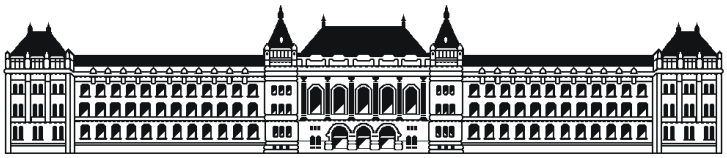


Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Audio jelek szűrése többféle zajmodell alapján

Szakdolgozat

Készítette
Turi Dániel

Konzulens
dr. Sujbert László

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

DIPLOMATERV FELADAT (ezt adják. . .)

Turi Dániel
szigorló villamosmérnök hallgató részére
(nappali tagozat villamosmérnöki szak)

Audio jelek szűrése többféle zajmodell alapján

(a feladat szövege a mellékletben)

A tervfeladatot összeállította és a tervfeladat tanszéki konzulense:

dr. Sujbert László
docens

A záróvizsga tárgyai:

Első tárgy
Második tárgy
Harmadik tárgy

A tervfeladat kiadásának napja:
A tervfeladat beadásának határideje:

dr. Diplomatervfelelős András	*dr. Tanszékvezető Gábor*
adjunktus, diplomaterv felelős	egyetemi tanár, tanszékvezető

A tervet bevette:
A terv beadásának dátuma:
A terv bírálója:

MELLÉKLET

Audio jelek szűrése többféle zajmodell alapján

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás. Fénymásolat nem jó, ezért mindenki igényeljen megfelelő számú eredeti iratot az adminisztrációban).

---

dr. Sujbert László
docens

HALLGATÓI NYILATKOZAT

Alulírott *Turi Dániel*, 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, amelyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Tudomásul veszem, hogy az elkészült diplomatervben található eredményeket a Budapesti Műszaki és Gazdaságtudományi Egyetem, a feladatot kiíró egyetemi intézmény saját céljaira felhasználhatja.

Budapest, 2009. december

Turi Dániel
hallgató

Tartalomjegyzék

1. fejezet

Kivonat

A hanganyagok zajszűrése a gyakorlatban előforduló fontos jelfeldolgozási probléma. A hangminőség megítélése alapvetően szubjektív, ezért sokszor már a zaj és a hasznos jel elvi elkülönítése nehéz feladat. A probléma megoldásához a sűrűn előforduló zajtípusokat megpróbáljuk modellekkel leírni, és a modellnek megfelelően zajszűrő módszereket megvalósítani. További követelmény ezekkel a rendszerekkel szemben, hogy on-line, azaz valós időben elvégezhetőek lehessenek.
A feladatom három gyakran előforduló zajtípus vizsgálata, és az elnyomásukra készíthető szűrőstruktúrák megvalósítása volt.
Régi, esetleg rossz minőségű felvételekre jellemző kis teljesítményű impulzusszerű zavarjelek (kattogások, sercegések) megszüntetésére a medián szűrő alkalmazása kézenfekvő.
Zajos környezetben fellépő, vagy az elektronikus jelátviteli rendszerre jellemző keskenysávú zajokat az LMS (Least Mean Squares) algoritmussal megvalósított adaptív vonaljavító struktúrával érdemes szűrni.
Végül nagyobb jelteljesítményű, de ismeretlen modellel jellemezhető sztochasztikus zajok szűrésére szűrőbankot alkalmazhatunk. Az egyes sávokban a jeleknek a jelteljesítményét vizsgáljuk, és ha ez meghalad egy a sávokra külön-külön definiált küszöbértéket, akkor a jelet változtatás nélkül átengedjük. Ha viszont a küszöbérték alatt marad, akkor zajként értelmezzük, és különböző karakterisztikák szerint nyomjuk el. Egy ilyen korábban megvalósított szűrőbankot fejlesztettem tovább a wavelet-transzformáción alapuló MRA (Multirate Analysis) segítségével. Az MRA segítségével a jelet kettő hatványai szerint változó szélességű sávokra bonthatjuk, kihasználva ezzel azt, hogy az emberi fül bizonyos frekvenciatartományokban található jeleket kevésbé tud megkülönböztetni, illetve kevésbé hordoz értékes információkat számára.
A vizsgált különböző típusú zajok a gyakorlatban együttesen jelentkeznek, ezért a három zajszűrő alrendszert összekapcsolva egy kombinált zajszűrő rendszert érdemes

létrehozni. A rendszerek összekapcsolásánál meg kell fontolni az alrendszerek összekapcsolási sorrendjét, mert az a rendszer stabilitását, és a kiadott jel torzítottságát befolyásolhatja. Az implementációt MATLAB-ban és az Analog Devices Blackfin DSP-n végeztem.

2. fejezet

Abstract

Filtering audio signals is a common problem of signal processing. In most cases it is subjective to decide which components of a signal are noise and which are information. To solve this problem several models for usual types of noise have been described. Several signal filtering methods have been invented according to these models. In most cases it is required to be able to process the signals on-line – that is in real time.
My task was to examine three different types of noise and to implement filtering methods accordingly.
In case of old or low-quality audio records it is common that they are distorted by low power impulse-like noise signals. These distortions can be filtered with median filters. Audio signals recorded in noisy environment or transferred through noisy channels the characteristic noise is usually narrow-band. In this case an adaptive line enhancer (ALE) structure which is based on the LMS (Least Mean Squares) algorithm can be used. In the case of high power noise about which only very little information is available, a filterbank structure can be used. The spectrum of the signal is divided into frequency bands and the average signal power in each of the bands is measured. If the signal power reaches a threshold defined for every band, than the signal can get through without any modification. If it is below the threshold than we can suppose that it is only noise in that band and it has to be attenuated respectively. The noise can be attenuated according to several characteristics. This filterbank structure can be developed further by using MRA (Multirate Analysis) which is based on wavelet transformation. Using MRA the spectrum is divided into different bandwith bands. This method is based on the fact that the human ear cannot detect much difference between signals within a specific domain of frequency.
As in the practical life the different types of noise can be found in signals simultaneously, it is necessary to combine the different filtering subsystems. At the connection of the subsystems care has to be taken of the sequence of the systems. In a wrong

sequence the system can become unstable, or the output signal can be too much distorted.
The filtering methods have been implemented both in MATLAB and on the Analog Devices Blackfin digital signal processor.

3. fejezet

Bevezetés

A hanganyagok zajszűrése a gyakorlatban sűrűn előforduló probléma. A zaj hasznos jelre szuperponáló zavarjel, mely csökkenti az így keletkező jel információtartalmát, valamint ha zenéről van szó, akkor a hanganyag élvezeti értékét. A hangminőséget befolyásolják a hanganyag felvételének körülményei (mikrofonok elhelyezkedése, terem akusztika), a hanghordozó minősége (pl. magnószalag), és a jelátviteli rendszer. A zajszűrés egyik fő nehézsége, hogy nehéz eldönteni mi számít hasznos és mi haszontalan jelnek, azaz zajnak. Ezért ahhoz hogy külön tudjuk választani a zajt a hasznos jeltől, legalább kevés előzetes ismerettel, feltételezéssel kell rendelkeznünk a zaj, vagy a hasznos jel jellemzőiről, azaz valamilyen zajmodellt kell alkotnunk. A szakdolgozatomban három gyakorlatban előforduló zajmodellt, és az ezek szűrésére alkotott módszereket vizsgáltam meg.
Régi felvételek esetében gyakran előfordulnak pattogások, sercegések. Ezek kis energiájú impulzusszerű zajok, és szűrésükre általánosan megoldás a mediánszűrők használata [4].
Zajos környezetben készített felvételeknél (pl. ipari berendezések közelében) előfordulhat, vagy a felvevő elektronikus rendszerben keletkezhet búgás, sípolás. Ezeknek a keskenysávú zavarjeleknek az eltávolítására alkalmasak lehetnek különféle adaptív szűrők. Egy ilyen már megvalósított struktúrát [8], az úgynevezett adaptív vonaljavítót valósítottam meg.
Végül előfordulhat, hogy gyakorlatilag semmilyen előzetes ismerettel nem rendelkezünk a zajról. Viszont mivel audiójelek zajszűrése a cél, kihasználhatjuk, hogy az emberi fül nem képes bizonyos frekvenciatartományokat élesen megkülönböztetni egymástól. Ezért jó megoldás, ha a szűrendő jelet szűrőbankokkal frekvenciasávokra osztjuk, majd az egyes sávokban egyenként valósítjuk meg a zajszűrést, méghozzá a sávokban található jelteljesítmény függvényében. Szakdolgozatomban egy elkészült diplomatervet [5] fejlesztettem tovább Wavelet szűrőbankok alkalmazásával.
A digitális jelfeldolgozás előrehaladtával egyre több, és egyre bonyolultabb mód-

szerek megvalósítása vált lehetővé. Ez eleinte a felvett hanganyagok *offline*, azaz utólagos feldolgozását jelentette, majd a technológia fejlődésével és a számítógépek teljesítményének növekedésével lehetővé vált az audiójelek valós időben történő, *online* feldolgozása. A digitális feldolgozási feladatokhoz a gyártók speciális beágyazott számítógépeket, ú.n. digitális jelfeldolgozó processzorokat fejlesztettek ki. A teljesítmény növekedését mutatja az is, hogy manapság már nem csak audiójelek feldolgozására van lehetőség, hanem akár videó jelekére is. A modern technikában már ritkán fordul elő, hogy analóg áramkörökkel végeznének el jelfeldolgozási a feladatokat.

A Méréstechnika és Információs Rendszerek Tanszék DSP laboratóriumában lehetőség nyílik korszerű digitális jelfeldolgozó processzorokkal való fejlesztésre, ezért a feladatom volt megvalósítani ezeket a zajszűrő módszereket az itt található egyik processzortípussal, az Analog Devices Blackfin processzorával.

Az első fejezetben egy röviden bemutatom az általam használt DSP kártya jellemzőit, valamint a programozáshoz felhasznált fejlesztői környezetet. Ezek után a második fejezetben foglalkozom a szűrőbankos zajszűréssel, majd a wavelet transzformáció rövid elméleti áttekintése után megvizsgálom, hogy hogyan lehet a már megvalósított struktúrát továbbfejleszteni. A harmadik fejezetben az adaptív zajszűrés rövid elméleti áttekintése után ismertetem az adaptív vonaljavító struktúráját, valamint ennek a megvalósítását. Ezek után megvizsgálom a medián szűrőt, és ennek a megvalósítási megfontolásait. Végül a negyedik taglalom az alrendszerek összekapcsolásának lehetőségeit, majd az ötödik fejezetben a mérési eredményeimet. Ezek után a hatodik fejezetben összefoglalással és kitekintéssel zárom a szakdolgozatomat.

4. fejezet

Digitális Jelfeldolgozó Processzorok

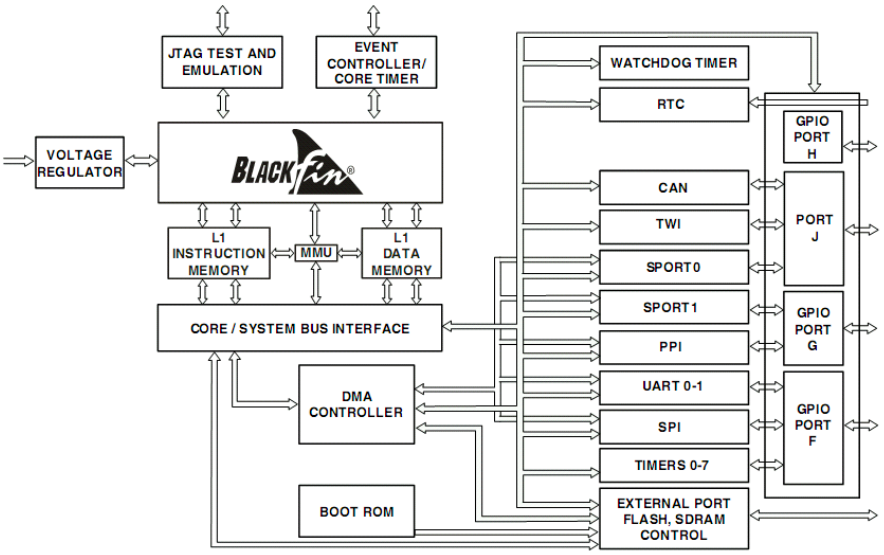
4.1. Áttekintés

A számítástechnika és az analóg-digitális átalakítás fejlődésével megjelent az igény arra, hogy a jelfeldolgozási feladatokra (pl. bonyolultabb szűrők) számítógépeket használjanak. Ezeknek a feladatoknak a sebesség és számításigénye szükségessé tette, hogy bizonyos jelfeldolgozásban gyakran előforduló feladatokra speciális processzorokat dolgozzanak ki, melyeknek mind az architektúrájuk, mind pedig az utasításkészletük a jelfeldolgozási feladatoknak megfelelően lett tervezve. Az ilyen, jelfeldolgozásra alkalmas architektúrával rendelkező processzorokat DSP-nek (Digital Signal Processor, azaz Digitális Jelfeldolgozó Processzor) nevezik.
Ezeknek a processzoroknak a segítségével lehetővé vált az ú.n. online - azaz valósidejű - feladatok ellátása, amelyeknek a számításigénye nagyon nagy is lehet. Éppen ezért a mai technikában a DSP-k elmaradhatatlan tartozékává váltak a fogyasztói szórakoztató elektronikának (pl. dekóder áramkörök DVD lejátszókban, vagy kézremegés-stabilizátorok kézi DV kamerákban), ipari környezetben használatos zajelnyomó rendszerekben, távközlésben, bonyolult elektronikus műszerekben (pl. oszcilloszkóp), vagy akár irányítástechnikai alkalmazásokban.

4.2. Architektúra

A DSP-k belső struktúrája a Harvard architektúra elveinek felelnek meg. Ez azt jelenti, hogy a kódmemória és az adatmemória elkülönülnek egymástól. Ez nagyfokú párhuzamosítást tesz lehetővé, mivel a két memória külön buszrendszeren érhető el, így válik lehetővé, hogy a processzor egyidejűleg adatot és utasításkódot tudjon felolvasni. Ennek köszönhetően tud egy órajel ciklus alatt egy utasítást elvégezni. A DSP-k speciális architektúráját a legjobban talán a konvolúción keresztül lehet

bemutatni, ami a jelfeldolgozásban az egyik leggyakrabban előforduló művelet (pl.: FIR szűrők). Ennek az elvégzésére cirkuláris bufferek használata terjedt el, vagyis egy olyan memóriaterület, ahol az éppen aktuálisan beírt, vagy kiolvasott adatra egy pointer mutat, amit minden műveletnél a címgenerátor léptet tovább. A modern DSP-k a cirkuláris buffereket speciális, ú.n. modulo címaritmetikával valósítják meg, méghozzá hardveresen, a speciális címgenerátorukkal. A konvolúció során a feldolgozandó mintasorozatokat két ilyen cirkuláris buffer tartalmazza. Az ezekből egymás után kiolvasott értékeket az aritmetikai-logikai egység egy lépésben képes összeszorozni, majd a keletkezett szorzatot egy akkumulátor előzetes értékéhez hozzáadni, és az akkumulátorban eltárolni. Ez a MAC (Multiply- Accumulate) művelet. A DSP-k ezen kívül rendelkeznek a mikrovezérlőktől elvárható tulajdonságokkal is, bár általában kevesebb perifériával rendelkeznek. Az utóbbi időben megjelentek a piacon a DSC-k (Digital Signal Controller), amik a a mikrokontrellektől megszokott perifériákat (pl. CAN buszvezérlő) a DSP-ktől megszokott architektúrával ötvözik. A most bemutatásra kerülő Blackfin DSP is egy ilyen sok perifériával rendelkező processzor, amint azt a 4.2 ábrán látható blokkvázlat is mutatja.

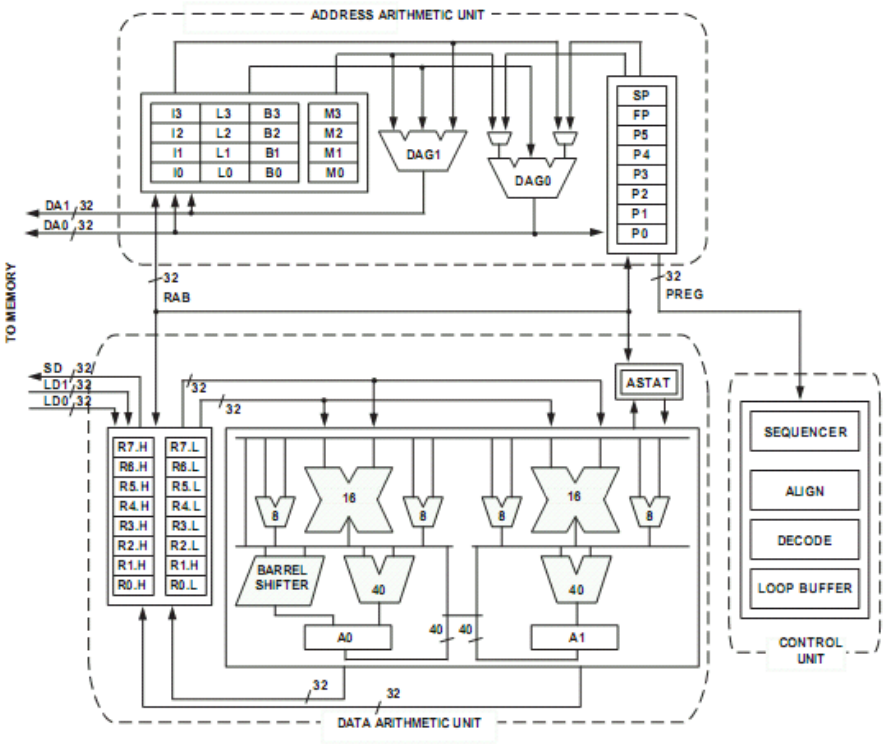


4.1. ábra. Az ADI BF-537 processzor blokkvázlata

4.3. Az ADI Blackfin 537 processzor bemutatása

A szakdolgozatomban az Analog Devices Blackfin 537-es modelljét [2] használtam, ami megtalálható a MIT tanszék DSP laboratóriumában. Ez egy nagyteljesítményű egyesített DSP-mikrokontroller architektúra, ami multimédia alkalmazásokra is

használható. A chip maga egy oktatási és kutatási célokra alkalmas kártyára van telepítve, egyéb segédáramkörökkel együtt (pl. AD-, DA-átalakítók, Ethernet chip, stb). Szintén ezen a kártyán találhatóak járulékos memória modulok, a belső memória modulokat kiegészítendő. A 4.2. ábrán áttekintést kaphatunk a Blackfin processzor magjáról.



4.2. ábra. Az ADI BF-537 belső architektúrája

4.3.1. Adat aritmetikai egység

Az adat aritmetikai egységre az erős párhuzamosítás jellemző. A processzor tartalmaz két 16 bites szorzót, két darab 40 bites akkumulátort, két 40 bites általános célú ALU-t, egy 40 bites barrel shiftert, és 4 db. 8 bites video ALU-t. Ezek az egységek egyidejűleg képesek műveleteket végezni operandusokon, így a MAC műveletek 16 bites számokra kétszer olyan gyorsan végezhetők. A 40 bit szélességű akkumulátorok nagypontosságú fixpontos számábrázolást tesznek lehetővé. A processzor 32 bites szavas, 16 bites félszavas és bájtos értékeket képes kezelni. Ezek mind lehetnek integer típusúak (előjeles, vagy anélküli), de csak a 16 és 32 bitesek lehetnek tört típusúak (a DSP C fordítója ezeket a számokat a speciális *fract16* és *fract32* típusokkal reprezentálja [3]).

Az ALU a mikroprocesszoroknál megszokott általános aritmetikai és logikai műveletek, szorzás, a szorzó eredményének az akkumulátorba összeadásos vagy kivonásos akkumulálása (MAC) mellett bonyolultabb műveleteket is támogat: pl. abszolút érték, kerekítés, maximum, minimum számítása. Utóbbiaknak az az érdekessége, hogy ezeket is egy ciklus alatt végzi el (aminek pl. medián szűrők készítésénél van jelentősége).
A 16 bites üzemmóddal SIMD műveletvégzést lehet megvalósítani, sőt mivel két független ALU is van, ezért akár négy darab 16 bites adaton végezhetjük el egyszerre a műveleteket.
Az adataritmetikai egységhez 8 db. 32 bites adatregiszter, vagy - mivel egy 32 bites regisztert két 16 bites regiszterként is lehet használni - 16 db. 16 bites regiszter tartozik.

4.3.2. Címaritmetikai egység

A címaritmetikai egység támogatja az általános processzoroknál megszokott indirekt címzést. Ezekre a 8 db. 32 bites pointer (P) regiszter szolgál. A cirkuláris bufferek működtetéséhez szükséges modulo címzéshez 32 bites index, lépéshossz, bufferhossz valamint báziscím regiszterekből a processzorban 4-4 db. található. A címaritmetika támogatja több regiszter egyidejű elérését, az adat aritmetikai egység párhuzamos feldolgozási képességének támogatására.
A DSP címaritmetikájának egy további speciális funkciója a bitcserélt (bit-reverse) címzés, ami az FFT algoritmus megvalósításához használatos.

4.3.3. Memóriaszervezés

A Blackfin processzorok módosított Harvard architektúrával rendelkeznek, ami azt jelenti, hogy habár az utasításmemória és az adatmemória el van különítve a címtérben, valamint a buszrendszerben, az utasításmemória tárolhat konstansokat. Kétféle memóriát implementáltak a chipen: az L1 memóriát, amit teljes processzor sebességgel lehet elérni, valamint az L2 memóriát, amit fél processzor sebességgel lehet elérni. Ezen kívül lehetőség van még a kártyán található külső memória modulok elérésére is, bár jóval lassabban a külső buszrendszeren keresztül.

4.3.4. Programvezérlés

A sebesség növelésére a processzor speciális ciklusszervezést használ, az ú.n. zero-overhead loop-ot. Ez azt jelenti, hogy a processzornak egy ciklus futtatása előtt nem

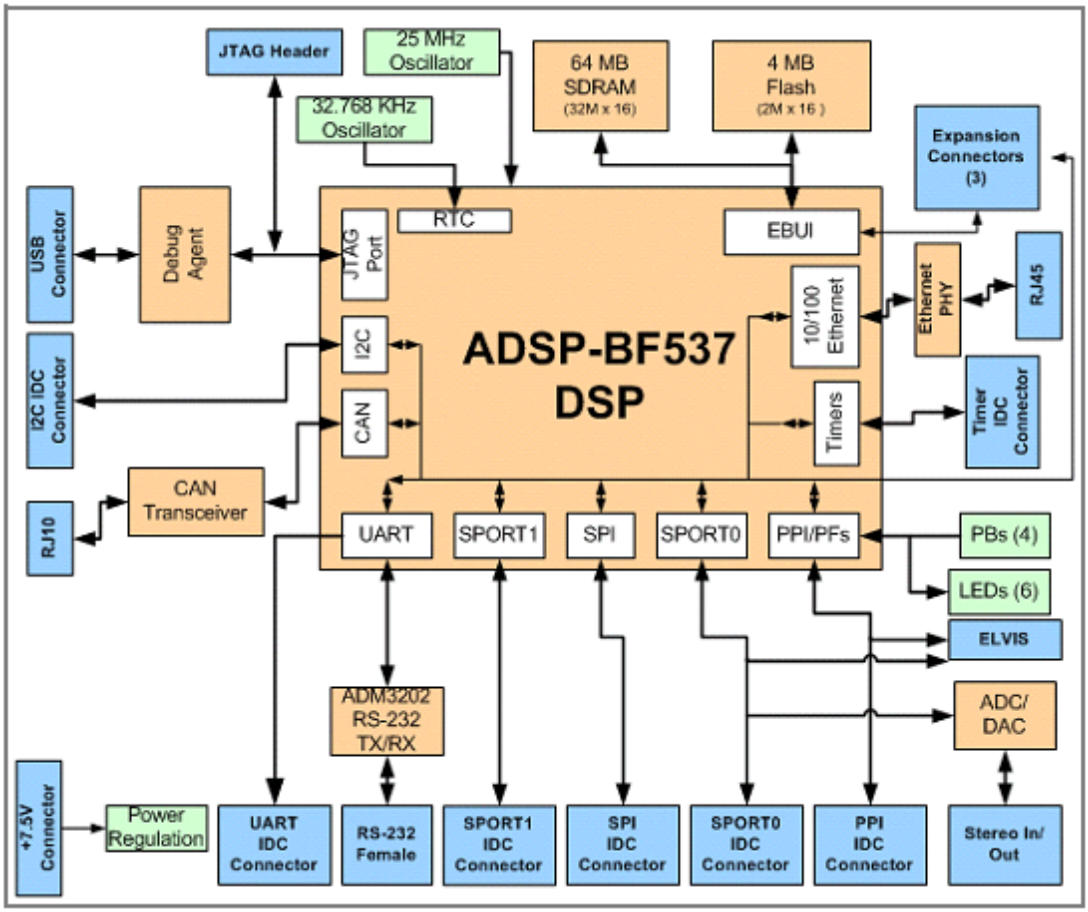
kell ellenőriznie egy ciklusszámláló változót, azon komparálási, majd feltételes ugrási műveleteket végrehajtani, hanem beépített ciklusvezérlő regisztereket használ. A zero-overhead loopnak főleg a konvolúciók gyors elvégzésénél van jelentősége.

4.4. Az ADSP-BF537 EZ-Kit kártya

Az ADSP-BF537 kártyának [1] a segítségével teljes funkcionalitásában lehet használni a DSP-t, nem kell a felhasználó által gyártott áramkörökbe beforrasztani.
Csatlakozók találhatóak hozzá számos adatátviteli szabványhoz, pl. USB, UART, CAN, SPI, JTAG, vagy Ethernet.
Bemeneti és kimeneti audiojelek csatlakoztatására 3.5 mm sztereó jack dugók szolgálnak. A bemeneti jelekből a kártyára telepített AD-átalakítók digitális adatot állítanak elő, majd az új adat készen állását megszakítással jelzik a processzornak. Erre a megszakításra reagálva szoftverből kell megoldani az adatok beolvasását. A 4.3.ábrán láthatjuk a kártya blokkvázlatát.

4.5. A fejlesztőkörnyezet

A Blackfin DSP-re való fejlesztéshez az Analog Devices saját integrált fejlesztői környezetet (IDE) biztosít, a *VisualDSP++*-t. Ennek a fejlesztői környezetnek a segítségével C nyelven, vagy a DSP saját assembly nyelvén lehet programot fejleszteni. A környezet azon kívül, hogy programkód szerkesztőt, és fordítót (mind C, mind assembly nyelvhez), egyéb funkciókkal is rendelkezik. Segítségével JTAG porton keresztül el lehet érni a DSP-t, és többek közt működés közben ki lehet olvasni a regisztereit, töréspontok segítségével meg lehet állítani a program futtatást, módosítani lehet változókat a memóriában. Ezen kívül lehetőség van a DSP működéséről dianosztikai adatokat nyerni, például a *tracing* funkció, melynek segítségével meg lehet állapítani, hogy az idő hány százalékában melyik programrészletet hajtja végre a processzor. Ezzel a DSP kihasználtságát, vagy éppen túlterheltségét lehet megállapítani.
A fejlesztői környezet ezen kívül tartalmaz még egy szimulátor modult is, arra az esetre, hogyha az fejlesztőknek nincs hozzáférésük a kártyához. A szimulált processzor regiszterei szintén vizsgálhatóak működés közben, nagyban megkönnyítve a fejlesztést.



4.3. ábra. Az ADSP-BF537 EZ-Kit kártya blokkvázlata

5. fejezet

Zajszűrés Wavelet-szűrőbankkal

5.1. Zajszűrés sávokra bontással

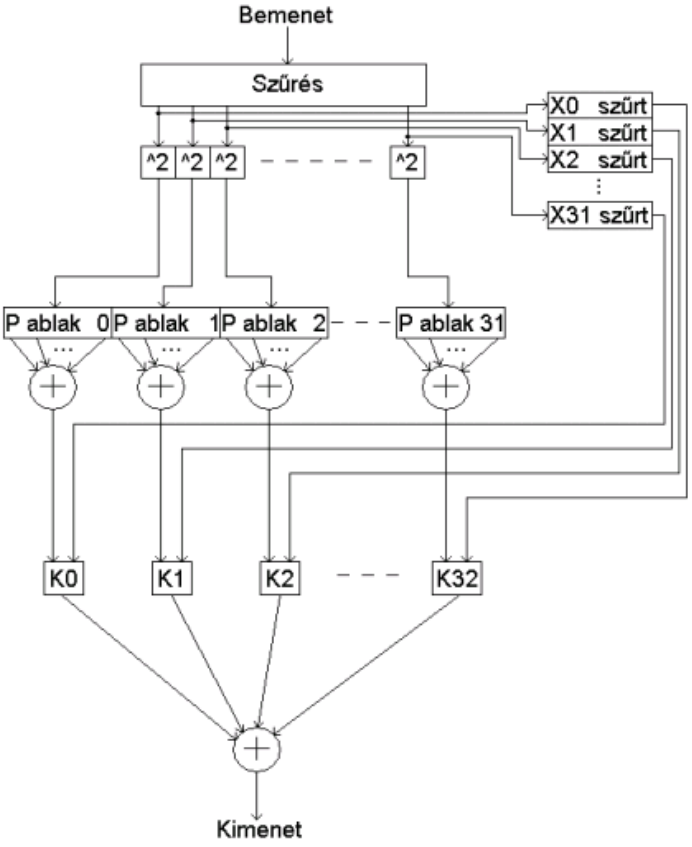
5.1.1. Bevezetés

Nagyteljesítményű szélessávú zajjal (pl. sistergés) terhelt zenei vagy beszédjelek szűrésére egy általánosan használt módszer az, hogy a jelet sávokra bontják, majd a szűrést sávonként végzik el. A szűrés módja az, hogy a zajszűrő rendszer az egyes sávok jelenergiájától függően különböző mértékben engedi át a jelet az adott sávban [5]. Tehát az egyes sávokra zajszűrő karakterisztikákat (azaz a jelteljesítménytől függő „átviteli függvényeket" definiálhatunk. A jelteljesítmény itt nem a a pillanatnyi jelteljesítményt jelenti, hanem egy rövid időtartamra (L mintára) nézve vizsgáljuk a W_L átlagos jelteljesítményt. Azt feltételezzük, hogy ha egy adott sávban alacsony az átlagos jelenergia, akkor a hasznos jel éppen nem ebben a sávban található, tehát az ebben a sávban zaj található, amit el kell nyomni.

5.1.2. A zajelnyomó karakterisztikák

Egy sávban tehát definiálnunk kell egy határteljesítményt, ami felett az adott jel változtatás nélkül képes átjutni. Arra viszont, hogy mi történjen a küszöbértékkel kisebb jelenergiájú összetevőkkel, többféle módszert lehet találni. A szakdolgozatomban négyféle karakterisztikát valósítottam meg:

- lépcsős
- hiszterézises
- lineáris elnyomású offsettel
- négyzetes elnyomású

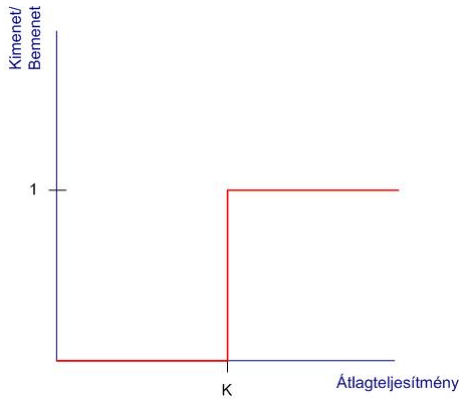


5.1. ábra. Az eredeti szűrőbank struktúrája

A legegyszerűbb a *lépcsős karakterisztika*. Ha a jelteljesítmény nem éri el a K küszöbszintet, akkor ebben a sávban a jelet teljesen elnyomjuk, vagyis a karakterisztika átvitele:

$$x_i = \begin{cases} 0 & \text{, ha } W_L < K \\ x_i & \text{, egyébként} \end{cases}$$

A karakterisztika az 5.2. ábrán látható. A probléma ezzel a karakterisztikával az, hogy ha a zajteljesítmény nagy, akkor a zaj csillapítás nélkül átjut a sávban, ha viszont a nagyra vesszük a küszöbértéket, akkor a hasznos jelünk esetleg túlságosan torz lesz. Azonkívül, ha fehér zajunk van, és a teljesítménye a küszöbérték körül ingadozik, akkor az egyenletes sistergés helyett impulzus-szerű zajok keletkezhetnek.



5.2. ábra. A lépcsős karakterisztika

Az utóbbi jelenségen segít az 5.3. ábrán látható *hiszterézises karakterisztika*. Két küszöbértéket definiálunk, egy K_H felső és egy K_L alsó küszöbértéket. Ha a jelteljesítmény alacsony volt, akkor a K_H küszöbértékig teljesen elnyomjuk a sávban a jelet. Ha előzetesen a küszöbérték fölött volt a jelteljesítmény, akkor K_L küszöbérték alá csökkenve fogjuk elnyomni. A K_H és a K_L közötti különbség megfelelően nagyra választásával elnyomhatóak a küszöbértékek környékén ingadozó teljesítményű zajok is.

A *lineáris elnyomású karakterisztika* az 5.4. ábrán látható. A „lineáris” szó a karakterisztika elnyomási tartományára utal. A küszöbérték alatti jelteljesítményű sávban a jelteljesítmény arányában nyomjuk el a jelet. Ez a karakterisztika mentes a lépcsős karakterisztikánál említett problémától, nem keletkeznek impulzusszerű zajok. Viszont a küszöbérték alatti teljesítményű zaj is átjut a szűrőbankon. Ezért definiálhatunk egy offset küszöbértéket, amivel eltoljuk a karakterisztikát. Ez alatt a küszöbérték alatt teljesen elnyomjuk a jelet:

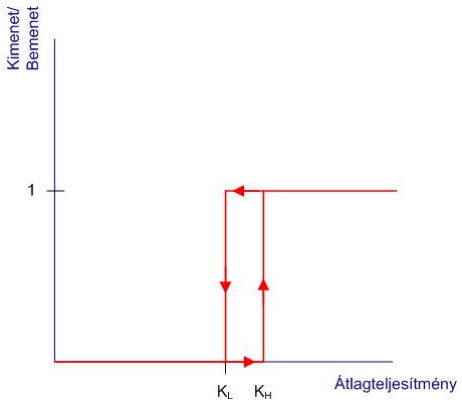
$$x_i = \begin{cases} 0, & \text{ha } \sum_{k=1}^{W_L}(x_{i(j-k+1)})^2 < K_L \\ x_i \cdot \frac{\sum_{k=1}^{W_L}(x_{i(j-k+1)})^2}{K_H - K_L}, & \text{ha } K_L \leq \sum_{k=1}^{W_L}(x_{i(j-k+1)})^2 < K_H \\ x_i, & \text{egyébként} \end{cases}$$

Egy jobb megoldása a fenti ötletnek az 5.5. ábrán látható négyzetes karakterisztika, ahol a küszöbérték alatt a jelet négyzetesen nyomjuk el:

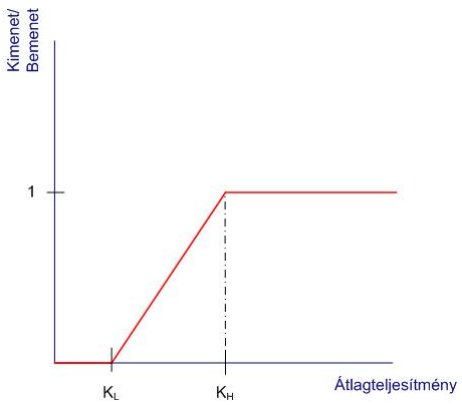
$$x_i \begin{cases} x_i \cdot (\frac{\sum_{k=1}^{W_L}(x_{i(j-k+1)})^2}{K})^2, & \text{ha } \sum_{k=1}^{W_L}(x_{i(j-k+1)})^2 < K \\ x_i, & \text{egyébként} \end{cases}$$

Kis teljesítménynél az elnyomás majdnem teljes, a küszöbérték közelében viszont kicsi.

A karakterisztikák kimenetén keletkező jeleket ezek után összeadva képezhetjük a szűrt jelet. A sávokra bontást egyszerű FIR szűrőkkel meg lehet oldani, viszont a módszer problémája az, hogy a sávok szélessége ugyanakkora, holott a hasznos jelnek általában a kisfrekvenciás komponensei hordozzák az információt. Ezért pontosabb zajszűrést kapunk, ha az alacsony frekvenciás tartományban a jelet több,



5.3. ábra. A hiszterézises karakterisztika



5.4. ábra. A lineáris karakterisztika

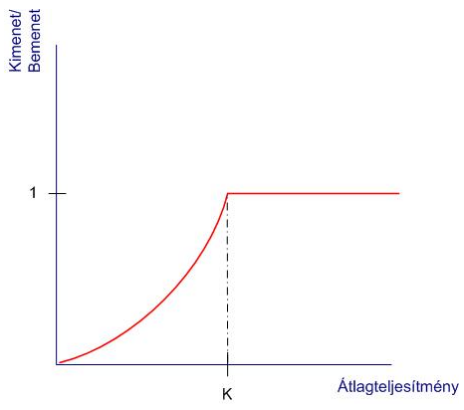
kisebb szélességű sávra bontjuk, a magas frekvenciás tartományban pedig kevesebb nagyobb frekvenciaszélességű tartományra. Erre kínál megoldást a wavelet traszformáció használata.

5.2. Zajszűrés egyenlőtlen szélességű sávokra bontással

A sávokra bontáshoz az MRA (Multirate Analysis) módszerét használtam fel, ami a Wavelet-transzformáción alapszik.
Mivel a MATLAB Wavelet Toolboxa sok különféle waveletet elő tud állítani, így az általam használt, és a DSP-be letöltött szűrő impulzusválaszokat is ezzel a toolboxszal generáltam. A MATLAB környezet nemcsak a szűréshez használt konstansok generálására használható, hanem egyben a módszer megismeréséhez, kipróbálásához, valamint a DSP által megvalósított feladat szimulálására is. A *Wavelet toolbox* beépített függvényeket tartalmaz bizonyos feladatok egyszerű megoldására, ezeket az elkészített szimulációs program teszteléséhez referenciaként használtam fel.
A módszerrel a szűrendő jelet diadikusan (2 hatványainak megfelelően) csökkenő méretű sávokra bonthatjuk, majd ezekben a sávokban a fent említett jelenergiától függő szűrőkarakterisztikákkal lehet eldöntetni, hogy teljes egészében átjut-e a jel abban a sávban, vagy esetleg csak valamilyen csökkentett mértékben. Ezek után inverz transzformációval tudjuk visszaállítani a szűrt jelet.

5.2.1. A rövid idejű Fourier transzformáció

A digitális jelfeldolgozás során mintavételezési időközönként vett mintákkal dolgozunk. Itt a jel időbeli lefutásáról pontos információink vannak, de a jel frekvenciatartománybeli viselkedéséről általában nem rendelkezünk közvetlenül információval (lásd az 5.6. ábra bal felső grafikonját), azaz rossz frekvenciafelbontást kapunk.
Ha egy végtelen hosszúságú jelsorozatot transzformálunk, akkor a jel időtartomány-



5.5. ábra. A négyzetes elnyomású karakterisztika

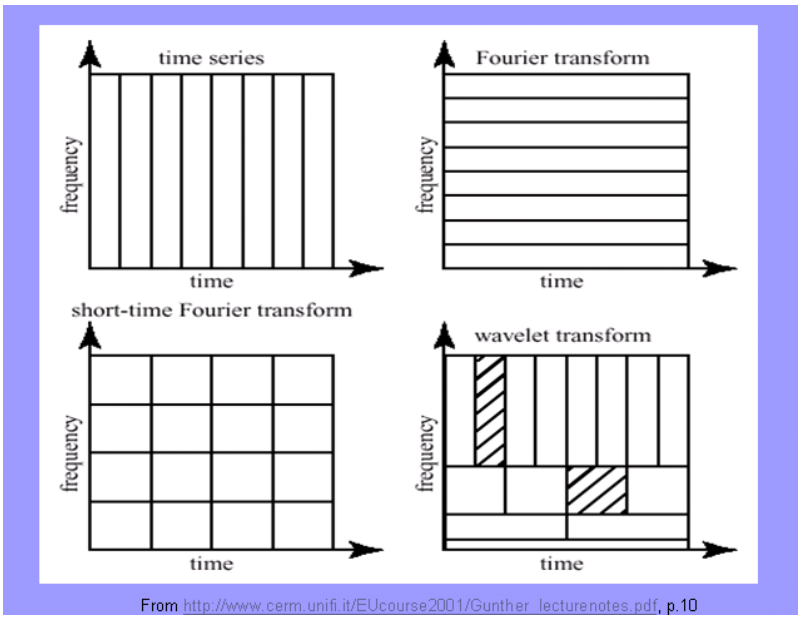
beli viselkedéséről fogunk túl kevés információval rendelkezni (lásd az 5.6. ábra jobb felső grafikonját), azaz rossz idő felbontást kapunk. Mivel a DSP-vel nem tudunk végtelen hosszú jelsorozatokat feldolgozni, hanem az AD-átalalkító által mintavételezett jeleket általában egy bufferbe töltjük, azaz egy csúszó ablakot hozunk létre. Így ahelyett, hogy egy végtelen mintasorozaton végeznénk transzformációt, egy véges időintervallumhoz tartozó mintasorozatunk van. Véges időintervallumon egy jel Fourier transzformáltját STFT-nek (Short-Time Fourier Transform) nevezzük. Mivel a véges időintervallumot matematikailag egy ablakfüggvénnyel való szorzással ábrázolhatjuk, ezért a STFT (5.1) összefüggés szerint alakul (feltéve, hogy a γ ablakfüggvény valós).

$$F_x^\gamma = \mathcal{F}\left\{x(t)\gamma(t-\tau)\right\} = \int_{-\infty}^{\infty} x(t)\gamma(t-\tau)e^{-j2\pi ft}dt \tag{5.1}$$

Ez a képlet átalakítható a transzformálandó $x(t)$ függvény és egy τ-val eltolt és f-fel modulált ablakfüggvénnyel való skaláris szorzatára, (5.2) szerint.

$$F_x^\gamma(\tau, f) = \left\langle x(t), \gamma(t-\tau)e^{j2\pi ft}\right\rangle_t \tag{5.2}$$

Ennek megfelelően a STFT az idő-frekvencia síkon egy két dimenziós ablakot hoz létre, aminek van egy időbeli és egy frekvenciabeli szélessége. Így már a jel egy adott időintervallumra eső részéről rendelkezünk némi frekvenciatartománybeli információval is. Viszont ennek az idő-frekvencia ablaknak a szélessége kötött, ami sok jelfeldolgozási alkalmazásban kényelmetlenséget jelenthet (lásd az 5.6. ábra bal alsó



5.6. ábra. Különböző jelfeldolgozási transzformációk összehasonlítása

grafikonját). Például, ha egy jelnél a kisfrekvenciájú komponensek viszonylag lassan változnak, a kisfrekvenciás komponenseket kisebb időfelbontással elegendő vizsgálni, míg a nagyobb frekvenciájú komponenseknél lényeges, hogy az időbeli lefutásukat pontosan ismerjük, viszont a frekvenciájukat elég kis pontossággal ismerni.

5.2.2. A Wavelet-transzformáció

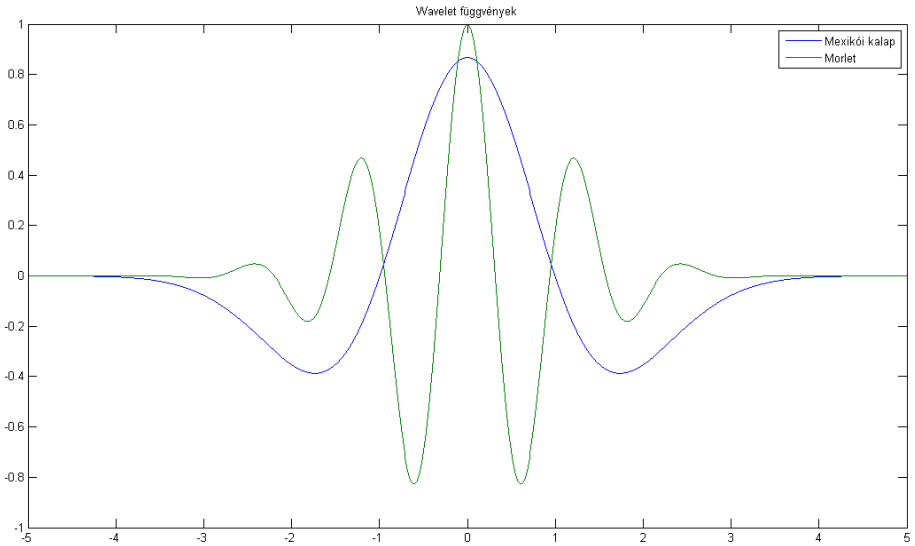
A Wavelet-transzformáció [6] az előbbi problémát oldja meg. A jelet az idő-frekvencia síkon arányosan változó szélességű ablakokkal képezi le. A Wavelet-transzformáció egy integráltranszformáció, a transzformálandó jelet a Fourier-transzformációval ellentétben nem szinuszjelekkel szorozzuk skalárisan, hanem úgynevezett waveletekkel. Az *x(t)* jel Wavelet transzformáltja tehát (feltéve, hogy a ψ wavelet valós értékű):

$$W_x^{\psi}(a,b) = \langle x(t), \psi_{a,b}(t) \rangle_t = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t) \psi \left(\frac{t-b}{a} \right) dt \tag{5.3}$$

Ahol a $\psi_{a,b}$ wavelet egy ψ ú.n. *mother wavelet* időben eltolt és skálázott változata (5.4) szerint (ahol a a skálatényező, b az időbeli eltolás). A *mother wavelet* egy olyan jel, aminek az abszolút középértéke nulla, viszont a közepes frekvenciája nem.

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi \left(\frac{t-b}{a} \right) \tag{5.4}$$

Az 5.7. képen két híres wavelet látható, az ú.n. *mexikói kalap*, valamint a *Morlet-wavelet*.



5.7. ábra. Két wavelet: a mexikói kalap és a Morlet-wavelet

A transzformált jel egyértelműen visszaállítható (5.5). képlet értelmében:

$$\hat{x} = \frac{1}{C_{\psi\tilde{\psi}}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} W_x^{\psi}(a,b) \cdot \tilde{\psi}_{a,b}(t) \frac{dadb}{a^2} \ , \tag{5.5}$$

ahol

$$C_{\psi\tilde{\psi}} = \int_{-\infty}^{\infty} \frac{\psi(f)\tilde{\psi}(f)}{|f|} < \infty \tag{5.6}$$

Ez utóbbi egyben az ú.n. megengedhetőségi feltétel, azaz csak ennek a feltételnek a teljesülésekor használhatjuk a $\psi(t)$ analízis waveletet (azaz a felbontáshoz használt waveletet) és a hozzá tartozó $\tilde{\psi}(t)$ szintézis waveletet (azaz a jel rekonstruálásához használt waveletet). Megjegyzendő, hogy ha a transzformálandó jel, valamint a waveletek valósértékű függvények, akkor a jel transzformáltja is valós értékű lesz. Ez egy további előny a Fourier-transzformációval szemben, hiszen sokkal kevésbé számításigényes.

5.2.3. A diszkrét Wavelet-transzformáció

Mivel a digitális jelfeldolgozásban nem folytonos idejű, hanem mintavételezett jelekkel foglalkozunk, ezért a wavelet transzformációnak is létezik diszkrét formája, amit Diszkrét Wavelet Transzformációnak (DWT - Discrete Wavelet Transform) hívnak. Először is a waveleteket úgy diszkretizáljuk, hogy a b időeltolást és a a skálatényezőt diszkrét lépésekben változtatjuk, méghozzá diadikusan (2 hatványai szerint). A T diszkrét időlépéssel a k. skálatényező és a k. időbeli eltolás:

$$a_k = 2^k, \ b_{m,k} = m \cdot T_k = m \cdot 2^k T \tag{5.7}$$

Így az egyes waveletek a következő összefüggés szerint alakulnak:

$$\begin{aligned} \psi_{m,k}(t) &= 2^{k/2}\psi(2^{-k}(t - m2^kT)) = 2^{k/2}\psi(2^{-k}t - mT) \ , \text{ és} \\ \tilde{\psi}_{m,k}(t) &= 2^{k/2}\tilde{\psi}(2^{-k}(t - m2^kT)) = 2^{k/2}\tilde{\psi}(2^{-k}t - mT) \ , \end{aligned} \tag{5.8}$$

ahol ψ az analízis wavelet és $\tilde{\psi}$ a szintézis wavelet.
Egy folytonos jel diszkrét Wavelet-transzformáltja tehát a jel és az analízis wavelet skaláris szorzata:

$$W_x^{\psi}(m,k) = \langle x(t), \psi_{m,k}(t) \rangle_t = \int_{-\infty}^{\infty} x(t) \cdot 2^{-k/2}\psi(2^{-k}t - mT)dt \tag{5.9}$$

Visszaállítani a jelet a szintézis wavelettel lehet a következő képlet szerint:

$$\tilde{x}(t) = \sum_{k=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} W_x^{\psi} \cdot \tilde{\psi}_{m,k}(t) \tag{5.10}$$

Mivel a feldolgozandó jelünk egy mintasorozat, ezért át kell térni az $x(n)$ diszkrét jelre. A diszkrét idejű jelek esetén diszkrét T időlépés célszerűen a t_s mintavételi periódus, így

$$T_k = 2^k \cdot t_s \ . \tag{5.11}$$

A frekvencialépés ennek megfelően:

$$F_k = 2^{-k} F < 2^{-k} \frac{1}{f_s} \ , \tag{5.12}$$

a Nyquist kritérium betartása mellett. Az $x(n)$ diszkrét idejű jel diszkrét Wavelet-transzformációja így a jel és a wavelet skaláris szorzataként a következőképpen értelmezhető:

$$W_x^{\psi}(m,k) = \langle x(n), \psi_{m,k}(n) \rangle = \sum_{n=-\infty}^{\infty} x(n) \cdot 2^{(-k/2)} \psi(2^{-k} n - m) \tag{5.13}$$

5.2.4. Az MRA elméleti alapja

A következőkben feltételezzük, hogy az analízis- és a szintézis-waveletek azonosak, vagyis

$$\tilde{\psi}_{m,k}(t) = \psi_{m,k}(t) \ , \tag{5.14}$$

azonkívül a skálázott, időben eltolt waveletek ortonormált bázisrendszert alkotnak:

$$\langle \psi_{m,k}(t), \psi_{m',k'}(t) \rangle = \delta(m - m') \cdot \delta(k - k') \ \forall k. \tag{5.15}$$

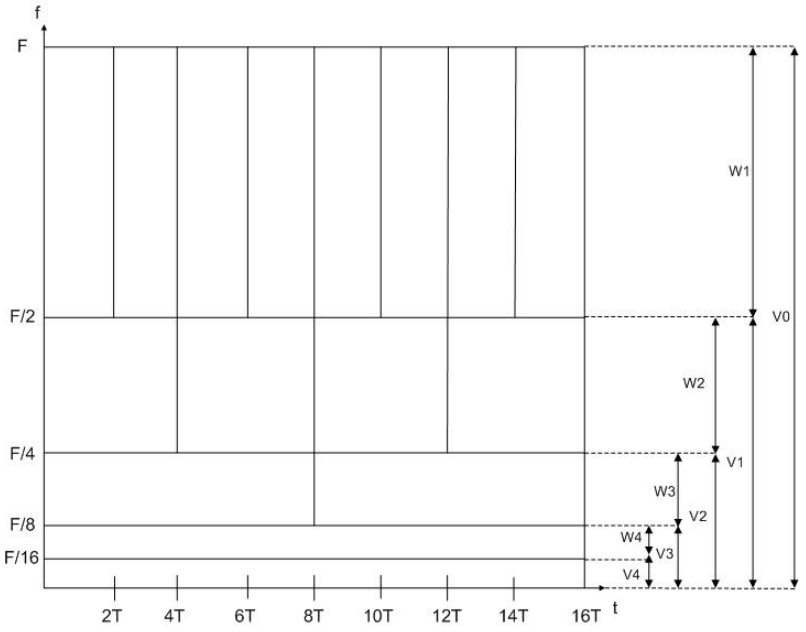
A gyakorlati életben a jelek, amiket feldolgozunk, nem rendelkeznek egy bizonyos F_0 maximális frekvenciánál nagyobb frekvenciával (mivel Nyquist tétele alapján a mintavételezett jel legmagasabb frekvenciája nem lehet nagyobb, mint az f_s mintavételi frekvencia fele), valamint rendelkeznek egy T_0 maximális időbeli szélességgel is, hiszen csak véges mennyiségű mintát tudunk tárolni. Így a feldolgozandó jelünket az idő-frekvencia síknak egy $T_0 \times F_0$ behatárolt részén ábrázolhatjuk.
A módszer folyamán a mintavételi frekvenciát diadikusan csökkenjük, ezzel a jelet a síknak egy $T_k \times F_k$ részén ábrázolhatjuk. Mivel a k növelésével a síkrészlet 2 hatványai szerint méreteződik át, a síkot a frekvenciatengely mentén megfelezzük.
A jelet a $T_0 \times F_0$ síkrészletben $y(k)$ bázisfüggvénnyekkel vett skaláris szorzata alapján

írhatjuk fel

$$x_0 = \sum_{k=1}^{\infty} y_k(t)$$

A függvényteret minden lépésben megfelezzük, két altérre bontjuk: egy aluláteresztőre (amit a textitk. felbontásnál V_k-val) és egy sáváteresztőre (amit a k. felbontásban W_k-val jelölünk), mivel az ábrázolt függvénynek csak a határok közé eső frekvenciájú komponensei találhatóak itt meg. A felbontást az 5.8. ábra szemlélteti. A k. felbontás után tehát az aluláteresztő tartományok frekvenciahatárai $\left[0\ ,\ F \cdot 2^{-k}\right]$, a sáváteresztő tartományé pedig $\left[F \cdot 2^{-k}\ ,\ F \cdot 2^{-k+1}\right]$.

Ahogy azt már említettem, a waveletek olyan függvények, amelyeknek a közepes



5.8. ábra. A függvénytér felbontása MRA segítségével

frekvenciája nem 0. Ezért a waveletek a sáváteresztő alteret feszítik ki. Az aluláteresztő tartományokat ϕ függvények, ú.n. skálafüggvények feszítik ki. Ahol a skálafüggvények a

$$\phi_{m,k} = 2^{-k} \cdot \phi(2^{-k}t - mT) \tag{5.16}$$

alakban írhatóak fel, valamint rájuk is igaz, hogy ortonormális bázisrendszert alkotnak.

Így az egyes alterekre (5.15) miatt igaz, hogy:

$$\begin{gathered} V_{k-1} = Vk \cup Wk \\ \cdots \subset V_{k+1} \subset V_k \subset V_{k-1} \cdots \subset V_0 \cdots \end{gathered} \tag{5.17}$$

Az egyes aluláteresztő és sáváteresztő tartományokra igaz továbbá az ortogonalitás:

$$\begin{gathered} V_k \cup W_k = \emptyset \text{ , azaz} \\ V_k \perp W_k \end{gathered} \tag{5.18}$$

A feldolgozandó jel egy adott altérre eső része (az altérre vett vetülete) felírható a jel adott bázisfüggvényekre eső vetületével. Ezek a bázisfüggvények a sáváteresztő altérben éppen a waveletek:

$$\begin{gathered} y_k(t) = \sum_{m=-\infty}^{\infty} d_k(m)\psi_{m,k}(t) = \text{Proj}_{W_k}\{x(t)\} \in W_k \text{ , ahol} \\ d_k(m) = W_x^{\psi}(m,k) \end{gathered} \tag{5.19}$$

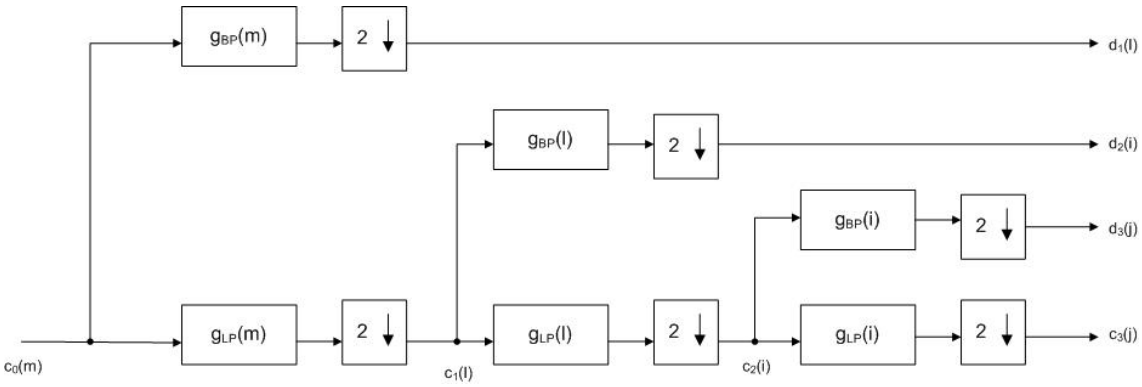
Azaz a sáváteresztő altér bázisfüggvényeire eső vetületek megegyeznek a wavelet transzformáltakkal.
Hasonlóképpen a skálafüggvények által kifeszített altérben:

$$x_k(t) = \sum_{m=-\infty}^{\infty} c_k(m)\phi_{m,k}(t) = \text{Proj}_{V_k}\{x(t)\} \in V_k \tag{5.20}$$

A wavelet függvényekkel és a skálafüggvényekkel alkotott vetületet (a függvénnyel alkotott skaláris szorzatot) némi átrendezés után az alábbi egyenletek szerint lehet kifejezni:

$$\begin{aligned} c_{k+1}(l) &= ck(2l) * g_{LP}(2l) \\ d_{k+1}(l) &= ck(2l) * g_{BP}(2l) \end{aligned} \tag{5.21}$$

Vagyis a transzformációkat FIR szűréssel és diadikus (2 hatványai szerinti) decimálással lehet megvalósítani. Minden sáv egy következő szűrőfokozattal két sávra bontható, tehát egy szűrőbank struktúra alakítható ki. A szűrők impulzusválaszai minden fokozatban megegyezik. A szűrőbank struktúráját az 5.9. ábra illusztrálja. Egy adott aluláteresztő sávban található jel visszaállításához összegeznünk kell azt



5.9. ábra. A felbontást végző szűrőbank

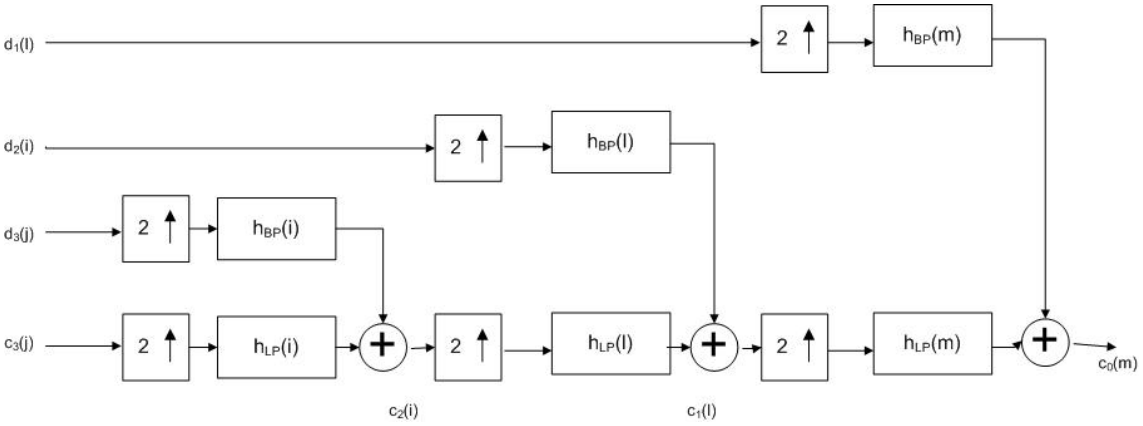
a két jelet, ami felbontott két sávban található:

$$x_k(t) = x_{k+1}(t) + y_{k+1}(t) \tag{5.22}$$

A visszaállítás szintén egy FIR szűrőbankkal végezhető:

$$c_k(m) = (c_{k+1} * h_{LP} + d_{k+1} * h_{BP}) \tag{5.23}$$

Az így a kialakítható visszaállító szűrőbank az 5.10. ábrán látható.



5.10. ábra. A visszaállítási szűrőbank

5.3. Megvalósítás MATLAB-ban

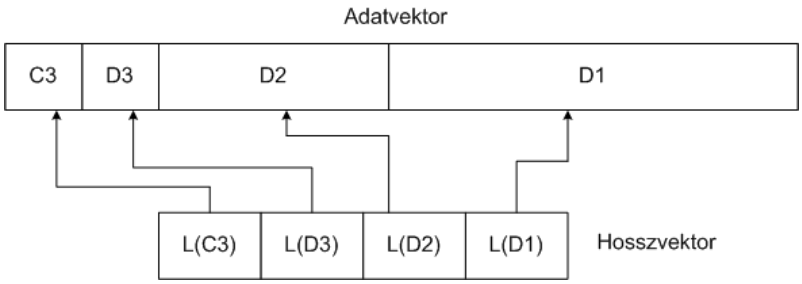
5.3.1. Bevezetés

A rendszer megvalósítását a MathWorks MATLAB programcsomaggal érdemes kezdeni, hiszen egyrészt a *Wavelet toolbox*ban olyan függvények állnak rendelkezésre, melyekkel a rendszerben felhasznált transzformációk elvégezhetőek, másrészt a tesztelés sokkal kényelmesebben végezhető. A rendszert alkotó szűrőbankot egyszerű műveletekre lebontva valósítottam meg, majd megvizsgáltam a rendszer különböző bemenetekre adott válaszát, és összehasonlítottam a referenciaként használt beépített függvényekkel.
A Wavelet toolboxban a következő beépített függvények állnak rendelkezésre:

- wavedec: a bemeneti jelet a megadott szűrőtípusokkal és megadott számú sávokra bontja, és egy speciális struktúrában eltárolja.

- waverec: a *wavedec* függvénnyel sávokra bontott jelet a létrehozott struktúrából visszaállítja.

- dwt: a megadott szűrőtípussal a jelet egy felbontási fokozatban két sávra felbontja.
- idwt: a *dwt*-vel felbontott jelet visszaállítja.
- wextend: a mintasorozat kiterjesztését végzi.
- wkeep: a mintasorozat szélén megadott számú mintát elhagy.
- wfilters: előállítja a wavelet szűrőbank impulzusválaszait.



5.11. ábra. A wavelet felbontás adatstruktúrája

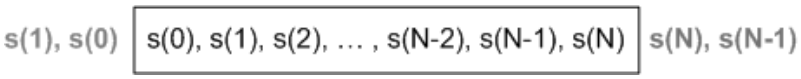
5.3.2. A tárolási struktúra

A felbontott jelet egy speciális struktúrában tároltam el, a könnyű kezelhetőség céljából. Ez a struktúra az 5.11. ábrán láthatóan két vektorból áll: egy adatvektorból, ami az egyes sávokba eső transzformált mintákat tartalmazza, valamint egy hosszvektorból, ami az adatvektor mezőinek a hosszát tartalmazza. Az utóbbira azért van szükség, hogy az adatvektor mezői könnyen indexelhetőek legyenek. Ez a struktúra megegyezik a *wavedec* és *waverec* beépített függvények által használt struktúrával, vagyis az általam megírt felbontási és visszaállítási programok akár párban is használhatóak a beépített függvényekkel, ami a további tesztelhetőséget segíti elő.
Az adatvektor N fokozatú felbontásnál $N+1$ mezőt tartalmaz, a legelső mező (C_N) mindig a legutolsó fokozat aluláteresztő szűrőjének a kimeneti mintáit tartalmazza. A következő mezők a sáváteresztő szűrők kimeneteti mintáit tartalmazzák, méghozzá balról jobbra haladva a legutolsó fokozattól a legelsőig $(D_N, D_{N-1}, \cdots, D_1)$.
Az egyes mezőkhöz tartozó minták hossza nem diadikusan változó, mivel a *conv* függvénnyel egy véges hosszú mintasort konvolválva, a mintasor határaihoz közelítve a hiányzó minták miatt számítási hibák, tranziensek lépnek fel. Ezért a mintasort a ki kell terjeszteni, azaz olyan járulékos mintákat kell generálni és a mintasorhoz hozzá venni, hogy a hasznos jel lehetőleg minél kevésbé torzuljon. A jelenség nagyobb mintaszámú jeleknél kevésbé érvényesül, viszont diadikusan csökkenő mintaszámnál ez

a hatás már néhány szűrőfokozat után jelentőssé válik, és a visszaállított jel torz lesz.

A kiterjesztés feladatotát a MATLAB-ban a *wextend* függvény látja el. Mivel ha egy a hosszúságú mintasort egy b hosszúságú mintasorral konvolválunk, az eredmény $a+b-1$ hosszú mintasor lesz. A 2-es decimálás után a minták száma $\frac{a+b-1}{2}$-re csökken. Ha a decimálandó mintasor mintáinak a száma páratlan lenne, akkor a kiterjesztés egy mintával kisebb lesz. Az így nyert hosszérték kerül a mezők hosszait leíró vektorba.

A kiterjesztett mintasorba tehát a hiányzó minták helyére mintákat kell generálni.



5.12. ábra. Szimmetrikus kiterjesztés

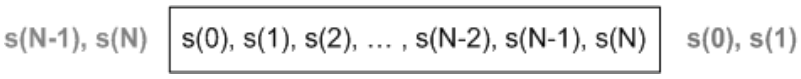
Erre több módszer létezik, (hivatkozás: MATLAB wavelet toolb.) a legegyszerűbb, ha nem terjesztjük ki a vektorokat, ami annak felel meg, mintha 0-kal terjesztettük volna ki (zero-padding). Ez a jel mesterségesen generált ugrásszerű változásait okozza, ami esetenként nagy hibákhoz vezethet.

Egy sokkal jobb módszer az ú.n. szimmetrikus kiterjesztés (symmetrization), amikor a mintasor szélén álló megfelelő számú mintát fordított sorrendben hozzávesszük a mintasor két széléhez, ahogy az az 5.12. ábrán látszik. Ezzel a módszerrel mesterségesen ugrást hozunk létre a jel első deriváltjában, viszont ez általában megfelel az alkalmazásoknak.

A lineáris kiterjesztést használó módszer (1st. order smooth padding) a jel első deriváltja szerint képez új mintákat, a konstans kiterjesztés (0 order smooth padding) pedig az első mintát és az utolsó mintát ismétli meg.

A periodikus kiterjesztés a mintasorozatot egy periodikus jel egy periódusának fogja fel, és az 5.13. ábrának megfelelően terjeszti ki.

A felsorolt módszerek közül a legszélesebb körben használt a szimmetrikus kiter-



5.13. ábra. Periodikus kiterjesztés

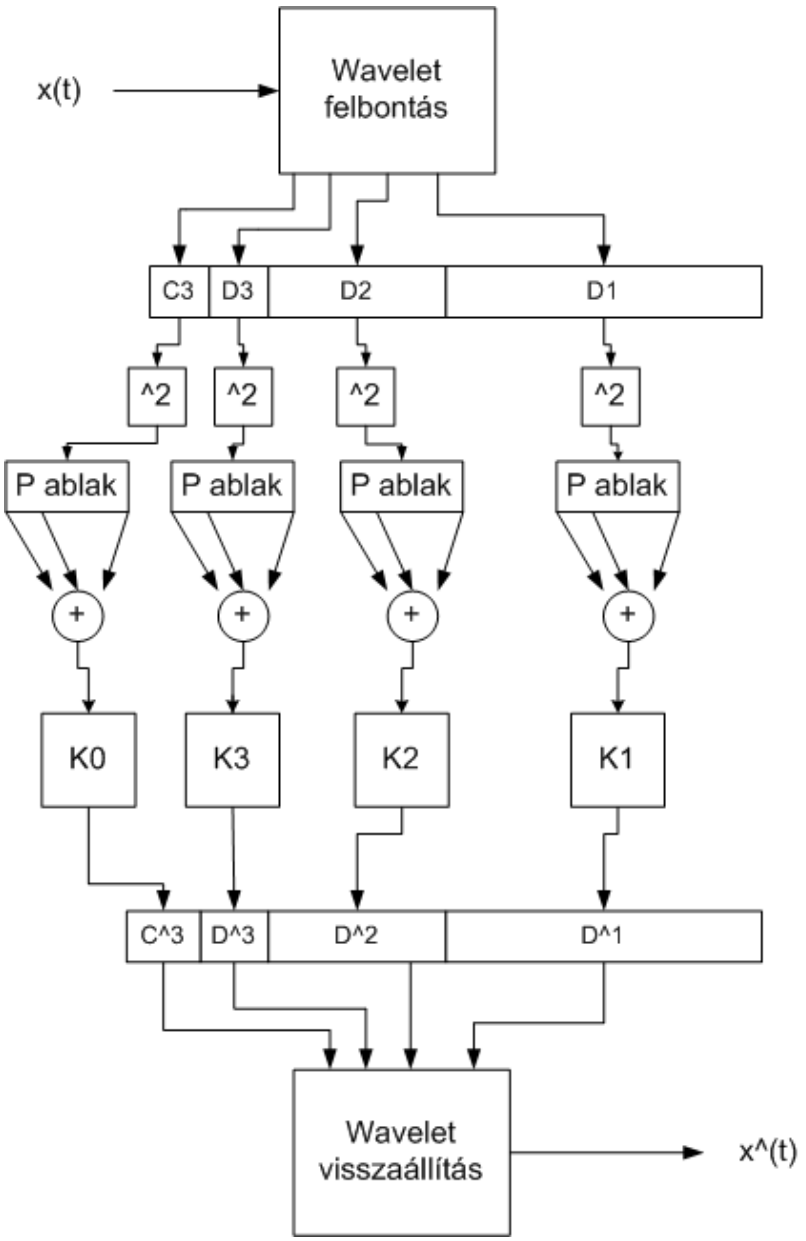
jesztés, a MATLAB beépített megoldásában is ezt a módszert használják, így én is ezt használtam.

5.3.3. A MATLAB-os megvalósítás rendszerterve

A megvalósított rendszer tehát három részből áll, blokkvázlata az 5.14. ábrán látható. A $x(t)$ szűrendő jelet először is sávokra bontjuk a wavelet transzformáció se-

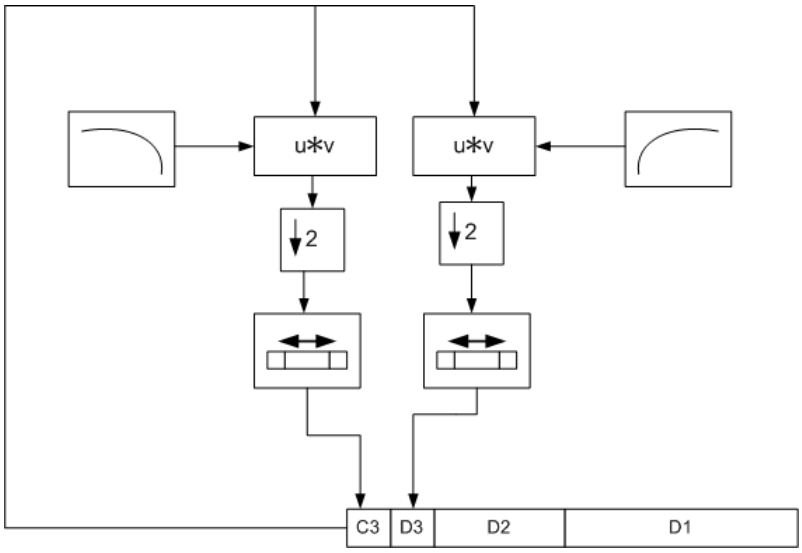
gítségével. Az így nyert sávokban képezzük a jelteljesítményt, azaz a minták négyzetösszegét, és a jelteljesítmény alapján a sávokra definiált szűrőkarakterisztikáknak megfelelően a szűrést elvégezzük, majd az $\hat{x}(t)$ szűrt jelet visszaállítjuk.
A felbontás lépései az 5.15. ábrán láthatóak. Az egyes fokozatok az előző fokozatok alacsony frekvenciás tartományába eső (C_{i-1}) mintasorozatából állítják elő a megfelelő C_i és D_i mintasorozatokat. Természetesen az első fokozat a szűrőbank $x(t)$ bemeneti jeléből indul ki.
Első lépésként a fokozat bemeneti jelét konvolváljuk a szűrő impulzusválaszokkal. A felbontáshoz két ilyen függvényt használunk, egy aluláteresztő és egy felüláteresztő impulzusválaszt. Így ugyanabból a bemeneti mintasorozatból két mintasorozatot kaptunk, melyeken a fokozaton belül minden lépésben ugyanazokat a műveleteket végezzük el. A konvolúció után a minták számát felére csökkentjük, majd az így kapott jelet a fent említett módszerrel kiterjesztjük. Ez a kiterjesztés teszi lehetővé a következő fokozatban, hogy a konvolúció kis tranzienssel mehessen végbe.
Az egyes fokozatok eredménye ezután beíródik az adatvektor megfelelő mezőjébe, és amennyiben nem értük el az utolsó fokozatot, a folyamat megismétlődik. Azért, hogy a memóriával takarékosabban bánjunk, kihasználva, hogy C_i közelítőleg C_{i-1} fele, és D_i hossza megegyezik C_i hosszával, valamint, hogy mindig csak a legutolsó fokozat aluláteresztő sávjába tartozó mintákat kell megtartanunk, ezért az adatvektorból elhagyjuk C_{i-1} mintáit, és C_i-t és D_i-t az így felszabaduló helyen tárolhatjuk el. Ha nem használnánk kiterjesztést, és így a mintaszám minden fokozat után diadikusan csökkenne, az adatvektor hossza megegyezhetne a bemeneti jel mintaszámával, és ezeknek a mintáknak a helyére elférne az összes jel, a fokozatok számától függetlenül. Mivel kiterjesztést használunk, így némileg több helyet kell az adatvektornak előzetesen lefoglalni.

A visszaállítás az előzőekben leírtakhoz hasonlóan történik, bár az 5.16. ábrán látható struktúrája már jóval bonyolultabb. A felbontási adatstruktúrában visszafele haladva az előző fokozat C_{i+1} és D_{i+1} mintáiból állítjuk vissza C_i mintáit.
A visszaállítás tulajdonképpen egy interpolálási műveletet jelent. Interpoláló szűrőként a *wfilters* által előállított visszaállítási szűrő impulzusválaszokat használhatjuk. Mivel a konvolúció elvégzésekor itt is felléphetnek tranziensek a véges mintaszám miatt, ezért a mintákat első lépésben ismét ki kell terjeszteni. Az interpolációnál jelentős mennyiségű műveletet takaríthatunk meg, ha az egyszerű interpolálásnál jelentkező 0-val feleslegesen elvégzett szorzások helyett polifázisú szűrőket alkalmazunk. Ehhez a *wfilters* által generált interpoláló szűrő impulzusválaszokat egy külön függvénnyel két-két polifázisú szűrőre bontjuk. Mivel az ismételt kiterjesztés miatt a mintaszám megnőtt, ezért, hogy a további fokozatokban is a felbontásnál kapott

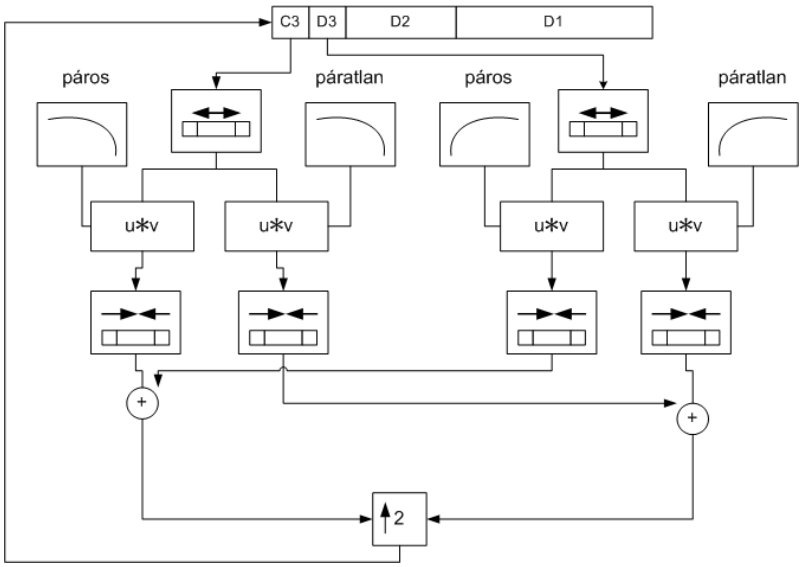


5.14. ábra. A wavelet alrendszer blokkvázlata

számú mintánk legyen, a jelből meghatározott számú szélső mintáit el kell hagynunk. Ha a mintaszám páratlan, akkor ez a "csonkítás" asszimmetrikus lesz.
Utolsó lépésként, hogy az előállítandó C_i mintáit megkapjuk, a páros sorszámú mintákhoz a páros fázisokkal előállíott minták összegét, a páratlan sorszámúakhoz pedig a páratlan fázisokkal előállított minták összegét vesszük. Ezután az adatvektorban beírhatjuk C_i-t C_{i+1} és D_{i+1} helyére.



5.15. ábra. A wavelet felbontás lépései



5.16. ábra. A wavelet visszaállítás lépései

5.4. Megvalósítás DSP-n

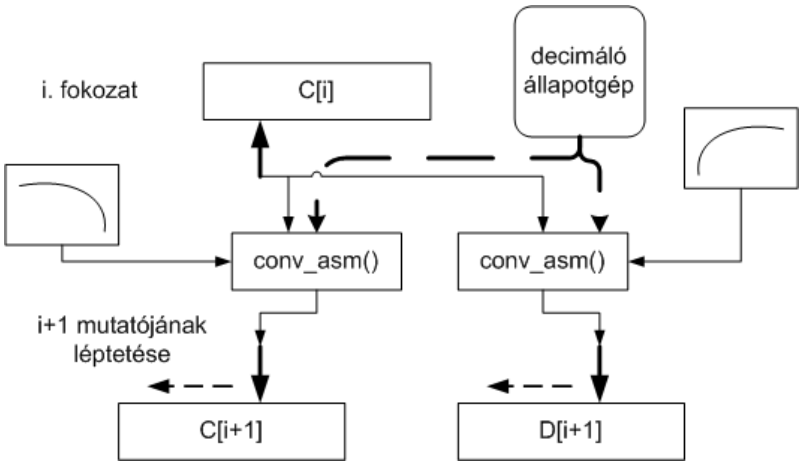
5.4.1. A DSP-s megvalósítás rendszerterve

A DSP-n történő megvalósítás struktúrálisan eltér az előzőekben ismertetett megoldástól. Ennek az oka az, hogy a megvalósított rendszer online működésű, azaz folyamatosan érkezik bemeneti jel, a kimeneten pedig a folyamatosan kell produkálni a feldolgozott jelet. Ez azt eredményezi, hogy a szűrőbank egyes fokozatai párhuzamosan működnek. Mivel a DSP memóriájában nem tudunk végtelen számú mintát tárolni (és ez felesleges is lenne), ezért célszerű mérettel rendelkező változókat foglalunk a memóriában, és ezekbe folyamatosan töltjük a legújabb bejövő mintákat, a legrégebbieket pedig elhagyjuk. Az ilyen változókat cirkuláris buffereknek hívják. A rendszerterv nagyban hasonlít az 5.14. ábrán látható MATLAB-os megvalósítás rendszertervéhez. A különbség az, hogy az adatstruktúra más. Mivel a szűrőfokozatok párhuzamosan futnak, ezért fokozatonként két cirkuláris buffert kell fenntartani a C_i aluláteresztő és D_i sáváteresztő szűrőkimeneteknek. Továbbá sávonként szükség van egy teljesítményablakra is, amit szintén egy cirkuláris buffer valósít meg, ahova beíródnak az adott sávban található jel mintáinak négyzetei. Ezért a DSP-s megoldás több memóriát, és több műveletet igényel, mint a MATLAB-ban megvalósított. A DSP-re letöltött szűrőkarakterisztikák a *MATLAB Wavelet Toolbox wfilters* függvényével generálhatóak, majd egy speciális függvénnyel átkódolhatóak a DSP-n használt fixpontos *fract16* formátumba.

5.4.2. A wavelet felbontás

A módszer elvi lépései megegyeznek a MATLAB-os megvalósításnál bemutatotakkal. A DSP kártyán található AD-átalakító 48 kHz-es mintavételi frekvenciával állítja elő a mintákat. A mintavételezés befejeztével az új mintákat egy változóban helyezi el, és egy megszakítást kér a processzortól. Ezek után bármilyen feldolgozási rutin a megszakítási rutinból kell, hogy elinduljon.
Mivel az online jelfeldolgozásnál nagyon fontos a sebesség, különösen a DSP-nél, hiszen itt minden feldolgozási művelet el kell, hogy végződjön a mintavételi periódus belül (különben a feldolgozási rutint önmaga kellene, hogy megszakítsa), ezért itt további egyszerűsítési lehetőségeket kell keresni. A konvolúció után következő decimálásnál minden második mintát elhagyjuk, azaz minden második pozícióból elvégzett konvolúció felesleges volt. Ezért jelentős mennyiségű műveletet takaríthatunk meg, ha csak minden második konvolúciót végezzük el. Ehhez felhasználhatunk például egy két állapotú állapotgépet (amelynek minden fokozathoz tartozik egy állapota), aminek az egyik állapotában elvégződik a konvolúció, a másik állapotában

pedig nem. Az aktuális jelet kijelölő mutató viszont minden ütemben lép egyet, tehát így minden második pozícióból decimálunk.
A megvalósított struktúrában (lásd az 5.17. ábrát) minden fokozat a következő fokozat buffereit írja. Először kiolvassa az előző fokozat által feltöltött cirkuláris bufferekt, majd ha a az adott fokozatban a decimáló állapotgép állapota 1 – elvégzi a konvolúciókat, eggyel lépteti a következő fokozat mutatóját a cirkuláris bufferben. Az így kapott adatokat beírja a következő fokozat buffereibe a pointer által mutatott pozícióba és átbillenti a fokozat állapotgépét. Így a következő fokozat csak minden második beírt jel után végzi el a konvolúciót. A műveletek felét így megspóroltuk, viszont a módszer még nem tökéletes.
A nagyobb sorszámú fokozatokban a decimálás aránya már nagyon nagy (a fokozatok sorszámával exponenciális arányban nő), ezért a decimáló állapotgép sok ütemen keresztül tartózkodna az egyes állapotokban. Ez azt jelentené, hogy 1-es állapotban, amikor az adott fokozatban a konvolúció elvégződik, egyrészt a következő fokozat mutatója folyamatosan növelődne, másrészt ugyanaz a konvolúció többször végződne el. Ez hibás működéshez vezethetne, ezért egy adott fokozathoz tartozó állapotgépnek csak egy lépés erejéig szabad tartózkodnia az egyes állapotban.



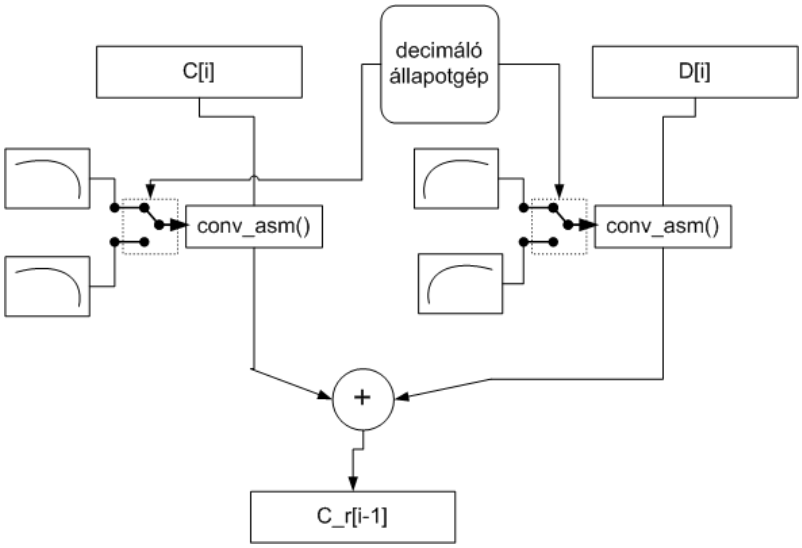
5.17. ábra. Wavelet felbontás DSP-n

További eltérés a MATLAB-os megvalósítástól, hogy a cirkuláris bufferek hossza minden fokozatnál ugyanakkora, méghozzá a szűrők tapszámával megegyező hosszúságú. Nem érdemes hosszabb bufferekkel dolgozni, hiszen a tapszámnál régebbi mintákkal már nem végzünk műveleteket, előbb-utóbb úgyis kifutnak a bufferből. Szintén nem kell a bufferekben található mintákat kiterjeszteni, hiszen, mivel a futás során a legrégebbi minták helyett folyamatosan új minták érkeznek, csak a bekapcsoláskor keletkeznek tranziensek, amik véges idő alatt kifutnak a bufferekből.

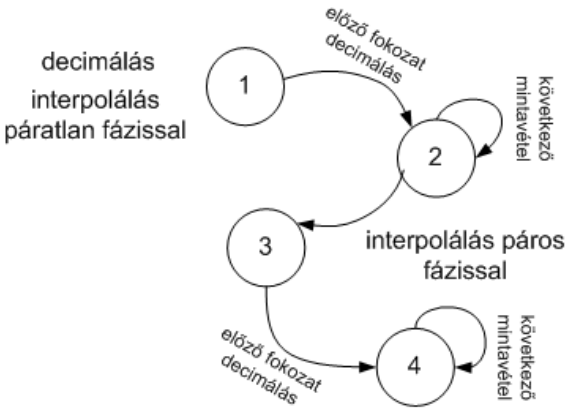
5.4.3. A wavelet visszaállítás

A visszaállításnál – ugyanúgy, mint a MATLAB-os megvalósításnál – polifázisú szűrőket használunk. A visszaállítást az 5.17. ábra szemlélteti. A szűrésnél a fázisok kiválasztásához a felbontásnál bemutatott decimáló állapotgép adott fokozathoz tartozó állapotait lehet felhasználni. A fent említett decimáló állapotgépnek két állapota volt, és a magasabb sorszámú fokozatok az idő nagy részében az egyikben találhatóak (amelyikben nem történik decimálás). Viszont a polifázisú szűrőnél is biztosítani kell, hogy ne történjen felesleges konvolúció, ezért még két állapotot célszerű bevezetni.
Az állapotgép állapotgráfja az 5.19. ábrán látható. Az adott fokozaton az 1. és 3. állapotban történik műveletvégzés, ezekbe az állapotokba az előző felbontási fokozat billenti be (amikor konvolúció történik). Egyébként a következő mintavételnél az összes fokozathoz tartozó állapot kiértékelődik, és az 1. vagy 3. állapotból 2. és 4. állapotba billen, majd az ezt követő összes mintavételnél ezekbe az állapotokba ragad, mindaddig, amíg az előző fokozat be nem lépteti a következőbe.
Tehát összefoglalásként egy adott felbontási fokozat esetében az állapotgép 1. állapotában történik a felbontási konvolúció. Az ezzel megegyező sorszámú visszaállítási fokozatban az állapotgép 1. állapotában történik a páratlan szűrőfázissal való interpoláció, a páros szűrőfázissal történő interpoláció pedig a 3. állapotban. A 2. és 4. állapotban nem történik semmi az adott szűrőfokozatban. Mivel a szűrőfokozatok párhuzamosan futnak, ezért a visszaállításnál fokozatonként egy járulékos cirkuláris buffert kell bevezetnünk. A visszaállításkor ugyanis a két legalacsonyabb frekvenciájú sávból állítunk elő egy sávot, viszont mivel a szűrőkarakterisztikák módosították az ebben található jelet, nem írhatjuk felül a felbontási szűrők által használt buffert.

A zajszűrési karakterisztikák szintén a MATLAB-ban generálhatóak, majd letölthetőek a DSP-re.



5.18. ábra. Wavelet visszaállítás DSP-n



5.19. ábra. A decimáló állapotgép állapotgráfja

6. fejezet

Az adaptív vonaljavító rendszer

6.1. Adaptív szűrés

A megvalósított zajszűrő rendszer egyik alrendszere az adaptív vonaljavító, amivel a periodikus zavarjeleket (pl. búgást) el lehet távolítani pl. egy beszédfelvételről. Az adaptív vonaljavító az ú.n. LMS algoritmuson alapszik. A fejezet során áttekintem ennek az LMS algoritmusnak az elméleti hátterét, majd bemutatom az adaptív vonaljavító rendszert, valamint annak megvalósítási módját.

6.2. Elméleti áttekintés

Az adaptív algoritmusok [7] [9] feladata egy rendszerről előzetes ismeretek alapján alkotott modell paramétereinek a becslése a rendszer kimeneti jeleinek alapján. Ezt modellillesztésnek nevezik. A cél a modell paramétereinek a meghatározása, és a paraméterek változásának a valósidejű követése. Esetünkben nem követelmény a paraméterek pontos meghatározása, amihez nagy mennyiségű adatot kellene gyűjteni. Ez esetenként egy rendkívül időigényes feladat lehet. Az adaptív szűrők esetében általában a sebesség sokkal fontosabb követelmény, hiszen teljesülnie kell a valósidejűségnek. Ezért általában egyszerűbb modelleket alkalmazunk, valamint a becsült paraméterek is pontatlanabbak.

6.2.1. Wiener-szűrők

A modellillesztés egy speciális feladata a regressziószámítás, aminek segítségével bizonyos változók közötti determinisztikus kapcsolatok találhatóak meg. Esetünkben ez a modellezendő rendszer be- és kimenete. Mivel a rendszer kimenetei, valamint a méréseink zajjal terheltek, a változók közötti függvénykapcsolat nemcsak determinisztikus, hanem sztochasztikus összetevőket is tartalmaz. Mivel a modellünk kizárólag determinisztikus kapcsolatot modellez, ezért ennek a függvénykapcsolatnak

a kialakítása általában bizonyos szabad paraméterek megfelelően megválasztott értékre történő beállítását jelenti.
A paraméterek hangolásához általában egy költségfüggvényt szokás definiálni, ami a modellünk illesztetlenségének valamilyen definiált mértékét adja meg. A paraméterkészletünket tehát olyan módon kell változtatnunk, hogy ez a költségfüggvény minimális legyen. Ennek a költségfüggvénynek általában megfelelő megfogalmazása a kimenetek négyzetes különbségének valamilyen lineáris függvénye, azaz:

$$\epsilon = E\left\{(\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}})\right\} \tag{6.1}$$

Ez a definíció, amellett, hogy matematikailag egyszerűen kezelhető, fizikailag a hibajel teljesítményének minimalizálását jelenti. Az olyan szűrőket, amiket az így definiált költségfüggvény minimalizálásával hangolunk, *Wiener-szűrő*nek nevezzük.
A paraméterhangolás folyamán persze nemcsak a vizsgált jelek pillanatnyi értékét használjuk fel, hanem ezeknek a jeleknek egyéb statisztikai jellemzőit is. Az illesztendő modell bemenete és kimenete között a következő összefüggés érvényes:

$$\hat{\mathbf{y}} = \hat{\mathbf{g}}(\mathbf{u}) = \hat{\mathbf{g}}(\mathbf{W}, \mathbf{u}) \tag{6.2}$$

ahol $\mathbf{u}$ a bemenőjel-vektor, $\mathbf{W}$ a hangolható paraméterek mátrixa, $\hat{\mathbf{y}}$ pedig a modell kimeneteinek a vektora. A modell dinamikus, és esetleg nemlineáris részét célszerű rögzíteni ($f(\mathbf{u}) = \mathbf{x}$) és különválasztani a lineáris, adaptálható résztől ($\mathbf{W}$):

$$\hat{\mathbf{y}} = \hat{\mathbf{g}}(\mathbf{W}, \mathbf{u}) = \mathbf{W}^T f(\mathbf{u}) = \mathbf{W}^T x \tag{6.3}$$

ahol $\mathbf{x} = f(\mathbf{u})$ a modell rögzített része, amely az $\mathbf{u}$ bemenő jel-vektorból előállítja a az $\mathbf{x}$ regressziós vektort, amely ezután az adaptálandó rendszer bemenetét képezi. Ezek után, a sztochasztikus folyamatokról az időfüggvényekre áttérve, és egy kimenetet és egy bemenetet feltételezve a kimenet a következőképpen adódik:

$$\hat{y}(n) = \mathbf{w}^T(n)\mathbf{x}(n) = \sum_{i=1}^{M-1} w_i(n)x_i(n) \tag{6.4}$$

ahol M a $\mathbf{w}$ és a $\mathbf{x}$ vektorok hossza, a mátrix-vektor szorzat pedig a vektorok skaláris szorzatává egyszerűsödik, és (6.4) alapján számítható.

A szétválasztás eredményeképp az adaptálható rész paramétereiben lineáris, és a paraméterek megváltoztatása által okozott tranziens véges idő alatt kifut a rendszerből. A bemenet és kimenet már felírt összefüggései alapján a kritériumfüggvény:

$$\epsilon(n) = E\left\{(y(n) - \hat{y}(n))^2\right\} = E\left\{(y(n) - \mathbf{w}^T\mathbf{x}(n))^2\right\} = \\ E\left\{y^2(n)\right\} - 2\underbrace{E\left\{y(n)\mathbf{x}^T(n)\right\}}_{\mathbf{p_T}}\mathbf{w} + \mathbf{w}^T\underbrace{E\left\{\mathbf{x}(n)\mathbf{x}^T(n)\right\}}_{\mathbf{R}}\mathbf{w} \tag{6.5}$$

ahol $\mathbf{p}^T$ a kimenet és a regressziós vektor közötti keresztkorrelációs vektor, illetve $\mathbf{R}$ a bemeneti vektor autokorrelációs mátrixa. Ezek szerint a kritériumfüggvény egy $M+1$ dimenziós térben elhelyezkedő paraboloidot ír le. A kritériumfüggvény minimumát az N dimenziós $\mathbf{w}$ vektor szerinti differenciálás adja meg, a következő képpen:

$$\frac{\partial \epsilon}{\partial \mathbf{w}} = -2\mathbf{p} + 2\mathbf{R}\mathbf{w} = 2(\mathbf{R}\mathbf{w} - \mathbf{p}) = \mathbf{0} \tag{6.6}$$

Ennek a megoldása a *Wiener-Hopf egyenlet*, mely megadja az optimum helyét a $\mathbf{w}$ szűrőegyütthatók terében:

$$\mathbf{w}_{opt} = \mathbf{R}^{-1}\mathbf{p} \tag{6.7}$$

A szűrőegyütthatók adaptálása ezek alapján egy szélsőérték-keresési feladat. A módszer hibája viszont, hogy a be- és kimeneti jelek statisztikai tulajdonságairól *a priori* ismeretekkel kell rendelkezzünk, a korrelációs vektorok és mátrixok meghatározásához.

6.2.2. Az LMS-algoritmus

Ha a jelek sztochasztikus jellemzőiről egyáltalán nem, vagy csak kevés *a priori* ismerettel rendelkezünk, akkor az adaptáció a pillanatnyi költségfüggvény alapján iteratív módon végezhető el. Az adaptáció a legmeredekebb lejtő módszer szerint történik. Ez azt jelenti, hogy minden iterációs lépésben meg kell határozni, hogy mekkora a hibafelület gradiense, és a legnagyobb negatív gradiens irányába mozdulva módosítjuk a paraméterkészletet, egy bátorsági tényező által megszabott mértékben. Ez az algoritmus az LMS (Least Mean Squares). A pillanatnyi adatok alapján számolt $\hat{\mathbf{R}}$ autokorrelációs mátrix és $\hat{\mathbf{p}}^T$ keresztkorrelációs vektor tehát a következőképpen alakul:

$$\begin{aligned} \hat{\mathbf{R}} &= \mathbf{x}(n)\mathbf{x}^T(n) \\ \hat{\mathbf{p}}^T &= y(n)\mathbf{x}^T(n) \end{aligned} \tag{6.8}$$

A pillanatnyi hibán alapuló becslés azonban nem feltétlenül a negatív gradiens irányába módosítja a paraméterkészletet, viszont hosszabb időtávon ezek a módosítások kiátlagolódnak, és a pillanatnyi hiba a hibafelület minimuma környékén található, a hibagradiens nulla körül mozog, viszont soha nem éri el a nullát, hiszen a jel sztochasztikus jellemzőinek a becslése pontatlan. Az LMS algoritmus blokkvázlata a 6.1 ábrán látható. Az LMS algoritmusban a paraméterkészlet adaptálása a következő

egyenletek alapján történik:

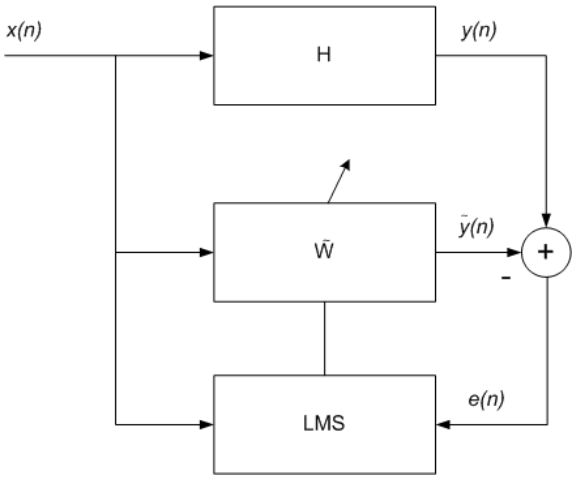
$$e(n) = y(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n) \tag{6.9}$$

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + 2\mu e(n)\mathbf{x}(n) \tag{6.10}$$

ahol a $\hat{\mathbf{w}}(n)$ vektor egy FIR szűrő n. ütemben rendelkezésre álló hangolható együtthatókészlete, a μ a bátorsági tényező, az $y(n)$ modellezendő rendszer kimenete, az $\mathbf{x}(n)$ a modellezendő rendszer és egyben a szűrő bemenete, $e(n)$ pedig a különbségükből képzett hibajel. A μ értéke befolyásolja egyrészt az adaptáció sebességét, túl nagy értéke viszont a rendszer instabilitását okozhatja. Az LMS algoritmus tehát az $\mathbf{R}$ autokorrelációs mátrix, valamint a $\mathbf{p}^T$ keresztkorrelációs vektor ismerete nélkül képes a $\hat{\mathbf{w}}$ szűrőegyüttható kkal megközelíteni a $\mathbf{w}_{opt}$ optimális szűrőegyüttható készletet.

6.2.3. Az adaptív vonaljavító rendszer

Az adaptív vonaljavító (ALE - Adaptive Line Enhancer) egy olyan struktúra, ami egy keskenysávú jelet (amiről kevés a priori ismerettel rendelkezünk) szétválasztunk egy szélessávú jeltől, ideális esetben egységnyi erősítéssel. Ehhez egyedül azt a követelményt kell teljesíteni, hogy a keskenysávú jel megfelelően keskeny sávú legyen. Az, hogy melyik jelet értelmezzük információt hordozó hasznos jelként, és melyiket zajként, az az alkalmazástól függ. Például szonároknál a hasznos jel keskenysávú, a zaj szélessávú, de egy beszédről készített felvétel esetében, ami búgást tartalmaz, a szélessávú jel hordozza az információt, a keskenysávú búgás a zaj. Az ALE struktúrája a 6.2. ábrán található. A bemeneti $x(n)$ jel a keskenysávú $u(n)$ és a szélessávú

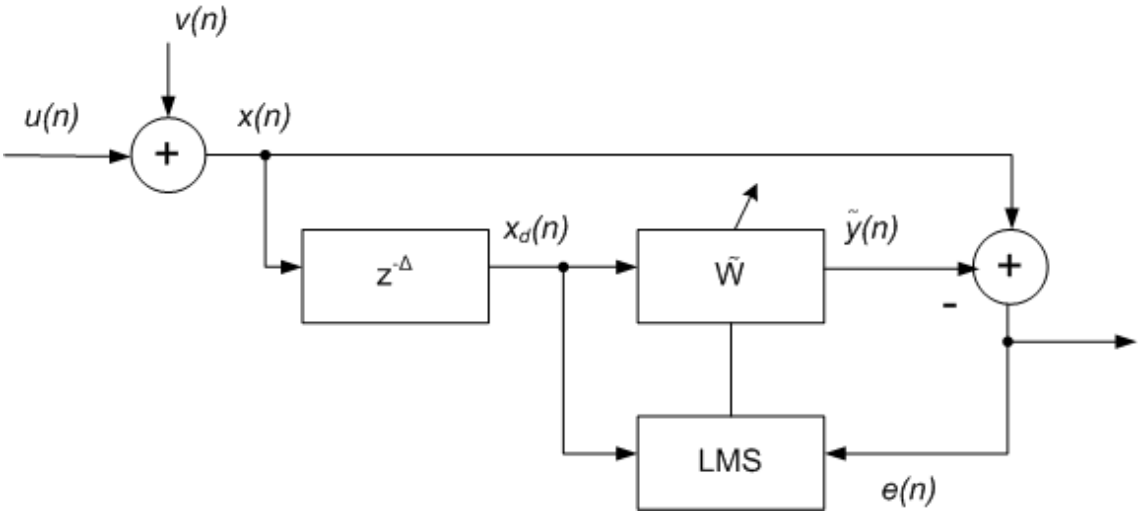


6.1. ábra. Az LMS struktúrája

$v(n)$ jelből tevődik össze. A prediktor szűrő hangolását az LMS algoritmus végzi. A prediktor szűrő és az LMS algoritmus bemenetére az $x_d(n)$ – Δ-nel késleltetett jel jut. A prediktor kimenetén keletkező $\hat{y}(n)$ jel és a késleltetés nélküli $x(n)$ jel különbségéből keletkezik a $e(n)$. A késleltetés mértékét úgy kell meghatározni, hogy a bemeneti jel $v(n)$ szélessávú komponense, valamint annak a $v(n-\Delta)\cdots v(n-\Delta-M)$ késleltetett változata (M a prediktor FIR szűrő együtthatóinak számát jelenti) korrelálatlan legyen. Enélkül a feltétel nélkül előfordulhatna olyan eset is, hogy a szűrő olyan irányba hangolódik, hogy a szélessávú $v(n)$ jelet is átengedje a hiba jobb minimalizálásának érdekében. Annak megfelően, hogy az alkalmazás szempontjából az $u(n)$ vagy a $v(n)$ hordozza az információt, az $\hat{y}(n)$-t vagy az $e(n)$-t csatoljuk ki. Ebben az alkalmazásban az $e(n)$-t használjuk kimenetnek, mivel a keskenysávú jelet értelmezzük zajként.

6.3. Megvalósítás DSP-n

A DSP-n megvalósított struktúra a 6.3. ábrán látható. A működéshez szükséges egy bemeneti zajos jel, valamint ennek egy késleltetett változata. Minden új bemeneti mintát egy cirkuláris bufferbe írunk. A Δn késleltetést egy késleltetési mutató létrehozásával valósítottam meg. Ez a mutató az aktuális mintát kijelölő mutató után halad n mintával szintén cirkulárisan. A cirkuláris buffer megvalósításának egyik C-nyelvben általában használatos módja a modulo operátor (%). Ezt a műveletet ténylegesen maradékos osztással megvalósítani viszont nagyon számításigényes lenne, ezért a fordítóban be kell állítani (*force circbuf*), hogy egy adott formában modulo operátorral megadott címzés esetében felismerje, hogy egy cirkuláris bufferről van szó, és optimálisan fordítsa le a kódot. Egy N elemű cirkuláris buffert



6.2. ábra. Az ALE struktúrája

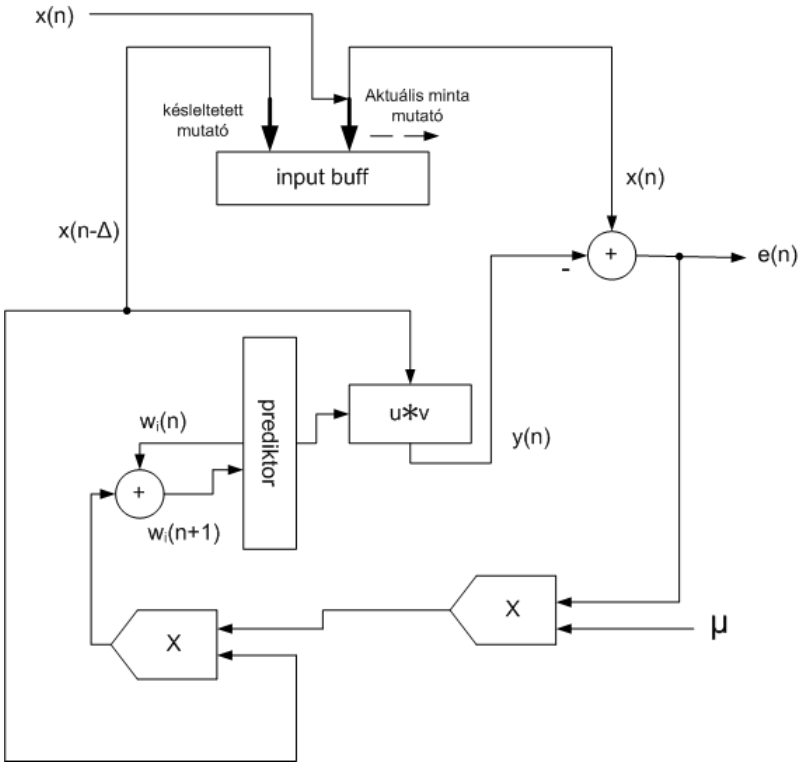
például a következő kódrészlettel lehet létrehozni(ahol a pointer egy *int* változó):

```
buffer[pointer % N];
pointer++;
```

A késleltetési pointer létrehozásához viszont nem elég kivonnunk a késleltetési mintaszámot a pointerből, hiszen, ha a pointer a buffer egy megfelően alacsony pozíciójára mutat, akkor kicímezhetünk a bufferből. Ehelyett inkább hozzáadjuk a pointerhez a N bufferhossz, és a késleltetés mintaszámának összegét, majd ennek vesszük a N modulóját. Hogyha definiálunk egy *DELAY* makrót a késleltetési mintaszámra, akkor a következőképpen néz ki a cirkuláris buffer definíciója.

```
buffer[(pointer + N - DELAY) % N];
pointer++;
```

Ahogy a wavelet szűrőbanknál, itt is meg kell fontolni, hogy milyen lehetőségek adódnak a sebesség növelésére. A prediktor szűrő minden egyes tapjét felül kell írni az LMS algoritmusban minden egyes lépésben. Ha a prediktor tapjeit (6.10) . képlet közvetlen megvalósításával frissítjük, akkor az $e(n)$ hibajel és a μ bátorsági tényező szorzatát miden szűrőtapnél kiszámoljuk, holott az értékük egy lépésben állandó. Ezért ezt célszerű az adott lépésben kiszámolni, és egy változóban eltárolni.



6.3. ábra. Az ALE megvalósítása DSP-n

7. fejezet

Zajszűrés medián szűrővel

7.1. Bevezetés

A medián szűrő az egyik legelterjedtebb nemlineáris szűrő. Impulzusszerű zajok szűrésére, kiugró minták eltávolítására használják.
A medián szűrő egy zajos jel egy mintát kicserél az adott mintaszámú környezetében a minták egy nagyság szerint rendezett sorozatában a középsőre (a sorozat mediánjára). Vagyis, egy N hosszúságú mediánszűrő esetében, ha $\tilde{x}$ a szűrő kimenete, és ha $x_1, x_2, \cdots, x_N$ egy nagyság szerint rendezett mintasorozat, akkor:

$$\tilde{x} = \begin{cases} x_{\frac{N+1}{2}}, & \text{ha } N \text{ páratlan} \\ \frac{1}{2}(x_{\frac{N}{2}} + x_{\frac{N}{2}+1}), & \text{ha } N \text{ páros} \end{cases}$$

A mediánszűrő ablakhosszának megállapításakor figyelembe kell venni, hogy a hasznos jel is tartalmazhat rövid idejű jelszintváltásokat, amit ha a mediánszűrő kiszűr, torz jelet, információveszteséget eredményez.

7.2. Megvalósítás

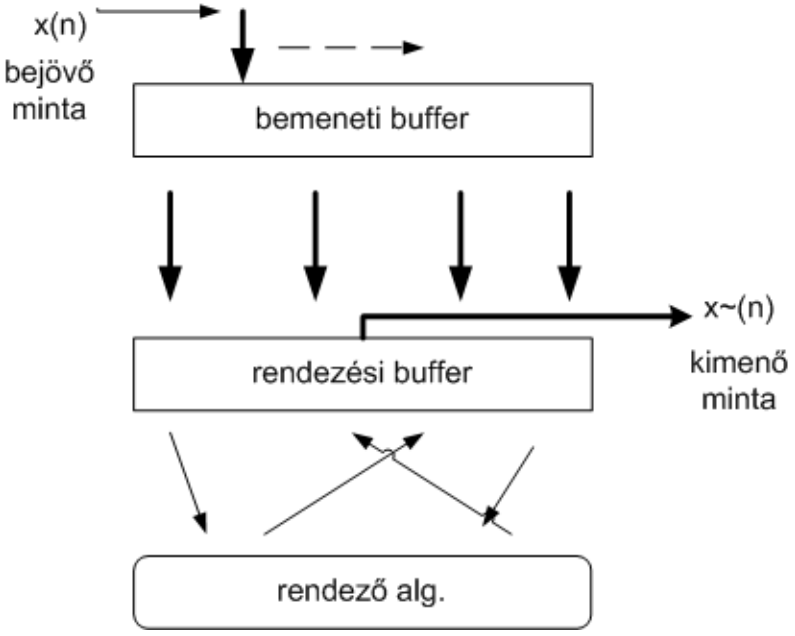
A DSP-ben megvalósított struktúra a 7.1. ábrán látható. Az új bejövő mintát egy cirkuláris bufferbe írjuk. Ezek után a buffert lemásoljuk egy rendezési bufferbe, ahol egy rendező algoritmus növekvő sorrendbe rendezi a mintákat. Azért van szükség külön egy rendezési bufferre, mert különben nem tudnánk a legrégebbi minta helyére beírni az új mintát.

A rendezéshez kétféle algoritmust próbáltam ki:

- quicksort

- buborékrendezés

A quicksort alkalmazása kézenfekvő, hiszen a *qsort()* szabványos C függvény, és mint ilyen a DSP C fordítójának függvénykönyvtára is tartalmazza [3]. A függvénykönyvtárban található függvények már eleve lefordítottan találhatóak meg, ami azt jelenti, hogy az Analog Devices fejlesztőmérnökei ügyeltek az optimális megvalósításra. A függvény igényel egy a felhasználó által megírt összehasonlító függvényt, amit a *qsort()* meghív minden lépésben.
A buborékrendezés egy ismert és könnyen megvalósítható algoritmus, viszont lépésszáma legrosszabb esetben négyzetesen függ a minták számától.
Az algoritmus a rendezési bufferben az elejétől indulva minden pozícióban beolvassa az ott található mintát és a következőt, majd ha a következő kisebb, akkor felcseréli őket, majd a következő pozícióra lép. Ha elérte a buffer végét, akkor a buffer elejéről újra kezdi, kivéve, ha az előző végighaladás alkalmával nem kellett az elemeket felcserélnie. Ez azt jelenti, hogy egy rendezett buffer esetében is le kell futnia egyszer.
A megvalósított medián szűrő hossza paraméterezhető. Abban az esetben, ha a minták száma páros, akkor az átlagszámítás helyett az $\frac{N}{2}$. pozíción található elem lesz a szűrő kimenete, ezzel egyrészt megspórolva a az összeadás és az osztás műveletét, másrészt a kimeneten található minta mindig olyan lesz, ami a bemeneti minták között is található.
A bufferek átmásolására is adódik megoldás a C szabványos függvénykönyvtárában, amiről szintén feltételezhető, hogy a lehető leggyorsabban oldottak meg a C fordító fejlesztői. Ez a függvény a *memcpy()*, melynek a definíciója a *string.h* header fájlban



7.1. ábra. A medián szűrő megvalósítása

található.

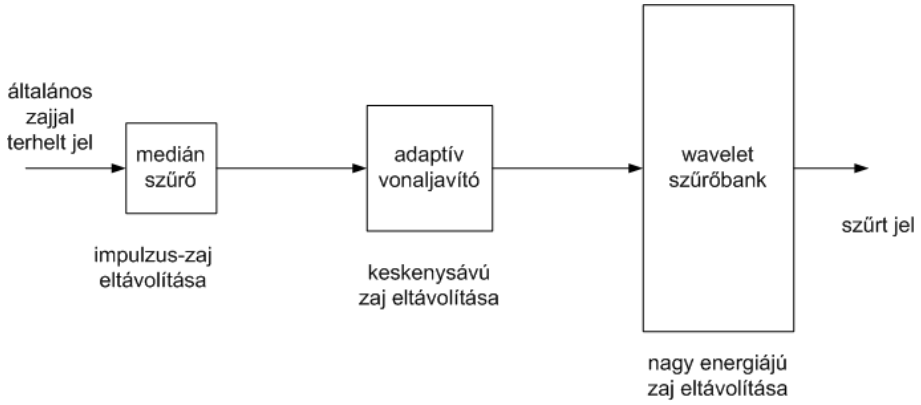
7.3. Értékelés

Mind a két rendezési algoritmus viszonylag lassúnak bizonyult a DSP-n, a buborékrendezés a magas lépésszáma miatt, a *qsort* algoritmus pedig a működéshez szükséges sok függvényhívás miatt. Az összehasonlításhoz különálló függvényt kell írni, ami eleve minden lépésben egy függvényhívást jelent, azonkívül a quick sort algoritmus rekurzív, tehát saját magát hívja. A függvényhívás regiszterek stackbe mentésével, memóriából való feltöltéssel, majd visszatéréskor a stackből való feltöltéssel jár, tehát az online jelfeldolgozási algoritmusok sebességigényéhez képest lassú művelet.

8. fejezet

A részrendszerek összekapcsolása

A három zajtípus, amiket a dolgozatom során vizsgáltam, a gyakorlatban együtt lép fel a zajos jelekben. Ezért célszerű a három alrendszert összekapcsolni, és egyidőben működtetni.
Az alrendszerek összekapcsolási sorrendjének kialakításánál figyelembe kell venni, hogy milyen hatással vannak egymás működésére, illetve hogy mennyire torzítják a következő fokozat bemeneti jelét.
Az egyik legfontosabb szempont a rendszer stabilitása. A medián szűrő és a wavelet szűrőbank bármilyen bemenetre megtartja a stabilitását, viszont az adaptív vonaljavító rendszerre jellemző az instabilitásra való hajlam bizonyos körülmények között. Ha a bemenetre jutó jelben impulzusszerű zaj van, nagy ugrásokkal, akkor a stabilitás megtartásának érdekében csak kis bátorsági tényezővel tudnánk működtetni. Ez viszont alapvetően lassú beállást eredményez, ami a szűrés minőségét csökkenti. Ezért célszerű a medián szűrőt az adaptív alrendszer elé helyezni, és így a bátorsági tényező kellő mértékben megnövelhető. Az adaptív vonaljavító rendszer eltávolítja a keskenysávú zavarjelet, de a nagyteljesítményű, szélessávú zajjal terhelt jelet nem módosítja. Ezért a vonaljavító kimenetén keletkezett jel megtisztítható a wavelet szűrőbank segítségével. A szűrőbankot azért érdemes utolsó fokozatként üzemeltetni, mert az impulzus-szerű zaj esetleg meghaladhatná a küszöbértékeket az egyes sávokban, és így a sávban jelen lévő egyéb zajok (pl. sistergés) csillapítás nélkül áthaladhatnának.
A fenti megfontolások alapján létrehozott struktúra a 8.1. ábrán látható. A DSP-n való futtatáskor figyelembe kell venni, hogy a három alrendszer esetenként már külön-külön túlságosan megterhelheti a DSP-t. Az általam tanulmányozott diplomatervben a szűrőbank struktúra eleve úgy lett megtervezve, hogy az teljesen kihasználja a DSP adta lehetőségeket. Ezért, ahhoz, hogy a teljes rendszer a DSP számítási képességein belül maradjon, kompromisszumokra kényszerülünk.



8.1. ábra. Az összekapcsolt alrendszerekből alkotott teljes rendszerterv

9. fejezet

Mérési eredmények

Szakdolgozatom utolsó fázisaként az implementált rendszereket különböző méréseknek vetettem alá. Mivel az egyes alrendszereket mind MATLAB-ban, mind DSP-n megvalósítottam, ezért mindkettőt méréseknek vetettem alá. A MATLAB-ban megvalósított rendszert egyben mérem be. A szakdolgozat leadásának időpontjakor a DSP-n megvalósított wavelet szűrőbank hibásan működik, ezért a DSP-n implementált teljes szűrőrendszert nem tudtam bemérni, csak az egyes alrendszereket egyenként. A mérésnek két módja van. Az egyik a megfelelően választott vizsgálójelek alkalmazása, melyek akár műszeren könnyen megfigyelhetővé, és látványossá teszik a rendszer működését. Másfelől, mivel a szűrőrendszer célja audiojelek zajcsökkentése, ezért érdemes megvizsgálni a rendszer működését olyan hanganyagokkal, amihez mesterségesen adtunk hozzá a vizsgált zajmodelleknek megfelelő zajt. A szűrés után meg tudjuk vizsgálni, hogy milyen volt az eredeti zajmentes jel, majd pedig a rendszer kimenetén keletkezett zaj, így szubjektíven tudjuk megítélni a szűrés sikerességét, valamint a hanganyag esetleges torzulását az eredetihez képest. A speciális vizsgálójelekkel végzett méréseket ábrák formájában jelenítem meg, a hanganyagokat, pedig a szakdolgozat DVD mellékletén helyezem el.

9.1. Mérési eredmények MATLAB-ban

9.1.1. Medián szűrő

A medián szűrő MATLAB-os megvalósításának működését láthatjuk a 9.1. ábrán. Az ablakszélesség 10 minta volt. Látható, hogy egy kiemelkedés keletkezik a szűrt jelen, de a impulzus-szerű zaj által keletkezett tüske eltűnik.

9.1.2. Adaptív vonaljavító

A 9.2. ábrán az adaptív vonaljavító MATLAB-os megvalósításának szemléltetése látható. A rendszert szinuszjel és fehér zaj összegével vizsgáltam. A felső két jel egy zajmentes szinuszjel és a prediktor által becsült szinuszjel. A hibajel a rendszer kimenete, látható, hogy kicsatolja a fehér zajt.

9.2. A teljes rendszer

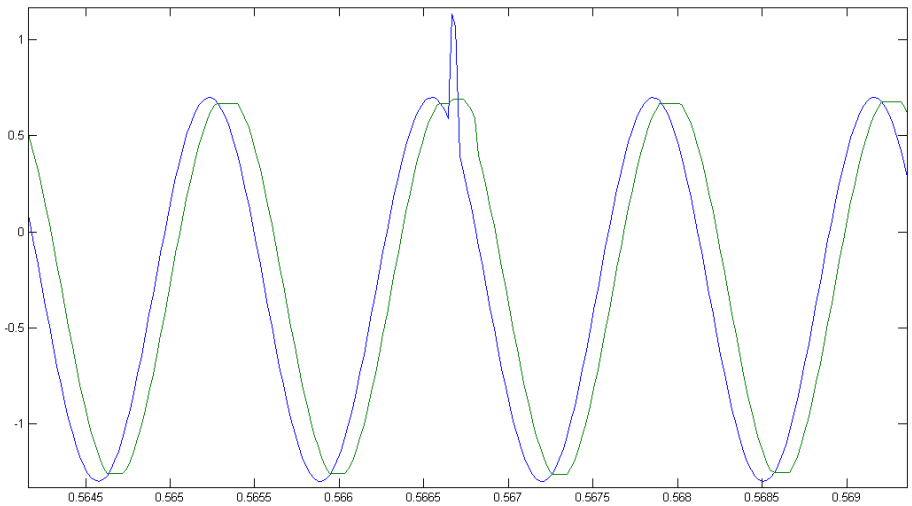
A teljes rendszer működését vizsgálandó egy rövid beszédjel pattogással, sistergéssel és szinuszos zavarral ellátott hangfájl (*matlab_rendsz_teszt.wav*) található a DVD mellékleten.

9.3. Mérési eredmények DSP-n

A DSP mérését laboratóriumban, az ott rendelkezésre álló függvénygenerátorokkal és oszcilloszkóppal végeztem.

9.3.1. Medián szűrő

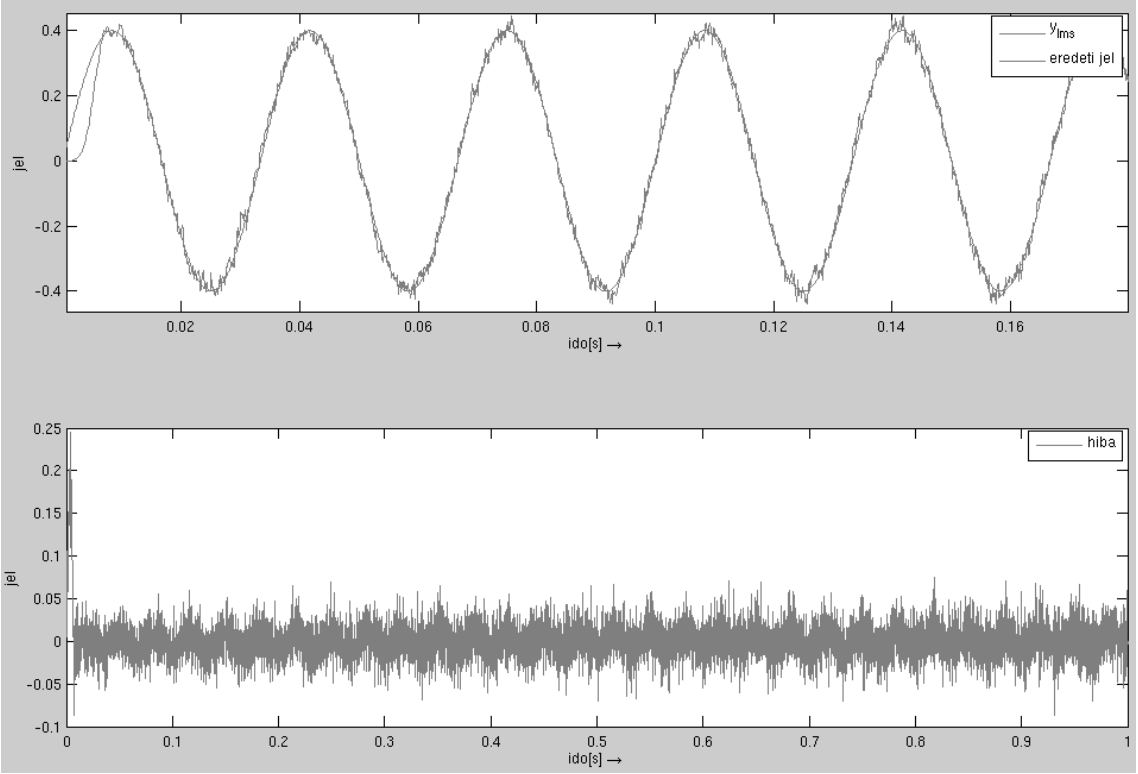
A medián szűrő beméréséhez szinuszjelre szuperponált impulzusokat használtam. A vizsgálójel előállításához két jelgenerátort kapcsoltam párhuzamosan. Az egyik egy 500 Hz-es szinuszjelet, a másik pedig 80 μs szélességű impulzusokat állított elő.



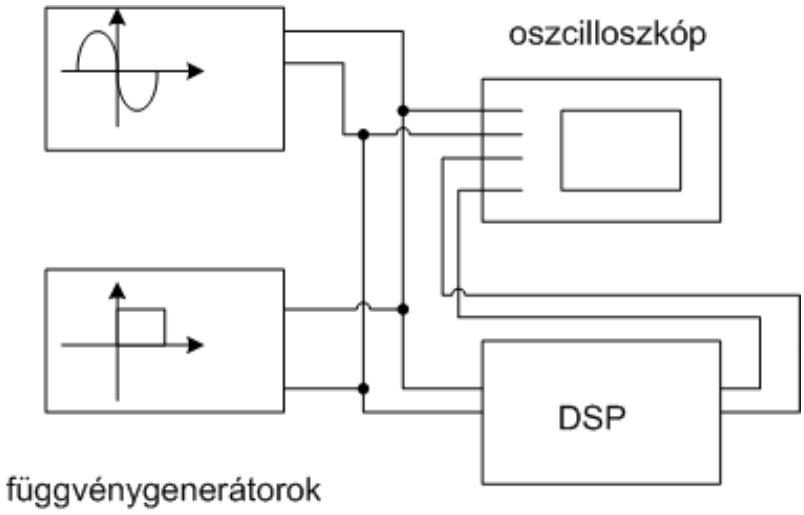
9.1. ábra. A MATLAB-os medián szűrő működésének vizsgálata

Az elrendezés a 9.3.ábrán látható. A medián szűrő ablakszélességének 10 mintát állítottam be. Az eredmény a 9.4.ábrán látható. A felső csatorna a szűrt jel, az alsó csatorna az eredeti jel. Látható, hogy az impulzusokat a szűrő kiszűrte, elenyésző mértékű torzulás maradt hátra. Ha az impulzusok szélessége 170 μs növekedik, akkor ahogy azt a 9.5.ábra felső csatornáján láthatjuk, az impulzusok megmaradnak.

A medián szűrő tesztelésére MATLAB-bal generált pattogó zajt adtam zenéhez, és ezt szűrtem először 10 minta szélességű, majd 5 minta szélességű medián szűrővel.



9.2. ábra. A MATLAB-os adaptív vonaljavító



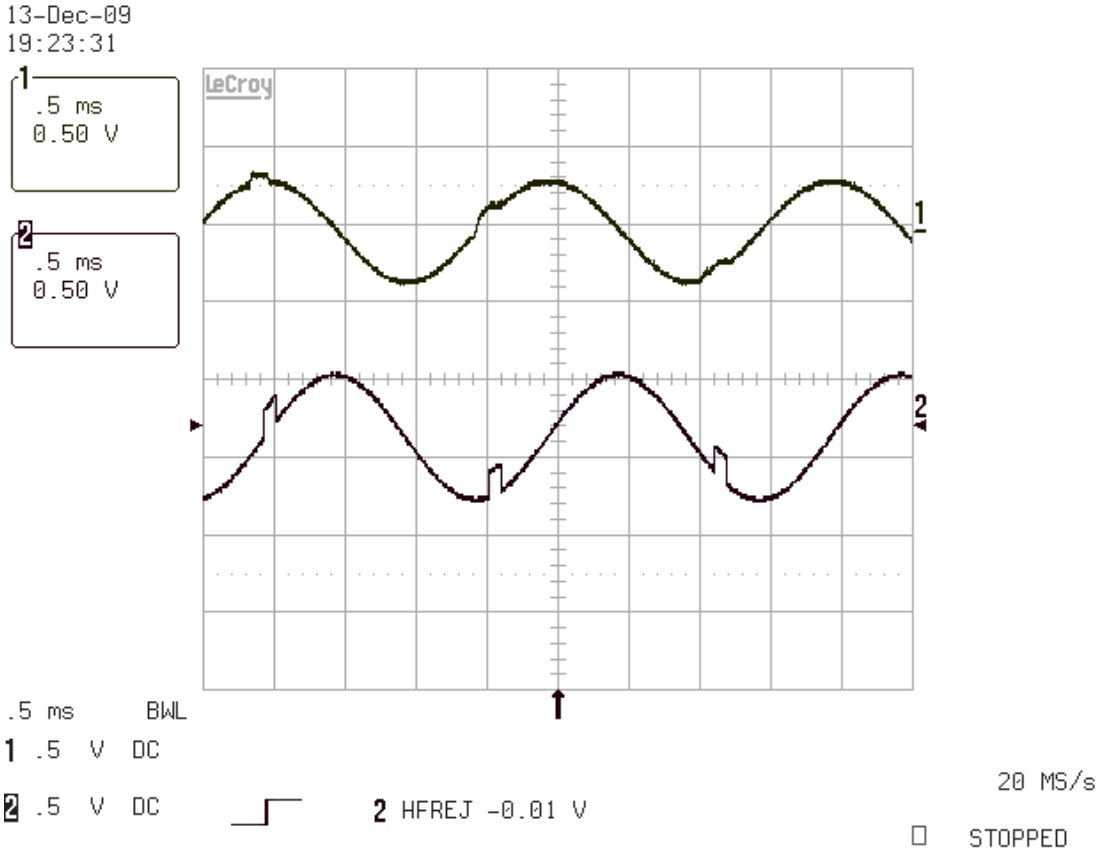
9.3. ábra. Mérési elrendezés DSP-vel történő méréshez

A DVD mellékleten megtalálható az eredeti (*median01.wav*), a hosszabb medián szűrővel szűrt (*median02.wav*), valamint a rövidebb szűrővel szűrt hanganyag is (*median03.wav*).

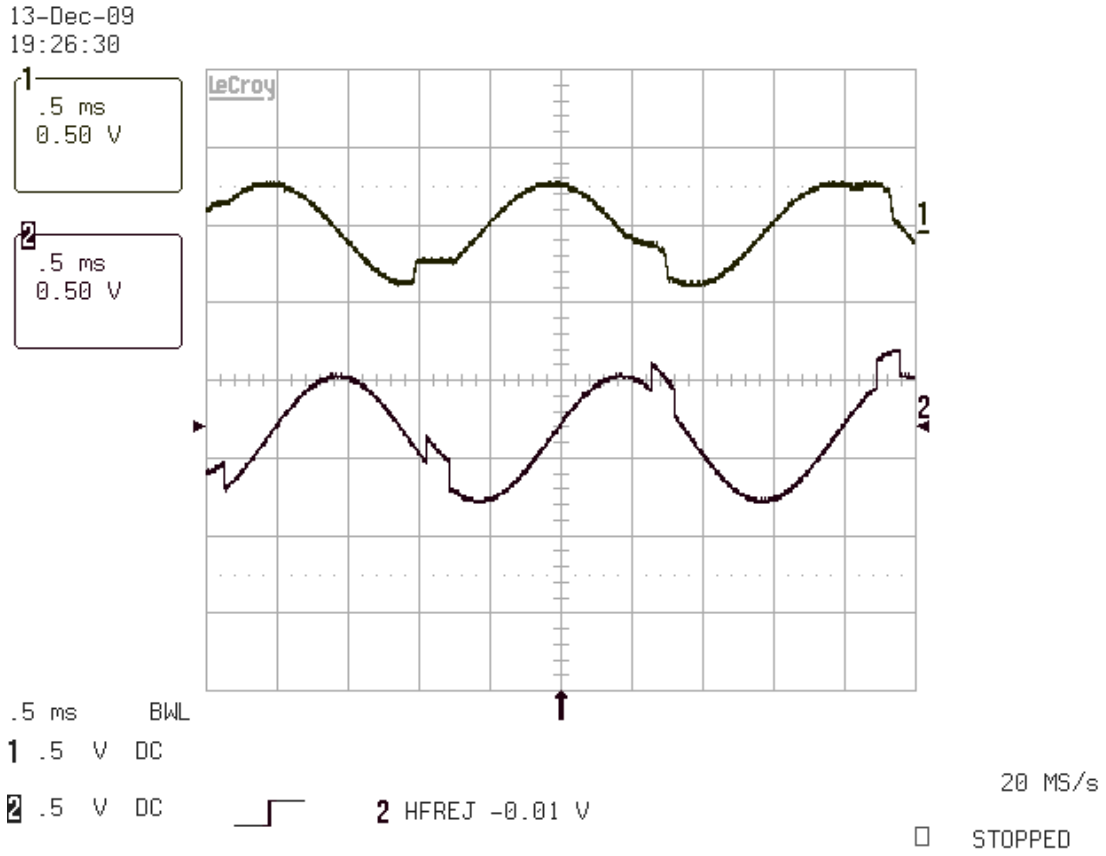
9.3.2. Adaptív vonaljavító

Az adaptív vonaljavítót a mediánszűrőéhez hasonló elrendezésben használtam. Itt az impulzusgenerátor helyett egy speciális zajgenerátort csatlakoztattam a szinuszgenerátorhoz. A vonaljavító prediktorát 100 tapesre állítottam be, mert ez már elég jól le tudja választani a szinuszjelet, de még nem igényel túlságosan sok lépést az adaptálása. A bátorsági tényezőt $\frac{1}{32}$-re állítottam be, a késleltetés itt mindegy, hiszen a véletlenszerű zaj bármilyen késleltetésnél korrelálatlan lesz önmagával. Az eredmény a 9.6. ábrán látható. A felső csatorna a zajjal terhelt szinuszjel, az alsó csatorna a zajgenerátor leválasztott jele.
Abban az esetben, ha a μ-t nagyon kicsire választjuk, az adaptív vonaljavító lassan áll be. A rendszer beállás közbeni átmeneti állapota látható a 9.7. ábrán.

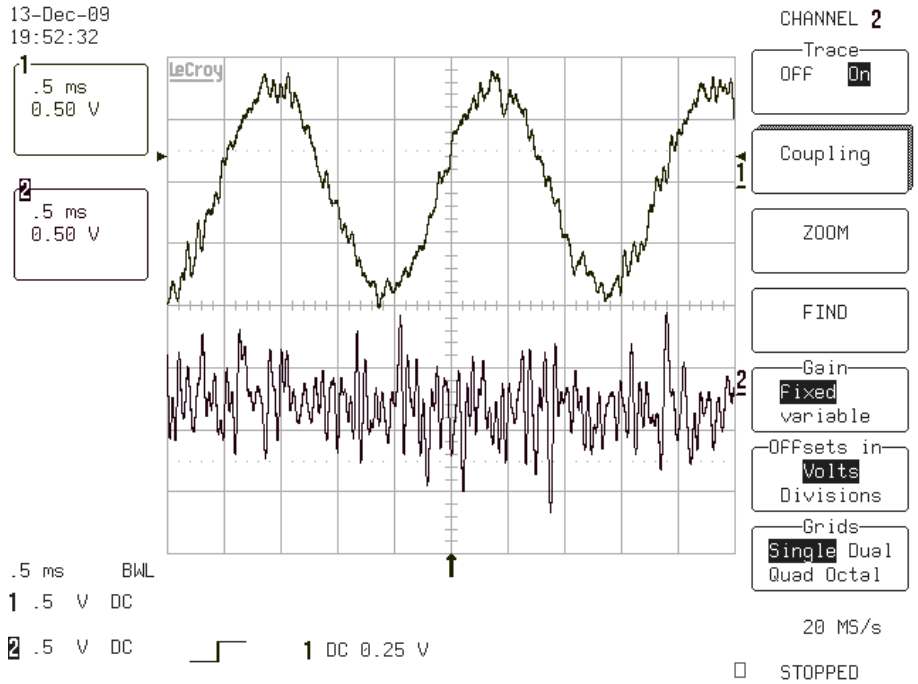
Az adaptív vonaljavító rendszerhez is tartoznak hangfájlok a DVD mellékleten. 100 tapes adaptív szűrőket vizsgáltam különböző késleltetésekkel. A vizsgálatot be-



9.4. ábra. A medián szűrő mérése

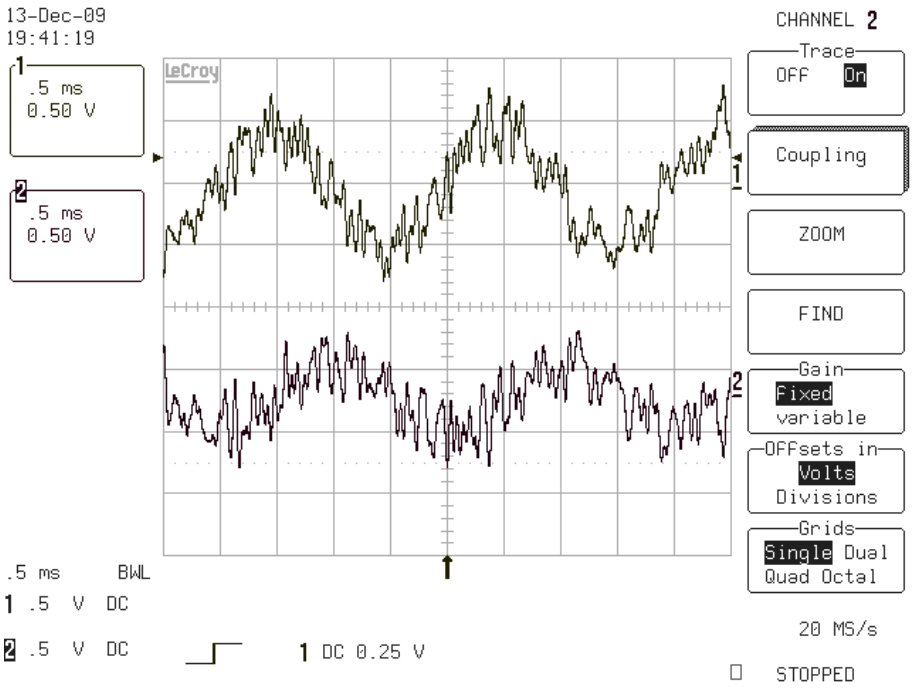


9.5. ábra. A medián szűrő mérése DSP-n



9.6. ábra. Az adaptív vonaljavító mérése DSP-n

széd felvétellel és hozzákevert chirp jellel végeztem. A fájlok tartalmazzák az eredeti bemeneti és a szűrt kimeneti jeleket is.



9.7. ábra. Az adaptív vonaljavító mérése DSP-n

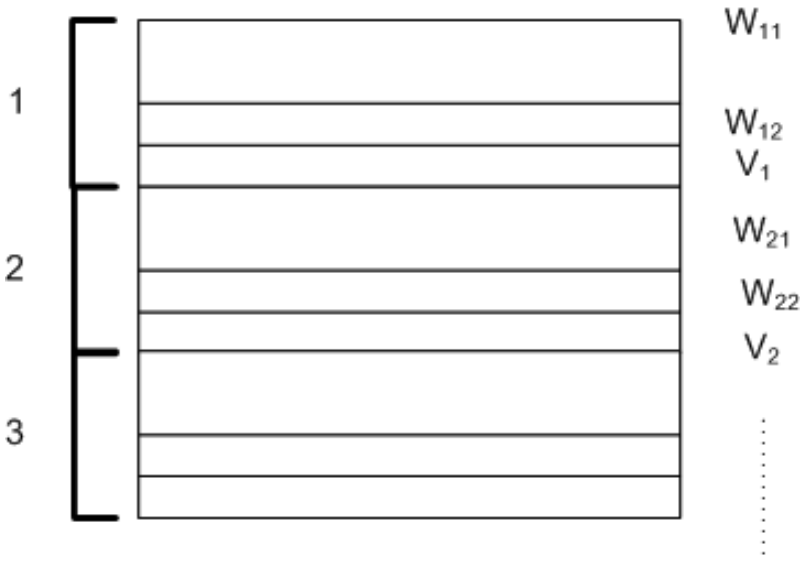
10. fejezet

Összefoglalás, kitekintés

Szakdolgozatom során a feladatom három különböző modellel leírható zajtípus tanulmányozása, az ezek kiküszöbölésére felhasználható általánosan elterjedt zajszűrő struktúrák megismerése, továbbfejlesztése volt. Munkám során felhasználtam egy diplomatervet, egy szakdolgozatot, valamint támaszkodtam az önálló labor alatt végzett munkámra is. Mélyebben megismertem a MATLAB programcsomag, különösen a Wavelet toolbox használatát, és a Blackfin DSP működését, a DSP-re való fejlesztés eszközeit, módszereit.

A szakdolgozat továbbfejlesztése alapvetően a sávokra bontás módszerének a módosításával lehetséges. Az általam megvalósított Multirate Analysis-on alapuló felbontás 2 hatványai szerint osztja sávokra a frekvenciatartományt. A felosztás viszont nem igazodik elég jól az emberi fül által érzékelhető 20 Hz – 20 kHz-es tartományhoz. A DSP kártyán található AD-átalakítók mintavételi frekvenciája 48 kHz. Tehát a szűrőbank bemenetén a Nyquist kritérium miatt 24 kHz a legmagasabb frekvenciájú komponens 24 kHz-es lehet. Ez azt jelenti, hogy a 11. szűrőfokozat után túllépem a 20 Hz-es alsó határt, tehát felesleges tovább decimálni, az már nem a hallható tartomány.

Ezért jobb megoldás lehet egy hibrid felbontás (lásd 10.1. ábra), vagyis a Bodor diplomatervben megoldott állandó szélességű sávokat az MRA módszerrel párhuzamosan bontjuk fel néhány diadikusan csökkenő méretű sávra. Ezzel a módszerrel biztosítható, hogy még rugalmasabban tudjuk megszabni, mely tartományokat érdemes jobb felbontással szűrni, melyeket kevésbé.



10.1. ábra. A frekvenciatartomány hibrid felbontása

Ábrák jegyzéke

Irodalomjegyzék

[1] Adsp bf-537 ez-kit lite evaluation system manual. http://www.analog.com/static/imported-files/eval_kit_manuals/ADSP-BF537_EZ-KIT_Lite_Manual_Rev_2_4.pdf.

[2] Adsp bf-537 hardware reference. http://www.analog.com/static/imported-files/processor_manuals/bf537_hwr_Rev3.2.pdf.

[3] C/c++ compiler and library manual for blackfin processors. http://www.analog.com/static/imported-files/software_manuals_legacy/626415916952668393726132936904386 45_Blackfin_comp_man.pdf.

[4] Bakó Tamás Béla. *Zenei anyagok zajcsökkentése.* http://home.mit.bme.hu/~bako/ZAOZ/zaz.htm.

[5] Bodor József. *Audio jelek zajcsökkentése szűrőbankok segítségével.* Budapesti Műszaki és Gazdaságtudományi Egyetem, 2007.

[6] Uwe Kiencke. *Methoden der Signalverarbeitung.* Universität Karlsruhe Institut für Industrielle Informationstechnik, 2007.

[7] BME Tanszéki Munkaközösség. *Digitális jelfeldolgozás, Segédlet.* Budapesti Műszaki és Gazdaságtudományi Egyetem, 2006.

[8] Balogh Tibor. *Adaptív jelfeldolgozás - Hallgatói mérés tervezése.* Budapesti Műszaki és Gazdaságtudományi Egyetem, 2008.

[9] Wallach Widrow. *Adaptive Inverse Control.* Prentice Hall PTR, Upper Saddle River, New Jersey, 1996.