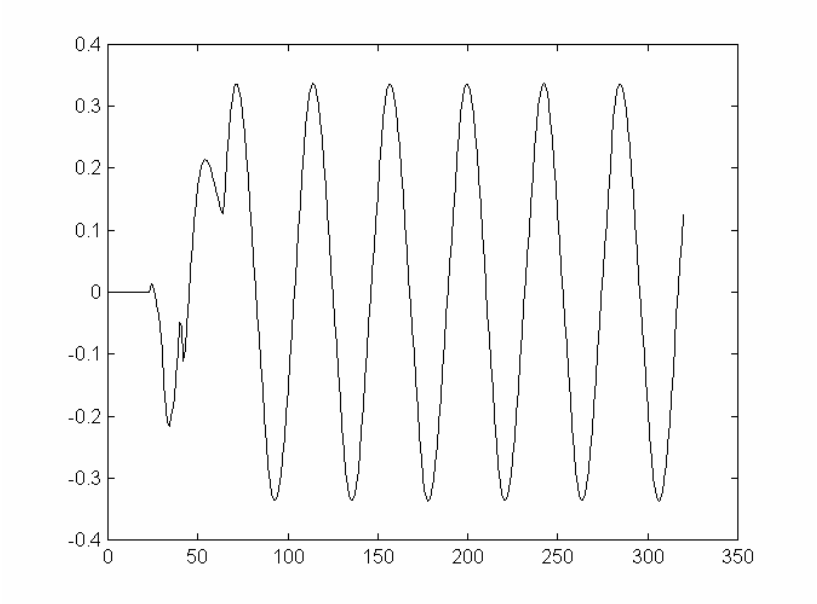
4.2. ábra Frekvenciatartománybeli ablakozás szemléltetése Hann-ablakra

Az 4.2. ábrán látható az ablakozás megoldási sémája. A *j.* ablakozott állapotváltozó az eredeti állapotváltozókból úgy áll elő, hogy a j. állapotváltozó egyszeres és a *j+1.* és *j-1.* állapotváltozók -½-es súllyal szerepelnek az ablakozott állapotváltozóban. Az *N.* állapotváltozónál az *N+1.* állapotváltozó az *1.* és az *1.* állapotváltozónál az *1-1.* állapotváltozó az *N.* (Ez jelenti a cirkularitást.), mint a mellékelt ábra is mutatja.

Az 4.3. ábrán egy szűrt jel látható ablakozás nélkül. Az eredeti jel egy szinusz jel, melynek bemeneti frekvenciája pont két szomszédos rezonátor frekvenciájának átlaga és amplitúdója egységnyi. Az 4.3. ábrán látható, hogy a szűrt jel alakja torz, és amplitúdója is 0.35 körüli. Ez a nemrég említett 3 dB-nél több (hozzávetőleg 9 dB), tehát nagymértékű amplitúdó hiba is fellép.



4.3. ábra Ablakozás nélküli szűrt jel

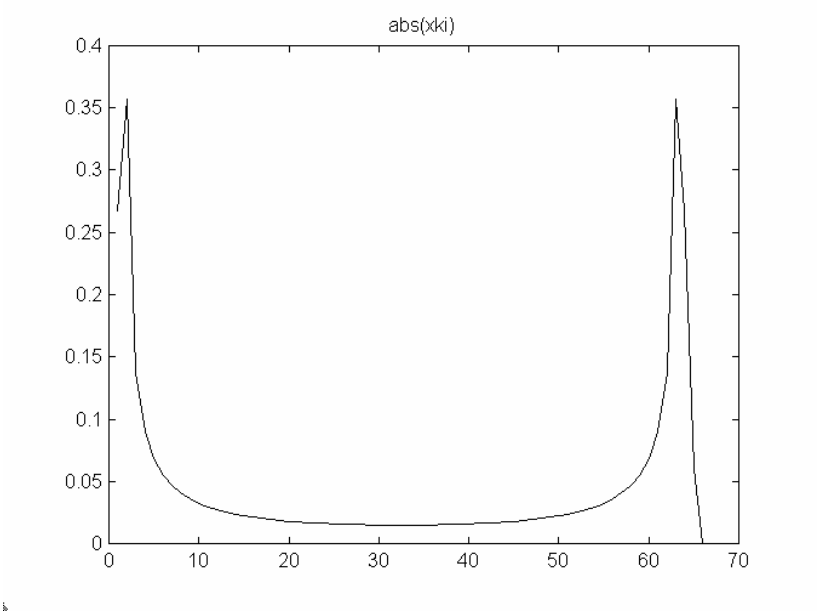


4.4. ábra Ablakozott szűrt kimeneti jel

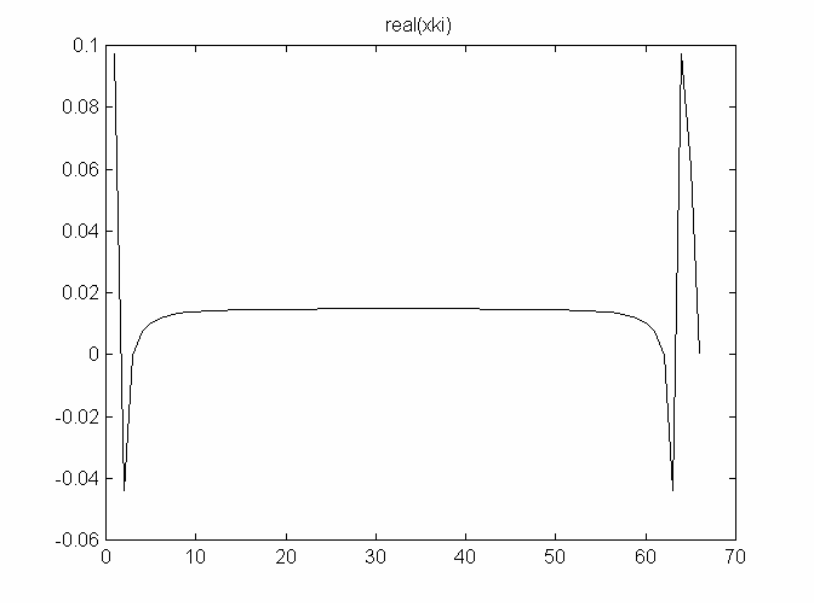
Az 4.4. ábrán látható jelalak az előbbi jel szintén szűrt változata, azonban ebben az esetben ablakfüggvény *(Hann)* is van alkalmazva. Látható, hogy ebben az esetben a jelalak helyesen áll elő, de az amplitúdó hiba jelentős. Ennek oka (jelalak hiba megszűnése) pedig az, hogy a *Hann* ablakfüggvény alkalmazása nagymértékben csökkenti a spektrumszivárgást, ezáltal közel csak a szinusz spektrumvonala marad meg a spektrumból. Mindkét esetben 65 rezonátor volt alkalmazva és ezért az első 65 érték (vízszintes skála 1-65-ig) a beálláskor fellépő jelalakot mutatja.

4.6.2 Az amplitúdó hiba vizsgálata

A jelnagyság csökkenés okának megismeréséhez meg kell vizsgálni az állapotváltozók valós részét, mivel minden időpillanatban a kimeneti jel az állapotváltozók valós részének összegéből áll elő (a rezonátoros struktúra esetében). Az 4.5 ábrán az állapotváltozó abszolút értéke látható egy pillanatban (Ebben a pillanatban a bemeneti jel a maximumát veszi fel.). A bemeneti jel egy szinusz jel egységnyi amplitúdóval, amelynek frekvenciája két szomszédos rezonátor átlaga. Az 4.5. ábráról látható, hogy jelentős spektrumszivárgás lép fel. Ennek valós része az 4.6. ábrán látható. Az ábrákon látszik, hogy az állapotváltozók nem nullára csökkennek le, hanem egy bizonyos szintre, ebben az esetben 0.015 környékére. Az állapotváltozók értékeinek csökkenése *sinc* "alakúnak" kell lennie, mivel nincsen ablakfüggvény használva (négyszögablak). Emiatt azt lehetne várni, hogy a távoli rezonátorok értékei közel nulla értéket vegyenek fel. Ebben az esetben a távoli rezonátorok értékei nem nulla értéket vesznek fel, hanem a bemeneti jel pillanatértékének függvényében valamilyen értéket. Minél nagyobb a bemeneti jel pillanatértéke, annál nagyobbat.



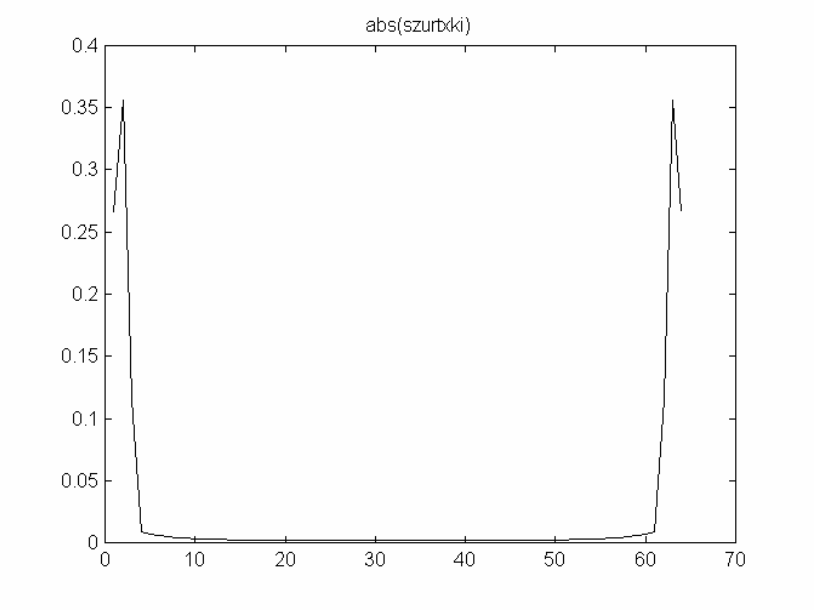
4.5. ábra Állapotváltozó abszolút értéke



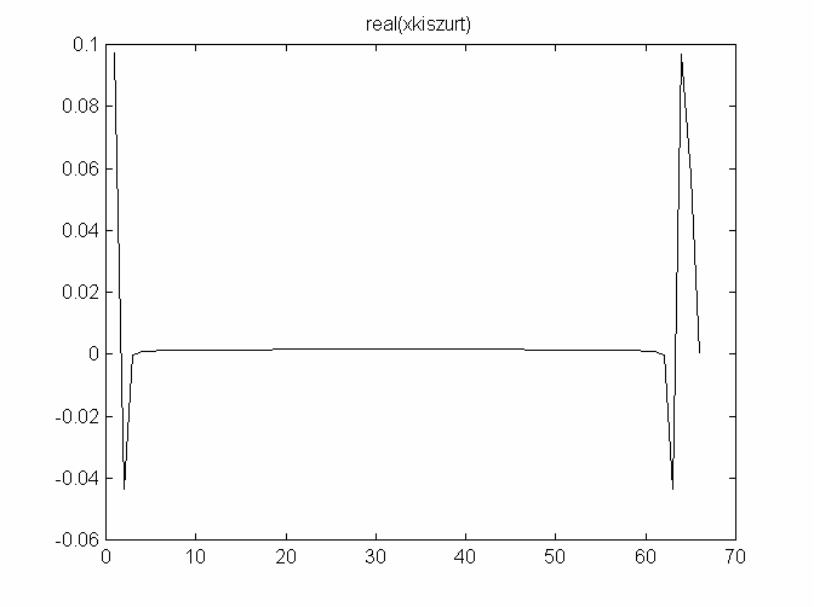
4.6. ábra Állapotváltozó valós része

Az állapotváltozó valós része is így viselkedik (Ez a 4.6. ábrán látható.).

A szűrés elvégzése utáni állapotváltozók abszolút értéke a 4.7. és ennek a valós része a 4.8. ábrán látható. Az 4.6. és 4.8. ábra közötti különbség lényegében annyi, ami a szűrésből is adódik, hogy a szűrési paraméter alatti rezonátorok értékeit beszorozza 0.1-del, mivel módosított soft-elbow algoritmussal van szűrve a jel. Ha összegezzük azt a kb. 50 darab állapotváltozót, ami 0.015 körüli értéket vesz fel, akkor ennek értéke 0,75, tehát ezek az állapotváltozók a kimeneti jel előállításában nagy súllyal szerepelnek. A 4.8. ábrán látható szűrt állapotváltozók valós részének szűrési paraméter alatti állapotváltozók értékei hozzávetőleg 0.015*0.1=0.0015, ezt az értéket itt is kb. 50 állapotváltozó veszi fel, így ezek értéke 0,075 lesz. Ennélfogva a szűrés miatt a kimeneti jel nagymértékű amplitúdó csökkenést fog szenvedni. Rezonátorpozícióba eső jelek esetén ez a jelenség nem lép fel, hiszen ott nincs spektrumszivárgás.



4.7. ábra Szűrt állapotváltozó abszolút értéke



4.8. ábra Szűrt állapotváltozó valós része

4.6.3 Tesztelés

A tesztelés itt is a többi programban használt Ady-versrészletet felhasználva történt. A teszteléshez használt matlab program az *mfdftrezszckonv.m* program volt, amely *Hann* ablakfüggvényt használ az ablakozáshoz. Ezzel a programmal végeztem az előző fejezetben ismertetett hibák kijavítása utáni tesztelést.

A zajparaméter a tesztelések során 0.05 volt, és 65 rezonátor volt alkalmazva.

A szűrési paraméterre nem volt olyan érzékeny ez az algoritmus, mint az előbbiek, hiszen 0,0012-es szűrési paraméternél (ami azt jelenti, hogy ha az állapotváltozó értéke ennél kisebb, akkor beszorozza a maradékzaj paraméterrel, ha ennél nagyobb akkor meg kivonja belőle a szűrési paraméter értékét) is érthető volt a szöveg, nem torzított, de nagymértékű volt a jel amplitúdójának csillapítása. Igaz, hogy az algoritmus nagyobb szűrési paraméterek mellett is helyesen működik, de a nagy szűrési paraméter alkalmazásának nagy hátránya, hogy a hasznos jelet is nagymértékben csökkenti. Ennek kiküszöbölését egy a szűrési paramétertől függő kompenzáló erősítés bevezetésével lehetne esetleg megoldani. Azonban ilyenkor nemcsak a hasznos jelet, hanem a zajt is felerősítenénk valamilyen mértékben. Viszont a hasznos jelet az algoritmus kisebb mértékben csökkenti, mint a zajt, ebből következik, hogy erősítés bevezetésével nem állna vissza a kiindulási állapotnál rosszabb eset.

Az algoritmus legérzékenyebb pontja a maradékzaj paraméter volt, hiszen 0.1-nél kisebb szűrési paramétereknél nagyon torzított a jel, igaz, nagyobb szűrési paraméter értékeknél nem lépett fel torzítás. 0.25-ös paraméternél már szinte semmilyen szűrési paraméternél nem torzított a beszéd. Ennélfogva 0.25 az a maradékzaj paraméter aminél kisebbet nem érdemes alkalmazni.

5 Valós idejű megvalósítás

5.1 Hardware ismertetése

A valós idejű megvalósításhoz használt hardware egy, az *Analog Devices* cég által kifejlesztett *ADSP-2181* típusú jelfeldolgozó processzorral felszerelt DSP kártya: *ADSP-2181 EZ-KIT Lite*™ néven.

Ezen a kártyán helyezkedik el a processzor az AD, DA konverter *(AD1847)* és a kiegészítő elemek. A kártya RS232 porton keresztül kommunikál a számítógéppel. [5]

5.1.1 Az ADSP-2181 főbb jellemzői [6][7]:

- Fixpontos, 16 bites számábrázolás
- 20MHz-es órajel esetén 40 MIPS sebességű, de a kártyán 16 MHz-es órajellel van ellátva, ezért 33 MIPS a sebessége.
- 80K Byte On-Chip Ram, amely áll
    - 16K Szó (24 bites) program és
    - 16K Szó (16 bites) adat memóriából.
- Egy utasítás egy órajel-ciklus alatt hajtódik végre.
- Maximum három utasítás egyidejű végrehajtása (pl.: két regiszter összeszorzása és ez alatt a következő két szorzandó szám behozatala a memóriából, szűrések tipikus utasítása).
- Egymástól független *ALU*-val (összeadó egység), *MAC*-cal (szorzó egység) és *BARREL SHIFTER*-el rendelkezik
- Két címképző egységgel rendelkezik.
- Van két soros portja és DMA portja

5.1.2 Az AD1847 főbb jellemzői [8]:

- Szigma-delta átalakító
- Sztereo
- 16 bites
- Maximális mintavételi frekvencia: 48 kHz
- Soros interfacen keresztül kommunikál a jelfeldolgozó processzorral

A programokat egy algebrai assembly nyelven lehet fejleszteni. Külön szerkesztő programja nincs, akármilyen szövegszerkesztővel lehet szerkeszteni. A lefordítását DOS-os batch file-al lehet megoldani. A lefordított programot a jelfeldolgozó processzorra Windows-os környezetben az Ezkitapp programmal lehetett letölteni, majd futtatni. E letöltő programban vissza lehet olvasni a memóriák tartalmát, de nem lehet a regiszterek tartalmát figyelemmel kísérni vagy változtatni, mint egyes újabb környezeteknél (Visual DSP). Ez a fejlesztői környezet állt rendelkezésre.

A processzor választásának főbb okai:

- megfelelően gyors az adott feladathoz
- elegendően nagy memória kapacitással rendelkezik
- az integrátoros struktúra már meg volt valósítva ezen a processzoron
- nem elhanyagolható szempont, hogy a tanszéken rendelkezésre áll
- gyakorlati megvalósítás esetén árban kedvező

5.2 Megvalósítás

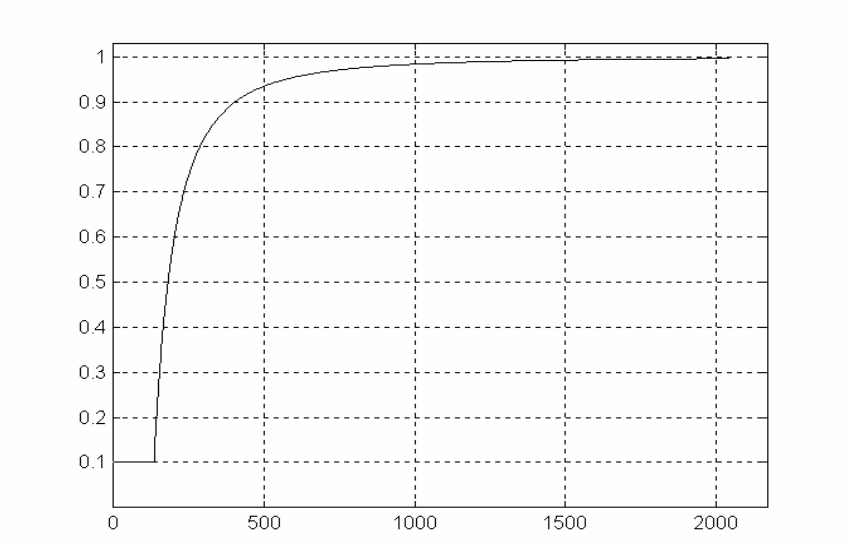
Miután az integrátoros struktúra meg volt valósítva a jelfeldolgozó processzorra, ezért ebben az esetben csak a zajszűrő algoritmust kellett beleágyazni. Ez az IMTC konferencia egyik előadásának demonstrálására megvalósított program volt [9].

A zajszűrő algoritmus nemlineáris, de a jelfeldolgozó processzor miatt külön kezelt mindkét értéket (valós és képzetes rész) egyforma mértékben csökkenti. Ez a csökkentés egy szorzótényezővel vehető figyelembe, amelynek értéke az állapotváltozók abszolút értékétől függ. Ezeket a szorzótényezőket egy táblázatban tároljuk a processzor memóriájában. Mivel az abszolút értéket számítani időigényes, ezért az abszolútérték-négyzetet számítjuk ki, és ennek megfelelően előtorzítjuk a táblázatot.

5.2.1 Szűrési táblázat

Ebben az alfejezetben a szűréshez használt táblázatot fogom ismertetni.

A táblázat azt adja meg, hogy az állapotváltozók abszolútérték-négyzete függvényében, mekkora értékkel kell beszorozni az állapotváltozókat, hogy a szűrési paraméternek megfelelő szűrés álljon elő.



5.1. Ábra A szűréshez használt táblázat

A táblázatban a következő függvény van megvalósítva:

$$táblázat(x) = \begin{cases} \dfrac{x^2 - \text{szp}}{\text{x}^2} & ha \quad x > szp \\ K \quad ha \quad x < szp \end{cases} \tag{5.1}$$

szp: szűrési paraméter

A táblázatban az 5.1-es egyenlet által meghatározott értékek kerülnek. A négyzetes karakterisztika azért kell, mivel az abszolútérték-négyzetet használjuk fel a táblázat címzéséhez. (*szp* = 128, *K*= 0.1 esetben a táblázatot a 5.1. ábra mutatja)

5.2.2 Az integrátoros struktúrájú program ismertetése

A kiindulási alap egy már előre megírt és működő integrátoros struktúrát megvalósító algoritmus volt, e programba lett beépítve a szűrési algoritmus.

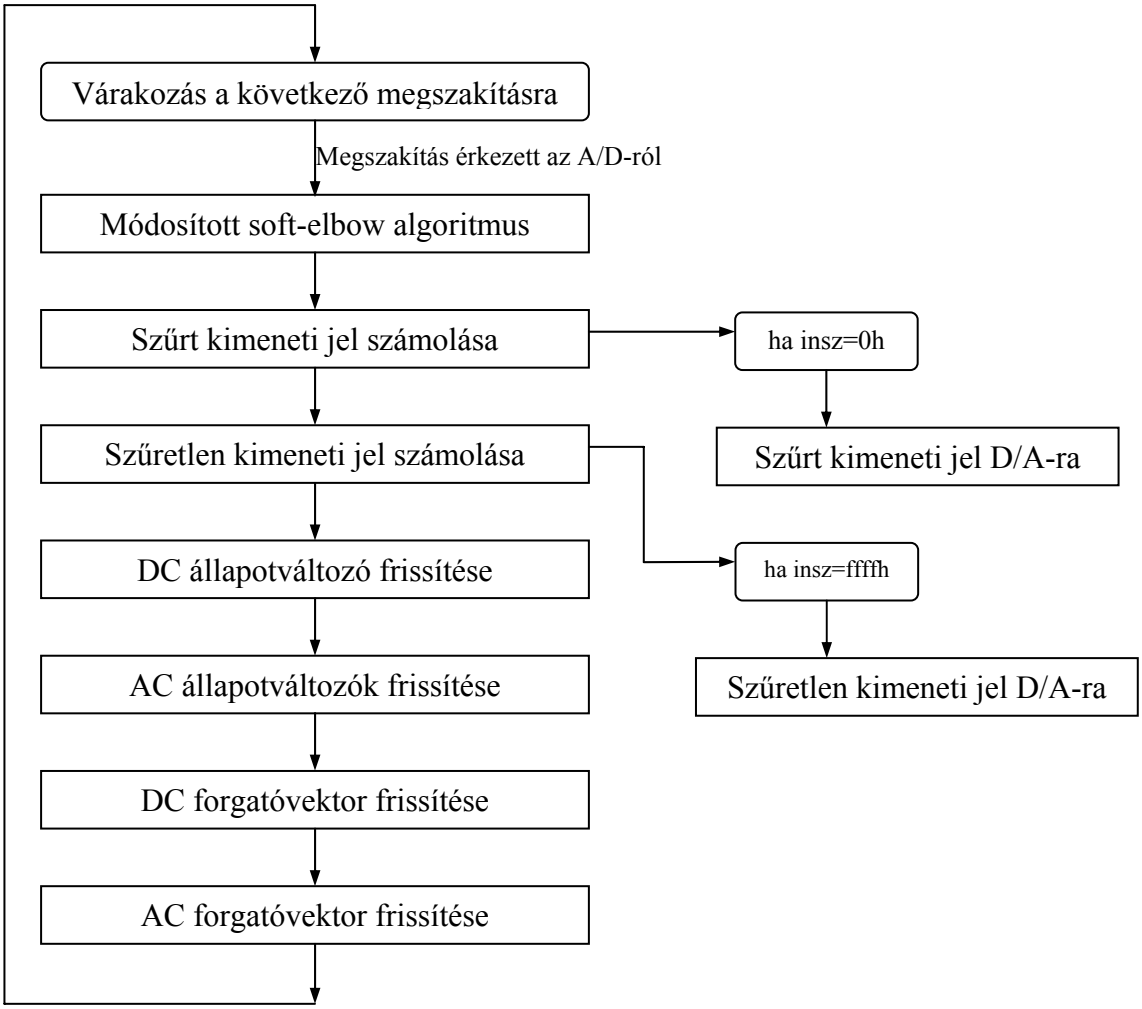
Ez a program 8kHz mintavételi frekvencia mellett 512 rezonátorral dolgozott. A rezonátorszámot a Matlab programok tesztelése során megfelelőnek talált 65-re állítottam be. Az alap program a rezonátorokat két csoportba osztja, és ezeket külön kezeli. Az egyik csoportba a DC, a másikban az AC rezonátor(ok) vannak. Az AC rezonátoroknak a fele van fizikailag megvalósítva, mivel a rezonátorok páronként egymásnak komplex konjugáltjai. Tehát a program 1 DC és 31 AC rezonátorból állítja elő a kimeneti jelet. A rezonátoroknak komplex értékeik vannak és a jelfeldolgozó processzoron a komplex szám két számként van kezelve. A programban a módosított soft-elbow szűrési algoritmus lett beleépítve.

A program első lépésben 6 file-t tölt be, amik a függelék 10.2.1-es pontjában vannak ismertetve.

Majd a konstansoknak és buffereknek történik a helyfoglalás, ezután a címregiszterek beállítása a megfelelő értékre, majd a bufferek kezdőértékeinek beállítása. A konstansok és bufferek értelmezése a mellékletben található. A *konstansok*, bufferek a 10.3.2. és 10.3.3. pontokban vannak ismertetve.

A kártyán van két gomb, az egyikkel resetelni lehet a kártyát, a másikkal egy megszakítást lehet adni, ami a program végén elhelyezkedő *time_it* részre ugrasztja a futást. Itt egy kis *subrutin* van, amely az *insz* értékét állítja vagy 0h-ra vagy ffffh-ra annak megfelelően, hogy a gomb be van-e nyomva, vagy nincs. Ez az *insz* paraméter dönti el, hogy a szűrt vagy a szűretlen jelet rakjuk ki a D/A-ra.

A program a megszakítás rutinban fut, amelyre az analóg-digitális átalakító által szolgáltatott megszakítás hatására ugrik (az *input_sample* cimkére). Az 5.2 ábrán a megszakítás rutin lépései láthatóak.



5.2. Ábra A megszakítás rutin lépései

5.2.2.1 A megvalósított szűrési algoritmus ismertetése

A szűrési algoritmus a módosított soft-elbow algoritmus, amely az 5.2.1. alfejezetben ismertetett táblázatot használja fel a szűrés elvégzéséhez.

Mivel a táblázat a programban 256 elemű, ez pedig 8 biten ábrázolható, ezért az abszolútérték-négyzet felső 8 bitjére van szükség. Ezért a számot meg kell szorozni 256-tal, hogy a felső 8 bitet külön tudjam kezelni. Az így kapott számhoz hozzáadok 1000h-át (mivel a táblázat kezdőcíme 1000h) és így előáll az a cím, amivel a táblázatból ki tudom venni a megfelelő értéket. Ezután ezzel az értékkel elvégzem a szűrést. A szűrés egy ciklusban foglal helyet, ami *C*-szer (31-szer) fut le. A ciklus után a DC állapotváltozó szűrése következik ehhez hasonló módon.

5.2.2.2 Szűrt és szűretlen kimeneti jel számítása.

Azért van szükség mindkettő kiszámolására, mert a rezonátorok frissítéséhez kell a szűretlen kimeneti jel értéke, a hibajel meghatározásához. Így nem lehet csak egy ciklust használni a kimeneti jel előállításához, és annak megfelelően, hogy szűrt vagy a szűretlen jelet rakjuk ki a kimenetre, átállítani a címregisztereket a másik bufferekre.

A szűrt kimeneti jel előállítását az *outsz1* és *outsz2*, míg szűretlen kimeneti jel előállítását az *out1* és *out2* ciklusok végzik. Az 1. számú ciklusok a *re_x* illetve *reszurt_x* buffereket szorozzák skalárisan a *re_r* vektorral. Az eredmény az *mr* regiszterben (MAC kimeneti regisztere) marad. A 2. számú ciklusok az *im_x* illetve *imszurt_x* vektorokat szorozzák az *im_r* vektorral és minden lépésben kivonják a szorzat eredményét az *mr* regiszterből. Majd az így kapott számhoz hozzáadja a *dc,* illetve *dcszurt* értékét. Azt, hogy melyik jel kerüljön ki a kimenetre, azt a kimeneti jel kiszámolása utáni *insz* változó értékének vizsgálatával dönti el.

5.2.2.3 DC és AC állapotváltozók frissítése

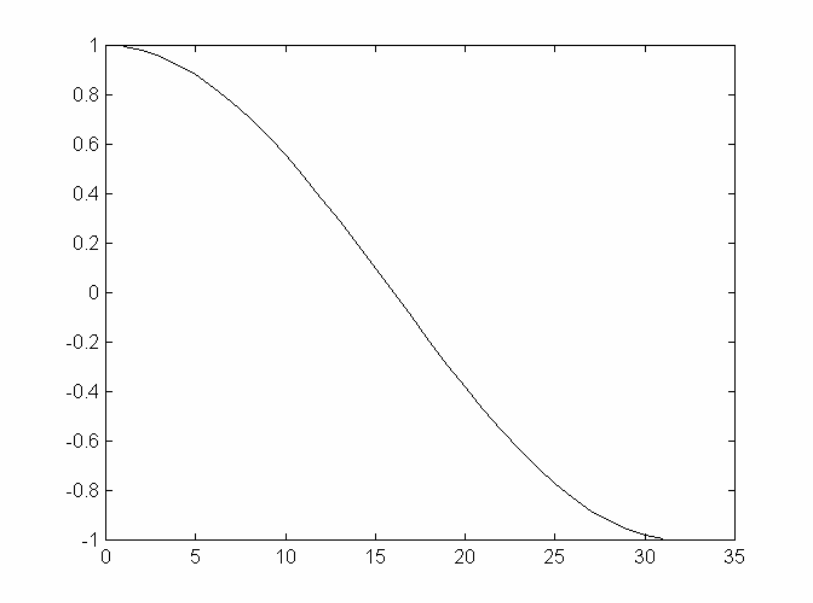
Az állapotváltozók frissítése a 3.7. egyenlet alapján történik a *res* nevű ciklusban. A ciklus előtt ki van számolva a hibajel a és az $\frac{1}{\mathrm{N}}\{y(n) - \mathbf{c}_n^T \hat{\mathbf{x}}(n)\}$ érték, ami

az *my0* regiszterbe kerül, és az *my1* regiszterbe ennek a -1-szerese kerül. Erre azért van szükség, mivel az $\frac{1}{N}\{y(n) - \mathbf{c}_n^T \hat{\mathbf{x}}(n)\}$ kifejezést $c_{i,n}$-nek a konjugáltjával kell megszorozni. Ez a képzetes részeknél egy -1-gyel való szorzásban tér el a valós részek számolásától, és a rezonátorok képzetes részeinek számolásánál az *my1* regiszter van felhasználva.

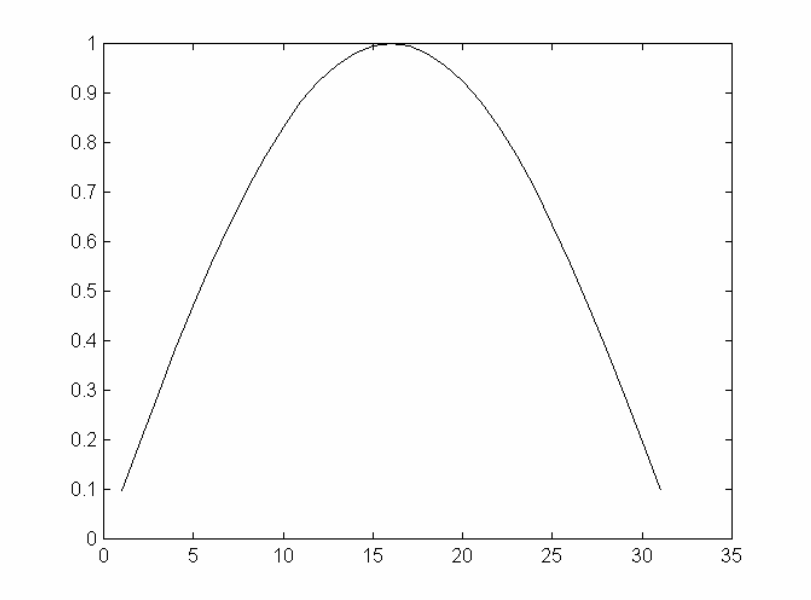
5.2.3 A rezonátoros struktúrájú program ismertetése

Ez a program a 2. fejezetben ismertetett rezonátoros struktúrát valósítja meg, a 2.7. egyenletnek megfelelően. Ennek a struktúrának a megvalósítása sokkal egyszerűbb, mint a 5.2.2. alfejezetben ismertetett integrátoros algoritmus. Ennek oka az, hogy a c_i-k időinvariánsak, azaz időtől függetlenek. Így elmarad a forgatóvektorok frissítése programrészlet, a kimeneti jelet lényegesen egyszerűbb kiszámolni (állapotváltozók összegzésével), viszont az állapotváltozók frissítése kicsit bonyolultabb, hiszen komplex szorzást kell alkalmazni.

A programok az *input_samples* részt kivéve szinte teljesen ugyanazok. Egy kis eltérés abban van a közös részekben, hogy milyen buffereket használ a program. Mivel itt a c_i-k állandók, ezért nem kell azokat frissíteni. Ezáltal nem kell a frissítéshez használt szinusz tábla sem, hanem az állandó c_i-ket kell berakni a memóriába. A c_i-nek van képzetes és valós része, a valós részt a *re_ft* buffer tartalmazza a 0h címtől, a képzetes részt az *im_ft* tartalmazza a 20h címtől.



5.2. Ábra Re_ft tartalma



5.3. Ábra Im_ft

A 5.2. ábra a c_i-k valós részét a 5.3 ábra a képzetes részt ábrázolja. Mindegyik táblázat 31 elemű, mivel ennyi darab AC rezonátor van, és a DC rezonátor ebben a programban is külön van kezelve. A konstansok és állandók megegyeznek a két programban, kivéve

azok, amelyek a frissítéshez vannak használva, mert azok nincsenek ebben a programban. A szűrés is megegyezik a két programban.

5.2.3.1 A kimeneti jelek előállítása

Ebben a struktúrában a 2.6 egyenletnek megfelelően a kimeneti jel előállítása az állapotváltozók összegzésével oldható meg. Ez azt jelenti, hogy elég az állapotváltozók valós részeit összegezni, mivel az állapotváltozók komplex konjugált párokat alkotnak, és ha a képzetes részt összegezzük csak, akkor nullát kapunk eredményül. Mindkét kimeneti jel (szűrt és szűretlen) is ki van számolva a 5.2.2.2. alfejezetben ismertetett okok miatt.

A szűrt jelet az *outcalszurt* ciklus, a szűretlen jelet az *outcal* ciklus állítja elő. Azt, hogy melyik kimeneti jel kerül a D/A-ra, az a másik programban ismertetett módon történik.

5.2.3.2 Az állapotváltozók frissítése

Az állapotváltozók frissítése ebben az esetben a 2.6. egyenletnek megfelelően történik. Ez bonyolultabb, mint a másik esetben, hisz itt a z_i-k komplex számok, és ezért komplex szorzást kell alkalmazni.

6 A jelfeldolgozó processzorra írt programok tesztelése, vizsgálata

A program megírása után megvizsgáltam, hogy a ráadott szinusz jelet visszaállítja-e helyesen, és a hibajel megfelelő-e a különböző frekvenciájú bemeneti jelek esetén. Mikor meggyőződtem, hogy helyesen működik a program, akkor elkezdtem a beszédjel zajszűrését tesztelni. A program a végleges formáját az ebben a fejezetben bemutatott módosítások során érte el.

6.1 Beszédjel szűrésének tesztelése integrátoros struktúrával

A teszteléshez a 4.1-es alfejezetben ismertetett mintát használtam fel, a számítógép hangkártyájáról a jelfeldolgozó processzor bemenetére adtam a jelet és a kimenetet egy aktív hangsugárzóra kötöttem. Mivel a felvétel nagy részben elég kis jelszinteket tartalmazott, de voltak benne „nagy” jelszintek is, ezért az algoritmust egy picit megváltoztattam, mivel túl nagy volt a szűrés mértéke, és nem tudtam vizsgálni a bugyborékolás jelenség felléptét és megszűntét. Ezért sokkal nagyobb szűrési paramétereket használtam, mivel 32 bites állapotváltozó abszolútérték-négyzet alsó 16 bitjét használtam a táblázat megcímzéséhez, nem pedig a felső 16-ot. A tesztelés során szerzett tapasztalatokat a függelék 1. táblázata foglalja össze. Az eredmények a vártaknak megfelelően alakultak. A szűrési paraméter csökkentésével egyre jobban a zajszint alá kerül a paraméter, ebből következik, hogy (pl.: szp = 0,0469 esetben) a bugyborékolás már hallható. Minél nagyobb a maradékzaj paraméter, annál több zaj marad meg a szűrés során, de 0,25 maradékzaj paraméter esetén torzítás lép fel. Minél kisebb a paraméter, annál kisebb torzítás.

6.1.1 Frekvenciatartománybeli vizsgálat

A frekvenciatartománybeli vizsgálatok során derült ki az átviteli függvény vizsgálatakor, hogy az eredeti program eggyel kevesebb AC rezonátort valósít meg,

mint kellene, 31 helyett csak 30-at. Ennek a problémának a kiküszöbölése egyszerűen megoldható, mivel a *C* konstanst kellett beállítani a megfelelő értékre. Ekkor már 31 AC rezonátorral dolgozott az algoritmus. Az átviteli függvény vizsgálatakor szűrés üzemmódban (szűrt jelet rakjuk ki a kimenetre) kiderült, hogy rezonátorfrekvenciák között (két szomszédos rezonátorfrekvencia átlagánál) 10-20 dB-es leszívása van az algoritmusnak, szűrési és maradékzaj paraméter függvényében. Ezt nem tudtuk mivel magyarázni, mert két rezonátor frekvencia közötti jelre (a 2.2 pontban említett legrosszabb esetben is) a két szomszédos rezonátor értékének az egységnyi átvitelhez képest csak (a 2.7 ábrán látható módon) 30%-kal szabadna csökkennie, ami 3 dB körüli csökkenést jelent. A másik felvetődött probléma az volt, hogy az algoritmus nagyon csillapítja a jelet. Az algoritmus akkor is 10-20 dB-es leszívásokat eredményezett rezonátorfrekvenciák között, ha a táblázat első elemének kivételével egyeseket tartalmazott, az első elem 0 volt (ez lényegében egy hard-elbow algoritmus).

6.1.2 A fellépett problémák okainak vizsgálata

6.1.2.1 Jelcsillapítás kiküszöbölése

A túl nagy jelcsillapítás oka a következő volt: az állapotváltozó értéke –0.5-től 0.5-ig változhat, ennek négyzete 0-tól 0.25-ig. A táblázat azt tételezi fel, hogy az állapotváltozók abszolút-érték négyzete 0-tól 1-ig változhat.

Ezen probléma megoldása történhet:

- abszolút-érték négyzet 4-gyel való szorzásával
- a táblázatot úgy határozzuk meg, hogy azt feltételezze, hogy az állapotváltozó abszolútérték-négyzet értéke 0-tól 0.25-ig tart.
- az állapotváltozó értékét beszorozzuk 2-vel

A programban az utolsó megoldás van megvalósítva, méghozzá úgy, hogy az állapotváltozó meghatározásánál használt *1*/*N* helyett *2/N*-el van szorozva, hiszen mindegy, hogy hol szorozzuk be az állapotváltozó értékét. Így nem kellett külön utasításokat használni a probléma megoldására.

6.1.2.2 Rezonátorok közötti jelcsökkenés vizsgálata

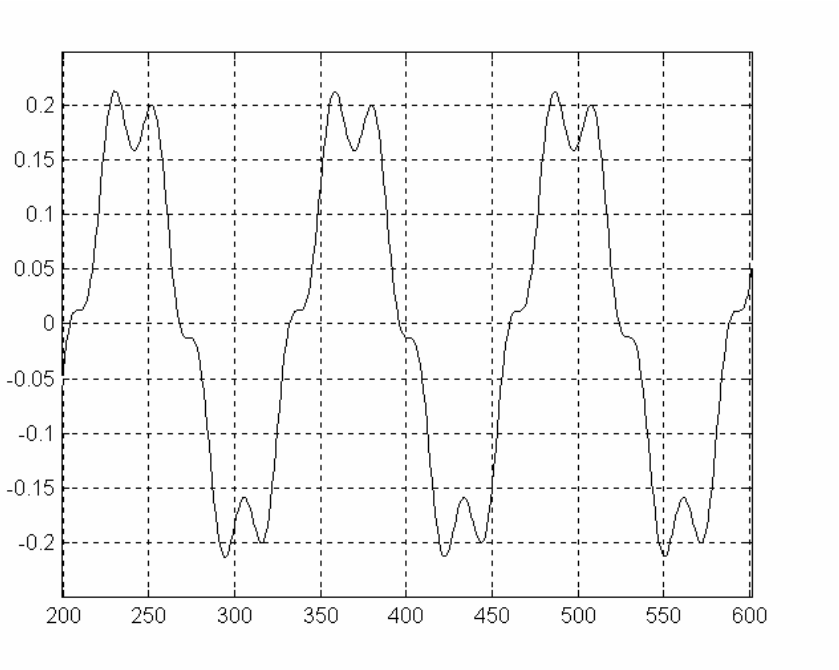
Különböző paraméterekre a leszívás mértéke más volt. A leszívás igazából a maradékzaj paraméter nagyságától függött, a szűrési paramétertől nem annyira. Olyan jelenséget eredményezett, mintha a rezonátorok között az állapotváltozók nagyon kis értéket vettek volna fel, és csak a táblázat alját tudták volna megcímezni. Ennélfogva a szűrési ciklus részletes vizsgálat alá került. Minden a szűrési algoritmusban kiszámolt eredményt az adat memóriába raktam el egy teszt vektorba, majd a program futásának megállítása után az Ezkitapp program segítségével feltöltöttem a memóriából a disk-re. Ily módon bebizonyosodott, hogy a szűrési algoritmus helyesen működik, megfelelő „bemenetre" megfelelő eredményt ad. Az állapotváltozók felvettek akkora értékeket, hogy az értékeikből előállított cím a táblázat második felében lévő elemre mutatott, tehát a táblázat jól van illesztve az állapotváltozók értékeihez.

A vizsgálatok során kiderült, hogy az állapotváltozó értéke rossz a programban, ezért minden fontosabb jelet ellenőriztem az algoritmusban. Elsőként a *re_r* és *im_r* bufferek elemeinek időtartománybeli viselkedését. *Re_r* vagy *im_r* bufferek egy elemének egy szinuszt kell adni, ha minden lépésben kirakjuk az aktuális értéket a D/A-ra. A valós és képzetes bufferek elemeinek időtartománybeli szinusz jelei között 90^o fázistolásnak kell lennie, frekvenciájának pedig a hozzá tartozó rezonátor frekvenciájának kell lennie (pl.: az első elem az első rezonátorhoz tartozik, ennek a rezonátornak a frekvenciája a mintavételi frekvencia *N*-ed része.

A k. rezonátor frekvenciája:

$$f_{k.rezonátor} = k * \frac{f_s}{N} \tag{.1}$$

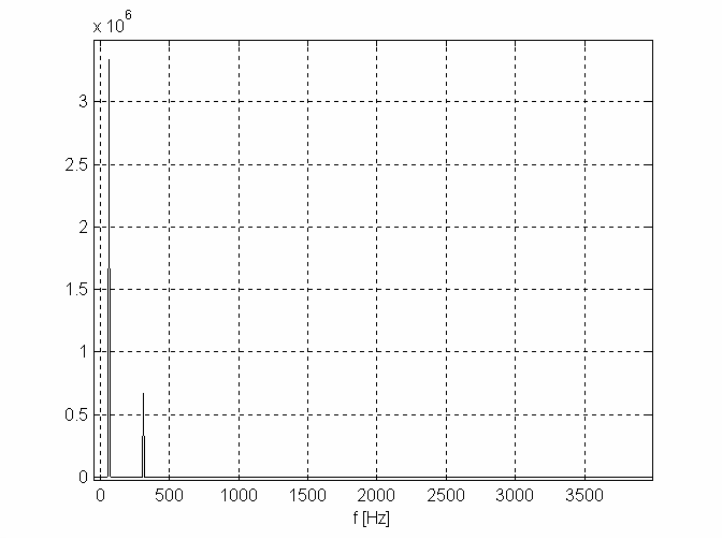
ahol f_s a mintavételi frekvencia, *N* a rezonátorok száma).



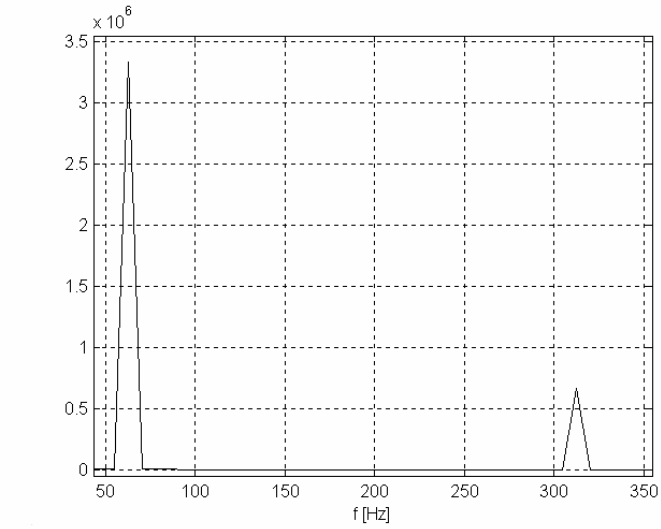
6.1.ábra Első rezonátor állapotváltozójának reális része

A frekvencia mérését Lissajous-ábrás módszerrel és egy B&K 1051-es szinusz generátorral végeztem. Minden rezonátor frekvenciája megfelelt a számolt értéknek, maximum egy-két tized Hz-es eltérések voltak. Az utolsó négy rezonátor esetében a szinusz jel „belengett”, aminek az oka a sigma-delta D/A volt.

Ezután a hibajelet vizsgáltam meg, ez jó volt. Majd az *1/N*(y(m)-c*x.')* jelet és ez is helyes volt. Majd az állapotváltozót raktam ki a kimenetre. Az állapotváltozóknak az időtartományban egy szinuszt kell adniuk minden tetszőleges frekvenciájú bemeneti jel esetén. Az állapotváltozó két szinusz jel szorzatából áll elő. Két szinusz jel szorzatának eredménye a két szinusz jel frekvenciájaiból képzett különbségi és összeg-frekvenciájú jelek összege. Viszont az algoritmusnak a 2.9-es képletben lévő *g*-vel egy alapsávba történő lekeverést kellene megvalósítania és az összeg frekvenciának nem „kellene” megjelennie. A 6.1-es ábrán a 125 Hz-es rezonátorhoz tartozó állapotváltozó reális része látható. A bemeneti jel 187,5 Hz volt. A szorzás miatt az alapharmonikus (különbségi jel) 187,5 - 125 = 62,5 Hz-es a felharmonikus (összeg jel) 187,5 + 125 = 312,5 Hz-es lesz.



6.2. ábra Az állapotváltozó spektruma



6.3. ábra A állapotváltozó spektrumának egy része

A 6.2. ábrán a 6.1-es ábra spektruma látható, a 6.3. ábrán a spektrum egy kinagyított része van ábrázolva. A 6.3. ábrán látható első spektrumvonal az előbbi számításnak megfelelően 62,5 Hz körül van. A másik spektrumvonal az előbb számolt 312,5 Hz közelében helyezkedik el. Ez a jel (állapotváltozó) a kimenet jel előállításakor fel lesz keverve a megfelelő $c_{i,n}$ paraméterrel. Ennél a keverésnél a jel újra meg lesz szorozva egy 125 Hz-es szinusz jellel. Ebből következik, hogy kimeneti jelben ezen rezonátor által előállított komponensek száma 4 lesz, ezek frekvenciája pedig: 125 – 62,5 = 62,5

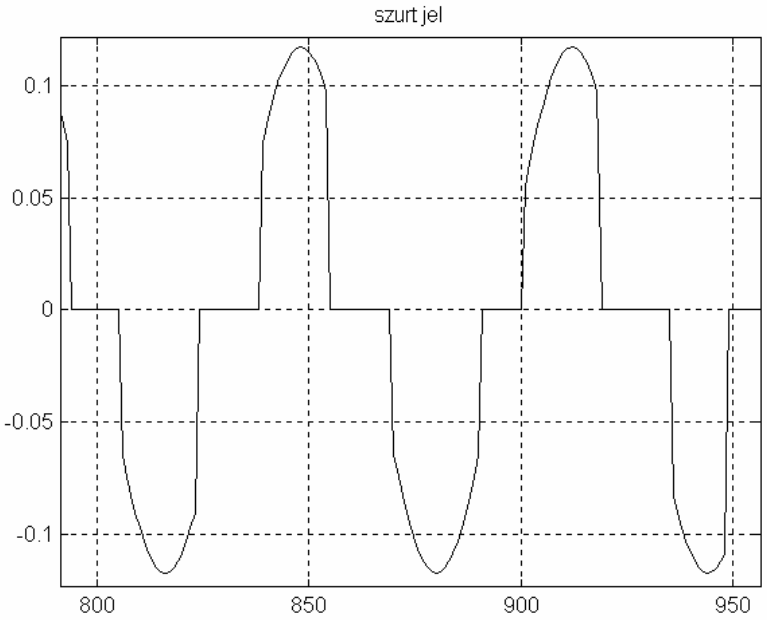
Hz, 125 + 62,5 = 187,5 Hz, 312,5 – 125 = 187,5 Hz, 312,5 + 125 = 437,5 Hz. Így már magyarázható a tesztelésekkor kapott eredmény, ami az volt, hogy nem rezonátorpozíciókba eső bemeneti jel esetén az algoritmus nem állította vissza alakhelyesen a jelet.

Matlabbal is meg lett vizsgálva az állapotváltozó időbeli viselkedése, és ott is ezt az eredményt kaptam. Matlabbal megvizsgáltam, hogy ha megszorozzuk az állapotváltozó értékét a rezonátorhoz tartozó frekvenciájú szinusz jellel (visszakeverés a rezonátor frekvenciatartományába), akkor az előbb számolt frekvencia komponenseket kaptam. Viszont, ha a lekeverésnél csak az összeg frekvencia maradna meg, akkor a felkeverés során előállna a 62.5 és 187.5 Hz-es jel a szorzás miatt, de ebből is az algoritmus szerint csak az összeg jelnek kell a kimenetben szerepelnie (mivel frekvencia felkeverésről van szó). Ezen probléma kiküszöbölése nem megoldható, ebből következik, hogy az algoritmus nem működik helyesen szűréssel együtt a rezonátor közötti frekvenciákra. A rezonátorfrekvenciákra az állapotváltozóknak állandó az értéke. Mivel nem lehet megoldani a problémát, ezért a másik, 2.7-es egyenlet szerinti struktúrát kezdtem el megvalósítani. Ebben a struktúrában az állapotváltozók időtartományban szinuszt adnak.

6.2 Rezonátoros struktúra tesztelése

Miután a 2.6 egyenletnek megfelelő algoritmust beprogramoztam, és első lépésben úgy ítéltem meg, hogy helyesen működik, azután elkezdtem részletesen tesztelni az algoritmust. Első lépésben a hibajel alakulása lett vizsgálva, minden rezonátorfrekvencián ellenőrizve lett, hogy a hibajel tényleg nulla-e. Hiszen, ha ez nem teljesül, akkor biztosan nem működik helyesen az algoritmus. A hibajel minden frekvencián „nulla” volt (nem volt mérhető), tehát az algoritmus ebből a szempontból helyesen működik. Ezután az állapotváltozó lett megvizsgálva. Ez szinusz volt, tehát nem lépett fel a másik algoritmusnál lévő jelenség, mivel ott nem volt szinuszos az állapotváltozó jele. Ezután a szűrési algoritmus lett beépítve a programba. Először hard-elbow algoritmussal lett tesztelve. Rezonátorfrekvenciákra az algoritmus jól működött, lehetett találni egy 20 mV-os sávot, ahol (a táblázatban ilyenkor az állapotváltozók

abszolútérték-négyzetből előállított cím pont szűrési paraméter közvetlen közelébe esett, volt amikor felette volt, volt amikor alatta volt, a bemeneti jel nagyságának függvényében) szinusz bemeneti jel esetén a kimeneti jel a 6.4. ábrának megfelelően alakult. Ennek oka pedig az volt, hogy az állapotváltozó abszolútérték-négyzete állandó bemeneti jel esetén nem volt állandó, hanem egy szinusz volt, ami kis amplitúdójú volt és egy egyen-szinttel el volt tolva a nullához képest (lényegében egy egyen-jelen ült egy szinusz). Emiatt lépett fel a 6.4. ábrán látható jelenség.



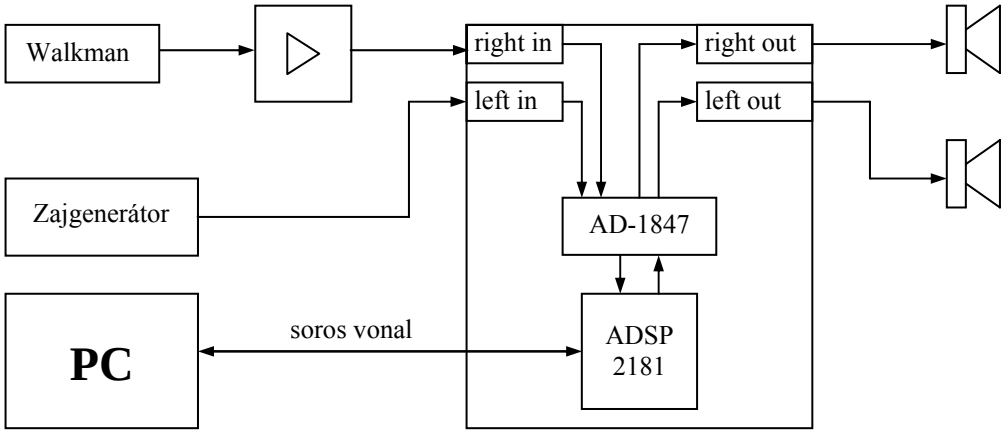
6.4. Ábra Szűrési határhelyzet

A jelenség Matlabbal is meg lett vizsgálva, és hasonló eredményt adott. Lehetett találni egy olyan határt, (ez a határ elég nehezen volt megállapítható) ahol fellépett a jelenség. Az állapotváltozó abszolútérték-négyzetének ingadozásának oka az volt, hogy a bemeneti jel frekvenciájának változásával az állapotváltozó valós és képzetes része nem változott egyformán. Amikor egy kicsivel a rezonátorfrekvencia alatt vagyunk, akkor a valós rész növekszik, a képzetes rész pedig csökken, ha a rezonátorfrekvencia felett vagyunk, akkor fordított a helyzet (valós rész csökken, és a képzetes növekszik). Az állapotváltozók megfelelő értéke csak kis ideig növekszik, utána elkezd csökkenni.

A rezonátorközi frekvenciákra ez az algoritmus sem működött teljesen helyesen. Jelalak és amplitúdó hiba lépett fel. Ezen hibák kiküszöbölésének esetleges módszereiről a 4.6. fejezetben olvashattunk.

A programban az ablakozás meg lett valósítva, mégpedig oly módon, hogy a táblázat megcímzéséhez minden állapotváltozóra kiszámoljuk az ablakozott állapotváltozót, nem pedig oly módon, hogy az egész ablakozott állapotváltozót kiszámoljuk és egy másik bufferbe tároljuk. Az ablakozás következtében a kimeneti jelalak helyesen ált elő, azonban az amplitúdó-hiba fellépett.

6.2.1 Tesztelés eredményei



6.5. ábra Mérési elrendezés beszédjel zajszűrésének teszteléséhez

A rezonátoros struktúra tesztelése beszédjelre rádió felhasználásával történt, a mérési elrendezés a 6.5 ábrán látható. A rádió használatát az indokolta, hogy nem lehetett felkészülni előre a beszédre, tehát egy ismeretlen beszédjel lett megszűrve, ami ismeretlen jel/zaj viszonyt is jelentett. Ezáltal a rádió felhasználása alkalmas arra, hogy olyan paraméter-beállítást lehetett megtalálni, amely többé-kevésbé minden, megfelelően nagy jelnagysággal rendelkező bemenő jel mellett kielégítő szűrést biztosít.

A rádió után be kellett iktatni egy erősítőt, mivel a rádió (walkman) kis jelszintet adott, és nem tudta megfelelően meghajtani a kártyát. A zajt egy zajgenerátorral

állítottam elő, amely jele a DSP egyik bemenetére kerül, míg a beszédjel a másikra. A két jel összegzésre került a processzorban.

A programal a tesztelések során megfelelően működött, a szimuláció során megismert összes jelenség itt is fellépett. A paraméterek megfelelő beállítása mellett sikerült kb. 10 dB-es zajcsökkenést elérni, viszonylag kis torzítás mellett. Ilyenkor a szűrési paraméter 0,001 és 0,002 között tartományban változott és a maradékzaj paraméter 0.2 körül volt. Ennél kisebb maradékzaj paraméter esetén, már nagyon jelentős volt a bugyborékolás és a torzítás, tehát nem volt képes a meghagyott zaj elfedni őket. Nagyobb maradékzaj paraméter esetén pedig nem volt megfelelő mértékű a zaj csökkenése.

Mivel a minél nagyobb zajszűrés a cél, ami azonban a maradékzaj paraméter csökkentésével oldható meg, ugyanakkor, ha a maradékzaj paramétert nagyon lecsökkentjük (0.2 alá), akkor már a visszamaradt zaj nem fedi el a bugyborékolást és a torzítást, tehát a minél nagyobb zajcsökkentésnek korlátjai vannak a programban. Ez a korlát a már megemlített 0.2..0.25 körüli maradékzaj paraméter. A szűrési paramétert minden esetben a zajszint fölé kell beállítani, hiszen, ha alatta van, akkor mindenképpen fellép a bugyborékolás, azonban túl nagyra nem érdemes beállítani, hiszen akkor sokkal jobban torzít a szűrt beszéd.

A táblázat 256 elemű volt, ezért a szűrési paraméter értéke az 5.1. táblázaton látható töréspont értékét adja meg.

A tesztelések során az is meg lett vizsgálva, hogy az algoritmus helyesen működik-e nem szélessávú zajok esetén. A nem szélessávú zajok előállítása a zajgenerátor sávszélességének és véletlenszám-generátor ismétlődési idejének változtatásával történt. Ebben az esetben is megfelelő mértékű volt a zaj csökkenése.

A program megfelelő működése nagymértékben függ a bemeneti jel amplitúdójától, ebből következik, hogy optimális zajszűrést csak az adott felvételhez, vagy beszédhez külön beállítva lehet elérni. A program érzékeny a nagy dinamikájú beszédekre, hiszen a halkabb részlet a beszédből kisebb állapotváltozókat eredményez, aminek következtében kis amplitúdók esetén a beszéd összes komponense a szűrési paraméter alá kerülhet. A programnak megfelelően nagy jelszintre van szüksége, ami képes meghajtani a programot.

7 Továbblépési lehetőségek

Az adott feladatban továbblépni többféle irányba lehet. Egyik ilyen a zajszűrő algoritmus fejlesztése, a másik lehetséges irány a hardware cseréje egy másik gyorsabb 32 bites processzorra (pl.: a tanszéken is megtalálható Analog Devices cég SHARC processzorra).

Ez első továbblépési lehetőség azért indokolt, mert léteznek jobb zajszűrő algoritmusok, de ezek vagy nem publikusak, vagy nem megoldhatóak jelfeldolgozó processzoron a bonyolultságuk miatt. Egyik ilyen továbblépési lehetőség lenne a maradékzaj csökkentésére a *Vaseghi* féle másodlagos szűrés alkalmazása, ami abból áll, hogy a spektrumvonalak élettartalmát vizsgálja, és ennek alapján dönti el, hogy zaj vagy hasznos jel az adott frekvencia komponens, mivel a zajkomponensek élettartama rövidebb, mint a beszéd komponensé.

Esetlegesen másik jelfeldolgozó processzor alkalmazása azért lenne indokolt, mert akkor nagyobb rezonátorszám alkalmazása lehetővé válik nagyobb bitszámú DSP esetén, és gyorsabb processzor esetén magasabb mintavételi frekvenciával is működhetne, amely szintén javulást eredményez. Bár lebegőpontos processzor alkalmazása esetén az előállítási költség jelentősen megnőhet, hiszen egy lebegőpontos processzor ára többszöröse egy fixpontosénak, de egy nagyobb bitszámú és gyorsabb processzor esetén javulás érhető el.

8 Összefoglaló

Munkám során a beszédjeleket terhelő zaj szűrésével foglalkoztam. Feladatom volt a zajszűrő algoritmusok áttekintése, a megfelelő kiválasztása, a zajszűrés Matlab szimulációjának megvalósítása és a kiválasztott algoritmus Analog Devices cég által gyártott ADSP-2181 és jelfeldolgozó processzorra való implementálása.

Mivel a kiválasztott algoritmusok a frekvenciatartományban végzik a szűrést, ezért első lépésben a feladatom kiírásában meghatározott rezonátoros megfigyelő struktúra elméletével ismerkedtem meg, majd ezután a struktúrát Matlabban szimuláltam. Itt megismerkedtem mind a rezonátoros, mind az integrátoros koncepcionális jelmodellel és a hozzájuk tartozó megfigyelőkkel, és ezen megfigyelők tulajdonságaival.

Ezt követően a kiválasztott zajszűrő algoritmusokkal ismerkedtem meg részletesen. Ezek az algoritmusok a soft- és hard-elbow algoritmusok és azok változatai voltak. Az algoritmusokat a rezonátoros megfigyelő struktúrába beleágyazva teszteltem. A tesztelés során összehasonlítottam az algoritmusokat és a legjobbat közülük kiválasztottam a jelfeldolgozó processzoron való implementálás céljából.

A szimulációk során a Matlab programmal is részletesebben megismerkedtem.

Miután kiválasztottam a legjobb algoritmust (módosított soft-elbow), azt be kellett programoznom a jelfeldolgozó processzorra, itt meg kellett ismerkednem a processzor algebrai assembly programnyelvével.

A program megírása után teszteltem annak helyes működését. Tesztelésnél első lépésben a hibajelet vizsgáltam, majd szinusz jel felhasználásával vizsgáltam a program működését. Végső tesztként pedig beszédjelet használtam fel a teszteléshez.

A tesztelés során kiderültek a program hibái. A fellépett problémák megoldásához először a hibák okait kerestem meg, majd azokat kiküszöböltem, ha ez lehetséges volt.

A hibák kiküszöbölése után a már véglegesnek tekinthető programmal végeztem a beszédjel tesztelését. A zajcsökkenés mértékét a maradékzaj paraméter határozta meg, hiszen a módosított algoritmusoknál ez a paraméter adja meg a szűrési paraméter alatti komponensek szorzótényezőjét. A tesztelések során tapasztalhatóak voltak a szimuláció során fellépett jelenségek, úgymint bugyborékolás és torzítás.

Sikerült egy olyan működő programot megírnom, amely képes kb. 10 dB-el zajszűrést megvalósítani megfelelően kis torzítás mellett. A feladatom másik előnye,

hogy viszonylag olcsón megvalósítható, mivel fixpontos jelfeldolgozó processzorra készült a program.

9 Irodalomjegyzék

[1] Kollár István, Péceli Gábor, Sujbert László, “Digitális jelfeldolgozás (Hallgatói segédlet)”, 5. Fejezet: Rekurzív DFT, A megfigyelőelmélet alapjai, pp. 42-62, BME, Budapest, 2000

[2] Sujbert László, “Periodikus zavarhatások csökkentésének aktív módszerei”, Ph.D. 2.4 fejezet, BME, Budapest, 1997

[3] http://www.mit.bme.hu/~bako/ZAOZ/digital.htm

[4] Bakó Tamás, “Régi hangfelvételek rekonstrukciója”, Diplomaterv, BME, Budapest, 2000

[5] Analog Devices, “ADSP-2100 EZ-KIT-Lite Reference Manual”, Analog Devices Inc, Norwood, 1995

[6] Analog Devices, “ADSP-2100 Family User’s Manual”, Analog Devices Inc, Norwood 1995

[7] http://www.analog.com/pdf/ADSP2181_d.pdf

[8] http://www.analog.com/pdf/ad1847.pdf

[9] L. Sujbert, G. Peceli, Gy. Simon, “Resonator Based Non-parametric Identification of Linear System”, IEEE Instrumentation and Measurement Technology Conference, Budapest, Hungary, May 21-23, 2001

10 Függelék

10.1 Integrátoros alapstruktúrájú program valós idejű teszteredményei

10.1. táblázat

szűrési paraméter	maradékzaj paraméter	Észrevételek
0,0469	0,25	zaj nagymértékben csökken alig van, bugyborékolás kicsit hallható, torzítás van
	0,375	zaj nagymértékben csökken bugyborékolás nem hallható, torzítás nem észlelhető
	0,5	zaj alig csökken bugyborékolás nem hallható torzítás nem hallható
0,0625	0,25	zaj nagymértékben csökken alig hallható bugyborékolás nem hallható, de mintha néha lenne torzítás jelentős kis jelszinteknél, de érthető a szöveg
	0,375	zaj nagymértékben csökken, de hallható bugyborékolás nem hallható torzítás kis jelszinteknél lép fel
	0,5	zaj csökkenése elég kis mértékű bugyborékolás nem hallható torzítás nem hallható
0,125	0,25	zaj nagymértékben csökken, "alig hallható" bugyborékolás nem hallható torzítás jelentős, és nem csak kis jelszinteknél lép fel
	0,375	zaj nagymértékben csökken, de hallható bugyborékolás nincs torzítás nem hallható, talán kis jelszinteknél egy kicsi van
	0,5	zaj csökkenése elég kis mértékű bugyborékolás nincs torzítás nem hallható
0,25	0,25	zaj nagymértékben csökken, "alig hallható" bugyborékolás nincs torzítás van kis jelszinteknél elég jelentős
	0,375	zaj jelentős mértékben csökken bugyborékolás nincs torzítás nem hallható, csak a nagyon kis jelszinteknél lép fel
	0,5	zaj kis mértékben csökken bugyborékolás nincs torzítás nem hallható

Ebben az esetben az abszolútérték-négyzet értéke az mr regiszterben van, és a táblázat megcímzéséhez az alsó 16 bit van használva a kis jelszintek miatt. ezért vannak ekkora szűrési paraméter értékek. A táblázat célja, hogy bemutassa az algoritmus a szimulációnak megfelelő eredményeket adott. A szűrési paraméter azt jelenti, hogy a táblázat annyiadik eleménél van a töréspont. A táblázat 2048 elemből áll. Az elvi szűrési paraméter a *seszp.m* Matlab függvénynek megadott értéket jelenti.

10.2 A rezonátoros Matlab program

```
function mfdft(N,R,Q,P,SZ,SZP,K,qwer)

%==============================
%szuresi tablazat eloallitasa
%==============================
% soft-elbow
M = K;
t = 1/M:1/M:1;
SZ2 = SZ/32767;
for i=1:M
   szp(i) = (t(i)^2-SZ2)/t(i)^2;
end

a = find(szp>SZP);
szpnew = SZP*ones(1,M);
szpnew(SZ:1:M) = szp(a(1):1:M-SZ+a(1));

%========================

t = 0:1/(R*N):1-1/(R*N);
y = qwer*sin((Q*R+P)*2*pi*t);
x(N+1) = .1;
xki(N+1) = .1;
yki(R*N+1)=0;
z(N,N)=0;

for n=1:N
   zm(n) = exp(j*2*pi*n/N);
   g(n) = (1/N)*zm(n);
   z(n,n) = zm(n);
end
c = ones(1,N+1);
c(N+1) = 0;
z(N+1,N+1) = 0;
g(N+1) = 0;

for m=1:R*N

    xki = (z*x.').'+g*[y(m)-c*x.'];
    yki(m) = c*xki.';
    xkimfel = -0.5*xki;
    xkickonv(2:1:N-1) = xki(2:1:N-1)+xkimfel(3:1:N)+xkimfel(1:1:N-2);
    xkickonv(1) = xki(1) + xkifel(2) + xkiminuszfel(N-1);
    xkickonv(N) = xki(N) + xkifel(1) + xkiminuszfel(N);
    absxki = abs(xkickonv);              % szures kezdete soft-elbow
    anglexki = angle(xkickonv);
    sortszp2 = ceil(K*absxki+1);
    sortszp2(N+1) = 1;
    xkickonv(N+1) = 1;
    szurtxki = szpnew(sortszp2).*xki;
    xkiszurt = szurtxki;
    xkiszurt(N+1)=1;
    ykiszurt(m) = c*xkiszurt.';               % szures vege
    realyki(m) = real(yki(m));
    hibajel(m) = realyki(m) - y(m);
    realykiszurt(m) = real(ykiszurt(m));
```

```
    hibajelszurt(m) = realykiszurt(m) - y(m);
    x=xki;
 end
figure;
plot(t,y,'b',t,realyki,'r',t,real(ykiszurt),'k');
title('mfdftrezszckonv');
zoom;
```

10.3 A rezonátoros jelfeldolgozó processzorra írt program

```
.module/ram/abs=0      mfdftszse;        {program starts at pm = 0}
{az egyik bemenetre a zaj a masikra a jel kerul}
.include    <c:\adi_dsp\21xx\ezkitl\2181\dsp\head2181\head.dsp>;
.include    <c:\adi_dsp\21xx\ezkitl\2181\dsp\head2181\it.tbl>;
.include    <c:\adi_dsp\21xx\ezkitl\2181\dsp\head2181\io_mac.dsp>;
.include    <c:\adi_dsp\21xx\ezkitl\2181\dsp\head2181\ad_mac.dsp>;
.include    <c:\adi_dsp\21xx\ezkitl\2181\dsp\head2181\init_mac.dsp>;

.const           N = 64;                      {rezonatorszám}
.const           M = N/4;
.const           C = N/2-1;             {number of AC resonators}
.const           ckonvmod = C-2;
.const           alpha2 = 2048;               {2/N = 32768/N}
.const           alpha = 1024;                {1/N = 32768/N}
.const           alpha3 = 512;                {alpha3 = 1/N}
.const           szpn = 256;            {szűrési táblázat mérete}
.const           K = 4096;                    {tesztvektor mérete}

.var/dm/circ/abs=0          re_ft[C];{forgatóvektor valós része}
.var/dm/circ/abs=0x0100     im_ft[C];{forgatóvektor képzetes része}

.var/dm/abs = 0x1000   szp[szpn];  {szuresi tablazatnak helyfoglalas}
.var/dm/circ/abs = 0x2000    teszt[K];   {test tabl. helyfoglalas}

.var/dm/circ     re_x[C];          {állapotváltozó valós része}
.var/dm/circ     im_x[C];          {állapotváltozó képzetes része}
.var/dm/circ     re_xszurt[C];     {szűrt állapotváltozó valós része}
.var/dm/circ     im_xszurt[C];   {szűrt állapotváltozó képzetes része}
.var/dm          dc;               {dc állapotváltozó}
.var/dm          dcszurt;          {szűrt dc állapotváltozó}
.var/dm          insz;
.var/dm          lpsz;
.var/dm          in1;
.var/dm          in2;

.var/dm          alfa;
.var/dm          alfa2;
.var/dm          alfa3;
.var/dm          ckonv;

.init alfa:      alpha;
.init alfa2:     alpha2;
.init alfa3:     alpha3;
.init ckonv:     ckonvmod;
```

```
.init re_ft:
.include    <c:\users\tl\adsp2181\ftre0064.dat>;{forgatót. valós része}
.init im_ft:
.include    <c:\users\tl\adsp2181\ftim0064.dat>;{ft. képzetes része}
.init szp:
.include  <c:\users\tl\adsp2181\szp256se.dat>; {szűrési táblázat}

start:      sample_rate(0x0);       {sample rate setting}

      init;                   {DSP and ADC initialization}

      i2 = ^re_x;             {cimregiszterek beállitása}
      i3 = ^im_x;
      i4 = ^re_ft;
      i5 = ^im_ft;
      i7 = ^teszt;

      l2 = C;
      l3 = C;
      l4 = C;
      l5 = C;
      l6 = N;
      l7 = %teszt;

      m2 = 1;                 {modósító regiszterek beállítása}
      m4 = 1;
      m3 = 0;
      m5 = 0;
      m7 = 1;

      ar = 0x0000;
      dm(insz) = ar;
      dm(lpsz) = ar;
      cntr = C;
      ar = 32767;
      do null1 until ce;            {re_x, im_x}
      dm(i2,m2) = ar;
null1:dm(i3,m2) = ar;

      ar = 0;
      dm(dc) = ar;
      ena ints;               {ITs enabled}

                  {* Main *}
      ax0 = 0;                {semafor for fl1}
      ax1 = 0;                {semafor for timer}
      set fl1;
main: idle;
      ay0 = b#0100000;        {normally waits for AD interrupt}
      ar = mstat;             {(input_samples}
      ar = ar and ay0;
      if eq jump main;        {single click on the on-board IT button:}

      ay0 = ax1;              {waits for timer interrupt (tim_it)}
      ar = ar and ay0;
      if ne jump main;        {double click: return to the monitor}
      rts;


input_samples:
```

```
      set fl0;
      ena sec_reg;

      cntr = C;                          {szures kezdete}
      i2 = ^re_x;
      i3 = ^im_x;
      i4 = ^re_xszurt;
      i5 = ^im_xszurt;
      ay0 = 0x1000;
      ax0 = dm(i2,m2);
      do szures until ce;                {ac komponensek szűrese}
            ar = dm(i2,m3);              {ablakozás}
            sr = ashift ar by -1 (hi);
            ay1 = sr1;
            ar = ax0 - ay1;
            ax0 = ar;
            m3 = ckonvmod;
            modify(i2,m3);
            m3 = 0;
            ar = dm(i2,m2);
            sr = ashift ar by -1 (hi);
            ay1 = sr1;
            ar = ax0 - ay1;
            mx0 = ar;                          {ablakozott valós rész}
            mr = mx0 * mx0 (rnd);
            ax0 = dm(i3,m2);
            ar = dm(i3,m3);
            sr = ashift ar by -1 (hi);
            ay1 = sr1;
            ar = ax0 - ay1;
            ax0 = ar;
            m3 = ckonvmod;
            modify(i3,m3);
            m3 = 0;
            ar = dm(i3,m2);
            sr = ashift ar by -1 (hi);
            ay1 = sr1;
            ar = ax0 - ay1;
            mx0 = ar;                  {ablakozott képzetes rész}
            mr = mr + mx0 * mx0 (rnd);
            mx1 = mr1;
            my1 = 2048;             {hivatalosan 256}
            mr = mx1 * my1 (ss);
            ar = mr1 + ay0,         my0 = dm(i2,m2);
            ax0 = ar;
            ay1 = 0x10ff;
            ar = ax0 - ay1;
            if le jump cimset;
            ax0 = 0x10ff;
cimset:     i6 = ax0;
            dm(i7,m7) = ax0;
            mx1 = dm(i6,m5);
            mr = mx1 * my0 (rnd),   my0 = dm(i3,m2);
            dm(i4,m4) = mr1;
            mr = mx1 * my0 (rnd),   ax0 = dm(i2,m2);
szures:     dm(i5,m4) = mr1;                   {ac rez. szűrés vége}
                                               {dc komp. szűrese}
            mx0 = dm(dc);
            mr = mx0 * mx0 (rnd);
            mx1 = mr1;
```

```
            my1 = 2048;
            mr = mx1 * my1 (ss);
            ay0 = 0x1000;
            ar = mr1 + ay0;
            ax0 = ar;
            ay1 = 0x10ff;
            ar = ax0 - ay1;
            if le jump cimset2;
            ax0 = 0x10ff;
cimset2:    i6 = ax0;
            my1 = dm(i6,m5);
            mx0 = dm(dc);
            mr = mx0 * my1 (ss);
            dm(dcszurt) = mr1;                  {dc szűres vége}

      {filtered output calculation}
      i2 = ^re_xszurt;
      i3 = ^im_xszurt;
      i4 = ^re_ft;
      i5 = ^im_ft;

      cntr = C-1;
      ar = 0;
      ay0 = dm(i2,m2);
      do outcalszurt until ce;
outcalszurt:      ar = ar + ay0,          ay0 = dm(i2,m2);
      ar = ar + ay0;
      ay0 = dm(dcszurt);
      ar = ar + ay0;
      ay0 = ar;                     { filtered output calculation end }

      ar = dm(insz);
      ar = pass ar;
      if ne jump tovabb;
      right_out(ay0);               {filtered output signal}
      left_out(ay0);                {filtered output signal}

tovabb: nop;

      right_in(ay1);                {noise in}
      left_in(ax1);                 {signal in}
      dm(in1) = ax1;
      ar = ax1 + ay1;
      ax1 = ar;                     {signal + noise}
      dm(in1) = ar;

      {output calculation}
      i2 = ^re_x;
      i3 = ^im_x;
      i4 = ^re_ft;
      i5 = ^im_ft;

      cntr = C-1;
      ar = 0;
      ay0 = dm(i2,m2);
      do outcal until ce;
outcal:     ar = ar + ay0,          ay0 = dm(i2,m2);
      ar = ar + ay0;
      ay0 = dm(dc);
      ar = ar + ay0;
```

```
      ay0 = ar;

      ar = dm(insz);
      ar = pass ar;
      if eq jump tovabb2;
      right_out(ay0);                    {output signal}
      left_out(ay0);                     {output signal}

tovabb2: nop;                            { output calculation end }

      ax1=dm(in1);
      ay1 = ay0;
      ar = ax1 - ay1;              {subtraction}

      mx0 = ar;
      my0 = alpha3;  {1/N}
      mr = mx0 * my0 (rnd);
      ay1 = dm(dc);
      ar = mr1 + ay1;
      dm(dc) = ar;                       {dc resonator update}

      my0 = alpha;                       {1/N}
      mr = mx0 * my0 (rnd);        {ax0 = 1/N*[y(m)-[1,1,...,1]*x]}
      ar = mr1;
      ax0 = ar;

      cntr = C;                    {komplex szorzás kezdete}
      my0 = dm(i4,m4);             {re_ft megfelelő eleme}
      my1 = dm(i5,m4);             {im_ft megfelelő eleme}

      do res until ce;
      ar = ax0;                          {1/N*[y(m)-[1,1,...,1]*x]}
      mr = ar * my0 (rnd),    mx0 = dm(i2,m3);  {re állapotv. eleme}
      ay0 = mr1;
      mr = mx0 * my0 (ss),    mx1 = dm(i3,m3);  {re állapotv. eleme}
      mr = mr - mx1 * my1 (rnd);
      ar = mr1 + ay0;              {állapotváltozó valós része}
      dm(i2,m2) = ar;
      ar = ax0;
      mr = ar * my1 (rnd);
      ay0 = mr1;
      mr = mx0 * my1 (ss);
      mr = mr + mx1 * my0 (rnd),    my1 = dm(i5,m4);
      ar = mr1 + ay0,               my0 = dm(i4,m4);
res:  dm(i3,m2) = ar;               {állapotváltozó képzetes része}

      reset fl0;
      rti;

change:
      ar = not ax1;
      ax1 = ar;
      pop sts;
      ay0 = 0xa000;
      dm(TCOUNT) = ay0;
      ena timer;
      push sts;
      rti;
tim_it:
      pop sts;
```

```
      dis timer;
      push sts;
      ax1 = 0;
      ar = not ax0;
      ax0 = ar;
      if eq jump be;
      if ne jump ki;
be:   set fl1;
      ay1 = 0x0000;
      dm(insz) = ay1;
      rti;
ki:   reset fl1;
      ay1 = 0xFFFF;
      dm(insz) = ay1;
      rti;

.endmod;
```

10.3.1 A programba betöltött file-k ismertetése

- *head.dsp*: itt történik az *init_cmds* regiszter beállítása, amelyben a működési paramétereket lehet beállítani, pl.: mintavételi frekvencia, számformátum beállítása. Itt történik még a *system.k* betöltése.
- *system.k*: itt az adat memóriába leképzett regiszterek nevei és azokhoz rendelt címek vannak megadva. E regiszterek címei 3e08h-tól 3fffh-ig tartanak. Ezeket a regisztereket nem szabad felülírni.
- *it.tbl*: megszakítás vektor tábla megadása
- *io_mac.dsp*: az adatbehozó *(left_in, right_in)* és kirakó *(left_out, right_out)* makrók vannak definiálva
- *ad_mac.dsp*: a mintavételi frekvencia beállító makrót definiálja.
- *init_mac.dsp*: itt történik a soros port, a *timer*, memória és rendszerbeállítások elvégzése.

10.3.2 Konstansok:

- *N*: rezonátorszám
- *C*: ac rezonátorok darabszáma
- *alpha*: a 3.8. és 3.9. egyenletben szereplő 1/*N* értéknek felel meg, az ac rezonátoroknál. Mivel a táblázatott értékeit illeszteni kell az állapotváltozó maximális értékéhez (ez 1/N helyett 2/N használatát jelentené), és mivel csak az

állapotváltozók felével működik az algoritmus, ezért az ac rezonátorokból kiszámolt kimeneti értéket szorozni kell kettővel, ezért 1/*N* helyett 4/*N* értéket vesz fel *alpha2*. A programban használt szám formátum 1.15-ös, ami azt jelenti, hogy a 16 bitből az első az előjelbit a többi bit az ½ hatványainak felel meg. A 15. bit ½ a 14. ¼ … 1. bit (LSB) értéke ½^15. Ennek megfelelően az 1 értéknek a 32768 felel meg így, *alpha2* értéke 2048 lesz.

- *alpha3*: ez is 1/N-nek felel meg, csak a dc rezonátornál van használva, értéke 1024, azaz 2/*N*, mivel a dc rezonátornak nincs komplex konjugált párja és így, csak a táblázatillesztési problémát kell megoldani.
- *szpn*: a szűréshez használt táblázat elemszámát jelöli

10.3.3 A programban használt bufferek:

- *re_ft[C]:* Az állapotváltozók frissitéséhez használt vektor valós részét tartalmazza
- *im_ft[C]:* Az állapotváltozók frissitéséhez használt vektor képzetes részét tartalmazza
- *szp[szpn]*: a szűréshez használt táblázatott tartalmazza az adat memóriában, hossza 256, kezdőcíme fix 1000h, mert a táblázatból olvasáshoz elő kell állítani a címet és ennek egy fix eltolásra van szűksége azaz 1000h. Mivel a 0h. címtől a szinusz tábla van ezért ide nem lehetett rakni. (a program memóriában pedig a program kezdődik ott)
- *re_x[C]*: az állapotváltozók valós részt tartalmazza
- *im_x[C]*: az állapotváltozók képzetes részt tartalmazza
- *re_r[C]*: a 3.12 képletben szereplő *c* vektor valós részét tartalmazza
- *im_r[C]*: a 3.12 képletben szereplő *c* vektor képzetes részét tartalmazza
- *re_xszurt[C]*: a szűrt állapotváltozók valós részt tartalmazza
- *im_xszurt[C]*: a szűrt állapotváltozók képzetes részt tartalmazza
- *dc*: egy-elemű buffer a dc rezonátor állapotváltozóját tartalmazza
- *dcszurt*: a szűrt dc komponenst tartalmazza
- *insz*: értéke eldönti, hogy a szűrt vagy szűretlen kimeneti eredmény kerül a D/A-ra
- *in1*: a bemeneti jel van elmentve benne

Szögletes zárójelekben a bufferek nevei után az elemszámuk szerepel.

10.3.4 A rezonátoros jelfeldolgozó processzorra írt program tesztelése

10.2. táblázat A rezonátoros jelfeldolgozó processzorra írt program tesztelésének eredményei

Szűrési paraméter	Maradékzaj paraméter	Zajszint [Vrms]	Észrevételek	
			Torzítás	Bugyborékolás
0,000244	0,1	0,1	van	nincs
		0,3	van	van
	0,25	0,1	van	nincs
		0,3	van	alig
		0,4	van	van
	0,375	**0,1**	nincs	nincs
		0,4	nincs	van
	0,5	0,1	nincs	nincs
		0,5	nincs	van
0,000488	0,1	0,1	van	nincs
		0,4	van, alig érthető	van
	0,25	0,1	nincs	nincs
		0,4	van	nincs
		0,5	van	van
	0,375	**0,1**	nincs	nincs
		0,4	nincs	nincs
		0,6	nincs	van
	0,5	0,1	nincs	nincs
		0,7	nincs	van
0,000977	0,1	0,1	van	nincs
		0,6	alig érthető	van
	0,25	**0,1**	nincs	nincs
		0,6	nincs	van
	0,375	0,1	nincs	nincs
		0,7	nincs	van
	0,5	0,1	nincs	nincs
		0,8	nincs	van

0,0015	0,1	0,1	van	nincs
		0,7	van	van
	0,25	0,1	van	nincs
		0,5	van	nincs
		0,8	van	van
	0,375	**0,1**	nincs	nincs
		0,8	nincs	van
	0,5	0,1	nincs	nincs
		0,7	nincs	van
0,002	0,25	**0,1**	nincs	nincs
		0,9	van	van

A felvételek egy része (vastag dőlt betűvel írt zajszint paraméterre rendelkezők) a cd mellékleten találhatóak.

A 10.2. táblázatból látható, hogy minden szűrési paraméter értéknél, 0.1-es maradékzaj paraméter esetén torzítás lép fel, és 0.25-ös paraméter esetén is sokszor fellép. A táblázatban minden maradékzaj paraméterhez tartozó legnagyobb zajszint érték a bugyborékolás felléptét jelenti, tehát ekkora zajszint értékek esetén a zaj nagysága a szűrési paraméter felett lesz, ebből következik, hogy a szűrt jelen marad vissza maradék zaj.

10.4 A CD mellékleten található hangminták ismertetése

A CD mellékletnek van egy *audio*, és egy számítógépes része. Az audio részen a 10.3.4-ban található 10.2. táblázat eredményeinek egy részét tartalmazza. A cd-re a szűretlen felvételek és az adott szűrési paraméterhez beállított legjobb maradékzaj paraméterrel készült felvételek kerültek fel. A cd-n meg azok a felvételek is megtalálhatóak, amelyek azt mutatják, hogy az adott szűrési paraméter mikor kerül a zajszint alá, hiszen itt mindenképpen fellép a bugyborékolás jelensége. Eddig a paraméterig használható az adott beállítás. Ugyanakkora szűrési és maradékzaj paraméter esetén a második felvétel, ami a bugyborékolást tartalmazza.

A felvételek sorrendje:

1. szűretlen felvétel, zajszint paraméter: 0.1 Vrms
2. szűretlen felvétel, zajszint paraméter: 0.4 Vrms
3. szűretlen felvétel, zajszint paraméter: 0.6 Vrms
4. szűretlen felvétel, zajszint paraméter: 0.7 Vrms
5. szűretlen felvétel, zajszint paraméter: 0.8 Vrms
6. szűretlen felvétel, zajszint paraméter: 0.9 Vrms
7. szűrt felvétel, szűrési p.: 0.000244, maradékzaj p.: 0.375, zajszint p.: 0.1 Vrms
8. szűrt felvétel, szűrési p.: 0.000244, maradékzaj p.: 0.375, zajszint p.: 0.4 Vrms
9. szűrt felvétel, szűrési p.: 0.000488, maradékzaj p.: 0.375, zajszint p.: 0.1 Vrms
10. szűrt felvétel, szűrési p.: 0.000488, maradékzaj p.: 0.375, zajszint p.: 0.6 Vrms
11. szűrt felvétel, szűrési p.: 0.000977, maradékzaj p.: 0.375, zajszint p.: 0.1 Vrms
12. szűrt felvétel, szűrési p.: 0.000977, maradékzaj p.: 0.375, zajszint p.: 0.7 Vrms
13. szűrt felvétel, szűrési p.: 0.0015, maradékzaj p.: 0.375, zajszint p.: 0.1 Vrms
14. szűrt felvétel, szűrési p.: 0.0015, maradékzaj p.: 0.375, zajszint p.: 0.8 Vrms
15. szűrt felvétel, szűrési p.: 0.002, maradékzaj p.: 0.25, zajszint p.: 0.1 Vrms
16. szűrt felvétel, szűrési p.: 0.002, maradékzaj p.: 0.25, zajszint p.: 0.9 Vrms