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

Adaptív FIR-szűrők hatékony megvalósítása

TDK DOLGOZAT

Készítette
Szőke Kálmán Benjamin

Konzulens
Dr. Sujbert László

2015. október 26.

Tartalomjegyzék

Ábrák jegyzéke

Táblázatok jegyzéke

Kivonat

Nagy konvergenciasebességű, nagy fokszámú, illetve magas mintavételi frekvenciával működő adaptív szűrők megvalósítása nem triviális, miközben sok alkalmazásban szükség van rájuk. A mai jelfeldolgozó processzorok számítási teljesítménye mellett néhány kilohertz mintavételi frekvencián valós időben néhány ezer együtthatós adaptív FIR-szűrő futtatható. Egyes feladatok megkívánják nagyobb fokszámú szűrők alkalmazását, vagy magasabb mintavételi frekvenciát. Ekkor az alapegyenletek implementálása eredeti formájukban nem célravezető, a frekvenciatartományban kell a szükséges konvolúciót elvégezni. A módszer alapja, hogy a diszkrét Fourier-transzformáció az FFT algoritmussal gyorsan végrehajtható, és elegendően nagy pontszám esetén a konvolúció kevesebb műveletet igényel, mint az időtartományban. Ezt a lehetőséget használják ki az OLA (overlap and add) algoritmusok, ezek azonban a transzformáció adatgyűjtése és végrehajtása miatt jelentős késleltetéssel szolgáltatják a kimeneti adatokat. Ez a késleltetés általában az adaptív szűrőt használó alkalmazásokban nem megengedett, így a munkám során erre a problémára kerestem a megoldást.

A szakirodalom számos frekvenciatartománybeli számítási módszert kínál FIR-szűrőkhöz, amelyek a kimenet és a bemenet közti késleltetést hivatottak csökkenteni. Ezeket tanulmányozva és megismerve egy olyan eljárást mutatok be a dolgozatomban, amely az idő- és a frekvenciatartománybeli számítás együttes futtatásával küszöböli ki a transzformáció adatgyűjtéséből származó késleltetést. Ez az általam választott eljárás már a frekvenciatartománybeli számításhoz használt OLA algoritmus adatgyűjtési ideje alatt is szolgáltat szűrési eredményt, az időtartományban végzett konvolúció és az előző OLA blokkokból származó részeredmények segítségével. A módszernek köszönhetően nagy fokszámú FIR-szűrők alkalmazásakor is lehetséges az adatgyűjtésből származó késleltetések teljes mértékű megszüntetése, valamint továbbra is gyorsabb, mint számos más időtartománybeli eljárás, amiknél ilyen fajta késleltetés nem lép fel.

Ezt az időtartománybeli konvolúcióval kiegészített OLA algoritmust MATLAB környezetben és ADSP-21364 jelprocesszoron valósítottam meg. Ezek után az elkészült algoritmust egy adaptív FIR-szűrőt használó alkalmazásba építettem be. Ehhez először MATLAB környezetben implementáltam a feladathoz hatékonyan használható frekvenciatartománybeli adaptációt végző Fast Block LMS algoritmust, majd ezt és a konvolúcióval kiegészített OLA algoritmust egy jelprocesszoros fejlesztőkártyán kialakított, adaptív FIR-szűrőt működtető visszhangcsökkentő struktúrába integráltam, amelyen mérésekkel demonstráltam a helyes működést, a rendelkezésre álló jelprocesszor lehetőségeihez mérten.

Abstract

The implementation of adaptive filters is not trivial with high convergence rate, large number of taps and high sampling rate, required by many applications. Several signal processors are able to run an adaptive FIR filter with few thousands of coefficients and few kilohertz sampling rate in real-time. Some tasks require the use of filters with more taps or higher sampling frequency, in this case the basic equations are not useful for implementation in their original form, therefore, we may need to do the necessary convolution in the frequency domain. The method is based on the fast Fourier transform (FFT) algorithm, as for large number of taps the convolution requires less operations than that in the time domain. This capability is used in the OLA (overlap and add) algorithms, however, they introduce a significant delay of the output, due to the data collection of the transformation. In general, this delay is not tolerated in many applications of adaptive filters, so in my work I have been looking for a solution to this problem.

There are a number of frequency-domain calculation methods for FIR filters published in the literature, which are designed to reduce the latency between the input and the output. Based on the known methods I introduce one that is able to eliminate this latency by a smart calculation of the convolution both in the time and the frequency. The method I have chosen provides filtering results even in the data collection period, thanks to the time domain convolution and partial results of the previous OLA's blocks. Besides the latency can be eliminated entirely, the calculation is still faster than other time domain filtering methods, where such delay does not occur.

I have implemented this algorithm in MATLAB environment and on an Analog Devices Sharc (ADSP-21364) DSP. After that I have embedded it to an adaptive FIR filter application. To this end first I have implemented an FBLMS (Fast Block LMS) algorithm, which can perform the adaptation in the frequency domain. Then I have integrated this module and the smart OLA filtering method into an echo cancellation procedure. Thus the viability and the effectiveness of the method are demonstrated by simulations and measurements, as well.

Bevezető

Motiváció

Manapság egyre több alkalmazás igényli az adaptív szűrők használatát, mint például aktív zajcsökkentő, visszhangcsökkentő rendszerek vagy épp méréstechnikai és szabályzástechnikai alkalmazások. Egyes feladatok, legfőképpen az akusztikai alkalmazások megkívánják a nagyobb fokszámú szűrők alkalmazását, vagy magasabb mintavételi frekvenciát, ezért a dolgozatomban bemutatom, hogy az adaptív FIR-szűrők használata során milyen problémák és korlátok lépnek fel ilyen követelmények teljesítése esetén, és ezek milyen jelfeldolgozási módszerekkel elégíthetők ki és valósíthatók meg hatékonyan.

Ezek az adaptív szűrőt valós időben működtetető alkalmazások ma már számos fajta hardveren és programozási környezetben implementálhatók. A legelterjedtebbek az FPGA-n, GPGPU-n vagy épp digitális jelprocesszoron történő megvalósítások. A mai jelfeldolgozó processzorok számítási teljesítménye mellett néhány kilohertz mintavételi frekvencián valós időben néhány ezer együtthatós adaptív FIR-szűrő futtatható. Ez azt jelenti, hogy az időtartományban végzett együtthatók adaptálása és az ezekkel való szűrés a konvolúció segítségével, két mintavételezés között csak néhány ezer együtthatóra végezhető el. Annak érdekében, hogy az ily módon adódó maximális együtthatószámot túllépjük, a frekvenciatartományban kell a szükséges konvolúciót elvégezni a hatékonyság érdekében.

Ehhez adott a diszkrét Fourier-transzformáció, amely az FFT algoritmussal gyorsan végrehajtható, és elegendően nagy pontszám esetén így a konvolúció kevesebb műveletet igényel, mint az időtartományban. Azonban az FFT algoritmussal történő transzformáció blokkonkénti adatgyűjtést és végrehajtást igényel, ami előnyökkel és hátrányokkal is jár egyaránt. Előnye, hogy az adaptív FIR-szűrő konvergenciasebességét nagyban megnöveli az együtthatók frekvenciatérben történő blokkonkénti adaptációja, hátránya pedig az, hogy a blokkonkénti adatgyűjtés és a transzformációs művelet az akusztikus alkalmazásokban nem megengedhető késleltetést okoz az adaptív FIR-szűrő kimenetén.

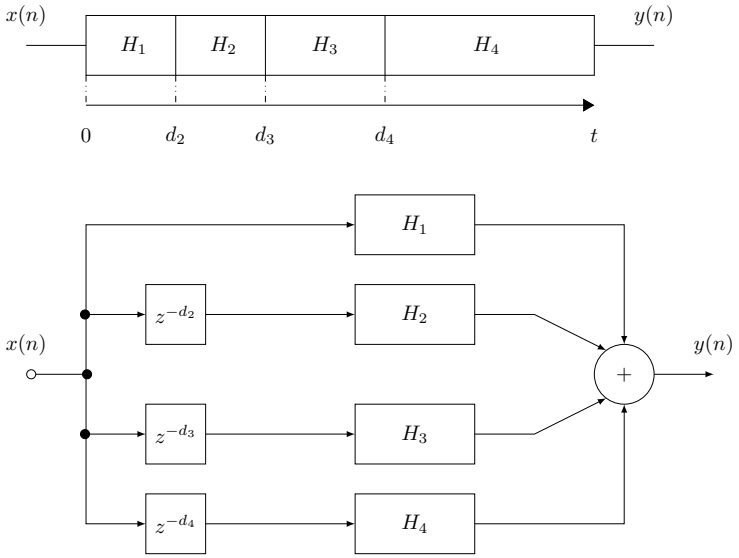
Megoldási módszerek

Az időtartományban implementált direkt konvolúció kedvező tulajdonsága, hogy nem jár késleltetéssel, viszont a kedvezőtlen az, hogy igen számításigényes a szorzás/összeadás műveletei miatt. Nem mellesleg ez a számítási igény mintánként lineárisan növekedik a FIR-szűrő működtetése során, amíg a rendszer el nem éri az állandósult állapotot, amikoris a FIR-szűrő hosszával egyezik meg ezen műveletek száma [7]. Nagy fokszámú FIR-

szűrők esetén tehát a direkt konvolúcióval megvalósított valós időjű szűrés nem célravezető megoldás.

Elegendően nagy pontszám esetén a frekvenciatartományban végzett konvolúció kevesebb műveletet igényel, mint az időtartományban. Ezt a lehetőséget használják ki az OLA (overlap-add) és az OLS (overlap-save) algoritmusok [10], melyek számítási igénye a direkt konvolúcióhoz képest sokkal kedvezőbb, mert ez logaritmikusan növekedik a blokkok méretének függvényében. Ennek a hatékony műveletvégzésnek azonban a kedvezőtlen tulajdonsága az, hogy a blokkonkénti adatgyűjtés késleltetést okoz, amely mértéke legalább akkora vagy nagyobb, mint a szűrés során alkalmazandó FIR-szűrő hossza [7].

E két módszer közös használatával a késleltetés megszüntethető/csökkenthető, miközben a számítási igény továbbra is kedvező szinten tartható [5]. Az alapelgondolás az, hogy a nagy fokszámú szűrő felbontható több párhuzamosan működő kisebb fokszámú szűrésre, ahogy ez az 1. ábrán is látható. Abban az esetben, ha a párhuzamos ágakban működő szűrők előtti késleltetés megegyezik a particionált a blokkhoz tartozó késleltetéssel, akkor a párhuzamosan működő szűrések kimenetének összegeiből előállítható a teljesen szűrt kimenet. A legegyszerűbb ilyen struktúra az, amikor a nagy fokszámú szűrő egyenletesen van particionálva. Ekkor minden egyes szűrési ágban egy ugyanakkora FFT elvégzésére van szükség a késleltetett bemeneteken.



1. ábra. *Nagy fokszámú FIR-szűrő impulzusválaszának nem egyenletes particionálása, és az ezekkel való párhuzamos szűrési struktúra. [6]*

William G. Gardner [7] bemutatott egy hatékony, nem egyenletesen particionált eljárást is, ami azon alapszik, hogy az impulzusválasz növekvő méretű blokkokra van felosztva. Ez azt használja ki, hogy a rövidebb blokk méretű impulzusválaszokkal való szűrések alacsony késleltetést okoznak, míg a hosszabbak, a frekvenciatartományban szűrve az FFT algoritmus által kevesebb műveletet igényelnek. Ezt a két kedvező tulajdonságot kihasználva a nem egyenletesen particionált impulzusválasszal való szűrés hatékonyabban alkalmazható a késleltetés megszüntetésére/csökkentésére.

Dolgozatomban egy olyan alapvető eljárást mutatok be, amely az idő- és a frekvenciatartománybeli számítás együttes futtatásával küszöböli ki a transzformáció adatgyűjtéséből származó késleltetést. Alapja, hogy a frekvenciatartománybeli számításhoz használt OLA algoritmus adatgyűjtési ideje alatt párhuzamosan végzett időtartománybeli konvolúció, és az előző OLA blokkokból származó részeredmények képesek késleltetés nélkül szolgáltatni a szűrési eredményt. Ez az általam választott eljárás egyaránt alkalmazható adaptív és nem adaptív FIR-szűrőkhöz is.

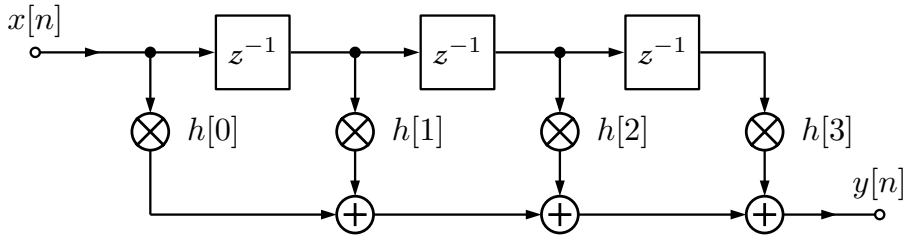
1. fejezet

Idő- és frekvenciatartománybeli FIR-szűrés

1.1. Időtartománybeli szűrés a direkt konvolúcióval

Egy FIR-szűrővel való időtartománybeli szűrésnél a bemeneti jel $x(n)$ előző értékeire és a szűrő impulzusválaszára h_i van szükségünk. A szűrési művelet matematikailag a konvolúcióval történik, vagyis az N hosszúságú impulzusválaszunkkal direkt konvolúciót hajtunk végre a bemenet aktuális és előző $N-1$ mintáin. Egy n-dik minta kiszámolásánál a műveletét elképzelhetjük úgy is, hogy a bemenetünk régebbi mintáit késleltetőkön keresztül szorozzuk meg az impulzusválasz értékeivel, majd ezeket összegezzük, amivel eredményül megkapjuk az n-edik ütemben számolt eredményét a szűrésnek.

$$y(n) = \sum_{i=0}^{N-1} h_i x(n-i) = (h * x)(n) \tag{1.1}$$



1.1. ábra. *A direkt konvolúció hatásvázlata, 4 együtthatós FIR-szűrő esetén.*

A direkt konvolúcióval végrehajtott időtartománybeli szűrésnek előnye, hogy a konvolúciós művelet minden egyes új minta beérkezése után azonnal elvégezhető, így adatgyűjtésből származó késleltetés nem áll fenn. Hátránya ennek a módszernek viszont az, hogy N nagyságú impulzusválasz esetén a direkt konvolúciós művelet szorzásai és összeadásai $\mathcal{O}(N^2)$ műveletet igényelnek. Ebből látható, hogy nagy fokszámú FIR-szűrők esetén

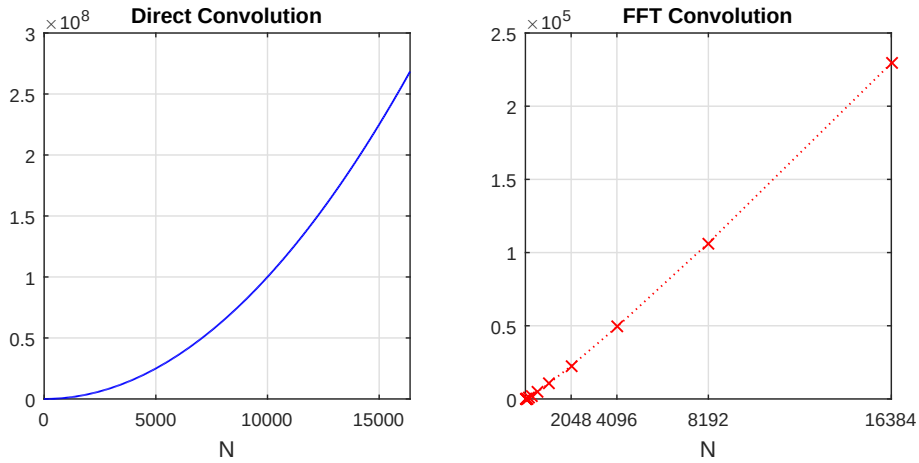
az időtartománybeli szűrés már gondot okozhat a nagy műveletigénye miatt, ezért célszerű megismerni frekvenciatartományban használható módszereket is.

1.2. Frekvenciatartománybeli szűrés az FFT algoritmussal

Lineáris rendszerek modellezése esetén az időtartománybeli konvolúció egyszerű szorzási műveletre egyszerűsíthető le a frekvenciatartományban. Ez azt jelenti, hogy a FIR-szűrő impulzusválaszának Fourier-transzformáltja és a bemeneti jelünk Fourier-transzformáltja szorzataként kiszámolhatjuk a szűrő kimenetének frekvenciatartománybeli eredményét, amely az időtartományba visszatranszformálva természetesen megegyezik az időtartománybeli szűrés eredményével.

$$\mathcal{F}(b * x) = \mathcal{F}(b) \cdot \mathcal{F}(x) \tag{1.2}$$

A diszkrét Fourier-transzformáció mátrixos koncepciójából következik, hogy N szám transzformációjához továbbra is $\mathcal{O}(N^2)$ szorzás szükséges, ami jól láthatóan megegyezik az előzőekben tárgyalt direkt konvolúciós eljárással. Azonban ezt a számítási igényt 1965-ben Cooley és Tukey matematikusok által kidolgozott gyors Fourier-transzformációs algoritmus [3] radikálisan lecsökkentette, amely forradalmat jelentett akkor a digitális jelfeldolgozásban. Az általuk kifejlesztett komplex szorzásokat végző lepkeműveletek lehetővé tették, hogy a teljes transzformáció elvégezhető $\mathcal{O}(Nlog_2(N))$ szorzási műveletből. Tipikusan jelprocesszorokon a Radix-2 FFT algoritmus érhető el, aminek megkötése mindössze annyi, hogy az algoritmus csak a kettő hatványaival megegyező nagyságú adatsorokat képes transzformálni a gyors Fourier-transzformáció során.



1.2. ábra. *Direkt konvolúció N^2 és az FFT algoritmussal számolt frekvenciatartománybeli konvolúció $N \log_2(N)$ műveletigénye.*

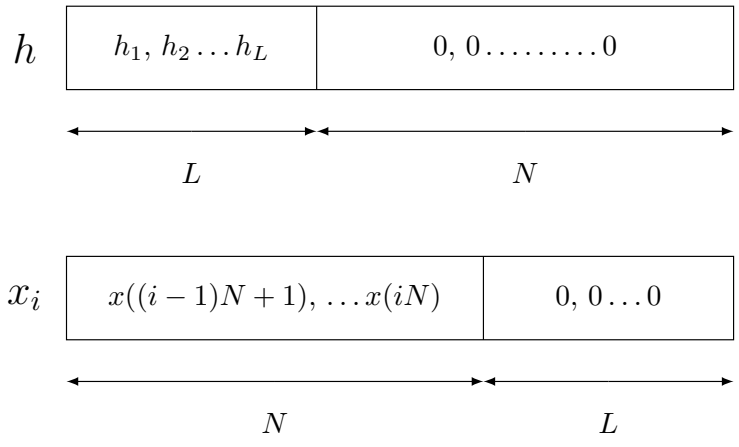
Tovább elemezve a gyors Fourier-transzformáció alkalmazási lehetőségeit, észrevehetjük, hogy ez a művelet nem alkalmas mintánkénti végrehajtásra. Vagyis valós idejű FIR-szűréshez csak úgy alkalmazható, ha a transzformációs számításokat megelőzően már elegendő adatot rögzítettünk ahhoz, hogy blokkonként végezhessük el a frekvenciatar-

tománybeli szűrést. A valós idejű FIR-szűréshez számos ilyen blokkonkénti gyors Fourier-transzformációs algoritmus lett kifejlesztve, ezek közül is a témához legfontosabbak az Overlap-Add és Overlap-Save algoritmusok, melyeket a továbbiakban ismertetek.

1.2.1. Overlap-Add módszer

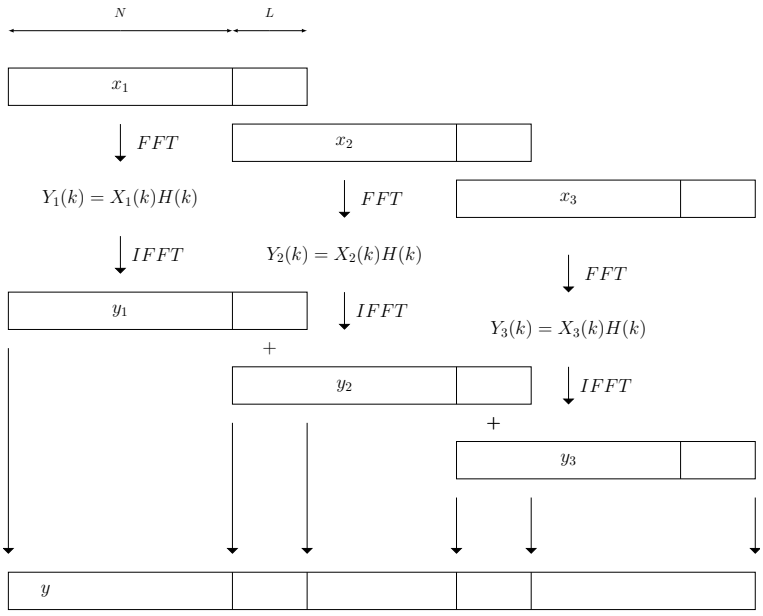
Az Overlap-Add az egyike azon blokkonkénti gyors Fourier-transzformációt használó eljárásoknak, amik hatékonyan használhatók valós idejű FIR-szűréshez [2]. Ennél a módszernél a bemeneti adatokat $x_i(n)$ blokkokba rögzítjük, melyek méretei $N_{fft} = N+L$ nagyságúak. Az adatgyűjtés során egy OLA blokkba N mintát rögzítünk, majd a fennmaradó L nagyságú részt, amely megegyezik a szűrés során használt FIR szűrő impulzusválasz hosszával, nulla elemekkel töltjük fel. A módszer során a cél az, hogy az N hasznos adattal kitöltött blokkon a szűrést a frekvenciatartományban is elvégezhessük, és L nagyságú átlapolást és összegzést használva az y_i kimeneti eredményblokkok között, előállíthassuk a szűrés eredményét, hatékonyabban, mint azt a direkt konvolúcióval tehetnénk az időtartományban.

Ehhez az adatgyűjtések előtt a FIR-szűrő impulzusválaszának Fourier-transzformáltjára is szükségünk van, amelynek mérete az FFT algoritmus miatt egyaránt $N_{fft} = N + L$ kell hogy legyen. Ennek érdekében az impulzusválaszt az $x_i(n)$ bementi blokkokhoz hasonlóan tároljuk, csak épp az $N + L$ felosztást fordítva alkalmazzuk. Itt az L nagyságú $h(n)$ impulzusválasz végét egészítjük ki annyi nullával, hogy a Fourier-transzformációhoz megfelelően N_{fft} méretű tömb álljon a rendelkezésünkre, vagyis ekkor az L mintából álló impulzusválaszunk tömbjét N darab nulla elemmel egészítjük ki. Az ily módon történő felosztás után még a szűrés előtt az FFT algoritmus segítségével hatékonyan előállíthatjuk a FIR-szűrőnk frekvenciatartománybeli diszkrét átviteli függvényét $H(k)$-t, amelyre a szűrés során folyamatosan szükségünk lesz.



1.3. ábra. *Az $x_i(n)$ és $h(n)$ felosztása az Overlap-Add eljárás során*

Az $x_i(n)$ és $h(n)$ megfelelő $N_{fft} = N + L$ felosztása után az Overlap-Add módszerrel való valós idejű FIR-szűrés a következő lépések és ábrák szerint történik.



1.4. ábra. *Overlap-Add módszer blokkonkénti szűrése*

1. A FIR-szűrés megkezdése előtt létrehozzuk $h(n)$ impulzusválasz $N + L$ nagyságú tömbjét, az előzőekben leírt felosztás szerint, majd az FFT algoritmussal kiszámoljuk ennek a Fourier-transzformáltját $H(k)$-t.

2. Az N minta rögzítése és az $x_i(n)$ blokkban használt $N + L$ felosztás helyes alkalmazása után, szintén az FFT-vel kiszámoljuk a bemeneti blokk Fourier-transzformáltját $X_i(k)$-t.

3. Az így kapott transzformáltak segítségével a i-edik blokk szűrését elvégezzük a frekvenciatartományban $Y_i(k) = X_i(k) \cdot H(k)$.

4. $Y_i(k)$-t inverz Fourier-transzformáljuk, amivel megkapjuk az i-edik bemeneti blokk szűrési eredményét az időtartományban, $y_i(n)$-t.

5. Az $y_i(n)$-nél használt L nagyságú átlapolás miatt az ábra szerint az előző $y_{i-1}(n)$ eredmény blokk utolsó L elemét hozzáadjuk $y_i(n)$ eredményblokk első L eleméhez. Ezzel az i-edik és az $i-1$-edik blokk átlapolási tartományában megkapjuk ugyanazt a szűrési eredményt, mint amit a direkt konvolúcióval kapnánk, vagyis ezek azok a minták a blokkos szűrés során amelyek a valós idejű szűrés alkalmazásakor a kimeneti mintákat szolgáltat a következő blokkos végrehajtás elkészültéig.

6. Az $y_i(n)$ átlapolási tartományába eső L értékeket elmentjük a következő $y_{i+1}(n)$ számításához, majd folytatjuk mintavételezést az $x_{i+1}(n)$ bementi blokkba, a 2. lépéstől folytatva.

A módszer bemutatásához olyan ábrákat használtam, ahol ez az L nagyságú átlapolás 50%-nál kisebb, de fontos megjegyezni, hogy ennek mértéke lehet ennél jóval nagyobb is. Ez akkor fordulhat elő, amikor használni kívánunk egy akkora impulzusválaszt, aminek

L fokszáma nagyobb, mint az FFT algoritmushoz tervezett N_{fft} blokk méretének fele. Ekkor fontos azt észben tartanunk, hogy ilyen esetben már nemcsak két $y_i(n)$ eredmény blokk lapolódik át, hanem a nagyobb L függvényében akár több is.

50%-nál nagyobb átlapolást használva, a kimenetnek szánt minták száma is változik az L növelésére. Ekkor az összegzések után az $y_i(n)$ blokknak csak az első N eleme az, ami megegyezik a direkt konvolúcióval történő szűrés eredményével, így ilyenkor a következő blokkos végrehajtás elkészültéig csak ez az N darab minta használható kimeneti adatként. Ebből még egy utolsó fontos dolog is adódik, amit fontos megemlíteni, mégpedig az, hogy ennek az N darab mintának a kiadása alatt, vagyis Nf_s valós idő alatt a következőleg elkezdett $y_{i+1}(n)$ kimeneti blokk eredményének mindenképpen el kell készülnie, hogy az eljárás folyamatosan és megszakítás nélkül tudja szolgáltatni az egymás után sorban következő N nagyságú kimeneti eredményeket.

Az FFT algoritmus előnyeit kihasználva az Overlap-Add módszer $\mathcal{O}(N_{fft}log_2(N_{fft}))$ műveletigénnyel bír, ami sokkal kedvezőbb annál, mintha direkt konvolúciót használnánk. Kedvezőtlen tulajdonsága viszont a blokkos végrehajtásból adódó N mintányi késleltetése, és a memóriaigényessége, ami abból adódik, hogy a módszer során tárolnunk a $H(k)$ komplex tömböt, és az előző $y_{i-1}(n)$ blokk eredményeket az átlapolásnál használt összegzés miatt.

1.2.2. Overlap-Save módszer

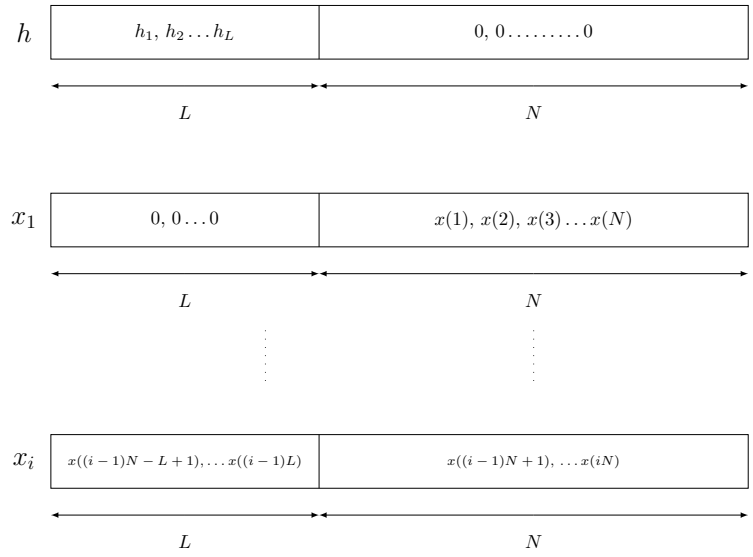
Az Overlap-Add módszernél mutatkozó memóriaigényt hivatott csökkenteni az Overlap-Save eljárás. Célja az, hogy a blokkos szűrés végrehajtás alatt ne legyen szükség az $y_i(n)$ kimeneti blokkok átlapolására, és az ebből adódó összegzésre. Ennek érdekében bemeneti adatokat $x_i(n)$ blokkokba egy másfajta $N_{fft} = L + N$ felosztás szerint rögzítjük, és az átlapolást most nem az $y_i(n)$ blokk eredményekre alkalmazzuk, hanem a bemeneti $x_i(n)$ blokkok között valósítjuk meg.

Ennél az eljárásnál $x_i(n)$ blokkot úgy osztjuk fel, hogy az új mintákat a blokk második N minta nagyságú része tartalmazza, mégpedig úgy, hogy ezt az új $x_i(n)$ blokkot a mintavételezés során az $N_{fft} - N$-edik helytől kezdjük el feltölteni. Az $x_i(n)$ blokk első L nagyságú része pedig az aktuális blokkok előtti mintavételezett bemeneti értékeket tartalmazza, vagyis itt az átlapolás olyan formában jelenik meg, hogy az $x_i(n)$ blokkban tárolt értékeket a következő $x_{i+1}(n)$ blokkban mindig N mintányit balra eltoljuk, és az így szabadon maradt N nagyságú helyre rögzítjük a bemenet új értékeit. A szűrés elején az $x_1(n)$ blokk első L elemét értelemszerűen nulla elemekkel tölthetjük fel.

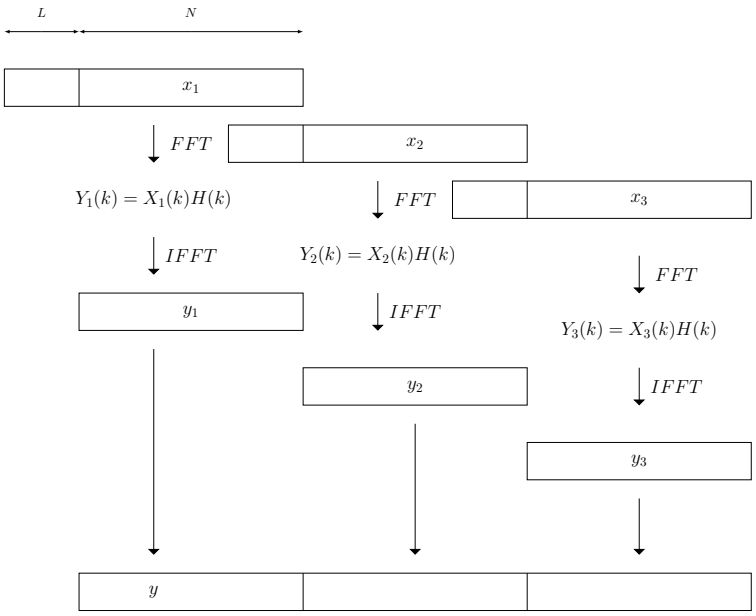
Ez az eltérő fajta blokkos szervezés az impulzusválasz tömbjének felosztásában viszont nem igényel különböző felosztást az Overlap-Add módszernél megismerthez képest. Továbbra is a $h(n)$ $N + L$ nagyságú tömbjét először az L darab FIR-szűrő együtthatóval töltjük fel, majd a maradék N mintányi helyre nulla értékeket tárolunk el.

Az $x_i(n)$ és $h(n)$ megfelelő felosztása után az Overlap-Save módszerrel való valósidejű FIR-szűrés a következő lépések és ábrák szerint történik.

1. A FIR-szűrés megkezdése előtt létrehozzuk $h(n)$ impulzusválasz $N + L$ nagyságú



1.5. ábra. *Az $x_i(n)$ és $h(n)$ felosztása az Overlap-Save eljárás során*



1.6. ábra. *Overlap-Save módszer blokkonkénti szűrése*

tömbjét, az előzőekben leírt felosztás szerint, majd az FFT algoritmussal kiszámoljuk ennek a Fourier-transzformáltját, $H(k)$-t.

2. Az N minta rögzítése és az $x_i(n)$ blokkban használt $N + L$ felosztás helyes alkalmazása után, szintén az FFT-vel kiszámoljuk a bemeneti blokk Fourier-transzformáltját, $X_i(k)$-t.

3. Az így kapott transzformáltak segítségével a i-edik blokk szűrését elvégezzük a frekvenciatartományban, $Y_i(k) = X_i(k) \cdot H(k)$.

4. $Y_i(k)$-t inverz Fourier-transzformáljuk, amivel megkapjuk az i-edik bemeneti blokk

szűrési eredményét az időtartományban, $y_i(n)$-t.

5. A bemeneti blokkokon alkalmazott átlapolás hatására az $y_i(n)$ eredmény blokk utolsó N mintája egyezik meg a direkt konvolúcióval történő eljárással, így ez az utolsó N adat az, amit a valósidejű szűrő kimenetén használhatunk a következő $x_{i+1}(n)$ blokk szűrésének elkészültéig.

6. Az $x_i(n)$ bementi blokk első N elemét elhagyjuk, és a megmaradt $N_{fft} - N$ adatot a következő $x_{i+1}(n)$ blokkban használjuk fel az első L mintának, majd folytatjuk az eljárást az $x_{i+1}(n)$ blokkra is, a 2. lépéstől.

Az Overlap-Save módszerrel a blokkos végrehajtásból adódó N mintányi késleltetés továbbra is megmarad, de a hatékonyabb átlapolási módszernek köszönhetően a memóriaigénye kevesebb, mint az Overlap-Add módszernek.

1.3. Késleltetés nélküli szűrés idő- és a frekvenciatartománybeli eljárás együttes használatával

Az előző blokkonkénti végrehajtást használó Overlap-Save és Overlap-Add eljárásnál láttuk, hogy ezeket használva a valósidejű FIR-szűrés csak N minta késleltetése után szolgáltat eredményt. A szakirodalom számos frekvenciatartománybeli számítási módszert kínál FIR-szűrőkhöz, amelyek az adatgyűjtésből eredő kimenet és a bemenet közti késleltetést hivatottak csökkenteni. William Gardner cikkére [7] alapozva egy olyan módszert mutatok be, ami az idő- és a frekvenciatartománybeli számítás együttes futtatásával küszöböli ki a transzformáció adatgyűjtéséből származó késleltetést.

Az általam javasolt módszer Gardner módszeréhez képest abban tér el, hogy az impulzusválaszt particionálatlanul használom fel a szűrés során, hogy az eljárás minél kevésbé legyen bonyolult, vagyis nagy komplexitású, és ennek révén könnyebb legyen a használhatósága adaptív FIR-szűrőt használó rendszerekben is. Ez a késleltetés nélküli szűrés azon alapszik, hogy az Overlap-Add módszerrel való frekvenciatartománybeli konvolúciót kiegészítem egy parciális konvolúciót használó időtartománybeli szűréssel.

A módszer úgy működik, hogy az Overlap-Add eljárásnál használt $y_{i-1}(n)$ előző kimeneti blokk eredmények átlapolási részéhez nem a következő $y_i(n)$ blokk ezen részhez tartozó eredményeit adjuk hozzá (mivel ezek még nem állnak rendelkezésre), hanem ezeket előállítjuk még az adatgyűjtés ideje alatt egy parciális konvolúcióval $x_i(n)$ blokk első N éppen mintavételezet értékeiből. A parciális konvolúció alatt ebben az esetben azt kell érteni, hogy az $x_i(n)$ első N elemére a konvolúciós szummát mindig csak az adott n-edik indexszámmal megegyező FIR-szűrő együtthatóig végzzük el minden egyes új mintavételezési értékre. A tömbök indexelésének kezdetét nullától értelmezve ez a következőképen írható le matematikailag:

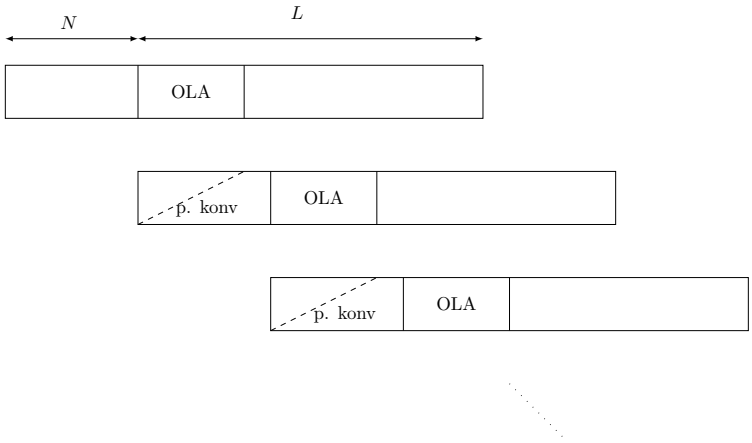
$$y_p(n) = \sum_{j=0}^{n} h_j x_i(n-j) \qquad n = 0, 1, 2 \ldots N-1 \tag{1.3}$$

Amint az $x_i(n)$ n-edik mintájára elkészül ez az $y_p(n)$ konvolúciós eredmény, az előző $y_{i-1}(n)$ blokk átlapolási részének $y_{i-1}(N+n)$ elemének hozzáadásával megkaphatjuk a helyes szűrési eredményt, már az adott n-edik minta beérkezése után azonnal. Ennek az eljárásnak a helyessége azzal igazolható, hogy az Overlap-Add eljárásnál az $y_i(n)$ átlapolási tartománya pont azt a részeredményt tartalmazza, amit a parciális konvolúció során $y_p(n)$ kiszámolásánál még nem végeztünk el.

Matematikailag ez az eredmény azon alapszik, hogy a konvolúciós művelet disztributív tulajdonságú, tehát $y(n)$ szűrési eredmény két külön végzett konvolúciós szumma segítségével is előállítható. (Ekkor $h(n)$ az OLA módszernek megfelelően N_{fft} nagyságú és $L+N$ felosztású, vagyis az utolsó N eleme nulla!)

$$y(n) = \sum_{j=0}^{n} h_j x_i(n-j) + \sum_{k=n+1}^{N_{fft}-1} h_k x_i(n-k) = y_p(n) + y_{i-1}(N+n) \qquad (1.4)$$

$$n = 0, 1, 2 \ldots N-1$$



1.7. ábra. *Valós időjű késleltetés nélküli szűrés Overlap-Add módszerrel*

A valós idejű FIR-szűrés során persze azzal is számolnunk kell, hogy az $x_i(n)$ első N mintájának mintavételezése után az FFT algoritmussal végzett frekvenciatartománybeli szűrésnek is még időre van szüksége, ahhoz, hogy elkészüljön, és így szolgáltatni tudja a következő $x_{i+1}(n)$ blokknál végzett parciális konvolúció kiegészítéséhez szükséges $y_i(n)$ i-edik blokk szűrési eredményét. Tehát valós időben gondoskodnunk kell arról is, hogy míg az FFT algoritmus dolgozik, továbbra is legyen eredménye a FIR-szűrésnek. Ehhez az eddigieket csak kis mértékben kell kiegészíteni azzal, hogy az új $x_{i+1}(n)$ blokk N nagyságú részének első néhány eleménél még a konvolúciós szummázást folytatjuk $N+K$ FIR-szűrő együtthatóig, ahol K azon minták darabszáma, amennyi adat rögzítése után majd elvégződik az FFT algoritmussal való szűrés $x_i(n)$-re. A valós idejű késleltetésmentes FIR-szűrés érdekében használt parciális konvolúció ezzel a kiegészítéssel így módosul, az állandósult állapotban leírva:

$$y_p(n) = \sum_{j=K}^{n} h_j x_i(n-j) \tag{1.5}$$

$$n = K, K+1, K+2 \ldots N-1+K$$

A valós időjű működés ezen kiegészítése annyiban korlátozza az eljárást, hogy $y_i(n)$ és $y_{i-1}(n)$ közötti átlapolásnak nagyobbnak kell lennie, mint 50%. Ezt azért kell alkalmazni, hogy az N minta ideje alatt működő parciális konvolúciók fel tudják használni az előző OLA blokk szűrési eredményét. Kevesebb, mint 50%-os átlapolás esetén, az OLA szűrésének ideje tovább tartana, mint az átlapolási tartomány, és ekkor már a párhuzamosan működő parciális konvolúciók nem tudnák szolgáltatni a késleltetésmentes eredményt, helyettük teljes konvolúciót kellene végezni, ami a hatékony megvalósítás teljes kárára történhetne csak meg.

2. fejezet

Adaptív FIR-szűrők

Adaptív FIR-szűrőt egyaránt működtethetünk idő- vagy frekvenciatartományban, de ekkor a hatékonyság érdekében jól meg kell választanunk azt is, hogy milyen rendszeridentifikációs eljárást használjunk a működése során. A legkézenfekvőbb a legkisebb négyzetek módszerével történő adaptáció, mely időtartományban, de akár frekvenciatartományban is elvégezhető. Ebben a fejezetben bemutatom ezek működését, és kitérek arra is, hogy a dolgozatomban megvalósított valós idejű alkalmazásomhoz miért volt célszerűbb a frekvenciatartományban történő adaptációt választanom.

2.1. Rendszeridentifikációs módszerek

2.1.1. Időtartománybeli adaptáció (LMS)

A modellillesztés során történő paraméterbecslés alatt azt értjük, hogy a valóságos rendszer és a hozzá megválasztott modell kimeneteinek eltérését kívánjuk minimalizálni. A fizikai rendszer nem lesz ismert teljesen, ezért paramétereit nagy pontossággal nem lehet meghatározni, így meg kell, hogy elégedjünk egy olyan eredménnyel, amely a valóságos rendszert számunkra elfogadhatóan modellezi.

Ezen feladat során tehát a paraméterek becsléséhez a valóságos rendszer és az illesztett modell kimeneteinek különbségétől függő költségfüggvényt definiálunk, és a paraméterek változtatásával ennek minimalizálására törekedünk. Ezzel a modellből származó becsült kimenet a lehető legjobban közelít majd a valóságos rendszer kimenetét. [9]

$$\epsilon = E\left[(\boldsymbol{y} + \hat{\boldsymbol{y}})^T \cdot (\boldsymbol{y} + \hat{\boldsymbol{y}})\right] = E\left[e^2\right] \tag{2.1}$$

Cél a hibajel teljesítményének minimalizálása. Az illesztendő modell bemenete, kimenete és paraméterei közötti összefüggés a következő ($\boldsymbol{x} = f(\boldsymbol{u})$ a bemenőjel vektora, $\boldsymbol{w}$ a változó paraméterek vektora, $\hat{\boldsymbol{y}}$ a modell becsült kimenetének vektora):

$$\hat{\boldsymbol{y}} = \boldsymbol{w}^T f(\boldsymbol{u}) = \boldsymbol{w}^T \cdot \boldsymbol{x} \tag{2.2}$$

Egy bemenet és egy kimenet esetén az n-dik időpontban a becsült $\hat{y}(n)$ kiemenet az n-edik időpontban rendelkezésre álló M minta hosszú $\boldsymbol{w}(n)$ paraméter és $\boldsymbol{x}(n)$ regressziós

vektorok skaláris szorzataként számolható.

$$\hat{y}(n) = \boldsymbol{w}(n)^T \cdot \boldsymbol{x}(n) = \sum_{i=1}^{M-1} w_i(n)\, x_i(n) \tag{2.3}$$

Ezt felhasználva az előzőekben definiált költségfüggvény az alábbi alakra hozható:

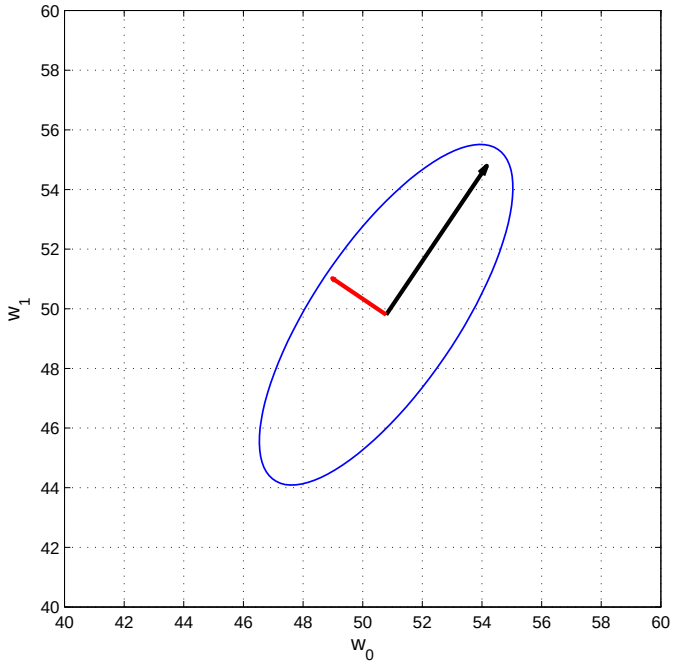
$$\epsilon = E\left[(y(n) + \hat{y}(n))^2\right] = E\left[\left(y(n) + \boldsymbol{w}(n)^T \boldsymbol{x}(n)\right)^2\right] \tag{2.4}$$

$$\epsilon = E\left[y(n)^2\right] - 2E\left[y(n)\,\boldsymbol{x}(n)^T\right]\boldsymbol{w}(n) + \boldsymbol{w}(n)^T E\left[\boldsymbol{x}(n)\,\boldsymbol{x}(n)^T\right]\boldsymbol{w}(n) \tag{2.5}$$

$$\epsilon = E\left[y(n)^2\right] - 2\boldsymbol{p}^T\boldsymbol{w}(n) + \boldsymbol{w}(n)^T\boldsymbol{R}\,\boldsymbol{w}(n) \tag{2.6}$$

Az eredményben azt kaptuk, hogy a költségfüggvényünkben megjelenik a kimenet négyzetes várható értéke $E\left[y(n)^2\right]$, a kimenet és a regressziós vektor közötti keresztkorrelációs vektor, $\boldsymbol{p}^T$, és a regressziós vektor autokorrelációs mátrixa, $\boldsymbol{R}$.

Ez a költségfüggvény egy M dimenziós ellipszis implicit egyenletét írja le, melyek változói a $\boldsymbol{w}$ pataméter vektor komponensei, és a főtengelyeinek irányát és nagyságát az autokorrelációs mátrix ($\boldsymbol{R}$) sajátvektorai és sajátértékei határozzák meg. Egy ilyen hibaellipszis számunkra legegyszerűbben abban az esetben vizualizálható a legkönnyebben, amikor a $\boldsymbol{w}$ paramétervektort úgy választjuk meg, hogy két komponenst tartalmaz, ugyanis ekkor egy 2 dimenzióban ábrázolható ellipszist ír le a költségfüggvény egyenlete [1].



2.1. ábra. *2 dimenziós hibaellipszis és a sajátvektorok ábrázolására egy példa*

Egy ilyen paraméterbecslési feladat során az a feladatunk, hogy megtaláljuk azt a $\boldsymbol{w}$

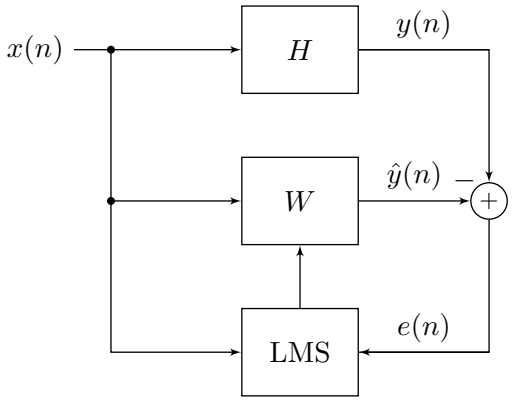
paramétervektort, amivel a valóságos rendszer a legjobban modellezhető matematikailag, más szóval a $\boldsymbol{w}$ paramétervektorban keressük azt az M darab szűrőegyüttható, amikkel a legnagyobb egyezést kaphatjuk a mérés során mért fizikai rendszerrel. Ezeket az optimális együtthatókat a költségfüggvény minimalizálásával kaphatjuk meg.

$$\frac{\partial \epsilon}{\partial \boldsymbol{w}} = -2\boldsymbol{p} + 2\boldsymbol{R}\,\boldsymbol{w} = 0 \tag{2.7}$$

$$\boldsymbol{w}_{opt} = \boldsymbol{R}^{-1}\boldsymbol{p} \tag{2.8}$$

Az előző fejezetben kapott keresztkorrelációs vektor és autokorrelációs mátrix inverzének számítása nagyon időigényes és nem célravezető megoldás. Ezért a költségfüggvényben az átlagos négyzetes hiba helyett csak a pillanatnyi hibát vesszük figyelembe, így a következő módosításokkal él a rekurzívan megoldható LMS-algoritmus (Least mean squares) [13]:

$$\hat{\boldsymbol{R}} = \boldsymbol{x}\,(n)\,\boldsymbol{x}\,(n)^T \qquad\qquad \hat{\boldsymbol{p}} = \boldsymbol{y}\,(n)\,\boldsymbol{x}\,(n)^T \tag{2.9}$$



2.2. ábra. *LMS algoritmus blokkvázlata*

$$e\,(n) = y\,(n) - \hat{\boldsymbol{w}}\,(n)^T\,\boldsymbol{x}\,(n) \tag{2.10}$$

$$\hat{\boldsymbol{w}}\,(n+1) = \hat{\boldsymbol{w}}\,(n) + 2\mu e\,(n)\,\boldsymbol{x}\,(n) \tag{2.11}$$

Az LMS-algoritmus által egy véges impulzusválaszú FIR-szűrőt kapunk a paraméterek becslése során. Minden iterációs lépésben az n-edik ütemben rendelkezésre álló szűrőegyüttható-készletet a $\hat{\boldsymbol{w}}\,(n)$ vektor definiálja. Az algoritmusban használt paramétertérben történő lépések mértékét μ (ez a lépésköz a költségfüggvény gradiens menti csökkentésének sebességét adja meg) az [1] hivatkozás alapján autokorrelációs mátrix legnagyobb sajátértékéből határozhatjuk meg, $0 < \mu < \frac{1}{\lambda_{max}}$. Viszont az előzőekben már kikötöttük, hogy ennek a mátrixnak a kiszámítása nem célravezető, így ezt kísérleti úton tudjuk csak meghatározni. A μ értékét célszerű a kívánt szűrőegyütthatók számából M megválasztanunk, vagyis az alábbi relációt betartva, tapasztalati úton megkeresni az optimális értékét annak érdekében, hogy az adaptív FIR-szűrő és a valóságos rendszer kimenetéből képzet hiba,

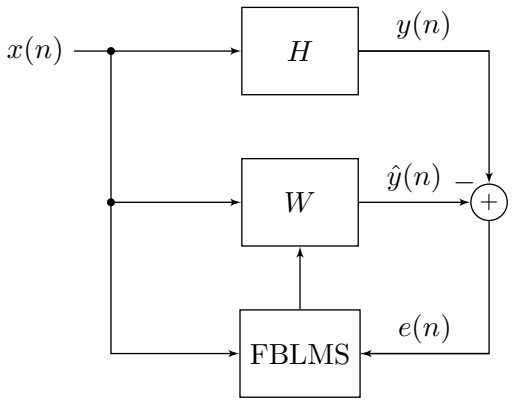
$e(n)$ a lehető legkisebb idő alatt csökkenjen le közel nulla értékre, azaz az algoritmus a lehető legjobb konvergenciasebességgel rendelkezzen.

$$0 < \mu \leq \frac{1}{M} \tag{2.12}$$

2.1.2. Frekvenciatartománybeli adaptáció (Fast Block LMS)

Számos adaptív szűrőt használó alkalmazásban, mint például az adaptív visszhangcsökkentésben és zajcsökkentésben igen nagy fokszámú FIR-szűrők adaptálására is szükség lehet [11] [4]. Az időtartományban működő nagy fokszámú együtthatók adaptálása az LMS algoritmussal már igen lassú lehet, így célszerűbb az eljárást blokkonként végezni [8].

Ezt a blokkonkénti LMS eljárást nevezzük Fast Block LMS (FBLMS) algoritmusnak, mely során a gyors Fourier transzformáció segítségével az együtthatók adaptálása a frekvenciatartományban történhet. Ez a frekvenciatartománybeli blokkonkénti adaptálás számottevően csökkentheti a számítási igényt, valamint a konvergenciasebességet is növelheti.



2.3. ábra. *FBLMS algoritmus blokkvázlata*

A frekvenciatartománybeli blokkos adaptáció során az adaptált FIR-szűrő együtthatókkal való szűrést kézenfekvő szintén frekvenciatartományban végrehajtani, az FFT algoritmus segítségével. Ehhez a dolgozatomban már bemutatott módszerek közül az Overlap-Save a legmegfelelőbb választás [14]. A következő blokkdiagram mutatja be az FBLMS algoritmust, és a további felsorolásban részletezem a működést lépésről lépésre.

1. A bemenő jel régi is új blokkjának összefűzése $2N$ méretű blokká.
2. A $2N$ méretű bemenő jel blokkjának transzformálása frekvenciatérbe FFT-vel.
3. Szűrés a frekvenciatérben az adaptív FIR-szűrő együtthatóival $\hat{W}(k)$.
4. A frekvenciatérben elvégzett szorzás eredményének inverz Fourier-transzformációja (IFFT) időtartományba.
5. Az utolsó blokk megtartása a kimenet eredményének, $\hat{y}(n)$.

6. Hibaképzés, a hibavektor, $e(n)$ kiszámítása a mért kimeneti vektor, $y(n)$ és az adaptív szűrés eredményvektora, $\hat{y}(n)$ segítségével.

7. Hibavektor, $e(n)$ elejének kiegészítése nullákkal, hogy biztosítva legyen a megegyező blokkméret a bemeneti blokkal.

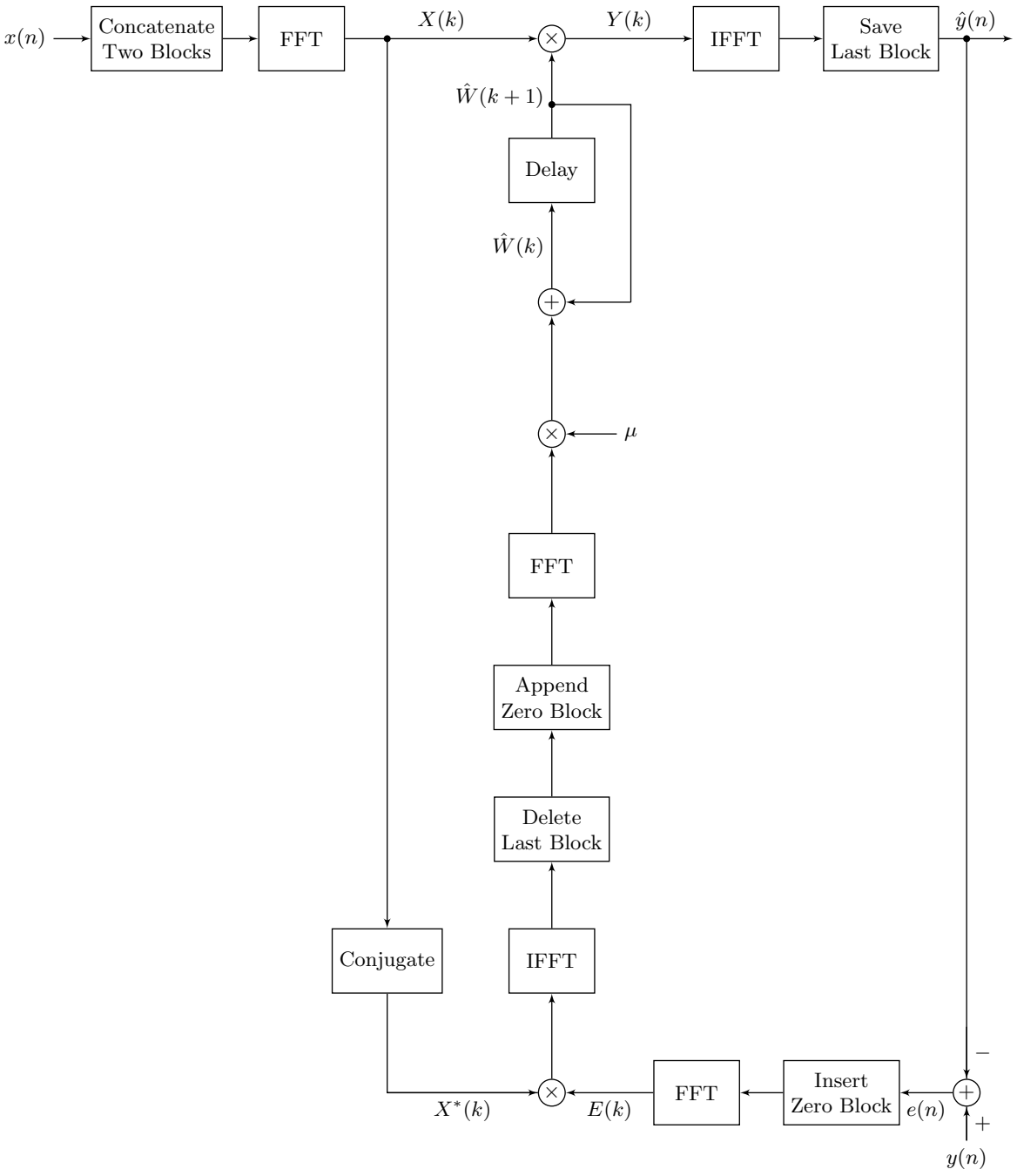
8. Hibavektor transzformálása frekvenciatérbe az FFT-vel.

9. Az eredmény összeszorzása a bemeneti blokk Fourier-transzformáltjának konjugáltjával a frekvencia térben.

10. Az szorzateredmény $X^*(k) \cdot E(k)$, vagyis a gradiens vektor inverz Fourier transzformációja.

11. Az időtartománybeli gradiens vektor utolsó blokkjának módosítása nulla elemekkel, majd ennek Fourier transzformálása újra a frekvenciatérbe.

12. A gradiensvektor szorzása a μ lépésközzel, majd hozzáadása az adaptív FIR-szűrőhöz $\hat{W}(k)$. Ez az utolsó lépés adaptálja az új együtthatókat, $\hat{W}(k+1)$.



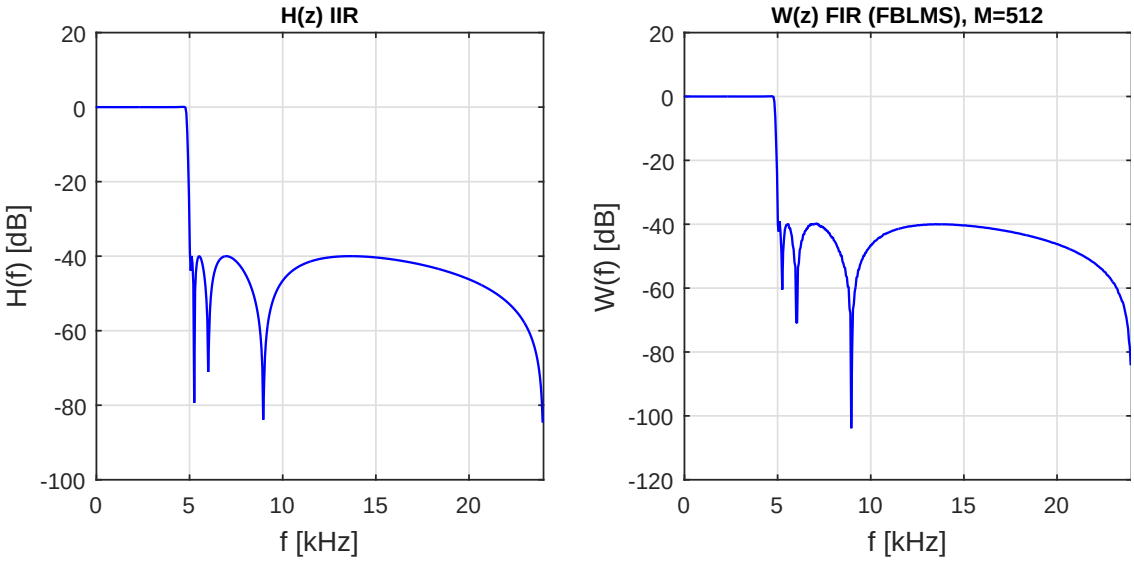
2.4. ábra. *Fast Block LMS algoritmus [12]*

2.2. LMS és Fast Block LMS algoritmusok összehasonlítása

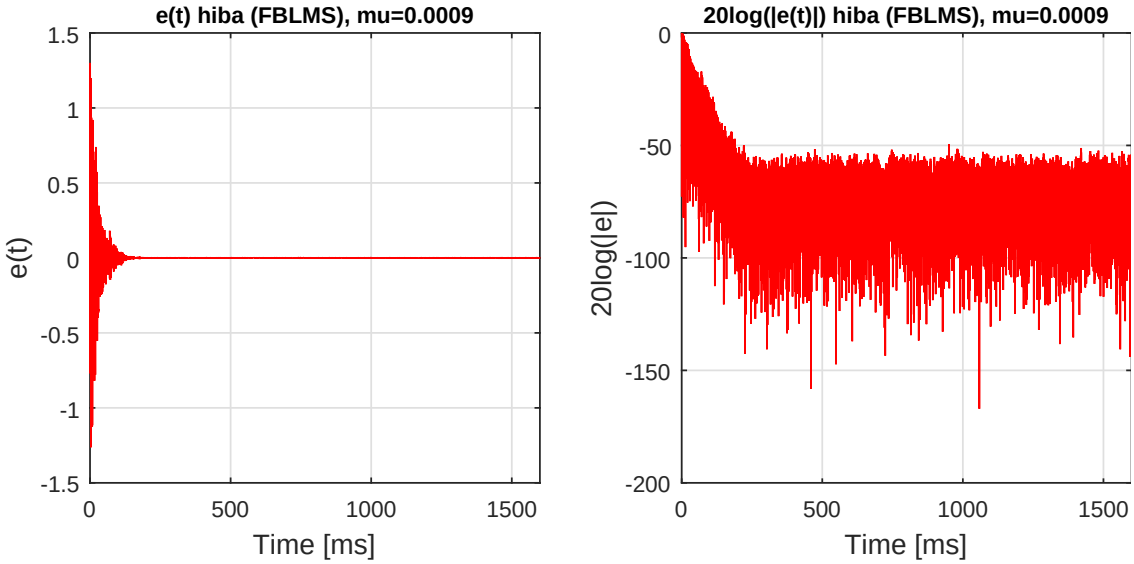
Az adaptív FIR-szűrők hatékony megvalósításához elengedhetetlen, hogy az adaptáció és a szűrés a frekvenciatartományban történjen. Ahhoz, hogy ezt a szükségességet jól demonstráljam, az LMS és FBLMS algoritmust megvalósítottam Matlab szimulációban is, és bemutatom a legfontosabb összehasonlítási eredményeket. Az LMS és az FBLMS közötti legfőbb eltéréseket pedig az alábbi felsorolásban részletezem.

- Az FBLMS algoritmus az együtthatókat a bemeneti jelből $x(n)$ vett blokkok segítségével végzi el (amik mérete megegyezik az adaptív szűrő hosszával), vagyis az együtthatók így blokkonként frissülnek. Ezzel szemben az időtartományban végzett LMS algoritmus esetén az együtthatók frissítése minden $x(n)$ mintavételezési pontban megtörténik. A blokkonkénti adaptációjának köszönhetően az FBLMS hibájának konvergenciasebessége nagyobb, mint az LMS algoritmus használata esetén.

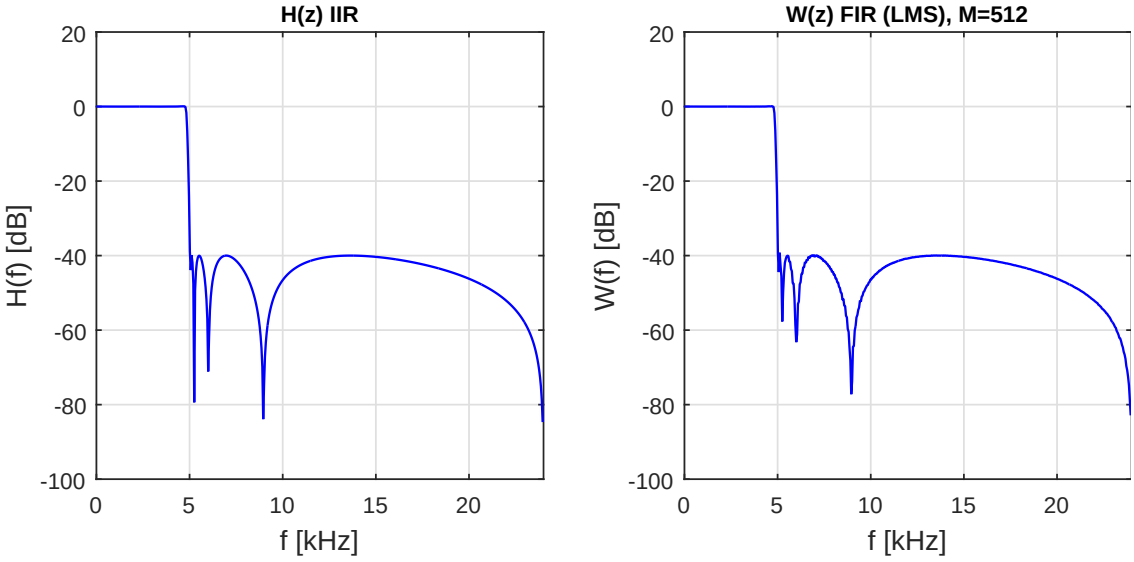
- Az FBLMS algoritmus kevesebb szorzási műveletet igényel, mint az időtartománybeli LMS. Abban az esetben, ha a blokkok mérete és az adaptív szűrő hossza, N megegyezik, akkor az LMS algoritmus $N(2N+1)$ db szorzási műveletet igényel, az FBLMS pedig csak $10N\log_2(2N)+26N$ szorzást. Konkrét értékkel bemutatva ez, azt jelenti hogy $N = 1024$ esetén az FBLMS algoritmus 16-szor gyorsabb, mint az LMS algoritmus. Ennek a hatékonyabb számítási eljárásnak köszönhetően adódik az, hogy a nagy fokszámú FIR-szűrők adaptálása esetén mindenképpen kifizetődőbb az FBLMS-sel a frekvenciatartományban elvégezni az adaptációt.



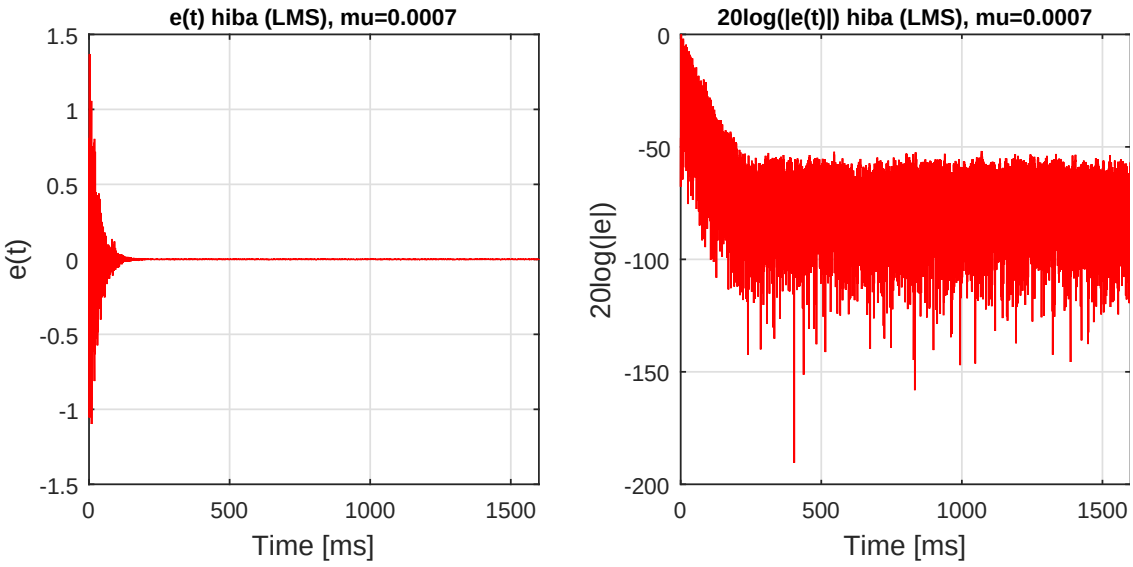
2.5. ábra. *Felhasznált IIR-szűrő és annak becsült adaptív FIR-szűrője az FBLMS algoritmussal, $f_s = 48kHz$*



2.6. ábra. *Hibalecsengés a Matlab szimulációban, FBLMS algoritmus esetén*



2.7. ábra. *Felhasznált IIR-szűrő és annak becsült adaptív FIR-szűrője az LMS algoritmussal, $f_s = 48kHz$*



2.8. ábra. *Hibalecsengés a Matlab szimulációban, LMS algoritmus esetén*

A Matlab szimulációs eredmények hibalecsengésein jól látszik, hogy a konvergenciasebesség az optimálisnak feltételezett μ lépésközöknél az FBLMS algoritmus esetén bizonyul valamivel jobbnak. A futási időket elemezve a különböző méretű FIR-szűrő együtthatók adaptálására, azt láthatjuk, hogy az FBLMS fölényesen győz az LMS-sel szemben. Ezekből az eredményekből látható, hogy az adaptív FIR-szűrők megvalósítása is csak úgy történhet hatékonyan, ha mind a szűrést mind pedig az adaptációt is a frekvenciatartományban végezzük.

N	N_{fft}	FBLMS futási idő [sec]	LMS futási idő [sec]
128	256	0.095	0.811
256	512	0.064	0.870
512	1024	0.049	0.992
1024	2048	0.043	1.239
2048	4096	0.041	1.910
4096	8192	0.041	4.607
8192	16384	0.044	7.118
16384	32768	0.043	11.049
32768	65536	0.041	16.600

2.1. táblázat. *FBLMS és LMS futási idő összehasonlítása* 204800 *minta nagyságú szűrt be- kimenetű adatpont adaptálására*

3. fejezet

Adaptív FIR-szűrő hatékony megvalósítása

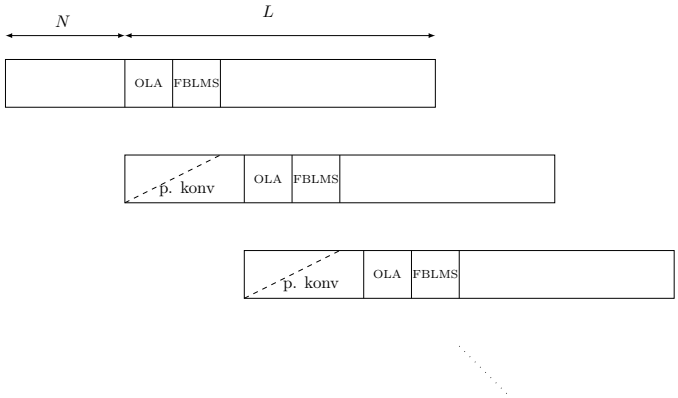
Az eddig bemutatott késleltetésmentes idő- és frekvenciatartományban működő FIR-szűréssel, valamint Fast Block LMS felhasználásával célom az volt, hogy megvalósítsak egy olyan adaptív FIR-szűrőt, amely nagy fokszámú együtthatók adaptálására is hatékonyan alkalmas, és emellett a tipikusan a frekvenciatartományban működő blokkosan végrehajtott konvolúciós eljárásoknál tapasztalható adatgyűjtés miatti késleltetéstől is mentes. Ebben a fejezetben részletezem ennek megvalósítási lehetőségét és a hatékonyságát.

3.1. Késleltetés nélküli szűrés alkalmazása adaptív FIR-szűrőhöz

Ahhoz, hogy az Overlap-Add eljárást és a parciális konvolúciót használó késleltetés nélküli FIR-szűrést adaptív FIR-szűrőkhöz használjuk, nem kell mást tennünk mint megfelelő programszervezéssel ezt egy Fast Block LMS algoritmussal kell párhuzamosan futtatnunk a valós idejű alkalmazás során.

Az 1. fejezetben szó volt arról, hogy gondoskodnunk kell arról is, hogy míg az Overlap-Add eljárás FFT algoritmusa dolgozik, továbbra is legyen eredménye a FIR-szűrésnek. Ez egyszerűen megoldható volt azzal, hogy a parciális konvolúciót az N mintagyűjtés után is folytattuk a következő blokk első néhány N minta adatgyűjtési ideje alatt, amíg az FFT algoritmus az előző bementi blokk szűrési eredményét el nem végezte, majd a parciális konvolúciót újra kezdtük az aktuális új blokkban.

Adaptív FIR-szűrő esetén ezt azzal tudjuk a legegyszerűbben kiegészíteni, hogy az Overlap-Add algoritmus után egyből, az újrakezdett parciális konvolúció alatt párhuzamosan a Fast Block LMS algoritmust folytatjuk, amely blokkonként végzi el az ismeretlen fizikai rendszer adaptációját. Ez a feladat egyébként a rendelkezésre álló rendszer és processzorarchitektúrától függően igen sokféleképpen valósítható meg. Ha több processzormagot is van módunkban használni a tervezendő rendszerünkben, akkor kézenfekvő megoldás lenne ezt a három algoritmust párhuzamosan futtatni. A munkám során egy Analog Devices SHARC ADSP-21364 jelprocesszor használatára volt lehetőségem, ami sajnálatos módon csak egy processzormaggal rendelkezik, így a legegyszerűbb nem többmagos meg-



3.1. ábra. *Késleltetés nélküli adaptív FIR-szűrő, OLA-t és FBLMS-t használva*

valósítás volt csak elérhető számomra a fejlesztés során.

A fejlesztésem során Multitasking-gal rendelkező operációsrendszer sem állt rendelkezésemre az ADSP-21364 jelprocesszor fejlesztő környezetén, így ennek a három algoritmusnak az egymás melletti működését a lehető legegyszerűbben törekedtem megvalósítani. Ez a lehető legegyszerűbb megoldás így csak az lehetett, hogy az adaptálást végző Fast Block LMS algoritmusom minden Overlap-Add szűrési eljárás után kezdi meg működését. Vagyis ez után, az új mintáknál fellépő, megszakítások alatt elvégződő újrakezdett parciális konvolúciók után a következő minta érkezéséig a fennmaradó szabad processzoridőben az előző blokkra a Fast Block LMS elkezdheti a részszámításokat, majd egy bizonyos mintavételezési szám után elkészül ezzel, és megtörténik az adaptív FIR-szűrő együtthatóinak frissítése, amit már azonnal a következő OLA eljárásnál és parciális konvolúciónál fel is lehet használni.

Az adaptív FIR-szűrő hatékony megvalósításánál a cél az, hogy a blokkos végrehajtás alatt működő módszerek a lehető legnagyobb számú L együtthatóra tudjanak működni. Ezt az igényt elsősorban az $N_{fft} = N + L$ blokkméretet használó OLA és FBLMS eljárás limitálja, mert a véges memóriaterület és processzorkapacitás miatt tetszőlegesen nagy N_{fft} méretű blokkokra nem tudjuk elvégezni a műveleteket. Másodsorban az is limitálja a maximálisan elérhető L együtthatót, hogy az N blokkméret csak limitált méretig csökkenthető a módszer során, mivel ezen N minta beérkezési ideje alatt az előző x_{i-1} blokkon végzett szűrést az OLA algoritmusnak és a blokkos adaptációt az FBLMS-nek el kell tudnia végezni a megszakítások után fennmaradó szabad időben még az előtt, hogy az éppen aktuális új N mintával feltöltődő x_i-edik blokkon ezek meg nem kezdenék az új szűrést és adaptációt.

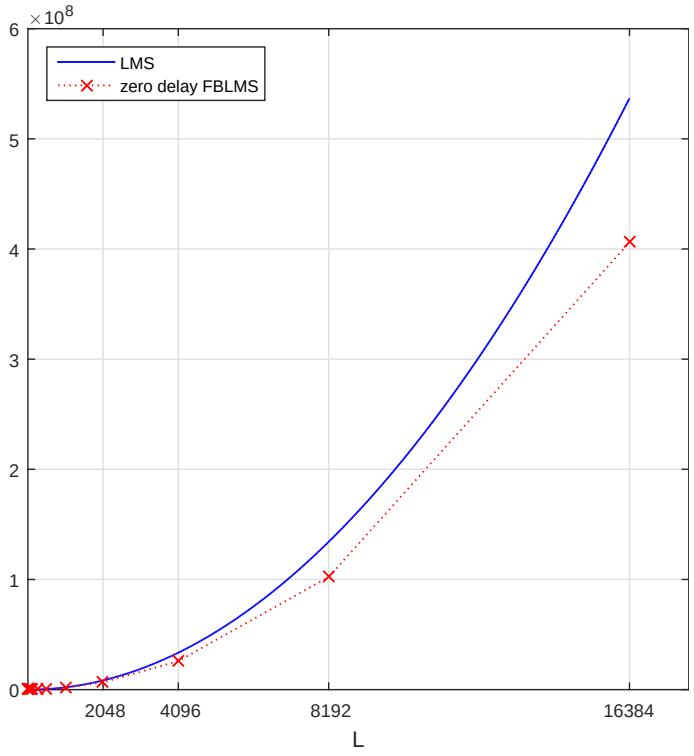
Ezt, hogy hogyan lehet a maximális L együtthatót megbecsülni az általam javasolt adaptív, késleltetés nélküli FIR-szűrőm használatakor egy adott memóriával és processzorteljesítménnyel rendelkező rendszerben, a következő alfejezetben fejtem ki bővebben.

3.2. Idő- és frekvenciatartománybeli, késleltetésmentes adaptív FIR-szűrés hatékony megvalósítása

Az időtartományban működő LMS algoritmussal való adaptálást az idő- és a frekvenciatartományban működő FBLMS-sel való adaptálással összehasonlítva kijelenthető az, hogy egy bizonyos L nagyságú FIR-szűrő adaptálása felett mindenképpen megéri az általam megvalósított idő- és a frekvenciatartománybeli, késleltetésmentes adaptív FIR-szűrőt használni, mert az FBLMS és az OLA eljárás még mindig elég hatékonyan képes csökkenteni a számítási igényt a parciális konvolúció mellett is. A két módszer számításigénye L függvényében az alábbi 3.2. ábrán látható (LMS-t használva a műveletigény N_{LMS}, FBLMS-t használva pedig N_{zerod}).

$$N_{LMS} = L(2L+1)$$

$$N_{zerod} = 12L\log_2(2L) + 27L + \frac{3}{2}L^2$$



3.2. ábra. *A késleltetésmentesen működő adaptív FIR-szűrő műveletigényének összehasomnlítása LMS és FBLMS algoritmust használva az L adaptált együtthatók függvényében (FBLMS-nél 50%-os átlapolással, $N_{fft} = L + L$).*

Annak érdekében, hogy hatékonyan a lehető legnagyobb L együtthatót tudjuk adaptálni, meg kell becsülnünk az adaptív FIR-szűrő maximális együtthatószámát egy adott N_{fft} blokkméret és processzorteljesítmény mellett. Ehhez paraméteresen ábrázoltam a processzor kihasznált számítási teljesítményét a mintavételi idők függvényében, amin további számításokkal ez a becslés jó közelítés eredményre vezethet. A modell során használt pa-

raméterek értelmezései a következők:

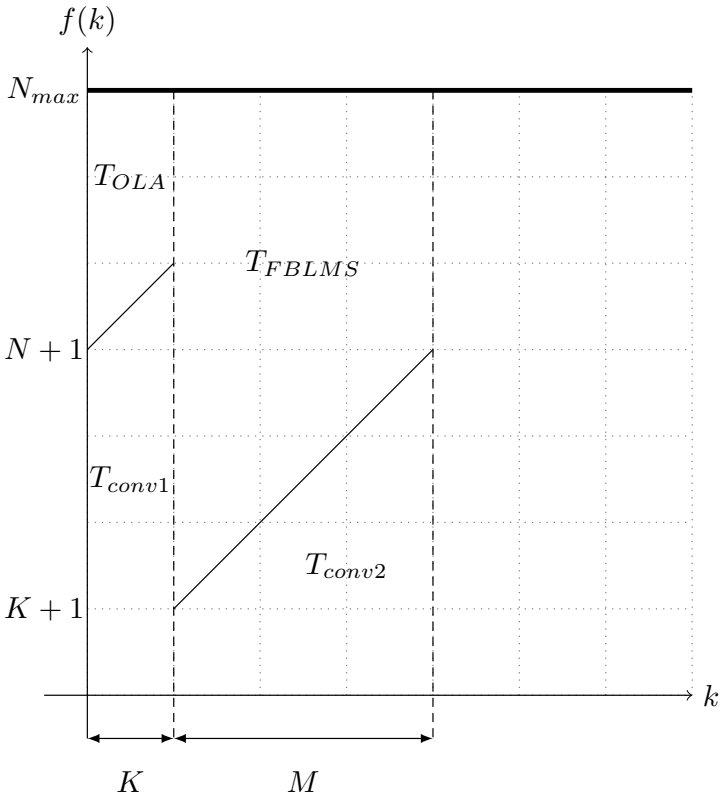
- k: A már mintavételezett minták száma az OLA blokk elejétől számolva.
- N_{max}: Egy adott k-adik mintavétel után a maximálisan elvégezhető műveletek száma a következő $k+1$ minta beérkezéséig. Ez a processzor teljesítményéből és a használt fs mintavételi frekvenciából származtatva egy állandó paraméter.
- N: Az OLA blokkos végrehajtásánál, egy blokkban mintavételezett bemeneti értékek száma.
- K: Az OLA algoritmus futási ideje alatt, a beérkező új minták száma.
- M: Az FBLMS algoritmus futási ideje alatt, a beérkező új minták száma.
- N_{OLA}: Az OLA algoritmus elvégzéséhez szükséges műveletek száma.
- N_{FBLMS}: Az FBLMS algoritmus elvégzéséhez szükséges műveletek száma.
- N_{fft}: Az OLA és FBLMS algoritmusok blokkméreteinek mintanagysága, amely a rendelkezésre álló maximális memóriaterület alapján szintén egy állandó paraméter.

A modell segítségével az a cél, hogy L együtthatószám maximális értéke megközelítőleg meghatározható legyen, vagyis az $N_{fft} = N + L$ összefüggés alapján N minimális értéke a kérdés, ami még épp elegendő lehet ahhoz, hogy a K és M mintaszámig tartó OLA és FBLMS algoritmus befejeződjön ezen minták beérkezése és parciális konvolúció számítása alatt. Tehát a modell során azzal a feltétellel élhetünk, hogy $K + M \leq N$.

A továbbiakban még azt tudjuk, hogy az OLA algoritmus kezdetekor a még folytatott parciális konvolúció szummájának felső határa megegyezik $N+1$-gyel, vagyis ekkor $N+1$ szorzást és összeadást végez el a processzor. Az is adódik, hogy a SIMD (single instruction, multiple data) módot kihasználva a jelprocesszorban, a parciális konvolúciós szumma felső határa minden egyes új minta beérkezése után 1-gyel növekszik, mivel eggyel több mintára végződik majd el a konvolúció. Ez a művelet igény egészen K minta beérkezéséig növekszik lineárisan 1 meredekséggel, amikor is befejeződik az OLA algoritmus, és elkezdhető a parciális konvolúció újbóli futtatása, immár $K+1$ felső szumma határral kezdve. Az ezután elkezdődő FBLMS algoritmus még M minta beérkezéséig fut a megszakításokban működő parciális konvolúciók mellett, aminek továbbra is 1-gyel nő a konvolúciós szumma felső határa minden új mintavételezés után.

Az előzőekben vázolt a priori információk alapján könnyen ábrázolhatjuk a 3.3. ábrán a beérkezett minták számában azt, hogy a két minta beérkezése között a processzornak mennyi további számítási kapacitása van a megszakításokban futó parciális konvolúciók mellett, az OLA és az FBLMS algoritmusok számára.

Az ábrán jól látszódik, hogy a parciális konvolúció kezdeti $N+1$ felső szumma határa, az OLA és az FBLMS algoritmushoz tartozó K és M mintányi futási időt befolyásolja, vagyis $K(N)$ és $M(N)$ értékei függnek N-től. Ahhoz hogy megkapjuk az összefüggést $K(N)$ és $M(N)$-re, egy egyszerű geometriai problémát kell megoldani az ábrán.



3.3. ábra. *A processzor kihasználtságának aránya a mintavételi időpontok függvényében.*

Láthatjuk hogy a $K+M$ minta beérkezése alatt a processzor $N_{max} \cdot (K+M)$ műveletet hajt végre. És azt is tudjuk, hogy az ez idő alatt a parciális konvolúciók során elvégzett műveletek összes száma (ami a lineáris növekedések alatti terület) plusz az N_{fft} blokkméretekre végzett OLA és FBLMS algoritmusok műveletszáma pontosan meg kell, hogy egyezzen az $N_{max} \cdot (K+M)$ összesen elvégzett művelet számával.

$$N_{max}(K+M) = T_{OLA} + T_{FBLMS} + T_{conv1} + T_{conv2} \tag{3.1}$$

$$T_{OLA} = N_{OLA} \qquad T_{FBLMS} = N_{FBLMS}$$

A területegyezésekből felírható egyenlet révén $K(N)$ és $M(N)$ kifejezhető egy adott N_{max} és N_{fft} állandó érték mellett. Ehhez nem kell más tennünk, csak integrálás útján meg kell határoznunk paraméteresen T_{conv1} és T_{conv2}-őt, és ez után a következő két egyenlet segítségével $K(N)$ és $M(N)$ N-től valló függését ki kell fejeznünk.

$$T_{conv1} = \int_0^K k + (N+1)dk = \frac{K^2}{2} + K(N+1) \tag{3.2}$$

$$T_{conv2} = \int_{K}^{K+M} k + (K+1)dk = \frac{M^2}{2} + 2KM + M \tag{3.3}$$

$$N_{max}K = N_{OLA} + T_{conv1} = \frac{K^2}{2} + K(N+1) + N_{OLA} \tag{3.4}$$

$$N_{max}M = N_{FBLMS} + T_{conv2} = \frac{M^2}{2} + 2KM + M + N_{FBLMS} \tag{3.5}$$

Az utóbbi két egyenletből $K(N)$ megkapható a másodfokú polinom gyökének paraméteres kifejezésével, és ezután $K(N)$-et felhasználva ugyanígy megkapható az $M(N)$-re való összefüggés is.

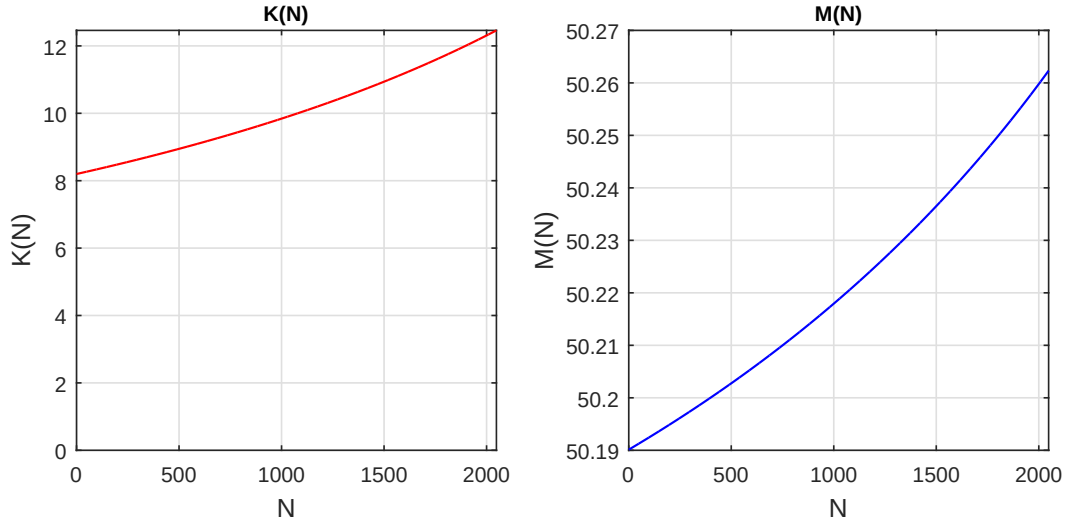
$$K(N) = (N_{max} - N - 1) \mp \sqrt{(N_{max} - N - 1)^2 - 2N_{OLA}} \tag{3.6}$$

$$M(N) = -(2K + 1 - N_{max}) \pm \sqrt{(2K + 1 - N_{max})^2 - 2N_{FBLMS}} \tag{3.7}$$

$$N_{OLA} = N_{fft} \log_2(N_{fft})$$

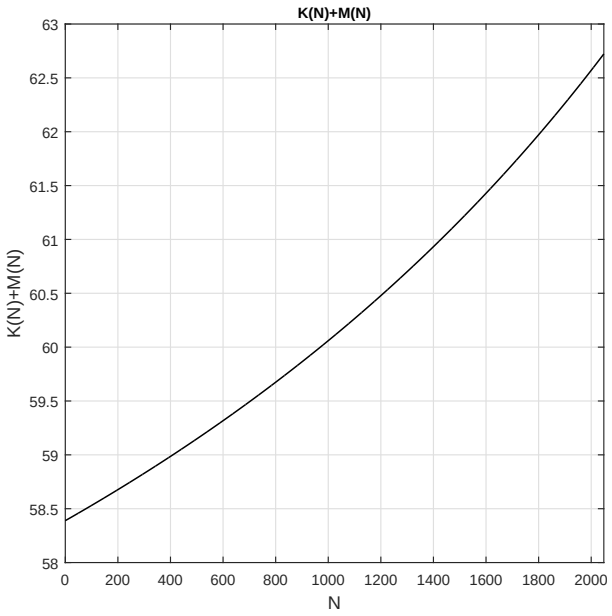
$$N_{FBLMS} = 5N_{fft} \log_2(N_{fft}) + 13N_{fft}$$

Ennek az eredménynek a hatására $K(N)$ és $M(N)$ függését N-től most már ismerjük egy adott processzorhoz és egy adott N_{fft} blokkméretre használva a módszert, tehát ha $f(N) = K(N) + M(N)$-t ábrázoljuk, akkor az adott fix paraméterekkel N_{fft} és N_{max}-xal, grafikusan megkaphatjuk azt az optimális N-et, ahol teljesül a $K(N)+M(N) \leq N$ reláció, és az adaptív FIR-szűrő L együtthatóinak számát a maximálisra választhatjuk.

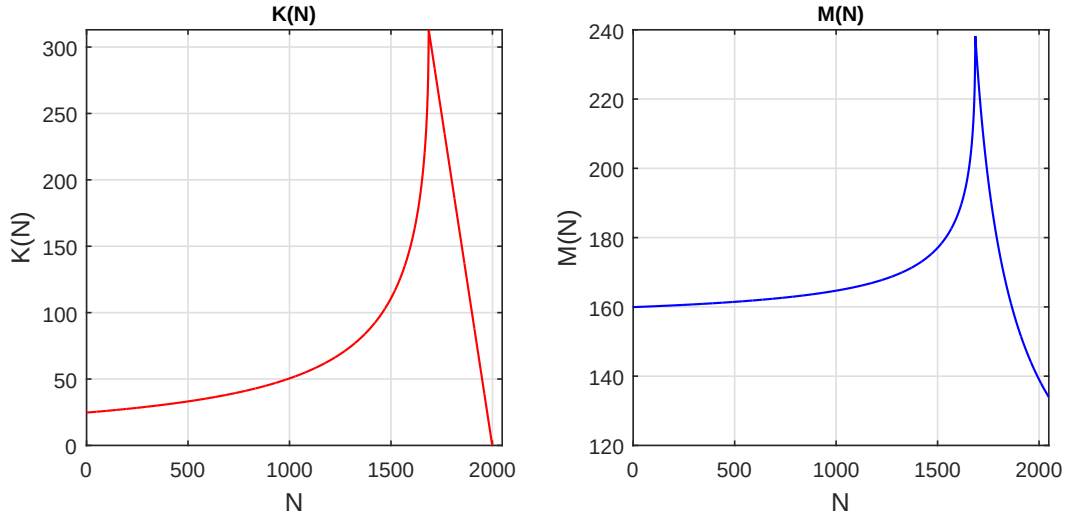


3.4. ábra. *$K(N)$ és $M(N)$ ábrázolása $N_{fft} = 4096$ és $N_{max} = 6000$ esetén.*

Konkrét példákkal ábrázolva az eredményt, $N_{max} = 2000$ és $N_{fft} = 4096$ rendszerpa-



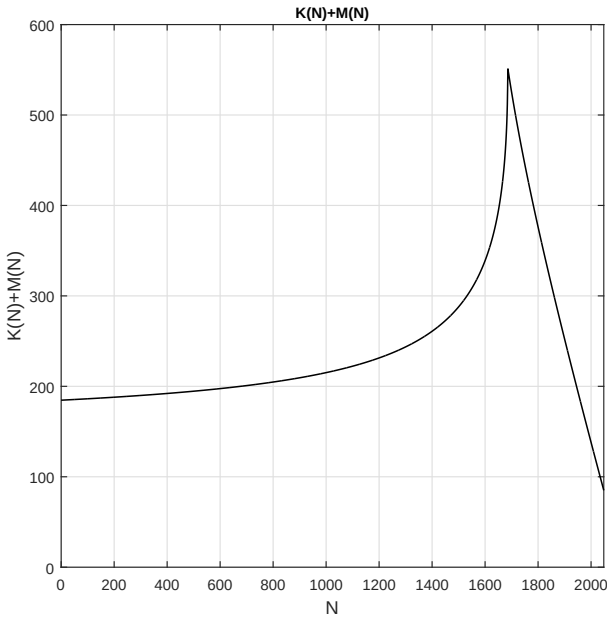
3.5. ábra. *$K(N) + M(N)$ ábrázolása $N_{fft} = 4096$ és $N_{max} = 6000$ esetén.*



3.6. ábra. *$K(N)$ és $M(N)$ ábrázolása $N_{fft} = 4096$ és $N_{max} = 2000$ esetén.*

raméterek esetén még az is jól megfigyelhető, hogy melyik az a maximálisan használható N méret, amikor még a megszakításokban futó parciális konvolúció szummájának felső határa az OLA algoritmus befejezésének pillanatában pont akkora, hogy a processzornak még éppen legyen rá kapacitása, hogy ezt elvégezze. Az ábrákon ez a $N + K(N) = N_{max}$ határesethez tartozó pont (csúcspont). Ha ennél a pontnál magasabbra választjuk meg N-et, a parciális konvolúció olyan nagy felső szumma határról indul, hogy ez már az OLA algoritmus befejezésének ideje előtt telítésbe viszi a processzorkihasználást, és egy mintavételi idő alatt már nem lesz rá elegendő kapacitása, hogy a processzor kiszámolja ezt. A matematikai modell esetén ez a határeset úgy jelentkezik, hogy ennél nagyobb N használatánál egyszerűen az eredmény komplexszé válik (csak a valós részt ábrázolva ez a

tartomány a csúcs utáni hirtelen csökkenő rész).



3.7. ábra. *$K(N) + M(N)$ ábrázolása $N_{fft} = 4096$ és $N_{max} = 2000$ esetén.*

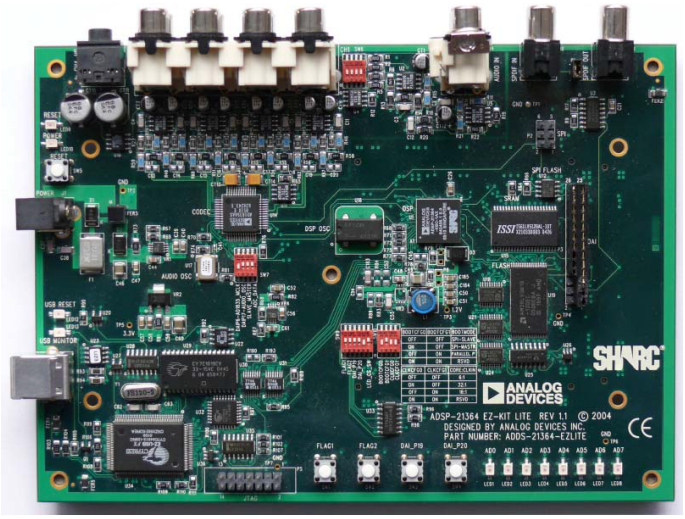
3.3. Késleltetésmentes adaptív FIR-szűrés implementálása ADSP-21364 jelprocesszoron

Az idő- és a frekvenciatartománybeli, késleltetésmentesen működő adaptív FIR-szűrést ADSP-21364 jelprocesszorra fejlesztettem ki, amelyhez a Méréstechnika és Információs Rendszerek Tanszék DSP laborjában rendelkezésre áll több ADSP-21364 EZ-KIT Lite fejlesztőkártya.

Ezen az ADSP-21364 EZ-KIT Lite fejlesztőkártyán az ADSP-21364 jelfeldolgozó processzorra történő fejlesztés hatékony és gyors. A fejlesztőkártyán kialakításra került a jelfeldolgozó processzor minden olyan szükséges perifériája és környezete, amelyre a leggyakoribb feladatok elvégzésekor szükség lehet. Az analóg és digitális hang ki- és bemenetek, LED-ek, kapcsolók, nyomógombok azonnal hozzáférhetőek, így a szoftverfejlesztés mindenféle hardvertervezési lépés nélkül azonnal elkezdhető, a fejlesztőkártya VisualDSP++ programozási környezetében. A kártya a következő hardverelemeket bocsátja a fejlesztők rendelkezésére. [13]

- Analog Devices ADSP-21364 processzor
- 4 Mbit SRAM
- 8 Mbit flash memória
- 2 Mbit SPI által kezelhető flash memória
- Analóg hanginterfész (AD1835A kodek, 1 db sztereó bemenet, 4 db sztereó kimenet)

- Digitális hanginterfész (1 db bemenet, 1 db kimenet)
- 11 db LED (ebből 1 db power, 1 db board reset, 1 db USB monitor, 8 db általános célú)
- 5 db nyomógomb (1 db reset, 2 db DAI lábra, 2 db FLAG lábra kötve)
- Bővítőinterfész (párhuzamos port, FLAG-ek, DAI, SPI)
- JTAG emulátor port
- USB port a PC csatlakoztatásához.

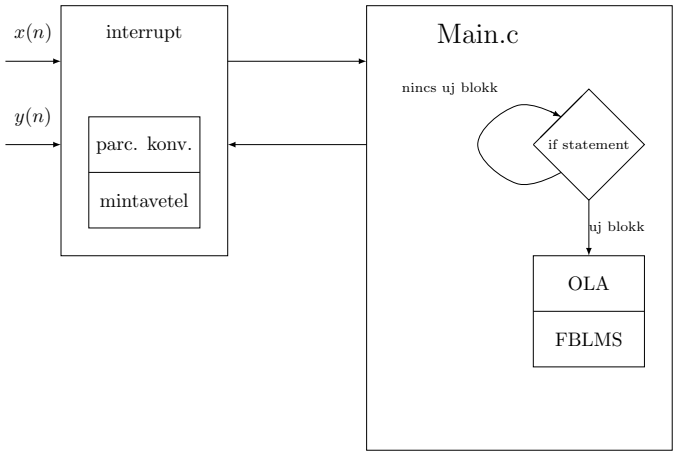


3.8. ábra. *ADSP-21364 EZ-KIT Lite fejlesztőkártya*

A késleltetésmentes adaptív FIR-szűrést egy igen egyszerű programszervezésben oldottam meg. Az adaptív szűrés alatt a gerjesztés, $x(n)$ és a rendszer válasza, $y(n)$ mintavételezése megszakítások után történik, a mintavételi frekvenciáknak megfelőlen, valamint ekkor történik meg a parciális koncolúcióval való FIR-szűrés is, ami egy mintavételi idő alatt bőven elkészül, és így az előző OLA blokk szűrési eredményének segítségével a D/A kimeneteire szolgáltatja a szűrés eredményét, adatgyűjtési késleltetés nélkül.

Mialatt a mintavételezések és a parciális konvolúciók a megszakítási időpontokban megtörténnek, a main függvény végtelen ciklusában az OLA és FBLMS algoritmusok várják blokkjaik feltöltődését, és amikor egy új blokk N mintával feltöltődik, akkor ezek ebben a ciklusban egymás után futtatásra kerülnek egy feltételvizsgálat révén. A mintavételi idők között, amikor már megtörténtek a mintavételezések és konvolúciók a megszakításokban, ezek tovább folytatják futásukat, és $K + M$ minta alatt befejeződnek, még a következő bemeneti blokkjuk feltöltése előtt, amikor futtatásuk újból megtörténik.

A fejlesztés során hamar kiderült, hogy az OLA és FBLMS algoritmusok igen memóriaigényessé teszik a módszeremet a blokkos végrehajtások miatt. Ezért a programfejlesztés során törekedtem minél jobban és hatékonyabban kihasználni a rendelkezésemre álló memóriakapacitást. Első lépésként arra törekedtem, hogy az OLA és FBLMS eljárások



3.9. ábra. *Programszervezés*

alatt működő FFT algoritmusok, átmeneti tömböket nem igénylő "in place" FFT-k legyenek. Később az is kiderült a fejlesztőkörnyezet alapos megismerése után, hogy a 4 Mbit SRAM-hoz képest, alapértelmezésben sokkal kevesebb memória használható a programfejlesztés során, így elengedhetetlen lett a fejlesztőkörnyezetben ennek átkonfigurálása is, amit végül a 3 Mbit nagyságig sikerült megnövelni. Ezeknek a módosításoknak köszönhetően maximálisan $N_{fft} = 4096$ méretű blokkokat tudtam használni az adaptív FIR-szűrő működtetésekor.

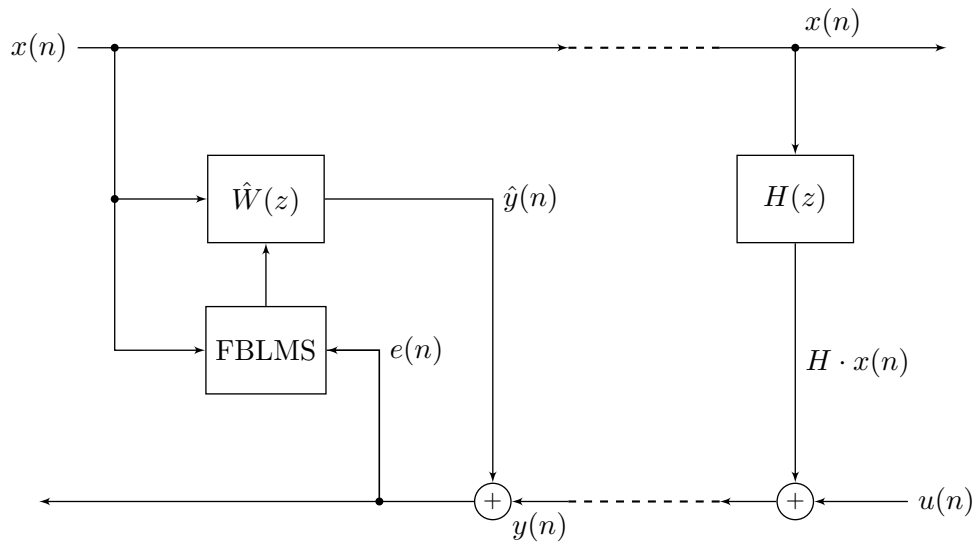
Az adaptív FIR-szűrő együtthatószáma, L, a blokkos végrehajtás miatt korlátozott az N_{fft} blokkméret által, mert ennél a méretnél csak kisebb alkalmazható a szűrés során. Az előző alfejezetben ábrázolt hatékonyságbeli különbségnél azt lehetett tapasztalni, hogy az $L = 4096$ együttható szám az, ami felett az elméleti számolások alapján a módszernek egyre jobban és hatékonyabban kellene működnie az időtartománybeli adaptív FIR-szűrőhöz képest. A rendelkezésemre álló ADSP-21364 processzoron ennek a hatékonyságnak az igazolása nem ígérkezett egyszerű feladatnak. Ezt a hatékonyságot és helyes működést mérésekkel próbáltam demonstrálni egy adaptív FIR-szűrőt működtető visszhangcsökkentő struktúrában, melynek eredményeit a következő fejezetben mutatom be.

4. fejezet

Adaptív FIR-szűrő alkalmazás

4.1. Adaptív visszhangcsökkentés

A fejlesztőkártyához már rendelkezésre állt egy adaptív visszhangcsökkentő algoritmus, amely az Információfeldolgozás laboratórium egyik mérési feladatában szerepel az időtartományban adaptáló LMS-algoritmussal megvalósítva. A helyes működést és a hatékony megvalósítást ezzel vetettem össze a munkám végén.

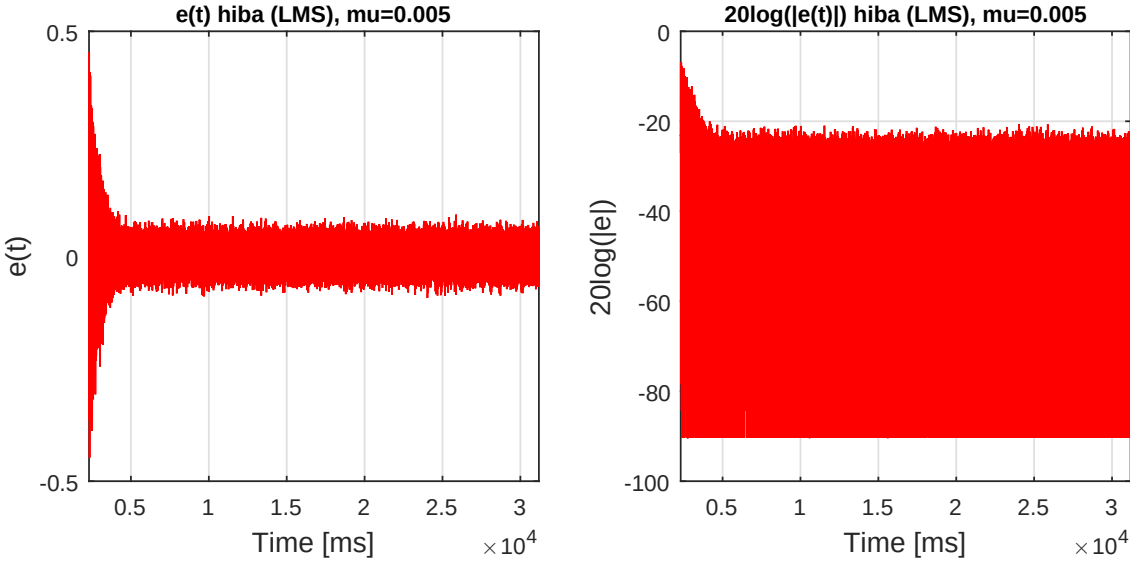


4.1. ábra. *Az adaptív visszhangcsökkentő struktúra blokkvázlata FBLMS használatával*

Az adaptív szűrők vizsgálata című méréséhez használható hibrid áramkör [13] segítségével megvalósítható a fenti ábrán látható struktúra. Ebben az alkalmazásban az adaptív szűrővel a feladat az, hogy a beszélő saját hangja, $x(n)$, amely visszhangként hallható vissza a saját oldalán, ne legyen hallható. Ehhez a vevő oldalán az adaptív szűrőnek adaptálnia kell a másik oldali $H(z)$ átviteli rendszert (amiben az akusztikai közeg is beletartozik, például egy szoba), hogy a beszélő oldalán az adaptált $\hat{W}(z)$ FIR-szűrővel való szűrés eredményével (ami a becsült visszhang) elnyomható legyen ez a visszhang.

4.2. Adaptív visszhangcsökkentés az ADSP-21364 jelprocesszoron, idő- és a frekvenciatartománybeli, késleltetésmentes adaptív FIR-szűrővel

Az általam megvalósított késleltetésmentes adaptív FIR-szűrőt olyan módon teszteltem, hogy a visszhangcsökkentő struktúrában egy zajgenerátort használtam az $x(n)$ bemeneten. Ezt az $x(n)$ bemenetet hangszórón kiadva egy mikrofonon keresztül visszavezettem az $u(n)$ jelként, és így próbáltam megvalósítani azt a valós helyzetet, amikor a másik oldali beszélőnél visszhang keletkezik, és más beszédforrás nem hallható éppen. Így egy valódi akusztikai rendszeren áthaladó visszhangot tudtam demonstrálni, és ennek elnyomását tudtam tesztelni a visszavezető ágban, és lévén, hogy más hangforrás nem adódott hozzá $u(n)$-hez, a hibalecsengés jól megvizsgálhatóvá vált. A helyes eredménynek ilyenkor annak kell lenni, hogy a hibakimeneten, $e(n)$-en a visszhangnak teljesen meg kell szűnni, ami ebben az esetben a zajgenerátorból származó zaj volt. Ezzel a teszttel az időtartománybeli LMS-sel való megegyező működés jól ellenőrizhető volt, és a rendszerek konvergenciasebességének összehasonlítása is megtörténhetett.

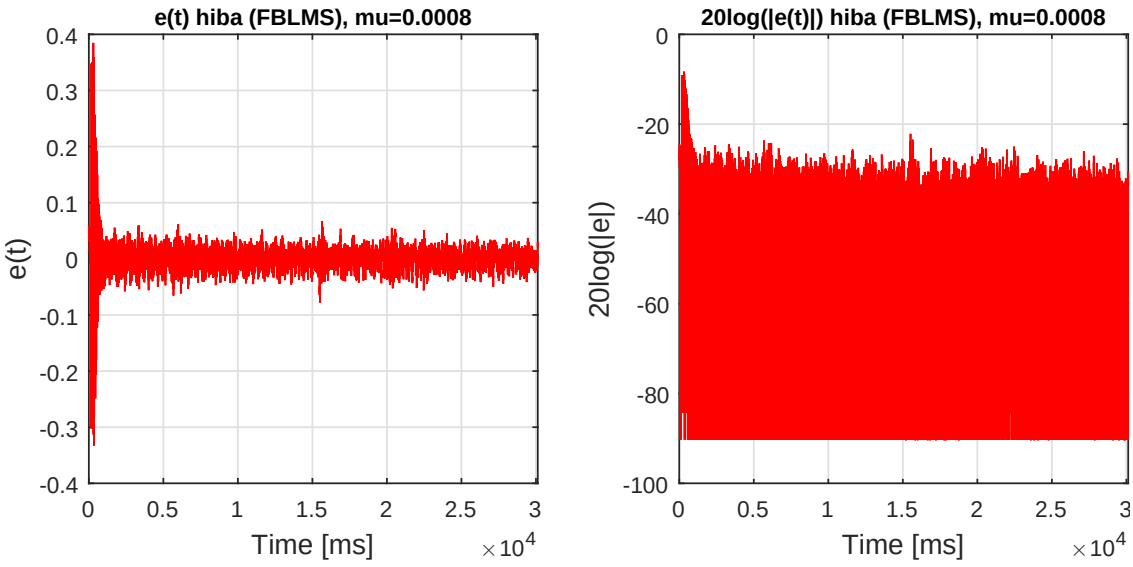


4.2. ábra. *Hibalecsengés az LMS algoritmust használva*

A 4.2. és 4.3. ábrákon látható mért eredmények a helyes működést jól igazolják. Az eredményben az is nagyon jól észrevehető, hogy az optimális μ-t használva mind a két módszerben, a konvergenciasebességekben jól megmutatkozik az, hogy az FBLMS ezt hatékonyabban tudja biztosítani, és ezáltal vele gyorsabb hibalecsengés érhető el.

Az ADSP-21364 jelprocesszoron elérhető 3 Mbit memória miatt viszont nem adódott lehetőségem arra, hogy a maximálisan elérhető szűrőegyütthatószámot bemutassam a késleltetésmentes FBLMS-algoritmussal működő adaptív FIR-szűrőmhöz. A blokkos műveletek sajnos emiatt csak $N_{fft} = 4096$ méretben hajtható végre, így az L együtthatószám nem növelhető meg nagyobbra, noha a processzor vélhetően nagyobb blokkos feldolgozására is alkalmas lenne.

A továbbiakban még egy összehasonlítást végeztem arra az esetre, amikor nagyobb min-



4.3. ábra. *Hibalecsengés az FBLMS algoritmust használva, a késleltetésmentes adaptív FIR-szűrővel*

f_s [kHz]	N_{fft}	L_{FBLMS}	L_{LMS}
48	4096	3750	9940
96	4096	2772	4010

4.1. táblázat. *Maximálisan elérhető szűrőegyütthatószám a késleltetésmentes FBLMS és LMS-algoritmusokkal az ADSP-21364 jelprocesszoron.*

tavételi frekvencián történik az adaptálás. A táblázatban szereplő eredményekből látható, hogy ha az időtartománybeli LMS-sel, kétszeres mintavételi frekvencián működik az adaptív szűrő, akkor ehhez mérten megközelítőleg a felére csökken a maximálisan adaptálható együtthatók száma. Az FBLMS-t használva kétszeres mintavételi frekvencián ez a csökkenés jóval kevesebb, ekkor a maximálisan adaptálható együtthatók száma csak a negyedével csökken le.

Összefoglalás

Dolgozatomban az adaptív FIR-szűrők hatékony megvalósításának módját vizsgáltam, amivel az akusztikus alkalmazásokban használt nagy fokszámú adaptív FIR-szűrők együtthatóigényének problémája megoldható. Munkám során körüljártam, hogy az idő- és frekvenciatartományban milyen eljárások szolgálnak az adaptív FIR-szűrők implementálására és ezek milyen hátrányokkal és előnyökkel járnak egymáshoz képest.

A szakirodalmat körüljárva egy olyan idő- és frekvenciatartományban működő eljárást dolgoztam ki, amivel megvalósíthatók a nagy fokszámú késleltetésmentesen működtethető adaptív FIR-szűrők. Elméleti számításaimmal igazoltam ennek hatékonyságát, és egy módszert is bemutattam, amivel egy adott processzoron és hardverkörnyezetben megbecsülhető rögzített blokkméret mellett a maximálisan elérhető együtthatók száma az adaptáció során, majd egy jelprocesszoron a helyes működést demonstráltam.

A fejlesztések során természetesen újabb és újabb a hatékonyságot növelhető ötletek fogalmazódtak meg. Közvetlen továbblépési lehetőség ugyanennek az algoritmusnak az ADSP Sharc processzorcsalád valamely több memóriával rendelkező tagján történő megvalósítása. Ez esetben érdemi programfejlesztés nélkül lehet újabb eredményeket elérni. További cél lehet a más platformon pl. FPGA-n, grafikus processzoron történő megvalósítás. Egy más irányú továbblépés lehet olyan zajcsökkentő rendszer megvalósítása, illetve zajelnyomási feladatok megoldása, amelyekre csak ezen hatékony struktúra alkalmazásával nyílik lehetőség.

Irodalomjegyzék

[1] Peter N. Stearns Bernard Widrow. *Adaptive Signal Processing.* Prentice-Hall, New Jersey, 1985.

[2] E. O. Brigham. *The Fast Fourier Transform.* Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

[3] Tukey John W. Cooley James W. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, April 1965.

[4] Nelson P.A Elliot S.J. Active noise control. *Signal Processing Magazine, IEEE*, 10(4):12–35, October 1993.

[5] E. Ferrara. Fast implementation of lms adaptive filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):474–475, August 1980.

[6] Guillermo Garcia. Optimal filter partition for efficient convolution with short input/output delay. In *Audio Engineering Society Convention 113*, Oct 2002.

[7] W. G. Gardner. Efficient convolution without input-output delay. *Audio Eng. Soc.*, 43(3):127–136, March 1995.

[8] National Instruments. *Least Mean Squares Algorithms (Adaptive Filter Toolkit).* National Instruments Ltd, June 2008.

[9] BME MIT Tanszéki Munkaközösség. Digitális jelfeldolgozás, 2006.

[10] Oppenheim and Shafer. *Digital Signal Processing.* Prentice-Hall, New Jersey, 1975.

[11] D. R. Morgan S. M. Kuo. Active noise control: a tutorial review. *Proceedings of the IEEE*, 87(6):943–973, June 1999.

[12] John J. Shynk. Frequency-domain and multirate adaptive filtering. *IEEE Signal Processing Magazine*, 9(1):14–37, January 1992.

[13] Balogh Tibor Sujbert László. Adaptív szűrők vizsgálata - 3. mérés, 2010.

[14] Aurelio Uncini. *Fundamentals of Adaptive Signal Processing.* Springer, Signals and Communication Technology Series, London, 2015.