Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Németh Péter

SZINKRONIZÁLÓ ÉS NAGY PONTOSSÁGÚ IDŐBÉLYEGEZŐ EGYSÉG TDOA ALAPÚ RENDSZEREKHEZ

KONZULENS

Moldován István

BUDAPEST, 2012

Tartalomjegyzék

Ábrajegyzék

1 Bevezetés

A XXI. században már megszokott és kényelmes szolgáltatás a műholdas navigáció, amivel pontosan tájékozódhatunk a Föld bármely pontján. Vannak viszont olyan helyzetek, amikor egy épületen belül lenne szükség a pozíciónk meghatározására, de ez a műholdas navigációs rendszer működéséből adódóan nem lehetséges. Szükségünk van tehát egy olyan beltéri lokalizációs rendszerre, amivel ez a probléma kiküszöbölhető.

Több megoldás is született az elmúlt időben, ezek egyike a Beérkezési Időkülönbség (a továbbiakban TDOA, Time Difference Of Arrival) alapú helymeghatározási módszer. A módszer lényege az, hogy a felhasználói készüléktől érkező jeleket a kiépített vezeték nélküli hálózatban több vevő gyűjti be egyidejűleg és ezeket egy-egy időbélyeggel ellátva küldik a feldolgozó egységnek, ami az időbélyegek különbségeiből számolja ki a felhasználó pontos helyzetét. A pontos helymeghatározás érdekében tudnunk kell a jelek nagyon pontos beérkezési idejét, ezért egy nagyon kis felbontású időbélyegező mechanizmus szükséges. A nagy pontosságot indokolja még, hogy az épületekben a közel fénysebességgel haladó rádió jelek pár nanoszekundum alatt több métert is meg tesznek.

Egy ilyen beltéri lokalizációs rendszer tervezésénél tehát nagyon fontos egy olyan egység megvalósítása, amely együtt tud működni a meglevő vezeték nélküli hálózati protokollokkal (pl. WiFi) és megoldja az elvárt nanoszekundum pontosságú időbélyegezést. Ehhez három fontos feladatot kell megvalósítani. Egyrészt az eszköznek meg kell oldania a vevő oldali szinkronizációt, ami a szoftverrádió (Software Defined Radio, SDR) **[1]** által vett jelek fázis demodulációjához szükséges. Ezután el kell végeznie a jelek dekódolását, és meg kell állapítania a kapott csomagok nagyon pontos beérkezési idejét. A dolgozatban ismertetett FGPA alapú hardver képes ezen feladatok megoldására, így nyújtva egy megfelelő alapot a beltéri TDOA alapú helymeghatározási rendszernek.

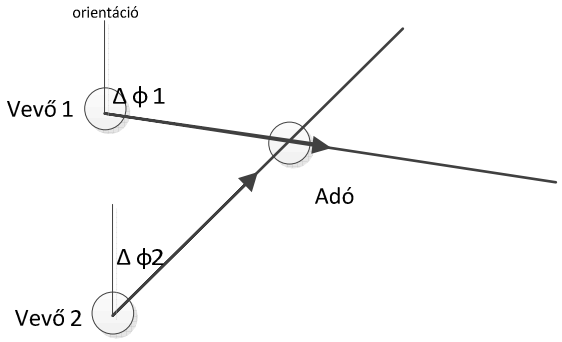
2 Beltéri pozicionálás

2.1 Beltéri pozicionálási módszerek

A beltéri lokalizáció többféle elv alapján elvégezhetjük, viszont mindegyikről elmondható, hogy a pontos pozíció kiszámításához az adó eszköz jeleit több vevőnek kell egyidejűleg észlelnie. Ebben a fejezetben egy összefoglalást adnánk a legelterjedtebb mechanizmusokról. **[2]**

2.1.1 Beesési szög alapú számítás

Ez a módszer a *Vett Jel Beérkezési Szögéből (AoA, Angle of Arrival)* határozza meg az adó helyzetét. A legegyszerűbben két irányított antennából álló rendszert kell telepíteni, amelynél ismernünk kell az egyes antennák orientációját is. Ekkor az egyes vevők által számított beesési szögekre fektetett egyenesek metszéspontja megadja az adó helyét, ahogy ez az **1. ábraán** is látható.



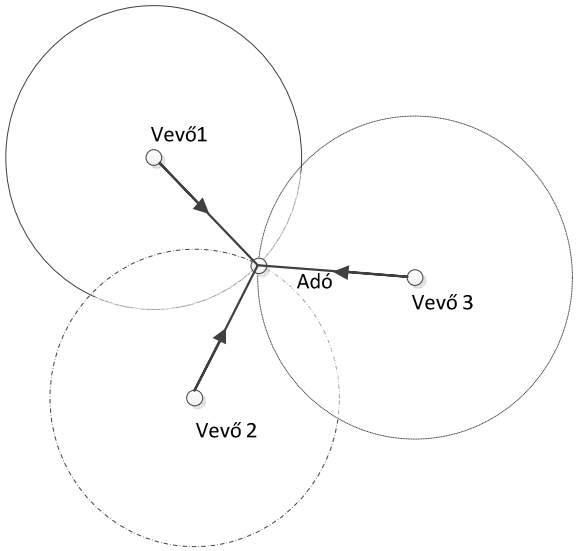
1. ábra Pozicionálás a beesési szögek alapján

Ez a viszonylag egyszerű megoldás csak akkor működik megfelelően, ha a vevőknek közvetlen rálátása van az adóra. Zegzugos helyeken, illetve olyan esetben, amikor pontosabb helymeghatározásra van szükségünk már egy komplexebb, több vevőből álló rendszert kell telepíteni. Ennek elkerülése végett a kereskedelmi és katonai rendszerekben több antennás vevőket alkalmaznak. Ekkor a beesési szög meghatározását egy külön számítási egység végzi el, abból az információból, hogy az antennaháló milyen eloszlással vette a jelet. A számítás része a beérkezési időkülönbség

(TDOA) számítása is mivel az antennaháló egyes elemei között fázis különbség léphet fel.

2.1.2 Laterális (távolság alapú) számítási módszerek

A fejezetben ismertetett módszerek közül a legkönnyebben implementálható a *Vett Jel Erősség (RSS, Received Signal Strength)* alapú pozicionálás. Ezzel a módszerrel a vevő által számolt *d* távolság csak azt mondja meg, hogy az adó a vevőtől egy *d* sugarú körben helyezkedik el. A pontos lokalizációhoz ezért minimum három vevőre van szükség: ekkor a három kör metszés pontja pontosan megadja az adó helyzetét, ahogy ezt az **2. ábra** is mutatja.

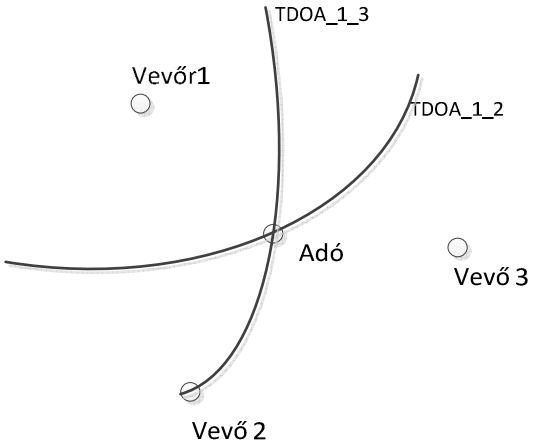


2. ábra Az adó helyének meghatározása három vevő esetén

Hátránya, hogy ismernünk kell az adó által kibocsájtott jel erősségét, illetve a lokalizáció pontosságát jelentősen ronthatja a több utas terjedés illetve az, hogy az átviteli közeg csillapítása eltérő lehet az épület egyes részein.

Az adó pozícióját a *Jelterjedési Időből (ToA, Time of Arrival)* is meghatározhatjuk. Az adó távolságának számítása egy (sebesség * terjedési) idő műveletté redukálódik, ha élünk azzal az egyszerűsítéssel, hogy a rádió jelek a levegőben közel fénysebességgel terjednek. A pontos lokalizációhoz, ahogy az RSS-nél is, itt is legalább három vevőre van szükség. A működéséhez szükséges és egyben a módszer hátránya is, hogy az adás kezdeti idejét nagyon pontosan ismernünk kell, illetve az adó és a vevők óráját szinkronban kell tartani.

Egy másik időmérésen alapuló módszer a *Beérkezési Időkülönbség (TDOA, Time Difference Of Arrival)* alapú lokalizáció. Előnye az előbb ismertetett ToA-hoz képest, hogy itt csak az egyes vevők által vett üzenet beérkezési idejét kell ismernünk, így az adók szinkronizációja nem szükséges ennél a módszernél. A lokalizációhoz itt legalább három vevőre van szükség: az adó helyét az ezek között vett hiperbolikus laterációval határozzuk meg, a **3. ábra** szerint. A két vevő („1” és „2”) által mért idők különbségéből egy olyan hiperbolát kapunk, ami mentén a két időkülönbség abszolút értéke konstans ($|T_1-T_2|$ = konst.). Az egyértelmű helymeghatározáshoz egy harmadik vevőre („3”) is szükség van: ekkor egy újabb méréssel (pl. „1” és „3”) egy második hiperbolát kapunk. Az adó pontos helyét a két hiperbola metszéspontja fogja megadni.



3. ábra A TDOA hiperbolikus helymeghatározási módszere

Ahhoz, hogy elkerüljük azt az esetet, hogy több megoldás is születik a háromszögelés után, egy negyedik vevőre és egy harmadik számításra is szükségünk lesz.

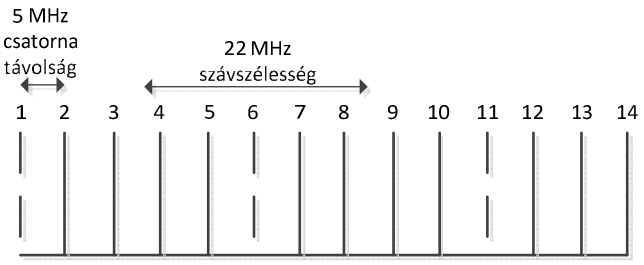
A módszer az előbb felsoroltakhoz képest számos előnnyel bír: a ToA-val ellentétben nem kell az adót a vevővel szinkronizálni, és nem kell tudnunk a csomag küldésének kezdetét. Pontosabb pozicionálást ad az RSS-nél, mivel nem szükséges a kibocsájtott jel erősségét ismernünk és kevésbé érzékeny az átviteli közeg csillapításának változására. A pontosság terén az AoA még egy jó alternatívája lehet, viszont komplex, több antennás vevő kellenek hozzá és a lokalizáció pontosságának növeléséhez a TDOA-t is szükséges implementálni.

2.2 Vezeték nélküli hálózati protokollok beltéri pozicionálásra

Fontosnak tartottuk, hogy a beltéri lokalizációhoz a már meglevő vezeték nélküli hálózati protokollokat használjuk fel, így biztosítva a kompatibilitást a meg levő hálózati eszközökkel. További feltétel volt, hogy a minél pontosabb pozicionálás végett nagy sávszélességen működő szabványokat használjunk fel. Az alábbi fejezetekben két megoldást mutatnánk be, melyek az előbb leírt követelményeknek megfelelnek.

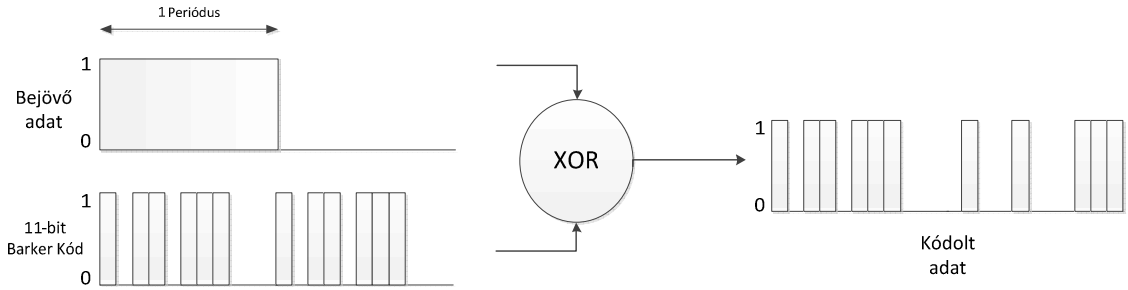
2.2.1 Wi-Fi

Azért, hogy a lokalizáció minél több eszközzel kompatibilisek legyen, a 802.11b protokollt **[3]** választottuk ki a Wi-Fi-s szabványok közül, amit 1999-ben szabványosítottak. 11Mb/s-os adatátviteli sebességre képes és a 2,4 GHz-es tartományban 5 Mhz-es eltolással 14 adatátviteli csatornát hoztak létre azért, hogy egy helyen egy időben egyszerre több adás is folyhasson. Egy csatorna sávszélessége 22 MHz és az interferenciamentes adatátvitelhez az 1, 6 és 11-es csatornát érdemes használni, ahogy ez az **4. ábra** is látszik.



4. ábra Az optimális csatorna kiosztás szaggatott vonallal

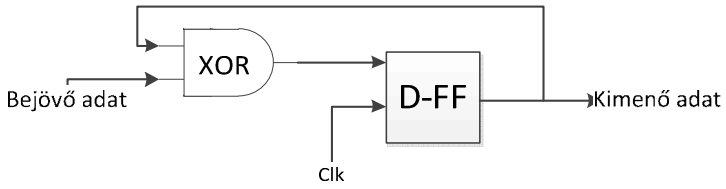
A küldendő adatot először DSSS (Direct-Sequence Spread Spectrum) kódolja, ami egy spektrum kiterjesztési eljárás. Lényege, ahogy az **5. ábra** is mutatja, hogy az egyes biteket XOR-oljuk (kizáró vagy művelet) a egy adott hosszúságú véletlen kóddal (chip kóddal), így nagyobb sávszélességet érünk el, ami a hibatűrésre van jótékony hatással. A kódolt adat bitsebességét chipfrekvenciának nevezzük. A chip kódot úgy kell megválasztani, hogy az önmagával vett autokorrelációja kicsi legyen. A 802.11b egy 11 bites Barker kódot használ ehhez a művelethez. A kódolt adatot ezután differenciális fázis modulálja (DBPSK, Differential Binary Phase-Shift Keying), aminek lényege, hogy a kimeneten akkor lesz fázisváltozás, ha bemeneten 1-es érték jött.



5. ábra A spektrum kiterjesztés (DSSS) működése

2.2.2 Vezeték nélküli protokoll RTLS rendszerekhez

Egy másik elterjedt protokollt a Valós Idejű Lokalizáció Rendszerekhez (RTLS, Real-Time Locating Systems) ajánlják 2,4 GHz-es vezeték nélküli kommunikációra **[4]**. Ellentétben a WiFi-vel itt csak egy csatornánk van, viszont a sávszélessége 60 MHz, aminek kisebb a zavarérzékenysége. Mivel a rendszerhez nincs ütközés elkerülési algoritmus specifikálva, ezért az egyes eszközök a csatornát meghatározott időközönként használják, alapesetben 5 másodpercenként 638 ms-es szórással, így csökkentve az ütközés lehetőségét. Az adatok kódolása és demodulációja is eltér a WiFi-nél látottaktól. A küldendő csomagot először differenciálisan kódolja, vagyis a kódoló kimenetét XOR-olja a bejövő bitekkel, ahogy a **6. ábra** látható.



6. ábra Differenciális kódoló

A DSSS kódolás a 802.11b-ben leírtak szerint történik, viszont a chipkód ebben az esetben 511 bites és a chipfrekvencia 30,521875 MHz.

3 A TDOA vevő

3.1 Követelmények

Célunk egy olyan TDOA vevő tervezése és megvalósítása volt, ami megvalósítja a rádiós jelek vételét és beérkezési idejük nagyon pontos meghatározását.

A megoldásnak az alábbi követelményeknek kell megfelelnie:

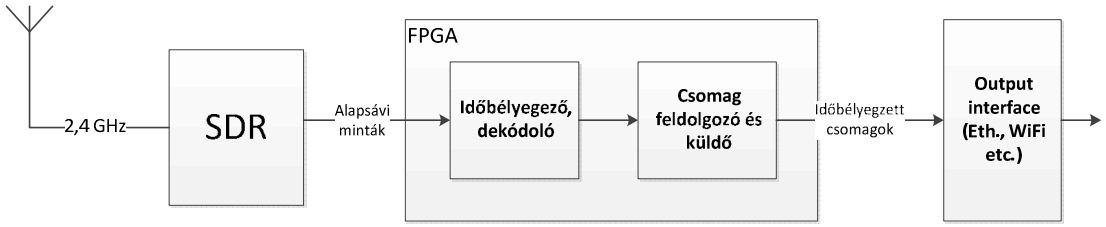
- Nagy sávszélességű rádiós jelek vétele és feldolgozása.
- A fontosabb, leginkább elterjedt vezeték nélküli hálózati protokollok támogatása.
- Nanoszekundum pontosságú időbélyegező rendszer használata a csomagok beérkezési idejének megállapítására.
- Azonosítók és pontos beérkezési idők elküldése egy központi szervernek

A rendszer tervezésekor ügyelnünk kellett arra, hogy a vevő moduláris felépítésű legyen, mivel ez számos előnnyel jár. Egy új protokoll implementálásakor nem kell az egész vevőt újra tervezni, illetve a meg levő modulok továbbfejlesztése is kényelmesebbé válik. További igény tervezői szemmel, hogy a számításigényes műveletek a lehető legkevesebb erőforrás használják fel, lehetővé téve az egyszerűbb alkatrészek felhasználását. A rendszer legfontosabb feltétele, hogy az egyes modulok késleltetését és így a feldolgozási időt konstans szinten tartsuk. Mivel a TDOA elv az időkülönbségekből számolja a pozíciónkat és nem az abszolút időből, ezért ez elengedhetetlen kritérium lokalizációhoz.

Ahogy azt már korábban megismerhettük a TDOA rendszerek alapvető gondolata, hogy a vevő képes a bejövő üzenetek nagyon pontos beérkezési idejének megállapítására. Ennek meghatározását és az időbélyeg elkészítését egy adott szimbólum, a referencia szimbólum felismeréséhez kötjük. Ennek a szimbólumnak egy olyan bitmintát érdemes választani, ami minden csomagban kötött helyen van, és így az azonosítása egyértelmű. A beérkezési időt ennek a szimbólumnak bármely részéhez köthet, de a legjobb, ha a szimbólum közepét választjuk, mert ekkor a legbiztosabb a döntés helyessége. Feladatunk tehát, ennek a referencia szimbólumnak a beérkezési idejének a meghatározása.

3.2 Rendszerterv

Az előbb ismertetett követelményeket szem előtt tartva a **7. ábra** látható felépítést terveztük meg.



7. ábra A teljes TDOA vevő felépítése

A bejövő rádiós jeleket először egy szoftverrádió (SDR) **[1]** dolgozza fel, aminek feladata a 2,4 GHz-es jelekből az alapsávi szimbólumok előállítása. A modul kimenete egy A/D átalakító, ami az alapsávi jelet négyszeresen túl mintavételezi. A mintákat az FPGA-ban dolgozzuk fel, ahol dekódoljuk az egyes üzeneteket, majd ráhelyezzük a nagyon pontos időbélyeget, ami a beérkezési időt jelöli. A csomagokat ezután továbbítjuk a központi feldolgozónak. Az ehhez szükséges interfészről alkalmazásáról még nincs végleges döntés. A vevő működését meghatározó legfontosabb paraméterek a **1. táblázatban** találhatóak. A 125 MHz-es működési és mintavételi frekvenciát egy PLL segítségével állítjuk elő az FPGA-ban a 40 MHz-es kvarc órajeléből.

Chip frekvencia ($1/T$)	30,521875 MHz
A/D átalakító mintavételi frekvencia ($1/T_s$)	125 MHz
A/D átalakító felbontása	12 bit
Kvarc oszcillátor frekvenciája (TCXO)	40 MHz
FGPA rendszer órajel	125 MHz

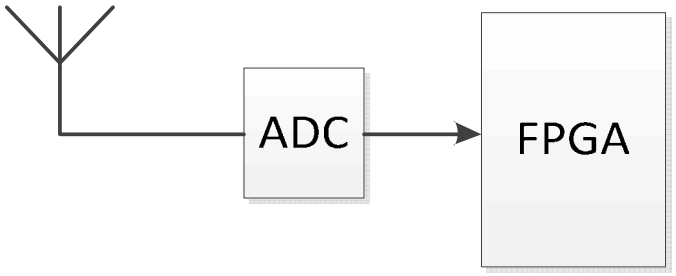
1. táblázat A vevő fő paraméterei

3.3 Felhasznált hardverek

Ebben a fejezetben azokról az eszközökről adnék átfogó képet, amiket felhasználtunk a vevő megvalósításához.

3.3.1 Az SDR vevő

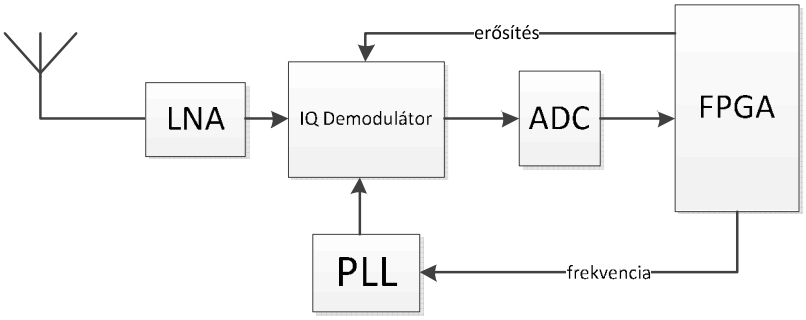
Mivel a vevő működését mi szeretnénk meghatározni, ezért kézen fekvő volt, hogy vevőnek egy szoftverrádiót (SDR, Software-defined Radio) használjunk. Az ilyen vevők előnye, hogy függetlenek a jel frekvenciájától, sávszélességétől, modulációs módoktól, így könnyen módosíthatóak a kívánt működés szerint. Az egyik legelterjedtebb megoldás az USRP (Universal Software Radio Peripheral) **[6]**, lásd **8. ábra**, egy antennából, egy analóg-digitális átalakítóból és valami programozható általános hardverből áll (FPGA, µP), amin az összes többi rádiós komponens van megvalósítva. Az eszköz általában egy USB vagy Ethernet csatlakozón keresztül kapcsolódik a PC-hez.



8. ábra Az ideális Szoftver rádió blokkvázlata

Sajnos ez a vevő 20 MHz-es maximális sávszélességre van korlátozva, ezért nem megfelelő RTLS jelek vételére. Ezért egy olyan általános SDR vevőből indulunk ki, amely 60Mhz-es sávszélességű jelek vételére is képes.

A vevő blokkvázlata a **9. ábra** látható. A bejövő jelek először egy kis zajú előerősítőn (a képen az LNA) vannak átvezetve, majd az IQ demodulátorba kerülnek. Az IQ demodulátort egy fáziszárt hurokkal, PLL-lel állíthatjuk a megfelelő vevő frekvenciára. A demodulátor kimenetén már az alapsávi fázismodulált jeleket kapjuk meg, amiből az analóg-digitális átalakító egy digitális jelet állít elő. Ezt a jelfolyamot kapja meg az FPGA, amin a szimbólumok dekódolása és a vett csomagok időbélyegezése történik. Meg kell még említeni, hogy a szimbólumközi áthallás elkerülése végett egy illesztett szűrőn kell átvezetni a bejövő jelfolyamot, amit tipikusan egy négyzetgyök emelt koszinusz szűrővel valósítanak meg.



9. ábra A szoftverrádió blokkvázlata

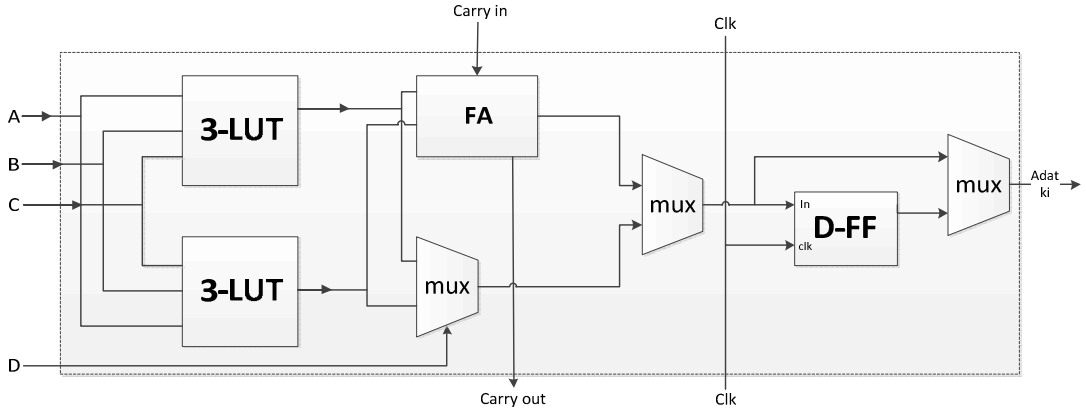
3.3.2 Rövid ismertető az FPGA-ról

Az FPGA (Field-Programmable Gate Array) egy olyan eszköz, ami programozható logikai blokkokból és programozható összeköttetéseket tartalmaz. A logikai blokkok tartalmaznak memória elemeket is, ezek általában flip-flopok, de lehetnek teljes memóriablokkok. A feladattól függően az egyszerű logikai funkcióktól (pl. ÉS kapu) kezdve egész bonyolult kombinációs feladatok is megvalósíthatók vele.

Az egyes blokkok összeköttetését egy hardverleíró nyelvvel valósíthatjuk meg (pl. Verilog, VHDL). Ezt a programot „égetjük” bele aztán a chipbe, létrehozva így az kapcsolatot az egyes blokkok között. A tetszőleges összekötések kialakítása viszont hátrányt is jelenthet a feladat specifikus áramkörökkel szemben (ASIC), azok ugyanis sokszor gyorsabbak és a fogyasztásuk is kevesebb. A folyamatos fejlesztéseknek köszönhetően viszont ezek a hátrányok ma már jelentősen csökkentek, segítve így az elterjedését. Az FPGA legnagyobb előnye, hogy a felprogramozás többször végrehajtható, így lehetőség van a működés módosítására, esetleges hibák javítására, ami fontos szempont volt a feladat elvégzésénél. Ma széles körben elterjed szokás, hogy a fejlesztések egy általános FPGA-n végzik, a sorozatgyártáshoz viszont már egy nem módosítható változatot használnak.

Felépítését tekintve programozható logikai blokkokból **[5]** (CLB, *Configurable Logic Block),* I/O kivezetésekből és programozható összeköttetésekből állnak, amik általában azonos szélességűek. Egy logikai blokk logikai cellákból (Slice-ból) áll, melyek általában egy négy bemenetű *Lookup Table*-t (LUT), egy összegzőt (FA, Full Adder) és egy D flip-flop-ot tartalmaz. A **10. ábra** egy négy bemenetű egy kimenetű cellát ábrázol, melyek két darab három kimenetű LUT alkot. Normál módban a négy jel multiplexelt (baloldali multiplexer) változata lesz a kimenet, Aritmerikai módban viszont az összegző kimenete. Az egyes módok között a középső multiplexerrel tudunk

váltani, a kimenet lehet aszinkron és szinkron, ekkor a D-tárolóból kerül a kimenetre a jel adott órajelre (ezt a jobb oldali multiplexerrel állíthatjuk). Manapság a csúcsmodellekben már hat bemenetű LUT-ot alkalmaznak.



10. ábra Egy logikai cella felépítése

4 A vevőben alkalmazott szinkronizációs megoldás

Alapsávi pulzus amplitúdó modulációnál sávkorlátozott jelek esetén nagyon fontos az optimális mintavételi idő meghatározása. A demoduláció során a döntésünk akkor lesz a legmegbízhatóbb, ha ezt az időt a szimbólum közepére választjuk, ahol az amplitúdója a legnagyobb. **[7]**

A bejövő szimbólumoknál háromféle hiba léphet fel: az IQ demodulátorban a lekeveréshez használt frekvencia nem biztos, hogy megegyezik a modulációs frekvenciával és ez Δf frekvencia hibát okoz. Másrészt a két jel fázisában is lehet eltérés, ami egy $\Delta\varphi$ nagyságú fázis hibát hoz a rendszerbe. Harmadrészt a mintavételi időpont is eltérhet; jelöljük ε-nal ezt a szimbólumidőtől való eltérést, aminek értéke 0 és 1 közötti. Ezeket az ismeretlen paramétereket kell a demoduláció során meghatározni. A differenciális kódolásból adódóan azonban eltekinthetünk a fázishibától, mivel a módszer fázistolásra érzéketlen. A frekvenciahiba számítását is elhagyhatjuk, mert a használt protokollokban meghatározott frekvenciahibák elhanyagolhatóan kicsit. A feladatunk tehát az ε időparaméter meghatározása volt, amihez egy visszacsatolás nélküli, spektrális becslésen alapuló módszert használtunk fel (NDA Timing Parameter Estimation by Spectral Estimation) **[8]**.

Vegyük az alábbi függvényt optimális mintavételezési időt számoló ML becslőnk alapjául:

$$L(\varepsilon) = \sum_{l=-L}^{L} |z(lT + \varepsilon T)|^2$$

Kellően nagy N számú szimbólumra ($N >> L$) a summán belüli rész ciklostacionárius, vagyis egy adott időpillanatban érvényes várhatóértékei periodikusak. Az ilyen függvények Fourier-sorba fejthetőek,

$$|z(lT + \varepsilon T)|^2 = \sum_{n=-\infty}^{\infty} c_n^{(l)} e^{j\frac{2\pi}{T}nT\varepsilon}$$

Ezt visszahelyettesítve $L(\varepsilon)$-be:

$$L(\varepsilon) = \sum_{l=-L}^{L} \sum_{n=-\infty}^{\infty} c_n^{(l)} e^{j\frac{2\pi}{T}nT\varepsilon} = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi n\varepsilon}$$

Bebizonyítható **[8]**, hogy ebben a sorban csak a c_0, c_{-1}, c_1 együtthatóknak van nullától eltérő értéke, így az egyenlet egyszerűsíthető:

$$L(\varepsilon) = c_0 + 2Re\left[c_1 e^{j2\pi\varepsilon}\right]$$

Definíció szerint az ML becslőnk a legnagyobb $\hat{\varepsilon}$-t fogja kiválasztani:

$$L(\hat{\varepsilon}) > L(\varepsilon), \forall\varepsilon$$

Továbbá tudjuk, hogy:

$$\hat{\varepsilon} = arg \max_{\varepsilon}\left(c_0 + 2Re\left[c_1 e^{j2\pi\varepsilon}\right]\right)$$

Bizonyítható **[8]**, hogy c_0 *és* $|c_1|$ függetlenek ε-tól, ezért:

$$\hat{\varepsilon} = -\frac{1}{2\pi} arg c_1$$

Így nem szükséges maximum keresés $\hat{\varepsilon}$ megtalálásához, mivel a Fourier-sor c_1 tag explicit tartalmazza. Az első tagot pedig könnyen felírhatjuk:

$$c_1 = \sum_{l=0}^{N-1} \int_0^1 |z(lT + \varepsilon T)|^2 e^{-j2\pi\varepsilon} d\varepsilon = \sum_l \frac{1}{M_s} \sum_{k=0}^{\lfloor M_s - 1 \rfloor} |z[round(lM_s + k)T_s]|^2 \, e^{-j\frac{2\pi}{M_s}k}$$

, ahol a *round* függvény az argumentum egész számra kerekített értékét adja vissza. Fontos még megemlíteni, hogy a módszer csak akkor működőképes, ha a rendszer impulzusválasza szimmetrikus, illetve a négyzetre emelt sorozat sávszélessége nem sérti a Nyquist kritériumot. A függvény implementálása az exponens periodicitása miatt T/T_s=4 –re jelentősen egyszerűsödik, szorzás nélkül megvalósítható, de sajnos a mi esetünkben ez nincs így. Ezek után még kapcsolatot kell teremteni a vevő mintavételezési órajele (T_s) és a vett szimbólum periódusideje (T) között, amit az alábbi bekezdésekben fogunk ismertetni.

Az adó T időközönként küld egy jelet és ebben az esetben a szimbólum közepe az nT időpontban lesz. A vevő ezt a jelet a jelterjedési idő miatt $nT+ \varepsilon T$ időpontban fogja észlelni, ami az ideális mintavételi időpont lesz, és a belső oszcillátorától függő T_s mintavételi órajele szerint fog mintát venni belőle. A két órajel aránya T/ T_s, amit jelöljünk M_s-el, viszont nem feltétlenül egész szám. Akkor, hogyan tudjuk T_s-el az $nT+ \varepsilon T$ időpontot megközelíteni? Először is az alábbi átalakítást kell megtenni:

$$nT + \varepsilon T = \left(n * \frac{T}{T_s} + \varepsilon * \frac{T}{T_s}\right) T_s = m_n * T_s + \mu_n * T_s$$

A jobb oldalon látható m_n, ami a zárójeles rész egész része megadja, hogy melyik T_s órajelre kell a szimbólumból mintát venni. A μ_n pedig, ami a tört rész megmondja, hogy a minta milyen messze van az optimális mintavételi időponttól.

Az m_n és μ_n számításához szükséges képletek meghatározásánál az alábbi egyenletből indultunk ki:

$$(n+1)T + \varepsilon_n T + \varepsilon_{n-1} T - \varepsilon_{n-1} T$$

, amit úgy kapunk, hogy az *n+1*-dik mintára felírt időből kivonunk és hozzáadunk $\varepsilon_{n-1}T$-et. Ekkor vegyük észre, hogy az $nT + \varepsilon_{n-1}T$-es tag az *n*-edik mintának a mintavételi ideje és írjuk fel rá a maradékos képletet:

$$m_n * T_s + \mu_n * T_s + T + (\varepsilon_n - \varepsilon_{n-1})T = \left[m_n + \mu_n + \frac{T}{T_s}(1 + (\varepsilon_n - \varepsilon_{n-1}))\right] * T_s$$

A kapott képletből m_n és μ_n már minden szimbólumra számítható csak azt kell figyelembe tartani, hogy ε átfordulása ne okozzon hibát a különbségképzésnél. Ennek elkerülése érdekében az alábbi fűrészfog függvényt vezettük be:

$$SAW(x) = x - \lfloor x + 0{,}5 \rfloor$$

Ezek alapján az m_n és μ_n számításához szükséges egyenletek:

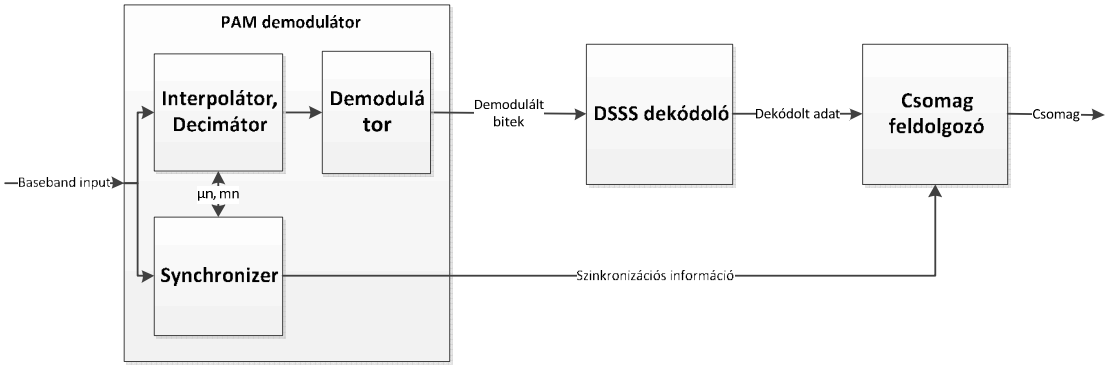
$$m_{n+1} = m_n + \lfloor \mu_n + M_s(1 + SAW(\varepsilon_n - \varepsilon_{n-1})) \rfloor$$

$$\mu_{n+1} = [\mu_n + M_s(1 + SAW(\varepsilon_n - \varepsilon_{n-1}))] \; mod \; 1$$

5 Az FPGA-n belüli rendszer megvalósítása

5.1 Keretrendszer felépítése

A FPGA-n üzemelő TDOA vevőnk felépítése három fő komponensre bontható a **11. ábra** szerinti módon.



11. ábra A TDOA vevő blokkvázlata

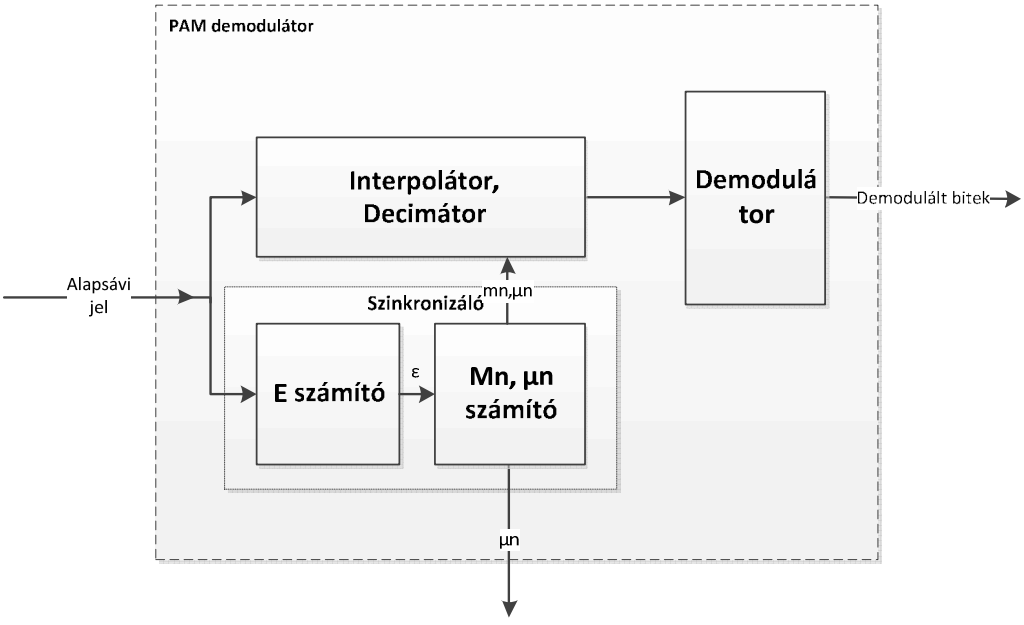
A PAM (Pulse Amplitude Modulation) demodulátorban történik a korábbi fejezetben ismertetett szinkronizációs mechanizmus, amit az interpolálási és decimálási feladatokat is ellátó egység végez el az szinkronizálótól kapott paraméterek alapján. Ezután történik a lekevert jel lineáris amplitúdó-fázis demodulációja, ami RTLS szabvány szerint BPSK (Binary Phase-Shift Keying) modulált jel. Mivel az RTLS jelek differenciális kódolásúak, ezért fontos megemlíteni, hogy nem szükséges koherens vevő alkalmazása. A DSSS dekóder a már említett módon visszaállítja a szórt spektrumú jelet, amihez a szabványban meghatározott ál véletlen kódot használja. RTLS jeleknél ezután történik még a differenciális dekódolás, miután az üzenet a csomag feldolgozóba kerül, ahol megvizsgáljuk a csomag helyességét (CRC ellenőrzéssel). Ekkor történik a referencia szimbólum keresése is, amivel a beérkezési idő határozható meg. A nanoszekundum pontosságú időbélyegek létrehozásának egy fontos eleme, hogy figyelembe vesszük a szinkronizáló által számított töredék időket is.

A vevőben jelenleg csak az RTLS szabványban leírt protokoll szerinti működés van implementálva; a bevezetőben tárgyalt Wi-Fi szabvány szerinti működés megvalósítása később fog történni.

5.2 Demodulátor és szinkronizáló egység

Ebben a modulban történik az alapsávi jel szinkronizálása és fázis demodulálása, aminek felépítése a **12. ábra** látható. Az interpolálási és decimálási feladatokat ellátó egység és a demodulátor egy modulban van megvalósítva, míg a szinkronizáló két alegységből lett létrehozva.

A bemenetre T_s időközönként érkeznek a mintáink, amiket szinkronizáló ε számító modulja és az interpolálási és decimálási feladatokat ellátó egység is megkap. Az *Interpolátor, Decimátor* egység először egy lineáris interpolációt hajt végre, ami függ μ_n értékétől, majd a kapott eredményt m_n időközönként a kimenetére rakja. Fontos tudni, hogy az ε és μ_n, m_n számításokat egyszerre kell elkezdeni, illetve mindig csak a következő decimálási időpontra (m_n értékre) kell az értéküket újraszámolni. A szinkronizáló így az adás bármely szakaszában el tudja végezni a vevő szinkronizációt. Az m_n érték egy globális engedélyező jelnek, pongyolán fogalmazva órajelnek is felfogható, ami a szinkronizáló utáni egységeknek fogja az érvényes adatot jelezni.



12. ábra A PAM demodulátor blokkvázlata

5.2.1 A szimbólum időhöz viszonyított eltolódást számító egység

A modul feladata, hogy a már korábban ismertetett ε paramétert m_n periódusonként újraszámolja az alábbi egyenletekből:

$$c_1 = \sum_l \frac{1}{M_s} \sum_{k=0}^{\lfloor M_s - 1 \rfloor} |z[round(lM_s + k)T_s]|^2 \, e^{-j\frac{2\pi}{M_s}k}$$

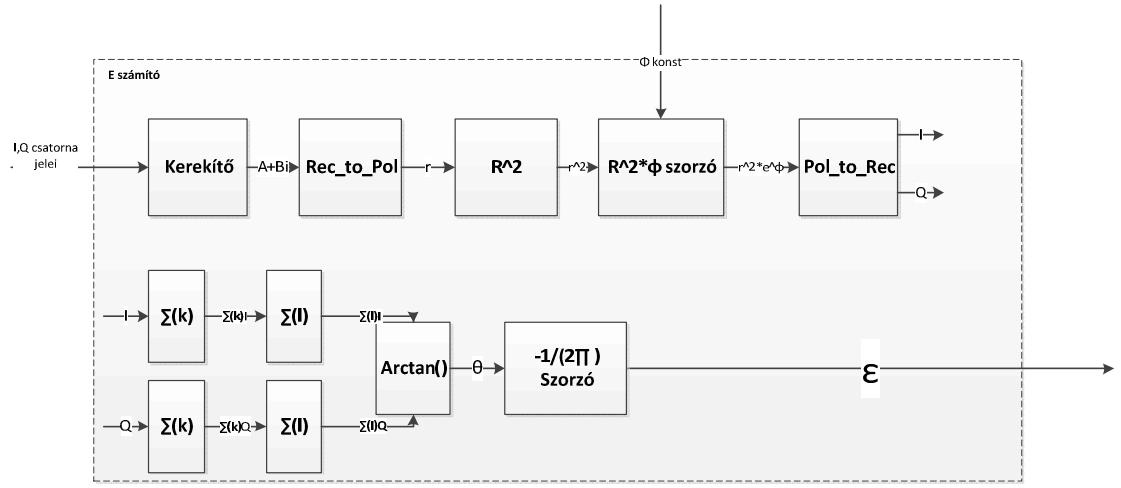
$$\hat{\varepsilon} = -\frac{1}{2\pi} arg c_1$$

Ezen egyenletek viszont nem, vagy csak nehezen és erőforrás pazarlóan implementálhatóak FPGA-ra. Az alábbi bekezdésekben azon szükséges módosításokat ismertetnénk, amivel az egyenletekben megfogalmazott működés optimálisan megvalósítható az FPGA-n.

A c_l egyenletnél vegyük észre, hogy M_s értéke egy konstans, T és T_s aránya. A mi esetünkben ez az érték 4,0954 (125MHz / 30,521875MHz) ebből következik, hogy k maximális értéke 3 lehet. Így az e^x tag négy konstans komplex számként fog megjelenni, melyeket az $1/M_s$ taggal megszorozva letárolhatunk négy konstans értékként. Az $|\ |^2$-en belüli tag azt mondja meg, hogy z komplex számnak - ami az I és Q csatorna mintái - mindig a $[round(lM_s + k)*T_s]$ időpillanatbeli tagját kell venni. Ezt a kalkulációt külön is elvégezhetjük és a bemenetet ez alapján egy engedélyező jellel vezérelhetjük. Az egyszerűsítés után az alábbi egyenletet kapjuk:

$$c_1 = \sum_l \sum_{k=0}^{3} |z(t)|^2 \, e(k)$$

, ahol $z(t)$ az engedélyező jellel vett minta $e(k)$ pedig az előbb ismertetett 4 darab konstans komplex szám. Az l paraméter maximális értéke szabadon megválasztható, a szimulációs eredmények szerint 300 felett érjük el a megfelelő pontosságot; az implementációban l_{max} = 400-at használtunk. Az $\hat{\varepsilon}$ számításánál ilyen egyszerűsítést nem tudunk elvégezni, de ennek a képletnek az implementálása nem körülményes. A megtervezett ε számító egység blokkvázlata a **13. ábrán** látható. A működéshez szükséges konstansokat, az M_s, e^{φ}-t tartalmazó $e(k)$ értékeket az FPGA-ban regiszterekben tároljuk és a működés során állandó értékűek.



13. ábra Az epszilon számító blokkvázlata

A bejövő minták először a *Kerekítő* modulba kerülnek. Mivel M_s értéke nem egész szám, ezért lesznek olyan szimbólumok, melyekből nem 4 mintát fogunk venni, hanem 5-öt. Az egység feladata, hogy ezt az ötödik mintát átugorja, ami akkor történik, ha a $round(lM_s + k)$ függvény 2-vel növekedik. Az implementációban ez úgy van megoldva, hogy külső M_s konstans tört részét iteráljuk minden egyes k ciklusban. Amikor a törtrész összege éppen elhagyja *0,5*-öt, akkor lesz 5 darab mintánk egy szimbólumból. Ilyenkor a kimeneti engedélyező jelet tiltó állásba állítjuk, így a számítási sorban következő modul nem fogja ezt a mintát feldolgozni.

Mivel a c_1 képletben komplex szorzást kell végrehajtani, ezért célszerű polár koordinátás, más néven Euler alakra áttérni. A *Rec_to_Pol* egység megvalósításához a Xilinx ISE fejlesztőkörnyezet **[9]** beépített CORDIC modulját **[10]** használtam, aminek bemenetére az I és Q csatorna mintáit kötve, a kimeneten ezek polár koordinátás alakja jelenik meg. Mivel a számításhoz nem szükséges az átalakítás során kiszámított szög, ezért ezzel a paraméterrel nem foglalkozunk. Az *R^2* és *R^2*φ* modulhoz szintén a fejlesztőkörnyezet által biztosított szorzó implementációt használtam. Az *R^2*-nél mindkét bemenetre a *Rec_to_Pol* egység *r* (vektorhossz) kimenetét kötöttem, így megvalósítva a négyzetre emelést. Az *R^2*φ* modul egyik bemenetére a kapott négyzetszám van kötve, míg a másikra a már tárgyalt *e(k)* konstansok a *k*-dik értéke. Mivel az ezt követő komplex összegzést Descartes alakban egyszerű elvégezni, ezért a kapott értékeket a *Pol_to_Rec* modullal visszaalakítja erre az alakra. Az egység megvalósításához itt is a CORDIC implementációt használtam. Az ezt követő összegző egységek nagyon egyszerűek. A ∑(k) egységek négy mintát összegeznek a c_1 képletnek megfelelően, a ∑(l) összegzők pedig egy ablakozó függvényt valósítanak meg: az utolsó

400 darab ∑(k) értéket adják a kimenetükön. Az így kapott eredményből már számítható az $\hat{\varepsilon}$, amihez először inverz tangens műveletet kell az összegen elvégezni. Ehhez szintén a beépített CORDIC egységet használtam fel. Ezután már csak egy (-1/(2∏))-vel valószorzást kell elvégezni, aminek eredménye lesz az aktuális ε.

Mivel a felhasznált számolási egységek feldolgozási sorban (pipeline) végzik a különféle matematikai műveleteket (pl. szorzás, polár koordináta rendszerbe való átalakítás, stb.), ezért a modul egy 89 órajelnyi kezdeti késleltetéssel rendelkezik, amit figyelembe kell venni a többi hozzákapcsolódó egységnél.

5.2.2 Az optimális mintavételi időt számító egység

Az előző fejezetben megismert modul kimentén megjelenő ε értékekből ez az egység határozza meg μ_n, m_n értékeket, melyek az optimális mintavételi időt fogják meg adni. Itt is a már korábban megismert képletekből kell kiindulni:

$$m_{n+1} = m_n + \lfloor \mu_n + M_s(1 + SAW(\varepsilon_n - \varepsilon_{n-1})) \rfloor$$

$$\mu_{n+1} = [\mu_n + M_s(1 + SAW(\varepsilon_n - \varepsilon_{n-1}))] \; mod \; 1$$

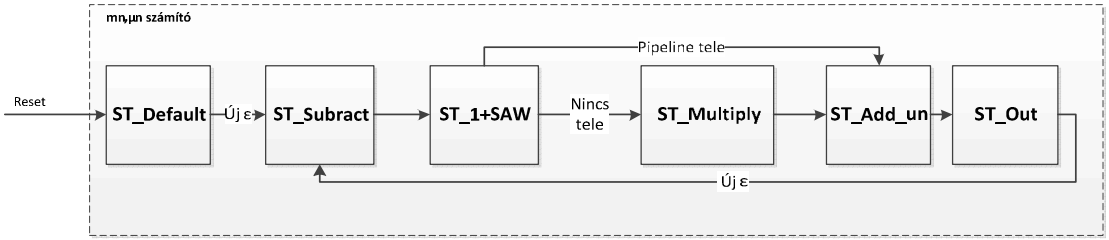
A két egyenletet jobban megfigyelve észrevehető, hogy a lényegi számítás mindkettőben megegyezik: az m_{n+1}-nél az előző m_n értékéhez kell az aktuális számítás egész részét hozzáadni, míg μ_{n+1} az aktuális számolás tört részét kapja eredményül. A két számolás ezért is lett egy modulban létrehozva. Az optimális implementálás érdekében, azonban módosítani kellett az (1 + *SAW (x)*) részszámítás implementálását. Ahogy már korábban ismertettük az ε átfordulásának a különbségképzésben okozott hibáját az alábbi függvénnyel küszöböltük ki:

$$SAW(x) = x - \lfloor x + 0{,}5 \rfloor$$

, ahol *x*-xel az $\varepsilon_n - \varepsilon_{n-1}$ különbséget jelöljük. Vegyük észre, hogy a függvény akkor fog a számított különbségtől eltérő értéket adni, ha az ε_n és ε_{n-1} közti differencia nagyobb, mint *0,5* vagy kisebb, mint *-0,5*. Előbbi esetben az (1 + *SAW (x)*) számítás $(1 + (\varepsilon_n - \varepsilon_{n1})-1)$, míg utóbbi esetben $(1 + (\varepsilon_n - \varepsilon_{n-1})+1)$ lesz. Egyébként pedig $(1 +(\varepsilon_n - \varepsilon_{n-1}))$. Vagyis az implementációban:

$$(1 + SAW(\varepsilon_n - \varepsilon_{n-1})) = \begin{cases} 2 + (\varepsilon_n - \varepsilon_{n-1}), & ha \; (\varepsilon_n - \varepsilon_{n-1}) < -0{,}5 \\ 1 + (\varepsilon_n - \varepsilon_{n-1}), & ha \; -0{,}5 < (\varepsilon_n - \varepsilon_{n-1}) < 0{,}5 \\ (\varepsilon_n - \varepsilon_{n-1}), & ha \; 0{,}5 < (\varepsilon_n - \varepsilon_{n-1}) \end{cases}$$

A kész modul az **alábbi** folyamatábra szerint működik:



14. ábra Az m_n, μ_n számító folyamatábrája

Az egység bekapcsolást követően és minden egyes reset jelre a *ST_Default* állapotban várakozik. Amint az ADC-n (Analog Digital Converter) értelmes adat jön az állapotgép a *ST_Subract*-be ugrik, ahol megtörténik az ($\varepsilon_n - \varepsilon_{n-1}$) különbség képzése. Az ezt követő állapotban, a *ST_1+SAW*-ban a fejezetben ismertetett módszerrel elvégezzük a (1 + *SAW (x)*) műveletet. Ezután következik az M_s konstanssal való szorzás, amit a Xilinx beépített szorzó végez el. Mivel ez a szorzó egység egy pipeline alapú, amit 8 órajel alatt lehet feltölteni, ezért az első számítási ciklusban a *ST_Multiply* állapotba ugrunk, ahol ezt a 8 órajelnyi késleltetést elvégezzük. A későbbi ciklusokban ezt már nem kell megtennünk, mivel a feltöltött feldolgozási sor órajelenként adja majd a szorzások eredményeit a kimenetén. Ekkor egyből a *ST_Add_un* állapotba lépünk, ahol megtörténik a μ_n hozzáadása az eddigi számoláshoz. Végül az *ST_Out* állapotban történik az m_{n+1} és μ_{n+1} meghatározása: a kiszámított érték egészrészét reprezentáló bitjeit hozzáadjuk az m_n-hez így kapva m_{n+1} értékét, a tört rész bitjeit pedig megkapja μ_{n+1} változó.

A modulnál figyelembe kell venni az ε számító kezdeti késleltetését is. A modul indulásától számított 88 órajelig *ST_Out* állapotban az előbb leírtak helyett az m_{n+1} értékét 4-re, a μ_{n+1} értékét 0-ra és az ε_{n-1} változót szintén 0-ra állítjuk. Így a feldolgozás kezdete oly módon tolható el időben, hogy közben a modul szorzójának a kezdeti késleltetését is eliminájuk. Mivel a szinkronizáló az adás bármely pontjától indulva képes a vevő szinkronizációra, ezért ez a művelet nem okoz hibát a demoduláció során.

5.2.3 Interpolátor, decimátor egység

A modul feladata, hogy az optimális mintavételi időpontban levő mintát a fázis demodulátornak továbbítsa. Ehhez a művelethez egységet használ fel, ami interpolálási és decimálási feladatot is ellát, és a működéséhez felhasználja az előbbi fejezetekben megismert μ_n és m_n szinkronizációs paramétereket. Az interpoláció megvalósítása egy

ideális alul áteresztő szűrővel történne, viszont ekkor végtelen sok mintára lenne szükségünk. Helyette egy lineáris interpolátort alkalmaztunk, ami a kimeneti mintákat az alábbi egyenlet szerint állítja elő:

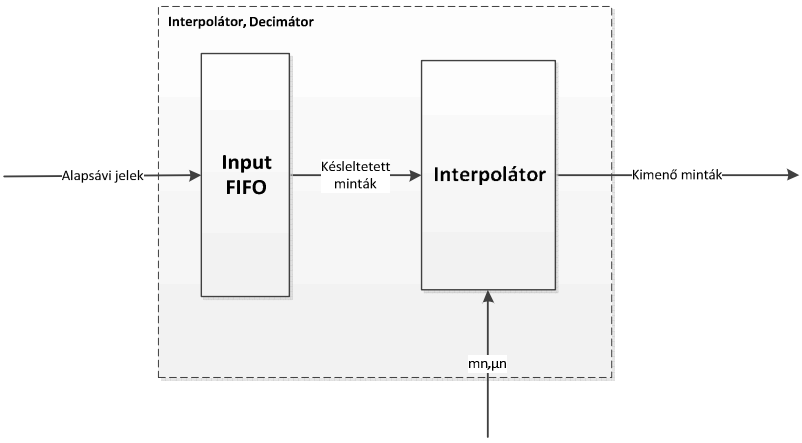
$$y(kT_s) = x(kT_s) - \mu\left[x\left((k+1)T_s\right) - x(kT_s)\right]$$

Mivel ez az egyenlet a jövőbeni mintát is felhasználja az aktuális kimenet elkészítésére, ezért az alábbi formára hoztuk, hogy a működést implementálni tudjuk:

$$y((k-1)T_s) = x((k-1)T_s) - \mu[x(\mathrm{k}T_s) - x((k-1)T_s)]$$

, vagyis az optimális mintavételi időpontbeli mintát a μ_n és m_n változását követő órajelben tudjuk meghatározni. A decimátor megvalósítása ennél jelentősen egyszerűbb: az m_n változását használjuk a decimálási időpontnak.

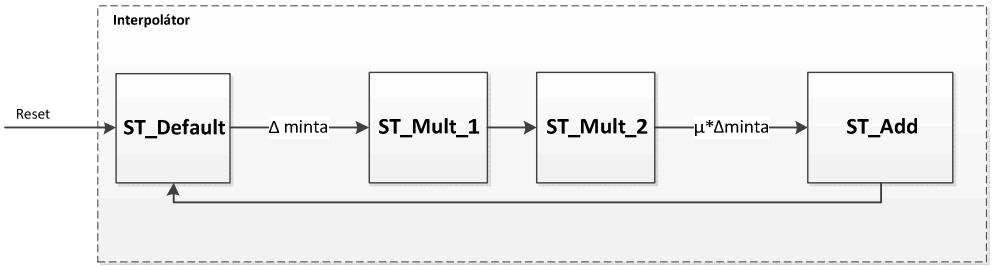
A modul felépítése a **15. ábra** látható. Mivel a megismert szinkronizáló egységnek egy kezdeti, konstans késleltetése van, ezért a modul bemenetére egy FIFO-t terveztünk, amivel kiküszöbölhető a késleltetés miatt előforduló mintavesztés. Az egységet a tervező környezet COREGEN programjával **[11]** állítottuk elő, ami *first-word-fall-through* működést valósít meg, vagyis az első beírt adat egyből megjelenik a sor kimenetén. A FIFO kiolvasásának indítására az első m_n változást használjuk, és az olvasás leállása a sor kiürülését jelző *empty* jel aktív szintjére történik.



15. ábra Az Interpolátor, Decimátor egység blokkvázlata

Az előbb ismertetett interpolációs eljárás az *Interpolátor* egységben van megvalósítva és a szorzás művelethez a tervezői környezetben elérhető implementációt használtuk. A működését a **16. ábra** is látható állapotgéppel valósítottuk meg, aminek a kezdeti állapota a *ST_Default.* Ekkor történik meg az aktuális és az előző minta különbségének a képzése, ami az egy órajellel késleltetett m_n változásra történik meg.

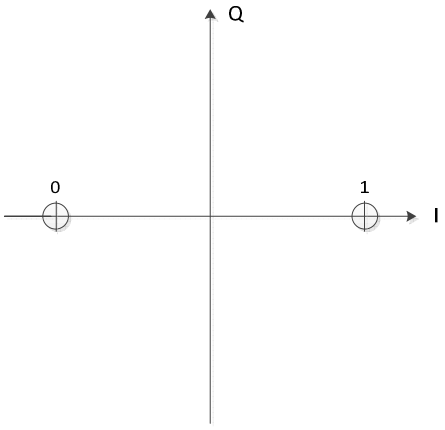
Az ezt követő két állapot, az *ST_Mult_1* és *ST_Mult_2* alatt történik meg a μ_n-nel való szorzás, ami után a *ST_Add* állapotba kerülünk. Ekkor történik az (n-1)-dik mintával való összegzés, aminek eredményeként megkapjuk az interpolált kimeneti értéket. Végül a rendszer menti az aktuális minta értékét a következő ciklushoz, majd visszatérünk alapállapotba.



16. ábra Interpolator folyamatábrája

5.2.4 Fázis Demodulátor

Az egység feladata az optimális mintavételi időpontban érkező minták demodulációja. RTLS protokollal küldött jeleknél ez BPSK (Binary Phase-Shift Keying), vagyis bináris fázis demodulációval történik, aminek konstellációs diagramja az **alábbi** ábrán látható.



17. ábra BPSK konstellációs diagram

Két fázist használ a modulációhoz, melyek 180 fokos fáziskülönbséggel van eltolva egymástól. A kezdeti fázisa a két pontnak nem számít csak a fázis különbségük; a fenti ábrán a 0 fázishoz vettük a kimenet logikai 1 értékét és -180 fokhoz a logikai 0 értéket. Ez a módszer egy nagyon robosztus felépítést hoz létre, aminek nagy előnye, hogy a zajjal szembeni tűrése nagyon nagy (a hibás döntéshez több mint 90 foknyi fázishiba kell).

A modul implementációja külön kihívást jelentett. A bemeneti mintáink egy a+bi algebrai alakú komplex számként foghatóak fel, amiből a jel fázisát (a komplex szám szöge) csak komplex módszerekkel lehet kinyerni, ráadásul a demodulációt m_n időközönként mindenképp végre kell végrehajtani. Végül az alábbi megoldást választottuk: az első minta I és Q értékét eltároljuk, és ez lesz a logikai 1 pontunk. Mivel a kódolása a protokollnak differenciális, ezért ezt minden további nélkül megtehetjük. A később beérkező szimbólumokat pedig úgy demoduláljuk, hogy vesszük aktuális és a referencia pontunk skaláris szorzatát: ha az eredmény pozitív, akkor logikai 1-re döntünk, ha negatív, akkor 0-ra. Ezzel a módszerrel a fázis demoduláció m_n időközönként végrehajtható, ráadásul az implementálásához sem kell túl sok erőforrás (két szorzóval, egy összegzővel és minimális logikával megvalósítható).

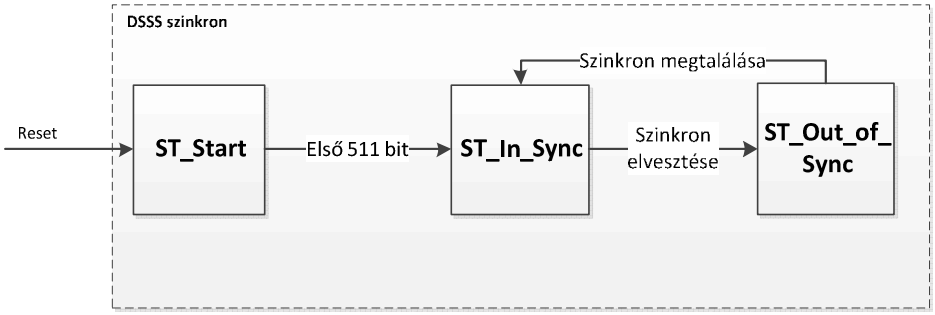
5.3 Dekódoló egység

Ebben a fejezetben ismertetnénk azon dekódolási eljárásokat, melyekkel a demodulált jelekből visszakaphatóak az adat csomag egyes bitjei.

5.3.1 DSSS dekóder

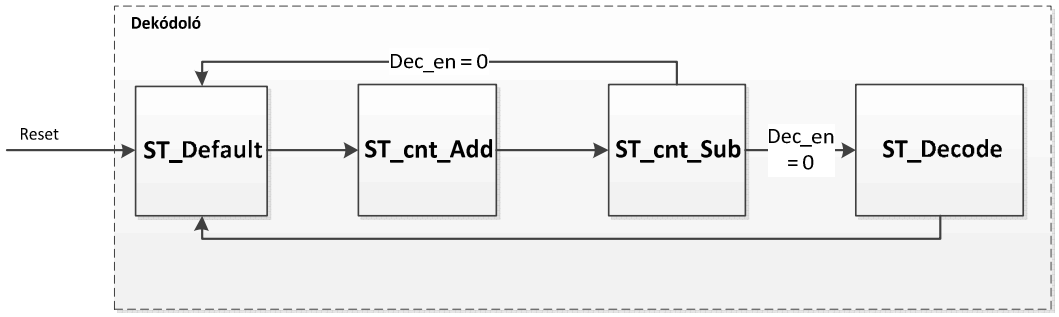
A modul feladata, hogy demodulált, spektrum kiterjesztett jelből kinyerje az üzenet bitjeit. Az elméleti összefoglalóban tárgyalt RTLS protokollra vetítve ez az alábbi feladatot jelenti: az előre meghatározott 511 bites chip kóddal 511 beérkezett bitet kell kizáró vagy-olni és megnézni a kapott értékben az egyesek számát. Ha az egyesek darabszáma 511 vagy egy ahhoz alulról közeli érték, akkor logikai 1-re döntünk, ha 0 vagy ahhoz egy felülről közeli érték, akkor logikai 0-ra. Mivel a chip kód önmagával vett autokorrelációja kicsi, ezért az egyesek száma egy 511 bites sorozatban csak egy helyen fogja e két érték valamelyikét megközelíteni, egyébként a chip kód hosszának felére, 255-re áll be. Ezért azt az állapotot, amikor chip kód hosszonként dekódolni tudunk egy bitet DSSS szinkronnak nevezzük.

A megvalósítást nehezítette, hogy előfordulhat olyan eset, amikor a rádióadás elejét nem tudjuk venni, illetve adás közben bitek vesznek el nem várt zavarok miatt. Ezért a modulban két nagyobb egységet hoztunk létre: egyik a fent leírt dekódolási mechanizmusért felel, míg a másik a DSSS szinkronizációt hajtja végre.



18. ábra A DSSS szinkron folyamatábrája

A DSSS szinkron alapvető feladata, hogy megtalálja az egyes chipsorozatok elejét, rászinkronizáljon a vételükre, illetve jelezze a szinkron elvesztését. Ehhez a **18. ábra** is látható állapotgépet valósítottuk meg. A *ST_Start* állapotban mindig bevárjuk az első 511 bit beérkezését és csak azután történik meg az első döntési ciklus kiértékelése. Ezzel elkerülhető, hogy az esetleges zavarok miatt módosult, hibásan demodulált jelek hibás döntést okozzanak. Sikeresen dekódolás esetén ezután addig tartózkodunk *ST_In_Sync* állapotban, ameddig a dekóder döntése érvényes kimeneti adatot eredményez. Ekkor egy döntési periódus hossza 511 beérkezett bit. Ha a dekóder nem képes dönteni, mert a bemeneti jel hibás, vagyis az egyeseket száma nem esik egyik döntési tartományba sem, akkor a *ST_Out_of_Sync* állapotba jutunk. Ebben az állapotban minden egyes beérkezett bit után megtörténik az egyesek számának kiértékelése mindaddig, amíg a szinkron helyre nem áll és vissza nem térünk az *ST_In_Sync* állapotba.



19. ábra A DSSS dekódoló folyamatábrája

A dekódert egy állapotgép valósítja meg, aminek folyamatábrája a **19. ábra** látható. A dekódoláskor ellentétben a fent leírtaktól, nem várunk be 511 beérkező bitet, hanem menet közben folyamatosan kizáró vagy-oljuk az egyes biteket a chip kód megfelelő értékével és egy *decoder_cnt* nevű számlálóban tároljuk az egyesek számát, amire a végén dönteni fogunk. A rendszer bekapcsolásakor és minden reset-re az

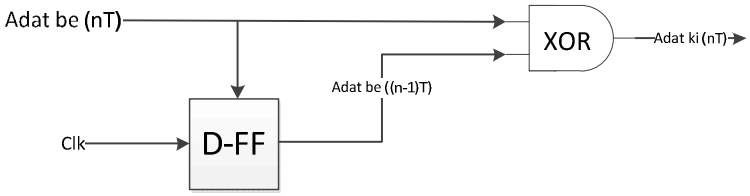
ST_Default állapotba kerülünk. Itt demodulátor engedélyező jelére megtörténik a kimenet olvasása és kizáró vagy-olása a PN kóddal. A chip kódot egy 511 hosszú regiszterben tároljuk, aminek értékén minden egyes ciklusban egyet rotálunk. Ha ez megtörtént, akkor a *ST_cnt_Add* állapotba lépünk, ahol megtörténik a kizáró vagy művelet eredményének letárolása egy shift register-ben, illetve ekkor adjuk hozzá az értékét a *decoder_cnt*-hez. Az ezt követő *ST_cnt_sub* azért szükséges, hogy a számlálónk mindig az utoljára beérkezett 511 bit XOR-jának az eredményét tárolja. Ezt úgy érjük el, hogy a shift register-ben tárolt bitek közül a legfelsőt (ami a legrégebben beérkezett) kivonjuk a számláló értékéből. Ez gyakorlatilag egy 511 széles ablakozó függvénynek feleltethető meg. Ezután, ha szinkronizáló megadta az engedélyező jelet (*Dec_en = 1*) megtörténik az egyesek számának kiértékelése. Itt meg kell említeni, hogy több féle döntési tartományt hoztunk létre. Az első 511 bit beérkezése után és a szinkron elvesztése esetén egy keskenyebb, a két döntési értékhez (511 és 0) közeli döntési sávot alkalmazunk, amit a szinkron megtalálása után kijjebb tolunk a középérték felé, amivel a zavarok ellen tudjuk a rendszert ellenállóbbá tenni. Döntéskor tehát azt az értéket adjuk a kimenetre, amelyik ahhoz a döntési sávhoz van hozzárendelve, amiben az egyesek számát jegyző *decoder_cnt* értéke van. A szinkron elvesztésekor a kimenetet 0-ba állítjuk és a kimenet engedélyező jelét tiltó állásba kapcsoljuk.

Az így bemutatott DSSS dekóder képes a bejövő demodulált jelekből gyorsan és pontosan előállítani a dekódolt adatbiteket.

5.3.2 Differenciális dekóder

Ahogy már az elméleti áttekintésben bemutatásra került az RTLS kommunikációs protokoll differenciális kódolást használ az átküldendő adat továbbításakor. Ebben a fejezetben ezt az egységet mutatjuk be.

Az egység implementációja egyszerű, a **6. ábra** látott működést kell a fordított irányba elvégezni, ahogy ez a **20. ábra** is látható. A DSSS dekóderből érkező aktuális és az eggyel korábbi adaton kell XOR (kizáró vagy) műveletet végrehajtani, ami után megkapjuk a konkrét csomag egyes bitjeit.



20. ábra A Differenciális dekóder blokkvázlata

5.4 Csomagfeldolgozó egység

Miután a dekóder előállította az üzenet egyes bitjeit ennek a modulnak a feladata, hogy detektálja a csomag pontos beérkezési idejét, majd a hibaellenőrzés után továbbítsa egy megfelelő vezetékes vagy vezeték nélküli interfészen keresztül a központi feldolgozóegységnek.

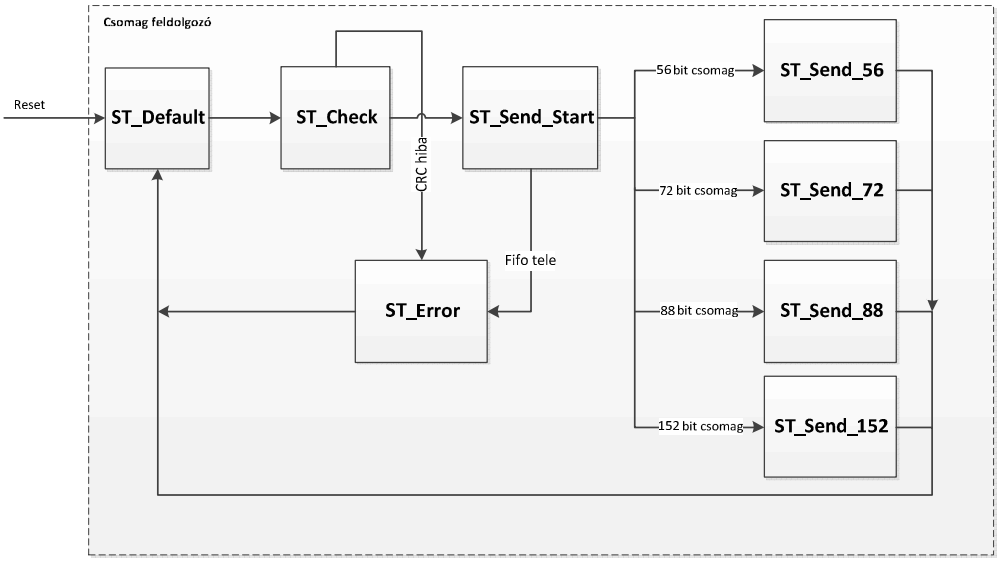
Mielőtt a jelenleg megvalósított RTLS csomagfeldolgozó működését ismertetném, egy rövid összefoglalót adnék a keretszerkezetről. Az RTLS protokollban négy féle hosszúságú csomag létezik: 56, 72, 88, és 152 bites. A keret felépítése az összesben azonos, csupán a hasznos adat (payload) hosszúságában van különbség, ahogy ezt a **21. ábra** is mutatja.

RTLS Packet				
Preamble (1 byte)	RTLS transmitter status (4 bit)	RTLS transmitter ID (4 byte)	Payload (0, 2, 4, 12 byte)	CRC (12 bit)

21. ábra RTLS csomag felépítése

A csomag minden esetben egy 1 bájtos *preamble*-lel kezdődik, aminek értéke binárisan 00000001. Ez az érték azért fontos a mi esetünkben, mivel ezt választottuk a referencia szimbólumunknak, ami a csomag beérkezési idejét jelöli. A másik fontos érték a CRC hibajavító értéke, mivel a modul feladata még a csomag hibamentességének vizsgálata. A CRC polinom hossza 12 bit, és csomagban a *preamble* utáni részre vonatkozóan szolgál információval.

A modulban egy nagy állapotgépben történik a csomag feldolgozása és az időbélyegezés, a **22. ábra** látható módon.



22. ábra A Csomagfeldolgozó folyamatábrája

A csomagok feldolgozását mindig az *ST_Default* alapállapotban kezdjük. Ekkor történik meg a referencia szimbólum észlelése is az alábbi módon: egy nullákkal feltöltött regiszterbe léptetjük jobbról a beérkező biteket. Amint a regiszter alsó négy bitjén előáll a b'0001 érték, ami a preamble alsó négy bitje és az üzenethossz számláló nem haladta meg a preamble hosszát, akkor előállítjuk az időbélyeget. Ezzel a megoldással kiküszöbölhető az az eset, amikor a csomag elejét, konkrétan az első 4 bitet nem tudjuk valamilyen oknál fogva venni. Az időbélyeg az FGPA-n futó óra aktuális értékét jelző 44 bites számból és a szinkronizáló aktuális μ_n értékéből generáljuk.

Az időbélyeg generálással párhuzamosan, ebben az állapotban történik még a csomag beolvasása is egy 152 bites shiftregiszterbe. A beolvasott bitek számát egy számlálóban tároljuk. A beolvasás befejezését két módon figyeljük: egyrészt a számláló értéke megegyezik-e a regiszter nagyságával, másrészt kiestünk-e a DSSS szinkronból. Ez azért használható, mivel az adás közbeni hibát a CRC ellenőrzés úgyis kiszűri, és egyébként, ha a csomag adásának végére értünk, akkor már nem fogunk több hasznos bitet dekódolni az üzenetből.

A csomag vételét követően a *ST_Check* állapotba kerülünk, ahol megvárjuk a CRC ellenőrzés végeztét. Ha a csomagban nem talált hibát az ellenőrző, akkor a *ST_Send_Start* állapotba kerülünk, ellenkező esetben a *ST_Error* állapotba lépünk. *ST_Send_Start* állapotban megvizsgáljuk a kimeneti FIFO-ba belefér-e a csomag: ha van elég hely, akkor megkezdjük a küldést és a csomagméretnek megfelelő állapotba (*ST_Send_56, _72, _88, _152*) lépünk, ahol a csomag végéig ciklikusan a FIFO-ba írjuk

a továbbítandó csomagot. Ha nem fér már el a csomag a FIFO-ban, vagy érvénytelen hosszúságú csomagot vettünk, akkor a *ST_Error*-ba lépünk, ahol az összes átmeneti regisztert alapállapotba állítunk.

A CRC ellenőrző egy külön modulként lett megvalósítva, ami az alábbi generátor függvény szerint működik:

$$x^{12} + x^{11} + x^3 + x^2 + x + 1$$

Az ellenőrzés a csomag beolvasásával párhuzamosan hajtódik végre úgy, hogy a *ST_Check* állapotra már meg tudjuk mondani, hogy a vett csomag hibás vagy nem.

Az egység kimeneti FIFO-ját szintén a tervező környezettel generáltuk, hogy a modul kimenete könnyen illeszthető legyen bármilyen vezetékes (pl. Ethernet) vagy vezeték nélküli interfész vezérlőjéhez.

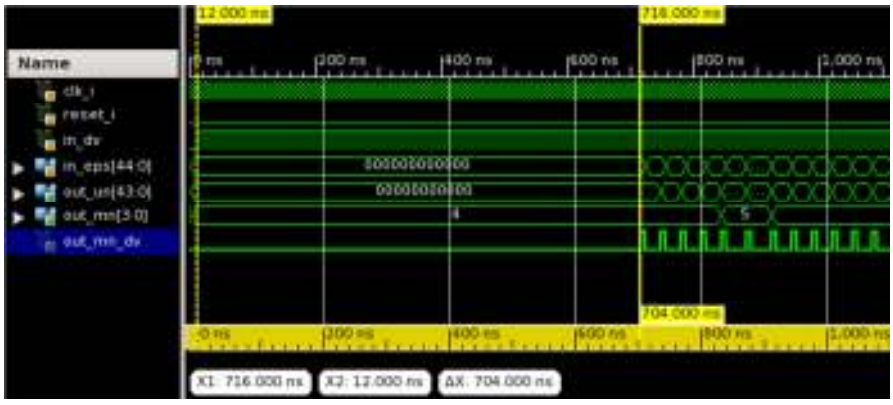
6 Tesztelés és értékelés

Az elkészült rendszer tesztelésekor nem csak arra törekedtünk, hogy bemutassuk a vevő egyes egységei megfelelően működnek, hanem a vevőben előállított időbélyegek és így az alkalmazott szinkronizációs megoldás pontossága is kielégíti a TDOA-hoz szükséges követelményeket.

6.1 Egységek működésének ellenőrzése

A megvalósítást nagyban segíti és felgyorsítja, ha az egyes modulok helyes működéséről, nemcsak az egész folyamat végén - amikor a kész egységeket az FPGA-ra töltjük - bizonyosodunk meg, hanem az egyes részegységek implementálása után ellenőrizzük, hogy azt a működést valósítják meg, amit elterveztünk. Ezt megtehetjük úgyis, hogy az egyes modulokat rátöltjük az FPGA-ra, majd ráadunk valamilyen bemenő jelet és közben figyeljük a kimenetet, de ennél sokkal gyorsabb, egyszerűbb a tervezőkörnyezet szimulátorát alkalmazni, amivel bármikor megállíthatjuk a futási folyamatot. Munkánk során mi is az utóbbival győződtünk meg az egyes egységek helyes működéséről, illetve a rendszer késleltetésének milyenségéről.

A funkcionális teszt egyik legfontosabb része az volt, hogy megvizsgáljuk az egyes modulok késleltetését a feldolgozási sorban. Az elméleti összefoglalóban már ismertettük, hogy a TDOA alapú lokalizáció a beérkezési idő alapján számolja a pozíciót ezért kritikus, hogy az egyes egységek késleltetései, ha nem is nulla, de állandó értéken legyenek.



23. ábra A szinkronizáló késleltetésének mérése

Ehhez a szimulációs fájlban egy teszt adatsort állítottunk elő, amit a rendszer bemenetére adva megvizsgálhattuk az egyes modulok működését és megmérhettük a késleltetéseiket, ahogy ez a **23. ábra** is látható. A mérések eredményei az **2. táblázatban** láthatóak.

Modul név	Késleltetés [órajel]
Epszilon és M_n, μ_n számító	88 órajel
Interpolátor, Decimátor	5 órajel
BPSK demodulátor	4 órajel
DSSS dekódoló	4 órajel
Differenciális dekódoló	1 órajel

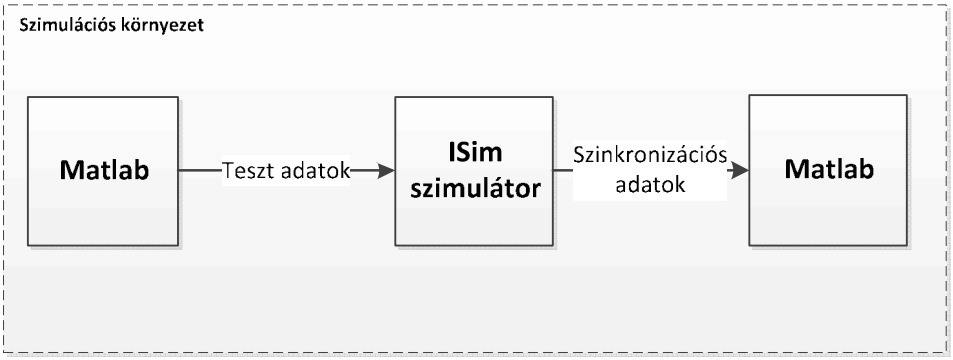
2. táblázat a rendszer bemeneti késleltetése modulokra bontva

A mérés alapján elmondható, hogy az egységek konstans késleltetéssel működnek, vagyis a lokalizáció pontosságát nem befolyásolják.

6.2 Időbélyegezés pontosságának meghatározása

A tesztelés során nem csak azt szerettük volna ellenőrizni, hogy az elkészített modulok helyesen működnek, hanem a szinkronizálás és így az időbélyegezés pontossága megfelel-e a TDOA által felállított nanoszekundumos felbontásnak. Ebben a fejezetben olyan szimulációkat mutatnék be, amivel a pontosság ellenőrizhető.

Mivel az időbélyeg előállítása csak a szinkronizáló egység paramétereitől (μ_n, m_n, ε) függ, ezért a mérés célja ezen értékek pontosságának meghatározása volt. Ehhez először egy külön szimulációs környezetet kellett tervezni. Az elméleti összefoglalóban tárgyalt szinkronizációs megoldást egy MATLAB **[12]** függvényként valósítottuk meg. A függvény képes az FPGA bemenetének megfelelő jelek előállítása oly módon, hogy rádiós jelekbe különböző hibaforrásokat is rakhatunk. Így megvizsgálhatjuk, hogy a szinkronizációs egység mennyire érzékeny a fázis és frekvencia hibára, a mintavételi idő (T_s) pontatlanságára, és hogyan tolerálja a bejövő jelhez adott Gauss zajt.



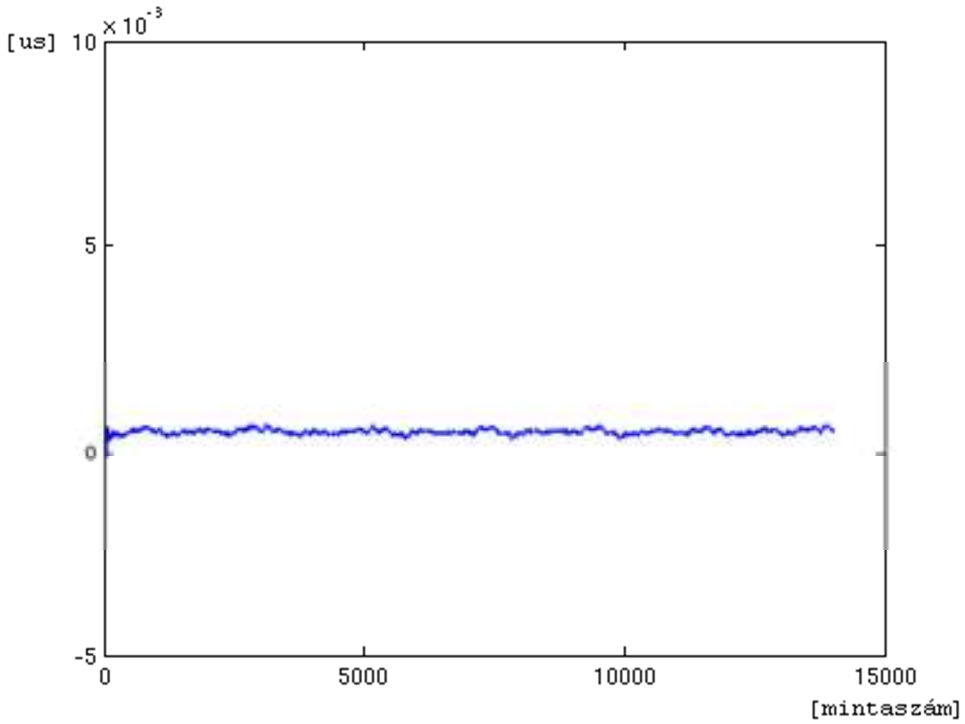
24. ábra A szimulációs környezet felépítése

A pontosság ellenőrzése a **24. ábra** látottak alapján zajlott. A MATLAB-ban létrehozott környezetben előállítottuk a különféle rádiós jeleket, majd ezeket egy szöveges fájlba mentetettük. Az FGPA tervező környezet szimulátora (Xilinx ISim) **[13]** ezeket a fájlokat kapta bemenetként, és a szinkronizációs adatokat szintén szöveges fájlokba mentette ki, amiket aztán visszatöltve a MATLAB-ba elvégezhettük az ellenőrzést. A pontosság méréséhez az alábbi feltételezéssel éltünk: mivel a szimbólum idő (T) adott, ezért ebből minden időpillanatra kiszámolhatjuk az optimális mintavételi időt (a szimbólum idő közepét). Az FPGA által számolt μ_n és m_n értékekből pedig megadható a szimuláció során kalkulált mintavételi időpontok. A szinkronizáció pontosságának és így az időbélyegező mechanizmus felbontásának nagyságát e két időpontvektor különbségének a szórása fogja megadni, mivel a konstans késleltetésre a rendszer érzéketlen.

A mérési eredményeket a **3. táblázat** tartalmazza. Ezek alapján kijelenthető, hogy hibamentes esetben a szinkronizáló és így az időbélyegező rendszer pontossága a kívánt 1 nano szekundumos pontosságon belül van, pontosan 0,1833 ns pontossággal, amit a **25. ábra** is megfigyelhetünk. A demodulátor fázis (Δf) és frekvencia ($\Delta\varphi$) hibája, ahogy ezt már korábban is sejtettük nem befolyásolja a szinkronizációt, a kapott eredmények csak elhanyagolható eltérést mutatnak. Ugyan csak minimális hibát okoz, ha a 20 dB jel-zaj viszonyú Gauss zajt keverünk a szimbólumokhoz.

Teszteset	Pontosság
Hibamentes átvitel	0,1833 ns
Gauss zajjal terhelt átvitel (SNR 20dB)	0,1876 ns
IQ demodulátor 25%-os frekvencia hiba	0,1839 ns
IQ demodulátor 45 fokos fázishiba	0,1848 ns

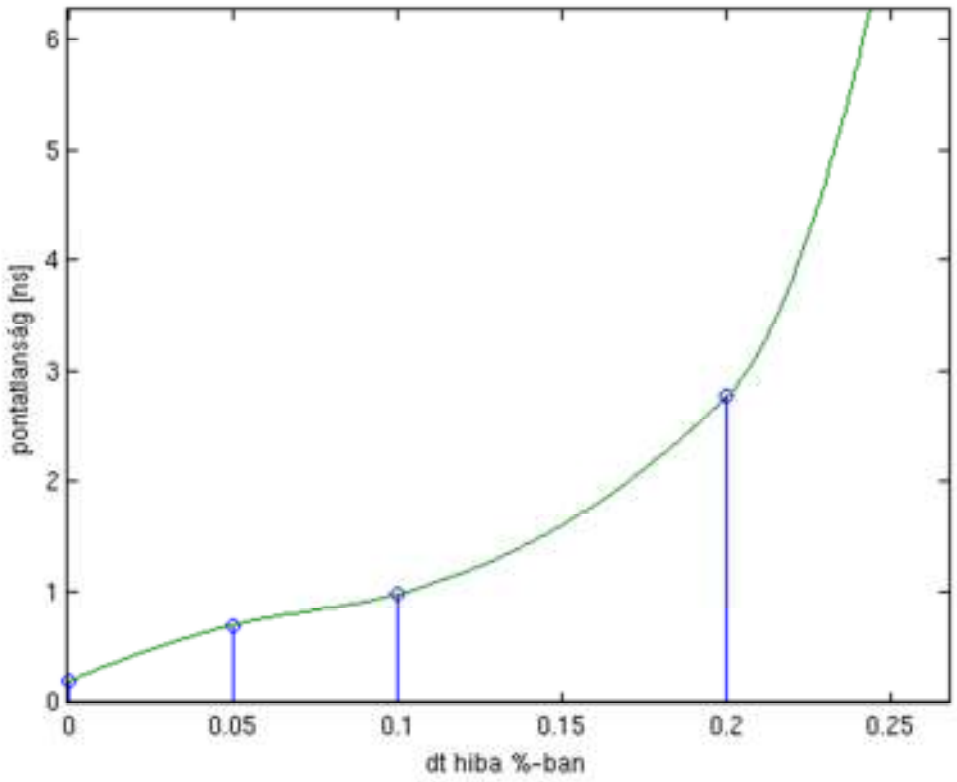
3. táblázat A szinkronizáció pontosságának alakulása



25. ábra Optimális mintavételi időtől valós eltérés hibamentes esetben

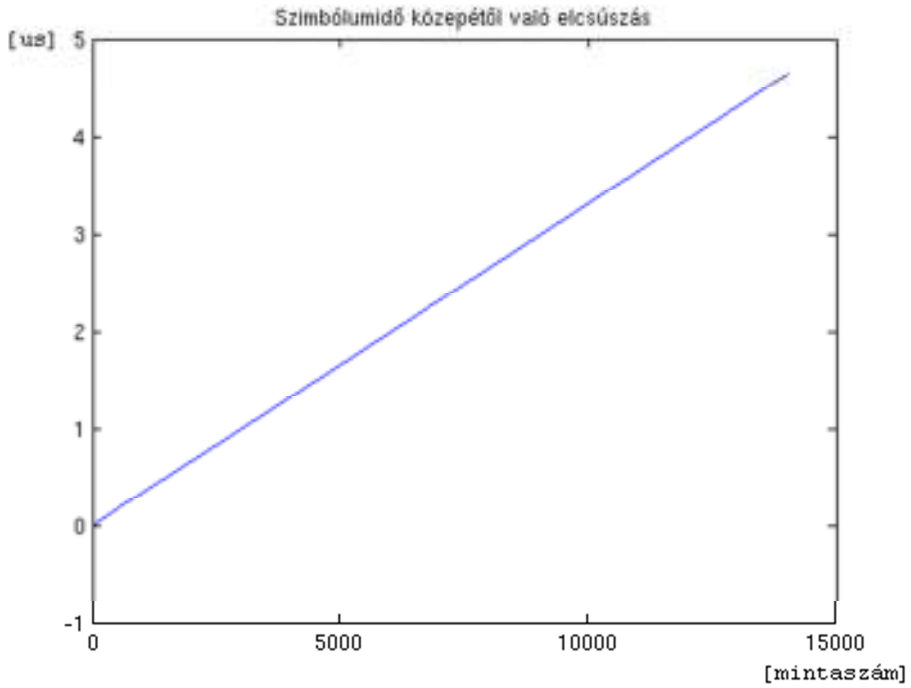
Ezek után megvizsgáltuk még, hogy a rendszer mennyire tolerálja a szimbólum időbe és a mintavételi időbe iktatott hibákat. Mivel a működéshez szükséges konstansokat, az M_s, e^{φ}-t tartalmazó *e(k)* értékeket az FPGA-ban fixen, regiszterekben tároljuk, ezért az időparaméterek hibája nagy hatással van a pontosságra.

Több mérést is végeztünk különféle nagyságú hibákkal, így vizsgálva minként hatnak az időbélyegező pontosságára, az eredményeket a **26. ábra** prezentálja. A mérések alapján elmondható, hogy a mintavételi és szimbólum időbeli hibák az egyes esetekben közel azonos mértékben rontották az időbélyegeket felbontását. Hibamentes esetben a szinkronizáló a már említett 0,1833 ns pontossággal működik, amivel a pontosság az előírt tartományon belül van. 0,05%-os hibánál a rendszer pontatlansága már az előző érték háromszorosa (kb. 0,7 ns), de ezzel még mindig a kívánt felbontáson belül marad. Ezután egy 0,1% hibát adtunk a mintavételi majd a szimbólum időhöz, és ekkor a szórásra 0,975 ns-ot kaptunk, ami még éppen megfelel a méteres pontosságú lokalizáció megvalósítására. 0,2%-os hibánál a viszont már több mint 2,5 nano szekundum fölé megy. Meg kell jegyeznünk, hogy ugyan ez a felbontás már nem felel meg a beérkezési idő meghatározására, de a jelek dekódolására még képes a rendszer.



26. ábra A mintavételi idő hibájának hatása az időbélyeg pontosságára

Van egy bizonyos hiba határ, ami fölött már a jelek dekódolása sem lehetséges. A **27. ábra** jól látható, hogy amikor a rendszer „kiesik” szinkronból. Ekkor a szinkronizáló nem képes már az optimális mintavételi idő meghatározására, folyamatosan késik és így a rendszer nem tudja a beérkező jeleket demodulálni.



27. ábra A mintavételi idő elcsúszása 0,5 %-os hiba esetén

A jövőbeni tervek között szerepel, hogy több utas terjedés és többes adás esetén hogyan viselkedik a rendszer és miként hatnak ezek az esetek a szinkronizálóra és áttételesen az időbélyegek pontosságára.

6.3 Az FPGA-s egység erőforrás felhasználása

A rendszer tervezésekor szem előtt kellett tartani, hogy a megtervezett egységeknek az erőforrás felhasználása a lehető legkisebb legyen. A **28. ábra** alapján elmondható, hogy a rendszer egy Spartan 6-os FPGA esetén **[14]** - ami egy belépő szintű chip a Xilinx palettáján – is csak az erőforrások 45%-át használja LUT-okból, regiszterekből pedig még kevesebbet, csak a teljes mennyiség 21%-át.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	11821	54576	21%
Number of Slice LUTs	12539	27288	45%
Number of fully used LUT-FF pairs	8209	16151	50%
Number of bonded IOBs	67	218	30%
Number of Block RAM/FIFO	12	116	10%
Number of BUFG/BUFGCTRLs	7	16	43%
Number of DSP48A1s	13	58	22%
Number of PLL_ADVs	1	4	25%

28. ábra Az FPGA-s modulok erőforrásigénye

7 Összefoglalás

A Tudományos Diákköri Dolgozatban tehát megterveztünk, megvalósítottunk és teszteltünk egy TDOA vevő FPGA-n belüli feldolgozó egységeit. Az elkészített rendszer amellett, hogy megoldja a bejövő rádiós jelek demodulációját, dekódolását és csomagszintű feldolgozását, képes a bejövő adatok beérkezési idejének nagy pontosságú meghatározására. Ehhez a számításhoz felhasználjuk a spektrum vonalas becslőn alapuló vevőszinkronizáló időzítési paramétereit, így elérve a nano szekundum pontosságú időbélyegezést, ami alapfeltétele a méteres pontosságú, TDOA alapú lokalizációnak. További előnye a vevőnek, hogy az FPGA használata és a moduláris keretrendszer egy olyan környezetet nyújt, amiben az egyes modulok továbbfejlesztése és új funkciók, más elterjedt vezeték nélküli protokollok szerinti működés implementálása egyszerűen megvalósítható.

A tesztelés során bizonyítottuk, hogy a működő rendszer egységei nem rendelkeznek dinamikus késleltetésekkel, melyek károsan befolyásolnák az időbélyegek felbontását. Ezután különféle hibaforrások beiktatásával teszteltük a rendszer pontosságát. Az eredmények alapján elmondható, hogy a vevő az előforduló hibák nagy részére érzéketlen. A szimbólum és mintavételi időbeli hibákra való érzékenység pedig további kutatást igényel, mert e paraméterek pontossága jelentősen függ az adó és vevő a hardverek végleges felépítésétől.

Mindezek alapján kijelenthető, hogy a rendszerünk képes a bejövő jelek dekódolására és a csomagok beérkezési idejének nano szekundum pontosságú meghatározására, így egy megfelelő alapot nyújt egy TDOA alapú helymeghatározási rendszer megvalósítására.

Az elkészült rendszer ugyanakkor felhasználható vezeték nélküli rendszerekben idő szinkronizációs célokra is, mivel pontossága messzemenően kielégíti a csomag alapú idő szinkronizációs protokollok (IEEE 1588v2 – PTP) **[15]** követelményeit is.

Irodalomjegyzék

[1] Peter B. Kenington: *RF and Baseband Techniques for Software Defined Radio*, ISBN 1-58053-793-6, Artech House Inc., 2005.

[2] Cisco, Wi-Fi Location-Based Services 4.1 Design Guide, Location Tracking Approaches, [online elérhető] http://www.cisco.com/en/US/docs/solutions/Enterprise/Mobility/wifich2.html

[3] IEEE 802.11b, [online elérhető], https://en.wikipedia.org/wiki/IEEE_802.11b-1999

[4] Information technology -- Real-time locating systems (RTLS) -- Part 2: 2,4 GHz air interface protocol, ISO/IEC 24730-2:2006, http://www.iso.org/iso/catalogue_detail.htm?csnumber=40508

[5] Field-Programmable Gate Array, [online elérhető] http://en.wikipedia.org/wiki/Field-programmable_gate_array

[6] Universal Software Radio Peripheral, [online elérhető] https://en.wikipedia.org/wiki/Universal_Software_Radio_Peripheral

[7] Umberto Mengali and Aldo N. D'Andrea: *Synchronization Techniques for Digital Receivers*, ISBN 0-306-45725-3, Plenum Press, 1997

[8] Heinrich Meyr, Marc Moeneclaey, Stefan A. Fechtel: *Digital Communication Receivers*, ISBN 0-471-20057-3, John Wiley & sons Inc., 1998

[9] Xilinx ISE fejlesztőkörnyezet, [online elérhető] http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm

[10] Xilinx CORDIC IP, [online elérhető] http://www.xilinx.com/products/intellectual-property/CORDIC.htm

[11] Xilinx COREgen, [online elérhető] http://www.xilinx.com/tools/coregen.htm

[12] MATLAB, [online elérhető] http://www.mathworks.com/products/matlab/

[13] Xilinx Isim szimulátor, [online elérhető] http://www.xilinx.com/tools/isim.htm

[14] Xilinx Spartan 6 FPGA, [online elérhető] http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/index.htm

[15] Precision Time Protocol, IEEE1588v2, [online elérhető] http://www.ieee1588.com/