Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

TDK DOLGOZAT 2012

Oszcillátorok zajának csökkentése FPGA alkalmazásával

Készítette:
Mészáros Gergely Zoltán
(VP5OIH)
Villamosmérnök V. évfolyam
gerimeszi@gmail.com

Konzulensek:
Dr. Ladvánszky János
Ericsson R&D
janos.ladvanszky@ericsson.com

Fehér Gábor
Szélessávú Hírközlés és villamosságtan tanszék
gabor.feher@mht.bme.hu

Tartalomjegyzék

1 Bevezetés:

Elektromos rendszerekben a zajcsökkentésre számos algoritmus létezik [1, 2, 3]. Ezek az algoritmusok átlagolást, és szűrést alkalmaznak. Konzulensem tavaly megjelent cikkében [4] bemutatta, hogy kapcsolat van a zajszoknya, és az amplitúdó- és/vagy fáziszaj korrelációja között. erre az eredményre alapozva egy új zajcsökkentő algoritmust mutattunk be [5].

A zaj nem detektálható/mérhető keletkezésének helyén. Feltételezzük, hogy létrejöttekor a zaj korrelálatlan. A korrelációt egy a zaj keletkezésének helye, és a zaj mérésének helye között „elhelyezkedő" időinvariáns szűrő átviteli függvényével modellezzük. Annak érdekében, hogy a mért zajt korrelálatlanná tegyük, meg kell határoznunk az említett szűrő átviteli függvényének reciprokát, majd alkalmazni kell azt a zajos jelen. Ennek érdekében kiszámítottuk az alapsávi zaj korrelációját, és minimalizáltuk a négyzetösszegét egy adaptív szűrő segítségével. Ezt követően az adaptív szűrő súlytényezőit alapsávról a jel frekvenciájára transzformáltuk, majd alkalmaztuk azt a zajos jelen. Eddig az algoritmust nem tudtuk valós időben alkalmazni, de kísérletileg bemutattuk a zajcsökkentés hatékonyságát egy 1,8 GHz-es tranzisztoros oszcillátor mért jelén Simulink modell alkalmazásával.

A sikeres eredményeket követően a cikkben bemutatott zajcsökkentő módszer FPGA segítségével történő valósidejű megvalósítása a következő cél. Mivel a jelfeldolgozáshoz az FPGA belső órajelét szeretnénk használni (ami 100 MHz), ezért olyan frekvenciájú oszcillátort kell alkalmazni a zajcsökkentéshez, amelyre teljesül a mintavételi tétel. Ennek érdekében a valós idejű zajcsökkentéshez jelforrásként nem a fent említett oszcillátor kerül felhasználásra, hanem egy 10 MHz-es alacsonyfrekvenciás kvarcoszcillátor jele, melyet korábban ipari konzulensem készített [4].

Első lépésként természetesen nem a teljes kétfokozatú zajcsökkentő realizálása a cél, hanem csak a zajparamétereket meghatározó blokk, valamint a zajparamétereket felhasználó zajcsökkentő blokk realizálása. Az FPGA-ra töltendő bitstream előállításához az ISE Design Suit System Generator for DSP komponensét használva az eddig használt Matlab/Simulink környezet használható.

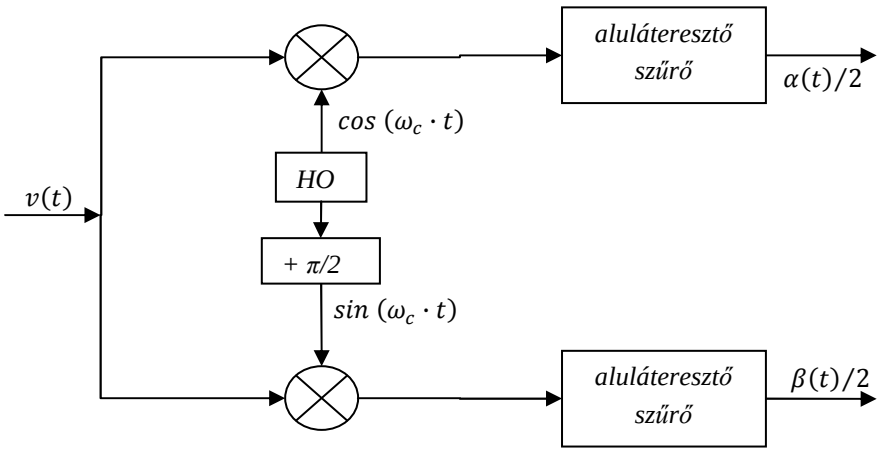
A Tudományos Diákköri konferencia dolgozat keretein belül röviden bemutatásra kerül az MSc-s diplomatervezés 1 tantárgy beszámolójában bemutatott oszcillátorok zajanalízisére használt off-line Matlab/Simulink modell, továbbá az általunk FPGA-n realizálni kívánt kiegészített modell is.

A dolgozatban kitérünk a xilinx komponenseket felhasználó rendszer elkészítésére, a szükséges módosításokra a Simulinkes modellhez képest, valamint a rendszer főbb részeinek kimeneteinek a Simulink modell megfelelő kimeneteivel történő összehasonlítására, melyekből kiderül, hogy a szükséges módosítások ellenére is eredményes a zajcsökkentés. A bitstream generálásával kiderül, hogy a rendszerhez nem szükséges több erőforrás, mint a használni kívánt Virtex-4 c4vfx20 –as FPGA-an rendelkezésre áll, azaz kipróbálható a rendelkezésre álló FPGA a zajcsökkentéshez (a bitstream generálásakor keletkezett figyelmeztetések kijavítását követően).

Jelenleg az említett figyelmeztetések kijavítása, valamint a rendszerhez szükséges órajel előállítása a cél. A későbbiekben szeretnénk az online zajcsökkentő rendszert más alkalmas frekvenciájú, rendelkezésre álló rendszerek vivőin is kipróbálni.

2 Kiindulási modell

Az MSc-s diplomatervezés 1 tantárgy keretein belül bemutatott oszcillátorok zajparamétereinek meghatározásával, majd további műveletek segítségével csökkenthető (ideális esetben megszüntethető) az oszcillátor amplitúdó- és fáziszaja. A zajparaméterek meghatározásához (a zajt moduláló jelnek tekintve) kvadratúra demodulátort használunk. A kvadratúra demodulátor látható a 2.1. ábrán. A bemeneti zajos jel: $v(t) = A \cdot \big(1 + \xi(t)\big) \cdot cos\big(\omega_c \cdot t + \varphi(t)\big)$, ahol A az eredeti (zajmentes) jel amplitúdója, ω_c a zajmentes jel körfrekvenciája, $\xi(t)$ a relatív amplitúdózaj és $\varphi(t)$ a fáziszaj. Az ábrán szereplő helyi oszcillátor (HO) előállítja az ω_c körfrekvenciájú szinuszos és koszinuszos jeleket a keveréshez (szorzáshoz), míg az aluláteresztő szűrők feladata, hogy eltávolítsák a szorzatból a $2 \cdot \omega_c$ körfrekvenciájú összetevőket. Így megkaphatjuk az $\alpha(t) = A \cdot \big(1 + \xi(t)\big) \cdot cos\big(\varphi(t)\big)$ és $\beta(t) = -A \cdot \big(1 + \xi(t)\big) \cdot sin\big(\varphi(t)\big)$ jeleket, melyek segítségével már könnyen meghatározhatók a zajparaméterek a (2.1)-es és a (2.2)-es egyenletek segítségével.

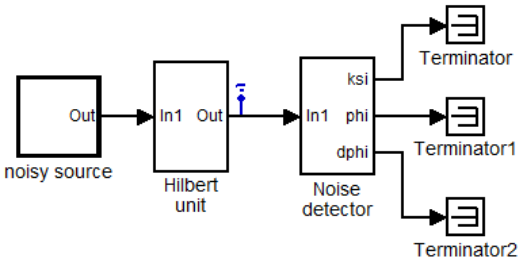


2.1. ábra: Kvadratúra demodulátor használata a zajparaméterek meghatározásához

$$\xi(t) = \frac{\sqrt{\alpha(t)^2 + \beta(t)^2}}{A}, \tag{2.1}$$

$$\varphi(t) = tan^{-1}\left(-\frac{\beta(t)}{\alpha(t)}\right). \tag{2.2}$$

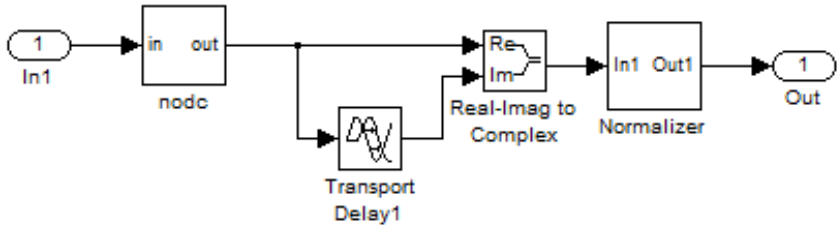
Ez alapján a bemutatásra kerülő zajcsökkentő eljárás kiinduló modellje a 2.2. ábrán látható Matlab/Simulink modell. A zajos valós jelforrásból a Hilbert egység blokk eltávolítja az egyen komponenst, komplex jellé alakítja egy 90°-os eltolással, majd a kapott komplex jelet normalizálja. A zajparaméterek meghatározását a Noise detector blokk végzi a fent bemutatott kvadratúra demodulátor elve alapján.



2.2. ábra: A használt kiindulási modell.

A Noise detector blokk az előbb előállított komplex jelből a kvadratúra demodulátorhoz hasonló elven előállítja az amplitúdó zajt ($\xi(t)$) és a fáziszajt ($\varphi(t)$) a (2.1) és (2.2) egyenleteknek megfelelően, ahol $\alpha(t)$ és $\beta(t)$ a Hilbert egység normalizált koszinuszos és szinuszos kimenete. Ezeken kívül egy harmadik kimenetet is találunk a blokkon, mellyel a blokkban található fáziszárt hurok fázisváltozását követhetjük.

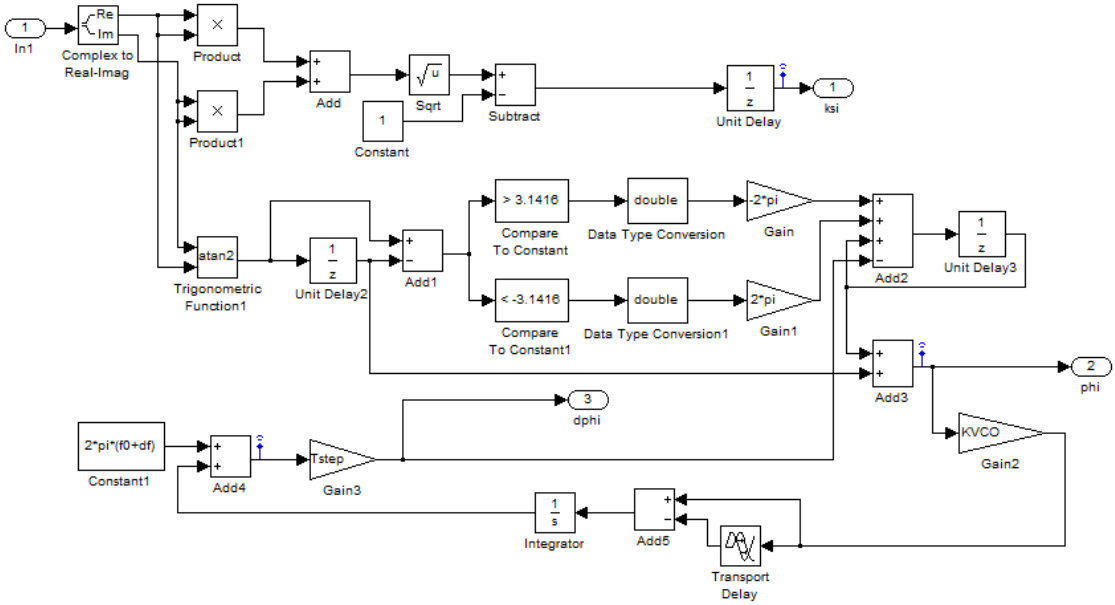
A Hilbert egység blokkvázlata a 2.3. ábrán látható. Az nodc blokk távolítja el a zajos jel esetleges egyenáramú összetevőit paraméteresen meghatározható egész számú periódus átlagolásával (mozgóátlag), és az eredeti jelből történő kivonásával. A Transport Delay1 blokk végzi az f_0 frekvencián (zajos jelforrás frekvenciája) történő 90°-os fázistolást, mellyel a későbbi komplex jel képzetes részét állítjuk elő. Ha változtatható frekvenciájú oszcillátor zaját kívánjuk csökkenteni, akkor a késleltető helyett Hilbert szűrőt alkalmaznánk. Erről kapta ez az egység a nevét.



2.3. ábra: Hilbert egység blokkvázlata

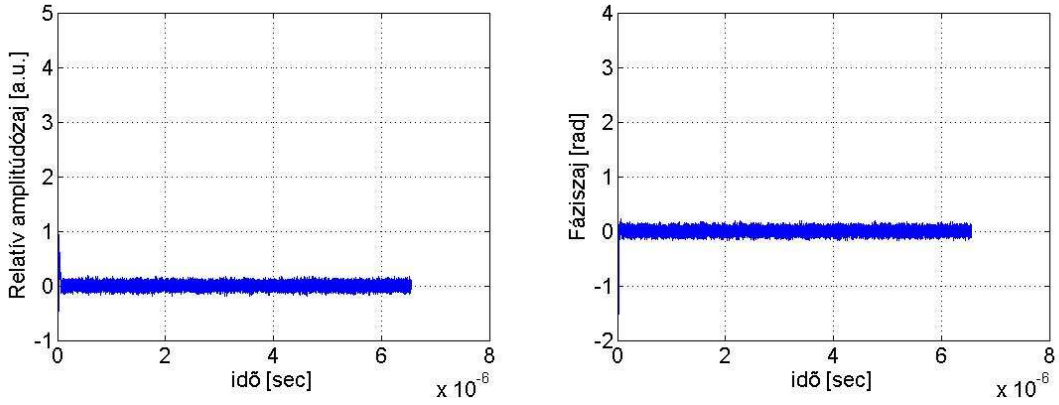
A Hilbert egység utolsó blokkja a normalizálást végző Normalizer blokk. A blokk a már komplex jel abszolút értékének egészszámú periódusra meghatározza az átlagát (mozgó átlag), mellyel az eredeti komplex jelet leosztja. Az átlagoláshoz használt periódusok száma az nodc blokkhoz hasonlóan paraméteresen meghatározható.

A zajparaméterek meghatározását végző Noise detector blokk látható részletesen a 2.4. ábrán. Az ábra felső részén a (2.1) egyenletnek megfelelően az amplitúdó zaj kiszámítását megvalósító rész látható, míg az alsó részen a (2.2) egyenletnek megfelelő fáziszaj meghatározása kiegészítve a már említett fáziszárt hurokkal.



2.4.ábra: Noise detector

Bemeneti zajos jelként a bevezetőben már említett 10 MHz-es [4] oszcillátor jelét használjuk, melyet oszcilloszkóppal rögzítettünk 40 GSa/sec mintavételi sebességgel. 10-10 periódusra végzett átlagolásokkal a 2.2. ábrán látható modellel kapott időfüggő zajparaméterek a 2.5. ábrán látható.



2.5. ábra: 10MHz-es oszcillátor amplitúdó- és fáziszaja 10 periódusra történő átlagolásokkal.

A fáziszaj grafikonján jól látható, hogy a 10 periódus (10^{-6} s) után megfelelően már jól zár a használt fáziszárt hurok.

3 Zajcsökkentés megvalósítása

A meghatározott zajparaméterek segítségével a Hilbert egységgel előállított normalizált, egyenárammentes, komplex jel zajának csökkentése/eltávolítása már könnyen elvégezhető. A Hilbert egység kimenő jele:

$$\big(1+\xi(t)\big)\cdot\big[cos\big(\omega_c\cdot t+\varphi(t)\big)+j\cdot sin\big(\omega_c\cdot t+\varphi(t)\big)\big], \tag{3.1}$$

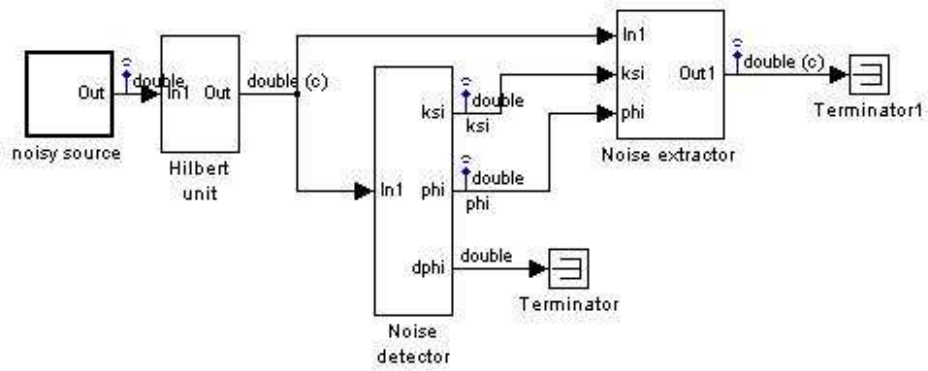
melyet átírhatunk a következő formára:

$$\big(1+\xi(t)\big)\cdot e^{j\cdot(\omega_c\cdot t+\varphi(t))}. \tag{3.2}$$

Ebből a formából jól látható, hogy az amplitúdó zaj eltávolításához a Hilbert egység kimenő jelét $\big(1+\xi(t)\big)$-vel kell osztani, míg a fáziszaj eltávolításához $e^{-j\cdot\varphi(t)}$-vel kell szorozni. Az így kapott „megtisztított” jel:

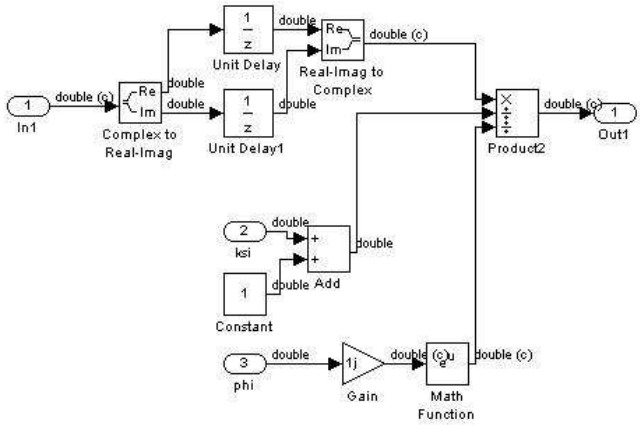
$$1\cdot e^{j\cdot(\omega_c\cdot t)}=1\cdot[cos(\omega_c\cdot t)+j\cdot sin(\omega_c\cdot t)]. \tag{3.3}$$

A zajcsökkentéssel kiegészített Matlab/Simulink modell a 3.1. ábrán látható.



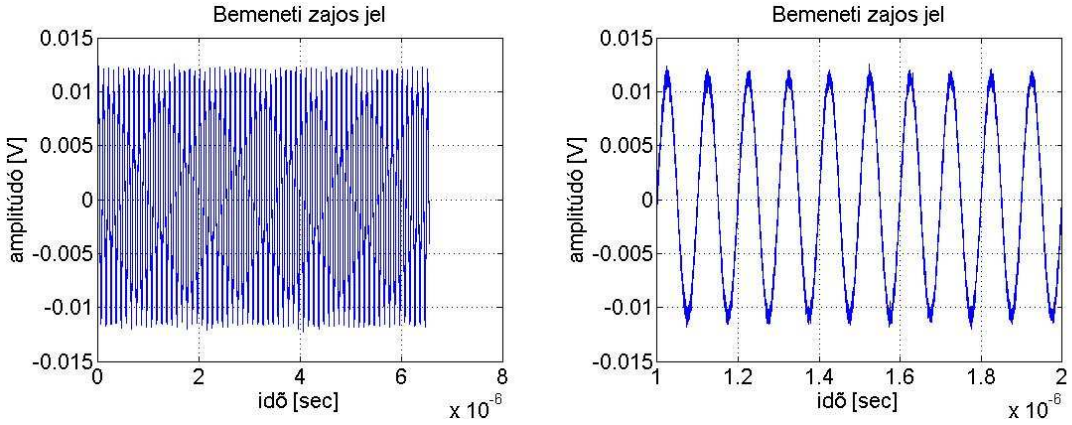
3.1. ábra: A zajcsökkentéshez használt modell.

A zajcsökkentéshez szükséges műveleteket a Noise extractor blokk végzi, mely részletesen a 3.2. ábrán látható. A blokkban szereplő egységnyi késleltetők kompenzálják a zajparaméterek meghatározását végző Noise detector blokk késleltetését a Hilbert egység kimeneti komplex jelén, hogy az adott komplex értéket az időben hozzá tartozó amplitúdó- és fáziszaj értékekkel legyen „megtisztítva”.

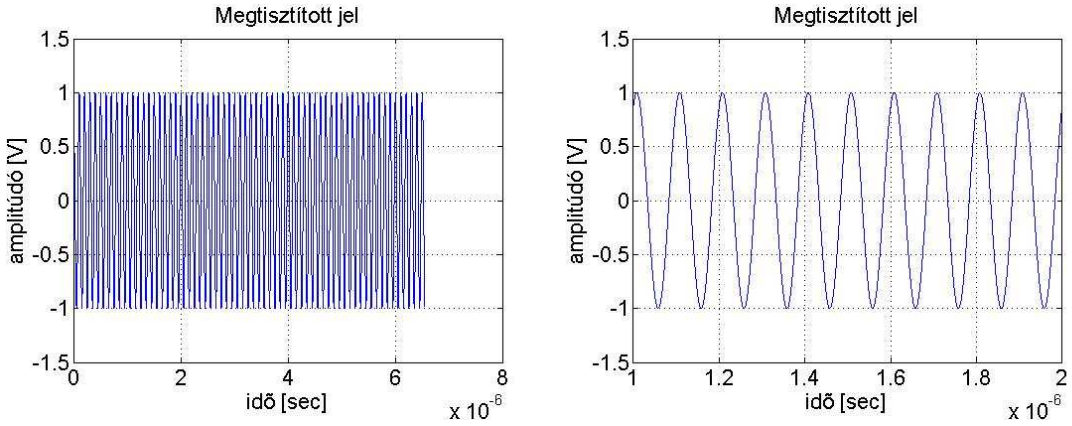


3.2. ábra: Noise extractor blokk.

A fenti modellhez használt bemeneti zajos jel látható a 3.3. ábrán, míg a zajtól megtisztított jel valós része a 3.4. ábrán látható.

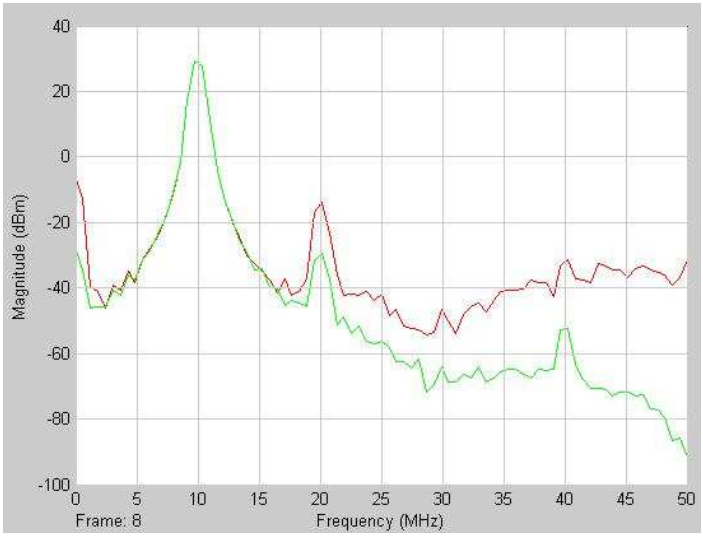


3.3. ábra: A modellhez használt bemeneti 10 MHz-es zajos jel.



3.4. ábra: A megtisztított jel.

A fenti ábrákon is jól látszik a zajcsökkentés hatása, különösen a jobb oldali ábrákon kiemelt 10 perióduson. A bemeneti zajos jel, és a kimeneti jel spektrumát megvizsgálva még szembetűnőbb a zajcsökkentés hatása. A két jel spektruma a 3.5. ábrán látható. A piros görbe mutatja a zajos bemeneti jelnek a spektrumát, míg a zöld görbe a zajcsökkentett kimeneti jelnek a spektrumát mutatja.



3.5. ábra: A zajos jel (piros görbe), és a „megtisztított” jel (zöld görbe) spektruma.

A 3.3. és a 3.4. ábrákat összehasonlítva láthatjuk, hogy a két jel amplitúdója között közel százszoros a különbség. Ez az amplitúdóbeli eltérés közel 40 dB-es eltérést eredményez a két jel spektruma között. E különbség elkerülése végett a 3.5. ábra elkészítésekor a bemeneti zajos jelet külön normalizáltuk, hogy annak is egységnyi legyen az amplitúdója. Az alapharmonikusok fedésén látható, hogy az alkalmazott kompenzálás megfelelő.

Az utolsó ábrák alapján meggyőződhettünk a zajcsökkentés működőképességéről. A következő lépés az FPGA-ra töltendő bitstream előállítása. Ehhez a feladathoz az ISE Design Suit System Generator for DSP komponensét használva lehetővé válik, hogy a már megszokott Matlab/Simulink környezetet használva elkészíthető legyen a szükséges bitstream.

4 System Generator for DSP alkalmazása

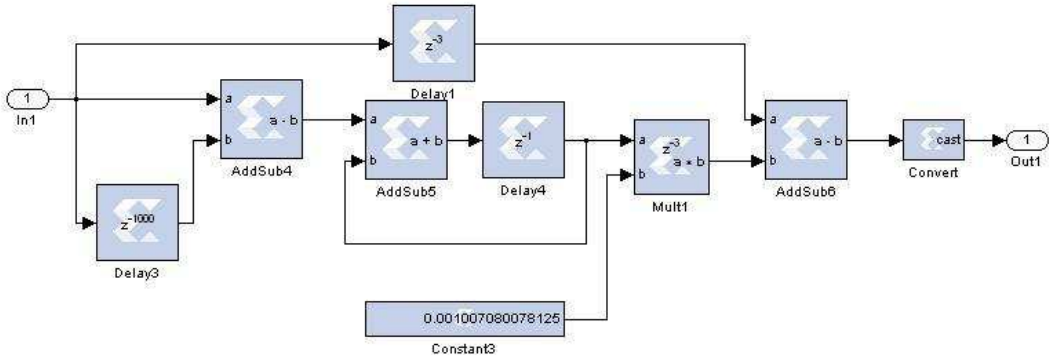
A System Generator for DSP ”toolbox-ként” jelenik meg a Simulink-ben, melynek a legalapvetőbb eleme a System Generator blokk. Ez a blokk kezeli a rendszer illetve a szimulációs paramétereket, illetve az elkészített modell alapján létrehozza a bitstreamet. A blokk paraméterei között meg kell adni a használni kívánt FPGA típusát, ami esetünkben a laborban rendelkezésre álló Virtex4 xc4vfx20-as típus, az FPGA órajelét, illetve az órajelhez tartozó pin helyzetét.

Ahhoz hogy elkészülhessen a kívánt bitstream előbb el kell készíteni a teljes modellt a Simulink xilinx könyvtárbeli komponensek segítségével. Ezt a feladatot a 3.1. ábrán látható blokkon külön-külön kerül megvalósításra, úgy, hogy blokkonként a kimeneti jelalakok összehasonlításra kerülnek a Simulink-es modell kimeneti jelalakjaival.

4.1 Hilbert egység blokk megvalósítása

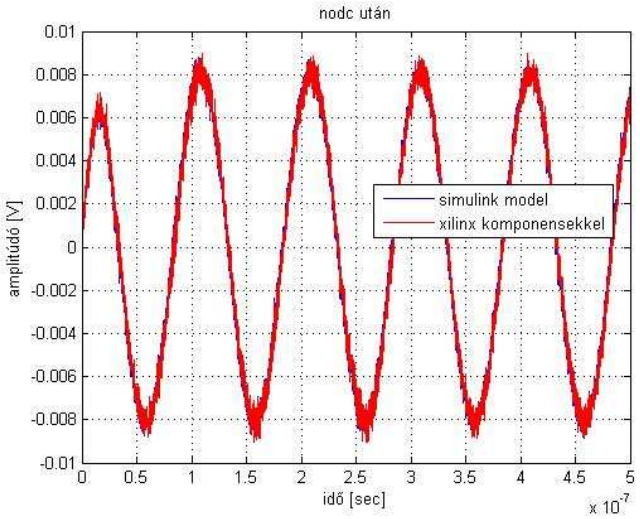
A Hilbert egység blokkjai közül az egyenáramú összetevőt eltávolító nodc blokk elkészítése a legegyszerűbb a xilinx blokkokkal, mert ahhoz csak az alapelemek: összegző (és különbségképző), késleltető (tároló) és szorzó blokkok, illetve konstanst tartalmazó blokkok szükségesek. Az egyes blokkok paraméterei olyan változók,

melyeket a szimuláció előtt egy M-fájl segítségével tudunk definiálni. Az említett blokk a 4.1. ábrán látható xilinx komponensekkel összeállítva.



4.1. ábra: Hilbert egység nodc blokkja xilinx komponensekkel összeállítva.

A blokk kimeneti jelének időfüggvényének, valamint a Simulink modell ugyanezen blokkjának kimeneti időfüggvényét összehasonlítva meggyőződhetünk a blokk helyes működéséről. A kimeneti időfüggvények a 4.2. ábrán láthatók. Az ábrán a kék görbe tartozik a Simulink modellhez, míg a piros görbe a xilinx komponensekből összeállított blokkhoz. Az ábrán láthatjuk, hogy a két görbe szinte teljesen fedi egymást, azaz megfelelően működik az új komponensekből összeállított modell.



4.2. ábra: Hilbert egység nodc blokkjának kimenete.

A korábbi időfüggvényekkel összehasonlítva láthatjuk, hogy ez az ábra nem a teljes bemeneti zajos adathoz tartozó kimeneti időfüggvényt mutatja, hanem annak csak az első néhány periódusát. Ennek oka, hogy a xilinx komponenseket használva ugyanazon bemeneti adatsorozat mellett a szimuláció jelentősen lelassul. Mivel ennyi periódusból is jól látszik a helyes működés, ezért a szimuláció további futtatása ezen blokk esetében felesleges.

A Hilbert egység másik blokkjának (Normalizer blokknak) xilinx komponensekkel történő megvalósítása már nehezebb feladat, mivel ez a blokk az nodc blokkal ellentétben nem csak az alap komponenseket (összeadó, szorzás, stb.) tartalmazza, hanem más műveleteket is. az egyik ilyen művelet az osztás, melyből kettő is kell (a valós részhez, és a képzetes részhez). A xilinx komponensek között található legtöbb osztást megvalósító komponens csak lebegőpontos számábrázolással működik, illetve csak egész számok osztását végzi el. Szerencsére egy másik xilinx komponenseket tartalmazó könyvtárban megtalálható néhány további előre elkészített adott műveletet elvégző blokk, melyek a CORDIC algoritmust alkalmazzák (COordinate Rotation Digital Computer). Ezen algoritmusokkal többek között szögfüggvény-, logaritmus-, négyzetgyökszámítást,

valamint osztást (CORDIC DIVIDER) is megvalósítanak. Ezen blokkok fel paraméterezéséhez elegendő megadni a bemeneti adatok számábrázolását (hány bitet használunk, valamint melyik bittől kezdődik a tört rész), valamint az algoritmus lineáris forgatásához hány darab iteratív fokozatot szeretnénk használni (ez utóbbi paramétert az alapértelmezett értéken hagytam). Az osztást megvalósító komponensnek a működése különböző konstans számokkal illetve különböző függvényekkel is le lett tesztelve. A komponens megfelelően működik, viszont a tapasztaltak alapján a jelentős késleltetése mellett (15 biten ábrázolt számok, és 11 darab iteratív fokozat esetén már 30 időegységgel késleltet) jelentősen lassítja a szimulációt egy komplexebb blokk esetében (már az alapműveletekkel is nagyon lelassul a szimuláció).

A másik gondot okozó művelet a Normalizer blokkban az abszolút érték képzés. Ez a művelet egy valós szám esetében nem jelentene problémát, de egy komplex szám esetében már nem annyira egyszerű, mint a Simulink modell esetében. Komplex számok esetében az abszolút érték képzést négyzetre emelésekkel, összeadással és gyökvonással helyettesíthetjük:

$$z = x + j \cdot y, \tag{4.1}$$

$$|z| = \sqrt{x^2 + y^2} = \sqrt{x \cdot x + y \cdot y}. \tag{4.2}$$

Ezen átalakításokkal már csak a gyökvonást tartalmazza a már említett alapműveletek mellett. Az előzőekben leírtak alapján a gyökvonásra is van megoldás a CORDIC algoritmus segítségével. Ez az algoritmus azonban az osztáshoz képest még komplexebb, amely tovább lassítja a szimulációt, és a jelfeldolgozást. A szögfüggvényeket megvalósító algoritmusok között kutatva rátalálhatunk a CORDIC ATAN komponensre, amely nem a matematikában használt arctan függvényt valósítja meg, hanem a számítástechnikában alkalmazott két-argumentumú arctan függvényt (arctan(y/x)). Ez az algoritmus a kétdimenziós koordinátarendszerben ábrázolható (x,y) pontot $\Delta\theta = \pm\, arctan(1/2^i)$ szöggel (i a paraméterként megadható iteratív fokozatok száma) addig forgatja, míg az y koordináta értéke nulla nem lesz. Eredményül az y=0 eléréséhez szükséges szöget adja, valamint további kimenetként megadja azt az x értéket is, ahol az x tengelyt (y=0 egyenes) metszi. Ez az x érték megfelel a bemeneti x, y számokkal megadott komplex szám abszolút értékének a megfelelő kompenzálás után. A blokk által számított abszolút érték:

$$|z| = K \cdot \sqrt{x^2 + y^2}, \tag{4.3}$$

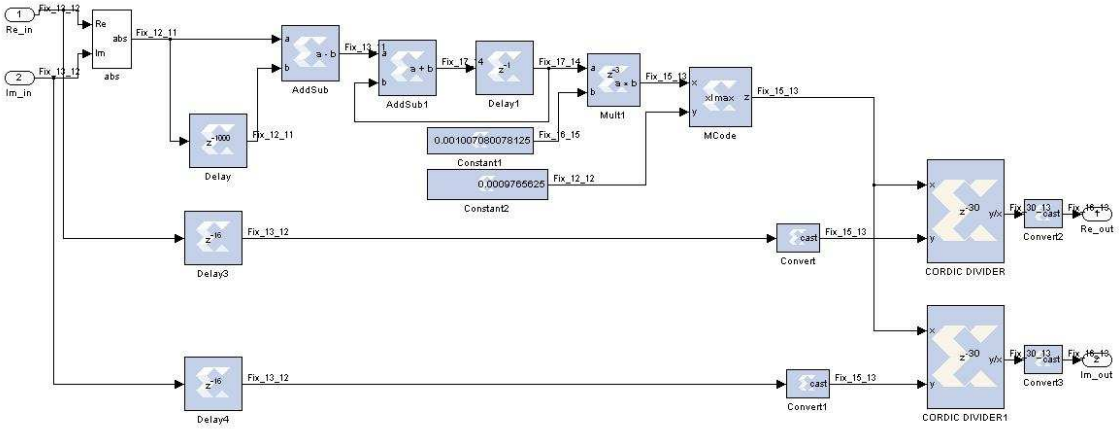
ahol

$$K = 1{,}646760\,. \tag{4.4}$$

A pontos eredményhez K-val osztani kell az ATAN komponens által kiszámított eredményt. Mivel K konstans szám, ezért elég a reciprokával szorozni, így egy további osztást végző komponenssel kevesebbet kell használni.

Ez a komponens a $\Delta\theta$-vel való forgatást összeadásokkal, és kivonásokkal éri el, így a gyökvonáshoz képest sokkal kevesebb késleltetést okoz (azonos paraméterek esetén a gyökvonás okozta késleltetés harmadát), valamint a szimulációt is sokkal kisebb mértékben lassítja. Ennek a komponensnek a zajparamétereket meghatározó blokkban is fontos szerepe lesz.

A Normalizer blokk xilinx komponenseket felhasználó megvalósítása a 4.3. ábrán látható.



4.3. ábra: Hilbert egység Normalizer blokkjának megvalósítása xilinx komponensekkel.

A blokk legelső egysége (abs) tartalmazza a CORDIC ATAN komponensét a megfelelő kompenzációval, valamint tartalmaz még egy maximum függvényt megvalósító blokkot a nullával való osztás elkerülése érdekében. Mivel a xilinx komponensek között nincs maximum függvényt megvalósító blokk, ezért ezt az M-Code blokkal valósítjuk meg, amellyel m-fájlba elmentett függvényeket lehet megvalósítani. A maximum függvényt megvalósító kód a következő:

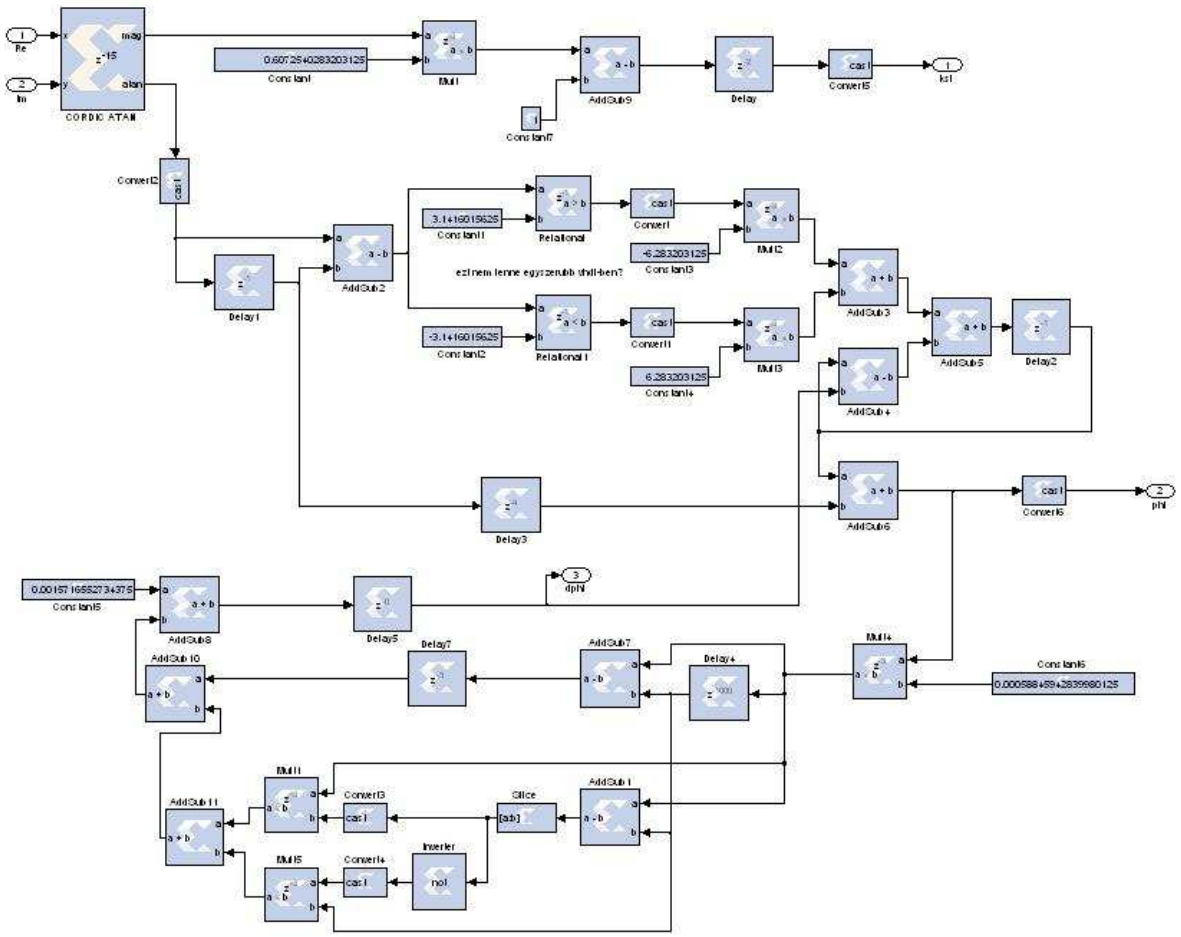
```
function z = xlmax(x, y)
      if x > y
         z = x;
      else
         z = y;
      end
```

Az osztást megvalósító CORDIC komponensek az ábra jobb oldalán láthatók.

A 40 GSa/sec mintavételi sebességgel rögzített zajos jel túl sok adatot tartalmaz így a blokk szimulációja a teljes adatsorral több napot venne igénybe, de már a kiértékelhető adatsor előállításával is memória problémák lépnek fel szimuláció közben. Ezen okok miatt a jelentősebb blokkok működését egyesével vizsgáltam, hasonlítottam össze a megfelelő Simulink blokkal. Az egyes blokkokkal kapott eredményeket nem mutatom be külön-külön, de a szimulációk során minden komponens megfelelően működést mutatott.

4.2 Noise detector blokk megvalósítása

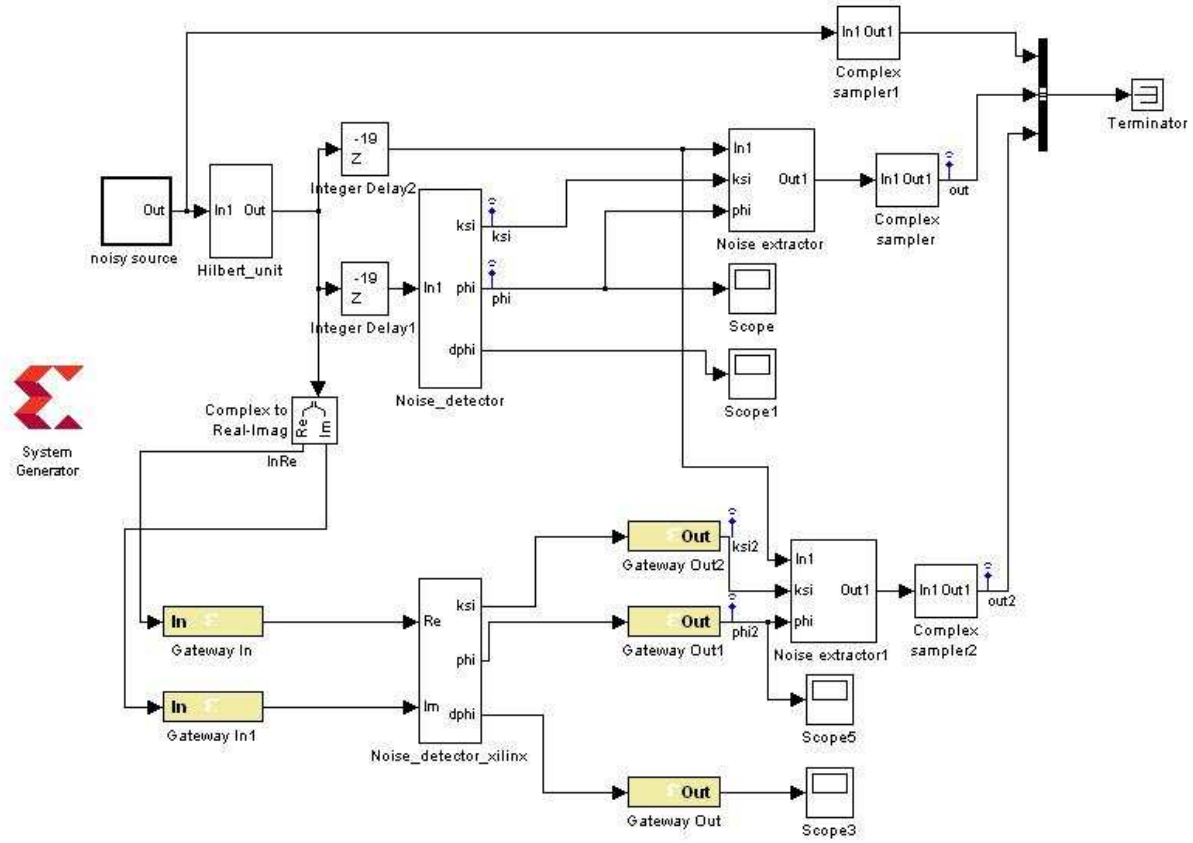
A Noise detector blokk xilinx komponensekkel történő megvalósításánál kulcs fontosságú szerepet tölt be a fent bemutatott CORDIC ATAN blokk. Ez a blokk egyszerre valósítja meg a 2.4. ábra elején található atan2 függvényt, amely megadja a valós és képzetes részekhez tartozó szöget, valamint a valós és képzetes rész által meghatározott vektor hosszát. Ez utóbbi a 2.4. ábrán a valós és képzetes részek négyzetösszegeiből vont gyökkel van megvalósítva. Ha az említett ábrán látható blokkokkal egyező komponensekkel kerül megvalósítása, akkor sokkal jelentősebb jelfeldolgozásbeli késleltetést eredményezne. A Noise detector xilinx komponensekkel történő megvalósítása a 4.4. ábrán látható. Ezen az ábrán már látható a CORDIC ATAN komponens által kiszámított abszolút értékhez szükséges kompenzáció, amely az előző fejezetben említésre került.



4.4. ábra: Noise detector blokk megvalósítása xilinx komponensekkel.

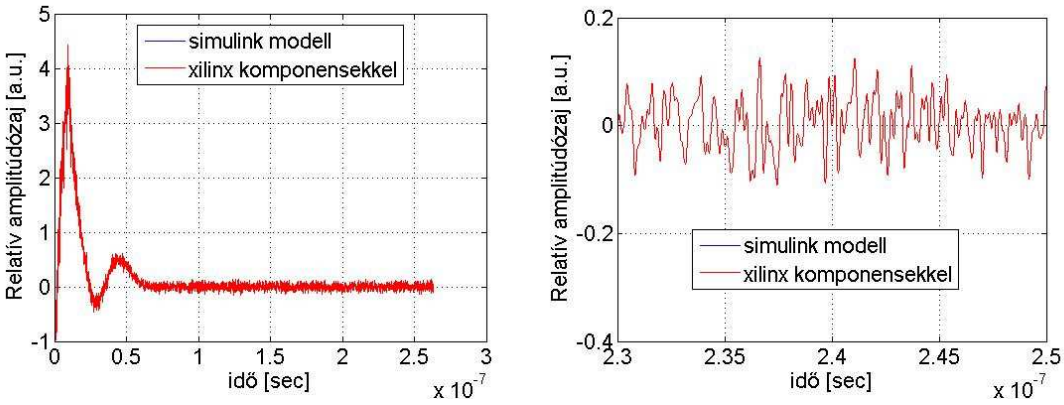
A blokk felső szakasza a CORDIC ATAN komponens kivételével megegyezik a 2.4. ábrán látható résszel, azonban az alsó szakasz már eltérő a két blokkban. Ez az alsó szakasz valósítja meg a fáziszárt hurkot. Az egyik eltérés, hogy a 2.4. ábrán látható Gain3 blokk által megvalósított szorzás a xilinx komponensekkel megvalósított blokkban hiányzik. Ennek az eltérésnek az oka, hogy ezt a szorzást annak érdekében, hogy minél kevesebb szorzó blokkot kelljen használni. Ezzel az összevonásokkal azt is elértük, hogy a fáziszárt hurokhoz szükséges paraméterekkel meghatározható konstansok ne legyenek túl nagy értékek (kevesebb biten ábrázolhatók legyenek), így a használt szorzók is kevésbé lesznek bonyolultak. A fáziszárt hurokban látható másik különbség az integrálást végző blokk elhagyása, mert a xilinx komponensekkel bonyolult ennek a blokknak a megvalósítása. Az integráló blokk területszámítással helyettesíthető, amely a 4.4. ábra legalsó részén látható. Egy másik eltérés a két fáziszárt hurok között (amely az ábrákat összehasonlítva nem látható), hogy a xilinx komponensekkel megvalósított hurokban elvégzett műveletek (pl.: szorzás) késleltetéssel járnak, így a Simulink modellhez hasonlóan nem lehet ideális (késleltetés nélküli) visszacsatolást megvalósítani. Ez utóbbi két blokkok közötti eltérés a zajparaméterek időfüggvényeibe is okozhat némi eltérést.

A blokk helyes működésének ellenőrzéséhez a 4.5. ábrán látható szimulációs elrendezés alkalmazható. Az ábrán látható, hogy a zajparamétereket meghatározó blokkok bemeneti jeleit azonos Simulink blokkokból összeállított Hilbert egység végzi. A Noise detectorok által meghatározott zajkomponensek időfüggvényei rögzítésre kerülnek, hogy a szimuláció végeztével azok összehasonlíthatók legyenek. Az időfüggvények megfelelő összehasonlítása miatt a Simulink blokkokkal megvalósított referencia modellt ki kell egészíteni késleltetésekkel a xilinx komponensek késleltetéseinek megfelelően. Ezt a 4.5. ábrán látható 19 időegységgel késleltető blokkok végzik.

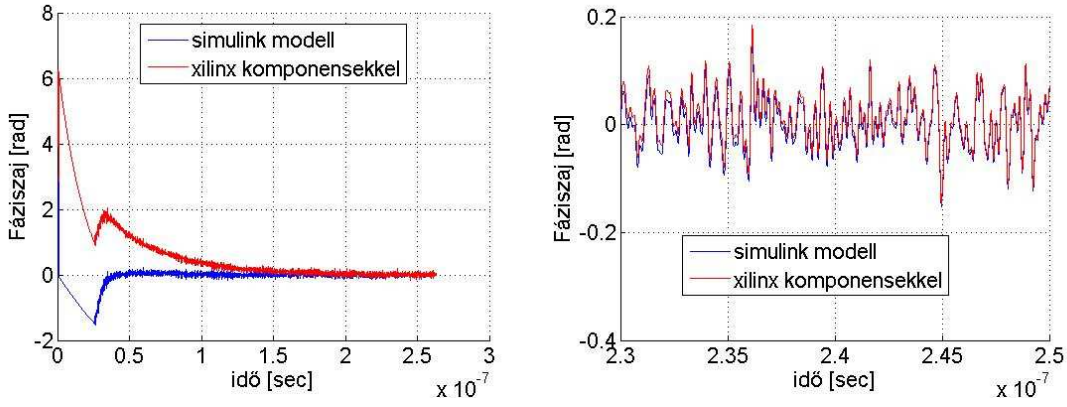


4.5. ábra: Noise detector blokk helyes működésének ellenőrzésére használt szimuláció.

A 4.5. ábrán látható szimulációs elrendezés eredményei a 4.6. ábra grafikonjain, és a 4.7. ábra grafikonjain láthatók. A 4.2. ábrán bemutatott összehasonlításhoz hasonlóan a 4.6. és a 4.7. ábrán látható összehasonlítás se a teljes bemeneti adatsorhoz tartozik, hanem csak a 10 MHz-es jel első két periódusához. A 4.6. ábra a relatív amplitúdózajokat összehasonlító grafikonokat mutatja, melyeken jól látható, hogy a két görbe (a Simulink modellhez, és a xilinx komponensekkel megvalósított blokkhoz tartozó) teljesen fedi egymást. Ez az egyezés igazolja a CORDIC ATAN komponens megfelelő működését is.

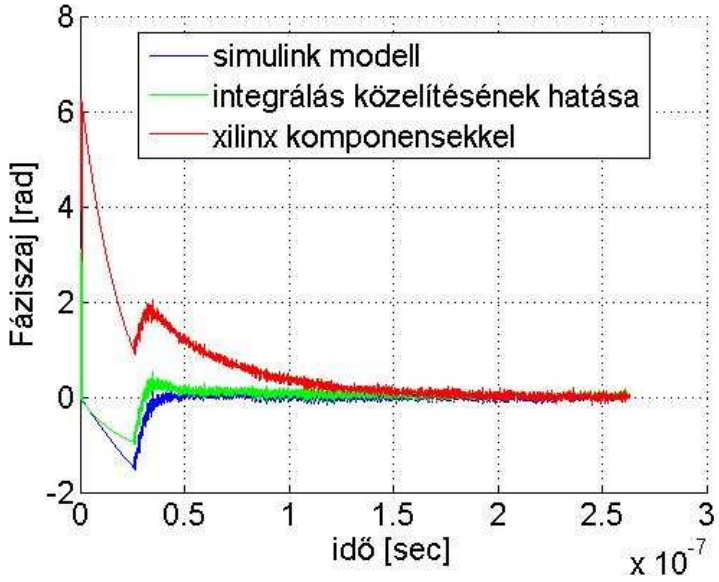


4.6. ábra: Noise detector blokkok relatív amplitúdózajainak összehasonlítása.



4.7. ábra: Noise detector blokkok fáziszajainak összehasonlítása.

A fáziszajokat összehasonlító grafikonok a 4.7. ábrán láthatók. A két görbe az első 100 ns-ban (1 periódusban) eltér egymástól, de ezt követően már megegyeznek egymással (jobb oldali grafikon). A kezdeti eltérésnek az okai az integrálás területszámítással való közelítése, valamint a visszacsatolásban lévő késleltetések. További szimuláció elvégzéséből kiderül, hogy az eltérés főbb oka a visszacsatolásbeli késleltetések. Az integrálás területszámítással történő közelítésének hatása a 4.8. ábrán látható.



4.8. ábra: Az integrálás területszámítással történő közelítésének hatása.

Az ábrán zölddel láthatjuk az integrálás közelítésének hatását. a zöld görbe sokkal közelebb van a kék (az integrálással számítotthoz tartozó) görbéhez, mint a piros (xilinx komponensekkel kapott) görbéhez. Ez bizonyítja, hogy a visszacsatolásban található késleltetések okozzák az eltérés jelentős részét.

A fáziszajt összehasonlító ábrákon azonban láthatjuk, hogy az első 2-3 periódust (200-300 ns-ot) követően már megegyezik a xilinx komponensekkel számított, és a Simulink modellel számított fáziszaj. Mivel az egyezés néhány periódust követően teljesül, ezért nyugodtan használható a xilinx komponensekkel összeállított blokk.

4.3 Noise extractor blokk megvalósítása

A zajcsökkentést végző blokk Simulink modellje a 3.2. ábrán már bemutatásra került. Az ábrán látható, hogy az összeadó alapegységen kívül osztás, és exponenciális műveletet, illetve függvényt kell xilinx komponensekkel megvalósítani. Mivel xilinx komponensekkel nem lehet komplex számokat kezelni (csak valós számokat lehet), és ennek a blokknak a Hilbert egységből érkező bemenete komplex, ezért az egyes műveleteket szét kell választani külön a valós, és külön a képzetes részre vonatkozóan. Ez azt jelenti, hogy a Hilbert egység normalizer blokkjához hasonlóan az osztást külön el kell bégezni a jel valós részén, és a jel képzetes részén is. Ez ismét jelentősen lassítja a szimulációt, valamint újabb nagyszámú erőforrást igényel az FPGA-tól. A helyzetünket tovább rontja, az a tény, hogy a xilinx komponensek között nincs olyan blokk, amely az exponenciális függvényt valósítaná meg.

Ez utóbbi nehézséget könnyedén kezelhetjük, ha az alábbi matematikai átalakítással élünk:

$$e^{-j\cdot\varphi(t)} = cos\big(\varphi(t)\big) - j\cdot sin\big(\varphi(t)\big). \tag{4.5}$$

Egy tetszőleges komplex számot ($x + j\cdot y$) megszorozva a fenti komplex számmal az eredmény:

$$\begin{aligned}(x + j\cdot y)\cdot e^{-j\cdot\varphi(t)} &= (x + j\cdot y)\cdot\big[cos\big(\varphi(t)\big) - j\cdot sin\big(\varphi(t)\big)\big] = \\ &= x\cdot cos\big(\varphi(t)\big) + y\cdot sin\big(\varphi(t)\big) + j\cdot\big[y\cdot cos\big(\varphi(t)\big) - x\cdot sin\big(\varphi(t)\big)\big].\end{aligned} \tag{4.6}$$

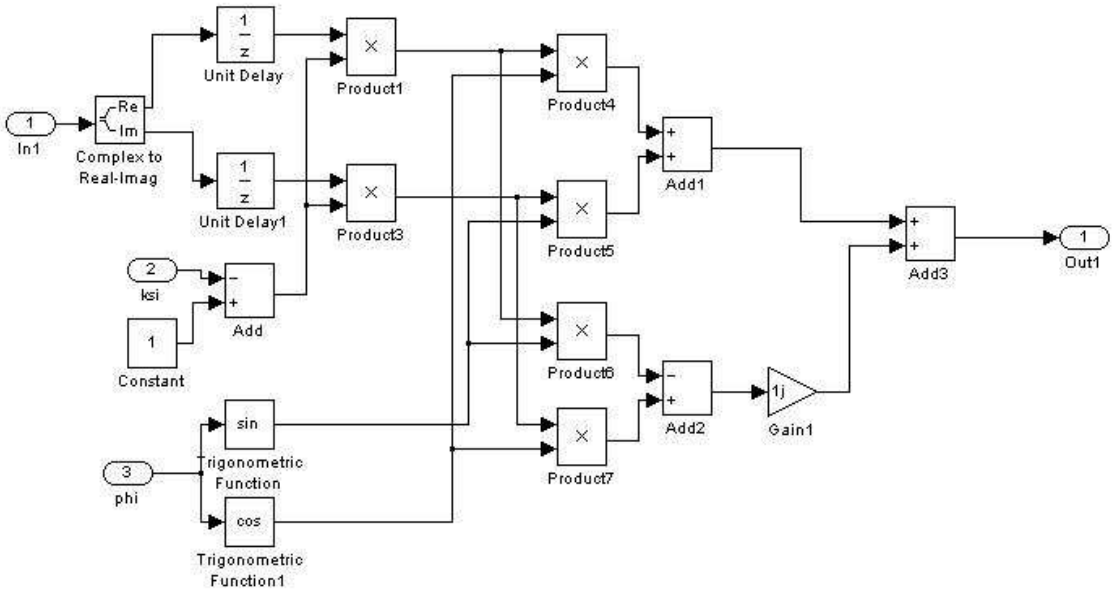
Ebből láthatjuk, a valós és képzetes rész előállításához külön-külön szükség van kettő szorzást megvalósító komponensre, egy összeadást/kivonást megvalósító komponensre, illetve elő kell állítani a fáziszaj adott időponthoz tartozó értékének szinuszát és koszinuszát. Ez utóbbi műveletet a már bemutatott CORDIC algoritmusokat megvalósító komponensek közül a CORDIC SINCOS komponens egyszerre előállítja. Így a fáziszaj eltávolítását visszavezettük olyan műveletekre, melyek a xilinx komponensek alapegységeivel megoldható (a CORDIC SINCOS komponens, mint minden más CORDIC algoritmust megvalósító komponensek szorzó, összeadó és kivonó műveleteket, valamint regisztereket tartalmaz).

Az amplitúdózaj eltávolításához, ahogy azt már korábban láthattuk $\big(1 + \xi(t)\big)$-vel kell osztanunk a Hilbert egység kimenő jelét. Mivel az $\big(1 + \xi(t)\big)$ értéke időfüggő, ezért nem lehet a reciprokát konstansként megadni, és azzal szorozni a kívánt jelet. Ahhoz, hogy minden időpontban a megfelelő számmal osszunk, azt a Normalizer blokkban már bemutatott CORDIC DIVIDER komponenssel meg lehet valósítani. Ha jobban szemügyre vesszük a 2.5. ábra, illetve a 4.6. ábra grafikonjait, akkor láthatjuk, hogy a kezdeti tranzienseket követően a relatív amplitúdózaj nullához közeli értékeket vesz fel (-0,2 és 0,2 között). Ezt a tényt felhasználva némi matematikai átalakítással kiváltható az $\big(1 + \xi(t)\big)$-vel való osztás. Az osztás kiküszöböléséhez először írjuk át az osztást a következő alakra:

$$\frac{1}{1 + \xi(t)} = \big(1 + \xi(t)\big)^{-1} \approx 1 + (-1)\cdot\xi(t) = 1 - \xi(t) \tag{4.7}$$

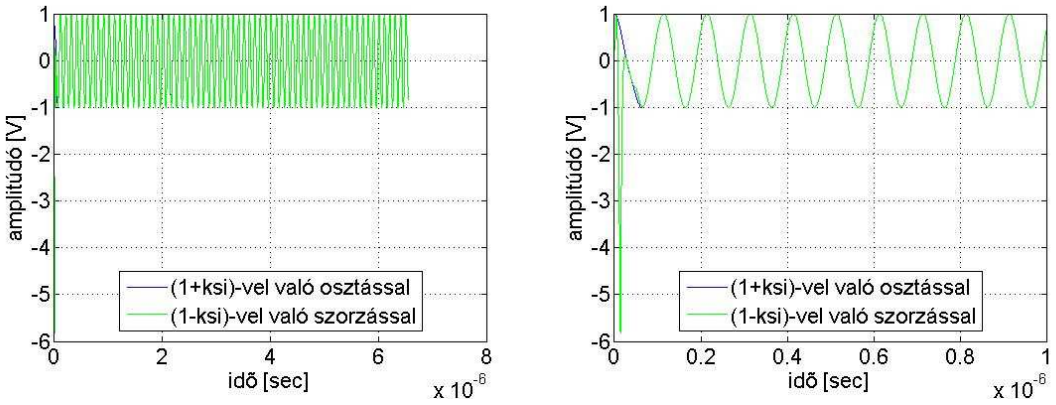
Ezt követően a relatív amplitúdózaj értékeit figyelembe véve kihasználhatjuk, hogy egy nulla közeli számot négyzetre emelve még inkább nullához közeli számot kapunk. Az ilyen kis számot elhanyagolva a nevezőben 1 marad, azaz az $\big(1 + \xi(t)\big)$-vel való osztás jó közelítéssel helyettesíthető $\big(1 - \xi(t)\big)$-vel való szorzással.

A bemutatott közelítés helyességének ellenőrzéseként a 3.1. ábrán látható modellhez egy másik Noise extractor blokk is került, mely a 4.9. ábrán látható. Az ábrán láthatjuk, hogy ez a blokk az osztás közelítése mellett a fáziszaj eltávolításához szükséges átalakításokat is tartalmazza.



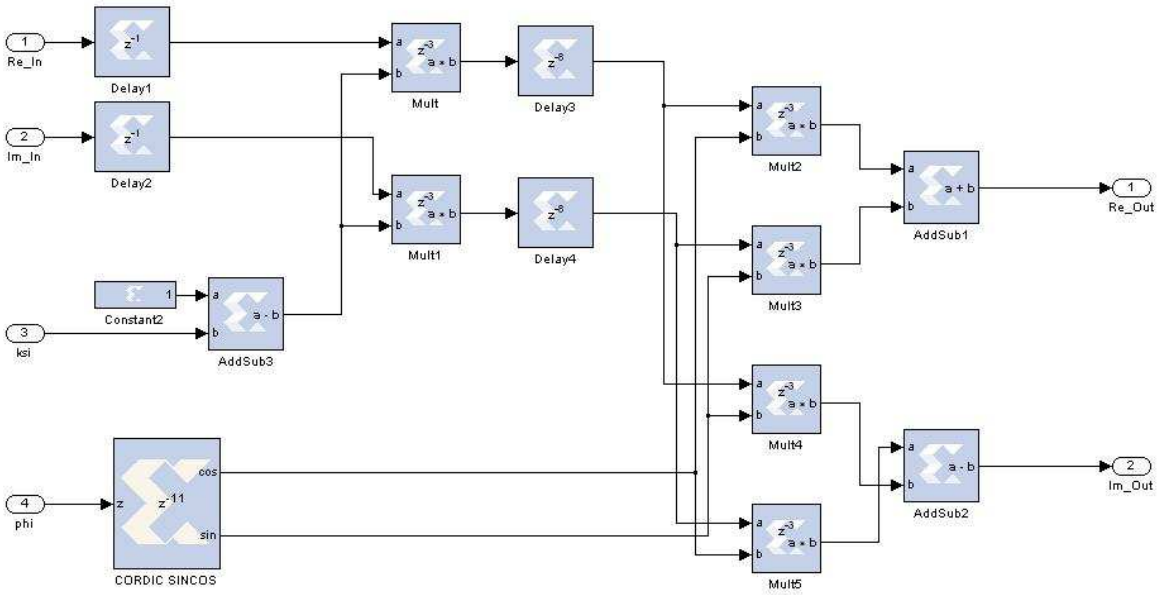
4.9. ábra: Az osztás közelítéséhez használt Simulink blokk.

A közelítés eredménye a 4.10. ábrán látható. Kékkel a 3.4. ábrán is látható osztással kapott kimenet látható, míg zölddel a közelítés eredménye. A bal oldali ábra a teljes bemeneti adatsorhoz tartozó eredményt mutatja, a jobb oldali ábra az pedig az első 10 periódust. A jobb oldali ábrán jól látható, hogy a kezdeti tranzienseket követően a zöld görbe teljesen lefedi a kéket.



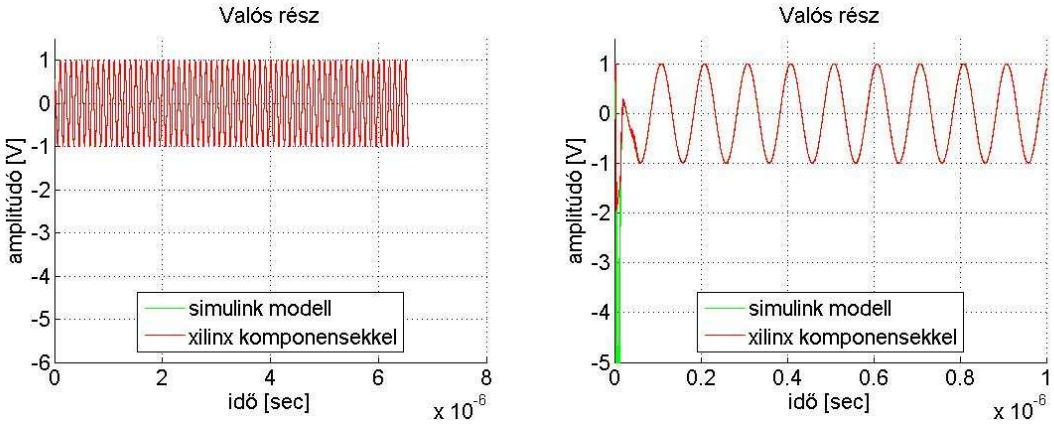
4.10. ábra: Az osztás közelítésének eredménye.

A kapott eredményeket követően xilinx komponensekkel is elvégeztem a szimulációt. A xilinx komponensekkel elkészült blokk a 4.11. ábrán látható. Az ábrán látható blokkot a korábbiakhoz hasonlóan Simulink–beli megfelelőjével le lett ellenőrizve, úgy hogy a bemeneti jeleket az eredeti Simulink-ben elkészített egységek szolgáltatják.



4.11. ábra: Noise extractor xilinx komponensekkel.

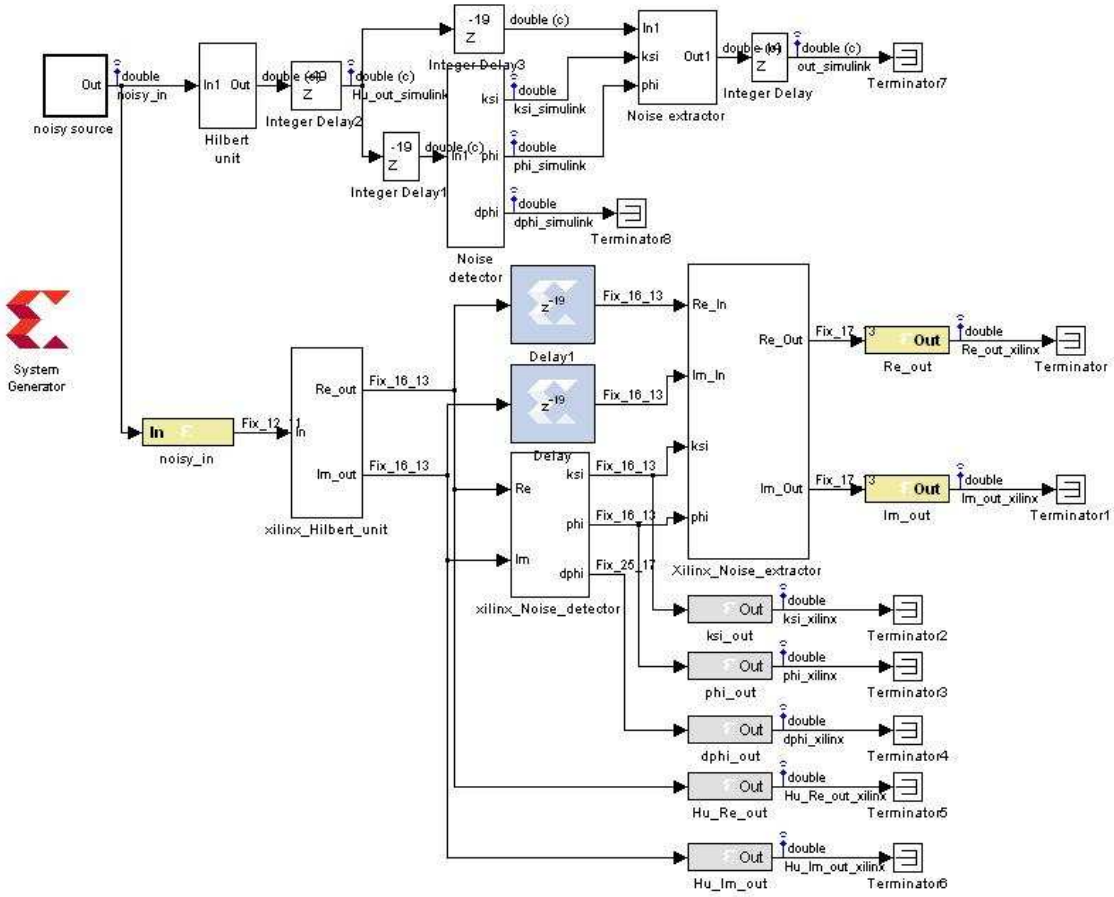
Az összehasonlítás eredménye a 4.12. ábrán látható. Zölddel az előző Simulink modellel bemutatott osztás közelítésének eredménye, pirossal pedig a xilinx komponensekkel megvalósított blokk eredménye. A grafikonokon jól látható, hogy a kezdeti tranzienst követően a Simulink-es blokk, illetve a xilinx-es blokk eredménye megegyezik (mindkét grafikonon a tranzienst követően teljesen fedi a piros görbe a zöldet). A tranziensnél látható különbség valószínűleg a xilinx komponenseknél alkalmazott számábrázolás paraméterei miatt van, de a szimuláció időtartamára való tekintettel erre az eltérésre vonatkozóan nem végeztem több szimulációt.



4.12. ábra: Noise extractor blokkok kimeneti jeleinek összehasonlítása.

5 Xilinx komponensekből összeállított rendszer

Az előző fejezetben bemutatott xilinx komponensekből már könnyedén összeállítható a 3.1. ábrán látható zajcsökkentést megvalósító rendszer. Az így elkészült szimulációs modell az 5.1. ábrán látható.



5.1. ábra: Xilinx komponensekből összeállított zajcsökkentő rendszer.

Az ábrán látható, hogy a 3.1. ábrához képest késleltetőkkel is kiegészült a modell. A xilinx komponensekből álló részben azért volt szükség a késleltető elemekre, hogy az utolsó (a zajcsökkentést végző) blokk egyszerre kapja meg a szükséges bemeneteket, míg a felső részen látható Simulink elemekből álló részben azért van szükség a késleltető blokkokra, hogy a két szimulációs részeredményei összehasonlíthatóak legyenek. Az ábrán az is látható, hogy a teljes mértékű összehasonlíthatóság miatt minden főbb alkotórész kimenete rögzítésre kerül.

Tekintettel arra, hogy a használt bemeneti adatsorral az egyes részegységek szimulációja is sok időt vett igénybe, ezért a teljes rendszer szimulációjára egy másik bemeneti adatsort érdemes használni, amely az eddiginél kevesebb időponthoz tartozó amplitúdót tartalmaz, azaz az előző mintavételi frekvenciánál (40 GHz) kisebbet használjunk. Mivel az a célunk, hogy az FPGA valós időbe végezze a zajcsökkentést, ezért az analóg bemeneti jel mintavételezését is célszerű az FPGA-val elvégeztetni. Kézenfekvő ötlet, hogy a használni kívánt FPGA órajelével megegyező mintavételi frekvenciával készüljön az új adatsor a szimulációhoz. Az általunk használni kívánt Virtex-4 c4vfx20 –as FPGA órajele 100 MHz. Ez azt jelenti, hogy a használt 10 MHz-es zajos jelforrás egy periódusából tíz darab mintánk lenne, amely megfelel a mintavételi kritériumnak. Ha figyelembe vesszük a zajcsökkentést végző rendszerünk főbb lépéseit láthatjuk, hogy nem választható akármekkora mintavételi frekvencia.

A rendszernek ebből a szempontból kritikus egysége a Hilbert egységben található, a bemeneti jel hullámhosszának negyedével késleltető egység, mellyel a 90°-os eltolást/késleltetést végezzük el a komplex jel képzetes részének előállításához. Azért nem használható tetszőleges mintavételi frekvencia, mert a diszkrét idejű rendszerekkel az említett 90°-os késleltetés elvégzéséhez az szükséges, hogy egész számú minta essen egy negyed periódusra. Azaz ebből a szempontból a következő kritériumnak kell megfelelnie a mintavételi frekvenciának:

$$f_s = n \cdot 4 \cdot f_0, \tag{5.1}$$

ahol:

f_s: a lehetséges mintavételi frekvenciák,

f_0: a bemeneti jel frekvenciája (esetünkben 10 MHz),

n: tetszőleges pozitív egész szám (hány darab minta tartozik negyed periódushoz).

Választásunk az FPGA órajelénél (100 MHz) kisebb (de a lehető legnagyobb) értékű mintavételi frekvenciára esett, amely 80 MHz. Ezzel a mintavételi frekvenciával minden negyed periódushoz két minta tartozik, így teljesül a mintavételi kritérium, és az általunk megvalósítandó 90°-kal történő eltolás kritériuma is.

A választott 80 MHz-es mintavételi frekvenciával új oszcilloszkópos mérés elvégzésére van szükség, vagy kihasználhatjuk, hogy a korábban használt adatsor 40 GHz-es mintavételi frekvenciája maradék nélkül osztható a választott új mintavételi frekvenciával. Így elegendő a már meglévő adatsort módosítani. Az adatsor módosítására egy Matlab függvényt készítettem, mely egy általunk megadható nevű *.mat kiterjesztésű fájlba elmenti a kívánt mintavételi frekvenciájú (maradék nélkül osztható legyen vele a 40 GHz) adatsort. A használt függvény a következő:

```
function new_input_data(fs_new,variablename,xy)
    fs=1/(xy(2,1)-xy(1,1));
    delta=fs/fs_new;
    L=length(xy(:,1))/delta;
    xy2=zeros(L,2);
    delta=int32(delta);

    xy2(1,1)=xy(1,1);
    xy2(1,2)=xy(1,2);

    for i=1:L
        xy2(i+1,1)=xy(i*delta+1,1);
        xy2(i+1,2)=xy(i*delta+1,2);
    end

    xy=xy2;
    save(variablename,'xy');
```

A függvénynek négy paraméter megadására van szükség:

fs_new: az új adatsor mintavételi frekvenciája,

variablename: a fájl neve, amelybe elmenti az új adatsort (pl.: ’xy_new.mat’),

xy: az eredeti adatsor.

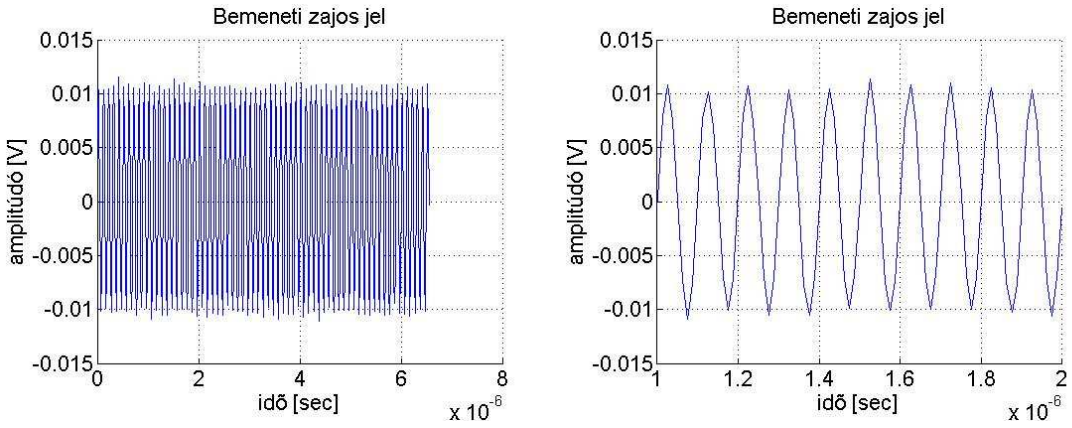
A függvény kiszámolja az eredeti adatsor mintavételi frekvenciáját, majd kiszámítja, hogy az eredeti adatsor mely mintáit kell megtartania, melyeket kigyűjt egy külön mátrixba, amit a kívánt fájlba elment. Az új fájl beolvasását követően már használható a korábbi rendszer paraméterezésére készített Matlab script.

A rendszer paramétereinek beállítására használt script:

```
%********************** initiation ************************
load('xy_new.mat');
LL=length(xy)/2;
f0=1e7;                          % signal freq
```

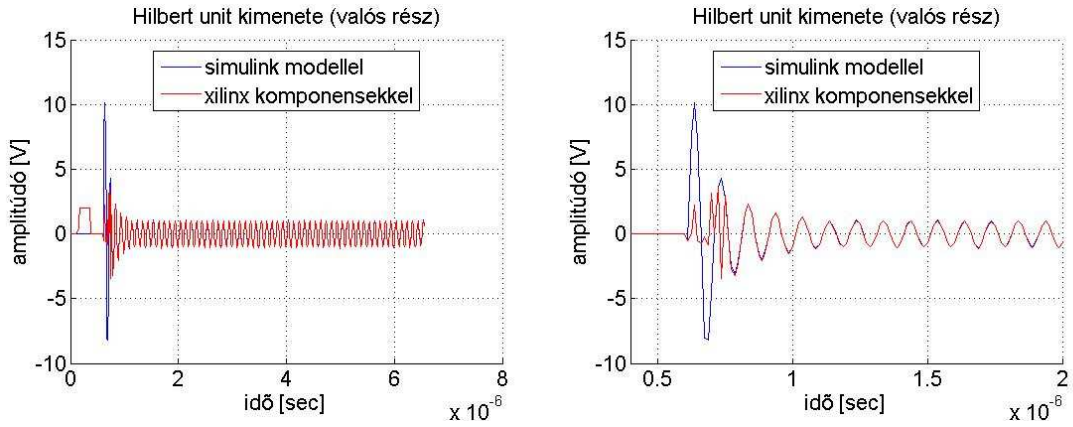
```
Ts=xy(2,1)-xy(1,1);              % sampling time
Fs=1/Ts;                         % sampling frequency
Tstep= Ts;                       % analysis timestep
Nscope=LL;                       % scope point number
Tstop= Nscope*Tstep;             % end of analysis

%********************* Hilbert egység *************************
Norm=double(int64(5*1/Ts/f0));      % number of averaging in normalizer
Nodc=double(int64(5*1/Ts/f0));      % number of averaging in nodc block
delay=double(int64(1/f0/4/Ts));     % transport delay

%******************** Noise detector ***********************
%************************  PLL  ****************************
om1=0e6;
om2=12e11;
v1=-1;
v2=1;
KVCO=2*pi*(om2-om1)/(v2-v1);      % VCO gain
df=0;                            % initial freq shift
Nave=double(int64(5*1/Ts/f0));  % averaged points at the loop
```

Az új adatsorral a szimuláció, már sokkal kevesebb idő alatt elvégezhető (néhány percet vesz igénybe). Az egyes blokkok kimeneti időfüggvényei a következő ábrákon láthatók. Elsőként az új adatsorból származó bemeneti zajos jel látható az 5.2. ábrán. Összehasonlítva a 3.3. ábrán látható zajos bemeneti jellel jól látható a mintavételi frekvencia csökkentésének hatása. Az új „jelforrás” zajos jele sokkal kevésbé tűnik zajosnak a korábban használthoz képest. Ennek oka, hogy korábban a 10 MHz-es jel egy periódusa alatt 4000 minta állt rendelkezésre, míg az új adatsornál periódusonként csak 8 minta. Ennek ellenére a bal oldali ábrán az amplitúdózaj hatása továbbra is jól látható.

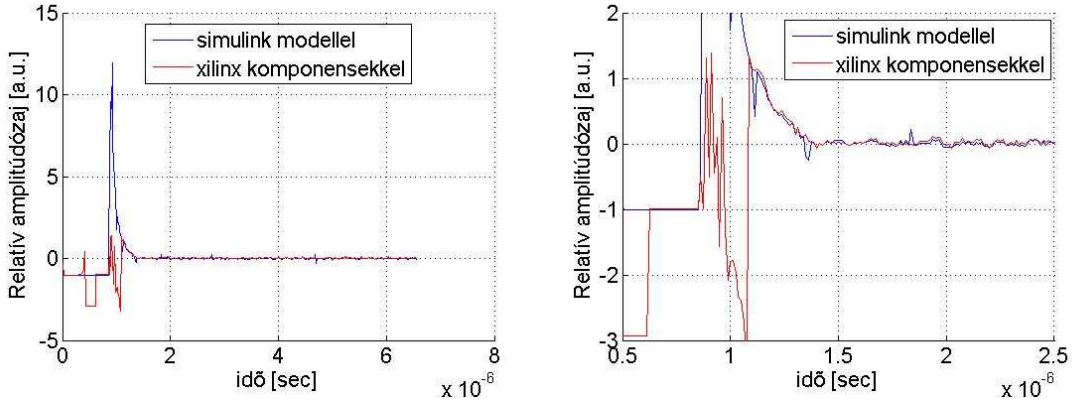


5.2. ábra: Az új adatsorból származó bemeneti zajos jel.

Az 5.3. ábrán láthatjuk a Simulink blokkokból (kék görbe), illetve a xilinx komponensekből álló (piros görbe) Hilbert egységek kimeneteinek összehasonlítását. A bal oldali grafikonon a teljes adatsorhoz tartozó kimeneti jel látható, míg a jobb oldali grafikonon jól láthatók a kezdeti tranziensek (1 µs előtti tartomány) eltérései. A bal oldali ábrán láthatjuk, hogy körül-belül 1 µs-tól a Simulink-es, illetve a xilinx komponensekből álló blokkok kimenetei megegyeznek. Az egységek kimeneteinek képzetes részénél ugyanezek a jelenségek figyelhetők meg a periódusidő negyedével késleltetve.



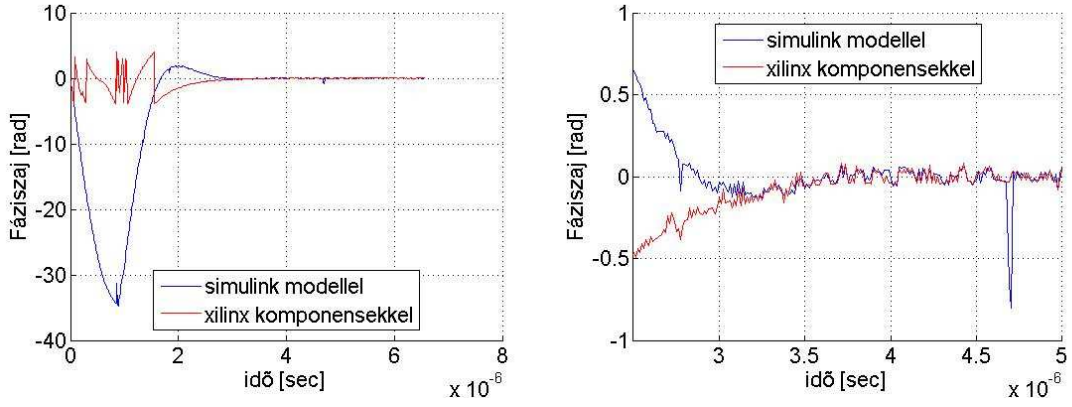
5.3. ábra: Hilbert egységek kimeneteinek összehasonlítása (valós részek)



5.4. ábra: A Simulink modellből és a xilinx komponensekből származó relatív amplitúdózajok összehasonlítása.

A Simulink modellel, és a xilinx komponensekkel előállított relatív amplitúdózajok időfüggvényeit az 5.4. ábra grafikonjai hasonlítják össze. Az eddigiekhez hasonlóan kékkel a Simulink blokkból származó relatív amplitúdózaj látható, míg pirossal a xilinx komponensekből álló rendszer amplitúdózaja. A jobb oldali grafikonon külön kiemelve látható a kezdeti tranziens jelenség, valamint az is látható, hogy 1,5 µs-tól véget érnek a tranziensek, a két görbe kisebb eltéréseket leszámítva illeszkednek egymásra, valamint elérik a nulla várhatóértéket. A 4.6. ábrával összehasonlítva látható, hogy a mintavételezés változása a relatív amplitúdózaj tranziens szakaszának jellegére is hatással van. Az 5.4. ábra tranziens szakaszánál láthatjuk, hogy a két görbe jelentősen eltér egymástól. Ennek az eltérésnek az oka a xilinx komponenseknél használt számábrázolás, ugyanis a teljes rendszer összeállításával az egyes egységek kimenetein használt bitmennyiségek minimalizálva lettek, hogy a feldolgozás további szakaszaiban végzendő műveleteket minél kevesebb bemeneti biten kelljen elvégezni (ezzel is gyorsítva az egyes műveleteket). Az amplitúdózaj egyes értékeihez 16 bitet használunk, melyből 13 bit az érték tört részét írja le, míg a maradék 3 bit az egész értékeket. Mivel a relatív amplitúdózaj negatív értékeket is felvehet, ezért kettes komplemens alakot kell használnunk, azaz a 3 egész értéket ábrázoló bit közül a legnagyobb helyiértékű az előjelet határozza meg. Ezért nem vehet fel akármekkora értéket a xilinx komponensekből származó amplitúdózaj.

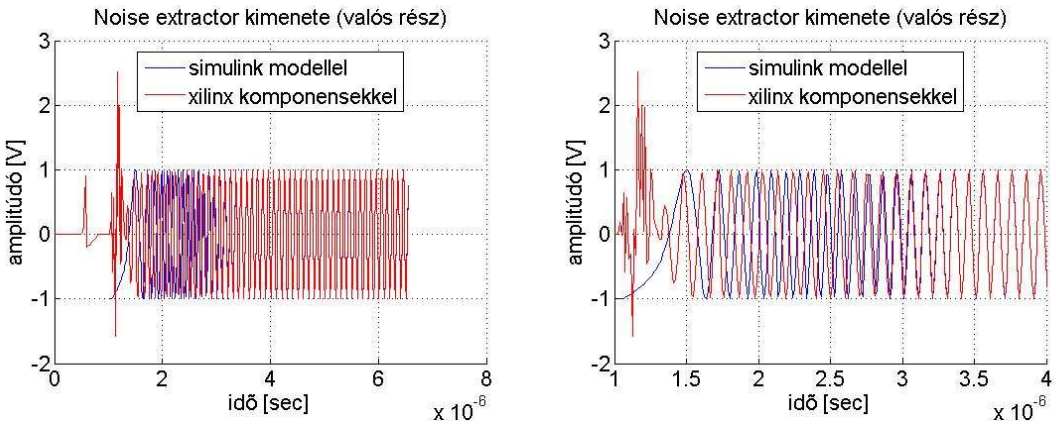
Az amplitúdózajhoz hasonlóan a fáziszajok is összehasonlításra kerültek, amely az 5.5. ábra grafikonjain látható az előzőekkel egyező színezéssel.



5.5. ábra: A Simulink modellből és a xilinx komponensekből származó fáziszaj összehasonlítása.

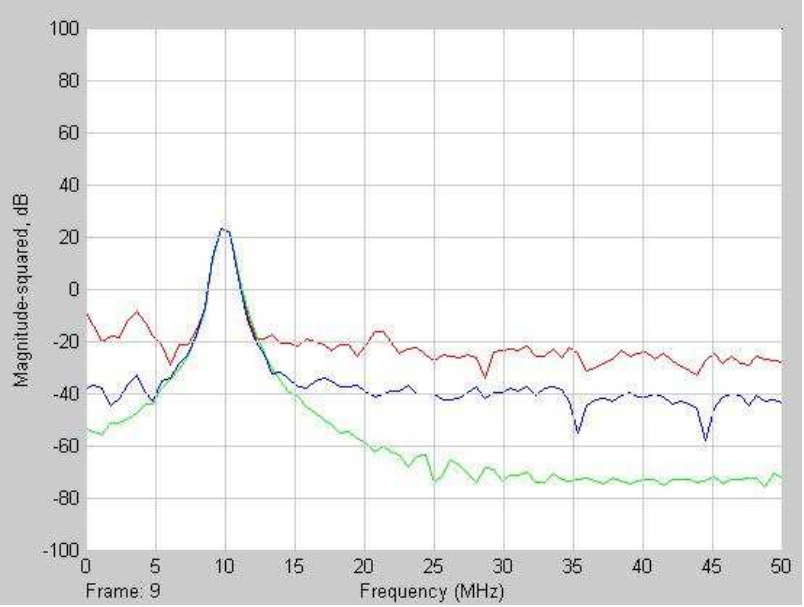
Az ábrán jól látható az amplitúdózajhoz hasonló eltérés a kezdeti tranziens szakaszban, valamint az is jól látható, hogy a két görbe a 4.7. ábra grafikonjaihoz képest 1-1,5 µs-mal később éri el a nulla várhatóértéket. Ennek oka, hogy az új adatsor miatt a fáziszárt hurok feszültség vezérelt oszcillátorának paraméterét módosítani kellett a megfelelő lockolás végett.

Ezeket az eredményeket követően nem meglepő a Noise extractor kimeneteinek eredményei, melyek az 5.6. ábrán. Ezeken a grafikonokon is jól látható a kezdeti tranziensbeli eltérés, valamint jól látható a zajparaméterek konvergenciájának hatásai is. A jobb oldali grafikonon jól látható a relatív amplitúdózajoknak nulla körüli együttes ingadozása 1,5 µs-tól (5.4. ábra), ugyanis ettől a ponttól kezdve a két görbe azonos amplitúdójú. Ezen kívül a grafikonon jól látható a fáziszajok együttfutásának hatása is 3-3,5 µs-tól (5.5. ábra), amikortól a két görbe fázisa is megegyezik. A fázisok eltérése (3-3,5 µs előtt), illetve egyezése (3-3,5 µs után) a bal oldali grafikonon is jól látható.



5.6. ábra: Noise extrctorok kimeneteinek összehasonlítása.

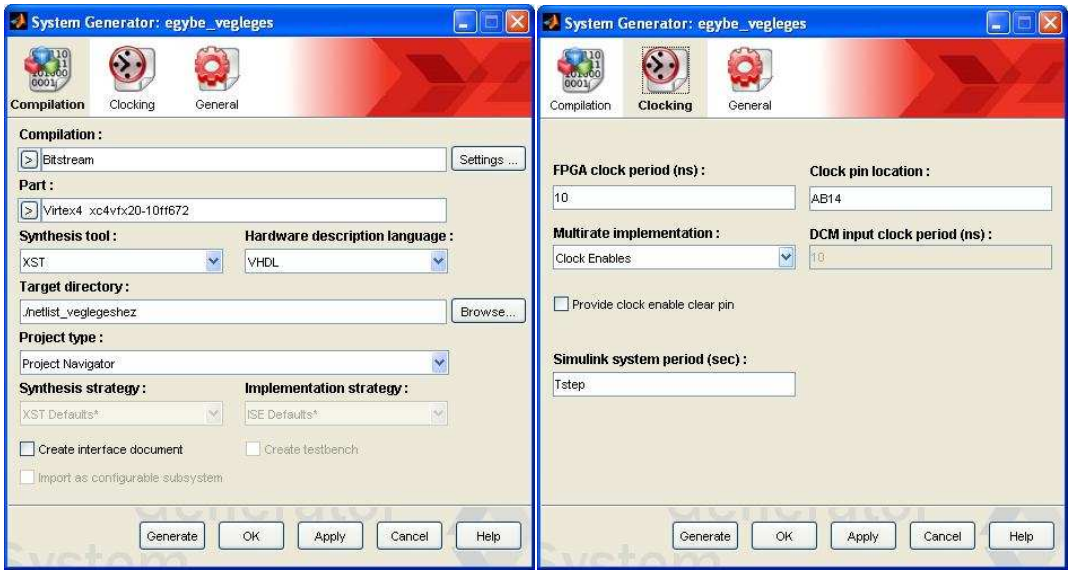
A fenti összehasonlító ábrákon jól látszik, hogy a xilinx komponensekből készített zajcsökkentő modul megfelelően működik az első 10-15 periódust (plusz a késleltetéseket) követően. Mielőtt tovább lépnénk érdemes a spektrumokra is kitérni. Az említett spektrum az 5.7. ábrán látható. Az ábrán a piros görbe mutatja a zajos jel spektrumát, a zöld görbe a Simulink modellel megtisztított jel spektrumát, míg a kék görbe a xilinx komponensekkel „megtisztított” spektrumot mutatja. Az ábrán látható, hogy 10 MHz-en mind a három spektrum megegyezik, de 10 MHz-től távolodva a zajszinteken megmutatkozik a három görbe közötti különbség. A zajos jel, és az ideálisan megtisztított (Simulink modellel) jel zajszintjei között 40-50 dB-es eltérés is van, míg a zajos jel, és a xilinx komponensekkel „megtisztított” jel zajszintje között közel 20 dB-es különbség van. A két „megtisztított” jel zajszintje közötti eltérés jelentős részét a fáziszárt hurokban alkalmazott integrál területszámítással végzett közelítése okozza.



5.7. ábra: Spektrumok összehasonlítása. Piros: zajos jel spektruma;
Zöld: Simulink modellel „megtisztított" jel spektruma; Kék: xilinx komponensekkel „megtisztított" jel spektruma

6 Xilinx komponensekből elkészített modell bitstreammé konvertálása

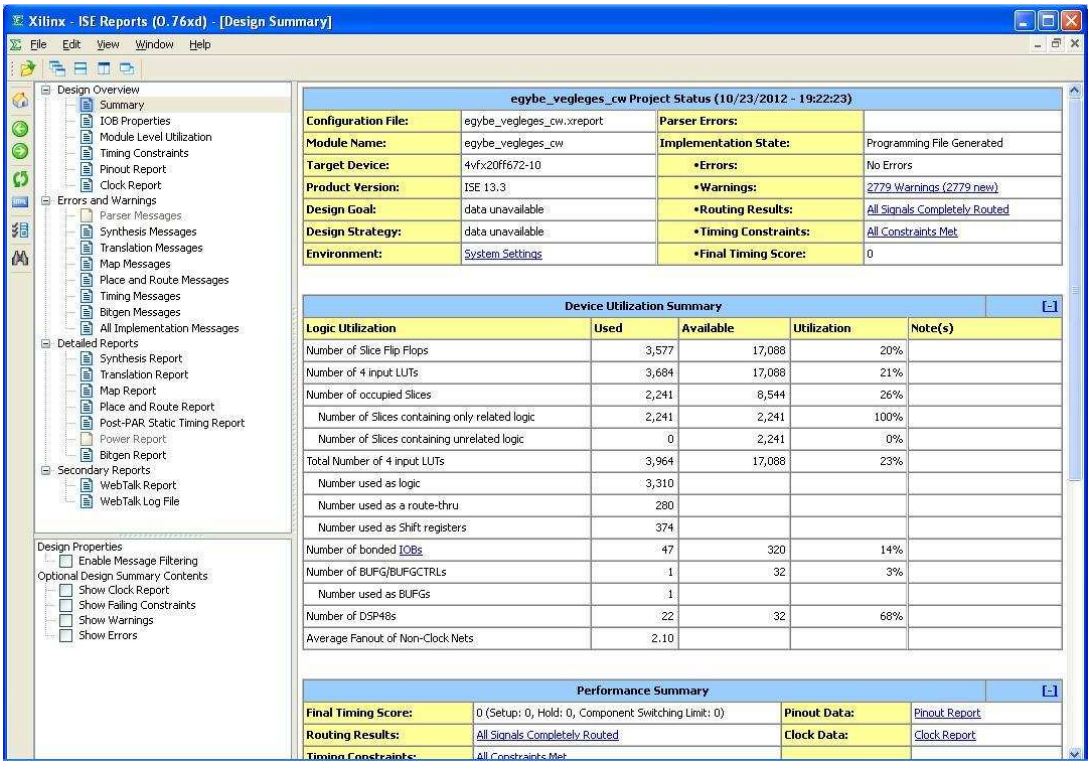
A xilinx komponensekből elkészült rendszer szimulációkkal történő ellenőrzését követően a következő feladat a rendszer bitstreammé konvertálása. Ezt a feladatot az 5.1. ábrán is látható System Generator tokennel lehet elvégeztetni. A konvertáláshoz a 6.1. ábrán látható ablakok paramétereit kell megfelelően beállítani.



6.1. ábra: System Generator token beállítása.

A bal oldali ablakban a legfontosabb paraméter a használni kívánt FPGA típusa, a hardver leíró nyelv kiválasztása. A jobb oldali ablakban definiálni kell a kapcsolatot az FPGA órajele, illetve a Simulinkben alkalmazott rendszer periódusideje (a T_{step} paraméterrel megadott érték) között. Az FPGA órajelének hozzáférhetőnek kell lennie, ehhez meg is kell adni pin-jének helyzetét. Jelen pillanatig még nem sikerült megoldani, hogy a tokennel olyan órajel legyen beállítható, melyet az FPGA órajeléből valamilyen módon előállítunk, ezért az ábrán beállított érték az FPGA központi órajele.

Az így generált bitstream összefoglalója a 6.2. ábrán látható. Az összefoglalóban láthatjuk, hogy hibaüzenet nincsen, viszont rengeteg figyelmeztetés van. Ezen figyelmeztetések kiküszöbölése jelenleg is folyamatban van. Az összefoglalóban láthatunk egy táblázatot is, amely az FPGA-n elérhető, és a használt különböző erőforrások számát mutatja. Ebből a táblázatból láthatjuk, hogy a zajcsökkentő rendszerhez szükséges erőforrások száma az elérhető erőforrások száma alatt van, azaz a használni kívánt FPGA-ra rátölthető a rendszer a figyelmeztetések kiküszöbölése után.



6.2. ábra: Generált bitstream összefoglalója.

A hibaüzenetek kiküszöbölése után az így átkonvertált rendszernek már képesnek kell lennie a szimulációhoz hasonlóan egy előre digitalizált adatsoron elvégezni a zajcsökkentő eljárást az FPGA belső 100 MHz-es órajelének megfelelően. A mi célunk azonban az online működés elérése, melyhez az előző fejezetben leírtak alapján a 100 MHz-es órajel helyett 80 MHz-es órajel szükséges.

A kívánt órajel az FPGA-ban DCM (Digital Clock Manager) segítségével előállítható a megfelelő szorzó (M) és osztó (D) értékeinek beállításával (egész számok legyenek). Az órajel előállításához szükséges VHDL kód a Xilinx CORE Generator segítségével a szükséges paraméterek grafikus felületen történő megadásával egyszerűen elkészíthető.

A félév további részében a System Generatorral létrehozott bitstream figyelmeztetéseinek ellenőrzését, az esetleg szükséges javításokat követően, egy közös projektben össze kell kapcsolni a CORE Generatorral összeállított DCM VHDL kódjával, majd az ellenőrzéseket, illetve teszteléseket követően az összevont bitstream rátölthető az FPGA-ra, és kipróbálható lesz a gyakorlatban is mérhető lesz a zajcsökkentő eljárás hatása. Ezt követően a kétfokozatú zajcsökkentő valósidejű realizálása a cél.

7 Összefoglalás

A dolgozat elején bemutatásra került egy a kvadratúra demodulátor elvén működő relatív amplitúdó- és fáziszaj meghatározó eljárás. A dolgozatban bemutattuk a zajparaméterek ismeretének egy lehetséges felhasználását, mellyel egy szinuszos jel „megtisztítása” a célunk.

A dolgozatban Matlab/Simulink program felhasználásával szimulációk segítségével bemutatásra került a zajcsökkentő eljárás (off-line) eredményei, melynek segítségével az eregeti jel zajszintje 40-50 dB-lel is csökkent. A félév során további célunk, hogy a bemutatott zajcsökkentő eljárást valós időben (online) a rendelkezésre álló Virtex-4 c4vfx20 –as FPGA-n megvalósítsuk. Az FPGA-ra rátöltendő bitstream előállítása továbbra is a Matlab/Simulink programmal történt a Xilinx System Generator for DSP programjának segítségével. Ez utóbbi program lehetővé teszi, hogy a már megszokott Simulink-es környezetbe xilinx komponensek felhasználásával összeállítsuk a kívánt rendszert, majd a System Generator token segítségével elkészítsük az FPGA-ra töltendő bitstreamet. A dolgozatban bemutatásra kerültek a zajcsökkentő rendszer főbb elemeinek működései, azok xilinx komponensekkel történő megvalósításai a szükséges módosításokkal együtt, valamint a főbb elemek szimulációs eredményeinek összehasonlításával meggyőződhettünk a xilinx komponensek felhasználásával készült rendszer helyes működéséről. A szimulációval kapott spektrumon láthattuk a szükséges átalakítások hatását az ideálistól való eltérésre, melyek ellenére a xilinx komponensekből álló rendszer alkalmas a zajcsökkentésre.

Miután meggyőződtünk a rendszer helyes működéséről a System Generator tokennel elkészíthettük az FPGA-ra töltendő bitstreamet. A jelenlegi cél a bitstream készítése közben keletkezett figyelmeztetések kijavítása, valamint az FPGA belső órajelétől eltérő, nem külső órajel használatának megoldása.

A továbbiakban az eddig használt 10 MHz-es oszcillátor online zajcsökkentésének kipróbálása a cél. Sikeres eredmények esetén a kétfokozatú zajcsökkentés megvalósítása a következő jelentős lépés, majd az elkészült online zajcsökkentést egyéb rendelkezésre álló rendszerek vivőjén is szeretnénk kipróbálni. Ezeket követően érdemes különböző modulált jeleken is kipróbálni a zajcsökkentést (pl.: GPS adatjelének zajcsökkentésére). Az FPGA-k órajelének növekedésével lehetővé válhat a 10 MHz-nél nagyobb frekvenciájú jelek zajcsökkentése is.

8 Irodalomjegyzék:

[1] H. W. Ott: “Noise reduction techniques in electronic systems”, Wiley, 1988

[2] S. V. Vaseghi: “Advanced signal processing and digital noise reduction”, Wiley, 1996

[3] E. Hegazi, H. Sjöland and A. A. Abidi: “A filtering technique to lower LC oscillator phase noise”, IEEE Journal of Solid-State Circuits, Vol. 36, No. 12, December 2001, pp. 1921-1930

[4] J. Ladvánszky, G. Kovács: “Software Based Separation of Amplitude and Phase Noises in Time Domain”, Proc. of ISCAS’2011, Rio de Janeiro, Brazil, pp. 769-772

[5] J. Ladvánszky, G. Mészáros, Cs. Fűzy: “Spectral cleaning for oscillators”, IEEE International Conference on Signal Processing, 21-25 October 2012, Beijing, China

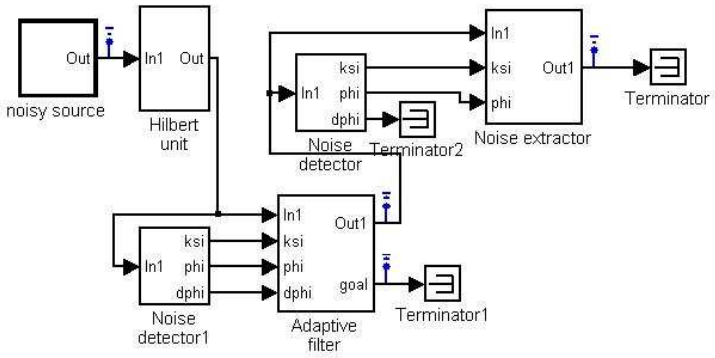
[6] G. Mészáros, Cs. Fűzy and J. Ladvánszky: “Does amplitude or phase noise arise first?”, Electronics Letters, July 7, 2012, pp. 692-693

[7] Bertsekas, Dimitri: “Convex Analysis and Optimization”, Athena Scientific, 2003

9 Melléklet

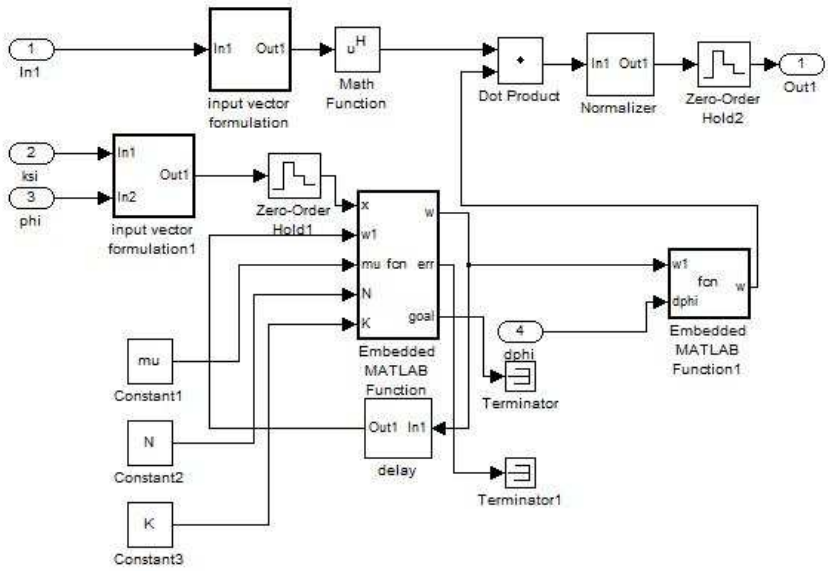
9.1 Kétfokozatú zajcsökkentés

A kétfokozatú zajcsökkentés végző rendszer a 9.1. ábrán látható. A mellékletben csak az adaptív szűrőt mutatom be, mivel a rendszer többi eleme a dolgozatban már részletesen megjelentek.



9.1. ábra: A kétfokozatú zajcsökkentést végző rendszer.

Az adaptív szűrővel meghatározható az alapsávi zaj korreláció négyzetösszegének minimalizálásához szükséges súlytényezőket, majd áttranszformálja a meghatározott alapsávi súlytényezőket a jel frekvenciájára. Az így megkapott súlytényezőkkel kompenzálja a zaj keletkezésének helye, és a zaj mérésének helye közötti átviteli függvényt. Itt kihasználjuk azt a tényt, hogy nincs korlátozva az amplitúdó- vagy a fáziszaj eltávolításának sorrendje [6]. Az adaptív szűrő felépítése a 9.2. ábrán látható.



9.2. ábra: Adaptív szűrő felépítése.

Az ábra felső részén az adaptív szűréshez szükséges blokkok láthatók. A szükséges súlytényezők meghatározását az ábra közepén található Matlab függvényt tartalmazó blokk végzi, míg a súlytényezők alapsávról a jel frekvenciájára történő transzformációt a jobb oldalon található Matlab függvényt tartalmazó blokk végzi.

A súlytényezők meghatározása:

Először kiszámítjuk az adaptív szűrő célfüggvényét, majd keressük annak minimumát. Az adaptív szűrő alapsávi komplex bemeneti, és kimeneti mintáit x_i-vel és y_{N+1}-gyel jelöljük, ahol i és N+1 az időréseket jelölik.

$$y_{N+1} = \sum_{i=1}^{N} w_i \cdot x_i \tag{9.1}$$

ahol w_i az adaptív szűrőt jellemzi, az x_i mintákat a fázis- és a relatív amplitúdózaj határozza meg. Az így előállítható mintasorozat y_k, ahol $k = N+1, N+2, \dots N+K$. Ez a sorozat a bemeneti mintákkal is kifejezhető:

$$y_k = \sum_{i=1}^{N} w_{i+k-N-1} \cdot x_{i+k-N-1} \tag{9.2}$$

ahol K a kimeneti minták száma. Ez az egyenlet azt mutatja, hogy a következő kimeneti minta az előző N db kimeneti mintának a súlyozott összege. A kimeneti minták autokorrelációja:

$$R_{yy}(m) = \sum_{k=N+1}^{N+K} y_k^* \cdot y_{k+m} \qquad m = 0, 1, \dots K-1 \tag{9.3}$$

ahol a * jelzi a komplex konjugálást. Megfelelően behelyettesítve a (9.2)-es egyenletet a (9.3)-as egyenletbe:

$$R_{yy}(m) = \sum_{k=N+1}^{N+K} \left(\sum_{i=1}^{N} w_{i+k}^* \cdot x_{i+k}^* \right) \cdot \left(\sum_{i=1}^{N} w_{i+k+m} \cdot x_{i+k+m} \right) \qquad m = 0, 1, \dots K-1 \tag{9.4}$$

A (9.4)-es egyenletből következik, hogy összesen $N + 2 \cdot K - 2$ darab w szükséges, melyeket vektorként kezelhetünk:

$$\underline{w} = [w_1 \quad w_2 \quad \cdots \quad w_{N+2K-2}]^T, \tag{9.5}$$

ahol T a transzponálást jelöli. A Stochastic gradient descent módszert [7] alkalmazva:

$$\underline{w}_{next} = \underline{w} - \left(\mu_1 \cdot grad_{Re\,\underline{w}}(\varepsilon) + \mu_2 \cdot grad_{Im\,\underline{w}}(\varepsilon) \right), \tag{9.6}$$

ahol μ_1 és μ_2 a lépések méretét meghatározó konstansok, és $grad_{\underline{x}}$ egy olyan oszlopvektor, amely az $\underline{x}$ elemei szerinti ε-ra vonatkozó parciális deriváltakat tartalmazza. A használt célfüggvény:

$$\varepsilon(\underline{w}) = \sum_{m=1}^{K-1} \left| R_{yy}(m) \right|^2 \tag{9.7}$$

Ezt a célfüggvényt (ε) a $\underline{w}$ értékeinek változtatásával minimalizáljuk, hogy Dirac-delta jellegű autókorrelációs függvényt kapjunk:

$$R_{yy}(m) = \begin{cases} \neq 0 & m = 0 \; esetén \\ = 0 & minden \; m \neq 0 \; esetén \end{cases} \tag{9.8}$$

Mivel a meghatározott súlytényezőket a jel frekvenciájára kell transzformálni alapsávról, ezért a (9.7)-es, a (9.9)-es, és a (9.4)-es egyenletek felhasználásával fejezzük ki a szükséges deriválásokat a következő átalakításokkal:

$$\frac{\partial \varepsilon}{\partial Re\, w_n} = \sum_{m=1}^{K-1} \frac{\partial R_{yy}^*(m)}{\partial Re\, w_n} \cdot R_{yy}(m) + R_{yy}^*(m) \cdot \frac{\partial R_{yy}(m)}{\partial Re\, w_n} \tag{9.9}$$

$$\frac{\partial \varepsilon}{\partial Im\, w_n} = \sum_{m=1}^{K-1} \frac{\partial R_{yy}^*(m)}{\partial Im\, w_n} \cdot R_{yy}(m) + R_{yy}^*(m) \cdot \frac{\partial R_{yy}(m)}{\partial Im\, w_n}$$

$$\begin{aligned} \frac{\partial R_{yy}(m)}{\partial Re\, w_n} &= \sum_{k=0}^{K-1} \left(\sum_{\substack{i=1 \\ i+k=n}}^{N} x_{i+k}^* \right) \cdot \left(\sum_{i=1}^{N} w_{i+k+m} \cdot x_{i+k+m} \right) + \\ &+ \sum_{k=0}^{K-1} \left(\sum_{i=1}^{N} w_{i+k}^* \cdot x_{i+k}^* \right) \cdot \left(\sum_{\substack{i=1 \\ i+k+m=n}}^{N} x_{i+k+m} \right) \end{aligned} \tag{9.10}$$

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

$$\frac{\partial R_{yy}^*(m)}{\partial Re\, w_n} = \left[\frac{\partial R_{yy}(m)}{\partial Re\, w_n} \right]^* \tag{9.11}$$

$$\begin{aligned} \frac{\partial R_{yy}(m)}{\partial Im\, w_n} &= -j \cdot \sum_{k=0}^{K-1} \left(\sum_{\substack{i=1 \\ i+k=n}}^{N} x_{i+k}^* \right) \cdot \left(\sum_{i=1}^{N} w_{i+k+m} \cdot x_{i+k+m} \right) + \\ &+ j \cdot \sum_{k=0}^{K-1} \left(\sum_{i=1}^{N} w_{i+k}^* \cdot x_{i+k}^* \right) \cdot \left(\sum_{\substack{i=1 \\ i+k+m=n}}^{N} x_{i+k+m} \right) \end{aligned} \tag{9.12}$$

Az alapsávi, és jel frekvenciás szűrők átviteli függvényei közötti kapcsolat levezetése:

A következőkben $h_1(\tau)$-val jelöljük az alapsávi szűrő átviteli függvényét, és $h_2(\tau)$-val az alapsávi szűrőnek megfelelő jel frekvenciás szűrő átviteli függvényét. A bemeneti jelet $x(t)$-vel, a kimeneti jelet $y(t)$-vel jelöljük.

Alapsávi szűrő esetén a szűrést megelőzi a bemeneti jel lekeverése ($e^{-j \cdot \omega_0 \cdot t}$-vel való szorzás):

$$x_1(t) = x(t) \cdot e^{-j \cdot \omega_0 \cdot t} \tag{9.13}$$

$$y(t) = \int_0^{\infty} x_1(t - \tau) \cdot h_1(\tau)\, d\tau \tag{9.14}$$

A jelfrekvenciás szűrő esetén előbb a szűrést végezzük el, majd utólag keverjük le alapsávba az összehasonlítás érdekében.

$$y_1(t) = \int_0^{\infty} x(t - \tau) \cdot h_2(\tau)\, d\tau \tag{9.15}$$

$$y(t) = y_1(t) \cdot e^{-j \cdot \omega_0 \cdot t} \tag{9.16}$$

Mivel a két esetben azonosaknak kell lenniük a bemeneti és kimeneti jeleknek, ezért:

$$e^{-j \cdot \omega_0 \cdot (t-\tau)} \cdot h_1(\tau) = h_2(\tau) \cdot e^{-j \cdot \omega_0 \cdot t} \tag{9.17}$$

Ebből az egyenletből következik, hogy a súlyfüggvényekhez szükséges transzformáció az $e^{j \cdot \omega_0 \cdot \tau}$-val való szorzás.