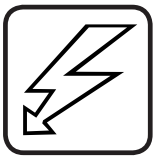

**Budapesti Műszaki és Gazdaságtudományi Egyetem**

**Villamosmérnöki és Informatikai Kar**



**Méréstechnika és Információs Rendszerek Tanszék**

# Szenzorhálózatos aktív zajcsökkentő rendszer felhasználóbarát kezelői felülettel

Készítette: **Székely Gábor**

E-mail: szg@ip6.hu

Konzulens: Dr. Sujbert László docens

2008

## Nyilatkozat

Alulírott **Székely Gábor**, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Budapest, 2008. május 15.

Székely Gábor

# Tartalomjegyzék

# Kivonat

A közelmúltban a *Méréstechnika és Információs Rendszerek Tanszéken* kifejlesztettek egy jelmodell alapú algoritmusra épülő aktív zajcsökkentő rendszert, amelynek hibamikrofonjai egy szenzorhálózat egy-egy elemén helyezkednek el. A szenzorhálózat elemei (az ún. mote-ok) egymással és a bázisállomással rádió segítségével kommunikálnak, működésük egyébként autonóm. A bázisállomás által gyűjtött adatokat egy lebegőpontos processzort tartalmazó jelfeldolgozó kártya fogadja. Itt fut a zajcsökkentő algoritmus is. A rendszer hibátlanul működik, és alkalmas az aktív zajcsökkentő rendszerekben és szenzorhálózatokban releváns méréstechnikai, jelfeldolgozási problémák vizsgálatára. Ugyanakkor működtetése sok nehézséget rejt magában. A sikeres üzembe helyezés megköveteli mind az alkalmazott fejlesztőeszközöknek, mind magának a programkódnak a mélységi ismeretét. A jelfeldolgozást két különálló DSP program valósítja meg (egyik az identifikációt, másik a zajelnyomást végzi). A rendszer átkonfigurálása, paramétereinek hangolása csak forráskódszinten lehetséges, így minden esetben a program újrafordítását és fejlesztőkártyára való letöltését vonja maga után.

Megoldásként a jelfeldolgozó algoritmusokat egyetlen, a rendszer összes funkcióját megvalósító DSP programban egyesítettem. A programmodulok strukturális átalakítása révén lehetővé tettem az üzemmódok közötti rugalmas átváltást, illetve a működésükbe való futás közbeni beavatkozást.

Napjainkban egy korszerű rendszer szinte elképzelhetetlen grafikus felhasználói interfész nélkül. A gördülékeny munkavégzés elősegítése végett a rendszert korszerű felhasználói felülettel láttam el. Ennek kivitelezése a GUI-tervezés ismert alapelveinek figyelembevételével véve történt.

A diplomaterv kidolgozása során sikerült egy olyan, korszerű technológiákon alapuló rendszerig eljutni, amelynek segítségével a problémát jól ismerő, de az alkalmazott eszközök fejlesztésében járatlan felhasználó is képes kísérletező munkát végezni.

# Abstract

Recently, a periodic signal model based active noise control (ANC) system was developed at the Department of Measurement and Information Systems. Particularly, the noise sensing microphones are placed on so-called 'motes', which are the elements of a wireless sensor network. These autonomous sensors communicate with each other and the base station by radio. A digital signal processor board equipped with a floating point processing unit receives the data collected by the base station, and the ANC algorithms are implemented on this board. The system works properly, and is suitable for the investigation of problems which appear in ANC systems and wireless sensor networks. However, setting up and running the system has some difficulties. Successful installation requires deep knowledge on the implemented algorithms and development tools applied. Furthermore, signal processing algorithms fall into two detached tasks: one for the identification of the acoustic system, and another for noise canceling. System reconfiguration or tuning of the working parameters is possible only on source code level, so each change implies a complete project rebuilding and program downloading.

To clear up the problems mentioned above, I merged/integrated the signal processing algorithms into a single project, realizing all the operational functions in a unified system. By the structural modifications made on the program modules, it is now possible to switch between the main operating modes, and modify the working parameters.

Nowadays, a modern system is hardly conceivable without a graphical user interface. To assist and advance the fluent work with our system, an up to par graphical user interface was fabricated, too. In pursuance of User Interface implementation, the well-known suggestions and principles of GUI design were considered.

In course of mapping out this thesis, I managed to establish a system based on modern technologies, which serves as an experimental system, even at expert level. However, the system can be managed by users, who are not familiar with the applied development tools.

# 1. fejezet

# Bevezetés

## 1.1. A diplomaterv célja

A Méréstechnika és Információs Rendszerek Tanszéken megismerkedtem az aktív zajcsökkentés és a szenzorhálózatok elméleti alapjaival, kérdéseivel, majd – önálló labor keretében – egy már működő rendszer tanulmányozásában és továbbfejlesztésében vettem részt.

Az elméleti felkészülés során megismertem az aktív zajcsökkentés eljárásait, leggyakrabban használt algoritmusait és struktúráit, illetve elsajátítottam az ehhez szükséges digitális jelfeldolgozási ismereteket. Megismerkedtem a digitális jelfeldolgozó processzorokkal és azok alkalmazásával, valamint a vezeték nélküli szenzorhálózatok jellemzőivel és a velük kapcsolatban felmerülő problémák körével.

Diplomatervem alapvető célja egy már létező – kísérleti – aktív zajcsökkentő rendszer ismertetése, továbbfejlesztése, hiányosságainak kijavítása, és felhasználóbarát kezelői felülettel való ellátása.

Jelen dolgozatnak nem célja valamennyi, a rendszerrel kapcsolatosan felmerülő problémára és hiányosságra generális megoldást adni. Célja viszont, hogy könnyebbé tegye a zajcsökkentő rendszer tanulmányozását és a vele végzett tudományos kísérleteket, valamint egy lépés megtétele afelé, hogy a kísérleti aktív zajcsökkentő rendszerből valós körülmények között is jól használható rendszer váljék.

Ezen írás célja továbbá, hogy összefoglalja és rendszerezze mindazon ismeretanyagot, amely az aktív zajcsökkentéssel és annak kapcsolódó területeivel összefüggésbe hozható, ezáltal hasznára legyen a szakirány jelenlegi és leendő hallgatóinak.

## 1.2. A dolgozat felépítése

Diplomatervem első részében áttekintem a téma elméleti alapjait és történelmének alapvető állomásait, bemutatom a legalapvetőbb struktúrákat, röviden felvázolom a modellillesztés problémakörébe tartozó feladatok két alapvető csoportját – az identifikációt és az adaptációt –, valamint röviden kitérek a jelfeldolgozó processzorok legfontosabb architekturális jellemzőire.

A második részben ismertetem a diplomaterv alapjául szolgáló kísérleti zajcsökkentő rendszer felépítését és működését, kitérve az abban alkalmazott hardver és szoftver komponensek jellemzőire, a rendszer egyes részegységeivel kapcsolatban felmerülő problémakörökre. Szó esik a napjainkban egyre népszerűbb elosztott adatgyűjtésről és az azt megvalósító szenzorhálózatokról, esettanulmányként bemutatva az alaprendszerben alkalmazott technikákat és megoldásokat. Itt kerül tárgyalásra maga a jelfeldolgozó algoritmus, illetve annak működése, valamint megemlítem a rendszerrel kapcsolatban felmerülő legfontosabb hiányosságokat, amelyek implikálják a továbbfeljesztésre való igényeket, törekvéseket.

Ennek a továbbfejlesztésnek az irányvonalát jelölöm ki a harmadik részben, kitűzve a legfontosabb célokat, majd ismertetve a megoldásukra irányuló főbb lehetőségeket és azok sajátosságait. Itt történik a tervezési döntések meghozatala és azok indoklása, valamint a rendszerterv elkészítése és a funkcionális blokkvázlat bemutatása.

A következő rész a rendszertervben foglaltak konkrét megvalósítását ismerteti. Bemutatja az alkalmazott technológiákat és azok legfontosabb jellegzetességeit, majd ezen ismeretanyagra támaszkodva az egyes funkcionális elemek realizálásának és a köztük lévő kapcsolatok kialakításának részleteit. Ezután összegzem a továbbfejlesztett rendszer tervezése és kivitelezése során szerzett tapasztalatokat, és értékelem annak teljesítőképességét. Majd – mintegy kitekintésképp – sorra veszem a továbblépés lehetséges irányait, és megfontolásokat teszek ezekkel kapcsolatban.

A dolgozat függelékében röviden összefoglalom a kész rendszer működtetésével és kezelésével kapcsolatos legfontosabb ismereteket, kitérek a szenzorhálózat élesztési mechanizmusára, a mótok felprogramozására, ismertetem a *TinyOS* és a kapcsolódó fejlesztőeszközök installálásának menetét az érdeklődő olvasó számára, majd – ízelítőül – bemutatom az eredeti rendszer (felhasználói felület nélküli) élesztési mechanizmusát és használatát.

# 2. fejezet

# Elméleti áttekintés

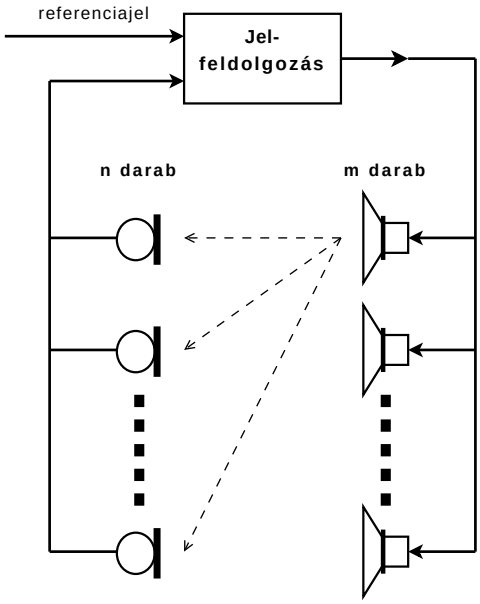
## 2.1. Az aktív zajcsökkentés helye és szerepe

A környezetünkből érkező zajok nemcsak zavaróak lehetnek, hanem hallásküszöb-emelkedéshez, illetve közvetett módon egyéb nemkívánatos élettani elváltozásokhoz – pl. magas vérnyomás – is vezethetnek. Az ipari fejlődés, a városiasodás és a motorizáció egyre nagyobb elterjedésével a velük járó nemkívánatos zaj napjainkban egyre nagyobb jelentőségű problémává válik. A nemkívánatos zaj elleni küzdelem terén a hagyományos megoldást a *passzív technikák* alkalmazása jelenti. Ezek lényege, hogy a zajforrást a védeni kívánt térrésztől hanggátló (hangelnyelő, hangvisszaverő) anyagból készült szerkezeti elemekkel választják el. Ez történhet a zajforrás, vagy a megvédeni kívánt terület teljes vagy részleges elkerítésével.

A passzív megoldások kétféle csoportja létezik: a *reaktív* és a *rezisztív* tompítók. Előbbiek terelőelemek – gátak és csövek – segítségével megváltoztatják az akusztikus impedanciát, ezáltal „elrontják" az akusztikus csatolást a forrás és a védendő térrész között; míg utóbbiak zegzugos szerkezetük segítségével felemésztik a zaj energiáját. Reaktív tompítók például a belső égésű motorok kipufogórendszereiben használt hangtompító dobok és furulyák, míg rezisztív tompító például a zenei stúdiók felvételi helyiségeinek falborítása.

Ezek a passzív módszerek széles frekvenciasávban igen jó csillapítást nyújtanak. Hatásosságuk feltétele, hogy a védőeszköz fizikai vastagsága nagyobb legyen a hang hullámhosszának negyedénél. Nagy hullámhosszak esetén a hatásosság csak a védőfalak méreteinek – és így tömegének – növelése árán őrizhető meg. Amellett, hogy sok alkalmazás nem teszi lehetővé a méret és a tömeg tetszőleges mértékű növelését (pl. egy fülvédő, vagy más mobil berendezések esetén), jelentős anyagköltséggel is számolni kell. Így a kisfrekvenciás tartományban a passzív módszerek nem hatékonyak. Ezek a hátrányos tulajdonságok teremtenek létjogosultságot az *aktív* zajcsökkentés (a továbbiakban ANC – *Active Noise Control*) számára, illetve lehetőségeinek vizsgálatára.

Mivel az akusztikai rendszerek széles dinamikatartományban *lineárisnak tekinthetők*, alkalmazható bennük a *szuperpozíció* tétele. Az aktív zajcsökkentő rendszer egy *elektroakusztikus* vagy *elektromechanikus* rendszer, amely a szuperpozíció elvét kihasználva próbálja kioltani a nemkívánatos zajokat. Ideális esetben a rendszer olyan *ellenzajt* generál, amelynek amplitúdója megegyezik, de fázisa ellentétes az eredeti zajéval. Megjegyzendő,




2.1. ábra. ANC rendszer általános fizikai felépítése

hogy – bár most csak akusztikai rendszerekkel foglalkozunk –, ez az elv bármilyen – lineáris – fizikai rendszerben használható rezgések kioltására.

Az aktív zajcsökkentő rendszerek általános fizikai felépítését a 2.1. ábrán láthatjuk, míg a 2.2. ábra az analitikus tárgyalás során használatos linearizált modellt mutatja. Bár az akusztikai rendszerekben nem különbségképzés, hanem összegzés szerepel, az ábrán jelölt különbségképzés nem jelent megszorítást, hiszen a rendszer kimenőjelét minden további nélkül megszorozhatjuk $-1$-gyel. A szabályozástechnikával foglalkozó irodalmakban azonban a különbségképző a megszokott elem.

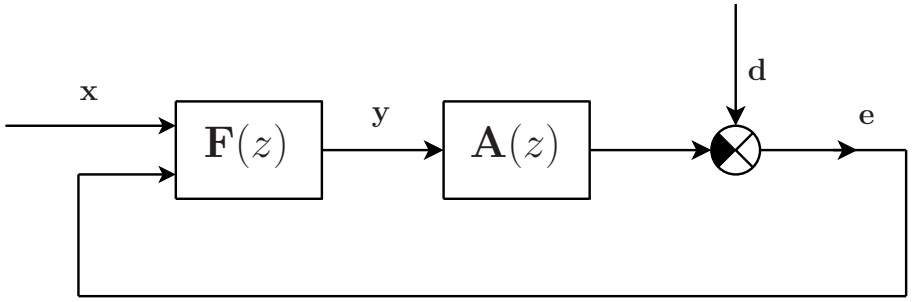
A rendszer működése röviden a következő: A jelfeldolgozó egység a hozzá kapcsolódó mikrofonok segítségével érzékeli a zajt. Egyes alkalmazásokban a zajcsökkentő algoritmusnak rendelkezésére áll még egy, az eredeti zajjal korrelált ún. *referenciajel* is. Ezen információk alapján a beavatkozó forrásokból – esetünkben a hangszórókból – kiadandó ellenzaj aktuális értéke a zajcsökkentő algoritmus alapján határozható meg. A zajforrásból érkező jelet ez a hangszórókból kiadott „inverz zaj" nyomja el úgy, hogy a térben a hanghullámok interferenciájának következményeként a hibamikrofonok környezetében *csendes zónák* alakulnak ki.

Ezekkel a rendszerekkel igen jó zajelnyomást érhetünk el, ám nem szabad elfeledkeznünk a módszer korlátairól sem. A hibamikrofonok körül kialakuló csendes zónák sugara a hullámhosszal arányos (körülbelül a zaj hullámhosszának negyedével egyezik meg), így a frekvencia növekedésével egyre csökken. Ez nagyjából be is határolja az aktív zajcsökkentés működési frekvenciatartományát: a módszer az infrahangoktól 1-2 kHz-ig alkalmazható eredményesen. Nem szabad figyelmen kívül hagyni azt sem, hogy az alkalmazási területek jelentős részénél a mérőmikrofonok a fülektől viszonylag távol helyezhetők csak el.

Maga az elv nem újdonság; az első ötletek 1936-ban láttak napvilágot. Az első tervek *Paul Lueg* nevéhez fűződnek. Ötletéből szabadalom is született, amely vázolta a destruktív elnyomás alapelvét, valamint egy rendszertervet is bemutatott, amely egy hangszórót és

egy mikrofont tartalmazott [10]. A sikeres kivitelezés azonban a kor technikai színvonalán nehézségekbe ütközött, így az elv gyakorlati alkalmazására nem került sor.

Az utóbbi néhány évtizedben sokféle megoldás látott napvilágot. A probléma megoldására eleinte *analóg áramköröket* alkalmaztak. Ilyen struktúrát ismertet *Harry Olson* és *Everet May* 1953-ban megjelent cikkükben [11]. Az ő rendszerükben a hibamikrofon jelét egy elektromos szűrővel megszűrve áll elő a beavatkozó hangszóró irányítójele. A kibocsájtott hanghullám az akusztikai közegen át eljut a hangszórótól a mikrofonig, létrehozva ezzel a visszacsatolást. Az igazi áttörést azonban a digitális jelfeldolgozó processzorok (DSP-k) ‘80-as évekbeli rohamos fejlődése és tömeges elterjedése jelentette.




2.2. ábra. Aktív zajcsökkentő rendszer linearizált modellje

A 2.2. ábrán látható linearizált modell jelöléseinek kifejtése:

- $\mathbf{F}(z)$: a zajelnyomó struktúra
- $\mathbf{A}(z)$: a fizikai rendszer átviteli függvénymátrixa
- **x**: a referenciajel, amely az elnyomandó jellel korrelált, és a zajelnyomó struktúra felhasználhatja (de nem feltétlenül használja fel)
- **y**: a zajcsökkentő struktúra kimenőjele
- **d**: az elnyomandó jel, amelyet a struktúra bemenőjelének tekintünk
- **e**: különbségi jel vagy hibajel

## 2.2. Az identifikációról

Amint azt a 2.2. ábrán is láttuk, az aktív zajcsökkentő rendszerekben jelen van egy *másodlagos út*nak is nevezett átvitel, amelyet az A/D-átalakító, a D/A-átalakító, az analóg erősítő fokozatok, a hangszóró és a mikrofon átvitele, valamint az akusztikai közeg átvitele határoz meg. Ez egy csatornás – egy hibamikrofont és egy beavatkozó hangszórót tartalmazó – rendszer esetén egy átviteli függvény, többcsatornás rendszer esetén pedig egy átviteli függvény-mátrix. A zajcsökkentő rendszerek legtöbbjének helyes működéséhez ezt valamilyen pontossággal ismerni kell. Tehát a másodlagos út átvitelét valamilyen módon mérni, identifikálni szükséges.

Lineárisnak feltételezett rendszer identifikációja során a rendszert reprezentáló átviteli függvény meghatározása a cél; a rendszer – ismertnek feltételezett – bemeneti és kimeneti jelei alapján. Időben állandó objektumot vizsgálunk, így a rendszert kellően hosszú ideig megfigyelhetjük, sok adat begyűjtésére van lehetőség. Nem a rövid mérési idő, hanem a nagy pontosságú eredmény a cél.

Lévén általános probléma, az átviteli függvény mérésére többféle módszer is kínálkozik. Ezek mindegyike bír bizonyos – természetéből fakadó – előnyökkel ill. hátrányokkal. Azt, hogy melyiket érdemes alkalmazni, mindig az adott feladat határozza meg. Alapvetően háromféle megközelítés különíthető el:

**Keskenysávú gerjesztőjel, kiértékelés pontonként:** Ilyen a léptetett szinusz vagy a lassan söprő szinusz. Minden mérési pontban egyetlen szinuszjellel gerjesztjük a rendszert, és az erre adott választ mérjük, aminek ismeretében kiszámítható az átviteli függvény értéke az adott frekvencián.

Szelektív mérési módszer, így könnyen felfedhető vele a rendszer nemlineáris viselkedése. Előnye a koncentrált jelteljesítmény, és az ennek köszönhető jó jel-zaj viszony. A mérés varianciája kicsi; az eredmény nagy pontosságú lehet. Az előnyökért a relatíve hosszú mérési idővel fizetünk.

**Szélessávú mérőjel, kiértékelés FFT-vel:** Ilyen a multiszinusz, a véletlen zaj (pl. fehér zaj), a pszeudo-véletlen zaj (sűrű, de vonalas spektrumú), vagy az impulzus gerjesztés (ez utóbbit inkább mechanikai rendszerekben alkalmazzák). A gerjesztés spektrumát természetesen ismertnek tételezzük fel.

A módszer előnye, hogy az FFT-s kiértékelés gyors mérést tesz lehetővé, viszont az alacsony teljesítmény-sűrűség, a dekoncentrált jelteljesítmény rossz jel-zaj viszonyt eredményezhet. A variancia igen nagy lehet. Ennek javítása érdekében több ($10 \ldots 100$) mérés eredményét célszerű átlagolni.

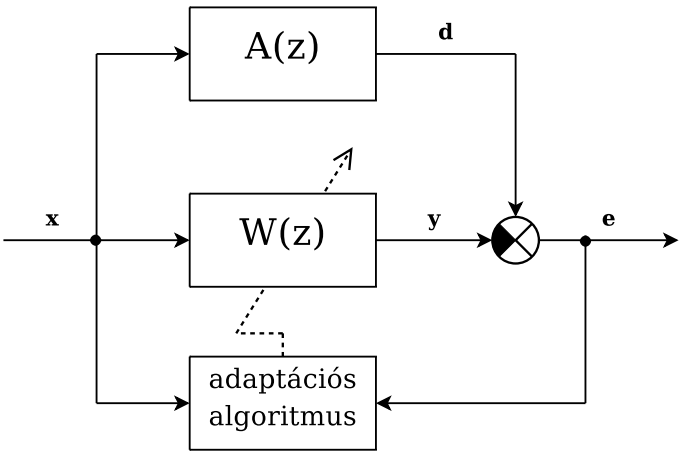
Multiszinuszos gerjesztőjel esetén további problémák forrásai lehetnek az azonos fázishelyzetbe kerülő komponensek. Ebben az esetben ugyanis a gerjesztőjel amplitúdója egyes időpillanatokban igen nagy lehet, ami az A/D ill. D/A átalakítókat, vagy az analóg erősítő fokozatokat telítésbe vezérelheti. Probléma forrása lehet az analóg áramköri részek esetleges intermodulációs torzítása is.

**Az átviteli út modellezése adaptív FIR szűrővel:** E szerint a megközelítés szerint egy adaptív szűrő együtthatóit hangolva próbáljuk előállítani az identifikált rendszer átvitelének *becslőjét*. A folyamatot a 2.3. ábra hivatott szemléltetni. Állandósult állapotban – ideális esetben – a hiba zérus, tehát azonos gerjesztésre $A(z)$ és $W(z)$ kimenete megegyezik. Ez csak $W(z) = A(z)$ esetén lehetséges, azaz $W(z)$ szűrő felhasználható $A(z)$ modelljeként.

Az adaptív szűrő által realizált modell számos tekintetben különbözhet az identifikált rendszertől, ez azonban – megfelelő tervezés esetén – a gyakorlati alkalmazásokban nem okoz problémát. A különbség okai lehetnek például, hogy

- végtelen impulzusválaszú (IIR) rendszert véges impulzusválaszú (FIR) szűrővel modellezünk,
- bizonyos frekvenciákon a gerjesztőjel nem vezérli ki megfelelően a rendszer bemenetét,
- zérus hiba általában eleve nem érhető el (lásd Wiener-megoldás).

Gerjesztőjelként szélessávú jelet célszerű alkalmazni, hiszen $W(z)$ modell csak azokon a frekvenciákon adaptálódik, ahol a gerjesztőjelnek van komponense. A gyakorlatban ezért mérőjelként általában fehér zajt alkalmazunk.




2.3. ábra. Adaptív szűrőn alapuló identifikációs eljárás blokkvázlata

## 2.3. Az adaptív jelfeldolgozásról

Az aktív zajcsökkentő rendszerek visszacsatolt struktúrát alkotnak. Emiatt igen érzékenyek a visszacsatoló ág – a másodlagos út – átviteli függvényény-mátrixának megváltozására. Ezt az ágat esetünkben az A/D- ill. D/A-átalaktó, az analóg erősítő fokozat, a hangszóró, a mikrofon, valamint a kettő közötti akusztikai közeg alkotja. A másodlagos út átvitelének megváltozásához leginkább ez utóbbi járul hozzá. Az akusztikai tér átviteli tulajdonságai időben változnak. Ezt nagyon sok tényező befolyásolja, amelyek közül példaként említhető a hőmérséklet, a légnyomás, a páratartalom, vagy a tereptárgyak helyzete, elhelyezkedése (egy szobában pl. a nyílászárók nyitott vagy csukott állapota). Maga az akusztikus zaj sem stacionárius; változhat a fázisa, amplitúdója, frekvenciatartalma. Emiatt olyan adaptív algoritmusra van szükség, amely az időben változó paramétereket is képes követni.

Adaptációra akkor is szükség van, ha nem az akusztikai tér átvitele változik, hanem maga az elnyomandó jel. Egyszerű példa erre az az eset, amikor egy személygépkocsi belső égésű motorja által keltett zajhatást kívánjuk elnyomni, és a motor fordulatszáma megváltozik. Ilyen esetben a zajelnyomó struktúra csak akkor működhet hatékonyan, ha ezeket a változásokat követni tudja.

A követés valós idejű követelményt támaszt: eliminálódjon gyorsan a hiba, az „együttmozgás” a fontos. A rendszerparaméterek pontos megfeleltethetősége a valóság és a modell között nem elsőrendű szempont, sőt sokszor a könnyebben kezelhető struktúrára és paraméterkészletre térés a cél – úgy, hogy a lényeges működési tartományban hasonlóan viselkedő modellt kapjunk.

Az adaptív szűrőkön alapuló zajcsökkentő rendszerek és algoritmusok megjelenését a ‘80-as években elterjedő korszerű jelfeldolgozó processzorok tették lehetővé. Ezek a rendszerek diszkrét rendszerek, amelyek a fizikai világgal A/D- és D/A-átalaktókon keresztül tartják a kapcsolatot (lásd a 2.4. ábrát). Megjelenésükkel az aktív zajcsökkentéssel foglalkozó fejlesztések és kutatások új lendületet vettek.


2.4. ábra. Digitális jelfeldolgozó rendszer blokkvázlata

## 2.4. A jelfeldolgozó processzorokról

A digitális jelfeldolgozás napjaink egyik leggyorsabban és legdinamikusabban haladó szakterülete. Alapvetően formálja és befolyásolja a *XXI.* század tudományának fejlődését; a tudományágak széles palettájára van hatással.

A digitális jelfeldolgozó algoritmusok sokszor megkívánják, hogy nagy számú matematikai műveletet végezzük el *gyorsan* egy adathalmazon. Sok alkalmazás ugyanis követelményeket, megkötéseket támaszt a késleltetésekkel és a feldolgozási idővel szemben – a

rendszer csak akkor marad működőképes, ha a jelfeldolgozási műveletek ezen határidőkön belül elvégezhetők.

Bár digitális jelfeldolgozási feladatokat általános célú mikroprocesszorokkal is megvalósíthatunk, egy direkt ilyen algoritmusok futtatására kifejlesztett a digitális jelfeldolgozó processzorral, vagy röviden *DSP*-vel (*Digital Signal Processor*) felépített rendszer sokkal hatékonyabban (kisebb késleltetéssel vagy kisebb költségek befektetése mellett) teljesíti ugyanezt a feladatot. A DSP-k is mikroprocesszorok, és legtöbb esetben tartalmaznak minden olyan funkciót, amikkel egy általános célú processzor bír. A DSP-ket azonban – mind architekturális kialakításukat, mind utasításkészletüket tekintve – kifejezetten valós idejű jelfeldolgozási feladatokra tervezték.

### 2.4.1. Architektúra

Memóriakiosztásukat tekintve a DSP-k az általános célú processzorok világában hagyományosnak számító Neumann-architektúra helyett *Harvard*, vagy ritkábban *módosított von Neumann* architektúrájúak. Ez azt jelenti, hogy külön program- és adatmemóriával rendelkeznek. Sőt, ez utóbbiból egyes DSP-k többel is rendelkeznek. Ez az osztott elrendezés, és a kapcsolódó külön buszrendszerek lehetővé teszik, hogy egy utasításciklus alatt egyszerre lehessen hozzáférni az utasításkódhoz, és akár több operandushoz is – elősegítve ezzel az egyciklusos utasítás-végrehajtást. Meg kell jegyezni, hogy ez a párhuzamos elérhetőség legtöbb esetben csak az *on-chip* memóriára igaz (ennek mérete általában néhány megabit; a külső memóriához fordulás csak szekvenciálisan és nagyobb késleltetéssel történhet. Programozás során tehát figyelemmel kell eljárnunk az adatstruktúráink memóriában való elhelyezését illetően, ha ki akarjuk használni ezen kedvező tulajdonságokat.

A párhuzamosítás architekturális szinten és az utasításvégrehajtás[1], sőt az utasításkészlet szintjén is megjelenik. Az utasításdekódolás és -végrehajtás gyorsítását többszintű *pipeline* segíti. Az utasításvégrehajtás folyamata kisebb elemi lépésekre tagolódik (pl. fetch, utasításdekódolás, operandusok előkészítése, stb.), és az egymás után következő utasítások elemi lépései szimultán hajtódhatnak végre. Vannak olyan processzorok, amelyekben a szükséges funkcionális blokkok, erőforrások (pl. ALU, hardveres szorzó, stb.) több példányban is rendelkezésre állnak, ami által több utasítás végrehajtása válik lehetségessé egyetlen gépi ciklus alatt. Ezt nevezik *szuperskalár* architektúrának. Az erőforrások további kihasználása és a prhuzamosítás további fokozása – ezáltal további sebességnövekedés elérése – érdekében abban az irányban is folynak kutatások, hogy az utasítások végrehajtása a programban rögzítettől eltérő sorrendben[2] is történhessen.

Ahogy az előző bekezdésben is megemlítettük, a párhuzamos műveletvégzés előnyeit akkor tudjuk igazán kihasználni, ha a párhuzamosság az utasításkészlet szintjén is megjelenik. Tipikus példája ennek a $MAC$[3] utasítás, amelyet szinte minden jelfeldolgozó processzor végre tud hajtani. Ilyenkor a processzor két szám összeszoroz, majd az eredményt hozzáadja egy harmadikhoz, és a következő két tényezőt beolvassa a memóriából a szorzóegység bemeneti regisztereibe *egyetlen utasításciklus alatt*. Leggyakoribb alkalmazása a

[1] ILP – instruction level parallelism
[2] Out-of-order execution
[3] Multiply-and-Accumulate

FIR szűrők megvalósításánál alkalmazott *konvolúció-számítás*, és egyes *mátrixműveletek*. Ennek kihasználásával egy FIR szűrés végrehajtása a szűrő tap-számával összemérhető számú utasításciklus alatt elvégezhető.

Egyes processzorokban adat-szintű párhuzamosítási technikákat is alkalmaznak. Az egyik ilyen technika a *SIMD*[4], amelynek előnyeit olyan esetekben tudjuk kihasználni, amikor ugyanazt a műveletet kell elvégezni egy nagy adathalmaz minden elemén.

### 2.4.2. Aritmetika

A modern jelfeldolgozó processzorok az összeadást, kivonást, szorzást, bitléptetést és a logikai műveleteket fejlett hardveres műveletvégző egységekkel támogatják. Ennek köszönhetően ezek az utasítások általában egyetlen gépi ciklus alatt végrehajtjatók. Gyakran rendelkezésre korlátozott pontosságú osztást megvalósító egység is. Az osztás azonban legtöbb esetben időkritikus művelet, ezért használatát célszerű elkerülni, amennyiben lehetséges.

Az aritmetika egyik leglényegesebb jellemzője a *szóhossz*, ami megmutatja, hogy a műveletvégző egységek hány bites operandusokkal dolgoznak, és az eredmény milyen bitszámú regiszterben áll elő. A processzorok *számábrázolás* tekintetében is különbözhetnek egymástól. Megkülönbözetünk *fixpontos* és *lebegőpontos* aritmetikával ellátott processzorokat. Utóbbi esetben az szolgál alapul, hogy a valós számok felírhatók normálalakban: normálalakban: $m \cdot a^c$, ahol $m$ a *mantissza*, $a$ az *alap*, és $c$ a *karakterisztika*. $m$ valós szám, $c$ előjeles egész, $a > 1$ és rögzített. Az alap lehet pl. 2, 8, 10, 16; ez adott, tehát nem kell letárolni. A rendelkezésre álló szóhozz tehát mantisszára és karakterisztikára tagolódik. A mantissza általában előjel-abszolútértékes, normalizált, aminek köszönhetően a „kis" és a „nagy" számok azonos pontossággal ábrázolhatók (értékes jegyeik száma megegyezik), ami által a műveletvégzés során nagy dinamikatartomány érhető el. A modern lebegőpontos processzorok megfelelnek az *IEEE 754*-es szabványban rögzített formátumoknak. Sok valós alkalmazásban azonban nincs szükség a lebegőpontos számábrázolás által nyújtott dinamikatartományra, így a legtöbb DSP-t fixpontos aritmetikával látják el. Ennek az áramköri felépítése ugyanis lényegesen egyszerűbb, ami nagyobb működési sebességet tesz lehetővé.

### 2.4.3. Címzési módok

Az általános célú processzorokhoz hasonlóan a DSP-kben is lehetőség van *direkt* és *indirekt* címzésre. A párhuzamos műveletvégzés lehetőségeinek maximális kihasználásához azonban a DSP-k *külön címaritmetikai egység*gel is rendelkeznek, amely a címregiszterek értékének módosítását automatikusan elvégzi. Lehet ez a módosítás egyszerű inkrementálás vagy dekrementálás, vagy más, ún. ofszetregiszterekben tárolt értékekkel való korrekció.

Gyakori a modulo-címzés hardveres támogatása is, aminek révén plusz szoftveres címszámítások nélkül valósíthatjuk meg a jelfeldolgozási feladatok során gyakran használt cirkuláris puffereket[5], vagy adatfolyamok csúszóablakos feldolgozását. Ezáltal hatékonyan valósíthatunk meg FIR szűrést, vagy konvolúció- és korreláció-számítást.

[4] Single Instruction, Multiple Data

[5] A cirkuláris puffer olyan – állandó méretű – átmeneti tároló, amelynek elemeit ciklikusan olvassuk ki, és ciklikusan írunk bele.

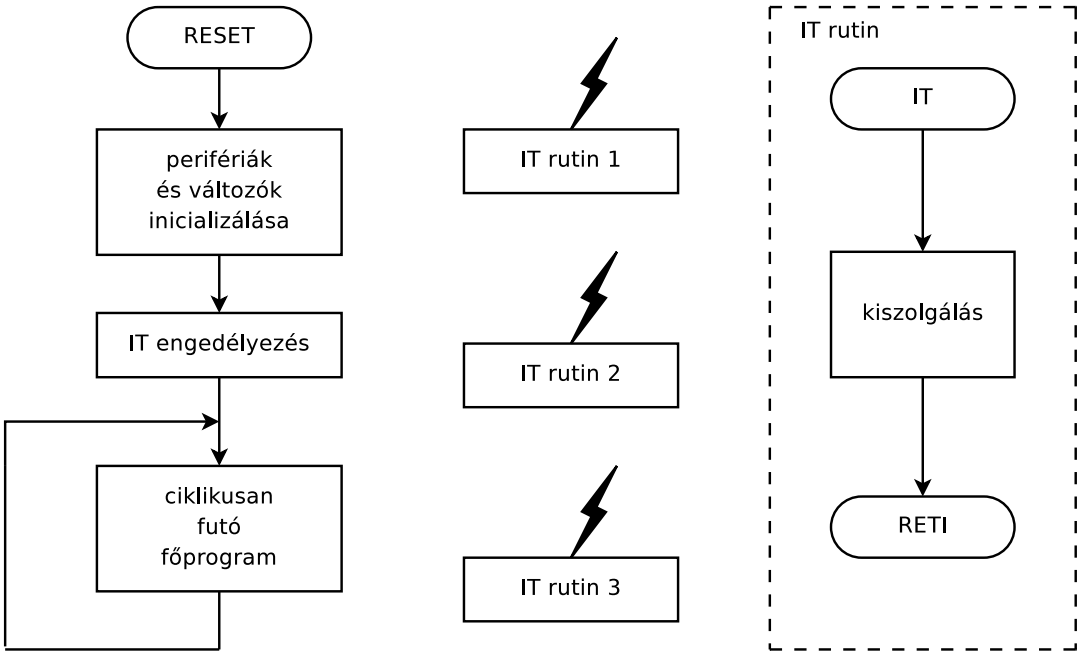
A jelfeldolgozó processzorok gyakran az ún. *többletszámítás nélküli cikluskezelés*t (Zero Overhead Loop) is támogatják, aminek köszönhetően a ciklus futása során nem szükséges a ciklusváltozó szoftverből történő ellenőrzése és kezelése. Ennek használatához általában ki kell jelölni a memóriában a ciklus elejét és végét, valamint beállítani a lépések számát. Ilyenkor valóban csak a ciklusmag utasításai hajtódnak végre iteratív módon.

A gyors FFT eljárások ún. *bitreverse* sorrendben állítják elő az eredmény bitjeit, így ennek könnyebb feldolgozhatósága érdekében a DSP-k címaritmetikai egysége általában támogatja a bitreverse címzési módot is.

### 2.4.4. Programszervezés

A jelfeldolgozó programok általános szerkezetét a 2.5. ábra mutatja. A főprogram a változókat és a perifériákat inicializáló programrésszel kezdődik. Ennek befejeztével kerül engedélyezésre globálisan az IT. A főprogram további része egy végtelen ciklus, amely lényegi jelfeldolgozási feladatot nem végez. Ennek futását időszakosan megszakítják a perifériák (pl. az A/D átalakító) kiszolgáló rutinjai. A helyes működéshez fontos, hogy a kiszolgálás még a kövtkező kérés érkezése előtt befejeződjön.

Tipikusan az A/D átalakító megszakítást kér, ha érvényes adat van a kimenetén. Ezek a megszakítások a mintavételi periódusidőnek megfelelő időközönként következnek be. A jelfeldolgozást végző rutin futási ideje így kritikus lehet. Előfordul, hogy a C fordító nem használja ki optimálisan a processzor képességeit. Ha a rutin futási ideje a megengedhetőnél nagyobbra adódik, érdemes megpróbálkozni a kritikus részek assembly nyelven történő implementálásával is, törekedve a gyorsítási és párhuzamosítási lehetőségek maximális kihasználására.




2.5. ábra. Jelfeldolgozó programok általános felépítése

## 2.5. ANC struktúrák

[1] Az aktív zajelnyomás alapproblémáira az évek során számos különböző megoldás született. Ezek általában a probléma különböző megközelítéseiből indulnak ki. Jelen fejezet két fontos struktúrát ismertet, röviden bemutatva azok legfontosabb sajátosságait.

### 2.5.1. Visszacsatolt (Feedback) struktúra

A visszacsatolt struktúra fontos sajátossága, hogy nem használja fel az $x$ referenciajelet. Blokkvázlatát a 2.6. *a)* ábra mutatja. A rendszer $d$ bemenőjele $x$-nek $A_1(z)$-vel való szűrése útján áll elő. $A_1(z)$ az ún *elsődleges út* átvitele, innen az index. $\hat{A}(z)$ a *másodlagos út* $A(z)$-vel jelölt átviteli függvényének a becslője, amelyet offline identifikációval határozunk meg. $H(z)$ adaptív szűrő, amelynek együtthatóit a hibajel pillanatnyi értéke alapján hangoljuk – valamilyen adaptációs algoritmus segítségével.

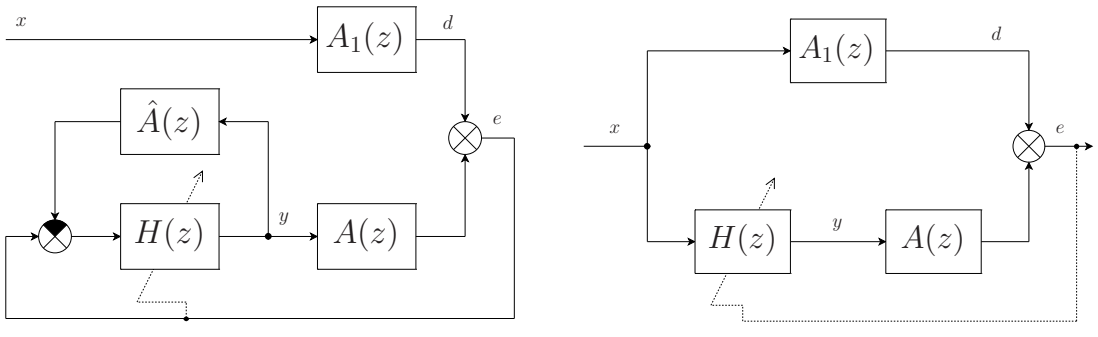
A visszacsatolt struktúrájú rendszernek pusztán a hibajelre támaszkodva kell predikciót végrehajtania, ami kevéssé korrelált minták – pl. szélessávú elnyomandó zaj – esetén nehéz feladat.

A struktúra előnye, hogy akkor is alkalmazható, amikor nem áll rendelkezésre a zajforrásból származó referenciajel.

### 2.5.2. Előrecsatolt (Feedforward) struktúra

Az előrecsatolt struktúra blokkvázlatát mutatja a 2.6. *b)* ábra. Fontos különbség a visszacsatolt struktúrához képest, hogy a rendszer megkapja az $x$ referenciajelet is. Ez még egy érzékelőt igényel, amely lehet egy, a zajforráshoz közel elhelyezett mikrofon. Így azonban a referenciajelre szuperponálódhat a beavatkozójel (parazita átvitel), ami akár a rendszer instabilitásához is vezethet. Ez elkerülhető, ha a referenciajelet nem mikrofonnal, hanem valamilyen egyéb érzékelővel vesszük.

A struktúra egyik nagy előnye, hogy az előrecsatolás miatt a szabályozás hibája elméletileg zérus is lehet.




(a) Visszacsatolt (b) Előrecsatolt

2.6. ábra. A visszacsatolt és az előrecsatolt struktúra blokkvázlata

# 3. fejezet

# A szenzorhálózaton alapuló zajcsökkentő rendszer ismertetése

Az általam tanulmányozott és továbbfejlesztett aktív zajcsökkentő rendszer meglehetősen komplex, több egységből felépülő elrendezés. A rendszer részei a következők:

**Vezeték nélküli szenzorhálózat:** A rendszerben vezetékes mikrofonok helyett vezeték nélküli szenzorhálózat felel az elnyomandó zaj érzékeléséért. Ez gyakorlati szempontból megalapozott, hiszen ebben az esetben nincs szükség hosszú jelvezetékekre. Ezáltal a szenzorok telepítése egyszerű, és elrendezésük dinamikusan változtatható. A mobilitás előnye mellett azonban a szenzorhálózat alkalmazása számos problémát is felvet. A szabályozási körként is tekinthető zajelnyomó struktúra stabilitása ugyanis megköveteli a jó minőségű visszacsatolást. Leggyakoribb nehézségek *a mintavételezés szinkronizálása*, valamint a továbbítandó adatmennyiség csökkentését megcélzó *előfeldolgozás* kapcsán merülnek fel. Ezzel a két kérdéskörrel a rendszer ismertetése során részletesebben is foglalkozunk, mintegy esettanulmány-szerűen bemutatva a jelen rendszerben alkalmazott megoldásokat.

**Jelfeldolgozó processzor és fejlesztőkártya:** A feladat bonyolultsága megkívánja nagyteljesítményű jelfeldolgozó processzor alkalmazását. Ennek megfelelő módon kell illeszkednie a fizikai világhoz, a szenzorhálózaton és a beavatkozó hangszórókon keresztül.

**Személyi számítógép:** A rendszer hatékony működtetéséhez praktikus egy PC alkalmazása, amelynek segítségével elvégezhetjük a bonyolult offline matematikai számításokat, beavatkozhatunk a zajelnyomó algoritmusok működésébe, és egyúttal grafikus felhasználói környezetet biztosíthatunk a rendszert üzemeltető személy számára. Felhasználói szemmel nézve a PC felelős az ember-gép kapcsolat megteremtéséért.

## 3.1. Elosztott adatgyűjtés

A jefeldolgozó rendszereknek szükségük van a fizikai környezetből a feldolgozandó jelek mérés útján történő megszerzésére. Az elző fejezet szerint legtöbb esetben a fizikai környezetet az akusztikus környezet képviseli, amelyben mikrofonokat kell elhelyezni a tér viszonylag távoli ponjaiban. Ezek jeleit hagyományosan vezetéken továbbítjuk. Napjainkra azonban a technológiai fejlődés lehetővé tette olcsó digitális rádiós adattovábbító rendszerek létrehozását, amelyeket különféle szenzorokkal kiegészítve mérésadatgyűjtő és -továbbító hálózatokat, elterjedtebb nevükön *szenzorhálózatokat* alakíthatunk ki.

A korszerű megoldásokban nem vezetékes mikrofonokkal látjuk el az észlelendő akusztikai környezetet, hanem helyette egy digitális adatgyűjtő hálózatot készítünk, amely már bizonyos mértékű jelfeldolgozási feladatokra is alkalmas számítási kapacitást biztosíthat az egyes csomópontokban.

A szenzorhálózatok révén ugyan lehetőség van a kábelezés elkerülésére, de sajnos további megoldandó problámákkal kell számolnunk, úgymint a kapcsolat nem megbízható volta, vagy a csomópontok óraszinkronizációs problémája.

Az alkalmazások terjedésével összhangban egy beágyazott rendszerekkel foglalkozó villamosmérnök számára elengedhetetlen, hogy tisztában legyen az adaptív jelfeldolgozási módszerek és algoritmusok felépítésével és működésével, valamint az általuk kínált lehetőségek mellett azok gyakorlati korlátaival is.

### 3.1.1. A szenzorhálózat felépítéséről

A szenzorhálózat egy digitális, vezérlő logikával ellátott mérésadatgyűjtő és -továbbító rendszer. Az általam használt rendszerben a szenzorhálózatot és a bázisállomást a *Crossbow Technology* által gyártott *MPR2400* (Berkeley MICAz) mote-ok alkotják [20]. A mótok moduláris felépítésűek – a központi panelhez egy csatlakozó segítségével illeszthetünk különféle szenzorkártyákat, vagy programozókártyát[1].

A központi panel lelke egy *ATmega128L* mikrokontroller [21], amely 7,3728 MHz frekvencián üzemel. Tartalmaz ezen kívül egy *CC2420*-as [22] típusú, *ZigBee* szabványnak megfelelő rádiós IC-t, és 512 kB flash memóriát. A rádió a 2,4 GHz-es ISM[2] sávban működik, és a *IEEE* 802.15.4-es szabványnak megfelelően 250 kbps maximális adatátviteli sebességre képes. Ezt a szabványt kifejezetten beágyazott rendszerek rádiós hálózatai számára fejlesztették ki, a telepről való működtethetőség érdekében szem előtt tartva a kis energiafelhasználást. A szórt spektrumú moduláció révén nem kell számolnunk a tereptárgyakról való reflexiók okozta többutas terjedés miatti frekvenciafüggő csillapítás kellemetlen hatásaival sem.

A gyártó többféle típusú és kiépítésű szenzorkártyát kínál. Rendszerünkben *MTS310*-es típusú [23], mikrofont is tartalmazó szenzorkártyák kerültek alkalmazásra.

A mótok felprogramozása soros porton keresztül, PC-ről történik, az *MIB510*-es típusú programozókártya segítségével [24]. A rendszerben a programozókártya szerepe kettős:

[1] A gyártó rövid prezentációban demonstrálja a szenzorhálózatok működését [19].
[2] Industrial, Scientific and Medical Radio Band

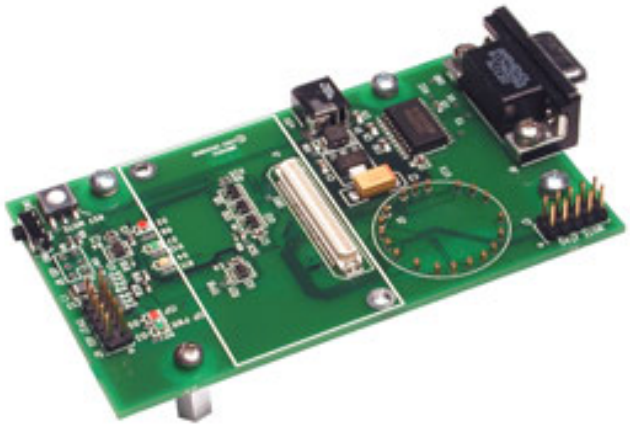
segítségével hozzáférhetünk a rácsatlakoztatott mote mikrokontrollerének USART perifériájához, így *RS-232* szabvány szerinti soros kommunikációra nyílik lehetőségünk. A programozókártya tehát szintillesztést is megvalósít. A szenzorhálózat bázisállomása tulajdonképpen egy ilyen programozókártyával összekapcsolt mote alapkártya, amely a programozókártya *RS-232* interfészén keresztül csatlakozik a jelfeldolgozó kártya soros perifériájához.

A mótokon *TinyOS* beágyazott operációs rendszer fut, amely nyílt forráskódú, és szabadon letölthető [25]. Legfőbb előnye, hogy – a komponensek és interfészek rendszerével, valamint az események kezelésével – lehetővé teszi a perifériák egyszerű elérését. A szenzorhálózat programjának fejlesztése *NesC* programozási nyelven történt, amely a C nyelv kiegészítése kifejezetten szenzorhálózatos rendszerekre – a *TinyOS* struktúrájának és koncepciójának figyelembe vételével [26].

A diplomatervem alapjául szolgáló zajcsökkentő rendszer alapvetően kétféle megvalósításban áll rendelkezésre. Ezek a zajcsökkentő algoritmus tekintetében teljesen megegyeznek; különbség köztük a szenzorhálózat kivitelezésében és a rendszerbe való integrációjában van. A kétféle adatgyűjtési eljárást (egyszerű mintatovábbítást, illetve Fourier-dekompozíciót) megvalósító adatgyűjtő hálózatot következő alfejezetek ismertetik.

(a) MICAz alapkártya

(b) MTS310 szenzorkártya



(c) MIB510 programozókártya

3.1. ábra. A szenzorhálózat elemei

### 3.1.2. Egyszerű adattovábbító hálózat

Az egyszerű adattovábbító hálózat egy olyan adatgyűjtő rendszert valósít meg, amelyben a szenzor mote-ok mintavételezik a saját szenzorkártyájuk mikrofonjának jelét, és ezeket a bázisállomás felé 25 mintát tartalmazó csomagokban továbbítják. A bázisállomás ezeket a mintákat összegyűjti, átcsoportosítja, és a soros portra az egy mintavételi időponthoz tartozó, különböző mote-októl származó adatokat küldi. Viselkedése tehát úgy is felfogható, hogy egy többcsatornás AD átalakító módjára, adott mintavételi frekvenciával szolgáltatja az adatokat. Ha egy mote nem szolgáltat mért adatot, a neki megfelelő bájtokban a bázisállomás nullákat küld. Normál működés során a zérus érték soha nem fordul elő, így ez az érték alkalmas az átviteli hibákból eredő adatvesztés jelölésére – elkerülve ezáltal a téves adatok feldolgozását.

A mótok 1,8 kHz-es mintavételi frekvenciával dolgozzák fel a szenzorkártyájukon található mikrofon jelét, a mikrokontroller beépített A/D alakítójának és időzítőinek segítségével. Ekkora mintavételi frekvencia a rendszer megfelelő működéséhez még elegendő sávszélességet biztosít.

A szenzorhálózat a referencia mote órájához szinkronizálódik. Az egyes mote-ok mintavételi frekvenciája megegyezik, és a bázisállomás is ugyanezzel a mintavételi frekvenciával továbbítja az adatokat.

A zajminták egyszerű továbbítása esetén a hálózatban – a tapasztalatok szerint – maximum 2-3 mote működtethető. Ennek oka, hogy a mótok számának növelésével nő a hálózatban egységnyi idő alatt továbbítandó adatmennyiség, a csatorna sávszélessége viszont korlátozott. Ront a helyzeten, hogy a keretezés és a szoftveres overhead miatt csupán 60% körüli kihasználtság érhető el.

Ezt a korlátot átléphetjük pl. úgy, ha a jelen olyan előfeldolgozást végezünk, amivel csökkenthető a továbbítandó adatmennyiség. A 8 bites, kis számítási teljesítményű mikrokontrollerek nem teszik lehetővé a bonyolult jelfeldolgozást, de egyszerű, kis számításigényű algoritmusok futtatása nem okoz problémát.

### 3.1.3. Rezonátoros adattovábbító hálózat

A rezonátoros adattovábbító hálózat ebben az esetben egy olyan adatgyűjtő rendszert valósít meg, amelyben a szenzor mótok mintavételezik a saját szenzorkártyájuk mikrofonjának jelét, és ezen minták, valamint a bázisállomástól kapott információ alapján kiszámítják az érzékelt – periodikusnak feltételezett – jel Fourier-együtthatóit, amelyeket azután a bázisállomásnak továbbítanak. Ez a felbontás – a 3.3. szakaszban ismertetésre kerülő – rezonátoros megfigyelő alakban kivitelezett DFT szűrővel történik, innen a *rezonátoros* elnevezés.

Mivel a Fourier-együtthatók általában lassabban változnak, mint maga a jel, továbbításuk ritkábban is lehetséges, így kevésbé jelent korlátozást a hálózat adatátviteli sebessége. Az így előállított együtthatókat a jelfeldolgozó kártyán futó algoritmus közvetlenül is fel tudja használni. A bázisállomás a kapott együtthatókat összegyűjti, és a soros porton továbbítja a DSP fejlesztőkártya felé. Ha egy mote valamilyen okból nem szolgáltat adatot, a neki megfelelő bájtokban a bázisállomás zérus értékeket küld. Normál működés során a

zérus érték soha nem fordul elő, így – az egyszerű adattovábbító hálózathoz hasonlóan – ez alkalmas módszer az érvénytelen adatok jelzésére.

A dekompozícióhoz a mótoknak ismerniük kell a Fourier-felbontáshoz szükséges bázisfüggvényeket. Mivel ezeket a DSP-n állítjuk elő (lásd később), szükség van egy DSP→mót-bázisállomás irányú kapcsolat létrehozására is. Maguknak a mótoknak a bázisállomás szolgáltatja a Fourier-dekompozíció alapjául szolgáló alapharmonikus bázisfüggvény fázisát illetve frekvenciáját. Elegendő az alapharmonikus rezonátor fázisát és frekvenciáját ismerni, hiszen a többi ebből származtatható.

A Fourier-dekompozíciót megvalósító szenzorhálózat a tapasztalatok szerint 6 db móttal is megfelelően és megbízhatóan működik, ami nyilvánvaló előnyt jelent az egyszerű adattovábbítást megvalósító hálózattal szemben. Így a rendszer továbbfejlesztése során a rezonátoros adattovábbító hálózatra épülő struktúrát tekintjük kiindulási alapnak.

### 3.1.4. A szinkronizáció megvalósulása

Bár a teljes rendszer működése a 3.3. szakaszban ismertetésre kerül, ott a szenzorhálózat szinkronizációs kérdéseit nem tárgyaljuk, így az jelen fejezetben kerül bemutatásra.

Ugyan a szenzorhálózatban lévő egységek, csomópontok az egymás közti kommunikáció révén kapcsolatban állnak egymással, ez a kapcsolat azonban meglehetősen laza, így az egységek bizonyos szempontból autonóm rendszereknek tekinthetők.

Az idő ismerete számos alkalmazás esetén kulcsfontosságú kérdés, így általában a hálózat minden egysége rendelkezik valamilyen önálló órával. Ezek az órák – bár névlegesen megegyező időalap szerint működnek – természetesen nem járnak teljesen azonosan, hiszen az időalapjukat előállító órajel-generátorok frekvenciái nem egyeznek meg pontosan. Ennek többféle oka is lehet. Például a gyártási bizonytalanságok, vagy az eltérő környezeti viszonyok.

Természetesen az órajel-generátorok pontossága különleges technikák alkalmazásával növelhető, ez azonban az árukat is jelentősen megnöveli. A növekvő pontosság egyébként sem szünteti meg a szinkronizáltság problémáját, legfeljebb a megfelelő pontosság tartásához szükséges szinkronizációk közötti időt növeli meg.

Szenzorhálózatunk időosztásos (TDM) elven működik. A bázisállomás szinkronüzeneteit (beacon) követően minden mót kap egy számára kijelölt időszeletet, ami alatt elküldheti az általa előállított $X_i$ együtthatókat. Ezeket a bázisállomás fogadja, és továbbítja a DSP felé. A szinkronüzeneteket a bázisállomás periodikusan küldi olyan gyakorisággal, hogy minden mótnak legyen elegendő ideje az adatok továbbítására. A beacon-üzenet egyrészt a *mintavételi időpontok* szinkronizációját, másrészt a Fourier-dekompozíció $c_i(n)$ *bázisfüggvényeinek* frissítését szolgálja.

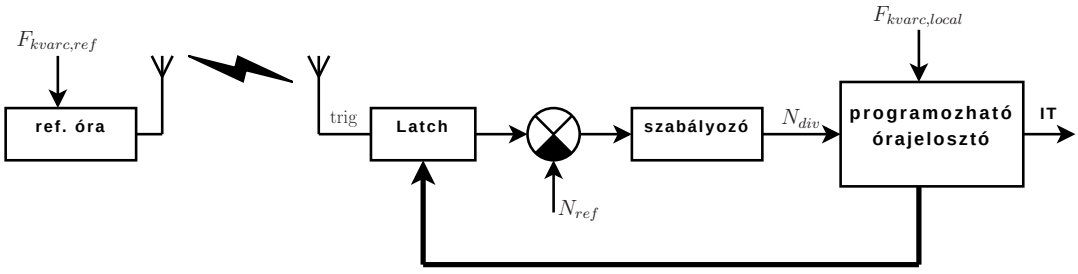
A *mintavétel* szinkronizálása azért szükséges, mert – habár a névleges mintavételi frekvenciák megegyeznek – az egyes mótok időzítőit működtető kvarcoszcillátorok órajel-frekvenciái eltérnek a névleges értéktől, így a mintavételi időpontok elcsúsznak a névleges adatokból adódókhoz képest. Ez az elcsúszás a másodlagos út átviteli függvény-mátrixát megváltoztatja, hiszen annak kialakításában az adatok keletkezése között eltelt idő is részt vesz.

A szinkronizációt valamelyest egyszerűsíti, hogy nem kell *abszolút szinkronizáció*t megvalósítani – tehát nem szükséges egy egységes referenciaidőt ismerni –, elegendő *a mótok együttfutásának* biztosítása. Sok szabályzási rendszerben követelmény, hogy a beavatkozó jel számítása időben összetartozó minták alapján történjen. Esetünkben azonban elegendő, ha a mintavételi időpontok mindig fix időkülönbséggel követik egymást. Ez arra vezethető vissza, hogy zajcsökkentő rendszerünkben az egyes csatornák identifikációja külön-külön történik, így a késleltetés miatti – állandó – fázistolás benne foglaltatik az identifikáció során meghatározott **W** mátrixban. Az sem jelent problémát, ha a szinkronizáció nem történhet meg minden egyes mintavételi időpontban, mert a frekvencia ismeretében a fázis a szinkronizációt követő időpontokban is előállítható.

Feltételezzük, hogy az adattovábbító hálózatban lezajló folyamatok (mintavételezés, előfeldolgozás, adattovábbítás, stb.) időigénye jó közelítéssel állandónak tekinthető. Ez a feltételezés azzal magyarázható, hogy a mótokon futó programok periodikus működésűek, a csomagok küldése a hálózatban pedig determinisztikus. Ki kell jelölni a szenzorhálózatban egy referenciaként szolgáló egységet, amelyhez a többi csomópont szinkronizálódni fog. Ez a kijelölés gyakorlati megfontolások alapján történhet. Esetünkben – praktikusan – a bázisállomás tölti be a referencia-csomópont szerepét. A hálózat többi tagja a referenciaállomás csomagjainak érkezését használja fel szinkronizációs célra – ahogy azt fentebb már említettük. Bár a referenciacsomag nem tartalmaz időinformációt, az implicit módon mégis benne foglaltatik azáltal, hogy a csomag elküldése meghatározott időpontokban történik.

A mótokon a mintavételi időalapot egy *hangolható időzítő* állítja elő. Az időzítőben egy számláló értéke növekszik az órajel ütemében, amíg el nem éri a mintavételi frekvenciát meghatározó (szabadon változtatható) $N_{div}$ értéket. Ennek bekövetkeztekor a számláló nullázódik, és a folyamat kezdődik elölről. A hangolható időzítő tehát tulajdonképpen egy *programozható órajelosztó*. A számláló időfüggvénye egy (digitális) fűrészjel, amelynek lefutó élénél *interrupt* generálódik. Ennek hatására történik meg a mikrofon jelének mintavételezése. Látható, hogy ha az egyes mótok órajelosztóinak fűrészjelét valamilyen módon fáziszárt állapotba hozzuk, akkor biztosított a szinkron mintavételezés. Rendszerünkben ezt egy PLL-szerű algoritmus segítségével próbáljuk elérni. Arra törekszünk, hogy a referenciaállomás szinkronizációs csomagja mindig az órajelosztó fűrészjelének egy bizonyos fázishelyzetében érkezzen. A fázisdetektort úgy valósítjuk meg, hogy a referencia-csomópont jelzéseikor – tehát a szinkronizációs csomagok megérkezésekor – a hangolható időzítő számlálójának értékét mintavételezzük. Amennyiben ez a szinkronizációs csomag a megfelelő időpontban érkezik (fáziszárt állapot), az adott csomópont változatlan mintavételi frekvenciával[3] működik tovább. Abban az esetben, amikor a referencia-csomag később érkezik az előírtnál (sietés), a $T_s$ mintavételi időköz megfelelő mértékű növelése révén *csökkentjük a mintavételi frekvenciát*. Amennyiben viszont a szinkronizációs csomag korábban érkezik az előírtnál (késés), $T_s$ értékének csökkentésével *megnöveljük a mintavételi frekvenciát*. Ezáltal tulajdonképpen az egyes mótok mintavételi frekvenciáinak PLL-szerű „összehúzását” való-

[3]Mintavételi frekvencia alatt itt nem egy abszolút frekvenciát kell érteni, hanem az adott mote saját órajelének egy bizonyos értékkel való leosztottját. A cél az, hogy minden móton megtaláljuk azt az $N_{div}$ osztási arányt, amellyel a helyi órajelet leosztva a kívánt (abszolút) mintavételi frekvenciát kapjuk. A leosztásnak természetesen követnie kell az alapórajel frekvenciadriftjét is.




3.2. ábra. A szinkronizáció mechanizmusának szemléltetése

sítjuk meg. Egy ilyen PLL-szerűen működő szinkronizációs mechanizmus blokkdiagramját szemlélteti a 3.2. ábra.

### 3.1.5. A TinyOS programozási funkciói

A TinyOS fejlesztői környezet beépített funkciói révén igyekszik a programozási feladatot minél jobban megkönnyíteni. Egyrészt közvetlenül támogat számos programozóeszközt (a számunkra fontos MIB510-et is), másrészt lehetővé teszi az egyes eszközök egyedi címmel való ellátását a programkód újrafordításának szükségessége nélkül. Az MIB-510 A Crossbow[4] soros porti programozóeszköze.

A TinyOS alapvető programozó szoftvere a „$\mu$ In-System Programmer", vagy `uisp`. Ennek bonyolult paraméterezését a felhasználónak azonban nem kell ismernie és használnia, mert azt a rendszer az alkalmazott eszköznek megfelelően önállóan meghívja.

### 3.1.6. A mótok felprogramozása

A mote-ok programozása az MIB510 programozókártya segítségével történik, melyet a számítógép soros portjához kell csatlakoztatni. Windows alatt a programozás a Cygwin-ben kiadott parancsok segítségével történik. A Cygwin indítása után be kell lépni a programok forráskódjait tartalmazó könyvtárba:
`cd /opt/tinyos-1.x/contrib/xbow/apps/`, majd a konkrét programot tartalmazó könyvtárba (a szenzorhálózat mote-jai esetében `cd ANCsensor` vagy `cd ANCsensorREZ` , a bázisállomás esetében `cd ANCbase` vagy `cd ANCbaseREZ`).

A programozáshoz minden egyes mote-ot egyedi azonosítóval, a hálózati címmel kell ellátni, amely a következőképpen számítható ki:

$$ID = 8 \cdot (OSSZES\_MOTE\_SZAMA - 1) + LOCAL\_ID,$$

ahol:

- $OSSZES\_MOTE\_SZAMA$: a hálózatban található mote-ok darabszáma, a bázisállomás beleszámítása nélkül
- $LOCAL\_ID$: egyedi azonosító, amely nullától indul, egyesével haladva felfelé.

[4] `http://www.xbow.com/`

A program fordítása és a programozókártyára helyezett mote felprogramozása a következő paranccsal indítható, az ID és a soros port megfelelő módosításával:

```
make micaz install,ID mib510,com1
```

A programozás menete a következő:

1. A programozókártya feszültség alá helyezése, majd a számítógép soros portjához való csatlakoztatása
2. A szenzorkártya eltávolítása a mote-ról
3. A telepek eltávolítása a mote-ból
4. A mote felhelyezése a programozókártyára
5. A fent ismertetett parancs kiadása a számítógépen
6. A mote eltávolítása a programozókártyáról.

A bázisállomás felprogramozása a következő paranccsal indítható:

```
make micaz install,OSSZES_MOTE_SZAMA mib510,com1
```

Példaként, egy három szenzort tartalmazó hálózat mote-jainak programozásához a következő parancsokat kell kiadni a megfelelő könyvtárban:

```
make micaz install,16 mib510,com1
make micaz install,17 mib510,com1
make micaz install,18 mib510,com1
```

A referencia-mote mindig a legkisebb azonosítóval rendelkező, a példában a 16-os hálózati azonosítóval (ID-vel) rendelkező darab.

A programozás folyamata a `Ctrl+C` billentyűkombinációval szükség, vagy rendellenesség esetén bármikor megszakítható. Ha a programozás nem működik, a jelenség oka az lehet, hogy a programozókártyára helyezett mote a soros porton kommunikál annak ellenére, hogy éppen programozási folyamat indult. Ha az MIB510 programozókártyán az `SW2` kapcsolót ON állásba kapcsoljuk, a ráhelyezett mote nem tud a soros porton üzeneteket küldeni.

Lefordított program felprogramozásakor a `reinstall` paramétert kell használni a parancsok kiadásakor. Programozás néküli fordításhoz a

```
make micaz
```

parancsot kell kiadni.

## 3.2. Az ADSP-21364 EZ-KIT Lite fejlesztői kártya [12]

A rendszer jelfeldolgozási feladatinak döntő része az *Analog Devices ADSP-21364 EZ-KIT Lite* néven ismert fejlesztőrendszeren lett implementálva. A kártya szíve egy *ADSP-21364*[13] típusú processzor, amely köré a gyártó olyan perifériákat és hardverelemeket épített, amelyekre az általános jelfeldolgozási feladatok során szükség lehet. A kapcsolósorok segítségével gyorsan és egyszerűen módosíthatjuk a konfigurációt (pl. kiválaszthatjuk, hogy egy buszon melyik perifériát szeretnénk a munka során használni, stb.). Az általános felhasználású nyomógombok és LED-ek a fejlesztés során történő hibakeresési folyamatokat könnyíthetik meg.

A fejlesztőkártya tervezésekor a gyártó arra törekedett, hogy a processzor képességeit és szolgáltatásait a lehető legnagyobb mértékben kipróbálhatóvá is megismerhetővé tegye. Külön kiemelendő a $8 + 2$ db analóg kimenet, ami által a kártya sokcsatornás rendszerek tervezésére és kialakítására is alkalmas.

Azáltal, hogy mindezen komponenseket egyetlen, jól átgondolt és megtervezett eszköz formájában megkapjuk, a szoftverfejlesztés szinte azonnal megkezdhető, anélkül, hogy hardvertervezési lépésekkel bajlódnunk kellene.

A következőkben sorra vesszük azokat a hardverelemeket, amelyekkel a gyártó igyekszik megkönnyíteni és hatékonyabbá tenni a fejlesztést:

- Analog Devices ADSP-21364 jelfeldolgozó processzor
- 512 kB SRAM[5] (8 bites szóhossz)
- 1 MB flash memória (8 bites szóhossz)
- 2 Mbit SPI buszon elérhető flash memória
- Analóg hanginterfész
  - AD1835A kodek; 96 kHz, 24 bit
  - 2 db sztereo bemeneti csatorna (RCA)
  - 4 db sztereo kimeneti csatorna (RCA)
  - 1 db sztereo fejhallgatókimenet (tetszőleges sztereo kimenetre kapcsolható)
- Digitális hanginterfész (1 db RCA bemenet, 1 db RCA kimenet)
- 11 db LED
  - 8 db általános célú
  - 1 db Power, 1 db Board reset, 1 db USB monitor
- 5 db nyomógomb (1 db Reset, 2 db DAI, 2 db FLAG lábakon)
- Általános bővítőinterfész
- JTAG In-Circuit Emulator csatlakozó
- USB porton keresztüli programletöltés és debugger interfész

[5] Static RAM

### 3.2.1. Az ADSP-21364 jelfeldolgozó processzor [13]

#### Architekturális felépítés

A 3.2. alfejezetben bemutatott fejlesztői kártya központi egysége egy ADSP-21364 típusú jelfeldolgozó processzor. Ez a DSP az Analog Devices egyik legszélesebb körben alkalmazott termékcsaládjának, a SHARC[6], családnak a tagja. Ez a felépítés a beágyazott rendszerekben alkalmazott processzorok körében elterjedt Harvard-architektúra továbbfejlesztett változata. A von Neumann architektúrával szemben itt külön van választva a program- és adatmemória, amelyek a külön buszrendszernek köszönhetően szimultán elérhetők. A szuper-Harvard processzorok négy önálló buszrendszerrel rendelkeznek – egy utasításbusszal, egy I/O busszal és két adatbusszal.

A SHARC elnevezés egy nagy számítási teljesítményű, 32 bites lebegőpontos aritmetikával ellátott processzorcsaládot képvisel. A gyártó olyan alkalmazásokhoz szánja ezeket a termékeket, ahol a teljesítmény és a nagy dinamikatartomány kulcsfontosságú. Jelenleg három generációt számlál ez a család, amelyek egymással kód-kompatibilisek. Találunk köztük olyan – belépő szintű – processzort, amelynek ára alacsonyabb $10-nál, de a 400 MHz-es, 2400 MFLOP számítási teljesíményű típusok is megtalálhatók a palettán.

#### Főbb jellemzők

- 333 MHz max. órajel-frekvencia, 2000 MFLOPS[7] (a fejlesztőkártyán $F_{clk} = 300$ MHz)
- 32 és 40 bites lebegőpontos számábrázolás
- SIMD[8] architektúra
- 3 Mbit on-chip SRAM: 1-1 Mbit-es a 0-ás és 1-es blokk; 0,5-0,5 Mbit-es a 2-ás és 3-as blokk, amelyek szimultán elérhetők a processzor és a DMA vezérlő számára
- 4 Mbit on-chip ROM (2 Mbit a 0-ás, 2 Mbit az 1-es blokkban)
- A processzormag sebességén (333 MHz) működő memóriabuszok
- Kettős címaritmetikai egység modulo- és bitreverse-címzéssel
- Hardver-támogatás 16 db cirkuláris pufferhez
- Hatszintű hardveres támogatású ciklusszervezés (zero overhead looping)
- 25 csatornás DMA vezérlő
- Digitális hanginterfész
- S/PDIF szabványú interfész
- Aszinkron párhuzamos port

[6] Super Harvard Architecture Single-Chip Computer
[7] Million Floating-point Operations Per Second
[8] Single Instruction, Multiple Data

- 2 db SPI interfész
- 3 db programozható időzítő; PWM és külső esemény számláló módokkal
- Két önálló műveletvégző egység (ALU, hardveres szorzó, léptető, és külön regiszterfájl)
- Kódkompatibilitás a SHARC család többi tagjával (az első generációs ADSP-2106x SHARC processzorokkal csak SISD[9] módban)

Minden utasítás egyetlen órajel-ciklus alatt végrehajtható, így maximális órajel-frekvencia mellett 3 ns ciklusidő is elérhető. A gyártó által megadott benchmark-adatokat foglalja össze a 3.1. táblázat. Az értékek maximális órajel-frekvencia mellett értendők.

| Feladat | szükséges idő |
|---|---|
| 1024 pontos komplex FFT (radix 4) | 27,9 $\mu$s |
| FIR szűrés (tap-enként) | 1,5 ns |
| IIR szűrés (másodfokú alaptagonként) | 6,0 ns |
| Mátrix-szorzás $[3 \times 3][3 \times 1]$ | 13,5 ns |
| Mátrix-szorzás $[4 \times 4][4 \times 1]$ | 23,9 ns |
| Osztás | 10,5 ns |
| Inverz gyökvonás | 16,3 ns |

3.1. táblázat. ADSP-21364 benchmark-adatok (333 MHz)

[9]Single Instruction – Single Data

### 3.2.2. A VisualDSP++ integrált fejlesztői környezet

Az ADSP-21364 EZ-KIT Lite fejlesztői kártyára történő szoftverfejlesztés a *VisualDSP++ IDDE*[10] programcsomag segítségével történik, amely az *Analog Devices* vállalat terméke. Tulajdonképpen egy személyi számítógépen (Windows környezetben) futó integrált fejlesztői környezetről van szó, amely – az ADSP-21364 SHARC processzor mellett – számos, az *Analog Devices* által gyártott jelfeldolgozó processzort támogat. Képes együttműködni a cég által gyártott fejlesztői kártyákkal, ezzel is könnyítve a fejlesztői munkát. Az alkalmazás két részre tagolódik: *Projekt szerkesztő*re és *Debugger*re.

A Projekt szerkesztő szolgál a program forrásfájljainak szerkesztésére, lefordítására és linkelésére. A projektet egymáshoz rendelt forrás- és leírófájlok alkotják, amelyekből felépül egy konkrét jelfeldolgozó program. A forrásfájlokat Assembly és C nyelven írhatjuk meg. A fordítás során keletkező tárgykódfájlokból a linker állítja elő az eszközre letölthető *futtatható állomány*t (`.exe`), illetve ennek *debuggolható verzió*ját (`.dxe`), vagy az *EEPROM-ba írható* `.bit` fájlt. A projekt részét képezi egy linkelést leíró LDF (*Linker Description File*) fájl, amely a linker számára tartalmaz fontos információkat. Ez a fájl írja le, hogy a programban szereplő kód- és adatrészek melyik memóriaszegmensbe töltődjenek be. A hatékony futtatás szempontjából ez alapvető jelentőséggel bír, hiszen a nem megfelelően megválasztott memóriatartományok – a párhuzamosítási lehetőségek kihasználatlansága miatt – a végrehajtási idő megnövekedését eredményezhetik.

A Debugger felelős a futtatható DSP programnak a processzor memóriájába való letöltéséért. Emellett hatékony hibakeresési funkciókat is biztosít. Segítségével lehetőségünk van töréspontok elhelyezésére, a program futásának tetszőleges helyen történő felfüggesztésére, elindítására, lépésenkénti végrehajtásra. Megállított program esetén megtekinthetjük és módosíthatjuk a regiszterek és a memória tartalmát, lehetőségünk van memóriatartományok fájlba mentésére vagy fájlból való feltöltésére. A szimbólumkezelőnek köszönhetően az egyes memóriaterületekre a forráskódban deklarált nevükkel is hivatkozhatunk. Nagyban segíti a fejlesztői munkát az a funkció is, hogy memóriaterületek tartalmát (pl. egy algoritmus kimeneti mintáit tároló vektor tartalmát) grafikusan megjeleníthetjük.

A VisualDSP++ beépített szimulátorával a PC-n szimulálhatjuk az eszköz majdani várható működését, így – bizonyos korlátokon belül – a hardver rendelkezésre állása nélkül is lehetőségünk van folytatni a fejlesztői munkát.

[10] Integrated Development and Debugging Environment

## 3.3. A jelfeldolgozó algoritmus ismertetése [4] [5]

A diplomaterv alapjául szolgáló zajcsökkentő rendszer periodikus akusztikus zajok elnyomására alkalmas, ún. *jelmodell alapú* algoritmuson alapszik. A következő alfejezetekben ezen algoritmusok részleteire térünk ki, a rendszer működésének megértéséhez szükséges mélységben. A téma iránt mélyebben érdeklődő Olvasó számára [4] és [] irodalmak szolgálhatnak további ismeretekkel.

### 3.3.1. Zajcsökkentés periodikus jelekre

Környezetünkben a zajok, zavarhatások jelentős része – köznyelven fogalmazva – zúgó, búgó hang. Gondoljunk pl. egy autó motorjának a hangjára, vagy a háztartási gépek által keltett zajokra. Azok a zajhatások tehát, amelyeket aktív zajelnyomással szeretnénk csökkenteni, túlnyomórészt periodikus komponensekből állnak. Természetesen a szélessávú zajcsökkentő eljárások is alkalmasak ilyen zavarok elnyomására, ám a periodikus jelek leírására szolgáló apparátus, valamint az ilyen rendszerek vizsgálati módszerei merőben mások, mint szélessávú jelek esetén. Ez hívja életre azt az törekvést, hogy az ilyen zavarok elnyomására szolgáló rendszerek struktúrája is különböző legyen.

A periodikus jelek leírására szolgáló *Fourier-reprezentáció* azt fejezi ki, hogy egy sávkorlátozott periodikus jel komplex exponenciálisok súlyozott összegeként is leírható:

$$p(n) = \sum P_k \cdot c_k(n)\,, \text{ ahol} \qquad (3.1)$$
$$c_k(n) = \mathrm{e}^{j\,2\pi f_1 k n}\,, \quad k = -K \ldots K$$

A fenti kifejezésekben $f_1$ a periodikus jel alapharmonikusának frekvenciája; a sávkorlátozottságot pedig $k$ véges sok értéke fejezi ki. $K$ értékére – a sávkorlátozottság miatt – a következő megkötést kell tennünk:

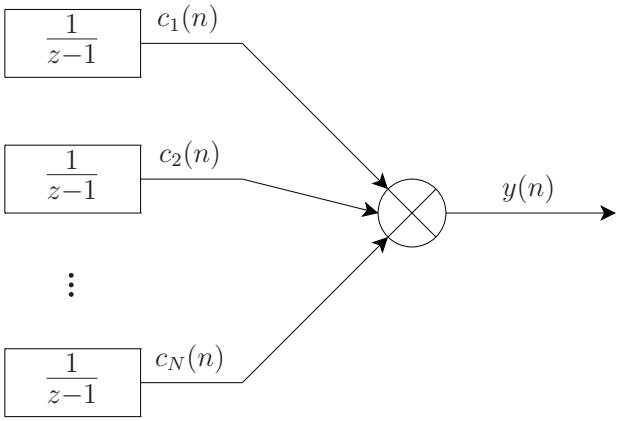
$$K \cdot f_1 \;<\; \frac{f_s}{2} \;<\; (K+1) \cdot f_1 \qquad (3.2)$$

A (3.1) sorfejtés alapján megalkotható a periodikus jelek egy ún. *koncepcionális modellje*[11]. Az $N$-re periodikus jelek egy olyan modelljét keressük, ahol az állapotváltozók az egyes Fouriér-komponensek. Ilyen modellt illusztrál a 3.3. ábra. Ebből kiindulva periodikus jelek generálására alkalmas struktúrát tervezhetünk.

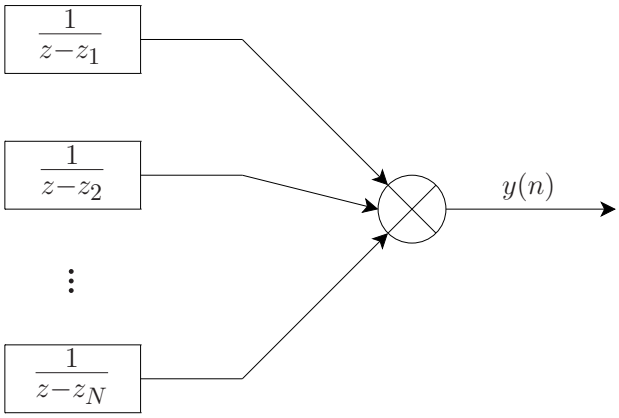
A 3.3. ábrán bemutatott modellben a sávkorlátozott periodikus jel úgy jön létre, hogy az integrátorokba a (3.1) egyenlet szerinti $P_k$ kezdőértékeket töltjük, majd bemenetükre a továbbiakban zérus értéket adunk. Így az integrátorok tartani fogják a kezdőértéket, aminek következtében a modell pontosan a (3.1) által leírtakat valósítja meg.

Sávkorlátozott jelek generálása egy másik, az előzővel ekvivalens modell alapján is történhet. Ezt – az ún. rezonátoros modellt – mutatja be a 3.4. ábra.

[11] A koncepcionális modellről a 3.3.2. alfejezetben lesz szó.

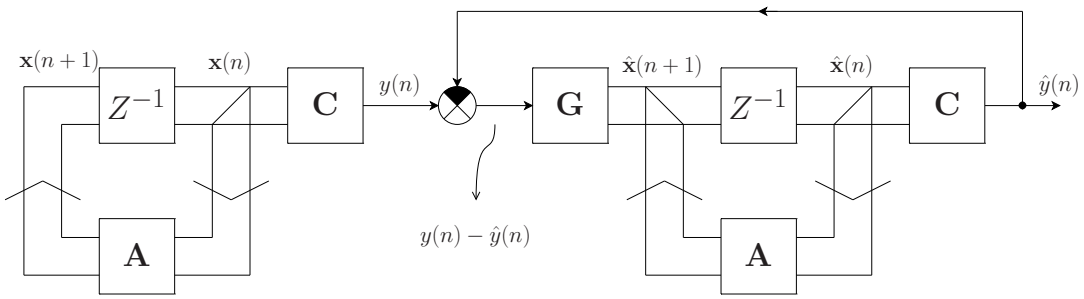



3.3. ábra. Sávkorlátozott periodikus jel koncepcionális modellje integrátorokkal




3.4. ábra. Sávkorlátozott periodikus jel koncepcionális modellje rezonátorokkal


3.5. ábra. Az integrátoros és a rezonátoros jelmodell ekvivalenciája




3.6. ábra. A mefigyelő és a megfigyelt rendszer

A fenti ábrán zi a következőképpen határozható meg:

$$z_i = \frac{c_i(n+1)}{c_i(n)}\,; \quad i = 1 \ldots N \tag{3.3}$$

A 3.5. ábra alapján könnyen belátható, hogy a két modell egymással ekvivalens.

A következő alfejezetekben felidézzük a megfigyelőelmélet alapjait, majd kitérünk arra is, hogy az ismertetett jelmodellekhez hogyan tervezhető megfigyelő. Ezután megmutatjuk, miként épül be a koncepcionális jelmodell a zajelnyomó eljárásba.

### 3.3.2. Jelmodell alapú megfigyelés

A megfigyelő olyan rendszer, amely alkalmas egy másik rendszer állapotváltozóinak (vagy ezekből származtatott mennyiségeknek) a meghatározására. Ezáltal végső soron olyan mérési eljárást valósít meg, amelynek során a megfigyelt rendszer *méréstechnikailag nem hozzáférhető* állapotváltozóinak *lemásolásával* teszi lehetővé azok „megmérését”.

A megfigyelő, amely képes követni a megfigyelt rendszer állapotát, az autonóm rendszer A megfigyelt rendszer állapotát követni képes megfigyelőt úgy állítjuk elő, hogy megvalósítjuk *az autonóm rendszer egy másolatát*, kiegészítve egy bemenettel. A bemenetet *a két rendszer kimenetének eltéréséből*, vagyis a *hibából* képezzük, valamilyen „becsatoláson” keresztül. A rendszerkópia állapotváltozói lesznek a megfigyelt rendszer állapotának becslői.

A másolást tehát a *predikciós-korrekciós elv* segítségével, a két rendszer állapotváltozóinak eltéréséből származtatott hiba becsatolásával hajtjuk végre. A hiba becsatolásának miként nem egyértelmű. A predikciós-korrekciós stratégiákat és a megfigyelőelmélet alapjait [3] és [2] irodalmak részletesebben tárgyalják.

A megfigyelőket jelfeldolgozási feladatra úgy lehet felhasználni, hogy megalkotjuk a feldolgozandó jelet létrehozó rendszer *koncepcionális modelljét*, mégpedig úgy, hogy a meghatározandó paraméterek kapcsolatban legyenek a koncepcionális modell állapotváltozóival. Ezután létrehozunk egy olyan megfigyelőt, amely ezeknek az állapotváltozóknak a meghatározására alkalmas.

A megfigyelő és a megfigyelt rendszer kapcsolatát a 3.6. ábra mutatja (a különbségképzőtől balra az autonóm rendszer, jobbra pedig a megfigyelő). A megfigyelt rendszer meghatározott struktúrájú és adott paraméterű, nem ismerjük azonban állapotát ($\mathbf{x}$). Az

**A** állapotátmeneti mátrix és a **C** kicsatoló mátrix ismeretében az állapotváltozós leírás:

$$\mathbf{x}(n+1) = \mathbf{A}\,\mathbf{x}(n) \tag{3.4}$$

$$\mathbf{y}(n) = \mathbf{C}\,\mathbf{x}(n) \tag{3.5}$$

A rendszerkópia állapotváltozós leírására pedig a következő adódik:

$$\hat{\mathbf{x}}(n+1) = \mathbf{A}\,\hat{\mathbf{x}}(n) + \mathbf{G}\,\{\mathbf{y}(n) - \hat{\mathbf{y}}(n)\} \tag{3.6}$$

$$\hat{\mathbf{y}}(n) = \mathbf{C}\,\hat{\mathbf{x}}(n) \tag{3.7}$$

(3.5) és (3.7) visszahelyettesítésével kiküszöbölhetők a kimeneti változók. Algebrai átalakítások után látható, hogy a megfigyelő állapotátmenet-mátrixa $(\mathbf{A} - \mathbf{G}\,\mathbf{C})$:

$$\hat{\mathbf{x}}(n+1) = (\mathbf{A} - \mathbf{GC})\hat{\mathbf{x}}(n) + \mathbf{G}\mathbf{y}(n) = (\mathbf{A} - \mathbf{G}\,\mathbf{C})\,\hat{\mathbf{x}}(n) + \mathbf{GC}\mathbf{x}(n) \tag{3.8}$$

Összességében a rendszer autonóm, és a követési hiba eltűnésével páronként megegyező állapotváltozókat kell kapnunk. A követési hibára (az állapotváltozók különbségeire) differencia-egyenlet írható fel; ezt a szakirodalom *hibarendszer*nek nevezi:

$$\mathbf{x}(n+1) - \hat{\mathbf{x}}(n+1) = (\mathbf{A} - \mathbf{GC})(\mathbf{x}(n) - \hat{\mathbf{x}}(n) = \tag{3.9}$$

$$= (\mathbf{A} - \mathbf{GC})^{n+1}(\mathbf{x}(0) - \hat{\mathbf{x}}(0)) \tag{3.10}$$

Igazolható, hogy $(\mathbf{A} - \mathbf{GC})^N = 0$ teljesülése esetén rendszerünk véges impulzusválaszú (FIR), ami számunkra kedvező, hiszen ez azt jelenti, hogy véges lépésben konvergál. Ez könnyen belátható, hiszen az előbbi feltétel teljesülésekor $(\mathbf{A} - \mathbf{GC})$ *nilpotens* mátrix, vagyis olyan – nullmátrixtól különböző – mátrix, amely legfeljebb $N-1$ önmagával történő szorzása után nullmátrixszá válik. A nilpotens mátrixokra igaz, hogy valamennyi sajátértékük zérus. A rendszer pólusai tehát mind az origóban vannak, azaz az átviteli függvénye felírható $z^{-1}$ polinomjaként [3].

A fentiek ismeretében nincs más hátra, mint a periodikus jelek koncepcionális modelljét használva parametrikusan felírni a megfigyelő állapotátmeneti mátrixát – az $(\mathbf{A} - \mathbf{GC})$ mátrixot –, majd annak nilpotens tulajdonságát felhasználva **G** becsatoló mátrix elemeire vonatkozólag megoldjuk az egyenletrendszert.

A 3.4. ábra alapján a koncepcionális modellünket leíró mátrixok a következők:

$$\mathbf{A} = \mathrm{diag}\,\langle z_i \rangle \tag{3.11}$$

$$\mathbf{C} = \mathbf{c}^{\mathbf{T}} = [1, 1, 1, \ldots, 1] \tag{3.12}$$

Véges beállású megfigyelőt tervezve, és az egységkörön egyenletesen elhelyezkedő rezonátorpólusok mellett – a részletszámításokat mellőzve – $\mathbf{G} = \mathbf{g}$ elemeire a következők

adódnak:

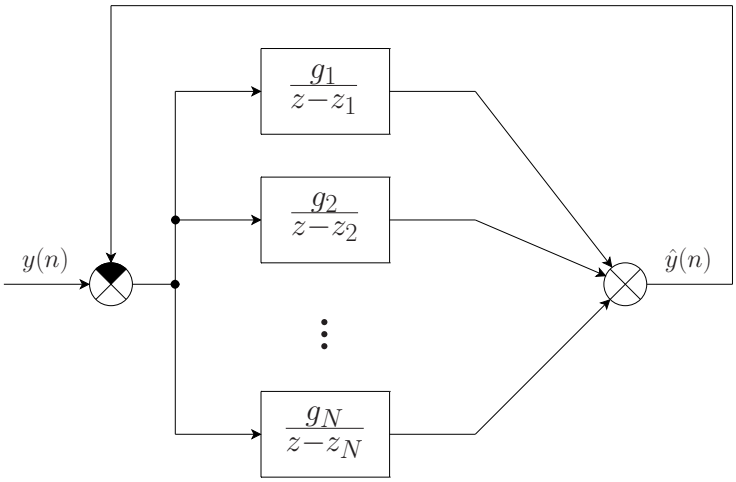
$$g_i = \frac{1}{N} \cdot z_i, \tag{3.13}$$

ahol $N$ a rezonátorok száma, $z_i$ pedig az $i$-edik rezonátorhoz tartozó pólus. Az így adódó elrendezést mutatja a 3.7. ábra.

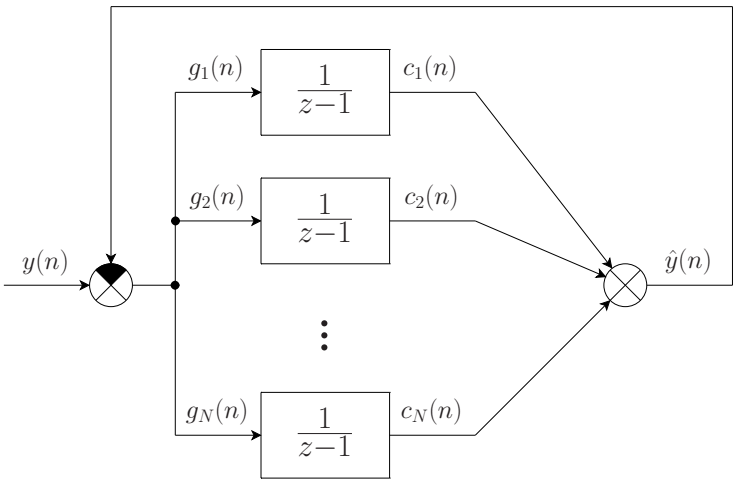
Természetesen a 3.3. ábrán látható integrátoros jelmodellhez is tervezhető megfigyelő. Ez az elrendezés látható a 3.8. ábrán. Az előző esetben deklarált feltételek mellett – ismét a részletszámítások mellőzésével – most a következő adódik $\mathbf{g}$ elemeire:

$$g_i(n) = \frac{1}{N} \cdot \overline{c}_i(n), \tag{3.14}$$

ahol $n$ a diszkrét időt jelöli. Ebben az esetben tehát $\mathbf{g}$ elemei időben változó szorzótényezők lesznek. Ez a struktúra tulajdonképpen egy *rekurzív diszkrét Fourier-analizátor*, amelynek állapotváltozói a transzformált értékeit adják.




3.7. ábra. Rezonátoros jelmodellre épülő megfigyelő




3.8. ábra. Integrátoros jelmodellre épülő megfigyelő

A struktúra működése röviden a következő: a hibajel $k \cdot f_1$ frekvenciájú komponensét a $k$-adik rezonátorcsatorna bemenetén jelen lévő $\overline{c}_k(n)$ szorzótényező zérus frekvenciára keveri le. Lévén integrátor DC átvitele végtelen, az adott rezonátorcsatorna állapotváltozója addig növekszik, amíg a hibajelben jelen van a $k \cdot f_1$ frekvenciájú összetevő. A $k$-adik rezonátorcsatorna által modellezett jelkomponenst a $c_k(n)$-nel való szorzás révén kapjuk meg.

A rezonátoros struktúra a rezonátorpólusok frekvenciáin így elvileg zérus hibával képes előállítani a jelet. Más rezonátorfrekvenciákon az átvitel zérus; a többi frekvencián a hiba valamilyen véges érték.

Mindkét bemutatott struktúra a gyakorlatban is sikeresen alkalmazható. A tárgyalt zajelnyomó rendszer – implementációs szempontok miatt – az integrátoros jelmodell alapú megfigyelőre épül.

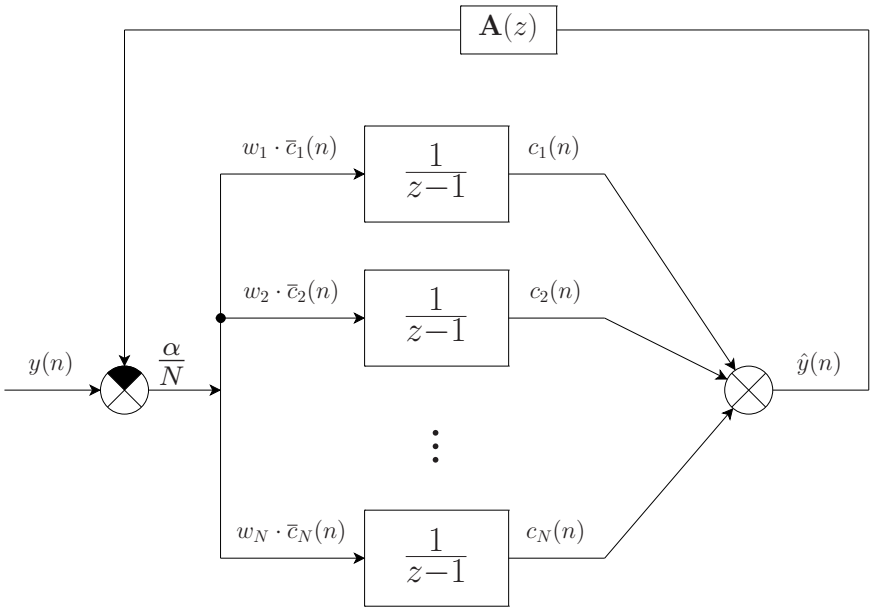
A struktúrát a 2.2. ábrával összevetve látható, hogy a rezonátoros struktúrát olyan zajelnyomó rendszernek tekinthetjük, amelynél $A(z) = 1$. A valóságban azonban ez nem teljesül, hiszen a rezonátorok visszacsatolása egy külső hurkon, a *másodlagos úton* keresztül valósul meg. Ennek kompenzálásaként a $\mathbf{G}$ becsatoló-mátrixban felhasználjuk $\mathbf{A}(z)$ inverzét. Így – a késleltetéstől eltekintve – teljesíteni tudjuk a visszacsatoló ág egységnyi átvitelét. Ezáltal $\mathbf{A}(z)$ tulajdonképpen beépül a modellbe.

Amennyiben $\mathbf{A}(z)$ nem pontosan ismert, a stabilitás biztosításához szükség van egy $\alpha \leq 1$ konvergencia-paraméterre is. Ennek megválasztása hatással van a rendszer tranziens viselkedésére és külső zavarok iránti érzékenységére is.

Egycsatornás esetben $\mathbf{G} = \mathbf{g}$ így a következőképpen alakul:

$$\mathbf{g} = \frac{\alpha}{N} \left[ w_1 \cdot \overline{c}_1(n),\ w_2 \cdot \overline{c}_2(n),\ \ldots,\ w_N \cdot \overline{c}_N(n) \right] \tag{3.15}$$

ahol $w_i$ az $\mathbf{A}(z)$ inverzét jelöli az adott rezonátor-frekvencián. A leírtak figyelembe vételével adódó zajelnyomó struktúra egy csatornáját szemlélteti a 3.9. ábra.




3.9. ábra. A zajelnyomó rendszer egy csatornája

### 3.3.3. Adaptív Fourier-analízis

Az előző alfejezetben ismertetett rezonátoros struktúra a rezonátorfrekvenciákon pontosan előállítja a koncepcionális modellel leírt jelet. Gyakorlati alkalmazások esetén azonban a bemenőjel frekvenciája változhat, eltérhet a feltételezettől. Jó példa erre egy villamos forgógép, vagy egy robbanómotor főtengely-fordulatszámának változása. Ha a zajelnyomást hasonló esetekben is biztosítani kívánjuk, biztosítanunk kell a bemenőjel hibamentes felbontását: a rezonátorok pólusait az elnyomandó jel komponenseinek frekvenciáira kell hangolnunk. Erre alkalmas az *Adaptív Fourier-analizátor*, vagy röviden *AFA*.

Segítségével a csillapítandó periodikus zavarhatás változásait a struktúra követni tudja, így a maradó hibára a zérus érték elméletileg megcélozható. Ehhez szükség van egy – a zajjal korrelációban lévő – referenciajelre, amit általában valamilyen szenzor (pl. Hall-szondás jeladó vagy gyorsulásmérő) szolgáltat.

Az *AFA* is egy rezonátoros struktúra. Működését legegyszerűbben a DFT-t megvalósító struktúrát ábrázoló 3.8. ábra segítségével érthetjük meg. A jelmodell ebben az esetben:

$$y(n) = \mathbf{c}^T(n)\mathbf{x}(n)\,, \text{ ahol} \tag{3.16}$$

$$c_k(n) = \mathrm{e}^{j\frac{2\pi}{N}kn}\,; \quad k = -K \ldots K\,; \quad N = 2K+1 \tag{3.17}$$

Ebben az esetben az alapharmonikus $f_1 = \frac{2\pi}{N}$. Az állapotváltozós leírás most a következőképp írható fel:

$$\hat{\mathbf{x}}(n+1) = \hat{\mathbf{x}}(n) + \frac{1}{N} \cdot \overline{\mathbf{c}}(n)\,\{y(n) - \mathbf{c}^T(n)\hat{\mathbf{x}}(n)\} \tag{3.18}$$

$$\hat{y}(n) = \mathbf{c}^T(n)\,\hat{\mathbf{x}}(n) \tag{3.19}$$

Egy-egy csatorna működése szemléltethető, hogy a hibajelet először $\hat{c}_k(n)$ együttható zérus frekvenciára keveri, majd integrálás után a megfelelő $c_k(n)$ együttható keveri vissza az eredeti frekvenciára. Amennyiben a megfigyelő és a jelmodell pontosan illeszkedik egymáshoz, az állapotváltozók nem változnak. Ha azonban a bemenőjel eltérő frekvenciájú, akkor az állapotváltozó – állandósult állapotban – egy olyan forgó komplex vektor lesz, amelynek a forgási sebessége megfelel az aktuális frekvenciakülönbségnek. Ez használható fel a frekvencia adaptálására:

$$f_1(n+1) = f_1(n) + \frac{1}{2\pi N} \cdot \arg\left(\hat{x}_1(n+1) - \hat{x}_1(n)\right) \tag{3.20}$$

(3.17) felhasználásával $\mathbf{c}(n+1)$ kifejezhető:

$$c_k(n+1) = c_k(n) \cdot \mathrm{e}^{j\,2\pi f_1(n+1)k} \tag{3.21}$$

Ez azt jelenti, hogy a frekvencia adaptálása a $c_k$ együtthatók frissítésével történik.

Fontos megemlíteni, hogy a frekvencia bizonyos határon túli növekedésekor a legnagyobb frekvenciájú rezonátorok megközelíthetik az $f = 0{,}5$ frekvenciát, azaz a mintavételi frekvencia felét. A rendszer stabilitása és a véges beállás szempontjából azonban fontos, hogy a rezonátorok egymástól egyenletes távolságra helyezkedjenek el. Így ha az adaptá-

ció során valamely rezonátor túlzottan megközelíti az $f = 0{,}5$ frekvenciát, akkor ő és a tükörfrekvenciáján lévő rezonátor egymáshoz nagyon közel kerülhetnek. Ez akár a rendszer instabilitását is okozhatja, így ilyenkor ezt a rezonátort meg kell szüntetnünk (ki kell léptetnünk).

Hasonlóképpen, ha az alapharmonikus frekvenciája csökken, és újabb egész számszorosa még „elfér" az $f = 0{,}5$ frekvenciáig, akkor az egyenletes rezonátor-eloszlást biztosítandó új rezonátorokat kell beléptetnünk a rendszerbe. Ezen rezonátorokat a következőképpen kell inicializálni:

$$x_K(n+1) = x_{-K}(n+1) = 0 \tag{3.22}$$

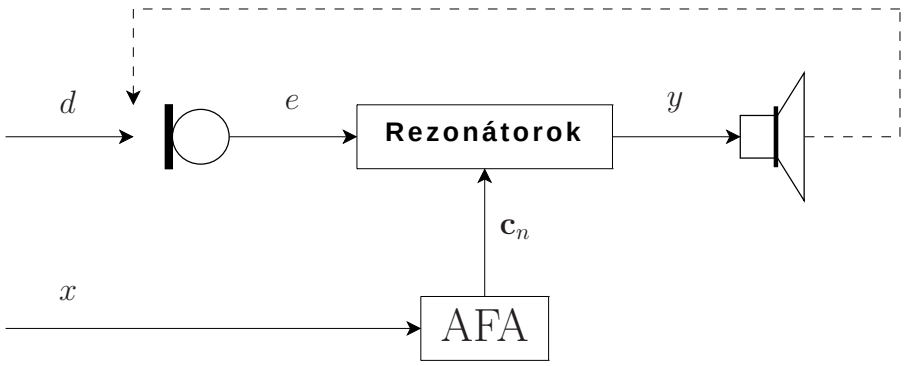
$$c_K(n+1) = c_{-K}(n+1) = 1 \tag{3.23}$$

A fentiekben az *AFA* jelen rendszerben használt formáját ismertettük. A gyakorlatban az adaptív Fourier analizátorok számos fajtáját alkalmazzák, ám a mi szempontunkból csak az itt elmondottak voltak lényegesek.

Fontos tehát, hogy a rezonátorokat AFA segítségével mindig a megfelelő frekvenciákra hangoljuk. Ez a hangolás alapvetően kétféleképpen történhet:

- Amennyiben adaptív rezonátorkészlettel valósítjuk meg a zajcsökkentő struktúrát, a teljes zajelnyomó hurok működik AFA-ként, ami lassabb frekvenciaadaptációt eredményez.
- A másik lehetőség, hogy a zajforrásból vett referenciajelet feldolgozó *külön rezonátoros struktúrával* valósítjuk meg az AFA-t, ami a zajelnyomásban részt vevő rezonátorokat hangolja (a $c$ együtthatók változtatása révén).

A két megoldás tulajdonképpen a 2.5. fejezetben már bemutatott *visszacsatolt* illetve *előrecsatolt* struktúrát valósítja meg, magában hordozva azok előnyeit és hátrányait. A mi zajcsökkentő rendszerünk az előre csatolt struktúrát követi, azaz a zajforrásból származó referenciajelet feldolgozó külön AFA blokk végzi a zajelnyomó rezonátorok adaptálását. A struktúra felépítését vázlatosan a 3.10. ábra szemlélteti. A jelölések a 2.2. ábra jelöléseit követik.




3.10. ábra. A zajcsökkentő struktúra vázlatos felépítése

# 4. fejezet

# Általános rendszerterv

## 4.1. A rendszer korlátai, hiányosságai

Az – eredetileg TDK dolgozat és diplomaterv keretein belül készült – aktív zajcsökkentő rendszer jelfeldolgozási szempontból kifogástalanul üzemelt, és teljesítette a tervezésekor megfogalmazott specifikációt, ám élesztése és üzemeltetése nehézségekbe ütközött. A sikeres üzembe helyezés megköveteli mind az alkalmazott fejlesztőeszközöknek, mind magának a programkódnak a mélységi ismeretét.

Bár az önálló laboratóriumi munka során készült egy kezdetleges kezelői felület, az nem vont magával alapvető strukturális változtatásokat, és nem adott megoldást a rendszer általános hiányosságaira – csupán áttekinthetőbbé tette azt a legfontosabb működési jellemzők és paraméterek rendszerbe foglalásával és struktúrált megjelenítésével. Bár a rendszerrel kapcsolatos legfontosabb paraméterek áttekintésére és kezelésére alapvető támogatást nyújtott, és védelmet biztosított a paraméterek inkompatibilis megválasztása vagy inkonzisztens kezelése ellen, ezek futásidejű megváltoztatása továbbra sem volt lehetséges. A jelfeldolgozást továbbra is két különálló DSP program valósította meg (egyik az identifikációért, másik a zajcsökkentésért felelős), és a paraméterek hangolása, megváltoztatása továbbra is csak forráskód szinten volt lehetséges, így teljes újrafordítást és újbóli programletöltést vont maga után. Az identifikált átvitel szintén csak a program újbóli lefordítása és letöltése révén kerülhetett a processzor memóriájába.

Ezek a tulajdonságok a zajcsökkentő rendszert rugalmatlanná és nehezen kezelhetővé teszik, az örökös újrafordítások pedig – még egy mai korszerű PC-n is – rengeteg időt emésztenek fel. Az ehhez szükséges idő pedig a hasznos munkára fordítható időt csökkenti.

## 4.2. Célkitűzések; a továbbfejlesztés irányvonala

A felhasználói felület tervezését megelőzően, a koncepció megalkotásakor alapvetően a következőket tűztem ki célul:

- az *identifikáció*ért és a *zajcsökkentés*ért felelős kódrészek *egyetlen* DSP programban való egyesítése,
- a két fő funkció közötti egyszerű átváltás lehetővé tétele,
- a fő funkciók működésébe való – futás közbeni – beavatkozás lehetővé tétele,
- a DSP program forráskód-szintű módosításának, valamint újbóli és újbóli lefordításának szükségtelenné tétele,
- a rendszer gyors üzembe helyezhetősége és egyszerű használata,
- egyszerű és áttekinthető átkonfigurálhatóság, skálázhatóság, akár működés közben is,
- az eddigiek során emberi beavatkozást igénylő műveletek gépesítése, automatizálása, ennek révén
- az üzemeltető által elkövethető hibák lehetőségének minimálisra csökkentése, továbbá
- a magas szintű programozási koncepció megtartása az egyszerű és gyors továbbfejleszthetőség érdekében, valamint
- a grafikus ellenőrzési módok elérhetőségének megtartása (pl. átviteli függvények ábrázolása).

A fenti szempontok alapján olyan kezelői felület megalkotása tehát a cél, amely – a konkrét megvalósítás részleteit elfedve – a rendszer lényegi funkciójával közvetlenül nem összefüggő, gépies műveletek elvégzésének terhét leveszi az üzemeltető válláról. Azáltal, hogy megvalósítja a rendszer részegységei köztötti átjárást és a rendszerben előforduló adatok és paraméterek átfogó és következetes kezelését, egyrészt felesleges többletmunkától szabadítja meg a kezelő személyt, másrészt a szükséges műveletek automatizált elvégzése révén elejét veszi az esetleges helytelen konfigurációkból vagy emberi figyelmetlenségből adódó hibás működésnek.

Az egységes felhasználói felület révén csökken az üzemeltetési „overhead”, valamint ami még fontosabb, használatával kikerülhető a teljes rendszer és a fejlesztéshez használt eszközök konkrét működésének mélységi ismerete, amelyet az eddigi sikeres beüzemelés szigorúan megkövetelt. Mindazonáltal a felület megléte nem zárja ki bármely rész mélyebb megismerésének lehetőségét sem. Ez alapján egy ilyen átfogó felhasználói felülettel rendelkező rendszer a későbbiek során alkalmas lehet demonstrációs célokra, vagy egy hallgatói laboratóriumi mérés keretében történő alkalmazásra, ahol nyilvánvalóan nem várható el, hogy a hallgatók vagy az üzemeltetők tisztában legyenek minden szoftverkomponens és fejlesztőrendszer működésével, és mélységi ismeretekkel rendelkezzenek mind a szenzorhálózat, mind a jelfeldolgozó algoritmusok működésének részleteivel kapcsolatban.

## 4.3. Tervezési döntések és indoklásuk

Az előző szakaszban leírtak fényében nyilvánvaló, hogy a meglévő – jelfeldolgozási funkciókért felelős – kódrészeket egyetlen, jól strukturált DSP programban (projektben) kell egyesíteni, és biztosítani kell a lehetőséget a külső vezérlésre, valamint az ehhez szükséges adatcserékre – mindezt a lehető legkevesebb *overhead* okozása mellett, úgy, hogy a jelfeldolgozás sebessége, hatékonysága a lehető legkisebb mértékben csökkenjen. Figyelemmel kell lennünk arra is, hogy a rendszeren végrehajtott módosításoknak ne legyenek olyan mellékhatásai, amelyek a jelfeldolgozó algoritmus működésében zavart okozhatnak.

A következőkben sorra vesszük azokat a sarkalatos kérdéseket, amelyek a továbbiakban alapvetően meghatározzák a fejlesztés menetét.

### 4.3.1. A felhasználói felület és annak kapcsolata a DSP kártyával

A tervezés során legelőször a következő kérdésekkel kapcsolatban kell meghoznunk a szükséges döntéseket:

- Milyen komplexitású felhasználói felületet szeretnénk biztosítani a kezelő személy számára? Mely kivitel felelne meg legjobban a célkitűzéseknek?
- Milyen megjelenítőeszközökre és beavatkozó szervekre van szükség a kitűzött célok eléréséhez?
- Egyáltalán milyen funkciókat, beavatkozási lehetőségeket nyújtson a felület?
- Hogyan kapcsolódjon egymáshoz a jelfeldolgozó rendszer és a felhasználói felület? Miként valósuljon meg a kettő közötti kommunikáció?

Könnyen beláthatjuk, hogy ezek a jellemzők alapvetően meghatározzák majd a megvalósítandó rendszer tulajdonságait. A kivitelt illetően alapvetően kétféle megoldás kínálkozik: tervezhetünk *vezérlő célhardver*t önálló beviteli- és megjelenítő-eszközökkel, vagy – a ma oly népszerűnek számító *virtuális műszer* koncepció jegyében – alkalmazhatunk egy közönséges PC-t, mint központi elemet – természetesen a megfelelő szoftver-elemekkel kiegészítve. Ez utóbbi esetben az alkalmazandó szoftverek kiválasztása is megfontolást igényel, hiszen számos kiváló alternatíva közül van módunk választani. A következőkben sorra vesszük ezeket a lehetőségeket, kitérve azok előnyeire illetve hátrányos tulajdonságaira.

**Vezérlő célhardver** alkalmazása esetén érnénk el a legkompaktabb kivitelt, ám egyúttal ez a választás eredményezné a legkötöttebb, legkevésbé rugalmas rendszert is.

Szem előtt kell tartanunk, hogy ez a választás olyan hardvertervezési és kivitelezési feladatok megoldását is megkívánja, amelyek jelentősen lelassíthatják a fejlesztés menetét. Emellett a rugalmas bővíthetőség és továbbfejleszthetőség lehetőségét is elveszíthetjük, hiszen a később felmerülő igények kielégítése, vagy új funkciók implementálása adott esetben a hardveres részek újratervezésének (és legyártásának) szükségességét vonhatja maga után.

Az identifikáció során keletkező átviteli mátrixok pszeudo-invertálását a DSP-n vagy a vezérlő célhardver mikrokontrollerén – alacsony szinten – kellene implementálni. Ez elvi nehézséget ugyan nem jelent, ám – a be- és kimenetek számának függvényében változó darabszámú méretű komplex mátrixokról lévén szó – kevéssé praktikus.

Bizonyos esetekben szükség lehet az adott elrendezés identifikációja során nyert adatok elmentésére. Erre az információmennyiségből adódóan valamilyen adattároló médiumot (pl. *Compact Flash* kártyát) célszerű használni. Napjainkban egyre olcsóbban kaphatók ezek a nagy tárolókapacitású flash memóriák. Kezelésüket megoldhatjuk külön céláramkör alkalmazásával, vagy a mikrokontrollerből szoftveresen.

A vezérlőegység kapcsolata a jelfeldolgozó kártyával praktikusan USART interfészen keresztül valósulhat meg, hiszen ezt mind a DSP, mind a legtöbb mikrokontroller támogatja. A rendszer szívét képező jelfeldolgozó processzor I/O feszültsége 3,3 V, ami szintillesztő nélkül csatlakoztatható pl. az Atmel „L”[1] jelzésű AVR kontrollereihez [21].

A zajcsökkentő rendszert felügyelő személlyel való kapcsolattartást a beágyazott rendszereknél megszokott beviteli- és megjelenítő-eszközök segítségével kell lebonyolítani (pl. $4 \times 4$-es tasztatúra és pontmátrix LCD-kijelző). Bár ez meglehetősen kompakt megoldás, kényelmetlenné teheti a rendszer kezelését. Az LCD-kijelző nem teszi lehetővé a rendszer állapotára jellemző adatok egyidejű, áttekinthető megjelenítését. Az egyes funkciók és információk elérése többszintű menürendszer használatát tesz szükségessé.

Összességében elmondhatjuk, hogy ez a kivitel nem jelentene optimális választást egy olyan rendszer esetében, ahol éppen a könnyű kezelhetőséget és az áttekinthetőséget jelöltük meg elsődleges szempontként.

**PC és saját fejlesztésű célszoftver** (pl. Visual C++ környezetben) alkalmazása esetén rendelkezésünkre áll a PC képernyője mint megjelenítő eszköz, ami a széles matematikai eszköztárral párosulva szinte korlátlan lehetőségeket nyit előttünk. Azt is szem előtt kell azonban tartani, hogy ebben az esetben a felhasználói felület funkcióit sajátkezűleg kell implementálni – gondoljunk itt átviteli függvények tetszetős grafikus megjelenítésre, és változó méretű struktúrákon végzendő komplex matematikai műveletekre is. Habár a Világhálón nagy választékban állnak rendelkezésünkre a különböző mintapéldák és függvénykönyvtárak, a korszerű programozói módszerek és a fejlesztési idő tekintetében alulmaradunk a napjaink korszerű szoftvertechnológiája által nyújtott lehetőségekhez képest. Célszerűbbnek látszik tehát valamilyen kifejezetten tudományos célokra kifejlesztett szoftver alkalmazása. A következő pontokban két olyan programcsomagot vizsgálunk meg, amelyeket tudományos problémák megoldására hoztak létre, és amelyekkel a villamosmérnöki képzés során módunk volt behatóbban megismerkedni: az egyik a National Instruments által kifejlesztett *LabVIEW*[2], amely a cég által propagált *virtuális műszer* koncepciót képviseli, a másik

[1] Low Power
[2] Laboratory Virtual Instrument Engeneering Workbench

pedig a MathWorks-féle *Matlab*, amelyhez rendelkezésre állnak az ún. *toolbox*-ok, amelyek a mérnöki tudományágak szinte minden lényeges problémakörét felölelik.

**PC és *LabVIEW*:** A National Instruments terméke az ún *virtuális műszer* koncepcióra épül. Ennek lényege, hogy a PC-t egy olcsó, univerzális „műszermagnak" tekintjük, jelentős számítási kapacitással, nagyméretű memóriával és háttértárakkal, gazdag ember–gép kapcsolattal (billentyűzet, egér, grafikus kijelző, nyomtató) és különféle kommunikációs interfészekkel (soros port, Ethernet, USB stb.). Szükség esetén az adott feladathoz szükséges speciális I/O eszközökkel kell kiegészíteni. Ebből az univerzális „műszermagból" és a kiegészítő perifériákból sokféle konkrét „műszer" alakítható ki, annak függvényében, hogy milyen alkalmazói program fut éppen a számítógépen.

A LabVIEW adatfolyam alapú, magas szintű grafikus programozási nyelv és környezet, amely az elmúlt két évtizedben jelentős fejlődésen ment keresztül. A programcsomag fejlesztése során szem előtt tartották, hogy a legtöbb tudományos területen felmerülő irányítási problémák megoldására alkalmas legyen. Alkalmazásával eszköztárunk kibővül az igényesen kidolgozott GUI[3] elemekkel és a fejlett, műszerek kijelzőire emlékeztető megjelenítő-elemekkel. Ezeket a komponenseket mérnöki feladatok során felmerülő igények kielégítésére tervezték, segítségükkel hatékonyan tudunk tetszetős és praktikus kezelői felületet előállítani. Ez nem csak a munka menetét egyszerűsíti – jelentősen csökkentve a fejlesztési időt –, hanem programunkat egy magasabb kvalitási szintre is emeli. További pozitívum, hogy ebben az esetben készen kapjuk a kommunikációt lekezelő modulokat is.

Bár a matematikai VI[4]-ok és függvények között találunk mátrixok kezelésére alkalmasakat, a jelfeldolgozási problémák megoldása nem tartozik a *LabVIEW* erősségei közé.

**PC és *Matlab*:** A MathWorks által kifejlesztett Matlab egy matematikai programcsomag, amely integráltan tartalmaz számítási, megjelenítési és programozási környezetet. A mérnöki és természettudományok számos területéhez kapcsolódóan nyújt támogatást és segédeszközöket. Sokféle gyakran felmerülő számítási feladat elvégzéséhez rendelkezik beépített függvényekkel, ami tetemes programozói munkától szabadíthatja meg használóját. Természetesen a felhasználónak lehetősége van saját függvények definiálására is.

A *MATLAB* név a *matrix laboratory* kifejezés rövidüléseként jött létre. Ez nem véletlen, hiszen alapvető adateleme egy olyan tömb, amelynek elemei dimenzió nélküli értékek. Eredetileg arra tervezték, hogy a mátrixokkal és vektorokkal formalizálható tudományos feladatokat gyorsabban és hatékonyabban lehessen vele megoldani, mint a hagyományos programozási nyelveken (pl. C-ben).

Leggyakoribb alkalmazási területei a következők:

[3]Graphical User Interface
[4]Virtual Instrument; a LabVIEW-ban így nevezik a programokat.

- offline numerikus matematikai számítások,
- algoritmusok fejlesztése és tesztelése,
- adatgyűjtés,
- modellezés, szimuláció,
- adatelemzés és vizualizáció,
- tudományos és mérnöki ábrázolás,
- alkalmazásfejlesztés, grafikus felhasználói felületek fejlesztése.

Elmondható, hogy az évek során a Matlab szinte szabvánnyá nőtte ki magát mind az oktatásban, mind pedig a természettudományok, a mérnöki tudományok, a kutatás, és az ipari alkalmazások területén. Kiegészítők (ún. *toolbox*-ok) egész sora is a rendelkezésünkre áll, amelyek egy-egy problémakört céloznak meg. Ilyen témakörök pl. – csak néhányat említve – a digitális jelfeldolgozás, az irányítástechnika, a neurális hálók, a wavelet-analízis, vagy a szimuláció különböző területei. Nem elhanyagolhaó az a tény sem, hogy szinte minden elképzelhető környezetbe könnyűszerrel integrálható.

Az elemzett lehetőségeket vizsgálva megállapíthatjuk, hogy a *PC + Matlab* választás tűnik a legígéretesebbnek, mert ötvözi mindazokat a hatékony eszközöket, amelyekre a megfogalmazott célok eléréséhez szükségünk lehet. Egyetlen kiforrott, jól integrálható környezetben áll rendelkezésünkre a fejlett matematikai eszköztár, a rugalmas programozási környezet, és a kor színvonalának megfelelő grafikus alkalmazásfejlesztést segítő eszköz.

A tervezés következő sarkalatos pontja annak eldöntése, hogy milyen módon valósuljon meg a Matlab és a fejlesztőkártya közötti kommunikáció. Ezen a téren két ésszerű lehetőség kínálkozik:

**Soros porti kommunikáció a DSP kártyával.** Ez a – ma már hagyományosnak számító – átviteli mód nem túl nagy, de a vezérlés megvalósításához elegendő átviteli sebességet biztosít. Legfőbb előnye a széleskörű elterjedtsége: szinte minden mikrokontroller és DSP tartalmaz USART perifériát (az ADSP-21364 is), és a legtöbb PC is rendelkezik *RS-232* szabványú soros porttal – bár napjainkban ez utóbbi már egyre kevésbé igaz. Így a fizikai kapcsolat létrehozásához legfeljebb egy szintillesztő áramkör közbeiktatására van szükség.

A soros port hatékony kezelését a Matlab beépített függvényekkel támogatja, így PC-oldalról ez nem jelent nehézséget. A jelfeldolgozó processzor oldaláról szükség van egy USART periféria felkonfigurására, és a jelfeldolgozó programban az azt kezelő részek implementálására. A DSP soros portját alapvetően kétféle módon kezelhetjük: *interruptosan*, vagy ismétlődő lekérdezésekkel (ún. *polling*gal). Az interruptos kezelés további megfontolásokat igényel; gondoskodnunk kell róla ugyanis, hogy a kommunikáció lekezelése a jelfeldolgozó rutinok futásában ne okozzon zavart. Pollingos kezeléskor ilyen probléma nem léphet fel, „csupán” a futásidő-növekedés történik. Ilyenkor arra kell ügyelnünk, hogy a rutinok lefutása worst case esetben is beleférjen a mintavételi időközökbe.

**Kommunikáció USB porton keresztül.** Előnyei a soros porthoz viszonyítva a nagy adatátviteli sebesség, valamint, hogy a fejlesztői kártya eléréséhez és programozásához amúgy is szükséges USB interfészen keresztül valósul meg a kommunikáció – így nincs szükség további kábelezésre.

Az *ADSP-21364 EZ-KIT Lite* fejlesztőkártya úgy van kialakítva, hogy USB interfészén keresztül teljeskörűen hozzáférhetünk a processzorhoz, és a kártyán elhelyezkedő minden perifériához. Ez azt jelenti, hogy beavatkozhatunk a processzor működésébe, elindíthatjuk vagy megállíthatjuk az egyes folyamatok futását, vagy írhatjuk és olvashatjuk a DSP memóriaterületeit. Ez rendkívül kedvező számunkra, hiszen a nagyméretű adatstruktúrák átvitele[5] közvetlenül a processzor memóriájába illetve a memóriájából történhet, mintegy DMA-szerűen, a processzormag igénybe vétele nélkül. Ennek és a gyors USB kapcsolatnak köszönhetően az adatátviteli idők jelentősen lerövidülnek.

Ezt a hozzáférést a fejlesztőkártya *eszközmeghatjó programja* (idegen szóval driver-e) teszi lehetővé számunkra, egy API[6] segítségével. Az API funkcióit COM[7] hívások útján érhetjük el.

A felvázolt lehetőségek közül az utóbbi, korszerűbbnek számító USB-kapcsolatot alkalmazó megoldás mellett döntöttem. Ebben az esetben a jelfeldolgozó program működésébe legpraktikusabban változók és flag-ek módosításával tudunk beavatkozni. Ezt a mechanizmust vázolja fel a 4.1. ábra.

### 4.3.2. A kezelői felület kialakításáról

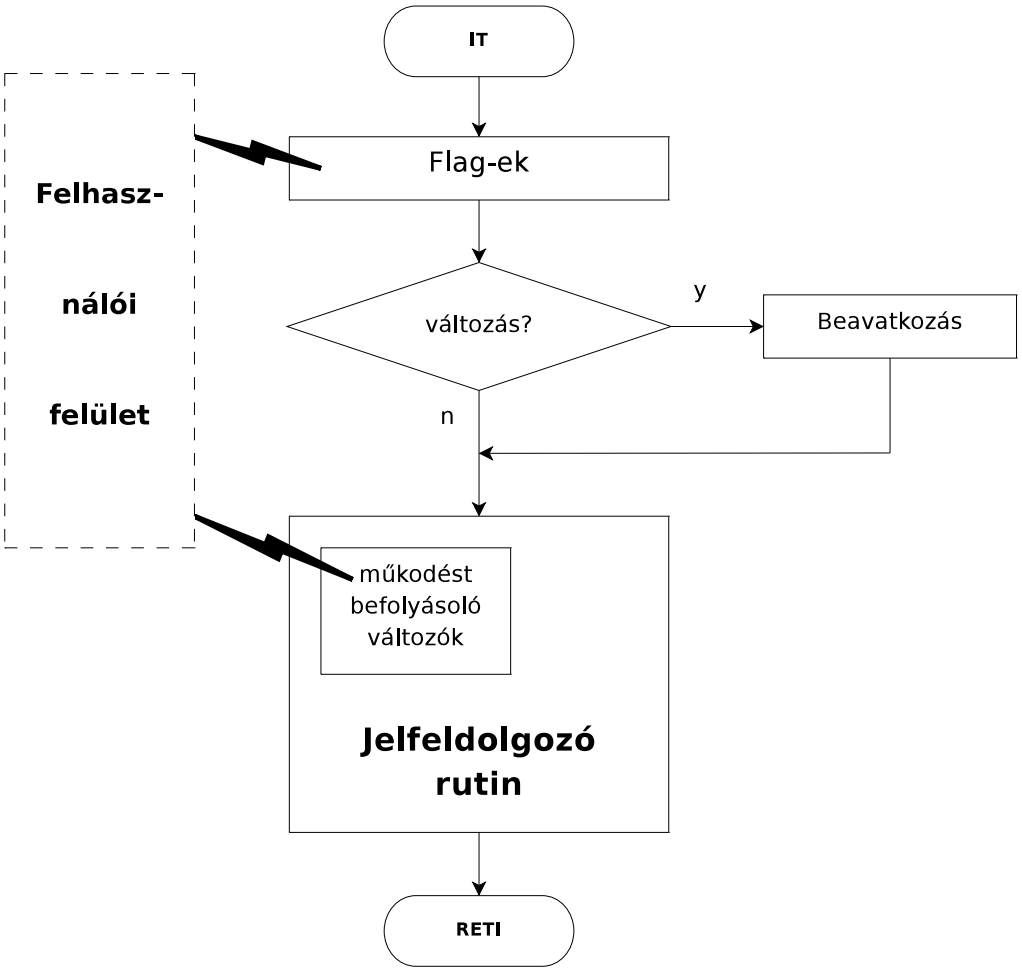
A grafikus megjelenítőeszközök az elmúlt évtizedekben jelentős fejlődésen mentek keresztül. Ez a fejlődés indította el azt a törekvést, hogy rendszereinket az emberi gondolkodásmódhoz közelebb álló, a vizuális megjelenítés előnyeit kiaknázó kezelőszervekkel lássuk el. Ez eleinte a menüvezérlés alkalmazását jelentette, ám napjainkban egy korszerű rendszer elképzelhetetlen tetszetős, grafikus felhasználói interfész nélkül.

Grafikus felhasználói felületek tervezésével számos könyv és egyéb irodalom foglalkozik. Ezek áttekintése során levonhatjuk azt a következtetést, hogy a fő alapelvek az idők során nem változtak. A téma rövid, összefoglaló áttekintését adja [16] és [17]. A tervezés során is az itt ismertetett szempontokat és alapelveket tekintettem irányadónak. Ezek közül talán az *egyszerűségre való törekvés* a legfontosabb: a kezelői felület legyen egységes kialakítású, letisztult, elegáns megjelenésű. Kerüljük a lényeges információt nem hordozó, esetleg figyelemelterelő elemek alkalmazását. Ne feledjük: a vizuális szemléletmód inkább a *minőségi*, mintsem a *mennyiségi* jellemzők hangsúlyozására épít. Törekedjünk a konzisztenciára, és arra, hogy a GUI használata könnyen elsajátítható legyen. Tartsuk szem előtt, hogy a felhasználói felület célja egy *adott személy* segítése, egy *adott feladat* megoldásában.

[5] Az identifikáció során kapott értékek kinyerése, és a pszeudo-invertált mátrixok visszatöltése.

[6] Application Programming Interface, azaz Alkalmazásprogramozási felület. Lásd bővebben az 5.1.1. fejezetben.

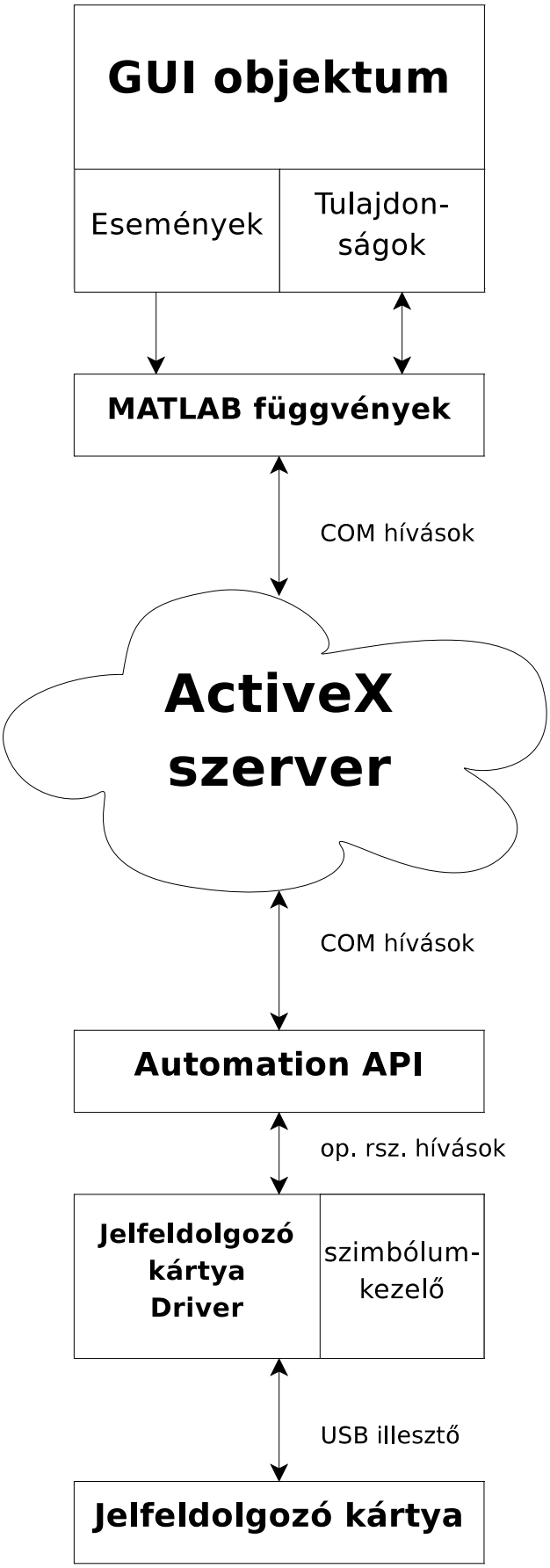
[7] Component Object Model, lásd az 5.1.1. és az 5.1.2. szakaszokat.




4.1. ábra. Beavatkozás a DSP program működésébe

### 4.3.3. Funkcionális blokkvázlat

A tervezési döntések összefoglalásaként 4.2. ábrán látható blokkvázlatot rajzolhatjuk fel, amely a rendszer funkcionális elemeit és a közöttük megvalósítandó kapcsolatokat hivatott áttekinthetően illusztrálni. Az alkalmazott komponensek és technológiák bemutatására az implementáció ismertetése során térünk ki.




4.2. ábra. A funkcionális elemek közötti kapcsolatrendszer

# 5. fejezet

# Megvalósítás

## 5.1. A megvalósítás során alkalmazott szoftver-komponensek bemutatása

Az implementáció tárgyalása előtt szükséges az annak során felhasznált két fontos alapelem: az *Automation API* és a *Component Object Model* rövid bemutatása.

### 5.1.1. Az Automation API

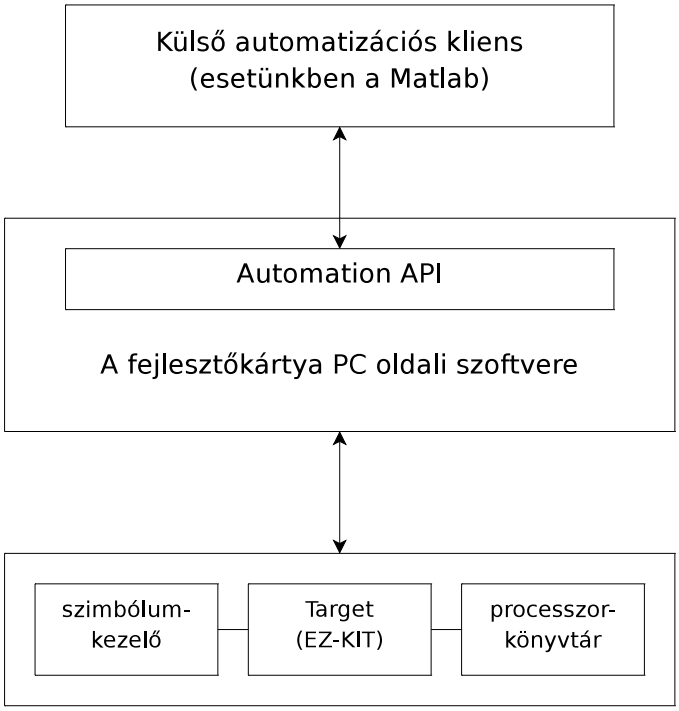
Az *EZ-KIT Lite* fejlesztőkártya PC oldali szoftvere biztosít számunkra egy ún. *alkalmazásprogramozási felület*et, idegen szóval *API*-t. Ennek segítségével lehetőségünk nyílik a VisualDSP++ fejlesztőeszköz használata nélkül hozzáférni a jelfeldolgozó processzorhoz, annak memóriablokkjaihoz, valamint a fejlesztőkártyán található egyéb perifériákhoz.

Az API elérése COM interfész-objektumokon keresztül valósul meg. Az interfész-objektumok *tulajdonságokkal*, *metódusokkal* és *eseményekkel* bírhatnak. A *tulajdonságok* egyszerű attribútumok, amelyek az adott objektumról hordoznak információt. Ilyen információ lehet pl. az objektum neve. A *metódusok* valamilyen – az adott objektummal kapcsolatos – operációt írnak le. Pl. egy memória-objektum metódusa lehet adott címen található szó olvasása vagy írása. Az *esemény*ekre megszakításokhoz lehet hasonlítani – bekövetkezésükkor értesítést küldenek a kliens számára.

A kliens és API közötti kapcsolat a Microsoft Windows beépített COM kezelő rendszerén keresztül valósul meg. Az API funkciói ezen kívül az ún. *Microsoft ActiveX* keretrendszer segítségével is elérhetők. Az *Automation API* kapcsolatát az egyéb szoftverkomponensekkel az 5.1. ábra szemlélteti.

### 5.1.2. Szoftvertechnológiai háttér – Néhány szó a COM-ról

A *Component Object Model* (COM), amely *ActiveX*-ként is ismert, a *Microsoft* által kifejlesztett technológia a komponens alapú fejlesztés támogatására, amely a különálló szoftverek közötti kommunikációt teszi lehetővé. Bár a technológiát több platformon is megvalósították, elsősorban Microsoft Windows operációs rendszereken használják. Az elődje az *Object Linking and Embedding* (OLE) technológia volt. Ma a COM szerepét a Microsoft *.NET* rendszer veszi át.



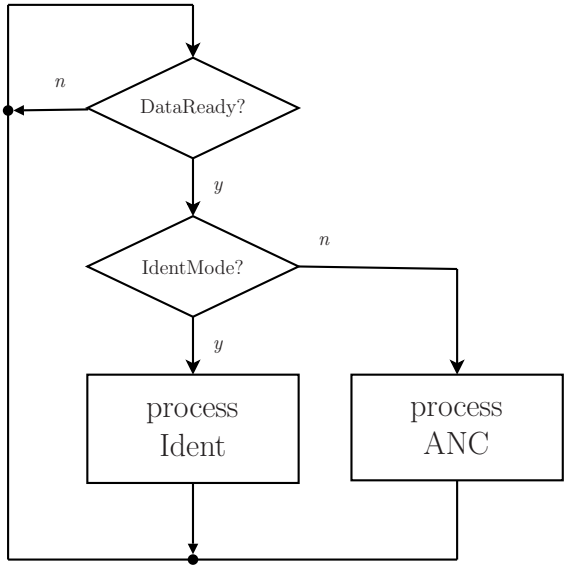

5.1. ábra. Az Automation API kapcsolata a többi szoftverkomponenssel

A különböző komponens-típusok osztályazonosítókkal (*class ID*, CLSID) azonosíthatók, amelyek globálisan egyéni azonosítók, avagy *GUID*-ok (Globally Unique Identifier). A globálisan egyedi azonosító, avagy GUID egy a szoftveralkalmazások által használt álvéletlen szám. Minden generált GUID egyedisége „matematikailag garantált". Ez azon az egyszerű törvényszerűségen alapszik, hogy az egyedi kulcsok száma annyira nagy ($2^{128}$, azaz $10^{38}$ nagyságrend), hogy két azonos szám generálásának a valószínűsége gyakorlatilag nulla. A GUID a Microsoft implementációja az *Univerzálisan egyedi azonosító* avagy *UUID* szabványnak, amelyet az *Open Software Foundation* (OSF) specifikált. Ez alapvetően egy 16 bájt hosszú szám, melyet hexadecimális formában írnak le. A GUID-ot 1 négybájtos, 3 kétbájtos és 1 hatbájtos szóval írják le, pl.: {3F2504E0-4F89-11D3-9A0C-0305E82C3301}. A Microsoft COM esetén a GUID-ok egyértelműen megkülönböztetik a különböző szoftverkomponens interfészeket. Ez azt jelenti, hogy egy komponens két verziójának azonos lehet a neve, de különböznek a felhasználók felé, ha a GUID-ok változtak. A Microsoft Office programok GUID-ot illesztenek be a dokumentumokba, mert azokat objektumoknak tekintik.

Minden COM komponens egy vagy több interfészen keresztül szolgáltat funkciókat. A különböző, komponens által támogatott interfészek interfész-azonosítókkal különböztethetőek meg egymástól (*interface ID*, IID), amelyek szintén *GUID*-ok.

A COM interfészeknek több programozási nyelvhez van kötése, például *C*, *C++*, *Visual Basic*, valamint több Windows platformon implementált script nyelvhez. A komponensekhez minden hozzáférés az interfészek metódusain keresztül történik. Ez olyan lehetőségeket kínál, mint a folyamatközi illetve a számítógépközi programozás.

A komponensek COM típuskönyvtárakkal (*COM Type Library*) tudják önmagukat leírni. A típuskönyvtár olyan információkat tartalmaz, mint a komponens CLSID-je, a kompo-




5.2. ábra. Az üzemmódok kezelése

nens által megvalósított interfészek IID-ja, valamint ezen interfészek metódusainak leírása. A típuskönyvtárakat jellemzően RAD[1] környezetek használják, mint a *Visual Basic* vagy a *Visual Studio*, hogy segítsék a fejlesztőt a komponensek létrehozása során.

A COM osztályokat Globálisan egyedi azonosítók (GUID) azonosítják. A COM osztályokat regisztrálni kell a Regisztrációs adatbázisban. COM objektumokra folyamaton belül, folyamatok között és hálózaton keresztül lehet hivatkozni.

## 5.2. A jelfeldolgozó algoritmusok egyesítése

Mint azt a rendszer hiányosságainak tanulmányozása során említettük, a hatékony munka egyik legnagyobb korlátja, hogy a működéshez szükséges jelfeldolgozó algoritmusok implementálva vannak ugyan, de az identifikációért és a zajcsökkentésért felelős kódrészek külön programot képeznek. A két projektben a fejlesztőkártya perifériáit inicializáló és kezelő rutinok megegyeznek, ezekhez kapcsolódik a kétféle feladatot megvalósító jelfeldolgozó függvény, és az őket „kiszolgáló” függvények. Eredetileg az identifikáló program függvényei a `processRezIdent.c`, a zajcsökkentő program függvényei pedig a `processRezANC.c` forrásfájlban foglalnak helyet. Az egyesítés során ezeket a függvényeket kellett egyetlen projektbe foglalni, és megoldani, hogy mindig az éppen kiválasztott funkciónak megfelelő rutinok hívódjanak meg. Ezt az `IdentMode` flag bevezetésével érjük el. Amikor új adat érkezik a mótoktól, ennek a flagnak a vizsgálatával állapítjuk meg, milyen módban vagyunk, és a megfelelő feldolglozó függvényt hívjuk meg. Ezt a folyamatot szemlélteti az 5.2. ábra.

[1]Rapid Application Development,
lásd: `http://www.blueink.biz/RapidApplicationDevelopment.aspx`

## 5.3. A futás közbeni átkonfigurálhatóság lehetővé tétele

A zajcsökkentő rendszerrel való kísérletezés során gyakran van szükség a rendszer méretének – a bemenetek és kimenetek számának – a megváltoztatására. Az eredeti rendszerben ez is csak forráskód szinten volt lehetséges, ami minden átkonfiguráláskor a teljes zajelnyomó program újrafordítását vonta maga után.

Első lépésként definiálnunk kell a be- és kimenetek *maximális* számát, amelyek kezelésére a rendszert felkészítjük. A mikrofonszám növelésének elsősorban az adatgyűjtő hálózat szab határt. A tapasztalatok azt mutatják, hogy a Fourier-dekompozíciós előfeldolgozást végző szenzorhálózat legfeljebb 6 db csomóponttal működik üzembiztosan és megbízhatóan. A kimenetek számát a fejlesztőkártya analóg kimeneteinek száma határolja. Mivel a 7. és 8. csatornát hibakeresési célokra tartjuk fent, a beavatkozó hangszórók maximális számát is 6 db-ban állapítjuk meg.

A bemenetek számának változtatását két szinten kell megoldanunk: a *szenzorhálózat* szintén, és a *jelfeldolgozó algoritmusok* szintjén.

A szenzorhálózat felépítéséből és kezeléséből adódóan a maximálisnál kevesebb csomópont használata nem okoz problémát; emiatt nincs szükség a mótok újraprogramozására[2]. Csupán arra kell odafigyelni, hogy mindig a kisebb azonosítóval rendelkezőket használjuk, a legnagyobbakat elhagyva. A jelfeldolgozó program szintjén úgy oldjuk meg a kérdést, hogy ha az aktuálisan kijelölt bemenetszám a maximális értéknél kisebb, akkor a beavatkozó jel aktuális értékének kiszámítása során mindig csak a beállított mikrofonszámnak megfelelő mótoktól származó adatokat vesszük figyelembe.

A beavatkozó hangszórók számának változtatása a másodlagos út identifikált átvitelét feldolgozó eljárásba épül be. A DSP memóriájából kinyert együttható-mátrixok pszeudoinvertálását Matlab függvény végzi (`atvKarInv`). A művelet során úgy módosítjuk az átviteli függvények együtthatóit, mintha a nem használt beavatkozó hangszórók és a mikrofonok között zérus volna az átvitel. Ennek a módosításnak a menetét szemlélteti az 5.1. lista. A beavatkozó hangszórók számának változtatása után tehát az együtthatók feldolgozását és letöltését újra el kell végezni.

Lista 5.1. Az átviteli karakterisztika módosítása a hangszóró- és mikrofonszámnak megfelelően

```
for hangszoro=1:hangszoroszam
    for mikrofon=1:mikrofonszam
        [...]
        if ((hangszoroszamUse < hangszoro) | (mikrofonszamUse <
            mikrofon))
            kar{hangszoro, mikrofon} = zeros(size(kar{hangszoro,
                mikrofon}));
        end
    end % for hangszoro
end % for mikrofon
```

[2] lásd a szenzorhálózat működéséről szóló 3. fejezetet.

## 5.4. A kezelői felület kivitelezése

Amint azt tervezési döntések indoklásánál már láttuk, a Matlab rendelkezik egy grafikus felhasználói felületek fejlesztésére alkalmas környezetettel. A *GUIDE* eszköz objektumok széles skálájával igyekszik megkönnyíteni a kezelői felületek tervezésének és programozásának folyamatát.

### 5.4.1. A grafikus felhasználói felület alapelemei

A *Matlab GUIDE* a Matlab promptról a

```
>> guide
```

paranccsal indítható. Ennek kiadása után a `GUIDE Quick Start` ablak fogad minket, ami két lapra tagolódik. A `Create New GUI` lapon kiválaszthatjuk, milyen típusú felületet tervezését szeretnénk megkezdeni, míg az `Open Existing GUI` lapon az előzőleg szerkesztett projekteket nyithatjuk meg. Kezelőfelületünk tervezését az első lapon szereplő `Blank GUI (Default)` opció kiválasztásával kezdjük meg. Az `Ok` gombra való kattintás után az 5.3. ábrán látható ablak fogad minket. Egy a *Visual Studio*-ra hasonlító – ám annál jóval egyszerűbb – layout szerkesztőt kapunk. Az ablak bal oldalán látható ún. *komponens eszköztáron* találjuk a rendelkezésünkre álló komponenseket, amelyek felhasználásával programunk felületét kialakíthatjuk. Ezek a következők:

**Push Button:** egyszerű nyomógomb, amely megnyomásakor generál eseményt.

**Toggle Button:** speciális nyomógomb, amely tulajdonképpen egy kétállapotú kapcsoló. Minden megnyomáskor eseményt generál; aktuális állapotát a callback függvényében olvashatjuk ki.

**Radio Button:** a grafikus kezelői felületekről jól ismert rádiógomb. Jellegzetessége, hogy az egy gombcsoporthoz (Button Group, lásd lentebb) tartozó rádiógombok között kölcsönös kizárás áll fenn, azaz közülük mindig csak egy lehet aktív.

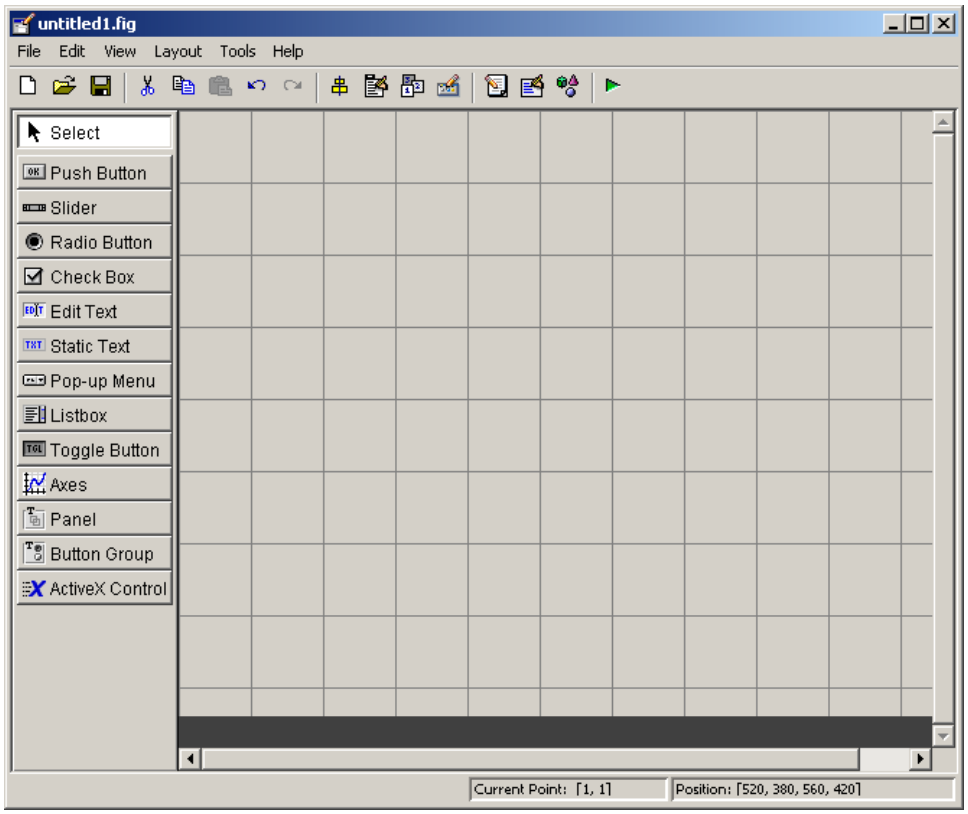
**Check Box:** a rádiógombokkal ellentétben ennél az elemnél nem áll fenn kölcsönös kizárás; egy csoporton belül egyszerre több is aktív lehet.

**Edit Text:** a felhasználó által szerkeszthető szövegmező. Legtöbbször adatbeviteli célokat szolgál.

**Static Text:** szöveges információ megjelenítésére szolgál. A felhasználó által interaktív módon nem szerkeszthető. Tipikusan ilyen elemek a felhasználói felület címkéi, feliratai.

**Slider:** csúszka, amelynek mozgatásával a felhasználó numerikus értéket választhat ki egy adott tartományból. A csúszka helyzete a kiválasztott érték adott tartománybeli elhelyezkedését illusztrálja.

**List Box:** elemek egy listáját tartalmazza, amelyekből a felhasználó egyet vagy többet kiválaszthat.




5.3. ábra. Új GUI létrehozása

**Pop-Up Menu:** legördülő menü. Lehetőségek egy listáját kínálja fel, amelyek közül a felhasználó választhat.

**Axes:** függvények, grafikonok vagy képek megjelenítésére szolgáló elem. Tujaldonságai és megjelenése széles körűen konfigurálható.

**Panel:** GUI komponensek csoportba foglalására, rendszerezésére szolgál. Vizuálisan elkülöníthetjük vele a logikailag összetartozó felületi elemeket. Az egy *Panel*en elhelyezett objektumok pozíciója a panel pozíciójához viszonyított relatív értékként értelmezett, aminek köszönhetően a kérdéses *Panel* áthelyezésekor a hozzárendelt komponensek vele együtt mozognak.

**Button Group:** „Toggle Button” és „Radio Button” komponensek olyan csoportokba foglalására szolgál, amely csoportokon belül érvényes a kölcsönös kizárás.

Az egyes komponensek különféle *tulajdonságok*kal és *esemény*ekkel rendelkeznek. A tulajdonságok közé tartozik pl. egy nyomógomb színe, felirata, mérete, pozíciójának koordinátái, stb. Az eseményekhez ún. *callback* függvények vannak rendelve, amelyek az esemény bekövetkeztekor (egy nyomógomb esetén pl. annak megnyomásakor) kerülnek meghívásra. Minden objektum egyedi azonosító névvel, ún. *tag*-gel rendelkezik, amelynek segítségével hivatkozhatunk rá. Ennek a névnek a segítségével férhetünk hozzá az objektum tulajdonságaihoz, illetve rendelhetünk függvényeket az egyes eseményeihez.

### 5.4.2. A GUI leírása

A GUIDE eszköz két, szorosan összetartozó fájlban írja le a felhasználói felületünket:

**Az az ún. *FIG* fájl** (`.fig` kiterjesztés) a GUI kinézetét (az ún. *layout*-ot) írja le. Ez a fájl bináris formátumú; közvetlenül nem szerkeszthető.

**Az ún. *M-fájl*** (`.m` kiterjesztés) tartalmazza a függvényeket, amelyek segítségével elvégezhetjük a GUI programozását. A felülethez kapcsolódó inicializációs résszel kezdődik, amit érintetlenül kell hagyni; a felhasználó ezt nem szerkesztheti. Ezt követi a GUI megnyitásakor végrehajtódó *Opening function*, amely tulajdonképpen a felület legelőször végrehajtódó callback függvénye, így ebben helyezhetjük el saját inicializációs kódunkat. Ez a függvény ugyanis közvetlenül a felület megjelenítése előtt hajtódik végre. Lehetőség van egy *Output function* definiálására is, amelynek segítségével visszatérési értéket adhatunk át a Matlab parancssor számára.

Ebben a fájlban kell elhelyezni továbbá az objektumaink eseményeihez rendelt ún *callback* függvényeket, és az egyéb, eseményhez nem rendelt Matlab függvényeinket is.

Célszerű ezt a fájlt a Matlab beépített M-fájl szerkesztőjével szerkeszteni; ilyenkor rendelkezésünkre áll a szintaxisnak megfelelő kiemelés, a zárójel-párok kiemelése, és a cellás megjelenítési mód.

Komponenseket úgy adhatunk hozzá a layout-hoz, hogy az eszköztáron kiválasztjuk a kívánt típust, majd az egér segítségével a layout-on elhelyezzük. Az aktív objektum tulajdonságait a *Property Inspector* nevű eszközzel tekinthetjük meg ill. módosíthatjuk. Ez az eszköz a `View` menüből, a felső eszköztárról, vagy magán az objektumon való dupla kattintással aktiválható. Az egyes komponensek tulajdonságait [18] irodalom „Uicontrol Properties” című fejezete kimerítően ismerteti.

### 5.4.3. A felhasználói felületet tagolása

A felhasználói felület vezérlő- és látványelemeinek kialakítása a 4.3.2. szakaszban taglalt irányelvek szerint történt. Az 5.4. ábra tanulmányozása során hamar szembeötlik, hogy a logikailag összetartozó komponensek csoportokat alkotnak. Minden csoportot egy-egy *panel* reprezentál, amely egyúttal a kezelőszervek vizuális elkülönítésére is szolgál. Összesen hat ilyen panel került kialakításra, amelyek – az egyszerűség és az áttekinthetőség jegyében – rövid, lényegre törő neveket kaptak:

**DSP program:** Ezen a panelon kaptak helyet a jelfeldolgozó programmal illetve annak futásával kapcsolatos legalapvetőbb műveletek. A `Fut` és `Megáll` nyomógombokkal indíthatjuk el illetve állíthatjuk meg a DSP program végrehajtását. A fejlesztőkártya inicializálására és a programletöltésre is itt van lehetőségünk.

A megállítás és az újbóli elindítás semmilyen más műveletet nem von maga után (megállításkor az állapotváltozók megtartják az adott pillanatbeli értékeiket).

**A rendszer mérete:** Itt állíthatjuk be a rendszer mindenkori méretét – a mikrofonok és a beavatkozó hangszórók számát – legördülő menük segítségével. A rendszer futás közbeni átkonfigurálása minden esetben a jelfeldolgozó program futásának átmeneti felfüggesztésével jár. Ilyenkor ugyanis – a rendszer új méreteit figyelembe véve – újra kell számolni a másodlagos út átviteli függvény-mátrixát, és le kell tölteni azt a processzor memóriájába.

**Együttható-készlet:** A zajelnyomó rendszerrel való kísérletezés során gyakran felmerül az az igény, hogy bizonyos – gyakran használt, vagy valamilyen szempontból fontosnak tartott – elrendezések identifikációja során nyert adatokat félretehessünk későbbi felhasználás vagy analízis céljából. „Kézi" módszerekkel természetesen ez eddig is lehetséges volt, ám ez a rendszer mélységi ismeretét feltételezte, és felhasználóbarátnak sem volt nevezhető.

A grafikus felület erre a következő megoldással szolgál: a rendszer kezelőjének rendelkezésére áll 6 db tárolóterület – ún. *bank* –, amelyek közül mindig pontosan egy lehet kiválasztva. A DSP kártya és a PC közti memóriaműveletek végrehajtásakor – PC oldalról – mindig az aktív tárhely vesz részt az adatcserében.

Ezekkel a tárolóterületekkel szoros kapcsolatban áll a panel fölött elhelyezett két nyomógomb: Az `Identifikált együtthatók kiolvasása` feliratú a legutóbbi identifikáció során mért értékeket olvassa ki a DSP memóriájából, és eltárolja őket a kiválasztott tárterületre. Az `Együtthatók betöltése zajcsökkentéshez` feliratú pedig az aktuális tárolóban lévő adatokat átalakítja a zajelnyomó algoritmus számára megfelelő formátumúra, majd letölti a DSP memóriájába. Ezen átalaktás során a rendszer aktuális méreteit is figyelembe kell venni.

A rendelkezésre álló tárolóterületek korlátozott száma – 6 db – megkötést jelent ugyan, ám gyakorlati tapasztalatok alapján ez a szám az esetek döntő többségében elegendő. E mellett a mellett szól, hogy általa gyorsabban és egyszerűbben kezelhetjük identifikált átviteleinket, mintha minden alkalommal fájlokat kellene mentenünk és betöltenünk.

**Funkció:** A panel rádiógombjai segítségével válthatunk a rendszer két fő üzemmódja között.

*Identifikáció* állásban a másodlagos út átviteli függvény-mátrixát tudjuk identifikálni. Az átviteli függvények mérése léptetett szinuszjellel történik, egyidejűleg egy hangszóró és az összes mikrofon között. A DSP kiadja a hangszórón a megfelelő frekvenciájú gerjesztőjelet, majd a mótoktól érkező adatok ismeretében kiszámítja az átviteli függvényeket. Az így nyert adatokat az előző pontban leírtak szerint ki kell olvasni a DSP memóriájából, majd a szükséges offline feldolgozást követően vissza kell tölteni azokat.

*Zajcsökkentés* állásban a zajelnyomó algoritmus futtatására nyílik lehetőség.

Az üzemmódok nevei mellett található egy-egy `Alaphelyzet` feliratú nyomógomb. Ezek a megfelelő rendszerrész állapotváltozóit hivatottak visszaállítani az alapértelmezett értékre.

**Identifikáció kontroll:** Ez a panel az identifikáció folyamatába való beavatkozást teszi lehetővé.

A hangszórók léptetésére akkor lehet szükség, ha egy már identifikált elrendezésnek valamilyen okból csak egy-egy csatornáját szeretnénk újraidentifikálni. Ez az eset áll fenn pl. akkor, ha egy beavatkozó hangszóró helyzetét megváltoztattuk, ám a többit érintetlenül hagytuk. Mivel a teljes rendszer identifikációja hosszadalmas folyamat, időt takaríthatunk meg azzal, ha csak a szükséges csatornákra ismételjük meg azt.

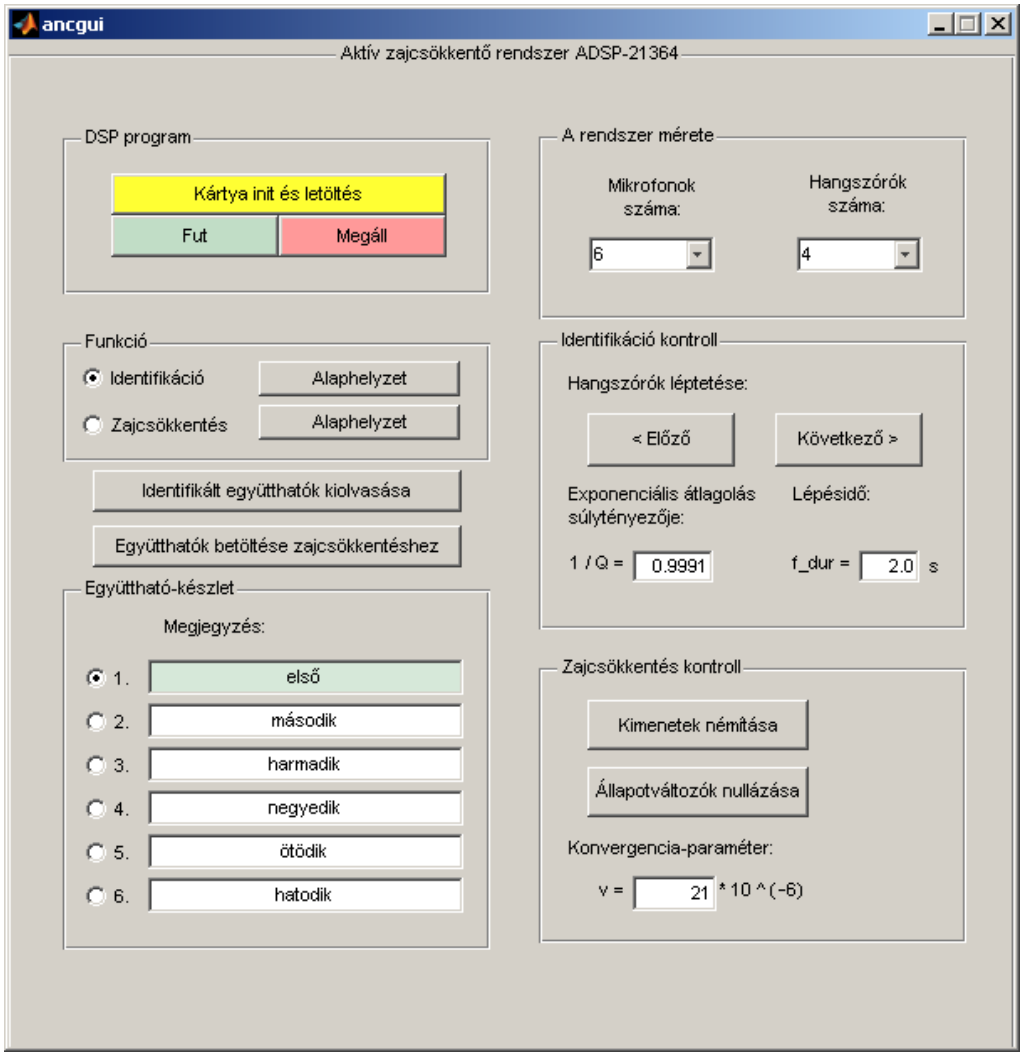
Identifikáció során a variancia csökkentése végett a mótoktól érkező Fourier-együtthatókra exponenciális átlagolást valósítunk meg. A súlytényező megválasztása során szem előtt kell tartani, hogy az első minták beérkezésekor még tarthatnak a teremben a frekvenciaugrás miatt lejátszódó tranziensek.

Egy frekvencián a mérés $t_{fdur}$ ideig tart. Amint az adott frekvencián töltött idő eléri ezt az értéket, az aktuális becslő mentésre kerül. $_{fdur}$ megválasztásánál úgy kell eljárni, hogy minden frekvencián legyen elegendő idő az állandósult állapot kialakulására, ám a teljes mérési idő ne adódjon túlságosan hosszúra.

**Zajcsökkentés kontroll:** A kísérletezés során gyakran van szükség a konvergencia-paraméter ($\nu$) megváltoztatására. Ezzel a rendszer tranziens viselkedését és stabilitási tulajdonságait hangolhatjuk.

A `Kimenetek némítása` feliratú kétállapotú nyomógomb leállítja a beavatkozójelet előállító rezonátorok frissítését, és némítja a beavatkozó hangszórókat. Segítségével a zajelnyomás ki- és bekapcsolt állapota közötti különbséget vizsgálhatjuk.

Az `Állapotváltozók nullázása` segítségével a rendszer beállási tulajdonságait vizsgálhatjuk. Ez a nyomógomb annyiban különbözik a `Funkció` panel `Alaphelyzet` nyomógombjától, hogy ennél csak a beavatkozó jelet előállító rezonátorok állnak alaphelyzetbe, míg a másiknál az AFA blokk állapotváltozói is újrainicializálódnak.




5.4. ábra. A grafikus felhasználói felület layout-ja

## 5.5. A működés ismertetése

Ebben a fejezetben a grafikus felhasználói felület belső működési mechanizmusainak ismertetésére kerül sor, kitérve a jelfeldolgozó kártyával való kapcsolat felépítésére és kezelésére. Nem cél a teljes forráskód listázása és taglalása; helyette kiragadott részletek elemzésén keresztül tárulnak fel a lényeges mozzanatok az Olvasó előtt.

Ahogy a legtöbb program, a GUI programja is inicializációs részekkel kezdődik. Ezeknek a kódoknak egy része a GUI indítása után automatikusan lefut, más részeit pedig maga a felhasználó hívja meg, a tényleges használatba vételt megelőzően.

Az indulást követő inicializálás[3] során beállítjuk a rendszer kezdeti méretét. A konzisztencia biztosítása érdekében ez úgy történik, hogy a `constants.h` fájlból kiolvassuk a jelfeldolgozó program fordítása során alkalmazott értékeket, és ezek alapján állítjuk be `A rendszer mérete` panel legördülő menüinek kezdeti értékeit. A folyamatot az 5.2. lista szemlélteti.

Lista 5.2. A kezdeti paraméterek beállítása

```
% init -- a rendszer inditaskori meretenek kinyerese a forrasfajlokbol
paramFileList = {'constants.h'};
handles.mikrofonUse   = findInHeaders('IN_NUM',     paramFileList);
handles.hangszoroUse = findInHeaders('OUT_NUM',    paramFileList);
handles.Data_Count    = findInHeaders('ID_TOP_VAL',paramFileList) *
    findInHeaders('IN_NUM',paramFileList) *
    findInHeaders('OUT_NUM',paramFileList);

% A 'handles' struktura frissitese
guidata(hObject, handles);

set(handles.num_mics,    'Value', handles.mikrofonUse);
set(handles.num_speaks, 'Value', handles.hangszoroUse);
```

A `findInHeaders()` egy Matlab függvény, ami az argumentumában megadott forrásfájlokban megkeresi az első argumentumban megnevezett konstant, és annak értékével tér vissza. Ezen kívül talán csak a `guidata(hObject, handles)` sor igényel magyarázatot. Ez frissíti a globális `handles` struktúra tagváltozóit, aminek révén azok aktuális értékei más függvényekből is „láthatók” lesznek. A `handles` struktúra tehát a függvényközi adatátadásokért felel.

[3] *Opening function*; lásd az 5.4. szakasznak az „M-fájl” részeiről szóló bekezdését.

A felhasználói felület elindításakor a jelfeldolgozó program még nincs letöltve a DSP memóriájába, és a fejlesztőkártyával való kapcsolat sincs még felépítve. Emiatt a felhasználói felület kezelőszervei az indítást követően még inaktív állapotban vannak. A DSP program letöltése, valamint a felhasználói felület és a jelfeldolgozó kártya közötti kapcsolat kiépítése a `DSP program` panelon található `Kártya init és letöltés` nyomógombra való kattintás után meg. A gombnyomáskor meghívódó *Callback* függvényt az 5.3. lista mutatja be. Röviden a következő történik:

- A Windows ActiveX szerver segítségével létrehozzuk az *ADspApplication* objektum egy példányát. Ez az objektum lesz az ún. *belépési pont*, ami biztosítja a fejlesztőkártyával való kapcsolatot.
- Nyitunk egy új munkamenetet (session-t). Itt adjuk meg az alkalmazott processzor és a fejlesztőkártya típusát.
- Megnyitjuk a zajcsökkentő projektet, és letöltjük a futtatható kódot a processzor memóriájába.
- Létrehozzuk a processzor és a memória kezeléséért felelős objektumok egy-egy példányát. Ezek belépési pontjait globális tárolási osztályban helyezzük el, hogy bármely függvényből hivatkozhatók legyenek.
- Engedélyezzük a felhasználói felület kezelőszerveit.
- Szót kell még ejteni az `Is_Running` flag-ről, amelyet szintén ebben a részben inicializálunk. Ennek segítségével fogjuk nyilvántartani, hogy a jelfeldolgozó program éppen fut-e egy esemény bekövetkeztekor. Erre azért van szükség, mert a processzor memóriájába csak akkor tudunk írni, amikor annak működése éppen fel van függesztve. Ha tehát a jelfeldolgozó program futása közben történik memóriaműveletet igénylő interakció, a memóriához való hozzáférés előtt meg kell állítani egy pillanatra a program futását, majd a művelet befejezte után újra engedélyezni kell azt.

  A kezelő személy egyébként bármikor megállíthatja vagy újra elindíthatja a jelfeldolgozó program futását a – szintén a `DSP program` panelon található – `Fut` és `Megáll` nyomógombokkal. Ezek megnyomása egyúttal az `Is_Running` flag-et is frissíti.

Lista 5.3. A „Kártya init és letöltés” nyomógomb ún. callback függvénye

```
%% --- Executes on button press in anc_init.
function anc_init_Callback(hObject, eventdata, handles)
% hObject    handle to anc_init (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global Is_Running processor memory_type_list memory_type anc_path;

anc_path = '.\';

h = actxserver('VisualDSP.ADspApplication');

app = h.invoke('IADspApplication');
app.Interactive = 0;

session = app.CreateSession('ADSP-21364 ADSP-21xxx EZ-KIT Lite',
    'EZ-KIT Lite(ADSP-21xxx)', 'ADSP-21xxx EZ-KIT Lite', 'ADSP-21364');

project_list = app.ProjectList;
project_list.Close()
project_list.invoke('AddProject',strcat(anc_path,'ANCrez.dpj'));
project = project_list.ActiveProject

processor = session.get('ActiveProcessor');

processor.LoadProgram(strcat(anc_path,'Debug\ANCrez.dxe'));

Is_Running = 0;

memory_type_list = processor.MemoryTypeList;
memory_type = memory_type_list.Item('Data(DM) Memory');

% Sarga szin levetele az init gombrol
    set(handles.anc_init, 'BackgroundColor', [0.831, 0.816, 0.784]);

    set(handles.text17, 'String', handles.mikrofonUse);
    set(handles.text18, 'String', handles.hangszoroUse);

% Kezeloszervek engedelyezese
    set(handles.anc_run,     'Enable', 'on');
    set(handles.anc_halt,    'Enable', 'on');
    set(handles.nu,          'Enable', 'on');
    set(handles.afil,        'Enable', 'on');
    set(handles.num_mics,    'Enable', 'on');
    set(handles.num_speaks,  'Enable', 'on');
```

A jelfeldolgozó program futását *flag-ek*, illetve *a működést befolyásoló változók* módosításával tudjuk befolyásolni. Flag beállítására mutat példát az 5.4. lista.

Ahhoz, hogy a flag-et módosítani tudjuk, szükségünk van annak – a DSP memóriablokkja beli – címére. Ezt a *szimbólumkezelő* használata segítségével deríthetjük ki, a `FindSymbol()` metódus alkalmazásával. Figyeljük meg, hogy a DSP memóriájába csak ún. *ADspValue* típusú értéket lehet írni. A művelet megkezdése előtt az `Is_Running` flag segítségével ellenőrizzük, hogy éppen fut-e a jelfeldolgozó program. Amennyiben igen, futását a memóriaművelet időtartamára fel kell függeszteni.

Lista 5.4. Flag bebillentése

```
IdentInit_Flag = actxserver('VisualDSP.ADspValue');
set(IdentInit_Flag, 'Value', '1');

IdentInit_list = memory_type_list.FindSymbol('IdentInit');
IdentInit = IdentInit_list.Item(0);

if Is_Running == 1
    processor.Halt(1)
end % if

    memory_type.FillMemoryWithValue(IdentInit.Address, 1, 1,
        IdentInit_Flag);

if Is_Running == 1
    processor.Run(0)
end % if
```

Változók értékét hasonlóképpen tudjuk módosítani. A különbség annyi, hogy a beírandó értéket megfelelő formátumú sztinggé kell konvertálnunk. Egész számok esetén ez a `num2str` Matlab függvény segítségével történhet. Lebegőpontos esetben az *IEEE 754*-es szabványnak megfelelő alakra van szükség. Ezt a `num2hex` Matlab függvénnyel állíthatjuk elő. Mivel a DSP 32 biten ábrázolja a lebegőpontos számokat, a `num2hex` argumentumában szereplő értékre a `single` függvényt is meg kell hívni. A konverzió folyamatát az 5.5. lista szemlélteti.

Lista 5.5. Lebegőpontos szám átalakítása a megfelelő formátumúra

```
uj_nu = actxserver('VisualDSP.ADspValue');
set( uj_nu, 'Value', num2hex(single(str2double(get(hObject,'String'))
    * 10^(-6)) ));
```

## 5.6. Értékelés, összefoglalás

Diplomatervem első részében rövid áttekintést adtam azokról a tényezőkről, amelyek az aktív zajcsökkentés létjogosultságát megteremtették. Röviden ismertettem a téma történetét, fejlődésének fontosabb állomásait. Bemutattam az aktív zajcsökkentés alapelveit, áttekintetést adtam az elméleti alapokról, a felmerülő problémákról, valamint az alapvető struktúrákról.

Ismertettem a diplomaterv alapját képező – a Méréstechnika és Információs Rendszerek Tanszéken kifejlesztett – szenzorhálózatos aktív zajcsökkentő rendszert, kitérve az annak egyes részegységeivel kapcsolatosan felmerülő problémákra, problémakörökre. Szó esett a jelfeldolgozó processzorokról illetve azok felépítéséről, valamint az egyre nagyobb népszerűségnek örvendő szenzorhálózatokról. Kitértem az elosztott adatgyűjtéssel kapcsolatosan felmerülő korlátoka és problémákra, majd – mintegy esettanulmányszerűen – ismertettem az alaprendszerben alkalmazott megoldásukat.

Összegeztem a rendszerrel kapcsolatban felmerülő legfontosabb hiányosságokat, majd a továbbfejlesztés irányvonalának kijelölése után elemeztem a különböző lehetőségeket. Ezek sajátosságainak figyelembevételével hoztam meg a tervezési döntéseket és a választottam ki a megvalósítás során alkalmazandó eszközöket.

A jelfeldolgozó algoritmusokat egyetlen, a rendszer összes funkcióját megvalósító DSP programba foglaltam össze. Megoldást adtam a különböző üzemmódok közötti rugalmas átváltásra, és a programmodulok strukturális átalakítása révén lehetővé tettem, hogy működésükbe futás közben is be lehessen avatkozni.

Annak érdekében, hogy a jelfeldolgozási–akusztikai problémát jól ismerő, de *az alkalmazott eszközök fejlesztésében járatlan* felhasználó is képes legyen kísérletező munkát végezni, a rendszerhez felhasználóbarát kezelői felületet is készítettem. Ennek kivitelezése során célom volt, hogy a kísérletező munkát gördülékennyé tegye, és a kezelése könnyen elsajátítható legyen. Ezt hivatott elősegíteni a függelék formájában közreadott „Rövid kezelési leírás” is.

A megvalósítás során alkalmazott eszközök és technikák bemutatása azt az igényt is szem előtt tartva történik, hogy megfelelő kiindulási alapot képezzen a rendszer esetleges későbbi továbbfejlesztéséhez.

Összességében levonható az a tanulság, hogy egy komplex aktív zajcsökkentő rendszer fejlesztése multidiszciplináris ismereteket igényel. Megköveteli mind az akusztikai, jelfeldolgozási, szabályozástechnikai jártasságot, mind az alkalmazott hardver-elemek, mind pedig a fejlett szoftvertechonógiai módszerek ismeretét.

## 5.7. Továbblépési lehetőségek, kitekintés

A felhasználóbarát rendszer kifejlesztésével a munka még nem ért véget, a kísérletes tapasztalatok alapján szükség lehet a rendszer szolgáltatásainak kisebb módosítására, illetve bővítésére. Az aktív zajcsökkentés területén hosszú utat kell még bejárni az olyan sokcsatornás rendszerek létrejöttéig, amelyek a mindennapi élet során is egyszerűen és sikeresen alkalmazhatók.

Rendszerünknél maradva vegyük sorra a továbblépés irányait!

Az első, szinte triviálisan adódó lehetőség a sokcsatornás rendszer maximális méretének növelése. Ennek felső korlátja elsősorban a jelfeldolgozó processzor teljesítőképessége. Amennyiben rádiós szenzorhálózatot alkalmazunk, a mikrofonok számát korlátozhatja annak átviteli sebessége.

További törekvés lehet, hogy a rendszert ne csak inkrementálisan lehessen átkonfigurálni, hanem a hibamikrofonok és a beavatkozó hangszórók tetszőlegesen kiválaszthatók legyenek.

Problémát jelenthet, hogy a léptetett szinuszos identifikáció a hangszórószám növekedésével egyre több időt vesz igénybe. Bizonyos méret fölött ez kezelhetetlenné teheti a rendszert, így szükséges lehet az identifikáció valamilyen módon való rugalmasabbá tétele, vagy egy teljesen új, gyorsabb identifikációs eljárás kidolgozása.

# Ábrák jegyzéke

# Forráslisták jegyzéke

# Irodalomjegyzék


[1] S. M. Kuo, D. R. Morgan, „Active Noise Control: A Tutorial Review", *Proceedings of the IEEE*, vol. 87, No. 6, pp 943-973, 1999

[2] Luenberger, D. G., „An Introduction to Observers", *IEEE Trans. on Automatic Control*, Vol. AC-16, No.6, 1971.

[3] BME MIT Tanszéki Munkaközösség, *Digitális jelfeldolgozás – Segédlet a „Digitális jelfeldolgozás" c. tárgyhoz*, kézirat, Budapest, 2006.

[4] Sujbert László, *Periodikus zavarhatások csökkentésének aktív módszerei*, Ph.D. dolgozat, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 1997.

[5] Lajkó László, Orosz György, *Aktív zajcsökkentő rendszerek megvalósítása szenzorhálózattal*, TDK dolgozat, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 2005.

[6] Orosz György, „Egyszerű adattovábbító szenzorhálózahoz tartozó leírás", `http://home.mit.bme.hu/~orosz/wirelessANC/Leiras/ANCsimaAdat.pdf`

[7] Orosz György, „Fourier-együtthatókat továbbító szenzorhálózahoz tartozó leírás", `http://home.mit.bme.hu/~orosz/wirelessANC/Leiras/ANCrezAdat.pdf`

[8] Orosz György, „Zajcsökkentő algoritmust futtató DSP leírása", `http://home.mit.bme.hu/~orosz/wirelessANC/Leiras/DSP_ANC.pdf`

[9] Elek Kálmán, *Adaptív jelfeldolgozás*, Muegyetemi Kiadó, 2005.

[10] P. Lueg, „Process of silencing sound oscillations", United States Patent Office, 1936.

[11] S. J. ELLIOTT, P. A. NELSON: „Active Noise Control", *IEEE Signal Processing Magazine*, October, pp 12-35, 1993

[12] Analog Devices, Inc., „ADSP-21364 EZ-KIT Lite Evaluation System Manual", Revision 3.0, 2006. augusztus, `http://www.analog.com/UploadedFiles/Associated_Docs/33168315105663ADSP_21364_EZ_KIT_Lite_Manual_Rev._3.0.pdf`

[13] Analog Devices, Inc., „SHARC Processors – ADSP-21362/ADSP-21363/ADSP-21364/ADSP-21365/ADSP-21366", Rev. D,

http://www.analog.com/UploadedFiles/Data_Sheets/ADSP_21362_21363_21364_21365_21366.pdf

[14] Analog Devices, Inc., „SHARC Processor Architectural Overview”, http://www.analog.com/processors/sharc/overview/archOverview.html

[15] Analog Devices, Inc., „Getting Started With SHARC Processors”, http://www.analog.com/UploadedFiles/Associated_Docs/352228244SHARC_getstart_online.pdf

[16] The MathWorks, Inc., „Building GUIs with MATLAB”, 5. kiadás, http://www.apc.univ-paris7.fr/~beau/IMG/pdf/buildgui.pdf

[17] The MathWorks, Inc., „MATLAB – The Language of Technical Computing – Creating Graphical User Interfaces”, 7. kiadás, http://www.ee.hacettepe.edu.tr/~solen/Matlab/MatLab%207.x/Matlab%207%20-%20Creating%20Graphical%20User%20Interfaces.pdf

[18] The MathWorks, Inc., „Full Product Family Help”

[19] Crossbow, „How Wireless Sensor Networks Work”, http://www.xbow.com/Technology/Images/wsn_demo.swf

[20] Crossbow, „MPR-MIB Users Manual”, Revision A, 2007. június http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf

[21] ATmega128(L) Data Sheet http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

[22] CC2420 2.4 GHz IEEE 802.15.4 ZigBee-ready RF Transceiver http://inst.eecs.berkeley.edu/~cs150/Documents/CC2420.pdf

[23] MTS Sensor & Data Acquisition Boards http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0047-01_B_MTS.pdf

[24] MIB510 Serial Interface Board http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB510CA_Datasheet.pdf

[25] TinyOS Site http://www.tinyos.net/

[26] nesC: A Programming Language for Deeply Networked Systems http://nescc.sourceforge.net/

[27] Analog Devices, Inc., „VisualDSP++ 4.1 Help” „Automation Overview” és „VisualDSP++ Automation API Reference” című fejezetei

# Függelék

## F.1. TinyOS telepítése Windows környezetben

A szenzorhálózatot, és a bázisállomást alkotó mote-ok mikrokontrollerein TinyOS beágyazott operációs rendszer fut. A mote-ok programozásához, valamint a szenzorhálózat szoftverének fejlesztéséhez és fordításához a számítógépre elsőként fel kell telepíteni a TinyOS rendszerét [25]. A munka során az 1.1.11-es verziót használtam, az arra telepített 1.1.15 frissítéssel, Windows XP operációs rendszer alatt.

A TinyOS rendszer főbb komponensei a következők:

- Cygwin – kötelező (Windows esetén)[4]
- nesC – kötelező
- TinyOS – kötelező – a TinyOS beágyazott operációs rendszer állományai
- TinyOS-tools – opcionális (része a párhuzamos porti programozást lehetővé tevő driver is, melyet nem használunk)
- Java SE (Standard Edition) Development Kit (JDK)[5]
- JavaCOMM (Java Communications API)[6]
- AVRtools (avr-binutils, avr-libc, avr-gcc, avarice, avr-insight).

A telepítés befejezése után egyes állományokon bizonyos rendszerfüggő módosításokat kell elvégezni, annak érdekében, hogy a fordításra kerülő kódok a mikrokontrollerek regisztereit megfelelően inicializálják. Ezek a módosítások a következők:

1. Az USART sebességét át kell állítani 115200 kbps-ra:
   (USART Baud Rate Register – UBRR[21]):
   A `HPLUART0M.nc` állományban (`/tinyos-1.x/tos/platform/micaz/`)
   67. sor: `outp(7,UBRR0L);`

2. A rádiócsatorna átállítása – a gyakorlatban az alapértelmezettként megadott 11-es csatorna a külső zavarhatások miatt nem megfelelő. Szükség esetén nagyobb stabilitás érhető el, ha a `CC2420Const.h` állományban (`/tinyos-1.x/tos/lib/CC2420Radio/`) átállítjuk a csatornát, pl:
   `#define CC2420_DEF_CHANNEL 26 //channel select`

Amennyiben a számítógépre a Cygwin környezet rendszergazdaként került telepítésre, a többi felhasználónak nincs joga a fenti állományokat módosítani. Ehhez a Cygwin fastruktúrájában látható összes állományt mindenki számára olvasási, írási és futtatási joggal kell felruházni, amely a Cygwin felületen kiadott `chmod -R 777 /*` parancs segítségével tehető meg.

[4] `http://cygwin.com/`, Linux-szerű felületet biztosít
[5] `http://java.sun.com/javase/`
[6] `http://java.sun.com/products/javacomm/`

### Alternatív megoldások

A TinyOS telepítése hosszadalmas folyamat, mert sok különböző gyártótól illetve készítőtől származó komponens összehangolt működését igényli. Ha a TinyOS fent leírt módon történő telepítését valamilyen okból kifolyólag nem tartjuk szükségesnek, lehetőség van a XubunTOS[7] használatára is. A XubuntTOS egy Xubuntu Linux-disztribúción alapuló live-CD, amely tartalmazza a TinyOS 1.1 és 2.0 verzióját egyaránt.

Ha a számítógépet a XubunTOS live-CD-ről indítjuk be, és a szükséges konfigurációkat elvégezzük, a mote-ok egyszerűen átprogramozhatók.

[7] `http://toilers.mines.edu/Public/XubunTOS/`

## F.2. Rövid kezelési leírás

### F.2.1. Hardver összeállítás

1. Csatlakoztassuk a jelfeldolgozó kártyát a PC-hez USB porton keresztül! A kapcsolat létrejöttéről az `USB monitor` feliratú LED tájékoztat.
2. A referenciajelet a fejlesztőkártya 1-es számú bemenetére kell csatlakoztatni (fehér csatlakozó).
3. A beavatkozó hangszórókat sorban a fejlesztőkártya kimeneti csatornáihoz csatlakoztatjuk. Maximálisan 6 db hangszóró csatlakoztatható; a 7. és 8. csatorna hibakeresési célokat szolgál.
4. A szenzorhálózat bázisállomásának *RS-232* portját – az annak illesztésére szolgáló csatlakozó segítségével – a fejlesztőkártya `DAI` jelzésű csatlakozósorához csatlakoztatjuk, a következő módon:
   - DAI2 – kék,
   - DAI4 – sárga,
   - DAI6 – zöld.
5. Indítsuk el a szenzorhálózatot.
6. Indítsuk el a kezelői felületet!

### F.2.2. A szenzorhálózat indítása

Az összes mote felprogramozása után a bázisállomást a programozó kártyára kell helyezni, amely ekkor soros porti illesztőként szolgál. A programozókártya `SW2` kapcsolóját OFF állásba kell kapcsolni annak érdekében, hogy a mote által küldött üzenetek a soros porton megjelenjenek, és azok a DSP (vagy hibakeresési célből egy PC) számára hozzáférhetőek legyenek.

A szenzorhálózat indítása a következő lépések végrehajtásával történik:

1. A bázisállomás programozó kártyára való helyezése, majd feszültség alá helyezése.
2. `SW2` OFF állásba kapcsolása.
3. Csatlakoztatás a DSP kártyához.
4. `RST MOTE` nyomógomb használatával a bázisállomás reset-elése (ekkor minden más mote-nak kikapcsolt állapotban kell lennie).
5. A szenzor mote-ok bekapcsolása sorra egymás után, legutoljára hagyva a legkisebb azonosítóval programozottat – a hálózati működést ennek az első üzenete indítja el.

### F.2.3. A felhasználói felület használatba vétele

A kezelői felület elindítása történhet az `ancgui.exe` bináris futtatásával, vagy a *Layout editor*-ból a `Tools`→`Run` menüpont segítségével (`CTRL-T` billentyűkombináció).

A fejlesztőkártya PC-re való csatlakoztatása után várjuk meg, amíg az `USB monitor` feliratú LED világítani kezd, az alkalmazást csak ezután indítsuk.

Indítás után az első teendő a jelfeldolgozó program letöltése a fejlesztőkártyára. Ez a `Kártya init és letöltés` nyomógombbal tehető meg.

### F.2.4. A felhasználói felület kezelésének rövid összefoglalása

A kezelői felület hat ún. panelra tagolódik. Az egyes panelokon elhelyezett beavatkozószervek funkcióinak rövid áttekintése:

**DSP program:** Itt van lehetőségünk a fejlesztőkártya inicializálására és a jelfeldolgozó program letöltésére. A `Fut` és `Megáll` nyomógombok segítségével bármikor megállíthatjuk, vagy újra elindíthatjuk annak futását.

A program futásának megállításakor az állapotváltozók megtartják az adott pillanatbeli értékeiket, így újbóli elindítás esetén a megállításkori üzemállapotban folytatódik a DSP program működése.

**A rendszer mérete:** Itt állíthatjuk be a rendszer mindenkori méretét – a mikrofonok és a beavatkozó hangszórók számát. Átkonfigurálás után – a rendszer új méreteit figyelembe véve – újra kell számolni a másodlagos út átviteli függvény-mátrixát, és le kell tölteni azt a processzor memóriájába (`Együtthatók betöltése zajcsökkentéshez` feliratú nyomógomb).

**Együttható-készlet:** A rendszer kezelőjének rendelkezésére áll 6 db tárolóterület – ún. *bank* –, amelyek közül mindig pontosan egy lehet kiválasztva. A DSP kártya és a PC közti memóriaműveletek végrehajtásakor – PC oldalról – mindig az aktív tárolóhely vesz részt az adatcserében.

Ezekkel a tárolóterületekkel szoros kapcsolatban áll a panel fölött elhelyezett két nyomógomb: Az `Identifikált együtthatók kiolvasása` feliratú a legutóbbi identifikáció során mért értékeket olvassa ki a DSP memóriájából, és eltárolja őket a kiválasztott tárterületre. Az `Együtthatók betöltése zajcsökkentéshez` feliratú pedig az aktuális tárolóban lévő adatokat átalakítja a zajelnyomó algoritmus számára megfelelő formátumúra, majd letölti a DSP memóriájába.

**Funkció:** A panel rádiógombjai segítségével válthatunk a rendszer két fő üzemmódja között. Az üzemmódok nevei mellett található egy-egy `Alaphelyzet` feliratú nyomógomb. Ezek a megfelelő rendszerrész állapotváltozóit hivatottak visszaállítani az alapértelmezett értékre.

**Identifikáció kontroll:** Ezen a panelon tudunk beavatkozni az identifikáció folyamatába.

A variancia csökkentése végett a mótoktól érkező Fourier-együtthatókra exponenciális átlagolást valósítunk meg. Itt tudjuk állítani ennek súlytényezőjét ($\frac{1}{Q}$), valamint a mérés során egy frekvencián töltött időt.

**Zajcsökkentés kontroll:** Megváltoztathatjuk a konvergencia-paramétert ($\nu$), illetve a rendszer különböző tulajdonságait vizsgálhatjuk.

A `Kimenetek némítása` feliratú kétállapotú nyomógomb leállítja a beavatkozójelet előállító rezonátorok frissítését, és némítja a beavatkozó hangszórókat. Segítségével a zajelnyomás ki- és bekapcsolt állapota közötti különbséget vizsgálhatjuk.

Az `Állapotváltozók nullázása` segítségével a rendszer beállási tulajdonságait vizsgálhatjuk. Ez a nyomógomb annyiban különbözik a `Funkció` panel `Alaphelyzet` nyomógombjától, hogy ennél csak a beavatkozó jelet előállító rezonátorok állnak alaphelyzetbe, míg a másiknál az AFA blokk állapotváltozói is újrainicializálódnak.

## F.3. Élesztési mechanizmus a felhasználói felület nélkül

Ebben a szakaszban – ízelítőül – röviden áttekintjük az eredeti rendszer (felhasználói felület nélküli) élesztési mechanizmusát és használatát.

Az élesztés előtt a következő paramétereket, konstansokat kell meghatározni, és azokat a forrásfájlokban – minden szükséges helyen – következetesen módosítani. Bizonyos paramétereket más paraméterek segítségével kell származtatni.

`MOTE_NUM`: A hálózatban található szenzor mote-ok száma

`OUT_NUM`: A kimenetek, azaz a rendszerben használt zajelnyomó hangszórók száma

`ID_LENGTH`: Az identifikáció során a frekvenciafelbontás: $f_{felb} = \frac{f_{s,DSP}}{2 \cdot ID\_LENGTH}$, ahol $f_{s,DSP}$ a DSP decimált mintavételi frekvenciája, amely 2 kHz.

`ID_TOP_VAL`: Az identifikált frekvenciák száma. A mért legnagyobb frekvencia: $f_{top} = f_{felb} \cdot ID\_TOP\_VAL$.

`NU`: A konvergencia-paraméter értéke.

`IDENT_FROM_FREQ`: Az identifikáció lényegi kezdeti frekvenciája. Amíg nulláról ezt az értéket nem éri el a rendszer, egy frekvencián nagyon rövid időt tölt.

`ID_END_SIG`: Egy adott hangszóró esetén az `ID_END_SIG`-edik frekvencialépés után kigyullad a DSP kártyán található LED5 nevű LED.

`IDENT_DUR`: Az identifikáció során egy frekvencián töltött idő hossza másodpercben.

`IDENT_DUR_LONG`: Az első identifikált frekvencián töltött idő hossza másodpercben.

`FILTER_FROM`: A bejövő adatok szűrése `FILTER_FROM` s idő elteltével kezdődik meg. A funkció célja, hogy a bejövő adatokat akkortól kezdjük el szűrni, miután azok beálltak az állandósult értékük körül. Ez gyorsabb identifikációt tesz lehetővé, ugyanis a jel/zaj viszony javításához szükséges, de a beállást lassító szűrés jobb kezdeti értékről indul.

`REZ_IN_FILT`: Egyszerű adattovábbító hálózat esetén használt paraméter. Megadja, hogy a `FILTER_FROM` idő leteltével milyen mértékű legyen a Fourier-analizátorok átlagolása az identifikációban. Megadja a megfigyelő hibajelének leosztását (a stabilitáshoz szükséges $1/N$-es leosztáson felül, ahol $N$ a rezonátorok száma).

`NOISE_FREKI_MIN`: A zaj minimális frekvenciája Hz-ben. Ha a referenciajel ez alá megy, a zajcsökkentő rendszer kikapcsol, és az állapotváltozók nullázódnak.

`NOISE_FREKI_MAX`: A zaj maximális frekvenciája Hz-ben. Ha a referenciajel efölé megy, a zajcsökkentő rendszer kikapcsol, és az állapotváltozók nullázódnak.

`A_FIL`: Rezonátoros adattovábbító hálózat esetén használt paraméter. Megadja, hogy a `FILTER_FROM` idő leteltével milyen mértékű legyen a Fourier-együtthatók átlagolása. A bejövő együtthatókon exponenciális átlagolást alkalmazunk.

A zajcsökkentő rendszer manuális élesztési folyamata a következő:

1. A VisualDSP++ fejlesztői környezet elindítása

2. A megfelelő session, azaz munkamenet kiválasztása – a session-ök a működési módot és a hardver típusát határozzák meg. Az adott rendszer esetében az `ADSP-21364 ADSP-21xxx EZ-KIT Lite` munkamenetre van szükség. A session listából történő kiválasztására, vagy hozzáadására a `Session` menü parancsai nyújtanak lehetőséget.

3. A projekt megnyitása – az alkalmazásokat `.dpj` kiterjesztésű projektfájlok fogják össze. A projektek megnyitása a `File` → `Open` → `Project` paranccsal kezdeményezhető. A megnyitzott projekthez rendelt forrás-, fejléc-, és linker-állományok listája a `Project` ablakban jelenik meg.

4. Az identifikáció vagy zajcsökkentés feladattól függően, a megnyitott projekthez a feladatnak megfelelő forrásfájlt hozzá kell adni, míg az oda nem illőt el kell távolítani. Identifikáció esetén a `ProcessIdent.c`, zajcsökkentés esetén pedig a `ProcessANC.c` állományra van szükség. Az állomány hozzáadása a `Project` → `Add to Project` → `File(s)` paranccsal történik, míg az eltávolítás az adott fájlra előhívható helyi menüben szereplő `Remove File from Project` parancs használatával.

5. Ha a projekt már megfelelő állományokat tartalmaz a `Project` → `Build Project` paranccsal, vagy az `F7` billentyű lenyomásával a projekt lefordítható. A fejlesztőkörnyezet a fordítás után létrejövő `.dxe` állományt azonnal felprogramozza a DSP-be.

6. A program futása a `Debug` → `Run` paranccsal, vagy az `F5` billentyűvel indítható, és a `Debug` → `Halt` paranccsal, vagy az `Shift+F5` billentyűkombinációval állítható le.

7. Elsőként az identifikációt kell végrehajtani, tehát az ennek megfelelő állományt a projekthez kell adni, a projektet lefordítani, a kódot letölteni, majd a program futását elindítani, majd az identifikáció végeztével megállítani az előzőekben leírt módon.

8. Az identifikáció befejeztével a DSP memóriájában létrejött átviteli függvények mért adatait fájlba kell menteni a `Memory` → `Dump` paranccsal, a következő beállításokkal a függvény valós része esetén:

   **Address:** `atvitel_re`

   **Memory:** Data(DM) Memory

   **Format:** Floating Point 32 bit

   **Count:** `ID_TOP_VAL` · `IN_NUM` · `OUT_NUM`

   **Stride:** 1

   **Dump to a file:** OK

   **Show Address:** NOT

   **File name:** /atvitel/atvitel_re.dat

   **Write format to file:** NOT.

A függvény képzetes részére pedig:

**Address:** `atvitel_im`

**Memory:** Data(DM) Memory

**Format:** Floating Point 32 bit

**Count:** `ID_TOP_VAL` · `IN_NUM` · `OUT_NUM`

**Stride:** 1

**Dump to a file:** OK

**Show Address:** NOT

**File name:** /atvitel/atvitel_im.dat

**Write format to file:** NOT.

9. Az ilyen módon elmentett átviteli függvények feldolgozása az `atvKarInv.m` MATLAB script futtatásával történik, amelyben a következő beállításokat kell megadni:

   **hangszoroszam:** `OUT_NUM`

   **mikrofonszam:** `MOTE_NUM`

   **adatSzam:** `ID_TOP_VAL`

   **fsVirt:** `ID_TOP_VAL` · `FS / ID_LENGTH`

   A script futása során kirajzolja az átviteli függvényeket, és előállítja a zajcsökkentő program számára az inverz átvitelt tartalmazó állományokat.

   Ha a kirajzolt amplitúdókarakterisztikák teteje „lapos”, azaz sok frekvencián azonos értéket vesz fel, a mote-ok mikrofonja telítésbe ment, az identifikációt kisebb hangerőn meg kell ismételni.

10. Az adatkonverzió után van lehetőség a zajcsökkentés feladatának végrehajtására. Ilyenkor az identifikációt végző forrásfájlt el kell távolítani, a megfelelő zajcsökkentő állományt pedig a projekthez kell adni, a projektet le kell fordítani, a kódot letölteni, majd a program futását elindítani.

    A munka során ügyelni kell arra, hogy mind az identifikációt, mind a zajcsökkentést végző programban, mind a MATLAB scriptben a fent ismertetett konstansok, paraméterek megegyezzenek. Inkonzisztens paraméterezés esetén valamely fázisban hibaüzenetet kapunk.

A rendszer működtetése során szükség lehet egyes paraméterek futás közbeni módosítására is. Ez a gyakorlatban legtöbbször a `NU` átcsatoló együttható értékének változtatását jelenti. A memóriatartalom megváltoztatása a következőképpen végezhető el:

1. A `Memory` → `Three Column` parancs hatására megnyíló ablakban ki kell választani a változó nevét (`NU`)

2. A számsorra előhívott helyi menü segítségével többféle megjelenítési formátum közül lehet választani

3. A programot meg kell állítani a `Debug` → `Halt` paranccsal, vagy az `Shift+F5` billentyűkombinációval

4. Az előzőleg megjelenített memóriaterület értékére duplán kattintva az átírható

5. A program futása a `Debug` → `Run` paranccsal, vagy az `F5` billentyűvel újra elindítható.