Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Információfeldolgozás USRP platformon

Tóth Károly

2018

BSc, Beágyazott információs rendszerek ágazat

Konzulens:

Krébesz Tamás, tanársegéd
Bükkfejes András, National Instruments
Méréstechnika és Információs Rendszerek Tanszék

TARTALOMJEGYZÉK

1. Bevezetés, a téma bemutatása 5
1.1. Analóg FM rádiózás 6
1.2. Szoftveres alapú rádiózás 11
1.3. Analóg FM műsorszórás 13
1.4. Eszközök ismertetése 15
2. Alkalmazás megvalósítás 21
2.1. Tervezés 21
2.1.1. Funkcionalitás 22
2.1.2. Front End 23
2.1.3. USRP szoftveres működtetése 25
2.1.4. Back end és adatgyűjtés 28
2.1.5. Adatfeldolgozás 32
2.1.6. Spektrum ábrázolás 36
2.2. Implementáció 38
2.2.1. User Interface 38
2.2.2. Back end és queue 39
2.2.3. Állapotgép 40
2.2.4. Spektrumanalizátor és AGC 43
2.2.5. DDC és Testing DDC 46
2.2.6. Demoduláció, hang kijátszás, tárolás 50
3. Összefoglalás, skálázhatóság, továbbfejlesztés 53
3.1. Megvalósult funkcionalitások 53
3.2. Skálázhatóság, továbbfejlesztés 54
Irodalomjegyzék 57
Függelék 58

HALLGATÓI NYILATKOZAT

Alulírott Tóth Károly, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2018. 12. 16.

.....

Tóth Károly

SZAKDOLGOZAT FELADAT

Tóth Károly (JQ2AP9)

Villamosmérnök hallgató részére

Információfeldolgozás USRP platformon

Napjaink egyik népszerű szoftverrádiós platformja az USRP (Universal Software Radio Peripheral). Az USRP alkalmazása jellemzően a kutatás, oktatás és gyors prototipusfejlesztés területén kiemelkedő, ahol a fő cél egy szoftverdefiniált adatátviteli rendszer vizsgálata, fejlesztése.

A feladat során a hallgatónak alkalmaznia kell ezt a platformot, és a LabVIEW programozási nyelvet használva kiaknázni az USRP lehetőségeit, és tesztelni a határait. A feladat analóg FM műsorszóró adók demodulálása, hangkártyával történő kijátszása és rögzítése párhuzamosan, több rádiócsatornát együttesen kezelve. A felhasználó beállíthatja, hogy egyszerre hány csatornát akar hallgatni és rögzíteni is, mindezt dinamikusan, működés közben változtatva. A hallgató feladata olyan információfeldolgozó alkalmazás elkészítése, amely az adott HW eszközön maximalizálja a rögzíthető adók számát.

A feladat tartalmazza az USRP, a LabVIEW és az analóg modulációs, illetve demodulációs technikák megismerését, a digitális jelfeldolgozás alapjainak elsajátítását, mindezt egy komplex, széleskörűen elterjedt műsorszóró rendszeren keresztül.

A hallgató feladatának a következőkre kell kiterjednie:

- Mutassa be a USRP platformot
- Vizsgálja meg az USRP képességeit, és határozza meg a maximális teljesítőképességét
- Készítsen olyan szoftver alapú alkalmazást az USRP platformra, amely o FM adók jeleit képes feldolgozni és demodulálni,
    - kijátssza azokat hangkártyára és párhuzamos rögzít,
    - képes több rádiócsatornát is kezelni
    - dinamikus felhasználói hozzáférést biztosít

- Vizsgálja meg, hogy az eszköz képességeit maximálisan kihasználta-e és azonosítsa be a szűk keresztmetszeteket

Tanszéki konzulens: Krébesz Tamás István, tanársegéd
Külső konzulens: Bükkfejes András (National Instruments)
Budapest, 2018. október 05.

……..
Dr. Dabóczi Tamás
tanszékvezető
habilitált egyetemi docens

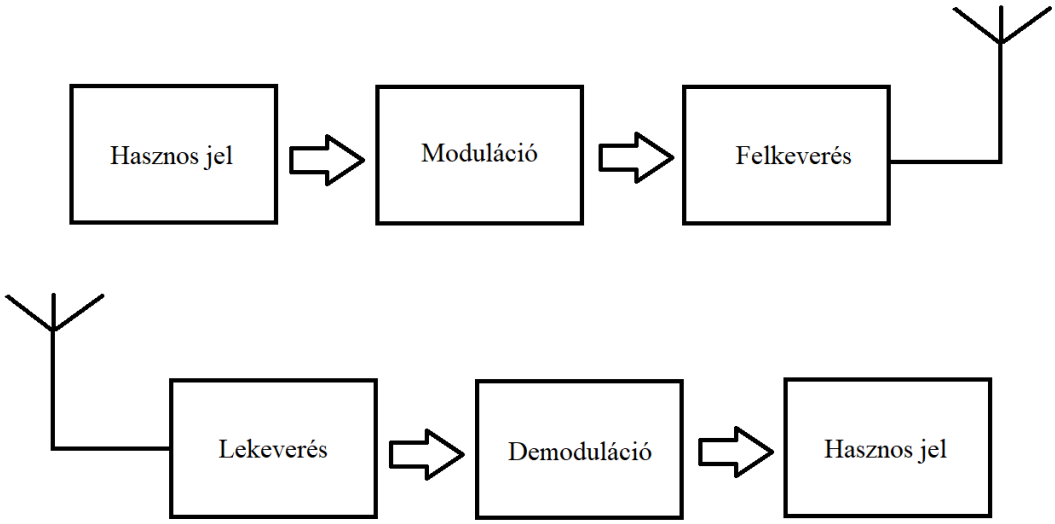
1. Bevezetés, a téma bemutatása

A szakdolgozatom témáját már a Témalabor, illetve az Önálló laboratórium című tárgyak megalapozták. A témalabor során azt a feladatot választottam, hogy MATLAB-on belül leszimuláljak egy BPSK adatátviteli rendszert. Ennek az volt a célja, hogy az adatátviteli rendszereket megismerjem alapszinten, egy könnyen implementálható digitális moduláció segítségével. Az Önálló laboratórium már sokkal konkrétabban foglalkozott a jelenlegi témával, amelyet tulajdonképpen megalapozott. Az Önálló labor keretein belül kerültem a National Instrumentshez, ahol megismerkedtem a LabVIEW-val, illetve az USRP-vel is, amely egy szoftver alapú rádiós eszköz, és szerettem volna ezzel foglalkozni. A feladatom pedig ennek megfelelően az volt, hogy először készítsek egy analóg FM szimulációt a modulációtól kezdve az IQ keverésen és csatornazajon át az IQ lekeverésig és demodulációig, majd, ha már megértettem a teljes folyamatot, akkor használjam az USRP-t, és vessem össze az elméleti eredményeket a gyakorlatival. Kiaknázva a platform nyújtotta lehetőségeket, a megvalósítandó feladat az volt, hogy egyszerre több rádiócsatornát hallgassak, és tároljak a számítógépen. Ehhez az analóg FM rádiószolgáltatás kiváló segítséget nyújt, hiszen a feladat szempontjából nem az adás megvalósítása a releváns, hanem az, hogy az architektúrát megfelelően lehessen tesztelni. Az analóg FM rádiószolgáltatás pedig kellően sok csatornát biztosít ehhez, hogy tényleg csak a feladatra lehessen koncentrálni.

A dolgozat ennek megfelelően 2 fő részből fog állni. Az első részben megadom az elméleti keretét a gyakorlati implementációnak. Bemutatom a szoftver alapú rádiózást, majd ebből következően, hogy miért az analóg FM rádiókkal foglalkoztam ezen belül, és ennek megfelelően be is mutatom a szolgáltatást, és a mögötte álló matematikát. Utána pedig azt, hogy milyen eszközöket használtam a munkám során. A második részben pedig specifikálom, hogy pontosan milyen funkciókat valósítok meg, ennek megfelelően hogyan terveztem az architektúrát, majd bemutatom a konkrét implementációt a tervnek megfelelően lépésről lépésre, figyelembe véve a tervezésnél felmerült problémákat, észrevételeket. A végén pedig adok egy kitekintést arról, hogy hogyan skálázható a projekt, milyen irányokban lehet továbbfejleszteni, majd pedig összefoglalom 1 oldalban a tartalmát.

1.1. Analóg FM rádiózás

Mielőtt bemutatom a szoftver alapú rádiózást, definiálom a rádió, és az ahhoz kapcsolódó fogalmakat. Rádió alatt egy olyan technológiát értünk, ami arra képes, hogy információt továbbítson az egyik helyről a másik helyre, legfőképpen mérési adatokat, audió információkat. Ez elektromágneses hullámok manipulációjával történik meg, ezt hívjuk modulációnak. Ilyenkor az információt analóg esetben a hullám amplitúdójába, frekvenciájába vagy fázisába képezik le. A demoduláció pedig az, amikor a modulált jelet visszaalakítjuk, és kinyerjük belőle a szükséges információt.[1] A dolgozat során FM modulált (FM) jelekkel fogunk foglalkozni, ezért arra fogok kitérni részletesebben.



1. ábra Rádiózás blokkvázlata

Rendszerben nézve a rádiózást, a következő történik. Előállítjuk az úgynevezett hasznos jelet, amit továbbítani akarunk, ha rádióadót építünk, illetve ezt akarjuk kinyerni az adásból, ha rádióvevőt implementálunk. Tehát adás oldalon elindul az adás, ezt tetszőleges modulációval megváltoztatjuk. Eddig a jel alapsávban van, ami azt jelenti, hogy a 0 frekvenciához „közel” dolgozunk. A „közel” moduláció függő, a később bemutatandó FM rádiózásnál ez akár 200 kHz is lehet. Alapsávból áthelyezzük a jelet egy másik frekvenciatartományba, ezáltal az kisugározhatóvá válik. Vivőfrekvenciának hívjuk azt a frek-

[1] https://en.wikipedia.org/wiki/Radio

venciát, amire ráültetjük az alapsávi jelünket. Az adatátviteli csatornában haladva a jelünket különböző hatások érik. Majd a vételi oldalon először lekeverjük a vivőfrekvenciáról alapsávba a jelünket, majd utána demodulálva visszakapjuk a hasznos jelet.

Ezt a blokkvázlatot kövessük végig FM jelre specifikusan, matematikailag megalapozva. Bár az alkalmazás során nem fogunk modulációval és felkeveréssel foglalkozni, viszont a lekeverés és demoduláció megértéséhez, amivel viszont hangsúlyosan fogunk foglalkozni, szükséges. Először is kezdjük az alapsávi modulációval. A modulátor bemenetére érkezik egy tetszőleges jel, amit át szeretnénk küldeni a rádió segítségével. Ezt nevezzük m(t)-nek, mint moduláló jelnek.

Moduláció esetén a jelnek 3 összetevőjét lehet változtatni. Ezt az alábbi összefüggés szemlélteti:

$$A(t)*\cos(2\pi*f(t)*t + \theta(t)).$$

Amplitúdómodulációról beszélünk, ha a moduláló jel (modulate signal) az A(t) amplitúdót változtatja. Hasonlóképpen f(t) változtatása esetén frekvenciamodulációról, θ(t) változtatása esetén pedig fázismodulációról van szó. Azonban az utóbbi kettő nem független egymástól, mert a frekvencia a fázisnak a deriváltja, képlettel,

$$\frac{d\theta}{dt} = f$$

Azért írhatjuk fel ebben a formában a modulációt, mert olyan jelekkel dolgozunk, amelyeknek létezik Fourier transzformáltja.[2]

Az általunk igénybe vett analóg FM rádiószolgáltatás úgy működik, hogy fázismodulációba viszi át a jelet, így először integrálja azt.[3] Integrálás után pedig jöhet a konkrét fázismoduláció. Az integrálás utáni jelet nevezzük M(t)-nek. Ahhoz, hogy meghatározzuk pontosan mekkora mértékű legyen a moduláció, M(t)-t meg kell szorozni 2π-vel (körfrekvencia), majd pedig k_p-val, ami a frekvenciaérzékenységet jelenti, és a frekvencialöket (frequency deviation) és a moduláló frekvencia (modulate frequency) hányadosa. A frekvencialöket azt jelenti, hogy egységnyi moduláló frekvenciához képest mekkora fáziskilengést okozhat maximum a moduláló jel. Tehát az alábbi jelet fogom kapni szorzás után:

$$\varphi = 2\pi k_p * M(t)$$

[2] Forrás: Géher Károly, *Hiradástechnika*
[3] Forrás: Géher Károly, *Hiradástechnika*

Ezután vegyünk egy sávátereszto, rádiófrekvenciás jelet. Ezt bontsuk fel két részre. Egy lassan változó komplex burkolóra, amit x(t)-vel jelölünk, és egy exponenciális vivőfrekvenciás jelre, amit jelöljünk $\tilde{X}(t)$-vel. A komplex burkolót általánosan az alábbi alakban írjuk le[4]:

$$x(t) = x_I(t) + j * x_Q(t)$$

Itt az I In-Phase-t jelöl, ami azt jelenti, hogy fázisban van, míg a Q Quadrature-Phase-t jelöl, ami azt jelenti, hogy a fázisban lévő komponenshez képest kvadratúrában, azaz 90◦ fázishelyzet különbségben vannak. A szinusz és a koszinusz jelek IQ jelek, és ezt fel fogjuk használni.

A nagyfrekvenciás vivő jelet pedig az alábbi szerint írjuk le:

$$\tilde{X}(t) = \exp(j\omega_c * t)$$

Használva az Euler-képletet:

$$\exp(j\omega_c * t) = \cos(\omega_c * t) + j * \sin(\omega_c * t)$$

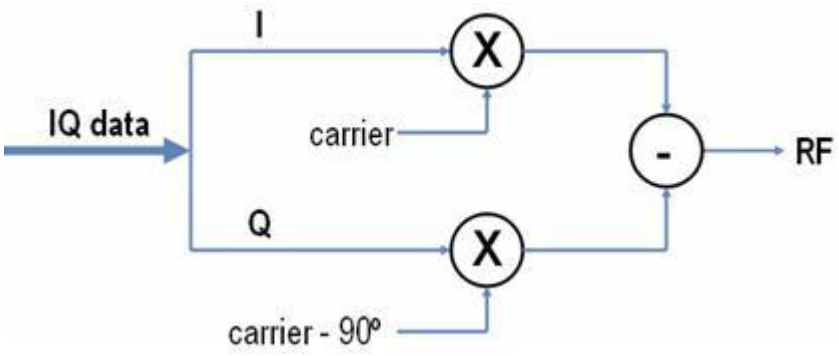
A komplex burkolók számos szolgáltatást kínálnak, ami miatt célszerű a használatuk:

- Minden rádiófrekvenciás információ elérhető alapsávban is. Mivel minden RF sávátereszto típusú jel felbontható egy lassan változó komplex burkolóra és egy nagyfrekvenciás vivőre, ezért a komplex burkoló hordozza az összes hasznos információt, amely így az RF jel alapsávi ekvivalense
- Mivel ez egy reprezentáció, ezért ebből az alapsávi rádiófrekvenciás jel hiba nélkül visszaállítható
- Mivel alapsávi jel, ezért biztosítja, hogy a mintavételi frekvencia minimális lesz a digitális jelfeldolgozás során.[5]

Következő lépésként meg kell valósítani a felkeverést, ami annyiból áll, hogy frekvenciatartományban gondolkodva eltoljuk a jelet egy tetszőleges frekvenciára, amiről a későbbiekben sugározni fogjuk az adást. Pontosabban a modulált jelünket ráültetjük egy vivő jelre (carrier signal), aminek a legmeghatározóbb paramétere a vivő frekvencia (carrier frequency) lesz. Ehhez IQ keverést használunk.

[4] Forrás: Géza Kolumbán, Tamás István Krébesz, Francis C.M. Lau, *Theory and Application of Software Defined Electronics,*

[5] Forrás: Géza Kolumbán, Tamás István Krébesz, Francis C.M. Lau, *Theory and Application of Software Defined Electronics,*



2. ábra IQ keverés blokkvázlata[6]

Az IQ keveréshez szinusz és koszinusz jeleket használunk. Ez az Euler-képletből következik, aminek a segítségével az exponenciális jelünket felbontottuk szinuszra és koszinuszra. Az IQ jeleket úgynevezett Lokál Oszcillátor (Local Oscillator) segítségével állítjuk elő. Ez lesz a vivő jelünk. A komplex burkolós jelünk az alábbi alakban írható fel:

$$\exp(j\varphi) = \cos(\varphi) + j * \sin(\varphi)$$

A kisugárzott RF jelet matematikai levezetés alapján kapjuk meg. A blokkvázlatot követve:

$$\mathrm{I} = \cos(\varphi)$$

$$Q = \sin(\varphi)$$

$$carrier = \sin(\omega_c * t)$$

$$carrier - 90\ ^\circ = \cos(\omega_c * t)$$

Majd utána összeszorozzuk a megfelelő tagokat a blokkvázlat szerint, aminek az eredménye:

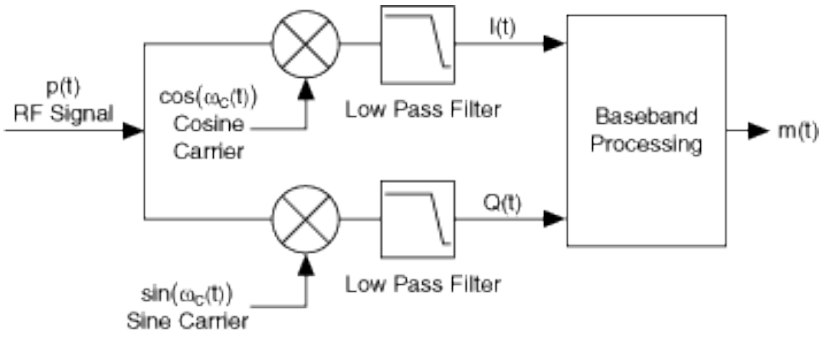
$$\cos(\varphi) * \sin(2\pi * f_c * t) - \cos(2\pi * f_c * t) * \sin(\varphi)$$

Azt fogjuk kapni a szinuszra vonatkozó addíciós tétel segítségével, hogy:

$$\cos(\varphi) * \sin(2\pi * f_c * t) - \cos(2\pi * f_c * t) * \sin(\varphi) = \sin(\omega_c * t - \varphi)$$

Ez a jel fog kisugározódni, ami valamilyen csatornába kerül, majd onnan át a vételi oldalra. Ezután jöhet a lekeverés alapsávba.

[6] Forrás: http://www.ni.com/tutorial/4805/en/



3. ábra IQ lekeverés és szűrés blokkvázlata[7]

Ahhoz, hogy visszakapjuk a moduláló jelet, először is le kell szedni vivőfrekvenciáról, ezáltal visszahozva azt alapsávba. Ezt úgy tudjuk megtenni, hogy a beérkező FM jelet megszorozzuk a korábban felhasznált LO-val, majd utána aluláteresztő szűrő segítségével kiszűrjük a nem kívánt nagyfrekvenciás komponenseket. Hogy miért van erre szükség, arra matematikai levezetés alapján rájövünk:

$$\sin(\omega_c * t - \varphi) * \sin(\omega_c * t) = \frac{\sin(2\omega_c * t - \varphi)}{2} - \frac{\sin(\varphi)}{2}$$

$$\sin(\omega_c * t - \varphi) * \cos(\omega_c * t) = \frac{\cos(2\omega_c * t - \varphi)}{2} + \frac{\cos(\varphi)}{2}$$

Mint látható, visszakaptuk a jelünket alapsávban feleakkora amplitúdóval, valamint feleakkora amplitúdóval megjelent egy kétszeres vivőfrekvenciás komponens is, amire viszont nincs szükség, csak zavarja a rendszerünket. Éppen ezért ezt a jelet ki kell szűrnünk valahogyan, erre pedig egy aluláteresztő struktúra megfelelő lesz. Ennek a megvalósítását nem tekintettem a feladatkör szerves részének, ezért a megvalósítását a LabVIEW beépített könyvtári függvényeire bíztam a későbbiek során, melyek között szerepelnek szűrőtervező függvények is.

A szűrés után a demoduláció maradt hátra, mint ahogyan azt 1. ábra tartalmazza. Ehhez arra van szükség, hogy az IQ jelünket visszaalakítjuk a komplex burkoló alakjára, aminek vesszük az exponenciális alakját, ahol a φ tartalmazza a hasznos információt, amelynek alakja az alábbi volt:

$$\varphi = 2\pi k_p * M(t)$$

Ennek megfelelően deriválni kell a φ-t, aminek az eredménye:

[7] Forrás: http://zone.ni.com/reference/en-XX/help/372058U-01/nirfsa/iq_modulation/

$$\varphi = 2\pi k_p * m(t)$$

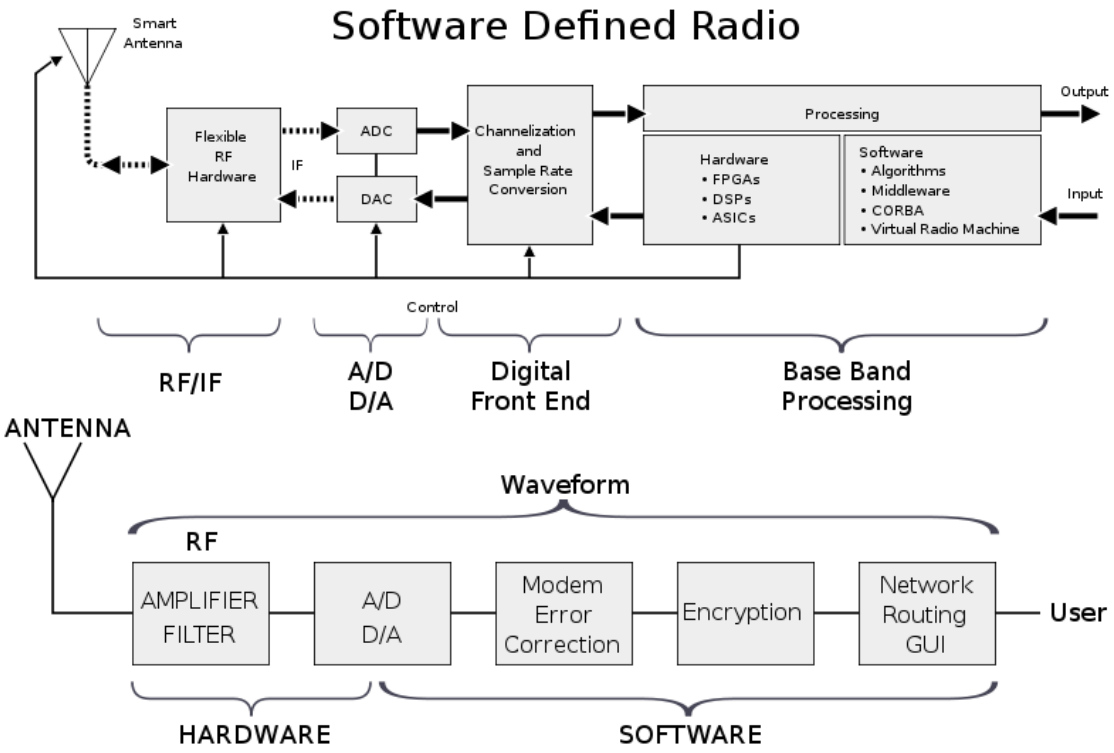
hiszen a $2\pi k_p$ konstans, az M(t) pedig a moduláló jelnek, m(t)-nek az integrálja. Már csak normalizálni kell φ-t a $2\pi k_p$-vel, és visszakapjuk az m(t)-t.

A fentieknek megfelelően az analóg rádiózás egy komplex folyamat, azonban van olyan része, ami általánosabb, és van, amely kevésbé. Az általános része a felkeverés és lekeverés, míg a moduláció és demoduláció nem. Éppen ezért nem tűnik célszerűnek együtt kezelni a kettőt. Erre kínál megoldást a szoftver alapú rádiózás.

1.2. Szoftveres alapú rádiózás

Az analóg FM rádiózás, és úgy általánosságban a rádiózás ismertetése után rátérhetünk a szoftver alapú rádiózásra. Angolul Software-defined radió (SDR), egy olyan technológia, ami nem csak hardveresen közelíti meg a rádiós kommunikáció témakörét, hanem erőteljesen szoftver alapon is. Régebben a rádiók teljesen hardveres alapon működtek, ami azt jelentette, hogy rádióadó esetén elvégezte a modulációt, majd pedig felkeverés után kisugározta a jelet az antennán keresztül, vagy rádióvevő esetén, ami lekeverte a jelet, majd pedig demodulálta az. A hardveres megoldás egy adott problémára nyújt egy konkrét megoldást. Ez nem igazán skálázható, illetve fejleszthető tovább, hiszen hardverelemek cserélésével lehet javítani, gyorsítani stb., de új funkciókat nehezen vagy egyáltalán nem lehet megvalósítani vele.

Az SDR erre kínál megoldást. Számítógép és célhardver segítségével egy olyan rádiós rendszer kialakítása a cél, ami több funkciót tud ellátni, szükség esetén átprogramozható, adaptálható a megváltozott környezethez.



4. ábra Szoftveres alapú rádiózás blokkvázlata

A 4. ábra látható, hogy hogyan épül fel az SDR architektúrális, illetve jel szinten. Az antennától a kimenet felé irány jelenti azt, hogy vevőként működik, és először lekeveri a jelet, majd utána alapsávban feldolgozza. A bemenettől az antennáig irány pedig azt, hogy adóként működik, először modulálja a jelet, majd alapsávról felkeveri és kisugározza az adatátviteli csatornába. Vagyis az SDR mind adó, mind vevő szempontjából implementálható. A később bemutatásra kerülő USRP egy adó-vevő egység, mert egy egységen belül képes adás és vétel megvalósítására. Alapvetően három fő részre különíthető el. Van benne egy ADC, illetve DAC az analóg-digitális konverzió érdekében. Adás esetén a számítógép digitális adatait kell analóg hullámformába átkonvertálni, míg vétel esetén fordítva. Adás esetén következik a felkeverés, vétel esetén pedig a lekeverés (a használattól függ).

Ezek után jön az a rész, ami miatt az SDR teret nyert magának. Az alapsávi jelfeldolgozást már hardver és szoftver együttműködésével végezzük. Az adatokat valamilyen hardverrel gyűjtjük, és a feldolgozását pedig erre implementált szoftverre bízzuk. Ha igazán nagy számítási igényre van szükségünk, akkor ez az egész implementálható nagy teljesítményű mikrokontrollereken, FPGA-n vagy DSP-ken. De az egész megvalósítható számítógépen is. És a nagy előnye ebben mutatkozik meg. Nem egy egész nagy hardvereszköz felel a

teljes folyamatért, aminél a keverésekre és az alapsávi jelfeldolgozásra kell együttesen optimalizálni. Ezt azonban hardveresen véges szintig lehet csak végezni. Ehelyett a kettő szét van választva, és az adatfeldolgozásban a szoftvernek nincsen szüksége nagy mintavételi frekvenciára. Hardver esetén azért van rá szükség, hogy teljesüljön a Nyquist – Shannon mintavételi tétel. Képlet formájában:

$$B \leq \frac{f_s}{2}$$

ahol B a sávszélességet, f_s a mintavételi frekvenciát jelöli. Ha a felkevert jelet kellene mintavételezni, akkor azt azért nehéz megoldani, mert a vivőfrekvencia miatt az alapsávhoz képest jelentősen nagyobb mintavételi frekvenciára van szükség. Az analóg FM rádiók esetében, amiről a dolgozatban szó lesz, a worst case esetet nézve, ami 108 MHz, 216 MHz a minimális mintavételi frekvencia. Ezzel szemben alapsávban maximum pár MHz mintavételi frekvencia elegendő. És emellett szoftveresen tudunk olyan problémákat megoldani, mint a demoduláció, amelynek köszönhetően igen gyorsan és jó minőségben lehet megvalósítani a feldolgozást.

A 4. ábra alsó részén pedig az látható, hogy magával a jellel pontosan mi történik a folyamat során. Vétel esetén a jelet szűrni kell a legtöbb alkalmazáshoz, mert a lekeverés után megjelennek nagyfrekvenciás komponensek, illetve az alapsávi jelre is kerülhet zaj. Utána A/D átalakító segítségével digitalizáljuk, hogy a számítógép fel tudja dolgozni. Majd jön a hálózati korrekció, hiszen az SDR és a számítógép valamilyen módon kommunikál egymással. Az esetek jelentős részében Ethernet kapcsolaton keresztül. Végül a felhasználó alakítja tetszés szerint, pl. FM demodulálja. Adás esetén pedig ugyanez, csak megfordul az irány pl. FM modulálja, majd pedig a végén szűrés helyett erősít rajta, hogy megfelelő teljesítménnyel történjen a kisugárzás.

Az SDR tehát egy univerzálisan használható rádióadó vagy -vevő egység, ahol a szoftveres rész jól skálázhatóvá és bővíthetővé, alakíthatóvá teszi a rendszert. Ezen a téren a National Instruments az élen jár, ezért választottam azt, hogy náluk végezzem az Önálló laboratórium tárgyamat, és erre építve, írjam meg a szakdolgozatomat is.

1.3. Analóg FM műsorszórás

A projekt során olyan területet választottam, ami széleskörűen használt, és a párhuzamos rögzítéshez szolgáltat számos rádióadást. Így jutottam el az analóg FM műsorszóráshoz,

ami annyit jelent, hogy a kereskedelmi rádiócsatornák vételével kívánom tesztelni az architektúrámat. A demoduláció pedig ehhez a szolgáltatáshoz könnyen megvalósítható.

Magyarországon jelenleg analóg FM rádióműsorszórás zajlik, a digitális átállás (DAB+) még csak gyerekcipőben jár, ezért analóg FM tesztelésre még sokáig használható. Itthon a rádiócsatornák a 87,5 MHz – 108 MHz sávban helyezkednek el. Lévén, hogy Budapesten készítem a szakdolgozatomat, ezért a Budapesten megtalálható rádiócsatornák a relevánsak számomra.

Budapest rádióállomások

Kattints a rádióállomás nevét hallgatni online

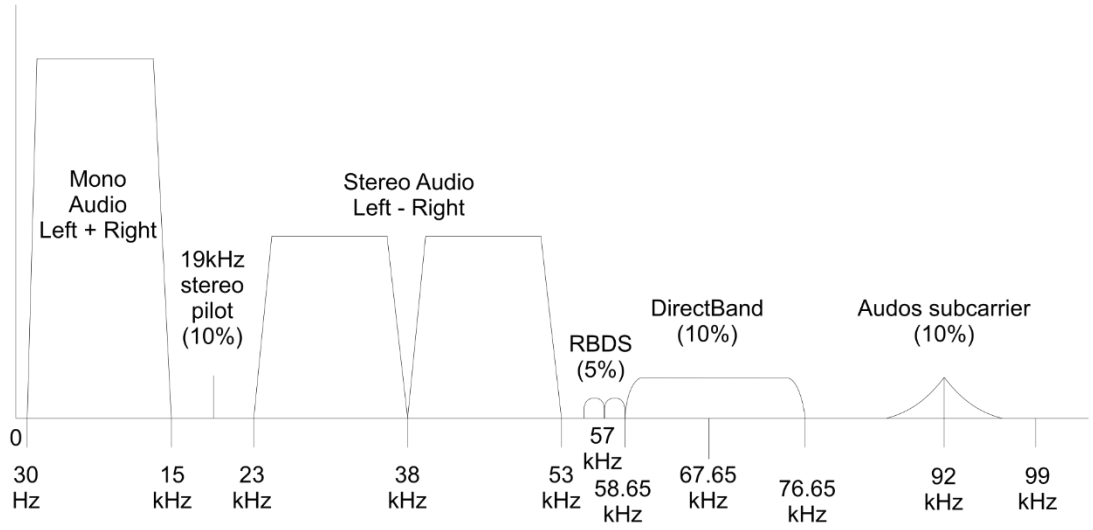
URH,MHz	Rádióadó neve	Adó helye
87.60	Broadway Rádió	Terézváros
88.10	InfóRádió	Nagyvárad tér, SOTE épület
88.80	Mária Rádió (Magyarország)	Gellért-hegy, Citadella rádióadó állomás
89.50	Music FM	Széchenyi-hegyi adótorony
90.10	RTVS Rádio Slovensko	Szlovákia, Besztercebánya, Száraz-hegy
90.30	Tilos Rádió	Gellért-hegy, Citadella rádióadó állomás
90.70	Bartók Rádió	Kékes tető adótorony
90.90	Jazzy Rádió	Sashegy adótorony
91.70	Mária Rádió (Magyarország)	Pécel
92.10	Klasszik Rádió	Gellért-hegy, Citadella rádióadó állomás
92.90	Klubrádió	Sashegy adótorony
93.90	Petőfi Rádió	Kab-hegy
94.20	Trend FM	Sashegy adótorony
94.60	RTVS Pátria Rádió RTVS Rádio Devín	Szlovákia, Érsekújvár, Elektrosvit Kémény
94.80	Petőfi Rádió	Széchenyi-hegyi adótorony
95.50	Kossuth Rádió	Kékes tető adótorony
95.80	Rock FM	Széchenyi-hegyi adótorony
96.40	Rádió 1	Gellért-hegy, Citadella rádióadó állomás
97.00	Első Pesti Egyetemi Rádió	VIII. kerület, ELTE

5. ábra Rádiócsatornák egy része Budapesten[9]

Látható, hogy tényleg sok van, ami így lehetőséget ad az architektúra képességeinek felmérésére (benchmark készítése). Az is látható, hogy két rádiócsatorna között minimálisan

[9] Forrás: http://radiomap.eu/hu/budapest

200 kHz távolság felfedezhető, de inkább a több jellemző. Ennek indokaira az FM sugárzás alapsávi spektrumképéből tudunk következtetni, ami a 6. ábra található.



6. ábra FM sugárzás alapsávi spektrumképe

A rádiócsatorna frekvenciája tulajdonképpen a vivőfrekvencia, amelyen ez a spektrumkép ül. Az első 15 kHz tartalmazza a mono audio jelet, ami arra elég, hogy a hasznos jelet kijátsszuk egy darab hangszóróra. A spektrum következő része 53 kHz-ig ahhoz kell, hogy sztereóban, azaz egyszerre két hangszórón tudjuk hallgatni a rádióadást. Végül pedig 100 kHz-ig egyéb szolgáltatásokat tartalmaz, mint például a Radio Broadcast Data System (RBDS), ami egy szöveget tartalmazó digitális jel. Olyan információkat hordoz magában, mint például a rádiócsatorna neve.[10]

Az alkalmazás szempontjából elég a mono hang kinyerése. Az architektúra vizsgálatához a többi nem ad hozzá, viszont arra építve később, ha bővíteni akarjuk a rendszert, érdekes lehet.

1.4. Eszközök ismertetése

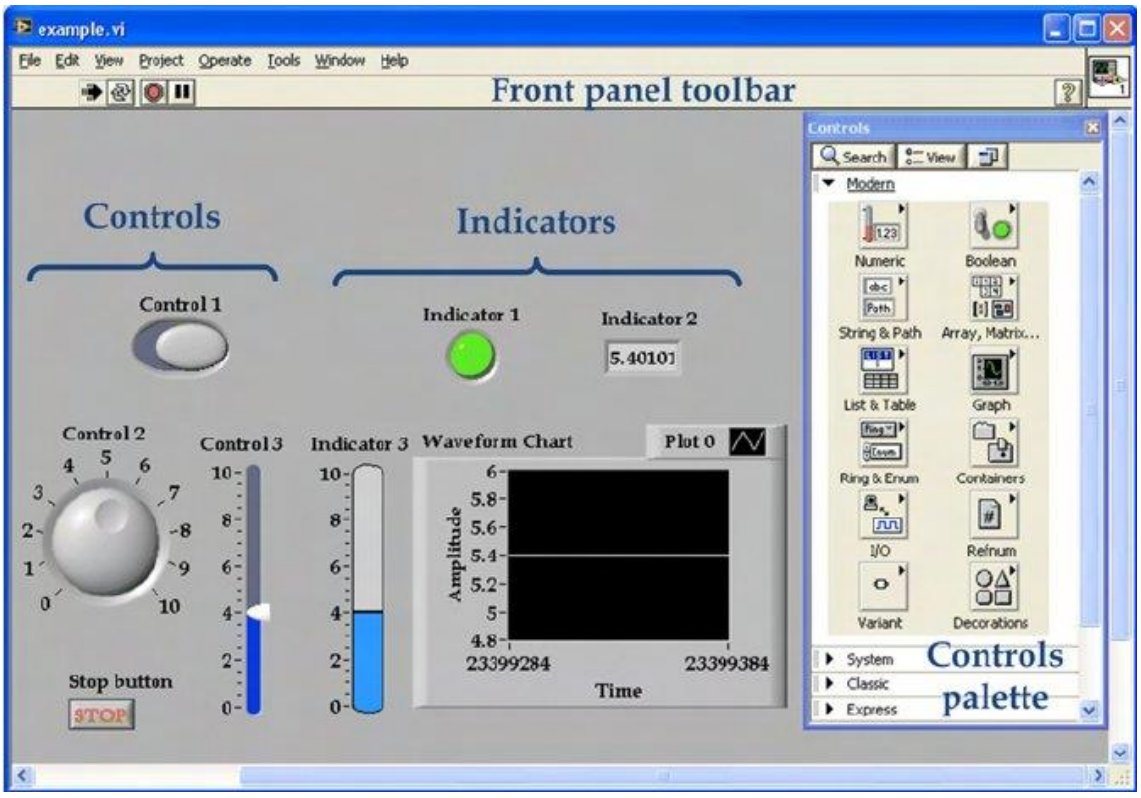
Ebben a részben bemutatom azt az eszköztárat, amit a National Instruments biztosított nekem a munkavégzés során ahhoz, hogy a lehető legtöbbet hozzam ki a szakdolgozatomból.

A LabVIEW egy adatfolyam alapú, grafikus programozói nyelv. Egy integrált fejlesztői környezetet biztosít, mellyel lehet kapcsolódni különböző mérő és vezérlő hardverekhez,

[10] https://en.wikipedia.org/wiki/Radio_Data_System

elemezni lehet a mért adatokat, akár publikálni is azokat, valamint olyan elosztott rendszereket lehet létrehozni, amivel számos ipari folyamatot lehet vezérelni. Nem csak NI termékeket, hanem más cég gyártotta eszközöket is lehet a LabVIEW-val vezérelni, de értelemszerűen az NI-os termékeket a legkönnyebb integrálni, hiszen minden berendezéshez saját driver program elérhető, ami a LabVIEW kiegészítő szoftvere, és megkönnyíti annak használatát.

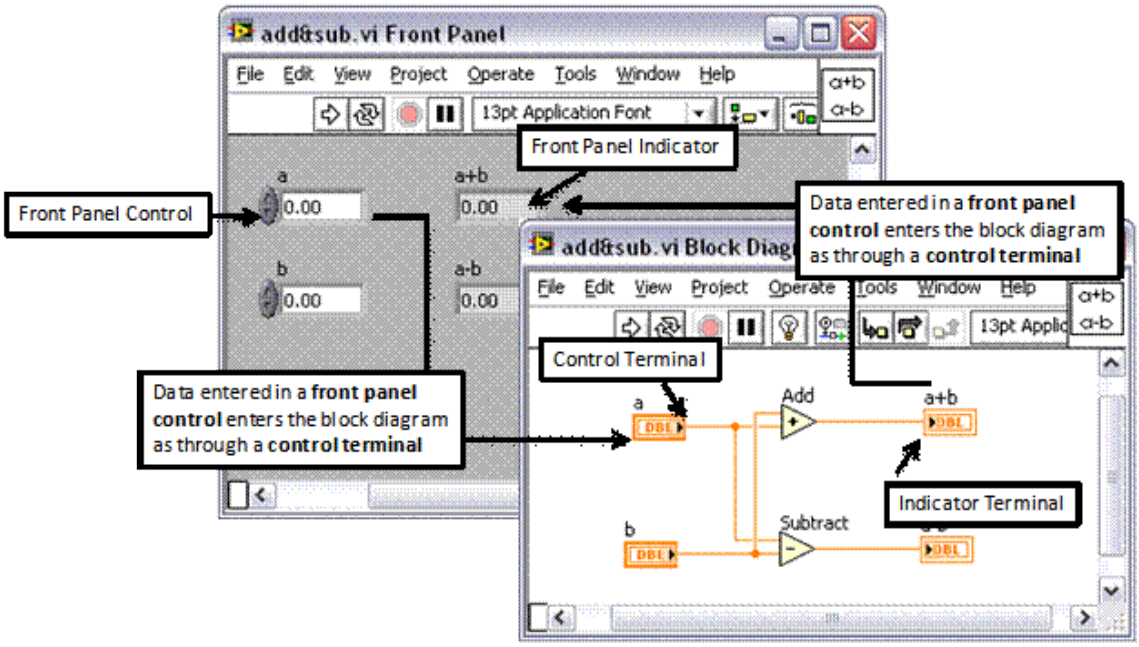
A LabVIEW-t két részre lehet bontani. Az első az úgynevezett Front Panel. Ez egy olyan felület, ami lehetővé teszi a programunk vezérlését, az eredmények megjelenítését és azoknak a kiértékelését. Ez egy olyan felület, amit az is tud kezelni (ha jól lett megcsinálva a program), aki nem is ért a programozói részéhez. A 7. ábra ezt mutatja be. A Control-ok segítségével lehet irányítani, változtatni a különböző paramétereket, míg az indikátorok kijelzőként funkcionálnak, eredményeket mutatnak be.



7. ábra Front Panel vázlata[11]

[11]https://www.researchgate.net/figure/LabVIEW-front-panel-example-and-the-controls-palette_fig1_221914350

A másik része pedig az úgynevezett Block Diagram, ahol a konkrét programozás történik a különböző dobozok, amiket VI-oknak nevezünk, és az azokat összekötő vezetékek segítségével.



8. ábra A Block Diagram és a Front Panel kapcsolata[12]

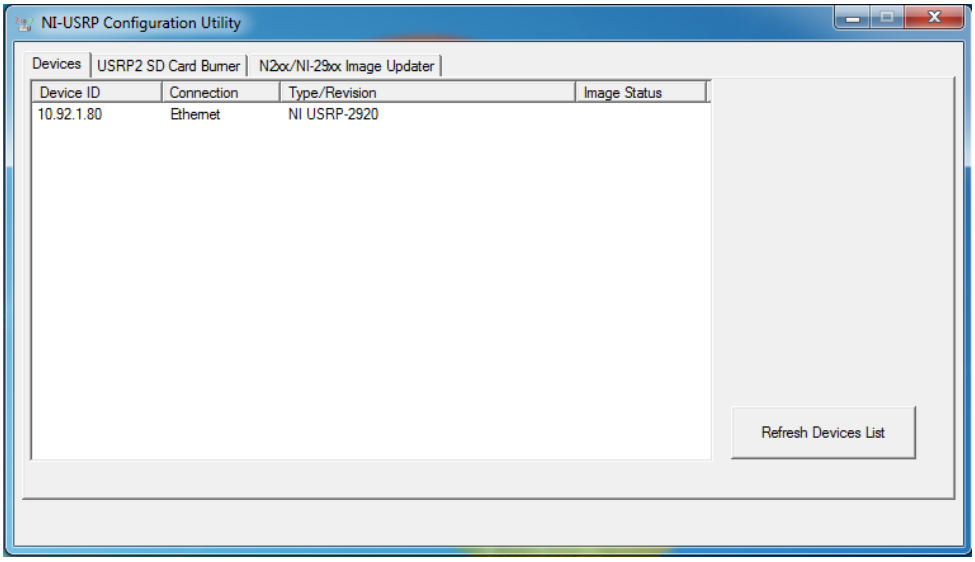
A 8. ábra látható, hogy kapcsolódik össze a Block Diagram és a Front Panel. A Control-ok segítségével beállítjuk a paramétereket, meghívjuk a függvényt a Block Diagramon, majd utána az Indicator-ok segítségével a végeredményt kijelezzük a Front Panelen.

A munka során a 2018-as, 32-bites verziót használtam. Azért ezt, mert a Front Panelhez tartozó NXG Style, azaz Next Generation Style olyan szép és letisztult formát kínál a rádiós panelnek, ami sokkal szofisztikáltabb megjelenést ad neki. Ezt mutatja be a 9. ábra És azért a 32-biteset, mert biztonságosabb, stabilabb működést biztosít, mint a 64-bites, pedig szóhossz és számítás szempontjából az utóbbi lenne a hasznosabb.

[12] http://www.ni.com/tutorial/7565/en/

9. ábra LabVIEW NXG Style[13]

Az SDR a National Instruments keretein belül a Universal Software Radio Peripheral (USRP) névre hallgat, azaz szoftveresen vezérelhető univerzális rádió adó-vevő egység. Az USRP-t a LabVIEW-ból vezérelhetjük, van hozzá külön driver csomag, így az USRP tetszőlegesen módon felkonfigurálható. A számítógépes kapcsolathoz egyedül gigabites Ethernet kapcsolatra van szükség (pár kivitel USB-vel is működik), és onnantól már körülbelül plug-and-play szinten működik. Van egy Configuration Utility, ami a különböző USRP eszközök kezelésére szolgál, ezt mutatja be a 10. ábra.



10. ábra USRP Utility Config

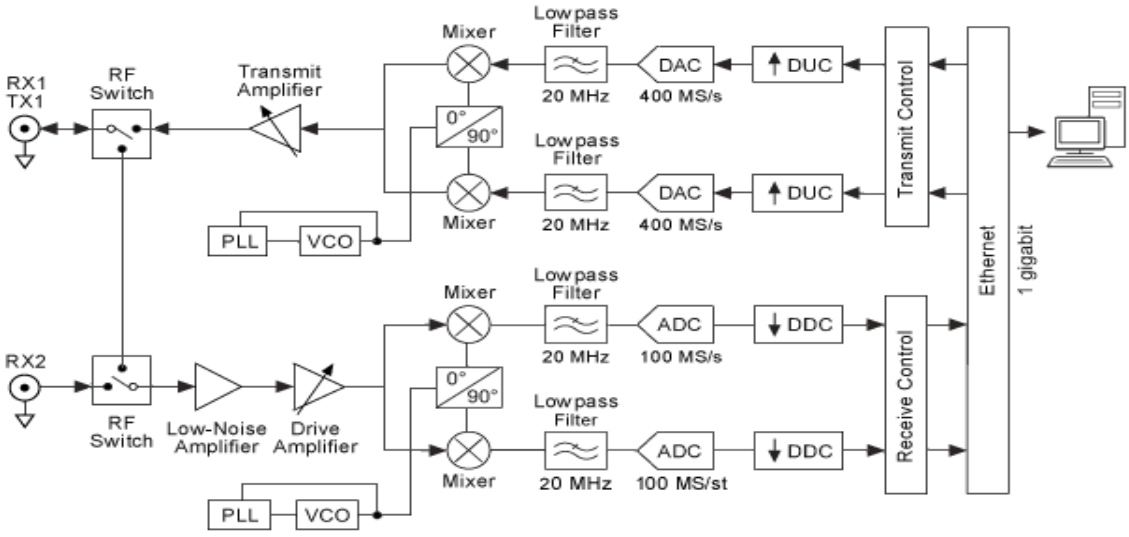
[13] https://www.electronicdesign.com/industrial-automation/introducing-labview-nxg

Ezen a felületen lehet beállítani az USRP IP címét, illetve feltölteni rá adatokat, amelyeket pl. FPGA-s alkalmazás esetén alkalmazhatunk.

Az USRP-t alapvetően kutatásra, fejlesztése, és oktatásra használható. Gyors, kompakt, relatíve egyszerű kezelni, cserébe sokrétűen lehet használni. Én a munkám során az USRP 2920-as verzióját használtam.

11. ábra USRP kezelői felülete

A 11. ábra az USRP kezelői felülete látható. A fémezett kivezetések a különböző be- és kimenetek. A felső 2 az Rx és Tx bemenetek/kimenetek, ahova a rádióantennát csatlakoztatni kell. A bal oldali 2 pedig konfigurálásra, kalibrálásra használható. Továbbá található rajta egy tápbemenet (jobb oldal), egy Ethernet csatlakozó, illetve egy MIMO csatlakozó a kaszkádosításhoz.



12. ábra USRP blokkvázlat

Az 8. alábbi ábrán pedig az USRP blokkvázlata található. Van benne A/D és D/A átalakító, hogy tetszőlegesen tudjon digitális és analóg jeleket kezelni. Emellett a jobb oldalon látható a kommunikáció a számítógéppel, ami a szoftver rádiózás lényege. Középen pedig a felkeveréshez és lekeveréshez használható IQ keverő látható, amiről Analóg FM rádiózás részben már volt szó. A lekeveréshez aluláteresztő szűrő is szükséges, ami 20 MHz frekvencián üzemel, ennek megfelelően a maximális sávszélessége 20 MHz. A lokáloszcillátor órajelét biztosítja a PLL, és a VCO.

Receiver

Frequency range	50 MHz to 2.2 GHz
Frequency step	<1 kHz
Gain range[5]	0 dB to 31.5 dB
Gain step	0.5 dB
Maximum input power (P_{in})	0 dBm
Noise figure	5 dB to 7 dB
Frequency accuracy[6]	2.5 ppm
Maximum instantaneous real-time bandwidth[7]	
16-bit sample width	20 MHz
8-bit sample width	40 MHz
Maximum I/Q sample rate[8]	
16-bit sample width	25 MS/s
8-bit sample width	50 MS/s
Analog-to-digital converter (ADC)	2 channels, 100 MS/s, 14 bit

13. ábra USRP 2920 rendszerjellemző paraméterei

Az alkalmazás szempontjából fontos, hogy az USRP 2920 milyen tulajdonságokkal rendelkezik. Az 50 MHz – 2,2 GHz tartományban használhatjuk, amiben megtalálható a 88 – 108 MHz, és hogy ezen belül 1 kHz pontossággal tudunk lépkedni. Ez bőven elég a használathoz. Az USRP maximum 20 MHz sávszélességet biztosít, ami elméletileg pont lefedi azt a sávot, amit használni fogunk, de a benchmark eredményei után kiderül, hogy ki tudjuk-e használni.

2. Alkalmazás megvalósítás

A témakör és a felhasznált eszközök részletezése után térjünk rá a témakör gyakorlati alkalmazására. Az általam meghatározott és a konzulensem által finomított feladat az volt, hogy megvalósítsak egy olyan rádiós szoftvert, amivel képes vagyok párhuzamosan egyszerre több csatornát hallgatni, illetve párhuzamosan felvenni és tárolni őket fájlban a számítógép háttértárolóján. Emellett lehessen vizsgálni az általam kiválasztott frekvenciatartomány spektrumát. Szintén fontos szempont volt, hogy a felhasználói interfész (User Interface, a továbbiakban UI) úgy nézzen ki, mint egy hétköznapi rádió. Emellett pedig a felhasználó dinamikusan, azaz a program futása közben változtathassa a paramétereket. Ezen paraméterek közé tartozik, hogy melyik csatornákat szeretné hallgatni, illetve felvenni a háttértárolóra, hogy mekkora sávszélességen belül kíván csatornákat választani, illetve legyen lehetősége megvizsgálni az adott sáv spektrumát.

A következők során taglalni fogom a megvalósítás lépéseit, kezdve a tervezési fázistól az egyes subVI-ok funkcióinak és megvalósításának bemutatásáig.

2.1. Tervezés

A projekt első lépése az, hogy megtervezzük, mit akarunk pontosan megvalósítani, és hogy erre pontosan milyen eszközök állnak rendelkezésre.

Mint minden szoftverfejlesztésnél, a legelső lépés az, hogy lebontjuk, pontosan milyen funkcionalitást valósít meg a program. Ha ez megvan, akkor ennek megfelelően kell tervezni UI-t a felhasználónak, ami tartalmazza azokat a funkciókat, amelyeket pontosan meghatározunk az első lépésben. Ha a UI tervezés készen van, akkor utána jöhet az ennek megfelelő Back End megtervezése, ami ténylegesen az adott funkcionalitás megvalósítása. Ehhez össze kell írni, hogy milyen eszközök állnak rendelkezésünkre, és hogy ezekből hogyan valósíthatóak meg a különböző funkcionalitások.

A konkrét esetben tehát azt kell meghatározni, hogy milyen rádiós UI-t kap a felhasználó, ez milyen funkciókkal bír, utána át kell tekinteni az USRP szoftveres kereteit, majd a LabVIEW adta lehetőségeket, ez alapján meghatározni, hogy milyen pontosan valósíthatóak meg a funkciók, és milyen megkötésekkel kell élnünk. Ezen lépéseket fogjuk most végigvenni a következő fejezet során.

2.1.1. Funkcionalitás

Első lépésként meg kell határozni, hogy pontosan milyen funkciókat valósítunk meg a fenti feladatkiírás alapján. Ehhez tisztázni kell a terminológiát.

- Rádióállomás az analóg FM műsorszolgáltatás által kiadott hanganyag (13. oldal)
- Párhuzamos működés: ugyanazon idő alatt egyszerre több, önálló feladat fut
- Tárolás: Hanganyag kiírása fájlba a számítógépen
- Dinamikus működés: a felhasználó a program megállítása nélkül tudjon paramétereket változtatni
- Spektrum: az adott jelek frekvencia-tartománybeli reprezentációja

Ezen fogalmakat tisztázva meg lehet határozni a konkrét funkciókat, amit elvárunk a programtól.

- Párhuzamos csatornahallgatás és rögzítés:
  A hallgatható csatornák pontos száma attól függ, hogy milyen számítógépet használunk. Elsősorban a számítógép processzora és buszrendszere határozza meg, illetve feladat specifikusan a hangkártya és a Hard Drive Disk/Solid State Drive (HDD/SSD). Ezek adatátviteli sebessége határozza meg az elérhető maximumot. Ezek alapján lehet azt mondani, hogy egyszerre 2, 3, 5 vagy 10 csatorna.
- Dinamikus működés:
  Az architektúra tervezés során figyelni kell arra, hogy a UI tényleg érzékeny legyen a beavatkozásra, amelyet adott időegység alatt képesnek kell lennie figyelembe venni. Ezen időegység kiszámolását a megtervezett architektúra teszi lehetővé, de alapelv, hogy ne pufferelődjön sok adat, és 1 másodpercen belül reagáljon az interakcióra
- Spektrum:
  Olyan grafikon legyen, ami a beállított sávszélességre jeleníti meg a spektrumképet. Elvárás, hogy a grafikon ne ugráljon minden apró teljesítmény ingadozásra, hanem tényleg csak akkor változzanak meg a tengelyértékek, ha annak konkrét kiváltó oka van. Erre a Property Node-ok kiváló megoldást nyújtanak, amik a Back Enden találhatóak, és biztosítják, hogy egyes elemek a Front Enden milyen formában jelenjenek meg.

A funkciók konkretizálása és számszerűsítése után lehet rátérni az egzakt tervezési fázisra.

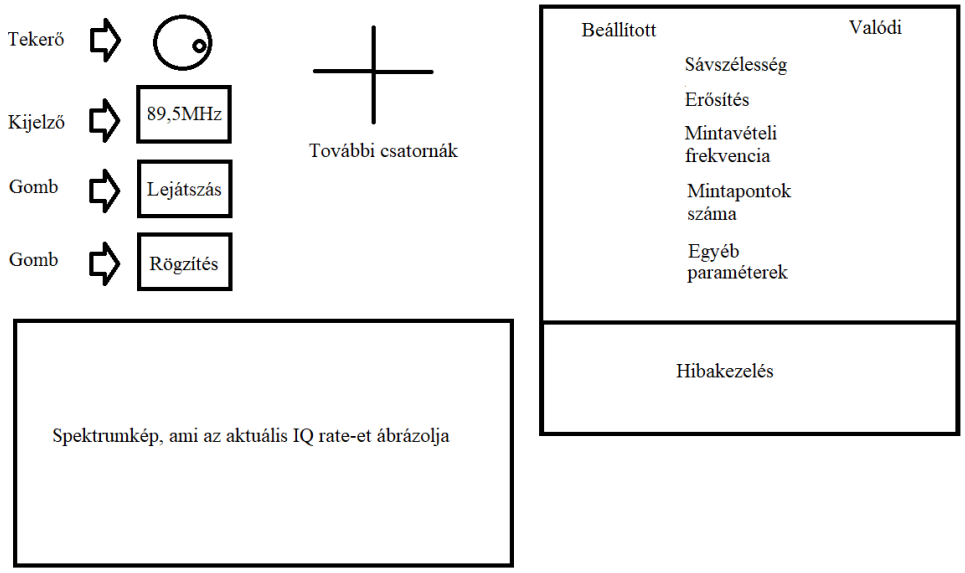
2.1.2. Front End

A tervezési fázis következő lépése a funkcionalitás tisztázása után az, hogy milyen felületet kap a felhasználó, amin keresztül a programot használni tudja. Ezt nevezi a LabVIEW Front Endnek, vagy általánosan a User Interface (UI) kifejezés használható. Ez kritikus része a programnak, hiszen ez alapján fog megítélést kapni a program. Lehet akármilyen jó a Back End, ha a felhasználó ezt nem látja viszont a UI-n, akkor rossz programnak lesz elkönyvelve.

Az előző részben taglalt funkcionalitás alapján a következőt várja el a felhasználó a UI-tól:

- Ahány csatornát lehet maximum használni, annyi szabad csatorna legyen feltüntetve a UI-n. A LabVIEW-ban a nem aktív, de használható elemeket szürkével jelöli.
- Minden egyes csatornához tartozzon egy gomb, amivel be lehet kapcsolni a hangszórót, meg egy másik gomb, amivel a tárolást lehet elindítani
- Minden egyes csatornán tetszőlegesen lehessen állítani a frekvenciát az itt használt 88 – 108 MHz között, egy adott pontosság szerint. Illetve legyen egy kis csúszka, amivel adott pontosságnyit lehet lépkedni. Ha valamelyik határt elértük, akkor a csúszka léptetése esetén vigyen át a másik határértékre folytonosan
- Egy grafikon, ami az általunk használt frekvenciatartomány spektrumképét jeleníti meg. Az X-tengelyen a frekvenciatartomány legyen, az origó a frekvenciatartomány közepét reprezentálja. Az Y tengely pedig az adott frekvencián található jelnek a teljesítményét jelenítse meg dBm-ben.. Az X-tengely ne ugráljon minden kis dBm változás miatt, csak akkor változzon, ha nagyobb dBm változás történt (ez a programban 5 dBm változás szerint lett beállítva).
- Egy sávszélességet szabályozó csúszkát. Ez elméletileg 0 – 20 MHz között változhat, tetszőleges pontossággal, azonban az implementáció során változhat
- Egy gombot, amivel meghatározzuk, hogy az USRP melyik antenna kimenetét kívánjuk használni
- Ezen kívül még érdekes lehet egy-két indikátor, ami az USRP működésére, vagy az által közvetett kapcsolatban lévő paraméterre vonatkozik. Ez majd tárgyalásra kerül az USRP szoftveres részénél

Ezek voltak a főbb tervezési szempontok, amiket meg lehet fogalmazni a korábbi funkcionalitás alapján. A tervezett UI-t az alábbi ábra mutatja meg:



14. ábra A tervezett UI

Az 10. ábra felső részében találhatóak a csatornák. Minden csatornához tartozna egy tekerő, amivel lehet állítani a frekvenciát a megadott pontosság szerint. Ezalatt a kijelző, ahol kézzel is be lehet írni a kívánt frekvenciát. Utána pedig két gomb, egyik a hangkártyára való kijátszásra, a másik pedig a háttértárolón való mentésre.

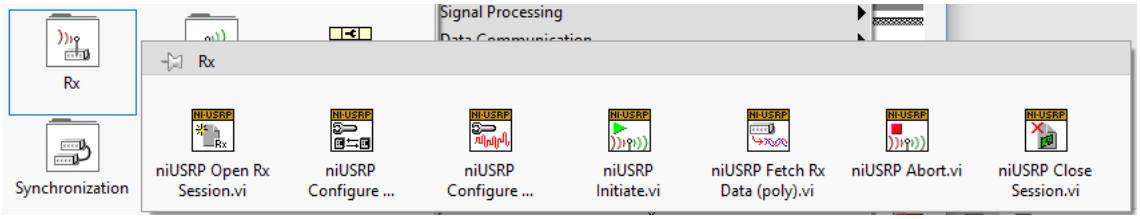
A csatornák alatt található a spektrumkép, ami mindig az aktuális sávszélességet jeleníti meg az X-tengelyen, míg az Y-tengelyen a teljesítmény. Ezek mellett pedig állítható paraméterek, mint a sávszélesség, az erősítés, a mintavételi frekvencia, a mintapontok száma, illetve egyéb, később felmerülő paraméterek felkerülhetnek még. Az egyik oldalon az, amit a felhasználó beállít, a másik oldalon pedig az, ami valójában a rendszerben van. Ezt az USRP hardveres kerekítése okozza. Van még egy panel, a hibakezelés, ami kiírja, hogyha valamilyen hiba miatt megáll a rendszer. A megállítás pedig a panel bezárásával történne, nem külön dedikált Stop gommbal, hanem a panel bezárásával. A felhasználónak nincs szüksége arra, hogy lássa a megállított LabVIEW-t.

2.1.3. USRP szoftveres működtetése

Következő lépésként át kell nézni, hogy az USRP szoftveresen hogyan van implementálva a LabVIEW-n belül. Ennek megfelelően kell megtervezni a későbbi architektúrát, hiszen azt az USRP köré kell építeni.

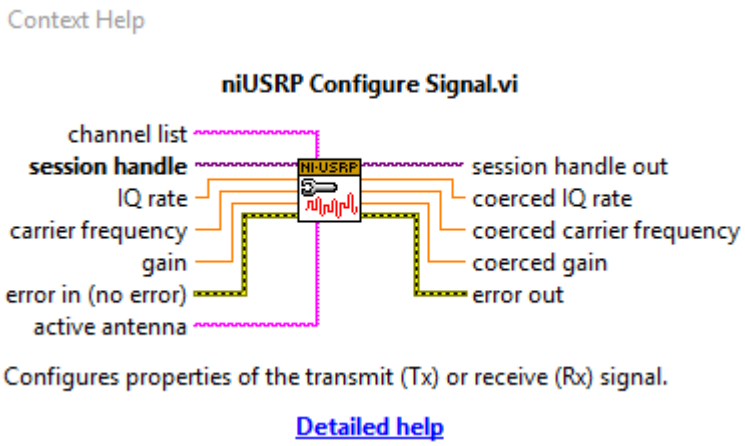
Az USRP, mint ahogyan arról az eszközök ismertetése résznél is írtam róla (15. oldal), egyszerre egy rádióadó és -vevő egység is. Éppen ezért értelemszerűen mindkét üzemmód szerint konfigurálható. Ezen kívül még van pár VI, ami segít az USRP szinkronizációjában, hibakezelésben, illetve bármilyen paraméter változtatás véglegesítésében. Ezek megtalálhatóak a Synchronization, illetve a Utility fül alatt.

Nekünk az alkalmazás szempontjából az Rx-re, azaz a rádióvevő részre van szükségünk.



15. ábra Az USRP Rx fülhöz tartozó VI paletta

Beszédes ez az ábra, hiszen a VI-ok nevei egyben magukban hordozzák a funkcionalitásukat, másrészről pontosan olyan sorrendben vannak, ahogy egy USRP Session, azaz munkaciklus felépül. Az Open Rx Session az, mint ami nevének szó szerinti fordítása, megnyitja a munkafázist. Átad egy session handle-t, ami lényegében a sessiont azonosítja a későbbi VI-ok számára, ez a legfontosabb paraméterük. A következő egy olyan VI, ami a Number of Samples-t, azaz a minták számát állítja be.. A következő a Configure Signal meghatározza, hogy az USRP milyen paraméterekkel dolgozik.

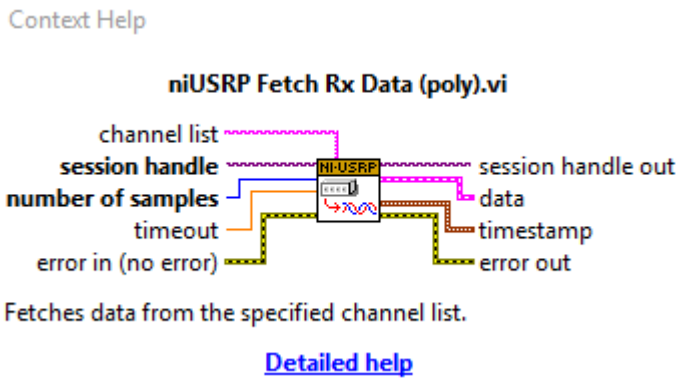


16. ábra A Configure Signal által beállított paraméterek

Az alábbiakban részletesen kitérek a 11. ábrán bemutatott Configure Signal elnevezésű VI-ra. A VI a channel list paraméterén keresztül megadható egy lista a csatornákról. Ezzel paraméterrel a fejlesztés során nem kellett foglalkozni. Az IQ rate meghatározza az alapsávi IQ adat mintavételezési frekvenciáját, amely meghatározza a sávszélességet (a továbbiakban IQ rate-ként fogok rá hivatkozni), ami maximum 20 MHz lehet. Ilyenkor 16 bites adatokat használhatunk (15. oldal). A carrier frequency az a vivőfrekvencia, amire a hasznos jel „rá van ültetve”, és amiről az USRP lekeveri a jelet az alapsávra. Ez 50 MHz és 2,2 GHz között lehet, a frekvenciafelbontás legfeljebb 1 kHz (15. oldal). A Gain az RF Erősítést jelenti dB-ben. Ez az érték 0 és 31,5 dB között változhat legfeljebb fél dB felbontással.[14]

Visszatérve az xy ábrára a következő VI az Initiate, ami pusztán annyit csinál, hogy az előbb definiált paraméterek alapján elindítja az RF jel vételét. Ha ez is sikeresen lefutott, akkor utána következik a komplex burkoló mintáinak meghatározása majd feldolgozása. Erről a Fetch Rx Data VI gondoskodik.

[14] http://www.ni.com/pdf/manuals/375839c.pdf



17. ábra a Fetch Rx Data VI paraméterei

Kritikus bemeneti paraméter a Number of Samples, azaz a minták száma, ez megadja, hogy az adott IQ rate-et milyen felbontással kapjuk vissza. Ez egyben egy időzítést is rak a rendszerbe a következő egyenlet végett:

$$IQrate * Timing = Number\ of\ Samples,$$

ahol Timing az időzítést jelenti. Az IQ rate-et már korábban beállítottuk, és ennek a VI-nak az időzítését az fogja meghatározni, hogy mit kötünk be a Number of Samples helyére, hiszen az időzítés a kettő hányadosa. Éppen ezért célszerű az egyenlet formájában bekötni ezt a paramétert, hogy szabályozni tudjuk ezt az időzítést.

A Timeout egy olyan paraméter, ami meghatározza, hogy mennyit várjon a VI a mintákra, mielőtt hibát jelezne. Ha nincs bekötve, akkor default érték szerint nincs timeout.

Kimenetként pedig kiadja a kért adatokat az USRP. Több formátumban is vissza lehet kérni az adatokat, és ez kritikus jelentőségű a későbbi tervezés szempontjából. Az adatformátumok az alábbiak:

- CDB Cluster: Az IQ adatot komplex, dupla-pontosságú (64 bit) lebegőpontos számformátumú cluster (csoport), ami tartalmazza az adatpontokat a fent említett formátumban, a mintapontok közötti időintervallumot (ezt hívjuk dt-nek, és $dt = \frac{1}{IQ\ rate}$), illetve egy t0 indulási időt
- CDB Waveform: Ugyanaz, mint a CDB Cluster, csak ez Waveform (hullámforma) formátumban adja ki az adatot
- CDB: Ez a cluster helyett csak az adatpontokat adja vissza
- I16: Az IQ adatot egész formátumban, 16 biten ábrázolva adja vissza. Ez azt jelenti, hogy -32768 – 32767-ig bármilyen egész számot felvehet. A tömb páratlan

elemei I adatot, a páros elemei Q adatot adnak vissza. A tömb mérete kétszer akkora, mint a minták száma, mert egy I és egy Q adat együtt alkot olyan adatot, ami szükséges a működéshez

Ezen adatformátumokból kell kiválasztani azt, hogy melyiket célszerű használni a rendszerben. Ehhez azonban specifikálni kell, hogy milyen műveleteket kell a rendszeren belül végrehajtani, ahol nyerhetünk erőforrást azzal, hogy kisebb bitszámú reprezentációt választunk, cserébe viszont számolni kell azzal, hogy ez a minőséget is rontani fogja.

Az adatgyűjtés után már csak az maradt, hogy mit kell csinálni akkor, ha le akarjuk állítani ezt a folyamatot. Erre van az utolsó két VI a palettáról. Az Abort VI az Initialize ellentéte, leállítja az adatgyűjtés folyamatát, míg a Close session VI a korábban megkezdett Sessiont zárja le és felszabadítja az eszközt mint erőforrást. Fontos ezeket meghívni, hogy ne maradjon adatszemét a memóriában és hogy az eszköz más alkalmazás számára is elérhetővé váljon..

Mint láthatjuk, a LabVIEW kényelmes használatot biztosít az USRP számára. Következő lépésként ezen elemek használatával kell megtervezni az architektúrát.

2.1.4. Back end és adatgyűjtés

Eddig tehát meghatároztuk, hogy a program milyen funkciókkal rendelkezzen, milyen legyen a UI, és hogy az USRP milyen funkciókat kínál fel nekünk a programozás során. Ebben a részben azzal fogunk foglalkozni, hogy milyen lesz a Back End, vagyis milyen elemekből fog felépülni a kód

Először is, az USRP szoftveres reprezentációjából következik a működési mechanizmus: elindítunk egy sessiont, felkonfiguráljuk a megfelelő kezdeti paraméterekkel, ennek megfelelően elindítjuk az adatgyűjtési fázist, ezt folyamatosan használjuk, közben az adatokat feldolgozzuk, majd pedig, ha már nem akarjuk tovább használni a programot, akkor megállítjuk az adatgyűjtést és adatfeldolgozást, lezárjuk a sessiont, bezárjuk a LabVIEW-t.

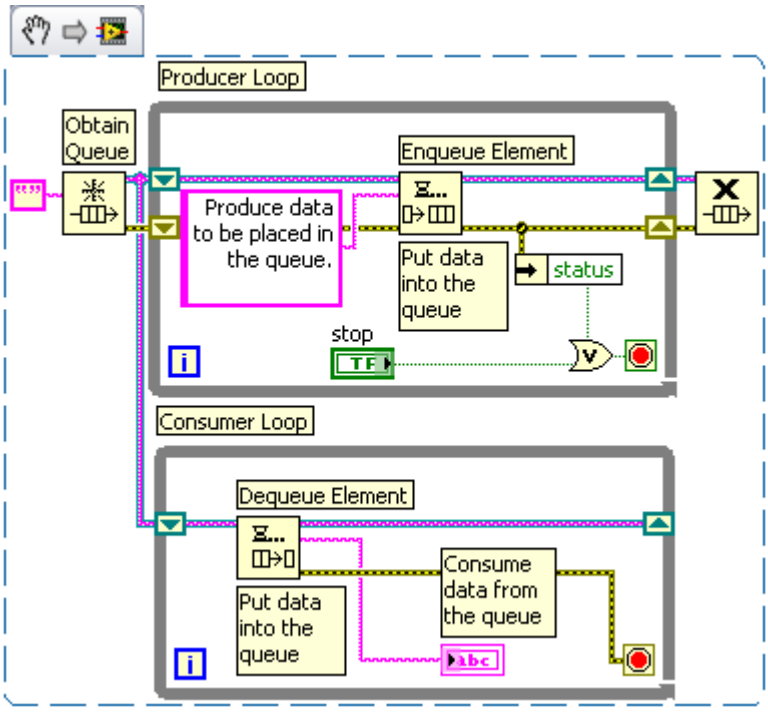
Ebből két alapvető dolog következik:

1) Lesz kettő While ciklus, az egyik az adatgyűjtésért felel, a másik pedig az adatfeldolgozásért. A kettő függ egymástól, ha valamelyik hibára fut, akkor az egész program megáll. Ugyanezen elv mentén, ha a felhasználó leállítja az adatgyűjtést (ezzel pedig a programot), akkor az adatgyűjtés huroknak meg kell állítania az

adatfeldolgozás hurkot is. Mindenesetre a két huroknak kommunikálnia kell egymással, a gyűjtőnek el kell juttatnia az adatokat a feldolgozó egységnek. A feladatra van egy bevált és jól használható architektúrája, ami a Producer & Consumer Loop névre hallgat. Ez lényegében azt valósítja meg, amit az előbb definiáltunk. A kommunikációt Queue, azaz Sor segítségével valósítjuk meg, erről ebben a fejezetben lesz szó

2) A működés során a program több állapotban működik. Egyrészt van az inicializálás állapota, amikor beállítjuk az USRP-t a kívánt paraméterek szerinti működésre. Az adatgyűjtés állapota, amikor a Producer Loopban gyűjtünk adatot, elküldjük az Consumer Loop-nak, ami feldolgozza azt. Van a leállási állapot, amikor a felhasználó úgy dönt, hogy befejezi a program használatát. Ehhez azonban figyelembe kell még venni, hogy a felhasználónak a program dinamikus működést nyújt, ami azt jelenti, hogy paramétereket menet közben megváltoztathat. Ez azonban csak úgy működik, ha újra felkonfiguráljuk az USRP-t, mert más lehetőséget nem kínál fel. Ez szül meg még egy állapotot, az úgynevezett újraindítás (reset) állapotot. Ez arra van, hogy megállítja az adatgyűjtést, lezárja a sessiont, majd pedig a program visszatér az inicializálási állapotába

A következő két ábra az ezekből következő architektúrát, illetve az ennek megfelelő állapotgépet fogja bemutatni.



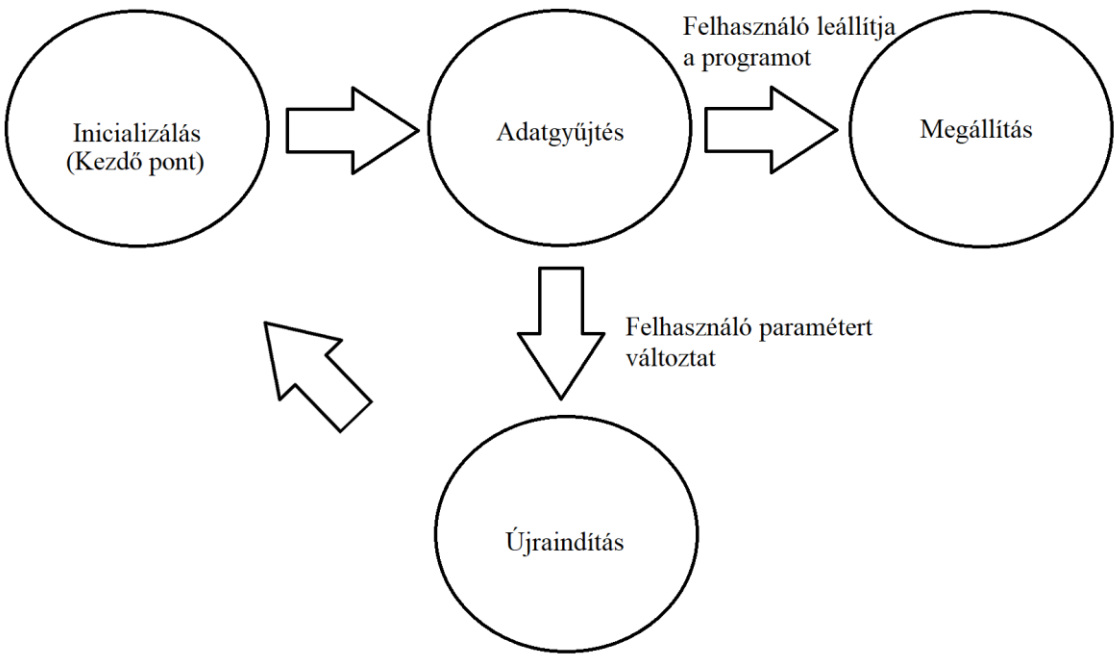
18. ábra Producer/Consumer Loop architektúra[15]

Az architektúra tehát tartalmaz két While Loopot, az egyik a Producer, a másik pedig a Consumer. A kettő közötti kommunikációért pedig a Queue, azaz Sor felelős. A Queue konkrétan egy FIFO (First in – First out, azaz ami először be, az először ki) puffer, ami azt jelenti, hogy az adatokat továbbítja a két Loop között, továbbá tud adatokat tárolni, ha esetleg az adatgyűjtés gyorsabb, mint a feldolgozás. A Queue egyszerűen használható:

1) Az Obtain Queue első lépésében létrehoz egy referenciát, amivel megszületik a Queue, és meghatározza mekkora legyen a puffer, és milyen adattípussal működjön
2) Az Enqueue Element fogja a továbbításra szánt adatot, és berakja a pufferbe
3) A Dequeue Element a legelső adatot kiveszi a pufferből, és továbbadja azt az őt használó feldolgozó egységnek
4) A Release Queue törli a referenciát (viszont a pufferben maradt elemeket nem, arra külön VI, a Flush Queue van)

A Queue a mi feladatunkra ideális eszköz, hiszen nekünk egy olyan kommunikációt kell megvalósítanunk, ami adott esetben akár pufferelni is tudja az adatokat.

[15] http://www.ni.com/white-paper/3023/en/



19. ábra Az állapotgép terve

Ez az állapotgép a fenti specifikáció alapján. Kezdünk az inicializálási állapotból, amiből, ha minden rendben van, átlépünk az adatgyűjtési állapotba. Ebben megy a program egészen addig, amíg a felhasználó nem akarja leállítani a programot, vagy nem akar megváltoztatni egy paramétert. Utóbbi esetben az Újraindítás állapotba kerül, ahol is lezárja és törli a korábbi sessiont, majd pedig átlép az inicializálási állapotba, és közben ez az állapot megkapja az új paramétert.

Az állapotgépet eseményekkel tudjuk vezérelni. Eddig ezek az események előbukkantak említés szintjén, de specifikusan az események a következőképpen jelennek meg:

- Frekvenciaváltoztatás, új csatorna aktiválása:
  Ebben az esetben adatgyűjtésről újraindítás állapotba lépünk át, majd onnan tovább az inicializálásra
- Erősítés változtatása:
  Ha inicializálás közben történik, akkor ignoráljuk, utána pedig nem okoz állapotváltást (40. oldal)
- IQ rate változtatása:
  Ebben az esetben ugyanaz az eljárás, mint a frekvenciaváltozás esetén
- Panel bezárás:
  Ha a felhasználó a panel bezárása gombra kattint, akkor először kap egy biztonsági üzenetet, ami szerint vagy visszatér az adatgyűjtéshez, vagy pedig tényleg

bezárja a programot. Ha bezárja, akkor a Megállítás állapotba kerül az állapotgép, ami leállítja az USRP adatgyűjtését, ezáltal az Adatgyűjtés Loop-ot, ezt követően pedig hibaüzenet segítségével az Adatfeldolgozás Loop-ot. Erről részletesebben az Implementációnál lesz szó. (40. oldal)

- Timeout:
  Ezt a LabVIEW generálja, nem használjuk semmire, maradjon az aktuális állapotban

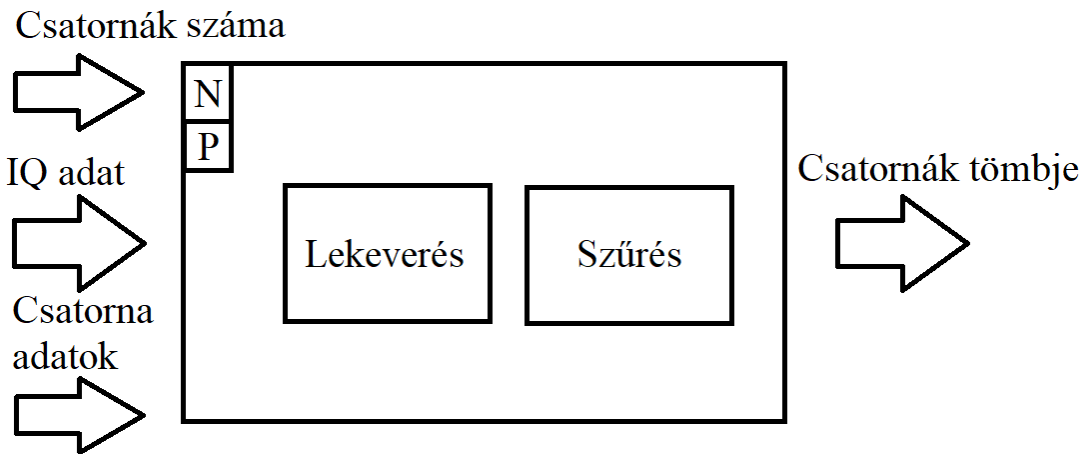
2.1.5. Adatfeldolgozás

Ez logikailag még szintén a Back Endhez tartozik, azonban célszerűnek érzem itt külön fejezetre bontani, mert ez nem csak programozási technika alapvetően, hanem matematikai összefüggések implementálása.

Az FM moduláció és demoduláció, illetve az USRP ismertetése kapcsán már láttuk azt, hogy egy rádióadás hogyan jut el a végfelhasználóhoz, és hogy ebben az USRP milyen feladatot lát el. Azonban egy kicsit mást kell tenni ahhoz, hogy párhuzamosan, egyszerre több csatorna is eljusson a felhasználóhoz. Az USRP ugyan lekever egy adott frekvenciáról az alapsávba, de ezt kvázi egyszerre kellene megtennie több csatornával, ami viszont hardver szinten nem lehetséges. Szoftveres lehetőség azonban van. Célszerű úgy beállítani az USRP középfrekvenciáját, hogy az az átlaga legyen a megadott frekvenciáknak. Erről a frekvenciáról az USRP lekever az alapsávba, ami az IQ rate-nek megfelelő adatot adja vissza 0 középfrekvenciával. Ezen sávon belül találhatóak azok a rádiócsatornák, amelyeket a felhasználók beállítottak. Éppen ezért az alapsávi spektrumot minden csatorna esetén annyival kell megszorozni, hogy a csatorna frekvenciája éppen l a 0-án legyen, majd pedig megszűrni, hogy csak az érdemi információ maradjon meg, ne legyen átlapolódás. Figyelni kell az előjelekre, mindig ellentétes előjelű frekvenciával kell szorozni ahhoz, hogy a 0-ba kerüljünk. Ennek egyszerű az oka: szinuszokkal és koszinuszokkal szorzunk, amelyeket Fourier-transzformáció alá vetve két tüskét kapunk. Az egyik tüske a szinusz/koszinusz frekvenciáján található az összteljesítmény felével, a másik tüske pedig ugyanez a negatív frekvenciatartományban. Ha viszont IQ jelet használunk, ami már komplex jel, akkor már csak a pozitív frekvenciás komponens marad meg. Éppen ezért szinuszokkal és koszinuszokkal való szorzás a frekvenciával való összeadásba és kivonásba megy át, ezért kell az ellentettet venni.[16]

[16] Forrás: S. Haykin, *Communication Systems*

Ezt minden csatornára alkalmazva megkapjuk a csatornákat külön-külön. Fontos, hogy ez párhuzamos működés, éppen ezért szükség van csatornánként egy processzor magra. Ha a csatornafrekvenciák a 0-ra kerültek, akkor utána az adott jelet meg kell szűrni az FM műsorszórás szerint. (13. oldal) Azt az egységet, ami elvégzi ezen műveleteket, Digital Down Converternek (továbbiakban DDC) hívjuk, ami Digitális Lekeverőt jelent. Ez az egység felelős azért, hogy a beérkező jeleket az alapsávba lekeverje.

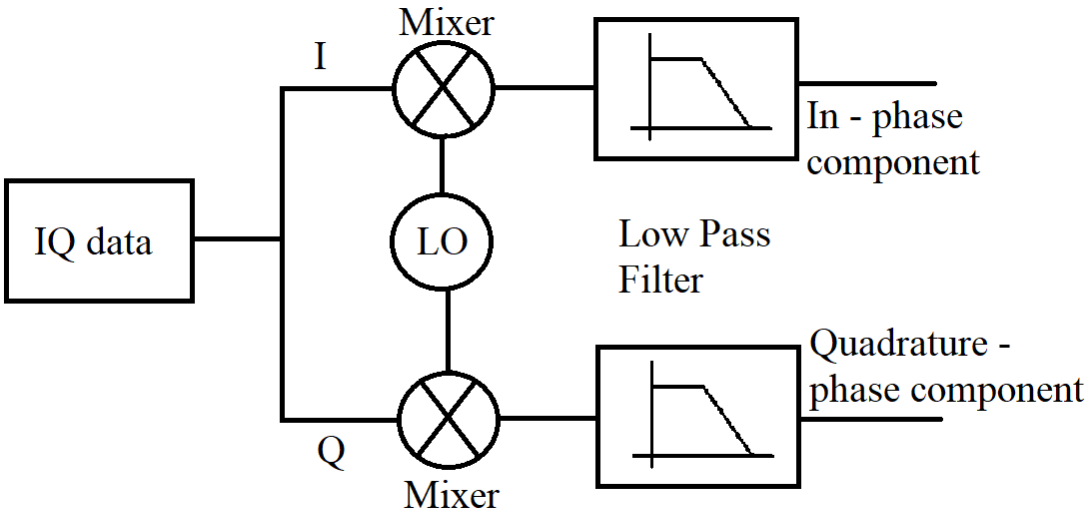


20. ábra DDC blokkvázlata

A DDC bemenetként megkapja, hogy éppen hány csatornát használunk, az IQ adatot, valamint egyéb csatorna adatot, amire szükség lesz. Ilyen adat a csatorna frekvenciája, a csatorna aktuális fázishelyzete, valamint a mintavételezési adatok. A frekvencia egyértelmű, a fázishelyzetre azért van szükség, mert a DDC mindig kap X számú mintát, és az utolsó minta egy adott fázishelyzetben van. Ezt meg kell őrizni, hogy a következő adag minta az előző törésmentes folytatása legyen. A mintavételi adatok pedig a mintavételi frekvenciát, illetve az adatpontok számát jelenti. Előbbinek teljesítenie kell a Nyquist-Shannon mintavételi tételt. Itt már csak az alapsávot kell mintavételezni, ez a lekeverés lényege. Ha nem kevernénk le, akkor a vivőfrekvencia kétszereséig kellene mintavételezni. Ezen a ponton az USRP azonban már lekeverte nekünk a jelet alapsávba. Az adatpontok számát pedig már meghatároztuk a Fetch Rx Data VI-nál. (25. oldal)

Az előbb felsorolt adatok bekerülnek egy For Loopba, ami annyiszor fut le, ahány csatornát használunk (N = Iteration Number, iterációk száma). A P pedig a párhuzamosságot (Parallel) jelenti. LabVIEW-ban lehetőségünk van feltételes For Loop hívásra (Condition Terminal), aminek következtében beállíthatjuk, hogy minden csatorna külön processzor

magon legyen feldolgozva. A For Loopon belül az adatok alapján elvégezzük az újabb lekeverést az ebben a fejezetben taglalt módon, utána a lekevert adatok átesnek egy szűrésen. Kimenetként pedig megkapjuk a lekevert csatornák tömbjét, ami demodulálásra, hangkártya kijátszásra és tárolásra vár.



21. ábra Lekeverés és szűrés

A lekeverés az 21. ábra blokk diagramja szerint történik. Beérkezik az IQ adat, ezt szétbontjuk I-re és Q-ra, majd utána a korábban ismertetett eljárás szerint Local Oscillator (LO) segítségével megszorozzuk az I és Q jelet, majd szűrjük azt, végül megkapjuk az alapsávi szűrt IQ jelünket. (6. oldal)

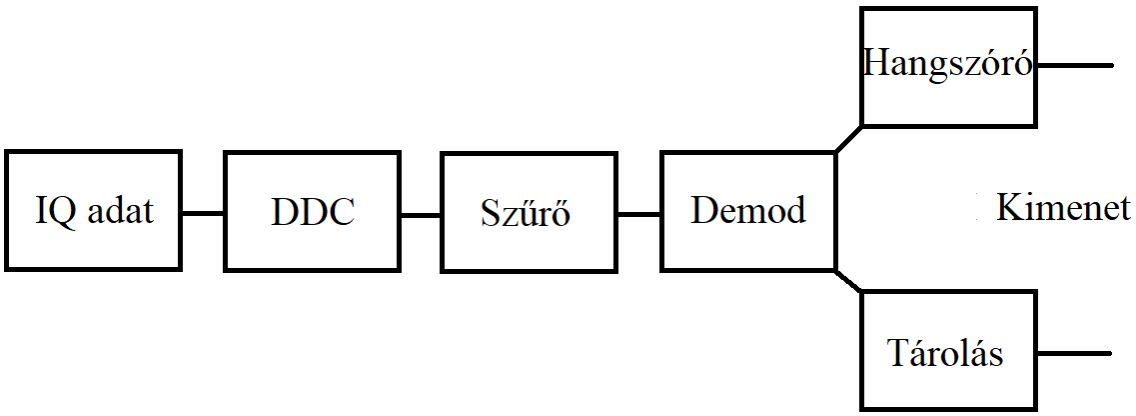
Ez az egység a rendszer egészét illetően kritikus jelentőséggel bír. Rengeteg matematikai művelet történik ezen a modulon belül, a mixer lényegében szorzást valósít meg. Ráadásul komplex szorzást, hiszen az IQ jel valójában egy komplex jel, ami valós és képzetes részre van bontva, az LO pedig szintén, ezek pedig már valós jelek. Két komplex szám szorzata az alábbi képlet szerint történik:

$$(a + b * i) * (c + d * i) = a * c - b * d + i * (a * d + b * c)$$

Ez azt jelenti, hogy a modul elvégez 4 szorzást, egy összeadást és egy kivonást. Ez sok adatra nézve rengeteg számítási időt emészt fel, hogyha nagyon precízek akarunk lenni. A LabVIEW alapértelmezés szerint dupla precizitású lebegőpontos számábrázolást használ, és minden ilyen szám 64 biten van ábrázolva. Ezt az teszi még nehézkessé, hogy a LabVIEW 32 bites program, ami miatt jelentősen terheli a memóriát a 64 bites számok ábrázolása.

Itt jön tehát képbe a számábrázolás fontossága. Az USRP adatgyűjtése ennek megfelelően konfigurálható. (25. oldal) Ennek megfelelően az I16-os adattípust kell választani, ami 16 bites egész számokat ad vissza. Utána ezen adatokat konvertáljuk át 16 bites, fixpontos számokká, melyből 1 egész rész, 15 törtrész. Ennek az az oka, hogy egységnyi nagyságú amplitúdójú szinuszos jelekkel dolgozunk, a törtrészt kell a lehető legprecízebben ábrázolni. A magasabb bitszám jobb minőséget jelentene, és bár hangról van szó, ahol ez fontos, nem erre optimalizál az architektúra. A fixpontos számábrázolásnak két oka van. Egyrészt ezzel lehet minimálisra csökkenteni a szorzásokkal járó memóriaigényt, másrészről pedig ilyen módon a rendszer FPGA-ra is átvihető, amelyről a Skálázhatóság, továbbfejlesztés fejezetnél írok.

Ha a számábrázolás is tisztázódott, utána jön a szűrőtervezés. A szűrőtervezés önmagában felérne egy diplomamunkával, de mivel nem ez volt a fókuszban, ezért erre a LabVIEW szűrőtervezési algoritmusait használtuk, erről részletesebben az Implementáció fejezetben lesz szó.



22. ábra Az adatfeldolgozás blokkvázlata

Az adatfeldolgozás nagy részét már tisztáztam, ezért fontos, hogy ezen a ponton lássuk az egészet. Az adatgyűjtéssel foglalkozó blokk IQ adatot szolgáltat az adatfeldolgozással foglalkozó blokknak. Az IQ adatot az előbb ismertetett módon ráengedjük a DDC-re, utána pedig szűrjük. A szűrés után az I és Q adatot komplex jellé alakítjuk vissza, ahol I a valós, Q a képzetes rész. A szűrő tehát komplex jelek tömbjét adja tovább a demodulációs blokknak. A tömb elemszáma attól függ, hogy a felhasználó egyszerre hány csatornát

hallgat. A jelek demodulációja matematikai szinten kifejtve már szerepelt (6. oldal), úgy-hogy ezt kell implementálni.

Demoduláció után a kimenet lehet a hangszóró vagy a tárolás. Előbbi esetben annyit kell csinálnunk, hogy összeadjuk azokat a csatornákat, amelyeket a felhasználó kijelöl a UI-on, míg utóbbinál szintén, annyi különbséggel, hogy itt külön fájlokba írjuk a külön csatornákat.

2.1.6. Spektrum ábrázolás

Logikailag a Back Endhez és az Adatfeldolgozás Loophoz tartozik, viszont ez teljesen különálló egység az előző pontban ismertetetthez képest. A spektrumanalizáláshoz ugyanis az az IQ adat kell, ami a DDC-nek is. Ezt az IQ adatot át kell alakítani a komplex jelalakjára. Ha ez megtörtént, akkor utána el kell végezni a számábrázolás és a valódi fizikai paraméterek összehangolását. Ehhez ki kell számolni, hogy 1 amplitúdó egység a USRP-ben mekkora feszültséget jelent, másrészről pedig meg kell állapítani a dBm és a dBV közötti matematikai kapcsolatot.

Az amplitúdó – Volt konverzió esetén meg kell határozni azt, hogy a maximális amplitúdó értékhez milyen feszültség érték tartozik, és utóbbit elosztani az előbbivel. A maximális amplitúdóérték I16 esetén 32767, a feszültség pedig szinuszos jelek esetén -1 és 1V között ingadozik.

A dBm és a dBV között az alábbi összefüggés áll fent:

$$P_{dBm} = 10 * \log \frac{10^{\frac{P_{dBV}}{10}}}{Z_0 * 10^{-3}}$$

Ebből a logaritmus azonosságait felhasználva, és Z_0 = 50 Ohm referenciaértéket behelyettesítve:

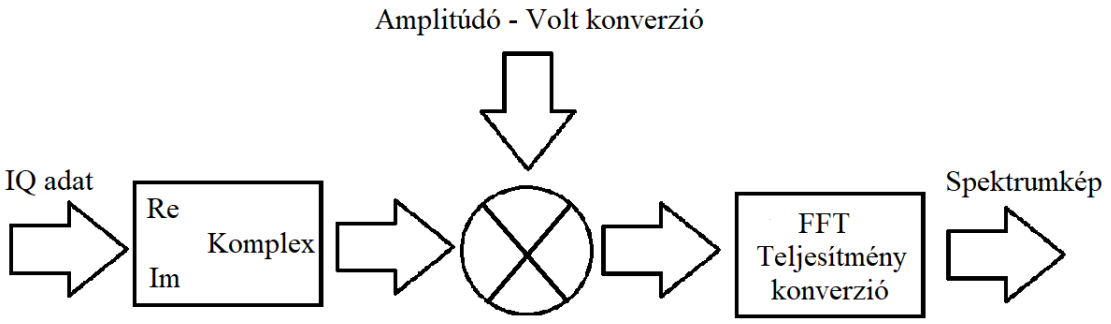
$$P_{dBm} = 10 * \log 10^{\frac{P_{dBV}}{10}} - 10 * \log 50 * 10^{-3} = 10 * \log 10^{\frac{P_{dBV}}{10}} - 13.01\ dB$$

Mindent átkonvertálva dB-re, azt kapjuk, hogy

$$dBm = dBV + 13dB$$

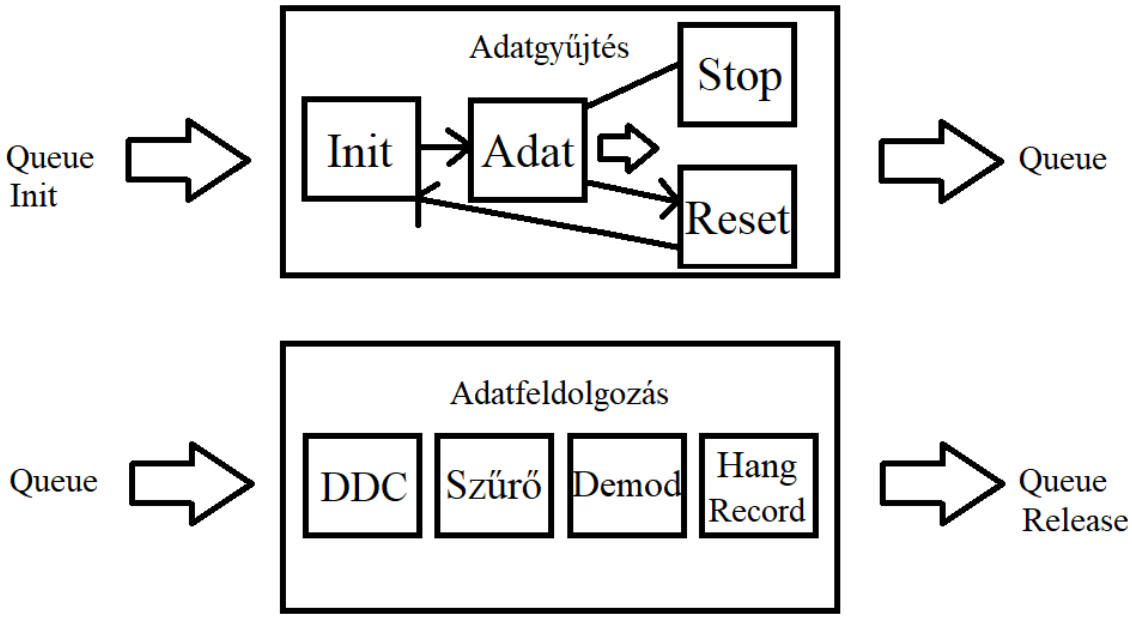
Tehát mivel dBm-ben akarjuk ábrázolni, nem pedig dBV-ben, ezért az Y-tengelyen kiszámolt értékeket 13 dB-vel meg kell növelni. Továbbá arra is figyelni kell, hogy a Gain

szintén változtat az eredeti jelnek a teljesítményén, ezért ezt le kell vonni. Így megkapjuk a sávszélességünk által meghatározott spektrumképet.



23. ábra Spektrumszámolás blokkvázlata

A megvalósítandó blokkdiagramot tartalmazza a 24. ábra.



24. ábra A teljes architektúra blokkvázlata

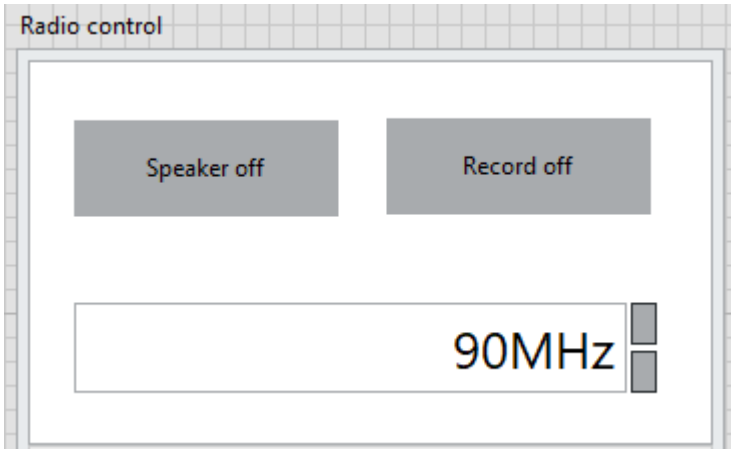
A tervezés után jöhet az implementáció. A tervezés hagy teret az implementációnak, hogy némi változás legyen az elképzelt és a megvalósítható architektúra között. Illetve lehetnek olyan részei az architektúrának, mely a tervezés során nem bukkant fel, de az implementáció során felbukkant.

2.2. Implementáció

Az előző fejezet során a tervezést taglaltam a funkcionalitástól kezdve a kisebb blokkokig. Ebben a fejezetben ennek a megvalósításáról lesz szó, kezdve a Front End-del, majd utána a Back End-el, és az előző fejezetben jellemző struktúra szerint. A megvalósítás során végig a LabVIEW 2018 Next Generation kinézetét használtam.

2.2.1. User Interface

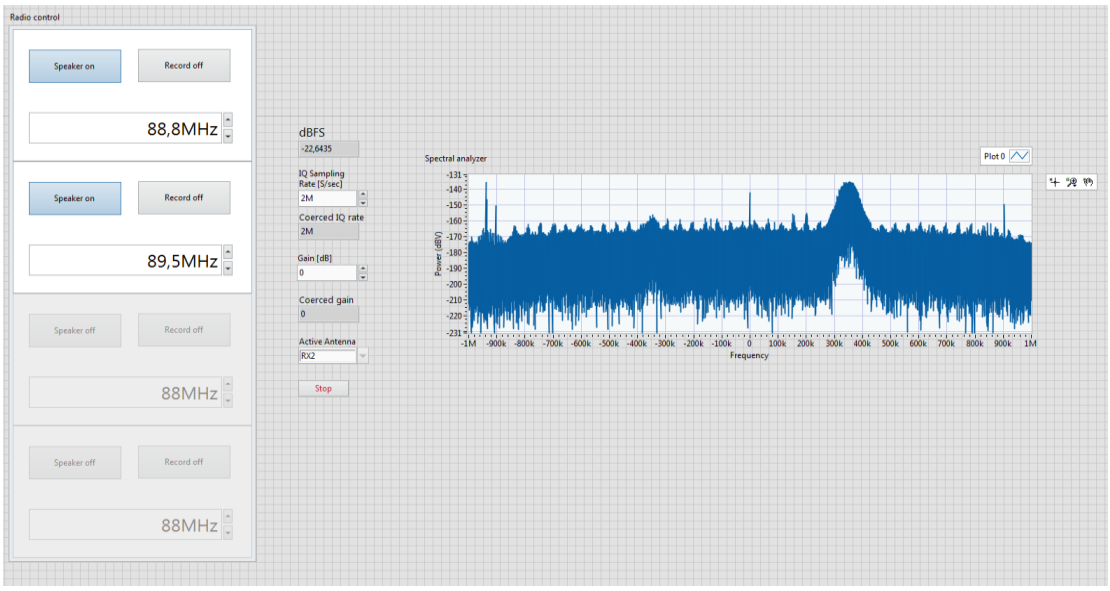
Az első lépés, hogy a UI tervet megvalósítsuk. Ehhez először meg kellett valósítani a csatornákat.



25. ábra Csatorna megvalósítás

Ez a legfontosabb eleme az UI-nak, ez a csatorna megvalósítása. Az eredeti tervtől egy kicsit eltér, de funkcióban ugyanazt nyújtja. Középen be lehet állítani a kívánt frekvenciát. Ez 88 MHz – 108 MHz között van, mást nem tud a felhasználó beírni. Mivel a rádióadók egy tizedes pontosságon mennek, ezért a csúszkával 0,1 MHz-et lehet lépkedni. Ha a felhasználó ennél pontosabbat akar beütni, akkor a legközelebbi tizedre fog kerekítődni az érték. Emellett megtalálható a hangkártya és a felvétel gomb is, ami mind felirattal, mind színnel jelzi, hogy éppen aktív-e a szolgáltatás.

A csatornát Radio Control.ctl typedef-ként mentettem el. A typedef a LabVIEW-n belül egy olyan sablon, amit használhatunk több helyen is, és ha változtatunk rajta, akkor minden érintett helyen változik. A Radio Control egy 1 dimenziós tömb, aminek az elemei a csatornákból állnak. A csatornák pedig clusterek, amik tartalmazzák a frekvenciát állító numerikus irányítót, illetve a 2 gombot.



26. ábra Az elkészült UI

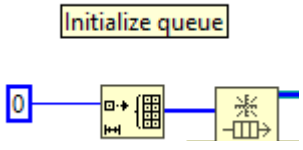
Ezen az ábrán a teljesen elkészült UI található. Az ábrán megtalálhatóak mind az aktív, mind a passzív csatornák, a spektrumnak a grafikonja, a sávszélesség irányító csúszkája, az antenna választó, illetve még kettő paraméter, ami az USRP működéséből származtatható, a Coerced Gain, azaz a kerekített erősítés, illetve az aktuális dBFS-nek az indikátora. Utóbbi teljesen újonnan felbukkant paraméter, amiről a későbbiekben lesz szó. (43. oldal)

A tervben eredetileg az szerepelt, hogy nincsen dedikált STOP gomb a megállításra, hiszen egy rádiót sem így használunk, hanem aki végzett a program használatával, az rámegy az ESC gombra, ott jön egy ellenőrző panel, majd, ha itt a felhasználó a Close, azaz Bezár gombot választja, akkor a program leállítja magát, majd pedig a LabVIEW bezár. Viszont a továbbfejlesztés érdekében egyelőre megmaradt.

2.2.2. Back end és queue

Ettől a résztől kezdve a Back End megvalósítással foglalkozunk. A tervezés szerint meg kell valósítani egy Queue-t, egy Adatgyűtjő, illetve egy Adatfeldolgozó Loopot, meg ezeken belül a kisebb blokkokat. Haladjunk ebben a sorrendben.

A Queue-t először inicializálni kellett. Ehhez először inicializálni kellett egy I16-okból álló tömböt, hiszen ezt az adattípust kell továbbítania a későbbiek folyamán. Majd pedig ezt kell rákötni az Obtain Queue VI-ra. Ha minden jól ment, utána egy érvényes Queue referenciát ad vissza. Utána ezt a



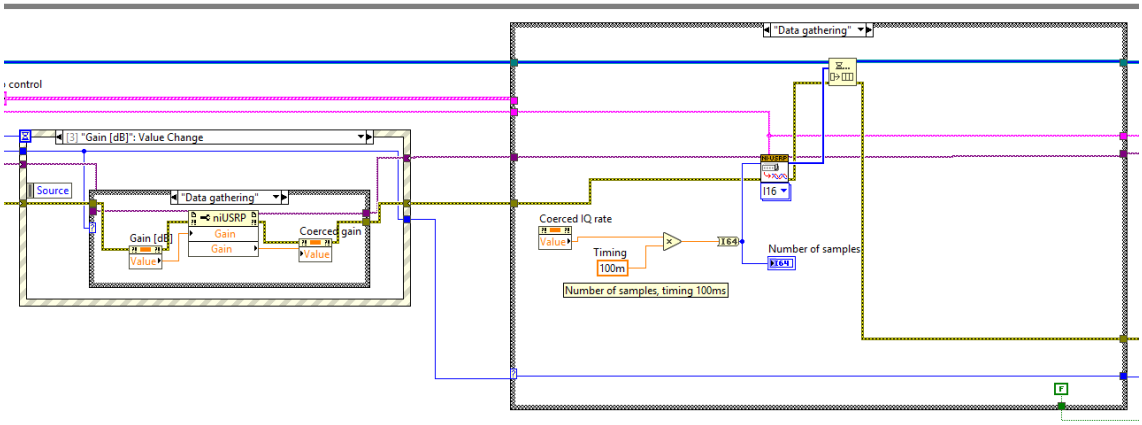
27. ábra Queue megnyitása

referenciát adja át az Enqueue Element VI-nak az Adatgyűjtés Loop-on belül, ami a begyűjtött USRP adatokat berakja a Queue-ba. Majd ezen referencia szerint működik a Dequeue Element VI is, ami visszaadja FIFO szerint az Enqueue-ba került elemeket, ezzel működik az Adatfeldolgozás Loop. A Queue-t meg kell szüntetni abban az esetben, ha bezárjuk a programot, ne maradjon a memóriában szemét. Mindkét Loop végére tettem egy Release Queue-t gondolva a hibakezelés eshetőségére. Az alapvető megállási folyamat az, hogy megáll az Adatgyűjtő Loop, így az Adatfeldolgozó Loop hibát dob, és ezért megáll. Ezt a hibát a Release Queue lenyeli, és megszünteti a referenciát. Ha viszont valamilyen nem várt hiba miatt áll le a program, ami az adatgyűjtés során következik be, akkor már ott helyben törölje a referenciát.

2.2.3. Állapotgép

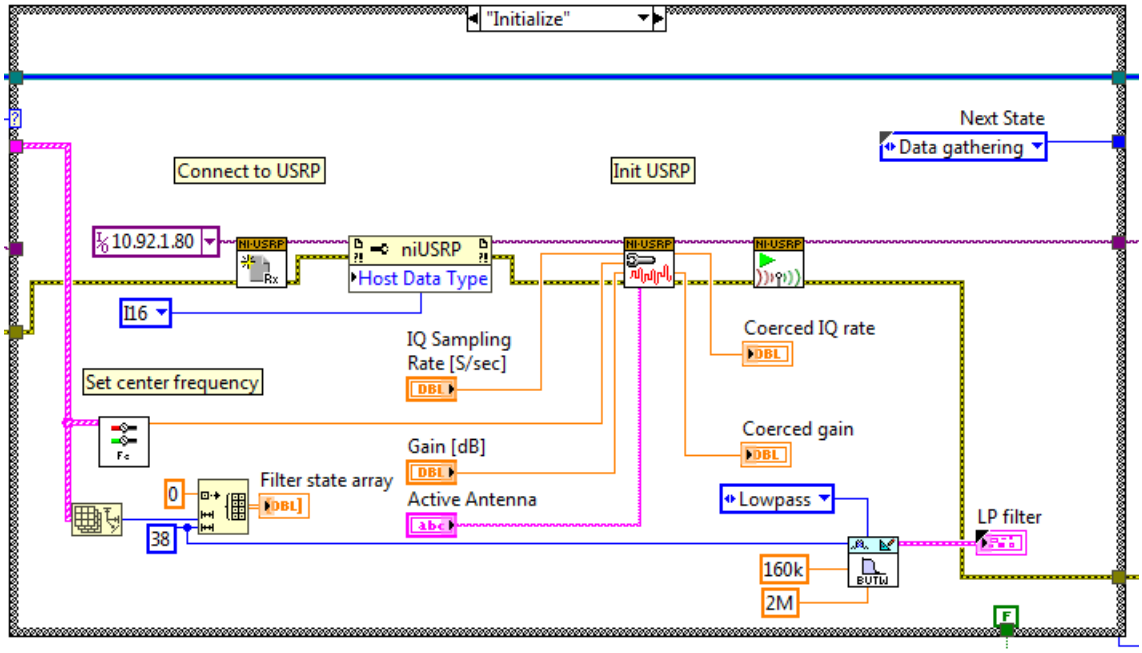
A következő részben az Adatfeldolgozás Loop megvalósítása kerül górcső alá. Ez a Loop egy While ciklus, amin belül pedig két fő rész található:

- Event Structure:
  A bekövetkezett események kezelésére szolgál, és ami alapján ki lesz választva, hogy melyik állapotban működjön a program
- Case Structure:
  Ez az állapotgép. Rendelkezik egy Case Selectorral, ami kiválasztja, hogy melyik állapot legyen éppen. Ezt vezérli az Event Structure



28. ábra Event Structure (bal oldal) és Case Structure (jobb oldal)

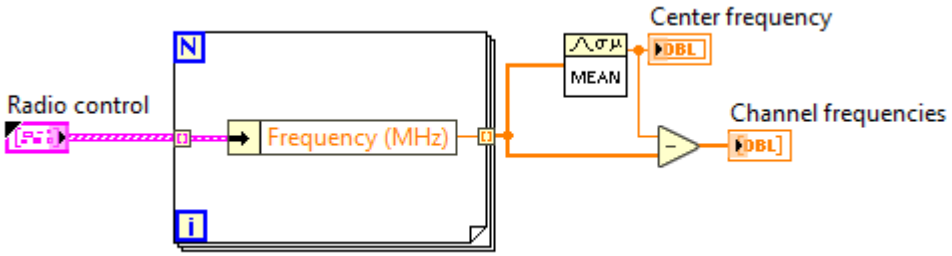
A Case Structure esetében a tervezésnél felvázolt állapotgépet kell létrehozni.



29. ábra Az inicializálás állapot megvalósítása

Elsőként létre kell hozni egy sessiont az Open Rx VI segítségével. Az IP cím egy statikus paraméter, az USRP jelenleg ezen a címen található meg az NI belső hálózatán. Megnyitás után be kell állítani, hogy I16-os adatokkal dolgozzon az USRP, ezt szolgálja a Property Node. Utána jön a Configure Signal VI, amelyen az IQ rate és az Active Antenna kikerül a Front Panelra, mint változtatható paraméterek. A Gain is Control, viszont azt más fogja irányítani, erről később lesz szó. (43. oldal)

A Center Frequency vagy más néven Carrier Frequency (f_c) paramétere érdekes, hiszen mindenképpen szükség lesz még keverésre több csatorna esetén. Viszont a többi keverés attól is függ, hogy pontosan mennyi az f_c. Ezt a feladatot oldja meg a Frequency Control VI, amelyet a 30. ábra mutat be.

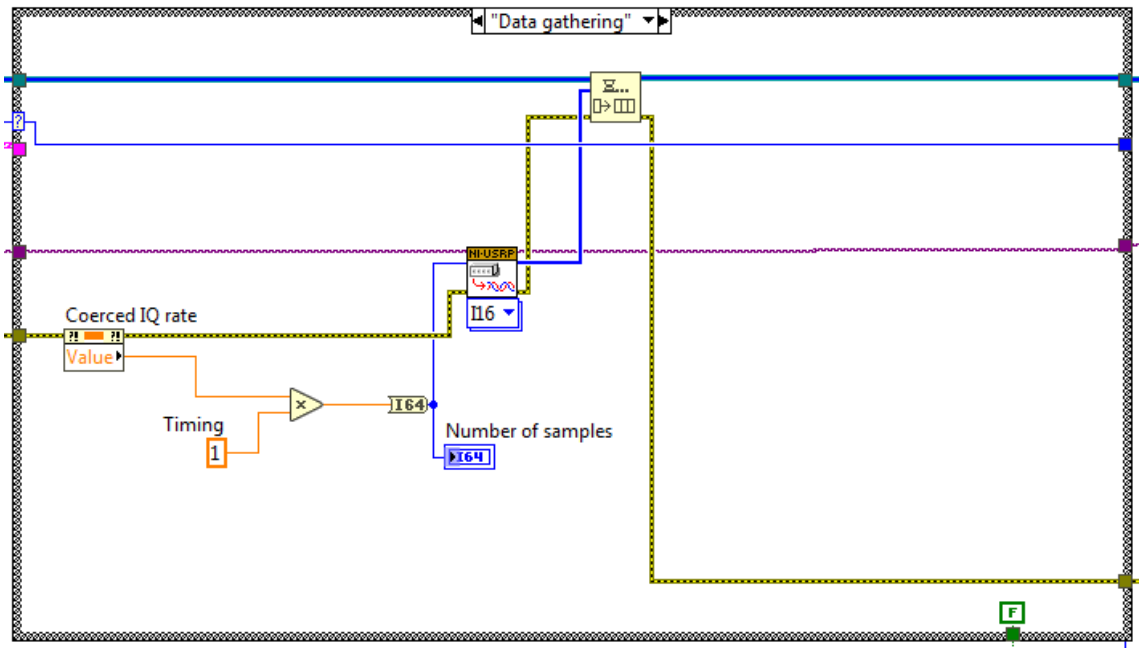


30. ábra Frequency Control VI

Az f_c-t beállítja a csatornafrekvenciák átlagára, és ezt kiadja kimenetnek, illetve mindegyik csatornafrekvenciából kivonja az f_c-t, ezáltal beállíthatjuk azt, hogyha az USRP elvégzi a lekeverést az f_c-re, akkor utána a DDC-ben pontosan mennyit kell keverni ahhoz, hogy az egyes csatornákat specifikusan adja vissza. Ezért adja ki a kimenetén a frekvenciatömböt, ami csatornánként ezeket a különbségi frekvenciákat tárolja.

Ennek megfelelően a csatornák átlagos frekvenciáját adjuk át az USRP-nek, ami az IQ rate-nek megfelelő nagyságú frekvenciasávot fog visszaadni, aminek a közepe az f_c.

Ezek után a Configure Signal Vi visszaadja a valóságos értékeit az előzőnek beállított paramétereknek. Van még egy szűrő is az inicializálási állapotban, erről a DDC és Testing DDC fejezetben lesz szó. Mivel az inicializálás állapot után mindig az adatgyűjtés következik, ezért ezt nem kell lekezelni az Event Structure-ön belül.



31. ábra Adatgyűjtés megvalósítása

A másik jelentős állapot az adatgyűjtés állapota, a megvalósítása nem bonyolult. A Fetch Rx Data VI megkapja a session-t, és a gyűjtött I16-os adatot átadja az Enqueue Element VI-nak. A minták száma paramétert meg kell határozni, ez lényegében egy időzítés az IQ ratenek köszönhetően. A LabVIEW hatékonyan működik 100ms időzítés esetén.

Ezen kívül még maradt a Reset meg a Stop állapot. Mindkettő esetében meghívjuk az Abort és a Close Session VI-t. Annyi a különbség, hogy előbbi állapotot mindig az inicializálási állapot követi, utóbbit meg semmi, illetve az, hogy a Stop állapotnál a Loop

Conditionra (ami megállítja a While Loopot) True logikai változót kötünk, míg a többi állapotban False-t.

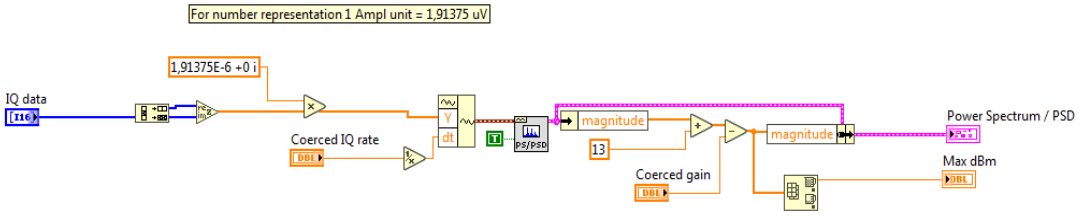
Az Event Structure az, ami kezeli az eseményeket, melyek az alábbiak:

- Timeout:
  Igény esetén be lehetne állítani, hogy mennyit várjon egy állapotban a rendszer, és ha timeout-ra fut, akkor egy másik állapotba lép. Én ezt nem használtam a feladatomban, nem volt rá szükség
- Panel Close?:
  Ez kezeli le azt, ha a felhasználó az Esc-et választja, vagyis be akarja zárni a programot. Ekkor felugrik egy ablak, ami megkérdezi a felhasználót, hogy tényleg be akarja-e zárni a programot (vagy csak véletlen volt az odakattintás), és ha igent mond, akkor a rendszert átlépteti a Stop állapotba
- IQ rate vagy Radio Control Change:
  Ez az esemény takarja az IQ rate változást, illetve csatornák hozzáadását, törlését, frekvenciaváltozását, tárolását és hangkártyára való kijátszását. Bármelyik is következik be, a Reset állapotba kell átléptetni a rendszert, ami onnan továbblép az Init állapotba
- Gain Change:
  Szerencsére ennek a változásához nem kell újrainicializálni az USRP-t a Configure VI segítségével. Ezen az eseményen belül Property Node segítségével állítjuk be az új Gain értéket

Az Event Structure ezzel teljes, minden eseményt lefedtünk, ami a rendszerrel történhet. Ezzel az Adatgyűjtés Loop ismertetése a szűrőn meg a hibakezelésen kívül megtörtént, ezek a DDC és Testing DDC fejezetben kerülnek meghatározásra.

2.2.4. Spektrumanalizátor és AGC

A Queue és az Adatgyűjtés Loop után az Adatfeldolgozás Loop-al folytatjuk. Ezen belül pedig a különálló spektrumanalizátort és az AGC-t (Automatic Gain Control), ami viszont nem került elő a tervezési fázis során.



32. ábra Spektrumanalizátor megvalósítása

A megvalósítás az alábbi folyamat szerint történik:

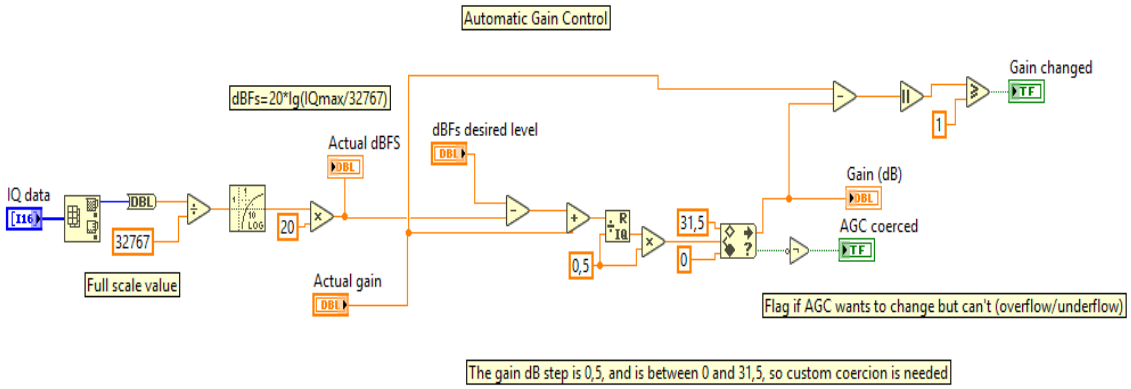
1. Az IQ adatot komplex adattá alakítjuk
2. Megszorozzuk a komplex adatot a konverziós számmal
3. Az adat és az IQ rate alapján előállítjuk a komplex Waveform-ot
4. Ennek a Waveformnak kiszámítjuk a Fourier-transzformáltját az FFT Power Spectrum and PSD VI segítségével
5. Ez a VI dBV-ben adja meg az egyes frekvenciák jelteljesítményét, amihez először hozzáadjuk a 13 dB-s konstanst, majd levonjuk az aktuális erősítést
6. A kapott adatokat ábrázoljuk grafikonon, ahol az X tengely a frekvencia, az Y-tengely pedig dBm

Ezen a ponton került elő az, hogy a felhasználó milyen alapon állítsa be magának az erősítést? Ha látja a spektrumképen, hogy van egy jel, de kicsi a jelszintje, akkor kézzel ugyan állíthatja, de ez nem kényelmes, cserében az is lehet, hogyha túl magasra állítja, akkor túlvezéreli a rendszert, és hibás számadatokat kap vissza. Vagy alapból a rendszer túlvezérlésből indul. Ennek elkerülése és a felhasználói élmény növelése végett tudunk olyan alprogramot létrehozni, aminek az a dolga, hogy automatikus állítsa be a Gain-t az optimálisra. Ha pedig erre képtelen, mert nem tudja csökkenteni vagy tovább növelni a Gain-t, akkor erre adjon egy jelzést. Ilyen egység az AGC, ami automatikusan állítja a Gain-t, csupán meg kell neki adni, hogy milyen feltételt kell vizsgálnia. Alapvetően a kivezérlés a legnagyobb probléma, amiért erre szükség van, ezért ilyen feltételt kell meghatározni.

A kivezérlés legjobb indikátora a dBFS, ami kifejezi a jelteljesítmény és a kivezérlés viszonyát dB-ben.[17] A 0 dBFS jelenti azt, hogy a rendszer elérte a csúcsértéket. A pozitív

[17] Forrás: S. Haykin, *Communication Systems*

dBFS már túlvezérlést jelöl. Számunkra célszerű az, hogyha kicsivel a csúcsérték alá tervezünk. Éppen ezért lett a dBFS = -5 kritérium az AGC számára (ez azonban állítható paraméter)



33. ábra AGC megvalósítása

A dBFS-es képletbe foglalva:

$$dBFS = 20 * \lg(\frac{IQ}{32767})$$

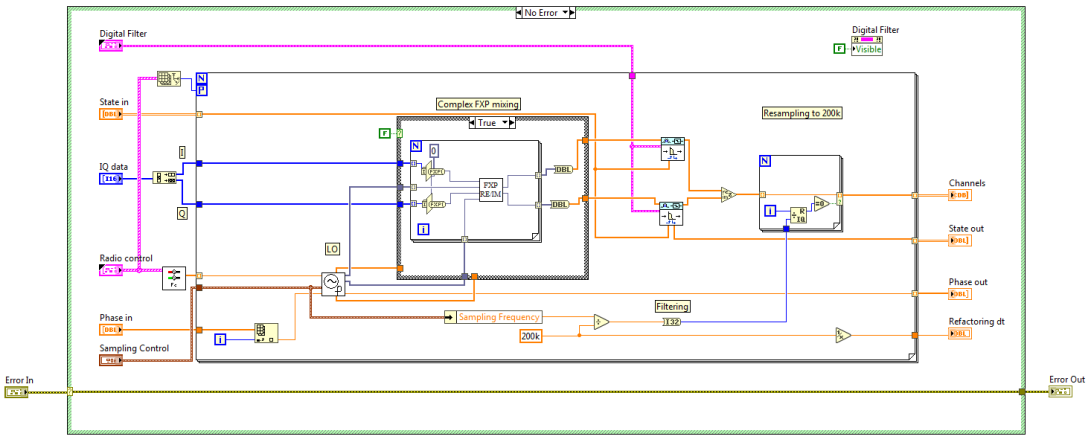
Ennek a csúcsértékét akkor kapjuk meg, ha vesszük az IQ tömbnek a maximális értékét, ekkor a dBFS értéke 0.

Ez a képlet került leprogramozásra az első részben. Az elején VI segítségével kinyerjük az IQ maxot, majd pedig elvégezzük a matematikai műveletet. Utána összehasonlítjuk a jelenlegi dBFS-t az elvárt szinttel, és ezt a különbséget hozzáadjuk a jelenlegi Gain-hez, hogy megnézzük, mennyit kell változnia annak érdekében, hogy a megfelelő szintet kapjuk. Ez lényegében egy szabályozási kör, ahol visszacsatolás által érjük el a kívánt Gain szintet. Utána egy kerekítést végző programkód következik. A LabVIEW nem kínál megoldást egyéni kerekítésre, márpedig az USRP erősítését 0,5 dB pontosan lehet megadni 0 és 31,5 dB között. Ezért kell az elvárt Gain-t 0,5-re kerekítve visszaadni. A flagek (jelzőbitek) pedig azt adják vissza, ha a Gain változott, illetve, hogy az AGC elért-e egy határt, amin túl már nem tud mit csinálni (alulcsordulás/túlcsordulás). Ebben az esetben jelzi, hogy addig nem tud javítani a felhasználáson, ameddig valamilyen paraméter nem változik meg.

Az AGC tehát egy olyan eleme a programnak, ami a tervezésben nem szerepelt, a spektrum implementáció közben jött elő a fontossága. Azonban hasznos a későbbiekben is, hiszen ilyen módon a gyengébb jeleket a rendszer automatikusan erősíteni fogja, vagy gyengíteni, ha hirtelen megváltozik a jelszint, és feleslegesen sokat erősít rajta az USRP.

2.2.5. DDC és Testing DDC

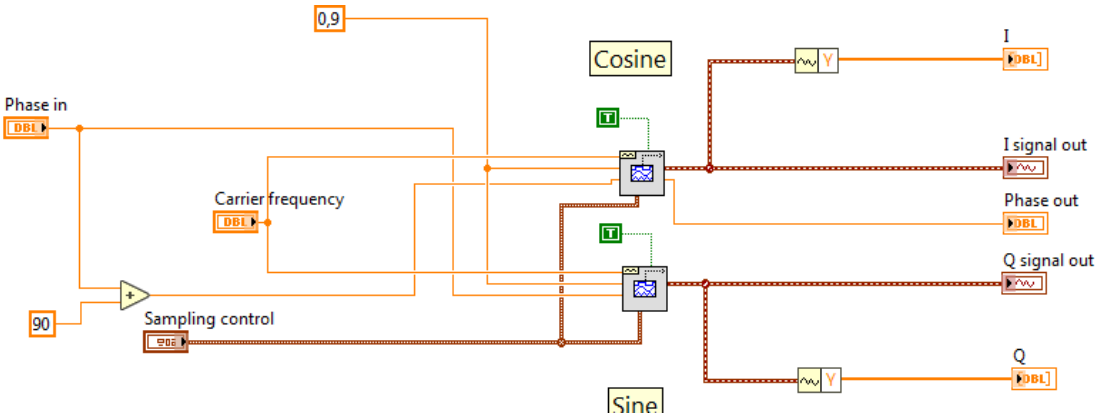
Az Adatfeldolgozás Loop különálló része után tekintsük át az egybefüggő részt, azon belül is a legkritikusabb és legösszetettebb pontját, a DDC-t.



34. ábra DDC és szűrés megvalósítása

Az ábrán látható, hogy egy Error Case Structure-ben található a DDC. Ennek az az oka, hogyha valamilyen error-t jelez a rendszer, akkor az álljon meg, és ez a blokk ne fusson le, mert rengeteg időt felemészt. Ezen belül a DDC áll egy Conditional Terminal-lal ellátott For Loop-ból, ahol az elemszámot az aktív csatornák száma határozza meg, és ezzel ekvivalensen a használt processzor magok számát is. Megkapja a Radio Control-t is, ahol az egyes csatornákhoz tartozó frekvenciákat (amit a Frequency Control kiszámolt) a megfelelő iterációban adja át. Ezen kívül van egy fázistömb is, ami mindig a megfelelő, elmentett fázist adja oda a további részeknek. A Sampling Control magában foglalja a mintavételi frekvenciát és a mintapontok számát. A Szűrőről a következő fejezetben lesz szó.

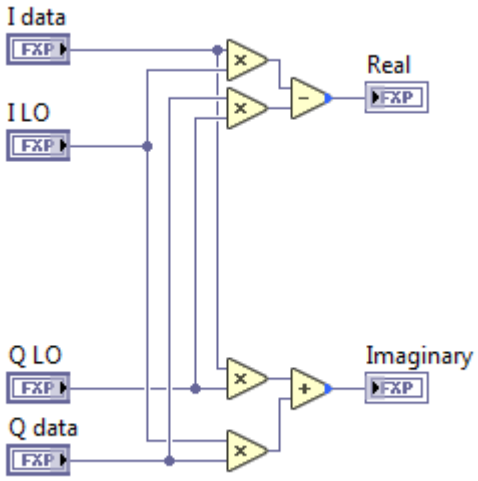
Ezen paraméterek után annyi a dolga ennek a blokknak, hogy a beérkező IQ adatot felbontja I tömbre és Q tömbre, majd pedig megszorozza őket az LO-val. Az LO-ról már beszéltünk (6. oldal) ,egy aspektusát még fontos kiemelni, a fázist.



35. ábra Az LO és a fázihelyzet

A lokáloszcillátornak fázisfolytonosnak kell lennie az egyes iterációk során, ezért a fázist minden iteráció után el kell menteni, majd pedig átadni a következőnek.

Maga a szorzás már fixpontos számokkal történt. A végeredmény a szorzás után 32 biten ábrázolt, maximum értéke pedig 3. A szorzás után pedig meg kell szűrni az I meg a Q jelet is. Fontos, hogy teljes mértékben azonos paraméterű szűrőt kell használni mindkét esetben. Itt vissza kellett térnünk a lebegőpontos számábrázolásra, ugyanis a LabVIEW nem kínál fel fixpontos szűrési lehetőséget. A szűrőtervezés nem volt benne a projekt és a szakdolgozat fókuszában, így ebből kifolyólag fixpontra sem. Ennek megfelelően először átkonvertálással próbálkoztam, és hogy annak megfelelően működni fog a szűrés, de sajnos végül nem szólalt meg a rádió, míg abban az esetben, ha a keverést lebegőpontos számokkal végeztem, működött. A keverésről már korábban beszéltem. (32. oldal)



36. ábra A keverés fix pont szerint

A keverést ebben a formában valósítottam meg fixpont szerint. Miután ez nem működött, annyit változtattam, hogy fix pont helyett az LO dupla-pontosságú lebegőpontos számokat adott, míg az adat I16-os formában érkezett. Hogy ez milyen változásokat okoz, arról a Skálázhatóság, továbbfejlesztés részben fogok beszélni.

A szűrés tervezésre és végrehajtásra beépített VI-okat használtam. A Classical Filter Express VI segítségével meghatároztam, hogy a szűrőnek a kívánt specifikáció szerint (160 kHz-es Butterworth aluláteresztő szűrő, ami 200 kHz-ig 60 dB elnyomást hajt végre a jelen) milyen rendű IIR szűrő szükséges. Mint ahogyan a fázisnál is fontos volt, itt is el kell menteni a korábbi állapotot, hogy a megfelelő helyen folytassa a szűrő a munkát az új mintapontok eseténél is. Majd pedig végre kell hajtani a szűrést egy VI segítségével.

Ennek megfelelően az inicializálásnál létre kell hozni a szűrőt, illetve egy tömböt, ami a szűrő állapotait elmenti a későbbiek során. Utána ezt a tömböt és a szűrőt is át kell adni az Adatfeldolgozás Loop-nak. A szűrő átadásával a DDC-ben meghívjuk a szűrés végrehajtás VI-t, ami elvégzi a szűrést ennek megfelelően. Ami meg a tömböt illeti, ott figyelnünk kell arra, hogyha egy új rádiócsatorna érkezik, annak újabb állapotai lesznek, amit szintén tárolni kell. Erre készült egy logika, amit egy subVI hajt végre. A program első futásánál az inicializált tömböt adja vissza, amúgy meg utána mindig az előzőleg tárolt állapotot kapja meg. Ha viszont érkezett egy új csatorna, akkor az előző kétdimenziós tömbbe (ahol az egyik dimenzió a csatornák száma, a másik pedig a csatornákhoz tartozó állapotok tömbje) be kell szúrni az új csatorna tömbjét.

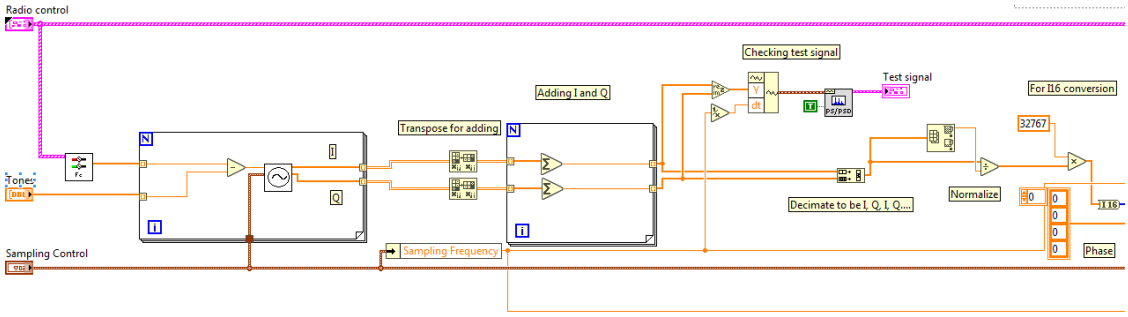
A szűrés után a jelet visszaalakítjuk a komplex alakjába. Utána jöhet a demoduláció.

Végig fontos szempont volt, hogy a számításokat minimalizáljuk annak érdekében, hogy a lehető legtöbb csatornát tudjuk kinyerni. Éppen ezért a demodulátornak ne adjunk nagyobb spektrumot csatornánként, mint amire szüksége van. Az FM műsorszórás alapján ez 200 kHz-nek felel meg. (13. oldal) Ezért ezen a ponton a mintavételi frekvenciát el kell osztani 200 kHz-cel, ezt az értéket kerekíteni, majd pedig egy olyan For Loopot kell létrehozni, ami csak azokat a mintapontokat menti el, ami az előbb kiszámolt értéknek egész számú többszöröse.

Minden egyes blokknál fontos a modularitás, és az önmagában való tesztelhetőség. Ennél a blokknál ez hatványozottan igaz, mert nagyon összetett, és sok számítást végez el. A főprogramkódból szinte lehetetlen lesz kihámozni, ha valami probléma van, annyi minden függ össze egymással. Ez hívta életre a Testing DDC VI-t.

A Testing DDC VI-jal tulajdonképpen annyit csinálunk, hogy szimulálunk egy olyan spektrumképet, amit az USRP ad vissza, és az ezt alkotó jeleket egy picit elhangoljuk arról a frekvenciáról, amit a Radio Control-on belül megadunk neki. A DDC maga úgy működik, hogy minden csatornát lekever az alapsávba, és minden, a Radio Controlon megadott frekvenciát vivőfrekvenciaként kezel, tehát a lekeverés során ez lesz a 0 frekvencia. Ha egy picit félrehangoljuk, akkor szintén ez történik meg, csak látszik benne a félrehangolás, és hogy a megfelelő frekvenciákat szoroztuk meg egymással.

Első lépésként elő kell állítani egy szimulált USRP jelet, ehhez azonban ismerni kell, hogy az USRP időtartományban nézve milyen jelet ad nekünk át. Annyit csinál az USRP, hogy a középfrekvenciától +/- $\frac{B}{2}$ sávszélességben veszi a mintapontok általi felbontás szerinti frekvenciákon lévő jeleket, és ezeket összeadja.[18] Ezt megteheti, ugyanis, ha Fourier – sorba fejtük az összeg jelet, akkor visszakapjuk a különböző frekvenciájú szinuszos jeleket, amivel pedig a spektrumképet. Ezt kell tehát megvalósítani.

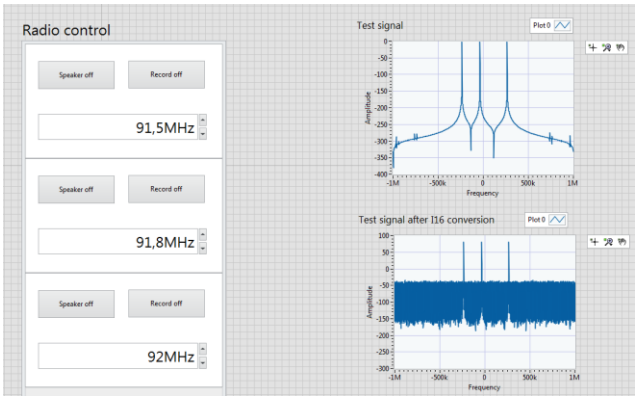


37. ábra Testing DDC, USRP jelszimulálás

A 37. ábra látható, hogy első lépésként vesszük a Radio Control-t, és kinyerjük belőle a felhasználó által beállított frekvenciákat. Ezen frekvenciákat hangoljuk el a Tone-ok segítségével a For Loopon belül. Ha az így kapott frekvencia megvan, akkor használjuk az LO-t, hiszen az I és Q jelet ad vissza. Ezután visszakapjuk csatornánként az I és Q jeltömbünket. Ezek kétdimenziós tömbök, ahol az egyik dimenzió a csatornák számát jelöli, míg a másik az I vagy a Q mintapontokat. Ezen tömböket kell összeadni minden csatorna szerint úgy, hogy minden mintapontot a vele azonos sorszámon elhelyezkedő mintaponttal kell összeadni. A LabVIEW adatábrázolása alapján először transzponálnunk kell a mátrixunkat, és utána mehet az összeadás.

[18] Forrás: S. Haykin, *Communication Systems*

Az összeadás után van egy ellenőrzés, hogy tényleg a megfelelő eredményeket kaptuk-e. Az alábbi ábrán látható az eredménye:



38. ábra Teszt jel

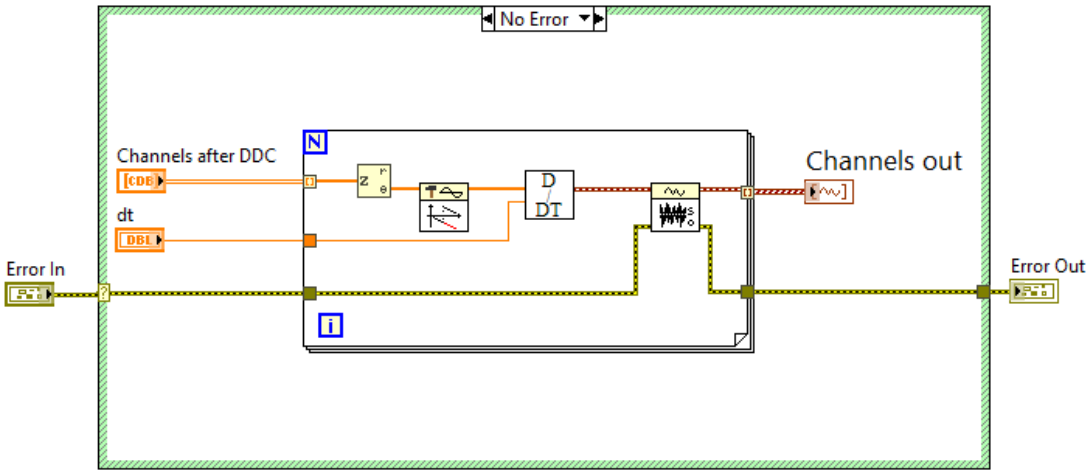
Ezek után megkaptunk egy megfelelő I és Q tömböt. Ezt kell úgy átadni a programnak, ahogyan az USRP adagolja az I16 esetén. Ezért kell első lépésként decimálni, és I, Q, I, Q… szerint összefűzni, és csinálni egy tömböt. Utána ezt a tömböt normalizálni kell, hogy -1 és +1 között vegye fel az értékeket. Ehhez megkeressük a maximális értéket a tömbben, és leosztunk vele, így az lesz az 1 érték. Aztán következik az I16 adattípusra való áttérés, ami miatt szorzunk az I16-ban előforduló legnagyobb értékkel. Végül megcsináljuk a dupla pontosságú lebegőpontos számról az I16-ra való konverziót.

Ezzel lényegében lefedtük az USRP jelszimulálását. A DDC által kért egyéb paramétereket könnyen meg lehet adni.

Mivel a DDC nagyon sokat számol, ezért, ha már valamilyen hibaüzenet érkezik, ami miatt leáll a program, akkor a DDC ne fusson le, erre van az Error Case.

2.2.6. Demoduláció, hang kijátszás, tárolás

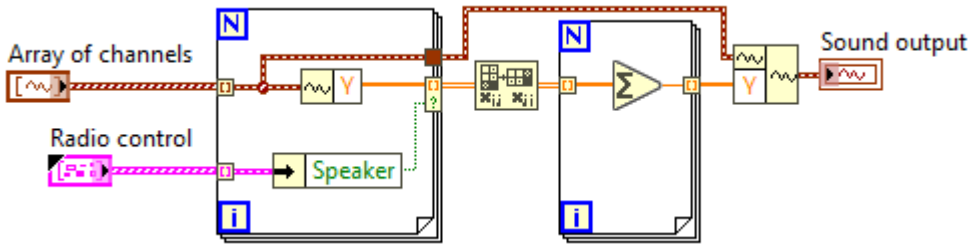
A sikeres DDC után a jeleket demodulálni kell. A séma adott hozzá, azt kell megvalósítani.



39. ábra Demoduláció

A csatornákat kapja meg, illetve az IQ rate által meghatározott dt-t. Először a komplex jelnek leválasztja a szög részét, majd azt az Unwrap Phase VI korrigálja, folytonossá téve a fázis átmenetet. Ha ez megvan, akkor deriváljuk a jeleket, a deriválás után hullámformában adjuk tovább. A hullámformát még normalizáljuk +1 és -1 szerint, végül kiadjuk a demodulált csatornákat Waveform formátumban tömb szerint. Ez a VI is sokat fut normál esetben, hiba esetén le kel állítani minél hamarabb, ezt a célt szolgálja az Error Cluster. Az Analóg FM rádiózásban részletesen kifejtem, hogy ennek miért ez a folyamata.

A demoduláció után következik a hangkártyára ültetés logikája, ami szerint eldöntjük, hogy melyik csatornát játsszuk ki, és melyiket nem.



40. ábra Hangkártya kijátszás

A 40. ábra szereplő VI annyit csinál, hogy megkapja, hogy melyik csatornákat kell kijátszania, és ezeket választja ki az első For Loop-ban. A tömbműveletek matematikája és ábrázolása miatt ezeket transzponálni kell, majd pedig a tömb elemeit összeadni. Ezzel a megoldással lehet egyszerre csak 1 csatornát hallgatni, de lehet egyszerre többet is összeg formájában.

Ezek után újramintavételezésre van szükség, mivel a hangeszközök 44,1 kHz mintavételi frekvenciával működnek. Erre a LabVIEW szintén kínál beépített megoldást, a Resample Waveforms VI-t, ami annyit tesz, hogy a mintákat a kívánt dt mintavételi idővel rendelkező mintasorozattá alakítja. Majd miután ez megtörtént, meg kell hívnunk még egy VI-t, ami ki is játssza a hangot.

A tárolás megvalósítására végül nem került sor, erről a Skálázhatóság, továbbfejlesztés részben fogok értekezni.

3. Összefoglalás, skálázhatóság, továbbfejlesztés

Ez a fejezet értékeli a megépített architektúrát, összevetve az elején specifikált elvárásokkal, utána kitér a skálázhatóság és a további fejleszthetőség kérdéskörére is.

3.1. Megvalósult funkcionalitások

A megvalósításra szánt funkciók nagy része működőképes, csak kérdéses, hogy milyen sebességgel, és hogy ezt hogyan lehet feljavítani. Térjünk ki a konkrét eredményekre, majd utána a funkciókra és a megvalósulásukra lépésről lépésre:

- Eredmények:
  Nehéz számszerűsíteni, hogy pontosan mire képes az architektúra. Az USRP-nek szüksége van egy jó minőségű, a 88 – 108 MHz sávszélességre hangolt antennára is, hogy a gyengébb csatornákat is tisztán, kevésbé zajosan lehessen hallgatni. Sajnos vannak csatornák, amelyeknek kicsi a jelteljesítménye, és ahhoz, hogy több csatornát hallgassunk, több MHz sávszélességre van szükség. Jelenleg egy, kettő és három csatorna hallgatása esetén kb. 5 MHz a sávszélesség, ahol még értelmezhető a rádiócsatorna által közvetített információ, és nem szakadt meg az adatfolyam, és a negyedik csatornánál ez az érték lecsökkent 4,6 MHz-re. Ez azt jelenti, hogy nem a csatornák száma, hanem a sávszélesség az architektúra gyenge pontja. Emellett ki kell emelni, hogy az újramintavételezés az IQ rate alapján történik, ami viszont nem vehet fel tetszőleges értéket. Ebben az esetben pontosan ki kell számolni az újramintavételezést, mert a hangkártya akkor fogja jól kijátszani az adatokat, hogyha 44,1 kHz a mintavételi frekvencia. Hallható volt az, amikor az újramintavételezés bizonyos beállított sávszélesség esetén pontatlan volt
- Párhuzamos csatornahallgatás és rögzítés:
  Ez a funkció a fentiek következtében részlegesen valósult meg. A párhuzamos csatornahallgatás működőképes, a rögzítést megvalósítását áthelyeztem a továbbfejlesztések közé, mert vannak olyan továbbfejlesztések, melyek előbbre valók lehetnek.
- Dinamikus működés:
  Jelenleg a felhasználó egyszerre 1 paramétert tud megváltoztatni, utána csak akkor, ha a rendszer már az új paraméterek szerint üzemel. Ha változás érkezik az-

előtt, hogy beállt a rendszer, a session handler invalid lesz, és a program hibaüzenettel leállt. Le kell kezelni ezt az esetet azzal, hogy tiltva legyen a UI minden beavatkozás után arra a rövid időintervallumra

- Spektrum:

  A Spectral Analyzer VI ezért a funkcionalitásért felelős. Kirajzol egy dBm szerinti teljesítményspektrumot a teljes sávszélességre. A 0 frekvencia pedig a csatornafrekvenciák átlaga. Ezt a VI-t csak minden 10. iterációban futtatom le, hogy spóroljak a számítási kapacitáson, hiszen ez nem változik olyan dinamikusan, ami miatt gyakrabban lenne rá igény.

3.2. Skálázhatóság, továbbfejlesztés

Ebben a részben kitérek az eredmények elemzésére, és az ebből fakadó lehetőségekre, továbbfejlesztési irányzatokra.

A LabVIEW-ban számos adatelemző funkció található. Ilyen a profilozás is, ami megmutatja, hogy egy adott VI pontosan hányszor futott le, mennyi időbe telt egy lefutása, mennyi memóriát használt el működése során stb. Ebből meghatározható, hogy melyik VI-okat célszerű fejleszteni, cserélni.

3		VI Time	Sub VIs Time	Total Time	# Runs	Average
4	niUSRP Open Rx Session.vi	3606,4	0	3606,4	5	721,3
5	niUSRP Abort.vi	399,4	0	399,4	4	99,9
6	niUSRP Fetch Rx Data (I16).vi	34481,1	0	34481,1	346	99,7
7	Double mixing.vi	8154,9	0	8154,9	690	11,8
8	NI_MAPro.lvlib:ma_Two-sided Power Spectrum (CDB).vi	151,4	0	151,4	35	4,3
9	niUSRP Close Session.vi	14,4	0	14,4	4	3,6
10	NI_MAPro.lvlib:ma_Convert to dB.vi	70,1	0	70,1	35	2
11	Sound Output Configure.vi	1,8	0	1,8	1	1,8
12	niUSRP Configure Signal.vi	7,2	0	7,2	5	1,4
13	AGC.vi	30,8	0	30,8	35	0,9
14	NI_AALBase.lvlib:Sine Wave.vi	746	0	746	1382	0,5
15	Speaker.vi	166,5	0	166,5	345	0,5
16	NI_AALPro.lvlib:Scaled Time Domain Window (CDB).vi	15,5	0	15,5	35	0,4
17	Derivate_v2.vi	177	0	177	690	0,3
18	NI_DigFilter_Design.lvlib:dfd_Combine Single Orders.vi	1,5	0	1,5	5	0,3
19	niUSRP Initiate.vi	1,5	0	1,5	5	0,3
20	NI_DigFilterLicensed.lvlib:dfd_Read Filter Coefficients.vi	79,2	0	79,2	1380	0,1
21	NI_DigFilterLicensed.lvlib:dfd_SOS to TF.vi	0,5	0	0,5	5	0,1

41. ábra Profilozás alapján a legrosszabb mutatókkal rendelkező subVI-ok

A 41. ábra bal oldalán találhatóak a VI-ok, melyekhez minden sorban több adat is tartozik. Ezek közül jelenleg nekünk a Total Time, azaz a VI futásával töltött idő a fontos, a Runs, azaz hányszor futott le ezen idő alatt, és az Average, ami azt jelenti, hogy futásonként átlagosan mennyi időt futott. Fontos, hogy a SubVIs Time idő ezeknél 0, mert ez azt jelenti, hogy ténylegesen ezen VI-ok hajtódnak végre. A Total Time a VI Time és a SubVI Time összege.

Látható, hogy az USRP-t hardverszinten igénybe vevő VI-ok a leglassabbak, viszont szerencsére ezeket nem futtatjuk sokszor. Kivéve ez alól a Fetch Rx Data, de ennek a VI-nak a futási sebessége pontosan attól függ, hogy mennyi adatot szolgáltat, ezt mutatja be a 17. ábra tartozó összefüggés a mintapontok számát illetően. Plusz ez a VI is kötelező minden USRP-s alkalmazáshoz.

Utána következik a double mixing VI, ami foglalkozik a DDC-n belül a keveréssel. Erre terveztünk fixpontos számábrázolást, ami viszont nem működött a jelenlegi architektúrában a sok konverzió miatt, amit a fixpontos szűrők hiánya, illetve az, hogy az LO-n belül is konvertálnunk kell, mert a szinusz függvénygenerátor dupla-precizitású lebegőpontos számábrázolás szerint adja ki magából a jelet. Az LO-t egyébként mindenképpen javítani kell, mert a SineWave VI is ahhoz tartozik, és ennek a lefutása is az átlagosnál hosszabb.

Emellett megjelenik még a Spectral Analyzer VI-on belüli FFT-számoló VI is, meg az AGC. Ezeken azonban nem lehet spórolni azon kívül, amit már eddig is megtettünk, miszerint ritkábban futtatjuk le. A demodulációhoz tartozó deriválás is megjelenik, de ezen szintén nem lehet spórolni.

	A	B	C	D	E	F
3		VI Time	Sub VIs Time	Total Time	# Runs	Average
4	NI_DigFilterLicensed.lvlib:dfd_Filter Signal by Coef.vi	9604,3	85,5	9689,9	1380	7
5	DDC.vi	3368,2	19191,5	22559,8	345	9,8
6	Demodulation_v2.vi	1075,1	290	1365,1	345	3,1
7	Local Oscillator.vi	328,9	878,9	1207,8	690	0,5
8	Local Oscillator_v2.vi	127	1207,8	1334,8	690	0,2
9	Normalize Waveform.vi	111,8	1,2	113	690	0,2
10	NI_MAPro.lvlib:ma_resample single-shot (linear interpolation).vi	110,4	2,3	112,7	345	0,3
11	NI_MABase.lvlib:Basic Function Generator.vi	82,6	796,3	878,9	1380	0,1
12	Sound Output Write (DBL).vi	48	9,3	57,4	346	0,1
13	Spectral analyzer.vi	42,8	242,9	285,7	35	1,2

42. ábra Profilozás alapján a legrosszabb mutatóval jelentkező fő VI-ok

Érdemes még megnézni a főbb VI-okat (olyan VI, aminek van subVI-ja) aszerint, hogy melyik üzemel a legtöbbet önmagában, ezt mutatja a 42. ábra. Ehhez kivontam a Total Time-ból a Sub VIs Time-ot. Ezek közül szignifikánsan nagyobb értékkel jött ki a szűrő. Mivel ez egy beépített könyvtári függvény, ezért valószínű, hogy az adott célfeladatra van jobb is. A többi VI már előbukkant a subVI-jai miatt, különösképpen a DDC.

Az adatokra való tekintettel meghatározható, hogy mik a fejlesztési irányzatok. Mindenképpen fontos megtervezni a szűrőt, és az újramintavételezést is. Előbbi a számítások gyorsítása miatt fontos, utóbbi pedig azért, hogy jó minőségű hangmintákat tudjunk feldolgozni. Az előbbi miatt az egész architektúrát célszerű lenne FPGA-ra átvinni, hiszen

az FPGA pont az ilyen számításokra lett kifejlesztve, emellett pedig az NI fixpontos szűrőket csak FPGA-ra tervezett ugyanezen okból kifolyólag. És ha ez megtörtént, akkor célszerű megvalósítani a tárolást is, hogy nagy sebességen, sok csatornát, egyszerre lehessen rögzíteni.

Emellett az architektúrán még lehet fejleszteni. Ki kell küszöbölni a felhasználásból származó hibákat, amire példa volt a gyors változás miatt bekövetkező referencia hiba a dinamikus működés során. Az LO-t is lehet optimalizálni, hogy ne fusson le ilyen sokszor, és mégis megőrizze a fázisfolytonosságát.

Irodalomjegyzék

[1] Géher Károly, *Hiradástechnika* Műszaki Könyvkiadó Kft., Budapest, 2000. január 1.

[2] S. Haykin, *Communication Systems*, 3rd ed. New York: Wiley, 1994.

[3] Géza Kolumbán, Tamás István Krébesz, Francis C.M. Lau, *Theory and Application of Software Defined Electronics*, IEEE Circuits and System Magazine, 2012

[4] [Online] http://www.ni.com/pdf/manuals/375839c.pdf

[5] [Online] http://www.ni.com/pdf/manuals/376358a.pdf

[6] [Online] http://zone.ni.com/reference/en-XX/help/373380G-01/usrpviref/niusrp_fetch_rx_data_poly/#niUSRPFetchRxData(CDBWDT)

[7] [Online] http://zone.ni.com/reference/en-XX/help/373380J-01/usrpviref/niusrp_configure_signal/

[8] [Online] http://www.ni.com/white-paper/3023/en/

Függelék

1. ábra Rádiózás blokkvázlata 6
2. ábra IQ keverés blokkvázlata 9
3. ábra IQ lekeverés és szűrés blokkvázlata 10
4. ábra Szoftveres alapú rádiózás blokkvázlata 12
5. ábra Rádiócsatornák egy része Budapesten 14
6. ábra FM sugárzás alapsávi spektrumképe 15
7. ábra Front Panel vázlata 16
8. ábra A Block Diagram és a Front Panel kapcsolata 17
9. ábra LabVIEW NXG Style 18
10. ábra USRP Utility Config 18
11. ábra USRP kezelői felülete 19
12. ábra USRP blokkvázlat 19
13. ábra USRP 2920 rendszerjellemző paraméterei 20
14. ábra A tervezett UI 24
15. ábra Az USRP Rx fülhöz tartozó VI paletta 25
16. ábra A Configure Signal által beállított paraméterek 26
17. ábra a Fetch Rx Data VI paraméterei 27
18. ábra Producer/Consumer Loop architektúra 30
19. ábra Az állapotgép terve 31
20. ábra DDC blokkvázlata 33
21. ábra Lekeverés és szűrés 34
22. ábra Az adatfeldolgozás blokkvázlata 35
23. ábra Spektrumszámolás blokkvázlata 37
24. ábra A teljes architektúra blokkvázlata 37
25. ábra Csatorna megvalósítás (kép csere) 38
26. ábra Az elkészült UI 39
27. ábra Queue megnyitása 39
28. ábra Event Structure (bal oldal) és Case Structure (jobb oldal) 40
29. ábra Az inicializálás állapot megvalósítása 41
30. ábra Frequency Control VI 41

31. ábra Adatgyűjtés megvalósítása 42
32. ábra Spektrumanalizátor megvalósítása 44
33. ábra AGC megvalósítása 45
34. ábra DDC és szűrés megvalósítása 46
35. ábra Az LO és a fázihelyzet 47
36. ábra A keverés fix pont szerint 47
37. ábra Testing DDC, USRP jelszimulálás 49
38. ábra Teszt jel 50
39. ábra Demoduláció 51
40. ábra Hangkártya kijátszás 51
41. ábra Profilozás alapján a legrosszabb mutatókkal rendelkező subVI-ok 54
42. ábra Profilozás alapján a legrosszabb mutatóval jelentkező fő VI-ok 55