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

Elosztott adaptív jelfeldolgozó rendszerek vizsgálata

DIPLOMATERV

Készítette
Varga Balázs

Konzulens
Dr. Orosz György

2018. május 28.

HALLGATÓI NYILATKOZAT

Alulírott *Varga Balázs*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2018. május 28.

Varga Balázs
hallgató

Tartalomjegyzék

Kivonat

Elosztott rendszernek nevezzük a hálózatba kapcsolt, egymással kommunikáló, közös feladat megoldásán dolgozó számítógépek összességét. Ez a számítási paradigma teszi lehetővé többek között a mikrobiológiai vagy csillagászati számításokat végző számítógép-klaszterek működését, a peer-to-peer fájlmegosztást, illetve a kriptovaluta-tranzakciók megmásíthatatlan könyvelését.

Egy algoritmus elosztott architektúrába történő szervezésének leggyakoribb motivációja a magas erőforrásigény elosztása, valamint a rendszer hibatűrésének javítása. Ezen tényezők a digitális jel- és információfeldolgozó rendszerekben is igen gyakran megjelennek – gondoljunk csak az önvezető járművek szenzorai által érzékelt temérdek adatra, és az ezek alapján autonóm módon meghozandó döntésekre.

Dolgozatomban digitális jelfeldolgozási algoritmusok elosztott módon történő megvalósíthatóságának lehetőségét vizsgálom egy konkrét alkalmazás, a többcsatornás aktív zajcsökkentés példáján keresztül. Részletesen ismertetem a témakörben használatos algoritmusokat, majd bemutatom egy működő, elosztott aktív zajcsökkentő mintarendszer implementálásának lépéseit.

Abstract

A distributed system is a collection of networked computers communicating with each other in order to achieve a common goal. This computing paradigm enables computer clusters to work on complicated microbiological or astronomical computations, it makes peer-to-peer file sharing possible, and it is also what guarantees the authenticity of cryptocurrency transaction ledgers.

The most common motivations for implementing an algorithm in a distributed fashion are improved fault tolerance and the requirement for distributing the computational load among multiple computers. These factors also play an important role in several digital signal and information processing systems – just think of the vast amount of data collected by the sensors of a self-driving vehicle, and the autonomous decisions-making based on this data.

The main focus of this thesis is the implementation of distributed digital signal processing algorithms, illustrated by a specific example application: multiple channel active noise cancellation. In my work, I first introduce and analyze the algorithms most commonly used in such applications, then I describe the steps of implementing a working prototype of a distributed active noise cancellation system.

1. fejezet

Bevezetés, célkitűzések

1.1. Elosztott rendszerek, elosztott jelfeldolgozás

Az *elosztott rendszer (distributed system)* fogalmára nem létezik a tudományos irodalom által egységesen elfogadott, precíz definíció, de általánosságban kijelenthetjük: elosztott rendszerről beszélünk, ha több, egymással összeköttetésben álló, egymással kommunikáló számítógép azonos feladaton, azonos cél érdekében dolgozik [1]. Számítógép alatt itt számos különböző architektúrájú feldolgozó egységet érthetünk a hagyományos PC-től kezdve, az általános célú mikrokontrolleren és a grafikus processzoron át, akár az imperatív kódot nem is futtató FPGA áramkörig. Sőt egyes szakemberek szorosan csatolt elosztott rendszernek tekintik a fizikailag ugyanazon eszközön, de több végrehajtási szálon futó alkalmazásokat is. Jelen dolgozat keretében ezzel szemben csak az egymástól térben elkülönülő, hálózatba kapcsolt feldolgozó egységek rendszerét tekintjük elosztottnak.

Az ily módon definiált elosztott rendszerek kialakításának céljai – és egyben a velük azonos feladatot ellátó, nem elosztott architektúrájú rendszerekkel szembeni előnyei – a következők lehetnek:

- **Számítási teljesítmény fajlagos költségének csökkentése.** Egy különösen nagy erőforrásigényű feladat rendkívül magas költségvonzatú – vagy akár teljesíthetetlen – követelményeket támaszthat az elvégzésére dedikált számítógéppel szemben. Ilyen esetekben érdemes megvizsgálni, hogy a probléma vajon dekomponálható-e olyan egyszerűbb részfeladatokra, amelyek „kiszervezhetők" különálló, olcsóbb számítógépekre.

- **Erőforrások hatékonyabb allokálása és kihasználása.** Centralizált rendszerekben gyakran előforduló probléma, hogy egyes erőforrások (CPU, memória, háttértár, kommunikációs sávszélesség stb.) szűk keresztmetszetté válnak, miközben mások kihasználatlanul maradnak. Ezzel szemben egy elosztott ar-

chitektúrában az erőforrások szétosztására sok esetben flexibilisebb lehetőségek kínálkoznak.

- **Skálázhatóság.** Egy jól megtervezett elosztott rendszer elemeinek száma – és esetleg kapcsolódásuk topológiája – könnyen megváltoztatható. Ezáltal lehetőség nyílik a rendszer dinamikus fejlesztésére, valamint új feladatok elvégzésére történő felkészítésére.

- **Hibatűrés, redundancia.** Centralizált rendszerek általános problémája, hogy egy központi elem meghibásodása akár a rendszer teljes funkcionalitásának kiesését is okozhatja. Ezzel szemben az elosztott rendszerek nyilvánvaló előnye a redundáns működés kialakításának lehetősége.

- **Biztonság.** A közelmúltban óriási népszerűségre szert tevő blockchain alapú technológiák remek példái olyan elosztott rendszereknek, amelyek kialakításának fő szempontja a biztonság volt.

Az előnyök mellett az elosztott rendszerek hátrányait is meg kell említenünk: egy elosztott számítási architektúra kiépítése és üzemeltetése, valamint egy algoritmus decentralizált végrehajtásának megtervezése erősen specializált szaktudást igényel. Továbbá az utolsóként említett előny egyben hátrány is lehet: egy elosztott rendszer általában nyíltabb struktúrájú, mint a centralizált megfelelője, ez a tény pedig egyes alkalmazásokban biztonsági aggályokat vethet fel.

Néhány közkeletű példa elosztott rendszerekre:

- Peer-to-peer fájlmegosztás
- Elosztott adatbázisok
- Pénzügyi, banki információs rendszerek
- Blockchain technológiák (kriptovaluták megmásíthatatlan tranzakciói, okosszerződések, önvezető járművek közti kommunikáció stb.)
- Cluster computing, grid computing

Nem utolsó sorban pedig természetesen a digitális jelfeldolgozás, adatfeldolgozás területén is kiemelkedő szerep jut az elosztott rendszereknek. Ez nem meglepő annak tükrében, hogy a technológiai fejlődésnek köszönhetően egyre több és több információ zúdul ránk, mi pedig szeretnénk abból egyre többet és többet feldolgozni. Ha belegondolunk, hogy egy „önvezető” (részben autonóm működésű) városi autó mennyi különböző szenzor segítségével érzékeli a környezetét (kamera, LIDAR, radar, ultrahangos távolságmérők, GPS, iránytű, gyorsulásmérő, giroszkóp stb.), akkor

szintén nem meglepő az a tény, mely szerint egy ilyen jármű másodpercenként háromnegyed gigabyte adatot gyűjt [2]. Ez az az adatmennyiség, amelyet érzékelőnként nagyon különböző előfeldolgozási feladatoknak kell alávetni, majd tanulóalgoritmusokon alapuló osztályozás után ebből kell rövid idő alatt meghozni egy olyan döntést, amin potenciálisan emberéletek múlhatnak. Nyilvánvaló, hogy egy ehhez hasonló alkalmazásban az elosztott rendszerek céljai/előnyei között felsorolt tulajdonságok mindegyike kiemelt szerepet kap.

A digitális jelfeldolgozás és információfeldolgozás tudományába tartozó feladatok köre igen sokrétű, és nem ritkán rendkívül számításigényes. Néhány példa a teljesség igénye nélkül:

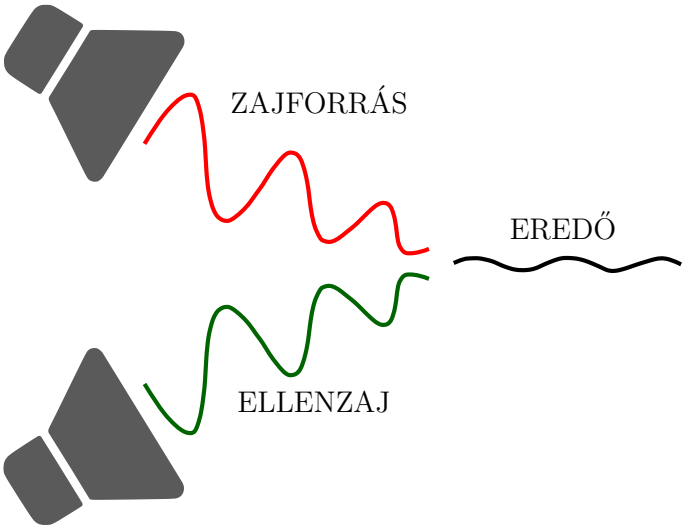
- Szűrési feladatok egy és több dimenzióban (lineáris FIR és IIR szűrők, mediánszűrő és egyéb nemlineáris szűrők, Gábor-szűrő)
- Jeltranszformációk (Fourier-transzformáció, koszinusz transzformáció, wavelet transzformáció, Hilbert-transzformáció, Radon-transzformáció, főkomponens-analízis)
- Adaptív algoritmusok (LMS típusú algoritmusok, rezonátoros jelfeldolgozás, Kálmán-szűrő)
- Tanulóalgoritmusok (klaszterezések és egyéb nem felügyelt módszerek, döntési fa, diszkriminanciaanalízis, előrecsatolt, rekurrens és konvolúciós neurális hálózatok, szupport vektor gép)

Az alábbiakban egy konkrét jelfeldolgozási problémát mutatok be részletesebben, amely az elosztott jelfeldolgozó rendszerek tanulmányozására több szempontból is alkalmas.

1.2. Aktív zajcsökkentés

Akusztikus zajok elnyomására több lehetőség is kínálkozik. Passzív módszereknek nevezzük azokat a megoldásokat, amelyek a közvetlen zajforrás által keltett hanghullámoknak a vizsgált pontba való terjedését igyekeznek megakadályozni. Ide soroljuk többek között a járművek kipufogóit, a forgalmas utak zajcsillapító falait, a hangszigetelő falakat és nyílászárókat.

Aktív zajcsökkentésről beszélünk, ha a zajforrás jelét egy tudatosan generált „ellenzaj" segítségével szeretnénk kioltani, amely a kérdéses pontban (a kioltási helyen) éppen ellentétes fázisban találkozik az eredeti zajjal, destruktív interferenciát hozva létre [3]. Az elvet az 1.1. ábra szemlélteti.



1.1. ábra. *Az aktív zajcsökkentés elve*

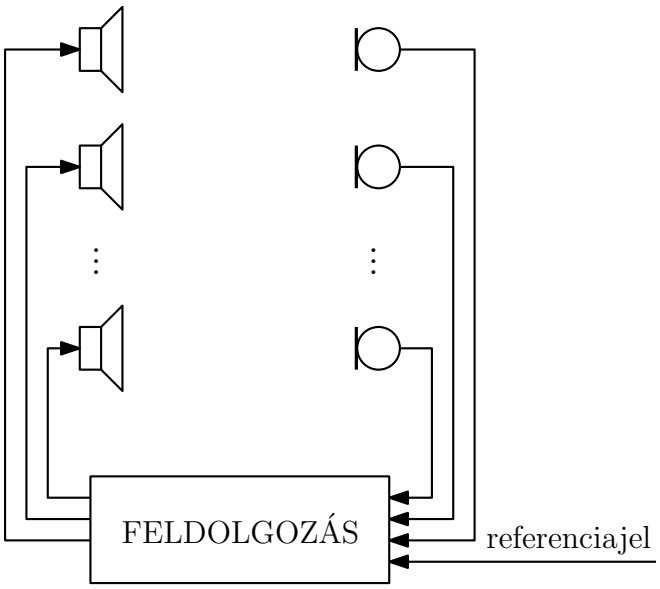
Aktív zajcsökkentést alkalmazó komfortjavító megoldásokkal találkozhatunk például egyes autók és repülőgépek utasterében (menetzaj/hajtóműzaj csökkentése a fejtámlákban elhelyezett hangszórók segítségével), valamint fejhallgatókban és elektronikus orvosi fonendoszkópokban [4].

Az aktív zajcsökkentő rendszer egy szabályozási kört valósít meg: a kívánt kioltási helyen egy mikrofonnal folyamatosan méri a zaj teljesítményét, és közben saját hangszórójának segítségével olyan jelet igyekszik generálni, amely a destruktív interferencia révén minimalizálja ezt a teljesítményt. A módszer elvéből adódóan a zavarás kioltása a legegyszerűbb esetben a tér egyetlen pontjában (illetve annak a hang hullámhosszával összemérhető kiterjedésű környezetében) valósul meg. Ezzel szemben a gyakorlatban természetesen többnyire egyél több pontban kívánjuk megvalósítani a kioltást, ez pedig több hangszórót és mikrofont – valamint összetettebb algoritmust – igényel. Így tehát a feladat hamar az 1.2. ábrán látható architektúrává skálázódik fel[1].

Éppen ez a tulajdonsága teszi az aktív zajcsökkentést az elosztott jelfeldolgozó rendszerek kiváló tesztalkalmazásává: egy jól skálázható (sőt természetes módon skálázódó), több bemenetű, több kimenetű (MIMO) rendszerről van szó, amely – mint később látni fogjuk – több lépésben is decentralizálható.

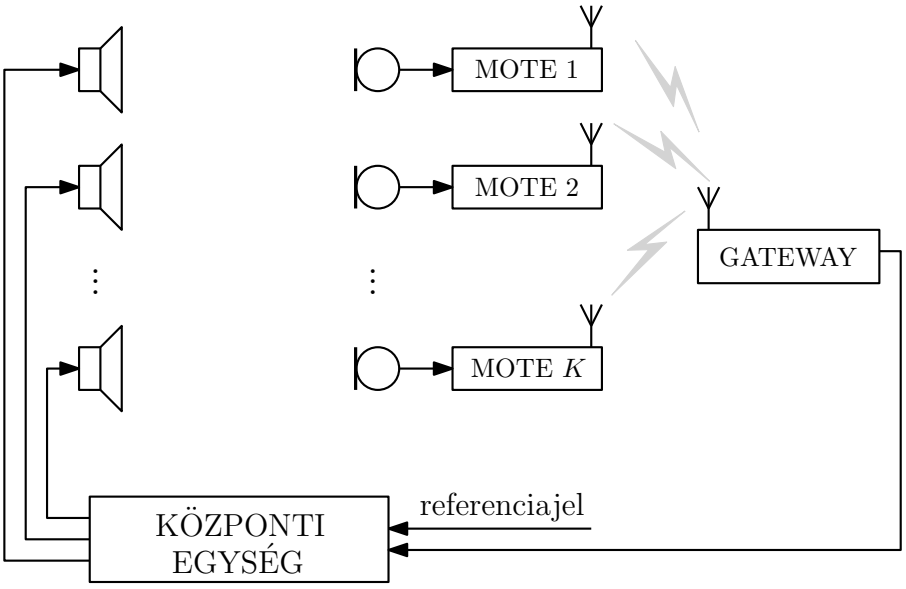
Az 1.2. ábrán látható, legalapvetőbb felépítésű zajcsökkentő rendszer teljes mértékben centralizált: minden mérést, jelfeldolgozást, beavatkozást egy központi feldolgozó egység végez, amely tipikusan egy jelfeldolgozó processzort és egy hozzá illesztett többcsatornás audio codecet tartalmaz.

[1] Az ábrán feltüntetett „referenciajel” felirat magyarázata, hogy az aktív zajcsökkentést megvalósító algoritmusnak szüksége van a zavarást generáló forrás jelére (az algoritmust a következő fejezetben részletesen ismertetem).



1.2. ábra. *Teljesen centralizált architektúra*

A centralizált rendszerek már ismertetett hátrányait kis részben orvosolhatjuk, ha az 1.3. ábrának megfelelően megtesszük az első lépést az elosztott működés irányába, és egy vezeték nélküli szenzorhálózatot alakítunk ki [5].

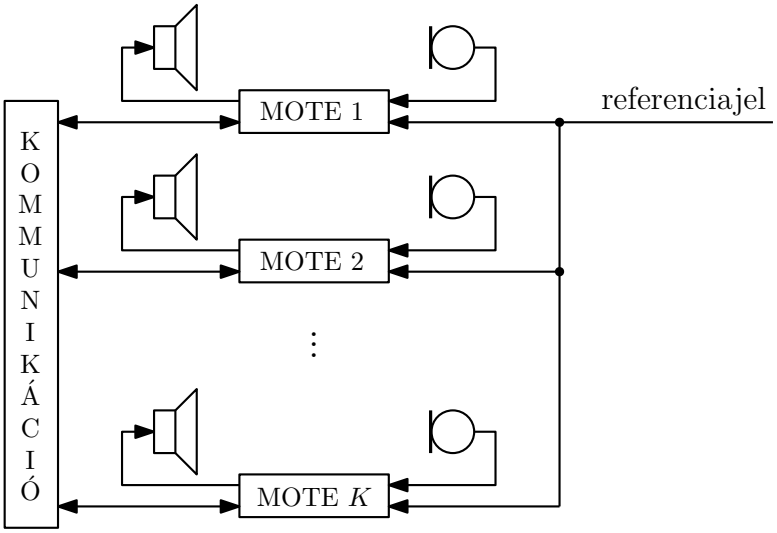


1.3. ábra. *Részben elosztott architektúra*

Látható, hogy itt minden egyes mikrofonhoz tartozik egy kis mikroprocesszoros egység (*mote* – így nevezzük a szenzorhálózatok elemeit), amely a mintavételezett jelet vezeték nélküli hálózaton továbbítja egy dedikált vevőegységen keresztül a központi egység felé. A lényegi algoritmus futtatását ebben a modellben is ez a központi egység végzi, azonban a mintavételezés – és opcionálisan némi előfeldolgozás, adattömörítés – már a mote-okon történik. Az erős centralizáltság azonban még mindig

jellemző az architektúrára, a központi egység kiesése esetén továbbra sincs esély a működés fenntartására. (Sőt, az ábrán vázolt topológiában a gateway kiesése esetén sincs rá esély.)

Az 1.4. ábra az immár teljesen elosztott működésű architektúrát mutatja, ebben már nem található semmilyen kritikus, centrális szerepű elem[2]. Az aktív zajcsökkentést megvalósító algoritmus teljes mértékben az egyes mote-okon, elosztott módon fut. A konkrét algoritmus részletezése nélkül a mote-ok szükséges számítási teljesítményéről és a rendszer hibatűrő képességéről természetesen nem tudunk értekezni, azonban annyi kijelenthető, hogy az algoritmus hatékony dekomponálása esetén a rendszer komponenseinek összköltsége csökkenhet a centralizált esethez képest (egy drága DSP-t sok olcsó mote-tal váltunk ki). Továbbá az elvi lehetőség megvan valamilyen szintű hibatűrő működés kialakítására is.



1.4. ábra. *Teljesen elosztott architektúra*

A [6, 7] publikációkban a Valenciai Műszaki Egyetem kutatói bemutatnak egy szimulációk során működőképesnek mutatkozó elosztott aktív zajcsökkentési algoritmust, amelynek magas szintű vázlata teljes mértékben megfeleltethető az 1.4. ábrának. A cikkben ismertetett algoritmus ugyan a jelen dolgozatban bemutatottaktól némileg eltérő módon (blokkosan, frekvenciatartományban implementált szűrésekkel) működik, az elosztott architektúra megvalósíthatóságáról mindenképpen pozitívan tanúskodik.

[2]A KOMMUNIKÁCIÓ feliratú dobozt tekintsük a kommunikációs folyamat általános modelljének, így abba tetszőlegesen redundáns hálózati topológia beleérthető.

1.3. Célkitűzések

Munkám kezdetekor célul tűztem ki, hogy megismerkedem az aktív zajcsökkentés elméletével, valamint a megvalósításához legszélesebb körben használt (LMS típusú) algoritmusokkal, méghozzá az 1.1-1.4. ábrákon vázolt „lépcsőfokokat" bejárva: az egycsatornás alapesettől eljutva a többcsatornás, elosztottan működő algoritmusig. A különböző módszerek tulajdonságait először numerikus szimulációk segítségével vizsgáltam, majd elkészítettem egy elosztott, kétcsatornás aktív zajcsökkentő mintarendszer működő prototípusát.

Dolgozatom 2. fejezetében részletesen bemutatom az elosztott aktív zajcsökkentés implementálásához szükséges algoritmusokat, viselkedésüket MATLAB szimulációkkal illusztrálom.

A 3. fejezetben ismertetem a gyakorlati implementáció platformjául szolgáló hardvereszközök, valamint a fejlesztéshez szükséges szoftvereszközök kiválasztásának folyamatát.

A 4. fejezetben bemutatom a megvalósítás menetét, kitérve néhány érdekesebb implementációs sajátosságra.

Az 5. fejezetben bemutatom az elkészült, működő rendszert, valamint az azon elvégzett mérések eredményeit.

Végül a 6. fejezetben értékelem az elért eredményeket, és felvázolom a téma esetleges folytatásának, jövőbeli kutatásának lehetséges perspektíváit.

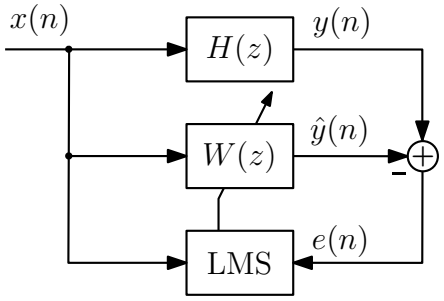
2. fejezet

A vizsgált algoritmusok áttekintése

Az aktív zajcsökkentés témakörében használatos adaptív módszerek többsége az *LMS (Least Mean Squares)* típusú algoritmusok közül kerül ki. Ezeknek az iteratív gradiensmódszereknek az első változatát az 1960-as évek elején dolgozta ki a Stanford Egyetem oktatója, Bernard Widrow, valamint PhD. hallgatója, Ted Hoff [8]. Azóta ezek az algoritmusok nagy népszerűségre tettek szert, és számos különböző változatuk született – ilyen módszerekkel találkozhatunk például neurális hálózatok tanítása során, illetve bizonyos szűrőtervezési feladatoknál is [9].

2.1. Rendszeridentifikáció LMS algoritmussal

Az egyszerű LMS algoritmus feladata a $H(z)$ diszkrét idejű átviteli függvénnyel jellemezhető lineáris, időinvariáns (LTI) rendszer identifikációja, azaz annak a $W(z)$ szűrőnek a megkeresése, amely közös gerjesztés esetén a $H(z)$ rendszer kimenetéhez képest minimális négyzetes hibájú kimenetet produkál.



2.1. ábra. *Az LMS algoritmus vázlata*

Az algoritmus működési vázlata a 2.1. ábrán látható. Az identifikálandó $H(z)$ rendszert és a keresett $W(z)$ szűrőt közös $x(n)$ jellel gerjesztjük, majd képezzük a kimeneteik közötti $e(n) = y(n) - \hat{y}(n)$ hibajelet[1].

[1] A jelek argumentumában n a diszkrét időt jelenti.

A továbbiakban feltételezzük, hogy a $W(z)$ szűrőt véges impulzusválaszú (FIR) formában keressük[2]. Ekkor az illesztendő szűrő átviteli függvénye, valamint a kimenete az alábbi alakban írható:

$$W(z) = w_0 + w_1 z^{-1} + \cdots + w_N z^{-N}$$
$$\hat{y}(n) = w_0 x(n) + w_1 x(n-1) + \cdots + w_N x(n-N) = \mathbf{x}^\mathsf{T}(n)\mathbf{w},$$

ahol $\mathbf{w} = [w_0 \quad w_1 \quad \dots \quad w_N]^\mathsf{T}$ és $\mathbf{x}(n) = [x(n) \quad x(n-1) \quad \dots \quad x(n-N)]^\mathsf{T}$. Az optimalizálás célja a várható négyzetes hiba minimalizálása:

$$\begin{aligned} C = \mathbb{E}\left\{e^2(n)\right\} &= \mathbb{E}\left\{(y(n) - \hat{y}(n))^2\right\} = \mathbb{E}\left\{\left(y(n) - \mathbf{x}^\mathsf{T}(n)\mathbf{w}\right)^2\right\} = \\ &= \mathbb{E}\left\{y^2(n)\right\} - 2\mathbf{w}^\mathsf{T}\,\mathbb{E}\left\{\mathbf{x}(n)y(n)\right\} + \mathbf{w}^\mathsf{T}\,\mathbb{E}\left\{\mathbf{x}(n)\mathbf{x}^\mathsf{T}(n)\right\}\mathbf{w} = \\ &= \mathbb{E}\left\{y^2(n)\right\} - 2\mathbf{w}^\mathsf{T}\mathbf{p} + \mathbf{w}^\mathsf{T}\mathbf{R}\mathbf{w} \end{aligned}$$

Itt az utolsó lépésben bevezettük a $\mathbf{p} = \mathbb{E}\left\{\mathbf{x}(n)y(n)\right\}$ keresztkorrelációvektort és az $\mathbf{R} = \mathbb{E}\left\{\mathbf{x}(n)\mathbf{x}^\mathsf{T}(n)\right\}$ autokorrelációs mátrixot. A költségfüggvény deriváltját nullával egyenlővé téve zárt alakban is megkaphatjuk az optimális $\mathbf{w}$ paramétervektort:

$$\frac{dC}{d\mathbf{w}} = -2\mathbf{p} + 2\mathbf{R}\mathbf{w} = 0 \qquad \Longrightarrow \qquad \boxed{\mathbf{w}_{opt} = \mathbf{R}^{-1}\mathbf{p}}$$

A zárt alakú megoldást megadó ún. Wiener–Hopf-egyenlet egyszerűsége ellenére számos gyakorlati alkalmazásban csak korlátozottan használható. Ha az $\mathbf{R}$ és $\mathbf{p}$ mennyiségekről nem rendelkezünk a priori ismerettel, akkor becslésükhöz szükség van a jel regisztrátumának offline feldolgozására, amely gátat szab a valós idejű működésnek. Ezért sok esetben célravezetőbb egy iteratív eljárás alkalmazása – ilyet kapunk, ha a hibafelület minimumát oly módon keressük, hogy minden ütemben a gradiens ellentettjével arányos nagyságú lépést teszünk (*gradient descent*). Az LMS algoritmus esetében a költségfüggvény pillanatértékéből indulunk ki:

$$\begin{aligned} C(n) = e(n)^2 = (y(n) - \hat{y}(n))^2 &= \left(y(n) - \mathbf{x}^\mathsf{T}(n)\mathbf{w}(n)\right)^2 = \\ &= y^2(n) - 2\mathbf{w}^\mathsf{T}(n)\mathbf{x}(n)y(n) + \mathbf{w}^\mathsf{T}(n)\mathbf{x}(n)\mathbf{x}^\mathsf{T}(n)\mathbf{w}(n) \end{aligned}$$

A gradienst pedig a költségfüggvény pillanatnyi deriváltjával közelítjük:

$$\begin{aligned} \frac{dC(n)}{d\mathbf{w}(n)} &= -2\mathbf{x}(n)y(n) + 2\mathbf{x}(n)\mathbf{x}^\mathsf{T}(n)\mathbf{w}(n) = -2\mathbf{x}(n)\left[y(n) - \mathbf{x}^\mathsf{T}(n)\mathbf{w}(n)\right] = \\ &= -2\mathbf{x}(n)e(n) \end{aligned}$$

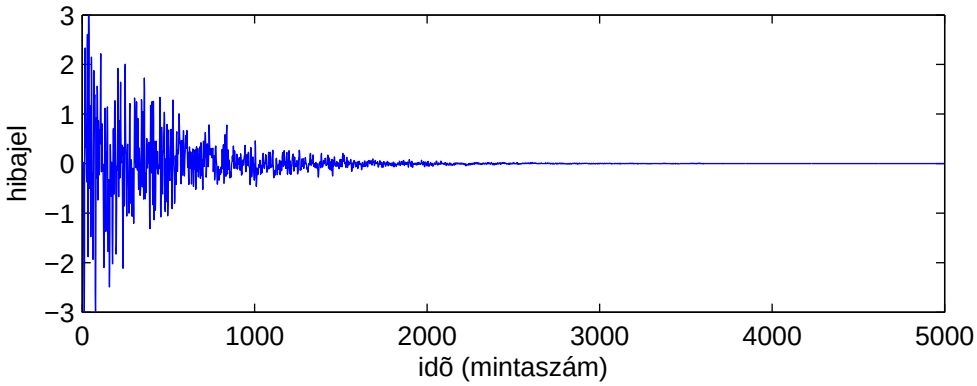
[2]Néhány ügyes trükk (pl. Equation Error Formulation) segítségével az LMS algoritmus végtelen impulzusválaszú (IIR) szűrő illesztésére is alkalmassá tehető [10].

Ezzel eljutottunk az LMS algoritmus számítási szabályához:

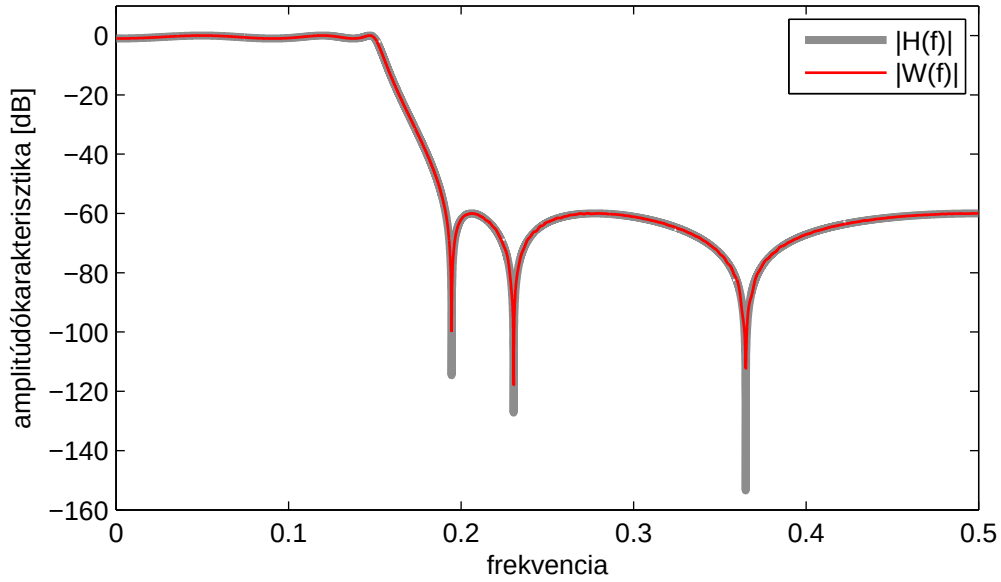
$$\boxed{\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{x}(n)}$$

Az egyenlet szemléletes: az algoritmus minden ütemben „tanul” a hibából, és ennek megfelelően módosít az adaptív FIR szűrő együtthatókészletén. Hogy mennyit, az a μ *bátorsági tényezőtől*[3] függ, ez határozza meg az algoritmus stabilitását, konvergenciasebességét, valamint a paramétervektor maradó hibáját.

A 2.2. és 2.3. ábrákon az LMS algoritmus működésének MATLAB szimulációja látható (hatodfokú elliptikus IIR szűrő identifikációja 300 együtthatós FIR szűrőként, normális eloszlású fehér zaj gerjesztéssel).



2.2. ábra. *A hibajel beállása*

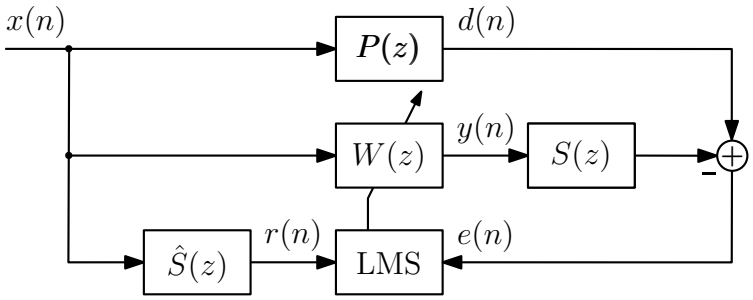


2.3. ábra. *Az eredeti (szürke) és az identifikált (piros) átvitel*

[3]Implementációs megfontolásból a μ bátorsági tényezőbe gyakran beleértjük az egyenletben szereplő konstans kettes szorzótényezőt is, így minden ütemben megspórolunk egy szorzást.

2.2. Aktív zajcsökkentés FxLMS algoritmussal

Az általános zavarjel[4] kioltására alkalmas aktív zajcsökkentő alkalmazások egyik tipikus algoritmusa az *FxLMS (Filtered-x LMS)*, amelynek blokkvázlata a 2.4. ábrán látható.



2.4. ábra. *Az FxLMS algoritmus vázlata*

A modell elemei a következők:

- Az $x(n)$ jel a *referenciajel*. Ez a távoli zajforrás által kibocsátott jel, illetve amit abból érzékelünk (esetleg egyszerű esetekben modellezünk).
- Az $y(n)$ jel a *beavatkozó jel*. Ezt adjuk ki a hangszórón, ezzel szeretnénk megvalósítani a destruktív interferenciát.
- Az $e(n)$ jel a *hibajel*. Ez maga a kioltandó zaj, ezt érzékeljük a mikrofonnal, ennek nullába szabályozása a célunk[5].
- A $d(n)$ jel az a *zavarás*, amely a zajforrás jeléből a kioltási helyre eljut.
- A $P(z)$ átvitel az *elsődleges akusztikus út (primary path)* modellje. Ezen keresztül terjed a zajforrás jele a kioltás helyéig.
- Az $S(z)$ átvitel a *másodlagos akusztikus út (secondary path)* modellje. Ezen keresztül terjed a megszólaltatott beavatkozó jel a kioltás helyéig (beleértve az ADC-k, DAC-k, hangszórók és mikrofonok átvitelét is.).
- Az $\hat{S}(z)$ átvitel az algoritmus része, ideális esetben megegyezik $S(z)$-vel. Mivel azonban $S(z)$ általában nem ismert analitikusan, $\hat{S}(z)$ meghatározása a gyakorlatban a másodlagos út identifikációjával – tipikusan a 2.1. pontban ismertetett LMS algoritmussal – történik.

[4]Speciális – például periodikus – zavarjel esetére léteznek más megközelítések is [5, 11, 12].

[5]Az FxLMS algoritmus blokkvázlatát bemutató 2.4. ábrán a témában szokásos konvencióhoz hűen a hibajel különbségképzéssel van számítva. Természetesen a valóságban a kioltási helyre eljutó akusztikus jelek összeadódnak. Ez implementációs szempontból mindössze annyit jelent, hogy a kiszámított beavatkozó jelet negatív előjellel kell kiadni a hangszórón.

- A $W(z)$ átvitel maga az adaptív szűrő (általában FIR), amely ideális esetben a $\frac{P(z)}{S(z)}$ átvitelhez konvergál, ekkor valósul meg ugyanis a zavarás kioltása:

$$E(z) = X(z)P(z) - X(z)W(z)S(z) = 0 \quad \Longrightarrow \quad W(z) = \frac{P(z)}{S(z)},$$

  ahol $E(z)$ és $X(z)$ rendre az $e(n)$ és $x(n)$ jelek z-transzformáltját jelöli.

- Az LMS algoritmus adaptációs szabálya:

$$\boxed{\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{r}(n)}$$

Az FxLMS algoritmus működése az alábbi lépésekben foglalható össze:

Inicializálás:

N_s, N_w, μ_i, μ_z beállítása a megválasztott értékre

$\hat{\mathbf{s}} \quad \leftarrow \mathbf{0}_{N_s \times 1}$

$\mathbf{w} \quad \leftarrow \mathbf{0}_{N_w \times 1}$

$\mathbf{b}_y \quad \leftarrow \mathbf{0}_{N_s \times 1}$

$\mathbf{b}_{xs} \quad \leftarrow \mathbf{0}_{N_s \times 1}$

$\mathbf{b}_{xw} \quad \leftarrow \mathbf{0}_{N_w \times 1}$

$\mathbf{b}_r \quad \leftarrow \mathbf{0}_{N_w \times 1}$

Identifikáció (feltételezzük, hogy ekkor nincs zavarás, azaz $d = 0$):

Egy adott feltétel teljesüléséig ismételjük (pl. előre beállított idő letelt, e kellően kicsire csökkent, a felhasználó megnyomta a gombot stb.):

Minden új e minta beérkezésekor ismételjük:

$y \quad \leftarrow \text{randNorm}$

$\mathbf{b}_y \quad \leftarrow [y \quad b_{y,1} \quad b_{y,2} \quad \dots \quad b_{y,N_s-1}]^\mathsf{T}$

$\hat{\mathbf{s}} \quad \leftarrow \hat{\mathbf{s}} + 2\mu_i \left(e - \mathbf{b}_y^\mathsf{T}\hat{\mathbf{s}}\right) \mathbf{b}_y$

Zajcsökkentés:

Minden új e, x minta beérkezésekor ismételjük:

$\mathbf{b}_{xw} \quad \leftarrow [x \quad b_{xw,1} \quad b_{xw,2} \quad \dots \quad b_{xw,N_w-1}]^\mathsf{T}$

$y \quad \leftarrow \mathbf{b}_{xw}^\mathsf{T}\mathbf{w}$

$\mathbf{b}_{xs} \quad \leftarrow [x \quad b_{xs,1} \quad b_{xs,2} \quad \dots \quad b_{xs,N_s-1}]^\mathsf{T}$

$r \quad \leftarrow \mathbf{b}_{xs}^\mathsf{T}\hat{\mathbf{s}}$

$\mathbf{b}_r \quad \leftarrow [r \quad b_{r,1} \quad b_{r,2} \quad \dots \quad b_{r,N_w-1}]^\mathsf{T}$

$\mathbf{w} \quad \leftarrow \mathbf{w} + 2\mu_z e\mathbf{b}_r$

Az algoritmusban szereplő véges impulzusválaszú szűrők működtetése matematikailag a buffervektorok jobbra léptetését, majd egy-egy skaláris szorzat képzését jelenti, azonban a gyakorlati megvalósítás során több lehetőségünk is van a szűrések implementálására. Egyes platformokon (pl. MATLAB, FPGA) akár ez a triviális megközelítés is kielégítő lehet, azonban az időtartománybeli konvolúció végrehajtására általában hatékonyabb megoldást jelent a *cirkuláris buffer* használata. Ekkor a buffervektorokat nem lineárisan töltjük fel, hanem mindig a vektor legrégebbi elemét cseréljük le, és nyilvántartjuk, hogy éppen melyik elem jelenti a vektor „elejét”. Ezáltal szükségtelenné válik a vektorelemek ciklusban történő jobbra léptetése, ráadásul számos végrehajtó egység hardveresen is támogatja a cirkuláris bufferek kialakítását.

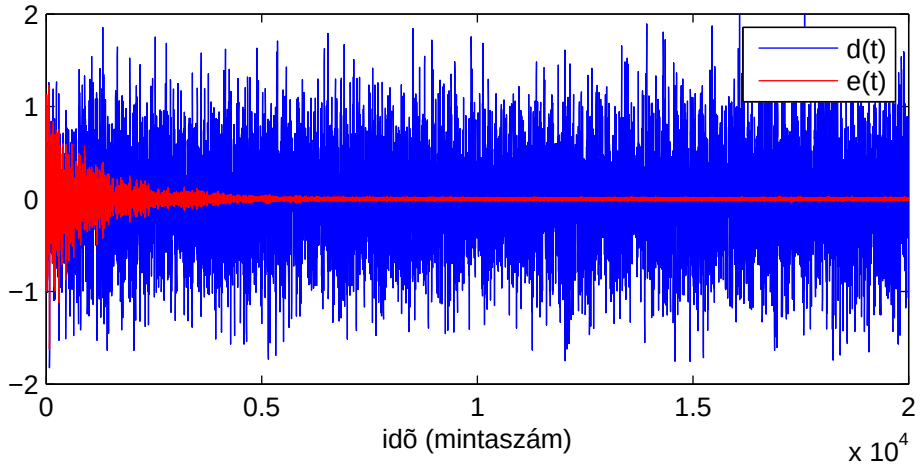
A FIR szűrések elvégzésének egy másik lehetséges módja a frekvenciatartománybeli számítás, amely a Fourier-transzformáció konvolúciótételét használja ki:

$$\mathbf{y} = \mathrm{IDFT}\left\{\mathrm{DFT}\left\{\mathbf{w}\right\} \circ \mathrm{DFT}\left\{\mathbf{x}\right\}\right\},$$

ahol a $\circ$ operátor a vektorok elemenkénti szorzását jelöli. Jóllehet a frekvenciatartománybeli módszer $\mathcal{O}\left(N \log_2 N\right)$ számításigénye nagy együtthatószámú szűrők esetén lényegesen kedvezőbb az időtartománybeli konvolúció $\mathcal{O}\left(N^2\right)$ számításigényénél, a DFT alapú szűrés hátránya, hogy blokkosan működik: csak minden N-edik ütemben keletkezik kimeneti minta (természetesen akkor N darab). Az adott alkalmazástól függ, hogy az így bevezetett késleltetés tolerálható-e.

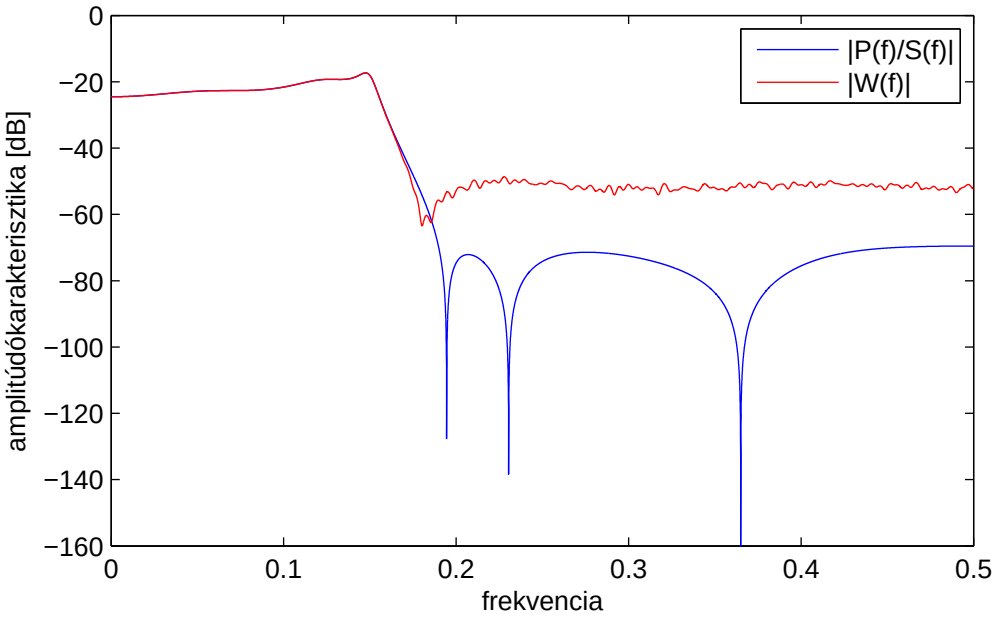
A későbbiekben bemutatott aktív zajcsökkentő rendszer implementációjában minden digitális szűrés időtartománybeli, cirkuláris buffer alapú számítással került megvalósításra, azonban a [6, 7] irodalomban ismertetett megoldás például DFT alapú számítást használ.

A 2.5–2.6. ábrákon az FxLMS algoritmus MATLAB szimulációja látható fehér zaj referenciajel esetén.



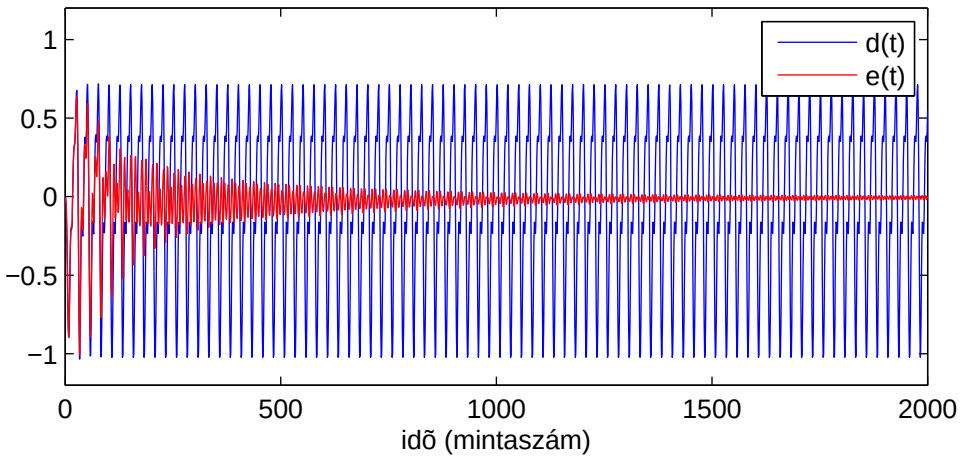
2.5. ábra. *Szélessávú zavarjel elnyomása*

Jól látható, hogy beállás után a $W(z)$ adaptív szűrő átvitele a várakozásoknak megfelelően nagyon jól közelíti a $\frac{P(z)}{S(z)}$ átvitelt.



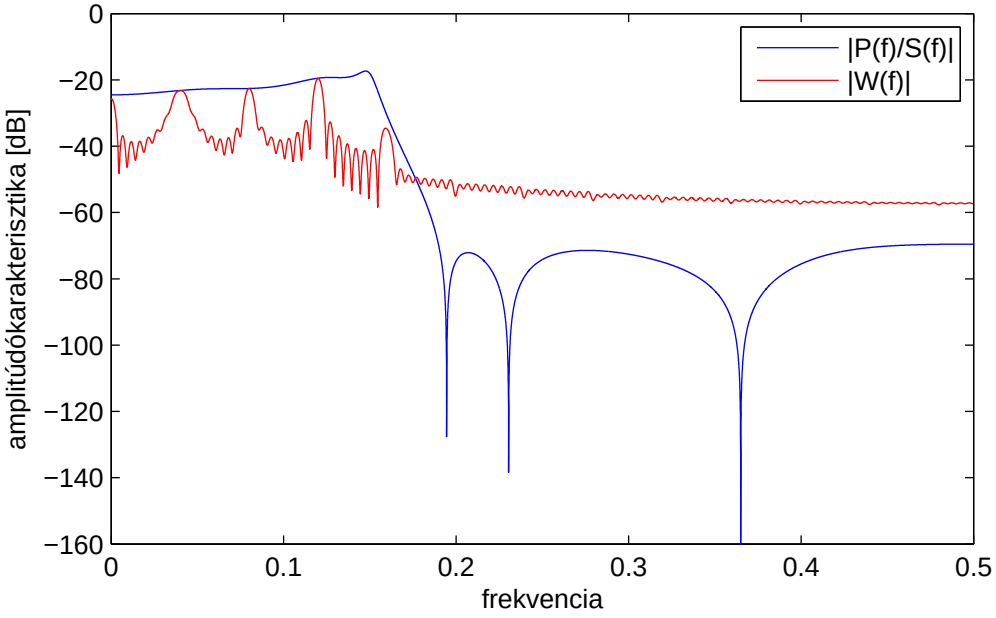
2.6. ábra. *Az adaptív szűrő átvitele szélessávú zavarjel esetén*

Érdekes a periodikus zavarás esete is. A 2.7–2.8. ábrákon egy olyan szimuláció eredménye látható, amelyben a referenciajel háromszögjel volt.



2.7. ábra. *Periodikus zavarjel elnyomása*

Ebben az esetben az adaptív szűrő csak a releváns pontokban „tanulta meg” a $\frac{P(z)}{S(z)}$ átvitelt, vagyis azokon a frekvenciákon, amelyeken a zavarásnak volt komponense.



2.8. ábra. *Az adaptív szűrő átvitele periodikus zavarjel esetén*

2.3. Többcsatornás aktív zajcsökkentés

Az FxLMS algoritmus abban az esetben is felhasználható az aktív zajcsökkentés megvalósítására, amikor egynél több pontban kívánjuk kioltani a zavarjelet, egynél több beavatkozó jel felhasználásával[6]. Természetesen ekkor a probléma dimenziója megnő, a többcsatornás zajcsökkentési feladat általános esetben nem kezelhető több független egycsatornás problémaként, hiszen minden beavatkozó jel eljut minden kioltási helyre (minden mikrofon érzékeli minden hangszóró jelét).

Tegyük fel, hogy M darab hibamikrofon jelének teljesítményét minimalizáljuk, H darab hangszóró által kiadott ellenzajok segítségével. Ekkor a 2.2. pontban bemutatott FxLMS algoritmus az alábbiakban módosul ($h = 1 \dots H$, $m = 1 \dots M$):

- $P_m(z)$ a zajforrás és az m-edik mikrofon közötti elsődleges akusztikus út modellje
- $S_{h,m}(z)$ a h-adik hangszóró és az m-edik mikrofon közötti másodlagos akusztikus út modellje
- $\hat{S}_{h,m}(z)$ az $S_{h,m}(z)$ átvitel LMS algoritmussal identifikált becslője (az identifikáció a hangszórók egyenkénti gerjesztésével történik)

[6]Egyes szerzők a többcsatornás FxLMS algoritmusra MLMS (Multiple Error LMS) néven hivatkoznak [12, 13]. Ennek a rövidítésnek azonban több különböző feloldása is megtalálható az irodalomban (pl. a súlyvektor korábbi módosításait is figyelembe vevő Momentum LMS [14]), ezért a továbbiakban az egyértelműség érdekében az MLMS rövidítést nem használom.

- $r_{h,m}(n)$ az $\hat{S}_{h,m}(z)$ átvitellel szűrt referenciajel. Az eddigiekben alkalmazott vektoros jelöléssel[7]:

$$r_{h,m}(n) = \mathbf{x}^{\mathsf{T}}(n)\hat{\mathbf{s}}_{h,m}$$

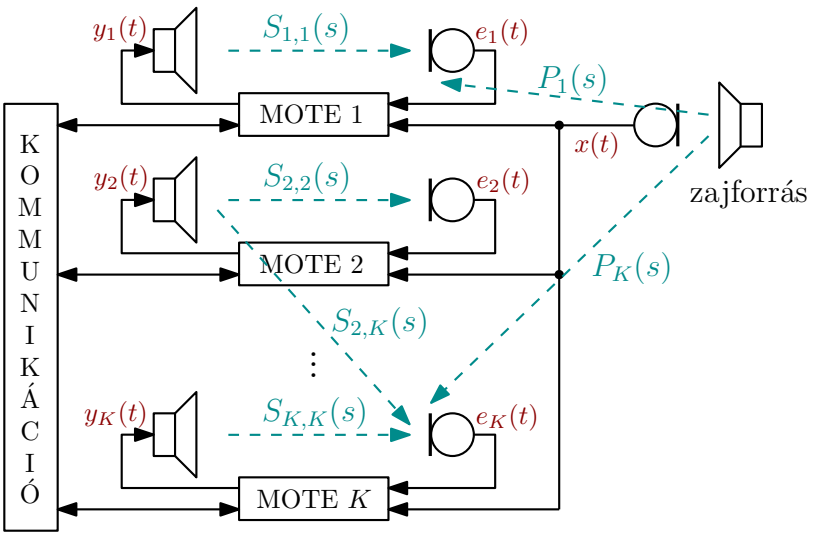
- $\mathbf{w}_h(n)$ a h-adik hangszóróhoz tartozó adaptív szűrő
- $y_h(n)$ a h-adik hangszóró által kiadott beavatkozó jel:

$$y_h(n) = \mathbf{x}^{\mathsf{T}}(n)\mathbf{w}_h(n)$$

- $e_m(n)$ az m-edik mikrofon által érzékelt hibajel
- A szűrők adaptációs szabálya:

$$\boxed{\mathbf{w}_h(n+1) = \mathbf{w}_h(n) + 2\mu \sum_{m=1}^{M} e_m(n)\mathbf{r}_{h,m}(n)}$$

A jelöléseket a 2.9. ábra illusztrálja arra a speciális esetre, amikor azonos számú mikrofon és hangszóró van a rendszerben (K darab mote, mindegyik egy-egy hangszóróval és mikrofonnal rendelkezik). Az ábrán az átláthatóság érdekében csak néhány akusztikus út van feltüntetve.



2.9. ábra. *A jelölések magyarázata*

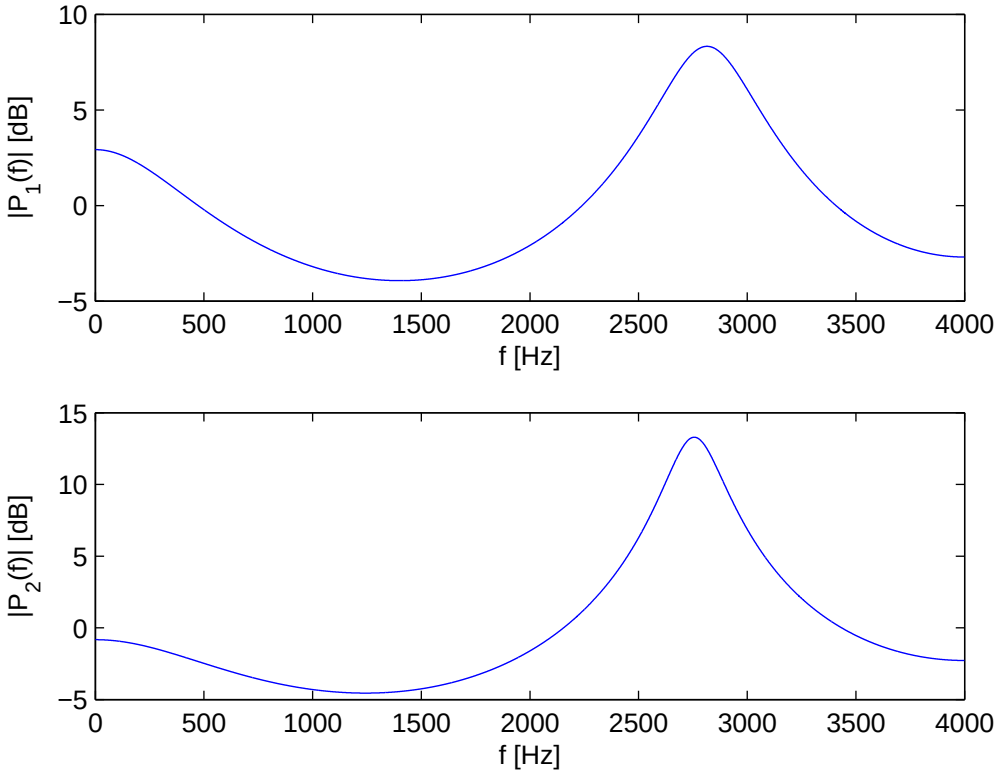
Az időfüggvények a folytonos időt jelentő t argumentummal szerepelnek az ábrán, az átviteli függvények pedig a folytonos idejű rendszereknek megfelelő s argumentumot kapták, hangsúlyozandó, hogy a jelzett pontokban a modellnek megfelelő fizikai

[7] Itt egyetlen közös referenciajelet feltételeztünk. Az általánosabb esethez lásd [3].

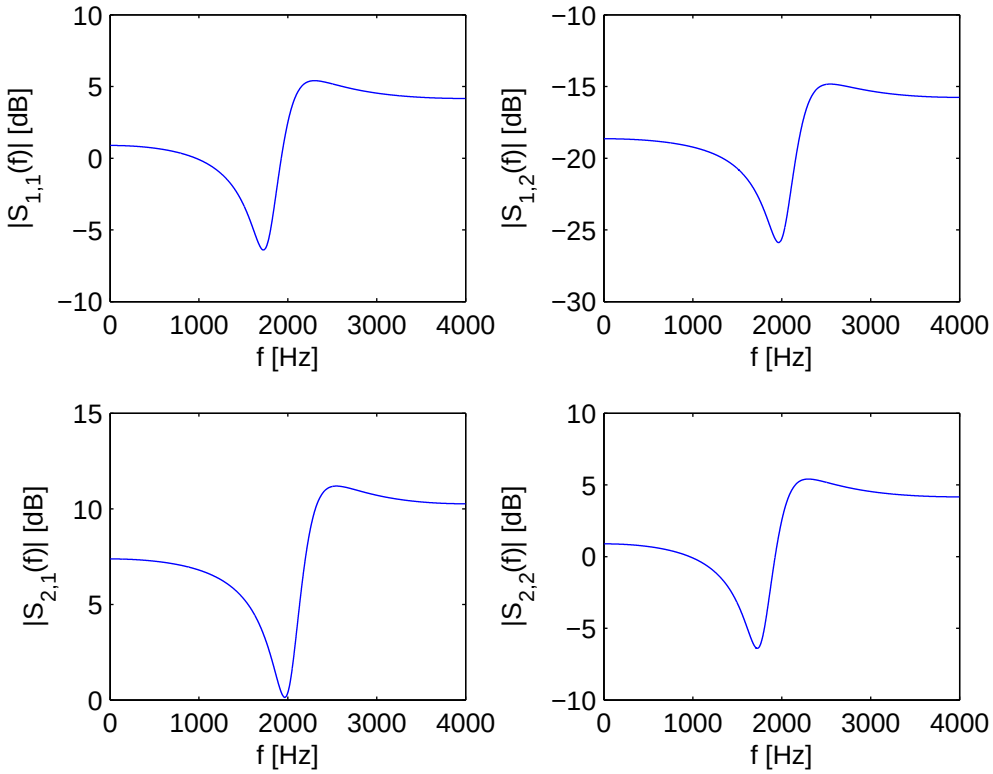
rendszerben analóg jelek vannak. Fontos továbbá, hogy az ábra analóg jelei és átvitelei csak részben felelnek meg az FxLMS algoritmus hasonló nevű digitális jeleinek, illetve rendszereinek, hiszen amíg $S_{h,m}(s)$ egy tisztán akusztikus rendszer átviteli függvénye, addig $S_{h,m}(z)$ tartalmazza a digitális-analóg átalakító, a hangszóró, a mikrofon, valamint az analóg-digitális átalakító hatását is.

A 2.10-2.12. ábrákon az algoritmus többé-kevésbé valóságos körülményeket modellezni igyekvő szimulációja látható két hangszóró és két mikrofon esetére. A zavarást a kioltási helyekre becsatoló elsődleges utak egymáshoz hasonló jellegű, harmadfokú IIR rendszerek, a hangszórók és a mikrofonok közötti másodlagos utak másodfokú IIR rendszerek. A mintavételi frekvencia 8 kHz.

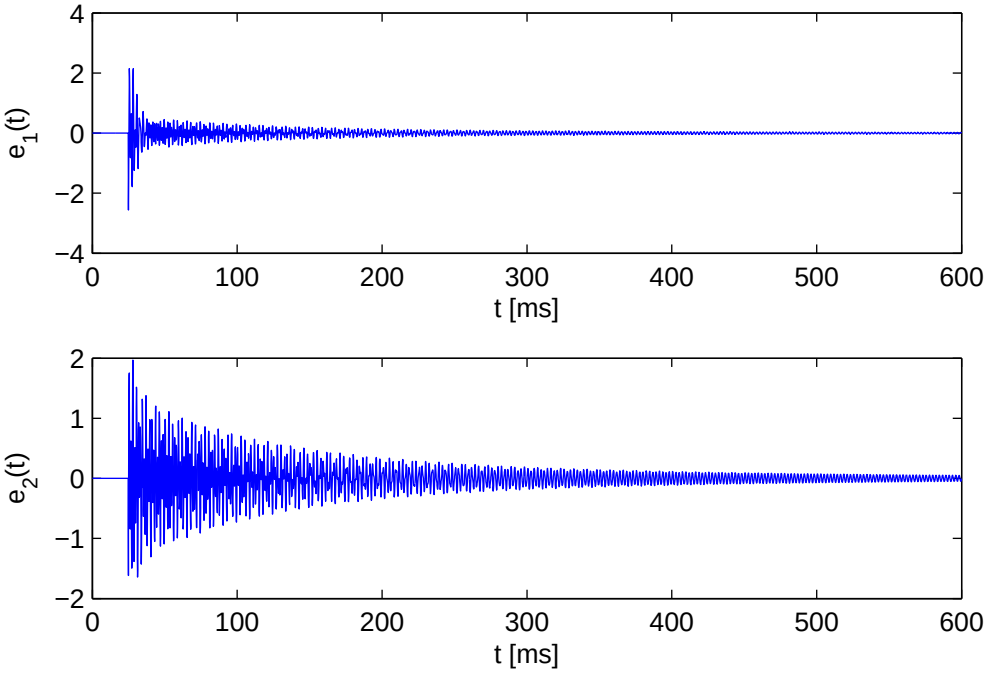
Az $S_{1,2}(z)$ és $S_{2,1}(z)$ átvitelek számottevően eltérő csillapításúak, és ez az aszimmetria a beállási időkben is megnyilvánul. Az átvitelek szimmetriáját helyreállítva a beállási idők is közel azonosak. Szélsőséges esetben, amikor a „keresztutak" végtelen nagy csillapításúak ($S_{1,2}(z) = S_{2,1}(z) = 0$), a probléma két független FxLMS algoritmusra csatolódik szét.



2.10. ábra. *Elsődleges utak*



2.11. ábra. *Másodlagos utak*



2.12. ábra. *Konvergencia*

2.4. Elosztott működés

Ebben a szakaszban feltételezzük, hogy az aktív zajcsökkentő rendszer architektúrája az 1.4. és a 2.9. ábrák szerinti, azaz a hibamikrofonok és a hangszórók száma megegyezik, és minden mote pontosan egy saját mikrofonnal és egy saját hangszóróval rendelkezik. Ez nem feltétlenül szükséges megkötés, azonban a tárgyalást leegyszerűsíti, és ez az architektúra valóságos rendszerekben is tipikusnak mondható. Ezekkel a megfontolásokkal számos mennyiséget mote-okhoz rendelhetünk:

- A k-adik mote-nak $\mathbf{w}_k(n)$ a *saját* adaptív szűrője
- A k-adik mote-nak $e_k(n)$ a *saját* hibajele
- A k-adik mote-nak $y_k(n)$ a *saját* beavatkozó jele
- A k-adik mote ismeri (tárolja) az $\hat{S}_{1,k}(z)$, $\hat{S}_{2,k}(z)$, ..., $\hat{S}_{K,k}(z)$ átviteleket.

Az utolsó pont némi magyarázatra szorul. Az $\hat{S}_{j,k}(z)$ átvitel megméréséhez a j-edik mote hangszóróját kell gerjeszteni, és a k-adik mote mikrofonját mintavételezni. Az $y_j(n)$ és $e_k(n)$ jelek tehát különböző mote-okban állnak elő, azonban az identifikációt végző LMS algoritmus e két jel időben összetartozó értékeinek ismeretét feltételezi. Ez a probléma többféleképpen megoldható:

- A tényleges zajcsökkentés elkezdése előtti identifikációs folyamat egyébként is speciális, „adminisztratív jellegű" fázisa a működésnek, ezért elképzelhető, hogy ekkor még megengedhető a külső beavatkozás, a rendszer megbontható. Ha ez így van, akkor az $\hat{S}_{j,k}(z)$ átvitel megmérésének erejéig csatlakoztathatjuk például a j-edik mote hangszóróját a k-adik mote-hoz, így utóbbi egyszerűen, önállóan el tudja végezni az identifikációt (természetesen az átcsatlakoztatást a hangszórók és a mikrofonok elmozdítása nélkül kell elvégezni).
- Ha ez a triviális megoldás nem megvalósítható, akkor az identifikáció időtartama alatt biztosítani kell a j-edik és a k-adik mote működése közötti szinkronizációt. Erre számos jól kidolgozott módszer létezik, azonban számolni kell a feldolgozó egységekkel és a kommunikációs réteg sávszélességével szemben támasztott követelmények növekedésével.
- Elképzelhetők köztes megoldások is (pl. mindkét mote számára ismert gerjesztés, szinkronizált indítás). Az adaptív szűrő valamilyen mértékben az identifikáció tökéletlenségét is kompenzálni tudja (már láttuk, hogy egycsatornás esetben az optimális szűrő $W(z) = P(z)/S(z)$).

Az identifikáció sikeres elvégzése után a k-adik mote-on futó, elosztott, többcsatornás FxLMS algoritmus működése az alábbi lépésekben összegezhető (az áttekinthetőség érdekében itt már nem a 2.2. szakaszban látott részletességgel, hanem a szövegben is használt matematikai jelölésekkel):

Normál futás:

Minden új e_k, x minta beérkezésekor ismételjük:

$y_k(n) \leftarrow \mathbf{x}^\mathsf{T}(n)\mathbf{w}_k$

Minden $j = 1 \dots K$-ra ismételjük:

$r_{j,k}(n) \leftarrow \mathbf{x}^\mathsf{T}(n)\hat{\mathbf{s}}_{j,k}$

$\Delta\mathbf{w}_{j,k} \leftarrow \Delta\mathbf{w}_{j,k} + 2\mu_z e_k(n)\mathbf{r}_{j,k}(n)$

Ha üzenetet kell küldeni a j-edik mote-nak:

$\mathrm{Send}_j(\Delta\mathbf{w}_{j,k})$

$\Delta\mathbf{w}_{j,k} \leftarrow \mathbf{0}$

Ha üzenet érkezett a j-edik mote-tól:

$\mathbf{w}_k \leftarrow \mathbf{w}_k + \mathrm{Recv}_j() + \Delta\mathbf{w}_{k,k}$

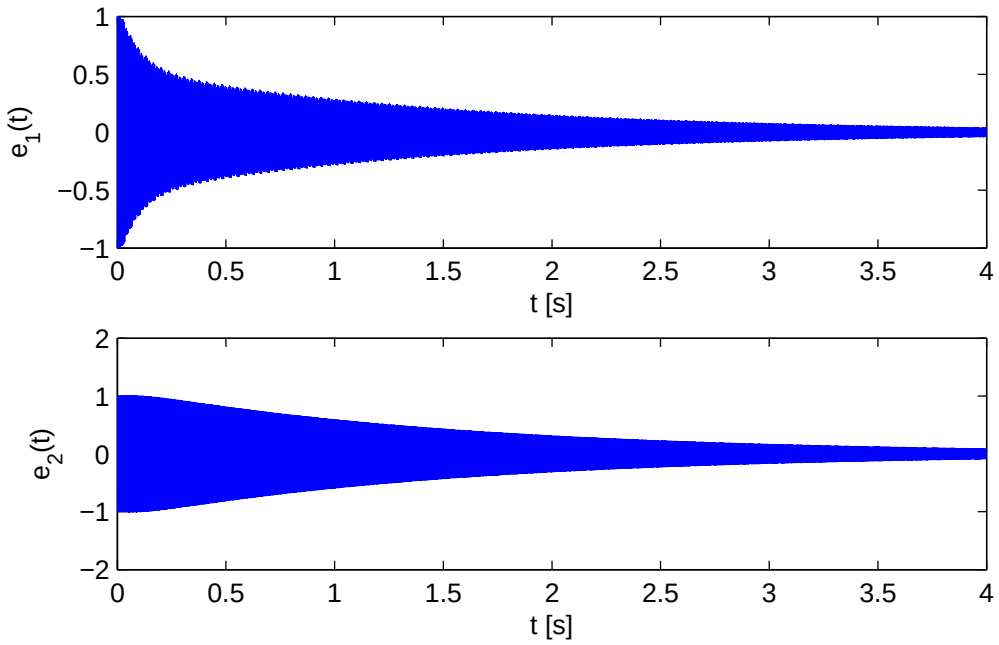
$\Delta\mathbf{w}_{k,k} \leftarrow \mathbf{0}$

Látható tehát, hogy minden mote akkumulálja a szűrőegyüttható-módosítások azon komponenseit, amelyeket ő maga a számára rendelkezésre álló adatok alapján ki tud számítani, majd ezeket a komponenseket a kommunikációs hálózaton eseményvezérelten elküldi a többi mote-nak. A küldést kezdeményezheti például belső időzítő/számláló, globális hálózati esemény (broadcast üzenet), vagy akár maga a címzett is. Amikor a k-adik mote megkap egy módosításkomponenst, akkor azzal frissíti a saját együtthatókészletét. A fenti pszeudokódban szintén ugyanekkor történik meg a mote saját maga számára előállított módosításvektorának ($\Delta\mathbf{w}_{k,k}$) felhasználása is, azonban elképzelhető az algoritmusnak olyan változata is, amely ezeket a komponenseket minden ütemben felhasználja.

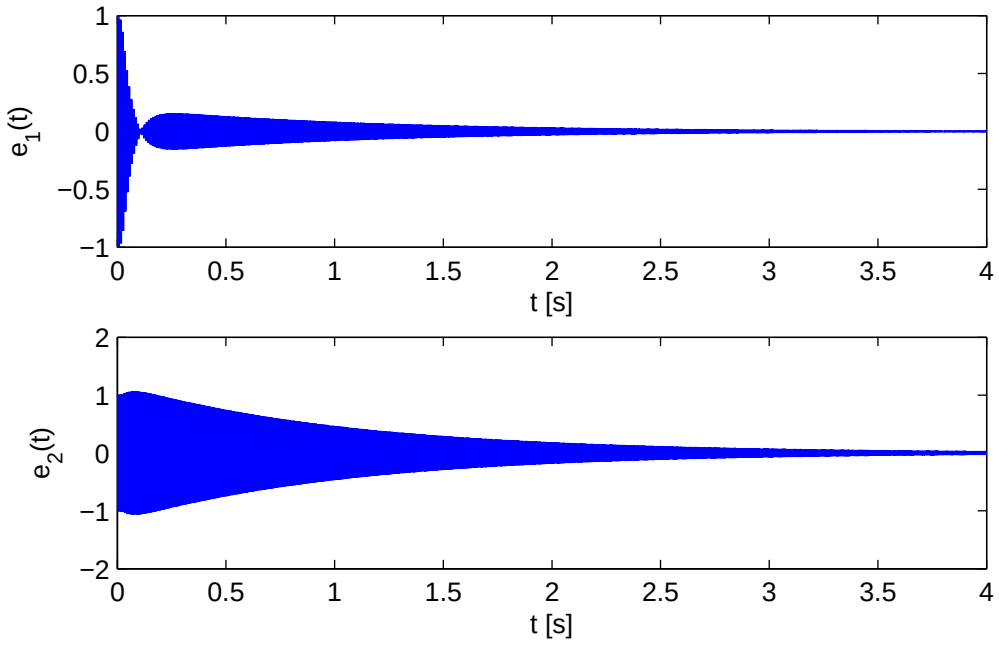
A többcsatornás, elosztott FxLMS algoritmust futtató mote-ok a fentiek alapján tehát egyenként $K + 1$ darab FIR szűrési műveletet, valamint K vektorösszeadást hajtanak végre egy normál ütem során. Ezzel szemben ha végiggondoljuk a 2.3. szakaszban leírtakat a $H = M = K$ speciális esetre, akkor azt vehetjük észre, hogy az algoritmus centralizált változata $K(K+1)$ FIR szűrés, valamint K^2 vektorösszeadás elvégzését igényelné. Az elosztott működés révén tehát az eredetileg négyzetesen skálázódó problémát lineárisan skálázódóvá tettük.

Az itt megfogalmazott algoritmus tekinthető a [6, 7] publikációkban tanulmányozott megoldás egy speciális esetének – azzal a már említett különbséggel, hogy a feldolgozás itt időtartományban történik. A blokkos jellegű végrehajtás elhagyásával lehetőség nyílik az üzenetküldési gyakoriságok tetszőleges beállítására is. Ezért ebben az elosztott FxLMS algoritmusban a centralizált változathoz képest az üzenetküldési gyakoriságok új paraméterekként jelennek meg, amelyeket esetleg fel lehet használni a beállási idők hangolására.

A 2.13-2.14. ábrákon látható szimulációs eredmények illusztrálják, hogy ezen paraméterek megválasztása milyen jelentős hatással lehet a konvergenciatulajdonságokra. A két esetben az elrendezés és a bátorsági tényező azonos, de a második esetben a 2-es mote háromszor gyakrabban küldi az együttható-módosításokat, mint az 1-es. Mivel adaptáció mindig vételkor történik, a 1-es mote gyakrabban tudja frissíteni az együtthatókészletét, ezért hibajelének csökkenése felgyorsul.

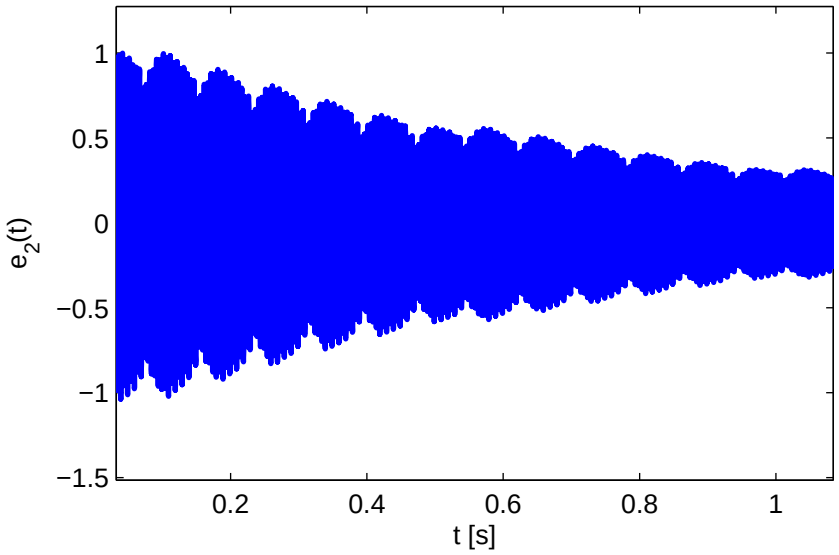


2.13. ábra. *Beállás azonos üzenetküldési gyakoriságok esetén*



2.14. ábra. *Beállás különböző üzenetküldési gyakoriságok esetén*

Érdekes megfigyelni, hogy bizonyos esetekben jól tetten érhetők az üzenetek megérkezéskor eszközölt együttható-frissítések hatásai a konvergáló hibajelekben. A 2.15. ábrán egy ilyen jel részlete látható.



2.15. ábra. *„Darabos" beállás*

3. fejezet

Eszközök

Ebben a fejezetben áttekintem az elosztott jelfeldolgozási feladatok elvégzésére alkalmas platformokat, valamint összefoglalom ezek előnyeit és hátrányait. Ezután ismertetem az általam vizsgált algoritmusok kipróbálásához választott konkrét hardvereket és szoftvereszközöket.

3.1. Végrehajtó egységek

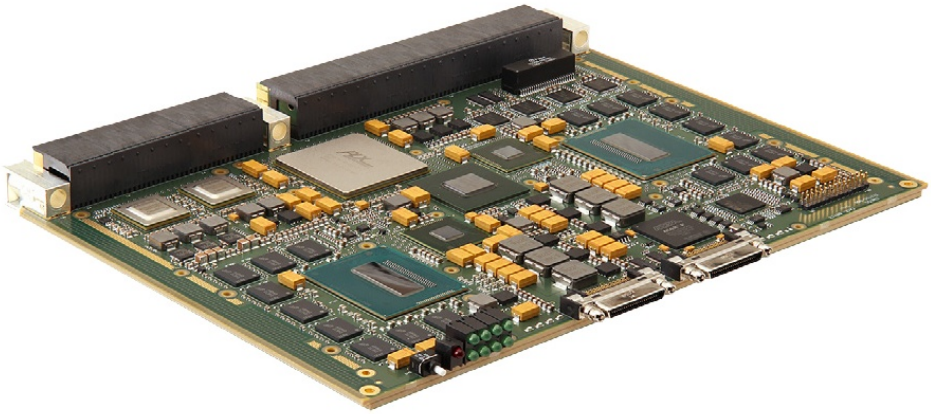
3.1.1. Hagyományos és ipari PC-k

Kétségtelen, hogy még viszonylag nagy számítási és memóriaigényű jelfeldolgozási algoritmusok is hatékonyan implementálhatók – például x86 architektúrájú processzort tartalmazó – hagyományos számítógépen. Egy Intel Core i7 CPU Wide Dynamic Execution Unit névre keresztelt szuperskalár végrehajtó magja órajelciklusonként hat mikroutasítás végrehajtására alkalmas, amelyekből három adatmozgató, három pedig aritmetikai jellegű lehet (utóbbiak akár SIMD utasítások is lehetnek). A többszintű utasítás- és adatchache segítségével akár 25,6 GBps memóriasávszélesség is elérhető. Ezen paraméterek egy – egyetlen processzormagot kihasználó – FFT algoritmus 20-30 GFLOPS sebességű elvégzését is lehetővé teszik, amely az elméleti hatékonyság közel 90%-a [15]. Az ilyen megoldások további előnye a „kulcsrakészség": az ipari kivitelű, számos interfésszel ellátott hardverek külső alkatrészek nélkül, azonnal beüzemelhetők.

Ezen kedvező tulajdonságoknak köszönhetően az x86 alapú jelfeldolgozó rendszerek még katonai és légiközlekedési, valamint ipari irányítástechnikai alkalmazásokban is helyet kapnak [15].

Egyértelmű hátrányuk azonban az áruk, amely az itt összefoglalt lehetőségek között a legmagasabb (néhány százezertől több millió forintig), valamint a magas fogyasztásuk. Ebből kifolyólag ilyen rendszereket általában csak akkor választunk, amikor az alkalmazás valóban megköveteli a robusztus kivitelt, illetve a rendelke-

zésre álló számítási teljesítményt. További hátrány, hogy a hagyományos operációs rendszerek általában nem, vagy csak korlátozottan támogatják a valós idejű működést; ha garantálni szeretnénk, hogy minden bemeneti adat időben feldolgozásra kerüljön, és minden ütemben generálódjon kimeneti adat, speciális operációs rendszerekhez vagy különleges szoftveres megoldásokhoz kell folyamodnunk.



3.1. ábra. *Curtiss-Wright CHAMP-AV9 DSP alaplap (Intel Core i7)*

A jelen dolgozat témáját képező elosztott jelfeldolgozó algoritmusok esetén azért sem lenne praktikus választás egy ilyen platform, mert az egyik legfőbb cél a költségcsökkentés; amennyiben minden mote szerepét egy-egy ipari PC töltené be, a jelenlegi megoldásoknál is sokkal drágább rendszert kapnánk. Ha pedig egyetlen PC végezné az összes szükséges számítást, akkor az algoritmusok elosztott jellegét legfeljebb szimulálni lehetne. Továbbá ilyen eszközök munkám során nem álltak rendelkezésemre.

3.1.2. Jelfeldolgozó processzorok (DSP-k)

Jelfeldolgozási feladatokra a legkézenfekvőbb megoldás lehet egy kifejezetten ilyen alkalmazások céljára dedikált processzor használata. A DSP-k utasításkészletét és architektúráját valós idejű működésre, illetve a digitális jelfeldolgozás területén leggyakrabban előforduló algoritmusok hatékony végrehajtására optimalizálták. Néhány ilyen különleges jellemző:

- **Moduláris címaritmetika.** Speciális címzési mód, amely többek között a konvolúció hatékony végrehajtásához szükséges cirkuláris bufferek kialakítását teszi lehetővé.
- **Bitreverse címzés.** Az FFT algoritmus legelterjedtebben alkalmazott implementációi (mint például a Cooley–Tukey-algoritmus) a bemeneti adatokat in-

dexeik fordított bináris alakjának megfelelő sorrendben választják ki. A DSP-k ezt általában hardveresen támogatni tudják.

- **Multiply-Accumulate (MAC) utasítás.** Egy konvolúció végrehajtása során minden lépésben összeszorzunk két számot, majd az eredményt hozzáadjuk egy akkumulátorregiszterhez. A legtöbb DSP ezt egyetlen utasításciklus alatt el tudja végezni (akár több adatra is).

- **Kiterjesztett pontosságú akkumulátor.** Egy konvolúció számításakor az egyes MAC műveletek részeredményeit egy akkumulátorregiszterben tároljuk, majd a végeredményt a számítás végeztével áttöltjük egy másik regiszterbe, vagy kiírjuk a memóriába. Általában a részeredményeket nagyobb bitszámmal tároljuk, mint a végeredményt, így ugyanis kisebb kerekítési hibát követünk el, mint ha az akkumulátor bitszáma megegyezne a végeredményével.

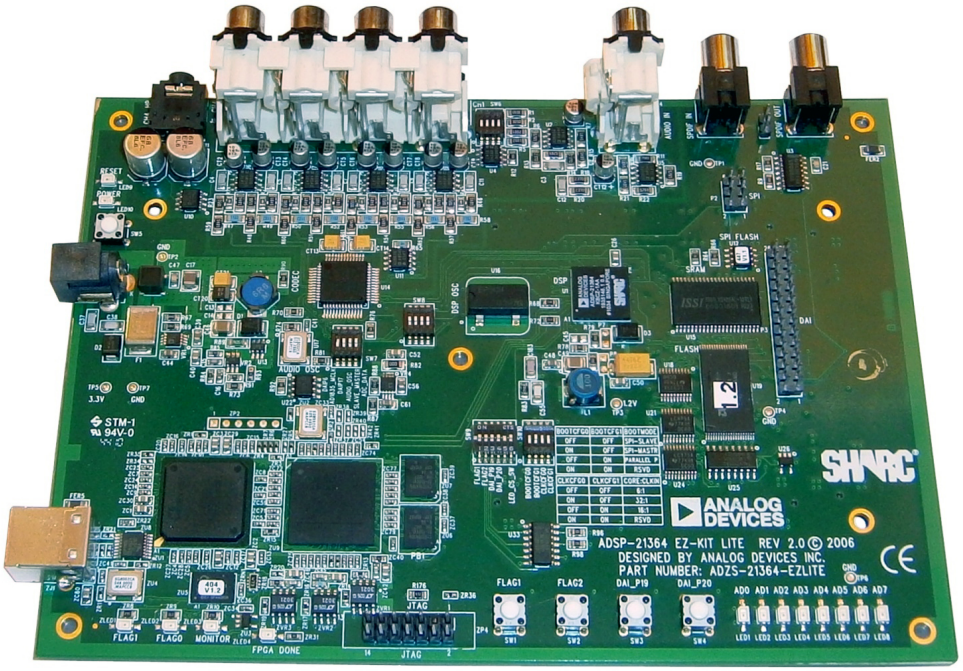
- **Párhuzamos buszrendszerek.** A MAC művelet önmagában nem feltétlenül lenne hatékony, ha az általa használt adatokhoz egyenként kellene hozzáférni. Emiatt a DSP-k gyakran rendelkeznek többszörözött buszrendszerrel, amely lehetővé teszi egy utasítás és akár több operandus egyetlen utasításciklus alatt történő beolvasását.

- **Hardveres ciklusszervezés.** Ciklusok kialakításának legtriviálisabb módja a ciklusváltozó minden lépésben történő növelése és a ciklusszámmal való komparálása. Ez azonban a pipeline alapú utasításvégrehajtás szempontjából hátrányos, mivel amíg nem dől el, hogy folytatni kell-e a ciklust, vagy pedig kilépni belőle, addig az sem tudható, hogy melyik utasítást kell következőként betölteni a pipeline-ba. A DSP-k legtöbbje ezért rendelkezik olyan speciális utasításokkal, amelyek lehetővé teszik egy programrészlet adott számú végrehajtását feltételes ugróutasítások alkalmazása nélkül.

- **Utasításszintű párhuzamosság.** A DSP-k egyes utasításai egymással párhuzamosan is végrehajthatók. Ennek feltétele, hogy az utasítások fizikailag külön egységeket használjanak, valamint hogy egyetlen hosszú utasításkódba egyszerre belekódolhatók legyenek (VLIW architektúra).

A DSP-k tehát számos szempontból ideális platformját alkothatják egy jelfeldolgozási feladat megoldásának. Azonban egy jelfeldolgozó processzor beüzemelése egy általános célú mikrokontrollernél jóval bonyolultabb hardveres környezetet igényel (külső memóriák illesztése, debugger áramkör stb.), ezért kutatási és prototípusfejlesztési célra általában a gyártók által kínált fejlesztőkártyák használata praktikusabb, ezek ára azonban viszonylag magas (a 3.2. ábrán látható fejlesztőkészlet ára

kb. 120 ezer forint). Továbbá az elosztott algoritmusok vizsgálatával kapcsolatban itt is felmerül az a probléma, miszerint egyetlen DSP használata esetén az elosztottságnak legfeljebb a szimulációjáról beszélhetnénk, több fejlesztőkártya pedig munkám keretét számottevően meghaladó költségeket jelentene.



3.2. ábra. *Analog Devices ADSP-21364 EZ-KIT Lite fejlesztőkártya*

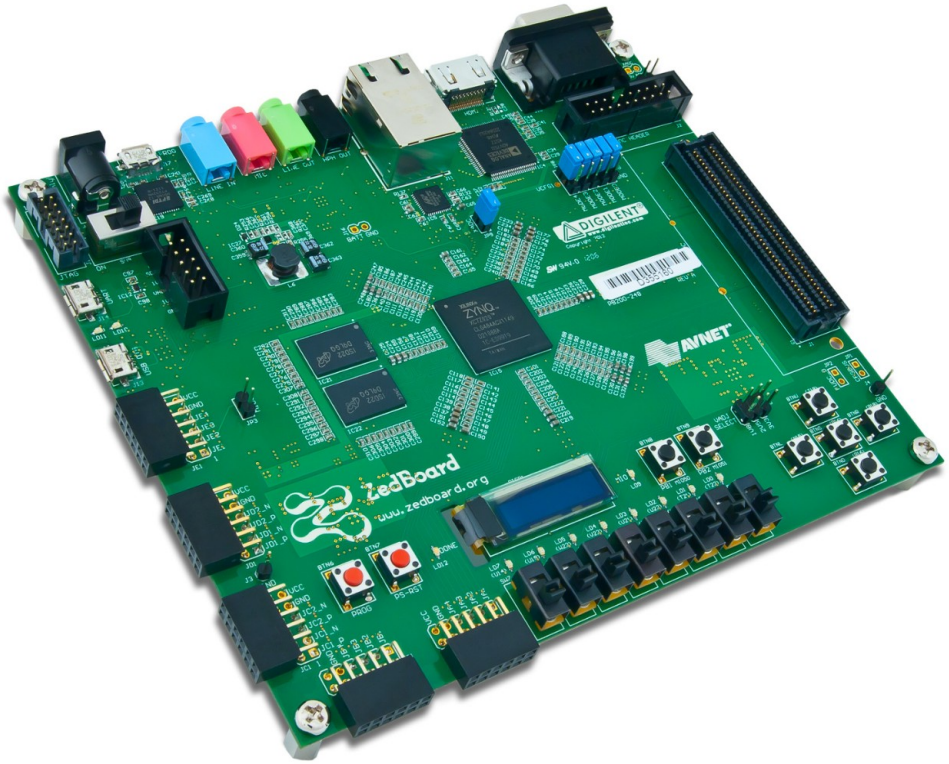
További hátrányként említhető a fejlesztéshez szükséges szoftverek viszonylagos bonyolultsága, a nyílt forráskódú – vagy legalábbis ingyenesen használható – megoldások hiánya, valamint a szegényes támogatottság.

3.1.3. FPGA áramkörök

Az FPGA áramkörök belsejében olyan digitális célhardver szintetizálható, amilyet csak a választott módszerrel (tipikusan hardverleíró nyelvekkel) előírunk – természetesen az adott áramkör erőforráskészletének erejéig. Emiatt számos alkalmazásban nagyon hatékonyan használhatók, különösen a nagyfokú párhuzamosságot megkövetelő feladatok esetében mutatkozik meg az előnyük.

A modern FPGA-k alkalmasak arra is, hogy az egyedi hardverek mellett egy vagy több processzormagot is szintetizáljunk bennük. Sőt, az utóbbi években ez a tervezési minta vált tipikussá, a legtöbb FPGA dizájn tartalmaz valamilyen procedurális végrehajtó egységet, mivel bizonyos feladatokat lényegesen egyszerűbb szoftveresen megvalósítani (inicializálás, kommunikációs protokollok kezelése, felhasználói interakció stb.). A gyártók – felismervén ezt a trendet – bevezették az olyan SoC áramköröket, amelyek a hagyományos programozható logikai rész mellett már eleve tartal-

maznak beépített (ún. *hard core*) processzormagokat is, tehát azokat nem az FPGA rész logikai erőforrásainak felhasználásával kell kialakítanunk. Ilyen például az Xilinx 2011-ben bevezetett áramkörcsaládja, a Zynq-7000. Ezek a chipek a programozható logikai rész mellett két ARM Cortex-A9 processzormagot is tartalmaznak. Egy elosztott jelfeldolgozó rendszer FPGA-s megvalósításában célszerűen egy processzormag végezhetné a kommunikáció kezelését, amely vétel esetén egy szabványos buszon keresztül átadná a vett adatokat a programozható logikai részben egyedi hardverként megvalósított jelfeldolgozó egységnek.



3.3. ábra. *Avnet ZedBoard (Xilinx Zynq-7000)*

Az FPGA-kra fokozottan igaz az – a DSP-knél is említett – szempont, mely szerint a működésükhöz minimálisan szükséges hardveres környezet viszonylag összetett: sokféle, nagy terhelhetőségű tápfeszültségszintre, a konfigurációt tároló memóriára, debugger interfészre van szükség, ráadásul a modern FPGA-k többsége csak ipari körülmények között beültethető (tipikusan BGA) tokban elérhető. Ezért kutatási jellegű projektek során saját áramkör építése helyett általában itt is kész fejlesztőeszközökhöz folyamodunk, ám ezek ára is viszonylag magas (a 3.3. ábrán látható ZedBoard ára kb. 100 ezer forint, bár ennél egyszerűbb FPGA fejlesztőeszközök 30-50 ezer forinttól is elérhetőek).

További hátrány, hogy a hardverleíró nyelvek ismerete, az ilyen chipek áramköri sajátosságai, az FPGA-ban szintetizálódó hardver és a processzormag(ok)on

futó szoftver együttes fejlesztésének kérdései *(hardware-software codesign)* speciális szakértelmet igényelnek, és általában a fejlesztési idő is hosszabb, mint egy tisztán szoftveres megoldás esetében.

3.1.4. Általános célú mikrokontrollerek

Általánosságban elmondható, hogy az itt összefoglalt lehetőségek közül az egyszerű mikrokontrollerek ára a legalacsonyabb, valamint ezekkel az eszközökkel érhető el a legkisebb fogyasztás is. A legtöbb architektúrához rendelkezésre állnak ingyenes fejlesztőeszközök és példaalkalmazások, továbbá a gyártók többsége kínál kedvező árú, egyszerűen beüzemelhető fejlesztőkártyákat termékeik kipróbálásához. A mikrokontrollerek széleskörű elterjedtségének további hasznos következménye, hogy a fejlesztés közben felmerülő problémáinkra internetes tudásbázisokban és fórumokban az esetek többségében gyorsan megoldást találhatunk.

Az általános célú mikrokontrollerek piacának napjainkban legjelentősebb – csaknem egyeduralkodó – szereplője az ARM architektúra[1]. RISC utasításkészletű processzormagjaik rendkívül széles választékban elérhetőek: általános beágyazott alkalmazások szempontjából legjelentősebb a Cortex-M család, azonban léteznek még többek között valós idejű működésre, alacsony teljesítményfelvételre, biztonságkritikus alkalmazásokra, valamint mobil eszközökben való felhasználásra optimalizált processzormag-családok is. Az ARM architektúrájú maggal rendelkező mikrovezérlők az egyes gyártók kínálatában igen széles skálán mozgó perifériakészlettel szerepelnek: elérhetőek alacsony árú, csak a legszükségesebb perifériákat (időzítők, számlálók, alapvető kommunikációs interfészek stb.) tartalmazó, egyszerű feladatokra szánt modellek, illetve speciális alkalmazásokban kihasználható, komplex egységekkel (MPEG codec, kamerainterfész, kriptográfiai célhardverek) felszerelt változatok is.

Egy elosztott jelfeldolgozó rendszer részeként működő mikrokontrollerben az alábbi tulajdonságok és perifériák megléte jelent előnyt:

- **DSP utasítások.** Számos általános célú mikrokontroller utasításkészlete tartalmaz néhány, hagyományosan a jelfeldolgozó processzorokra jellemző utasítást, valamint természetesen az azok hatékony végrehajtására képes belső egységeket. A leggyakoribb példák: MAC utasítás, hardveres ciklusszervezésre és moduláris címzésre szolgáló utasítások.

- **SIMD utasítások.** Egyes mikrokontrollerek tartalmaznak olyan utasításokat, amelyek segítségével ugyanazon művelet egyszerre több adaton is elvégezhető.

[1] Az ARM Holdings nem chipgyártó, náluk csak az architektúra fejlesztése zajlik, bevételük a gyártási licencek eladásából származik.

Például SIMD módban végrehajtható MAC utasításokkal nagyon hatékony szűrési és mátrixalgoritmusok szervezhetők.

- **I^{2}S interfész.** Hangfrekvenciás jelek feldolgozását célzó alkalmazásokban a D/A és A/D átalakítók illesztésének legelterjedtebb interfésze az I^{2}S, így az ilyen rendszerekben használt mikrovezérlőknek elengedhetetlen része a protokollt hardveresen implementáló periféria.

- **DMA egység.** Egy jelfeldolgozó rendszer a környezetével folyamatos információs kapcsolatban áll, szüntelen az adatáramlás a külvilág és a feldolgozó egység között. A DMA mentesíti a processzormagot az adatoknak a memória és a szenzorok/beavatkozó szervek közötti mozgatásának terhétől, értékes számítási teljesítményt spórolva ezzel a jelfeldolgozási feladatok javára.

- **Lebegőpontos aritmetikai egység (FPU).** A jelfeldolgozó algoritmusok működésének minőségét hátrányosan befolyásolják a fixpontos számábrázolás sajátosságaiból eredő kvantálási hibák, sőt szélsőséges esetekben akár stabilitási problémákhoz is vezethetnek. Lebegőpontos aritmetika használatával sokat javíthatunk a helyzeten, azonban ezt csak akkor tudjuk hatékonyan megtenni, ha a mikrovezérlő rendelkezik a lebegőpontos számítások hardveres végrehajtásához szükséges aritmetikai egységgel.

- **Ethernet támogatás.** Egy elosztott rendszer működésének kritikus eleme a rendszert alkotó számítógépek közötti kommunikáció, illetve azt azt lehetővé tévő összeköttetések hálózata. Az Ethernet erre kínál egy kézenfekvő megoldást, flexibilitásának és nagy sávszélességének köszönhetően.

3.2. Kommunikációs réteg

Elosztott rendszerek közös tulajdonsága a rendszer elemei közötti kommunikáció szükségessége, azonban az egyes konkrét alkalmazások nagyon különböző követelményeket támasztanak a kommunikációs hálózattal szemben. Egy elosztott rendszer elemei közötti összeköttetések megtervezésekor az alábbi szempontokat kell mérlegelnünk:

- **A kialakítani kívánt topológia.** Az elosztott rendszerbe kapcsolt számítógépek között olyan kapcsolatokat kell kialakítani, amelyek biztosítani tudják a konkrét alkalmazás által megkívánt mértékű redundanciát. Amennyiben a rendszer állandó rendelkezésre állásának biztosítása másodlagos szempont, akkor elegendő lehet egy egyszerű busz kialakítása is (pl. RS-485), ellenkező esetben azonban hibatűrőbb összeköttetésrendszer szükséges (pl. FlexRay). A

hálózat topológiájára hatással vannak továbbá a kommunikációs folyamat sajátosságai, valamint a rendszer egyes elemeinek különböző szerepei; elképzelhető például, hogy bizonyos kommunikációs útvonalakat, hálózati csomópontokat tehermentesítés céljából kell többszörözni.

- **Sávszélesség.** A hálózat sávszélességével szemben támasztott követelmény a különböző elosztott rendszerekben széles skálán változhat. Egyes alkalmazásokban mindössze néhány jelzés továbbítására, vagy az időnként előálló számítási eredmények megosztására van szükség, míg más esetekben a hálózat sávszélessége az egyik legfontosabb erőforrás (pl. peer-to-peer fájlmegosztás, elosztott adatbázisok).

- **Az adatátvitel sajátosságai.** Néhány alkalmazásban megengedhető az adatátvitel best effort jellege, más esetekben azonban feltétlenül biztosítani kell az elküldött adatok hiánytalan megérkezését, valamint sorrendhelyes vételét (pl. UDP vs. TCP). Ide sorolható szempont még az üzenetek késleltetése, prioritásos, unicast, broadcast, multicast üzenetek küldésének lehetősége stb.

- **A rendszer elemeinek távolsága.** A kommunikációs réteg kialakítását nagymértékben befolyásolja, hogy az elosztott rendszerben működő számítógépek térben egymástól milyen messze helyezkednek el; más követelmények érvényesek egy zárt téren belül működő (pl. „okosotthon” vezeték nélküli szenzorhálózata) és egy esetleg országhatárokon átívelő kiterjedésű rendszerre (pl. grid computing).

- **Bővíthetőség.** Elosztott rendszerekben gyakori követelmény a struktúra flexibilitása, új számítógépek bekapcsolásának és meglévők eltávolításának lehetősége, akár minimális beavatkozással, plug-and-play módon. Ezt lehetővé teszi a legtöbb vezeték nélküli hálózat (pl. Bluetooth, ZigBee)

- **Biztonság.** Elképzelhetők olyan elosztott alkalmazások, amelyekben olyan hálózat kialakítása szükséges, amely biztosítani képes a kommunikáció külső fél általi lehallgathatatlanságát, visszafejthetetlenségét (pl. Ethernet + TLS).

- **Bonyolultság, költség.** A hálózat kiépítéséhez és a fejlesztéshez szükséges eszközök beszerzésének, valamint a hálózat üzemeltetésének költsége egyes esetekben igen magasra rúghat (pl. GSM, FlexRay, optikai átvitel), míg egyszerűbb, kevésbé speciális hálózatok esetén jóval alacsonyabb lehet (pl. RS-485, Ethernet).

Egy digitális jelfeldolgozási – speciálisan: aktív zajcsökkentési – feladatot ellátó elosztott rendszer kialakításakor a redundáns, hibatűrő működés és a plug-and-play képesség nem érdektelen szempontok, de a jelen dolgozat témáját adó kutatási fázisban másodlagosak. Annál lényegesebb viszont a szükséges eszközök könnyű hozzáférhetősége, egyszerű kezelhetősége, valamint alacsony költsége, kiemelt fontosságú továbbá a hálózat sávszélessége.

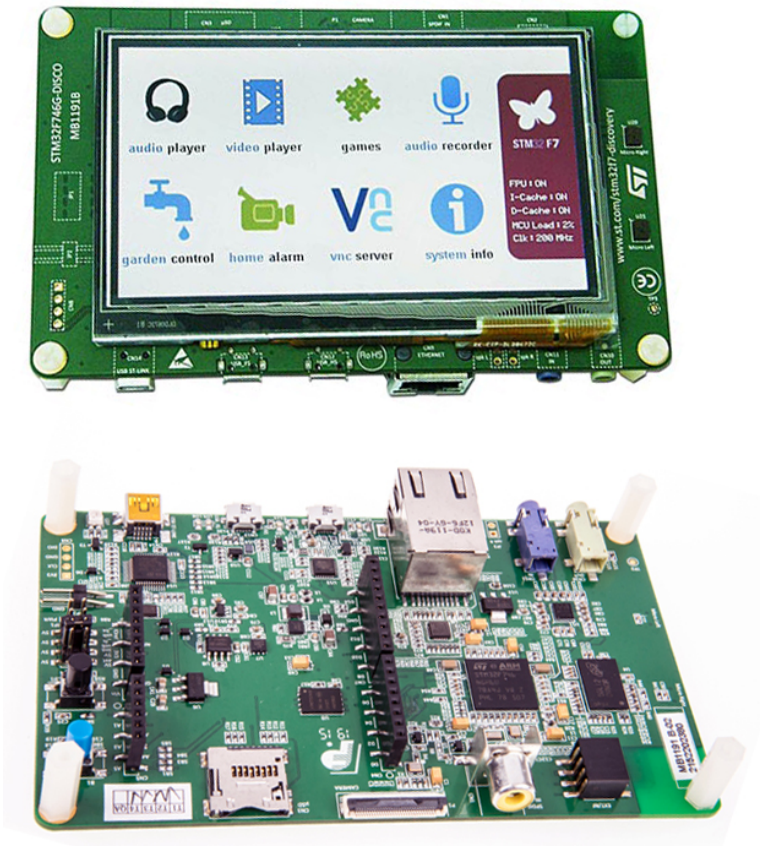
Tételezzük fel, hogy egy elosztott FxLMS algoritmust 8 kHz mintavételi frekvenciával futtató mote az algoritmus üzenetküldési fázisához érkezett. Az adaptív szűrők 300 együtthatósak, az együtthatók 32 bites, lebegőpontos számok. Ez azt jelenti, hogy egyszerre egy másik mote-nak 9600 bitnyi adatot kell elküldenie – a kommunikációs overheadet nem számítva. Ha azt szeretnénk, hogy egyetlen feldolgozási ütem alatt célba érjen az üzenet, akkor a küldésre 125 µs idő áll rendelkezésre, ez pedig 76,8 Mbit/s hasznos sávszélességet jelent. Ez a követelmény jelentősen leszűkíti a szóba jövő kommunikációs megoldások listáját. Noha a gyakorlatban nem szükséges elvárni az együttható-módosítások késleltetés nélküli megérkezését, illetve a zajcsökkentés általában ennél alacsonyabb mintavételi frekvencia és együtthatószám mellett is működőképes, a példa adatai mindenképpen reálisak.

Ezek alapján kijelenthető, hogy nem érdemes olyan kommunikációs hálózatot választani, amelynek átviteli sebessége legfeljebb néhány száz kbit/s, mert bár egy működőképes próbarendszer kialakítását valószínűleg lehetővé tenné, jelentősen korlátozná a jövőbeli továbbfejlesztési lehetőségeket. Ezzel szemben egy Ethernet alapú megoldás teljesíti a megfogalmazott követelmények mindegyikét: alacsony költségű, hétköznapi eszközökkel is kialakítható Gbit/s nagyságrendbe eső sávszélességű hálózat, illetve – bár az Ethernet és a hozzá kapcsolódó protokollok működésének részletei összetettek – számos modern mikrokontroller rendelkezik beépített Ethernet vezérlővel, amely nagymértékben leegyszerűsíti a hálózat kezelését. Elérhetők továbbá olyan middleware szoftverkomponensek, amelyek implementálják az Ethernethez kapcsolódó magasabb szintű protokollokat, így a kapcsolatok felépítésének és a kommunikáció folyamatának részletei szinte teljes egészében rejtve maradnak a programozó előtt.

3.3. A választott eszközök

3.3.1. STM32F746G Discovery kártya

Az előzőekben ismertetett szempontok mérlegelése után az elosztott FxLMS algoritmuson alapuló aktív zajcsökkentési feladat gyakorlati megvalósításának platformjául az STMicroelectronics cég *STM32F746G Discovery* névre keresztelt fejlesztőkártyáját választottam. A kártya fotója a 3.4. ábrán látható.



3.4. ábra. *Az STM32F746G Discovery kártya*

A fejlesztőkártyán egy ARM Cortex-M7 architektúrájú mikrokontroller kapott helyet, amely lebegőpontos aritmetikai egységgel, 1 MB belső flash memóriával, valamint 340 kB RAM-mal rendelkezik, és legfeljebb 216 MHz frekvenciájú órajellel képes működni. A Cortex-M család egyszerűbb képviselőihez képest egyik legjelentősebb újdonsága az utasítás- és adatcache megléte, amely jelentős mértékben felgyorsíthatja a programok végrehajtását[2]. A mikrokontroller utasításkészlete tartalmaz néhány, kifejezetten a jelfeldolgozási célú algoritmusok elvégzését támogató

[2]A cache bizonyos esetekben hátrányt is jelenthet, mivel a végrehajtási időt ugyan lerövidíti, de egyben bizonytalanná, nehezen becsülhetővé is teszi. Ez problémát jelent olyan valósidejű alkalmazásokban, amelyekben kritikus az egyes programrészletek végrehajtási idejének pontos ismerete. Az STM32F7 mikrokontrollerekben az utasítás- és az adatcache is külön-külön letiltható.

utasítást (pl. MAC), valamint SIMD utasításokat is – bár utóbbiak csak 8 vagy 16 bites egész számokra alkalmazhatók[3].

Az aktív zajcsökkentési feladat elvégzésére különösen alkalmassá teszi a fejlesztőkártyát, hogy megtalálható rajta egy Wolfson WM8994 audio codec, amely rendelkezik egy sztereó bemenettel, egy sztereó kimenettel, egy SPDIF bemenettel, valamint két digitális MEMS mikrofon is van hozzá csatlakoztatva[4]. Az elosztott működéshez szükséges kommunikáció megvalósítására lehetőséget kínál a kártyán szintén megtalálható 10/100 Mb/s Ethernet port, amelyet a mikrokontrollerhez RMII interfészen keresztül csatlakozó Microchip LAN8742A Ethernet transciever szolgál ki.

A fejlesztőkártya tartalmazza a mikrokontroller felprogramozásához és a program debugolásához szükséges ST-Link debugger áramkört. A futási idejű hibakeresésben és a futó program viselkedésének analízisében szintén sokat segít a 480×272 felbontású, színes kijelző, amely kapacitív érintőfelülettel is rendelkezik. A kártyán helyet kapott egy 8 MB méretű külső SDRAM is, amely nagyon hatékonyan el tudja látni a kijelző képernyőbufferének feladatát. Található továbbá a kártyán két USB On-The-Go port, egy microSD foglalat, valamint egy kamerainterfész.

Az STM32F746G Discovery kártya ára megközelítőleg 15 ezer forint. Ennél jelentősen alacsonyabb összköltségre egy saját célhardver elkészítése esetén sem számíthatnánk, még akkor sem, ha az csak a kitűzött feladat megvalósítása szempontjából feltétlenül szükséges elemeket tartalmazza (mikrovezérlő, tápegység, analóg jelek illesztése, kommunikációs interfész, programozó interfész). A fejlesztőkártyából – a Méréstechnika és Információs Rendszerek Tanszék támogatásával – két darabot szereztem be, célom tehát egy elosztottan működő, kétcsatornás aktív zajcsökkentő mintarendszer létrehozása volt.

3.3.2. Fejlesztőeszközök és szoftverkomponensek

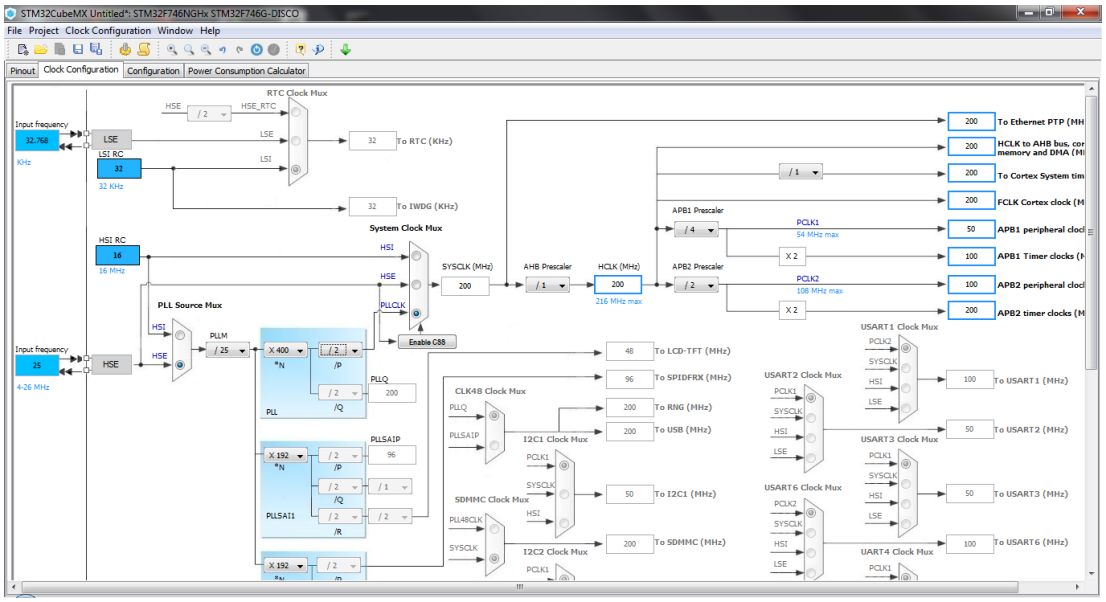
Az ARM architektúrájú maggal rendelkező mikrokontrollerek használatának egyik előnye az ingyenesen elérhető fejlesztőeszközök széles választéka. Teljes funkcionalitású, grafikus fejlesztőkörnyezetet mi magunk is összeállíthatunk, ennek legtipikusabb példája az architektúrának megfelelő fordító és debugger toolchain Eclipse alá történő telepítése. Számos cég kínál hasonló módon készült, előre összeállított, egyszerűen telepíthető és használható megoldásokat, ilyen például a CooCox CoIDE, valamint az Atollic TrueSTUDIO. Az elosztott aktív zajcsökkentő rendszer fejlesztéséhez ez utóbbit választottam.

Az STMicroelectronics maga is kínál a mikrovezérlőire és fejlesztőkártyáira törté-

[3] A lebegőpontos adatokon is használható, „valódi” SIMD utasításokat az ARM csak a Cortex-A családban vezette be, ARM NEON néven.

[4] A fejlesztés során azonban kiderült, hogy a ki- és bemenetek nem használhatók tetszőleges kombinációban – a részleteket lásd a következő fejezetben.

nő fejlesztést megkönnyítő segédeszközöket; egy ilyen program az STM32 CubeMX, amelynek segítségével könnyedén generálhatunk kiinduló kódot a projektünkhöz. Tartalmazza a fejlesztőkártyákon megtalálható perifériákat kezelő drivereket, tartalmaz számos middleware szoftverkomponenst többek között fájlrendszerek kezeléséhez, kommunikációs protokollok és beágyazott operációs rendszerek használatához, valamint képes legenerálni a mikrokontrollerünk belső egységeit előírásainknak megfelelően inicializáló kódot.



3.5. ábra. *Órajelkonfiguráció CubeMX segítségével*

Az aktív zajcsökkentő rendszer fejlesztése során a perifériák többségét kezelő programrészleteket magam készítettem el, azonban a CubeMX nagy segítségemre volt a különböző konfigurációs lehetőségek áttekintésében.

Az Ethernet alapú kommunikációhoz middleware komponensként felhasználtam az lwIP (lightweight IP) nevű nyílt forráskódú TCP/IP stack implementációt. Felhasználtam továbbá a CMSIS nevű, ARM mikrovezérlőkhöz készült, gyártófüggetlen hardverabsztrakciós réteg (HAL) néhány – digitális jelfeldolgozási feladatokat támogató – függvényét.

4. fejezet

Megvalósítás

Ebben a fejezetben bemutatom a többcsatornás, elosztott FxLMS algoritmus STM32F746G Discovery kártyákon történő implementálásának főbb lépéseit, valamint a fejlesztés közben felmerülő érdekesebb kérdéseket.

4.1. Az analóg jelek illesztése

Az aktív zajcsökkentő mintarendszer kialakításakor továbbra is éltem azzal a már bevezetett feltételezéssel, hogy egy mote egyetlen mikrofon-hangszóró párért felel (mint az 1.4. ábrán). Ekkor egy mote-nak három analóg jelet kell tudnia kezelni: a saját hibajelét és beavatkozó jelét, valamint a zajforrás által előállított referenciajelet. Az előző fejezetben a fejlesztőkártya egyik előnyeként említettem az audio codec meglétét, amely – a MEMS mikrofonokkal és az SPDIF bemenettel együtt – összesen hét analóg csatorna kezelésére alkalmas. A fejlesztés során azonban kiderült, hogy külön-külön a ki- és bemenetek mindegyike működik ugyan, de nem használhatók tetszőleges kombinációban. Például egy MEMS mikrofon használata esetén az analóg vonalbemenet már nem áll rendelkezésre. Sőt az analóg vonalbemenet és a hangszórókimenet sem használható egyszerre.

A korlátozások oka a codec és a mikrokontroller közötti illesztés sajátosságaiban keresendő[1]. Az STMicroelectronics hivatalos anyagaiban – a gyártó saját drivereinek kódjában található kommentektől eltekintve – a jelenség nincs dokumentálva, ezért valószínűsíthető, hogy nem tudatos tervezői döntés eredménye. Fejlesztői fórumokat böngészve találhatók a problémát részben áthidaló félmegoldások, azonban ezek csak erős kompromisszumok árán használhatók (például az analóg be- és kimenet bizonyos feltételek betartása esetén működtethető egyszerre, kis jelszintekkel és nagyon rossz jel/zaj viszonnyal).

[1] Az egymást kölcsönösen kizáró csatornák a TDM üzemmódban működő interfész azonos időszeleteit használják.

Ha a fejlesztőkártya teljesen képtelen volna analóg be- és kimenetek egyidejű kezelésére, az a jelfeldolgozási alkalmazások döntő többségében igen komoly hátrányt jelentene. Szerencsére azonban a probléma viszonylag egyszerűen megoldható, ugyanis a kártyán található mikrokontroller rendelkezik három különálló 12 bites A/D átalakítóval, amelyek összesen 24 csatorna multiplexelt mintavételezésére képesek, akár 2,4 MSPS sebességgel.

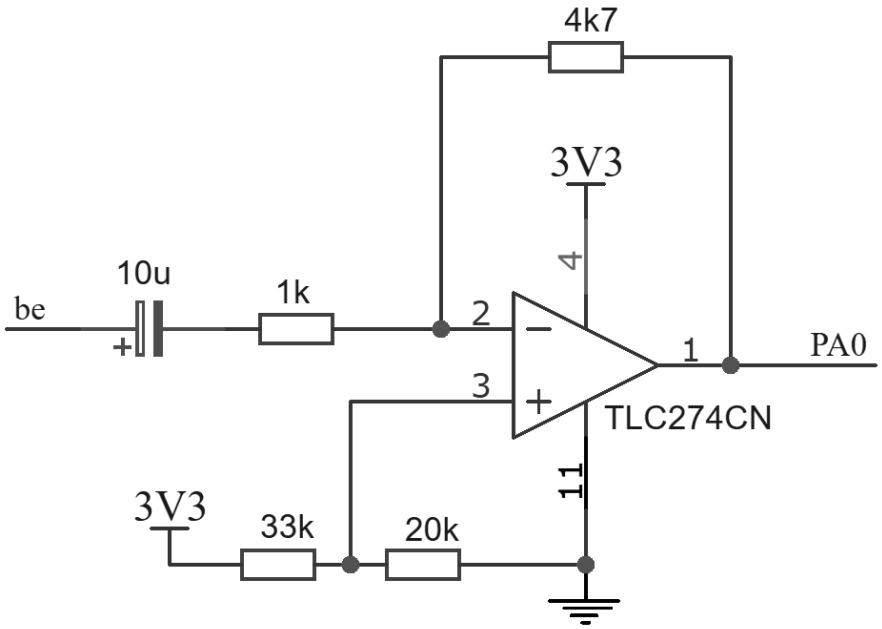
A fenti megfontolásokat szem előtt tartva az analóg jelek illesztését az alábbi módon terveztem meg:

- Beavatkozó jel: a codec analóg kimenetének jobb csatornája
- Hibajel: bal MEMS mikrofon
- Referenciajel: az ADC1 periféria 0. csatornája (PA0 láb)

Ez a kombináció megvalósítható, azonban felmerül egy kisebb nehézség: az A/D átalakító a bemeneti feszültségtartománya 0...3,3 V, ezzel szemben a vonalszintű hangfrekvenciás jelek tipikusan nulla középértékűek és kisebb amplitúdójúak. Vagyis egy szokványos audio jelet nem adhatunk rá nyersen az ADC bemenetére, mert egyrészt ki sem vezérelné azt kellőképpen (ami a kvantálásból eredő jel/zaj viszony romlásához vezetne), másrészt pedig a negatív jelszakaszok jó esetben csak csonkolódnának, rosszabb esetben pedig akár a bemenetet is károsíthatnák az esetlegesen fellépő latch-up jelenség következtében. Tény ugyan, hogy az ADC a referenciajelet mintavételezi, amelyet célszerűen egy függvénygenerátor állít elő, azonban ezt a jelet a zajforrást képviselő hangszórón is ki kell adni. Aktív hangfal alkalmazása esetén tehát a referenciajelnek vonalszintűnek kellene lennie, passzív hangszóró esetén pedig a 0...3,3 V tartományban mozgó jel nulla középértéke okozna gondot, mivel folyamatosan hatásos teljesítményt disszipálna a tekercs ohmos ellenállásán, ráadásul a nem az egyensúlyi helyzete körül lengő membrán rontaná a hangszóró átviteli tulajdonságait.

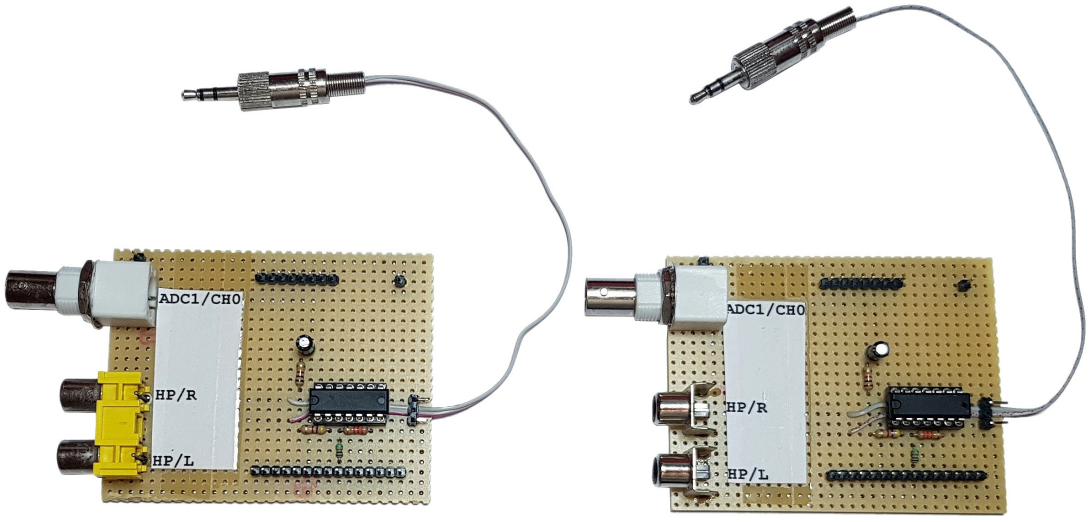
A probléma orvosolására elkészítettem a 4.1. ábrán látható egyszerű szintillesztő áramkört – méghozzá két példányban, egyet-egyet mindkét mote-hoz. Az áramkör a vonalszintű referenciajelen affin transzformációt (eltolást és erősítést) végez, ezáltal a kimenő jel biztosan az A/D átalakító bemeneti tartományába esik (már csak az erősítő táplálása miatt is), viszonylag jól kivezérelve azt. A TLC274CN típusú műveleti erősítő kiválasztását több szempont is indokolta. A tápellátást szerettem volna megoldani a fejlesztőkártyán hozzáférhető 3,3 V-os feszültségforrásról, azonban a legtöbb műveleti erősítő ennél magasabb szintű és gyakran szimmetrikus tápforrást igényel; a TLC274CN ezzel szemben már 3 V-tól működőképes. Ilyen tálplálás mellett fontos az erősítő közelítőleg rail-to-rail jellege: a kimenet nulla voltig teljesen

le tud menni, maximális értéke pedig kb. 2,7 V. A szinteltolást beállító ellenállások értékét úgy választottam meg, hogy a kimeneti tartományt közelítőleg teljesen ki lehessen használni, vagyis a felső és az alsó telítés egyszerre kezdődjön el. Nem utolsó szempont továbbá, hogy ebből a típusból több darab is azonnal rendelkezésemre állt. Tény azonban, hogy a TLC274-es tokjában négy műveleti erősítő is van, amiből három itt kihasználatlan, így egy végleges, nem kísérleti jelleggel fejlesztett rendszerbe nem ez lenne a legjobb választás.



4.1. ábra. *Az ADC szintillesztő kapcsolási rajza*

Az áramköröket próbapanelen építettem meg úgy, hogy illeszkedjenek a fejlesztőkártya alsó oldalán elhelyezett – a népszerű Arduino kártyákkal kompatibilis elrendezésű – bővítőcsatlakozókhoz, amelyeken keresztül hozzáférhető mindhárom szükséges jel (a tápfeszültség, a föld, valamint a PA0 láb). A paneleken a későbbi használat kényelmét szem előtt tartva a referenciajel illesztésén kívül néhány egyéb jelet is kivezettem: az analóg kimenet jobb és bal csatornáját külön-külön RCA csatlakozókon keresztül tettem egyszerűbben hozzáférhetővé, kivezettem továbbá egy kimenetként konfigurált GPIO lábat az algoritmusok futásidejének méréséhez. Az elkészült áramkörök fotója a 4.2. ábrán látható.



4.2. ábra. *A kiegészítő panelek*

4.2. A jelfeldolgozási műveletek implementálása

Az LMS típusú algoritmusok futtatása során véges impulzusválaszú szűrőket működtetünk, melyek közül néhánynak az együtthatókészletét időnként frissítjük is. Amint a 2.2. szakaszban említettem, dolgozatomban a FIR szűrési műveleteket időtartománybeli konvolúcióként tekintem, amely megvalósításának bevett és hatékony lehetősége a cirkuláris buffer alapú implementáció.

Az alábbi kódrészlet erre mutat példát. Az `ident()` függvény végrehajtja a zajcsökkentést megelőző identifikációs folyamat egyetlen ütemét. Maga a cirkuláris bufferrel megvalósított konvolúció a 13. sorban, az adaptív szűrő S együtthatóvektorának frissítése pedig a 15. sorban látható.

```
1  float S[N_S];
2  // ...
3  void ident(void){
4      static float y_buf[N_S];
5      float y, e;
6      static uint16_t i=0;
7      uint16_t j, ix;
8      e=micL;
9      y=rand();
10     outR=y;
11     y_buf[i]=y;
12     ix=i;
13     for(j=0;j<N_S;j++,ix++) e-=y_buf[ix%N_S]*S[j];
14     e*=MU;
15     for(j=0;j<N_S;j++) S[j]+=e*y_buf[(j+i)%N_S];
16     i=i?(i-1):(N_S-1);
17 }
```

A CMSIS absztrakciós réteg DSP könyvtára tartalmaz különböző szűréseket megvalósító függvényeket, sőt az LMS algoritmus implementációja is megtalálható benne. Ezek a függvények a legtöbb esetben nagyon hatékonyan képesek működni, mert az adott architektúra jellemzőit figyelembe vevő megoldásokat eszközölnek[2]. Egy példa erre a ciklusoknak az utasítás pipeline szerkezetétől függő mértékű kibontása (unroll), amely a ciklusmag minden egyes végrehajtását követő feltételvizsgálatokból fakadó ciklusszervezési overheadet képes csökkenteni. Továbbá természetesen a CMSIS függvények használatával egyszerűsödik programunk kódja, illetve általában az átláthatósága is javul.

Az alábbi kódrészlet funkciójában ekvivalens a fenti identifikációs algoritmussal, azonban ez az implementáció a CMSIS DSP könyvtár által kínált lehetőségekkel él. Az `ident()` függvény törzse itt mindössze három sor, az LMS algoritmus egyetlen függvényhívássá egyszerűsödött. A CMSIS függvények és típusok nevében az `_f32` postfixum a 32 bites lebegőpontos adattípust jelöli.

```
float S[N_S];
arm_lms_instance_f32 lmsS;
float lmsSState[N_S];
// ...
arm_lms_init_f32(&lmsS, N_S, (float32_t*)S, (float32_t*)lmsSState, MU, 1);
// ...
void ident(void){
    y=rand();
    outR=y;
    arm_lms_f32(&lmsS, (float32_t*)&y, (float32_t*)&micL, (float32_t*)&yS,
    (float32_t*)&e, 1);
}
```

Sajnos a CMSIS DSP könyvtárban megtalálható LMS implementáció nem képes szűrt referenciajellel működni; az adaptív szűrő bemenetén és az adaptációs szabályban szükségképpen ugyanaz a jel szerepel. Az FxLMS algoritmus kódját tehát nem tudjuk három sorra egyszerűsíteni, azonban a FIR szűrésekhez azért felhasználhatjuk a megfelelő CMSIS függvényeket. Az egycsatornás FxLMS algoritmus egy ütemét megvalósító függvény kódja alább olvasható:

```
float S[N_S], W[N_W];
arm_fir_instance_f32 firS, firW;
float firSState[N_S], firWState[N_W];
// ...
arm_fir_init_f32(&firS, N_S, (float32_t*)S, (float32_t*)firSState, 1);
arm_fir_init_f32(&firW, N_W, (float32_t*)W, (float32_t*)firWState, 1);
// ...
void fxlms(void){
    static float r_buf[N_W];
```

[2]Az architektúra fordítási idejű azonosítása a CMSIS kódokban makrók definiálásán keresztül történik.

```
10      static uint16_t iW=0;
11      uint16_t j, k;
12      float e, y, r, x;
13      x=adcGet();
14      /* r=S*x */
15      arm_fir_f32(&firS, (float32_t*)&x, (float32_t*)&r, 1);
16      /* y=W*x */
17      arm_fir_f32(&firW, (float32_t*)&x, (float32_t*)&y, 1);
18      /* dW=mu*e*R */
19      e=MU*micL;
20      r_buf[iW]=r;
21      k=N_W-1;
22      for(j=iW;j<N_W;j++,k--) W[k]+=e*r_buf[j];
23      for(j=0;j<iW;j++,k--) W[k]+=e*r_buf[j];
24      /* frissites a kovetkezo utemre */
25      iW=iW?(iW-1):(N_W-1);
26      outR=-y;
27      adcStart();
28  }
```

Megfigyelhető az itt bemutatott kódrészletek azon közös tulajdonsága, hogy az adott algoritmusnak mindig csak egyetlen feldolgozási ütemét hajtják végre. Ez a jelfeldolgozó rendszerekben futó programok tipikus szervezési módjából adódik. A beavatkozó és a hibajel mintáinak a codec és a memória közötti továbbítását a DMA egység végzi, a főprogram pedig végtelen ciklusban várakozik a DMA vezérlő jelzésére. Ez a jelzés megszakításvezérelt módon következik be akkor, amikor beérkezett a hibajel új mintája, és kiküldésre került a beavatkozó jel előző mintája. Ekkor tehát fel kell dolgoznunk az új bemeneti mintát, és elő kell állítanuk egy új kimeneti mintát. Fontos, hogy a hibajel új mintájának kiolvasása előtt az adatcache-ben érvénytelennek jelöljük ezt a memóriaterületet, máskülönben nem garantált, hogy a kiolvasás valóban a legfrissebb adatot adja vissza. A referenciajelet a mikorvezérlő saját A/D átalakítója mintavételezi, a konverzió indítása mindig a feldolgozási ütem végén történik, az eredmény kiolvasása pedig a következő ütem elején. Az alábbi kódrészlet a program főciklusát mutatja:

```
1   while(1){
2       if(inTransferComplete){
3           ledOn();
4           inTransferComplete=0;
5           SCB_InvalidateDCache_by_Addr((uint32_t*)inBuffer, 4);
6           micL=inBuffer[1];
7           outBuffer[1]=outR;
8           fxlms();
9           ledOff();
10      }
11  }
```

4.3. Kommunikációs interfész

Az előző szakaszban bemutatott kódrészletek összeillesztésével lényegében megkapjuk az egycsatornás zajcsökkentő rendszer implementációját, amelyet már csak a kommunikáció megszervezése választ el a többcsatornás, elosztott működéstől.

Az Ethernet alapú kommunikáció implementációjához a már említett lwIP nevű middleware-t használtam, amely a protokollok bonyolult részleteit elfedve egyszerű interfészt kínál a fejlesztő számára a hálózat funkcióinak használatához. Sőt valójában az lwIP használatakor három különböző szintű programozói interfész közül is választhatunk (raw, netconn, socket), melyek közül én a legalacsonyabb szintű, eseményvezérelt függvényekkel operáló változatot választottam.

A megvalósított mintarendszerben az egymáshoz routeren[3] keresztül csatlakozó mote-ok előre beállított, statikus IP címmel rendelkeznek, és egy adott porton beérkező UDP üzenetcsomagokat várnak. Ennek beállítása mindössze néhány függvényhívást igényel:

```
1 struct udp_pcb *pcb;
2 pcb=udp_new();
3 udp_bind(pcb, IP_ADDR_ANY, PORT);
4 udp_recv(pcb, recvCallback, NULL);
```

Az lwIP ún. Protocol Control Block struktúrákkal azonosít egy-egy kapcsolatot, a fenti kódrészlet 2. sorában egy ilyen létrehozása látható. Ezután előírjuk, hogy a megadott (itt makróként definiált) porton bármilyen IP címről érkező üzeneteket elfogadjuk, végül pedig beállítjuk, hogy üzenet vétele esetén melyik függvény hívódjon meg. Ennek a callback függvénynek az implementációjában valósíthatjuk meg az üzenetek feldolgozásának első lépését; ez esetünkben a vett adatok kimásolását, valamint a vétel tényének jelzését foglalja magában. A vett adatokat egy láncolt listában kapjuk meg, első lépésben ennek tartalmát kell átmásolnunk egy globális tömbbe. Amikor elkészültünk, egy flag beállításával jelezzük a program főciklusának, hogy van új, feldolgozatlan adat a bufferben, ideje felhasználni az adaptív szűrő együtthatókészletének frissítésére.

```
1 static void recvCallback(void *arg, struct udp_pcb *pcb, struct pbuf *p,
2                           ip_addr_t *addr, uint16_t port){
3     copyPayload(p);
4     pbuf_free(p);
5     recvFlag=1;
6 }
```

[3]Két mote esetében nem feltétlenül lenne szükséges routert használni, azonban egy kiterjedtebb rendszerben valamilyen útválasztó csomópont alkalmazása megkerülhetetlen. Továbbá a fejlesztési, debugolási folyamatot jelentősen megkönnyítették az OpenWrt-t futtató router szolgáltatásai.

Az adatok küldése hasonlóan egyszerű módon megvalósítható: összeállítjuk a küldendő üzenetet, majd egy IP cím és egy port szám segítségével megcímezzük a másik mote-ot. A különbség, hogy míg a vétel eseményvezérelt módon történik és tényét a főciklus „veszi észre”, addig a küldést a maga főciklus kezdeményezi periodikusan, adott idő (ütemszám) letelte után.

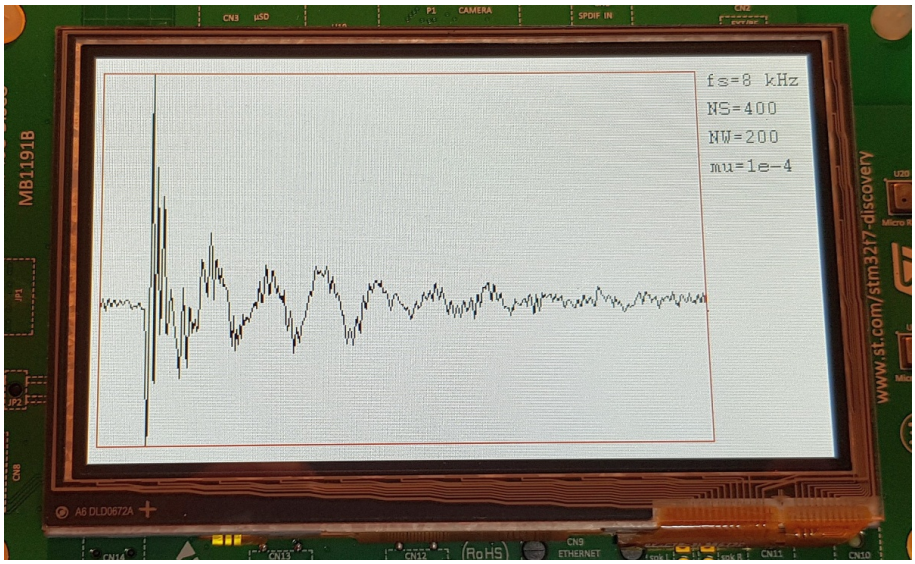
```
p=pbuf_alloc(PBUF_TRANSPORT, N_W, PBUF_REF);
p->payload=(void*)dWs;
SCB_InvalidateDCache_by_Addr((uint32_t*)dWs, N_W*sizeof(float));
udp_sendto(otherPCB, p, &otherIP, OTHER_PORT);
pbuf_free(p);
```

Ebben a kétcsatornás, elosztott alkalmazásban a program főciklusa tehát három feltételt ellenőriz: érkezett-e adat a másik mote-tól, kell-e már adatot küldeni a másik mote-nak, valamint előállt-e a hibajel új mintája.

4.4. Az elkészült rendszer működése

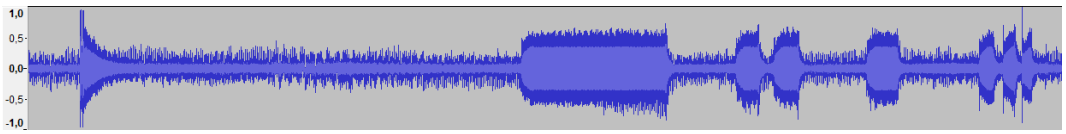
A fentiekben ismertetett fejlesztési lépések végrehajtása után előállt a kétcsatornás, elosztott, aktív zajcsökkentő rendszer működő prototípusa. A mintarendszer beüzemelése a fizikai elrendezés kialakításával kezdődik: a kártyákhoz csatlakoztatjuk a kiegészítő paneleket, bekötjük a referenciajelet és a hangszórókat, a kártyák Ethernet portjait routeren keresztül összekötjük és elvégezzük a beállítások harmonizációját (statikus IP címek, azonos alhálózat), csatlakoztatjuk a tápforrásokat, a futásidő mérhetősége érdekében kivezetett GPIO lábat opcionálisan oszcilloszkópra vezetjük.

A tényleges zajcsökkentést célzó működés megkezdése előtt el kell végezni a másodlagos akusztikus utak identifikációját. A 2.4. szakaszban megemlítettem ennek nehézségeit: a keresztutak identifikációjához az érintett mote-ok működésének szinkronizációja lenne szükséges. A jelenlegi rendszerben csak a probléma legegyszerűbb megoldása került implementálásra: az identifikációs fázis erejéig átmenetileg módosítani kell a rendszer struktúráján. A mote-ok kijelzőjén megjelenik, hogy melyik átvitel identifikációja következik éppen; az ennek megfelelő hangszórót kell az audio kimenet jobb csatornájára csatlakoztatni (természetesen magának a hangszórónak az elmozdítása nélkül). Az identifikáció ezután a kártyák alsó oldalán található nyomógomb lenyomásával indítható, és a gomb felengedéséig tart. Ekkor a mote a kijelzőjén megjeleníti az identifikált impulzusválaszt, ennek alapján megítélhető az identifikáció sikeressége vagy sikertelensége. A 4.3. ábrán egy sikeresen identifikált, tipikus impulzusválasz látható.



4.3. ábra. *Sikeres identifikáció*

Az identifikáció végeztével elkezdődik magának az elosztott FxLMS algoritmusnak a futtatása. A zajcsökkentés szüneteltethető a nyomógomb lenyomva tartásával, ezalatt az adott mote nem ad ki beavatkozó jelet, és nem adaptálja együttható-készletét. Fülünket a mikrofon környezetében tartva, és a gombot nyomogatva az algoritmus működéséről hallhatóan is meggyőződhetünk.



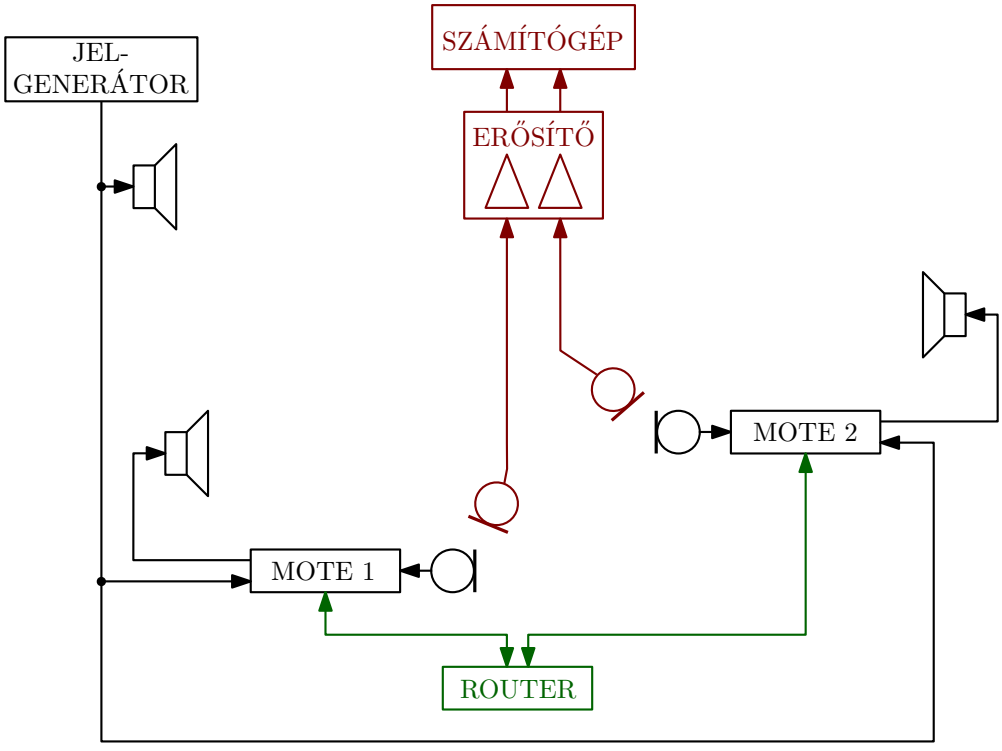
4.4. ábra. *Kezdeti beállás, illetve a gomb nyomogatásának hatása*

A megvalósított tesztalkalmazásban szereplő valamennyi adaptív algoritmusnak elkészítettem a CMSIS függvényeket használó és azok nélkül működő változatát is, így a futásidejük közötti eltérés tanulmányozható. Ha működés közben a mote-ok az ST-Link debuggeren keresztül számítógéphez csatlakoznak, akkor programjaik futása bármikor megállítható. Ekkor memóriáik tartalma is hozzáférhetővé válik, lehetővé téve például az identifikált átvitelek kiexportálását és MATLAB segítségével történő vizsgálatát.

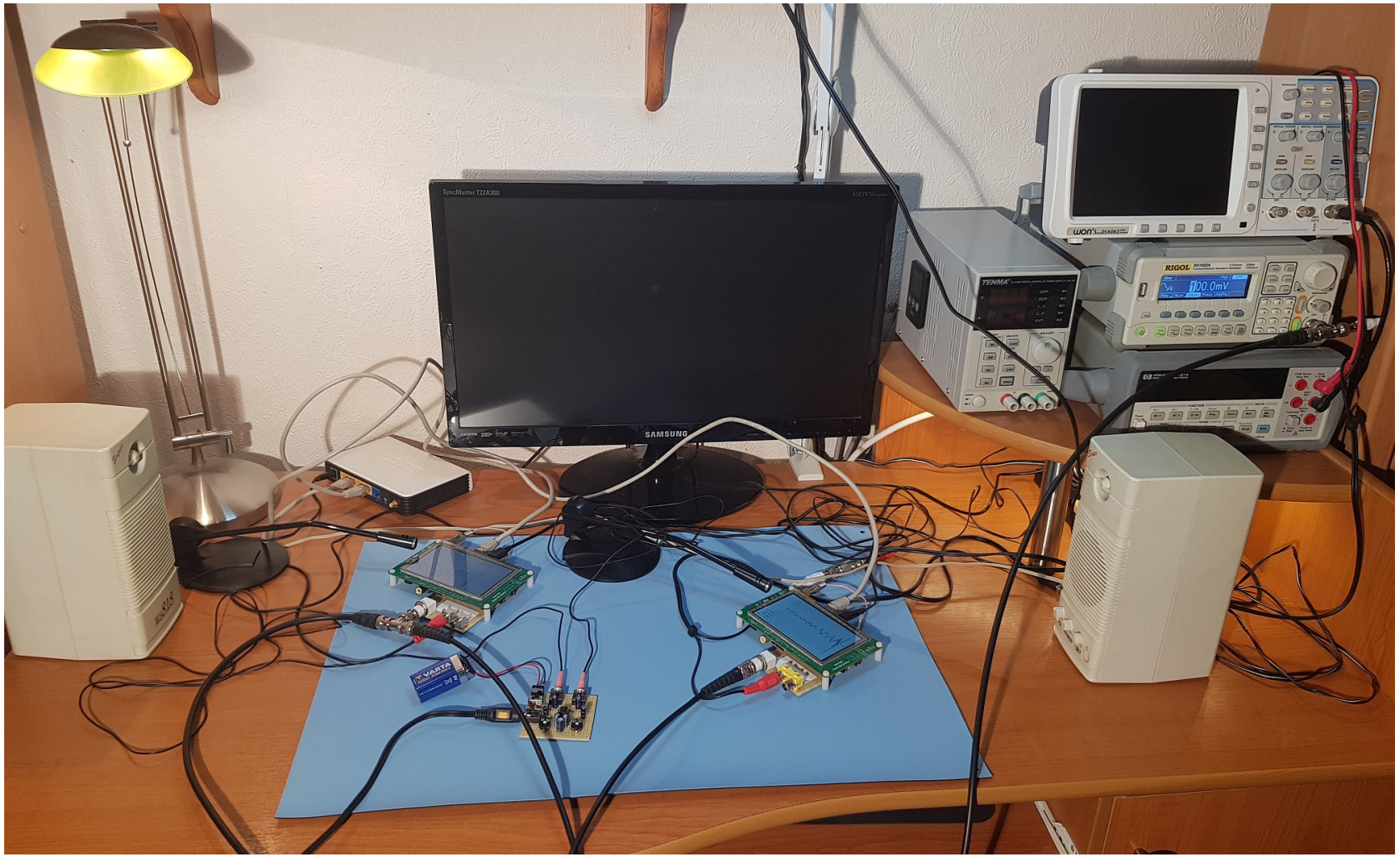
5. fejezet

Mérések

Az elosztott aktív zajcsökkentő rendszer elkészült prototípusának tulajdonságait az 5.1. ábrán vázolt elrendezésben vizsgáltam. A referenciajelet digitális függvénygenerátor állította elő, és egy aktív hangszóró szólaltatta meg. A hibajelek mérése a mote-ok saját MEMS mikrofonjainak közelségében elhelyezett egyszerű PC mikrofonokkal történt. A mikrofonok jelét saját építésű előerősítő kombinálta a számítógép vonalbemenetéhez illeszkedő szintű sztereó jellé, amelynek rögzítéséhez az Audacity nevű alkalmazást használtam, utólagos feldolgozásához pedig MATLAB-ot. Ezen kívül az algoritmus egyes részleteinek végrehajtási idejét esetenként oszcilloszkóp segítségével mértem, ez az ábrán nincs feltüntetve. A mintavételi frekvencia minden mérés során 8 kHz volt.



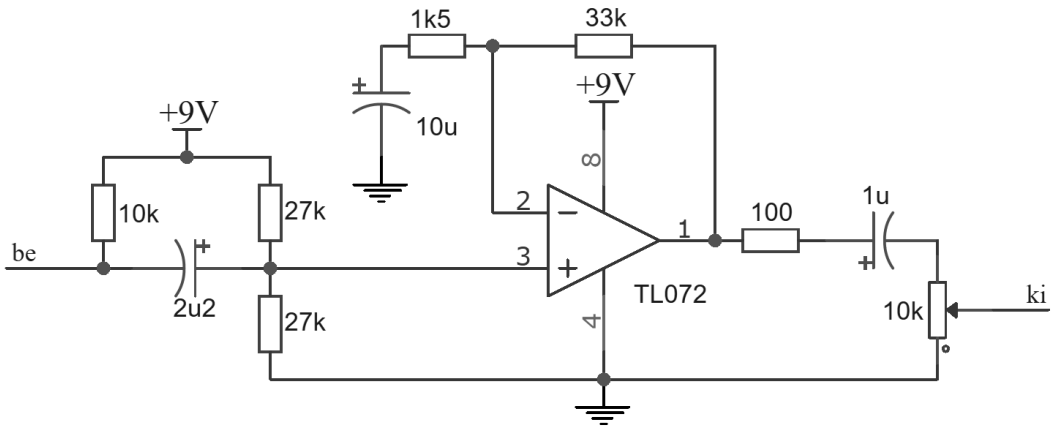
5.1. ábra. *A mérési elrendezés vázlata*



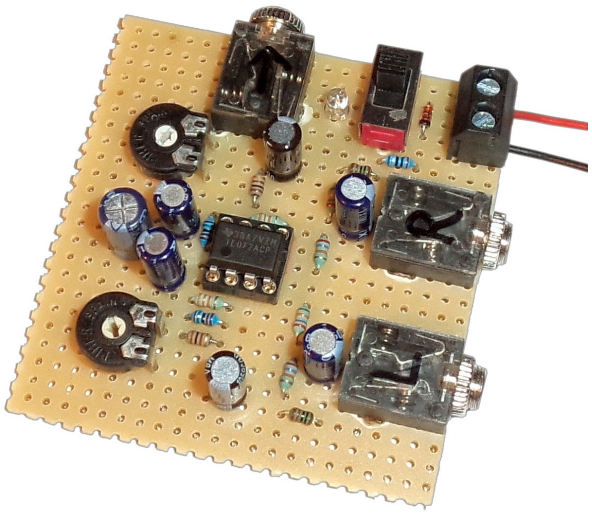
5.2. ábra. *A mérési elrendezés fotója*

Az 5.2. ábrán a konkrét fizikai elrendezés fotója látható. A képen nem kapott helyet a referenciajelet megszólaltató hangszóró, valamint a mikrofonok erősített jeleit rögzítő számítógép (előbbi a monitor fölötti polcon, utóbbi pedig az asztal alatt helyezkedett el). A futásidők mérésére alkalmas oszcilloszkóp ugyan látható a kép jobb felső sarkában, de a fotó elkészítésének idején nem volt használatban. Említést érdemel továbbá, hogy a rendszer komponensei a fotó erejéig némileg össze lettek zsúfolva; a képen láthatónál általában nagyobb fizikai kiterjedésű elrendezéseket vizsgáltam.

A hibajeleket rögzítő mikrofonok számára az 5.3-az 5.4. ábrákon látható előerősítő áramkört készítettem el. Erre azért volt szükség, mert a rendelkezésemre álló számítógépek egyike sem volt felszerelve sztereó mikrofonbemenettel, amely lehetővé tette volna a két hibajel egyidejű, eltolásmentes rögzítését, rendelkeztek viszont sztereó vonalbemenettel. Ezen felül az erősítő kényelmessé tette a hibajelek oszcilloszkópon történő megfigyelését is.



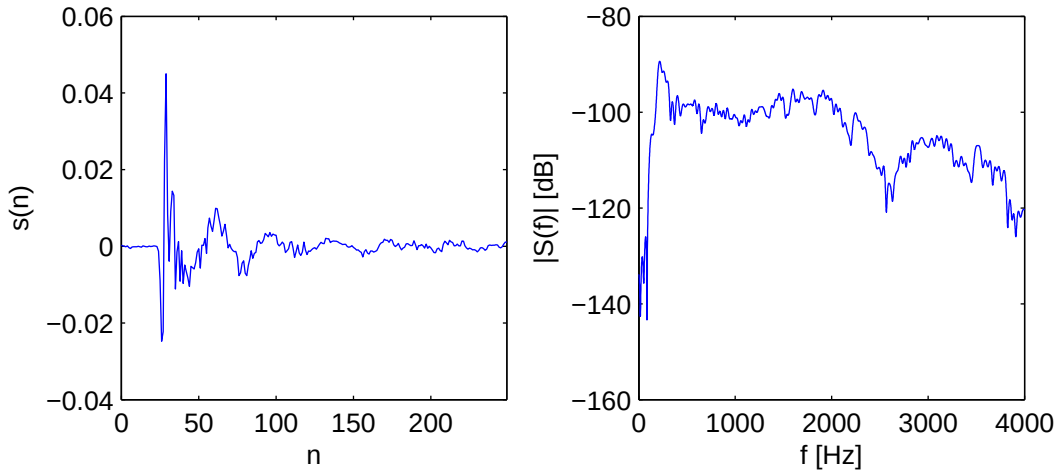
5.3. ábra. *Az előerősítő kapcsolási rajza*



5.4. ábra. *Az előerősítő fotója*

Az érdemi mérések megkezdése előtt az egyes kódrészletek CMSIS függvényeket használó, valamint azok nélkül működő implementációjának összehasonlítását végeztem el. Azonos együtthatószámok esetén az LMS algoritmus CMSIS alapú implementációjának futásideje megközelítőleg háromnegyede volt a saját implementáció futásidejének. Az FxLMS algoritmus esetében a CMSIS előnye még nagyobbnak bizonyult: a futásidő majdnem 40%-kal csökkent. Ezen eredmények ismeretében a későbbi mérések során az algoritmusoknak csak a CMSIS alapú változatát használtam fel. Az egycsatornás FxLMS algoritmus 400 együtthatós $S(z)$ és 200 együtthatós $W(z)$ átvitel mellett a feldolgozásra fordítható idő (125 µs) 85-90%-os kihasználtsága mellett futott. Hasonló eredmény a kétcsatornás, elosztott esetben $N_s = 200$ és $N_w = 200$ beállítás mellett adódott. A mért futásidő csekély mértékű ingadozása származhat a cache memóriák működéséből, valamint a jelfeldolgozáson kívüli feladatok kiszolgálásából (üzenetküldés, megszakítási rutinok stb.).

Az elkészült rendszer bemérését a zajcsökkentést megelőző identifikációs folyamat eredményének vizsgálatával kezdtem. A másodlagos utak identifikációját többféle elrendezésben, több különböző együtthatószám (50-400) mellett is elvégeztem, ezt követően pedig a kiadódó szűrőegyütthatókat (azaz a valóságos átvitelt becslő FIR rendszer impulzusválaszát) a fejlesztőkörnyezet debugger funkcióinak segítségével kiolvastam a mikrokontroller memóriájából. Az 5.5. ábrán egy tipikus ilyen impulzusválasz (balra) és a neki megfelelő amplitúdókarakterisztika (jobbra) látható.



5.5. ábra. *Az identifikált átvitel*

Az amplitúdómenetben megfigyelhető 100 dB körüli csillapítás az identifikált átvitelek mindegyikének közös jellemzője volt. Ez azonban nyilvánvalóan nem akusztikus eredetű csillapítás – hiszen a mikrofonok és a hangszórók távolsága minden esetben legfeljebb két-három méter volt –, sokkal inkább skálázási, számábrázolási okokra lehet visszavezethető. Az amplitúdómenet jellegre mindenképpen helyes: nagyon alacsony frekvenciákon nulla közeli az átvitel, ettől eltekintve enyhe aluláteresztő

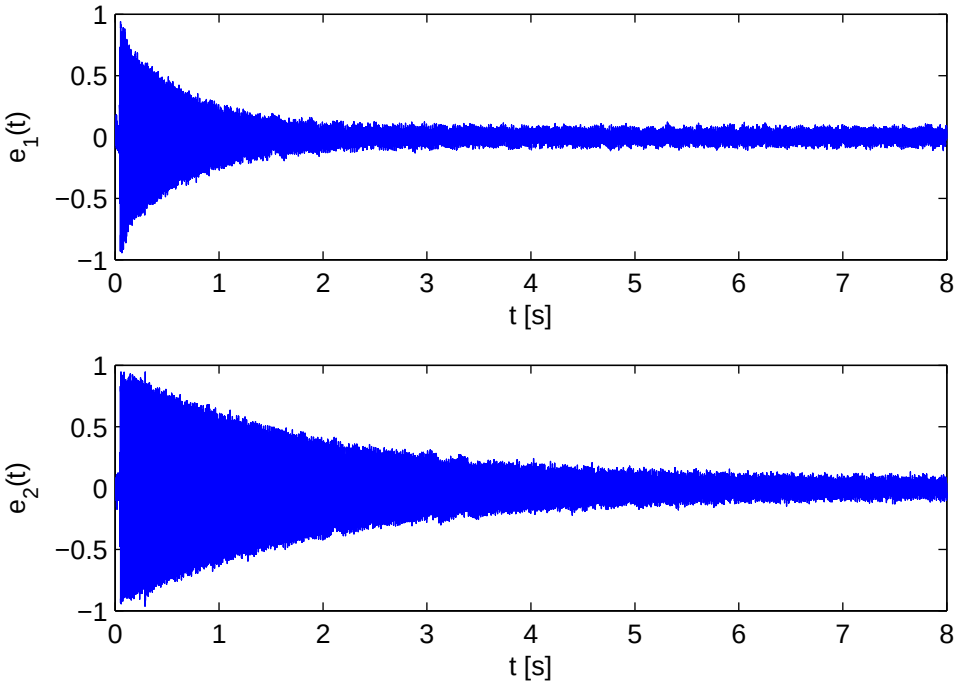
jelleget mutat, megtalálható benne néhány kiemelési és leszívási pont, valamint az impulzusválaszban is megjelenik néhány mintányi késleltetés, amely függ az identifikált út hosszától.

Az identifikáció eredményeinek vizsgálata után a rendszer analízisét a megvalósított aktív zajcsökkentés konvergenciatulajdonságainak tanulmányozásával folytattam. Ezen mérések legtöbbje $N_w = 150$ és $N_s = 200$ együtthatószámok mellett történt, a változó paraméter pedig az üzenetküldések gyakorisága volt. Az egyes mote-ok hibajelének rögzítését a már említett módon, számítógéppel végeztem, a regisztrátumokat pedig MATLAB segítségével ábrázoltam.

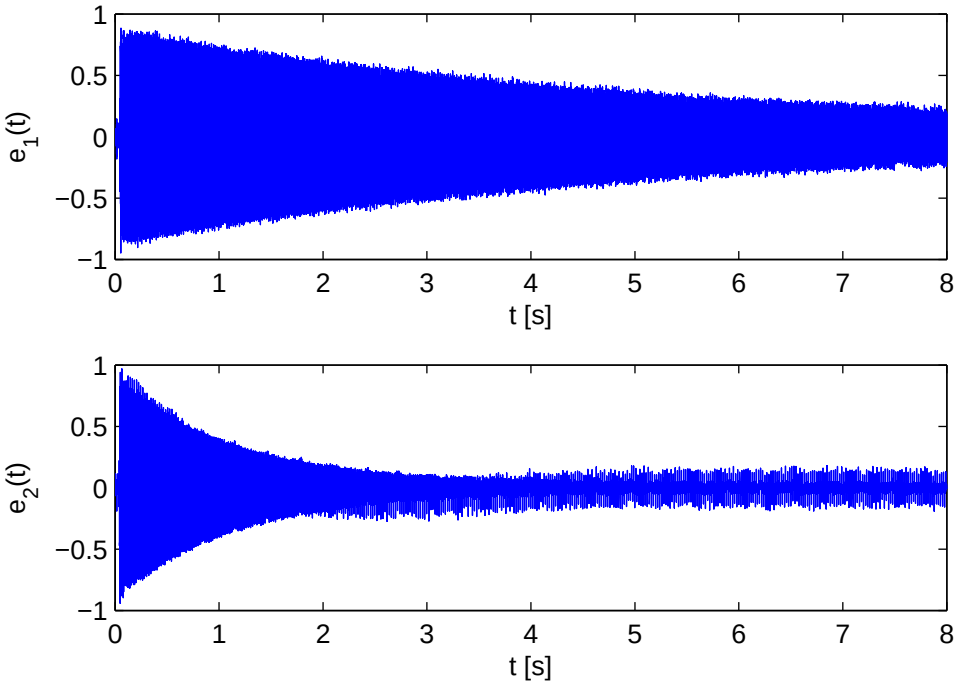
Hamar kiderült, hogy a 2.4. szakasz végén ismertetett szimulációs eredményeket könnyű reprodukálni: csupán az üzenetküldési gyakoriságok változtatásával adott elrendezés és zavarjel mellett is nagyon különböző beállási idők idézhetők elő. A 2.15. ábrán látotthoz hasonló mintát azonban nem sikerült megfigyelnem, a rögzített hibajelekben nem látszanak markánsan az együttható-módosítások időpontjai. Az 5.6. és az 5.7. ábra a két mote hibajelének beállását mutatja egy periodikus (fűrészfogjel) zavarás kioltása közben, a referenciajel első bekapcsolásának pillanatától kezdve. A mote-ok nagyjából két méterre helyezkedtek el egymástól a földön, a hangszóróik különböző irányokba néztek.

Az első esetben az adatküldések gyakorisága megegyezett: mindkét mote 40 ütemenként küldte el a másik számára előkészített együttható-frissítést, adaptív szűrőiket tehát azonos gyakorisággal frissítették. A beállások ennek ellenére jól láthatóan eltérő sebességűek, amelyet valószínűleg a térbeli elrendezés sajátosságai magyaráznak. Az aktív zajcsökkentési kísérleteket egy viszonylag kicsi, zárt szobában végeztem, ahol megszólaltatott referenciajel mellett fel-alá járkálva emberi füllel is könnyen észlelhetők voltak kioltási és kiemelési helyek (a zajcsökkentő algoritmus működése nélkül is). Ennek tükrében nem meglepő, hogy a különböző pontokban elhelyezett mote-ok zajelnyomásának konvergenciasebessége olykor jelentősen eltérőnek mutatkozott.

A második esetben az 1-es mote 10, a 2-es pedig 100 ütemenként kezdeményezett adatküldést. Mivel az együtthatókészlet adaptációja mindig üzenetek vételekor valósul meg, most a 2-es mote 10, az 1-es pedig 100 ütemenként frissítette az adaptív szűrőjének paramétervektorát. Ez a különbség láthatóan megmutatkozott a beállási időkben is: az 1-es mote konvergenciasebessége sokat romlott, a 2-esé ezzel szemben némileg javult.



5.6. ábra. *Beállás azonos üzenetküldési gyakoriságok esetén*



5.7. ábra. *Beállás különböző üzenetküldési gyakoriságok esetén*

Az itt bemutatott mérések reprodukálhatóságáról meggyőződtem, az elrendezést és a referenciajelet változatlanul hagyva a zajcsökkentő rendszer többszöri elindítása minden esetben a fentiekhez hasonlóan alakuló jeleket eredményezett. Ez megerősíti a 2.4. szakaszban megfogalmazott azon feltevést, mely szerint az üzenetküldési gyakoriságok hangolásával befolyásolhatók a konvergenciatulajdonságok.

6. fejezet

Összefoglalás, értékelés

Diplomamunkám elkészítése során megismerkedtem a jelfeldolgozó rendszerekben – azon belül speciálisan az aktív zajcsökkentő alkalmazásokban – használatos adaptív algoritmusokkal, valamint megvizsgáltam az elosztott architektúrában történő végrehajthatóságuk lehetőségét. Leírtam a többcsatornás aktív zajcsökkentést megvalósító FxLMS algoritmus elosztott implementációjának egy lehetséges változatát, majd működőképességét MATLAB szimulációk segítségével igazoltam. Ezután az elosztott rendszerek kialakítása során figyelembe veendő szempontokat szem előtt tartva kiválasztottam az algoritmus gyakorlatban történő kipróbálására alkalmas eszközöket. Végül elkészítettem egy kétcsatornás, elosztott, aktív zajcsökkentő rendszer egy működő prototípusát, amelynek tulajdonságait mérésekkel vizsgáltam.

A kialakított mintarendszer működése igazolta az Ethernet alapú hálózatok elosztott jelfeldolgozó rendszerekben történő alkalmazásának életképességét, a számítógépes szimulációk eredményeivel jól egybevágó mérési eredmények pedig megerősítették az FxLMS algoritmus elosztott végrehajtó egységekre történő dekomponálásának helyes implementációját. A beállási idők vizsgálatának tanulságai igazolták, hogy az elosztott adaptív jelfeldolgozó rendszerek komponensei közötti adatcsere megszervezésének részletei jelentősen befolyásolhatják az algoritmusok konvergenciatulajdonságait.

Az elosztott jelfeldolgozó rendszerek tanulmányozása és a tesztalkalmazás kifejlesztése rávilágított néhány olyan kérdésre, amelyek ugyan meghaladják jelen dolgozat kereteit, azonban kijelölhetik munkám folytatásának és a témakörben végezhető jövőbeli kutatásoknak az irányát. Ezek az alábbiak:

- **A beállási idők és az üzenetküldési gyakoriságok közötti összefüggés kvantitatív analízise.** Az elvégzett mérések rámutattak az összefüggés meglétére, illetve arra, hogy az esetek többségében annak jellege meg is felel a várakozásoknak – célszerű lehet tehát a matematikai részletek feltárásával is megpróbálkozni.

- **Összetettebb adatküldési rend.** Jelenleg az együtthatómódosítás-vektorok továbbítása periodikusan, az FxLMS algoritmus futásának teljes ideje alatt állandó periódusidővel történik. Érdekes lehet megvizsgálni, hogy vajon található-e ennél hatékonyabb, adaptív küldési gyakoriságokat implementáló módszer. Például: bekapcsoláskor gyakrabban küldünk adatokat, az állandósult állapot eléréséhez közelítve pedig egyre ritkábban; lassabban konvergáló mote-oknak gyakrabban küldjük az adatokat stb. Elképzelhető, hogy egy ilyen megoldás a beállási időket és a hálózat terhelését is mérsékelni tudná.

- **Bővíthetőség.** Noha az elkészített mintarendszer mindössze két mote-ból áll, a 2.4. pontban általános formában adtam meg az elosztott FxLMS algoritmus leírását. Ez alapján a rendszer újabb mote-okkal való bővítése viszonylag egyszerűen kivitelezhető, azonban a komponensek azonosítása jelenleg meglehetősen mereven, előre beprogramozott állandók és fix IP címek alapján történik. Dolgozatomban nem vizsgáltam az aktív zajcsökkentő rendszer dinamikus bővítésének lehetőségét, azonban egy ilyen opció kialakítása kétségtelenül fontos és elegáns továbbfejlesztése lenne a meglévő implementációnak.

- **Automatikus identifikáció.** Az elosztott aktív zajcsökkentő rendszer jelenlegi implementációjában a másodlagos utak identifikációja meglehetősen manuálisan zajlik, elkerülhetetlen a felhasználói beavatkozás. Egy automatikus, vagy legalábbis minimális külső beavatkozást igénylő identifikációs folyamat kidolgozása jelentősen kényelmesebbé tenné a rendszer használatát, ami egy esetleges kereskedelmi alkalmazásban kulcsfontosságú szempont. Ehhez azonban nélkülözhetetlen az elosztott rendszert alkotó mote-ok működésének valamilyen mértékű szinkronizációja, amely korántsem triviális feladat.

- **Távfelügyelet.** A már eleve Ethernet hálózatba kapcsolt mote-ok szabad számítási kapacitásának egy részét külső számítógépekről érkező kérések kiszolgálására dedikálva lehetségessé válna a rendszer állapotának lekérdezése. Sőt az automatikus identifikáció megvalósítása esetén lehetőség nyílna a zajcsökkentő rendszer teljes körű távfelügyeletére is, beleértve a távolról való indítást, újrakalibrálást.

- **Hibatűrő működés.** Jelen dolgozat keretében nem vizsgáltam a rendszer hibatűrését. Az elosztott FxLMS algoritmus jellegéből adódóan egy mote kiesése nem okozza a teljes rendszer működésének leállását, azonban a többi komponens nem képes detektálni a meghibásodást, a kieső mote együtthatómódosításának számítását továbbra is elvégzik, és meg is kísérlik továbbítani a rendszerből már hiányzó mote felé, ezzel is feleslegesen terhelve a kommunikációs hálózatot.

- **Más algoritmusok és alkalmazások vizsgálata.** Az aktív zajcsökkentés és az FxLMS algoritmus az elosztott jelfeldolgozó rendszereknek egy nagyon kézenfekvő, de speciális példája. Számos egyéb alkalmazásban is jogosan merülhet fel a jelfeldolgozási feladatok eloszthatóságának kérdése, így más algoritmusok elosztott implementációját is érdemes megvizsgálni – akár a munkám során elkészített mintarendszer felhasználásával.

Köszönetnyilvánítás

Köszönöm konzulensemnek, Dr. Orosz Györgynek, hogy rendszeresen és lelkiismeretesen megtartott konzultációkkal, valamint a konzultációs időpontokon kívül is hasznos tanácsokkal járult hozzá munkám sikerességéhez.

Köszönöm továbbá a Méréstechnika és Információs Rendszerek Tanszéknek a dolgozatom elkészítéséhez szükséges tárgyi erőforrások beszerzését és rendelkezésemre bocsátását.

Rövidítések jegyzéke

- **ADC:** Analog-to-Digital Converter
- **ANC:** Active Noise Cancellation (vagy Active Noise Control)
- **BGA:** Ball Grid Array
- **CMSIS:** Cortex Microcontroller Software Interface Standard
- **CPU:** Central Processing Unit
- **DAC:** Digital-to-Analog Converter
- **DFT:** Discrete Fourier Transform
- **DMA:** Direct Memory Access
- **DSP:** Digital Signal Processor (vagy Digital Signal Processing)
- **FFT:** Fast Fourier Transform
- **FIR:** Finite Impulse Response
- **FLOPS:** Floating Point Operations per Second
- **FPGA:** Field Programmable Gate Array
- **FPU:** Floating Point Unit
- **FxLMS:** Filtered-x Least Mean Squares
- **GPIO:** General Purpose Input/Output
- **GPS:** Global Positioning System
- **GSM:** Global System for Mobile Communications
- **HAL:** Hardware Abstraction Layer
- **I^2S:** Inter-IC Sound

- **IIR:** Infinite Impulse Response
- **IP:** Internet Protocol
- **LIDAR:** Light Detection and Ranging
- **LMS:** Least Mean Squares
- **LTI:** Linear Time-Invariant
- **MAC:** Multiply-Accumulate
- **MEMS:** Microelectromechanical System
- **MIMO:** Multiple Input, Multiple Output
- **MLMS:** Multiple Error Least Mean Squares (vagy Momentum Least Mean Squares)
- **MSPS:** Million Samples Per Second
- **UDP:** User Datagram Protocol
- **PC:** Personal Computer
- **RAM:** Random Access Memory
- **RISC:** Reduced Instruction Set Computer
- **RMII:** Reduced Media-Independent Interface
- **SDRAM:** Synchronous Dynamic Random Access Memory
- **SHARC:** Super Harvard Architecture Single-Chip Computer
- **SIMD:** Single Instruction, Multiple Data
- **SPDIF:** Sony/Philips Digital Interface Format
- **SoC:** System on a Chip
- **TCP:** Transmission Control Protocol
- **TDM:** Time-Division Multiplexing
- **TLS:** Transport Layer Security
- **VLIW:** Very Long Instruction Word

Ábrák jegyzéke

Irodalomjegyzék

[1] G. Coulouris, J Dollimore, T. Kindberg, G. Blair, *"Distributed Systems: Concepts and Design"*, 5th Edition, Addison-Wesley, Boston, 2011, ISBN 0-132-14301-1.

[2] `www.kurzweilai.net/googles-self-driving-car-gathers-nearly-1-gbsec`, hozzáférés: 2018. május 1.

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

[4] D. Miljković, *",Active Noise Control: From Analog to Digital – Last 80 Years"*, 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1358–1363, June 2016.

[5] Orosz Gy., *„Rezonátor alapú jelfeldolgozás szenzorhálózatokban"*, Doktori disszertáció, Budapesti Műszaki és Gazdaságtudományi Egyetem, 2012.

[6] C. Antoñanzas, M. Ferrer, M. de Diego, A. Gonzalez, *"Blockwise Frequency Domain Active Noise Controller Over Distributed Networks"*, Applied Sciences, 6 (5), 124, April 2016.

[7] J. Lorente, C. Antoñanzas, M. Ferrer, A. Gonzalez, *"Block-based distributed adaptive filter for active noiose control in a collaborative network"*, 23rd European Signal Processing Conference (EUSIPCO), pp. 310–314, Nice, France, 2015.

[8] B. Widrow, S. D. Stearns, *"Adaptive Signal Processing"*, Prentice Hall, 1985, ISBN 0-13-004029-0.

[9] S. S. Haykin, B. Widrow (ed.), *"Least-Mean-Square Adaptive Filters"*, Wiley, 2003, ISBN 0-471-21570-8.

[10] S. L. Netto, P. S. R. Diniz, P. Agathoklis, *"Adaptive IIR Filtering Algorithms for System Identification: A General Framework"*, IEEE Transactions on Education, vol. 38 no. 1, pp. 54–66, February 1995.

[11] L. Sujbert, G. Péceli, *"Periodic noise cancelation using resonator based controller"*, The International EAA Symposium on Active Control of Sound and Vibration, ACTIVE '97, pp. 905–916, Budapest, Hungary, August 1997.

[12] Sujbert L., *„Modellalapú jelfeldolgozás és aktív zajcsökkentés"*, habilitációs tézis, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 2016.

[13] L. Sujbert, *"A new filtered LMS algorithm for active noise control"*, The International EAA Symposium on Active Control of Sound and Vibration, ACTIVE '99, pp. 1101–1110, Fort Lauderdale, Florida, USA, December 1999.

[14] M. A. Tugay, Y. Tanik, *"Properties of the momentum LMS algorithm"*, Signal Processing, Volume 18, Issue 2, , pp. 117–127, October 1989.

[15] *"Intel x86-Based Digital Signal Processing"*, `http://www.rtcgroup.com/whitepapers/files/intelwp1.pdf`, hozzáférés: 2017. december 10.