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

Grafikusprocesszor-alapú aktív zajcsökkentő rendszer fejlesztése

DIPLOMATERV

Készítette
Gajdács András

Konzulens
dr. Sujbert László

2016. december 15.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Gajdács András*, szigorló hallgató kijelentem, hogy ezt a diplomatervet 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, 2016. december 15.

_____

Gajdács András
hallgató

Kivonat

Az akusztikus zajok elnyomására alkalmazott hangszigetelő anyagok rendszerint veszítenek hatékonyságukból az alacsony frekvenciájú tartományokban. Ez a hiányosság az aktív zajcsökkentő rendszerek (ANC) használatával hidalható át, amelyek az interferencia jelenségére építenek, és hangsugárzók segítségével keltett hanghullámok révén képesek kioltani a zajt egy meghatározott térrészben.

A sztochasztikus zajok hosszan zengő terekben történő elnyomására léteznek algoritmusok, amelyek viszont nagy fokszámú adaptív szűrők megvalósítását követelik meg, így a számításigényük eddig nem tette lehetővé a gyakorlati alkalmazásukat. Az általános célú programozásra alkalmas grafikus processzorok (GP-GPU) elterjedésével azonban a korábban elérhető hardverelemekhez mérve nagyságrendekkel nagyobb számítási teljesítmény vált széles körben hozzáférhetővé.

A valós idejű szabályozási körökben történő alkalmazás, mint amilyen az ANC-rendszer is, ugyanakkor nem evidens, hiszen a GPU jellemzően egy számítógépes környezetben található meg, emiatt pedig az operációs rendszer késleltetésével és annak ingadozásával is számolni kell. Ebből kifolyólag a grafikus processzor működését is figyelembe véve módosítanom kellett az alkalmazni kívánt zajcsökkentő algoritmuson, elkülönítve az időkritikus, kis számításigényű részt a kevésbé időkritikus, nagy számításigényű résztől.

A megvalósíthatósággal kapcsolatos kérdések tisztázása után megterveztem az ANC-rendszert, illetve a szükséges, PC-n kívüli hardverelemeket. A központi egység szerepét egy Zynq-7020 alapú fejlesztői kártya tölti be, amely az illesztési és kommunikációs feladatokon kívül időkritikus jelfeldolgozási feladatokat is végez. A zajcsökkentéshez szükséges nagyszámú referenciajel biztosítására MEMS-mikrofonokkal szerelt modulok felhasználását javasoltam, amelyek egy lineáris buszon keresztül csatlakoznak a rendszerhez. Emellett 8-8 analóg be- és kimeneti csatorna használatára van lehetőség.

Az áramköri megvalósítást és a tesztelést követően akusztikai méréseket végeztem a rendszer működőképességének igazolására. Hat referenciajel felhasználásával, 8192 együtthatós szűrők mellett az FeLMS-algoritmussal 19,5 dB-es zajelnyomást értem el. A teremben korábban egy másik ANC-rendszerrel végzett méréssorozat, valamint egy teremszimulációs programcsomag segítségével végzett ellenőrzés kimenetele egybevág a mostani eredményekkel, ami a helyes működésre enged következtetni.

Az elkészült rendszer bizonyítja, hogy a GPU alkalmas valós idejű ANC-rendszerekben történő felhasználásra, azonban a benne rejlő potenciál kiaknázásához kifinomultabb algoritmusok implementálása szükséges, amelyek így várhatóan nagyobb konvergenciasebességet és zajelnyomást eredményeznek.

Abstract

Soundproofing materials used for damping acoustical noises tend to lose their efficacy in the low frequency range. This drawback can be aided by utilizing active noise control (ANC) systems that rely on the phenomenon of destructive interference, thus are capable of cancelling noise in a certain spatial domain by emitting anti-noise via loudspeakers.

Even though algorithms have been known for cancelling stochastic noise in highly reverberant spaces, the computational demand emerging from operating high-order adaptive filters made them impractical for real-life applications in the past. General purpose programming of graphics processing units (GP-GPU) has recently gained popularity and as a result, computational capacity of more than a magnitude higher than previously applied devices provided became widely available.

Utilization of GPUs in real-time control loops (e.g. an ANC system) requires careful design as the GPU generally resides in a personal computer, therefore the delays and jitter introduced by the operating system cannot be neglected. Hence, I had to modify the noise cancelling algorithm in a way that it can take the formerly mentioned fact into account and also fits to the architecture of the graphics processing unit. This led to separation of the task into a time-critical part with low computational cost and a less time-critical part with high computational cost.

After addressing the questions related to feasibility, I designed the ANC system including the necessary external hardware components. The central unit of the system is based on a Zynq-7020 System-on-Chip equipped developer board which forms a bridge between the PC and the other external components and handles communication-related and signal processing tasks. ANC requires large number of reference signals which can be efficiently addressed by means of MEMS microphones mounted on modules that connect to the system via a linear bus. Besides these, the system also has 8-8 analog inputs and outputs.

Following electronic circuit realization and testing, I conducted acoustical measurements in order to prove the functionality of the system in a real-life application. Using 6 reference signals and 8192-tap filters, the system was able the achive 19.5 dB of noise reduction with the FeLMS algorithm. The results are consistent with the outcome of a former experiment that took place in the same room but utilized a different ANC system. I also had a chance to verify the performance of the built system with a simulation based on acoustic field modeling. Both of these serve as indirect proofs of proper operation.

The built system proves that GPUs are suitable for using in real-time ANC systems but in order to exploit all of its benefits, more sophisticated algorithms have to be implemented which are expected to yield faster convergence as well as higher noise suppression.

Bevezető

A dolgozat címe, bár nem hosszú, összetettsége révén mégis temérdek információt hordoz magában: *grafikusprocesszor-alapú aktív zajcsökkentő rendszer tervezése*. Érdemes ezt egy kicsit alaposabban is szemügyre venni, miáltal majd remélhetőleg körvonalazódik, hogy miről is lesz szó a továbbiakban.

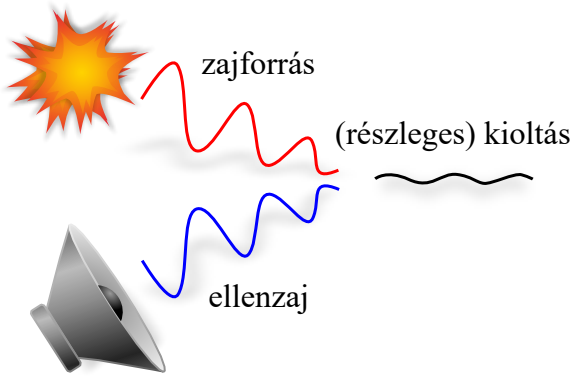
Először is tisztáznunk kell, hogy mi az a *zaj*, amelyet csökkenteni szeretnénk. Általában a zaj szó hallatán – hacsak nincs mérnöki képzettségünk – az akusztikai értelemben vett zajokra asszociálunk. Nehéz pontosan megfogalmazni, hogy mit is értünk akusztikus zaj alatt, amelyet jól mutat az is, hogy a szakirodalomban fellelhető definíciók nem egységesek. Egy ponton azonban minden szerző egyetért, nevezetesen, hogy az akusztikus zaj nemkívánatos hangjelenségek összessége. Itt máris egy szubjektív tényező merül fel, ugyanis a környezettől és a szituációtól is függ, hogy egy adott hangjelenséget „nemkívánatosként", vagyis zajként érzékelünk-e. Például egy klubban a hangos zene nem zaj az ott szórakozók számára, azonban a két emelettel fentebb aludni készülő egyén a beszűrődő hangokat minden bizonnyal zajként érzékeli. Persze léteznek olyan hanghatások is, amelyek szinte minden esetben nemkívánatosnak bizonyulnak. A teljesség igénye nélkül következzék néhány példa [1] aszerint csoportosítva, hogy a mindennapok során mely területeken szembesülhetünk velük:

- *Autóipari:* kipufogó-, motor- és menetzaj
- *Háztartási:* légkondicionáló berendezés, hűtőgép, konyhai elszívó, mosógép, fűnyíró, porszívó zaja; szobák közötti áthallás
- *Ipari:* forgógépek (ventilátorok, áramfejlesztők, kompresszorok, szivattyúk), transzformátorok zaja, egyterű irodákban a munkaállomások közötti áthallás
- *Utasszállítás:* autóbuszok, repülőgépek, helikopterek, dízelmozdonyok, hajók zaja

A zajokat csoportosíthatjuk a jelalakjuk periodicitása alapján is, így beszélhetünk *periodikus*, illetve *sztochasztikus* zajokról. Az említett példák közül rendszerint a forgógéppel rendelkező berendezések, motorok, transzformátorok keltenek periodikus vagy kváziperiodikus zajokat. A szakirodalom ezeket kissé pongyolán *„keskeny sávú" (narrowband)* zajoknak nevezi [1], amely azt hivatott kifejezni, hogy a teljesítményspektrumuk vonalas, vagyis az alapharmonikuson és a felharmonikusokon kívül ezeknek a jeleknek nincs más összetevője. Természetesen ez nem jelenti azt, hogy egy periodikus zaj nem lehet a klasszikus értelemben széles sávú. A sztochasztikus zajok teljesítményspektruma ezzel szemben tetszőleges lehet, és ezeket nevezzük *„széles sávú" (broadband)* zajoknak. A fenti példák közül a légkondicionáló berendezés vagy a porszívó zaját sorolhatjuk ide.

Bármilyen akusztikus zajról is beszélünk, az egészségre gyakorolt negatív hatásai vitathatatlanok. A tartósan fennálló, 80 dB_A feletti zajterhelés hallászcsökkenéshez és fülzúgáshoz vezethet, ugyanakkor az ennél jóval kisebb teljesítményű környezeti zajok is okozhatnak hosszú távon nem hallószervi panaszokat. A legfrissebb kutatások [2] szerint ingerlékenység, alvászavarok, magasvérnyomás-betegség, valamint szív- és érrendszeri betegségek írhatók a zajterhelés számlájára, továbbá a kognitív képességek (tanulás, memória) romlását is megfigyelték. Ez tehát kellő motivációt szolgáltat ahhoz, hogy foglalkozzunk a bennünket érő zajterhelés minimalizálásával, vagyis a *zajcsökkentéssel*.

A mechanizmust tekintve beszélhetünk *aktív* és *passzív* zajcsökkentésről. Passzív zajcsökkentés esetén hangelnyelő tulajdonsággal bíró anyagokkal (például üveggyapottal) igyekszünk elválasztani a zajforrást a környezetétől [3, 4]. Ez a megoldás széles frekvenciatartományban jó elnyomást biztosít, viszont hátránya, hogy sok helyet foglal, sokszor fizikailag nem is telepíthető az alkalmazás helyén, illetve a néhány száz Hertz alatti frekvenciájú zajok esetén elveszti hatékonyságát. Ezekben az esetekben jöhet szóba az aktív zajcsökkentő rendszerek (a továbbiakban ANC-rendszerek) alkalmazása, amelyek a destruktív interferencia révén működnek. Ez azt jelenti, hogy amennyiben egy hangsugárzóval olyan hangot keltünk, amely a tér egy pontján a zajjal azonos amplitúdóval, ellentétes fázisban találkozik, akkor ott elvben tökéletes kioltás jöhet létre. A megoldás hátránya a

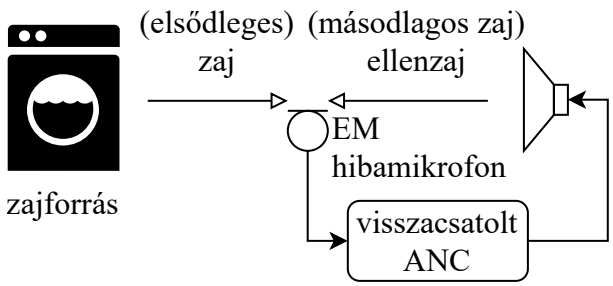


1. ábra: *Az aktív zajcsökkentés alapgondolata [5]*

leírásból következik, hiszen az interferencia révén – néhány speciális esettől eltekintve – egy hangsugárzóval elméletileg is csak a tér egyetlen pontjában tudunk tökéletes kioltást elérni. Az ANC-rendszerek használata azonban különösen kis frekvenciájú zajok esetén indokolt, amelyek hullámhossza a méteres nagyságrendbe esik, ebből eredően pedig a teljes kioltás helyének egy környezetében még mindig számottevő elnyomás érhető el.

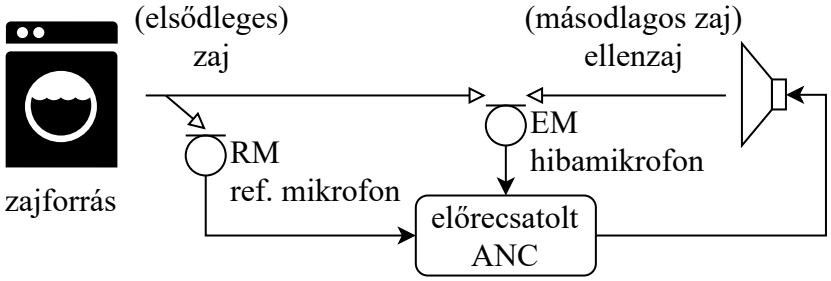
Egy ANC-rendszer nem más, mint egy szabályozási kör. Mint az ismeretes, egy szabályozási körnek tartalmaznia kell egy érzékelőt, amely esetünkben egy mikrofon lesz, illetve egy beavatkozó szervet, amelynek szerepét most a hangsugárzó tölti be. A legegyszerűbb, *visszacsatolt* ANC-rendszert a 2. ábra szemlélteti [6].

A cél az, hogy az EM hibamikrofon helyén teljes kioltás jöjjön létre a zajforrásból



2. ábra: *Visszacsatolt struktúra*

érkező elsődleges, illetve a hangsugárzóból érkező másodlagos zaj interferenciája révén. A struktúrát alaposabban szemügyre véve észrevehető, hogy a visszacsatolt zajcsökkentő rendszerek használhatósága korlátozott, ugyanis a hangsugárzó hibamikrofontól való távolsága miatt fellépő τ_s késleltetéssel, továbbá a τ_p feldolgozási idővel hamarabb mintát kellene vennünk a zajból ahhoz, hogy egyszerre érkezhessen meg a hibamikrofonhoz a zaj és az ellenzaj. Az, hogy ugyanott szeretnénk elérni kioltást, mint ahol mintát veszünk a zajból, természetesen egy akauzális rendszert eredményez általános, sztochasztikus jelek esetén. Periodikus jelek mellett azonban működőképes a megoldás, hiszen az előző periódusok alapján jósolhatók a jövőbeli értékek, és így időben kiadható a szükséges ellenzaj.



3. ábra: *Előrecsatolt struktúra*

Ha sztochasztikus zajok elnyomása (is) a célunk, abban az esetben a 3. ábrán látható *előrecsatolt* struktúrát alkalmazhatjuk. A visszacsatolt verzióhoz képest ez a rendszer tartalmaz még egy RM referenciamikrofont is, amely egy referenciajelet szolgáltat az elsődleges zajból legalább $\tau_p + \tau_s$ idővel hamarabb, mielőtt az az EM-hez megérkezne, ezáltal biztosítva a kauzalitást. A dolgozatomban is egy ilyen, előrecsatolt struktúrán alapuló zajcsökkentő rendszer megvalósítását tűztem ki célul. Ennek oka, hogy egy, a sztochasztikus zajok elnyomására alkalmas megoldás a periodikus zajok elnyomására is alkalmas, ezáltal általánosabb, másfelől pedig a jelenlegi állás szerint nincs igazán kiforrott megoldás a sztochasztikus zajok hosszan zengő terekben történő elnyomására [7, 8].

A digitális jelfeldolgozást alkalmazó előrecsatolt zajcsökkentő rendszerek általában *adaptív szűrőn* alapulnak [9, 10], amely a referenciajelből gyakorlatilag FIR-szűréssel állítja elő a kiadandó ellenzajt, miközben a szűrőegyütthatók folyamatos hangoláson esnek át annak érdekében, hogy a hibajel – valamilyen normában – a lehető legkisebb

legyen. A szükséges szűrőegyütthatók száma függ a mintavételi frekvenciától, valamint az akusztikai térre jellemző impulzusválasz hosszától. Például egy nem túl magas, 10 kHz-es mintavételi frekvencia mellett, és egy közepesnek mondható, 1-2 másodperces zengési idejű terem esetén már 10-20 ezer együtthatóval kell számolnunk. Ha figyelembe vesszük, hogy ezt a szűrőt valós időben kell tudnunk futtatni, illetve MIMO (több be- és kimenetű) rendszer esetén több példányban is, rádöbbenhetünk, hogy ez akár több GFLOPS (milliárd lebegőpontos utasítás másodpercenként) számítási teljesítményt követel meg a rendszert futtató hardvertől.

Ennek az igénynek a kielégítésére számos lehetőség kínálkozik, ugyanis a napjainkban kereskedelmi forgalomban kapható FPGA-k, DSP-k, általános célú felhasználást lehetővé tevő GPU-k, illetve vektorutasítások (SSE, AVX) használata és többszálú programkód mellett még a CPU-k is rendelkeznek az említett nagyságrendbe eső számítási teljesítménnyel. Ezeken belül a GPU-k névleges számítási teljesítménye akár 10 TFLOPS is lehet [11], viszont ez csak akkor aknázható ki hatékonyan, ha a futtatott kód bizonyos, később ismertetett követelményeknek megfelel. Szerencsére a szóban forgó architektúrákban közös, hogy a FIR-szűrés mindegyiküknél „natív” feladat, vagyis hatékonyan végrehajtható. Tudván, hogy az általános célú programozásra alkalmas GPU-k széleskörűen elterjedtek, ma már szinte minden PC-ben fellelhetők, kimagasló ár-teljesítmény arányuk és flexibilitásuk folytán vonzó alternatívát kínálnak a többi architektúrával szemben, kiváltképp akkor, ha egy jól skálázható, bővíthető ANC-rendszer megépítése a cél.

Gyakorlati szemszögből nézve azonban számos megoldandó feladat tárul elénk. A GPU-k általában egy PC részét képezik, amelyeken többnyire nem valós idejű operációs rendszer fut, holott az ANC egy valós idejű szabályozási körön alapul. A kereskedelmi forgalomban kapható audio interfészekkel elérhető legkisebb késleltetés legjobb esetben is 3 ezredmásodperc körül várható [12], később részletezett okokból kifolyólag viszont számunkra 1 ezredmásodpercnél kisebb válaszidő lenne kedvező, amely már a τ_p számításokra fordított időt is tartalmazza. A munkám célja tehát egy olyan rendszer tervezése, megvalósítása és tesztelése, amely alkalmas egy kis késleltetésű, sokcsatornás ANC-rendszer üzemeltetésére, kihasználva a GPU-k hatalmas számítási kapacitásában rejlő lehetőségeket.

Az első fejezetben ismertetem az adaptív előrecsatolt ANC-rendszerek alapkoncepcióját, illetve a leggyakrabban alkalmazott, LMS alapú adaptív szűrők elméleti hátterét. Ezután kitérek a másodlagos út átvitelének kompenzálási lehetőségeire, amely a gyakorlati alkalmazás szempontjából bír jelentőséggel. Ez szintén elmondható a szűrők konvergenciasebességéről, ezért ennek gyorsítására is ismertetek egy lehetséges eljárást. Az adaptív rendszerek több be- és kimenetet is kezelni képes változatának bemutatása után szimulációk segítségével vizsgálom meg a rendszer különböző paramétereinek hatását az elérhető zajelnyomásra és a konvergencia sebességére. Végül pedig az egy és a több referenciajelet felhasználó rendszerek teljesítménye közötti ugrásszerű különbségre keresek magyarázatot.

A második fejezet a grafikus processzorra történő programozásból kíván ízelítőt nyújtani, felfedve a hasonlóságokat és a különbségeket a különböző adat- vagy utasításpárhuzamosságot felvonultató architektúrákkal. A hatékony programkód elkészítéséhez kulcsfontosságú a GPU belső felépítésének alapvető ismerete. Ennek birtokában pedig bemutatom, hogyan lehetséges az FeLMS-algoritmus megvalósítása GPU-n, illetve az implementáció milyen átalakításokat tesz szükségessé.

A megvalósíthatóságról való meggyőződést követően a zajcsökkentő rendszer tervezése következik, amellyel a harmadik fejezet foglalkozik. Először a rendszerrel szemben támasztott követelményeket tisztázom, és felvetem a sok referenciajel igényének kielégítésére a MEMS-mikrofonok használatát. A grafikus processzor PC-s környezetével kapcsolatos kérdések szintén ebben a fejezetben kerülnek megválaszolásra. A kommunikációs és időkritikus jelfeldolgozási feladatokat végző központi elem kiválasztását követően bemutatom a digitális kimenetű MEMS-mikrofonokkal kapcsolatos jelfeldolgozási teendőket, majd a számításigényt figyelembe véve mikrokontrollert választok a feladatok ellátására. A mikrofonos modulok kommunikációjához egy lineáris buszrendszert tervezek, kitérve a hozzá kapcsolódó protokollra és a kínálkozó szinkronizációs lehetőségekre. Végül pedig a központi egységhez kapcsolódó kiegészítő panel tervét mutatom be, amely analóg be- és kimeneti csatornák kezelését teszi lehetővé, és a mikrofonos modulok jelét is fogadja.

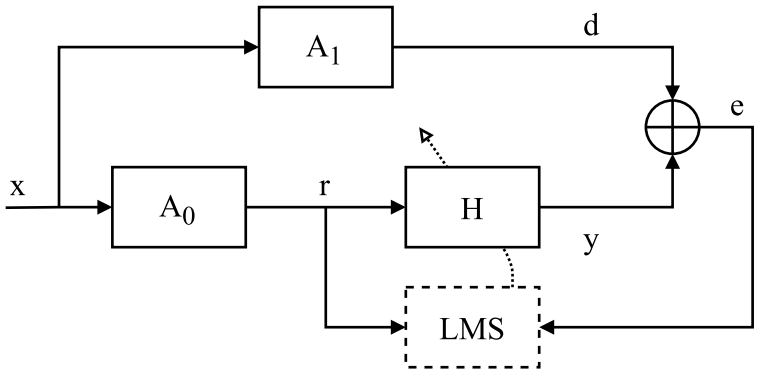
A tervezési fázis lezárása után az áramköri megvalósítással és a firmware-fejlesztéssel kapcsolatos gyakorlati kérdések kerülnek előtérbe a negyedik fejezetben. Bemutatom a MEMS-mikrofonos modulok és a kiegészítő panel realizációjának lépéseit, és az adatfolyamokba ékelődő szűrők implementációjával is foglalkozom. A mikrofonos moduloknál a firmware, a kiegészítő panelnél pedig az illesztést végző FPGA-modulok készítéséről számolok be. Az áramkörök élesztésekor és tesztelésekor szerzett tapasztalatok megosztása után a központi egységre fejlesztett alkalmazást ecsetelem, kitérve a kommunikációs és jelfeldolgozási feladatok processzormagok közötti szétosztására. A rendszer működtetéséhez C++ nyelven egy PC-s alkalmazást is fejlesztenem kellett, ennek birtokában láthattam hozzá a komponensek együttes élesztéséhez és teszteléséhez.

A kész rendszerrel ezután akusztikai méréseket is végeztem a működőképesség igazolásához, amelyről az ötödik fejezet számol be. A mérési elrendezés ismertetését követően értékelem a mérés során elért eredményeket, majd összevetem egy másik ANC-rendszer ugyanazon konfigurációban felmutatott teljesítményével. Befejezésképp egy teremszimulációs programcsomag lehetőségeit kihasználva szimulációval is alátámasztom az épített rendszer megfelelő működését.

1 Adaptív, előrecsatolt zajcsökkentő rendszerek

1.1 Alapkoncepció

Az adaptív, előrecsatolt zajcsökkentő rendszerek alapmodelljét [9, 10] az 1.1-es ábra hivatott szemléltetni. Az x bemenet egy hipotetikus jelet reprezentál, amelyet a pontsze-



1.1. ábra: *Adaptív, előrecsatolt zajcsökkentő rendszer elvi blokkvázlata*

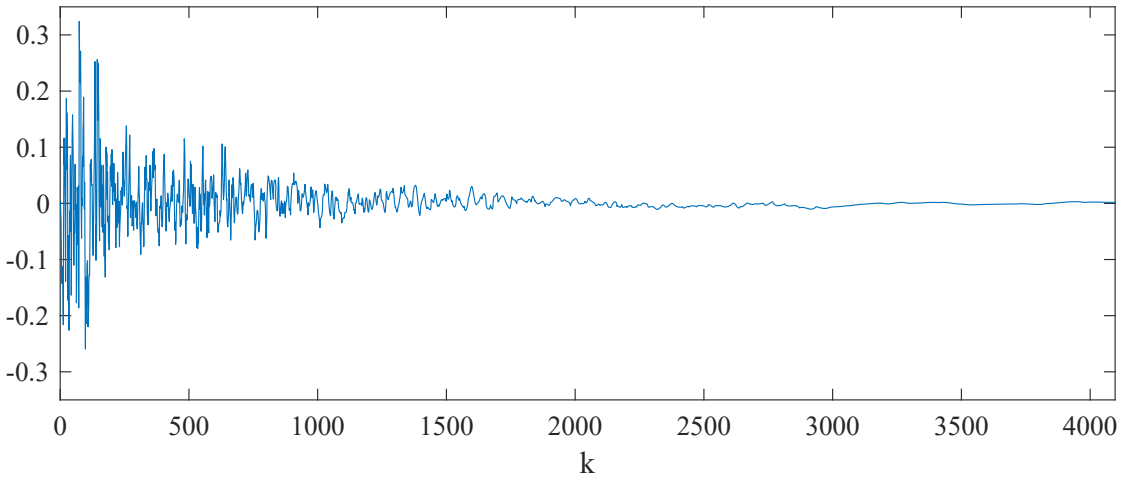
rűnek tekintett zajforrás helyén mérhetnénk a környezeti hatások (reflexiók) nélkül. Az r referenciajel, amelyet valójában mérni tudunk, az x bemenet A_0-lal szűrt változata. Az A_0 tartalmazza a zajforrástól az RM-ig terjedő akusztikus átvitelt, valamint a mikrofon átviteli karakterisztikáját. Az utóbbihoz hozzátartozik a jelútban lévő analóg (erősítő, szűrő, AD-átalakító) és digitális (buffer, decimáló szűrő) elemek átvitele is. A d a kioltandó zajt jelöli, amely egy másik akusztikus átvitelt és az EM karakterisztikáját magában foglaló A_1 segítségével fejezhető ki az x bemenetből. A H egy adaptív szűrőt takar, amelynek kimenete a (jelenleg ideálisnak tekintett) beavatkozó szervet, azaz a hangsugárzót hajtja meg. A d kioltandó zaj, valamint a hangsugárzóból érkező ellenzaj eredőjét az EM méri, az általa szolgáltatott e hibajel alkalmas az adaptív szűrő hangolására.

A működés során a cél az, hogy az e hibajel várható értéke minimális legyen, az egyszerűség kedvéért négyzetes normában. Könnyen belátható, hogy e feltétel mellett az adaptív szűrő optimális megoldására a

$$H = -\frac{A_1}{A_0} \tag{1.1}$$

adódik. Mivel A_0 inverze szerepel ebben a kifejezésben, és egy akusztikus átviteli függvény csak a legritkább esetekben minimálfázisú [13], az inverz csak akkor lesz jól közelíthető, ha A_1 késleltetése nagyobb A_0 késleltetésénél. A késleltetés itt azt jelenti, hogy a zajforrásból kiadott impulzus mikor érkezik meg leghamarabb (vagyis közvetlen úton, légvonalban) az adott mikrofonhoz. Ha tehát kellően nagy a késleltetésbeli különbség A_1 javára, akkor a H

közelíteni fogja A_0 ún. *késleltetett inverzét*. Az A_0 akauzális inverze ugyanis rendelkezne nemnulla értékű negatív indexű mintákkal is, a késleltetés hatására pedig ezek a negatív indexű minták pozitív irányba tolódnak, és így reprezentálhatóvá válnak. A közelítés akkor lesz jó, ha a késleltetett inverz negatív irányban is lecseng, vagyis a levágásra kerülő negatív indexű mintákban hordozott teljesítmény csekély.

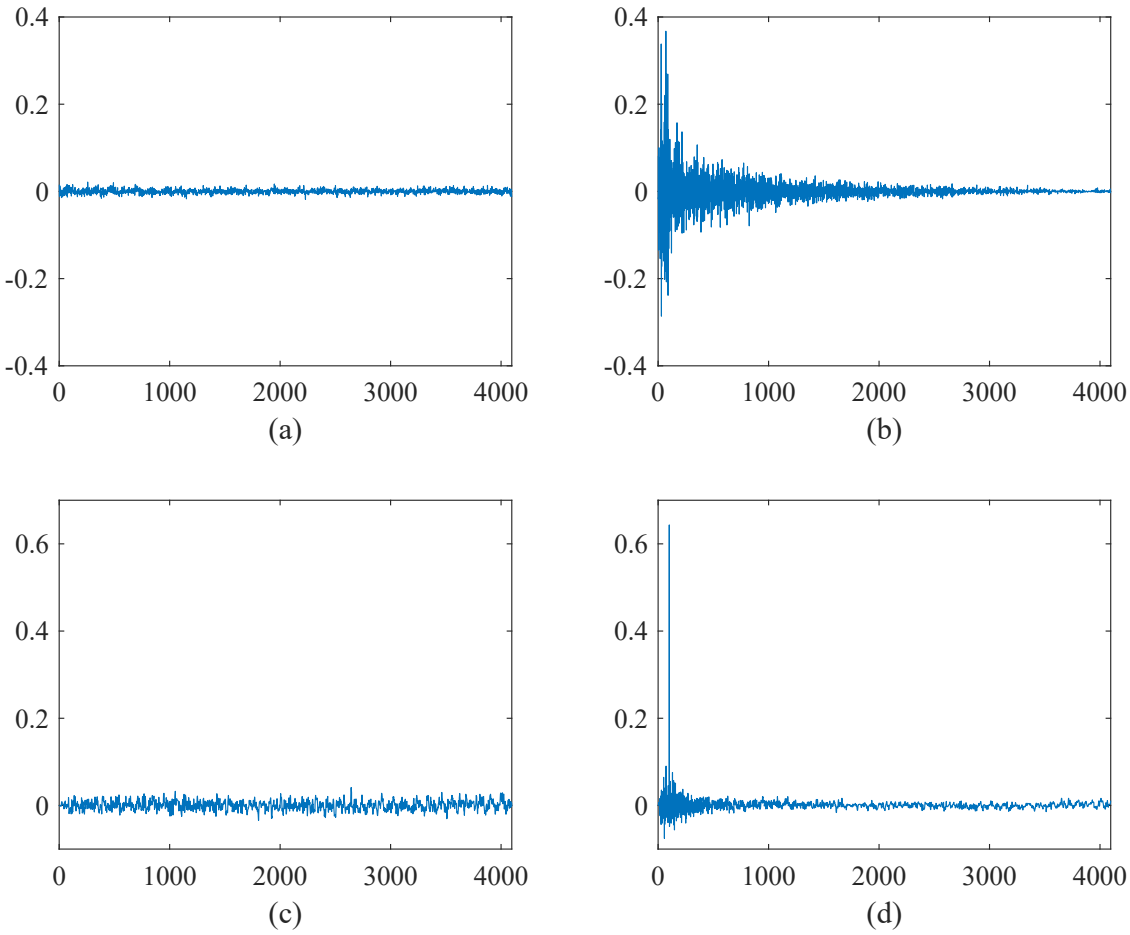


1.2. ábra: *Példa egy teremben mérhető impulzusválaszra*

Az 1.2-es ábrán egy valóságos akusztikai térben mért, 16 kHz-en mintavett impulzusválasz látható, az 1.3-as ábra *(a)* és *(b)* részében pedig a késleltetett inverzek közelítései figyelhetők meg, rendre 0 és 100 mintányi ($\sim$2 méter távolságnak megfelelő) késleltetés mellett. A *(c)* és a *(d)* részben a késleltetett inverzek és az eredeti impulzusválasz konvolúciója alapján következtethetünk a közelítés jóságára, mivel ideális esetben itt egy (késleltetett) egységimpulzusnak kellene kirajzolódnia. Az *(a)* és *(c)* részek alapján kijelenthetjük, hogy a késleltetés nélküli esetben elégtelen a közelítés, 100 mintányi késleltetés mellett azonban már egészen elfogadható az approximáció, jóllehet, az ábra *(b)* részén még nem látható a korábban említett negatív irányú lecsengés, vagyis nagyobb késleltetés mellett ennél még sokkal jobb eredményre is juthatnánk.

A fentiekből kiderült, hogy a kauzalitás biztosítása szükséges, de nem elégséges feltétele a rendszer megfelelő működésének. Elviekben annál jobb elnyomást érhetünk el, minél közelebb helyezzük az RM-et a zajforráshoz az EM-hez képest. Ugyanakkor az is világossá vált, hogy miért kell igyekeznünk minimalizálni a zajcsökkentő rendszeren belül fellépő késleltetéseket, hiszen ha nem tudjuk tetszőlegesen közel helyezni az RM-et a zajforráshoz, a gyorsabb feldolgozással ugyanúgy időt nyerhetünk, miáltal javul az invertálhatóság és az elérhető zajelnyomás.

Nem szabad megfeledkeznünk a véges hosszúságú adaptív szűrő együtthatószámáról sem, amely hasonló jelentőséggel bír. Az 1.2-es ábrán egy nagyon rövid, nagyjából negyed másodperc alatt lecsengő impulzusválaszt láthattunk, és már ehhez is több mint 4 ezer együtthatóra volt szükségünk 16 kHz-es mintavételi frekvencia mellett. Egy nagyobb,

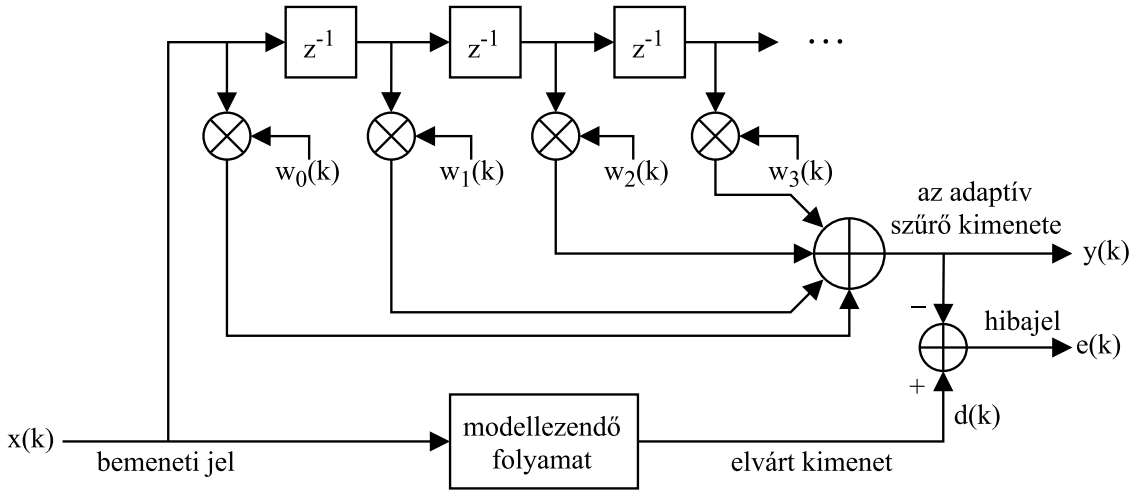


1.3. ábra: *A késleltetett inverz közelítése (a) 0 és (b) 100 mintányi késleltetés mellett, illetve konvolúciójuk az eredeti impulzusválasszal*

hosszan zengő teremben ugyanakkor több másodperces impulzusválaszra is számíthatunk, ami akár több tízezer együtthatós szűrők működtetését is megkövetelheti. Ha a szükségesnél rövidebb szűrőt alkalmazunk, akkor a rendszer a közelítendő impulzusválasznak csak az elejét lesz képes adaptálni, és az elhagyott farokrészhez tartozó jelkomponens úgy jelenik meg a hibajelben, mintha a referenciajellel nem korrelált, független zajforrásból származó zaj lenne.

1.2 LMS alapú adaptív szűrők

Az adaptív szűrők egyik legelterjedtebb realizációja az *adaptív lineáris kombinátor* [10]. Az elnevezés onnan ered, hogy a bemenetére adott értékek egy késleltetővonalra kerülnek, a kimenete pedig a megcsapolások súlyozott összegeként (lineárkombinációjaként) áll elő, amint az az 1.4-es ábrán is nyomon követhető. A működését tekintve tehát egy FIR-szűrő, amely azonban a szűrőegyütthatók (súlytényezők) megfelelő hangolása révén alkalmas lineáris folyamatok modellezésére, vagyis adaptációra.



1.4. ábra: *Adaptív lineáris kombinátor*

A matematikai háttér ismertetése megköveteli néhány jelölés bevezetését. A késleltetővonal megcsapolásain a k-adik diszkrét ütemben az

$$\mathbf{x}(k) = [x_0(k) \;\; x_1(k) \;\; \ldots \;\; x_{N-1}(k)]^T \tag{1.2}$$

értékek vannak, ahol az elemek alsó indexe a késleltetés mértékét jelöli mintákban kifejezve, az N pedig az adaptív szűrő hossza. A szűrőegyütthatók vektora az adaptáció miatt változik, így a k-adik ütemben

$$\mathbf{w}(k) = [w_0(k) \;\; w_1(k) \;\; \ldots \;\; w_{N-1}(k)]^T . \tag{1.3}$$

A szűrő kimenete pedig az

$$y(k) = \mathbf{x}^T(k)\mathbf{w}(k) \tag{1.4}$$

összefüggés szerint alakul. A hibajel a k-adik ütemben

$$e(k) = d(k) - y(k) = d(k) - \mathbf{x}^T(k)\mathbf{w}(k), \tag{1.5}$$

amelyből származtatható egy négyzetes kritériumfüggvény ($E\{\}$ a várhatóérték-operátor; a k-tól való függést pedig a továbbiakban nem jelölve):

$$\epsilon = E\{e^2\} = E\{d^2\} - 2\mathbf{w}^T E\{d\mathbf{x}\} + \mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w} \tag{1.6}$$

Bevezetve az $\mathbf{p} = E\{d\mathbf{x}\}$ és $\mathbf{R} = E\{\mathbf{x}\mathbf{x}^T\}$ jelöléseket, az alábbi forma adódik:

$$\epsilon = E\{d^2\} - 2\mathbf{w}^T\mathbf{p} + \mathbf{w}^T\mathbf{R}\mathbf{w} \tag{1.7}$$

A $\mathbf{p}$ a bemeneti jel és az elvárt kimenet közötti keresztkorrelációs vektor, az $\mathbf{R}$ pedig a bemeneti jel autokorrelációs mátrixa, amely egy szimmetrikus, pozitív szemidefinit

mátrix. Utóbbiból adódik, hogy a hibafelület egy paraboloid lesz, és az adaptáció során a szűrőegyütthatókat úgy kell hangolni, hogy a kritériumfüggvény értéke az $\epsilon = 0$ síkhoz a lehető legközelebb kerüljön. Ez tehát egy szélsőérték-feladat, amelynek megoldásához ki kell fejeznünk a gradiensvektort, majd egyenlővé kell tennünk a nullvektorral:

$$\nabla \triangleq \left[\frac{\partial \epsilon}{\partial w_0} \quad \frac{\partial \epsilon}{\partial w_1} \quad \dots \quad \frac{\partial \epsilon}{\partial w_{N-1}} \right]^T = -2\mathbf{p} + 2\mathbf{R}\mathbf{w} = \mathbf{0} \tag{1.8}$$

Ha $\mathbf{R}$ pozitív definit és invertálható, akkor az így kapott egyenletből kifejezhető az optimális megoldás, amely egyben a *Wiener-Hopf egyenlet* mátrixokkal felírt alakja[1]:

$$\mathbf{w}^* = \mathbf{R}^{-1}\mathbf{p} \tag{1.9}$$

Az (1.8)-as és az (1.9)-es egyenletek alapján a megoldást kifejezhetjük az $\mathbf{R}$ inverzének és a gradiensnek a segítségével is, amely bármely $\mathbf{w}$ kezdőállapotból egy lépésben az optimális megoldáshoz vezet:

$$\mathbf{w}^* = \mathbf{w} - \frac{1}{2}\mathbf{R}^{-1}\nabla \tag{1.10}$$

Egy gyakorlatban is jól használható eljáráshoz juthatunk az (1.10)-es összefüggés további elemzésével. Feltűnhet ugyanis, hogy a kifejezés hasonlít a *legmeredekebb lejtők módszerére* (*gradient descent*), mindössze annyi a különbség, hogy itt az $\mathbf{R}$ inverze is szerepel, amely egy lineáris transzformációként hat a gradiensre. Valóban erről van szó, ugyanis a hibafelületetet az $\mathbf{R}$ határozza meg [10], amelynek a sajátvektorai ortogonális bázist alkotnak $\mathbb{R}^N$-ben, lévén, hogy egy szimmetrikus mátrix. Ezek a sajátvektorok jelölik ki a paraboloid hibafelület főtengelyeinek irányait, amelyek mentén egy-egy szűrőegyüttható változtatásával mozoghatunk. Az $\mathbf{R}^{-1}$ tehát a sajátértékei szerinti skálázást hajt végre a kiindulási pontra jellemző gradiensen a paraboloid főtengelyei mentén, ezáltal mindegyik változó szerint minimalizálva a hibát, amelynek eredményeképpen egy lépésben eljutunk az optimális megoldáshoz.

Sajnos a gyakorlatban általában nem áll rendelkezésünkre az $\mathbf{R}$, legfeljebb becsülni tudjuk. A mátrix invertálása azonban így is számításigényes feladat, ezért alternatívaként alkalmazhatjuk közvetlenül is a legmeredekebb lejtők módszerét az (1.10)-es kifejezésből kiindulva, amellyel egy iteratív eljáráshoz jutunk.

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu\nabla(k) \tag{1.11}$$

A konvergencia feltétele, hogy a μ bátorsági tényező kisebb legyen mint az $\mathbf{R}$ legnagyobb sajátértékének a reciproka, de a túl kis érték nagyon lassú beállást eredményezhet, emiatt az optimális bátorsági tényező meghatározása egy alkalmazástól függő, gyakorlati probléma.

[1] Ha $\mathbf{R}$ nem invertálható, akkor nincs egyértelmű megoldás, viszont a feladat továbbra is megoldható marad a pszeudoinverz segítségével.

A gradiens kiszámításához továbbra is szükségünk lenne **R**-re és **p**-re, ezért egy közelítéssel kell élnünk. Az (1.8)-as összefüggésből az **R**-t és **p**-t visszahelyettesítve, majd a várhatóérték-képzéseket elhagyva eljuthatunk az ún. *LMS-módszerhez* (*Least Mean Square*).

$$\nabla = -2\mathbf{p} + 2\mathbf{R}\mathbf{w} = -2E\{d\mathbf{x}\} + 2E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w} \approx -2d\mathbf{x} + 2\mathbf{x}\mathbf{x}^T\mathbf{w} = -2e\mathbf{x} \tag{1.12}$$

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu e(k)\mathbf{x}(k) \tag{1.13}$$

A várhatóérték-operátor elhagyása meglehetősen pontatlan közelítésnek tűnhet, azonban nem szabad elfelejteni, hogy ergodikus folyamatokat feltételezve a várhatóérték-képzés kiváltható időbeli átlagolással kellően kis μ használata mellett. Az LMS-módszer kedvező számításigénye és egyszerű implementálhatósága révén széles körben elterjedt.

Amennyiben az adaptív szűrő bemeneti jele több nagyságrenden belül ingadozhat a működés során, érdemes az **x**-szel normálni a korrekciós tagot, különben a kis jelszintek esetén túl lassú lehet a konvergencia, vagy éppen nagy jelszinteknél divergenssé válhat a rendszer [14]. Az így kapott eljárás az *NLMS-módszer* (*Normalized LMS*):

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \frac{\mu}{a + \mathbf{x}^T(k)\mathbf{x}(k)} e(k)\mathbf{x}(k) \tag{1.14}$$

Az a regularizáló konstans a túlzottan alacsony bemeneti jelszinteknél játszik szerepet, ugyanis megakadályozza, hogy végtelenné váljon az effektív bátorsági tényező.

Fontos megemlíteni, hogy az akusztikai alkalmazás során a hibajel az (1.5)-ös egyenlettel ellentétben az $e = d + y$ összefüggés szerint alakul, hiszen a beavatkozó hangsugárzó által kiadott hang csak összeadódni tud a zajjal. Emiatt ilyenkor az (1.13)-as és az (1.14)-es összefüggéseknél a korrekciós tagot negatív előjellel kell szerepeltetni.

1.3 A másodlagos út átvitelének kompenzálása

Az 1.1-es ábrán bemutatott blokkvázlat kiegészítésre szorul, mert nem tartalmazza a másodlagos út átvitelét. Az adaptív szűrő kimenete és a hibajel közé ugyanis beékelődik még a DA-átalakító és a hozzá tartozó interpoláló szűrő, a hangsugárzó és az azt meghajtó erősítő, az EM és a jelét feldolgozó áramköri elemek, valamint a hangsugárzó és az EM közötti akusztikai átvitel. Mindez összefogható egyetlen A_2 jelű blokkba. Világos, hogy ebben az esetben az adaptív szűrőnek a

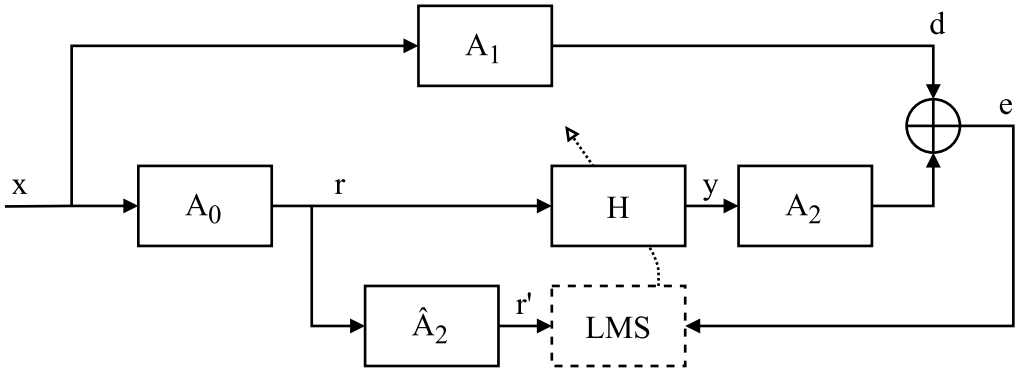
$$H = -\frac{A_1}{A_0 A_2} \tag{1.15}$$

átvitelt kell közelítenie, vagyis megjelenik a másodlagos út inverze is a kifejezésben. Ha az A_2 dinamikája nem túl nagy, akkor az inverz közelítése nem jelent majd különösebb problémát az A_0 inverzének közelítéséhez képest. Viszont az A_2 által okozott késleltetés a

szabályozási körben a frekvenciával egyenesen arányos fázistolást is eredményez, amely ha valamilyen frekvencián eléri a 180°-ot, ott pozitívvá teszi a visszacsatolást, ezáltal pedig instabillá válik a szabályozási kör [9]. Ez nem engedhető meg, emiatt némi módosítást kell eszközölnünk a struktúrán. Az alábbiakban a *filtered reference LMS* (*FxLMS*) és a *filtered error LMS* (*FeLMS*) névre hallgató módszereket ismertetem, amelyek alkalmasak a másodlagos út átvitelének kompenzálására [10]. Megjegyzendő, hogy mindkét eljárás torzítatlan olyan értelemben, hogy az adaptív szűrő továbbra is a minimális négyzetes hibájú megoldáshoz konvergál.

1.3.1 Filtered reference LMS (FxLMS)

Az FxLMS-algoritmus blokkvázlatát az 1.5-ös ábra szemlélteti. Az $\hat{A}_2$-t még a struktúra



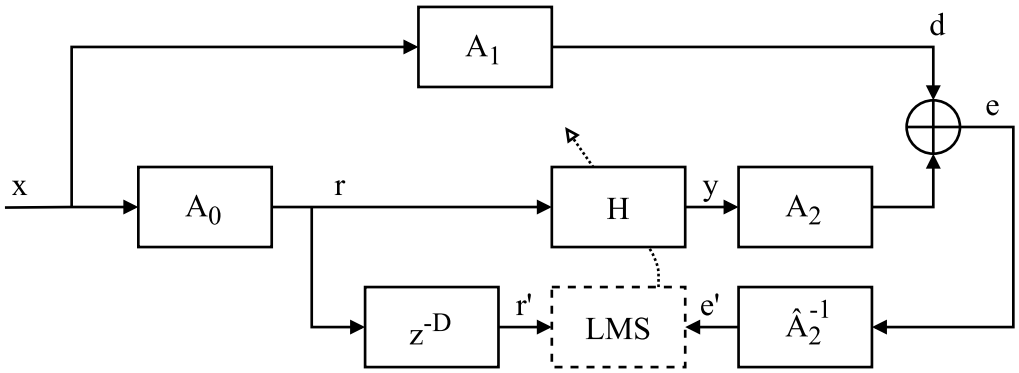
1.5. ábra: *FxLMS-algoritmuson alapulú struktúra*

működtetése előtt identifikáljuk (*offline* identifikáció), majd pedig a működés során az LMS-algoritmusnak szolgáltatott referenciajelet előszűrjük vele. Könnyen meggyőződhetünk a helyes működésről, ugyanis ha az adaptív szűrő H átvitele kellően lassan változik, akkor a linearitás miatt az utána lévő A_2-vel felcserélhető, és a két ágból az A_2 és az $\hat{A}_2$ kiemelhető közvetlenül az A_0 mögé. Ezáltal visszajutottunk az 1.1-es ábrán megismert struktúrához, mindössze annyi a különbség, hogy az r referenciajel az x bemenetből az A_0A_2 eredő átvitellel származtatható. A stabilitás szempontjából az $\hat{A}_2$-nek kizárólag a késleltetése játszik szerepet, tehát szélsőséges esetben akár egyetlen késleltetőelemmel is kiváltható lenne. Ennek az egyszerűsítésnek viszont az a hátránya az (1.13)-as összefüggés szerint, hogy az A_2 dinamikájától függően a különböző frekvenciákon más és más lesz a bátorsági tényező effektív értéke, és így a konvergenciasebesség is széles határok között változhat.

1.3.2 Filtered error LMS (FeLMS)

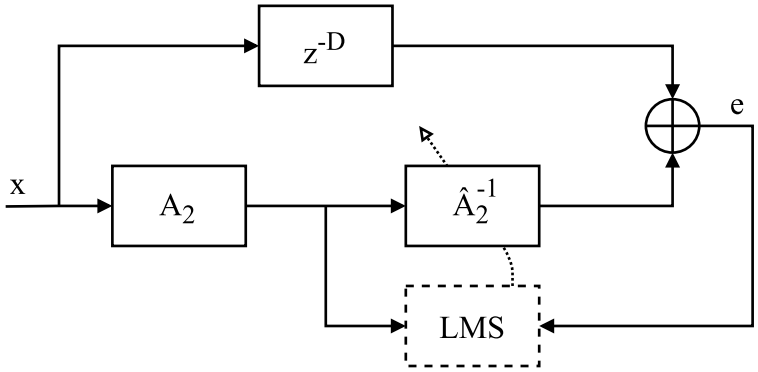
Egy másik alternatíva is kínálkozik a másodlagos út átvitelének kompenzálására, nevezetesen, hogy nem az adaptív szűrő bemeneti jelét szűrjük elő $\hat{A}_2$-vel, hanem a hibajelet $\hat{A}_2^{-1}$-zel, mint ahogyan az 1.6-os ábrán is látható. Ha az A_2 nem minimálfázisú, akkor

nem invertálható, azonban a késleltetett inverze akkor is létezik. Az $\hat{A}_2^{-1}$ ennélfogva ezt a késleltetett inverzet hivatott közelíteni. Mivel a rendszer lineáris, gondolatban az adaptív



1.6. ábra: *FeLMS-algoritmuson alapulú struktúra*

szűrő y kimeneténél lévő A_2-t áthelyezhetjük az összegző mögé, amennyiben a felső ágban szerepeltetjük az inverzét. Az e ágban lévő $\hat{A}_2^{-1}$-zel az A_2 együttesen egy z^{-D} késleltetést eredményez ideális esetben, emiatt az adaptív szűrő hangolásához alkalmazott bemeneti jel útjába is be kell iktatnunk D mintányi késleltetést, ezáltal biztosítva a két jel közötti szinkronitást. Észrevehető, hogy az adaptáció folyamatában lévő késleltetésektől eltekintve az 1.1-es ábrán látott struktúrához jutottunk azzal a különbséggel, hogy az elsődleges út eredő átvitele most $A_1 A_2^{-1}$.



1.7. ábra: *A késleltetett inverz identifikációjára alkalmas struktúra*

A késleltetett inverz identifikációja offline módon történik, például az 1.7-es ábrán ismertetett adaptív struktúra segítségével. Gerjesztőjelként többek között fehér zaj vagy ún. *„chirp”* jel alkalmazható. A szükséges D késleltetés kísérleti úton választható meg az identifikáció végére állandósuló átlagos hibaérték alapján. Általában a nagyobb késleltetés kisebb állandósult hibával párosul, és az 1.6-os struktúrában felhasználva minden frekvencián egyenletesebb konvergenciát eredményezne, viszont a késleltetés ténye negatívan befolyásolja az adaptív szűrő működését, amelyről a következő pont számol be részletesen.

1.3.3 A késleltetés negatív hatásai és kiküszöbölésük

Mind az FxLMS-, mind pedig az FeLMS-struktúra az LMS-algoritmus késleltetett változatát valósítja meg (*Delayed LMS, DLMS*). Az előbbi esetében az A_2 késleltetése, utóbbinál pedig az általunk kézben tartott D mintányi késleltetés befolyásolja a működést. A részletes analízis fényt derített arra [15], hogy az ennek felhasználásával működtetett szabályozási kör a növekvő késleltetés mellett csak a μ értékének drasztikus csökkentése árán tud stabil maradni. Ez a konvergenciasebesség visszaeséséhez vezet, illetve az 1.9-es ábrán bemutatásra kerülő szimulációs eredmények alapján elmondható, hogy a gyakorlatban elérhető maximális zajelnyomást is korlátozza az állandósult hiba nagysága révén.

A probléma gyökere abból ered, hogy a k-adik ütemben érvényes $\mathbf{w}(k)$ szűrőegyütthatókat az $\mathbf{x}(k-D)$ és $e(k-D)$ hibajelek alapján frissítjük, holott az ezekből képezhető korrekciós tag a $\mathbf{w}(k-D)$ szűrőegyütthatókra vonatkozik. Ha összevetjük a DLMS- és az LMS-algoritmusok hibajelét [16], akkor észrevehetjük, hogy némi extra számítás árán a két algoritmus egymásba alakítható. Az eljárás alkalmazható az FxLMS- és az FeLMS-struktúráknál egyaránt, mivel azonban a munkám során az FeLMS-struktúrát alkalmaztam, ezért csak az ehhez kapcsolódó konkrét lépéseket ismertetem. Az 1.6-os ábrán szereplő r jelből képzett $\mathbf{r}(k) = [r(k)\ \ r(k-1)\ \ \ldots\ \ r(k-N+1)]^T$ vektorjelölést bevezetve:

(1) Az $e'(k)$ pillanatnyi értéke szinkronban van $y(k-D)$-vel, amelyből becsülhetjük a $\hat{d}(k-D)$ elnyomandó zaj pillanatértékét, pontosabban annak az adaptív szűrő közvetlen kimenetére vetített $\hat{d}'(k-D)$ változatát:

$$\hat{d}'(k-D) = e'(k) - y(k-D) \tag{1.16}$$

(2) Az $y(k-D)$ kimenet még a $\mathbf{w}(k-D)$ alapján lett az $\mathbf{r}(k-D)$-ből kiszámítva, a legfrissebb $\mathbf{w}(k)$ szűrőegyütthatók alapján viszont már az $\hat{y}(k-D)$ adódna:

$$\hat{y}(k-D) = \mathbf{w}^T(k)\mathbf{r}(k-D) \tag{1.17}$$

(3) Ezeket felhasználva képezhetünk egy új $\hat{e}'(k)$ értéket, amelynek segítségével az aktuális $\mathbf{w}(k)$ szűrőegyütthatókhoz előírhatjuk a korrekciót:

$$\hat{e}'(k-D) = \hat{d}'(k-D) + \hat{y}(k-D) \tag{1.18}$$

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu\hat{e}'(k-D)\mathbf{r}(k-D) \tag{1.19}$$

Ezt az algoritmust nevezi a szakirodalom *módosított FeLMS-algoritmusnak* (*MFeLMS*).

1.4 Az LMS alapú szűrők konvergenciasebessége

Az ANC-rendszerek hatékonyságát egyrészt az elérhető elnyomás, másfelől pedig a *konvergencia sebessége* jellemzi. Emiatt célszerű feltárni az ez utóbbit befolyásoló tényezőket is [17]:

- A konvergenciasebesség *növekszik a μ értékével* egészen addig, amíg el nem érjük a stabilitás határhelyzetét jelentő érték felét. Ez az érték többek között függ a bemeneti jel korreláltságától, a szűrőhossztól és a szűrőegyütthatók kiindulási állapotától. A konkrét érték megválasztásakor kompromisszumot kell kötnünk, miszerint a gyors, vagy pedig inkább minimális hibájú beállást részesítjük előnyben.
- A konvergenciasebesség *csökken a rendre nagyobb szűrőhosszak (N) mellett*. Ez egyrészt azzal magyarázható, hogy amennyiben a bemeneti jel önmagával korrelált, úgy az $\mathbf{R}$ autokorrelációs mátrix kondicionáltsága romlik, vagyis jelen esetben a legnagyobb és a legkisebb sajátértékének a hányadosa növekszik [18]. A konvergencia feltétele, hogy a μ kisebb legyen a legnagyobb sajátérték reciprokánál, viszont a kis sajátértékekhez tartozó módusok esetében az így beállított érték nagyon lassú konvergenciát eredményez. A szűrőhosszak növelése ugyanakkor korrelálatlan bemeneti jel esetén is lassítja a konvergenciát, amelyre a legegyszerűbb magyarázat, hogy az (1.9)-es Wiener-megoldásban szereplő $\mathbf{R}$ csak akkor lehet teljes rangú, ha már legalább N különböző mintán keresztül átlagoltuk az $\mathbf{x}\mathbf{x}^T$ diádot, következésképp hosszabb szűrők esetén több ütemen át kell átlagolnunk.

A μ értékének megfelelő beállítására nincs általánosan érvényes formula. Bershad [19] rámutat egy összefüggésre, amely fehér zaj esetén megadja az állandósult állapotbeli hiba átlagos négyzetes értékét a μ transzcendens függvényeként. Ebből numerikus módszerek segítségével meghatározható az igényeknek és a körülményeknek is megfelelő érték. Emellett különböző heurisztikus módszerek is léteznek, amelyek változó bátorsági tényezőt alkalmaznak, mint például [20]. Az esetek egy jelentős részében ezek kellően gyors beállást ígérnek, amely után az állandósult hiba nagysága is kedvezően alakul.

Más a helyzet a szűrőhosszak tekintetében, amelyet jelen alkalmazás során alapvetően az akusztikai tér impulzusválaszának hossza határoz meg. Az 1.2-es ábrán is kivehető, hogy az impulzusválasz burkolója egy exponenciális lecsengést mutat, amelyet akkor tekintünk befejezettnek, amikor a burkolónak a kezdeti csúcsértékkel vett aránya eléri a -60 dB-t. Ezt nevezi a szakirodalom az adott tér RT_{60} paraméterének (*Reverberation Time 60*, [21]). Az impulzusválasz farokrészében összpontosuló energia általában töredéke az elején lévő energiának, ezért a konvergencia gyorsítása érdekében alkalmazhatunk az impulzusválasznál rövidebb adaptív szűrőket úgy, hogy közben az elérhető zajelnyomás nem romlik szignifikánsan. Egy másik megoldás pedig az lehet, hogy kiegészítjük az LMS-algoritmust annak érdekében, hogy több információt legyen képes kinyerni egy

adott hosszúságú bemeneti mintasorozatból a számítási igény esetleges növekedése árán. Elméletileg ez lehetséges, hiszen korábban már említésre került, hogy zajmentes esetben akár N mintán keresztüli átlagolás után kiszámíthatjuk az $\mathbf{R}^{-1}\mathbf{p}$ segítségével a Wiener-megoldást. A következő pontban egy ilyen algoritmus kerül bemutatásra.

1.4.1 NLMS ortogonális korrekciós tényezőkkel (NLMS-OCF)

Az LMS alapú adaptív szűrők konvergenciája akkor a leggyorsabb, ha az egymást követő bemeneti mintákból képzett N hosszú vektorok mind merőlegesek egymásra. Az adaptáció során ugyanis a $\mathbf{w}$ szűrőegyütthatóknak L^2 értelemben a lehető legközelebb kell kerülniük a $\mathbf{w}^*$ optimális megoldáshoz, ehhez pedig elengedhetetlen, hogy egy $\mathbb{R}^N$-beli bázist alkotó vektorrendszer minden iránya mentén[2] legalább egyszer korrekciót végezzünk. Minél inkább távol áll a bemeneti jel spektruma a fehértől, a belőlük alkotott vektorok annál inkább párhuzamossá válnak. Ez közvetve és közvetlenül is a konvergenciasebesség csökkenéséhez vezet, mivel egyrészt a párhuzamos vektorok között el kell osztanunk a μ-t a stabilitás megtartása végett, lévén, hogy azok ugyanabban az irányban végeznek korrekciót, másfelől pedig időben tovább kell átlagolnunk, hogy minden irány reprezentálva legyen a bemeneti vektorok által.

Ezt a jelenséget hivatott mérsékelni az *NLMS-OCF* (*NLMS with orthogonal correction factors*) algoritmus [22], amely az aktuális $\mathbf{x}$ és még az előző M darab bemeneti vektorból egy ortogonális vektorrendszert képez (például Gram-Schmidt ortogonalizációval), majd pedig minden irány mentén elvégzi a korrekciót. Az algoritmus további paramétere az S, amely a vektorok közötti időbeli távolságot fejezi ki, ugyanis a szerzők úgy találták, hogy minél nagyobb időszeletet fog át az M darab vektor, annál gyorsabb lesz a konvergencia, egészen $S = N/2$-ig. A számítások során szükség van a $\hat{d}(k)$ értékekre, ezért remekül használható együtt az MFeLMS- és az MFxLMS-struktúrákkal, amelyeknél ezt az adatot egyébként is használjuk. A szerzők identifikációs módban ismertették az eljárást, az alábbiakban viszont az 1.4-es ábrán látott zajcsökkentő alapstruktúra használatakor végrehajtandó lépéseket mutatom be:

(1) Kiszámítjuk az elnyomandó zaj aktuális mintájának becsült értékét, amelyre a későbbiekben lesz majd szükség:

$$\hat{d}(k) = e(k) + y(k) \tag{1.20}$$

(2) Változatlanul elvégezzük az első frissítést az aktuális bemenet és hibajel alapján:

$$\mathbf{w}_1(k) = \mathbf{w}(k-1) + \frac{\mu e(k)}{\mathbf{x}^T(k)\mathbf{x}(k)}\mathbf{x}(k) \tag{1.21}$$

[2]Véletlenszerű kiindulási állapotot feltételezve.

(3) Ismételjük az (a)-(c) lépéseket $i = 1, 2, \ldots, M$-re!

(a) Gram-Schmidt ortogonalizáció segítségével kiszámítjuk $\mathbf{x}(k - iS)$-ből az $\mathbf{x}_i(k)$-t úgy, hogy merőleges legyen $\mathbf{x}(k), \mathbf{x}_1(k), \ldots, \mathbf{x}_{i-1}(k)$ mindegyikére.

(b) Becslést adunk, hogy a szűrőegyütthatók jelenlegi állása mellett mekkora lett volna a hibajel a $k - iS$ ütemben:

$$\hat{e}_i(k - iS) = \hat{d}(k - iS) - \mathbf{x}^T(k - iS)\mathbf{w}_i(k) \tag{1.22}$$

(c) A becsült hibajel alapján előírjuk a korrekciót, de már csak az előzőekre merőleges irányban:

$$\mathbf{w}_{i+1}(k) = \mathbf{w}_i(k) + \frac{\mu\hat{e}_i(k)}{\mathbf{x}_i^T(k)\mathbf{x}_i(k)}\mathbf{x}_i(k) \tag{1.23}$$

(4) A ciklus végére megkapjuk az új szűrőegyütthatókat:

$$\mathbf{w}(k) = \mathbf{w}_{M+1}(k) \tag{1.24}$$

1.5 Több bemenetű és több kimenetű rendszerek (MIMO)

Az eddig ismertetett struktúrák könnyedén kiterjeszthetők olyan esetekre is, amikor több hibajellel és beavatkozójellel kell dolgoznunk [6], mindemellett több referenciajel figyelembevételére is van lehetőség [23]. Több hibamikrofont akkor használunk, ha a tér több pontjában is zajcsökkentésre van szükség, amely igazolhatóan csak akkor lehet hatékony, ha ilyenkor legalább ugyanannyi beavatkozó hangszórót is működtetünk. A több referenciajel használatát indokolhatja, hogy több zajforrás is jelen van egyidejűleg, de a gyakorlati tapasztalatok alapján kedvező hatása van az elérhető elnyomásra egyetlen zajforrás esetén is. Az 1.12-es szimulációs ábrán látható, hogy drámai különbség mutatkozott az egy és a két referenciajelet felhasználó rendszer teljesítménye között. Az ezzel kapcsolatos okfejtés az 1.7-es pont alatt olvasható.

Az adaptív szűrők egyik lehetséges MIMO-kiterjesztése minden beavatkozó hangsugárzóhoz és minden referenciajelhez egy-egy, korábban már bemutatott adaptív szűrőstruktúrát működtet, és több hibajel szerint végez korrekciót. Ez a változat azonban nem számol a referenciajelek korreláltságával, ami esetleg a későbbiekben befolyásolhatja a rendszer teljesítményét. Formálisan I hibajel, J hangsugárzó és L referenciajel mellett a következő összefüggésekkel írható le az FeLMS struktúrát feltételezve (1.6-os ábra), továbbá a

másodlagos út $\hat{A}_2^{-1}$ késleltetett inverzét egy N hosszú FIR-szűrővel ($\mathbf{a}_2^{-1}$) modellezve:

$$\mathbf{e}_i(k) = [e_i(k) \;\; e_i(k-1) \;\; \ldots \;\; e_i(k-N+1)]^T, \qquad 0 < i \leq I \tag{1.25}$$

$$e'_{ij}(k) = \mathbf{e}_i^T(k)\mathbf{a}_{2,ij}^{-1}(k), \qquad 0 < i \leq I, \;\; 0 < j \leq J \tag{1.26}$$

$$\mathbf{w}_{jl}(k) = \mathbf{w}_{jl}(k-1) - \mu \sum_{i=1}^{I} e'_{ij}(k)\mathbf{r}_l(k-D), \qquad 0 < j \leq J, \;\; 0 < l \leq L \tag{1.27}$$

$$y_j(k) = \sum_{l=1}^{L} \mathbf{w}_{jl}^T(k)\mathbf{r}_l(k), \qquad 0 < j \leq J \tag{1.28}$$

1.6 Szimulációs eredmények

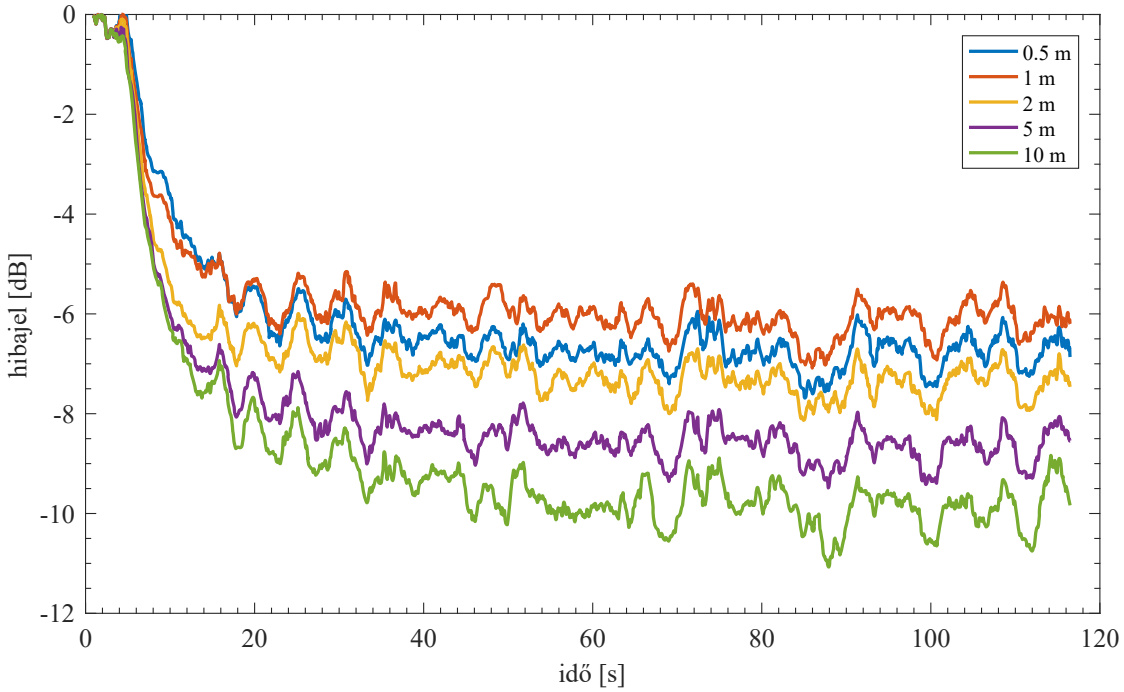
Érdemesnek tartottam az eddig ismertetett eljárások kipróbálását egy valósághű szimuláció keretein belül, amellyel egy átfogó képet kaphatunk arról, hogy hogyan teljesítenek az egyes algoritmusok különböző feltételek mellett, illetve a készülő zajcsökkentő rendszer teszteléséhez is majd viszonyítási alapként szolgálhat. Mivel jelen dolgozat a több referenciajelet felhasználó ANC-rendszerekre összpontosít, ezért eltekintek a több hibajelet és beavatkozó hangszórót alkalmazó változatok vizsgálatától. A szimulációhoz az *Open AIR Library* nevű weboldal impulzusválasz-gyűjteményéből választottam egy nagyjából 1 másodperces RT_{60} paraméterű csomagot, amely egy létező teremben, 8 különböző pozícióban felvett impulzusválaszt tartalmaz. A mikrofonok egy felülnézetből 4 méter élhosszúságú négyzet sarokpontjaiban és az élközepeinél lettek elhelyezve, a hangforrás pedig a négyzet középpontjában volt. Az impulzusválaszok közül 7 darabot használtam fel, amelyeket $f_s = 8$ kHz-es mintavételi frekvenciára decimáltam. Ezután az impulzusválaszok első csúcsát egymáshoz igazítottam, közülük egynek pedig Q mintányi késleltetést állítottam be a többihez képest. A késleltetett szűrő fogja szolgáltatni a szimuláció számára az elsődleges zajt, ami $c_{hang} = 343$ m/s-nak feltételezett hangsebesség mellett megegyezik azzal, hogy az RM-ek fizikailag kb. $(Q \cdot c_{hang})/f_s$ méterrel közelebb helyezkednek el a zajforráshoz az EM-hez képest.[3] A másik 6 szűrőt a referenciajelek előállításához használtam fel. Az x mintasorozatként 2 perc hosszú, Gauss-eloszlású fehér zajt alkalmaztam. A beállási görbék kinyeréséhez előbb a hibajel abszolút értékét képeztem, ezután 300 tap-es, kétirányú (fázistolás nélküli) aluláteresztő szűrésnek vetettem alá, majd pedig 16000 mintás ablakú mediánszűrést végeztem az adatfolyamon. A mediánszűrő előnye, hogy miközben simít, megtartja az éleket, így jól vizsgálhatóvá válik a konvergenciasebesség. A szimulációban az A_2 átvitelt egy késleltetéssel helyettesítettem, ami jó minőségű hangsugárzó használatát feltételezi. Szintén feltételeztem, hogy a beavatkozó hangsugárzó közel van az EM-hez, ezáltal csekély az általa kibocsátott hang erőssége, és így az RM-ek jelére érdemben nincs

[3] A τ_p feldolgozási időt elhanyagolhatónak tekintve.

hatással. Minden szimulációs ábrán $t = 4$ s-nál lép működésbe a rendszer, a hibajel 0 dB-es szintje a zajcsökkentés nélküli állapotra jellemző. A μ értékét minden konfiguráció mellett igyekeztem úgy beállítani, hogy a vizsgálat tárgyának szempontjából a lehető legkedvezőbb eredményt szolgáltassa.

1.6.1 Az EM és az RM-ek közötti útkülönbség hatása

Az 1.1-es pontban kifejtettem, hogy nem elegendő az előidejűség biztosítása az EM és az RM-ek között, mivel az A_2 átvitel egységkörön kívüli zérusainak kellő pontosságú invertálásához több késleltetésre van szükség. Nem tudjuk viszont, hogy mennyivel kell növelnünk az időkülönbséget ahhoz, hogy érdembeli változást tapasztaljunk az elnyomásban. Az 1.8-as ábrán különböző, egy átlagos teremben még kivitelezhetőnek mondható



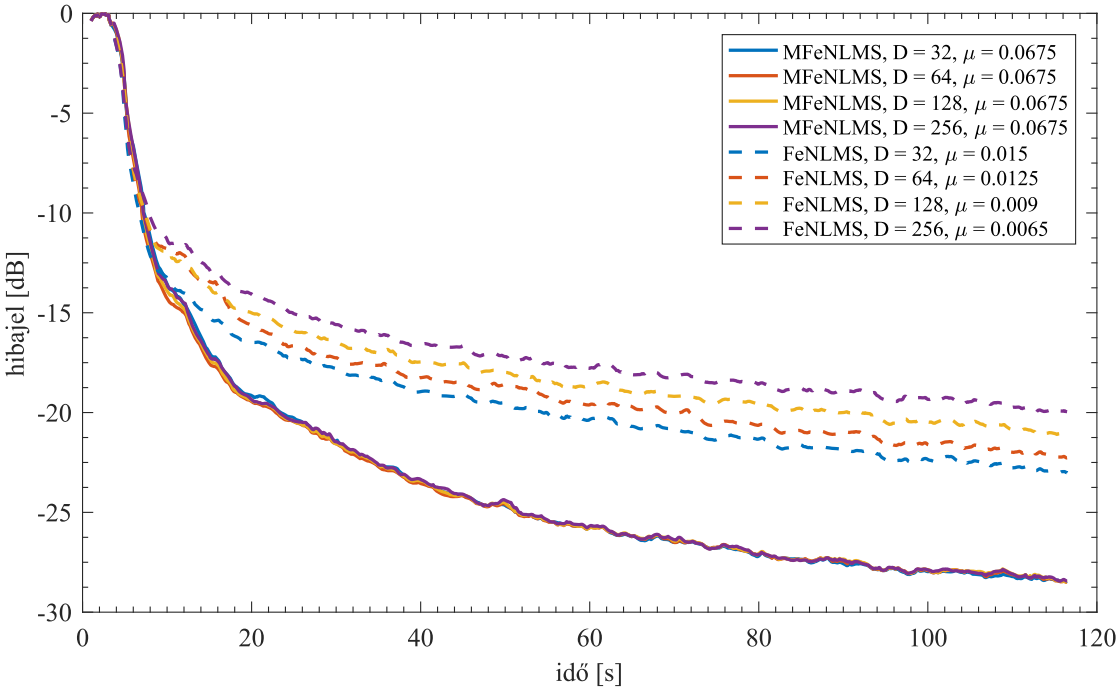
1.8. ábra: *Az EM és az RM-ek közötti távolság hatása (MFeNLMS | 1 ref. | N=8192 | D=192 | Q=12,24,48,120,240 | $\mu = 0.125$)*

útkülönbség esetén láthatjuk a hibajel alakulását. Az alkalmazott adaptív szűrő 8192 tap-es, amely meghaladja a szimulációhoz használt impulzusválaszok hosszát. Megfigyelhető, hogy a távolság minden egyes duplázásával nagyjából 1-1 dB-t nyerhetünk, bár jelen konkrét esetben fél méternél jobb eredmény adódott, mint egy méternél. Érdekességképp megvizsgáltam, mi történik extrém (50-100 méteres) útkülönbségek esetén, viszont az elérhető elnyomás ekkor sem volt több 15-16 dB-nél. Hozzá kell tenni, hogy a szimuláció nem teljesen valósághű a vizsgált szemszögből, hiszen ha közelebb kerülnek az RM-ek a zajforráshoz, akkor nő a közvetlen hang aránya a terem zengéséhez képest, amely általában sokat javít az invertálhatóságon. A szimuláció tanulsága, hogy a zajelnyomás átütő javulását

nem feltétlen az EM és az RM-ek jelei közötti időkülönbség növelésétől várhatjuk.

1.6.2 A korrigálatlan késleltetés hatása

A szabályozási körök stabilitása általában romlik a megjelenő késleltetések hatására. Az 1.3.3-as pontban bemutattam, hogyan lehet az FeLMS- és FxLMS-struktúrák használatából eredő késleltetések hatását kompenzálni. Az ismertetett eljárás növeli a számításigényt, ezért érdemes megvizsgálni, mennyit nyerhetünk a stabilitás, a konvergenciasebesség és az elérhető elnyomás terén. Az eredmények az 1.9-es ábrán követhetők nyomon. Jól látható,



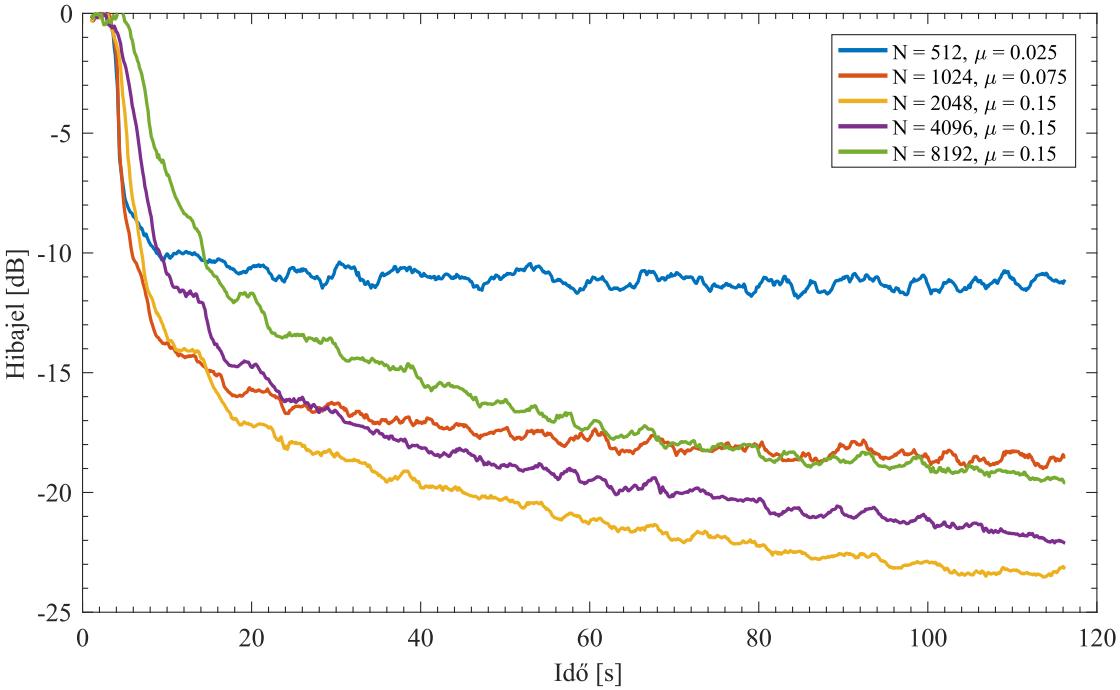
1.9. ábra: *A késleltetés hatása (6 ref. | N=2048 | Q=24)*

hogy a késleltetés növekedésével egyre kisebb μ-t kellett alkalmazni, vagyis csökkent a kör stabilitása. A különböző konfigurációk hasonlóan viselkedtek a 10 dB-es határ eléréséig, viszont a nem korrigált struktúra konvergenciasebessége innentől kezdve jelentősen visszaesett, ami korlátozza a gyakorlatban (vagyis észszerű időn belül) elérhető elnyomást is. A korrigált struktúra viselkedése ugyanakkor a várakozásoknak megfelelően a késleltetéstől függetlennek bizonyult. Érdemes tehát az FeLMS-struktúra helyett az MFeLMS-változatot használni, amennyiben a rendelkezésre álló számítási kapacitás ezt megengedi.

1.6.3 A szűrőhossz hatása

Már többször is szó esett az alkalmazott adaptív szűrők hosszának – adott mintavételi frekvencia mellett együtthatószámának – szerepéről, amely kulcsfontosságú lehet az elérhető elnyomás maximalizálásában. Elméletileg minél nagyobb hányadot képes modellezni

az alkalmazott szűrő a terem impulzusválaszának egészéből, annál jobb elnyomásra számíthatunk. Az együtthatószám növelése viszont kétélű fegyver, az 1.3.3-as pontban elmondottak alapján ugyanis a hosszabb szűrők velejárója a lassabb konvergencia. Jól tükrözi az eddigi megállapításokat a szimulációkról készült 1.10-es ábra. Az 512, 1024

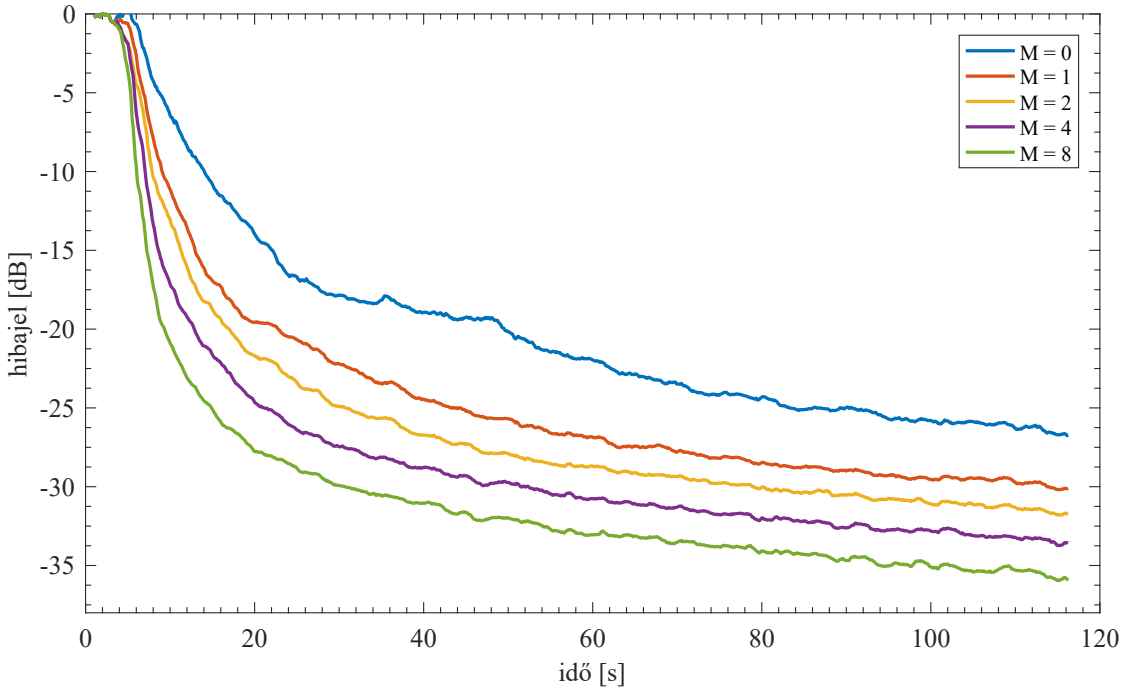


1.10. ábra: *A szűrő hosszának hatása (MFeNLMS | 2 ref. | D=192 | Q=24)*

és 2048 együtthatós szűrők esetében a μ növelésével kompenzálni lehetett a növekvő szűrőhosszakból adódó lassulást, miáltal persze az állandósult állapotbeli elnyomásból vesztünk néhány decibelt, de még így is látványos maradt a javulás. A 4096 és 8192 együtthatós szűrőknél viszont már nem lehetett tovább növelni a μ-t stabilitási problémák miatt, így pedig az állandósuló hiba nagyságának leolvasásához hosszabb megfigyelésre lett volna szükség. A tendencia ennek ellenére világosan látszik, vagyis az eddigi következtetések a konvergenciasebesség és az elérhető elnyomás fordítottan arányos mivoltáról helytállóak.

1.6.4 Az ortogonális korrekciós tényezők hatása

Az előző pont alapján azt mondhatjuk, hogy valóban jó szolgálatot tehetne a gyakorlatban egy olyan algoritmus, mint amilyen 1.3.3-as pontban bemutatott NLMS-OCF. A kérdés természetesen itt is az, hogy a várható javulás milyen arányban áll a megvalósításhoz szükséges extra számítási igénnyel. A Gram-Schmidt ortogonalizáció a korrekciós lépések M számától és az N szűrőhossztól függően $O(NM^2)$ műveletet igényel, illetve minden korrekciós lépésnél a hiba becsléséhez el kell végezni újra a szűrést, valamint módosítani a szűrőegyütthatókat, ami ütemenként további $O(MN)$ műveletet jelent. Az 1.11-es ábrát megpillantva azonban kijelenthetjük, hogy az algoritmus valóban drámai hatást képes

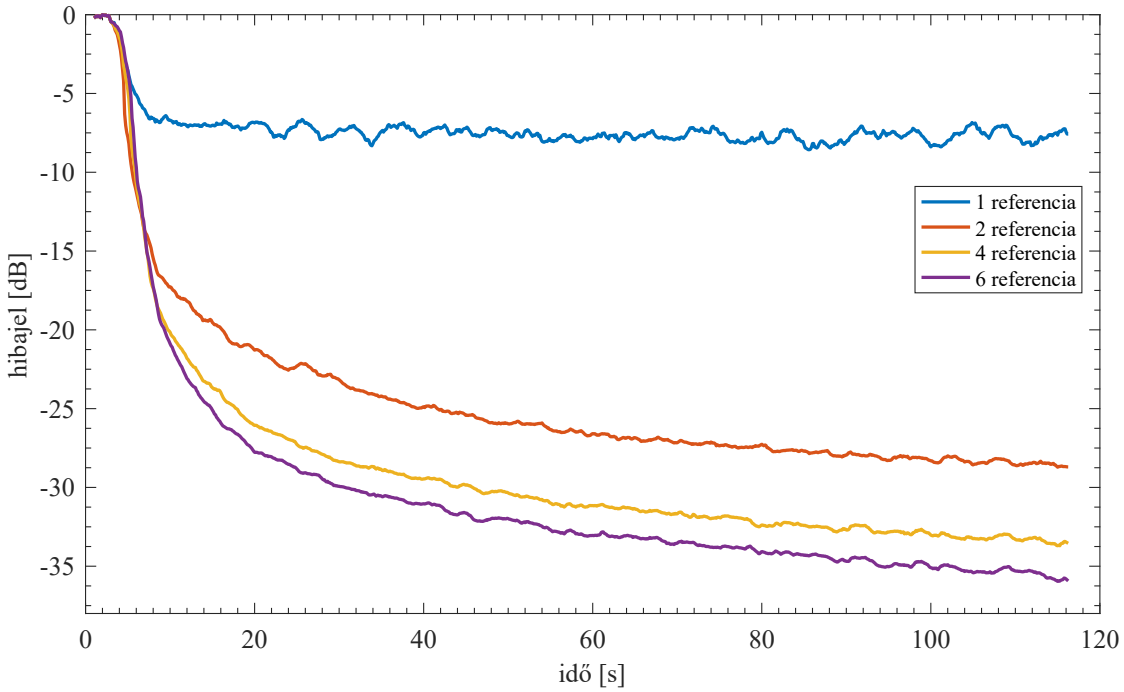


1.11. ábra: *Az NLMS-OCF hatása (MFeNLMS-OCF | 6 ref. | D=192 | Q=24 | N=4096 | S=256 | μ=0.1)*

gyakorolni mind a konvergenciasebességre, mind pedig a gyakorlatban elérhető zajelnyomásra, ezért érdemes erőforrásokat áldozni az implementációjára. Megjegyzendő, hogy az egymást követő korrekciós vektorok közötti időbeli távolságot mintákban kifejező S paraméter változtatása csak nüansznyi eltéréseket eredményezett a beállási görbéken, ezért az értékét végig 256-nak választottam, miáltal $M = 8$ korrekciós faktor esetén is csak fél másodpercnyi adatmennyiséggel többre volt szükség a referenciajelekből.

1.6.5 A felhasznált referenciajelek számának hatása

Az 1.6.1-es pontban láthattuk, hogy egy referenciajel felhasználásával nem lehet kimagasló elnyomást elérni még akkor sem, ha irreálisan nagy előidejűséget biztosítunk az RM jelének az EM jeléhez képest. Érdemes azonban megvizsgálni, hogy milyen hatással van a rendszer működésére a további RM-ek bevonása. Az 1.10-es ábrán látható szimuláció során az RM-ek és az EM közötti útkülönbség 1 méter volt ($Q = 24$), ami csak 6-7 dB elnyomásra volt elegendő egyetlen referenciajel mellett, viszont a második referenciajel bevonásakor lenyűgöző javulás mutatkozott, amellyel az elnyomás a 25 dB-es határt is átlépte két perc után (1.12. ábra). További referenciajelek felhasználásával ez az eredmény is túlszárnyalható, azonban a referenciajelek számának növekedésével egyre kevésbé lesz látványos a változás.



1.12. ábra: *A referenciajelek számának hatása (MFeNLMS-OCF | D=192 | Q=24 | N=4096 | M=8 | S=256 | μ=0.1)*

1.7 Inverz átvitel több referenciajel felhasználásakor

Az 1.6.5-ös pontban látott jelenség, amely szerint lényeges különbség mutatkozott az egy és a két referenciajelet felhasználó rendszer elnyomása között, további vizsgálódásra adott okot, ugyanis elképzelhető, hogy az 1.1-es pontban bemutatott késleltetett inverzen kívül más mechanizmus is létezik az inverz átvitel előállítására több bemeneti csatorna esetén.

Példaképp vizsgáljunk meg egy fehér zajjal gerjesztett, két kimenettel rendelkező lineáris rendszert, amelyhez egy két bemenetű invertáló rendszert illesztve a célunk a gerjesztő fehér zaj visszaállítása. Az elemzés megköveteli néhány jelölés bevezetését, így a gerjesztőjel mintáiból alkotott vektor legyen

$$\mathbf{x}(k) = [x_0(k) \;\; x_1(k) \;\; \dots \;\; x_{2N-2}(k)]^T, \tag{1.29}$$

és feltételezzük, hogy a megfigyelt rendszer két kimenete ebből két N hosszú FIR-szűrővel előállítható, így az ezekhez tartozó, $\mathbb{R}^{N\times(2N-1)}$-es konvolúciós mátrixok:

$$\mathbf{A} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{N-1} & 0 & 0 & \cdots & 0 \\ 0 & a_0 & a1 & \cdots & a_{N-2} & a_{N-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_0 & a_1 & a_2 & \cdots & a_{N-1} \end{bmatrix} \tag{1.30}$$

$$\mathbf{B} = \begin{bmatrix} b_0 & b_1 & b_2 & \cdots & b_{N-1} & 0 & 0 & \cdots & 0 \\ 0 & b_0 & b1 & \cdots & b_{N-2} & b_{N-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & b_0 & b_1 & b_2 & \cdots & b_{N-1} \end{bmatrix} \tag{1.31}$$

A célunk tehát egy olyan invertáló rendszer működtetése, amely a megfigyelt rendszer bemenetét közelíti. A hibajel ilyenkor az

$$e = \left(\mathbf{c}^T - \mathbf{w}_1^T\mathbf{A} - \mathbf{w}_2^T\mathbf{B}\right)\mathbf{x} \tag{1.32}$$

összefüggésből számítható, ahol a $\mathbf{c} \in \mathbb{R}^{2N-1}$ a megfigyelt és az invertáló rendszer eredő átvitelét, tehát egy egységimpulzust vagy annak késleltetett változatát takarja:

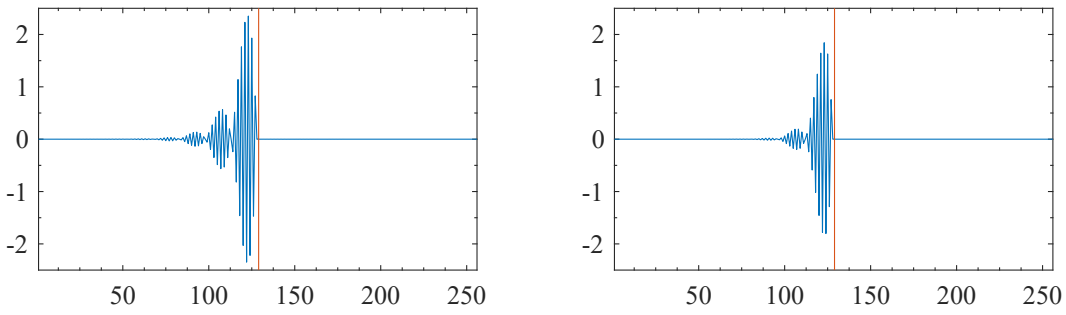
$$\mathbf{c} = [1 \;\; 0 \;\; \ldots \;\; 0]^T, \tag{1.33}$$

a $\mathbf{w}_1$ és a $\mathbf{w}_2$ pedig az invertáló rendszer N együtthatós szűrőinek vektorai. A hibajel ismeretében az 1.2-es pontban leírtakhoz hasonló gondolatmenettel eljuthatunk a legkisebb négyzetes analitikus megoldáshoz, amely az

$$\begin{bmatrix} \mathbf{A}\mathbf{A}^\mathbf{T} & \mathbf{A}\mathbf{B}^\mathbf{T} \\ \mathbf{B}\mathbf{A}^\mathbf{T} & \mathbf{B}\mathbf{B}^\mathbf{T} \end{bmatrix} \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{A}\mathbf{c} \\ \mathbf{B}\mathbf{c} \end{bmatrix} \tag{1.34}$$

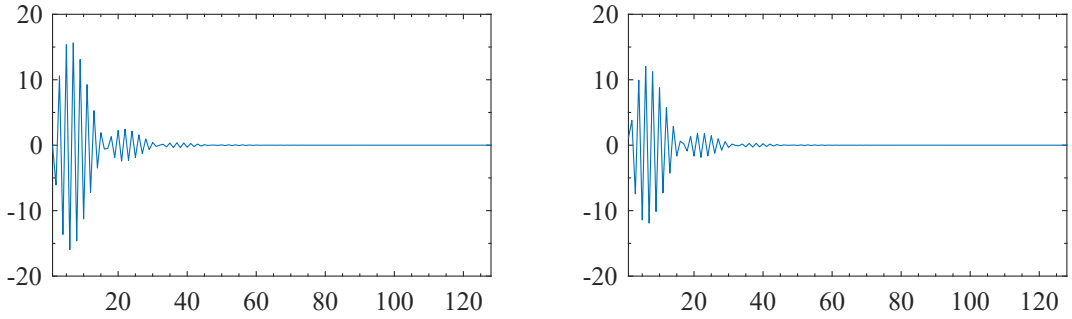
egyenletrendszer megoldását teszi szükségessé. Ez általánosságban nem teljes rangú, ezért a megoldás pszeudoinverz segítségével történhet.

Példaképp legyenek az egyik csatornához tartozó szűrő együtthatói $\mathbf{a} = [1 \;\; 2{,}1519 \;\; 1{,}21]$, a másiké $\mathbf{b} = [1 \;\; 2{,}2558 \;\; 1{,}3225]$! Mindkét átvitel egy-egy egységkörön kívülre eső zérus-párt tartalmaz, így ezeknek külön-külön csak késleltetett inverzük létezik. Az inverzeket



1.13. ábra: *A csatornák külön-külön vett késleltetett inverzei*

az 1.13-as ábra szemlélteti $D = 128$ mintányi késleltetés mellett. Ha azonban megoldjuk a fenti egyenletrendszert, és a két jel együttes felhasználásával próbáljuk közelíteni az egységnyi eredő átvitelt, akkor késleltetés nélkül az 1.14-es ábrán látható szűrők adódnak megoldásképp, az invertáló rendszer kimenetére vonatkozó átvitel pedig a számábrázolás precizitásán belül egy tökéletes egységimpulzus lesz.



1.14. ábra: *Az invertáló rendszer szűrőegyüttható (*$\mathbf{w_1}$ *és* $\mathbf{w_2}$*)*

Jól mutatja a példa, hogy két (vagy több) különböző referenciajel rendelkezésre állása esetén a késleltetett inverztől eltérő mechanizmus is szerepet játszik az inverz átvitel előállításában. A hatékonysága valószínűleg azzal magyarázható, hogy két referenciajel felhasználásakor az eredő átvitel a

$$W_1 A + W_2 B = C \tag{1.35}$$

szerint alakul, vagyis polinomok összegeként áll elő, amelynek eredő zérusai a matematika jelenlegi állása szerint megjósolhatatlanok a tagok zérusainak ismeretében.

További analízisre így nincs lehetőség, azonban támaszkodhatunk a zajcsökkentéshez kapcsolódó terület, a *„visszhangtalanítás”* (*dereverberation*) terén elért eredményekre [24]. Eszerint ugyanis egy reflexiókat is tartalmazó térben két különböző pozícióban elhelyezett mikrofon által vett jelfolyamból egy konstans szorzó bizonytalanságáig meghatározhatók a forrástól a mikrofonokig terjedő átviteli függvények, és így visszanyerhető a forrás által kibocsátott „nyers” hang is, amennyiben *a két átviteli függvénynek nincsenek közös zérusai*. Egy ANC-rendszer esetén azonban annyival egyszerűbb a feladat, hogy nem szükséges a zajforrás jelének rekonstruálása, csupán a referenciajelek segítségével kell kifejezni az EM-nél érzékelhető zajt, így például az A és B átvitelekben egyaránt jelen lévő zérusok szükség esetén kiolthatók úgy, hogy a kioltó pólusokat a W_1 és W_2 átvitelekben is szimuláljuk. Ebből viszont észrevehető, hogy az A és B átvitelekben egyaránt jelen lévő, egységkörön kívülre eső zérusok invertálása már valóban nem lehetséges ily módon. További referenciajelek bevonásával ezeknek a közös zérusoknak a számát csökkenthetjük, amelyek így egyre fogynak, ez pedig magyarázatot ad arra, hogy miért lesz egyre kevésbé szignifikáns a javulás további referenciajelek bevonásakor.

2 A GPU-programozás alapjai

2.1 Út a szekvenciális kódtól a GPU-kódig

A digitális jelfeldolgozás során az egyik leggyakrabban felmerülő feladat a véges impulzusválaszú szűrők futtatása, vagyis a FIR-szűrés. Ha klasszikus, szekvenciális programkódban gondolkodunk, akkor bizonyára mindenkinek az alábbihoz hasonló *C nyelvű* implementáció fordulna meg a fejében először.

```
void fir_c( float *coeffs, int N,   // N darab (fordítva tárolt) szűrőegyüttható
            float *data, int L,     // L hosszú bemeneti adatsor
            float *out)             // szűrt kimeneti adatsor (L-N hosszú)
{
    for(int i = 0; i < (L-N); i++)
        for(int j = 0; j < N; j++)
            out[i] += coeffs[j] * data[i+j];    // konvolúció
}
```

2.1. lista: *FIR-szűrés C nyelven*

Kevesebben tudják ugyanakkor, hogy a mai PC-k döntő hányadában fellelhető CPU-k támogatnak valamilyen *SIMD* (*single instruction – multiple data*) jelegű vektorutasításkészletet, leggyakrabban az *SSE* (*Streaming SIMD Extension*) különböző fokozatait. Az SSE 128 bites adatvektorokkal történő műveletvégzést tesz lehetővé, ami IEEE 754/1985 szabványra épülő, egyszeres pontosságú (32 bites), lebegőpontos (vagyis `float`) típusok esetén négy *egymástól független* adaton tudja ugyanazt a műveletet végrehajtani. Ezeket a vektorutasításokat a fordítóprogramok többsége is támogatja, és már gépi utasítások használata nélkül, ún. *intrinsic függvényhívások* segítségével elérhetők, ezáltal a kód hordozható marad. A fenti programrészletet is átalakíthatjuk úgy, hogy kiaknázza a vektorutasítások nyújtotta előnyöket.

```
void fir_sse( /* ... */ )
{
    for(int i = 0; i < (L-N); i += 4)   // egyszerre 4 kimenetet számol
    {
        __m128 acc = _mm_setzero_ps();  // akkumulátor inicializálása nullára
        for(int j = 0; j < N; j++)
        {
            __m128 c = _mm_load1_ps(coeffs + j);    // egy szűrőegyüttható betöltése
                                                    // mind a 4 pozícióba
            __m128 d = _mm_loadu_ps(data + i + j);  // 4 egymást követő adat betöltése
            __m128 res = _mm_mul_ps(c, d);          // aktuális részeredmény
            acc = _mm_add_ps(acc, res);             // hozzáadás az akkumulátorhoz
        }
        _mm_storeu_ps(out + i, acc);    // a 4 kimeneti adat tárolása
    }
}
```

2.2. lista: *FIR-szűrés SSE-utasításokkal*

Bár első ránézésre úgy tűnhet, hogy ez a változat sokkal több utasítás végrehajtását követeli meg, valójában a 2.1-es listán látható egyetlen kódsor ugyanennyi gépi utasításra fordul le.

Az SSE-t használó változatnál viszont a külső ciklusváltozó négyesével növekszik, ezért elméletileg (és általában a gyakorlatban is) négyszer gyorsabb kódhoz jutottunk. Persze van némi megkötés, hiszen az $(L - N)$ osztható kell, hogy legyen 4-gyel, ezt viszont minden esetben biztosítani tudjuk a szűrőegyütthatók és/vagy a bemeneti adatsor nulla értékekkel történő kiterjesztésével (*zero padding*).

A C++ nyelv 2011-es veziójától, vagyis a C++11-től kezdve platformfüggetlen szálkezelést is egyszerűen meg tudunk valósítani, ezáltal egy *SMT*-re (*Simultaneous Multithreading*) alkalmas, többmagos processzoron további gyorsulást érhetünk el.

```
1  void fir_smt( /* ... */ )
2  {
3      // logikai processzorok száma
4      unsigned k = std::thread::hardware_concurrency();
5
6      std::thread *t = new std::thread[k];
7
8      int slice = N/k;    // ekkora szelet jut egy szálnak
9
10     for(int i = 0; i < k; i++)  // k szál elindítása, mindegyik egy szeleten dolgozik
11         t[i] = std::thread(fir_sse, coeffs, N, data + i*slice, slice, out + i*slice);
12
13     for(int i = 0; i < k; i++)  // minden szálat meg kell várni kilépés előtt
14         t[i].join();
15
16     delete[] t;
17 }
```

2.3. lista: *FIR-szűrés több szálon (SMT)*

A szálak a teljes adatsort egymás között felosztják, és mindegyik egy-egy szeletet számol a korábbi, SSE-t használó megvalósításra támaszkodva. Egy négymagos processzor mellett ez elméletileg akár 16-szoros gyorsulást is eredményezhet az optimalizálatlan verzióhoz képest. Esetünkben minden szál ugyanazt a függvényt hívta meg, azonban tudni kell, hogy SMT esetén egymástól függetlenül futnak a szálak, és így eltérő feladatokat is végezhetnek.

A GPU-k, vagyis grafikus processzorok általános célú programozása (*GP-GPU*; *General purpose computing on GPU*) során az eddig bemutatott párhuzamosítási metodikák egyfajta ötvözetével szembesülünk. Az alapvető programozási egység a *szál* (*thread*), viszont a CPU-knál megszokott 2-8 szállal ellentétben itt 512-1024 szál egyidejű futtatására kell gondolni, vagyis egy sokkal finomabb felosztású párhuzamosításról van szó. Különbözik még az SMT-től, és inkább a SIMD-re hasonlít a tekintetben, hogy a szálak mind ugyanazt a kódot futtatják, alapvetően ugyanazt a műveletsort hajtják végre. Lehetőség van viszont elágazások (például `if-else`) használatára, viszont ilyenkor az eredő futásidő annyi lesz, mintha minden szál minden ágon lefutott volna, ezért lehetőleg kerülni kell a használatukat.

A GPU programozása történhet a *CUDA* vagy pedig az *OpenCL* keretrendszer segítségével. A CUDA az Nvidia saját GP-GPU platformjának, illetve a hozzá tartozó programozói interfésznek a neve is egyben [25]. A C és C++ nyelveket néhány kulcsszóval bővíti ki, amelyek lehetővé teszik a GPU-kód (*kernelkód*) hívását a processzoron futó kódból. A CUDA esetén nagy hátrány programozói szempontból, hogy a hatékony implementációhoz elengedhetetlen a mögöttes hardverarchitektúra ismerete, amely ráadásul folyamatosan

változott az évek során. Az egyes Nvidia GPU-kon elérhető képességekről az adott eszköz *compute capability* (*CC*) verziószáma árulkodik. Az OpenCL ezzel szemben egy szabványosított heterogén számítási keretrendszer, amely a platformfüggetlen párhuzamosítást helyezi előtérbe. Az OpenCL-lel történő fejlesztés éppen az egyetemesség miatt valamelyest körülményesebb, illetve az Nvidia grafikus processzorain általában sokkal jobb eredményeket lehet elérni a CUDA-val, mivel az Nvidia (érthető okokból) hanyagolja az OpenCL támogatását. A rövidebb fejlesztési ciklus, a többéves programozói tapasztalat, illetve az egyetemi és a saját számítógépemben lévő grafikus kártya típusa folytán a CUDA platformot használtam a vizsgamunkám elkészítéséhez.

```
__global__ void fir_gpu( /* ... */ )    // kernelfüggvény
{
    // Kezdőcím kiszámítása az adott szálhoz
    int offset = blockIdx.x * blockDim.x + threadIdx.x;

    float acc = 0.0f;

    for(int i = 0; i < N; i++)  // egy kimeneti minta előállítása
        acc += data[offset + i] * coeffs[i];

    out[j] = acc;   // eredmény tárolása
}

void execute_fir( /* ... */ )   // hoszt kód, amely meghívja a kernelt
{
    int threads = 512;
    int blocks = L / threads;
    fir_gpu<<< blocks, threads >>>(coeffs, N, data, out);
}
```

2.4. lista: *FIR-szűrés GPU-n*

A 2.4-es listán a FIR-szűrő legegyszerűbb CUDA-s kernelfüggvényét, illetve az azt meghívó CPU-kódot (*hoszt kódot*) olvashatjuk. A kernelfüggvényt a `__global__` kulcsszó különbözteti meg a hoszt kódtól, és a GPU-n futó szálak mindegyike ezt a műveletsort fogja végrehajtani. A meghívásához speciális szintaktika tartozik; a(z) `<<< >>>` jelek között adhatjuk meg, hogy hogyan szeretnénk felosztani a végrehajtandó feladatot. Egy *blokkban* általában 256-1536 szál futtatását szokás kitűzni célul, amelynek miértjéről még szó fog esni a továbbiakban. Az egy blokkon belül lévő szálak elérnek egy közös, gyors memóriaterületet, a *shared memory*-t, amelyen keresztül lehetőség van a szálak közötti kommunikáció lebonyolítására, illetve lehetőség van az egyébként egymáshoz képest aszinkron futó szálak explicit szinkronizációjára. A feladat dimenzionalitásától függően a blokk szálait 1, 2 vagy 3 dimenziós egységekbe szervezhetjük, indexelhetjük, ami kizárólag a programozó kényelmét szolgálja. A felosztás ezután a kernelfüggvényben a `threadIdx` struktúra `x,y,z` változóin keresztül érhető el, így tehát minden szál hozzáfér egy egyedi azonosítóhoz, amely alapján más-más memóriacímekre tudnak hivatkozni, ezáltal egyfajta SIMD-jelleget kölcsönözve az architektúrának. Az ANC során kizárólag egydimenziós szűrési feladatokat kell ellátni, ezért a többdimenziós szervezés ismertetésétől eltekintek. A szálak egy blokkjának definiálása után nincs más dolgunk, mint lefedni a teljes feladatot azokkal, vagyis annyi blokkot futtatni, amely a teljes feladat megoldására elegendő. Az

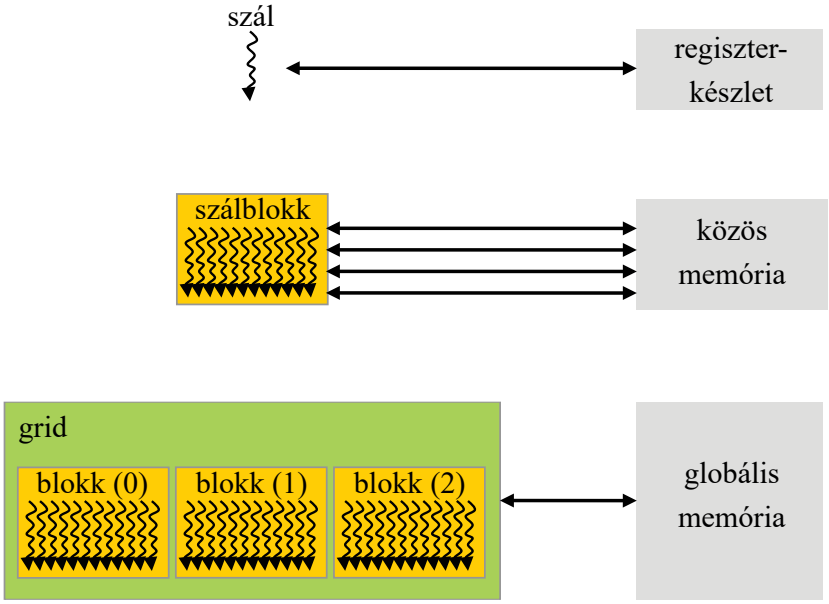
összes blokkot együttesen *gridnek* nevezzük. Tudnunk kell, hogy a különböző blokkokba tartozó szálak között csak nehézkes és lassú kommunikáció valósítható meg, illetve a blokkon belüli szinkronizáció is nagymértékben rontja a hatékonyságot. A példakódban 512 szálból áll egy blokk, minden szál egy kimeneti mintát számol, vagyis egy L hosszú adatfolyam esetén $\lceil L/512 \rceil$ blokkból áll a grid. A kernelfüggvény hívásánál lévő felosztás futás közben is elérhető; esetünkben a `blockDim.x` írja le, hogy hány szál van egy blokkban (512), illetve a `gridDim.x`, hogy hány blokkot futtatunk összesen ($\lceil L/512 \rceil$). Látható, hogy az `offset` változó értéket, amely a bemeneti adatfolyamban jelöli ki az adott szál számára a kezdőpozíciót, ezeknek a beépített struktúráknak a segítségével számítottam ki. A `threadIdx.x` az adott szál blokkon belüli pozícióját adja meg, viszont a címzéshez tudnunk kell, hogy hány (egyenként 512 szállal rendelkező) blokk dolgozik még a szóban forgó előtt, ez pedig megegyezik a `blockIdx.x * blockDim.x` értékével.

A bemutatott GPU-s FIR-szűrő, bár valószínűleg sokkal gyorsabb futásidőt eredményezne, mint bármelyik CPU-s változat, korántsem aknázza ki a GPU összes képességét. A további optimalizáció viszont megköveteli az architekturális alapok ismeretét.

2.2 Az architektúra rövid bemutatása

Az első probléma a 2.4-es listán olvasható kóddal az, hogy nem használja hatékonyan a memóriát. A bemeneti adatok tömbje ugyanis általában a grafikus kártya fedélzeti, *globális memóriájában* kerül elhelyezésre a kernelhívás előtt, amely a GPU-hoz 100-500 GB/s-os sávszélességű buszon keresztül csatlakozik. Bár a CPU mellett lévő memória esetében megszokott 10-50 GB/s-os sávszélességhez képest ez egy nagyságrendbeli különbséget jelent, azonban a GPU több TFLOP/s számítási teljesítményének fényében ez még mindig meglehetősen kevés, és az alkalmazások egy jelentős hányadánál ez jelenti a szűk keresztmetszetet. A FIR-szűrés esetén egy kimeneti minta számításához N darab bemeneti adat és szűrőegyüttható beolvasása szükséges, tehát az algoritmus memóriaintenzív. Egyetlen bemeneti adatot ugyanakkor N darab kimeneti minta előállításához is felhasználunk, vagyis a memóriaolvasások redundánsak. Szerencsére a globálismemória-műveleteket már a CC2.0 (Fermi architektúra) óta egy L2 gyorsítótár is segíti, áthidalva a tartományok közötti sebességkülönbséget. Az igazán hatékony megoldás viszont abban rejlik, hogy a gyakran használt adatokat betöltjük a GPU chipjén belüli közös memóriába. Ehhez az egy blokkon belül lévő összes szál hozzáfér, és a mérete legfeljebb 48 kB lehet blokkonként az újabb architektúrák esetében, tehát egy viszonylag szűkös erőforrás. Még ennél is egy nagyságrenddel gyorsabb a *regisztereken* történő műveletvégzés, amely viszont minden szál számára egy privát erőforrás. A teljes memóriahierarchiát a 2.1-es ábra szemlélteti.

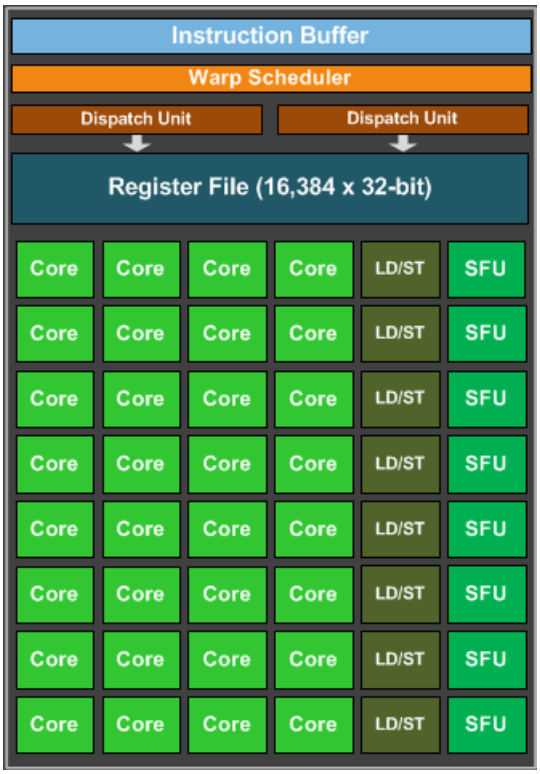
Nem kerülhetjük meg továbbá azt a kérdést sem, hogy hogyan valósul meg a szálak futtatása a GPU-ban. Elméletileg minden szál egymáshoz képest aszinkron módon fut, és a



2.1. ábra: *A GPU memóriahierarchiája [25]*

`__syncthreads()` függvényhívással lehet az egy blokkon belüli szálakat szinkronizálni, amelynek hatására a hívás helyén a szálak bevárják egymást. A kernelfüggvényen belüli szinkronizációk számát minimalizálni kell, mert ez akadályozza a műveletvégző egységek hatékony kihasználását később részletezett okokból. Tudnunk kell, hogy a tényleges realizáció során nem szálakat futtat a GPU, hanem 32 szálból álló *szálkötegeket* (*warp*), amelyeken belül minden szinkron történik. Felmerül a kérdés viszont, hogy miként kerül kiszolgálásra az egy warpon belül lévő 32 szál, amikor memóriaműveletet kezdeményeznek. A warpos szervezés végett a közös memória *32 bankra* lett felosztva, amelyek 4 bájtonként váltják egymást az egymást követő memóriacímek mentén. Ha az egy warpon belül lévő minden szál más bankhoz fordul, vagyis például egymást követő címekről olvasnak `float` típusú változókat, akkor ütközés nélkül, egy ciklus leforgása alatt megkapják a kért adatokat. *Bankütközés* esetén viszont annyiszor zajlik le a ciklus, ahányszoros az ütközés, legrosszabb esetben tehát 32 ciklust kell kivárni. A globális memóriához történő hozzáféréseknél az az előnyös, ha a szálak egymást követő címekre hivatkoznak, ugyanis ilyenkor a memóriavezérlő 128 vagy akár 256 bájtos kérést is tud generálni, a véletlenszerű hozzáférés viszont jelentős teljesítménybeli visszaeséshez vezethet.

A GPU legfelsőbb szintű építőblokkja a *Streaming Multiprocessor* (*SMM* a CC5.2-es Maxwell architektúra esetén). Egy SMM 4 darab, a 2.1-es ábrán látható warpütemezőből és a hozzá kapcsolódó 32 darab ún. CUDA-magból, 8 load/store egységből, és 8 SFU-ból (Special Function Unit; szinusz, koszinusz, négyzetgyök és egyéb függvények számítására szolgáló egység) áll, emellett pedig egy 16K méretű, 32 bites regiszterfájlon osztozhatnak az egységhez rendelt warpok. Az SMM-hez tartozik még 96 KB közös memória, illetve az L1 gyorsítótár. A különböző képességű GPU-változatok különböző számú SMM-et



2.2. ábra: *Az SMM egy ütemezője és a hozzá tartozó erőforrások [26]*

tartalmaznak, illetve eltérő a memória sávszélessége is. Amikor tehát egy kernelfüggvényt meghívunk, akkor minden blokkja, illetve a blokkokon belül minden warp egy SMM-hez, azon belül pedig egy ütemezőhöz kerül hozzárendelésre. Az egy ütemező által kezelt warpok száma attól függ, hogy a fordítás során keletkezett kód mekkora regiszterigényt támaszt. Például, ha minden szál a maximálisan hozzárendelhető 255 regisztert használja, akkor mindössze 2 warp kerülhet egy ütemezőhöz. Ugyanilyen korlátozó tényező a blokkok közösmemória-igénye, amely megszabja, hogy egy SMM-en hány blokk lehet jelen egyidejűleg. Az egy blokkhoz tartozó összes warp szükségképpen mindig egyazon SMM-en belül ütemezhető, hiszen másképp nem férhetnének hozzá a közös memóriához.

Fontos hangsúlyozni, hogy a műveletvégző egységek erősen pipeline-osítottak, és sok művelet elvégzése 15-20 ciklust is igénybe vehet. Tulajdonképp ennek a *késleltetésnek az elfedése* végett kell törekednünk arra, hogy egy blokk legalább 256 vagy még ennél is több szálat futtasson, hiszen ily módon amíg egy warp például adatfüggőség miatt egy adott utasítás eredményére vár, addig sok más warpból ütemezhető a következő utasítás. Mivel a regiszterfájl statikusan lett szétosztva a warpok között a kernel indításakor, ezért nincs szükség kontextusváltásnál a regiszterek mentésére, és ez teszi lehetővé a hatékony működést. Amikor a `__syncthreads()` függvényhívással szinkronizációt kérünk, korlátozzuk az ütemezőt, hiszen amíg minden warp el nem ér a szinkronizációs pontig, addig egyre csak fogy az ütemezhető warpok száma.

Most, az alapvető kérdések tisztázása után módosíthatjuk a korábbi FIR-szűrő kódot a

hatékonyság jegyében. Az eredmény a a 2.5-ös listán olvasható.

```
template<int N, int threads>
__global__ void fir_gpu_opt( /* ... */ )    // optimalizált FIR-szűrő
{
    int offset = blockIdx.x * threads + threadIdx.x;
    __shared__ float d_slice[N+threads];    // átmeneti tár a közös memóriában

    #pragma unroll                                  // a szükséges adatok tárolása
    for(int i = 0; i < N / threads + 1; i++)    // a közös memóriában
        d_slice[i*threads + threadIdx.x] = data[i*threads + offset];

    __syncthreads();    // csak akkor lehet szűrni, ha már minden warp végzett

    float acc = 0.0f;

    #pragma unroll 8
    for(int i = 0; i < N; i++)  // egy kimeneti minta előállítása szálanként
        acc += d_slice[threadIdx.x + i] * coeffs[i];

    out[j] = acc;   // eredmény tárolása
}

void execute_fir( /* ... */ )
{
    int threads = 512;
    int blocks = L / threads;
    fir_gpu<N, threads><<< blocks, threads >>>(coeffs, data, out);
}
```

2.5. lista: *FIR-szűrés GPU-n (optimalizált változat)*

Az első szembetűnő különbség, hogy a szűrő hosszát és a blokkban lévő szálak számát nem argumentumként adtam át a kernelfüggvénynek, hanem *template* paraméterként. Célszerű ugyanis a fordítási időben ismert konstansokat beépíteni a kódba, amely így lehetőséget ad statikus tömbök létrehozására, illetve az iterációszám ismeretében `for` ciklusok kifejtésére. A függvény elején egy tömböt deklarálok a közös memóriában, amely a bemeneti adatok átmeneti táraként fog szolgálni. Az adatok beolvasása után szükséges egy `__syncthreads()` utasítás kiadása, mivel addig egyetlen warp sem haladhat tovább, amíg nem biztosított, hogy az összes szükséges adat be lett írva az átmeneti tárba. További teljesítményjavulás várható a `for` ciklusok teljes, vagy részleges kifejtésétől, hiszen a ciklusmag egyetlen *MAC* (*Multiply and Accumulate*) utasításból áll, amelyhez képest számottevő többletterhelést jelent a ciklusváltozó karbantartása és a feltételvizsgálat.

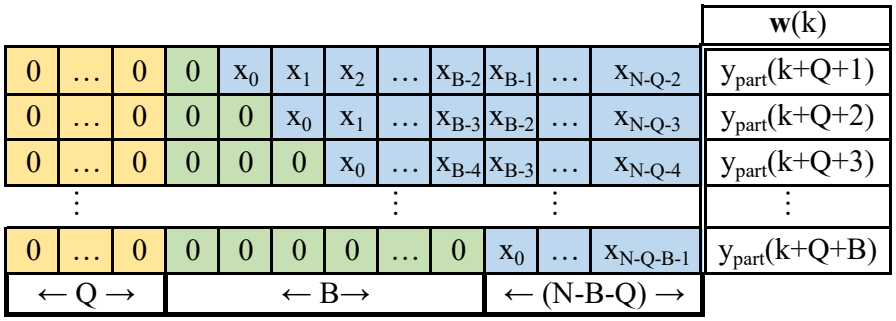
2.3 Az FeLMS algoritmus megvalósítása GPU-n

Egy valós idejű zajcsökkentő algoritmus megvalósításakor figyelembe kell vennünk, hogy a grafikus kártya általában egy PC-s környezetben férhető hozzá, ezért számolnunk kell azzal, hogy *késleltetés* és késleltetésingadozás (*jitter*) egyaránt fel fog lépni a működés során, ezekre pedig fel kell készülnünk. Legjobb tudomásom szerint a szakirodalomban jelenleg egyetlen GPU-n alapuló ANC-rendszer terve lelhető fel, amely Lorente és társai nevéhez fűződik [27, 28, 29]. Az általuk vázolt megoldás frekvenciatartománybeli MFxNLMS algoritmuson alapul, amely ortogonális korrekciós tényezőket és blokkos feldolgozást

alkalmaz. A kísérleteik során a zajforrást szerepét betöltő hangszóró közvetlen bemeneti jelét csatolták ki referenciajelként, ami a való életbeli alkalmazásnál általában nem áll rendelkezésünkre. A blokkos feldolgozás során fellépő késleltetés a legkisebb blokkméret esetén is több ezredmásodperc volt, a kauzalitás fenntartásához pedig ebben az esetben az RM-nek akár több méterrel közelebb kell elhelyezkednie a zajforráshoz az EM-hez képest, ami bizonyos esetekben irreális követelmény lehet. A szerzők továbbá nem tettek említést arról, hogyan kezelik az operációs rendszer jelenlétéből adódó jittert.

A blokkos feldolgozás valóban elengedhetetlen egy grafikus kártyán futó algoritmus esetén, hiszen ha egyetlen kimeneti minta számítását több szál között osztjuk szét, akkor a részeredményeket a procedúra végén egyazon akkumulátorhoz kéne hozzáadni egy-egy atomikus művelet keretében, az atomicitás miatt pedig ez csak sorosan történhet meg, vagyis minden szál egy külön ciklusban adhatná hozzá a saját részeredményét. Tudjuk azonban, hogy valójában mindig csak az egy warpon belüli szálak futnak ténylegesen párhuzamosan, ebből pedig következik, hogy ha az egy warpban lévő 32 szál mindegyike más kimeneti mintát számol, akkor hatékony lesz az implementáció. Célszerű tehát 32-t vagy annak egész számú többszörösét választani blokkméretként (B).

A másik problémakörbe a fellépő késleltetés és jitter tartozik, amelyeket orvosolnunk kell. A késleltetés javarészt a PC-s feldolgozás velejárója, hiszen a bemeneti adatfolyam pufferelt, az operációs rendszer és a CUDA keretrendszer használatából eredő jittert pedig további pufferelléssel lehet „tiszta”, determinisztikus késleltetéssé alakítani. A teljes jelút késleltetése így már több ezredmásodperc lenne, ezért némi átalakítást kell eszközölni az algoritmuson. A szükséges módosítások evidenssé válnak, ha áttekintjük, hogy tulajdonképp miben rejlik a probléma gyökere. A GPU egy nagy számítási kapacitású erőforrás, amelyhez azonban nagy késleltetés párosul, viszont a PC-ben található még egy közepes számítási teljesítményű, gyakorlatilag kihasználatlan CPU is, amelynek közepes a késleltetése, illetve amint majd látni fogjuk a későbbi fejezetekben, a ki- és bemeneti csatornák adatainak fogadására szánt FPGA vagy SoC még kisebb késleltetéssel fér hozzá a jelfolyamokhoz, és bizonyos feladatok elvégzésére még ezek is elegendőnek bizonyulhatnak. Ahogyan a számítógépben lévő háttértárat, RAM-ot és gyorsítótárat együttesen használjuk a tárterületek méretbeli és sebességbeli különbségeinek áthidalására, úgy most a különböző számítási teljesítményű és késleltetésű tartományokat is be kell vetnünk az optimális megoldáshoz, vagyis a szűrési feladatot fel kell bontani egy nagy számításigényű és egy kis késleltetést igénylő részre. Ha feltesszük, hogy az adott mintavételi frekvencián legfeljebb Q mintányi késleltetés léphet fel a PC-s feldolgozás során, akkor a GPU-val kiszámíttathatjuk egy jövőbeli kimeneti minta részeredményét a még nem ismert bemeneti mintákat nullákkal helyettesítve. A blokkos feldolgozás által okozott B mintányi késleltetés szintén kiküszöbölhető azáltal, hogy olyan kimeneti mintákat számoltatunk a grafikus processzorral, amelyekhez még nem áll rendelkezésünkre minden szükséges bemenet. Végeredményül egy olyan eljáráshoz



2.3. ábra: *Jövőbeli részleges kimeneti minták számítása* $\mathbf{x}(k)$ *birtokában*

jutunk, amelynél a blokk első részleges kimeneti mintáját $(Q + 1)$ darab, az utolsót pedig $(Q + B)$ darab szorzattal kell utólag kiegészíteni. Az 1.2-es pontban bevezetett jelölésekkel az y_{part} részleges kimeneti minták formálisan az

$$y_{part}(k + Q + m) = \mathbf{x}^T(k + Q + m)\mathbf{w}(k) \quad \Big| \quad \{x_i(k + Q + m) = 0 \quad : \quad 0 \leq i < Q + m\} \quad 0 < m \leq B \tag{2.1}$$

összefüggés szerint számíthatók (ezt szemlélteti a 2.3-as ábra), majd pedig a legkisebb késleltetésű hardverelemen a hiányzó minták megérkezésekor befejezzük a szűrést a hiányzó mintákra:

$$y(k + Q + m) = y_{part}(k + Q + m) + \sum_{i=0}^{Q+m} x_i(k + Q + m)w_i(k), \qquad 0 < m \leq B \tag{2.2}$$

A vázolt algoritmus viszont több ponton eltér a korábban bemutatott LMS-algoritmustól, ugyanis egyrészt az összefüggésekből is látszik, hogy a $(k + Q + 1)$-től a $(k + Q + B)$ ütemig a kimeneti minták számítása a k-adik ütemben érvényes szűrőegyütthatókkal történik, másfelől pedig a szűrőegyütthatók frissítése is praktikusan blokkokban történik, emiatt az algoritmus csak késleltetve kap visszacsatolást a hangolás eredményéről. Ez az 1.3.3-as pontban elemzett, DLMS-algoritmusra jellemző negatívumokat vonja maga után. Az MFeLMS-algoritmus ugyanakkor képes kiküszöbölni ezeket a hátrányokat, ezért a munka előrehaladtával indokolttá válik a használata, és így a GPU-kernelként történő megvalósítása. A rendszer teszteléséhez azonban az FeLMS-algoritmus korrigálatlan változata is elegendő.

Az általam implementált GPU-kód blokkonként 512 szállal, vagyis 16 warppal dolgozik, az egyszerűség kedvéért pedig 512 egész számú többszörösének megfelelő szűrőhosszakat támogat. Mivel a célarchitektúra a Maxwell (CC5.2), ezért az SMM-enkénti 4 warpütemező kihasználása érdekében a rendelkezésre álló 96 kB közös memóriát 4 blokk között kellett szétosztanom, ami blokkonként 24 kB-ra adódott, ez pedig 6K 32 bites `float` típusú változó tárolására elegendő. A kernelfüggvény, mint majd olvasható, a bemeneti mintákat a

feldolgozás előtt a közös memóriába tölti be, így annak limitált mivolta hosszabb szűrők esetén szükségessé teszi a több szeletben (*slice*) történő feldolgozást. Mivel az algoritmus meglehetősen memóriaintenzív, ezért a memóriaolvasások számának minimalizálása érdekében használtam a CC3.0 óta elérhető `__shfl()` intrinsic függvényt és variánsait, amelyek regiszterek közötti közvetlen adatcserét tesznek lehetővé a warpokon belül. A bemutatott kód összesen kb. 9 millió szűrőegyüttható valós idejű futtatását tette lehetővé egy középkategóriás, GTX 960-as videokártyán, ami több mint elegendő számunkra.

```
// FILTER_SLICES: ennyi szeletre kellett felosztani a szűrőt
// ROUNDS_PER_WARP: ennyi kört kell megcsinálnia minden warpnak ahhoz,
//      hogy az adott szelethez tartozó összes bemeneti minta fel legyen dolgozva
// TH_PER_WARP: 32, ennyi szál van egy warpban
// TH_PER_BLOCK: 512, ennyi szál van egy blokkban
// B: blokkméret, alapesetben 32
// Q: tartalék minták, alapesetben 64
// [...] : hiányzó kódrészletet jelöl

__global__ void felms(/* ... */)
{
    // thread azonosítója a warpon belül (0-31)
    const unsigned threadInWarp = threadIdx.x % TH_PER_WARP;

    __shared__ float x[SLICE_LENGTH];
    __shared__ float e[B];
    __shared__ float y[B];

    // [...] A nyers hibamintákból az A2 inverzével FIR-szűréssel állítjuk
    // elő a szűrt hibamintákat a korábban bemutatott FIR-szűrő kóddal,
    // ez kerül az "e" tömbbe.

    for (int slice = 0; slice < FILTER_SLICES; slice++)
    {
        // [...] A szelethez tartozó bemeneti minták betöltése a közös memóriába.

        __syncthreads();

        float w[ROUNDS_PER_WARP];   // a szűrőegyütthatók regiszterekbe kerülnek
        for (int i = 0; i < ROUNDS_PER_WARP; i++)
            w[i] = global->w[threadIdx.x + i * TH_PER_BLOCK + slice * SLICE_LENGTH];

        for (int i = 0; i < ROUNDS_PER_WARP; i++)
        {
            for (int j = 0; j < B / TH_PER_WARP; j++)
            {
                float er = e[j * TH_PER_WARP + threadInWarp];

                float xr1 = x[threadIdx.x + i * TH_PER_BLOCK];
                float xr2 = x[threadIdx.x + i * TH_PER_BLOCK + TH_PER_WARP];

                // Itt történik a szűrőegyütthatók korrekciója; a warpon belüli
                // szálak egymás között, a közös memóriát kihagyva cserélgetik
                // a hibaértékeket és a bemeneti mintákat shuffle utasításokkal.
                for (int k = 0; k < TH_PER_WARP; k++)
                {
                    float sel = __saturatef(TH_PER_WARP - (threadIdx.x + k));
                    float xa = sel * __shfl_down(xr1, k)
                                + (1.0f-sel) * __shfl_up(xr2, TH_PER_WARP - k);
                    float ea = __shfl(er, k);

                    w[i] += xa * ea;
                }
            }
        }

        for (int i = 0; i < ROUNDS_PER_WARP; i++)   // a szűrőegyütthatók visszaírása
            global->w[threadIdx.x + i * TH_PER_BLOCK + slice * SLICE_LENGTH] = w[i];

```

```
60          float yr[B / TH_PER_WARP];  // a kimeneti minták részeredményei
61                                      // szintén regiszterekben tárolva
62          for (int i = 0; i < B / TH_PER_WARP; i++)
63              yr[i] = 0.0f;
64          for (int i = 0; i < ROUNDS_PER_WARP; i++)
65          {
66              for (int j = 0; j < B / TH_PER_WARP; j++)
67              {
68                  // Itt a blokkmérettel és Q-val előrébb kell indexelni a bemeneti
69                  // minták tömbjét, hiszen jövőbeli kimeneti mintákat állítunk elő.
70                  float xr1 = x[B + Q + threadIdx.x + i * TH_PER_BLOCK];
71                  float xr2 = x[B + Q + threadIdx.x
72                              + i * TH_PER_BLOCK + TH_PER_WARP];
73
74                  // FIR-szűrés a közös memória kihagyásával, shuffle utasításokkal.
75                  for (int k = 0; k < TH_PER_WARP; k++)
76                  {
77                      float sel = __saturatef(TH_PER_WARP - (threadInWarp + k));
78                      float xa = sel * __shfl_down(xr1, k)
79                                     + (1.0f - sel) * __shfl_up(xr2, TH_PER_WARP - k);
80                      float wa = __shfl(w[i], k);
81
82                      yr[j] += wa * xa;
83                  }
84              }
85          }
86
87          // A szeletben keletkezett részeredményeket összegezni kell minden warpra.
88          for (int i = 0; i < B / TH_PER_WARP; i++)
89              atomicAdd(y + threadInWarp + i * TH_PER_WARP, yr[i]);
90
91          __syncthreads();    // A következő szelet elkezdése előtt szinkronizáció.
92      }
93
94      // Az összes szelet feldolgozása után a kimeneti minták tárolása.
95      if (threadIdx.x < B)
96          global->y[threadIdx.x] = y[threadIdx.x];
97  }
```

2.6. lista: *FeLMS-algoritmus GPU-n*

3

Rendszertervezés

3.1 Követelmények

Az előző fejezet végére bizonyossá vált, hogy egy GPU-alapú ANC-rendszernek lenne létjogosultsága. A késleltetések minimalizálása végett azonban nem közömbös, hogy miként csatlakoztatjuk a PC-hez a mikrofonokat, illetve a beavatkozó hangsugárzókat. Szoftveres oldalon *ASIO* (*Audio Stream Input/Output*) hanginterfész-protokoll használatával érhetnénk el a legkisebb késleltetést, viszont a kereskedelmi forgalomban kapható interfészek feltérképezése során azt találtam, hogy még ASIO-meghajtóprogrammal sem lehet garantált 3 ezredmásodperc alatti késleltetésre számítani [12]. Az algoritmus átalakításával, és az időkritikus feladatrész PC-n kívüli hardverelemre történő hárításával azonban enyhíthetők a követelmények a PC-n belül fellépő késleltetésekkel szemben, viszont továbbra is fontos, hogy a jelútba ékelődő effektív késleltetés ne legyen több 1 ezredmásodpercnél.

A legjobb hangminőséget kondenzátormikrofonok használatával érhetjük el, azonban ezek magas költsége, illetve fizikai kiterjedése miatt a rendszer elvesztené a praktikusságát, főképp sok referenciajel használata esetén. Az elmúlt évtized viszont kedvezett a *MEMS* (*mikroelektro-mechanikus*) eszközök fejlődésének, és a napjainkban már elérhetők olyan mikrofonok is, amelyek néhány köbmilliméteres térfogatuk és rendkívül kedvező áruk ellenére viszonylag alacsony, 30 dB_{SPL} körüli ekvivalens zajszinttel (*EIN*) és a hallható hangtartományban lineáris átvitellel rendelkeznek. További előnyük, hogy digitális kimenetű változatban is kaphatók, ami leegyszerűsíti a rendszerhez történő illesztésüket. A szakirodalomban már fellelhető olyan publikáció, amelyben sikeresen alkalmaztak MEMS-mikrofonokat ANC-rendszerekben [30], ezért úgy döntöttem, az általam fejlesztett rendszerben is megvizsgálom az alkalmazhatóságukat. Elképzeléseink szerint 8-16 darab MEMS-mikrofonos modul lenne felfűzve egy kb. 20 méter hosszú, lineáris buszvezetékre.

Viszonyítási alapként kondenzátormikrofonokkal is tesztelni szeretnénk a rendszert, amihez 8 darab *vonalszintű* analóg bemenet szükséges. Habár jelen dolgozat a több referenciajelet felhasználó ANC-rendszerekre összpontosít, a későbbiekben nem kizárt több EM és beavatkozó együttes használata sem, emiatt a rendszert 8 darab vonalszintű analóg kimenettel is szándékozom ellátni. A tervezés során 100-110 dB_{SPL} körüli maximálisan előforduló hangnyomásértéket vettem alapul, és tekintve, hogy a felhasznált kondenzátormikrofonok EIN-je sem kevesebb 10 dB_{SPL}-nél, a 16 bites mintavételezés nyújtotta 98 dB-es dinamikatartomány minden esetben elegendő lesz. A rendszernek az 50 és 4000 Hz közötti frekvenciatartományban kell működnie, így egy nem túl meredek átlapolódásgátló szűrő mellett előreláthatólag 10 kHz-es mintavételi frekvencia elegendőnek ígérkezik.

3.2 A GPU-t tartalmazó PC-s környezet

3.2.1 Csatlakozás

A külső hardverelemek PC-hez történő illesztésére alkalmas interfész kiválasztása során számolnom kellett azzal, hogy a 16 darab 16 bites bemeneti csatorna 10 kHz-es mintavételi frekvencia mellett 2,5 Mb/s-os, a 8 darab szintén 16 bites kimeneti csatorna pedig 1,25 Mb/s-os adatátviteli sebességet követel meg, továbbá az elosztott számítás miatt a legfrissebb $(Q + B)$ mintához tartozó szűrőegyüttható kat is el kell küldeni B mintánként, ami `float` adattípus, $B = 32$ és $Q = 64$ mellett további 8 Mb/s kimenő sávszélességet igényel. Továbbra is fontos szempont a kis késleltetés elérése (hiszen az előbb feltételezett Q érték 6,4 ezredmásodpercnek felelt meg), viszont az elsődleges döntési kritérium az volt, hogy belátható időn belül működőképes rendszert kell alkotnom, ezért csak olyan interfészek jöhettek szóba, amelyekre viszonylag egyszerű a fejlesztés, és szinte minden PC-n hozzáférhetők.

Ezeknek a követelményeknek leginkább a *gigabites Ethernet* hálózati csatlakozó interfész, valamint az *USB* (*Universal Serial Bus*) interfész felel meg. A rendszer fejlesztésének kezdetén az USB 2.0-s verziója volt széleskörűen elterjedt és hozzáférhető. Ez fizikailag egy *half-duplex*, differenciális, bitsoros buszt használ, amelynek elméleti adatátviteli sebessége *high speed* módban 480 Mb/s. Az adatátviteli protokoll 125 µs-onként teszi lehetővé átvitel kezdeményezését, ami egy alsó korlátot szab a minimális késleltetésre. A fejlesztéshez hozzátartozik az eszközillesztő (*driver*) fejlesztése is, ami nagyban hozzájárult ahhoz, hogy végül nem az USB-interfészt választottam.

A gigabites Ethernet alatt az IEEE 802.3ab szabványban definiált, 5, 5e vagy 6-os kategóriájú *UTP* (*unshielded twisted pair*) kábelen alapuló, RJ45-csatlakozófelülettel rendelkező hálózati interfészt értem. Lévén, hogy ez *full-duplex* átvitelt valósít meg, az USB 2.0-val szemben bármikor képes átvitelt kezdeményezni. Az említett szabvány az *OSI-modell* (*Open Systems Interconnection*) számos rétegét megvalósítja, ezért nincs szükség eszközmeghajtók írására. Lehetőségünk van „nyers”, csak az adatkapcsolati réteget megvalósító Ethernet-csomagok küldésére és fogadására, amelyek az elérhető legkisebb késleltetéssel kecsegtetnek, de az *IP* (*Internet Protocol*) hálózati réteget és a *TCP* (*Transmission Control Protocol*) szállítási réteget megvalósítva már bizonyos kényelmi szolgáltatások is elérhetők, ami megbízható, kapcsolatorientált kommunikációt tesz lehetővé. A különböző hálózati és szállítási rétegek kezelése általában része az operációs rendszer szolgáltatásainak, illetve annak hiányában támaszkodhatunk külső programkönyvtárakra. A jóval egyszerűbb fejleszthetőség miatt az Ethernet-interfész használata mellett döntöttem.

3.2.2 Operációs rendszer

A tervezése során szükségesnek láttam az Ethernet-interfészen keresztül elérhető késleltetés meghatározását mérésekkel. Ehhez egy *Digilent Spartan 3-E Starter Kit* FPGA fejlesztői kártyát használtam, amely ugyan gigabites helyett csak 100 Mb/s-os linkkel rendelkezik, azonban a két változat közötti késleltetéskülönbség µs-os nagyságrendű, ezáltal pedig egy felső becsléshez juthatunk. Az FPGA-n egy MicroBlaze alapú, lágymagos (*soft-core*), egyprocesszoros rendszert alakítottam ki, amelynek rendszerbuszához egy Ethernet-vezérlőt és egy időzítőt (*timer*) illesztettem. A processzoron futó kódot ezután C nyelven implementáltam. Az alkalmazás egy végtelen ciklusban egyedi azonosítókkal ellátott, nyers Ethernet-csomagokat küldött a PC felé, ezzel egy időben elindítva a timert. Ezután a PC, illetve az azon futó fogadó alkalmazás a megérkezett Ethernet-csomagokat visszaküldte az FPGA kártyának. A visszaérkezés pillanatában a timer állásából kiszámítható az órajel ismeretében az ún. *round-trip time* (*RTT*), vagyis az oda-vissza út késleltetése, amelyben az operációs rendszer, a PC-s eszközillesztők, illetve a visszaküldő alkalmazás késleltetése is benne van.

A méréseket először *Windows 7 x64* operációs rendszer alatt végeztem, amely nem valós idejű operációs rendszer. A visszaküldést végző alkalmazást a *WinPcap* programkönyvtárra alapoztam, amely lehetővé teszi az Ethernet-interfészen fogadott csomagok gyors elérését. A mért átlagos késleltetés 100 µs körül alakult, az operációs rendszer preempciója miatt azonban a *worst-case* késleltetés a 15 ezredmásodpercet is meghaladta még a legmagasabb, „valós idejű” alkalmazásprioritás beállítása mellett is. Az ennek megfelelő Q érték beállítása MFeLMS algoritmus esetén nem jelentene problémát, viszont a teszteléshez írt GPU-s változat nem tartalmazza a késleltetés kompenzálását, ezért más, ténylegesen valós idejű ütemezést lehetővé tevő operációs rendszert kellett keresnem a feladatra.

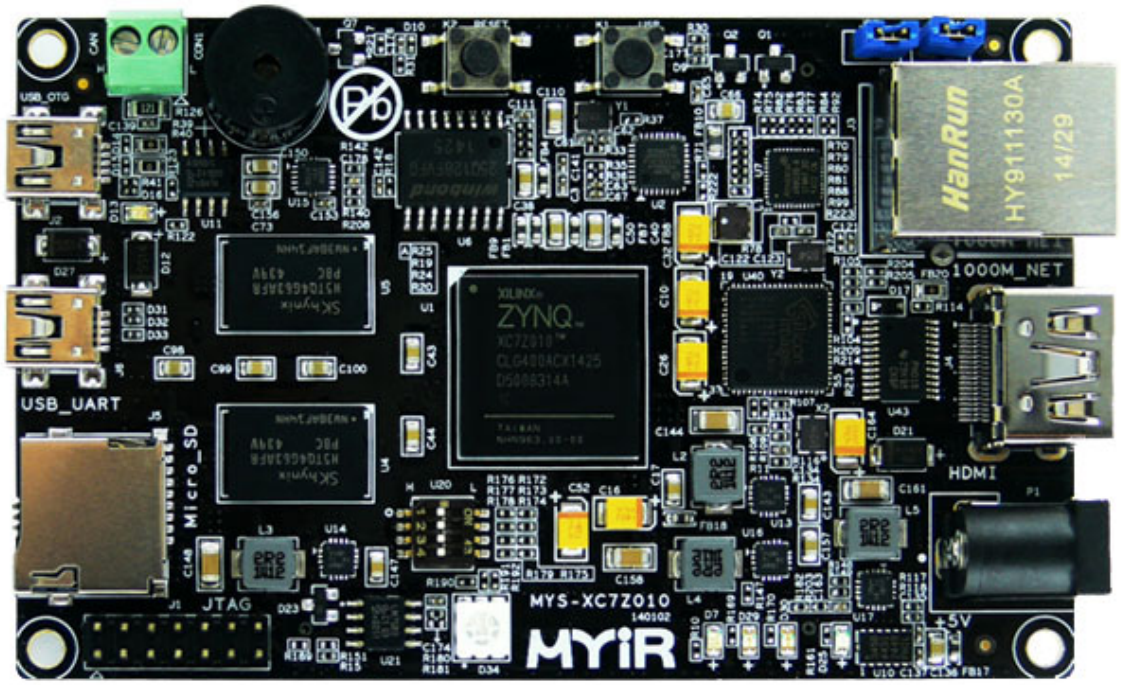
A Linux alapú operációs rendszerek rendelkeznek valós idejű ütemezővel, ezért a következő mérést az Ubuntu 16.04 disztribúció alatt végeztem el. Az erre írt csomagvisszaküldő-alkalmazásnál kihasználtam, hogy elérhetővé tehetők a felhasználói programok számára az eszközmeghajtók által használt cirkuláris pufferek memóriaterületei, ezáltal csökkentve a szükséges rendszerhívások és memóriamásolások számát (*zero-copy memory*). Az alkalmazás emellett az operációs rendszer valós idejű ütemezőjét használja. A mért átlagos oda-vissza késleltetés 50 µs körül alakult, és több percnyi mérés után, worst-case esetben is 80 µs alatt maradt. Az eredmények alapján kijelenthető, hogy az Ubuntu operációs rendszer viselkedése jobban kézben tartható, ami nagy előnyt jelent az élesztési és tesztelési fázisban, ezért a fejlesztés során ezt az operációs rendszert használtam. Természetesen ez nem zárja ki az alkalmazás későbbi, Windows rendszerre történő portolását.

3.3 Központi elem

A rendszer központi elemével szemben az alábbi elvárásokat támaszthatjuk:

- rendelkezzen gigabites Ethernet-interfésszel
- a számítási teljesítménye tegye lehetővé a részleges kimeneti minták kiegészítését
- legyen alkalmas 8 analóg be- és kimeneti csatorna, valamint a MEMS-mikrofonos modulok felől érkező jelek fogadására

A rendszer flexibilitását és a későbbi fejleszthetőségét is szem előtt tartva végül egy *Xilinx Zynq-7020 SoC* (*System on Chip*) alapú megoldást választottam egy *Myir MYS-7Z020-C* fejlesztői csomag keretében. A Zynq-7020 FPGA-tartománya a Xilinx Artix-7-nek felel meg, a benne található 85 ezer logikai cella pedig bármilyen illesztési feladatra alkalmassá teszi, ami az AD- és DA-átalakítók, valamint a MEMS-mikrofonos modulok buszillesztésénél tehet majd jó szolgálatot. Emellett rendelkezik 220 darab DSP-szelettel is, amelyek 18 és 25 bites egész vagy fixpontos számok szorzására, illetőleg MAC-művelet végrehajtására alkalmasak, így például az AD- és DA-átalakítók interpoláló és decimáló szűrőihez használhatjuk fel, de a részleges kimeneti minták kiegészítésénél a szűrés befejezésére is igénybe vehetők. A fejlesztői kártyán helyett kapott egy gigabites Ethernet-csatlakozó is. Az FPGA-tartomány mellett két darab *ARM Cortex-A9*-es mag áll a rendelkezésünkre, így köztük szétoszthatók a jelfeldolgozási és a kommunikációs feladatok. A fedélzeten található 1 GB méretű DDR3 RAM előreláthatólag minden feladatra elegendő lesz.

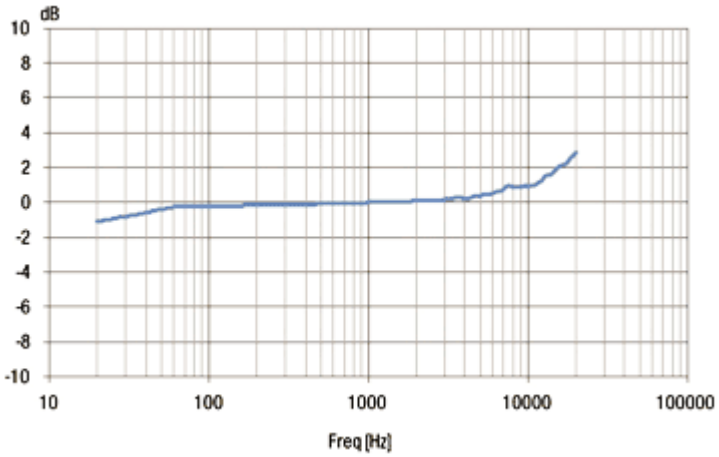


3.1. ábra: *Myir fejlesztőkártya Zynq-7020-as SoC-vel*

3.4 MEMS-mikrofonos modulok

3.4.1 A mikrofon típusának kiválasztása

Mivel az ANC-rendszerek épp a kis frekvenciájú zajok elnyomására nyújtanak hatékony megoldást, ezért fontos, hogy a felhasznált RM-ek és EM-ek frekvenciamenetében kis frekvenciák esetén se jelenjen meg nagy csillapítás, számszerűen az 50 Hz-es alsó határfrekvenciáig ne legyen 6 dB-nél nagyobb eltérés az 1 kHz-nél mért értékhez képest. A MEMS-mikrofonok esetében az alsó határfrekvenciát az akusztikai nyílás (*acoustic port*) keresztmetszete és a hátsó kamra geometriája befolyásolja [31]. A kereskedelmi forgalomban kapható típusok hasonló méretekkel és geometriával rendelkeznek, viszont az *ST Microeletronics MP34DT04* termékén 0,4 mm átmérőjű nyílás található, szemben az elterjedt 0,25-0,3 mm-rel. Sajnos az adatlapok többségében csak 100 Hz-től látható az eszközök frekvenciamenete, azonban az összehasonlítások során ennek a típusnak volt a legkisebb a csillapítása 100 Hz-en, így pedig várhatóan a 100 Hz alatti viselkedése is hasonló lesz. Egy nem hivatalos mérési eredmény szerint a szinte azonos paraméterekkel rendelkező elől, vagyis az *MP34DT01* 20 Hz-nél is csak 1 dB eltérést mutatott.

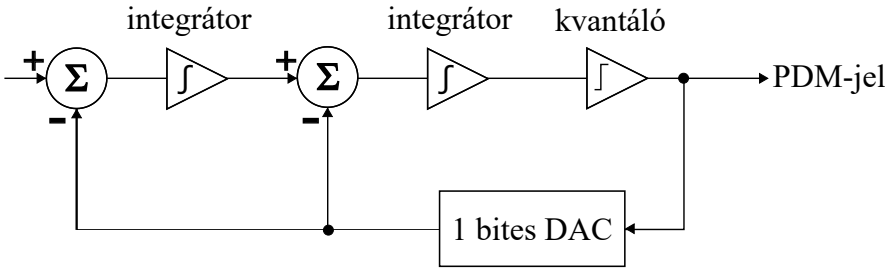


3.2. ábra: *Az MP34DT01 frekvenciamenete [32]*

Az MP34DT04 mikrofon közepesen jónak mondható, 65 dB_A-es jel-zaj viszonyt tud nyújtani 94 dB-es, 1 kHz-es mérőjel mellett, vagyis az ekvivalens zajszintje 29 dB_{SPL}. Előnye, hogy soros, digitális *PDM* (*Pulse Density Modulation*) kimenettel rendelkezik, tehát az AD-átalakítás már a tokon belül megtörténik. A teljes harmonikus torzítás és a zaj együttesen *(THD+N)* 1% alatt marad a számunkra releváns frekvenciatartományban egészen 100 dB_{SPL}-ig, tehát ekkora hangnyomásértékek mellett még remekül használható. Az adatlap ajánlása szerint 2,4 MHz körüli órajellel célszerű üzemeltetni a mikrofont, amely megegyezik a kimeneti adatfolyam bitsebességével is.

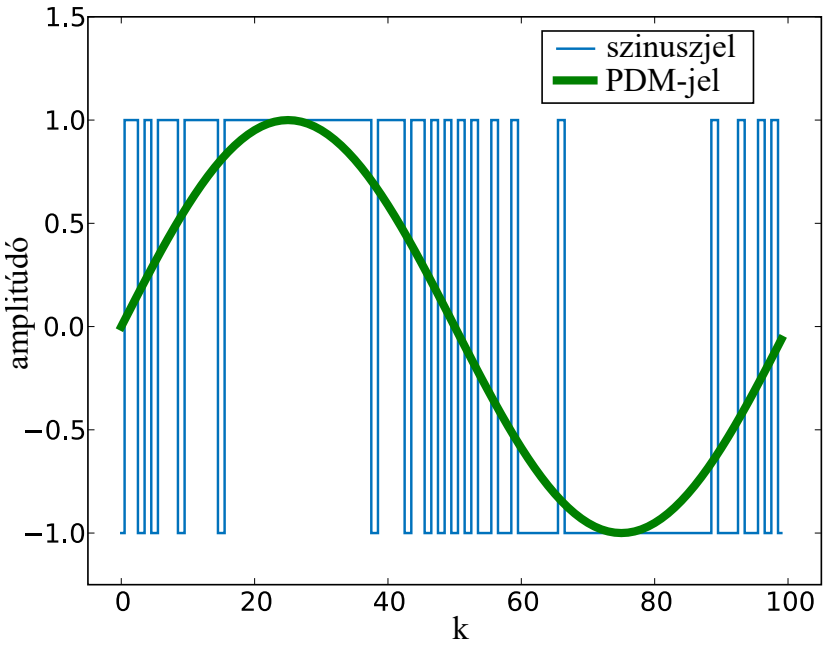
3.4.2 A PDM-jel feldolgozása

A digitális MEMS mikrofonok kimenete egy impulzussűrűség-modulált jel, amely tulajdonképpen egy ΣΔ AD-átalakító belső, nagyfrekvenciás, egybites jele. A 3.3-as ábrán látható



3.3. ábra: *PDM-jel egy másodrendű ΣΔ-modulátorból [33]*

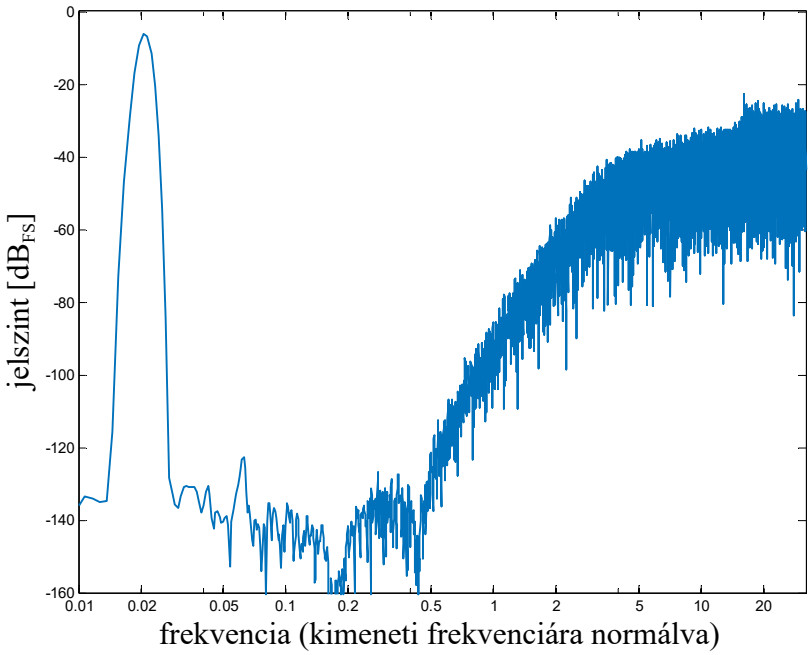
egy példa, hogy hogyan áll elő a PDM-jel egy másodrendű ΣΔ-modulátor esetében.[1] Az átalakítás befejezéséhez decimáló szűrésre van szükség, miáltal előállnak a kimeneti minták. A PDM-jel esetén a hasznos információt az impulzusok sűrűsége hordozza (3.4. ábra). Egy



3.4. ábra: *Szinuszjel és az annak megfelelő PDM-jel [34]*

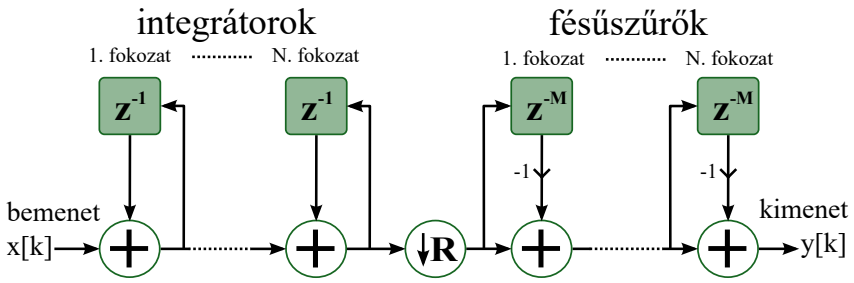
szinuszos mérőjel PDM megfelelőjének a spektrumát (3.5. ábra) megvizsgálva látható, hogy a ΣΔ-modulátor zajformáló szűrőként funkcionál, és a kvantálási zajt a hallható tartomány fölé transzponálja. Egy ilyen kimenettel rendelkező eszköz esetén a felhasználó feladata az AD-átalakítás utolsó szakaszának, vagyis a digitális szűrésnek a megvalósítása, ahol levágásra kerül a nagyfrekvenciás zaj, és egyben kisebb mintavételi frekvenciára decimálva nagyobb bitszélességű adatfolyamot kapunk.

[1] A MEMS-mikrofonoknál ennél bonyolultabb, negyed vagy ötödrendű struktúrát alkalmaznak.



3.5. ábra: *Példa egy szinuszból képzett PDM-jel spektrumára [35]*

A PDM-jeleket két lépésben célszerű a végleges kimeneti frekvenciára átalakítani. Elsőkörben egy *CIC-szűrőt* (*Cascaded Integrator-Comb*) alkalmazhatunk, amely egy meglehetősen hatékony interpoláló/decimáló szűrőstruktúra, lévén, hogy alkalmas kizárólag összeadások és kivonások elvégzésével a mintavételi frekvencia akár nagyságrendekkel történő megváltoztatására [36]. A 3.6-os ábrán látható a struktúra decimáló változata.



3.6. ábra: *Decimáló CIC-szűrőstruktúra [37]*

A szűrő átviteli függvénye (a nagyfrekvenciás oldalról nézve) egy fésűs aluláteresztő karakterisztikát takar:

$$H(z) = \left(\frac{1 - z^{-RM}}{1 - z^{-1}} \right)^N \tag{3.1}$$

Az M paraméter a differenciális késleltetést szabja meg, amelyet jelen alkalmazásnál célszerű egynek választani, az R a decimálási ráta és az N az integráló-decimáló fokozatok száma. Az összefüggést tanulmányozva észrevehető, hogy egy geometriai sorösszegről van szó, és ez valójában egy szimmetrikus FIR-szűrő, amely eredendően lineáris fázismenettel

rendelkezik. A DC erősítést a

$$H(z = 1) = (RM)^N \tag{3.2}$$

összefüggés írja le. Ennek ismeretében meghatározható a szűrő mentén használt regiszterek szószélessége, amelyet legalább akkorára kell választani, hogy a kimeneti regiszterben túlcsordulás nélkül reprezentálható legyen a teljes dinamikatartomány. A szükséges bitszámot (B) a következőképp határozhatjuk meg:

$$B = 1 + \lceil N + \log_2(RM) \rceil \tag{3.3}$$

A CIC-szűrő átviteli függvényébe a $z = e^{j2\pi \frac{f}{fs}}$ kifejezést behelyettesítve, némi átalakítás és abszolútérték-képzés után megkapjuk az amplitúdókarakterisztikát:

$$|H(f)| = (RM)^N \left| sinc\left(\pi RM \frac{f}{f_s}\right) \right|^N \tag{3.4}$$

A függvény zérushelyei az $f = kf_s/(RM)$ frekvenciáknál vannak ($k \in \mathbb{Z}$). Ebből tehát láthatjuk, hogy az első leszívási zóna a decimálás utáni mintavételi frekvenciával esik egybe, emiatt pedig elkerülhetetlen az átlapolódás, másfelől pedig a $sinc^N$ függvény révén már jóval a Nyquist-frekvencia előtt jelentős csillapítás jelentkezik az átvitelben. A megoldást az fogja jelenteni, hogy a CIC-szűrővel csak egy köztes, átmeneti mintavételi frekvenciára decimálunk, amelynél az átviteli függvény a számunkra értékes frekvenciatartományban még kis csillapítással rendelkezik, illetve az átlapolódó tartományok csillapítása már megfelelő, majd innen egy kis késleltetésű FIR decimáló szűrővel térünk át a végleges mintavételi frekvenciára.

3.4.3 Mikrokontroller választása

A mikrofonos modulokon a tervek szerint egy mikrokontroller (μC) fogja ellátni a PDM-jel fogadását, feldolgozását, valamint a központi egység felé való továbbítást. A MEMS-mikrofonok digitális kimenete a bemenő órajellel szinkron, egy konfigurációs láb beállításától függően a felfutó vagy a lefutó éleknél mintavételezhető. A jel fogadására általában az I^2S soros audiointerfész ajánlott [38], annak híján azonban egy közönséges SPI-interfész is alkalmas a feladatra, ezáltal a µC-ek jóval nagyobb választéka tárul elénk.

A bemeneti adatfolyam 2,4 Mb/s-os sebessége nagy terhelést ró a hardverre, ugyanis minden egyes végrehajtott utasítás 2,4 MIPS (*Million Instruction Per Second*) számítási teljesítményt követel meg, amennyiben egy órajel alatt végrehajtható utasításról van szó. A dedikált SPI-interfész nagy segítséget jelent az adatok fogadásánál, ugyanis a bitenként történő beléptetés helyett már csak 8-16 bit széles szeletekkel kell dolgoznunk. Feltételezve, hogy egy belépőszintű, *ARM Cortex-M0* magú mikrovezérlővel megoldható a feladat,

adható egy előzetes becslés a számításigényre vonatkozóan. A CIC-szűrő integrátorait mindenképp érdemes gépi utasításokkal, *assembly* betétként implementálni a hatékonyság jegyében. A 3.1-es listán látható egy négy fokozatú struktúrához ($N = 4$) tartozó, 16 bit feldolgozására alkalmas kódrészlet, amely tehát bitenként 5 utasítást végez, ezáltal pedig összesen kb. 12 MIPS erőforrásigényt támaszt a hardverrel szemben.

```
1 .rept 16          // ismétlés egy 16 bites szó minden bitjére
2 add %[itgr3], %[itgr2]  // integrátorok futtatása
3 add %[itgr2], %[itgr1]
4 add %[itgr1], %[itgr0]
5 lsrs %[value], #1       // egy bit kiléptetése, amely így a carry flagbe (C) kerül
6 adcs %[itgr0], %[zero]  // az első integrátorhoz a (C+0) hozzáadása
7 .endr             // ismétlés vége
```

3.1. lista: *Assembly-betét a CIC-szűrő integrátoraihoz ($N = 4$)*

Célszerű a decimálási rátát $R = 96$-ra választani, ezáltal 6 darab 16 bites szó feldolgozása után kell csak lefuttatni a fésűszűrő fokozatokat, miáltal egy köztes, 25 kHz-es mintavételi frekvenciához jutunk. A differenciátorok és egy 10-15 tap-es decimáló FIR-szűrő működtetéséhez ezen a frekvencián már hozzávetőleg 2-3 MIPS is elegendő.

A kommunikációs feladatok ellátására szintén egy SPI-egységet lesz célszerű bevetni, ezzel tehermentesítve a µC-t. Összességében tehát 20 MIPS számítási teljesítmény és 2 darab SPI-periféria a követelmény. A választásom az *ST Microelectronics STM32F030* szériájú, 32 bites µC-ének LQFP-48 tokkal ellátott verziójára esett. A legtöbb utasítást egy órajel alatt végrehajtja, és emellett legfeljebb 48 MHz lehet a magórajel, amely így 48 MIPS-es számítási kapacitást jelent. A 64 kB flash memória és a 8 kB RAM szintén elegendőnek ígérkezik minden kitűzött feladatra. Az SPI-vezérlők legfeljebb 18 Mb/s-os átviteli sebességig használhatók, amelyet figyelembe kell majd venni a magórajel kiválasztásánál, ugyanis az SPI-perifériák órajele abból csak 2 egész kitevőjű hatványaival történő leosztással származtatható.

3.5 Buszrendszer a MEMS-mikrofonos modulokhoz

3.5.1 Adatátvitel

A mikrofonos modulok a követelmények alapján egy kb. 20 méter hosszú, lineáris buszon kapcsolódnak a központi egységhez. A kommunikációhoz a mikrokontroller SPI-egységét lenne kényelmes használni, amely két adatvezetékkel (*MOSI* és *MISO*), egy órajellel (*SCK*) és egy opcionális kiválasztójellel (*CS*) rendelkezik. A buszvezetékek számát minimalizálni szeretnénk, emiatt egy *időosztásos* (*TDM*; *Time-Division Multiplexing*), *multi-master* konfigurációt képzeltem el a kommunikáció megvalósításához, amelyben az egyes modulok körforgó (*Round-Robin*) módon kapják meg a busz használati jogát a mintavételi frekvencia minden egyes ütemében. Ennek előnye, hogy az SPI-vonalakból csak a MOSI (a továbbiakban *DATA*) és SCK vezetékekre van szükség, a CS vezetékek

(és a MISO) pedig elhagyhatók. Szükség van még továbbá egy közös alapórajelre (*CLK*) ahhoz, hogy a modulok egymáshoz képest szinkron tudjanak működni. A szinkronizációról bővebben a 3.5.2-es pontban lesz szó.

A fizikai megvalósításnál törekedni kell az adatintegritás megőrzésére, ezért *RS-485*-ös, differenciális érpáron történő átvitel megvalósítását tervezem a DATA, az SCK és a CLK vonalon egyaránt. A 16 modul 16 bites minták esetén, 10 kHz-es mintavételi frekvenciát feltételezve legalább 2,6 Mb/s-os adatátviteli sebességet igényel. A *Texas Instruments SN65HVD75* RS-485 transceiver-áramköre 20 MBaud/s-os átviteli sebességet kínál, vagyis legfeljebb 10 MHz-es frekvencián üzemelhet, ezáltal alkalmas a feladatra, és emellett még kellően nagy tartalékkal is gazdálkodhatunk. A maximális megengedett kábelhossz 30 méter, amihez legfeljebb 5%-os jitter tartozik. Az elképzelt adatátviteli procedúra az alábbi:

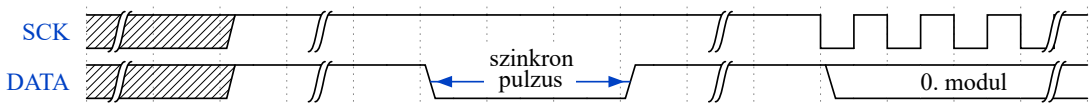
(1) A rendszer indulásakor, vagyis a tápfeszültség megjelenése és reset után a modulok mindegyik buszvonalához tartozó meghajtója inaktív, a vevők viszont folyamatosan aktívak. A busz minden vonalán alapértelmezésben logikai magas érték van kinn meghajtás hiányában. Minden modul tisztában van a buszon lévő összes egység darabszámával, és mindegyiknek egyedi sorszáma is van.

(2) A modulok figyelik, hogy mikor jelenik meg egy negatív impulzus felfutóéle a DATA vonalon, amely az első kommunikációs ciklust hivatott indítani. Ezt az impulzust a központi egység adja ki, és azért van rá szükség, hogy a rendszer éledése után a µC-ben lévő *PLL-egységnek* (*Phase Locked Loop*) maradjon ideje ráhangolódni a külső órajelre, még mielőtt megkezdődik a kommunikáció. Az impulzus megérkezésekor indul a mikrofon jelének feldolgozása és a kimeneti minták előállítása.

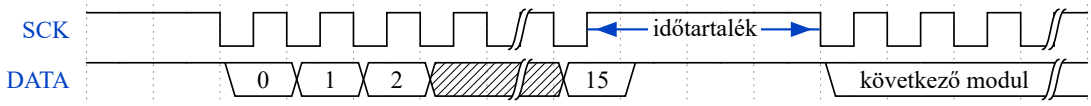
(3) Amint kész a 0. azonosítójú modul a küldendő 16 bites mintával, bekapcsolja a meghajtóit, és megkezdi az SPI-szabvány szerint ($CPOL = 1$, $CPHA = 1$) a küldést. A többi modul ezalatt számolja az SCK vonalon bekövetkező felfutóéleket.

(4) A 0. azonosítójú modul az adás végeztével azonnal kikapcsolja a meghajtóit, közben az 1. azonosítójú modul már észleli az SCK-n bekövetkezett élváltások számából, hogy a busz vezérlési joga hozzá került, ezért 1-2 bitidő késleltetés után aktiválja a meghajtóit, és megkezdi az adást (3.8. ábra). A többi modul esetén hasonlóan történik a busz átadása.

(5) Az utolsó modul adása után ismét a 0. azonosítójú modul következik, és amint előállt a következő küldendő minta, újabb kommunikációs ciklus indul (3)-tól.



3.7. ábra: *A rendszer indulása*



3.8. ábra: *A busz átadása két modul között*

A vázolt kommunikációs protokoll kísérleti jellegű, a kialakításánál az egyszerű kivitelezhetőséget tartottam szem előtt. Egy külső zavarjel hatására könnyen megzavarodhat, sőt akár meg is rekedhet a kommunikációs ciklus. Ez ellen időkorlátokkal (*timeout*) lehetne védekezni egy olyan rendszer esetében, ahol a megbízhatóság is fontos. A központi egység ugyanis érzékelni tudja, ha egy modul nem kezd el adni időben, vagy nem fejezi be az adást, és ilyenkor kiadhatja a többi modul normális működéshez szükséges impulzussorozatot az SCK vonalon. Ha az adatintegritás megőrzése is célunk, akkor ugyanazt a 16 bites mintát egymás után kétszer vagy háromszor is elküldhetik a modulok egyazon mintavételi ütemben, hiszen a rendelkezésre álló sávszélesség erre lehetőséget ad. Külső zavarok jelenléte esetén fennállhat az a veszély is, hogy a csatornák a vevőoldalon „megkeverednek”, ugyanis a küldött adatok nem tartalmaznak információt a küldő modul azonosítójáról. Ezt a hibalehetőséget úgy háríthatjuk el, hogy minden mintavételi ütem végén elvégezzük a (2)-es pontban leírt szinkronizációt a központi egység bevonásával.

3.5.2 Szinkronizáció

Az ANC-rendszerekben használt algoritmusok megkövetelik, hogy az összes be- és kimeneti eszköz ugyanazon a mintavételi frekvencián, egymáshoz képest szinkron üzemeljen. A mikrofonos modulok esetében az egyik lehetőség, hogy minden modulhoz eljuttatunk egy alapórajelet, amelyet ezután a µC-ek saját, belső PLL-egysége szoroz fel, előállítva ezzel a működéshez szükséges magórajelet. A buszon terjesztett órajel alkalmazásában számos előny rejlik. A jelfeldolgozási láncban nincs szükség többletfeldolgozásra, illetve az algoritmusok módosítására, ezáltal pedig az előállított minták minősége is az elérhető legjobb lesz, a PLL ugyanis lassan követi az alapórajelet a fázisdetektora utáni aluláteresztő szűrő hatására, és így kevésbé érzékeny például az alapórajel jitterére. A megoldásnak azonban van egy negatívuma is, amely szerint az egész rendszer leállhat, ha a µC-ben lévő PLL-egység egy külső zavar vagy a túl nagy jitter miatt nem tudja tovább követni az alapórajelet. Sajnos a választott µC adatlapja nem specifikálja egyértelműen, hogy mekkora bizonytalanságot visel el a PLL-je a periódusidők ingadozásában, ezért alternatív

szinkronizációs megoldásokat is kerestem. A differenciális átvitel ugyanakkor jól véd mind a kapacitívan, mind pedig az induktívan becsatolódó zajok ellen, ezért valószínű, hogy az ismertetett megoldás működőképes lesz.

Egy másik lehetőség, hogy minden μC belső, rövid távon stabil órajelet biztosító RC-oszcillátorról, egymáshoz képest aszinkron működik, és a buszon terjesztett órajel éleit szinkronizációra használjuk. A szinkronpulzus beérkezésekor érvényes kimeneti mintát ilyenkor kizárólag interpolációval tudjuk előállítani, ez pedig több szempontból hátrányos. Az interpoláció minőségromlást okoz a jelfolyamban, amely leginkább harmonikus torzítás formájában jelentkezik. Ez minimalizálható, amennyiben jó minőségű *spline* interpolációt alkalmazunk, és legalább 5-6 mintára illesztjük a görbét. Ehhez viszont 2-3 jövőbeli mintára is szükség van, ami plusz késleltetést jelent, ugyanakkor a késleltetése mértéke felére-negyedére csökkenthető, ha az interpolációt még a decimáló FIR-szűrés előtt, a köztes mintavételi frekvencián végezzük el. A szinkronizációs órajel jittere fáziszajként közvetlenül megjelenik a jelben, vagyis az elektromos zavar hatása a hangminőség romlásában nyilvánul meg. Az egyedüli előnye ennek a megközelítésnek, hogy zavarok jelenléte esetén sincs veszélyeztetve a rendszer működése.

A harmadik alternatíva, hogy a legfeljebb 50 MBaud/s-os (tehát 25 MHz-es) sebességű *SN65HVD78* RS-485 transceiver-áramkört alkalmazzunk a busz órajelvonalának meghajtására, és közvetlenül erről működtetjük a μC-t, a PLL-egység kihagyásával. Ebben az esetben az alapórajelre csak a 40 és 60 százalék közötti kitöltési tényező a követelmény, amely szinte biztosan teljesül. Ez a megoldás közel ugyanazokat az előnyöket nyújtja, mint a PLL-t használó változat. Az órajel jittere ebben az esetben is zajként jelenik meg a mintákban, a rendszer működőképességét azonban nem befolyásolja.

A dolgozat írása közben még egy lehetőség felmerült, ugyanis kihasználhatjuk, hogy a mikrofontól érkező adatok fogadására használt SPI-interfész *slave* módban is működhet, így pedig tokon kívülről is kaphat órajelet. A busz alapórajelét így elegendő a MEMS-mikrofonhoz, illetve az SPI-egység SCK lábához vezetni, a μC pedig üzemelhet a belső RC-oszcillátorról. Az előző megoldáshoz képest ez annyival előnyösebb, hogy a magórajel tetszőlegesen állítható, illetve a buszon is csak egy viszonylag alacsony frekvenciájú órajelet kell továbbítani, amelyre az SN65HVD75 is elegendő.

	a buszórajel felhasználási módja			
	PLL-alapórajel	**szinkronizációs jel**	**μC-magórajel**	**mikrofonórajel**
extra feldolgozás	nincs	interpoláció	nincs	nincs
extra késleltetés	nincs	van	nincs	nincs
minőségromlás	minimális	zavar-/interpolációfüggő	zavarfüggő	zavarfüggő
zavar hatása	leállhat a rendszer	csökkenő SNR	csökkenő SNR	csökkenő SNR
buszórajel	1-4 MHz	10-100 kHz	≥ 25 MHz	1-3 MHz

3.1. táblázat: *A szinkronizációs lehetőségek összehasonlítása*

3.5.3 A busz fizikai megvalósítása és tápellátás

Az eddig tervezett buszvonalak 3 darab differenciális érpárt igényelnek, valamint a tápfeszültés és a föld elvezetéséhez még egy további érpár szükséges. Az Ethernet alapú adatátvitelnél is alkalmazott, 5-ös kategóriájú UTP-kábelek 4 darab csavart érpárral rendelkeznek, amelyek a TIA/EIA-568-as szabvány szerint legalább 100 MHz-es sávszélességet biztosítanak. Az érpárok karakterisztikus impedanciája 100 Ω, a reflexiók elkerülése végett ezért mindkét végükön 100-100 Ω-os ellenállással kell lezárni minden vonalat. Egyetlen tisztázatlan kérdés maradt, miszerint alkalmas-e egy darab csavart érpár a 20 méter hosszú buszon lévő 16 modul tápellátására.

Az SN65HVD75 transceiver-áramkörök 3,3 V-os tápellátást igényelnek, amely megfelelő az STM32F030 szériájú μC-eknek, illetve az MP34DT04 mikrofonok számára is. A buszt ellátó tápáramkörökkel kapcsolatos követelmények feltárásához meg kell határoznunk, hogy mekkora áramfelvétellel számolhatunk egy-egy modul esetében. A következő alkatrészek áramigényét kell figyelembe vennünk:

- μC: 48 MHz-es magórajel mellett, az összes perifériát engedélyezve legfeljebb 23 mA lehet az adatlap tanulsága szerint
- mikrofon: 0,7 mA legfeljebb
- 3 darab diagnosztikai LED: 10 mA összesen
- DATA, SCK és CLK transceiver: előbbi kettő kb. 80 mA-t igényel együttesen, utóbbi viszont csak vevőként funkcionál, ezért 1 mA-nél nem több az áramfelvétele

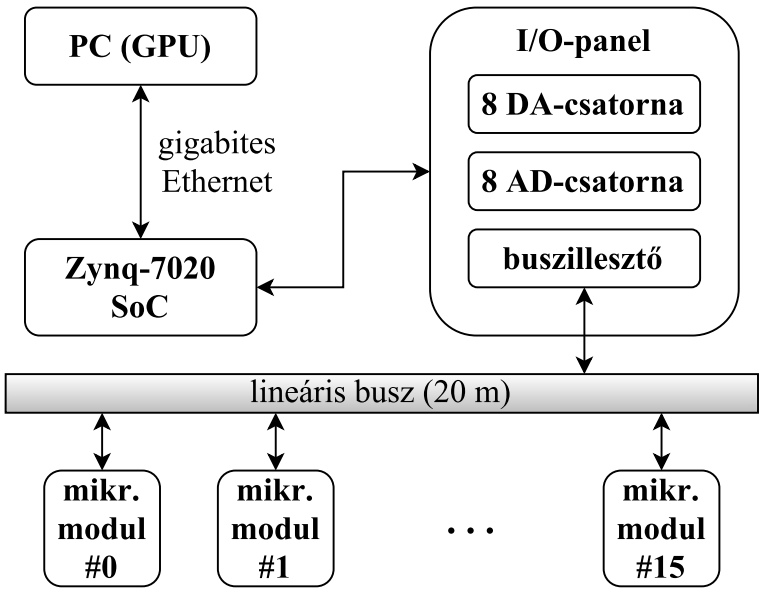
Az SN65HVD75-ös áramkörök adás közbeni fogyasztása abból ered, hogy a differenciális vonalak lezárására használt 100 Ω-os ellenállások eredője 50 Ω, és ezeken a meghajtóáramkör összesen kb. 40 mA-es áramot képes áthajtani, miközben a vezetékpár között 2 V körüli feszültségkülönbség jön létre. A buszt ellátó tápegység méretezésekor figyelembe kell venni, hogy a DATA és az SCK vonalakat egyidejűleg mindig csak egy modul hajtja meg. Célszerű lesz a buszt úgy kialakítani, hogy a központi egység középen helyezkedjen el, így a két irányban a 8-8 modul legfeljebb 10 méterre helyezkedik el, ezáltal pedig kisebb lesz a feszültségesés a tápvezetékeken. Mindent összegezve egy modul áramfelvétele legfeljebb 118 mA lehet, a busz egyik oldalára csatalakozó 8 modul esetén legfeljebb 374 mA-rel, mind a 16 modul esetén pedig összesen 667 mA-rel számolhatunk.

Egy átlagos minőségű, 5-ös kategóriájú UTP-kábelben lévő 0,2 mm^2 keresztmetszetű rézhuzal DC hurokellenállás nem több 0,2 Ω/m-nél, vagyis a busz egyik 10 méteres karjának végén összesen 2 Ω-on keresztül csatlakozhatunk a tápágakra. Azzal a közelítéssel élve, hogy mind a 8 modul a busz végén helyezkedik el, a rajtuk keresztülfolyó 374 mA áram kb. 750 mV feszültségesést okoz. A modulokon lévő alkatrészek megkövetelik a stabil tápfeszültséget, ezért a fedélzetükön helyet kell kapnia egy feszültségstabilizátor áramkörnek

(*LDO*, *Low Dropout Regulator*) is. A választásom a *Micrel MIC5504-3.3* típusra esett, amely legfeljebb 300 mA-t tud leadni, így a DATA és SCK vonalak aktív meghajtásakor is el tudja látni a feladatát. A maximális áramfelvétel mellett a rajta bekövetkező feszültségesés az adatlap szerint nem több 160 mV-nál. Összesen tehát legalább 4,2 V-ot kell biztosítania a központi egységnek a buszon lévő modulok számára, célszerű azonban egy kis tartalékot is hagyni a rendszerben, és felkerekíteni az egyébként is gyakrabban használatos 5 V-ra. A stabilizátor-IC tokjának hőellenállása 250 °C/W, legrosszabb esetben 1,8 V eshet rá, és az egy modulra jutó áramfelvétel időátlaga 42 mA, így tehát a disszipált teljesítmény 75 mW, amely legfeljebb 19 °C-kal emeli meg az eszköz hőmérsékletét, vagyis 25 °C-os szobahőmérséklet mellett maximum 44 °C-ra melegedhet fel, amely jócskán a megengedett működési tartományon belül van.

3.6 I/O-panel a központi egységhez

A Zynq-7020 SoC-vel szerelt Myir fejlesztőkártya hátoldalán 2 darab 80 csatlakozópontos hüvelysor található, amelyeken keresztül hozzáférhetünk az eszköz szabad kivezetéseihez. A feladatom egy olyan panel tervezése volt, amelyik képes 8 analóg, vonalszintű be- és kimeneti csatornát kezelni, valamint csatlakozni képes a MEMS-mikrofonok buszrendszeréhez, illetve el tudja látni azokat a működéshez szükséges tápfeszültséggel.



3.9. ábra: *A teljes rendszer blokkvázlata*

3.6.1 AD-átalakító

Az AD-átalakító kiválasztása során kompromisszumot kellett kötnöm az elérhető jelminőség és az átalakítás jelentette késleltetés között. Rendszerint a ΣΔ-családba tartozó

AD-átalakítók képesek a legjobb minőségű minták előállítására. Például a *Texas Instruments PCM4220*-as áramköre -108 dB-es THD+N értékkel kecsegtet, ugyanakkor a beállítható átlapolódásgátló szűrők közül a legkisebb csoportkésleltetéssel rendelkező változat is $21/f_s$ másodpercnyi, vagyis 10 kHz esetén 2,1 ezredmásodpercnek megfelelő késleltetést ékel a jelútba. A követelményekben foglaltak szerint a feladat megoldásához elegendő a 16 bites mintákkal elérhető 98 dB-es dinamikatartomány is, ezért olyan megoldást kerestem, ahol némi minőségbeli áldozatért cserébe a késleltetés értéke jobban kézben tartható.

A *Maxim MAX11046*-os IC egy *szukcesszív approximációs* (*SAR*), 16 bites AD-átalakító, amely 8 csatorna egyidejű mintavételezésére alkalmas. A bemenetek aszimmetrikusak, bipolárisak, és ±5 V közötti jelek fogadását teszik lehetővé. Az adatlap szerint az eszközt tipikusan 92 dB-es *SINAD* érték (*Signal to Noise and Distortion*, ami a THD+N-nel reciprok mennyiség) jellemzi. A legnagyobb elérhető mintavételi frekvencia 250 kHz. A ΣΔ-megvalósítással szemben az SAR típusú átalakítókon belül nincs átlapolódásgátló szűrő, vagyis ennek megvalósítása a felhasználó feladata.

Analóg áramköri elemekkel nehéz meredek levágású karakterisztikát kivitelezni, ezért az eszköz képességeit kihasználva túlmintavételezést kell alkalmazni. Minden bemenetre három, egymással sorba kapcsolt RC-tag fog kerülni, amelyek eredő törésponti frekvenciája *SPICE-szimulációk*[2] alapján 6 kHz, az eredő csillapítás pedig 245 kHz-nél 70 dB körüli érték. Ez azt jelenti, hogy az AD-átalakítást 250 kHz-en végezve a hasznos tartományra 70 dB-es csillapítással lapolódik át a bemeneti jelben feltehetőleg egyébként sem számottevő, 250 kHz körüli zaj. A végső mintavételi frekvenciára egy decimáló FIR-szűrővel lehet áttérni, amelynek paraméterei jól kézben tarthatók, és FPGA-ban implementálható.

Sajnos a követelményekben nem lett pontosan rögzítve, hogy a szóban forgó vonalszintek a professzionális (3,472 V_{pp}, +4 dB_u) vagy fogyasztói (0,894 V_{pp}, -10 dB_V) szinteket [39] jelentik, a tervezés során pedig a professzionális értékeket vettem alapul. A bemenetek 1 GΩ fölötti impedanciája ugyanis lehetővé teszi a bemeneti erősítők (puffererősítők) elhagyását az előbbi feltételezéssel élve. Így ugyan elveszítünk 1,5 bitet azáltal, hogy az AD-átalakító nincs teljesen kivezérelve, viszont ezt vissza is nyerjük a kb. 20-szoros túlmintavételezéssel. Ez a megoldás működőképesnek bizonyult professzionális jelszintek mellett, azonban utólag visszatekintve nem hagynám el a bemeneti puffereket, és kapcsolóval állíthatóvá tenném az erősítésüket, így mindkét jelszint mellett az elérhető legjobb minőséget kapnánk.

Az eszköz analóg részei szimpla 5 V-os tápellátást igényelnek, amin maximum 50 mA-es áramfelvétellel számolhatunk, míg a digitális tartomány kompatibilis a 3,3 V-os feszültségszinttel, és legfeljebb 7 mA-t igényel. Az áramkör belső órajelforrással dolgozik, így ezt már nem kell biztosítanunk a működéshez.

[2]Az analitikus megoldás egy harmadrendű RC-szűrő esetén meglehetősen hosszadalmas lett volna, ezért inkább szimulációval határoztam meg a pontos R és C értékeket.

3.6.2 DA-átalakító

A DA-átalakító kiválasztásánál az volt az elsődleges szempont, hogy az AD-átalakítóhoz hasonló paraméterekkel rendelkezzen, illetve lehetőleg egy tokon belül biztosítson 8 kimenetet. A választás az *Analog Devices AD5676R* típusára esett. Az áramkör 16 bites mintákat fogad, amelyeket egy szinkron, soros adatbuszon keresztül lehet beléptetni, és lehetőség van az összes kimenet egyszerre történő frissítésére. A várható SINAD 80 dB körüli, amely ugyan valamelyest gyengébb az AD-átalakítóéhoz képest, azonban az ANC-algoritmusok szimulációi alapján valószínűsíthető, hogy az elérhető zajelnyomást nem ez fogja korlátozni. A kimeneti feszültségszint választhatóan 0-2,5 V vagy 0-5 V, így tehát bármelyik vonalszinttel kompatibilis. Habár az ANC-rendszer működése szempontjából közömbös, a kimeneti spektrum ismétlődése miatt a beavatkozó hangszóró jelentős zajt bocsátana ki a tükörfrekvenciákon, ami ellen interpolációval védekezhetünk. A kimeneti értékek beállási ideje legrosszabb esetben 8 µs, így praktikusan 125 kHz-ig használható. Számunkra elegendő, ha a legkisebb tükörfrekvenciát a hallás felső határa fölött tudjuk tartani, amelyet ha 20 kHz-nek feltételezünk, akkor 3-szoros interpolációval biztosítható.

Lévén, hogy a kimenetek unipolárisak, egy felülátereresztő RC-tagot kell alkalmazni az ofszet leválasztására. Egy vonalszintű bemenet impedanciája legrosszabb esetben 10 kΩ lehet, ezért a DA-átalakító kimeneteire 7,5 kΩ – 22 µF tagok kerülnek, amelyek törésponti frekvenciája így 1 és 2 Hz közé esik a terheléstől függően. Az eszköz legfeljebb 15 mA-es áramot tud leadni a kimenetein, ezáltal elhagyhatók a kimeneti erősítők. Természetesen itt is egy univerzálisabb megoldást eredményezett volna, ha megtartom a kimeneti erősítőket, és lehetőséget biztosítok az erősítés beállítására a vonalszintnek megfelelően.

Az áramkör analóg oldalon 5 V-os tápfeszültséggel működőképes, terheletlen kimenetek mellett 2 mA-nél is kevesebb áramot igényel, de vonalszintű bemeneteket meghajtásakor is 10 mA-rel felülről becsülhető. A digitális logika 3,3 V-tal kompatibilis, és elhanyagolható az áramfelvétele.

3.6.3 RS-485 adó-vevők és tápellátás

A MEMS-mikrofonos modulok buszának DATA és SCK vonalához történő csatlakozás a moduloknál már ismertetett SN65HVD75 típusú transceiver-áramkörökkel történik, amelyek közül az előbbit adóként és vevőként egyaránt, utóbbit pedig csak vevőként kell használni. A CLK vonal illesztésére egy nagyobb jelzési sebességre hitelesített típus, az SN65HVD78 szolgál a 3.5.2-es pontban ismertetett összes eljárás megvalósíthatósága végett. Ezt a vonalat az FPGA kizárólag meghajtja. Legrosszabb esetben tehát két áramkör ad és egy vesz egyidejűleg, ami összesen kb. 82 mA áramfelvételt jelent a 3,3 V-os tápfeszültségről.

A Zynq-7020-as fejlesztőkártya a saját tápegységéről üzemel, így azt a számítások során

figyelmen kívül lehet hagyni. Az eddigiek alapján az I/O-panel tápegységének a következő feszültségtartományokat kell biztosítania:

- 5 V @ 670 mA a MEMS-mikrofonos moduloknak
- 5 V @ 60 mA a panelen lévő analóg részekhez
- 3,3 V @ 90 mA a panelen lévő digitális logikához

A rendszer számára egy külső tápegység állítja elő a hálózati feszültségből azt a névlegesen 6 V-os egyenfeszültséget, amelyből a panel tápegysége a felsorolt tápágakat származtatja. Túláram ellen egy 0,9 A-es regenerálódó biztosítóbetét védi az áramkört, a túlfeszültségvédelmet pedig egy záróirányban előfeszített, 6 V-os TVS-dióda látja el (SMBJ6.0A), amely kb. 7 V-os túlfeszültség fölött a túláramvédelem aktiválásával fejti ki a hatását, illetve fordított polaritású tápfeszültség ellen is véd.

A különböző tápágak stabilizátorai erről a védett forrásfeszültségről üzemelnek. A mikrofonos modulok és a 3,3 V-os LDO számára egy *Micrel MIC29301-5.0* típusú stabilizátor szolgáltatja az 5 V-os tápfeszültséget, amely névlegesen 3 A leadására is képes. Az adatlap szerint a tok hőellenállása nyomtatott huzalozású lemezre (*NYHL*) szerelve kisebb 56 °C/W-nál, így enyhe túlfeszültség esetén (7 V) a rajta eső 2 V és a rajta átfolyó 760 mA-es áram hatására 1,52 W-ot disszipál, amelynek eredményeképp 85 °C-kal melegszik fel, és 25 °C-os szobahőmérséklet esetén legfeljebb 110 °C lesz a hőmérséklete, így a 125 °C-os megengedett értéken belül marad. Hangsúlyozni kell, hogy ez csak egy worst-case érték, továbbá a tok és környezet közötti hőellenállás az NYHL gondos tervezésével ennél tovább is csökkenthető. A stabil működéshez legalább 10 mA-es átfolyó áram szükséges, amelyet egy 470 Ω-os terhelő ellenállás formájában lehet biztosítani. A kimeneti pufferkondenzátor mérete legalább 10 µF kell, hogy legyen.

Ebből a stabilizált 5 V-os feszültségből a mikrofonos moduloknál már bemutatott *MIC5504-3.3* típusú LDO állítja elő a panel digitális logikáinak működéséhez szükséges tápfeszültséget. Közvetlenül a 6 V-os forrásfeszültségről nem tud üzemelni, mert a megengedett legnagyobb bemeneti feszültsége 5,5 V. Az áramkör 300 mA leadására is képes, ezt kb. 220 mA-re korlátozza a tok hőellenállása és a maximális megengedett hőmérséklet, viszont a panelen csak 90 mA-re lesz szükség, ezért megfelel a célra. Már 1 µF-os kerámia kimeneti pufferkondenzátorral, minimális kimeneti áram nélkül is stabil.

Az AD- és DA-átalakítók analóg tartományai számára külön-külön egy *MIC5205-5.0* LDO állítja elő az 5 V-os tápfeszültséget a 6 V-os védett forrásfeszültségből. Az LDO-k előtt az AD-átalakító esetében egy másodfokú, a DA esetében pedig egy elsőfokú LC-szűrő választja le a nagyfrekvenciás zajokat. A törésponti frekvencia kb. 9 kHz-re lett beállítva. Az előszűrésre azért van szükség, mert az LDO adatlapjából kiderül, hogy az alsó, nagyjából 10 kHz-es frekvenciatartományba eső feszültségzajok elnyomására alkalmas, ezután kb. 40 dB/dekáddal romlik a hatékonysága. Az alkalmazott induktivitások valójában tekercsek,

emiatt parazitakapacitás is jelen van, és ezért használhatóságuk a rezonanciafrekvencia fölött korlátozott. A felhasználni kívánt 33 µH-s tekercs rezonanciafrekvenciája 22 MHz, efölött kapacitásként viselkedik. Mivel az AD-átalakító különösen érzékeny a zajokra, ezért hasznosnak ítéltem meg egy második LC-tag beiktatását, amelyben az 1 µH-s tekercs rezonanciafrekvenciája 210 MHz, ez pedig egy nagyságrenddel nagyobb az előző tagnál.

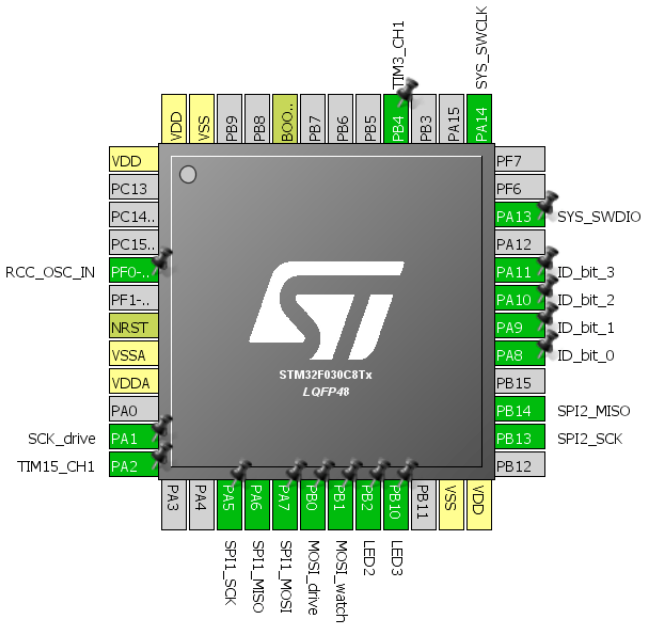
Mindhárom LDO-típus rendelkezik engedélyező bemenettel, amely hasznos segítség, hiszen így az I/O-panel és a busz tápellátását FPGA-ból, majd később PC-ről is lehet vezérelni. Ezzel szükség esetén újraindíthatók a buszon lévő mikrofonos modulok, ami jó szolgálatot tehet a fejlesztés során.

4 Áramköri és firmware-realizáció

4.1 MEMS-mikrofonos modulok

4.1.1 A µC és környezete

Az egyik elsődleges szempont a µC kiválasztásakor az volt, hogy tartalmazzon 2 darab SPI-egységet, amelyek közül az egyiket a MEMS-mikrofon illesztéséhez, a másikat pedig a kommunikáció lebonyolításához lehet alkalmazni. Az STM32F030 család C8-as tagja megfelel ennek a követelménynek, továbbá a számítási teljesítménye is elegendőnek ígérkezik annak fényében, hogy hardveres szorzóegység is található benne, amely egy órajelciklus alatt képes a műveletet elvégezni egész számokon. A kommunikációhoz szükséges még 2 időzítő egység is. Az egyik időzítő feladata az SCK vonalon bekövetkező felfutóélek számlálása, amely alapján mindig egyértelmű, hogy melyik azonosítójú modulnál van a busz használati joga, ezáltal lehetővé téve az időmultiplexelt kommunikációt. A másik időzítő szintén az SCK vonalra csatlakozik, de ennek az a feladata, hogy a magórajel ütemében mérje a legutóbbi felfutóél óta eltelt időt. Ennek segítségével észlelhetjük, ha *időtúllépés* történik, vagyis a buszon túl sokáig nem zajlik kommunikáció, és ekkor lehetőség van a µC feladatvégzésének leállítására, amely hibakeresésnél jöhet jól.



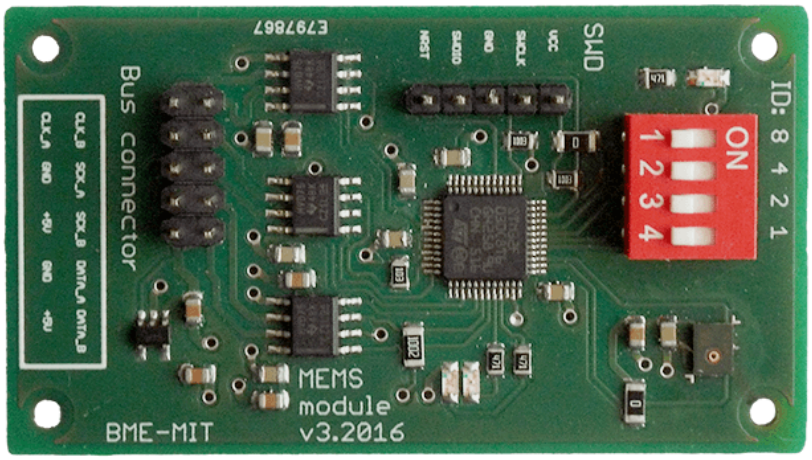
4.1. ábra: *A µC lábkiosztása*

Az *SCK_drive* és a *MOSI_drive* vonalak rendre az SCK és a DATA vonalak aktív meghajtását engedélyezik az RS-485 meghajtóknál. A master módban működő SPI1-egység

szolgál a kommunikáció lebonyolítására, amelynek *SPI1_MOSI* vezetéke a DATA vonal meghajtójához, az *SPI1_MISO* vezetéke a DATA vonal vevőjéhez, az *SPI1_SCK* vezetéke pedig az SCK vonal meghajtójához van kivezetve. A rendszer bekapcsolása után az adatátvitelt a központi egység felől érkező szinkron pulzus indítja, amiről egy megszakítás formájában célszerű értesíteni a firmware-t, így e célból a busz DATA vonalát figyelnünk kell egy külön lábon is (*MOSI_watch*). A szintén master módban működő SPI-interfész meghajtja a MEMS-mikrofont, és fogadja az általa küldött adatokat. A *TIM15_CH1* az időtúllépés-funkciót megvalósító időzítőre, a TIM3_CH1 pedig az élváltásokat számláló időzítőre vezeti a busz felől érkező SCK jelet.

4.1.2 NYHL tervezése

A µC esetében követelmény, hogy 100 nF-os kerámia táphidegítő kondenzátorokat helyezzünk el a táplábak közvetlen közelében, amelyek a nagyfrekvenciás áramok számára csökkentik a tápvonalak impedanciáját. A transceiverek esetén ugyanígy kellett eljárnom, amelyek közelében még további 10 µF-os kerámia kondenzátorokat is elhelyeztem, mivel buszvezetékeken a lezáró ellenállások miatt viszonylag nagy áramok haladnak át. A tápfeszültség meglétét egy piros LED jelzi. A panelen helyet kapott még egy 5 érintkezőből álló *SWD* (*Serial Wire Debugging*) csatlakozó is, amelyen keresztül a µC programozható, illetve hibakereséshez is használható. A buszhoz egy 2×5 pólusú tüskesoron keresztül van lehetőségünk csatlakozni. Az áramkörön helyet kapott egy 4 pozíciós DIP-kapcsolósor, amelyen a felhasználó által binárisan kódolható az adott modul azonosítója 0 és 15 között. A hibakeresésnél hasznos segítség, ha van néhány diagnosztikai célú LED is a panelen, ezért ebből is került 2 darab a µC mellé.



4.2. ábra: *Egy mikrofonos modul a beültetés után*

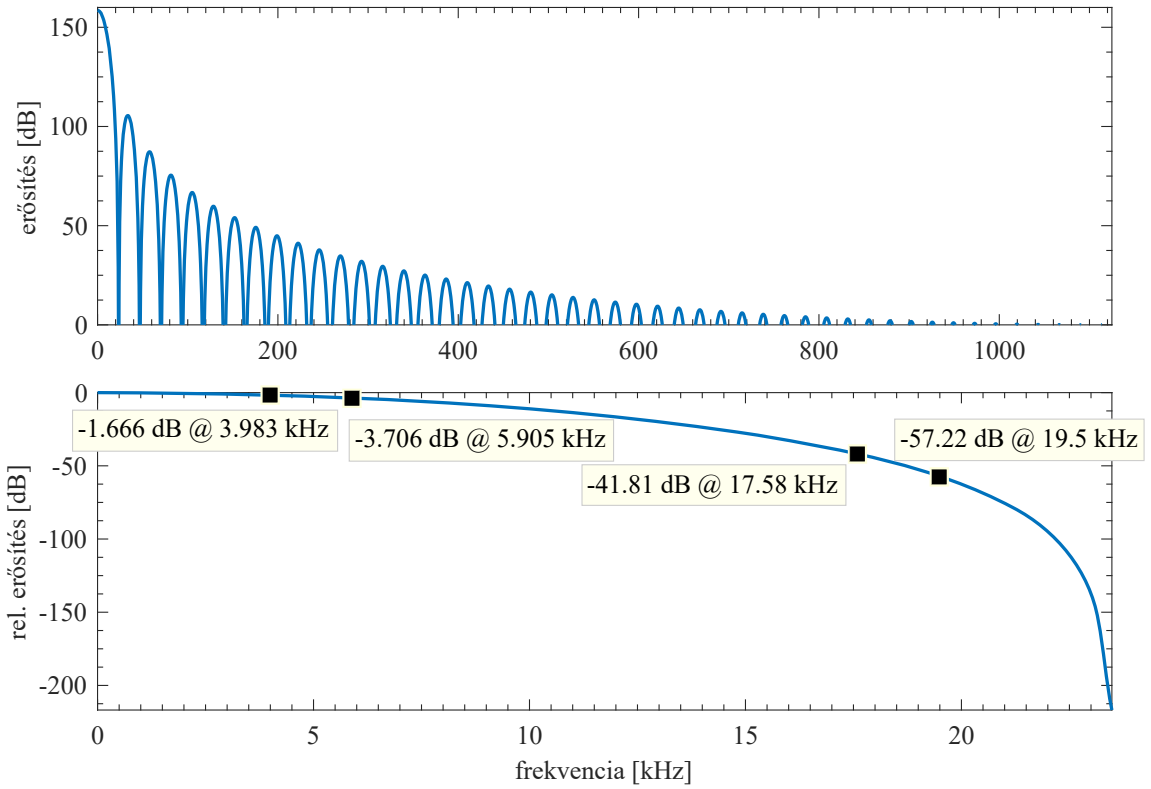
Az áramkör megvalósítása során törekedtem az alapvető *EMC* (*Electromagnetic Compatibility*) tervezési szabályok betartására, mint például az egyes adatvonalak közötti áthallás minimalizálása, és az alkatrészek számára jó minőségű föld és tápfeszültség biztosítása. A

kapcsolást sikerült egy kétrétegű panelen kiviteleznem, amelynek alsó rétege egy földréteg, így a felső rétegen vezetett adatvonalak által keltett áramok mindig a lehető legkisebb impedanciájú úton tudnak haladni. A felső réteg huzalozását egy tápfeszültségdomén öleli körül, amely váltakozó áramú szempontból szintén földként viselkedik, és valamelyest árnyékol a külső zavarok ellen. Az elkészült mikrofonos modulok kb. 64×36 mm-es lettek.

4.1.3 Mintavételi frekvencia választása és szűrőtervezés

A szinkronizációs lehetőségek közül az elsőként felmerült ötletet valósítottam meg. Ehhez a buszon 3 MHz-es órajelet terjesztettem a modulok felé a CLK vonalon, majd a µC-ek PLL-egységével ezt 12-vel felszorozva 36 MHz-es órajelhez jutottam. Ebből a többi periféria órajelét csak 2 egész kitevőjű hatványának megfelelő leosztással lehet előállítani, így a mikrofon számára az ajánlott 2,4 MHz-hez legközelebb eső 2,25 MHz-es értéket állítottam be, ami 16-tal történő osztásnak felel meg. Egyúttal a buszon történő kommunikációhoz használatos SPI-egység sebessége is eldőlt, a magórajelet 4-gyel leosztva ugyanis 9 MHz-es órajelet kapunk, amely közel van a buszmeghajtók maximális, 10 MHz-es megengedett sebességéhez.

A mikrofontól érkező jelet egy négyfokozatú CIC-szűrő decimálja az f_{cic} = 23.437,5 Hz-es átmeneti frekvenciára. Itt a 96-os osztásarányt az indokolja, hogy az SPI-egység

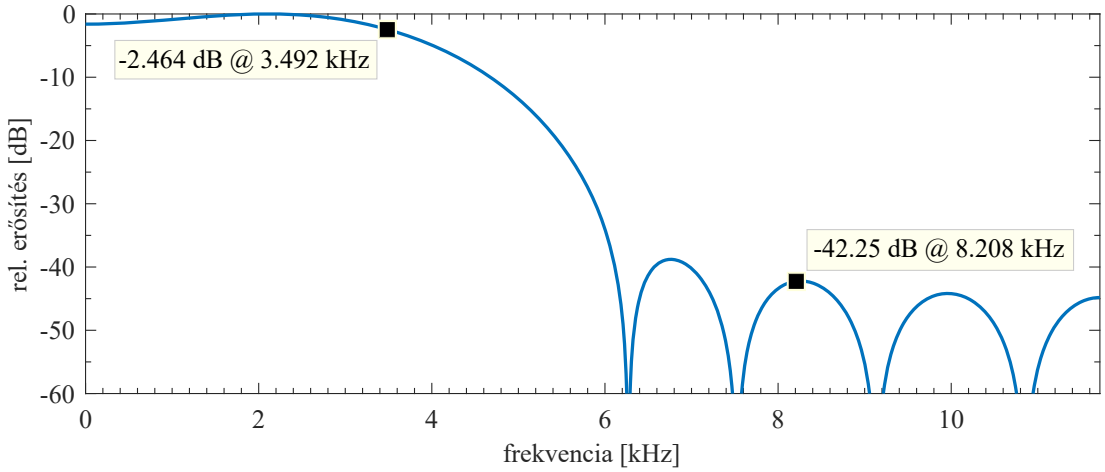


4.3. ábra: *A CIC-szűrő amplitúdómenete (alul nagyítva)*

legnagyobb vételi szóhossza 16 bit, a hatékony feldolgozás érdekében pedig célszerű volt ennek egész számú többszörösével dolgozni. A végső kimeneti mintavételi frekvencia így f_{ki} = 11.718,75 Hz lesz, amely nem sokkal több a tervezés során feltételezett 10 kHz-nél. A 4.3-as ábrán megtekinthető a CIC-szűrő amplitúdókarakterisztikája. Ez a spektrum eltolva is megjelenik a szűrő kimenetén a $k f_{cic}$ helyeken ($k \in \mathbb{Z}$), vagyis jól láthatóan lesz átlapolódás. A legnagyobb amplitúdóval az f_{cic} körülötti tartomány kerül átfedésbe a számunkra hasznos alsó 4 kHz-es szegmenssel. Az ábra alsó részében kinagyítva is látható a 0 és f_{cic} közötti szakasz. Az alsó 4 kHz-es tartomány felső határán 1,7 dB körüli a csillapítás a DC átvitelhez képest, az ezzel átfedésbe kerülő (f_{cic} − 4 kHz)-en pedig már 57 dB, vagyis legalább 55 dB-es jel-zaj viszonyra számíthatunk. Az f_{ki}-n értelmezett Nyquist-frekvencián már ennél valamivel kedvezőtlenebb a helyzet, mert a 3,7 dB-lel csillapított jelhez 41,8 dB-lel csillapított zaj adódik, amely már csak 38 dB-es különbség. Természetesen a későbbiekben ezen lehet módosítani, így például nagyobb f_{cic} választása esetén jobb minőségű mintákhoz juthatunk. A szűrő DC-erősítése a (3.2)-s összefüggés szerint 96^4, amely (3.3) alapján 27,34 bitet igényel, vagyis elfér a 32 bites regiszterekben. A CIC-szűrő csoportkésleltetése hozzávetőleg 85 µs.

A mikrofonokat 100 dB_{SPL}-ig szeretnénk használni, ehhez pedig legfeljebb −20 dB_{FS} kivezérlés tartozik (az adatlap szerinti érzékenység −26 dB_{FS} @ 94 dB_{SPL}), a bináris bemeneti értékek miatt keletkező $96^4/2$ ofszet levonása után tehát azt kapjuk, hogy a felső 3,34 bit kihasználatlan, így az alsó 8 bitet eldobva 16 bites mintákhoz jutunk.

A CIC-szűrőből érkező minták a késleltetés minimalizálása végett egy 11 tap-es, szimmetrikus decimáló FIR-szűrőn haladnak keresztül, amely már az f_{ki} kimeneti mintavételi frekvencián működik. A szűrőt *Matlab*-ban, *LS-módszerrel* (*Least Squares*) terveztem, majd 2^{13}-szoros erősítést állítottam be, végül egész értékű együtthatókat képeztem. A



4.4. ábra: *A FIR decimáló szűrő amplitúdókarakterisztikája*

kvantálás utáni relatív amplitúdómenet a 4.4-es ábrán követhető nyomon. Az átlapolódás 3,5 kHz környékén a legjelentősebb, hiszen csak 40 dB a hasznos jel és a vele fedésbe

kerülő zaj közötti különbség, viszont a tükörfrekvencián a CIC-szűrő is rendelkezik kb. 7,5 dB-es elnyomással, ami összességében már 47,5 dB-t jelent. A későbbiekben ez is módosítható az igényeknek megfelelően. A szűrő csoportkésleltetése 213 µs körüli. A szűrés során keletkezett érték alsó 13 bitjét eldobva kapjuk meg azt a 16 bites mintát, amelyet továbbítani lehet a központi egység felé.

4.1.4 Firmware fejlesztése

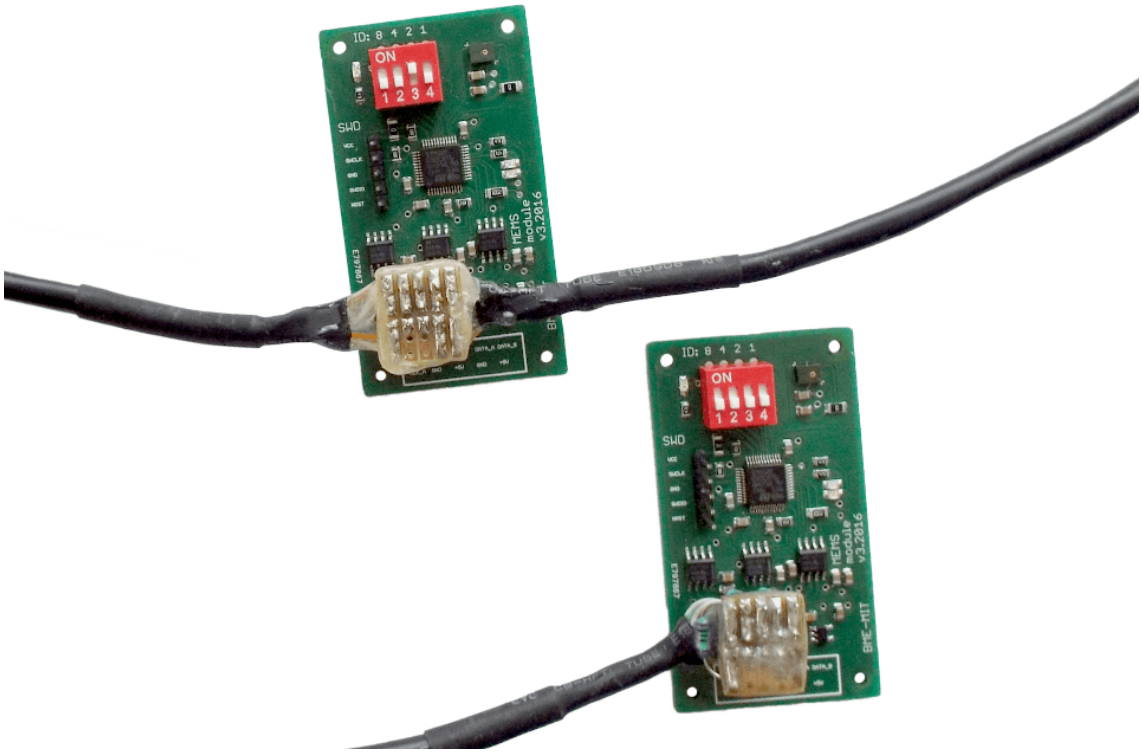
A firmware fejlesztését nagyban segítette az ST *CubeMX* névre keresztelt alkalmazása, amelynek grafikus felületén könnyedén tudtam konfigurálni a µC-t, ide értve a felhasznált perifériákat (SPI, TIMER), a lábkiosztást, továbbá az órajelhálózat (PLL, perifériák órajele) beállításait. Ebből C nyelvű kódot is generált a program, amelyet már csak a feladat érdemi megoldásával kellett kiegészítenem, ennek pszeudokódja pedig a 4.1-es listán olvasható. Az SPI-egységben csak 2 darab 16 bites adatcsomag tárolására van lehetőség, ezért a puffertúlcsordulás elkerülése végett gyakran ki kell olvasni a mikrofontól érkező adatokat. A jelenlegi implementáció a ciklusmagon belül számos ponton lekérdezi, hogy van-e új kiolvasásra váró csomag, és ha igen, akkor egy szoftveres FIFO-tárba helyezi a későbbi feldolgozásig. Ennél egy hatékonyabb megoldás lenne, ha a DMA-egységre bíznánk az adatok elvételét, amíg a feldolgozás zajlik.

```
belső egységek inicializálása, órajelforrás kiválasztása
ha a PLL-nek nem sikerült szinkronizálnia → ÁLLJ
modul azonosítójának beolvasása a DIP-kapcsolósor állásából

várakozás a buszillesztő mintavételt indító pulzusára

SPI2 engedélyezése a mikrofonhoz (megkezdődik az adat vétele)
SPI1 engedélyezése a kommunikációhoz (nincs küldendő adat, buszmeghajtó tiltva)
SCK élváltások számlálásának megkezdése
SCK timeout-időzítő indítása

[végtelen ciklus]
    [ismétlés 2-szer]
        [ismétlés 6-szor]
            ha rendelkezésre áll új adat SPI2-nél → elhelyezés a FIFO-tárban
            ha nem üres a FIFO-tár → következő feldolgozandó elem kivétele
                különben → várakozás új adat érkezéséig
            CIC-szűrő integrátorainak futtatása egy 16 bites szeletre
        [ismétlés idáig]
        CIC-szűrő differenciátorainak futtatása, köztes kimeneti minta előállítása
    [ismétlés idáig]
    FIR-szűrés, kimeneti minta előállítása

    [ismétlés, amíg nem kaptuk meg a hozzáférési jogot a buszhoz]
        ha van új adat SPI2-nél → tárolás FIFO-ban
    [ismétlés idáig]

    DATA és SCK vonalak meghajtásának engedélyezése
    küldendő minta beírása az SPI1-be (megkezdődik a küldés)
    várakozás a küldés befejezéséig
    DATA és SCK vonalak meghajtásának letiltása
[végtelen ciklus vége]
```

4.1. lista: *A µC firmware-jének pszeudokódja*

4.1.5 Élesztés, tesztelés

A mikrofonos modulokból a korábban tervezett 16 darab helyett csak 8 példányt gyártattunk, hiszen ennyi már elegendő lesz a rendszer teszteléséhez, ugyanakkor egy esetleges fatális tervezési hiba esetén kisebb a keletkezett anyagi kár. A modulokon lévő µC-ek programozását egy *STM32F411 discovery kit* segítségével végeztem el. A buszvezetéket egy 20 méter hosszú Ethernet-kábelből alakítottam ki, amelyet 2×5 pólusú hüvelysoros csatlakozókkal láttam el, a DATA, SCK és CLK vonalak érpárjait pedig mindkét végükön 100-100 Ω-os ellenállásokkal zártam le. Az élesztéshez szükséges buszillesztő áramkört az RS-485 transceiverekkel egy próbapanelen alakítottam ki, amelyet aztán az előbb említett discovery kittel vezéreltem.



4.5. ábra: *Mikrofonos modulok a buszra fűzve*

Sajnos a modulok és a buszrendszer kimerítő tesztelése az idő szűke miatt elmaradt, mindössze egy 1 kHz-es, szinuszos mérőjel mellett volt lehetőségem a fogadott jelalakokat összehasonlítani egy analóg mikrofon jelével. A mért harmonikus torzítások hibahatáron belül megegyeztek, és feltehetőleg a mérés során használt hangsugárzó torzítása dominált. A buszvonalakon lévő digitális jelalakok vizsgálatát mindenképpen érdemes lenne elvégezni a megoldás minősítéséhez. Mind a 16 modult felhasználva, 16 bites adatok és 11.718,75 Hz-es mintavételi frekvencia mellett 3 Mb/s a felhasznált sávszélesség, vagyis 66 %-os időtartalék van a rendszerben a buszátadások levezényléséhez. Ha a vizsgálatok során azt találnánk, hogy a buszjelek minősége nem kielégítő, úgy a jelzési sebességet meg lehetne felezni, miközben a fennmaradó időtartalék még mindig elegendő lenne a megfelelő működéshez.

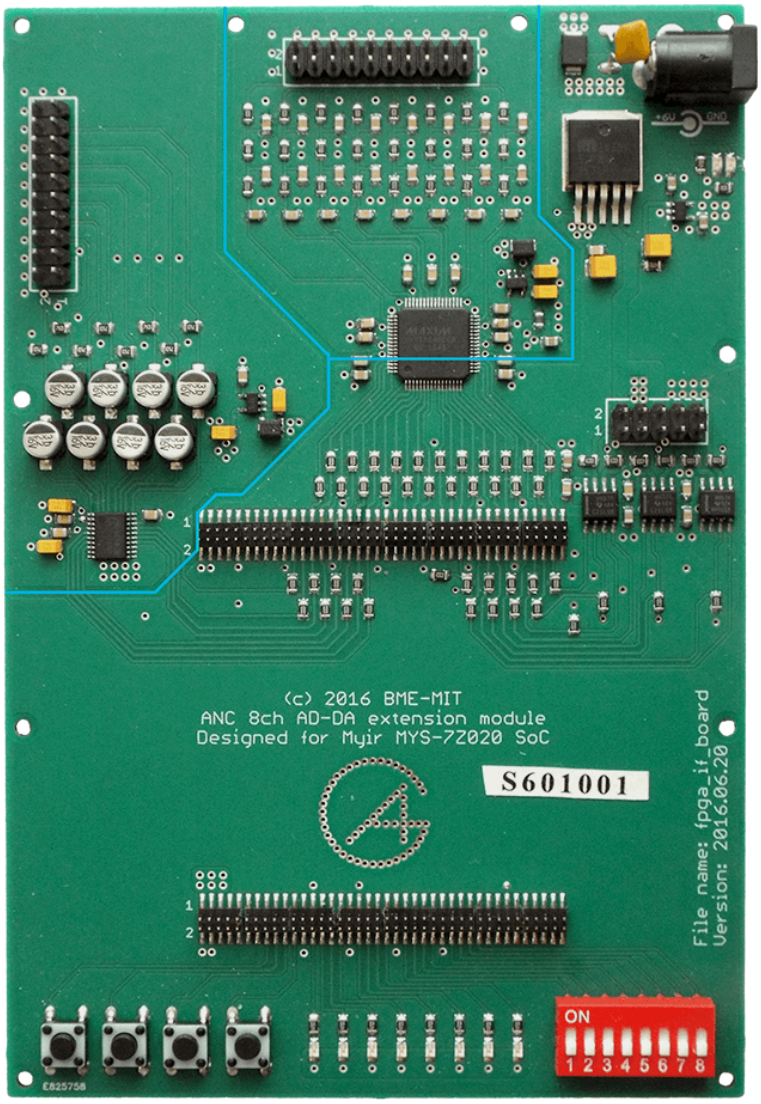
4.2 I/O-panel

4.2.1 Fizikai elrendezés, EMC-szempontok

Az analóg jelek és áramkörök jelenléte miatt különösen fontos mind az alkatrészek elhelyezése, mind pedig azok összehuzalozásának módja. Az analóg áramköröknek a lehető legtávolabb kell kerülniük a digitális vonalaktól és a zajos tápvonalaktól az áthallás csökkentése érdekében, amely tovább redukálható a huzalozási rétegek földkitöltésével, ráadásul ez még a melegedő alkatrészek hőelvezetésében is segít. A készülő áramkör 4 rétegű lesz, ezek közül a felső rétegen történik az alkatrészek összehuzalozása, az egyik belső réteg a 3,3 V-os tápfeszültséget, a másik belső réteg az 5 V-os analóg tápfeszültséget, az alsó réteg pedig a földet vezeti majd az a komponensekhez. Az utóbbi három réteg esetében evidensnek tűnhet, hogy teljes kitöltést alkalmazzunk, azonban az analóg területek alatt áthaladó tápvonalakra ülő zajok ilyenkor kapacitív módon csatolódnának, ami nem előnyös. Ennek elkerülése érdekében célszerű az AD- és DA-átalakítót, és azok analóg környezetét kikerülni a zajos 3,3 V-os és az 5 V-os tápréteggel egyaránt. Számolni kell továbbá azzal is, hogy a földrétegen visszafolyó áramok szintén zavart kelthetnek, ha éppen az analóg tartományok alatt haladnak át. Ennek elkerülése érdekében *csillagpontos földelést* alkalmaztam, amelynek eredményeképp az AD- és a DA-átalakító, valamint azok analóg környezetei egy-egy saját „földszigettel” rendelkeznek, a szigetek pedig egy-egy 0 Ω-os ellenálláson keresztül csatlakoznak a csillagpontban lévő földhöz, így ezeken keresztül tudnak folyni a kiegyenlítőáramok.

A Zynq-7020 fejlesztői kártya csatlakozóinak helyzete kötött, ezért a két 2×80 érintkezős, 1,27 mm-es raszterosztású tüskesor egymástól 53 mm-re helyezkedik el. A tervezés során egy olyan áramköri lapban gondolkodtam, amelynek szélessége megegyezik a Zynq-kártyáéval, és alul, illetve felül túllóg rajta magasságában. Az alsó túlnyúló részen helyezkedik el néhány kezelőszerv és diagnosztikai LED, a felső részen pedig a tápcsatlakozó, a buszcsatlakozó, továbbá az analóg csatlakozósorok. Az elkészült panel a 4.6-os ábrán tekinthető meg, ahol kék színnel kiemeltem a földszigetek határvonalait.

Az áramköri lap felső részének bal oldalán található a DA-átalakító áramköri részlete, amelynek 5 V-os stabilizátorát a zavarok minimalizálása érdekében a DA-átalakítóhoz közel, annak saját földszigete fölött helyeztem el. Középtájt található az AD-átalakító és a hozzá kapcsolódó komponensek, ahol jól megfigyelhető az is, hogy az IC kivezetései is gyakorlatilag két részre, egy analóg és egy digitális doménre tagolódnak. Az AD-átalakító belső referenciafeszültségének pufferkondenzátorai a chip alatt kialakított szigeten keresztül össze vannak kötve az adatlapi ajánlásának megfelelően. A táp- és földviák a lehető legközelebb kerültek az alkatrészlábakhoz, a 100 nF-os táphidegítő kondenzátorokkal egyetemben. A 10 µF-os hidegítő kondenzátorok közelsége másodlagos. Az 5 V-os tápellátáshoz szükséges stabilizátor és a kapcsolódó kiegészítő elemek az AD-átalakítóhoz



4.6. ábra: *Az elkészült I/O-panel (kék színnel a szigethatárok)*

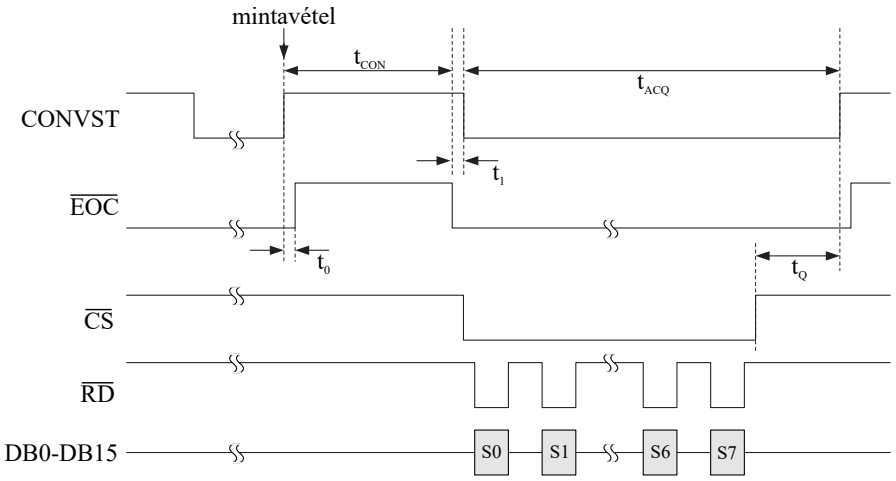
közel, saját földszigete felett kaptak helyet. A jobb felső sarokba került a tápcsatlakozó, a tápvédelem, a busz felé 5 V-ot biztosító LDO, a digitális logika számára 3,3 V-ot előállító LDO, valamint a bemeneti tápfeszültséget és az engedélyezést visszajelző LED-ek. Szintén a jobb oldalon, ezek alatt helyezkedik el a buszcsatlakozó és a hozzá tartozó RS-485 transceiverek. Utóbbiak esetében különösen fontos volt, hogy közel kerüljenek a buszcsatlakozóhoz, mert a tüskesortól a transceiverekig tartó vezetékszakaszok, mint hullámvezetők, lezáratlanok, és az ez által okozott reflexiók rontanak a buszjelek minőségén.

A kivitelezés során egyetlen apró hibát követtem el, nevezetesen, hogy az analóg jelutakba *X7R* kerámia dielektrikumú kondenzátorokat tettem *NP0* helyett, ezek kapacitása pedig erőteljesebben függ a rákapcsolt feszültségtől, amely egy nemlineáris viselkedést eredményez. A harmonikus torzítás minimalizálása végett érdemes lesz a későbbiekben lecserélni ezeket a kondenzátorokat.

4.2.2 Az AD-átalakító illesztése

Az AD-átalakító rendelkezik néhány beállítási lehetőséggel, amelyeket a CR0-CR3 lábakon keresztül tudunk beírni a konfigurációs regiszterébe. Egyetlen paramétert kellett az alapértelmezett konfigurációban átállítanom ahhoz, hogy az eszköz kettes komplemens adatábrázolással szolgáltassa a mintákat. Ez egy hagyományos, $\overline{\text{CS}}$ és $\overline{\text{WR}}$ jeleket alkalmazó regiszterírás, viszont az AD-átalakító CR lábai multiplexelve vannak a kimeneti minták alsó 4 adatvonalával (DB0-DB3), így az FPGA-ban ezek meghajtásához háromállapotú meghajtók szükségesek.

Az AD-átalakítót kezelő Verilog-modul tehát ezt a konfigurációt végzi el először, majd várakozik a gazdarendszertől érkező *start* jelzésre. Ha ez megtörtént, megkezdődik az adatok fogadása 20-szoros túlmintavételezéssel, vagyis 234.375 Hz-en. Az egy mintavételi ciklusra jellemző hullámformákat a 4.7-es ábra összegzi. A CONVST vonalat logikai

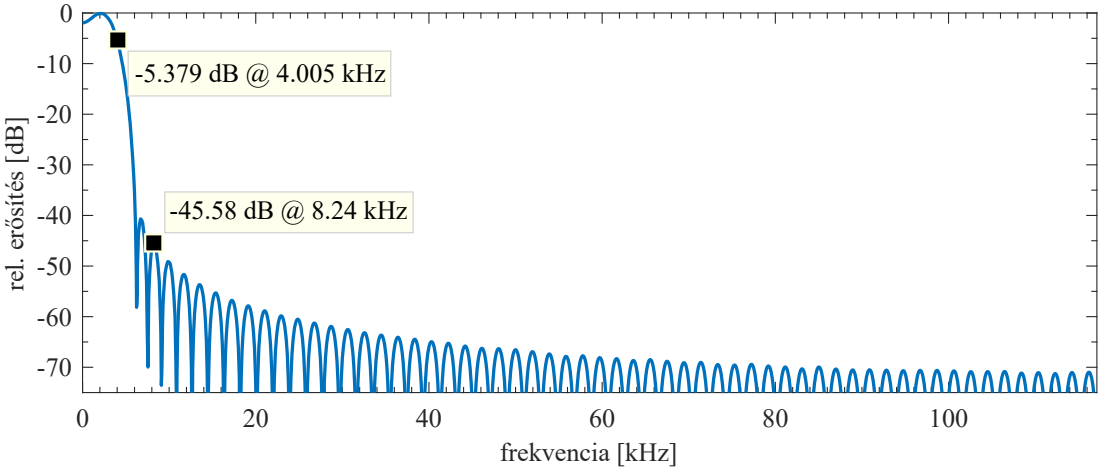


4.7. ábra: *Az AD-átalakítóra jellemző hullámformák [40]*

alacsony szinten tartva a *Track and Hold* áramkörök követni kezdik az analóg bemeneti jeleket, majd a felfutóél hatására megkezdődik a konverzió, amelynek végét az $\overline{\text{EOC}}$ felfutó éle jelzi. Ekkor történhet meg az adatok kiolvasása, amelyhez a $\overline{\text{CS}}$ vonalat logikai alacsony szinten kell tartani, és közben az $\overline{\text{RD}}$ vonal minden egyes lefutóélére egy-egy újabb csatorna mintája jelenik meg a DB0-DB15 párhuzamos adatvonalakon, amíg végig nem haladtunk mind a 8 csatornán.

A beérkező minták egy 16 minta mély FIFO-tárban kerülnek elhelyezésre, ahonnan egy FIR decimáló szűrő olvassa ki azokat, a keletkező kimeneti mintákat pedig egy másik, szintén 16 mély FIFO-tárba helyezi, amely a gazdarendszer felé puffereli az adatokat. A decimáló szűrő HDL-kódját *Xilinx FIR Compiler 7.2*-vel generáltam, ezzel felgyorsítva a fejlesztés folyamatát. Erről annyit érdemes tudni, hogy a 8 csatornát „*interleaved*” módban kezeli, vagyis egy adatfolyamon belül váltakoznak mind a betáplált, mind pedig a kimenő minták. A 120 tap-es decimáló szűrő együtthatóit Matlabban, LS-módszerrel állítottam elő,

az FPGA-ban lévő DSP-szeletek képességeihez igazítva 2^{19}-szeres erősítést eszközöltem, majd kvantáltam. Az így keletkezett szűrő relatív amplitúdómenetét a 4.8-as ábra szemlélteti. A 4 kHz-es határnál már ugyan közel 6 dB-es csillapítással rendelkezik, de egy ANC-

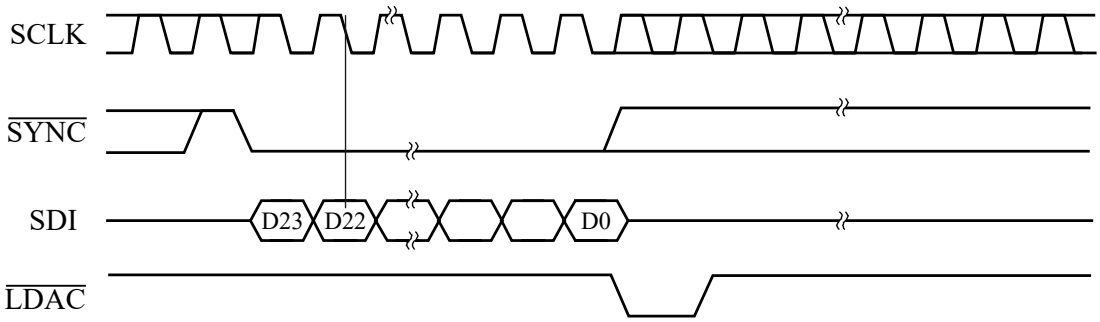


4.8. ábra: *Az AD-átalakítót követő decimáló szűrő amplitúdómenete*

rendszer esetén ez nem jelent nagy problémát, mert ezt az alkalmazott algoritmusok képesek korrigálni. A hasznos jelbe lapolódó legnagyobb csúcs 46 dB-es csillapítást szenved, tehát még ezen a frekvencián is legalább 40 dB-es jel-zaj viszonnyal számolhatunk, amely nem kiemelkedő, viszont valószínűleg nem lesz elsődleges korlátozó tényező a rendszer működtetése során. A szűrő csoportkésleltetése 254 µs körüli. A modul helyes működését hullámforma-szimulációkkal ellenőriztem.

4.2.3 A DA-átalakító illesztése

A DA-átalakító esetében egy szinkron soros interfészen keresztül írhatjuk be a kimeneti mintákat, amelyhez tartozik egy órajelvezeték (SCLK) és egy adatvezeték (SDI). A beírás

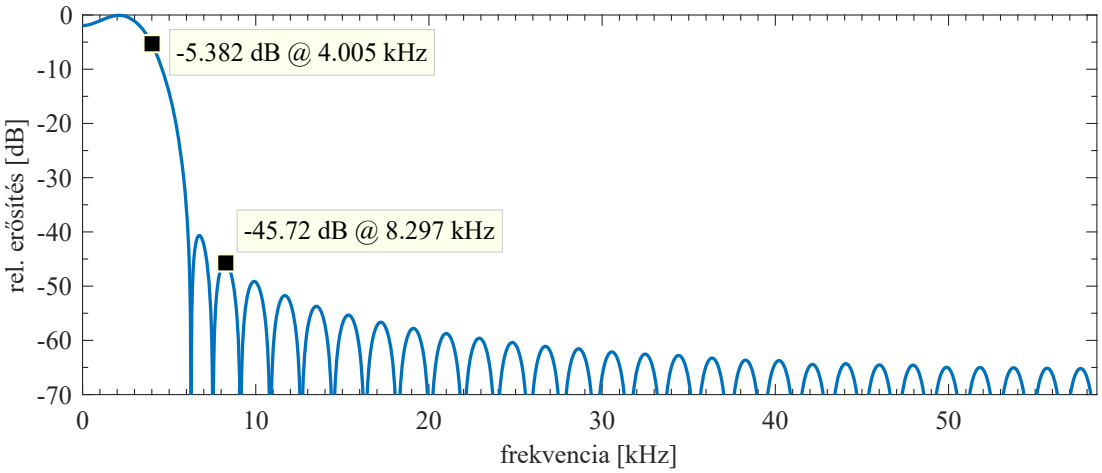


4.9. ábra: *A DA-átalakítóra jellemző hullámformák [41]*

kezdetét a $\overline{\text{SYNC}}$ vezeték logikai alacsony értékével jelezhetjük, ezután az eszköz a bejövő SCLK vonal minden egyes lefutóélénél mintát vesz az SDI vonalból (4.9. ábra), a vett bitek pedig egy belső shiftregiszterben kerülnek tárolásra. Az utolsó bit átvitele után a

$\overline{\text{SYNC}}$ vezeték logikai magasba állításával nyugtázhatjuk az adást. A 16 bites kimeneti mintákat egy 24 bites keretbe kell foglalni, ahol a legmagasabb helyértékű 4 bit (D23-D20) egy parancsot takar (*0001* a betöltésnél), az utána következő 4 bit pedig az éppen töltendő DA-csatorna indexét jelenti (*0000–0111*). Fontos, hogy a $\overline{\text{SYNC}}$ vezeték magasba állításakor még nem jut érvényre a beírt kimeneti minta, hanem ezt egy külön vezeték, az $\overline{\text{LDAC}}$ alacsonyba húzásával érhetjük el, ezáltal lehetőségünk van az összes csatorna beírása után a kimenetek egyidejű frissítésére.

A DA-átalakítót vezérlő Verilog-modul működése akkor kezdődik meg, amikor a gazdarendszer küldi az első kimeneti mintát. Innentől kezdve folyamatosan el kell látnunk a vezérlőt adatokkal ahhoz, hogy tartani tudja az ütemet a mintavételi frekvenciával, különben alulcsordulás következik be. A rendszer felől érkező mintákat egy 16 minta mély FIFO-tár puffereli, amelyből a 10-szeres frekvencián, vagyis 117.187,5 Hz-en működő interpoláló FIR-szűrő olvassa az adatokat. A szűrő HDL-realizációját szintén a Xilinx FIR Compiler 7.2-vel generáltam, és ez is *interleaved* módban működik, vagyis a helyes működéshez a gazdarendszernek kötelessége minden csatornához minden ütemben biztosítani a kimeneti értékeket. Az interpoláló szűrő 60 tap-es lett, melynek együtthatóit Matlabban, LS-módszerrel állítottam elő, 10×2^{15}-szeres erősítést eszközöltem, majd kvantáltam. A szűrő kvantálás utáni relatív amplitúdómenetét a 4.10-es ábra szemlélteti. A



4.10. ábra: *A DA-átalakító előtti interpoláló szűrő amplitúdómenete*

4 kHz-es határnál jelentkező 6 dB-es csillapítást az ANC-rendszer kompenzálni tudja, a hallható tartományba eső tükörfrekvenciák közül 8,3 kHz-en már közel 46 dB a csillapítás, így az itt keltett zaj már valószínűleg nem lesz zavaró. A csoportkésleltetés 252 µs körüli.

A szűrőt elhagyó minták az interpolációs arány miatt egy nagyobb, 256 minta mély FIFO-tárba kerülnek, ahonnan a vezérlő modul a kimeneti mintavételi frekvenciának megfelelő ütemben veszi el, és küldi az adatokat a DA felé. A modul helyes működését hullámforma-szimulációkkal ellenőriztem.

4.3 Zynq alapú processzoros rendszer

A következő lépés az AD- és DA-átalakító kezelését, valamint a mikrofonos modulok adatait fogadó, Verilog nyelven írt HDL-modulok összefogása volt, amit a *Xilinx Vivado HLS* grafikus fejlesztői felületének segítségével vittem véghez. Létrehoztam egy *AXI4 Lite* buszra csatlakoztatható *slave* perifériát, amely regiszteres hozzáférést biztosít az előbb említett modulokhoz. A rendszerbuszon keresztül a báziscímhez képest ezek a következő címeken érhetők el:

- **0x0**: (diagnosztikai célú regiszter)
- **0x4**: AD-átalakító felől érkező minták (csak olvasható)
- **0x8**: DA-átalakító kimeneti mintái (csak írható)
- **0xC**: mikrofonos modulok felől érkező minták (csak olvasható)

Egyazon regiszterhez többször hozzáférve tehát lehetőségünk van az összes csatorna elérésére, így például a 0x4 címen lévő regiszter első olvasásakor a 0 indexű AD-csatornától kapjuk vissza az adatot, a második olvasásra az 1-es indexűtől stb. Az összes csatorna végigcímzése után a következő mintavételi ütembe lépünk és elölről kezdődik az indexelés. A hibalehetőségek csökkentése végett kihasználtam, hogy a rendszer 32 bites adatokkal dolgozik, ezért a kiolvasott/beírt értékek alsó 16 bitje tartalmazza a mintát, az efölött lévő 3 bit pedig a csatorna indexét kódolja. Az AD-átalakítótól és a mikrofonos moduloktól érkező adatokon az előbbiek után még egy bit értelmezve van, amely azt kódolja, hogy érvényes-e a kiolvasott adat. Erre azért volt szükség, mert a rendszer a lehető legkisebb késleltetés elérése céljából nem megszakításokkal, hanem rendszeres lekérdezésekkel fogadja a perifériától az adatokat, ilyenkor pedig előfordulhat, hogy két lekérdezés között nem érkezett be újabb adat.

A kész perifériát ezután a Vivado HLS-ben egy Zynq-processzoros rendszerbe illesztettem, amelyben aktiváltam az Ethernet-vezérlőt is. A rendszer szintézise után kezdetét vehette a processzoros rendszer firmware-jének fejlesztése. Az elképzelt megvalósításban a két ARM Cortex-A9-es mag *AMP*-módban (*asymmetric multiprocessing*) működik együtt, vagyis a két mag más programkódot futtat, és a közös memórián keresztül kommunikálnak egymással. Erre alkalmas memóriaterület például a külső, 1 GB méretű DDR3 memória, valamint a chipen belül található 256 KB-os *OCM* (*on-chip RAM*).

A két mag közül a *CPU0* jelű látja el a jelfeldolgozási feladatokat. A főprogram tulajdonképpen egy végtelen ciklusból áll, amely folyamatosan lekérdezi a AD-átalakítótól és mikrofonos modulok felől érkező mintákat, új minták érkezésekor pedig elvégzi a (2.2)-s egyenlet alapján a részleges kimeneti minták kiegészítését, majd miután az adott mintavételi ütemben minden bemeneti csatorna lekérdezésre került, továbbítja a DA-átalakító felé a kész mintákat. Lehetőség van a kimeneti csatornákon fehér zaj kiadására, ami az ANC-rendszer

futtatását megelőző identifikációs fázisban hasznos. A nem használt kimenetek némíthatók. A részleges minták kiegészítését végző kódrészletben *ARM Neon* SIMD utasításokat használ, amely az SSE utasításkészlethez hasonlóan egy hatékony, vektoros feldolgozást tesz lehetővé.

A *CPU1* feladata, hogy minden szükséges adat a CPU0 rendelkezésére álljon addigra, amikor épp szüksége van rá. Ilyen adatok a részleges kimeneti minták, az utolsó $Q + B$ szűrőegyüttható az adaptív szűrőkhöz, az egyes kimeneti csatornák működési módja. A CPU0 emellett a bemeneti csatornáktól fogadott adatokat továbbítja a CPU1 felé, az pedig a PC felé küldi tovább. A CPU1 által futtatott kód az *lwIP* függvénykönyvtárra alapoz, amely egy viszonylag egyszerűen használható, ugyanakkor hatékony TCP/IP implementáció. A kommunikáció során ugyan nem TCP, hanem *UDP-protokollt* használtam a kis késleltetés igényét szem előtt tartva, de az lwIP még így is jó szolgálatot tett, mert megvalósítja az IP alapú címfeloldást (*ARP*; *Address Resoultion Protocol*), és leegyszerűsíti az Ethernet-csomagok küldését és fogadását.

4.4 PC-s alkalmazás fejlesztése

A rendszer működtetéséhez szükség volt egy olyan PC-s alkalmazásra, amely képes az alábbi feladatok ellátására valós időben:

- az ANC-rendszer csatornáinak feladatkiosztása
- az Etherneten érkező bemeneti adatok fogadása
- a GPU-n futó kernelkód meghívása a fogadott adatokkal
- a keletkezett részleges kimeneti minták és az utolsó $Q + B$ szűrőegyüttható elküldése Etherneten keresztül
- az adatfolyamok mentése háttértárra a későbbi kiértékeléshez
- üzemmódválasztás levezénylése (üresjárás/identifikáció/zajcsökkentés)
- a bátorsági tényezők menet közbeni módosításának lehetősége
- a szűrőegyütthatók nullázásának lehetősége (leginkább gerjedésnél)

A szükséges hatékonyságot és a rendelkezésemre álló időkeretet szem előtt tartva végül Ubuntu operációs rendszer alatt C++ nyelven egy konzolos alkalmazást fejlesztettem. A program induláskor egy külső konfigurációs fájlból olvassa be az ANC-rendszer csatornáinak konfigurációját, vagyis hogy az igénybe vett bemeneti csatornák közül melyek szolgáljanak referenciaként vagy hibajelként, illetve a felhasznált kimeneti csatornák közül melyek tartoznak zajcsökkentő vagy zajkeltő hangsugárzóhoz. Ez a fájl tartalmazza továbbá az induláskori bátorsági tényezőt mind az identifikációs, mind pedig a zajcsökkentő módhoz, és a másodlagos út késleltetett inverzének identifikálásánál bevezetett D paramétert.

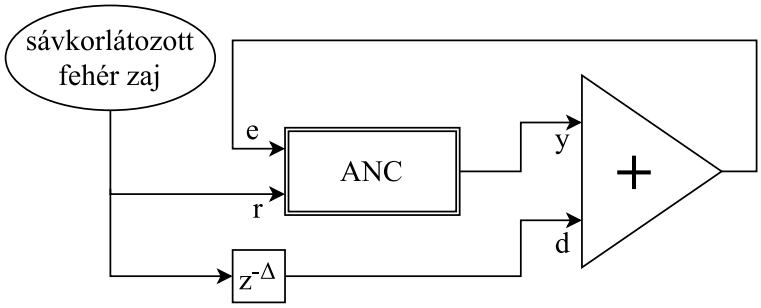
Kilépésnél a másodlagos utak szűrőegyüttható mentésre kerülnek, és amennyiben nem történt változás a rendszer hibajeleinek és beavatkozóinak számát, illetve a szűrőhosszakat illetően, a program következő indításakor ezek az adatok is betöltésre kerülnek, ami hasznos funkció, ha egy adott elrendezés mellett több mérést is szeretnénk elvégezni. Az alkalmazás 3 szálon fut:

- **#1**: a felhasználói felülettel történő interakciókért felelős, és utasításokat ad a fő programszálnak
- **#2**: az adatfolyamok mentését végzi
- **#0**: minden más feladat, amely valós idejű, periodikus végrehajtást követel meg

A fenti leírásból következik, hogy a #0 szál az operációs rendszer valós idejű ütemezőjét kell, hogy használja, és mivel gyakorlatilag sosem függeszti fel a saját futását *yield* rendszerhívással, egy külön processzormagot felemészt a végrehajtása. Ez azt is maga után vonja, hogy az alkalmazás működéséhez legalább két logikai maggal rendelkező CPU szükséges, különben a másik két szál és egyéb alkalmazások sosem kerülnének ütemezésre. A másik két szál tehát normál ütemezési osztályba tartozik, mivel az általuk végzett feladatok nem vagy csak mérsékelten időkritikusak.

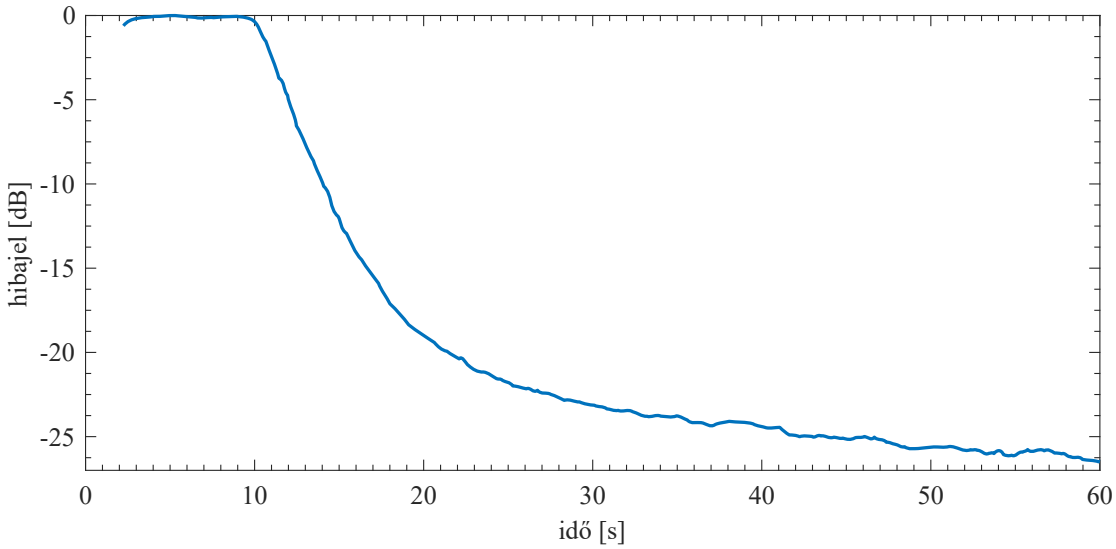
4.5 A rendszer élesztése, tesztelése

Az élesztési fázisban egy műveleti erősítőkkel megvalósított keverőáramkört vettem igénybe, amelynek egyik bemeneti csatornájára 4 kHz-re sávkorlátozott fehér zajt kapcsoltam, a másik bemenetére pedig az ANC-rendszer kimenetét, amelyet a 0-s indexű DA-csatorna szolgáltatott (4.11. ábra). A hibajel ebben a konfigurációban a keverőerősítő kimenete



4.11. ábra: *Az ANC-rendszer élesztéséhez használt konfiguráció*

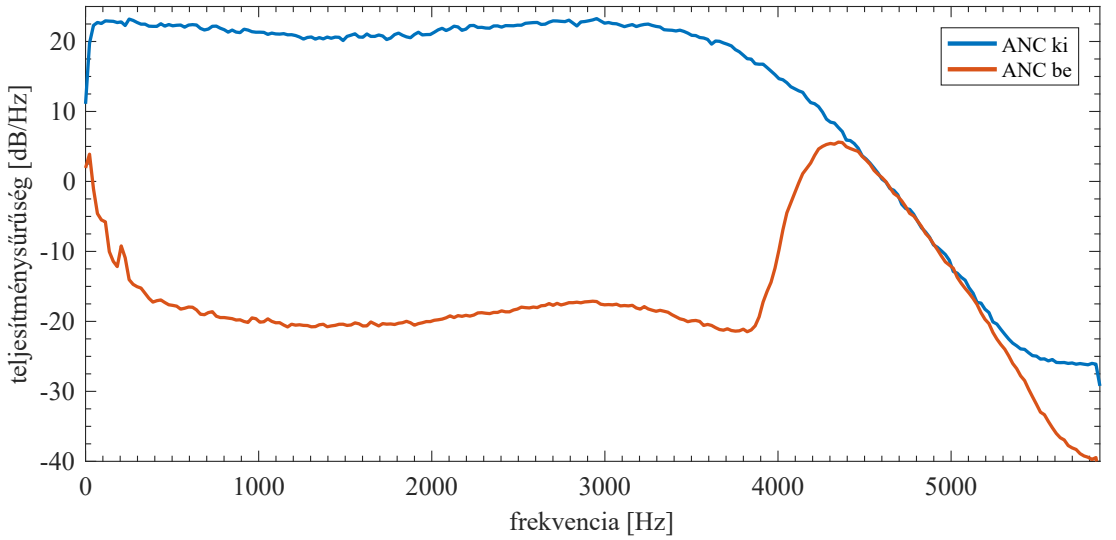
volt, amelyet az 1-es indexű AD-csatornán vezettem vissza a rendszerbe. Az elnyomandó fehér zajt egy másik PC audiointerfésze szolgáltatta, a sztereó hangkimenet bal csatornája a zajcsökkentő rendszer 0-s indexű AD-csatornájára csatlakozott, mint referenciajel, a jobb csatornán pedig a bal csatorna 10 ezredmásodperccel ($\Delta = 117$) késleltetett változata szólalt meg, amelyet a keverőerősítőre zajforrásként csatlakoztattam. A másodlagos út



4.12. ábra: *A keverőerősítőnél mért beállási görbe*

identifikációjánál $D = 256$-ot állítottam be, a késleltetett inverz impulzusválaszáról pedig közvetlenül leolvasható volt, hogy a DA-AD hurok teljes késleltetése jó közelítéssel 8 mintányi, amely $f_{ki} = 11.718,75$ Hz-es mintavételi frekvencián kb. 700 µs-nak felel meg. A DA-átalakító előtti interpoláló, illetve az AD-átalakító utáni decimáló szűrők késleltetése egyaránt 250 µs körüli, ami összesen 6 mintányi késleltetésre ad magyarázatot, a fennmaradó 2 mintányi késleltetés pedig a DA-átalakító előtti FIFO-tárnak tudható be, amely az alulcsordulás ellen véd. Kijelenthető tehát, hogy *sikerült a követelményekben lefektetett 1 ezredmásodperc alá szorítani a rendszer teljes késleltetését*, további késleltetéscsökkenés pedig már csak a mintavételi frekvencia növelésével lehetséges.

A keverőáramkörrel végzett mérés során kapott beállási görbét a 4.12-es ábra szemlélteti. A zajcsökkentő algoritmus $t = 10$ s-tól aktív. Az alkalmazott szűrőhossz $N = 8192$ volt,



4.13. ábra: *A hibajel spektrális teljesítménysűrűsége*

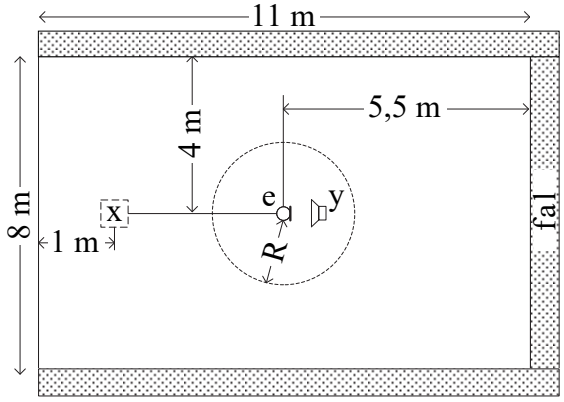
amely jól láthatóan limitálja a beállás sebességét. Érdemes a frekvenciatartományban is megvizsgálni a rendszer viselkedését a 4.13-as ábra alapján. Az alacsony frekvenciáknál az elnyomandó zaj és a referenciajel közötti Δ késleltetéskülönbség befolyásolta az elérhető elnyomást, kellően nagy (pl. 1000 mintányi) késleltetés esetén szinte függőleges meredekséget is el tudtam érni. A működés 4 kHz felett már nem specifikált, a köztes tartományban pedig valószínűleg a decimáló szűrők minősége korlátozta az elnyomást, amely így is szemmel láthatóan kb. 40 dB-es érték.

A rendszer működőképességét ugyanakkor sikerült igazolnom, az akusztikai térben kitűzött méréseket pedig a rendelkezésemre álló időkeret szűkössége miatt csak a GPU-kernelként már megvalósított, korrigálatlan FeLMS-algoritmussal volt lehetőségem elvégezni, jóllehet a fejlesztett ANC-rendszerben rejlő potenciált ez nem lesz képes teljes mértékben kiaknázni.

5 Akusztikai mérések

5.1 A mérési elrendezés ismertetése

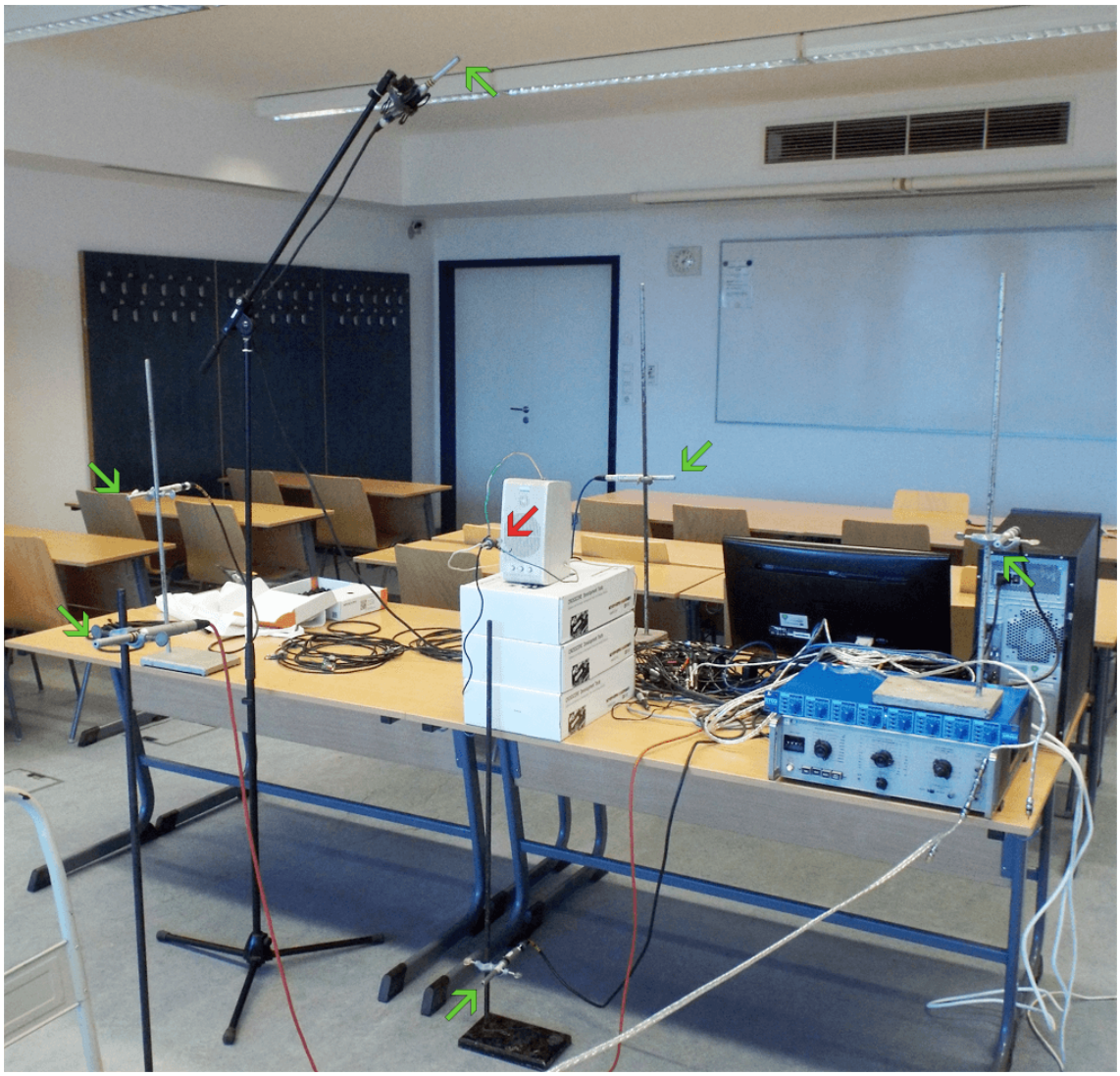
Az ANC-rendszer működőképességét a gyakorlatban is igazolnom kellett, ezért akusztikai méréseket végeztem az *IE224*-es egyetemi tanteremben. A választás azért épp erre a teremre esett elsősorban, mert a fizikai kiterjedése és a térben jelen lévő anyagok akusztikai tulajdonságai folytán hosszú, több másodperces zengésre számíthatunk, ezért a GPU-s feldolgozással elérhetővé vált, akár több tízezer együtthatós szűrők alkalmazásától itt várhatunk lényeges javulást a zajelnyomásban. További előny a teremmel kapcsolatban, hogy korábban már egy másik, ugyancsak FeLMS algoritmust alkalmazó ANC-rendszerhez fűződő méréssorozatnak is otthont adott, ennek köszönhetően pedig rendelkezésemre állt egy referencia-adatsor a különböző konfigurációk mellett elérhető zajelnyomásokról.



5.1. ábra: *A mérési konfiguráció az IE224-es teremben [42]*

A terem alaprajzát az 5.1-es ábra szemlélteti, amelyen a mérési elrendezés is látszik. Az ablakok előtt lévő *x* jelű hangsugárzó a zajforrás szerepét tölti be, az EM és a beavatkozó hangsugárzó pedig középtájt helyezkedik el, egymástól kb. 20 centiméterre. A hibamikrofon *R* sugarú környezetében összesen 6 RM került elhelyezésre, minden fal irányába (ideértve a padlózatot és a mennyezetet is) egy-egy darab. Az erről készült fénykép az 5.2-es ábrán tekinthető meg, ahol a rendszer épp *Behringer ECM8000* kondenzátor-mérőmikrofonokkal üzemelt. A zöld színű nyilak az RM-ek, a piros színű pedig az EM pozícióját mutatja.

Az 1.1-es pontban kifejtett gondolatmenet alapján akkor számíthatnánk a legnagyobb zajelnyomásra, ha az RM-ek a beavatkozó hangsugárzó és a zajforrás között, utóbbihoz minél közelebb helyezkednek el. Ha viszont a zajforrás helyzete nem ismert előre, akkor például az előbb bemutatott elrendezést alkalmazhatjuk. Jóllehet ilyenkor a zajforrásból érkező közvetlen hangra nem minden RM-nél teljesül a kauzalitási feltétel, a falakról



5.2. ábra: *Az elrendezésről készült fénykép*

visszavert hangok esetében, az adott falhoz legközelebbi RM-nél viszont már igen, vagyis a terem zengésének kioltásában az összes RM részt tud venni.

A kísérletek során Behringer ECM8000 mérőmikrofonokkal két mérési sorozatot végeztem, ahol az EM és az RM-ek közötti távolságot rendre 0,5 és 1 méterre állítottam be. Mindkét sorozatnál megvizsgáltam, milyen eredmények érhetők el 1, 2 és mind a 6 RM jelének felhasználásával. Az 1 RM-es esetben a zajforráshoz legközelebbi mikrofon volt aktív, 2 RM-nél pedig az azzal szemközti is. A méréseket 2048, 4096 és 8192 együtthatós szűrőkkel is megismételtem. Ennél hosszabb szűrők futtatására is volt lehetőség, azonban az előzetes mérések alapján a konvergenciasebesség olyan mértékben csökkent az alkalmazott algoritmusból kifolyólag, hogy ez nem lett volna praktikus. A zajelnyomást valós időben egy *Voltcraft SL-400* digitális zajszintmérővel követtem nyomon („C” súlyozószűrővel), és addig működtettem a rendszert, amíg az elért érték jó közelítéssel állandósult. A másodlagos út identifikációjánál alkalmazott késleltetés $D = 256$ mintányi volt.

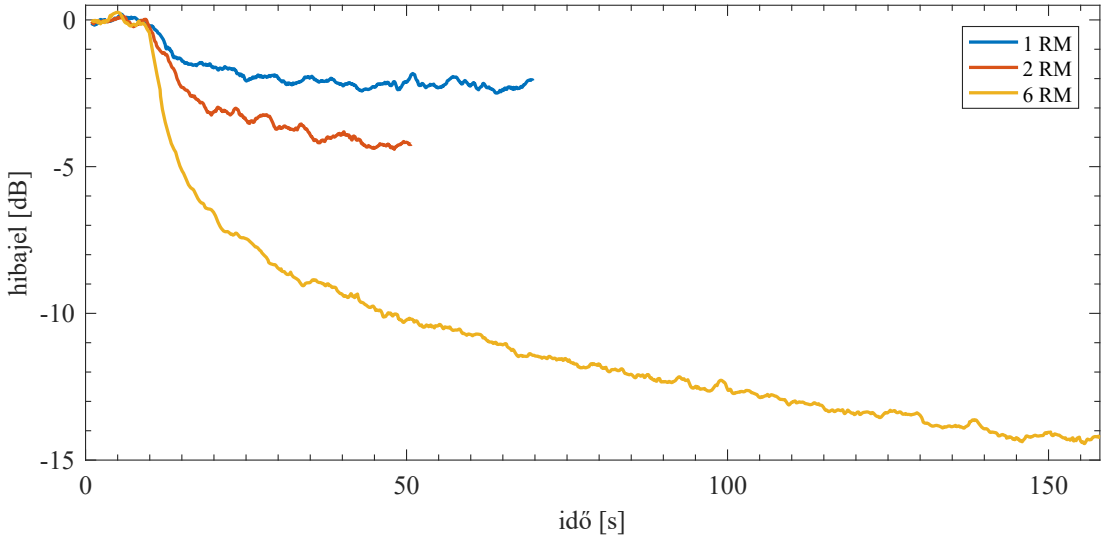
5.2 Mérési eredmények

Az elkészült ANC-rendszerrel az előbb vázolt mérési elrendezések és beállítások mellett 0,5 és 1,5 kHz sávszélességű fehér zaj elnyomását teszteltem. Az EM által mért zajszint 78 dB_C volt kikapcsolt zajelnyomás mellett. A terem 52 dB_C-es háttérzaja érdemben valószínűleg nem befolyásolta a méréseket. Az eredményeket az 5.1-es táblázat összegzi. A zajszintmérő leolvasási pontatlansága ±0,5 dB-nek vehető.

		1			**2**			**6**			*RM-ek száma*
		2K	**4K**	**8K**	**2K**	**4K**	**8K**	**2K**	**4K**	**8K**	*szűrőhossz (N)*
0,5 kHz	**0,5 m**	3	4	4	5,5	6,5	8	11,5	11	10,5	
	1 m	2,5	3	3,5	4,5	6,5	7,5	13,5	14,5	13	
1,5 kHz	**0,5 m**	2,5	3	3,5	4,5	7	7,5	14	15,5	16	
	1 m	2,5	3	3,5	4,5	5	7,5	14	18	19,5	
zaj sávszél.	*R*										

5.1. táblázat: *A kísérletek során mért elnyomásértékek decibelben*

Ahogyan az 1.6.5-ös pontban vázolt szimulációk alapján várható volt, az RM-ek számának növelésével jelentős javulás mutatkozik a zajelnyomásban még annak ellenére is, hogy a kauzalitási feltétel jószerivel csak az adott RM-hez legközelebb eső falról visszaverődő hangokra teljesül. Az 5.3-as ábrán látható, hogy 6 RM esetén a leállítás pillanatában

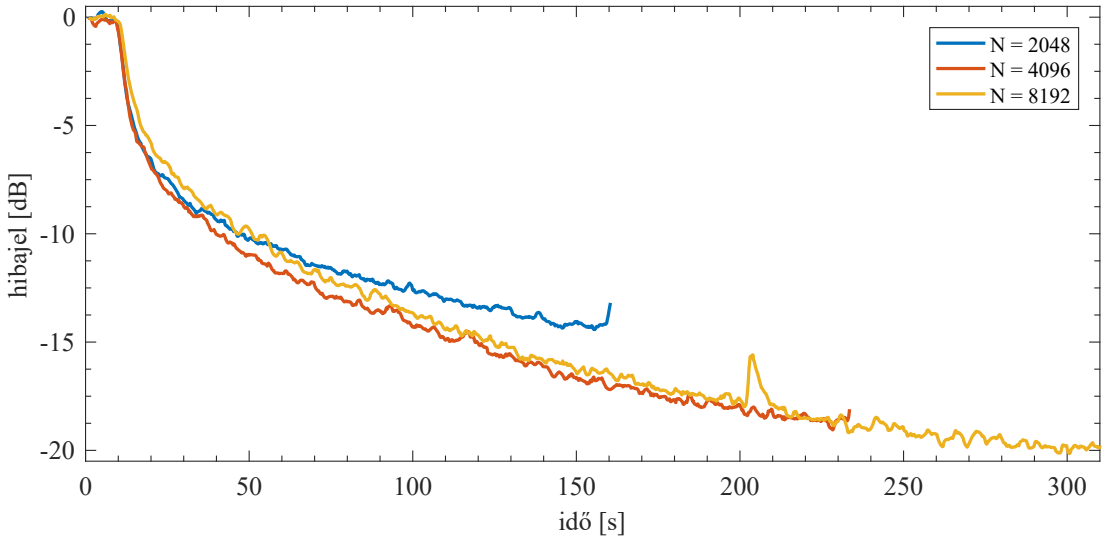


5.3. ábra: *Az RM-ek számának hatása (N=2048, BW=1,5 kHz, R=1 m)*

még nem állandósult teljesen az elnyomás, viszont a kevesebb referenciajelet használó mérésekhez képest így is jóval tovább, majdnem 3 percig működött a rendszer.

A szűrőhosszak tekintetében is többnyire helytállónak bizonyultak az 1.6.3-as pontban ismertetett szimulációk alapján levont következtetések, hiszen a rendre növekvő együtthatószám jóformán minden esetben nagyobb zajelnyomással párosult. A 6 RM-et használó beállítás mellett kapott eredmények látszólagosan ellentmondanak az előbbi megállapításnak, azonban a beállási görbéket tanulmányozva azt vettem észre, hogy ezeknél a méréseknél

valószínűleg túl nagy μ-t állítottam be a konvergencia gyorsításának reményében, és a stabilitás határhelyzetének közelében az elérhető zajelnyomás visszaesett. Ahogyan az várható volt, a szűrőhosszak növelésének jótékony hatása egyre kevésbé volt szignifikáns, amint a szűrőhosszak a terem impulzusválaszának hosszához közelítettek. A szimulációk alapján szintén várható volt, hogy a növekvő együtthatószám a konvergenciasebesség visszaesésével fog járni, ami a leghosszabb szűrőt alkalmazó beállításoknál volt kifejezetten érzékelhető (5.4. ábra).



5.4. ábra: *A növekvő szűrőhosszak hatása (6 ref., BW=1,5 kHz, R=1 m)*

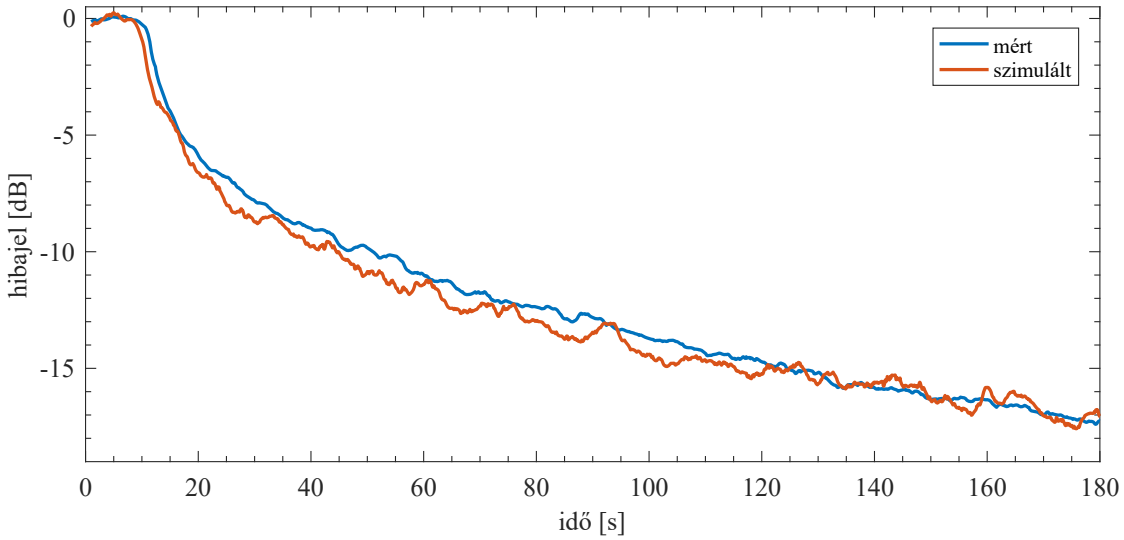
A zajforrás által kibocsátott fehér zaj sávszélessége elméletileg közömbös az elérhető zajelnyomás szemszögéből, amit a mérési eredmények is alátámasztani látszottak. Egyedül a 6 RM-et alkalmazó beállításnál vehető észre a leolvasási pontatlanságnál nagyobb különbség, azonban a korábban említett stabilitási gondok miatt ezek alapján nem érdemes messzemenő következtetéseket levonni. Az RM-ek EM-től vett R távolságának hatása csak a 6 RM-et alkalmazó beállításnál vált kimutathatóvá, és a mérések alapján úgy tűnik, hogy a kritikus 25 centiméterhez képest a fél méteres távolság már kellően nagy ahhoz, hogy elfogadható teljesítménnyel működjön a rendszer, de a várakozásoknak megfelelően jobb eredmények születtek 1 méteres távolságnál.

A MEMS-mikrofonos modulokat már csak azokkal a beállításokkal teszteltem, amelyek a legjobb zajelnyomást eredményezték a kondenzátormikrofonoknál, vagyis $R = 1$ m-es távolság, 1,5 kHz-es sávszélességű fehér zaj, 6 RM és $N = 8192$ hosszú szűrők mellett. A mérőmikrofonokkal elért 19,5 dB-es értéktől mindössze 2,5 dB-lel rosszabb, 17 dB-es elnyomást sikerült elérnem, ami elismerésre méltó annak fényében, hogy a MEMS-mikrofonok jel-zaj viszonya legalább 10-15 dB-lel alulmarad a kondenzátormikrofonokéhoz képest.

5.3 Az eredmények ellenőrzése

A rendszer teljesítményét érdemes összehasonlítani a teremben végzett korábbi, *DSP alapú méréssorozat kimenetelével* is [42]. Az alkalmazott zajcsökkentő rendszer szintén FeLMS algoritmust alkalmazott 8 kHz-es mintavételi frekvencián. A hardver nem tette lehetővé 6 RM mellett hosszú szűrők futtatását, így a mérési lehetőségek korlátozottak voltak. A konfiguráció terembeli elrendezése megegyezett a jelenlegivel $R = 1$ m mellett, az elnyomandó fehér zaj sávszélessége pedig 1 kHz volt. Két referenciajellel és 8000 együtthatós[1] szűrők alkalmazásával 6,6 dB-es elnyomást mértek, az általam fejlesztett rendszer 8192 együtthatóval 7,5-8 dB elnyomást tett lehetővé. Mind a 6 RM jelének a felhasználása mellett 2400 együtthatós[2] szűrőket tudtak futtatni, amely 15,7 dB-es elnyomáshoz volt elegendő, esetemben a 2048 együtthatós szűrő a legjobb esetben 14 dB-t, a 4096 együtthatós pedig 18 dB-t ért el, tehát kijelenthető, hogy a két rendszer hasonló hatékonysággal működik.

Szintén [42] nyomán nyilvánosan elérhető egy *teremszimulációs programcsomag*, amellyel korábban az IE224-es termet modellezték, és a 5.1-es pontban bemutatott konfigurációval kapcsolatban felmerülő átviteli függvények hosszú FIR-közelítései is rendelkezésemre álltak. Ezek felhasználásával lehetőségem nyílt egy *offline szimuláció*



5.5. ábra: *A mért és a teremszimulációk alapján kapott beállási görbe*

elvégzésére, amely figyelembe vette a beavatkozó hangszóró és az EM közötti visszacsatolásokat is. A beállási görbéket a legjobb eredményt szolgáltató beállítások mellett vetettem össze, 3 percen keresztül. Az eredményeket az 5.5-ös ábra összegzi. Jól láthatóan egymásra simul a két görbe, ami szintén a rendszer megfelelő működését támasztja alá.

[1] 11.718,75 kHz-en ez 11719 együtthatónak felel meg
[2] 11.718,75 kHz-en ez 3516 együtthatós szűrővel ekvivalens

Továbbfejlesztési lehetőségek

Jóllehet a feladatkiírás kitételeinek eleget tettem, és elkészült egy GPU-n alapuló ANC-rendszer, azonban az ezáltal elérhetővé vált számítási kapacitás jobbára még kihasználatlan. A teszteléshez használt FeLMS-algoritmus korrigálatlan késleltetése sokat ront a rendszer stabilitásán, emiatt drasztikusan csökkentenünk kell az alkalmazott μ értékét, miáltal pedig csökken a gyakorlatban elérhető zajelnyomás, és a konvergencia sebessége is visszaesik.

Az 1. fejezetben bemutatott algoritmusok közül mindenképp érdemes lenne implementálni az MFeLMS-algoritmust, amely mindössze egy szűrési lépéssel igényel többet az alapalgoritmusnál, viszont cserébe képes az imént vázolt probléma kiküszöbölésére. Ezen túlmenően az NLMS-OCF egy jól skálázható eljárásnak ígérkezik, amelyet a GPU-n hatékonyan megvalósítva átütő javulást érhetünk el mind az elérhető zajelnyomás, mind pedig a beállási sebesség tekintetében. Érdemes továbbá az 1.5-ös pont szerinti MIMO-kiterjesztést is újragondolni, hiszen az nem veszi figyelembe a referenciajelek korreláltságát, ez pedig szerepet játszhat a szűrők lassú beállásában.

Az 1.6.5-ös pontban láthattuk, hogy a szimulációk alapján jelentős különbség van az egy és a két referenciajelet felhasználó beállítások között. A 6 RM-et felvonultató konfigurációnál az EM-en átmenő, képzeletbeli vízszintes síkban elhelyezkedő zajforrásból érkező közvetlen hang általános esetben csak egyetlen RM-hez jut el a kauzalitási feltételt betartva, a többi mikrofon esetében pedig ez már csak a falakról visszavert hangokra áll fenn. Érdemes lenne ehelyett 8 RM-et elhelyezni egy szabályos kocka csúcsaiban, az EM-et pedig a középpontjában, ugyanis így a síkban bármilyen irányból érkező zajból legalább két mikrofon tudna időben mintát venni, ami elméletileg nagyobb elnyomást tenne lehetővé.

Nem kizárt, hogy még ennél is több referenciajel bevonásával további javulás lenne elérhető. A MEMS-mikrofonos modulok buszrendszere 16 egység kiszolgálására lett tervezve, vagyis elméletileg további 8 modul csatlakoztatására is van lehetőség. A rendszer bővítése előtt azonban célszerű lenne megvizsgálni a buszjelek minőségét, illetve a mikrofonos modulok átvitelét, torzítását, amelyekre idő hiányában már nem kerülhetett sor.

A rendszer egyszerűbb üzemeltetése végett érdemes fontolóra venni a fejlesztett alkalmazás portolását Windows rendszerre, hiszen az átalakított LMS-algoritmus már zömmel képes elviselni az említett operációs rendszernél felmerülő késleltetéseket. A könnyebb kezelhetőséget egy grafikus felhasználói felület is segíthetné, ahol valós időben nyomon lehetne követni a rendszer állapotát, és akár a beállási görbéket.

Szintén kényelmi okokból kifolyólag át lehet térni Ethernet alapú kommunikációról USB alapúra, mivel számos PC-n csak egyetlen Ethernet-aljzat található, ezért választani kényszerülünk az internetelérés és az ANC-rendszer működtetése között.

Összefoglalás

Az adaptív, előrecsatolt zajcsökkentő rendszerek irodalmának feltárása során megismerkedtem a másodlagos utak átvitelének kompenzálására alkalmas algoritmusokkal, amelyek lehetővé teszik az LMS alapú adaptív szűrők ANC-rendszerekben történő alkalmazását. Tanulmányoztam, hogyan lehet a fellépő késleltetés negatív hatását kompenzálni, illetve milyen lehetőségeink vannak a konvergenciasebesség növelésére, ami a hosszú szűrők futtatása miatt hangsúlyos. A több be- és kimeneti csatornára való kiterjesztés tisztázása után szimulációkat is végeztem, így képet kaptam arról, hogy milyen eredményekre számíthatok a különböző beállítások mellett.

Az ANC-rendszer tervezése előtt bemutattam, hogyan történik a GPU programozása, és mik a főbb irányelvek, szabályok, amelyeket szem előtt kell tartanunk. A számítási kapacitás hatékony kihasználása végett elkerülhetetlen volt a grafikus processzor architekturális alapjainak megismerése. Az FeLMS-algoritmuson módosításokat kellett eszközölnöm annak érdekében, hogy tolerálja a PC-s környezetből fakadó késleltetéseket és a blokkos feldolgozást, majd közöltem a GPU-kernel implementációját.

A rendszer tervezését a PC-hez történő csatlakozási lehetőségek feltérképezésével kezdtem, majd utána a követelmények ismeretében kiválasztottam a rendszer központi elemét, egy Zynq-7020 SoC-re alapuló fejlesztői kártyát. Javaslatot tettem egy MEMS-mikrofonokat alkalmazó adatgyűjtő hálózat kiépítésére, amely költséghatékony megoldást kínál, és különösen jól használható, ha sok referenciajelre van igény. Ehhez meg kellett ismerkednem a digitális kimenetű MEMS-mikrofonokhoz kapcsolódó jelfeldolgozási teendőkkel, amelyet egy mikrokontrollerrel terveztem megvalósítani. A következő lépés a modulok kommunikációs protokolljának és szinkronizációs mechanizmusának kidolgozása volt, majd pedig a fizikai megvalósítással kapcsolatos kérdéseket tisztáztam. A tervezési fázist az elképzelt I/O-panel ismertetésével zártam, amely a 8-8 analóg be- és kimeneti csatorna kezelése mellett hidat képez a központi egység és a mikrofonos modulok között.

Ezután következett az áramköri és a firmware-realizáció, amelyet a mikrofonos modulokkal kezdtem. A mikrokontroller szükséges egységeinek felderítése, és a lábkiosztás véglegesítése után hozzáláthattam az NYHL tervezésének az EMC irányelveket betartva. A CIC-szűrő és a decimáló FIR-szűrő megtervezése után elkészítettem az eszköz firmware-jét, majd pedig a modulokat a buszra felfűzve teszteltem. Az I/O-panel tervezése még inkább nagy körültekintést igényelt az analóg jelek jelenléte miatt. Az AD- és DA-átalakítók, továbbá a mikrofonos modulok buszának illesztéséhez Verilog nyelven HDL-modulokat írtam, amelyeket egy AXI4-buszra csatlakozó perifériává fogtam össze. Ezt a perifériát egy Zynq alapú processzoros rendszerbe foglaltam, majd a két Cortex-A9 processzormagra

fejlesztettem alkalmazást, amelyben az egyik mag a zajcsökkentéshez kapcsolódó szűrési feladatokért, a másik pedig a kommunikációs feladatokért felelős. A következő lépés egy PC-s applikáció fejlesztése volt, amely a központi egységgel történő adatcsere lebonyolítását, a GPU-s kernel periodikus futtatását és az adatok mentését végzi. A kész rendszert ezután egy keverőerősítő segítségével teszteltem.

Végül akusztikai méréseket végeztem a rendszer működőképességének igazolásához, amelynek során 8192 együtthatós szűrőkkel és 6 referenciajel felhasználásával 19,5 dB-es zajelnyomást tudtam elérni. Viszonyítási alapként egy korábbi, DSP-t alkalmazó zajcsökkentő rendszer, valamint egy teremszimulációs programcsomagra alapuló offline szimuláció szolgált, az eredményeket összehasonlítva pedig kijelenthető, hogy a most fejlesztett rendszer működése megfelelő.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani Bogár Istvánnak, hogy elvállalta az áramkörtervezéssel kapcsolatos folyamatok lektorálását, illetve hogy számtalan hasznos tanáccsal segítette a munkámat. Hálás vagyok a konzulensemnek, Sujbert Lászlónak a rugalmas hozzáállásáért és a példátlan rendelkezésre állásáért, hiszen az idő szűke miatt a nyári szabadságát is hajlandó volt félbeszakítani egy érdekfeszítő mérésért. A feleségem és a családom önzetlen támogatása nélkül pedig ez a dolgozat nem vagy nem ilyen minőségben jöhetett volna létre.

Betűszavak

µC mikrokontroller
AD analog-digital (converter)
AMP asymmetric multiprocessing
ANC active noise cancellation
ARP Address Resolution Protocol
ASIO audio stream input/output
AVX Advanced Vector Extensions
CC compute capability
CIC cascaded integrator-comb
CPU central processing unit
DA digital-analog (converter)
DIP dual in-line package
DLMS delayed LMS
DMA direct memory access
DSP digital signal processor
EIN equivalent input noise
EM hibamikrofon (error microphone)
EMC electromagnetic compatibility
FeLMS filtered error LMS
FIFO first in, first out
FIR finite impulse response (filter)
FLOPS floating point operations per second
FPGA field programmable gate arrays
FxLMS filtered reference LMS
GP-GPU general-purpose computing on graphics processing unit
GPU graphics processing unit
HDL hardware description language
I/O input/output
LDO low dropout regulator
LED light emitting diode
LMS least mean square (adaptive algorithm)
MAC multiply and accumulate
MEMS microelectromechanical systems
MFeLMS modified filtered error LMS
MIMO multiple input/multiple output
MIPS million instructions per second
NLMS normalized LMS
NLMS-OCF NLMS with orthogonal correction factors
NYHL nyomtatott huzalozású lemez
OCM on-chip memory
PC personal computer
PDM pulse density modulation
PLL phase locked loop
RM referenciamikrofon
RTT roundtrip time
SAR successive approximation
SIMD single instruction/multiple data
SINAD signal to noise and distortion (ratio)
SMM streaming multiprocessor
SMT simultaneous multithreading
SoC system on chip
SPI Serial Peripheral Interface
SSE Streaming SIMD Extensions
SWD Serial Wire Debugging (interface)
TCP/IP Transmission Control Protocol/Internet Protocol
TDM time division multiplexing
TVS transient voltage suppressor
UDP User Datagram Protocol
USB Universal Serial Bus
UTP unshielded twisted pair

Hivatkozások

Könyvek és folyóiratok

[1] S. M. Kuo and D. R. Morgan. *Active noise control systems : algorithms and DSP implementations*. English. New York : Wiley, 1996. ISBN: 0471134244.

[2] M. Basner et al. „Auditory and non-auditory effects of noise on health“. In: *Lancet* 383.9925 (Apr. 2014), pp. 1325–1332.

[3] C.M. Harris. *Handbook of acoustical measurements and noise control*. McGraw-Hill, 1991. ISBN: 9780070268685.

[4] L.L. Beranek and I.L. Vér. *Noise and vibration control engineering: principles and applications*. A Wiley Interscience publication. Wiley, 1992. ISBN: 9780471617518.

[6] P.A. Nelson and S.J. Elliott. *Active control of sound*. Academic Press, 1993. ISBN: 9780125154260.

[7] Y. Kajikawa, W. S. Gan, and S. M. Kuo. „Recent advances on active noise control: open issues and innovative applications“. In: *APSIPA Transactions on Signal and Information Processing* 1 (Aug. 2012). DOI: 10.1017/ATSIP.2012.4.

[8] M. R. F. Kidner. „Active Noise Control: A review in the context of the 'cube of difficulty'.“ In: *Acoustics Australia* 34.2 (2006), pp. 65–69.

[9] S. J. Elliott and P. A. Nelson. „Active noise control“. In: *IEEE Signal Processing Magazine* 10.4 (Oct. 1993), pp. 12–35. ISSN: 1053-5888. DOI: 10.1109/79.248551.

[10] B. Widrow and E. Walach, eds. *Adaptive Inverse Control*. Upper Saddle River, NJ, USA: Prentice Hall Press, 1996. ISBN: 0-13-005968-4.

[13] S. T. Neely and J. B. Allen. „Invertibility of a room impulse response“. In: *The Journal of the Acoustical Society of America* 66.1 (1979), pp. 165–169. DOI: 10.1121/1.383069.

[14] S. Haykin. *Adaptive Filter Theory (3rd Ed.)* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996. ISBN: 0-13-322760-X.

[15] G. Long, F. Ling, and J. G. Proakis. „The LMS algorithm with delayed coefficient adaptation“. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.9 (Sept. 1989), pp. 1397–1405. ISSN: 0096-3518. DOI: 10.1109/29.31293.

[16] R. D. Poltmann. „Conversion of the delayed LMS algorithm into the LMS algorithm“. In: *IEEE Signal Processing Letters* 2.12 (Dec. 1995), pp. 223–. ISSN: 1070-9908. DOI: 10.1109/97.475854.

[17] V. Madisetti and D. Williams. *Digital Signal Processing Handbook on CD-ROM*. Taylor & Francis, 1999. ISBN: 9780849321351.

[18] R. B. Wallace and R. A. Goubran. „Parallel adaptive filter structures for acoustic noise cancellation“. In: *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*. Vol. 2. May 1992, 525–528 vol.2. DOI: 10.1109/ISCAS.1992.230139.

[19] N. Bershad. „On the optimum gain parameter in LMS adaptation“. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.7 (July 1987), pp. 1065–1068. ISSN: 0096-3518. DOI: 10.1109/TASSP.1987.1165232.

[20] T. I. Haweel. „A simple variable step size LMS adaptive algorithm“. In: *International Journal of Circuit Theory and Applications* 32.6 (2004), pp. 523–536. ISSN: 1097-007X. DOI: 10.1002/cta.294.

[21] *Acoustics – Measurement of room acoustic parameters – Part 1: Performance spaces*. Standard. Geneva, CH: International Organization for Standardization, June 2009.

[22] S. G. Sankaran and A. A. Beex. „Normalized LMS algorithm with orthogonal correction factors“. In: *Signals, Systems amp; Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on*. Vol. 2. Nov. 1997, 1670–1673 vol.2. DOI: 10.1109/ACSSC.1997.679186.

[23] S. Elliott, I. Stothers, and P. Nelson. „A multiple error LMS algorithm and its application to the active control of sound and vibration“. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.10 (Oct. 1987), pp. 1423–1434. ISSN: 0096-3518. DOI: 10.1109/TASSP.1987.1165044.

[24] Y. Huang, J. Benesty, and J. Chen. *Acoustic MIMO Signal Processing*. Signals and Communication Technology. Springer, 2006. ISBN: 9783540376309.

[27] J. Lorente et al. „Real-time adaptive algorithms using a Graphics Processing Unit“. In: *Waves* 2012.4 (2012), pp. 59–68.

[28] J. Lorente et al. „GPU Implementation of Multichannel Adaptive Algorithms for Local Active Noise Control“. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.11 (Nov. 2014), pp. 1624–1635. ISSN: 2329-9290. DOI: 10.1109/TASLP.2014.2344852.

[29] J. Lorente et al. „The frequency partitioned block modified filtered-x NLMS with orthogonal correction factors for multichannel Active Noise Control“. In: *Digital Signal Processing* 43 (2015), pp. 47 –58. ISSN: 1051-2004. DOI: 10.1016/j.dsp.2015.05.003.

[30] E. Nogues, R. Le Bidan, and D. Pastor. „Active Noise Control with digital PDM MEMS mics“. In: *2015 International Symposium on Consumer Electronics (ISCE)*. June 2015, pp. 1–2. DOI: 10.1109/ISCE.2015.7177788.

[36] E. Hogenauer. „An economical class of digital filters for decimation and interpolation“. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29.2 (Apr. 1981), pp. 155–162. ISSN: 0096-3518. DOI: 10.1109/TASSP.1981.1163535.

[39] G. Ballou. „Chapter 26 - {VU} Meters and Devices“. In: *Handbook for Sound Engineers (Fourth Edition)*. Fourth Edition. Oxford: Focal Press, 2008, pp. 995 –1009. ISBN: 978-0-240-80969-4. DOI: 10.1016/B978-0-240-80969-4.50030-4.

[42] A. Szarvas and L. Sujbert. „Efficiency Testing of Active Noise Control by Acoustic Field Modeling“. In: *Periodica Polytechnica Electrical Engineering and Computer Science* 59.4 (2015), pp. 147–159. DOI: 10.3311/PPee.8453.

Elektronikus forrásanyagok

[5] URL: https://en.wikipedia.org/wiki/File:Active_Noise_Reduction.svg.

[11] Nvidia Corporation. *Pascal GP100 GPU Whitepaper*. 2016. URL: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[12] Focusrite. *RedNet PCIe System Performance Information*. 2016. URL: https://us.focusrite.com/rednet-pcie-system-performance-information.

[25] Nvidia Corporation. *CUDA C programming guide PG-02829-001_v8.0*. Sept. 2016. URL: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[26] Nvidia Corporation. *Maxwell GM204 GPU Whitepaper*. 2014. URL: http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF.

[31] ST Microelectronics. *Application Note AN4426 – Tutorial for MEMS microphones*. Jan. 2014. URL: http://www.st.com/content/ccc/resource/technical/document/application_note/46/0b/3e/74/cf/fb/4b/13/DM00103199.pdf/files/DM00103199.pdf/jcr:content/translations/en.DM00103199.pdf.

[32] URL: http://www.electronicproducts.com/Sensors_and_Transducers/Sensors_and_Transducers/MEMS_microphone_offers_top_performance_with_top-port_design.aspx.

[33] URL: https://en.wikipedia.org/wiki/File:DeltaSigma2.svg.

[34] URL: https://en.wikipedia.org/wiki/File:Pulse_density_modulation.svg.

[35] URL: http://users.ece.utexas.edu/~bevans/courses/rtdsp/lectures/10_Data_Conversion/AP_Understanding_PDM_Digital_Audio.pdf.

[37] URL: http://en.wikipedia.org/wiki/File:Cic-decimator.svg.

[38] Analog Devices. *Technical Article MS-2472 – Analog and Digital MEMS Microphone Design Considerations*. 2013. URL: http://www.analog.com/media/en/technical-documentation/technical-articles/Analog-and-Digital-MEMS-Microphone-Design-Considerations-MS-2472.pdf.

[40] Maxim Integrated. *MAX11046 datasheet*. 2013. URL: https://datasheets.maximintegrated.com/en/ds/MAX11044-MAX11056.pdf.

[41] Analog Devices. *AD5676R datasheet*. 2014. URL: http://www.analog.com/media/en/technical-documentation/data-sheets/AD5672R_5676R.pdf.