Aktív zajcsökkentő rendszerek megvalósítása szenzorhálózattal

Önálló labor zárójegyzőkönyv

Lajkó László és Orosz György
V. évf. vill. szakos hallgatók

2005.

Konzulens:
dr. Sujbert László egyetemi docens
Méréstechnika és Információs Rendszerek Tanszék

Feladatkiírások:

Lajkó László:

Cím: **Aktív zajcsökkentés on-line identifikációval**

Az aktív zajcsökkentő rendszerek működésének kulcsa a hangsugárzók és a hibamikrofonok közötti akusztikus átviteli függvény pontos ismerete. Ennek érdekében a zajcsökkentés bekapcsolása előtt ezt az átviteli függvényt identifikálni kell. Ha a mérés túl pontatlan, a rendszer labilis lesz, és a zaj csökkentése helyett erősíteni fog. Mivel a mérés a zajcsökkentés bekapcsolása előtt történik, ez off-line identifikációnak nevezhető. Ha a mérés jó volt, a zajcsökkentő rendszer működik, de működés közben az akusztikus átviteli függvény változhat, pl. a hőmérséklet változásának hatására, vagy mert kinyitnak egy ajtót. Ilyen esetekben a rendszer labilissá válhat. Az on-line identifikáció célja a rendszer stabilizálása ilyen kritikus esetekben is. A feladat során alaposan meg kell ismerni a zajcsökkentő rendszereket, valamint ki kell próbálni ismert és új on-line identifikációs eljárásokat. A feladat megoldása elméleti felkészültséget és kísérletező kedvet igényel.

Konzulens: Sujbert László.

Kiírás éve 2005

Orosz György:

Cím: **Aktív zajcsökkentő rendszerek demonstrációja**

Az aktív zajcsökkentés területén eddig született eredmények és működő programok birtokában célszerűnek látszik egy, a feladatot és lehetséges megoldásait demonstráló rendszer létrehozása. A feladat megoldása során a hallgatók megismerik ezeket a rendszereket, és egységes keretben megvalósítják azokat. A feladat kidolgozása során kisebb hardver építési munkákra is sor kerül. A feladatra elsősorban villamosmérnök szakos hallgatók jelentkezését várjuk, de informatika szakosok is jelentkezhetnek.

Konzulens: Sujbert László.

Kiírás éve 2005

Önálló laboratórium tárgy két féléves zárójegyzőkönyveként szeretnénk benyújtani a 2005. évi Tudományos Diákköri Konferenciára írt jegyzőkönyvünket.

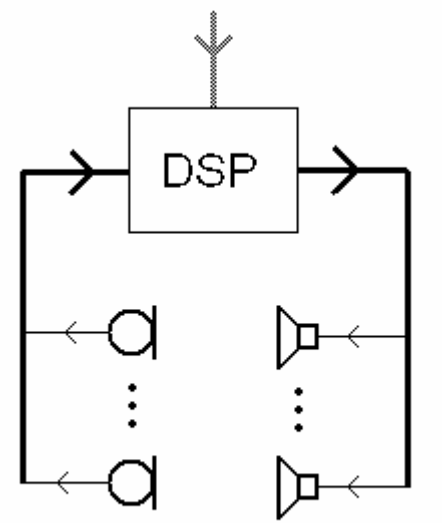
1. Bevezetés

1.1. Aktív zajcsökkentés helye és jelentősége

A zajcsökkentést hagyományosan úgy végzik, hogy a zajforrást és a kibocsátott zajtól megóvni kívánt területet fizikailag elválasztják egymástól különböző elnyelő anyagokkal. Ez történhet a zajforrás, vagy a megvédeni kívánt területet teljes vagy részleges elkerítésével. Ezeket passzív zajcsökkentő eljárásoknak nevezzük. Ezekkel az eljárásokkal nagy csillapítást érhetünk el, hátrányuk azonban, hogy meglehetősen költségigényesek, mivel használatukhoz speciális anyagokat kell alkalmazni, és az ilyen rendszerek telepítése (elrendezés megtervezése, az elnyelők megfelelő elhelyezése, rögzítése) igen nagy költségekkel jár. A passzív hangelnyelők másik hátránya, hogy sokszor elég nagy méretűek, amiből az következik, hogy szükség esetén elég nehezen mozgathatók, nem mobil berendezések. Ezen kívül ezek a rendszerek nem működnek jól alacsony frekvenciatartományban. Ennek az az oka, hogy például egy 100 Hz-es hang esetén a hullámhossz körülbelül 3,4 m (a levegőben, normál terjedési körülmények között). Ez a hullámhossz pedig igen nagy összemérve egy normál hangelnyelő felület vastagságával. Összefoglalva a passzív zajelnyelők hátránya, hogy igen költségesek (speciális anyag és telepítésigényes), nagy méretük miatt nehezen, vagy egyáltalán nem mozgathatók, és nem működnek jól az alacsony frekvenciákon.

Ahol a passzív rendszerek már nem működnek olyan jól, ott lép színre az aktív zajcsökkentés (azaz a 0-tól 1-2 kHz-ig terjedő tartományban). Az aktív zajcsökkentő rendszerek (ANC: Acive Noise Control) tulajdonképpen elektromechanikus vagy elektro-akusztikus rendszerek, amelyek a szuperpozíció elvén működnek. Ez azt jelenti, hogy a zavaró jelet (ami általában hallható zaj, de általánosabb értelemben lehet bármilyen rezgés) úgy szüntetik meg, hogy a zavarhatással egyező amplitúdójú, de ellentétes fázisú jelet juttatnak a fizikai rendszerbe (ahol a zavar fellép). Az általánosítást az indokolja, hogy például egy repülő szárnyának berezonálástól való mentesítése vagy egy szoba több pontjának zajmentesítése jelfeldolgozási szempontból nagyon hasonló feladat.

Egy aktív zajcsökkentő rendszer általános felépítését az 1.1. ábra mutatja.



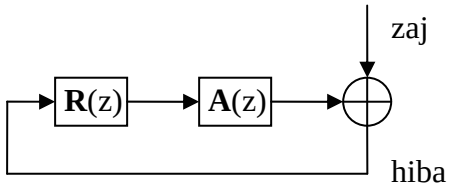
1.1. ábra. Egy aktív zajcsökkentő rendszer általános felépítése

A zajforrásból érkező jelet az úgynevezett beavatkozó hangszórókból kiadott „inverz” zaj segítségével nyomjuk el, oly módon, hogy a térben a hanghullámok interferenciájának eredményeként az úgynevezett hibamikrofonok környezetében csöndes zónák alakuljanak ki. Ezen csöndes zónák a mikrofonok *r* sugarú környezetében alakulnak ki, ahol *r* körülbelül az elnyomandó jel hullámhosszának a negyede. Ez tulajdonképpen meg is szabja az ANC rendszerek korlátját, mivel nagyobb frekvenciákon a hullámhossz egyre kisebb, ami azt eredményezi, hogy a kialakítható csöndes zónák mérete is jelentősen lecsökken.

A beavatkozó forrásokból kiadandó jelet a mikrofonok jeleinek és a zajforrásból például mikrofonnal vett úgynevezett referenciajel segítségével, bonyolult jelfeldolgozási algoritmussal határozhatjuk meg.

A feladat bonyolultsága miatt ANC rendszerek megvalósításához hagyományos mikrokontrollerek helyett jelfeldolgozó processzorokat alkalmaznak. Az ábrán ezért jelöltük úgy, hogy a mikrofonok és a beavatkozó hangszórók jelei között a digitális jelfeldolgozó processzor teremti meg a kapcsolatot. Ennek köszönhetően az aktív zajcsökkentő rendszerek diszkrét rendszerek, melyek az analóg fizikai világhoz AD- és DA-átalakítókon keresztül kapcsolódnak. Az egész rendszer vizsgálatához az analóg fizikai rendszert (átviteli közeg) is diszkrét modellel írjuk le. Ez a modellezés nem hibamentes, ezek a hibák azonban a gyakorlatban nem zavróak. Feltételezhető továbbá, hogy a fizikai rendszer átviteli tulajdonságai időben nagyon lassan változnak. A változás bekövetkezhet különböző okok miatt: például emelkedik a hőmérséklet, változik a páratartalom, egy szoba esetében, ha kinyitják az ajtót. A fizikai rendszerek

gyakran tartalmaznak nemlinearitásokat, melyeknek a kezelése nagyon nehéz. Szerencsére az akusztikus rendszerek igen széles tartományban lineárisnak tekinthetők. az aktív zajcsökkentő rendszereket tehát lineáris invariáns diszkrét idejű rendszereknek tekinthetjük. Ennek megfelelően a zajelnyomó rendszer modelljét mutatja az 1.2. ábra.



1.2. ábra. Aktív zajelnyomó rendszer blokkvázlata

Az ábrán **R**(*z*) maga a zajcsökkentő struktúra, **A**(*z*) pedig a fizikai rendszer átviteli függvénye abban az esetben, ha egy mikrofon és egy hangszóró van. Más esetekben, azaz, ha több mikrofon és több hangszóró van, akkor **A**(*z*) egy átviteli függvény mátrix.

A zajcsökkentő eljárás célja tehát, hogy a hibajelet minimalizálja. A következő fejezetekben a tárgyalás leegyszerűsítése érdekében egycsatornás rendszereket (egy mikrofon és egy hangszóró) vizsgálunk, hiszen ebben az esetben **A**(*z*) csak egy diszkrét idejű átviteli függvény, a zajcsökkentő struktúra kimenő jele egy skalár, ugyanígy a hibajel is. Természetesen az eredmények többcsatornás esetre is általánosíthatóak. Ezeket az eseteket is megvizsgáljuk.

1.2. Szenzorhálózatok általános bemutatása

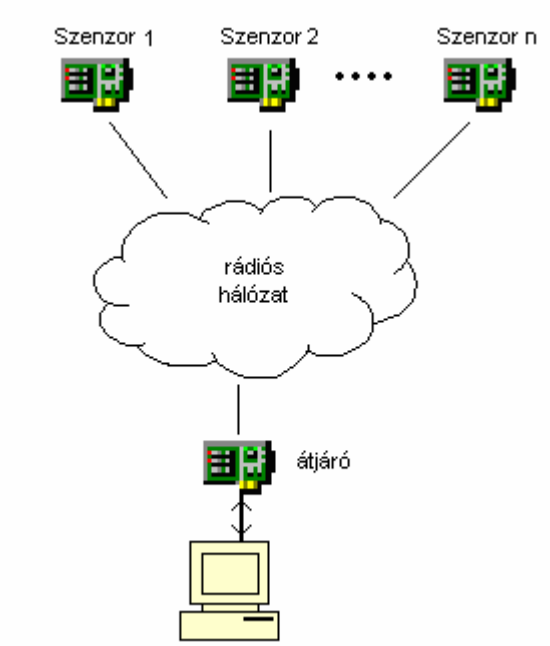
Az aktív zajcsökkentő rendszerek kialakításakor az egyik megoldandó feladat a mikrofonok és hangszórók kiválasztása, valamint meg kell oldani a hibamikrofonok jeleinek, valamint a beavatkozó jelek átvitelét. Hagyományosan ezeket a jeleket vezetéken továbbítjuk.

Manapság azonban a technológia fejlődése lehetővé tette az olcsó rádiós digitális adattovábbító berendezések kialakítását, melyeket különféle szenzorokkal és vezérlő logikával ellátva mérésadatgyűjtő és továbbító hálózatokat, elterjedt nevükön szenzorhálózatokat alakíthatunk ki. Ezen hálózati elemeket akár készen is megvásárolhatjuk, és megfelelő szenzorok illesztésével, melyeket a gyártók szintén készen kínálnak, viszonylag egyszerűen megoldható saját hálózat kialakítása.

A szenzorhálózatok elterjedésében nagy szerepet játszott könnyű kezelhetőségük, mobilitásuk. A hagyományos rendszerekkel ellentétben

telepítésük nem igényel meglévő hálózati infrastruktúrát, amely jelentős költségmegtakarítást jelent. Fokozottan igaz ez például azokra az esetekre, melyekben a hálózatot csak ideiglenesen, egyszeri mérés elvégzésére alakítjuk ki.

A szenzorhálózatok általános felépítése az 1.3. ábrán látható:



1.3. ábra. Szenzorhálózat általános felépítése

Látható, hogy az adatok begyűjtését végző szenzorok a mérési eredményeiket, esetleges alapvető adatfeldolgozást követően, továbbítják a hálózaton keresztül egy átjáróként szolgáló szenzorhoz, mely a rádiós hálózat adatait továbbítja az adatokat feldolgozó berendezéshez.

A szenzorhálózatokat manapság már az élet számos területén alkalmazzák. Néhány felhasználási példát itt is megemlítünk. Gyakori felhasználási terület a különböző környezeti és meteorológiai jellemzők, mint például hőmérséklet, fényerősség, légnyomás vagy páratartalom mérése. Találkozhatunk különböző rezgésanalízisban történő alkalmazással, például szeizmológiai megfigyelés, különböző szerkezetek, mint például épületek, hidak rezgéseinek megfigyelése. Segítséget jelenthetnek ezentúl egészségügyi, épületinformatikai és még számos alkalmazás fejlesztésében.

A fent említett felhasználások jellemzői általában, hogy az átvitt adatok mennyisége csekély a hálózat áteresztőképességéhez képest, illetve nagy mennyiségű adat továbbítása esetén az adatok átvitelére fordítandó idő nem kritikus.

Előfordulhatnak azonban olyan esetek, amikor nagy mennyiségű adatot határidőn belül továbbítani kell a hálózaton keresztül, aminek biztosítása igen nehézkes a rádiós kapcsolat bizonytalan volta miatt. Tipikusan ilyenek a

szabályozástechnikai alkalmazások. A fenti okok miatt a szenzorhálózatok felhasználása a szabályozási feladatokban még nem elterjedt. Általában a meglévő alkalmazásokban sem közvetlenül a folyamat kimenő jelének érzékeléséhez használják, hanem bizonyos környezeti paraméterek mérésével segítik a szabályzó optimális beállítását. Ha mégis kimenőjel érzékelőként használják a hálózat elemeit, akkor is lassú folyamatok szabályozásában, ahol nem játszik akkora szerepet az adatátvitel időbeni bizonytalansága.

A fent említett problémák miatt érdekes feladatnak bizonyult egy viszonylag nagy sebességgel működő aktív zajcsökkentő rendszer megvalósítása, mely bizonyos szemszögből szabályozástechnikai feladatnak is tekinthető, és igen érzékeny a visszacsatolás minőségére. Mivel az aktív zajcsökkentő rendszerek felhasználási tartománya néhány kHz-ig terjed, ezért a hálózat sebességét figyelembe véve még reménytelinek látszik a rendszer megvalósítása, bár nyilvánvalóan bizonyos kompromisszumok megkötése szükséges.

A rendszer kialakítása során a szenzorhálózattal kapcsolatban meg kellett oldani adatátvitellel és időzítéssel kapcsolatos problémákat valamint szinkronizációs feladatokat is. A zajcsökkentő algoritmust implementálni kellett egy jelfeldolgozó processzoron, és megoldást kellett kínálni az on-line identifikáció megvalósítására. Természetesen megoldandó feladat magában foglalja a rádiós hálózat átjáró elemének, és a jelfeldolgozó processzornak a fizikai összekapcsolását, valamint az összeköttetésen keresztül az adatok megfelelő továbbítását.

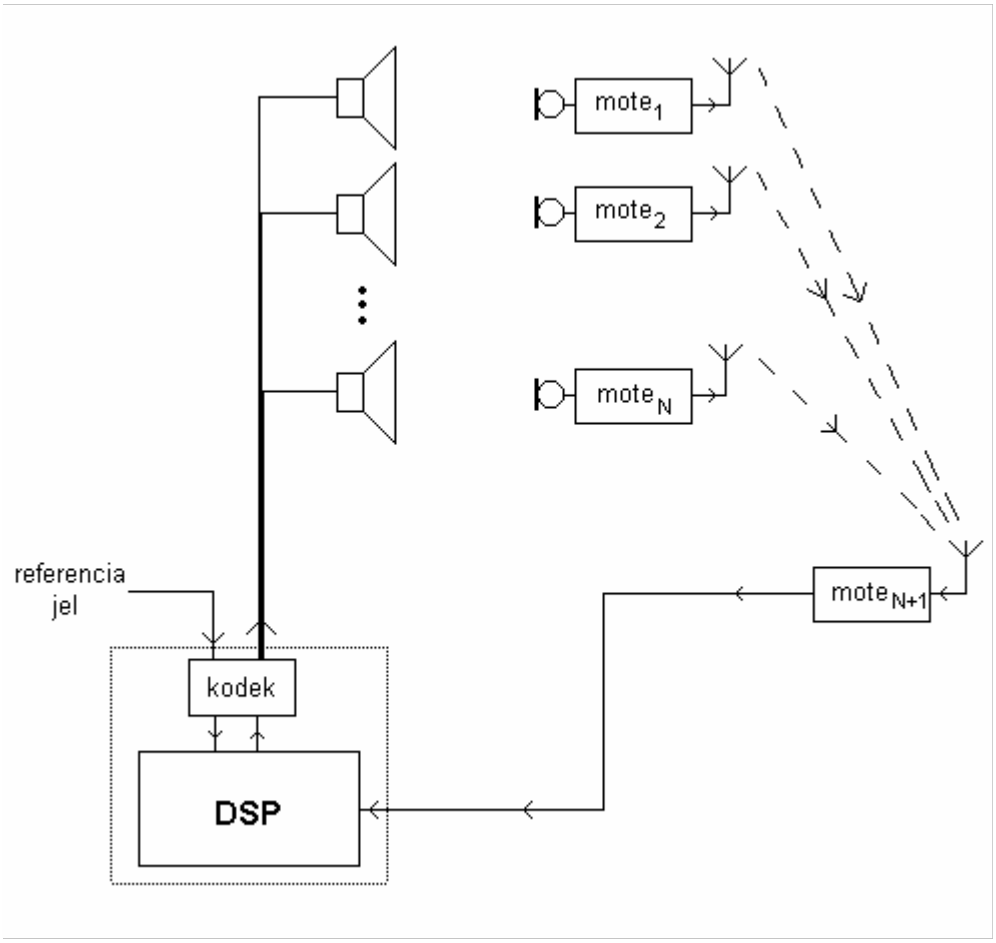
2. Specifikáció és rendszerterv

A feladatunk egy aktív zajcsökkentő rendszer megvalósítása szenzorhálózattal. A rendszernek a 0-tól 1 kHz-ig terjedő frekvenciatartományban kell képesnek lennie elnyomásra. A zajcsökkenés mértéke minimum 20-30 dB kell, hogy legyen. A fizikai környezet változásai a rendszer működését, azaz stabilitását és elnyomási tulajdonságait nem befolyásolhatják.

A rendszert egy Analog Devices 21061-es EZ-KIT Lite kártyán valósítottuk meg. Ez a processzor ugyanis lebegő pontos aritmetikai egységei miatt kiváló az akusztikai feladatok megoldására, ugyanis ezen területen igen nagy dinamikatartomány szükséges. A beavatkozó jeleket vezetéken keresztül továbbítjuk a hangszórók felé, a hibajeleket pedig Berkeley mote-ok segítségével érzékeljük és továbbítjuk a jelfeldolgozó kártyához. A választás azért esett a Berkeley-mote-okra, mivel ezek rendelkezésre állnak a tanszéken, és rendelkeznek a feladathoz szükséges mikrofont is tartalmazó szenzorkártyával. A mote-ok szinkronizálása, valamint jeleiknek a DSP kártyára továbbítása még nem megoldott, ezért ez is a feladat része. A specifikáció magában foglalja az összes nem rendelkezésre álló hardware és software megtervezését és megvalósítását.

A 0-1 kHz-es működési tartományt az indokolja, hogy az aktív zajcsökkentő rendszerek esetén a hibamikrofonok körül az elnyomandó jel hullámhosszának a negyedének megfelelő sugárban alakulnak ki a csöndes zónák. A 20-30 dB-es minimum elnyomás azért szükséges, mivel ez az olyan mértékű zajcsökkenés, ami köznyelven fogalmazva elég jelentős hangszintcsökkenésnek nevezhető. Az a feltétel, mely szerint a rendszer működését külső környezet változásai nem befolyásolhatják, azért nagyon fontos, mivel az 1.2. ábrán $\mathbf{A}(z)$-vel jelölt átviteli függvény mátrix megváltozása nagyban befolyásolja a zajcsökkentő struktúra stabilitását (ezzel egy későbbi fejezetben foglalkozunk részletesen). Az átviteli függvény mátrix megváltozása nagyon könnyen előfordulhat. Befolyásolhatja a hőmérséklet vagy a páratartalom változása, szobában alkalmazott zajcsökkentés esetén a függöny mint elnyelő felület elhúzása. Ezen szokásos tényezőkön kívül számolnunk kell a mikrofonok kismértékű elmozdulásával is, ami szintén befolyásolja az átviteli függvény mátrix tulajdonságait. Az elnyomó struktúrának az ezen változások miatt bekövetkező stabilitási problémákat tehát ki kell tudnia küszöbölni.

A zajelnyomó rendszer tervezett struktúrája a 2.1. ábrán látható. A hálózatban lévő N db mote ($mote_1 \ldots mote_N$) a hálózat és DSP közötti átjáróként szolgáló mote ($mote_{N+1}$) felé továbbítja a mikrofonszenzoraik által szolgáltatott zajjelet, menyet a $mote_{N+1}$ soros porton továbbít a DSP felé. A DSP a hozzá csatlakozó kodeken keresztül beolvassa a zajjal megegyező frekvenciájú referenciajelet, és a mote-ok adatait felhasználva előállítja, valamint kiadja a hangszórókra a beavatkozójelet.



2.1. ábra. A zajcsökkentő rendszer tervezett felépítése

3. A feladatmegoldás hardver és szoftver eszközei [9]

3.1 Az ADSP-21061 EZ-KIT Lite fejlesztőkártya

Az ADSP-21061 EZ-KIT Lite az ADSP-21061 jelfeldolgozó processzor köré van építve. Ezek a kártyák mentesítik a vásárlót a tesztáramkör építése alól, ami például egy lebegőpontos DSP esetében négyrétegű nyomtatott áramkör tervezését jelenti. Ehelyett a vásárló egy olyan kártyát kap kézbe, amelynek segítségével az adott termék rögtön hozzáférhető és működés közben kipróbálható. Ez nagymértékben elősegíti az adott alkatrész gyors megismerését.

Az ADSP-21061 EZ-KIT Lite fejlesztői kártya elsődlegesen arra szolgál, hogy az ADSP-21061 processzor minél több funkciója elérhetővé váljon, egyszerűen hozzáférhető legyen. A kártyán négy LED és három nyomógomb található. A LED-eket a DSP flag lábai hajtják meg. A három nyomógomb közül egy a Reset lábra, egy a flag1 lábra, egy pedig az IRQ1 lábra van illesztve. Ez utóbbi megnyomásakor megszakításkérés történik, amely lehetőség nagyon jól kihasználható a különböző alkalmazások implementálásakor.

A kártyán található portok egyike a JTAG emulátor port, amelyen keresztül hibakeresésre (debugging) nyílik lehetőség. A JTAG port használatáról bővebben a 3.4.2.-es pontban térünk ki

A DSP külső memóriabuszára a logikai áramkörön kívül egy gyárilag beégetett EPROM-ot is illesztettek. A processzor ennek segítségével hajtja végre a bootolási szekvenciát hardver reset után.

Egy AD1847-es sztereó codec-et is elhelyeztek a kártyán, amelyhez analóg oldalról két sztereó jack dugóval lehet csatlakozni (egy kimenet, egy bemenet). Ez két bementi és két kimeneti csatornát jelent. Az AD1847-es codec a DSP nulladik soros portjához (SPORT0) van illesztve.

A fejlesztői kártyán a DSP azon lábaihoz is vannak kivezetések, amelyekhez nincs alkatrész illesztve. Ezek a lábak akkor válnak elérhetővé, ha tüskesorokat forrasztunk a megfelelő helyekre.

3.2 ADSP -21061

A fejlesztői kártya legfontosabb eleme a DSP, a jelfeldolgozó processzor. Az ADSP-21061 a SHARC DSP-család egyik tagja, amely az ADI egyik leginkább elterjedt és széles körben használt termékcsaládja. A SHARC betűszó a Super Harvard ARhitecture Computer rövidítése. Ez a felépítés a DSP-k körében klasszikusnak számító Harvard-architektúra továbbfejlesztett változata. A Hardvard-architektúra a személyi számítógépek processzorainak Neumann-

architektúrájával szemben megkülönböztet kód- illetve adatmemóriát. A két memóriaterület külön buszon érhető el, így lehetőség van párhuzamos használatukra. Ez a lehetőség műveletek ciklikus elvégzésekor nagymértékben gyorsítja a program futását.

A szuper Harvard-architektúra esetében már négy független busz van: két adatbusz, egy utasításbusz és egy IO busz. A SHARC processzorok egy órajelütemben egy utasítást hajtanak végre, ezt a tokon belüli megvalósítású utasítás-cache teszi lehetővé.

Az ADSP-21061 a SHARC család első tagjának, az ADSP-21000-nek egy továbbfejlesztett változata. A processzor mag (processor core) megegyezik a két típusnál, de a 21061-es néhány fontos kiegészítővel látták el. Ezek a következők: SRAM, IO processzor, IO busz. Ezen egységek mindegyike tokon belül van megvalósítva.

A 21061-es nem a legfejlettebb modell a ma kapható DSP-k között, de sebessége és számítási pontossága kielégítő a legtöbb átlagos jelfeldolgozási alkalmazás szempontjából. Az iparban széles körben használják ezt a DSP-t, mert viszonylag alacsony ára mellett az összes olyan követelményt teljesíti, ami egy lebegőpontos jelfeldolgozó processzortól elvárható. Mindazonáltal várható, hogy a közeli jövőben ki fogja szorítani a piacról az ADSP-21065L processzor, amely a SHARC család újabb és népszerű tagja. Ez a jövőbeli processzorváltás várhatóan nem okoz majd nagy problémát, hiszen a két DSP felfelé kódkompatibilis, azaz a 21061-esre írt programok hiba nélkül futnak a 21065L-en is.

A távolabbi jövőben várható, hogy a SHARC család tagjait fel fogják váltani az újabb DSP családok, például a ma már kapható, de még ritkaságszámba menő TigerSHARC család tagjai.

3.2.1. Teljesítményadatok

A CMOS megvalósítású ADSP-21061 processzor maximális órajelfrekvenciája 50 MHz. Mivel a DSP órajelütemenként egy utasítást hajt végre az utasítás cache segítségével, így az utasításvégrehajtási sebesség 50 MIPS (Million Instructions Per Second).

3.2.2. Felépítés

A processzor fő részei a következők:

- **Műveletvégző egység**

A műveletvégző egység kétféle lebegőpontos számábrázolási formátum használatát támogatja, a 32 bites IEEE single-precision floating-point szabványos formátumot, illetve a 40 bites kiterjesztett (extended precision) formátumot. Lehetőség van 32 bites fixpontos formátum használatára is. A műveletvégző

egység részei: ALU (Arithmetic and Logic Unit), szorzó egység, shifter egység. Ezek párhuzamosan vannak megvalósítva, tehát lehetőség van egyidejű használatukra a műveletvégzés gyorsítása érdekében.

Az egység számítási kapacitása általános esetben a fentebb említett 50 MIPS. Ez azonban egyes tipikus jelfeldolgozási számítások során a műveletek párhuzamos végrehajtás miatt megnövekszik. Az ilyen módon elérhető maximális számítási kapacitás 120 MIPS vagy 120 MFLOPS (Million Floatingpoint Operation Per Second). Ez utóbbi mérőszám is gyakran használatos, a processzor lebegőpontos számítási műveletekre vonatkoztatott számítási sebességet jellemzi. Mivel az ADSP-21061-es alapértelmezésben lebegőpontos számokon hajtja végre az utasításokat, esetében a két mérőszám megegyezik.

- **Adatregiszter blokk**

32 általános célú regiszter áll rendelkezésre a különböző adatmozgatási és számítási feladatokhoz. Ez a 32 regiszter két 16-os csoportra oszlik, az elsődleges (primary) és a másodlagos (alternate) regiszterblokkokra. A két blokk közül mindig csak az egyik aktív, a másik nem hozzáférhető. A blokkok közti váltás egy utasítással történik. Ezt a funkciót tipikusan a megszakítási rutinok használják ki, olyan módon, hogy az IT rutin csak a másodlagos regiszterblokkot használja, a főprogram pedig az elsődlegest. Ekkor elkerüljük azt, hogy a két rutinnak közös regisztereken kelljen osztoznia. Tekintve, hogy a megszakítási rutinok a főprogramot tetszőleges helyen szakíthatják meg, a közös regiszterhasználat előre nem tervezhető konfliktusokat okozna. Ez a probléma a másodlagos regiszterblokk hiányában csak nehézkesen és nagy overhead árán lenne kivédhető a regisztertartalom rendszeres memóriába mentésével.

- **Utasítás cache**

A Harvard-architektúra nyújtotta előnyök ezen egység segítségével használhatók ki maximálisan. Az utasítás cache-nek köszönhető, hogy utasítások ciklikus végrehajtása esetén három párhuzamos adatmozgatás és két számítási művelet hajtódik végre egy órajelcikluson belül. Ezek részletesebben: Az adatmemória és a kódmemória buszain egy-egy adatot olvasunk be regiszterekbe. (A kódterületen is lehet adatot tárolni.) A szorzó és az összeadó egyidejűleg hajt végre egy-egy műveletet regiszterekben tárolt adatokon. A következő végrehajtandó utasítás kódját pedig az utasítás cache szolgáltatja.

- **Adatcímző egységek**

Az adatcímző egységek egyike a kódterülethez, a másik az adatterülethez tartozik. Ezen egységek hardver szinten támogatják cirkuláris bufferek megvalósítását. Egy egyszerű memóriatömb azáltal lesz cirkuláris buffer, hogy a bufferen belüli címeket az adatcímző egység speciális módon számítja ki. Ez úgy történik, hogy a cirkuláris bufferhez egy mutató regisztert rendel hozzá, amely segítségével a címzés történik. A mutató regiszter a cirkuláris buffer báziscíméhez képesti eltolást tárolja, értéke maximálisan a buffer hossza lehet. Amennyiben a

mutató regiszter olyan értékre módosulna, amely nagyobb, mint a buffer hossza, akkor a regiszter tartalma ezen értéknek a buffer hosszával osztott törtrésze lesz. Ilyen módon egy körkörös vagy cirkuláris buffer jön létre, amely nagyon kedvező néhány tipikus jelfeldolgozási algoritmus (FFT, FIR szűrés) megvalósításakor.

- **SRAM memória**

A tokon belül (vagy on-chip) megvalósított 1 Mbites SRAM memória miatt nem kell külön külső memóriát biztosítani a processzor működéséhez. Természetesen amennyiben nem lenne elegendő ennyi memória, lehetőség van további külső memória illesztésére. Az SRAM memóriaterület a Harvard-architektúrának megfelelően két részre van osztva, adat- illetve kódterületre. Ez a hagyományos elrendezés, de nincs semmiféle megkötés abból a szempontból, hogy milyen típusú adatokat tárolunk az egyes területeken. Az optimális működési sebesség elérésének érdekében mindig az adott alkalmazás szempontjából mérlegelni kell, hogy milyen elrendezéssel érhető el a lehető legtöbb párhuzamos memóriaművelet. Például FIR szűrés esetén a bemeneti adatokat és a szűrőegyütthatókat külön memóriaterületen érdemes tárolni.

- **Külső memória és periféria interfész**

Külső memória illesztésére a külső buszon (external bus) keresztül van lehetőség. Erre a külső buszra a belső buszok multiplexálva csatlakoznak, tehát egy külső eszköz mind adat, mind kód memóriaterületre illeszthető. Ilyen módon a DSP-hez külső memórián kívül tetszőleges periféria illeszthető. Ezen az interfészen keresztül csatlakoztatható például FPGA áramkörhöz, vagy többprocesszoros működés esetén (multiprocessing) másik DSP-hez.

- **Host interfész**

A host interfész nyújt lehetőséget szabványos mikroprocesszoros buszokhoz való illesztésére. Ezen az interfészen keresztül a DSP egyszerűen csatlakoztatható személyi számítógéphez.

- **DMA vezérlő**

A DMA (Direct Memory Access) vezérlő nagy tömegő blokkos adatmozgatások esetén használatos. Ekkor a processzor magjának igénybevétele nélkül a DMA vezérlő nagy sebességű adatmozgatást végez a belső SRAM memória és valamely más eszköz között. Ezen eszközök a következők lehetnek: külső memória, külső periféria, host processzor, soros port. Ezen kívül DMA kapcsolat létesíthető a külső memória és más külső eszközök között is

- **Soros portok**

A DSP két szinkron soros porttal rendelkezik, amelynek segítségével különböző digitális perifériákhoz való csatlakozásra nyílik lehetőség. Ezen soros port sebessége széles tartományban mozoghat, maximálisan 40 Mbit/sec.

- **JTAG interfész**

Az IEEE JTAG 1149.1 szabványnak megfelelő port a DSP tesztelésére szolgál. Ezen port segítségével emulálásra van lehetőség, ami azt jelenti, hogy az integrált áramkörön belül vannak megvalósítva az emulátor funkciók. Tehát megfelelő eszköz vagy alkalmazás segítségével e porton keresztül a DSP működése nyomon követhető és tesztelhető. Ezt a portot használja a 3.4.2. alfejezetben ismertetett EZ-ICE Emulátor is.

3.2.3. Értékelés

Kijelenthető, hogy a processzor felépítése és különböző funkciói kielégítők abból a szempontból, hogy a megvalósítani kívánt jelfeldolgozási algoritmusok széles körének implementálását hardver szinten támogatja. Még a legmagasabb mintavételi frekvencia mellett is csatornánként /* kb. 830 utasítás */ hajtható végre, amely az overhead miatt egy kb. 800 együtthatós FIR szűrő megvalósítására elegendő. Ezért az átlagos alkalmazások esetén elegendően hosszú jelfeldolgozási rutinok futtatására van lehetőség, ami biztosítja a problémamentes működést.

3.3. A VisualDSP++ fejlesztői környezet

A DSP programok fejlesztése az EZ-KIT kártyával nagymértékben együttműködni képes VisualDSP++ fejlesztői környezet segítségével történt. A VisualDSP++ az Analog Devices cég szoftver terméke. Ez egy személyi számítógépen futó integrált fejlesztői környezet, amely azt a célt szolgálja, hogy az AD DSP-in futó programokat fejlesszék ki segítségével. Az alkalmazás Windows grafikus környezetben fut.

A fejlesztői környezet két független alkalmazásból áll, a Projekt szerkesztőből és a Debugger-ből.

3.3.1. Projekt szerkesztő

A Projekt szerkesztőben projekt fájlokat lehet megnyitni és szerkeszteni. Egy ilyen projekt tulajdonképpen egymáshoz rendelt forráskód fájlokat jelent, amelyekből az alkalmazás egy DSP-re letölthető fájlt generál.

A kimeneti fájl típusa beállítható a Projekt tulajdonságok ablakban. A lehetőségek: debuggolható verzió, kész verzió, vagy EPROM-ba égethető bit-file. A kimeneti fájl kiterjesztése ettől a beállítástól függ, A kész verzió kiterjesztése .exe, a debuggolható verzióé .dxe. A forráskód fájlok lehetnek gépi nyelven megírtak (kiterjesztésük: .asm), C nyelven megírtak (.c), vagy a linkelést leíró úgynevezett Linker Description File-ok. (.ldf).

A Projekt szerkesztőn belül a következő funkciók vannak megvalósítva:

- **C fordító**

E funkció biztosítja, a DSP-k C nyelvű programozását, amely sok esetben kényelmesebb a nehézkesnek tűnő gépi kóddal szemben. Használata mégsem igazán elterjedt, mert a C fordító nem tudja maximálisan kihasználni a DSP hardver adta lehetőségeket. A DSP-ket pedig éppen azért használják a különböző időkritikus jelfeldolgozó algoritmusok megvalósítására, mert hardver szinten támogatják a jelfeldolgozási műveletek végrehajtását. Ezeket a speciális hardver lehetőségeket viszont csak egy assembly nyelven megírt program tudja igazán kihasználni. Sok esetben használatos az az egészséges kompromisszumot jelentő megoldás, hogy a DSP programok sokszor lefutásra kerülő részleteit gépi nyelven írja meg a fejlesztő, míg az egyszer lefutó inicializáló jellegű programrészleteket C-ben. Így a futási idő túlnyomó részében futó rutinok assembly nyelven vannak megírva, tehát a program futása nem lassul le lényegesen.

- **Compiler**

A compiler végzi a gépi kódban megírt forrásfájlok valamint a C fordító által lefordított fájlok szintaktikai és szemantikai ellenőrzését.

- **Linker**

A linker felelős azért, hogy az egy projekten belüli különálló fájlokat egybefűzze, és előállítsa a kimenet .exe vagy .dxe fájlt. Minden projekthez tartozik egy Linker Description File (LDF) is, amely szintén a projekt részét képezi. Amennyiben a felhasználó nem ad meg ilyen LDF-et, akkor a Projekt szerkesztő alapértelmezésként egy gyári LDF-et használ.

A Linker Description File a linker számára tartalmaz lényeges információkat, tehát nem közvetlenül a DSP-n futó program része. Az LDF egy egyszerű, néhány utasításból álló nyelven megírt leírófájl, amely alapján a linker összefűzi a forrásfájlok kód és adatrészleteit. Ez a leírófájl tartalmazza azt a lényeges információt, hogy a különböző forrásfájlok adat- és kódrészleteit a DSP-n fizikailag melyik memóriaterületre kívánja letölteni a felhasználó.

- **Loader és Splitter**

Ez a két funkció akkor használatos, ha a kész DSP programot EPROM-ba kívánjuk égetni. Ezen alkalmazások segítségével állíthatók elő a szükséges bit fájlok.

- **Debugger**

A fejlesztői környezet másik nagy részegysége a Debugger alkalmazás. Ez az alkalmazás lehetőség biztosít a megírt DSP programok kipróbálására futás közben. Lehetőség van a program futásának részletes vizsgálatára, töréspontok helyezhetők el, lehet léptetve végrehajtani az utasításokat, a memóriaterületek, valamint a rendszer regisztereinek tartalma folyamatosan monitorozható. A

Debugger „bemeneti adata” a Projekt szerkesztő által előállított .dxe kiterjesztésű fájl. A DSP program futtatásának helye beállítható, ez lehet emulátor vagy szimulátor.

- **Szimulátor**

Ebben az esetben a program fizikailag nem töltődik le a DSP-re, hanem az alkalmazás a személyi számítógépen szimulálja a DSP működését. Ez a szimuláció minden részletre kiterjed, tehát a DSP majdani várható működése vizsgálható. Ilyen módon azonban nem lehet kipróbálni sem a fizikai DSP alapú rendszer többi egységét, illetve az egységek összehangolt működését sem. Ezért ha lehetőség van rá, akkor az emulátor használata előnyt élvez.

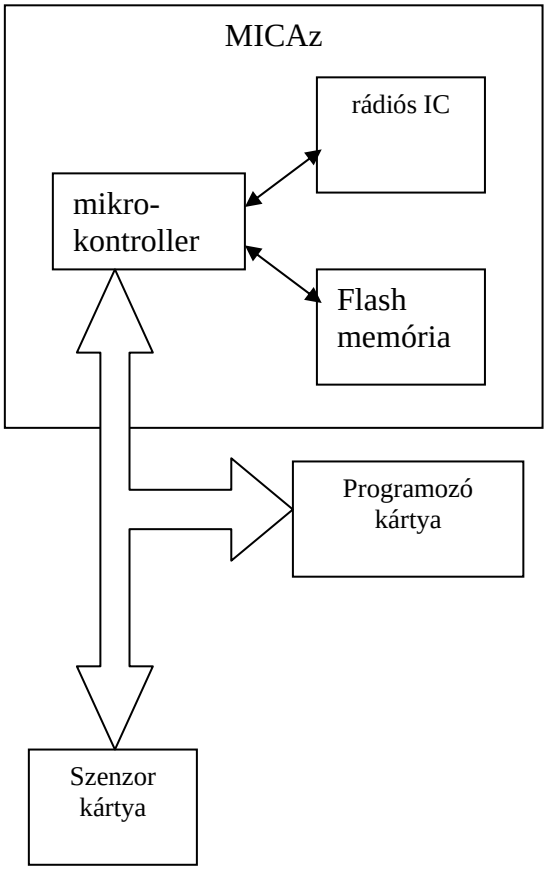
- **Emulátor**

Az emulátor használatára akkor van lehetőség, ha rendelkezésre áll az ADI által gyártott EZ-ICE emulátor kártya a személyi számítógépben. Ehhez tartozik egy csatlakozó, amelyet az ADI fejlesztői kártyáin található JTAG Emulátor csatlakozóval lehet összekötni. Erre a processzor belső (vagy on-chip) emulálását lehetővé tevő szabványos JTAG portjának lábai vannak kivezetve. Az emulátor ezen az interfészen keresztül vezérli a teljes vizsgálatot. Ugyanitt történik a Projekt szerkesztő által előállított program letöltése a DSP memóriájába. A letöltés után a processzort a Debugger Halt állapotban tartja. Ekkor lehetőség van a program futtatására, az utasítások léptetett elvégzésére vagy töréspontok elhelyezésére. A DSP összes regiszterének tartalma lekérdezhető és igény szerint módosítható. A memóriaterületek teljes területének tartalma fájlba írható (dump) vagy feltölthető (fill). A fenti műveletek elvégzésének mindegyikét a JTAG szabvány szerint a DSP hardveresen támogatja, ezért elvégzésükhöz nincs szükség külön monitorprogram letöltésére. A Debugger alkalmazás segítségével a program futásának teljes körű vizsgálata lehetővé válik.

Ha rendelkezésre áll az EZ-KIT Lite fejlesztői kártya, akkor emulálásra lehetőség van soros porton keresztül is, ugyanis a kártyán futó keretprogram – amit a boot szekvencia után az EPROM-ból tölt be – ezt lehetővé teszi. A lehetőségek ugyanazok, mint JTAG emulátor használata esetén, a különbség csak az, hogy az RS-232 kommunikációhoz szükséges kernel programnak a memóriában kell lennie.

3.4. A Berkeley mote-ok

Munkánk során a tanszéken is rendelkezésre álló Berkeley-moteokkal [5] dolgoztunk, melyekhez tartozik mikrofon szenzort is tartalmazó szenzorkártya; ennek típusa MTS310. A Berkeley-mote-ok tulajdonképpen egy családot jelölnek, melyeknek különböző tagjai vannak, mi a MICAz típust használjuk. Blokkvázlat szintjén a mote az 3.1. ábrán látható módon épül fel



3.1 ábra. A MICAz mote-ok felépítése

A következőkben a különböző részegységeket mutatjuk be.

- ***Flash memória***

A mote-on található egy AT45DB041B típusú memória, mely alkalmas a mérések során keletkezett adatok tárolására. Ennek köszönhetően 512 kB-nyi adatot lehet hosszútávon is tárolni, ha nincs lehetőség az adatok azonnali továbbítására. A mi általunk kifejlesztett alkalmazásban azonban a feladat jellege miatt ezt a flash memóriát nem tudjuk kihasználni.

- ***Rádiós IC***

A MICAz mote tartalmaz egy CC2420 típusú integrált adó-vevő áramkört, melynek segítségével lehetőség nyílik a mote-on futó alkalmazások számára az adatok vezeték nélküli továbbítására. A ZigBee szabványnak megfelelő rádió a 2.4 GHz-es ISM sávban működik, és adatátviteli képessége 250 kbps, melyet alkalmazásainkban természetesen nem lehet teljes mértékben kihasználni a kommunikációs overhead miatt. Ez a rádió az IEEE 802.15.4 szabvány szerint definiált fizikai és MAC réteget tartalmazó integrált áramkör. Ezt a szabványt

célirányosan a kisméretű helyi hálózatokkal szemben támasztott követelményeket szem előtt tartva alakították ki. Ilyen szempont például a nagy csomópontszámú hálózatok támogatása, amit a nagy címtartománnyal tesznek lehetővé. Fontos szempont a kis fogyasztású eszközök kialakításának támogatása, melyet a CC2420 például célirányos hardveres moduljaival is segít, mivel így nem szükséges a mikrokontroller erőforrásait bizonyos feladatokra feleslegesen használni.

Az IC kódolási és modulációs eljárásával is új irányvonalat követ. A rádiós adattovábbításban ugyanis ezekben az alkalmazásokban nagy problémát jelent a különböző tereptárgyak és reflexiók miatt kialakuló többutas terjedés, mely igen nagy csillapításokat eredményezhet bizonyos frekvenciákon. Ezen probléma megoldására spektrumkiterjesztést használnak. Ezzel az eljárással ugyan megnő a keskenysávú jel sávszélessége, így csökken a csatorna kihasználtsága, viszont a bizonyos frekvenciákon megjelenő csillapítások nem torzítják olyan mértékben a spektrumot, így az időtartománybeli jelet sem.

- ***Mikrokontroller***

A MICAz mote egy ATmega128 típusú mikrokontrollert tartalmaz. Ez egy általános célú 8 bites RISC architektúrájú mikrovezérlő. A következőkben felsorolásszinten bemutatjuk, hogy a mikrokontroller milyen fontosabb perifériakészlettel rendelkezik, és milyen szokásos paraméterekkel írható le:

- 128 kB program flash
- 4 kB SRAM
- 4 kB EEPROM
- szinkron és aszinkron soros interfész
- 4 db időzítő/számláló, melyeket PWM módban is használhatunk
- analóg komparátor
- 8 csatornás multiplexelt 10 bites AD-átalakító
- többféle energiatakarékos üzemmód
- legfeljebb 16 MHz-es órajel-frekvencia, az általunk használt mote órajel-frekvenciája 7.3728 MHz.

Már ezen adatokból is látható, hogy bonyolultabb jelfeldolgozási algoritmusok futtatására nem képes a mikrokontroller, a mote-on található perifériák kezelése és alapvető adatfeldolgozásra viszont kitűnően használható.

- ***Szenzorkártya***

A mote-okhoz, a rajtuk elhelyezett csatlakozón keresztül különféle szenzorkártyák csatlakoztathatók. Számunkra a legmegfelelőbb az MTS310-es típusú kártya, mivel ez tartalmazza a feladathoz szükséges mikrofont, valamint az ahhoz tartozó analóg jelkondicionáló áramköröket. A kártya lehetőséget kínál a mikrofonhoz tartozó erősítő erősítési tényezőjének bizonyos szintű állítására, mely egy a kártyán elhelyezkedő I^2C porton programozható ellenállás segítségével valósul meg. Ez a programozható értékű ellenállás egy műveleti

erősítő visszacsatoló ágában foglal helyet, így teszi lehetővé az erősítés beállítását.

- ***Programozói kártya***

Az általunk használt MIB510 típusú programozói kártya lehetővé teszi a mote-ok programozását a PC soros portja segítségével. A kártya üzemeltethető a mote-ok telepeiről, illetve lehetőség van tápegység csatlakoztatására is. Ez az eszköz azonban nem csak a programozásban játszik szerepet, ugyanis a rajta található áramkörök segítségével a mote-ok soros vonalán megjelenő logikai jelszinteket RS232-es jelszintekké alakítja, így lehetővé teszi más RS232-es soros interfészt használó eszközhöz való egyszerű csatlakoztatást. Ennek segítségével tehát egyszerűen kialakíthatunk olyan bázisállomásokat, melyek képesek továbbítani a rádiós hálózaton keresztül küldött adatokat az azokat feldolgozó egységhez.

3.5. A NesC nyelv és a TinyOS bemutatása

A témáról részletesebben a [3] és [4] helyeken olvashatunk, ebben a fejezetben csupán a feladat megoldásának megértéséhez szükséges információkat szeretnénk közölni.

3.5.1. Általános és szerkezeti leírás

A TinyOS kifejezetten a vezeték nélküli szenzorhálózatokhoz kifejlesztett operációs rendszer, mely nyílt forráskódú, ingyenesen letölthető és felhasználható. Alapvető kialakítási szempontjai az eseményvezérelt működés és a komponens alapú felépítés. Az operációs rendszert NesC nyelven fejlesztették ki, és ez is a hozzá kapcsolódó programozási nyelv, mely jól illeszkedik az operációs rendszer alapkoncepcióihoz.

A NesC alapvetően C szintaxisú programozási nyelv. A NesC-ben megírt programok komponensekből épülnek fel, ez egyszerűbbé teszi a program tervezését és tesztelését, hiszen a hierarchikus szemlélet megkönnyíti a rendszer áttekintését, az egymástól nagyjából függetlenül fejleszthető és tesztelhető komponensek pedig csökkentik a hibalehetőségeket.

A programjainkban felhasznált komponenseket interfészeken keresztül kötjük össze. Az egyes komponensek szolgáltathatnak, illetve felhasználhatnak interfészeket. Az interfészek parancsokat (command) és eseményeket (event) tartalmaznak. Az interfészt szolgáltató komponens az eseményeken keresztül értesíti az interfészt használó komponenst a megfelelő esemény bekövetkeztéről. A parancsokon keresztül pedig felhasználhatjuk az interfészt szolgáltató komponens szolgáltatásait. Az interfészeket a saját komponenseinkben új névvel láthatjuk el. Ez egyrészt megkönnyíti a program módosítását, másrészt lehetővé teszi azt, hogy azonos interfészeket szolgáltató komponenseket használhassunk.

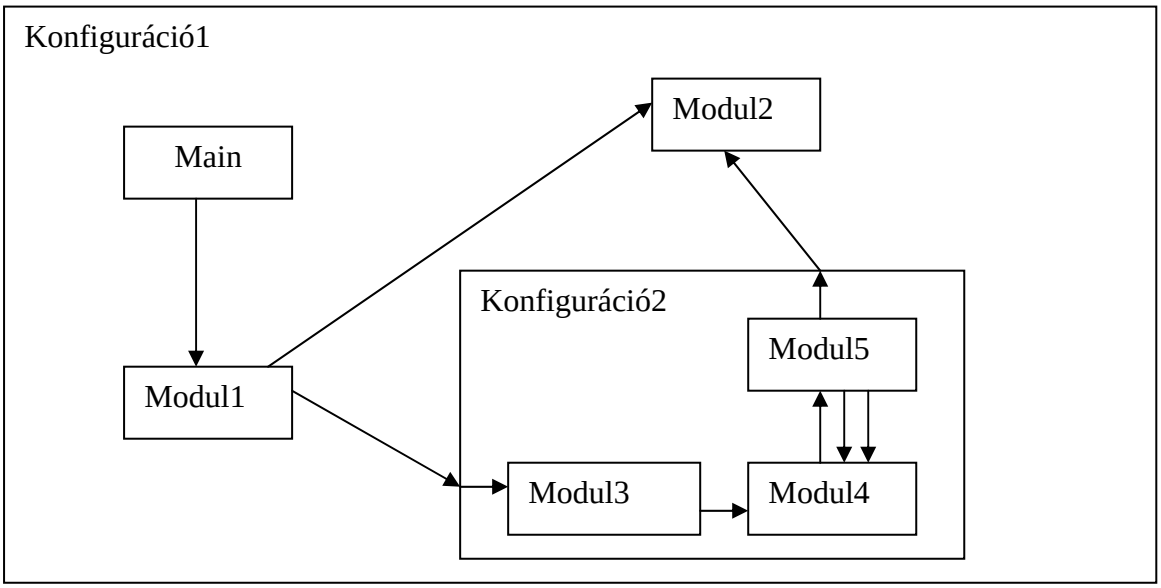
Ez például akkor következhet be, ha többféle, különböző szenzorhoz rendelt komponenst is használunk, melyek mindegyikének van a mintavételezést kezelő `ADC` interfésze. Az interfészek másik érdekes tulajdonsága a paraméterezhetőség, mely azt jelenti, hogy egy adott interfészből több azonos példány is készülhet. Ennek az a jelentősége, hogy az olyan perifériákhoz, amelyekből fizikailag csupán egy, vagy nem elegendő példány létezik (például időzítő vagy rádió), többszörös hozzáférést biztosítsunk szoftveres eszközökkel úgy, hogy a felhasználóknak ne kelljen kezelni az ezzel járó nehézségeket, és virtuálisan egy saját, kizárólagosan birtokolt interfészt lássanak.

A NesC nyelv két különböző típusú komponens kialakítását teszi lehetővé, ezek a modulok és a konfigurációk.

A modulokban a modulok által szolgáltatott interfészek parancsait valósítjuk meg, valamint a modul által használt interfészek eseményeit kezeljük le.

A konfigurációkban más konfigurációkat, illetve modulokat kötünk össze az azonos típusú interfészeken keresztül. Alapvetően kétféle konfiguráció létezik: top-level konfiguráció, valamint az újrafelhasználható komponensek. A top-level konfiguráció tulajdonképpen az elkészült programot jelöli. Ezen komponenssel kapcsolatos követelmény, hogy tartalmazza a Main modult, mely az `StdControl` interfészen keresztül végzi a programunk inicializálását, indítását valamint leállítását. Az újrafelhasználható konfigurációk szintén komponensek összekötésére szolgál, de oly módon, hogy a benne szereplő komponensek bizonyos interfészeit kivezetjük, és kívülről hozzáférhetővé tesszük más komponensek számára. Az eddig leírtak világossá tétele érdekében figyeljük meg az 3.2. ábrát, melyen egy egyszerű alkalmazás, a Konfiguráció1 felépítése látható. A Main modul a Modul1 `StdControl` interfészével van összekötve. A Konfiguráció2 három modult tartalmaz, és a Modul5 valamint Modul3 komponensek interfészei vannak kivezetve, melyeket a Modul2 szolgáltat, és a Modul1 használ. Az ábrán a nyilak az interfészt szolgáltató modul felé mutatnak.

A komponensekkel kapcsolatban is megemlítjük az interfészek esetén már ismertetett lehetőséget, miszerint a komponenseket a saját programunkban tetszőleges névvel láthatjuk el, és a későbbiekben így hivatkozhatunk rájuk.



3.2. ábra. Egy NesC-ben készült alkalmazás komponens diagrammja

A NesC nyelv által kínált lehetőség a taszkok használata. Ezen taszkokat a program bármely részéből indíthatjuk. A taszkok elindításuk kezdeményezésekor egy FIFO tárba kerülnek, amelyben bekerülésük sorrendjében végrehajtódnak. Egy új taszk indítása csak az előző taszk befejezésekor kezdődhet.

A programfejlesztés során fontos, hogy az adott környezet hogyan ütemezi a különböző típusú folyamatokat. A TinyOS operációs rendszerben a következő ütemezési szabályok vannak:

- A taszk nem szakíthat meg más folyamatot
- A hardver IT-t kezelő esemény bármilyen folyamatot megszakíthat

A taszkokat, fenti tulajdonságokból következően, a kevésbé időkritikus programrészek végrehajtásához használjuk, ugyanis végrehajtásuk késleltetéssel kezdődhet, így lefutásának időpontja nem biztosított.

Mint általában az eseményvezérelt rendszerekben, a TinyOS-ben is meg kellett oldani a kölcsönös kizárást. Ez kétféle képen történhet:

- Az atomic kulcsszó által védett területeken nem megszakítható a program, így nem használhatja más esemény az általunk használt közös erőforrást
- Ha a közös erőforrást csak taszkokban használjuk, akkor egyszerűen megoldható a kölcsönös kizárás, mivel a taszkok nem szakíthatják meg egymást.

A TinyOS segítségével lehetővé válik a mote különböző perifériáinak egyszerű kezelése, mivel a perifériát annak részletesebb ismerete nélkül használhatjuk. Az egyes perifériákat általában egy-egy komponens szimbolizálja,

melynek interfészein keresztül érhetjük el az eszközt. Ilyen komponensek interfészein keresztül érhetjük el például az időzítőt, a rádiót, a soros portot és a szenzorokat. Ezen modulok azonban nem minden esetben használhatók jól, főleg ha ki szeretnénk használni az eszközök maximális teljesítményét. Ekkor kénytelenek vagyunk a saját feladatunknak megfelelő komponenst készíteni, vagy a már meglévőket módosítani. Erre a munkánk során többször is sor került.

Ugyan nem képezi a TinyOS részét, azonban igen szoros kapcsolatban áll vele a PC-re kifejlesztett Java-s illetve Matlabos környezet, melyek segítségével egyszerűen kommuikálhatunk a mote-okon futó operációs rendszerrel, ha azok valamilyen módon csatlakoztatva vannak a PC-hez. Habár ezen fejlesztői környezetek használata közvetlenül nem képzi a feladat részét, a fejlesztés során nagyon hasznosak, ugyanis ezek igen megbízhatók és egyszerűen kezelhetők, ezért teszteléshez kiválóan alkalmazhatók.

3.5.2. A TinyOS speciális komponensei és jellemzői

Ezen részben mutatjuk be a TinyOS-ben alapelemként megtalálható néhány, a fejlesztés során fontos szerepet játszó főbb komponenst, illetve az azokkal kapcsolatos tudnivalókat és kötöttségeket. Ez azért fontos, mert így a későbbiekben jobban érthetővé válnak a felmerülő problémák, és az arra kínált megoldások. Emiatt néhány komponens esetén ismernünk kell működésének alapelveit is. A leírásoknál ugyan nem említjük, de az alábbiakban felsorolt komponensek mindegyike szolgáltatja az `StdControl` interfészt.

- ***TimerC: időzítő komponens***

Ez a komponens egy paraméterezhető, `Timer` nevű interfésszel csatlakoztatható más komponensekhez. Az különböző paraméterű interfészekhez különböző időközű, egyszeri vagy ismételt időzítést lehet beállítani, tehát virtuálisan több időzítendő elemet is ki tud szolgálni. Valójában azonban csupán a mikrokontroller Timer0-s, 8 bites időzítőjét használja, mely 32768 Hz-es külső kristályról üzemel, amihez az időzítőn belül egy 32-szeres osztót rendelnek. Ezzel a belső órajel-frekvencia 1024 Hz. Röviden annyit kell tudni a hardveres időzítés folyamatáról, hogy az időzítőben a belső órajel ütemében egy számláló működik, és ha ez a számláló elér egy általunk beállítható értéket, akkor nullázódik, és megszakítást ad. Ezzel tehát egy programozható órajelosztót valósít meg. Az adatokból látható, hogy a beállítható frekvencia 1024 Hz egész számú 8 bites számmal képzett leosztása lehet.
A többféle időzítés beállítását a `TimerC` szoftveresen kezeli le. A fent leírt adatokból két fontos tulajdonság következik:

- Az 1024 Hz-es órajel és a 8 bites leosztása egyrészt bizonyos alkalmazásokhoz nem biztosít elég magas frekvenciát, és a skálázás is elég durva a nagyobb frekvenciás tartományban
- A szoftveres időzítés miatt az időzítés nem pontos, ugyanis az eseményeket nem közvetlenül a hardveres megszakítás generálja. Ráadásul az időzítés pontossága változhat is az időzítőt használó komponensek számától, és azok aktuális állapotától függően.

- ***ADCC: AD-t kezekő komponens***

A komponens a mikrokontrolleren található AD-átalakítót kezeli. Az AD-átalakító 8 csatornás, de egyszerre csak az egyiken folyhat átalakítás. Az `ADCC` komponens viszont segítséget nyújt ennek egyszerű kezelésére, a hozzá kapcsolódó paraméterezhető `ADC` interfész segítségével. Az adott paraméterű interfészt hozzárendeljük a kívánt sorszámú csatornához. Az adott paraméterű interfészen beérkező kéréseket az `ADCC` komponens feljegyzi, és amint lehetséges kezdeményezi az adott paraméterhez tartozó csatornán az átalakítást, majd az átalakítás befejeztével értesíti a megfelelő paraméterű interfészt és ellenőrzi, hogy van-e újabb kérés. A viszonylag hosszadalmas algoritmus miatt itt sem indítható teljesen pontosan az AD-átalakítás, ami időben ingadozó mintavételhez vezethet, és a mintavételezés kezdete függ a processzor aktuális terhelésétől.

- ***MicC: mikrofont kezelő komponens***

A komponens két, számunkra fontos interfészt szolgáltat: `Mic` és `ADC` interfész. A `Mic` interfész egyik fontos tulajdonsága, hogy ezen keresztül beállítható a mikrofonhoz tartozó erősítő erősítési tényezője.

Az `ADC` interfész valójában a komponensen belül található `ADCC` konfiguráció által szolgáltatott megfelelő paraméterű interfész. Rajta keresztül indíthatunk a mikrofonhoz tartozó csatornán egy mintavételezést, és az ehhez tartozó esemény értesít a mintavétel eredményéről.

A `MicC` inicializáláskor felkonfigurálja a mikrokontroller megfelelő lábait, valamint az AD-t, mely kényelmes használatot biztosít.

- ***GenericComm: általános kommunikációs komponens***

A komponens segítségével kényelmesen használhatjuk a rádiós és a soros porti kommunikációt, melyhez két paraméterezhető interfész áll rendelkezésünkre: `SendMsg` és `ReceiveMsg`. Ezek segítségével üzenetet küldhetünk, és eseményt generálnak, ha üzenet érkezett. A paraméterezett interfész lényege ebben az esetben az, hogy egy másik eszközön csak annak a komponensnek generál eseményt az operációs rendszer, mely olyan sorszámú interfészhez csatlakozik, mint amilyen sorszámú interfészről az üzenetet küldtük. Ez azért lehet hasznos, mert így a programok egyes részei virtuálisan különböző csatornákon tudnak kommunikálni.

A kommunikáció konfigurálását lehetővé tevő egyik interfész a `MacBackoff`, bár ezt nem közvetlenül a `GenericComm`, hanem annak egy alkomponense szolgáltatja. Segítségével a közeghozzáférési réteg (MAC) paramétereit állíthatjuk. A MAC működése a következő: adás kezdeményezésekor a rendszer vár egy megadott ideig (`initialBackoff`), és megpróbálja elküldeni az üzenetet. Ha nem szabad a rádiós csatorna, akkor egy szintén megadható ideig (`congestionBackoff`) vár az üzenetküldés további kezdeményezéséig. Ezeket a paramétereket minden egyes küldés alkalmával a `MacBackoff` interfész megfelelő eseményeivel kérdezi le a rendszer, alapbeállításként ezek egy bizonyos tartományban előállított véletlenszámok.

A kommunikációval kapcsolatban meg kell említeni, hogy a TinyOS-ben az üzeneteket csomagokban küldjük, mely csomagok az operációs rendszer által megszabott formátumban kerülnek továbbításra. A TinyOS üzenetformátuma nagy vonalakban a következő: maximum 29 bájtnyi adat továbbítható egy csomagban, amelynek mérete a keretezéssel együtt ekkor 40 bájt. A keretben helyet foglal például a címzett azonosítója, csoportazonosító, üzenethossz, azon interfész száma, amelyikhez csatlakozunk, és CRC ellenőrző rész.

A fejlesztés során sok esetben nem elegendő csupán ezen modulok használatának megismerése, hanem tüzetesen tanulmányozni kell a forráskódjukat is, mivel a komponensek nem minden esetben alkalmasak az általunk kitűzött cél megvalósításához, így több-kevesebb változtatás vált szükségessé.

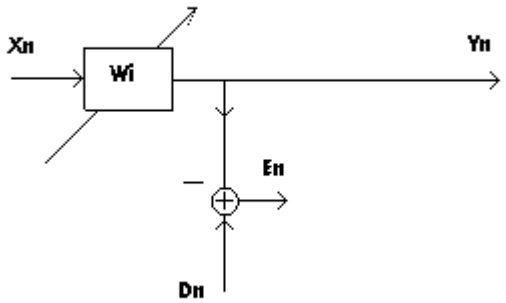
4. Aktív zajcsökkentő eljárások

4.1. Bevezetés

Ebben a fejezetben áttekintjük a legfontosabb zajcsökkentő struktúrákat. Először az adaptív szűrővel felépített zajelnyomó rendszereket mutatjuk be, valamint részletesen tárgyaljuk a nemrekurzív adaptív szűrők együtthatóinak gradiens alapú változtatását végző LMS (Least Mean Square) algoritmust. Ezt követően rátérünk a jelmodell alapú zajelnyomó struktúrák ismertetésére, amelyek periodikus zavaró jelek elnyomására nagyon hatékonyan alkalmazhatóak.

4.2. Az LMS algoritmus

Az LMS (Least Mean Square) algoritmus [10] egy olyan adaptív eljárás, melynek az a célja, hogy az adaptív szűrő kimenő jele (y_n) a lehető legjobban hasonlítson egy ismert, „megkívánt" jelre (d_n). Az, hogy a „lehető legjobban", az LMS algoritmus esetén azt jelenti, hogy a pillanatnyi hiba (e_n) négyzetének várható értéke a minimalizálandó költségfüggvény. Az LMS algoritmus egyszerű vázlatát a 4.1-es ábra mutatja:



4.1. ábra. Az LMS algoritmus blokkvázlata

Az LMS algoritmus tehát a pillanatnyi hiba négyzetét minimalizálja oly módon, hogy a „w_i" együtthatókkal rendelkező FIR szűrő együtthatóit módosítja, mégpedig a következőképpen:

$$e^2(n) = [d(n)-y(n)]^2 \tag{4.1}$$

$$[d(n)-y(n)]^2 = [d(n)-\mathbf{w}^T(n)\mathbf{x}(n)]^2 \tag{4.2}$$

A pillanatnyi derivált ebben az esetben:

$$e(n)' = -2[d(n)-\mathbf{w}^{T}(n)\mathbf{x}(n)]\mathbf{x}(n) \tag{4.3}$$

Azaz, a pillanatnyi deriváltban felismerve magát a hibát, ami rendelkezésre áll:

$$e(n)' = -2e(n)\mathbf{x}(n) \tag{4.4}$$

Így az LMS módszer képlete:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu \{-2e(n)\mathbf{x}(n) \} \tag{4.5}$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu 2e(n)\mathbf{x}(n) \tag{4.6}$$

ahol μ az úgynevezett bátorsági tényező, melynek nagysága befolyásolja a lépésközt, és ezzel együtt a konvergencia sebességet. Optimális értéke az autokorrelációs mátrix ismeretében meghatározható. Most vizsgáljuk meg azt az esetet, hogy a hiba négyzetének várható értékét minimalizálva melyek az optimális „w_i" együtthatók, és azt, hogy az együtthatókat LMS algoritmussal változtatva végül megkapjuk-e az optimális „w_i" készletet. A hiba négyzetének várható értékét minimalizálva:

$$\mathrm{E}\{e(n)^2\} = \mathrm{E}\{(d(n) - \sum w(n)_{\mathrm{i}}\, x(n)_{\mathrm{n-i}})^2\} = \tag{4.7}$$

$$= \mathrm{E}\{d(n)^2 - 2 \sum w(n)_{\mathrm{i}}\, x(n)_{\mathrm{n-i}} + \sum w(n)_{\mathrm{i}} \sum w(n)_{\mathrm{j}}\, x(n)_{\mathrm{n-i}}\, x(n)_{\mathrm{n-j}}\} \tag{4.8}$$

A várhatóérték-képzést az összeg tagjaira külön-külön elvégezve, és bevezetve az autokorreláció (**R**) és a keresztkorreláció (**b**) fogalmát, az egyenlet a következőképpen alakul:

$$\mathrm{E}\{e(n)^2\} = \mathrm{E}\{d(n)^2\} - 2 \sum w(n)_{\mathrm{i}}\, b_{\mathrm{i}} + \sum w(n)_{\mathrm{i}} \sum w_{\mathrm{j}}\, R_{\mathrm{i-j}} \tag{4.9}$$

A fenti egyenletet mátrixos formába átírva látható, hogy az egyenlet tulajdonképpen egy kvadratikus alak, azaz a hibafelület egy paraboloid a paraméterek síkja fölött. Az egyenlet mátrixos alakban ($\mathrm{E}\{d(n)^2\}=a$ helyettesítéssel):

$$E\{e(n)^2\} = a - 2\,\mathbf{w}^T\,\mathbf{b} + \mathbf{w}^T\,\mathbf{R}\,\mathbf{w} \tag{4.10}$$

Az optimális „w_i” együtthatókat a fenti kifejezés minimuma adja, ehhez deriválni kell a kifejezést **w** szerint, majd az eredményt egyenlővé téve nullával, a végeredmény, azaz az optimális „w_i” együtthatók:

$$\mathbf{w}_{opt} = \mathbf{R}^{-1}\,\mathbf{b} \tag{4.11}$$

Most vizsgáljuk meg, hogy az LMS algoritmus a pillanatnyi hiba minimalizálásával végül az optimális „w_i” készletet határozza-e meg. Ha az LMS algoritmus beért az optimumba, akkor:

$$E\{\mathbf{w}(n+1)\} = E\{\mathbf{w}(n)\} \tag{4.12}$$

Ekkor az LMS algoritmus képlete alapján:

$$E\{(d(n) - \sum w(n)_i\, x(n)_{n-i})\,\mathbf{x}(n)\} = 0 \tag{4.13}$$

$$E\{d(n)\,\mathbf{x}(n)\} = \sum w_i\, E\{x_{n-k}\, x_{n-i}\} \tag{4.14}$$

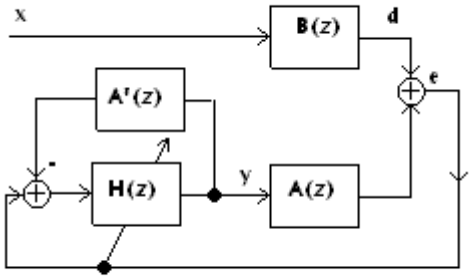
$$\mathbf{b} = \mathbf{R}\,\mathbf{w} \tag{4.15}$$

Ahogy a fenti egyenlet mutatja, az LMS algoritmus az optimális állapot felé konvergál. Tehát az LMS algoritmus az autokorrelációs mátrix és a keresztkorrelációs vektor ismerete nélkül képes megtalálni azokat a „w_i” együtthatókat, melyekkel az adaptív szűrő kimenete a lehető legjobban „hasonlít” a megkívánt jelre. Ez azért nagyon fontos, mivel általában a gyakorlati alkalmazások során sem az autokorrelációs mátrix, sem a keresztkorreláció nem ismert. Az LMS algoritmus viszont nagyon jól alkalmazható abban az esetben, ha valós időben ismert az adaptív szűrő kimenete és a megkívánt jel közti különbség. Az aktív zajcsökkentő eljárások ilyenek, hiszen az a cél, hogy az elnyomandó jellel azonos jelet adjunk ki (csak ellentétes fázisban), a hibamikrofonnal pedig azt a jelet vesszük, ami az LMS algoritmus számára szükséges. Ezért alkalmazzák előszeretettel az LMS algoritmust és különböző változatait az aktív zajcsökkentő eljárásokban.

A következő két alfejezetben bemutatunk két, LMS algoritmussal is használható zajcsökkentő struktúrát, a rendszer működését pedig szimulációs eredményekkel igazoljuk.

4.3. A visszacsatolt struktúra [11],[12]

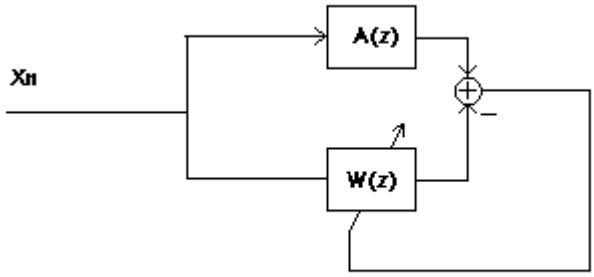
A visszacsatolt rendszer blokkvázlata az alábbi ábrán látható:



4.2. ábra. A visszacsatolt struktúra blokkvázlata

Az árán $A(z)$-vel jelölt átviteli függvény megegyezik az 1.2-es ábra $A(z)$ jelölésével, azaz a zajcsökkentő struktúra kimenetétől a hibamikrofonig terjedő út átvitelét képviseli. $B(z)$ jelölés a zajforrástól a hibamikrofonig terjedő út átvitele. $A'(z)$ átviteli függvény a $A(z)$ becslője, $A'(z)$-t a működés megkezdése előtt kell identifikálni, és így a hibamikrofon jelével állítva $H(z)$ együtthatóit, a rendszer kimenő jele az ábra szerint adódik. Fontos megjegyezni, hogy a visszacsatolt struktúra nem használ fel a zajforrásból érkező referenciajelet. Ez azt jelenti, hogy a rendszer állandósult állapotában a zajcsökkentő struktúrának a nulla, vagy nullához nagyon közeli hibajelből kell előállítania a nem nulla beavatkozójelet. A másik probléma az, hogy a beavatkozójel frekvenciáját szintén a hibajel alapján kell állítani. Ezeknek a feladatoknak a megfelelő pontossággal történő elvégzése pedig sajnos csak a rendszer sebességének csökkentésével érhető el. Ez az oka annak, hogy a visszacsatolt rendszerek általában lassabbak, mint az előrecsatoltak, előnyük azonban, hogy nincs szükségük semmilyen referenciajelre a zajforrásból.

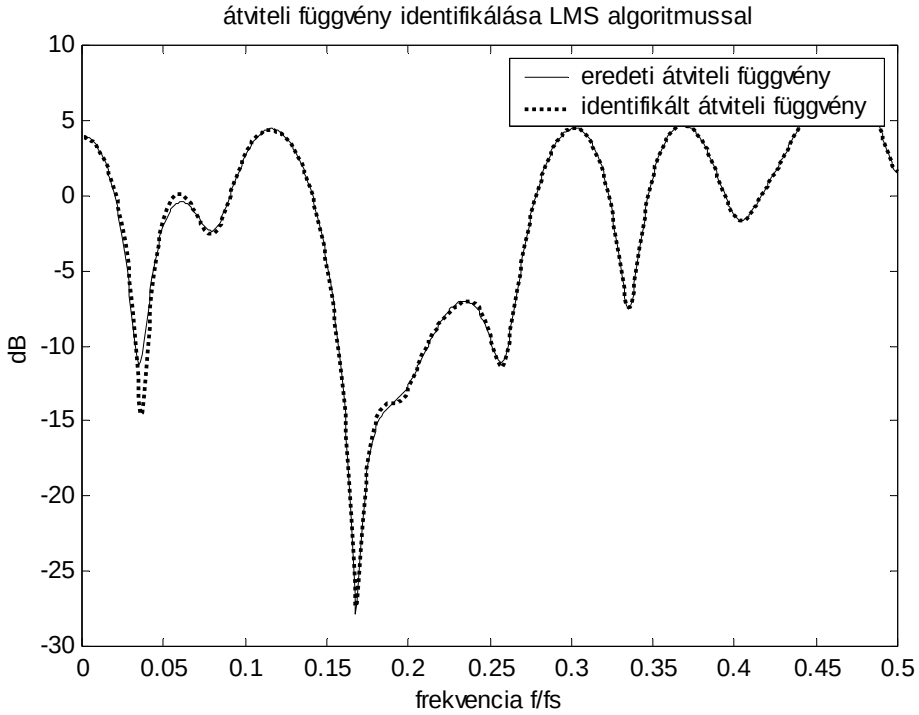
$A(z)$ meghatározása többféleképpen is történhet, mi a következő elrendezést alkalmaztuk:



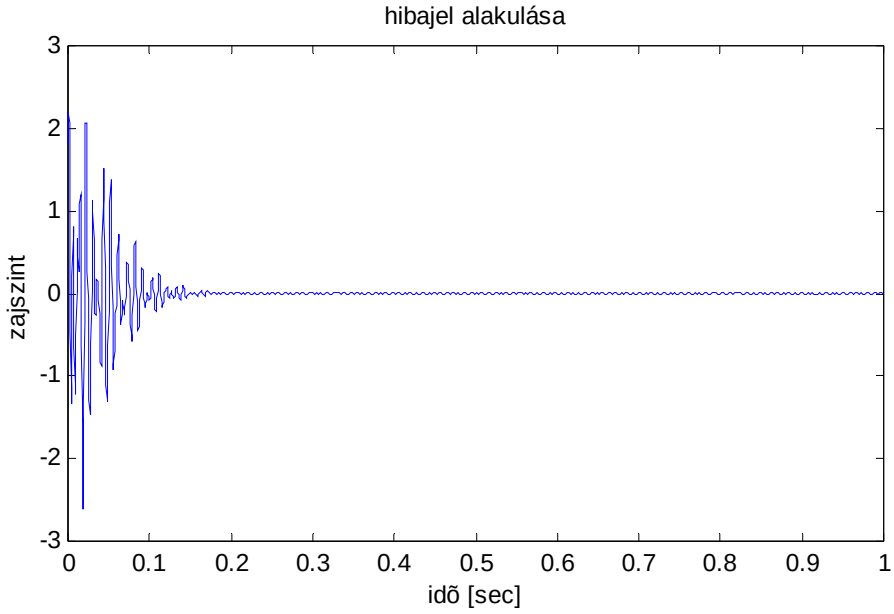
4.3. ábra. Átviteli függvény mérése LMS algoritmussal

Az ábrán x_n gaussi fehér zaj, így, ha a rendszer beállt, azaz a hibajel nullára csökkent, $W(z)$ együtthatói a $A(z)$ átvitel becslését adják.

Az alábbiakban egy visszacsatolt zajcsökkentő rendszer működését mutatjuk be MATLAB szimuláción keresztül. A 4.4. ábrán az identifikálandó akusztikus átviteli függvény és az identifikáció eredménye látható egymásra rajzolva, a 4.5. ábra a hibajel alakulását mutatja az idő függvényében. Az elnyomandó jel egyszerű szinuszjel:



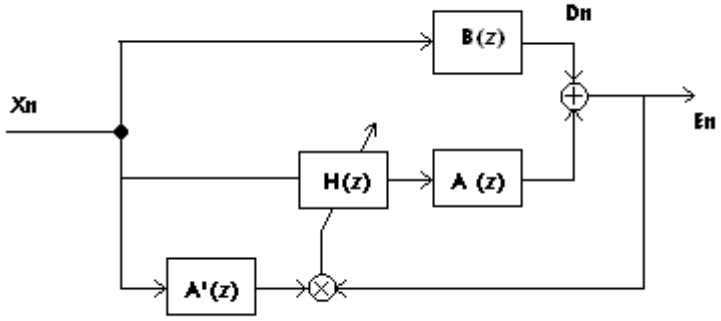
4.4. ábra. LMS algoritmussal mért és az eredeti átviteli függvény összehasonlítása



4.5. ábra. Visszacsatolt struktúra hibajelének alakulása az idő függvényében

4.4. Az előrecsatolt struktúra

Az előrecsatolt rendszer blokkvázlatát a következő ábra mutatja:



4.7. ábra. Az előrecsatolt rendszer blokkvázlata

Az ábrán jól látható, hogy az előrecsatolt struktúra a visszacsatolttal ellentétben megkapja a zajforrásból származó referenciajelet. Ez a jel akár egy, a zajforrás közvetlen közelébe rakott mikrofonnal is vehető, ekkor azonban számolni kell azzal, hogy a vett jelhez hozzákeveredhet a beavatkozó hangszóró

jele (főleg ha kicsi a távolság a zajforrás és a beavatkozó hangszóró között), ami a rendszer instabilitását is eredményezheti.

A fent említett parazita hatás nem léphet fel, ha a referenciajelet nem mikrofonnal, hanem valamilyen egyéb szenzorral vesszük. Ha így járunk el, a megvalósított rendszerünk stabilabb lesz.

Az előbbiekben ismertetett, LMS algoritmus alapú zajcsökkentő eljárásokat igen széles körben használják szélessávú zajok elnyomására. Az általunk megvalósított szenzorhálózatos struktúrában LMS alapú zajcsökkentő eljárásokat is meg lehetne valósítani, periodikus jelekre azonban kedvezőbb a következő fejezetekben bemutatásra kerülő rezonátoros struktúra alkalmazása. Rezonátoros struktúrával a zajforrásról információt hordozó referenciajel vétele is sokkal egyszerűbb, hiszen az LMS algoritmusnál szükség van az összes elnyomandó harmonikusra, mivel az LMS algoritmusnál tulajdonképpen egy FIR szűrőt alkalmazunk, ami nem generál felharmonikusokat. Ezzel szemben a rezonátoros struktúra akár az alapharmonikus alapján is képes hatékonyan felhasználni a referenciajelet.

4.5. Zajcsökkentés periodikus jelekre

A gyakorlatban a zajok, zavarhatások nagy része köznyelven fogalmazva zúgó, búgó hang, mint például egy repülő turbinájának vagy egy autó motorjának a hangja. Ez azt jelenti, hogy azoknak a zajhatásoknak a nagy része, amelyeket aktív zajcsökkentéssel szeretnénk csillapítani, periodikus jel. Természetesen ezekre a jelekre az előző fejezetben említett szélessávú zajcsökkentő eljárások is használhatóak, de mivel a periodikus jelek leírására szolgáló apparátus, valamint az ilyen rendszerek vizsgálatához szükséges módszerek is különböznek a szélessávú zajok leírásától és vizsgálati módszereitől, ez indokolja azt, hogy egy másfajta struktúrát használjunk a periodikus jelek elnyomására.

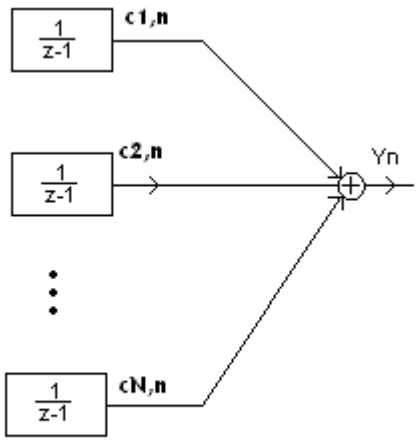
Periodikus jelek leírására a Fourier-reprezentáció használható. Ez azt jelenti, hogy egy sávkorlátozott, periodikus jel ($p(t)$) egyértelműen megadható az ő Fourier sorával:

$$p(n) = \sum P_k \, c(n)_k \tag{4.16}$$

$$c(n)_k = e^{j\,2\pi f k n}, \; k = -K \ldots K \tag{4.17}$$

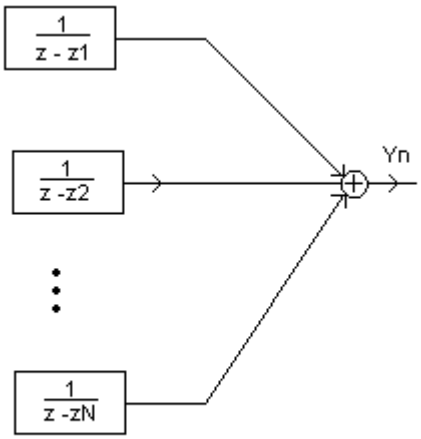
A fenti két képletben f a periodikus jel alapharmonikusának frekvenciája, k véges sok értéke pedig a sávkorlátozottságot fejezi ki.

A 4.16. és 4.17. képletek segítségével megalkothatjuk a periodikus jelek koncepcionális jelmodelljét, azaz meghatározhatunk egy olyan struktúrát, amely periodikus jelek generálására képes. Ez a modell látható az alábbi ábrán:



4.8. ábra. Sávkorlátozott periodikus jel modellje

A fenti ábra alapján egy $p(n)$ sávkorlátozott periodikus jel úgy jön létre, hogy az integrátorok bemenetére a nulla időpillanatban a 4.16 egyenletnek megfelelő P_k Fourier-együtthatókat adjuk, majd a továbbiakban nullát adunk az integrátorok bemenetére. Ennek következtében az integrátorok tartani fogják a nulla időpillanatban felvett értékeiket, így a modell pontosan a 4.16 egyenlet által megfogalmazottakat valósítja meg. Sávkorlátozott jelek generálására egy másik, az előzővel ekvivalens modell is használható, ezt mutatja a 4.9-es ábra:

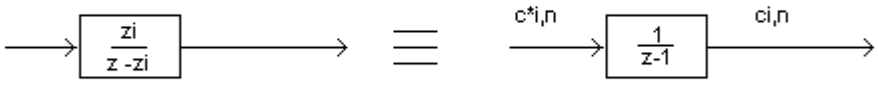


4.9. ábra. Sávkorlátozott periodikus jel modellje

A fenti ábrán z_i a következőképpen határozható meg:

$$z_i = c(n+1)_i / c(n)_i \tag{4.18}$$

A rezonátoros és az integrátoros modell megfeleltetését a következő ábra mutatja:

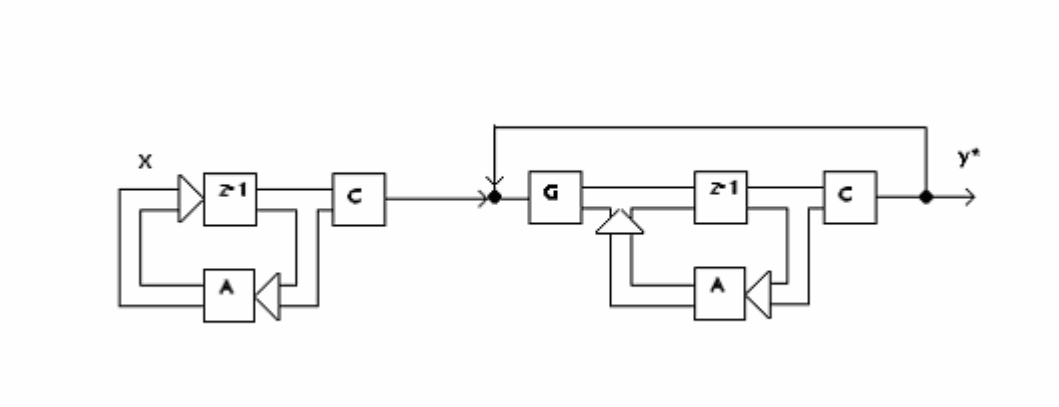


4.10. ábra. A rezonároros és az integrátoros jelmodell megfeleltetése

A következő fejezetben azt szeretnénk megmutatni, hogy a fenti jelmodellekhez hogyan tervezhető megfigyelő, és ez a struktúra miként használható zajcsökkentési feladatokra.

4.6. Jelmodell alapú megfigyelés, a rezonátoros struktúra

A megfigyelők olyan rendszerek, amelyek egy másik rendszer állapotváltozóit, vagy az azokból származtatott mennyiségeknek a meghatározását teszik lehetővé. Azaz olyan mérési eljárást valósítanak meg, melynek során egy rendszer méréstechnikailag nem hozzáférhető állapotváltozóinak lemásolásával azok mérését teszik lehetővé. A mérendő rendszer és a megfigyelő kapcsolatát mutatja a 4.11. ábra [13]:



4.11. ábra. A megfigyelő és a megfigyelt rendszer viszonya

Az ábrán a megfigyelt rendszer állapotváltozóit **x**-szel jelöltük, az állapotátmeneti mátrixot pedig **A**-val. A **C**-vel jelölt mátrixot kicsatoló mátrixnak nevezzük. Ezt a bal oldalon álló független rendszert figyeljük meg az ábra jobb oldalán található megfigyelővel.

A megfigyelőt úgy kell megalkotni, hogy fel kell állítanunk a mérendő rendszer egy koncepcionális jelmodelljét, ami az ábra esetén már megtörtént, majd az **A** és **C** mátrixok segítségével még egyszer meg kell valósítani a mérendő rendszert, kiegészítve az egy bemenettel. Ez a bemenet fogja szolgáltatni a mérendő rendszer és az általunk megépített kópia kimenete közti különbséget, amit egy **G** becsatoló mátrixon keresztülvezetve arra használunk, hogy az általunk megépített rendszer állapotváltozóit módosítsuk oly módon, hogy a valós rendszer és a kópia kimenete közötti különbség minél kisebb legyen.

A megfigyelt rendszer állapováltozós leírása tehát:

$$\mathbf{x}(n+1) = \mathbf{A}\,\mathbf{x}(n) \tag{4.19}$$

$$\mathbf{y}(n) = \mathbf{C}\,\mathbf{x}(n) \tag{4.20}$$

Most felírjuk a megfigyelő állapotváltozós leírását, a megfigyelő állapotváltozóit **x**'-vel jelölve:

$$\mathbf{x}'(n+1) = \mathbf{A}\,\mathbf{x}'(n) + \mathbf{G}\,\{\mathbf{y}(n)-\mathbf{y}'(n)\} \tag{4.21}$$

$$\mathbf{y}'(n) = \mathbf{C}\,\mathbf{x}'(n) \tag{4.22}$$

Ha a 4.21-be visszahelyettesítjük 4.20-at és 4.22-t, akkor algebrai átalakítással belátható, hogy a megfigyelő állapotátmeneti mátrixa **A-GC.** A megfigyelő akkor működik megfelelően, ha a kópia és a megfigyelt rendszer állapotváltozói közti különbség fokozatosan csökken, és előbb-utóbb nullává válik. A hibára a következő egyenlet írható fel:

$$\mathbf{x}(n+1) - \mathbf{x}'(n+1) = \{(\mathbf{A} - \mathbf{G}\,\mathbf{C})^{n+1}\,(\mathbf{x}(0) - \mathbf{x}'(0)\} \tag{4.23}$$

A fenti képletből látható, hogy a hiba akkor lesz nulla, ha **A-GC** már eleve nulla, ez abban az esetben lehetséges, ha a **C** négyzetes, azaz annyi kimenete van a megfigyelt rendszernek, ahány állapotváltozója, és még az is szükséges, hogy ezekből a kimenetekből **C** állapotváltozóira vissza lehessen következtetni. Ez tehát azt jelenti, hogy **C**-nek négyzetesnek kell lennie, és sor illetve oszlopvektorainak függetleneknek. Ha a fenti feltétel teljesül, akkor egylépéses konvergencia biztosítható, amennyiben a $\mathbf{G} = \mathbf{A}\mathbf{C}^{-1}$ választással élünk. Ha **C** nem négyzetes, akkor is nullává válhat a hiba, ha valamely n-re $(\mathbf{A} - \mathbf{G}\mathbf{C})^n = 0$ adódik. Ekkor az (**A** – **GC**) mátrix nilpotens. A nilpotens mátrixokra pedig az jellemző, hogy mindegyik sajátértékük nulla. Ez egyben azt is jelenti, hogy a rendszer pólusai az origóban helyezkednek el, vagyis a rendszerünk FIR. Ez számunkra is nagyon jó, mivel ha a megfigyelőnk véges lépésben beáll, akkor feltételezhető, hogy a megfigyelt rendszert igen gyorsan tudja követni.

A fentiek ismeretében a feladatunk már csupán annyi, hogy a periodikus jelek koncepcionális modelljét használva felírjuk parametrikusan az (**A** – **GC**) mátrixot, és kihasználva azt a feltételt, hogy mindegyik sajátértéknek nullának kell lennie, megoldjuk **G** elemeire az egyenletrendszert. Tehát a rezonátoros jelmodellre:

$$\mathbf{A} = \langle z_i \rangle \tag{4.24}$$

$$\mathbf{C} = \mathbf{c}^T = [111..1] \tag{4.25}$$

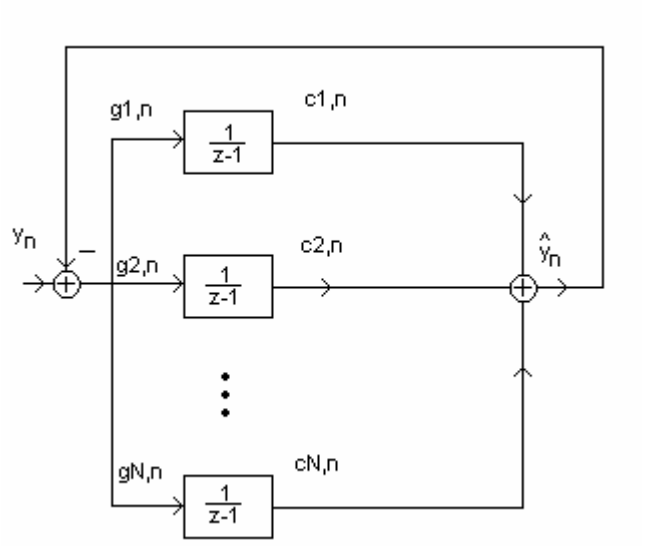
Az egyenleteket a fenti gondolatmenet szerint megoldva, véges beállású megfigyelőt feltételezve, valamint azt, hogy a rezonátorpólusok az egységkörön egyenletesen helyezkednek el, az eredmény:

$$g_i = z_i \ 1/N \tag{4.26}$$

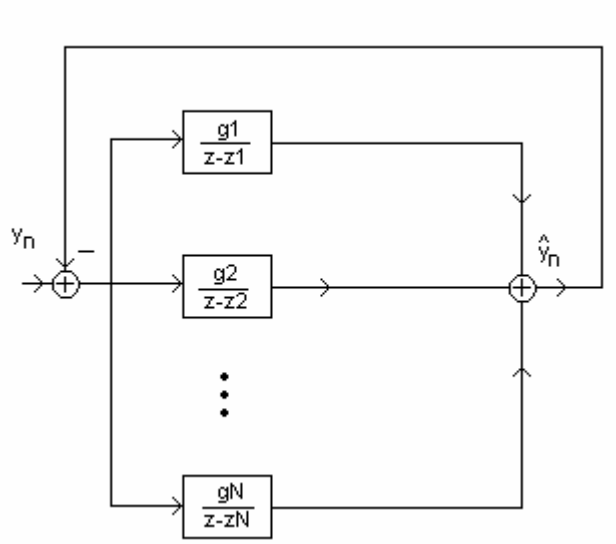
A fenti egyenletben N a rezonátorok száma, z_i az i-edik rezonátorhoz tartozó pólus. Természetesen az integrátoros jelmodellhez is tervezhető megfigyelő. Erre az eredmény az előző esetben használt feltételeket megtartva:

$$g(n)_i = c^*(n)_i \ 1/N \tag{4.27}$$

A fenti képletben a csillag a konjugálást jelöli, *n* a diszkrét időt jelöli, míg i azt, hogy hányadik rezonátorról van szó, N pedig a rezonátorszám. A jelmodellekhez tartozó megfigyelőket az alábbi két ábra mutatja:

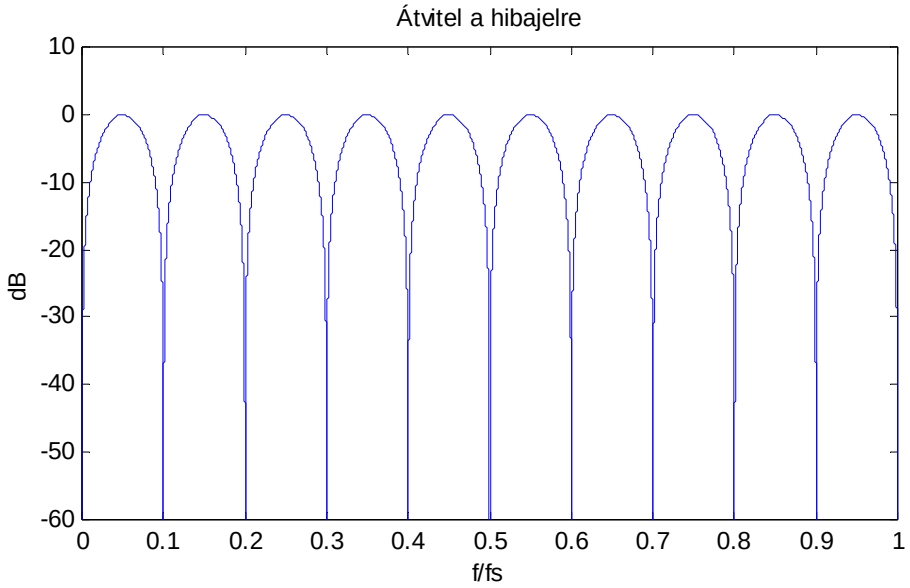


4.12. ábra. Rezonátoros megfigyelő integrátorokkal megvalósítva

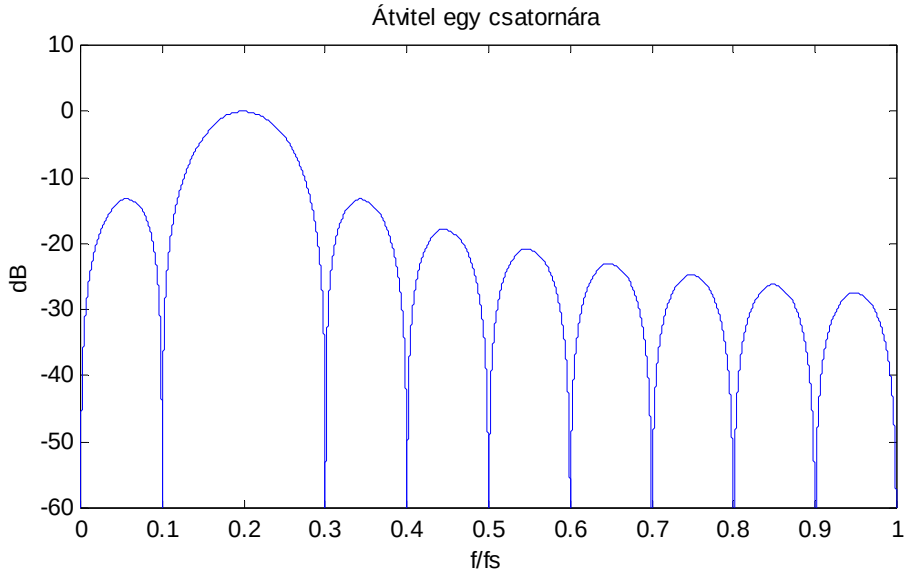


4.13. ábra. Rezonátoros megfigyelő rezonátorokkal megvalósítva

A rezonátoros struktúra ismertetését néhány jellemző átviteli függvény megadásával zárjuk. A következő két ábra közül a 4.14. a hibajel amplitúdó karakterisztikáját mutatja a frekvencia függvényében, a 4.15. az i-edik csatorna átvitelét mutatja N=10 esetén.



4.14. ábra. Rezonátoros struktúra átvitele a hibajelre a relatív frekvencia függvényében



4.15. ábra. Rezonátoros struktúra átvitele egy csatornára a relatív frekvencia függvényében

Az ábrákból jól látható, hogy a rezonátoros struktúra azokon a frekvenciákon, ahol rezonátorpólusok vannak, nulla hibával előállítja a jelet, más frekvenciákon a hiba valamilyen véges érték. A 4.15. ábra, ami egy csatorna átvitelét mutatja, jól mutatja, hogy egy csatorna csak a saját frekvenciáján biztosít egységnyi átvitelt, sőt a többi rezonátorfrekvencián az átvitele nulla.

A fentiekben bemutatott rezonátoros struktúra a gyakorlatban is sikeresen alkalmazható, mi saját alkalmazásainkban az integrátoros megfigyelőt használtuk egyszerűbb megvalósíthatósága miatt.

4.7. Adaptív Fourier-analízis [14]

Az előző fejezetben láttuk, hogy a rezonátoros struktúra a rezonátorfrekvenciákon pontosan előállítja a bemenő jelet. Gyakorlati alkalmazásokban azonban az elnyomni kívánt jel frekvenciája változhat. Ilyen eset például, ha egy autó motorjának túlságosan zúgó hangját szeretnénk elnyomni; ha az autó gyorsul, vagy fékez, az elnyomandó jel változik. Mivel az elnyomást ilyen esetekben is biztosítani szeretnénk, a rezonátorpólusokat az elnyomandó jel frekvenciájának megfelelően hangolnunk kell. Ezt a feladatot látja el az Adaptív Fourier Analizátor (AFA).

Az AFA a rezonátorok pólusait az elnyomandó jel frekvenciájára hangolja, ezzel biztosítja a rezonátoros struktúra számára, hogy az elnyomandó jel változásait követni tudja, így a zajelnyomásra a nulla hiba megcélozható. Ehhez természetesen szükség van a zajforrásból származó referenciajelre. Ez általában

nem okoz problémát, valamilyen egyszerű szenzor alkalmazásával könnyen megoldható.

Az AFA tulajdonképpen egy rezonátoros struktúra, működését legszemléletesebben a rezonátoros struktúra integrátorokkal való megvalósítását mutató 4.12. ábra segítségével lehet bemutatni. A jelmodell ebben az esetben:

$$y(n) = c(n)^{\mathrm{T}}\, x(n) \tag{4.28}$$

$$c(n)_{\mathrm{k}} = \mathrm{e}^{\mathrm{j}\, 2\pi\, 1/\mathrm{N}\, k\, n}\,,\ k = -\mathrm{K}\ldots\mathrm{K},\ \mathrm{N} = 2\ \mathrm{K}+1 \tag{4.29}$$

A fenti esetben az alapharmonikus relatív frekvenciája 1/N. Az állapotváltozós leírás a következőképpen írható fel:

$$x'(n+1) = x'(n) + 1/\mathrm{N}\ c^{*}(n)\ \{y(n)-c(n)^{\mathrm{T}}x'(n)\} \tag{4.30}$$

$$y'(n) = c(n)^{\mathrm{T}}\, x'(n) \tag{4.31}$$

Egy csatorna működése úgy magyarázható meg szemléletesen, hogy a hibajelet először a c^{*} együttható DC-re keveri, majd integrálás után a megfelelő c együttható keveri vissza a jelet az eredeti frekvenciára. Ha a megfigyelő pontosan illeszkedik a jel frekvenciájához, akkor az állapotváltozó nem változik; ha ez nem teljesül, akkor az állapotváltozó egy forgó komplex vektor lesz, a forgás sebessége pedig az aktuális frekvenciakülönbségnek felel meg. Ezért ezt (az állapotváltozó két időpillanat közötti szögeltérését) használjuk fel a frekvencia adaptálására:

$$f_1(n+1) = f_1(n) + 1/(2\pi\mathrm{N})\ \mathrm{phase}\{x'_1(n+1)-x'_1(n)\} \tag{4.32}$$

$$c(n+1)_{\mathrm{k}} = c(n)_{\mathrm{k}}\ \mathrm{e}^{\mathrm{j}2\pi f1(n+1)k} \tag{4.33}$$

Az előbbi két képlet azt fejezi ki, hogy a rendszer alapfrekvenciájának adaptálása után a c együtthatók 4.33 szerinti frissítésével a rendszer frekvenciája adaptálódott. Fontos megemlítenünk, hogy ha a rendszer frekvenciája nő, akkor a legnagyobb frekvencián lévő rezonátorok közelíthetnek f = 0,5 frekvenciához, azaz a mintavételi frekvencia feléhez. A rendszer stabilitása és a véges beállás szempontjából nagyon fontos, hogy a rezonátorok egymástól egyenletes távolságra helyezkedjenek el. Ha azonban a frekvencia adaptálás során az egyik rezonátor nagyon megközelíti az f = 0,5 frekvenciát, akkor ő és a tükörfrekvenciáján lévő rezonátor egymáshoz nagyon közel kerülhetnek. Ez akár

a rendszer instabilitását is okozhatja. Ezért, ha valamelyik rezonátor az $f = 0{,}5$ frekvenciát megközelíti, akkor azt 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-elhelyezkedést biztosítandó új rezonátorokat kell beléptetnünk a rendszerbe. Ezeknek a rezonátoroknak az inicializálása a következő:

$$x(n+1)_K = x(n+1)_{-K} = 0 \tag{4.34}$$

$$c(n+1)_K = c(n+1)_{-K} = 1 \tag{4.35}$$

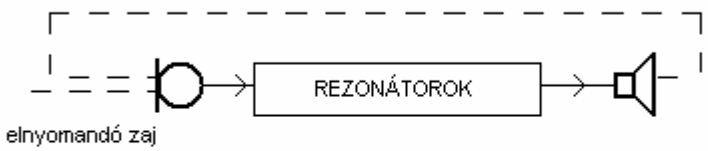
A fentiekben az AFA általunk és a gyakorlatban is használt formáját ismertettük. Természetesen az adaptív Fourier analizátoroknak számos fajtája megtalálható gyakorlati alkalmazásokban, a mi szempontunkból azonban csak a fentiek voltak lényegesek.

5. Rezonátoros struktúra alapú zajcsökkentő eljárások elemzése és megvalósítása

5.1. Bevezetés

Az előző fejezetben ismertetett, AFA-val kiegészített rezonátoros struktúra nagyon jól alkalmazható olyan zajelnyomási feladatokra, melyeknél az elnyomandó jel periodikus. A rendszer előnyei, hogy kis számítási igényt támaszt – a memóriában a legnagyobb helyet az exponenciálisok kiszámításához szükséges szinusztáblázat foglalja – valamint az elnyomandó jelet igen pontosan elő tudja állítani. Az előző fejezet eredményei alapján az általunk megvalósított zajcsökkentő rendszerekben a rezonátoros struktúrát részesítettük előnyben (természetesen próbaképpen megvalósítottunk LMS alapú rendszereket is).

Ahogy az előző fejezetben láttuk, ha a frekvencia-adaptálás pontos, azaz a bemenő jel komponenseinek frekvenciái megegyeznek a rezonátor frekvenciákkal, akkor a jel előállítás pontos, az elnyomás teljes lehet. Ha azonban a rezonátoros struktúrát összevetjük az aktív zajcsökkentő rendszer blokkvázlatát bemutató 1.2-es ábrával, akkor láthatjuk, hogy a rezonátoros struktúra esetén $A(z) = 1$. A valóságban a gyakorlati alkalmazások során azonban $A(z)$ mindenképpen jelen van. Egy ilyen zajcsökkentő rendszer blokkvázlatát mutatja az alábbi ábra:



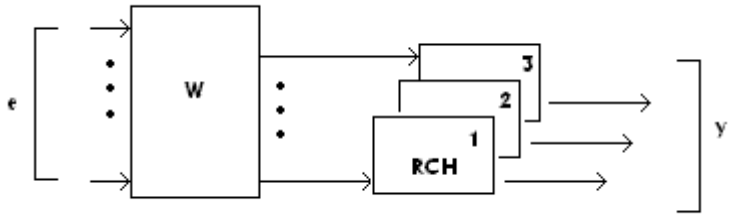
5.1. ábra. Egycsatornás zajcsökkentő rendszer rezonátoros struktúrával

Az ábra alapján jól látható, hogy az aktív zajcsökkentés esetén a rezonátoros struktúra $A(z)$-val szűrt kimenő jelének kell egyeznie a bemenő jellel. Ez azt jelenti, hogy a rezonátorok visszacsatolása egy külső hurkon keresztül valósul meg. A rezonátoros struktúra működéséhez azonban el kell érnünk, hogy a fenti feltétel ($A(z)=1$) teljesüljön. Ennek érdekében szükség van $A(z)$ inverzére, hiszen ezzel szűrve a rezonátoros struktúra kimenetét az mindenképpen keresztül megy $A(z)$-n, így az eredmény az lesz, hogy a késleltetéstől eltekintve biztosítottuk a $A(z)=1$ feltételt. $A(z)$ inverzének meghatározásáról a későbbiekben ejtünk szót.

A maximális konvergenciasebesség eléréséhez véges impulzusválaszú (FIR) rendszer tervezését tűztük ki célul, mivel feltételeztük, hogy a FIR rendszerek beállása gyorsabb, mint az IIR rendszereké. Az ilyen rendszerek hátránya, hogy a tranziensek során nagyok a túllövéseik, állandósult állapotban pedig nagyon érzékenyek a zajokra. A túllövések azért nagyon veszélyesek, mivel túlcsordulást okozhatnak. Akusztikus esetben azonban ez nem reális veszély.

A zajelnyomás csak akkor lehet teljes, ha – mint már említettük – a rezonátorokat AFA segítségével a megfelelő frekvenciákra hangoljuk. Ez a hangolás történhet kétféleképpen is: az első megoldás, hogy a zajcsökkentő eljárásban adaptív rezonátorkészletet alkalmazunk, ekkor azonban a teljes zajelnyomó hurok felel meg az AFA-nak, így a frekvenciaadaptáció lassabb. A másik megoldás, hogy a zajforrásból vett refereciajel segítségével valamint egy külön rezonátoros struktúrával működtetjük az AFA-t, a zajelnyomást végző rezonátorokat pedig a *c* együtthatók változtatásával adaptáljuk. Ez a két megoldás hasonló a már korábban említett előre és visszacsatolt struktúrához, azok előnyeit és hátrányait is hordozzák. Mi az előrecsatolt rendszer mellett döntöttünk, azaz a zajforrásból vett referenciajellel végeztük a frekvencia adaptálását.

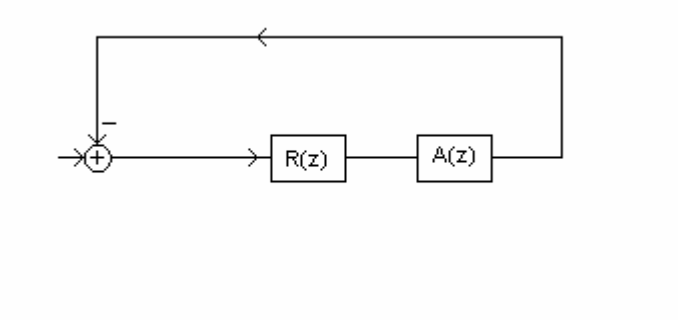
Az egycsatornás rendszerek mintájára megszerkeszthető a többcsatornás renszer blokkvázlata is. Ebben az esetben minden kimenethez tartozik egy független rezonátoros struktúra. A frekvencia adaptálására továbbra is használható az AFA. A hibajel azonban ebben az esetben egy vektor, ezt egy **W** becsatolómátrix segítségével vezetjük a rezonátorok bemenetére. Ezt mutatja az 5.2-es ábra:



5.2. ábra. Többcsatornás zajcsökkentő rendszer blokkvázlata

5.2. Stabilitás

Ebben a fejezetben szeretnénk ismertetni a rezonátoros struktúra stabilitásának szükséges feltételeit egy és többcsatornás esetben. A zajcsökkentő hurok stabilitásának vizsgálatához tekintsük az 5.3. ábrát:



5.3. ábra Zajcsökkentő rendszer blokkvázlata

Az ábrán $R(z)$ jelöli a visszacsatolatlan rezonátoros struktúrát, $A(z)$ pedig az akusztikus átviteli függvényt. $R(z)$ tulajdonképpen az egy csatornára vonatkozó átvitelek összege, azaz:

$$R(z) = \sum Q_i(z) = \sum r_i z_i / (z-z_i) \tag{5.1}$$

A következőkben azt szeretnénk kideríteni, hogy a rendszer milyen r_i választással lesz stabil, hiszen stabil rendszer és a rezonátorok megfelelő frekvenciára való hangolása esetén az elnyomás teljes. Tekintsük r_i-t a következő formában:

$$r_i = \alpha\, w_i \tag{5.2}$$

Egyrészt kínálkozik, hogy a hurokerősítést a szabályozási körökhöz hasonló módon kellően kicsire választva a rendszer stabil lehet. Ez α kicsire választását jelentené. Erről azonban belátható, hogy a rezonátoros struktúra esetén nem alkalmazható kizárólagosan, szükség van egy további feltételre.

A stabilitás vizsgálatára a Nyquist-kritériumot alkalmazzuk. Ha a rezonátorokat az egységkör belsejére soroljuk, akkor a Nyquist- kritérium szerint a stabilitás feltétele az, hogy a nyílt hurok Nyquist- görbéje a -1 pontot ne ölelje körül.

Belátható, hogy egyetlen visszacsatolt rezonátor α segítségével csak akkor stabilizálható a rendszer, hogyha w_i-re teljesül, hogy:

$$|\arc (w_i)| < \pi/2 \tag{5.3}$$

Ezek után a teljes hurok stabilitásáról belátható, hogy stabilitásának szükséges feltétele az, hogy:

$$|\arc A(w_i) + \arc(w_i)| < \pi/2 \tag{5.4}$$

Összefoglalva: w_i-k fenti kritériumnak megfelelő megválasztásával, valamint α konvergencia paraméter kellően kicsivé választásával a rendszer stabilitása biztosítható. Gyakorlati alkalmazásokban célszerű a rezonátor csatornákon a következő választással élni:

$$w_i = 1/A(z_i) \tag{5.5}$$

Többcsatornás rendszer esetén a nyílt hurkú rendszer átviteli függvénye a következő:

$$\mathbf{F}(z) = \mathbf{A}(z) \sum \mathbf{R}_i Q'_i(z) \tag{5.6}$$

Ahol $\mathbf{R}_i$ a hibajelvektort a rezonátorok bemenetére csatoló mátrix. A rezonátorokat továbbra is az egységkör belsejéhez sorolva a többdimenziós rendszerek stabilitására vonatkozó Nyquist- kritérium szerint a visszacsatolt rendszer akkor stabil, ha **F** sajátértékei (amelyek frekvenciafüggők) nem ölelik körül a -1 pontot. $\mathbf{R_i}$-t a következő alakban keressük:

$$\mathbf{R_i} = \alpha\, \mathbf{W_i} \tag{5.7}$$

A levezetésből azt kapjuk, hogy a többcsatornás rendszer stabilitásának szükséges feltétele:

$$| \arc \lambda_i | < \pi/2 \tag{5.8}$$

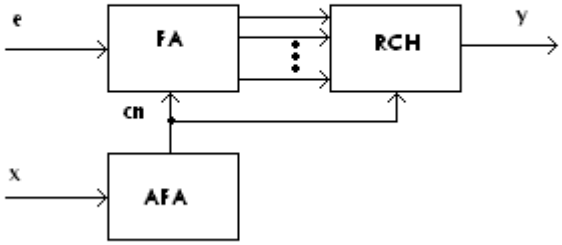
ahol az $\mathbf{A}$ (w_i) $\mathbf{W_i}$ mátrix sajátértékeit λ-val jelöltük. Ebből következik, hogy gyakorlati alkalmazásokban a következő paraméterválasztással érdemes élni:

$$\mathbf{W_i} = \mathbf{A}^{\#}(w_i) \tag{5.9}$$

ahol # a pszeudoinverz-képzést jelöli.

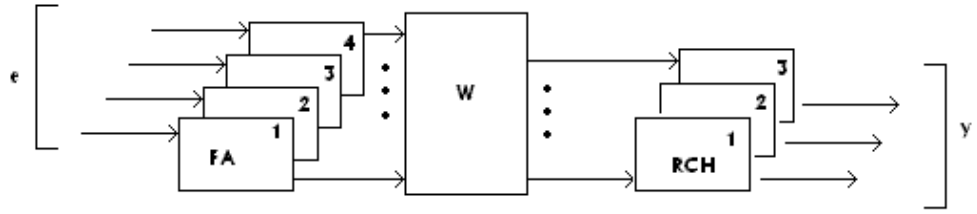
5.3. A konvergenciasebesség növelése

Ebben a fejezetben bemutatjuk, hogy hogyan növelhető a rezonátoros zajelnyomó rendszer konvergenciasebessége. Egy ilyen megnövelt sebességű egycsatornás rendszer blokkvázlata látható az 5.4. ábrán:



5.4. ábra. Megnövelt sebességű egycsatornás zajcsökkentő rendszer blokkvázlata

Az ábrán látható rendszer abban különbözik az eddig tárgyalttól, hogy a bemenetre érkező hibajelet az FA-val jelölt, Fourier analizátor komponenseire bontja, majd ezek a komponensek a zajcsökkentést végző RCH rezonátorok bemeneteire kerülnek. Mindkét rezonátoros struktúra rezonátorpozícióit AFA-val állítjuk. Ez az elrendezés általánosítható többcsatornás esetre is, ahogy az 5.5. ábra mutatja:



5.5. ábra. Megnövelt sebességű többcsatornás zajcsökkentő rendszer blokkvázlata

Az ábrán látható, hogy minden hibajelhez, azaz minden mikrofonhoz tartozik egy analizátor-készlet, és minden kimenethez egy rezonátor-készlet. Ez a gyakorlatban is nagyon jól használható struktúra tehát három rezonátor-készlet működtetését igényli. Igaz ugyan, hogy ez kicsit megnöveli a számítási igényt, de mindamellett jelentős sebességnövekedést jelent.

5.4. Az akusztikus átvitel mérése és a mért minták felhasználása

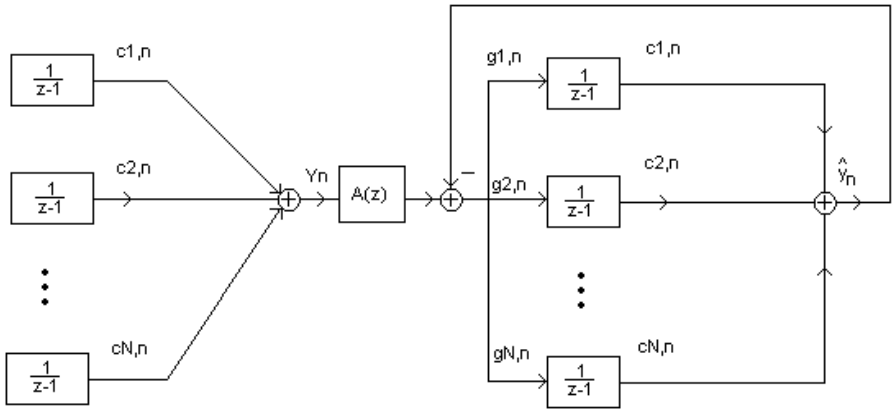
Az átviteli függvény mérése általános feladat, amelyre sokféle megoldás kínálkozik:

- gerjesztés fehér zajjal, kiértékelés FFT-vel
- gerjesztés multiszinuszos jellel, kiértékelés FFT-vel
- adaptív szűrő paramétereinek beállítása (például LMS algoritmussal).

Az első módszer jól alkalmazható, hátránya a mérés nagy varianciája, valamint az is elképzelhető, hogy azokon a frekvenciákon, ahol $A(z)$ átvitele kicsi, esetleg nem gerjeszti elég jól a rendszert.

A második módszer esetén a variancia kisebb, de a gerjesztőjel megkonstruálása nagyon bonyolult feladat. Ezen kívül multiszinusz esetén egy szinuszjelre nagyon kis teljesítmény jut, tehát kicsi lesz a gerjesztés. További hátrány, hogy $A(z)$-nek, amely az akusztikus rendszeren keresztül analóg elektronikát is tartalmaz, intermodulációs torzítása is lehet, emiatt egy adott frekvencián a kis amplitúdójú hasznos jelhez nagyobb amplitúdójú lekeverődő zavaró jel adódik, ezzel elrontva a mérést.

A harmadik módszert gyakran alkalmazzák, például LMS alapú zajcsökkentő eljárásokban is. Ennek ellenére mi egy negyedik, rezonátoros struktúra alapú átviteli függvény mérést alkalmaztunk. Azért választottuk ezt a struktúrát, mert zajelnyomásra rezonátoros struktúrát használunk, ezért maga a rezonátoros struktúra már adott, így egy nagyon kis módosítással megkaphattuk belőle az átviteli függvény mérésére alkalmas elrendezést. Ezen kívül a keskeny sávú (szinuszos) gerjesztés nagyon pontos mérést tesz lehetővé, és az átlagolás is könnyen megvalósítható. Ennek a mérésnek a blokkvázlatát a következő ábra szemlélteti:



5.6. ábra. Átviteli függvény mérése rezonátoros struktúrával

A méréshez az integrátoros jelmodellt használjuk, mivel ekkor állandósult állapotban az állapotváltozók maguk a Fourier-együtthatók. A bemutatott struktúrával az a probléma, hogy egyszerre csak N frekvencián tudja megmérni az átvitelt. Ennek a hibának a kiküszöbölésére a következő megoldást javasoljuk. Egyetlen AC rezonátorpárt kell működtetni, s az akusztikus rendszert egy sweepelő szinuszjellel kell gerjeszteni. A szinusznak annyi frekvencián kell végigugrálnia, ahány pontban az átvitelt meghatározni szeretnénk. Az esetlegesen előforduló DC offset miatt célszerű egy DC rezonátort is működtetni. Ebben az esetben a rezonátorok elhelyezkedése nem egyenletes, de ez a gyakorlatban nem okoz jelentős sebességnövekedést. Többcsatornás esetben a mikrofonok számával megegyező független AC rezonátorpárra van szükség, és a szinusszal való gerjesztést meg kell ismételni mindegyik beavatkozó hangszóróból. A mérési eredményeket megfelelően rendezve megkapható az átvitelifüggvény-mátrix.

Többcsatornás zajelnyomás esetén az átviteli függvény mátrix inverzét kiszámítva meghatározható a bemeneti analizátor-készletek és a kimeneti rezonátor-készletek közötti átcsatolást végző **W** mátrix. Az α konvergenciaparaméter állításával pedig a zajcsökkentő rendszer stabilitása biztosítható.

6. On-line identifikáció rezonátoros struktúránál

6.1. Bevezetés

A rezonátoros struktúrával való aktív zajcsökkentés esetén nagy szerepe van az átviteli függvény mátrix off-line identifikálásának, hiszen ennek alapján határozható meg az inverz átviteli függvény mátrix, aminek alkalmazásával a rezonátoros struktúra visszacsatoló ágában $A(z) = 1$ értékű visszacsatolás biztosítható a rezonátorfrekvenciákon. Sajnos az átviteli függvény időben változhat. A hőmérséklet vagy a páratartalom változása, egy szobában különböző tárgyak elmozdulása mind-mind befolyásolják a hibamikrofonok és a beavatkozó hangszórók közti akusztikus átvitelt. Ha ez a változás olyan mértékű, hogy az off-line meghatározott és a zajcsökkentésben alkalmazott inverz átviteli függvény mátrix ($\mathbf{W}(z)$) már nem teljesíti a stabilitásra vonatkozó, 5.8. egyenletben megfogalmazott feltételt, akkor a rendszer instabillá válik. Ha valamilyen módon biztosítani tudnánk azt, hogy $\mathbf{W}(z)$ megváltoztatásával az 5.8. egyenlet teljesüljön, akkor a rendszer továbbra is stabil maradna. Ha tehát az átviteli függvényt folyamatosan, on-line identifikálni tudnánk, akkor ez a fenti probléma megoldását jelentené és egy stabil, robosztus zajcsökkentő rendszert kapnánk. Ez tehát egy még nem megoldott probléma, ebben a fejezetben ismertetjük az általunk egycsatornás rendszerre kidolgozott megoldást. Léteznek más megoldások az on-line identifikációra [15], de ezek általában LMS alapú, szélessávú zajelnyomó rendszerek esetén alkalmazhatóak. A probléma ezekkel az eljárásokkal az, hogy úgy identifikálják az átviteli függvényt, hogy bizonyos időközönként szélessávú zajt kevernek a kimenethez. Ez azért nem megfelelő megoldás, mert az átviteli függvény méréséhez használt gerjesztőjel alkalmazása a hibamikrofonnál hallható zajt hoz létre, amivel a zajelnyomást rontjuk el. A gerjesztés alkalmazásának elkerülésére dolgoztuk ki egycsatornás rendszerre az általunk on-line identifikációs algoritmusnak nevezett eljárást. Ez a módszer nem alkalmaz gerjesztést, de tulajdonképpen nem is identifikációs eljárás. Az általunk kidolgozott algoritmus azt figyeli, hogy a rendszer mikor válik instabillá, ekkor beavatkozik, és a becsatoló együtthatók fázisának forgatásával teszi a rendszert újra stabillá. Ha azonban a rendszer stabil, akkor az on-line algoritmusunk nem avatkozik be, nem úgy mint más módszerek esetén, amelyek rendszeresen gerjesztést visznek a rendszerbe, elrontva ezzel a zajelnyomást. Ezzel az átviteli függvény abszolút értékének változását nem küszöböljük ki, de a gyakorlatban ez csupán a rendszer sebességét változtatja meg egy kis mértékben. Ezzel a módszerrel tehát a rendszer stabilitása és ezáltal a zajelnyomás biztosítható, célunk pedig nem az átviteli függvény pontos ismerete, hanem a zajcsökkentés biztosítása. A fentiekből következően a hibajavító beavatkozás volt az egyszerűbb dolog, a nagyobb feladat a rendszer instabilitásának detektálása volt. Ráadásul ezt néhány tized másodpercen belül kellett észrevenni, mivel megközelítőleg ennyi az

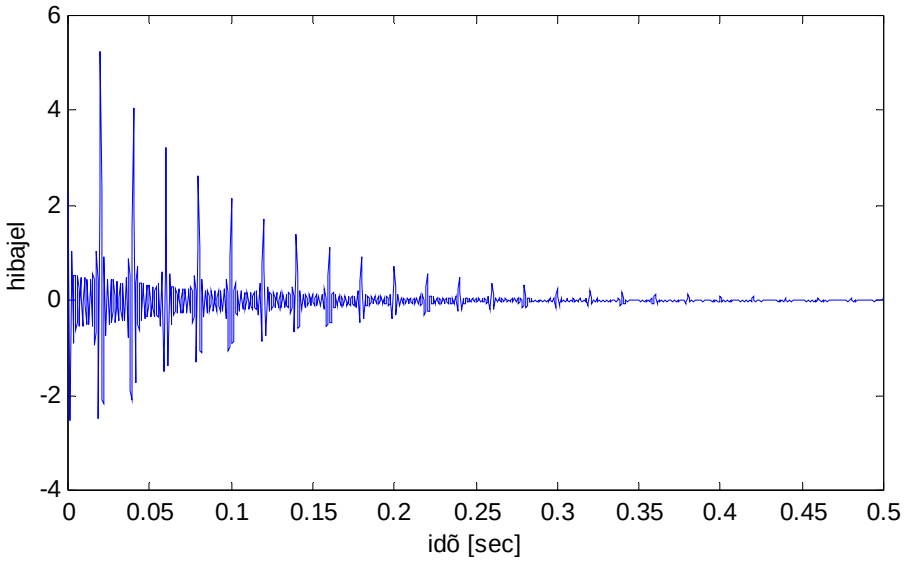
az idő, amíg egy DSP-n megvalósított rendszerben túlcsordulás következik be, és ha a rendszer már túlcsordult, akkor már nem lehet megállapítani, hogy melyik w_i együttható okozta a problémát.

6.2. On-line identifikáció egycsatornás rendszerre

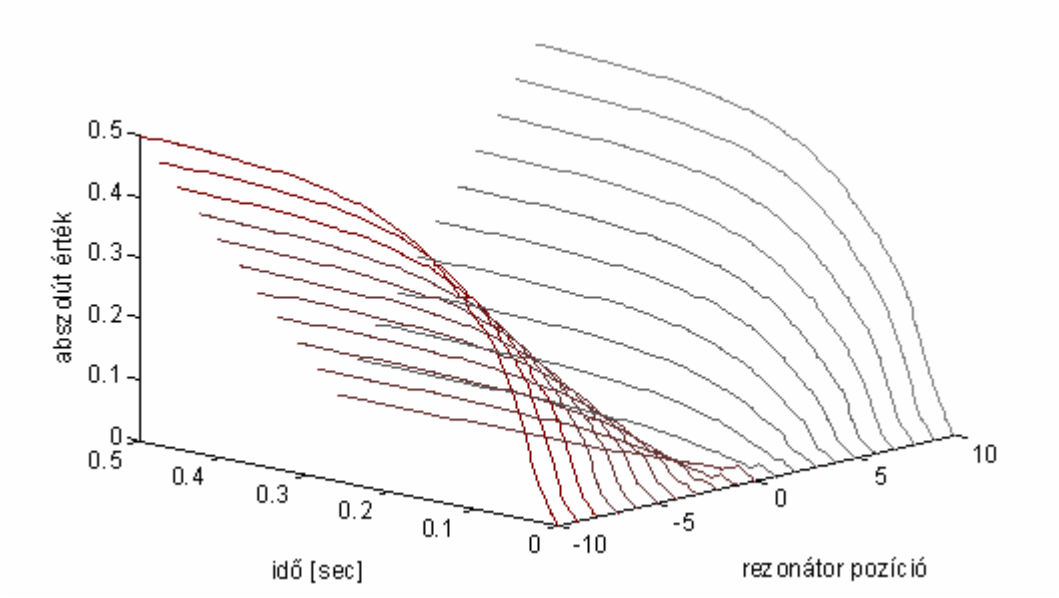
Alapötletünk az volt, hogy ha egy egycsatornás rendszer instabil, akkor az azt jelenti, hogy a rendszer valamely w_i becsatoló együtthatói nem teljesítik az 5.4. egyenletben megfogalmazott feltételt. Ekkor úgy tehetjük stabillá a rendszert, hogy az instabilitást okozó w_i együtthatók fázisát 180º-kal elforgatjuk.

Ahhoz, hogy ezt a beavatkozást elvégezhessük, meg kell tudnunk állapítani, hogy a zajcsökkentő rendszer instabil-e, és ha igen, akkor mely becsatoló együtthatók okozzák ezt az állapotot. A beavatkozási módszer tehát adott volt, ezután egy megfelelő algoritmust kellett találnunk arra, hogy miként állapítsuk meg azt, hogy mely becsatoló együtthatók okozzák a rendszer instabilitását.

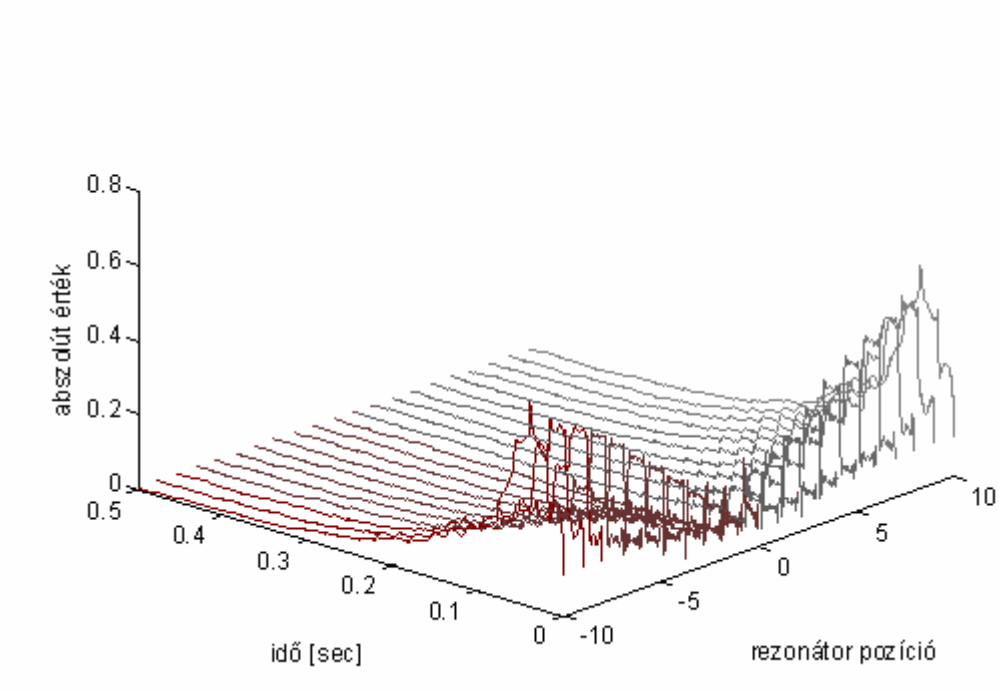
A probléma megoldására MATLAB-ban szimuláltuk egy egycsatornás zajcsökkentő rendszer viselkedését. A 6.1. ábra egy stabil rendszer működését mutatja:



6.1. ábra. Hibajel alakulása az idő függvényében, stabil rendszer esetén



6.2. ábra. Kimeneti rezonátorkészlet tárolók abszolút értéke az idő függvényében, stabil rendszer esetén



6.3. ábra. A hibajelet Fourier-komponensekre bontó rezonátorkészlet rezonátorai abszolút értékének alakulása az idő függvényében, stabil rendszer esetén

A rendszerben a mintavételi frekvencia 2 kHz. A szimuláció során nem alkalmaztunk adaptív Fourier-analízist, az elnyomandó jelet úgy generáltuk, hogy annak komponensei a rezonátor-frekvenciákra essenek. A rendszerben 21 rezonátor van, ebből 10 pár AC, és 1 DC rezonátor. Ez igaz mind az analizátor-készletre, mind a kimeneti rezonátor-készletre. A rezonátorok 50 Hz-enként helyezkednek el. Az akusztikus átvitelt egy másodfokú IIR szűrővel helyettesítettük, melynek átviteli függvényének nagy dinamikája miatt nagyon jó zajcsökkentő rendszerek tesztelésére. Átviteli függvénye a következő:

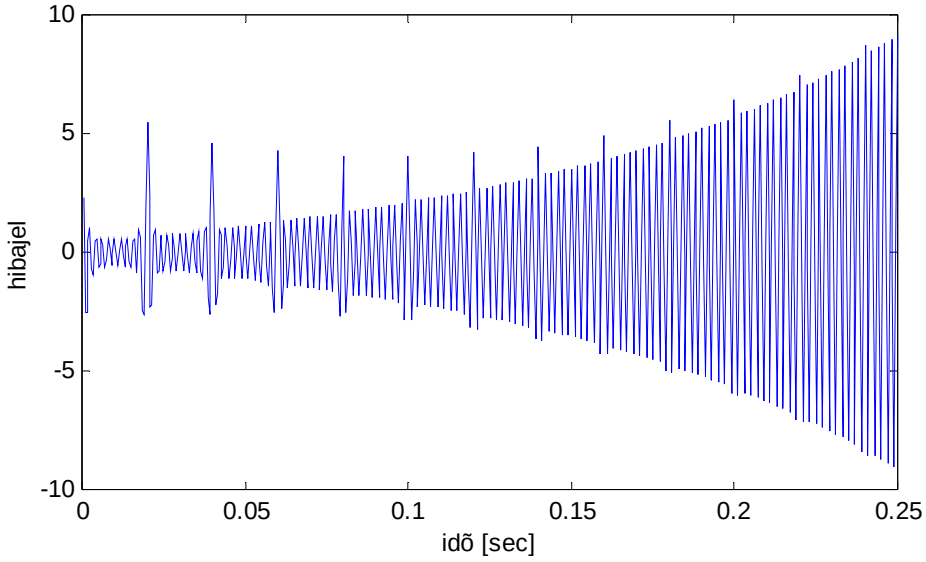
$$[z^2 - 0{,}4164z + 1{,}2346z]/[z^2 + 0{,}6627z + 0{,}6414] \tag{6.1}$$

Az elnyomandó jel a következő képlettel adott (ahol i=0...10):

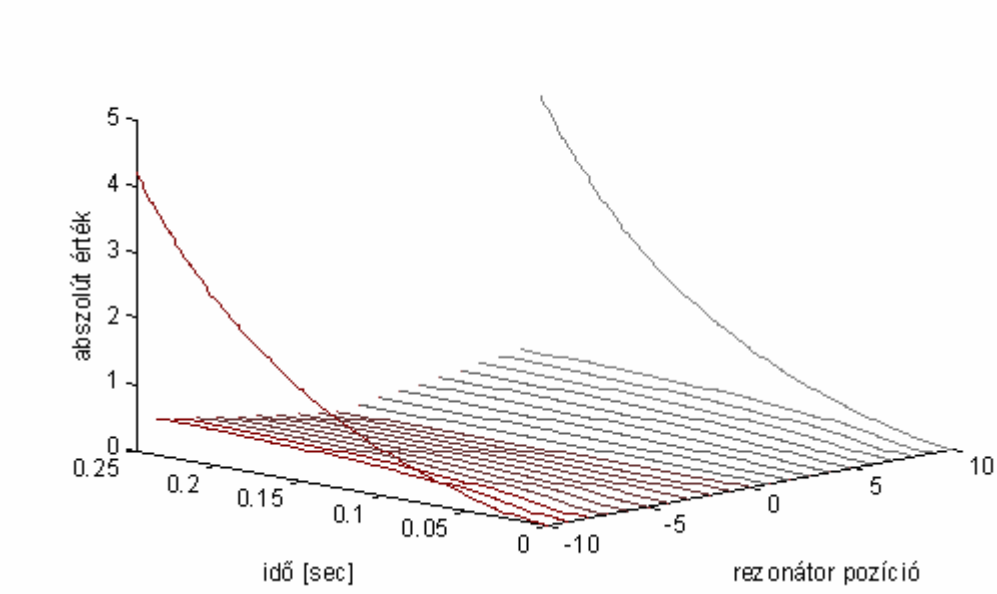
$$x(n)=\sum\cos(2^* \pi^*i^*50\ \text{Hz}^*n^*1/2000\ \text{Hz})^*i/10 + \text{gaussi fehér zaj}; \tag{6.2}$$

A 6.1-es ábra a hibamikrofon jelének alakulását mutatja az idő függvényében. Az ábrán jól látható, ahogy a hibajel exponenciálisan lecsökken, csak az elnyomandó jel mellé kevert zaj marad a kimeneten. A 6.2-es ábra a kimeneti rezonátorkészlet rezonátorainak abszolút értékének alakulását mutatja az idő függvényében, az összes rezonátorra. Látszik, hogy a rezonátorok rövid idő alatt beállnak egy konstans értékre, amit a továbbiakban tartanak. A 6.3-es ábra a hibajelet Fourier-komponensekre bontó rezonátorkészlet rezonátorai abszolút értékének alakulását mutatja az idő függvényében, az összes rezonátorra. Az ábráról leolvasható, hogy az analizátor-készlet összes rezonátora rövid idő alatt beáll a 0 értékre, azaz a hiba minimalizálódik.

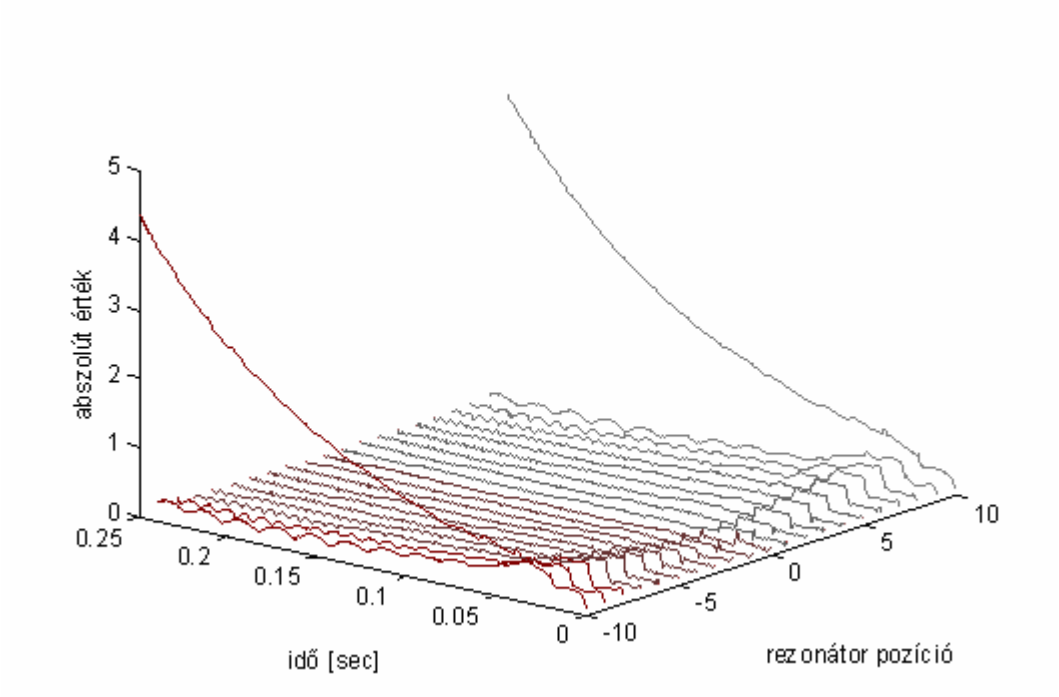
A következő ábrákon ugyanazokat ábrázoltuk, mint az előbbieken, azzal a különbséggel, hogy az f = 500 Hz-en működő rezonátorpár becsatoló együtthatóit „elrontottuk”, elforgattuk 180º-al. Ez ekvivalens azzal, mintha csak f = 500 Hz-en változott volna meg az akusztikus átvitel, méghozzá úgy, hogy az átviteli függvény fázisa f = 500 Hz-en 180º-ot fordult. A szimulációs eredmények a következők:



6.4. ábra. Hibajel alakulása az idő függvényében, instabil rendszer esetén



6.5. ábra. Kimeneti rezonátorkészlet tárolók abszolút értéke az idő függvényében, instabil rendszer esetén



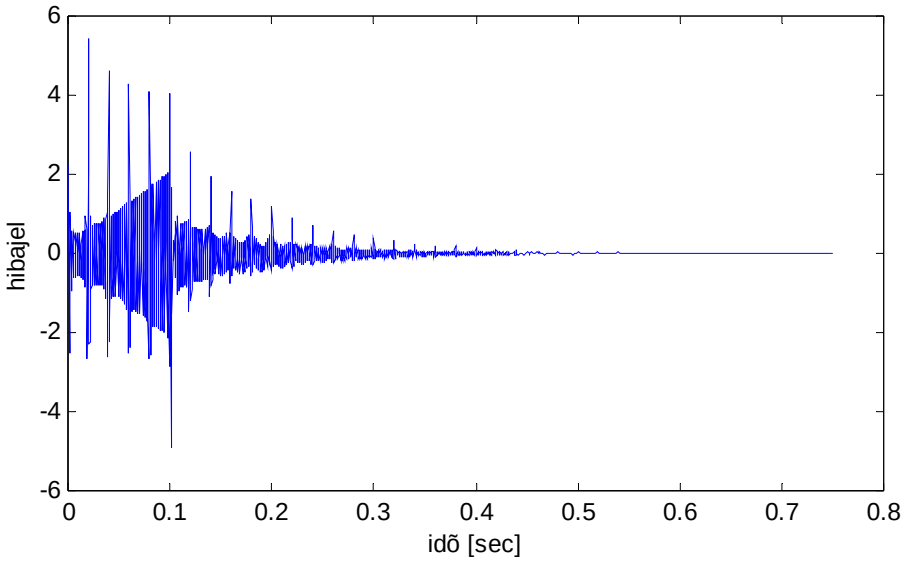
6.6. ábra. A hibajelet Fourier-komponensekre bontó rezonátorkészlet rezonátorai abszolút értékének alakulása az idő függvényében, instabil rendszer esetén

Az ábrákon jól látható, hogy a legnagyobb frekvenciás rezonátorpár tárolóinak abszolút értéke mind a bemeneti rezonátorkészletben, mind a kimeneti rezonátorkészletben exponenciális növekedésnek indul. Ezzel együtt a hibajel is exponenciálisan növekszik.

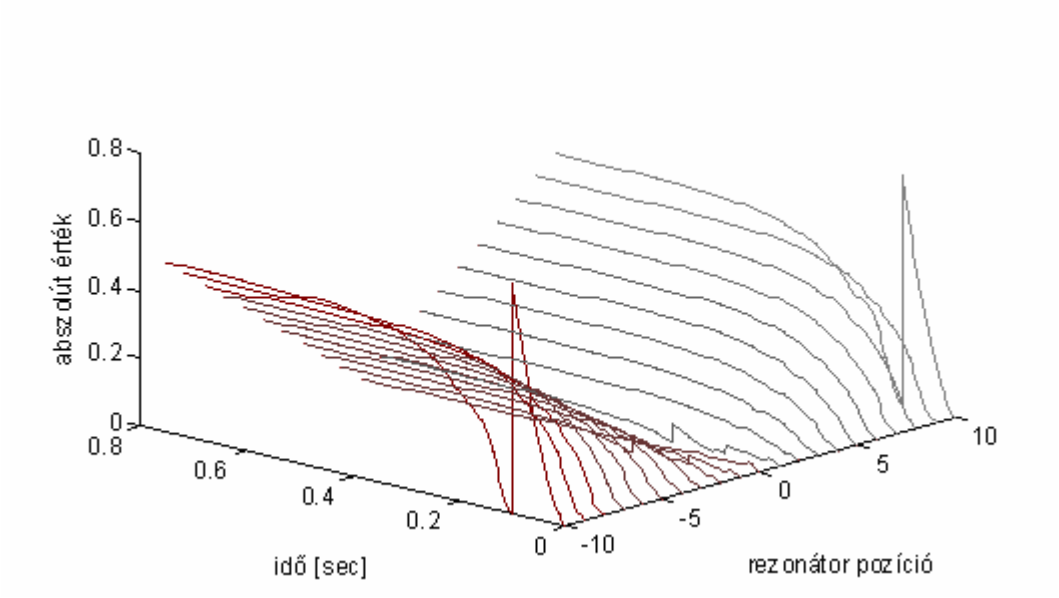
A stabil és az instabil állapot közötti különbséget vizsgálva azt vettük észre, hogy ha a rendszer stabil, akkor a kimeneti rezonátorkészlet tárolóinak abszolút értéke kezdetben nagyon gyorsan változik, majd később időegység alatt egyre kevesebbet. Ez azt jelenti, hogy stabil rendszer esetén a kimeneti rezonátor-készlet tárolói abszolút értékének deriváltja az idő függvényében csökken. Ellenben, ha a rendszer instabil, akkor a kimeneti rezonátorkészlet tárolóinak abszolút értéke exponenciálisan növekszik, azaz időegység alatt egyre többet változik, tehát deriváltja növekvő. Ezt a jelenséget használtuk fel a rendszer instabilitásának detektálására. Fontos megjegyeznünk, hogy egycsatornás esetben a bemeneti analizátor-készlet tárolóinak abszolút értéke a kimeneti rezonátor-készlet tárolói abszolút értékének a deriváltjával egyenesen arányos.

Hibadetektáló algoritmusunk tehát az, hogy figyeljük a kimeneti rezonátorkészlet tárolóinak deriváltját, és ha ez növekvő, akkor beavatkozunk és az adott w_i becsatoló együttható fázisát elforgatjuk 180º-kal. Előfordulhat azonban, hogy az instabil rendszerből a megfelelő w_i együtthatón végrehajtott 180º-os fázisforgatás instabil rendszert hoz létre. Ennek elkerülésére, ha az

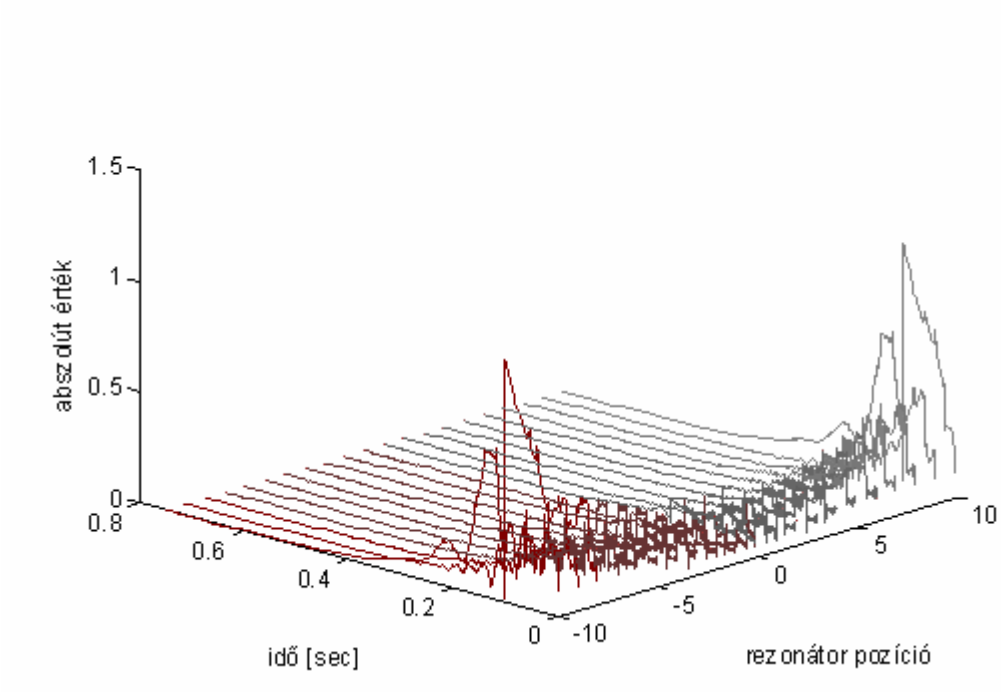
algoritmus azt észleli, hogy a rendszer nem stabil, akkor először 180º-kal elfogatja a megfelelő w_i együttható fázisát. Ha az így kapott rendszer instabil, akkor ugyannak a w_i együtthatónak a fázisát 20º-kal forgatja el. A 180º- és 20º-os fázisforgatást ugyanarra a w_i együtthatóra felváltva alkalmazza az algoritmus. a tapasztalat azt mutatta, hogy a stabil állapotot a rendszer a fent említett két forgatás változtatásával éri el a leggyorsabban. A gyakorlati alkalmazásokban szükség van arra is, hogy egy ilyen fázisforgató beavatkozás után az adott állapotváltozót nullázzuk, mivel a beavatkozás hatása csak néhány tized másodperc alatt jelentkezik, ami idő elég arra, hogy egy valós rendszerben túlcsordulást okozzon. Ezért kell minden fázisforgató beavatkozás után az adott állapotváltozót nullázni. Az algoritmus működését a következő három ábrán mutatjuk be. A helyzet ugyanaz, mint az előző esetben, azaz az f = 500 Hz-en működő rezonátorpár becsatoló együtthatói rosszak.



6.7. ábra. Hibajel alakulása az idő függvényében, instabil, de javított rendszer esetén



6.8. ábra. Kimeneti rezonátorkészlet tárolók abszolút értéke az idő függvényében, instabil, de javított rendszer esetén



6.6. ábra. A hibajelet Fourier-komponensekre bontó rezonátorkészlet rezonátorai abszolút értékének alakulása az idő függvényében, instabil, de javított rendszer esetén

Az ábrákon jól látható, hogy a hiba egy ideig nő, ugyanígy nő exponenciálisan az f = 500 Hz-en működő rezonátorpár tárolóinak abszolútértéke mind a kimeneti rezonátor-készletben, mind a bemeneti analizátor-készletben. Majd a hibadetektáló algoritmus észreveszi, hogy a rendszer instabil és beavatkozik. Az eredmény: a beavatkozást követően a tárolók beállnak egy konstans értékre, a hibajel lecsökken, a rendszer stabillá vált.

A fentiekben leírt módszer természetesen nem identifikációs módszer, az on-line identifikáció címet csak képletesen használtuk.

7. Adatgyűjtő hálózat felépítése

A fent bemutatott aktív zajcsökkentő rendszerekben különlegességet jelent, hogy a hibamikrofonok jeleit rádiós hálózaton keresztül továbbítjuk. A hálózat a bevezetőben az 1.3. ábrán vázolt általános felépítést követi. Ez azt jelenti, hogy a hálózatban lévő mote-ok által összegyűjtött adatokat rádión keresztül továbbítják egy átjáróként szolgáló mote felé, mely a fogadott csomagokat soros porton továbbítja a DSP felé. Bár itt már nem a TinyOS által szolgáltatott kommunikációs lehetőséget használjuk, de erről még később részletesebben beszélünk. Már azt is említettük, hogy ennek a hálózatnak a szokásostól eltérően on-line kell szolgáltatni az összegyűjtött adatokat, ráadásul minél nagyobb sebességgel. A tervezés minden lépését ennek szellemében kell végezni.

A hálózattal kapcsolatban számos megoldandó feladat volt: üzenetformátum meghatározása, topológia kialakítása, csomagvesztés kezelése.

A feladatok megoldásánál arra kellett törekedni, hogy minél jobban ki tudjuk használni a hálózat maximális áteresztőképességét, és biztosítanunk kellett, hogy a csomagok adott időn belül megérkezzenek az átjáróhoz. Ez a hálózatot használó zajcsökkentő rendszer miatt szükséges. Egyrészt ugyanis minél több adatot juttatunk át a hálózaton adott méretű hálózat esetén, nyilván annál nagyobb a mintavételi frekvencia az egyes mote-okon, ami növeli a zajcsökkentő rendszer felhasználási tartományát. Másrészt viszont nem engedhető meg az, hogy az adatok váltakozó időközönként érkezzenek meg, ugyanis ez azt eredményezné, hogy változna a mikrofon és a fogadó állomás közti késleltetés, ami végeredményben a visszacsatoló ág átviteli függvényének megváltozásához, így instabilitáshoz vezet.

A hálózatban továbbított üzenetekben kézenfekvőnek tűnt, hogy a lehető legtöbb adatot továbbítsunk egy csomagon belül, ami 29 byte. Ezzel érhetjük el, hogy optimálisan használjuk ki a csatornát, ugyanis így a legkisebb a számunkra „haszontalan” keret mérete az átvitt hasznos adat mellett. A csomagban azonban nem csak az összegyűjtött adatokat továbbítjuk. A biztonság és a hálózat kialakítása céljából az üzenetben elküldjük az aktuális csomag sorszámát, melyet folytonosan számozunk, valamint a saját azonosítónkat is. Ezek után az összegyűjtött adatok számára 25 bájtnyi hely áll rendelkezésre, tehát 25 minta összegyűjtése után továbbítunk egy csomagnyi adatot. Ez azt jelenti, hogy a rádión továbbított bájtokból 25/40*100%=62.5% a hasznos.

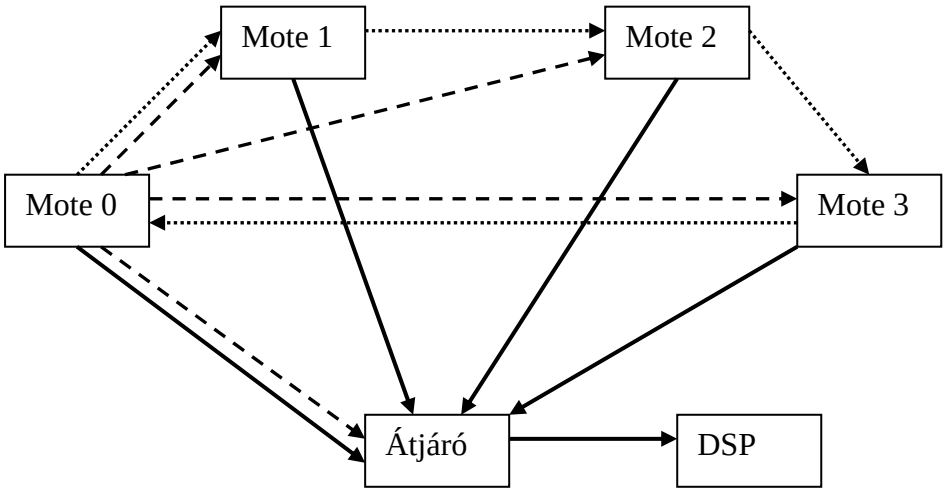
A csomag sorszámát azért kell elküldeni, mert így észlelhetjük, ha az adott mote-tól nem kapunk meg minden csomagot, ugyanis ekkor a fogadó állomáshoz érkező csomagok sorszáma nem folytonosan követi egymást.

A saját azonosító elküldése a hálózat kialakítása miatt fontos, ugyanis az általunk tervezett rendszerben tudniuk kell a mote-oknak, hogy mely mote küldte az adott üzenetet. Ez nyilvánvalóvá válik a hálózati működés ismertetése után.

A hálózat ugyanis gyűrűszerű topológiát követ, ez a 7.1. ábrán látható. Az egyes nyíltípusok a mote-ok közti különböző logikai kapcsolatokat jelölik. Ezeket a kapcsolatokat természetesen üzenetek valósítják meg, bár a különböző

kapcsolatokhoz nem rendelünk külön üzenettípusokat, hanem az adatok továbbítására szolgáló üzenetet használjuk fel több célra.
A nyíltípusok jelentése a következő:

- Folytonos vonal: adattovábbítás
- Pontozott vonal: adatküldés engedélyezése, a gyűrűs hálózat kialakításához kell
- Szaggatott vonal: szinkronizáció, erről később lesz szó



7.1. ábra. A hálózat felépítése

A hálózat a következőképpen működik. Az egyes mote-ok egymás után megfelelő sorrendben továbbítják az adatokat. Egy adott mote akkor küldheti el az aktuális csomagot, amennyiben a szomszédként kijelölt mote csomagja megérkezett. Szomszédnak egy csomópont azt a mote-ot tekinti, amelyik sorszáma eggyel kisebb, mint a sajátja, illetve a legkisebb sorszámú mote esetén a legnagyobb sorszámú mote lesz a szomszéd. Az ábrán a pontozott vonal köti össze a szomszédokat. Az, hogy hány mote van a hálózatban, és hogy melyik mote-nak mi az azonosítója, az programozáskor derül ki. Azonban nem kell minden mote-hoz külön programot írni, amiben szerepelnek ezek az adatok, ugyanis a fejlesztőkörnyezet lehetőséget kínál arra, hogy programozáskor a mote-oknak átadjunk egy paramétert. Ez a paraméter jelen esetben a következőképpen számítható: paraméter=[azonosító + 8*(hálózat elemeinek száma-1)].
Ebből a paraméterből visszanyerhető az azonosító és a hálózat mérete.

Állandósult állapotban tehát az adás joga folytonosan körbe fordul a hálózatban, ugyanúgy, mint a token-ring hálózatokban, és az átjáró mote-hoz sorrendben érkeznek meg a mote-októl az összetartozó csomagok.

Ez a hálózat ebben a formában azonban még nem lenne elég biztonságos, ugyanis egy csomag elvesztésekor megszakadna a lánc, és ezzel teljesen leállna a

hálózat, ami az egész rendszer működésképtelenségéhez vezetne. Erre a következő megoldás született. Az adás joga nem csak akkor kerül egy adott mote-hoz, ha már megkapta a szomszédja által küldött csomagot, hanem az utolsó csomag elküldése óta eltelt bizonyos idő múlva minden esetben. Ez azért lehetséges, mivel a mintavételi frekvenciát, és ezzel együtt a csomagküldési frekvenciát úgy választjuk meg, hogy mire egy mote-nál az utolsó csomag elküldése után összegyűlt egy csomagnyi adat, azaz 25 minta, akkorra újra rákerüljön az adás joga. Ha ez nem teljesül normális körülmények között, akkor a hálózat nem működőképes, és csökkenteni kell a mintavételi frekvenciát, ugyanis a hálózat átbocsátó képessége nem elég a keletkezett adatok továbbítására. Ezek szerint normális működés esetén található olyan felső korlát, amennyin belül minden esetben meg kell kapni az adás jogát. Tehát ezen idő leteltével biztosnak vehetjük, hogy már visszakerült az adás joga, csupán a csomagvesztés miatt ezt nem észleltük. Ez a korlát 25 mintányi időköz, de a biztonság kedvéért ezt kicsit nagyobbra választjuk, arra az esetre, ha valami miatt később érkezne be egy csomag.

A mintavételezési folyamat indítása azonban némi átgondolást igényel. Az eszközök bekapcsolásakor ugyanis egyből elkezd futni a rajtuk lévő program, és nyilvánvalóan nem lehet az összes mote-ot egyszerre bekapcsolni. Ez viszont nem kiszámítható állapothoz vezet, amennyiben nem teremtünk valami start feltételt a mintavételezést illetően. Ha ugyanis a mote-ok a bekapcsolásuk után egyből elkezdenék gyűjteni a mintákat, akkor a várakozási idő letelte után minden mote úgy vélné, hogy rajta van az adás joga, és elkezdenék az adást, ami miatt nem biztos, hogy beállna a hálózatban a normális állapot. Ennek elkerülésére a következő részletet építettük be a programba: amennyiben bekapcsolunk egy mote-ot, és annak azonosítója nem nulla, akkor nem kezdheti el az adását. Ha viszont az azonosító nulla, akkor a mote egyből elkezdi a mintavételezést, és egy csomag összegyűlte után elküldi az első üzenetet, ami pedig indítja a többi mote-on is a mintavételezést, és ezzel együtt kontrollált módon beindul a hálózat működése. Ez az algoritmus nyilván azzal jár, hogy a nulladik sorszámú mote-ot kell utolsóként bekapcsolni.

Felmerülhet a kérdés, hogy mi történik azokkal a csomagokkal, amelyek valamilyen okból kifolyóan nem érkeznek meg. Ezekkel a csomagokkal ebben az elrendezésben nem tudunk mit kezdeni, ugyanis a hálózatba nem építettünk be semmi nyugtázást arról, hogy egy üzenet megérkezett-e az átjáróhoz, a küldőnek viszont semmi információja nincs arról, hogy mi lesz az elküldött üzenet sorsa. A nyugtázás két ok miatt nem került kiépítésre.

A megerősítés ugyanis egy átjáró→mote irányú kapcsolat kialakítását is szükségessé tenné, ugyanis ebben az esetben nem használható az eddigi felfogás, miszerint az adatokkal együtt bizonyos hálózattal kapcsolatos paramétert is elküldünk, ugyanis az átjáróról semmilyen rádiós üzenetet nem küldünk. Ez tehát növelné a hálózat terheltségét, így csökkentené a hálózat hasznos adatátviteli képességét.

Másrészt, ha esetleg lenne is nyugtázás, akkor a csomag újbóli elküldése esetén bizonytalanná válna a hálózat terheltsége, és egy csomag többszöri ismétlésével megdőlhetne az az alapfeltevés, hogy egy csomagnyi minta összegyűlte után már visszakerül az adási jog egy adott mote-hoz, ugyanis adott idő alatt több adatot kellene átvinni ugyanazon csatornán, ami nem biztosítható minden esetben.

Ez a kissé bizonytalan adatátvitelt megvalósító megoldás, miszerint nem számít, ha az adatok egy töredéke elveszik, nem egyedi. Számos olyan alkalmazás van, ahol hasonló elvet követnek, miszerint inkább egy nagyobb sebességű, de kisebb megbízhatóságú csatornát használnak, mert az adatátviteli sebesség fontosabb, mint a biztonságos adatátvitel. Erre az a magyarázat, hogy egy kiesett csomag miatt az adatátvitelben bekövetkező megtorpanás nagyobb gondot okoz, mint a csomag elvesztése. Ilyen alkalmazás például az Interneten történő hangátvitel (ismert nevén Voice over IP), ami végeredményben igen hasonló a mi alkalmazásunkhoz. A zajcsökkentő rendszerben ugyanis, ha a kiesett csomag miatt valami helytelen adatot adunk át az algoritmusnak, azt úgy foghatjuk fel, mintha a bemeneten valami hirtelen keletkező zavar jelenne meg, ami egy pillanatra ugyan kizökkenti a rendszert az állandósult állapotból, de szabályozási rendszerről lévén szó ez egy tranziens szakaszt követően visszaáll az eredeti állapotába. Ha ez a hiba nem fordul elő túl gyakran akkor nem jelent nagy problémát. A szabályozás lassításával egyébként is csökkenthetjük a rövid tranziens szakasz hatását, ugyanis akkor a rendszer kisebb mértékben esik ki az állandósult állapotból.

A hálózat kialakításával kapcsolatban még meg kell említeni a MAC réteg konfigurálását. (lsd.: A NesC nyelv és a TinyOS bemutatása című fejezet kommunikációt bemutató része). Ebben az esetben az `initialBackoff` értékét állandóra választjuk. Ennek oka, hogy ebben a hálózatban a közeghozzáférés determinisztikus, tehát elvileg nem adhat egyszerre két állomás. A véletlenszerű `initialBackoff` értéket – mely alapbeállításként szerepel – azonban véletlen hozzáférésű hálózatokban alkalmazzák. Ez ebben az esetben szükségtelen várakozáshoz vezethet, amely a hálózat kihasználtsága miatt kedvezőtlen lehet, másrészt azt eredményezi, hogy az elküldés után az üzenet nem meghatározható idő alatt érkezik meg a címzettekhez. Ez nagy gondot okozhat a később bemutatásra kerülő szinkronizációs folyamat során.

8. Mintavételezés és időzítés megoldása

A munka során az elsőként elvégzendő, és fizikailag is megvalósítandó feladat a mikrofon kezelésének elsajátítása, és a mintavétel időzítésének elvégzése volt.

A mikrofon kezelése gyakorlatilag az inicializáló parancs meghívásából áll, mely elvégzi a megfelelő felkonfigurálást. Ez tehát nem okoz nehézséget. A mikrofon által kiadott jel lekérdezése a `MicC` modul `ADC` interfészén lehetséges. A mintavételt indító parancs kiadása után a `MicC ADC` interfésze egy `dataReady` esemény formájában tájékoztat az átalakítás végeredményéről. Az így kapott digitális érték 10 bites, mi azonban csupán a felső 8 bitet továbbítjuk. Miután összegyűlt 25 mintányi adat, ezeket pufferelјük, és a hálózati részben leírtak alapján továbbítjuk, valamint eközben természetesen tovább folytatódik a mintavételezési folyamat.

Megjegyzésként megemlítjük, hogy elvi lehetőségként felmerült az az alternatíva is, hogy ne a nyers mintákat továbbítsuk a hálózatban. Ehelyett, mivel a zajcsökkentő rendszert alapvetően periodikus jel elnyomására terveztük, elegendő lenne az érzékelt jel frekvenciáját, alap- és felharmonikusainak amplitúdóját és fázisát továbbítani. Ehhez azonban futtatni kellene a mikrokontrolleren egy adaptív Fourier analizátoros struktúrát, amihez nem elegendő ezen eszközök teljesítménye, ráadásul a hálózat és mintavételezés kezelése is sok időt foglal le.

Az első komoly probléma a mintavétel időzítése volt, ugyanis megfelelően gyors és pontos időzítési módot kellett kialakítani.

Elsőként a TinyOS bemutatásakor ismertetett `TimerC` komponenst használtuk. Az időzítés folyamata a következő: az inicializálás során a komponens `Timer` interfészén keresztül beállítjuk a mintavételi frekvenciát. Az időzítő ilyen ütemben generál eseményeket, melyek segítségével újabb mintavételt indítunk az AD-átalakítón.

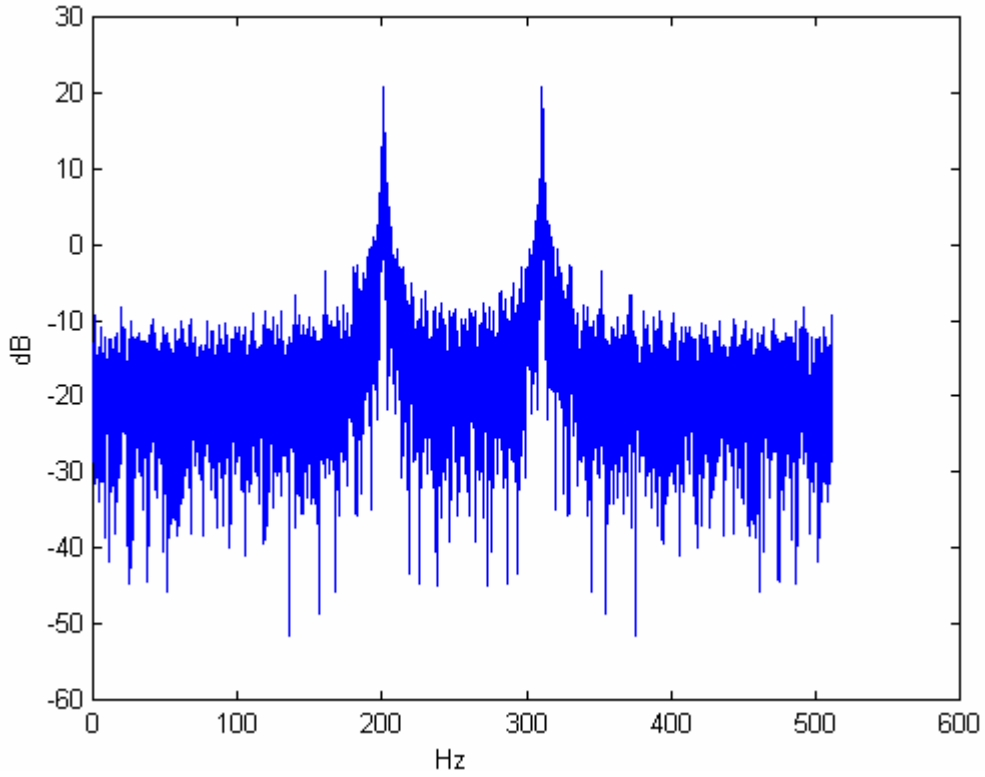
Hamarosan azonban kiderült, hogy ez a modul nem alkalmas a feladat ellátására. Ennek okait már a komponens ismertetésekor említettük, nevezetesen nem elég gyors, durva skálázású, és nem elég pontos. Ezek a hátrányos tulajdonságok a következők.

A legnagyobb mintavételi frekvencia 1024 Hz, mely a mintavételi tétel értelmében elvileg elég lehet egy néhány száz Hz-es periodikus jel alapharmonikusának és egy-két felharmonikusának helyes észlelésére, és így a zajcsökkentő rendszer adattal való ellátására. Ha azonban magasabb mintavételi frekvenciát szeretnénk használni, akkor ezzel az időzítővel ez nem lehetséges. Ha viszont a hálózatban viszonylag sok mote-ot helyeznénk el, és így az 1024 Hz-es mintavételi frekvencia is soknak bizonyulna, akkor a következő lehetséges mintavételi frekvencia 1024 Hz / 2=512 Hz lenne, ami már eléggé korlátozná a működési tartományt. Ez a durva skálázhatóság ráadásul egy később megoldandó szinkronizációs feladatban is komoly problémát jelentene.

A következő hátrányos tulajdonság az, hogy a mintavétel egyenetlen időzítése miatt nem lehet egyenletes mintavételezést megvalósítani. Ennek oka, hogy az időzítő jelentős szoftveres támogatást használ, így nagyban befolyásolják a mikrokontrolleren futó egyéb folyamatok, és például késleltethetik az események generálását. Ez a jelenség tükröződik a felvételek spektrumképén is.

Itt jegyezzük meg, hogy az adatátviteli rendszer tesztelésére a fejlesztés során szinuszos gerjesztést alkalmaztunk. Ez egyrészt lehetővé tette a minták spektrumának egyszerű kiértékelését, másrészt a rendszert egyébként is periodikus jelek továbbítására terveztük.

Visszatérve előző gondolatunkhoz, a `TimerC` komponens időzítésével előállított regisztrátum spektruma látható a 8.1. ábrán.

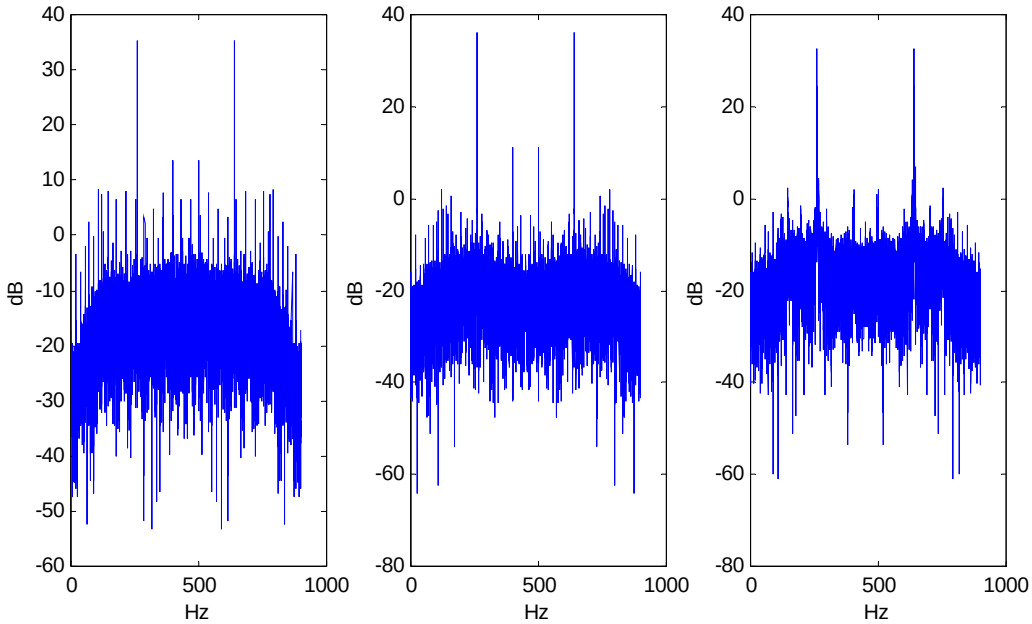


8.1. ábra. TimerC-vel időzített mintavételezés.
200 Hz-es szinuszjel 512 Hz-es mintavételi frekvenciával

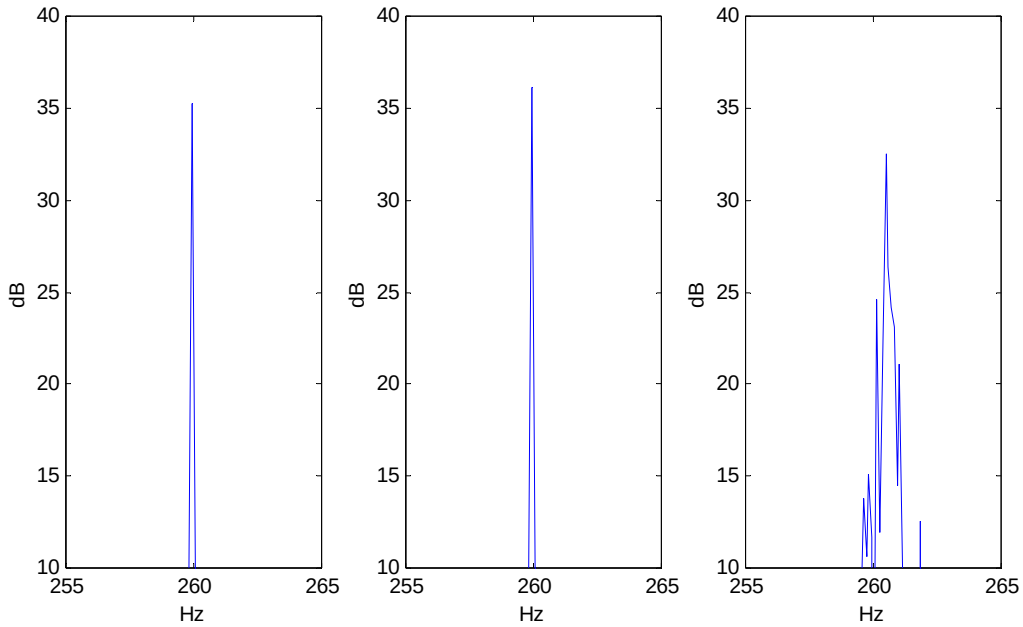
Megállapíthatjuk, hogy a spektrum nem igazán felel meg a várakozásnak. Mivel a gerjesztőjel szinuszos, így egyetlen éles spekrtumvonalat kellene látnunk. Habár a koherens mintavételezés valószínűleg nem biztosított, ezért a picket fence jelenség miatt nyilván ez a spektrumvonal kiszélesedik, de a kiszélesedett spektrumvonalon jelentős zajszerű ingadozás is található, melyet valószínűleg a helytelen mintavételezés okoz.

A fenti hátrányok kiküszöbölésére kézenfekvőnek tűnt egy új időzítő komponens elkészítése. Ehhez először is fel kellett deríteni, hogy a programban felhasznált többi komponens mely időzítőket használja még. Azt tudjuk, hogy a Timer0-t a fent említett `TimerC` használja, így azt nem használhatjuk, bár mivel 8 bites időzítő, ezért egyébként sem jöhet számításba. A Timer2 időzítőt a rádiót kezelő komponens használja, így ez is kizárható. Marad tehát a Timer3 és Timer1 időzítő, melyek egyébként azonos paraméterekkel rendelkeznek. Ezek közül a Timer3-t választottuk. Ez egy 16 bites időzítő, mely tehát viszonylag finoman hangolható időzítést tesz lehetővé. Az időzítőt egyébként a mikrokontroller 7372800 Hz-es órajel-frekvenciájáról üzemeltetjük. Ezzel tehát, ha 2 kHz-es mintavételezést szeretnénk megvalósítani, akkor azt 7372800/2000≅3686 osztásviszonnyal érhetjük el, melyhez 100%/3686=0.027%-os pontosság tartozik. Ez azt is jelenti, hogy ilyen mértékben hangolható a mintavételi frekvencia a 2 kHz-es érték környezetében, ami már megfelelőnek mondható.

A Timer3 segítségével kialakított komponens is hasonlóképpen működik, mint a `TimerC`, tehát a program indításakor inicializáljuk, aztán a beprogramozott ütemben periodikusan eseményeket generál, mely segítségével AD-átalakítást indítunk. Ez a modul abból a szemszögből kedvezőbb, hogy közvetlenül a hardveres megszakítás generálja az időzítő eseményeket. Ezen komponens segítségével felvett szinuszos jel spektrumát ábrázoltuk a 8.2. és 8.3. ábrákon.



8.2. a,b,c ábra. Mintavételezett 260 Hz-es szinuszjel, fs= 900 Hz



8.3. a,b,c ábra. Mintavételezett 260 Hz-es szinuszjel alapharmonikusának felnagyított spektrumvonala

A 8.2.a. ábrán lévő felvételen látható, hogy a spektrumban igen sok spektrumvonal jelenik meg. Ennek oka szintén abban keresendő, hogy a mintavételezés nem pontosan triggerelt. De nézzük, hogy ebben az esetben miért jelennek meg a diszkrét spektrumvonalak. A `TimerC` alkalmazása esetén már eleve a mintavételt indító parancs kiadása is jelentős bizonytalansággal történt meg. Az új időzítő segítségével ezt a problémát kiküszöböltük, azonban a komponensek tanulmányozásával megállapíthatjuk, hogy az AD-átalakítás indítása több függvényhíváson keresztül indul el, melynek időpontját ezért még szintén befolyásolják a mikrokontrolleren futó folyamatok.

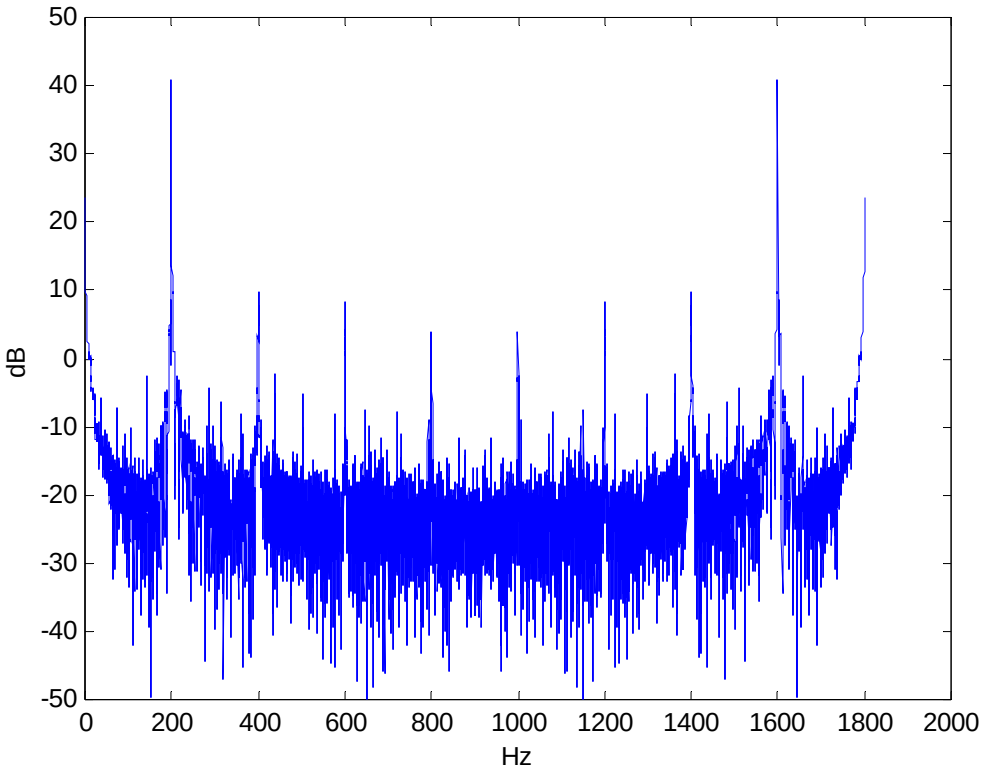
Az első esettel ellentétben itt azonban ez a zajszintet nem emelte meg a spektrumban, és az alapharmonikus helyén lévő spektrumvonal is sokkal élesebb (8.3.a. ábra), tehát az időzítő tökéletlenségén túl más hatás is érvényesül. A spektrumban lévő diszkrét vonalak valami periodikus folyamatról tanúskodnak. Ha részletesebben is tanulmányozzuk a spektrumképet, akkor megállapíthatjuk, hogy a spektrumvonalak, a felharmonikusoktól eltekintve, a mintavételi frekvencia huszonötöd részének egész számú többszöröseinél, illetve a mintavétel miatti belapolódásoknak megfelelő helyeken találhatók. Mivel a minták továbbítása 25 mintánként történik, ezért feltételezhetjük, hogy a rádió periodikus használata az annak megfelelő gyakorisággal torzítást visz a mintavételezési folyamatba, hiszen a rádiós IC által adott megszakítások késleltetik az AD-átalakítás indítását. Ennek a feltevésnek igazolására többféle megfigyelést is végeztünk. Egyrészt kipróbáltuk, hogy mi történik abban az esetben, ha a csomagokban nem 25 mintát továbbítunk. Ebben az esetben, ha például tíz adatot

továbbítunk csomagonként, akkor a mintavételi frekvencia tizede által meghatározott helyeken találhatóak meg a vizsgált spektrumvonalak. Másik kísérletünket a MAC réteg konfigurálásával végeztük el. Ennek eredményeit a 8.2., és 8.3. ábrán láthatjuk. A 8.2.a. és 8.3.a. ábrán látható spektrumú regisztrátum felvételét az `initialBackoff`=0 beállítással végeztük, tehát az operációs rendszer kezdeményezésünk után azonnal elkezdi az adatok rádión történő továbbítását. Ekkor a hibás spektrumvonalak igen markánsan jelennek meg. Amennyiben az `initialBackoff`=1 – melynek a 8.2.b. és 8.3.b. ábra felel meg – tehát egy kis szórású véletlen időtartamot követően kerül továbbításra a csomag, megállapítható, hogy a zavaró spektrumvonalak magassága csökken. A második legmagasabb spektrumvonal a jel első felharmonikusa. Ebben az esetben ráadásul az alapharmonikus spektrumvonala is megfelelő. A 8.2.c. és 8.3.c. ábrán az `initialBackoff`=15 beállítással készült felvétel látható. Ebben az esetben a spektrumkép jelentősen „kitisztult”, bár az alapharmonikus is jelentősen elmosódott, mivel a mintavétel kezdete még jobban ingadozik.

A kísérletek eredményeként tehát világossá vált, hogy a mintavétel időzítése még mindig nem kielégítő pontosságú, és egyéb folyamatok, mint például a rádió működtetése jelentősen befolyásolja.

Az eddig készített felvételek mindegyikének az volt a problémája, hogy a mintavétel időzítése nem volt megfelelő pontosságú. Ennek oka, hogy az időzítő nem közvetlenül indította az AD-átalakítást, hanem többszörös függvényhíváson keresztül, amely függővé teszi a triggerelés pontosságát a mikrokontroller terheltségétől.

A probléma megoldásához újabb átalakításokat kellett végezi mind az új időzítő, mind az AD-átalakítást vezérlő komponensen. Ezen átalakításokkal célunk az, hogy az időző megszakításrutinja közvetlenül indítsa az AD-átalakítás folyamatát, mely egyébként a program elején elvégzett megfelelő felkonfigurálás után csupán egy bit bebillentését jelenti az AD-átalakítóhoz tartozó vezérlő regiszterben. Ez tehát elvileg nem jelentene nagy feladatot. A nehézséget inkább az okozta, hogy ezt a műveletet az operációs rendszert megkerülve végeztük el. Ez viszont azzal jár, hogy az nem jegyzi fel az AD-átalakításra vonatkozó kérésünket, így a konverzió elvégezte után nem tudja, hogy milyen paraméterű interfészen keresztül értesítse a felhasználót. Meg kellett oldani tehát azt, hogy értesítést kapjunk az átalakítás eredményéről. A megoldást az jelenti, hogy nem az `ADCC` paraméterezett interfészét használjuk, hanem közvetlenül az AD-átalakítót kezelő modul által szolgáltatott interfészhez csatlakozunk. Egyetlen hátrány lehet ebben az esetben, hogy így a különböző csatornák kezelése kissé nehézkesebb lenne, ha több csatornán szeretnénk mintavételezni. Ebben az esetben ez azonban nem jelent nehézséget, ugyanis csupán a mikrofonhoz tartozó csatornát használjuk.



8.4. ábra. 200 Hz-es szinuszjel 1800 Hz-es mintavételi frekvenciával

A fenti ábrát az új időzítő modul segítségével készítettük. Észrevehetjük, hogy a hibásan megjelent spektrumvonalak szinte teljesen eltűntek. A 200 Hz egész számú többszöröseinél megjelenő vonalak a gerjesztőjel felharmonikusai. Habár szinuszos gerjesztést alkalmaztunk, a hangszóró torzítása miatt felharmonikusok is megjelentek.

Mint már említettük, a mintavételezett értékeket 8 bites formátumban továbbítottuk. Ennek hatására teljes kivezérlés esetén (8*6dB+1.76dB)≅50dB-es jel-zaj viszony várható a kvantálási zajmodell alkalmazásával. Mivel a 8.4. ábrán látható felvételt teljes kivezérlés mellett végeztük, ezért ebben az esetben is hasonló értéket várhatunk, és az ábrán láthatóan ez megközelítőleg teljesül is.

9. Szinkronizáció az adatgyűjtő hálózatban

A hálózat kiépítése során csakhamar felmerült a szinkronizáció szükségessége. Mivel a hálózat elemei mind saját órajel-generátort használnak, így semmilyen biztosíték nincsen rá, hogy az azonos névleges frekvenciájú kvarcok valójában is teljesen azonos frekvenciájú órajelet szolgáltatnak. Ez azt eredményezi, hogy az egyes csomópontok aszinkron járnak, az óráik tehát elcsúsznak egymáshoz képest. Ennek hatására a mintavételi frekvenciák eltérnek, tehát adott idő alatt különböző számú mintát gyűjtenek az egységek. Ez komoly problémákhoz vezet mind a hálózat mind a zajcsökkentő rendszer szempontjából. A következő részben ezeket a szinkronizálatlanság miatt bekövetkező hibalehetőségeket tárgyaljuk.

9.1. Szinkronizálatlanság hatásai

9.1.1. Zajcsökkentés hatékonyságának csökkenése

A fenti rész, és a hálózat felépítése alapján egyértelmű, hogy a mintavételezés, az adatátvitel és -feldolgozás sebessége nem egyezik meg, mivel ezek közül minden folyamatot más-más órajel-generátor lát el.

Az adattovábbítás sebességét a leglassabb mote határozza meg, ugyanis a token ring hálózatban minden körben minden csomópont csak egyszer adhat. Mivel a leglassabb egység addig vár minden körben, amíg nem gyűlik össze egy csomagnyi adat, így a többi csomópont is kénytelen ezt megvárni, addig ugyanis nem kapják meg az adás jogát, míg az őket megelőző mote-ok nem továbbították saját adataikat. Az adatokat feldolgozó DSP tehát más időközönként dolgozza fel a mintákat, mint amilyen időközönként azok a mintavételezés helyén keletkeznek.

Nézzük, hogy milyen eltérést okoz ez a normál működéshez képest. Mint már említettük, a DSP mindig a legkisebb mintavételi frekvenciának megfelelően kapja az adatokat a hálózat felől. A nagyobb frekvencián mintavételező egység ezért adott idő alatt több adatot szolgáltat, mint amennyit a DSP feldolgoz.

Nézzük, hogy a beolvasó egység szemszögéből mennyire változnak meg a beolvasott jel paraméterei. Az egyszerűség kedvéért szinuszos gerjesztést vizsgáljunk.

Legyenek adottak a következő paraméterek:

- a mintavételi időköz a lassabb mote esetén T_l, így a jel időfüggvényét $A \cdot \sin(2 \cdot \pi \cdot f \cdot n \cdot T_l)$ szerint kapjuk
- a gyorsabb mote esetén a mintavételi időköz T_{gy}, így a jel időfüggvényét $A \cdot \sin(2 \cdot \pi \cdot f \cdot n \cdot T_{gy})$ szerint kapjuk

ahol f a vizsgált szinuszos jel frekvenciája, A pedig az amplitúdója

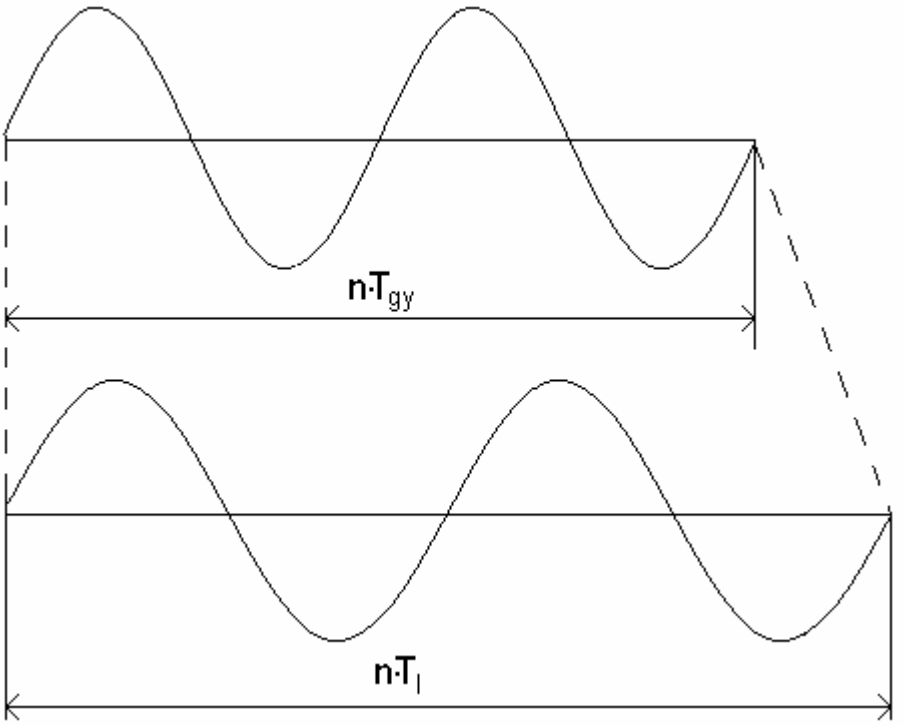
Az adatfeldolgozás, mint már említettük, T_l-nek megfelelő mintavételi időköz szerint történik, tehát egy $g(t)$ függvényű jelből érkező n-edik minta a függvény

$n \cdot T_l$ időpontbeli értékének felel meg a feldolgozás szempontjából, míg a valóságban a gyorsabb mintavételezés helyén ez az $n \cdot T_{gy}$ időpontot jelöli.
A jelenség értékeléséhez végezzük el a következő átalakítást:

$$A \cdot \sin(2 \cdot \pi \cdot f \cdot n \cdot T_{gy}) = A \cdot \sin(2 \cdot \pi \cdot f \cdot n \cdot T_{gy} \cdot \frac{T_l}{T_l}) = A \cdot \sin(2 \cdot \pi \cdot f \cdot \frac{T_{gy}}{T_l} \cdot n \cdot T_l) =$$

$$= A \cdot \sin\left[2 \cdot \pi \cdot \left(f \cdot \frac{T_{gy}}{T_l} \right) \cdot n \cdot T_l \right] = A \cdot \sin(2 \cdot \pi \cdot f_2 \cdot n \cdot T_l)$$

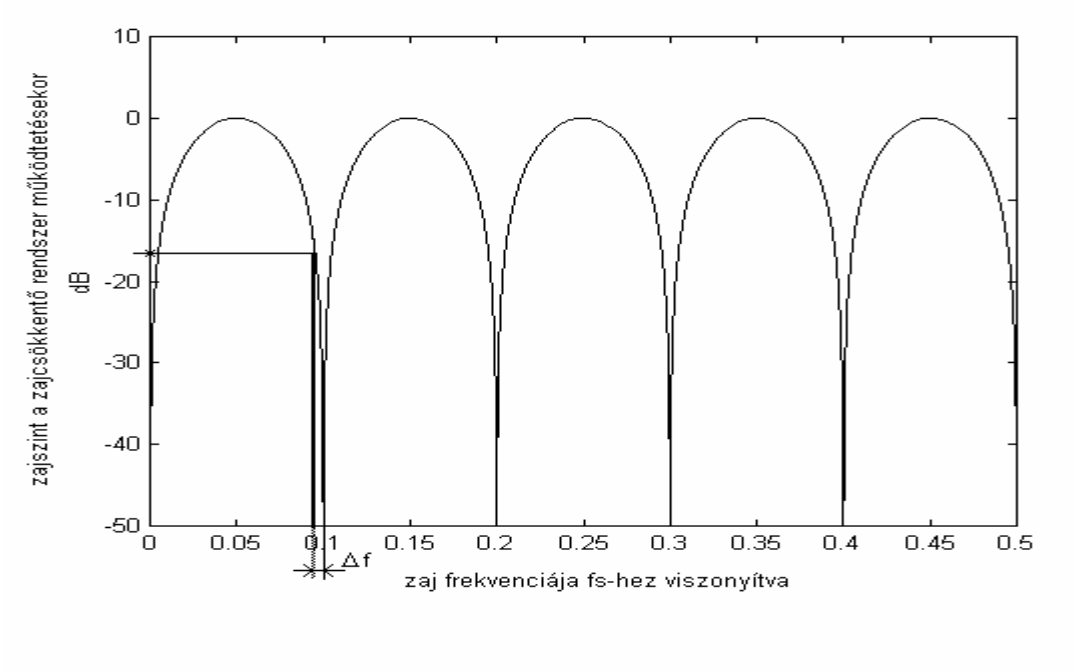
Ha tehát a mintavételi időköz T_{gy}, de mi ehelyett T_l-t feltételezünk, akkor a valójában f frekvenciájú jelet f_2 frekvenciájúnak mérjük, ahol $f_2 = \frac{T_{gy}}{T_l} \cdot f$. Mivel $T_{gy} < T_l$ ezért az így előálló jel frekvenciája csökken az eredeti jelhez képest.
Ezt a jelenséget láthatjuk a 9.1. ábrán. Látható, hogy ha egy $n \cdot T_{gy}$ hosszúságú regisztrátumot $n \cdot T_l$ idő alatt dolgozunk fel, az olyan, mintha „megnyújtanánk” a jelet, tehát csökkentenénk a frekvenciáját.



9.1. ábra. Eltérő mintavételi frekvencia hatása a jel frekvenciájára

Vizsgáljuk meg, mi lesz ennek a hatása a zajcsökkentő rendszerre. A rezonátoros zajcsökkentő algoritmus egyik lépése, hogy az AFA előállítja az elnyomandó jel frekvenciáját, melyet valamilyen mintavételi frekvenciához

képesti relatív egységben ad meg. Ha a hibajelet egy másik mintavételi frekvenciához viszonyítva mérjük, akkor ugyanarra a jelre kétféle frekvencia érték áll elő. Ha átgondoljuk a jelenséget, látható, hogy ez éppen az imént tárgyalt esetnek felel meg, ugyanis a referenciajel itt is két csatornán érkezik. Megjegyezzük, hogy a referenciajelnek ugyan nem kell megegyezni a hibajellel, frekvenciájuknak viszont egyenlőnek kell lenni, és jelen esetben ez számít. Ha megvizsgáljuk a zajcsökkentő rendszer hatékonyságát, tehát azt hogy bizonyos frekvenciákon milyen elnyomást biztosít, akkor megállapíthatjuk, hogy amennyiben az AFA pontosan becsli az elnyomandó jel frekvenciáját, akkor az elnyomás teljes, és minél jobban eltér a hibajel frekvenciája a becsült értéktől, annál kisebb lesz hatékonyság. A 9.2. ábrán látható, hogy ha a referenciajel frekvenciája a mintavételi frekvenciához képest például 0.1, akkor annak minden egész számú többszöröseinél teljes elnyomást biztosít, tehát a zajszint nullára (-∞ dB) csökken. Abban az esetben tehát, ha a zaj frekvenciája Δf értékkel eltér a referenciajel frekvenciájától, akkor csökken a zajcsökkentő rendszer hatékonysága, mivel magasabb lesz a zajszint. Ugyan esetünkben az eltérés nem jelenti azt, hogy a két jel frekvenciája fizikailag is eltér egymástól, hanem az eltérő mintavételi frekvencia miatt a jelfeldolgozó algoritmus számára látszik úgy, hogy nem egyezik meg egymással a két frekvencia, de ez is ugyanúgy a hatékonyság csökkenésében nyilvánul meg.



9.2. ábra. Zajcsökkentő rendszer átviteli függvénye

9.1.2. Adatfelhalmozódás

Az előző részben leírt jelenségben a szinkronizálatlanság ugyan rendellenes működést okozott, a rendszer hosszútávú működését azonban a vizsgált szempontból nem befolyásolja, csupán a hatékonyságot csökkenti.

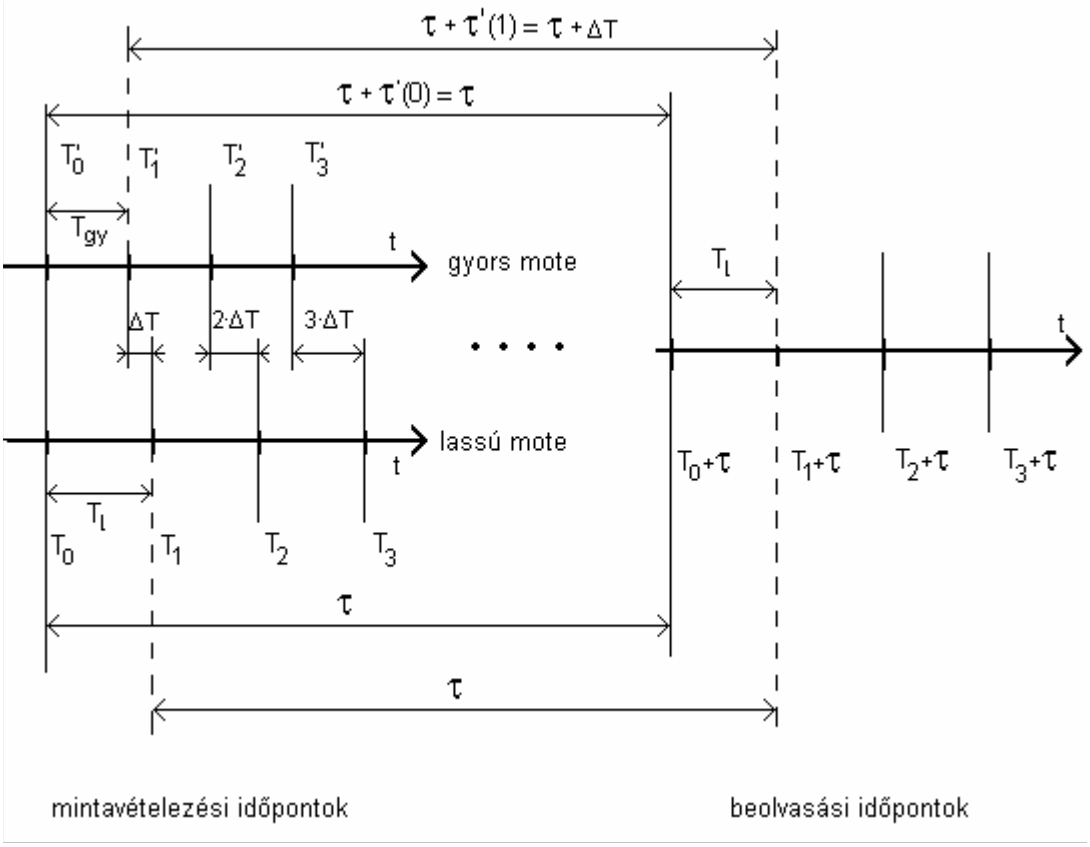
Az előzőekben már megvizsgáltuk, hogy az adattovábbítás a leglassabb mote-nak megfelelő ütemben történik, viszont a mintavételezés a többi csomópontban gyorsabban is történhet. Ennek következtében ezeken a mote-okon az adatok gyorsabban keletkeznek, mint ahogyan fogynak. Az el nem küldött adatokat viszont tárolni kell a memóriában, és mint látható, a tárolandó adatok száma egyre nő. Ez ahhoz vezet, hogy egy idő után a mikrokontrolleren a bufferelési célra fenntartott memória elfogy, ami nyilván nem megengedhető, mivel adatvesztéshez vezet.

9.1.3. Adatút átviteli függvényének változása

Az eddig tárgyalt két eset közül ugyan egyik sem kedvez a zajcsökkentő rendszer működésének, azt viszont egyik esetben sem láttuk be, hogy a jelenség a berendezés működésképtelenségét okozná. Hamar beláthatjuk azonban, hogy az eltérő mintavételi frekvenciák a jelenlegi formájában lévő hálózatot használva instabilitást okoznak a rendszerben. Ezt a fenti két esetben leírtakat felhasználva bizonyítjuk be.

Mint tudjuk, az aktív zajcsökkentő rendszerekben alapvető fontosságú a beavatkozójel és a hibajel bemenet közötti (másnéven visszacsatoló, vagy másodlagos út) átviteli függvényének pontos ismerete. Ez az átviteli függvény magában foglalja az AD-, DA-átalakító, jelkondicionáló áramkörök, a hangszóró, a mikrofon valamint az ezek közti akusztikus út és esetünkben a rádiós hálózat átviteli függvényét. Ez utóbbi gyakorlatilag a pufferelés és adattovábbítás miatt megjelenő késleltetés formájában jelenik meg.

Ugyan gyakorlatilag nehezen megoldható, de tételezzük fel, hogy a zajcsökkentő rendszer indításakor pontosan ismerjük a visszacsatoló út átviteli függvényét. Tegyük fel, hogy a hálózat által reprezentált átvitel egy τ késleltetésnek felel meg mindkét adatút esetén. Ezt az esetet ábrázolja a 9.3. ábra.



9.3. ábra. A késleltetés változásának bemutatása

ahol:

$T'_{n+1} - T'_n = T_{gy}$

$T_{n+1} - T_n = T_l$

Az ábrán a $\Delta T = T_l - T_{gy}$ jelölést használtuk, és T_i jelöli a lassú mote-on, T'_i pedig a gyors mote-on a mintavételi időket. A T_i-ben, és T'_i-ben vett adatok feldolgozása pedig a $T_i+\tau$ időpontokban történik.

A fent leírt eset a T_0 időpontban áll elő. A mintavételezést innen indítjuk mindkét mote-on, és az adatok beolvasása is ekkor kezdődik. A beolvasási időköz megegyezik a lassabb mote mintavételi időközével, azaz T_l-lel, míg a gyorsabb mote T_{gy} időközönként vesz mintát, de azt a vevő csak T_l időnként dolgozza fel.

Első ránézésre is nyilvánvaló, hogy a lassabb mote és a fogadó állomás közti késleltetés állandó, de vizsgáljuk meg, hogy a gyorsabb mote és a fogadó állomás közti késleltetés hogyan változik. Jelöljük ezt a változást τ'-vel. Mivel azt feltételezzük, hogy minden mintavételi frekvencia megegyezik, tehát ebből kifolyólag $T_n = T_n'$, ami az ábrából megállapíthatóan nem igaz, emiatt a gyorsabb mote és a fogadó állomás közti késleltetés minden mintavétel alkalmával ΔT-vel nő. Tehát megállapítható, hogy a késleltetés változása a következő képlet szerint alakul: $\tau'(n) = n \cdot \Delta T$

Mi is lesz ennek az állandóan változó késleltetésnek a hatása a zajcsökkentő rendszerre? A növekvő késleltetés az átviteli függvény fázistolásában jelenik meg. Egy $\tau'(n)$ késleltetést tehát egy $\exp[-j\cdot 2\pi\cdot\tau'(n)\cdot f]$-s szorzóval vehetünk figyelembe. A jelen helyzetben, amennyiben a mintavételi frekvenciák megegyeznek, akkor a $\tau'(n)$ állandó, és az identifikáció során meghatározható tehát nem okoz zavart. Habár a minél gyorsabb működés érdekében érdemes a késleltetést lehető legkisebb értéket elérni, hiszen egy szabályozási kör visszacsatoló ágában lévő késleltetés lassabb szabályozó tervezését teszi csak lehetővé. Így van ez a zajcsökkentő rendszereknél is. A valóságban azonban mint láttuk $\tau'(n)\cdot$nem állandó, ezért működés közben az általunk identifikált átviteli függvénytől el fog térni a valódi átviteli függvény. De mikor vezet ez működésképtelenséghez?

Belátható, hogy amennyiben a másodlagos út identifikált átviteli függvénye $\pm 90^{o}$-ra megközelíti a valóságos átviteli függvényt, akkor a zajcsökkentő rendszer stabilizálható a szabályzás lassításával. Ezek szerint, amikor a növekvő késleltetés miatt jelentkező változás eléri a 90^{o}-ot akkor a rendszer instabillá válik.

Igen informatív lehet, ha meghatározzuk, hogy ez az eset egy átlagos rendszerben mikor következik be. Ez az idő ugyanis elég nagy lehet ahhoz, hogy bizonyos esetekben ne kelljen foglalkozni a problémával, például azért, mert az instabilitás elérése sokkal tovább tart, mint a berendezés egy bekapcsolása alkalmával a folytonos működési idő. Nézzük, hogyan becsülhető az instabilitás eléréséhez szükséges idő. A mintavételt időzítő kvarcok által szolgáltatott mintavételi frekvencia eltérése a gyakorlatban normál kristályok esetén körülbelül h=10 ppm, tehát $h= 10^{-5}$, ahol h a hibát jelöli.
Az eddigi jelölésekkel:

$$f_{gy} = \frac{1}{T_{gy}}, \text{ valamint } f_l = \frac{1}{T_l}$$

f_{gy} és f_l a két mintavételi frekvenciát jelöli, melyeknek hibája megegyezik az órajel-generátorok hibájával, ugyanis a mintavételi frekvenciák és az órajel-generátorok frekvenciája között csupán egy konstans osztószám teremt kapcsolatot. Ekkor tehát felírható a két mintavételi frekvencia közti kapcsolat a hiba értékét felhasználva:
$f_{gy}=(1+h)\cdot f_l$

Behelyettesítve a mintavételi időközök és frekvenciák közti összefüggést

$T_l=(1+h)\cdot T_{gy} \rightarrow \Delta T= T_l - T_{gy} = h\cdot T_{gy}$

Tehát minden mintavétel alkalmával ekkora késleltetés jelenik meg a gyorsabb csatornában. Az instabilitás eléréséhez a késleltetés növekedésének el kell érnie

az üzemelési frekvencián a 90º-ot. Legrosszabb esetben ez a frekvencia megegyezik a mintavételi tétel által megengedett legnagyobb mintavételezhető frekvenciával, azaz f=f_{gy}/2-vel, ugyanis adott késleltetés itt okozza a legnagyobb fázistolást. Nem fontos, hogy ez a frekvencia megegyezzen a zaj frekvenciájával, az is elegendő, ha annak valamely felharmonikusa, hiszen a zajcsökkentést harmonikusonként végezzük, tehát minden egyes harmonikusra ismernünk kell az átviteli függvényt.

Ha tehát a jel frekvenciája f=f_{gy}/2, az azt jelenti, hogy $f = \frac{1}{2 \cdot T_{gy}}$.

τ'(n) késleltetés f frekvencián Δφ=360º· f·τ'(n) értékű fázistolást okoz. Stabilitás határhelyzetében Δφ=90º, tehát

$$\tau'(n) = \frac{\Delta\varphi}{360° \cdot f} = \frac{90°}{360° \cdot f} = \frac{1}{4 \cdot f} = \frac{1}{4 \cdot \frac{1}{2 \cdot T_{gy}}} = \frac{T_{gy}}{2}$$

mivel $\tau'(n) = n \cdot \Delta T = n \cdot (h \cdot T_{gy})$

így: $n \cdot h \cdot T_{gy} = \frac{T_{gy}}{2}$

tehát

$$n = \frac{1}{2 \cdot h} = \frac{1}{2 \cdot 10^{-5}} = 50.000$$

minta után érjük el a stabilitási határhelyzetet a feltételezett mértékű, de reálisnak mondható h hiba esetén.

Amennyiben az instabilitás bekövetkeztének idejére vagyunk kíváncsiak, akkor ezt az értéket meg kell szoroznunk a mintavételi időközzel, mely pedig T_{gy} . Tehát:

$$T_{instabil} = n \cdot T_{gy} = \frac{T_{gy}}{2 \cdot h}$$

Ez például 2 kHz-es mintavételi frekvencián 25 másodpercet eredményez, ami tehát mindenképpen szükségessé teszi a szinkronizációt, hiszen ez idő alatt még az identifikáció elvégzése sem lehetséges, tehát már az átviteli függvény mérésekor is hibás eredményt kapunk.

9.1.4. Doppler analógia

Bár új eredményeket már nem szolgáltat, mégis érdekes eredményre vezet, ha a jel frekvenciájának megváltozását egy más hatással modellezzük, mialatt a mintavételi frekvenciákat azonosnak tételezzük fel. Ismerősnek tűnhet akár a

hétköznapokból is, hogy egy hangforrás által keltett rezgést más frekvenciájúnak hallunk, mint amilyen annak a valódi értéke. Ezt az okozza, hogy a hangforráshoz képest bizonyos sebességgel elmozdulunk. Ezt nevezik Doppler-hatásnak.

A Doppler-hatás miatt egy jel frekvenciáját a következőképpen érzékeljük, ha hozzá képest távolodunk:

$$f_{érz} = f_{jel} \cdot \frac{c - v}{c} = f_{jel} \cdot (1 - \frac{v}{c})$$

ahol *c* a hang terjedési sebességét jelöli.
Nézzük, hogy az eltérő mintavételi frekvencia miatti változás miképpen írható le. Mint már bebizonyítottuk, az érzékelt jel frekvenciája:

$$f_2 = \frac{T_{gy}}{T_l} \cdot f = f \cdot \frac{(1-h) \cdot T_l}{T_l} = f \cdot (1-h)$$

tehát $h = \frac{v}{c}$ esetén a két módszer ugyanakkora változást eredményez a jel érzékelt frekvenciájában. Ez tehát azt jeleni, hogy az eltérő mintavételi frekvenciák hatása olyan, mintha az érzékelő mikrofon távolodna a hangszórótól. A távolodás sebessége pedig számolható, mégpedig *v=h·c*. Ha *h* értéke a megadott 10 ppm, a hangsebesség pedig körülbelül 340 m/s, tehát ez egy 3,4 mm/sec-os mozgást jelent. Ezen mozgás hatására nyilván folyamatosan változik a mikrofon és a hangszóró közti késleltetés, mivel a hangnak egyre nagyobb utat kell megtenni.

Számoljuk ki, hogy ezzel a modellel milyen érték adódik az instabillá válás időpontjára. Akkor lesz a rendszer instabil, ha a legnagyobb frekvenciájú jel esetén már több mint 90º fáziskésleltetés lép be. Ez annak az esetnek felel meg, hogy a mikrofon a hullámhossz negyedénél nagyobb távolságot tett meg. Tehát a határeset a hullámhossz negyedével történő elmozdulás, melynek ideje kiszámítható:

$$T_{instab} = \frac{\lambda_{kritikus} / 4}{\mathrm{v}}$$

A $\lambda_{\text{kritikus}}$ a mintavételi frekvencia felével megegyező frekvenciájú jel hullámhossza, mivel itt érjük el leghamarabb a 90º-os fázistolást.
Folytatva tehát:

$$T_{instab} = \frac{\frac{c}{f_{kritikus}}}{4 \cdot \mathrm{v}} = \frac{\frac{c}{f_{gy} / 2}}{4 \cdot \mathrm{v}} = \frac{2 \cdot c}{4 \cdot f_{gy} \cdot \mathrm{v}} = \frac{c \cdot T_{gy}}{2 \cdot h \cdot c} = \frac{T_{gy}}{2 \cdot h}$$

Ugyanakkora értéket kaptunk, mint az előző esetben, tehát a két eljárás ebből a szempontból is megfelel egymásnak.

9.2. Szinkronizáció megoldása

Az alapvető gondot tehát az okozza, hogy az adatok továbbítása azonos frekvenciával történik minden mote esetén, míg a mintavételezés az órajel frekvenciák eltérése miatt nem azonos ütemű. Ezek alapján kiderül, hogy két lehetőség kínálható a feladat megoldására.

Első alternatíva az lehet, hogy az adatok továbbításának sebessége minden mote esetén megegyezik az adatgyűjtés sebességével. Ez azonban számos problémát eredményezne. Egyrészt a fogadó állomást fel kellene készíteni arra, hogy a hálózati elemek aszinkron szolgáltatják az adatokat, mégpedig mindenki más-más ütemben. Ekkor előfordulhat, hogy néhány mote egyszerre szeretné továbbítani az összegyűjtött adatokat. Ennek bekövetkezésére igen nagy az esély, hiszen az egymáshoz képest elcsúszó mintavételi időpontok valamikor fedésbe kerülnek egymással. Ekkor elég nehéz lenne megoldani az adattovábbítást, ugyanis a véletlenszerűen adást kezdeményező mote-ok rádiós adásai nagyobb valószínűséggel ütköznének. Az ütköző csomagok miatt nagyobb lehet a bizonytalanság az adattovábbításban. Ez viszont a biztonságos adattovábbítás rovására menne, ami nem engedhető meg, hiszen ez az adatútban bekövetkező véletlenszerű változásokhoz vezetne, és így instabilitást okozna az aktív zajcsökkentő rendszerben. A másik probléma az, hogy az aszinkron érkező adatok bonyolultabbá teszik a DSP-n futó programot, ekkor ugyanis kezelni kellene azt a jelenséget, hogy nem egyező mintavételi frekvenciával érkeznek az adatok, így minden egyes mote-ot külön kellene kezelni.

Az eddigiekben kiépített hálózatunk tehát teljes mértékben alkalmatlan lenne a fent vázolt szinkronizálásra, mikoris a fogadó egységnél kell az adatokat szinkronizálni.

Másik lehetőség az, hogy a mintavételezés szinkronizálását az adatgyűjtési folyamatnál végezzük el. Ez azért kedvezőbb, mert így világosabban áttekinthető lesz az adattovábbítási folyamat, és leveszi a szinkronizációval járó terhet a fogadó állomásról. Talán ésszerűbb is a mintavétel szinkronizálását minden egyes mote-on külön-külön lekezelni, és így szétosztani az ezzel járó tevékenységeket, hiszen ez igen szoros kapcsolatban van a mote-ok egyik fő feladatával, a mintavételezéssel.

Nagy feladatot jelentett egy olyan eljárás kidolgozása, ami megfelelő pontossággal képes elvégezni a mote-ok szinkronizálását. Ne feledjük, hogy a rendszerben adott jelúton nem szabad 90º-nál nagyobb mértékűnek lenni az átviteli függvény eltérésének a valódi állapothoz képest. Ez 2 kHz-es mintavételi frekvencián például legalább 250 μsec-os pontosságot igényel, mivel ekkor a működési tartomány 1 kHz. Ezen jelnek a periódusideje ugyanis 1 msec, aminek negyede a kritikus érték. Ennél viszont még pontosabb szinkronizáció szükséges, hiszen egyáltalán nem biztos, hogy az identifikáció pontos, és ekkor kisebb szögeltérés esetén is kilépünk a megengedett tartományból. Ráadásul a zajcsökkentő rendszer dinamikája is függ az identifikáció pontosságától. Minél

jobban eltér az átviteli függvény modellje az eredetitől, annál lassabb beállás biztosítható.

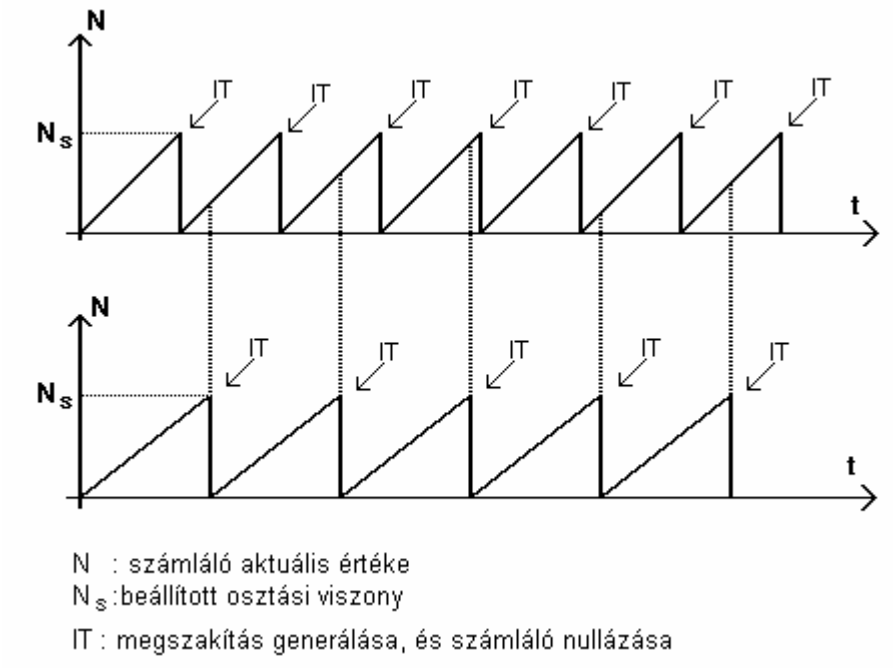
Első kérdés az volt, hogy egyáltalán mi legyen a szinkronizálási stratégia. Esetünkben nem feltétlenül kell hagyományos értelemben vett óra szinkronizálást végezni. Nem fontos ugyanis, hogy a mote-ok tudják az abszolút időt, elegendő, ha a mintavételi időpontok szinkronizáltak. Ráadásul az sem követelmény, hogy a mintavétel minden mote-on egyszerre történjen, nincs szükségünk az algoritmusban időben összetartozó mintákra. Ez azzal a könnyebbséggel jár, hogy a mote-ok mintavételi időpontjainak tehát nem kell egybeesni, csupán egymáshoz képest fix időbeni eltolással kell történni. Ez azért lehetséges, mert az átviteli csatornákat identifikáljuk, tehát csak azoknak időbeni állandósága a kritikus.

Az nyilván egyértelmű, hogy a szinkronizáció a rádió segítségével történik. Azon viszont el kell gondolkodnunk, hogy az eddigi felfogásunknak megfelelően hogyan oldjuk meg a szinkronizációt az adattovábbításban haszontalannak számító üzenetek nélkül.

A szinkronizáció végleges megvalósításának ismertetése előtt még tekintsük át röviden az időzítő működését. Ez a későbbiekben leírt folyamatok megértése miatt fontos.

Az időzítő a mikrokontroller órajelét felhasználva üzemel. Ez azt jelenti, hogy az időzítő engedélyezésével gyakorlatilag egy számláló kezd el üzemelni, és értéke minden órajel után eggyel nő. Az időzítő általuk használt módjában ez a számláló nullázódik és megszakítás generálódik amikor a számláló értéke elér egy, a felhasználó által beprogramozható értéket, melyet az időzítőhöz tartozó regiszterekben kell megadni. Jelöljük ezt az értéket N_s-el, mely egyébként egy osztásviszonyt képez az órajel és a megszakítások gyakorisága között, így tehát egy programozható órajelosztót valósítottunk meg. Ekkor tehát az időzítés gyakorisága $T_s=N_s \cdot T_{kvarc}$, ahol T_{kvarc} az órajel periódusideje. Ábrázoljuk ezt a folyamatot az idő függvényében. Ez látható a 9.4. ábrán.

Az ábrán két számláló működését is felrajzoltuk. A két számlálóban megegyezik az osztásviszony, az alsó grafikonon ábrázolt folyamatban azonban az órajel frekvencia kisebb, így a megszakítások gyakorisága is kisebb. Látható, hogy ha a lassabb időzítő megszakításainál kiolvassuk a gyorsabb számláló értékét, és ezeket a kiolvasott értékeket ábrázoljuk, akkor egy emelkedő fűrészjelet kapunk. Ha viszont a gyorsabb időzítő megszakításainál olvassuk ki a lassabb számláló értékét, akkor csökkenő fűrészjelet kapunk. (Ezt az esetet az átláthatóság érdekében nem ábrázoltuk, de az ábrából könnyen megállapítható) A két számláló által generált megszakítások gyakorisága pedig akkor egyenlő, ha az így kiolvasott értékek minden kiolvasás alkalmával megegyeznek. Ezek az általános elvek a későbbiekben fontos szerepet fognak játszani.



9.4. ábra. Az időzítő működése

A fentiek alapján talán már kezd körvonalazódni a stratégia. Meg kell oldani, hogy az egyes mote-okon generálódó megszakítások alkalmával kiolvassuk a többi mote-on a számláló értékét, annak ellenőrzésére, hogy mennyire járnak együtt óráik, és így mennyire egyeznek meg a mintavételi frekvenciák. A kiolvasott értékek alapján az is megállapítható, hogy az órajel kisebb vagy nagyobb-e az adott mote-on, mint azon, ahol a megszakítás generálódott (a kiolvasott értékek csökkennek vagy nőnek-e). Ezek után a kiolvasott adatok felhasználásával el kell érni, hogy a kiolvasott értékek minden kiolvasás alkalmával megegyezzenek, és ráadásul egy előre megadott értéket vegyenek fel. Az első feltétellel biztosítható, hogy a mintavételezés minden mote-on egymáshoz képest egyenlő időpontban, és így egyenlő frekvenciával történjen. A második feltétel pedig azért szükséges, mert ha a kiolvasott értékek a hálózat minden üzembe helyezése alkalmával más és más értéken stabilizálódnának, akkor minden ilyen alkalommal más lenne a hálózat csomópontjaiban a mintavételezések közti időköz. Ez viszont gondot okozhat akkor, ha egy előző üzembe helyezés alkalmával lemért átviteli függvényt szeretnénk használni. Tételezzük fel ugyanis, hogy a mote-ok és a hangszórók geometriai elhelyezése azonos, valamint a külső viszonyok sem változnak a hálózat két egymást követő indítása között. Ekkor a mikrofont tartalmazó mote-ok és hangszórók közötti átviteli függvénynek nem lenne szabad megváltozni. Ha azonban az egyik mote és a hangszózók közti átviteli függvény nem változik, akkor egy másik mote és a hangszórók közti átviteli függvény nem maradhat a régi, ha az első mote-hoz képest más időpontokban veszi a mintákat.

Felmerülhet a kérdés, hogy szükséges-e minden mote-párra megvizsgálni, hogy egymáshoz képest milyen az órajel frekvenciáiknak a viszonya. Gyakorlatban egyszerűbb, ha választunk egy referencia mote-ot, amelyhez viszonyítjuk a többi mote mintavételezését. Ezt tesszük mi is. Tehát a mote-okon az időzítő állását mindig a referencia mote megszakításának pillanatában olvassuk le. Ebből következően a referencia mote órajeléhez viszonyítva tudjuk megállapítani a többi mote órajelének értékét.

Nehézséget okozhat, hogy hogyan állapíthatjuk meg egy mote-ról, hogy éppen megszakítást generált a rajta lévő időzítő. Közvetlenül a megszakítás keletkezésének időpontját valóban elég nehéz lenne megállapítani. Kihasználhatjuk azonban, hogy minden mote a huszonötödik minta után elküldi a mintákból összeállított csomagot. A csomag érkezésének időpontját pedig felhasználhatjuk az időzítő aktuális állapotának lekérdezésére. Habár ez a csomag nem a megszakítás tényleges időpontjában érkezik meg, de remélhetően mindig fix késleltetéssel ami – mint már említettük – nem számít az alkalmazásban. Ebből is látszik, hogy mekkora jelentősége van a hálózatban a már oly sokszor emlegetett determinisztikusságnak, tehát annak, hogy például ne legyenek ki nem számítható időpontban érkező csomagok.

Kis eltérés az alapelvhez képest az, hogy az időzítő értékét nem minden megszakítás alkalmával, csak minden 25. megszakításonként olvassuk ki. Ez azt jelenti, hogy így huszonötször ritkábban kapunk képet a rendszer állapotáról, ami nem jelenthet problémát, mivel az órajelek eltérése igen kicsi.

Összefoglalva az eddig leírtakat, láthatjuk, hogy találtunk olyan módszert mely segítségével minden mote-on meg tudjuk állapítani, hogy egy kiválasztott referencia mote-hoz képest mekkora a mintavételi frekvenciája, és hogy ahhoz képest milyen késleltetéssel történik a mintavételezés. Ez a módszer azon alapszik, hogy a referenciaként kiválasztott mote csomagjainak érkezésekor kiolvassuk az időzítő számlálójának értékét. Már csak annyit kell elérnünk, hogy ezeket az értékeket egy megadott szinten tartsuk.

Megállapítottuk, hogy amennyiben a referencia mote-nál lassabb egy mote mintavételezése, akkor a referencia mote csomagjainak beérkezésekor az időzítőből kiolvasott érték csökkenő, míg ha gyorsabb akkor növekvő tendenciát mutat. Ha tehát el tudnánk érni, hogy változtathassuk a mintavételi frekvenciát, akkor be tudnánk állítani a kiolvasott értéket egy kívánt szintre. Ugyanis, ha a kiolvasott érték nagyobb, mint a megkívánt, akkor a mintavételi frekvenciát csökkentve közelíthetünk ahhoz. Fordított esetben is hasonló logikával járunk el. A megvalósított szinkronizációs algoritmusban a mintavételi frekvencia megváltoztatása arányos a számláló megkívánt szinttől való eltérésével, így gyors beállás érhető el. A mintavételi frekvencia beállítása ugyanis nem ütközik semmilyen problémába, egyszerűen meg kell változtatni az órajel leosztás mértékét.

A fentiek alapján már minden részegység működőképes a szinkronizáció megvalósításához.

Hasonló ez a működési elv a fáziszárt hurok (PLL) működéséhez. Ez az elv tekinthető meg a 9.5.a. ábrán. A fázisdetektor funkciót a számláló adott pillanatban történő kiolvasása látja el. Az ábrán ezt egy olyan latch-el jelöltük, melyet a referencia jel megszakítása engedélyez. Valójában ez az engedélyezés a megszakítás pillanatában küldött csomag. A változtatható mintavételi frekvenciát lehetővé tevő hangolható időzítő a VCO-t valósítja meg, melyet egy, a hibajel felhasználásával működő szabályzó vezérel.
A szabályzó a következő beavatkozó jelet adja ki: $N_b=N_s–(N_l–N_a)/32$, ahol:

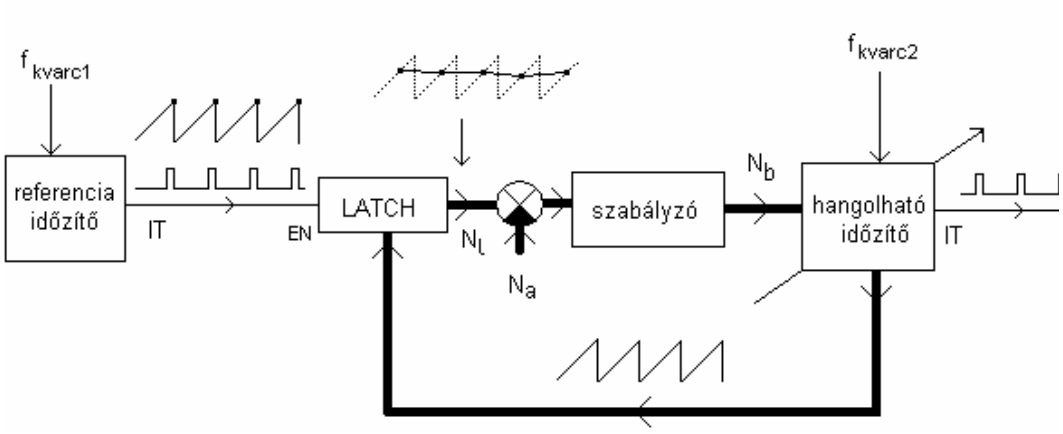
N_l: az időzítő számlálójának latch-elt értéke

N_a: az alapjel, amelynek meghatározására később térünk vissza

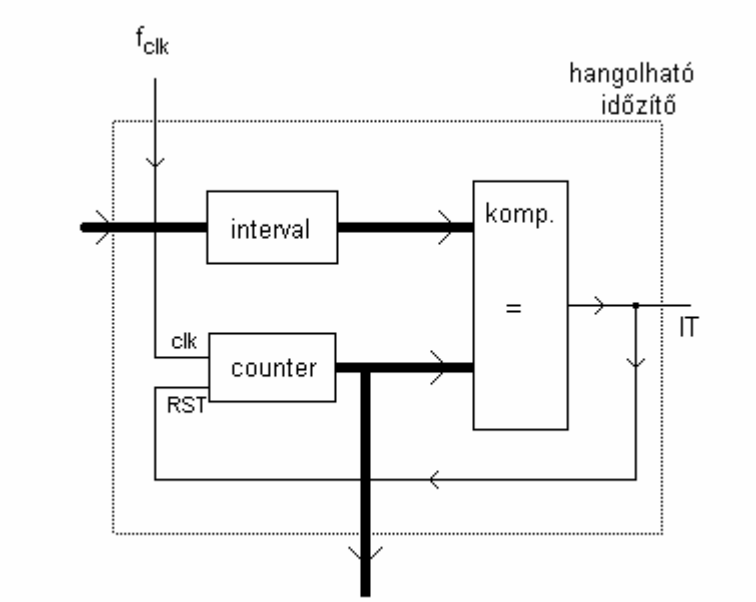
N_s: a névleges mintavételi frekvenciához tartozó osztószám, kiszámítása a következőképpen történik: $N_s=f_{kvarc}/f_s$, és f_s a mintavételi frekvencia

A képletben szereplő osztás egész számokra értelmezett osztás, mivel N_b csak egész szám lehet.

A fenti képlet szerint működő szabályzó negatív visszacsatolást biztosít, és stabil rendszert eredményez.



9.5.a. ábra. PLL analógia



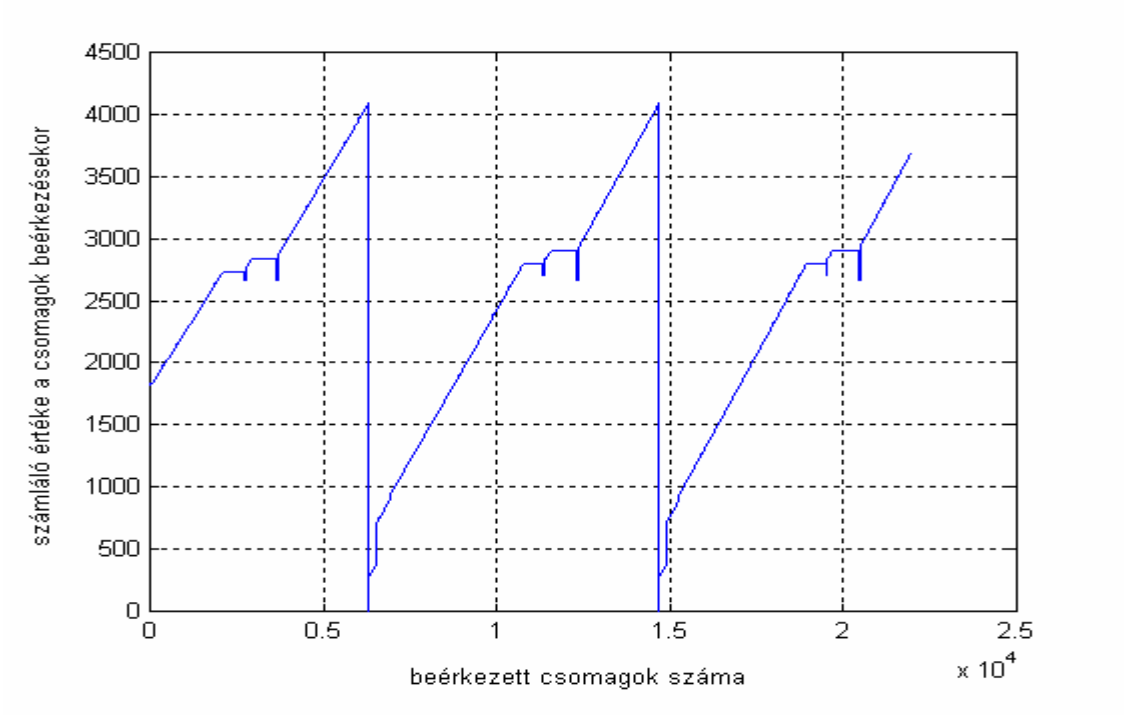
9.5.b. ábra. Hangolható időzítő

Az így előállt rendszerben tehát az egyes mote-okon üzemelő számlálókból kiolvasott fűrészjelek kerülnek fázismerev kapcsolatba, mely azt jelenti, hogy a fűrészjelek lefutó éleinél bekövetkező mintavételek állandó pozícióba kerülnek egymáshoz képest.

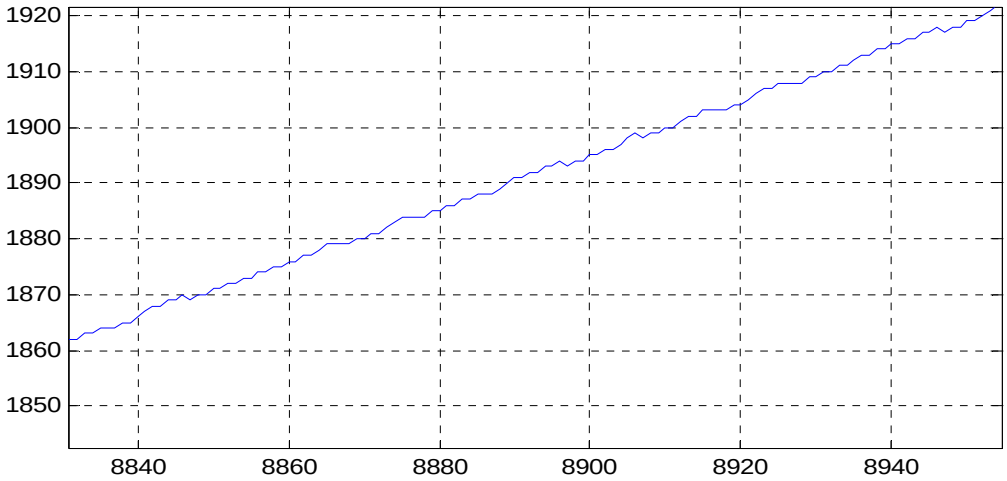
A 9.5.b. ábrán felvázoltuk a hangolható időzítő elvét is. A counter számláló az őt meghajtó órajel ütemében növekszik. A counter értékét egyrészt kivezetjük az időzítőből, így kívülről is hozzáférhető, másrészt bevezetjük egy komparátorba, melyhez csatlakozik egy interval nevű regiszter is. Ha a számláló és az interval értéke megegyezik, akkor abban az ütemben megszakítás generálódik, melyet felhasználunk a counter törlésére. Ez egy interval órajelnyi periodicitású folyamatot eredményez, tehát az interval regiszterben megadhatjuk a megszakítások generálásának intervallumát. Ez a működési elv van megvalósítva a mikrokontrolleren található időzítőkben is.

Az eddigiekben tehát áttekintettük a szinkronizáció elvi megvalósításának lehetőségét. Annak bizonyítására, hogy a módszer tényleg működőképes, méréseket is végeztünk. Első lépésben megvizsgáltuk, hogy a referencia mote csomagjainak érkezésekor milyen értékeket olvasunk ki az időzítőből. A mérés során két mote-ot működtettünk. Az egyik mote-ot kijelöltük referencia mote-nak, míg a másik mote a hangminták helyett a csomagokban a számlálóból a referencia mote csomagjainak érkezésekor kiolvasott értékeket továbbította, így azokat PC-n feldolgozhattuk az átjáró mote-ra csatlakozva. Ennek eredménye látható a 9.6. ábrán. Látható, hogy az elméletünk többé-kevésbé igazolódott a gyakorlatban is. Bár az így előállt ábra nem teljesen fűrészjel, mégis jellegében megegyezik azzal. A grafikonon látható törések azért vannak, mert a rádió által generált esemény más folyamatok miatt nem tud azonnal érvényre jutni. Az ábra egy részletét

kinagyítva látható a 9.7. ábrán, hogy a görbe nem teljesen monoton, és kis bizonytalanság tapasztalható a beérkezési időpontokban. Ez a bizonytalanság azonban jóval a tűréshatáron belül van.



9.6. ábra. A referencia mote csomagjainak beérkezésekor az időzítő értéke



9.7. ábra. A 9.6. ábra egy kinagyított részlete

A mérés alapján megállapítható az is, hogy a két mote-on lévő kvarc frekvenciája mekkora mértékben tér el egymástól. Ez a grafikon meredekségéből

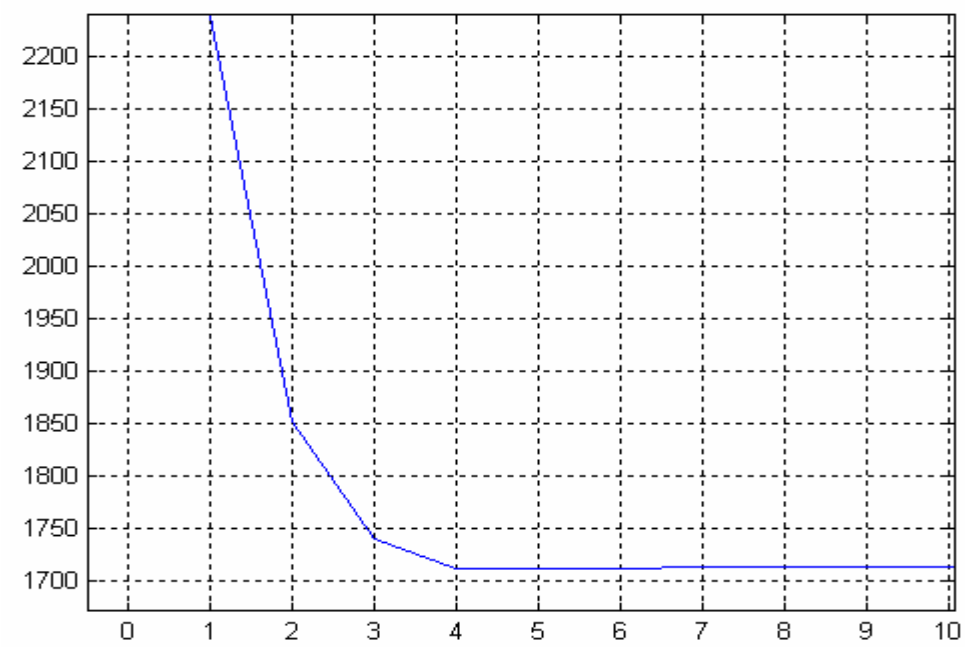
számítható. A meredekséget mindenképpen érdemes egy lineáris szakaszon leolvasni. A számítás gondolatmenete a következő:

A mintavételi frekvencia ebben az esetben 1800 Hz, ami 4096-os órajel leosztással érhető el. Két mintavétel között tehát 4096 órajel ütés történik.

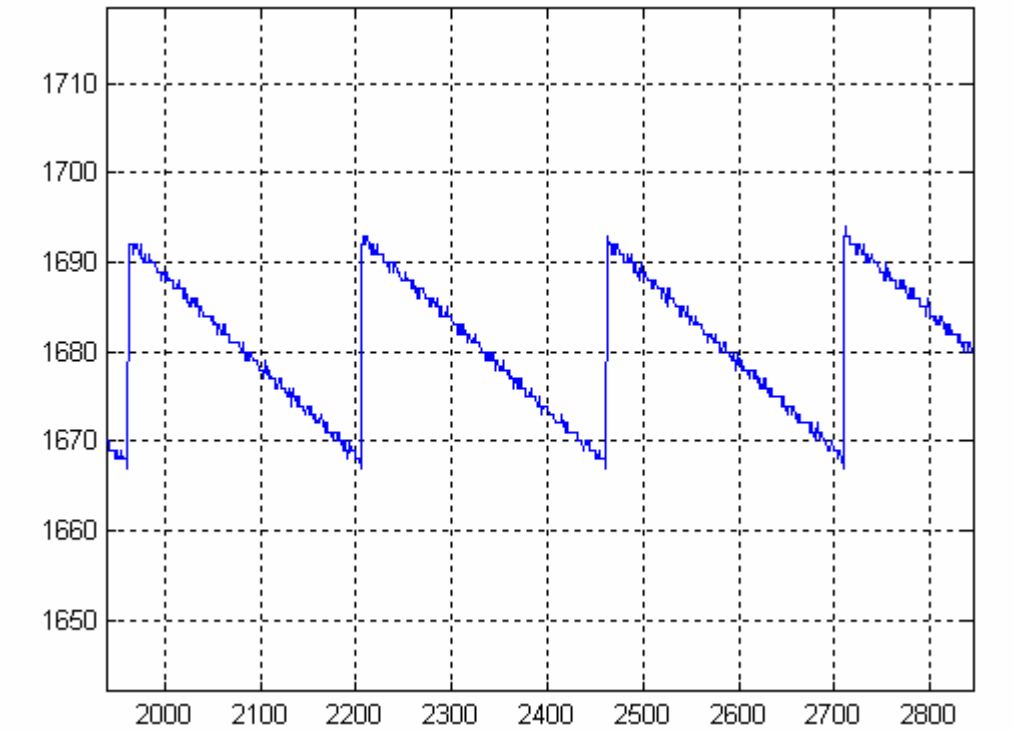
A függőleges tengelyen leolvasott érték azt mutatja meg, hogy mekkora a számláló értéke a csomagok beérkezésekor. Mi a 6212 és 10755-ös számú csomagok beérkezésekor mentett értékeket olvastuk le, amely 941 és 2799. Ez azt jelenti, hogy (10755-6212)=3843 csomag beérkezése alatt (2799-941)=1858 órajel periódussal csúszott el egymástól a két mote. A 3843 csomag közben (3843·25·4096)= 393523200 órajel periódus telt el. Egy órajel ütés alatt tehát h=1858/393523200=4.72e-006=4.72 ppm az eltérés. Ez megfelel a fejezet elején feltételezett 10 ppm-es nagyságrendnek. Tapasztalatunk szerint ez az érték azonban nem állandó, még az is előfordulhat, hogy a korábban nagyobb frekvenciájú kvarc később kisebb frekvenciájú lesz, mint a másik.

Látható tehát, hogy viszonylag pontosan tudjuk érzékelni a csomagok beérkezésének időpontjait.

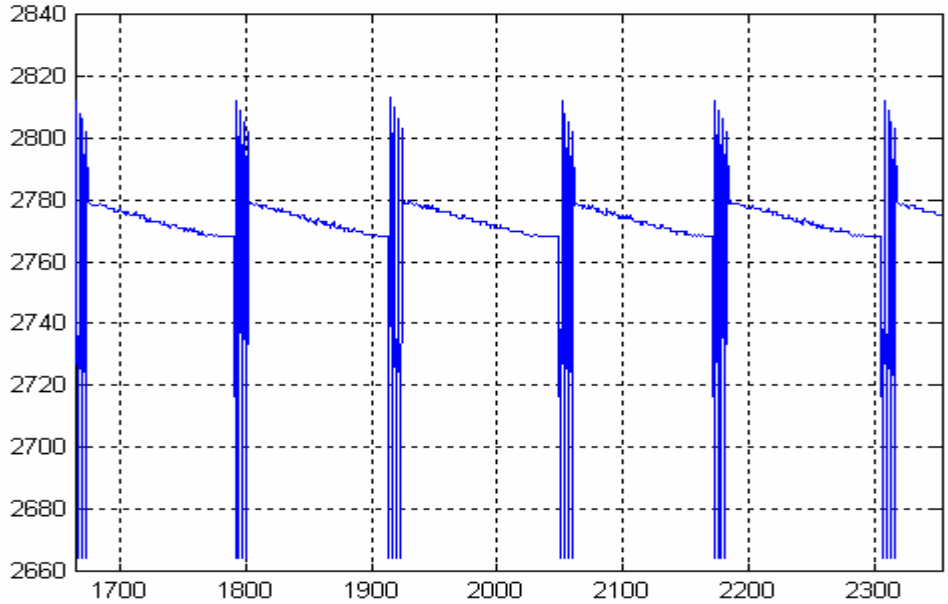
A szabályozási feladat megoldásában már csak egy nyitott kérdés marad, mégpedig az, hogy mi legyen az az érték, amire a számláló csomagok érkezésekor kiolvasott értékét szabályozni akarjuk, tehát egyszóval mekkora legyen az alapjel. A 9.6. ábrát szemlélve célszerűnek tűnik egy lineáris szakasz közepére helyezni az alapjelet. A megvalósított rendszerben ez az érték 1700. Ezen pont környezetében a görbe ugyanis folytonos, így ha nem teljesen pontos a szabályozás, és nem áll be az 1700-as alapjelre, akkor nem lesznek a szabályozandó értékben nagy ugrások. A 9.8., 9.9. és az 9.10. ábrán látható a szabályzási folyamat 1700-as alapjel, illetve 2780-as alapjel esetén. Ez utóbbinál látható, hogy a szabályozás igen rosszul működik, mivel itt a görbe nem folytonos. Ezen a részen a görbének szakadása van. Ez valószínűleg azért van így, mert ha a mintavétel után egy meghatározott idővel érkezik a rádión keresztül egy csomag, akkor a hatására generálódott esemény nem tud azonnal érvényre jutni más folyamatok miatt. Ha tehát ebbe a tartományba állítjuk az alapjelet, akkor nyilván nem működik jól a szabályzás, hiszen ebből a tartományból a mérés bizonyos időpontjaiban nem olvasunk ki értékeket, csak vagy felőle, vagy alóla, ami nullától különböző hibajelet eredményez. Ennek hatására viszont működésbe lép a szabályzó, és átlépünk az intervallum másik oldalára, ami szintén ellenirányú beavatkozáshoz vezet, stb…



9.8. ábra. Helyesen beállított szabályzó kezdeti beállása (behúzás)



9.9. ábra. Helyesen beállított szabályzó állandósult állapotban



9.10. ábra. Rosszul beállított szabályzó

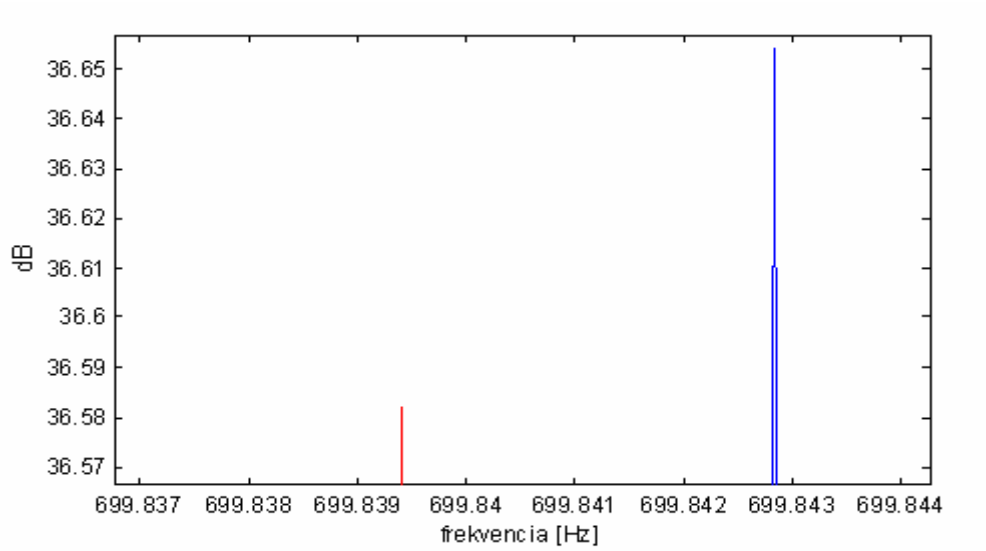
A szabályzás minőségével kapcsolatban megállapíthatjuk, hogy az 1700-as alapjel ±32-es tartományában tartja a szabályozandó értéket. Teljesen pontos szabályozást ezzel a módszerrel azonban nem is lehet készíteni. Ha ugyanis például a mintavételi időközt 4096-ról 4095-re állítjuk, akkor ez azt eredményezi, hogy a szabályozandó érték a következő csomag beérkezésekor már 25 egységet ugrik. Ha tehát 25 egységnél pontosabb szabályzót készítenénk, akkor fennállna az a lehetőség, hogy a szabályozott érték igen gyorsan ingadozna, ugyanis nem tudjuk pontosan beállítani az alapjelre, mivel csak 25-ös ugrásonként tudjuk állítani.

A konkrét szabályzó azért tud ±32-es tartományt tartani, mivel a leosztás változása a szabályozandó érték és az alapjel különbségének harminckettedével egyenlő. Ez azért kényelmes, mivel ez az osztás egy 5-szörös shifteléssel megvalósítható. Ekkor viszont mivel egész artimetikával dolgozunk, ha 32-nél kisebb számot osztunk 32-vel, akkor az eredmény nulla, tehát nem mutat eltérést az alapjeltől.

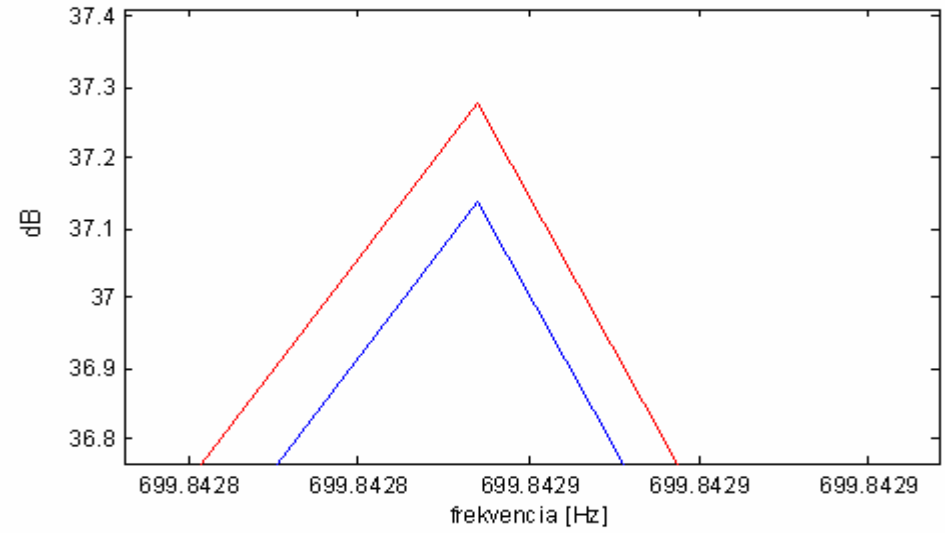
A szabályzás minőségére jellemző ±32 egységnyi ingadozás egy 64 egységnyi tartományt határoz meg. Ez a tartomány például 1800 Hz-es mintavételi frekvencia, tehát 4096-os órajel leosztás esetén $64/4096 \cdot 360^o = 5{,}625^o$-os ingadozást jelent az átviteli függvény becslésében, ami bőven a 90^o-os határon belül van.

Jó teszt lehet a szinkronizálás számára, ha megvizsgáljuk, hogy két csatornán felvéve ugyanazon szinuszjelet, az egyes csatornákon mekkorának mérjük az adott jel frekvenciáját. Mivel a mintavételi frekvenciák kis mértékben térnek el, így szinkronizálatlan esetben is csak kis mértékben fognak eltérni a becsült frekvenciák. Ehhez 1024^2, azaz körülbelül 10^6 mintát vettünk. Ez 1800 Hz-es mintavételi frekvencián körülbelül 10 perces mérési időtartamot

eredményez. A mért spektrumok a 9.11. és 9.12. ábrákon láthatóak. A gerjesztő jel 700 Hz-es szinuszjel. Kijelenthetjük, hogy a szinkronizáció sikeres, ugyanis szinkronizálatlan esetben eltér a két mote esetén a becsült frekvencia, míg szinkronizált esetben megegyezik. Az, hogy a mért jel nem pontosan 700 Hz, azért van, mert a szinuszjelet szolgáltató jelgenerátor, és a mote-ok órajel-generátora nem teljesen pontosak.



9.11. ábra. A két mote regisztrátumának spektruma szinkronizálatlan esetben

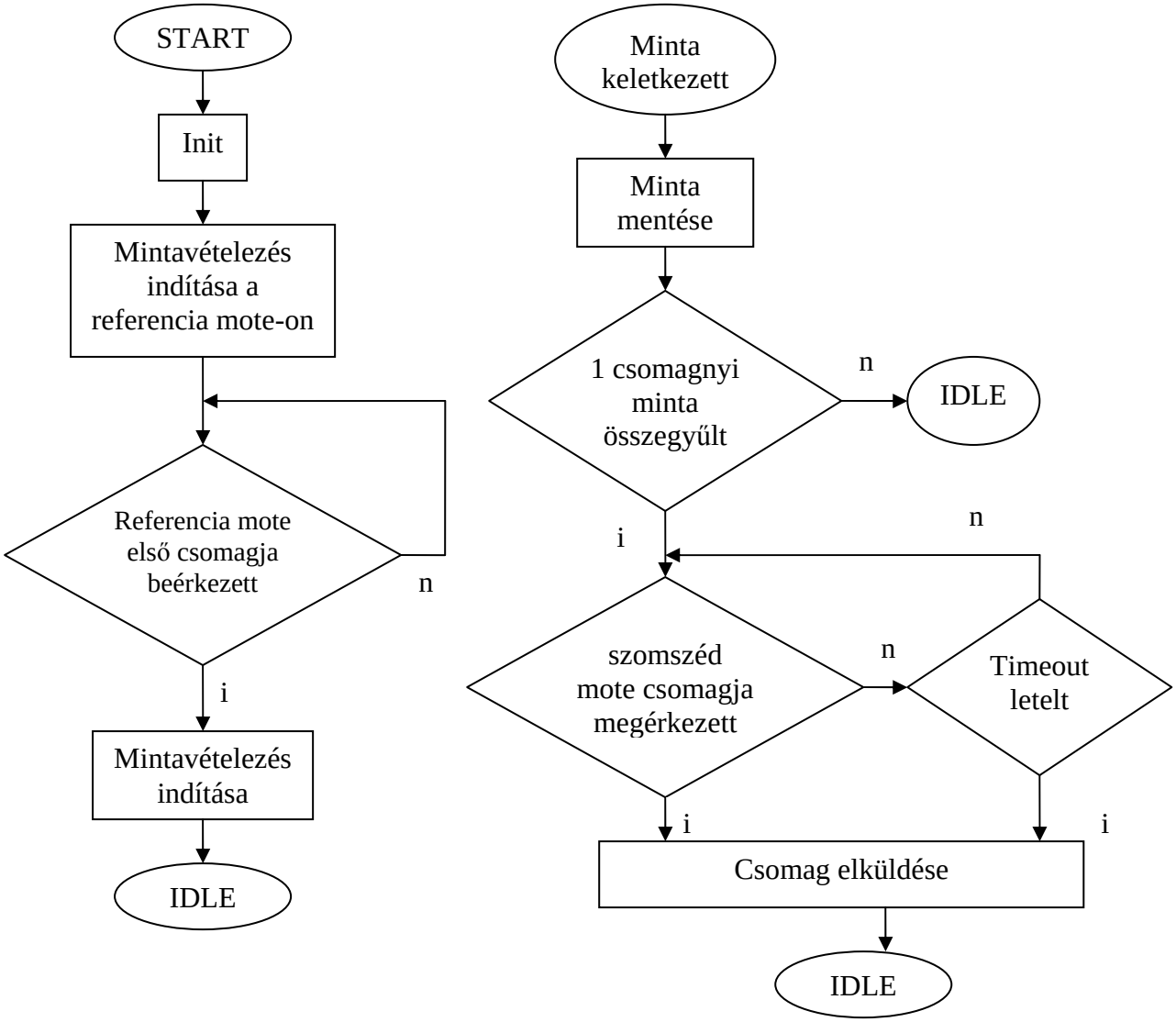


9.12. ábra. A két mote regisztrátumának spektruma szinkronizált esetben

9.3. Összefoglalás

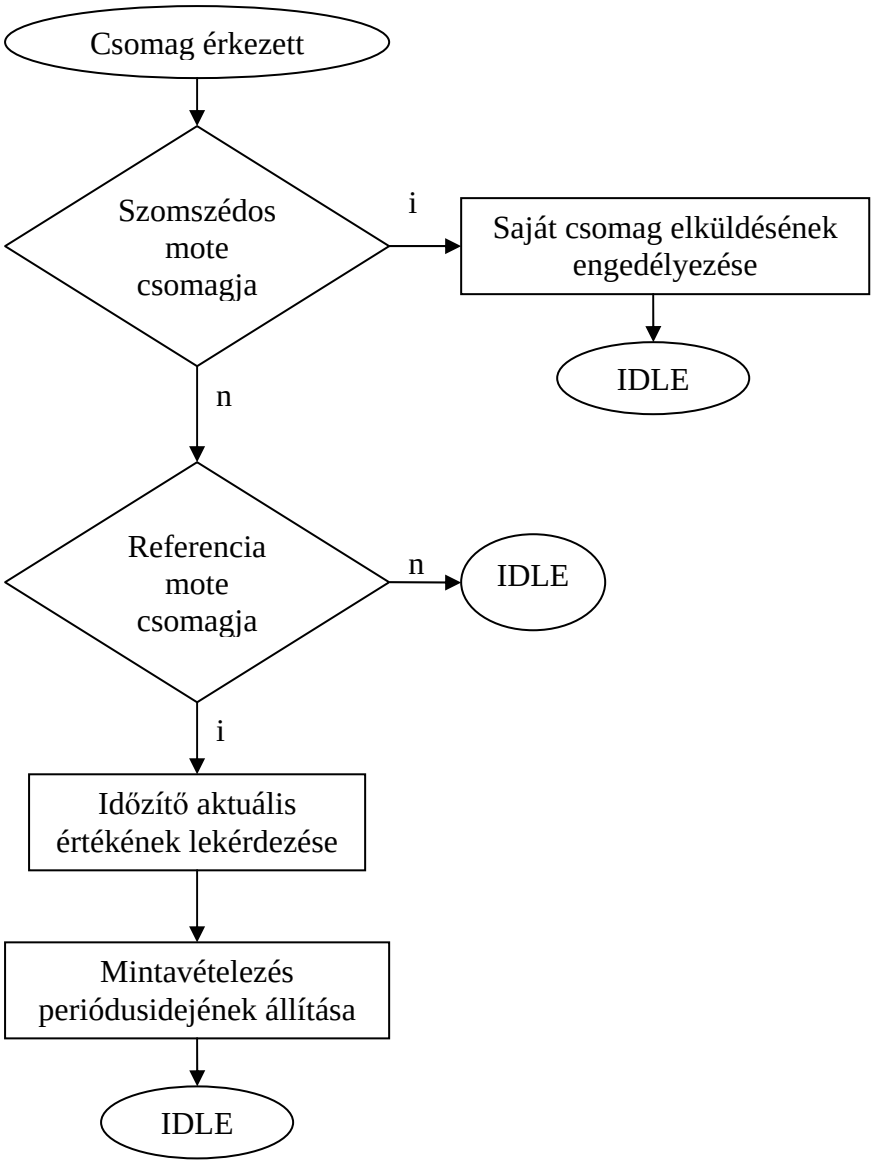
Az előző fejezetekben részletesen bemutattuk a hálózatban található mintavételezést végző mote-okon futó program főbb feladatait. Ahhoz, hogy átfogó képet kapjunk a program működéséről, segítséget nyújtanak az alábbi folyamatábrák.

A különálló folyamatábrák az ellipszisben megadott esemény hatására indulnak el, és a nyugalmi állapotba (IDLE) lépéssel zárulnak. A 9.13. ábrán látható két folyamatábra a program indítási szakaszát és a mintagyűjtés lefolyását ábrázolja.



9.13. ábra. A program indítási szakasza és a mintagyűjtés lefolyása

A 9.14. ábrán látható, hogy hogyan használják fel a mote-ok a hálózat egyes tagjainak adatcsomagjait a saját csomagjuk elküldésének időzítésére, valamint a szinkronizálásra.



9.14. ábra. A hálózati működés időzítése és szinkronizálás

10. A DSP és hálózat összekötése

A hálózat leírásánál láttuk, hogy a hálózati eszközök által továbbított adatokat egy átjáróként szolgáló mote gyűjti össze, és soros vonalon továbbítja a DSP felé. A DSP-n ezután egy keretprogram segítségével fogadjuk az adatokat, és bizonyos alapvető jelfeldolgozási lépés után átadjuk azt a felhasználói programnak. Ebben a fejezetben az egyes részfeladatok megoldásai olvashatók.

10.1. Az átjáró mote működése

10.1.1. A feladat általános bemutatása

Az átjáró mote gyakorlatilag nem szól bele a hálózat működésébe, csupán megfigyeli azt, és a hálózat tagjai által kisugárzott adatokat továbbítja az azokat feldolgozó eszköz felé. Mivel a hálózatot már szinkronizáltuk, így minden csomópont egyenlő intenzitással küldi csomagjait, tehát az átjárón csupán a fogadott csomagokat kell soros porton továbbítani, ha van olyan eszköz, mely a soros porton képes a TinyOS által megkívánt üzenetformátumot kezelni. Esetünkben ez például egy olyan PC volt, melyen egy megfelelő Java-s alkalmazás futott, mely figyeli a soros portot, és egy teljes csomag megérkezésekor értesíti a felhasználót erről az eseményről. A felhasználó ezek után feldolgozhatja az így megkapott adatokat. A fejlesztési munka első fázisában, a hálózat tesztelésekor, amikoris még nem a DSP felé küldtük az adatokat, hanem a PC felé, akkor ezt a struktúrát használtuk.

Miután azonban a hálózat megfelelően működött, és létre kellett hozni az átjáró és DSP közti kapcsolatot, a TinyOS formátumban történő adattovábbítás nem bizonyult elég optimálisnak. Ekkor ugyanis a DSP felé TinyOS formátumú üzeneteket küldünk, a DSP-nek tehát nemcsak az adatok fogadását kell megoldani, hanem fel kell dolgozni a csomagot, tehát el kell távolítani a kereteket, és a megfelelő időpontokban elő kell venni a megfelelő adatokat. Ez nem igazán kedvez a DSP program számára, ugyanis felesleges processzoridőt foglal le, mivel nem szorosan a jelfeldolgozáshoz tartozó feladatokat lát el ekkor.

Ehelyett ésszerűbbnek tűnt az átjáró mote-on alkalmazkodni a DSP-hez. Ez azt jelenti, hogy olyan eljárást kellett kidolgozni, mely nem csomagonként továbbítja az adatokat, hanem a hálózat mintavételi sebességének megfelelően úgy, hogy minden mintavételi ütemben továbbítja a hálózatban az aktuális ütemben összegyűjtött mintákat. Természetesen ezt csak valamekkora késleltetéssel tudja megtenni, hiszen a hálózatban meghagytuk a csomagszintű kommunikációt. Ennek megfelelően az átjáró mote-on össze kell gyűjteni a csomagokat, és azokat szét kell szedni bájtokra. Mivel a soros portot eddig a TinyOS kezelte, ezért egy új soros portot kezelő komponenst is létre kell hozni. Ezt a feladatot egyébként fejlesztési szempontból is egyszerűbb a mote-okon végezni. Ezeken ugyanis NesC-ben fejlesztjük a programokat, míg a DSP-n

assemblyben. A NesC magasabb szintű programnyelv, így például a tömbök használatával egyszerűbben megoldható a csomagok kezelése.

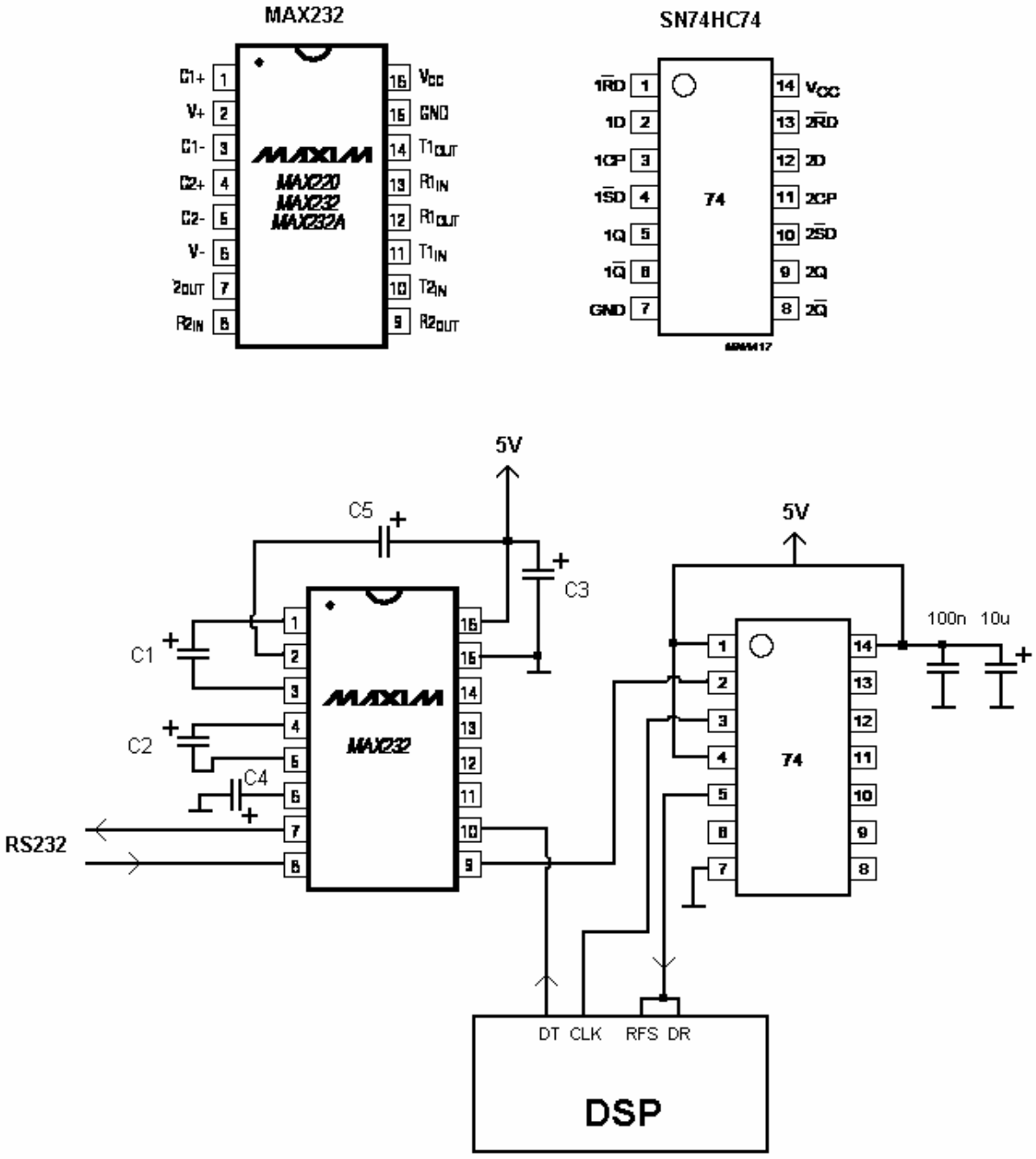
10.1.2. A mote és a DSP közötti fizikai kapcsolat

A két eszköz közötti kapcsolat aszinkron soros kommunikáción keresztül jön létre. Sajnos a DSP és a mote nem azonos tápfeszültséggel működik (5V és 3V), ezért a soros portok megfelelő lábait nem lehet közvetlenül összekötni, ezért szintillesztő áramkör használata szükséges. Elvileg lehetne a 3V-os és 5V-os jelszintek közötti konverziót is megvalósítani, de sokkal ésszerűbb, ha az átjáró mote-ot a programozó kártyára tesszük, amely a 0-3V-os jelszinteket az RS232-es jelszinteknek megfelelő szintekké alakítja. Az RS232-es szinteket 0-5V-os logikai szintekké alakító áramkörök pedig igen elterjedtek, ilyen például a MAX232-es típusú IC, amelyet mi is használtunk az illesztés során. Ennek segítségével a DSP-nek megfelelő jelszinteket kapunk.

Eddig nem említettük, hogy a DSP-n csak szinkron soros port található, azonban ez használható aszinkron soros portként is. Ennek érdekében a mote felől érkező, immáron 0-5V-os logikai szinteket egy SN74HC74 típusú D tárolóval szinkronizáljuk a DSP soros portjának órajeléhez, mely elérhető a fejlesztői kártya egyik kimenetén.

A [6] címen található dokumentáció ajánlásának megfelelően a biztonságos kommunikáció miatt a DSP soros portjának frekvenciáját a mote soros portja által használt működési frekvencia háromszorosára választjuk, így minden bit helyett három megegyező bitet olvasunk ideális esetben. Mivel azonban a két soros port nem szinkronizált, előfordulhat, hogy a DSP soros porti órajel adatbeolvasást időzítő élének megjelenésekor a mote még nem adta ki a megfelelő értéket. Ezért szükséges a megnövelt frekvencia. Ennek megfelelően minden bithármas közül csak a középsőt vesszük figyelembe. Az aszinkron port jelein túl a DSP-nek szüksége van egy receive frame sync. (RFS) jelre, mely szinkronjelként szolgál a bájtok átvitelének kezdetének jelzésére. Erre az aszinkron soros port START jelének lefutó élét használjuk fel, ez jelzi egy bájt átvitelének kezdetét.

A felhasznált alkatrészek lábkiosztása, illetve a illesztő áramkör kapcsolási rajza a 10.1. ábrán látható. Az alkatrészek adatlapjai az [7] [8] helyekről tölthetők le.



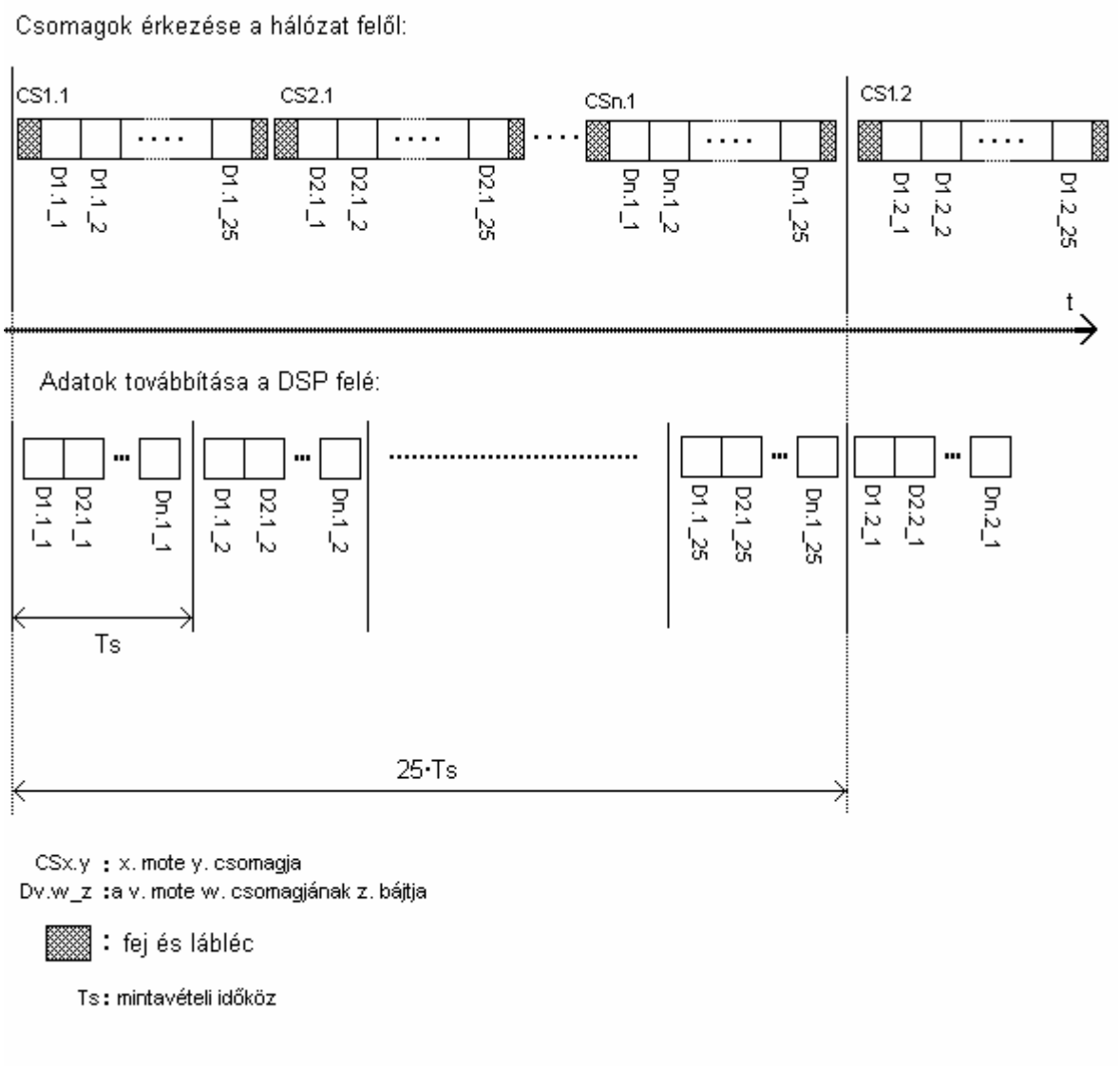
DT: Data Transmission: adatkimenet
CLK: soros port órajele
DR: Data Read : adatbemenet
RFS: Receive Frame Synchronisation: keretszinkronizáció

C1=C2=C3=C4=C5=1.0 uF

10.1. ábra. Soros porti illesztő áramkör

10.1.3. Csomagok fogadása és feldolgozása

A hálózatból a csomagok normális működés esetén a mote-ok azonosítóinak megfelelő sorrendben érkeznek. Az átjárónak is ilyen sorrendben kell továbbítani az adatokat, de már mintavételi időnként.

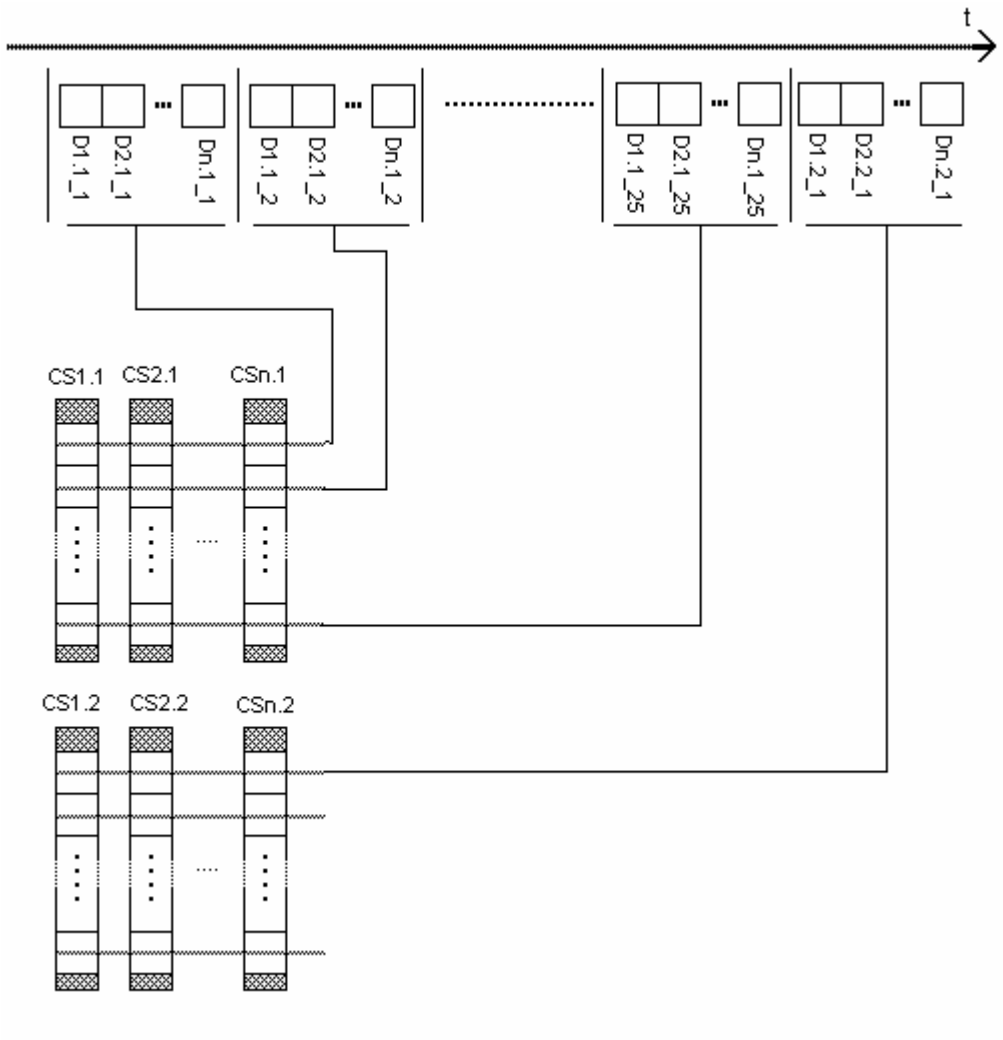


10.2. ábra. Az adatformátum átalakítása

A 10.2. ábrán látható, hogy hogyan kell a csomagokat átalakítani. Először is össze kell gyűjteni minden mote-tól azon csomagokat, melyek azonos időintervallumban gyűjtött mintákat tartalmaznak. Ezek után a mote-ok mintavételi frekvenciájával egyező sebességgel elő kell venni a csomagokból a soronkövetkező mintákat, amiket aztán továbbítani kell a soros porton a mote-ok

azonosítójának sorrendében. Miután a csomagokból továbbítottuk mind a huszonöt adatot, újabb csomagokat veszünk elő, és újból kezdjük a minták küldését a soros porton.

Ez ideális esetben nem lenne nehéz feladat. Ekkor ugyanis csak el kell tárolnunk a beérkező csomagokat egy pufferben, és a csomagokból megfelelő ütemben ki kell venni és továbbítni a mintákat. Azt is tudjuk, hogy az egymást követő csomagokat a token ring hálózatban egymás után következő mote-ok küldték, tehát ha N db mote van egy hálózatban, akkor a csomagokat N-es csoportokba kell gyűjteni, és ezek lesznek az azonos időintervallumban gyűjtött mintákat tartalmazó csomagok. Az, hogy hogyan képezzük ideális esetben a DSP-nek küldött adatokat a hálózatból érkező csomagokból, a 10.3. ábrán látható.



10.3. ábra. A DSP felé küldött csomagok kialakítása

Az ideális eset azt jelenti, hogy nincsenek kiesett csomagok a hálózatban. Ha viszont egy csomag kiesik, akkor az N-es csoportokban elcsúsznak a

csomagok, és az 1-től N-ig számozott mote-ok csomagjai nem lesznek sorrendben. Tehát például ha a 2. számú mote csomagja kiesik, akkor nem a [CS1.2 CS2.2 …. CSn.2] sorrendben továbbítjuk az adatokat, hanem mivel mindig N db csomagot gyűjtünk össze, az első mote következő csomagja is belekerül az előző csoportba. Ezek szerint [CS1.2 …. CSn.2 CS2.3] sorrendben továbbítjuk az adatokat, tehát a DSP-nek küldött csomagokban összekeverednek a csatornák adatai.

Ezt megelőzendő, egy kissé módosítjuk az algoritmust. A csomagokat nem közvetlenül a pufferből szedjük ki, hanem minden mote esetén elmentjük, hogy az általa küldött utolsó csomag hol foglal helyet a pufferben. Amikor továbbítjuk a soros porton az adatokat, akkor sorra megyünk a mote-okon, és a nekik megfelelő pufferpozícióból vesszük elő a csomagokat. Amennyiben minden csomag megérkezik, nincs különbség az előző algoritmushoz képest. Ha viszont egy csomag kiesik, akkor nem keverednek össze a csatornák adatai, ugyanis a kiesett csomagnak megfelelő helyen is továbbítunk valamit, mégpedig az adott mote-tól kapott régebbi adatokat. Ezzel tehát megoldottuk, hogy a DSP-hez küldött csomagokban az i. pozícióban az i. mote-nak megfelelő adatok vannak.

10.1.4. Adatok továbbítása a soros porton

Az előzőekben láthattuk, hogy elértük azt, hogy a mote-ok által küldött csomagokat a DSP számára kellemesebb formátumban tudjuk elküldeni. Ekkor az időben összetartozó adatokat küldjük el egy csomagban a mintavételi frekvencia sebességével. Ezt a frekvenciát azonban az átjáró mote generálja, így a kvarcok eltérése miatt ez nem fog megegyezni a hálózati szinkronizált mintavételi frekvenciával. Emiatt viszont nem egyenlő sebességgel érkeznek és fogynak az adatok, ami nyilvánvalóan hibás működést eredményez. Mivel azonban a hálózatban már megoldottuk a szinkronizációt, ezt csak be kellett építeni az átjáró mote-ba is, tehát a DSP a mintavételi frekvencia ütemében kapja a mote-ok által küldött adatokat.

Eddig nem hangsúlyoztuk, de mivel a DSP-nek mindenféle keretezés nélkül küldjük az adatokat, nem használhatjuk a TinyOS `GenericComm` komponensét, mivel az a soros porton a TinyOS keretezésnek megfelelően küldi az adatokat. Ezért készíteni kellett egy olyan komponenst, mely közvetlenül képes bájtokat küldeni a soros porton keresztül. A soros porton egyébként már korábban is kellett átalakításokat végezni, ugyanis meg kellett növelni az átviteli sebességet 57.6 kbps-ről 112.5 kbps-re, hiszen így nagyobb adatátviteli sebesség, így nagyobb mintavételi frekvencia valósítható meg. A mote-on lévő UART egyébként képes lenne nagyobb sebességre is, de a DSP csak ekkora sebességgel tud adatot fogadni aszinkron soros portról.

Annak ellenére, hogy az adatokat 8 bites felbontással továbbítjuk, a mote UART-ját 9 bites módban használjuk. Ez ugyan csökkenti a csatorna kihasználtságát, viszont elengedhetetlen a DSP és mote közötti biztonságos

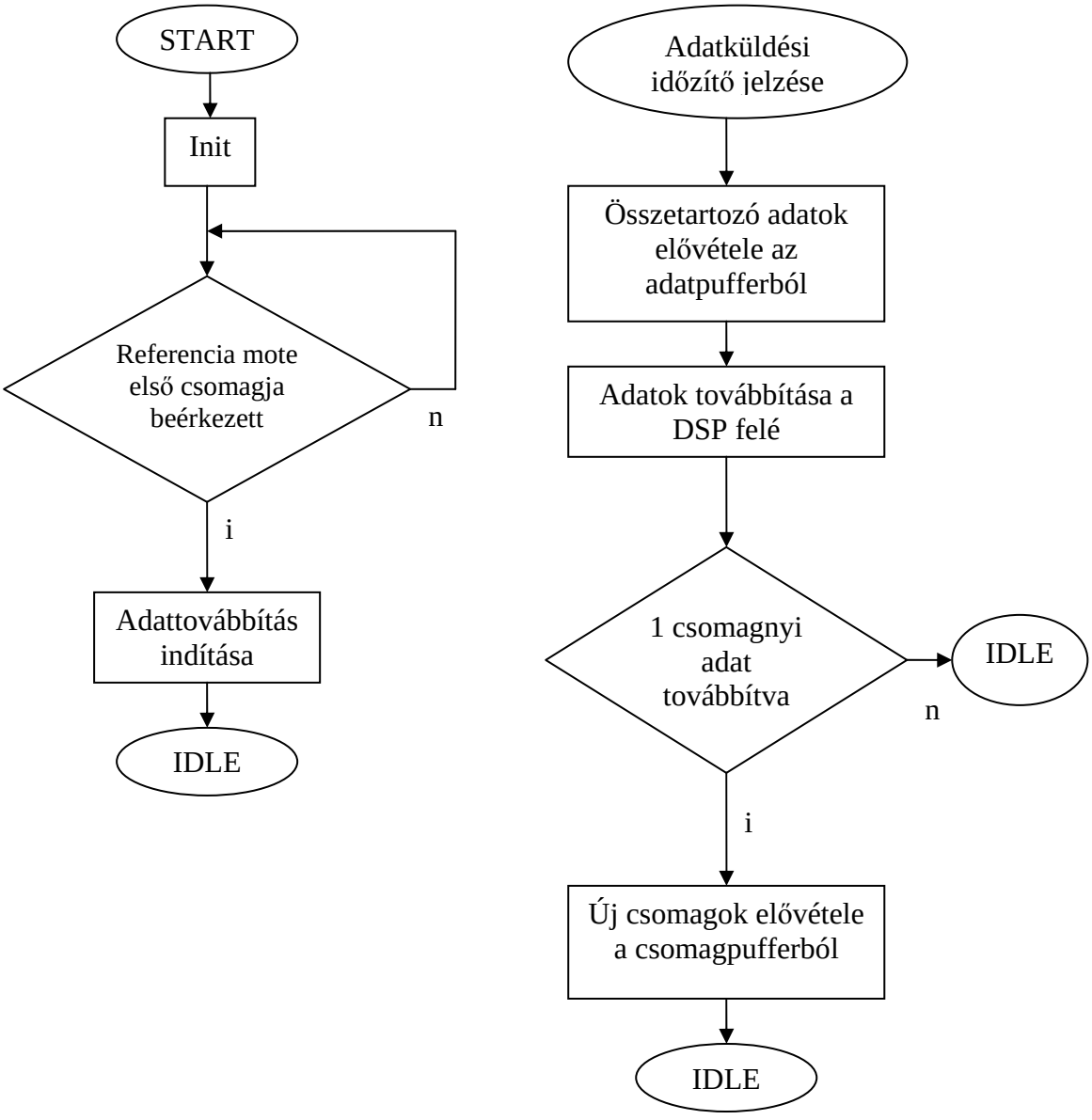
kommunikációhoz. Ha ugyanis elindítjuk a mintavételezést a hálózatban, akkor az átjáró mote egyből elkezdi továbbítani a soros porton a csomagokat. A DSP keretprogram az elindítása után N bájtos egységenként kezdi beolvasni az adatokat a soros port-ról, amennyiben a hálózatban N db mote van. Ha viszont rosszkor kezdi el olvasni az adatokat, akkor lehet, hogy nem egy csomag elején kezdi el a vételt, így nem az első mote adatát olvassa be az első helyen. Ez azt eredményezi, hogy összekeverednek a csatornák adatai.
Nézzünk erre egy példát. Jöjjenek az adatok
MD1, MD2, MD3, MD1, MD2, MD3, MD1, MD2…
sorrendben, ahol MDx az x. mote adatait jelöli.

Jelöljük CH1, CH2, CH3 –mal a DSP-n azokat a változókat, amelyekben az aktuális MD1, MD2, MD3 adatokat tároljuk, méghozzá szigorúan ebben a sorrendben. Ezekbe a változókba folyamatosan olvassuk be a soros porton érkező adatokat. Ha tehát a soros porton érkező csomagokat az elejüktől kezdve fogadjuk, akkor a CH1-be MD1, a CH2-be MD2, a CH3-ba pedig MD3 kerül. Ez a normális működés.

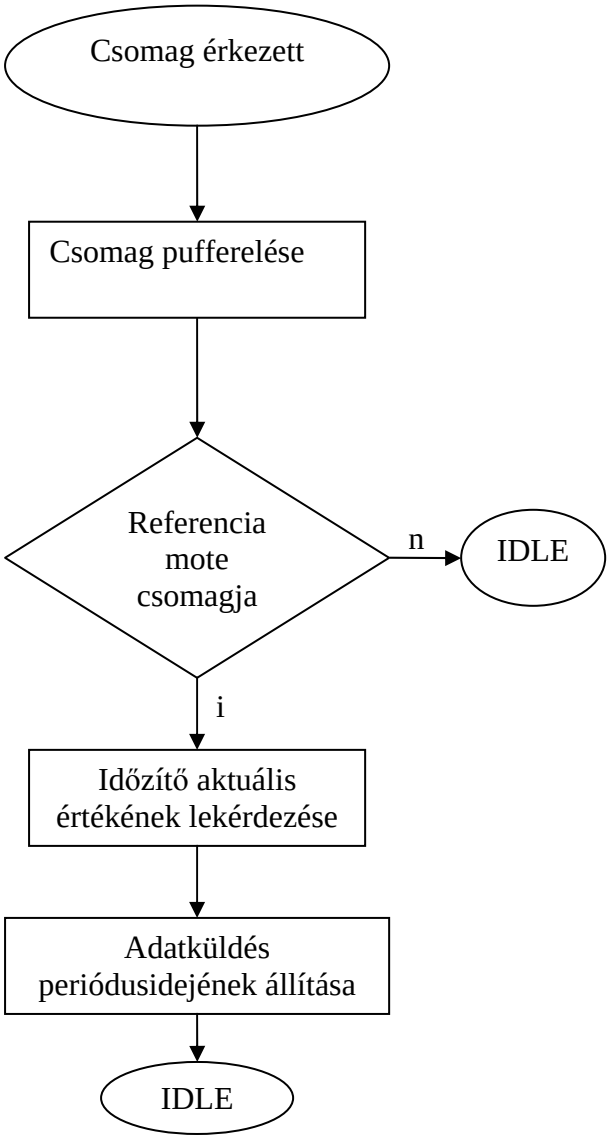
Amennyiben nem a csomag elején kezdjük beolvasni az adatokat, hanem például MD2-vel kezdve, akkor a CH1-be MD2, a CH2-be MD3, a CH3-ba pedig MD1 kerül, tehát felcserélődnek a csatornák.

Ennek alapvetően az az oka, hogy a mote által küldött csomagok határait végül is nem lehet megkülönböztetni, mivel nincs keretezés. A csomagon belül sem különbözteti meg semmi az egyes mote-ok adatait, tehát gyakorlatilag a csomagok csak logikailag léteznek, fizikailag semmi nem határolja azokat, a soros vonalon csak egy folytonos jelfolyamot észlelünk. A soros porton a 8 bites adatbájton kívül elküldött 9. bitet tehát keretezésre használjuk fel. Konkrétan minden csomag első adatában a 9. bit 1-es értékű, a többiben a 9. bit 0-s, így tehát megtalálható a keret eleje.

Ezzel áttekintettük az átjáró mote-on futó program fő komponenseinek a működését. A következő ábrákon a program folyamatábrái láthatók, így egészében is áttekinthetjük a működést.



10.4. ábra. Inicializációs, és adattovábbítást vezérlő rész



10.5. ábra. Adatfogadási és szinkronizációs folyamat

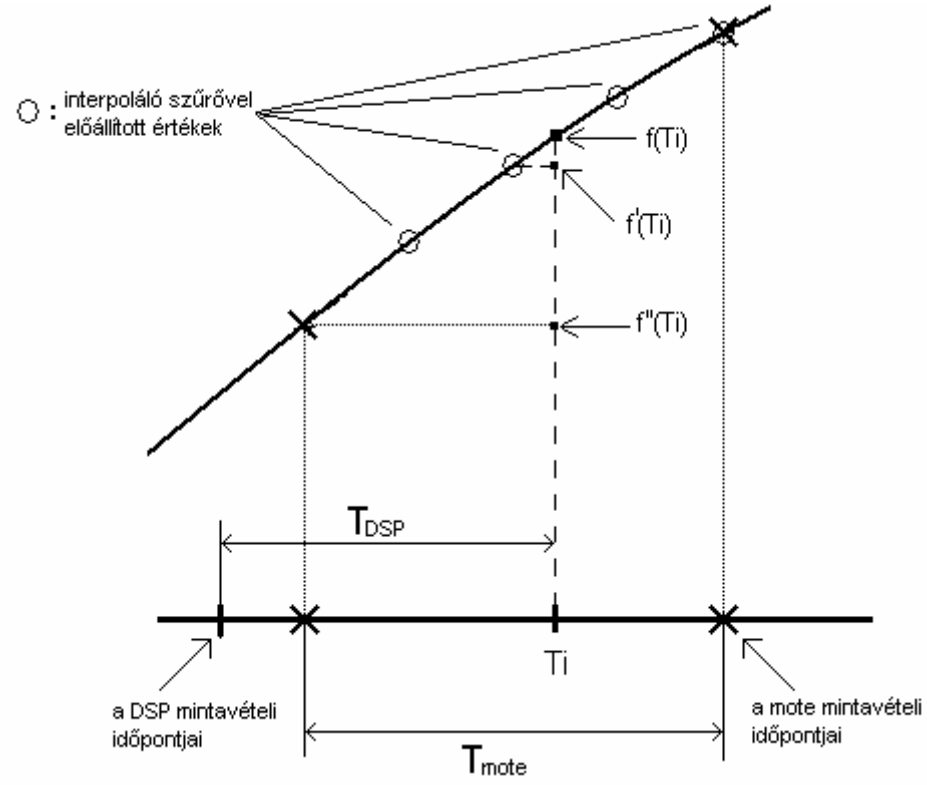
10.2. Adatfogadás a DSP-n

A hangminták továbbításában már eljutottunk addig a pontig, hogy a hálózatban összegyűjtött adatokat sikerült a mintavétel ütemével továbbítani soros porton keresztül a DSP felé. Utolsó lépésként el kell juttatnunk a DSP soros portján beérkezett adatokat a felhasználói program felé. Problémát okoz azonban, hogy az adatok nem olyan ütemben érkeznek, mint amilyenben a DSP-n futó alkalmazásoknak szükségük van rájuk. Ehhez ismerkedjünk meg kicsit a DSP-s programok struktúrájával.

Esetünkben a programok a következőképpen épülnek fel: a program kezdetén elvégezzük a változók és a rendszer inicializálását, aztán engedélyezzük a megszakításokat. A jelfeldolgozó algoritmusokat periodikusan hajtjuk vére, mégpedig a fejlesztői kártyán található AD-DA kodek által adott megszakítások alkalmával. Ekkor kiolvassuk a kodek AD-ja által előállított digitális értékeket, ezen új értékek felhasználásával lefuttatjuk az algoritmust, és a végeredményként kapott értékeket kiküldjük a kodek DA-ja számára.

Ehhez a struktúrához szeretnénk illeszteni a mote-ok által összegyűjtött adatok beolvasását, tehát azt szeretnénk, hogy a mote-ok a kodekkel egy időben szolgáltassák az adatokat. Ez a probléma is szinkronizációs feladathoz vezet. Ez azonban már nem oldható meg az eddig ismertetett algoritmussal. Ahhoz ugyanis vagy a kodek, vagy a hálózat mintavételi frekvenciáját kellene hangolni, ami nem lehetséges. A kodek ugyanis csupán bizonyos előre definiált frekvenciákon tud üzemelni, és működésébe egyébként sem tudnánk beavatkozni, hiszen az egy kész hardver, módosítására nincs lehetőség. A hálózati mintavételi frekvenciát szintén nem tudjuk befolyásolni, ugyanis akkor az átjáróként szolgáló mote-ról szinkronizáló üzeneteket kellene küldeni, ez viszont, mint már említettük csökkentené a hálózat hatékonyságát, így nem építettük be a hálózati protokollba.

Más módot kell tehát találni a szinkronizálásra. Erre egy jó megoldás, ha a mote-okról a hálózat mintavételi frekvenciájának megfelelően olvassuk be az adatokat, és valamilyen módszerrel a meglévő adatokból megbecsüljük, hogy mennyi lehet a minták értéke a kodek által generált megszakítások idejében. Ez a módszer az interpoláció.



10.6. ábra. Az interpoláció bemutatása

Az interpolációnak több fajtája ismeretes. Legegyszerűbb formája az, amikor a becsült érték mindig a legutolsó beolvasott érékkel egyenlő. Ez nulladrendű tartó módszere, ami tehát azt jelenti, hogy a legutolsó minta értékét tartjuk ki becslőként az újabb minták beérkezéséig. Ez igen durva becslés, bár ennek ellenére széles körben használt módszer, mégha sokszor nem is tudatosul bennünk, hogy ezt használjuk. A 10.6. ábrán *f* ''(Ti)-vel jelölt érték az *f*(Ti) nulladrendű tartóval becsült értéke.

A másik végletet jelenti az interpolációban, ha tökéletesen vissza tudjuk állítani a jelet minden megkívánt időpontban. Ehhez használhatjuk az interpoláló szűrőt. Ez egy olyan szűrő, melyet az eredeti fs mintavételi frekvencia valamilyen egész számú többszörösén (fs'=N·fs) működtetünk, és specifikációja a következő:

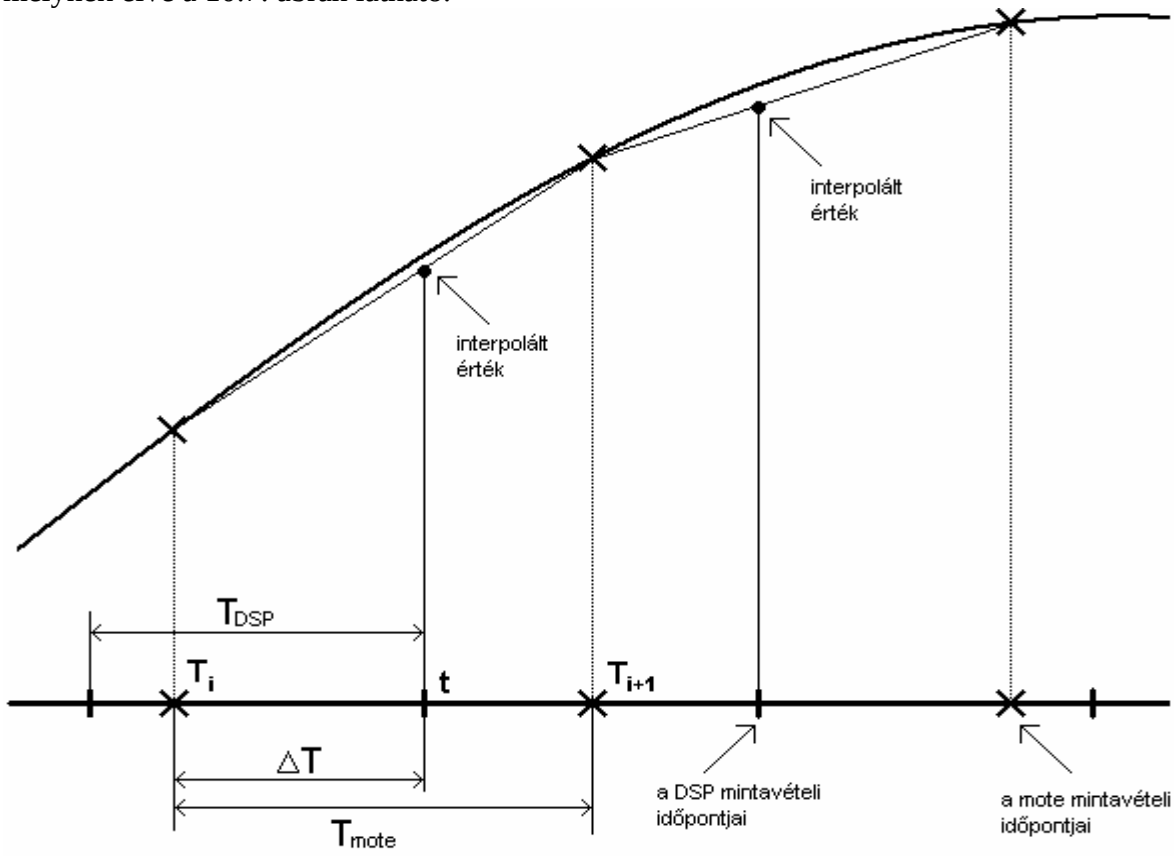
$$H(f) = \begin{cases} 1 & \text{ha f} < \text{fs'/N} \\ 0 & \text{egyébként} \end{cases}$$

Ahol fs' az a frekvencia, melyen az interpoláló szűrő működik. Ez az eredeti mintavételi frekvencia N-szerese. Tehát immáron az eredeti Ts időbeni felbontás helyett Ts/N felbontással ismerjük a jel értékét, ez azt jelenti, hogy az *f*(*n*·Ts) függvény helyett az *f*(*k*·Ts/N) függvényt ismerjük. A 10.6. ábrán egy négyszeres

mintavételi frekvencián működő interpoláló szűrő által előállított értékeket körrel jelöltük. Ez akkor jelenthet tökéletes megoldást, ha az alkalmazásunkban pontosan ilyen időpontokban van szükségünk a mintákra. A mi esetünkben viszont ez nem teljesül, hiszen a mote-ok és a kodek aszinkron járnak, így a kodek által időzített DSP program a mote-ok számára kötetlen időpontokban kér mintákat. Ennek ellenére nem elvetendő megoldás az interpoláló szűrő használata. Ha ugyan nem is érhetünk el vele hibátlan becslést, de mindenképpen javíthatjuk vele a becslés pontosságát. Ennek megértéséhez nézzük a 10.6. ábrát. Ha a Ti-ben szeretnénk becsülni a függvény értékét, akkor például interpoláló szűrő segítségével kiszámíthatjuk a Ti-t megelőző (k·Ts/N) időpontban a függvény értékét, és aztán erre alkalmazzuk a nulladrendű tartót. Így kapjuk meg az f ’(Ti) értéket, ami láthatóan jobb becslést ad, mint az egyszerű nulladrendű tartót alkalmazó f ’’(Ti) érték.

Mégis több probléma is van az interpoláló szűrő alkalmazásával. Egyrészt a szűrés folyamata értékes processzoridőt vesz el, valamint a szűrés alkalmazásával az eredeti jel nagy késleltetést szenvedhet, ami a zajcsökkentő rendszer dinamikáját rontja.

Egy kompromisszumos megoldást jelenthet a lineáris interpoláció alkalmazása, melynek elve a 10.7. ábrán látható.



10.7. ábra. Lineáris interpoláció szemléltetése

Ebben az esetben, ha egy [T_i T_{i+1}] intervallumba eső t időpontban szeretnénk becsülni f(t) értékét, akkor azt az $f(t) = f(T_i) + \left[f(T_{i+1}) - f(T_i)\right] \cdot \frac{t - T_i}{T_{i+1} - T_i}$ kifejezés alapján számolhatjuk. Ezen képlet másik formája: $f(t) = (1-a) \cdot f(T_i) + a \cdot T_{i+1}$, ahol $a = \frac{t - T_i}{T_{i+1} - T_i}$. Mi a képlet első formáját használjuk arra, hogy a kodek megszakításainak időpontjaiban előállítsuk a mote-okról érkező adatok adott időpontoknak megfelelő értékeit, mivel az kevesebb művelettel kiszámítható. Ehhez ismernünk kell a (T_{i+1}-T_i), az ábrán ΔT-vel jelölt (t-T_i), valamint az f(T_{i+1}) és f(T_i) mennyiségeket. (T_{i+1}-T_i) egyenlő a mote-ok mintavételi időközével: T_{mote}, mely egyébként egy ismert konstans érték. A ΔT idő mérhető a DSP-n található időzítő segítségével, f(T_{i+1}) és f(T_i) pedig az átjáró által soros porton küldött értékek.

Kissé nehezíti a feladatot, hogy ezt az interpolációt több csatornára is el kell végeznünk, hiszen az adatgyűjtő hálózatban több mote is található. Ez gyakorlatilag nem jelent problémát, hiszen a ΔT és T_{mote} minden csatornára megegyezik, csupán az egyes csatornáknak megfelelő f(T_{i+1}) és f(T_i) értékeket kell csatornánként külön kezelni.

Ez a módszer azonban tartalmaz még egy kis nehézséget, ugyanis az interpolációhoz ismernünk kellene f(T_{i+1}) értéket, amely még nem érkezett meg. Ennek kiküszöbölésére f(T_{i+1}) és f(T_i) helyett f(T_i) és f(T_{i-1}) értékeket használjuk, mely egy mintavételnyi késleltetést hoz be a rendszerbe.

Az interpoláció úgy tekinthető, mintha a mote-ok által küldött értékekből lineáris interpolációval egy folytonos függvényt állítottunk volna elő, melyet aztán a DSP mintavételi frekvenciájának megfelelően újra-mintavételeztünk. Így oldottuk fel azt a problémát, hogy az átjáró felől eltérő sebességgel érkeznek az adatok, mint ahogyan azokat a DSP feldolgozza.

Ezzel a fejezettel bezárult a feladatnak azon része, melyben meg kellett oldanunk a hálózatban található mote-okon lévő mikrofonok jeleinek továbbítását egy DSP-n futó program számára. A hálózatban meg kellett valósítani a szinkron történő pontos mintavételezést. Az összegyűjtött mintákat a hálózaton keresztül el kellett juttatni egy átjáróként szolgáló mote-hoz, mely az adatokat a DSP-felé megfelelő formátumban továbbítja soros porton keresztül, melyhez megterveztük a megfelelő szintillesztő áramkört. A megvalósított rendszerben sikerült kialakítani egy olyan keretprogramot, mely a felhasználó elől elrejtve az ezzel járó nehézségeket, a kodek mintavételi frekvenciájának megfelelő ütemben továbbítja a mote-októl érkező adatokat úgy, hogy minden esetben ismerjük, hogy melyik mintát melyik mote állította elő.

11. A zajcsökkentő program megvalósítása

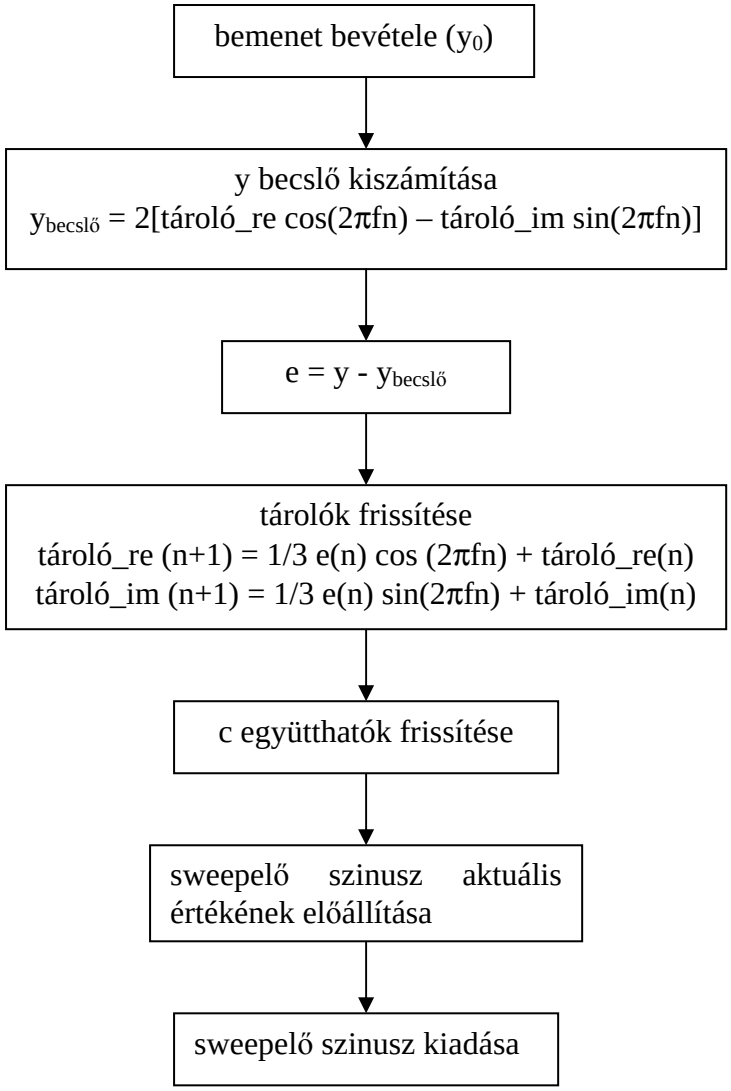
11.1. Bevezetés

A zajcsökkentő algoritmusokat az ADSP 21061-es EZ Kit Lite kártyán valósítottuk meg. A specifikáció valamint az algoritmus számítási igénye alapján f_s = 2 kHz-et választottuk mintavételi frekvenciának. Mivel a kártyán lévő AD-átalakító nem állítható 8 kHz-nél kisebb mintavételi frekvenciára, ezért négyszeres decimálást hajtottunk végre. Azért, hogy a jelfeldolgozó algoritmusok végrehajtására elég idő álljon a rendelkezésünkre, a jelfeldolgozást nem a timer megszakítás rutinban, hanem a főprogramban végeztük. A timer megszakítás rutinban csupán a bejövő adatok: referenciajel, hibamikrofonok jelei beolvasását és a kimenő adatok: beavatkozó hangszórók jelei kiírását végeztük. Az általános áttekintés után bemutatjuk az átvitelifüggvény-mérő, az egycsatornás zajcsökkentő on-line identifikációs, valamint a kétszer kétcsatornás (két hibamikrofon és két beavatkozó hangszóró) zajcsökkentő programot.

11.2. Az átviteli függvény mérő program

Az átvitelifüggvény-mérő program a 4. fejezetben leírtakkal összhangban két részből áll: az egyik rész „ugráló” szinuszos gerjesztőjelet állít elő a kimenet számára (azaz a szinuszjel ferkvenciája bizonyos időközönként ugrásszerűen változik), a másik rész pedig egy AC rezonátorpár és egy DC rezonátor segítségével az identifikálást végzi.

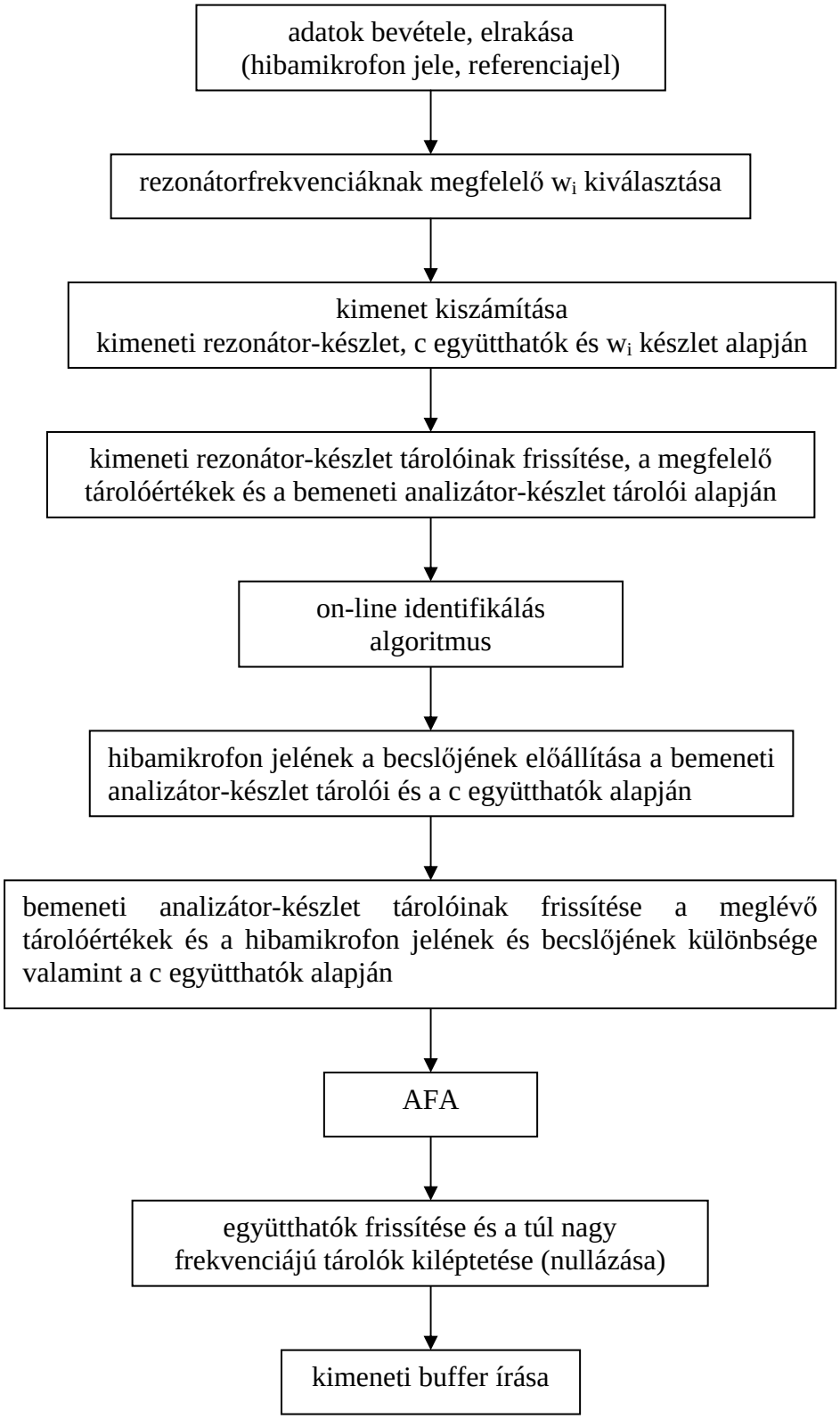
Mivel valós jeleket dolgozunk fel, ezért a rezonátoros struktúra megvalósításánál a következő „trükköt” vetettük be. Az AC rezonátorpár helyett egy darab AC rezonátort valósítottunk meg, és a hibajel kiszámításakor ennek a rezonátornak a tárolójának a valós részének a kétszeresével számoltunk. Ezt azért tehettük meg, mivel valós jelek esetén a pozitív frekvencián lévő jel a negatív frekvencián lévő jel konjugáltja, így összegük a valós rész kétszerese lesz. A program folyamatábrája a 11.1-es ábrán látható.



11.1. ábra. Az átviteli függvény mérő program folyamatábrája

11.3. Egycsatornás zajcsökkentő rendszert on-line identifikációval megvalósító program

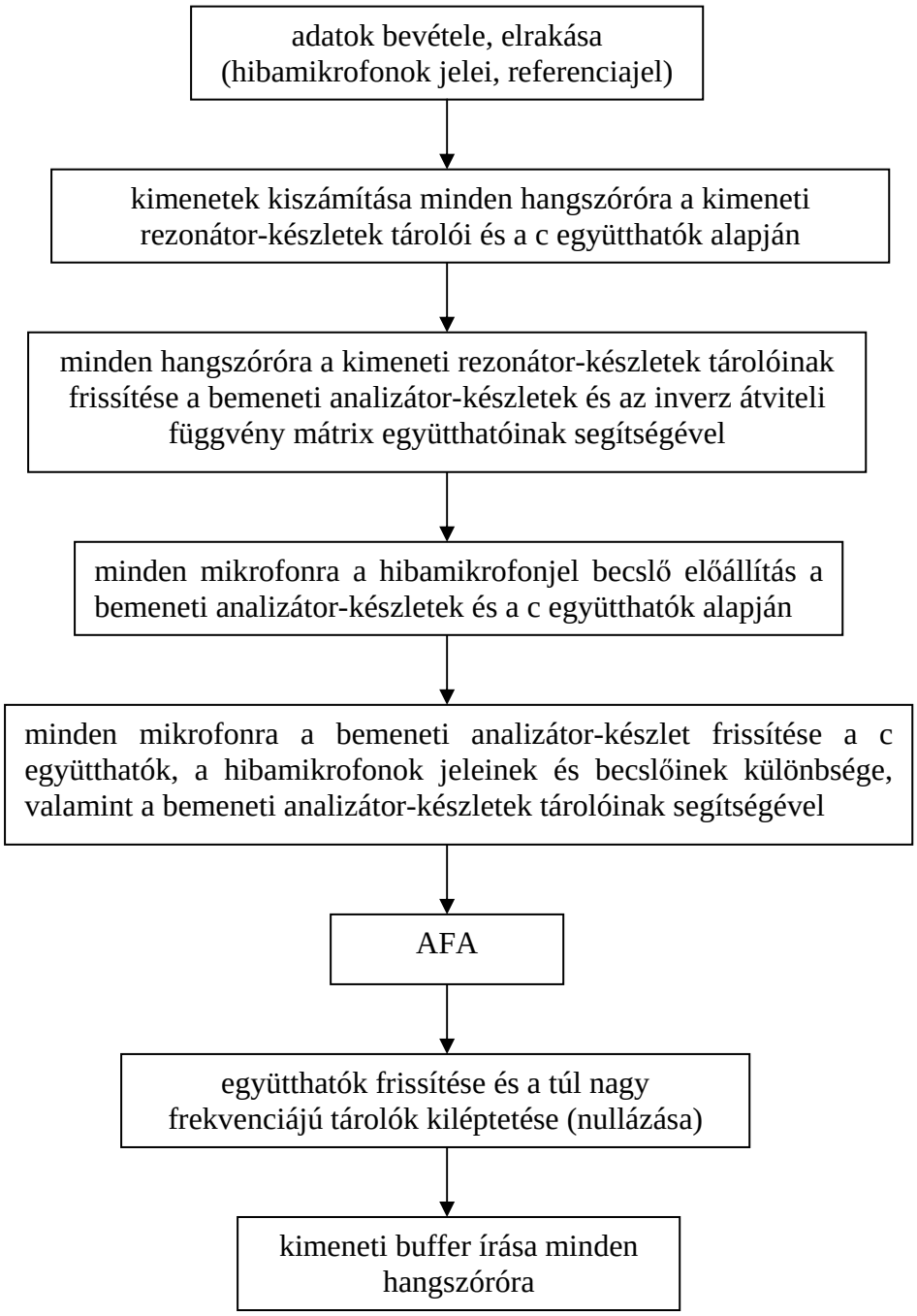
Az egycsatornás zajcsökkentő programot úgy valósítottuk meg, hogy egy bemeneti analizátor-készlet, egy kimeneti rezonátor-készlet és egy ezektől független, az AFA algoritmus számára szükséges rezonátorkészletünk volt. Az AFA algoritmushoz szükségünk volt egy referenciajelre a zajforrásból. A programot úgy konstruáltuk meg, hogy az AFA algoritmus a rezonátor-frekvenciákat a c együtthatók változtatásán keresztül állította. Így tulajdonképpen egy előrecsatolt rendszert valósítottunk meg. A zajcsökkentő struktúra megvalósításánál a jelmodell alapú megfigyelő integrátoros modelljét használtuk. Az on-line identifikációt megvalósító algoritmus során kihasználtuk azt, hogy egycsatornás esetben a kimeneti rezonátorkészlet tárolói abszolút értékének deriváltja a bemeneti analizátorkészlet tárolóinak abszolút értéke. A program folyamatábráját a 11.2-es ábra szemlélteti.



11.2. ábra. Az egycsatornás zajcsökkentő program folyamatábrája

11.4. Többcsatornás zajcsökkentő program megvalósítása

A többcsatornás zajcsökkentő program tulajdonképpen az egycsatornás általánosítása programozási értelemben. Felépítése nagyjából követi az egycsatornás rendszerét azzal a különbséggel, hogy egymásba ágyazott ciklusokat tartalmaz aszerint, hogy hány hangszóró illetve mikrofon van a rendszerben. A program folyamatábrája a 11.3-as ábrán látható.



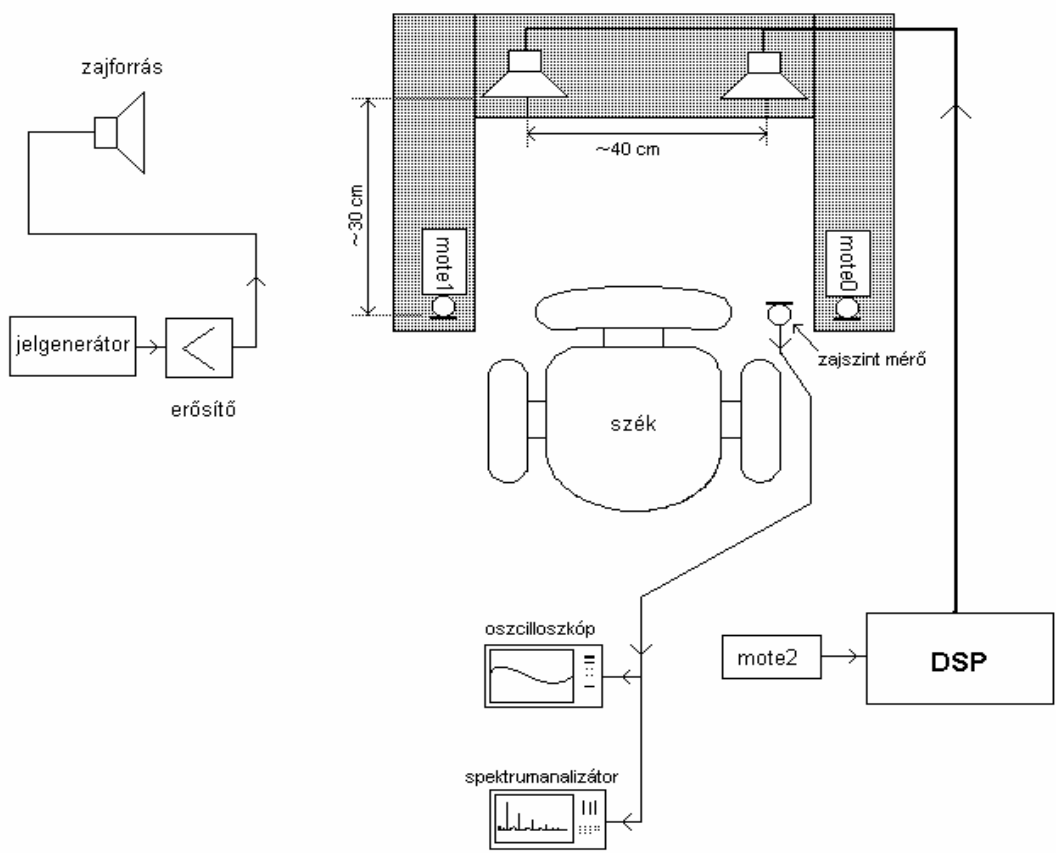
11.3. ábra. A többcsatornás zajcsökkentő program folyamatábrája

12. Az elkészült rendszer teszteredményei

12.1. Tesztkörnyezet bemutatása

A megvalósított zajcsökkentő rendszert több szempontból is teszteltük. A rendszert mind tranziens, mind állandósult állapotban is különböző mérőszámokkal jellemezhetjük.

A méréseket egy két bemenetű és két kimenetű rendszerre végeztük el. Habár a hálózat elvileg alkalmas lenne kettőnél több csatornáról történő adatgyűjtésre, a DSP kártyán található kodeknek csak két DA csatornája van, így csak két hangszórót tudunk meghajtani. Teljes zajelnyomás viszont általánosan csak akkor érhető el, ha egy rendszerben legalább annyi hangszóró van, mint ahány mikrofon. Ezért végeztük a tesztelést kétcsatornás esetre. A kísérleti elrendezést felülnézetből szemlélteti a 12.1-es ábra:



12.1. ábra. Zajelnyomó rendszer tesztkörnyezetének vázlata

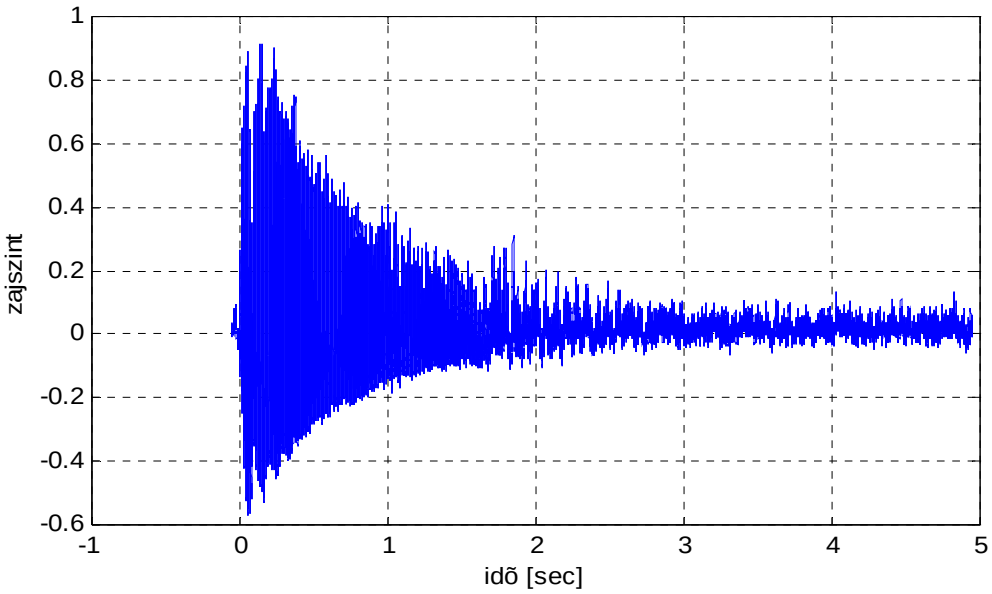
A tesztkörnyezet kialakításával egy olyan gyakorlati elrendezést szimuláltunk, melyben az érzékelő mikrofonok egy karosszék fejtámlájánál fülmagasságban helyezkednek el.

A méréseket egy zajszintmérő mikrofonnal mértük, melyet egy adott ponton elhelyezve mérhető az ott lévő zajszint. A mikrofon jelét spektrumanalizátorra illetve oszcilloszkópra vezetve mérhető a zajjel frekvenciatartománybeli illetve időtartománybeli viselkedése. A méréseket realisztikus körülmények között végeztük és nem süketszobában, így megvizsgálható, hogy hogyan viselkedik a rendszer a valós felhasználás esetén is jelentkező zavarjelekre, mint például beszélgetés, ajtócsapás… A rendszer ilyen körülmények között hosszú távon is stabilan működött.

Az elrendezés vázlata a 12.1. ábrán látható. A zajjel mintáit a mote0 és mote1 továbbítja a mote2 felé, amely soros porton továbbküldi ezeket a DSP-nek. A DSP-n futó zajelnyomó algoritmus szolgáltatja a beavatkozójeleket, melyeket vezetéken továbbítunk a hangszórók felé. A zajjelet egy jelgenerátor szolgáltatta, melyen különböző hullámformájú jel (szinusz, háromszög, négyszög, fűrészjel) előállítható.

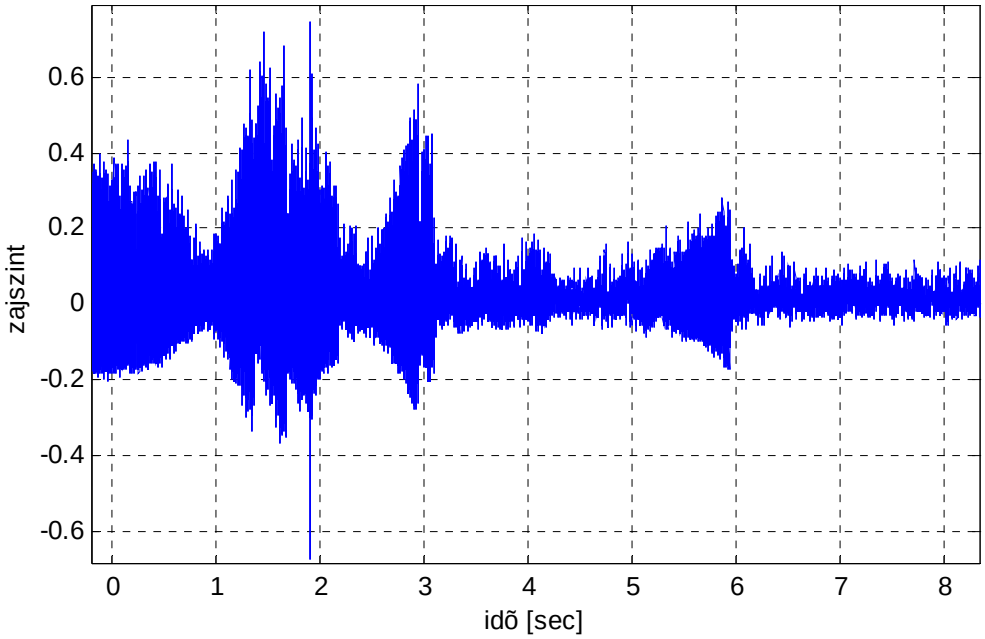
12.2. Időtartománybeli eredmények

Tranziens üzemben egy fontos adat a beállási idő, ezzel jellemezhető a rendszer dinamikája. Méréseink során egy kétcsatornás, és egy on-line identifikációs algoritmussal kiegészített egycsatornás rendszert teszteltünk. Ennek eredményeit mutatjuk be a következőkben.



12.2. ábra. kétcsatornás rendszer beállása

A 12.2. ábrán egy kétcsatornás zajcsökkentő rendszer egyik csatornájánál mért beállást láthatjuk. Megállapítható, hogy a beállási idő körülbelül 2 másodperc, ami egy elfogadható értéknek számít, és a vezetékes adattovábbítás esetén sem tapasztaltunk gyorsabb működést.

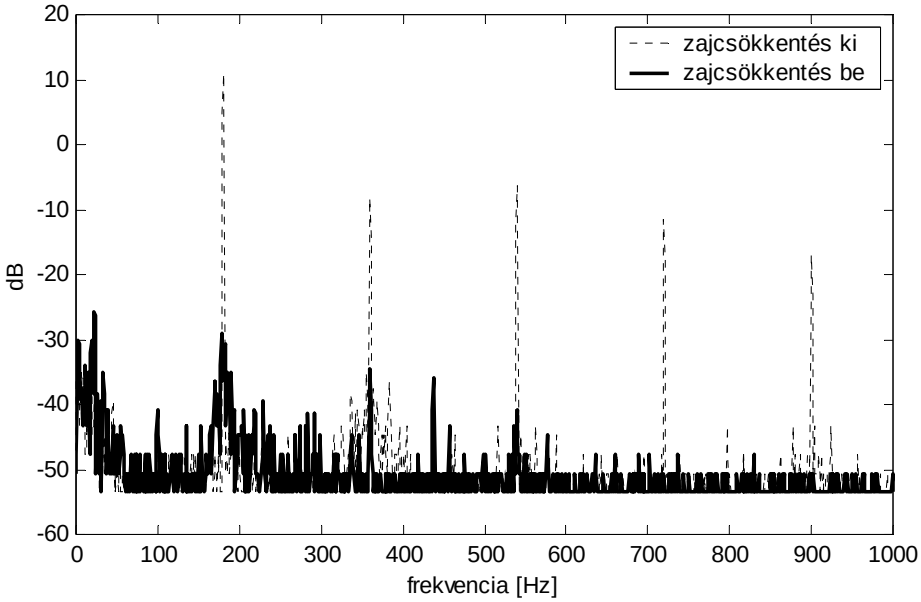


12.3. ábra. On-line identifikációs rendszer beállása

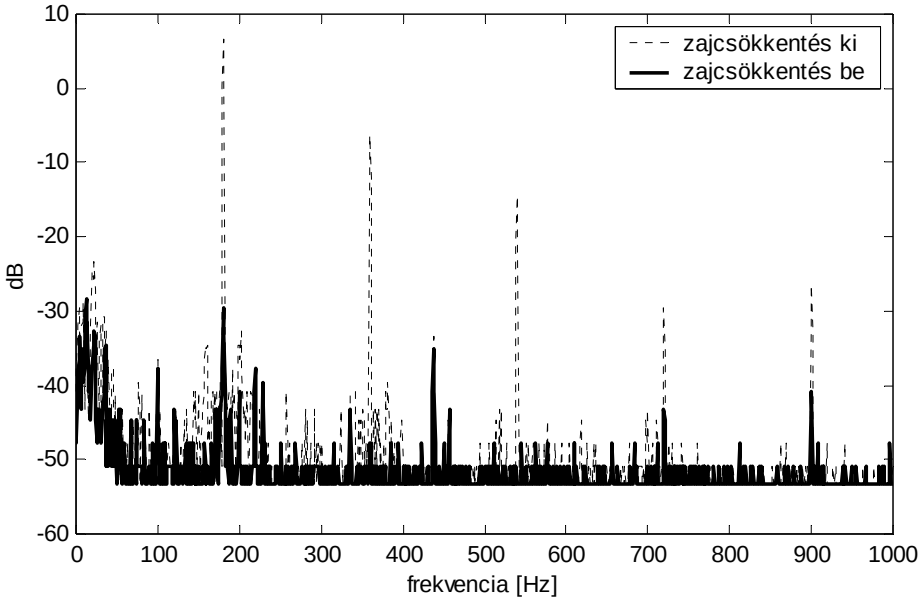
A 12.3. ábrán az on-line identifikációs algoritmussal kiegészített egycsatornás rendszer teszteredménye látható. A mérés során itt is a 12.1. ábrán látható elrendezést használtuk. Az állandósult állapotban lévő rendszert a mote0 elmozdításával tettük instabillá. Látható, hogy több instabil szakasz után, amikor a hibajel exponenciálisan nőni kezd, végül a rendszer megtalálja a helyes átviteli függvényt, és beáll a stabil állandósult állapotba.

12.3. Frekvenciatartománybeli eredmények

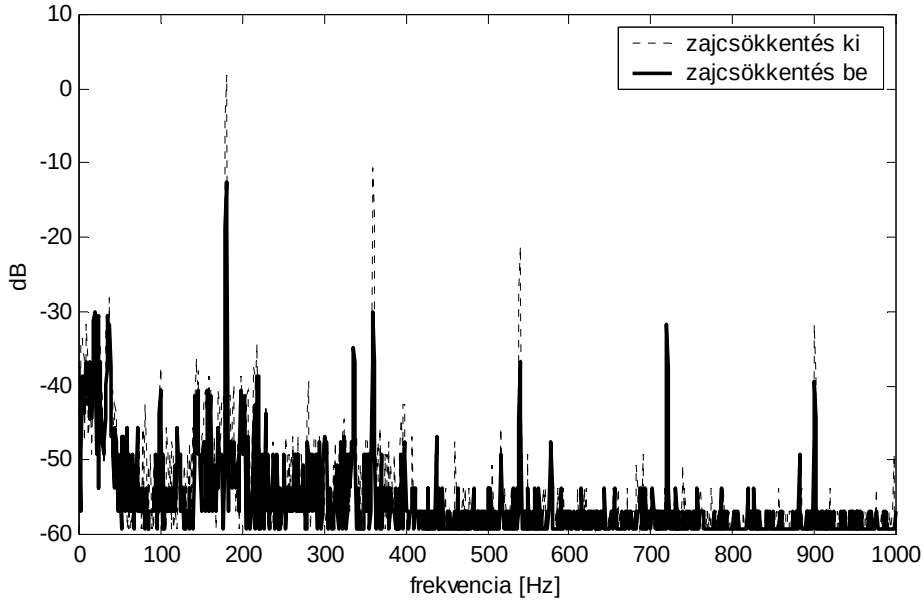
A következő ábrákon a frekvenciatartománybeli zajcsökkenést ábrázoltuk dB-ben. A kétcsatornás rendszer esetén 40 dB-es elnyomást értünk el az alapharmonikusra, ami jó eredménynek mondható. Mindez a 12.4. ábrán látható. A 12.5. ábrán on-line identifikáció esetén mérhető elnyomást vizsgálhatjuk, ami 36 dB. Az elnyomás azért nem lehet teljes, mert a zajt nem tudjuk közvetlenül a mikrofonoknál mérni, a mikrofontól távolodva viszont csökken az elnyomás mértéke.



12.4. ábra. Kétcsatornás rendszer zajelnyomása állandósult állapotban



12.5. ábra. On-line identifikációs rendszer zajelnyomása állandósult állapotban az AFA referencia jele is rádión keresztül érkezik



12.6. ábra. Kétcsatornás rendszer zajelnyomása,
ha a zajt nem közvetlenül a hibamikrofonnál mérjük

Ha a hibajelet érzékelő mikrofonoktól kb. 30 cm-re mérjük a zajszintet, az elnyomás jól láthatóan csökken. Ez az eset a 12.6. ábrán látható. Az elnyomás itt nyilvánvalóan kisebb, hiszen a zajcsökkentő rendszerrel csupán lokálisan, a zajérzékelő mikrofonoknál lehet teljes elnyomást elérni.

13. Összefoglalás, kitekintés

Munkák során sikerült egy stabilan működő kétcsatornás zajcsökkentő rendszert felépíteni. Habár az adatgyűjtő hálózat elvileg több csatornán is képes lenne üzemelni, azonban a rendelkezésre álló DSP-s fejlesztői kártya csupán kétcsatornás DA-átalakítót tartalmaz. Teljes zajkioltást általános esetben pedig csak akkor lehet elérni minden mikrofon környezetében, ha a hangszórók száma megegyezik a mikrofonok számával. A mérési eredmények alapján látható, hogy a rendszer paraméterei megfeleltek előzetes elvárásainknak, és gyakorlatban kipróbálva hallhatóan csökkentette az általunk generált zajokat, méghozzá megfelelő beállási sebesség mellett. Ehhez természetesen hozzájárult a megfelelő identifikáció is, melyet szintén saját fejlesztésű programmal végeztünk.

Abban az esetben, amikor nem ismerjük az átviteli függvényt, jó megoldást kínál az on-line identifikációs módszer. Ezt sikerült egycsatornás rendszerre megoldani, mely a teszteredmények alapján megfelelően működik, jó elnyomást biztosít.

További fejlesztések célja lehet az on-line identifikációs módszer kiterjesztése többcsatornás rendszerekre is, mely a csatornák közötti csatolás miatt komplexebb feladat, mint egycsatornás esetben. Amennyiben ez sikerül, érdekes lehet egy direkt zajcsökkentésre használható hardver kifejlesztése, eddigi munkánkat ugyanis egy általános célú fejlesztői kártyán végeztük.

A szenzorhálózat esetén is sikerült teljesíteni az előírt követelményeket. Megvalósítottuk a mote-ok közti szinkronizácót, mely lehetővé tette a szinkron mintavételezést. A DSP-vel való kapcsolat lehetővé tette, hogy a hálózatban viszonylag nagy sebességgel gyűjtött adatokat nagy számítási igényű algoritmusokban használjuk fel. Az adatgyűjtő hálózat a gyors működési sebesség mellett megfelelően biztonságos adatátvitelt biztosít, méghozzá on-line működést biztosítva. Ez azt jelenti, hogy a hálózati eszközök által gyűjtött adatokat egy mindig állandó értékű késleltetéssel tudjuk továbbítani a DSP felé.

Az elkészült rendszerben elértük, hogy a hálózatban két mote továbbítsa a hibajelet 2 kHz-es mintavételi frekvencián, amellyel 1 kHz-es sávszélességben tudunk zajcsökkentést elérni, mely sok alkalmazás számára teljes mértékben kielégítő lehet.

A jövőbeni továbbfejlesztéseknek több iránya is lehetséges. Egyik irányvonal lehet a mintavételi frekvencia megnövelésének vizsgálata. Erre egy alternatíva lehet például valamilyen tömörítési eljárás alkalmazása, mellyel garantáltan csökkenthető a hibajelből előállított adatok mennyisége. Egy másik lehetőség a fejlesztésre például olyan, a mote-okhoz csatlakoztatható kártya tervezése, mely segítségével vezérelhetjük a beavatkozó hangszórókat, ezáltal teljesen vezeték nélküli zajelnyomó rendszert lehetne megvalósítani. Ennek segítségével olyan szabadon mozgatható zajelnyomó rendszereket alakíthatunk ki, melyek csupán az érzékelőket és beavatkozókat tartalmazza, a zajelnyomó algoritmust pedig egy központi, nagy számítási teljesítményű processzor végzi,

mely akár több mozgó egységet is kiszolgálhat. Ezáltal csökkenhet a zajcsökkentő rendszer költsége, hiszen nem kell minden egységbe beépíteni a jelfeldolgozó algoritmust futtató DSP-t, hanem csupán csak egy, az összes egységet kiszolgáló modult kell készíteni. Ilyen alkalmazás lehet például aktív zajcsökkentést alkalmazó fülvédő, vagy a székek háttámlájára, a fülek magasságában felszerelhető zajcsökkentő rendszer tervezése, melyből egy zajos helységen belül többet is használhatnak (például járművekben), így egy központi egységgel is vezérelhetőek.

A hálózattal kapcsolatban is lehet fejlesztéseket végezni. Egyik ilyen lehetőség a hálózat struktúrájának automatikus felderítése. A jelenlegi alkalmazásban ugyanis a hálózat elemeit annak megfelelően kell programozni, hogy hány mote található a hálózatban, így újraprogramozás nélkül nem lehet a hálózatot módosítani. Bár ennek lehetővé tételére már eddig is tettünk bizonyos lépéseket, de egyelőre nem volt még szükségünk ezen funkció kiépítésére.

14. Felhasznált irodalom

[1] „*ADSP-2106x SHARC User's Manual (second edition)*”, Analog devices Inc. P.O Box 9106 Norwood, 1997.

[2] „*ADSP-2106x SHARC Reference Manual*”, Analog devices Inc. P.O Box 9106 Norwood, 1997.

[3] http://www.tinyos.net

[4] http://www.tinyos.net/nest/doc/tutorial/

[5] http://www.xbow.com/Support/Support_pdf_files/XBOW_Smart_Dust_ProductInfoGuide.pdf

[6] http://www.analog.com/UploadedFiles/Application_Notes/399447663EE191.pdf

[7] http://pdfserv.maxim-ic.com/en/ds/MAX220-MAX249.pdf

[8] http://www.semiconductors.philips.com/acrobat_download/datasheets/74HC_HCT74_3.pdf

[9] Lajkó, Grábler, „*Heterodin spektrumanalizátor megvalósítása DSP-n*”, TDK dolgozat, Budapest, 2004.

[10] Elek Kálmán, „*Adaptív jelfeldolgozás*”, Budapesti Műszaki Egyetem, jegyzet, 2005.

[11] B. Widrow, S.D.Stearns, „Adaptive Signal Processing”, Prentice-Hall, Inc 1985.

[12] S.j.Elliot, P.A. Nelson, „Active noise control”, *IEEE Signal Processing Magazine*, vol. 10, No. 4.October, 1993, pp. 12-35.

[13] BME MIT Tanszéki Munkaközösség: Digitális jelfeldolgozás.Budapesti Műszaki Egyetem, segédlet, 2004.

[14] F. Nagy, „Application of the nonlinear filter and observer theory in adaptive signal processing”, presented on the *IEEE Winter Workshop on Nonlinear Digital Signal Processing*, January, 17-20, 1993 Tampere, Finland, in workshop proceedings pages 6.2-3.1 to 6.2-3.6

[15] Widrow, Walach, „*Adaptive Inverse Control*”, Prentice Hall PTR, Upper Saddle River, New Jersey 07458, 1996.