Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Szélessávú Hírközlés és Villamosságtan Tanszék

Műholdfedélzeti S-sávú adó fejlesztése

TDK Dolgozat

Miklós Barnabás

2021

HALLGATÓI NYILATKOZAT

Alulírott Miklós Barnabás, szigorló hallgató kijelentem, hogy ezt a TDK Dolgozatot 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 autentikált 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.

Budapest, 2021. október 27.

Miklós Barnabás

Tartalomjegyzék

Kivonat

A magyarországi műholdfejlesztések folytatásaként fejlesztett 3 PocketQube osztályú, potenciálisan 6. magyar műhold, az 5x5x15cm-es MRC-100 előreláthatóan a jövő évben, 2022 decemberében startol majd egy Electron hordozórakéta segítségével. A start dátuma miatt a műhold hívójele HA100MRC, neve MRC-100, ugyanis ekkor lesz 100 éves a Műegyetemi Rádió Club, ahol a műhold fejlesztése megvalósul.

A műhold fedélzetén helyet kap az általam fejlesztett S-sávú adó is.Ennek oka, hogy a műholdfedélzeti hasznos terhekből adódóan, rövid idő alatt viszonylag sok adat fog keletkezni, és ezeket a földi állomások felett áthaladva nagyjából 100 kbit/s - 1Mbit/s adatátviteli sebességgel kell lesugározni, amelyre a normál UHF sávú telemetria-telekommand rádió link alkalmatlan, többek között a rádióamatőr sávszélesség korlátok miatt.

A TDK dolgozatomban leírom a rendszerrel támasztott követelményeket(energia viszonyok, modulációs mód megválasztás, környezeti paraméterek, stb.), az áramköri és nyomtatott huzalozású lemez tervezést, a prototípus panelek élesztésének, bemérésének és laborbeli validálásának egyes lépéseit.

A leírás érinti nemcsak a műholdfedélzeti adót, hanem a földi állomás oldalon található vevőt is amely az esetünkben szoftverrádiós alapokon nyugszik.

Az elsődleges földi S-sávú vevő állomás a BME Etetőn található automatizált, távvezérelt és autonóm energia ellátással rendelkező, jelenleg a SMOG-1-et vevő és vezérlő állomás lesz, amely rendelkezik a megfelelő 3 ill. 4,5 m-es forgásparaboloid apertúra antennákkal a hozzájuk tartozó X-Y forgatóval, amellyel a LEO pályás MRC-100 megfelelően követhető.

Abstract

As a continuation of Hungarian satellite developments, a 3 PocketQube class, 5x5x15 cm sized MRC-100 satellite will launch in next year's December, with the help of an Electron rocket.

It will potentially be the 6th hungarian satellite. The device's callsign is HA100MRC, its name is MRC-100.

This name comes from the fact, that the start date is the centenary of the place where these developments took place, Műegyetemi Rádióklub.

On-board the satellite, the downlink communication will be handled by my transmitter circuit. There is a need for this, because the useful payload of the satellite generates a lot of data in a short period of time. To communicate this, downlink data speeds of 100 kbit/s-1 Mbit/s are required.

These speeds can not be acquired by a normal UHF band telemetry-telecommand radio link, mostly because of the constraints of the radio amateur band.

In my TDK thesis, I will write about the requirements of the system (energy ratios, choosing the right modulation, environmental parameters, stb...), the schematic and circuit board design, the assembly, measurements and validation process.

The thesis will also contain a write up about the software defined radio based terrestrial receiver station.

The primary terrestrial receiver station will be located on BME Etető. The receiver, which also the receiver and controller for SMOG-1, is automated, remote controlled and has an autonomous power supply system. It also has a 3 m and 4,5 m paraboloid aperture antenna with an X-Y rotator, which is excellent for following the LEO orbit of the MRC100.

1. fejezet

Áttekintés

Az egyetemen zajlik épp Magyarország potenciálisan 6. műholdjának a fejlesztése, mely a 3PQ (5x5x15cm) méretű MRC-100 lesz. A fedélzeten az általam fejlesztett S-sávú adó fogja a Föld felé sugározni a műholdon keletkezett adatokat. Mivel rövid idő alatt nagy mennyiségű adat fog keletkezni szükség van 100kbit/s-1Mbit/s adatátviteli sebességre.

Ezt a sebességet egy robosztus rádió linkkel tervezzük elérni. Ez a rádió link a műhold fedélzetén lévő relatív nagy teljesítményű, jó hatásfokú és kis méretű S- sávú adóáramkörből és a BME Etetőn lévő automatizált forgásparaboloid antennából fog állni.

Az adó fejlesztés folyamatában két modulációt került szóba: QPSK vagy MSK. E két módszer között minimális különbség van, viszont rengeteg közös tulajdonsággal rendelkeznek. Mindkettő hatékony sávkihasználással rendelkezik és kifejezetten ellenáll az űr-Föld csatornán fellépő fading jelenségeknek.

2. fejezet

A rádió vevő

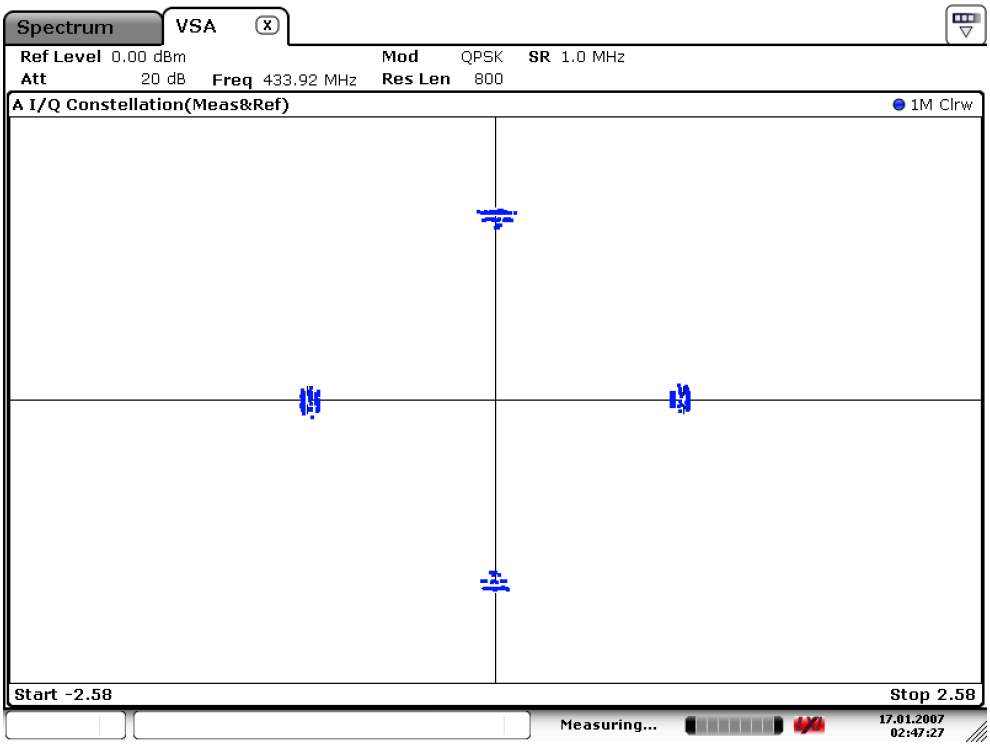
2.1. QPSK moduláció elmélete

2.1.1. A QPSK jelek leírása

Egy QPSK jel leírható:

$$s_n = \sqrt{2E_b/T_s}e^{j2\pi f_m t+\theta_n} \tag{2.1}$$

komplex körforgó vektorként Euler-formulával, f_m moduláló frekvenciával, ahol a fázis θ_n felvehet $m\pi/2 + \pi/4; m = 0, 1, 2, 3$ értékeket, T_s szimbólumidő intervallumokban E_b bitre leosztott energiával, ezek kimérve egy spektrum analizátoron (elforgatott verzió $-\pi/4$-el) a 2.1 ábrán látható



2.1. ábra. QPSK konstelláció(elforgatott verzió $\pi/4$-el)

Ez koszinuszos formában ennek felel meg:

$$s_n(t) = \sqrt{2E_b/T_s}cos(2\pi f_m t + \theta_n) \tag{2.2}$$

Additív Gaussi csatornában ehhez a jelhez hozzáadódik egy ω_n valószínűségi változó 0 várhatóértékkel és $\sigma = 10^{-SNR/20}$ szórással, ahol SNR a jel-zaj viszony(Szimulált zajos jel a 2.2 ábrán látható).

2.1.2. A QPSK jel spektruma

Mivel egy konkrét f_m frekvenciára felkevert négyszögjelről van szó ezért a spektrum ezekből adódik össze:

$$X(\omega) = AT_s sinc(\omega T_s) + 2\pi f_m \tag{2.3}$$

Eme spektrum látható az előző fejezetben.

2.1.3. A QPSK jel demodulációja(Costas-hurok)

Fázis és frekvenciahiba nélkül a demoduláció a moduláló vektor forgásirányával ellentétes irányú körforgóvektorral való visszaszorzásból áll.

$$\sqrt{2E_b/T_s}e^{j2\pi f_m t+\theta_n} \cdot e^{-j2\pi f_m t} = \sqrt{2E_b/T_s}e^{\theta_n} \tag{2.4}$$

Ha a két vektor fázisa közt különbség van akkor ez megjelenik a demodulált adatban is (ha a hiba frekvencia hiba akkor a θ_e ciklikusan 0-tól 2π felé változik ,ez a vektor(ezzel együtt a konstelláció) forgását idézi elő):

$$\sqrt{2E_b/T_s}e^{j2\pi f_m t+\theta_n} \cdot e^{-j2\pi f_m t+\theta_e} = \sqrt{2E_b/T_s}e^{\theta_n+\theta_e} \tag{2.5}$$

Ahhoz hogy ezt ki tudjuk kompenzálni ebből a hibából kell egy hibajelet generálni.A fázishiba jelet úgy képzem, hogy 2.4 ábrán látható módon szorzom a döntés előtti ($I_{inaccurate}, Q_{inaccurate}$) és utáni ($I_{limited}, Q_{limited}$) szimbólumok valós és képzetes részét és kivonom őket egymásból [5].

$$u_{phaseerror} = I_{inaccurate}Q_{limited} - Q_{inaccurate}I_{limited} \tag{2.6}$$

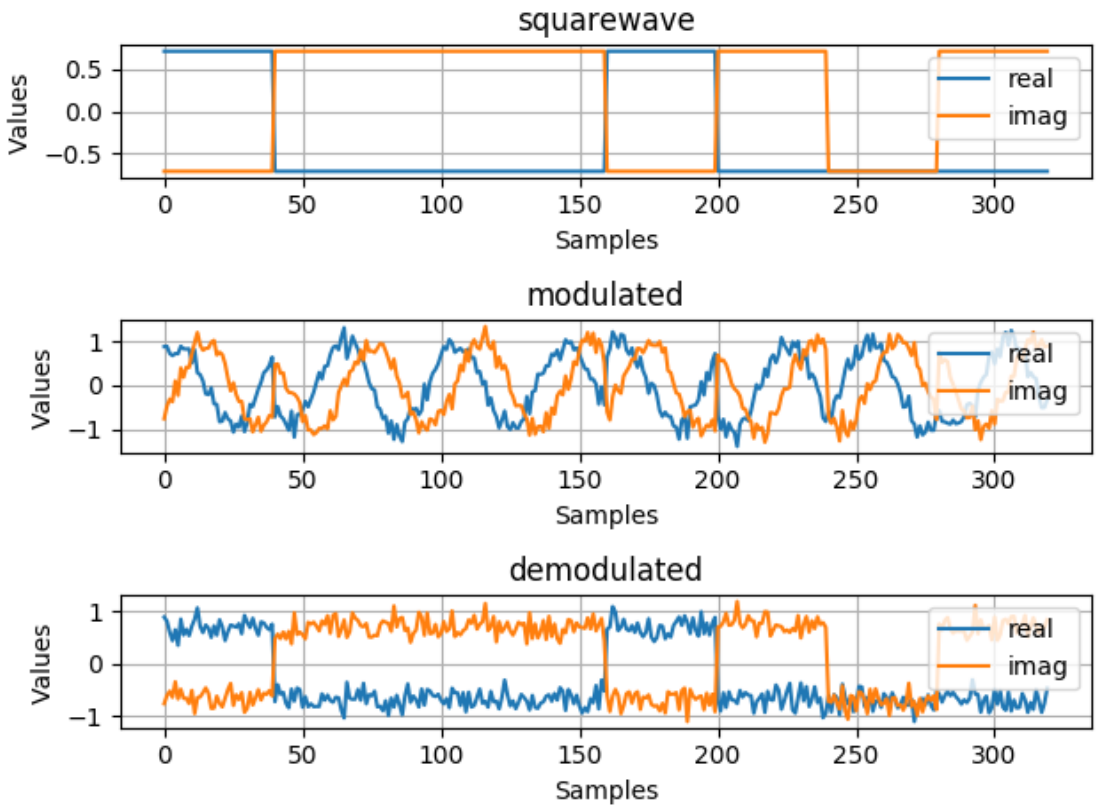
Ezt a hibajelet szűrve és integrálva használja a vevő a fázis- és frekvenciahiba kijavítására.

2.2. Adatátviteli lánc szimulációja C nyelven

A szimulációra azért van szükség mert ha elkészül a végleges realizációja az adónak és a vevőnek, ellenőrizni kell a hatékonyságát. Erre egy szimuláció ahol tudjuk hasonlítani a realizált működést az optimális szimulált működéshez a legalkalmasabb. Ehhez szimulálni kell az adatok előállításától a demodulációig a csatornában felszedett hibákkal együtt mindent. Emellett a vevő oldali szimulációs programkód a valós vételre is fel lesz használva; az adó oldali programkód részei pedig a különálló szintézeres megvalósításban a modulációt végző mikrokontrolleren futnak.

2.2.1. Ideális eset

Az elméleti részben leírtak alapján QPSK demoduláció ideális esetben csak a moduláló vektor forgásirányával ellentétes irányú körforgóvektorral való visszaszorzásból áll. Az 2.2 ábrán egy ilyen demoduláció látszódik AGWN-csatornán.



2.2. ábra. AGWN demoduláció

1. Először egy adott mennyiségű adat generálódik véletlenszerűen a randombytes.c (4.1) kóddal

2. Ezekből QPSK szimbólumok képződnek a byte2symbol.c(4.2) kóddal. A QPSK szimbólumok 00,01,10,11 bit kettősöket tartalmazzák.

3. Ezután az increment.c(4.4) kóddal inkrementálódik a jel(gyakorlatilag egy megadott szorzó szerint ismétlődnek ugyanazok a szimbólumok egymás után), hogy modulálható legyen

4. Majd felszorzódik egy komplex körforgó vektorral, ezt a cnco.c(4.5) végzi(Complex Number Controlled Oscillator)

5. Ezután Gauss-i fehér zaj adódik a jelhez az agwn.c(4.3) kóddal

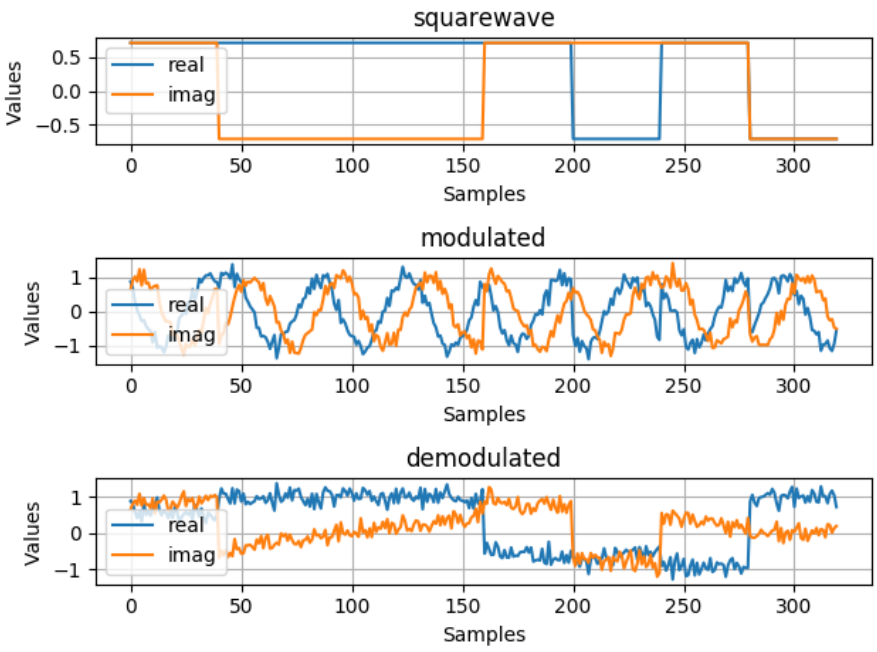
6. Egy másik CNCO-val ami az előző konjugáltját generálja(4.5) visszaszorzom a jelet

Látható hogy ez demoduláció tökéletes abból a szempontból, hogy a jelből teljesen eltűnt a moduláló jel frekvenciakomponense.

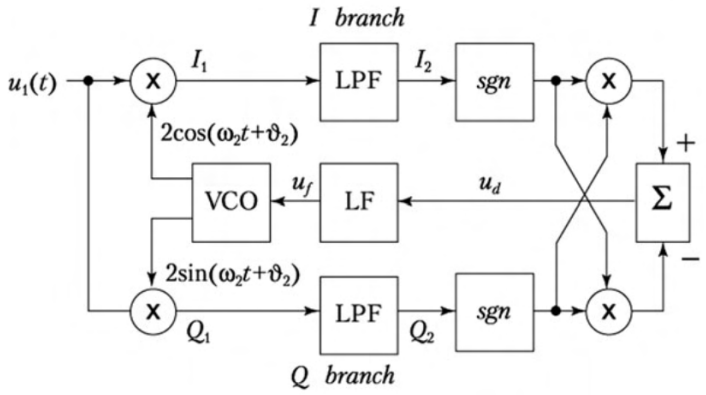
2.2.2. A frekvencia- és fázishiba

Egy például LEO pályán keringő műhold viszont kb 28000 km/h sebességgel kering a Föld körül ez a Doppler-effektus miatt több tíz kilohertzes csúszást eredményezhet 2 GHz körül, ahol a műhold tervezett adósávja elhelyezkedik.

Ez jelentős nemkívánatos frekvenciakomponenseket hoz be a spektrumba, sőt a földi vevőállomás és a műhold lokál oszcillátorának a frekvencia és fázisbeli eltérése is hibát okoz 2.3. Tehát szükség van egy eszközre ami kikompenzálja a fázis és frekvenciahibát. Ez a Costas-hurok 2.4 [5].



2.3. ábra. Frekvenciahibás(5 kHz) demoduláció



2.4. ábra. A Costas-hurok

1. Az előző szekcióban leírt módon generálva lesz egy modulált jel, majd nem a moduláló hanem egy ahhoz képest ofszettel rendelkező frekvenciával szorozódik vissza.

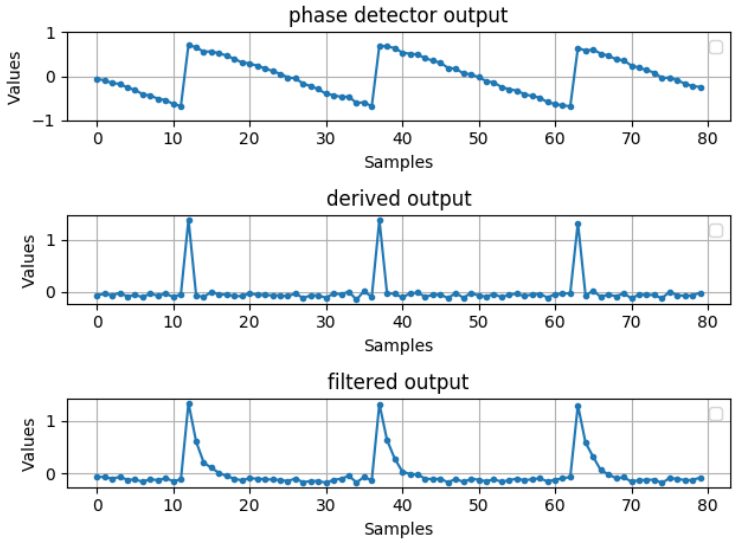
2. Ezután dekrementálódik a decrementbinary.c(4.7)kóddal

3. A dekrementált jelet vizsgálva döntök hogy valójában melyik szimbólum is érkezett meg.

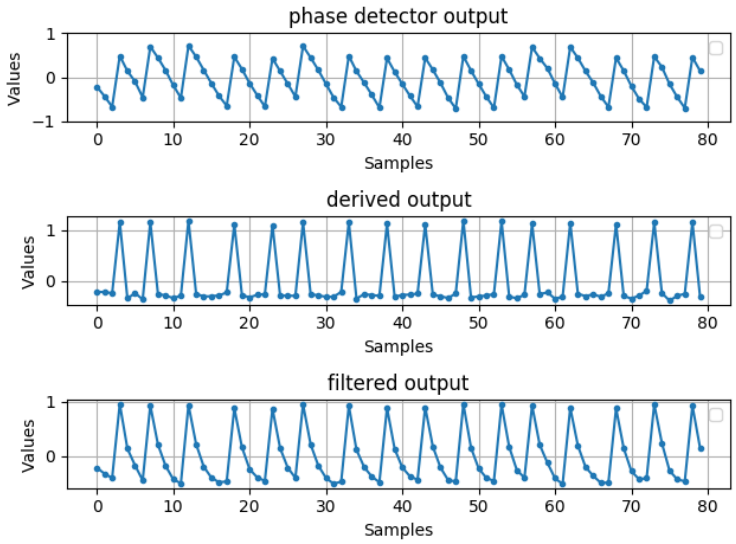
4. Az elméleti részben leírtak alapján a 2.4 ábrán látható módon szorzódik a döntés előtti és utáni szimbólumok valós és képzetes része ezután kivonódnak egymásból (4.9). Ez adja a fázishibát 2.5 2.6 .

5. Eza fázishiba jel utána integrálva lesz.

6. Az integrált fázishibával pedig egy CNCO-t(Complex numerically-controlled oscillator) léptetődik előre, és az oszcillátor jelével visszaszorzódik az aktuális jel adat. Az aktuális kódomban még nincs a hibajelen szűrő, ezzel majd optimalizálni lehet a működését és alkalmasabb lesz a zajosabb jelek kompenzálására is. A moduláló, modulált és demodulált jel(plusz az előbbi konstellációs ábrája) látszódik a 2.22 képen.



2.5. ábra. Fázisdetektor jele(1 kHz vevő ofszet)



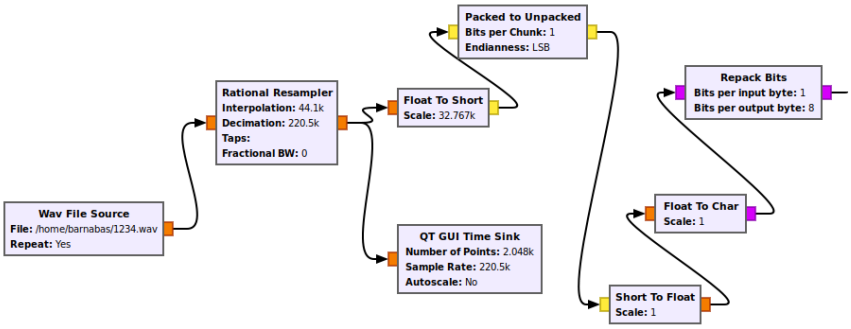
2.6. ábra. Fázisdetektor jele(5 kHz vevő ofszet)

2.3. Adatátviteli lánc szimulációja és tesztelése GNU Radio segítségével

Foglalkoztam az előbb leírt szimuláció megvalósításával GNURadio környezetben is. Ennek a célja az volt hogy legyen egy átfogó képem az egész QPSK adatátviteli láncról. Ezt a GNURadio grafikus felületet és jelábrázolási funkciói nagyban elősegítették.
Megvalósítottam egy .wav file átvitelét,először szoftveresen szimulálva, ezután pedig két B200 mini típusú szoftverrádióval le is teszteltem az átvitelt.
Az adó rész egy wav fájl forrásból áll, utána egy audió kodek megvalósítás található majd egy blokk mely komplex szimbólumokká alakítja a byte-okat.
A vevő rész először megkeresi a megfelelő mintavételi időpontot, egységkörre rakja a szimbólumokat majd kijavítja a frekvencia és fázishibát.

Az audió kodek

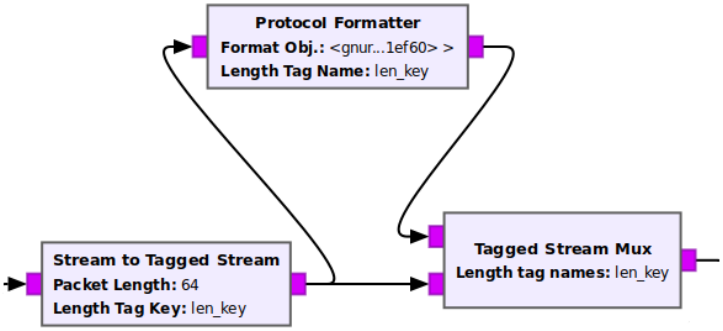
A .wav fájl forrás float értékeket küld ki magából, ezeket byte-okká kell alakítani. Ezt egy LPCM kodek végzi 2.7, azaz Linear Pulse Code Modulation. Gyakorlatilag a legegyszerűbb kodek amely a float értékeket kicsomagolja byte-okká, tehát egy 32-bites értéket 4 darab 8-bites értékre bont semmi más tömörítést nem végez.Az audiofájlt mintavételi értékek sorozataként továbbítja. Azért ezt a kodeket választottam mert egyszerű, GNURadio-s blokkokból megvalósítható és ebből kifolyólag könnyen érhető.



2.7. ábra. A .wav fálj float értékeinek 8-bitre bontását végző kodek része a programnak

A csomagokra bontás

Ahhoz, hogy akármilyen fájl eleje és vége meglegyen egy rádiós kommunikációban célszerű őket adott hosszúságú csomagokra bontani. Ezt végzi a következő része az adatátviteli láncnak 2.8. Ehhez a GNURadio Tagged stream funkcióját kell használni, és a Protocol formatter blokkot, mely egy "fejlécet" fűz minden packet-hez, tehát megjelöli az elejüket egy bitsorral.

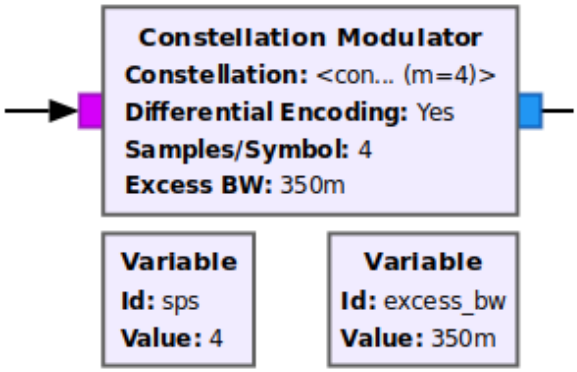


2.8. ábra. A csomagokra bontást a Tagged stream blokk és a Protocol Formatter végzi

A szimbólumokká alakítás

A QPSK moduláció komplex szimbólumokat használ, egy szimbólum két bites és megvan melyik két bit a konstelláció melyik értékének felel meg. A következő blokk, a mo-

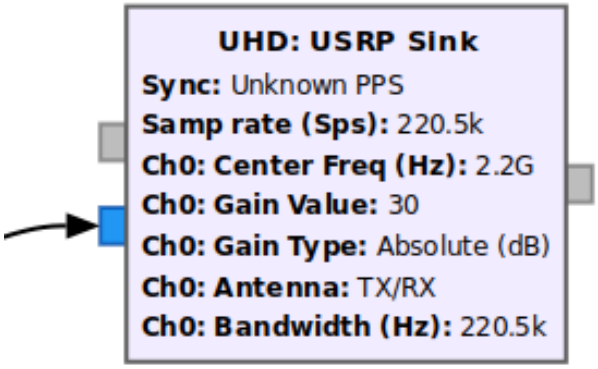
dulátor 2.9, ezt a megfeleltetést végzi Gray-kódolás alapján, tehát a kimenete komplex értékek.



2.9. ábra. A modulátor blokk, melyen be lehet állítani az négyzetgyök-emelt-koszinusz impulzus formálás lekerekítési paraméterét

USRP Sink

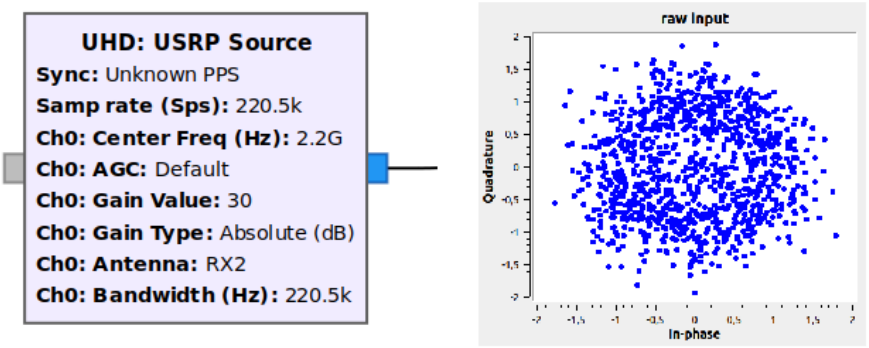
Ez a blokk 2.10 a B200 mini-be irányítja át az általam generált komplex szimbólumokat. A hardver pedig kiküldi ezt a jelet a beállított középfrekvencián.



2.10. ábra. Az USRP Sink blokk, beállítva 2.2GHz-re, mely az összes USRP szoftverrádiót támogatja

USRP Source

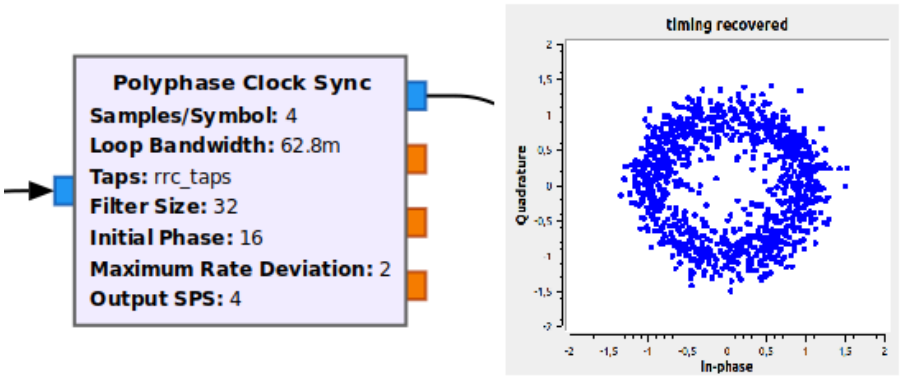
Ez a blokk 2.11 a vevő részen B200 mini-ből ,a megadott frekvenciáról lekevert, alapsávi jelet küldi tovább a programnak.



2.11. ábra. Az USRP Source blokk, beállítva 2.2 GHz-re és kimeneti jelének konstellációja

A legjobb mintavételi időzítés megkeresése

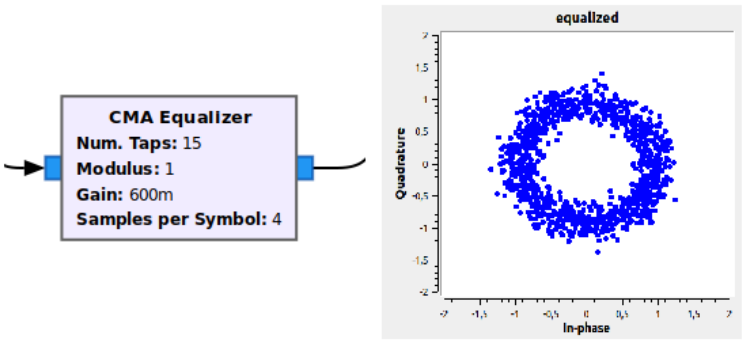
A következő blokk 2.12 arra szolgál hogy a bejövő szimbólumok pulzusainak a csúcsán mintavételezze ne pedig az átmenetkor, mivel a szimbólumokká alakításkor egy emelt koszinusz szűrővel "lekerekítette" a jelet ezért itt egy másik emelt koszinusszal illesztett szűrőként megpróbálja legjobban visszanyerni az eredeti jelett. Minimalizálva a szimbólumáthallást.



2.12. ábra. Polyphase Clock Sync blokk, mely a jó mintavételezési időzítés visszaállítására szolgál, és kimeneti jelének konstellációja

Egységkörre rakás

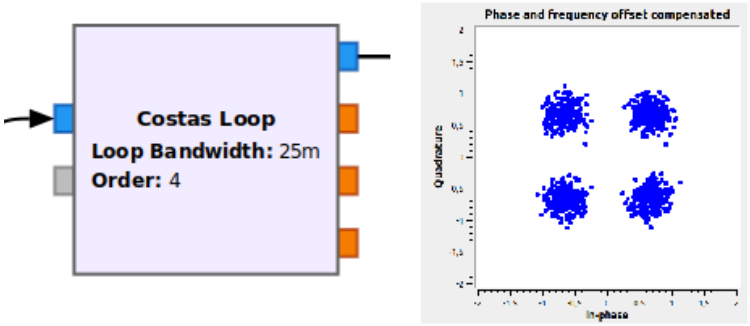
A CMA Equalizer blokk 2.13 pedig a zajos, nem konstans amplitúdójú jelet megpróbálja az egységkörre tenni, tehát 1-nél ne legyen nagyobb a komplex vektor hossza.



2.13. ábra. A képen a CMA Equalizer blokk(és kimeneti jelének konstellációja) látható, átlagoló ablak segítségével az összes szimbólumot az egységkörre rakja

Costas hurok

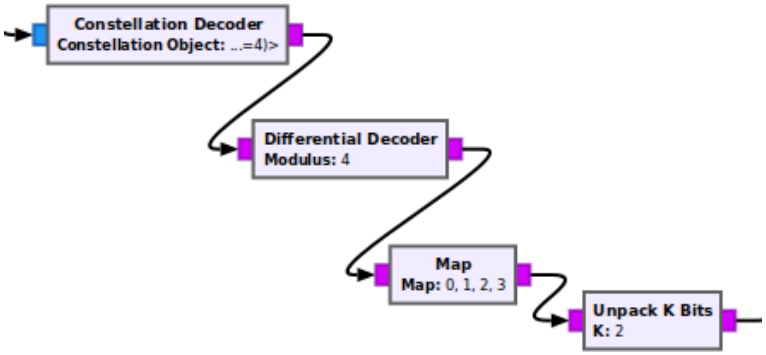
A Costas-hurok funkciójáról és működéséről már sok szó esett ebben a dokumentumban



2.14. ábra. A Costas-hurkot megvalósító blokk és kimeneti jelének konstellációja

A demodulátor

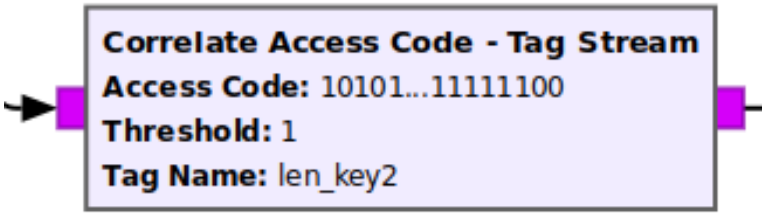
A különböző torzítási hibáktól mentesített szimbólumok pedig itt 2.15 demodulálódnak, tehát átalakulnak komplex szimbólumból byte folyammá.



2.15. ábra. A Costas-hurkot megvalósító blokk és kimeneti jelének konstellációja

A csomagok szétszedése

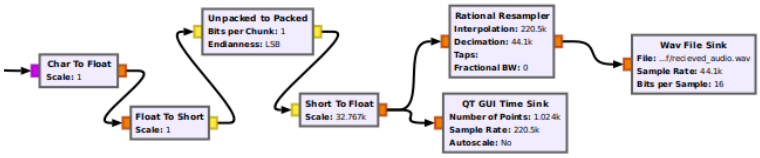
A következő blokk 2.16 megkeresi a fejlécet a csomagok eleje megtalálásának céljából, azután megfosztja a csomagokat a fejléc bitektől és kiadja már csak a hangfálj byte-jait.



2.16. ábra. A Correlate Access Code blokk a fejlécben megadott bitsort keresi, ha megtalálja akkor tovább engedi a stream-et és leszedi a fejlécet

A byte-okból hangfálj

Itt 2.17 az adat byte-okat már csak össze kell fűzni négyesével float-okká és kész a .wav fájl.



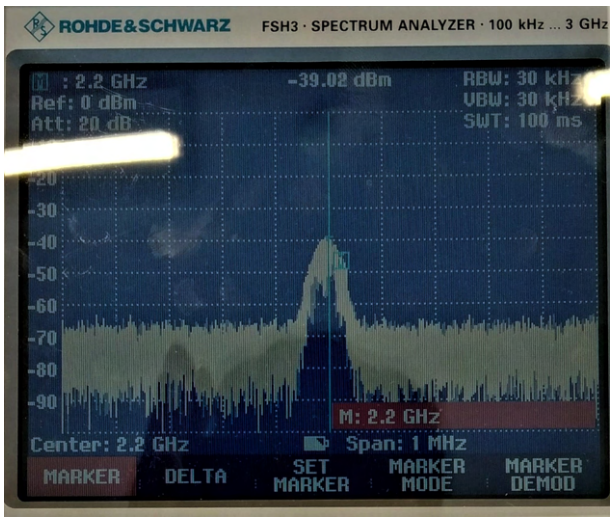
2.17. ábra. Az itt látható blokkok azt csinálják mint az adó oldalon lévő audió feldolgozó blokkok csak fordított sorrendben

Megvalósítás

Az adó és vevő szerepét is egy-egy B200 mini 2.19 típusú SDR töltötte be, a kimenő jel ki lett mérve spektrum analizátorral2.18.

Tapasztalatok

A GNURadio program nagyon hasznos segítség a különböző modulációk részletes működésének megértésére, de Python kódot futtat ami bizonyos sebességek után nem optimális. Ha a felhasználó komolyabb jelfeldolgozási láncot akar létrehozni benne ,személyes tapasztalataim alapján, az igényes dokumentáció hiánya nagyon meglassítja a munkát benne. Ezért írtam egy saját grafikus megjelenítő programot mely az általam már megírt szimulációs C kód jeleit rajzolja fel folytonos animációval. A jövőbeli jelfeldolgozó kódom hibakereséséhez írtam és hasznos lesz majd ehhez ez az ábrázoló környezet. A következő szekcióban röviden leírom az ábrázoló kód működését.

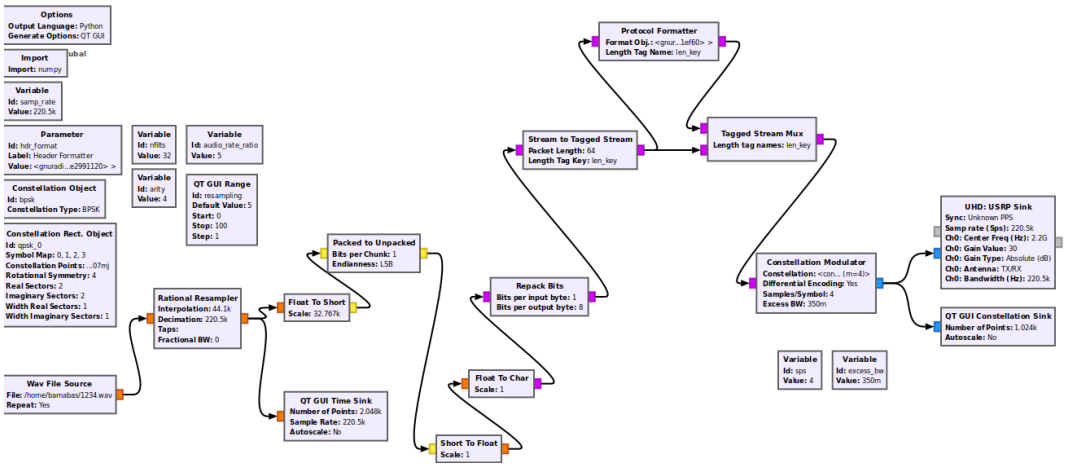


2.18. ábra. Az adó oldali SDR kimeneti jelének spektruma egy Rhode & Schwarz FSH3 hordozható spektrum analizátoron

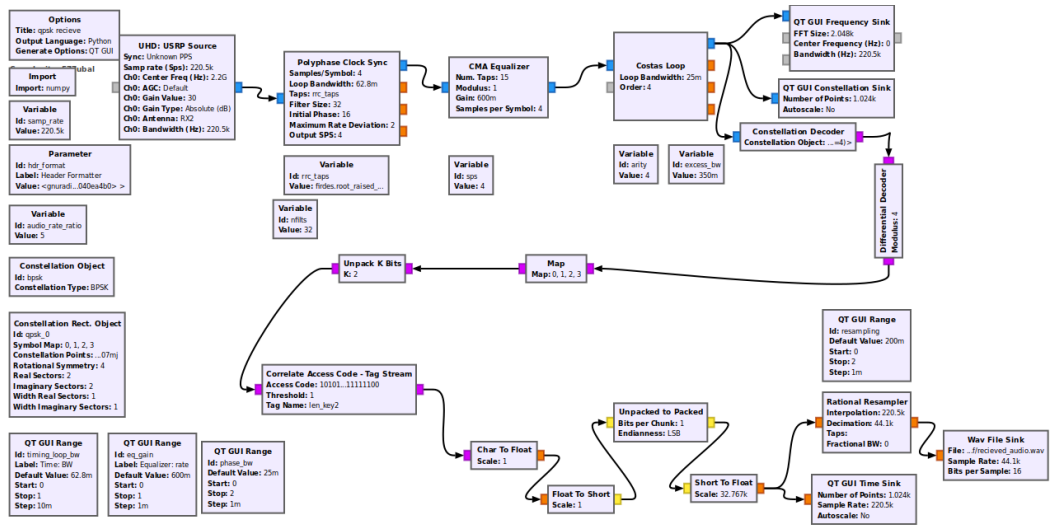
2.19. ábra. Az adó és vevő B200 Mini összekapcsolva a megfelelő csatalakozóikon keresztül

A szimulációs ábrázoló program

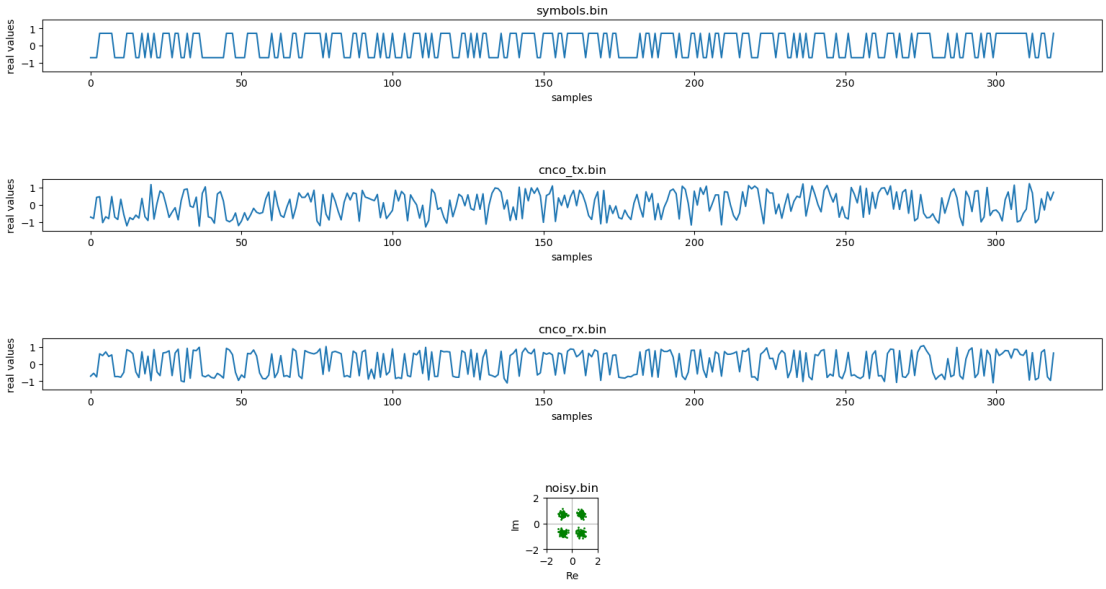
Az egész szimulációs kódot egy Shell script 4.12 fogja össze mely egymás után hívja meg a jelfeldolgozó C programokat és a kimenetüket a Pipe operátor(|) segítségével összeköti. A tee nevű program segítségével viszont minden jelfeldolgozási fázisban ki lehet írni egy bináris fájlba a kimenetet. Ezen bináris fájlokat olvassa és rajzolja ki a program, mely párhuzamos folyamatként fut a háttérben 2.22.
A animációs program Python-t használ, azon belül a matplotlib könyvtár animációs modulját 4.13, de mivel ez csak hibakeresésre és vizualizációra lesz felhasználva nem merülnek fel teljesítménybeli kérdések. A jelfeldolgozó részt nagyon egyszerű importálni egy összefüggő C kódba.
A ábrázoló kód úgy van megírva hogy könnyen lehessen új jeleket hozzáadni, egyenlőre két típusú ábrázolásra képes: időtartománybeli komplex és konstellációs.



2.20. ábra. Az adó teljes folyamábrája



2.21. ábra. Az vevő teljes folyamábrája



2.22. ábra. Pillanatkép a szimuláció működése közben

3. fejezet

A rádió adó

Az adó pontos tervezési paraméterei ez év áprilisában derültek ki, ez vezetett némi találgatáshoz és komplikációkhoz a tervezés és megvalósítás folyamatában.

Az adó végleges paraméterei:

- 2245-2290 MHz, S-sáv
- Alacsony fogyasztás($<$ 1 A)
- 4x4 cm
- Működjön a műhold szabályozatlan energiabuszáról(3.0-3.4 V)
- Hatékony sávkihasználás, 100kbit/s - 1Mbit/s kódolatlan adatsebesség
- Kimeneti teljesítmény: körülbelül 27 dBm

3.1. Különálló szintézeres áramkör

Ez az áramkör még kevésbé definiált paraméterekkel lett megvalósítva, egy sokrétűbb eszköz. Nagyobb frekvenciasávot lefed és képes többféle QAM moduláció megvalósítására(ebből csak QPSK lett tesztelve). Célja a szintézer IC tesztelése: megfelel-e egy műholdfedélzeti adó követeléseinek?

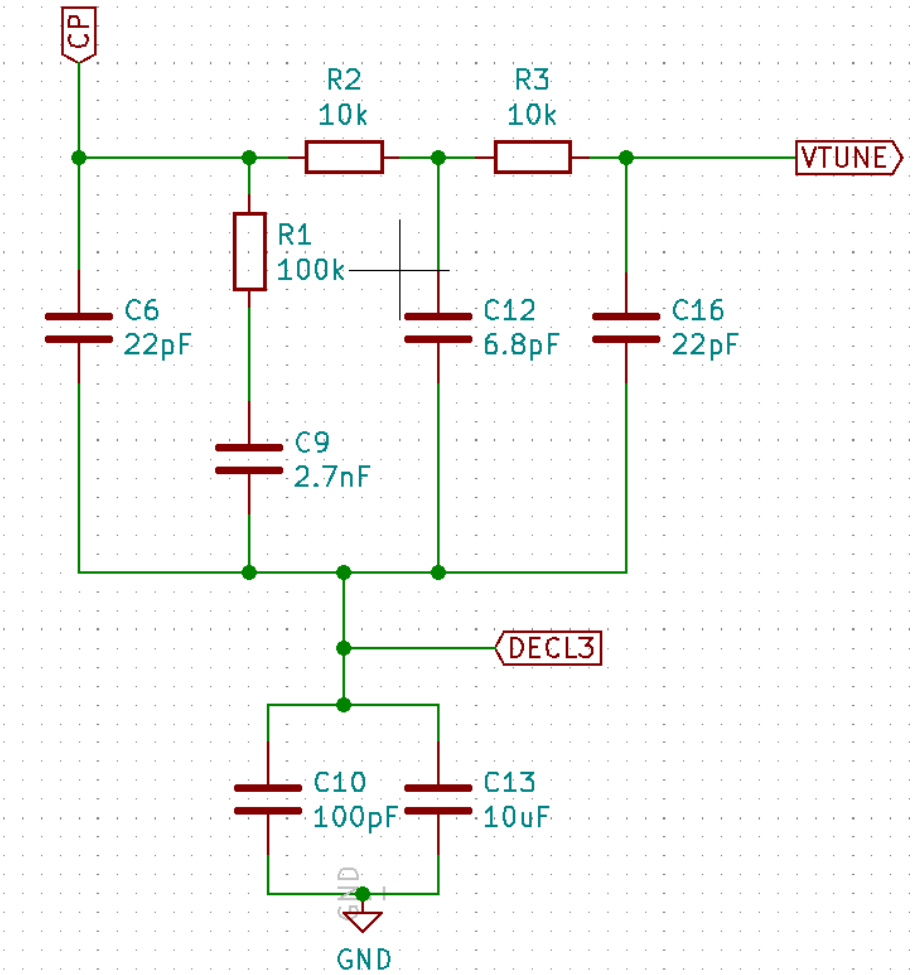
3.1.1. A szintézer

A szintézer IC-t Analog Devices kínálatából választottam mert megbízható és erre a célra megfelelő integrált lokáloszcillátorral ellátott IQ modulátor chippeket gyárt.

A tervezési irányelveket figyelembe véve 3 IC-re szűkült a választási lehetőség: ADRF6702, ADRF6703, ADRF6704. Ezek közül az ADRF6703 bizonyult a legjobb választásnak(az ADRF6703 jobb választás az ADRF6702 alacsonyabb fogyasztása de rosszabb rádiós paramétereivel szemben).

A referencia órajelet egy hőmérsékletkompenzált 20MHz-es oszcillátor szolgáltatja.

A hurokszűrő 3.1 , mely a VCO vezérlőfeszültségét szűri, 130 kHz-re lett beállítva.



3.1. ábra. A 130kHz hurokszűrő

3.1.2. Alapsávi jel előállítása

Az alapsávi jelet egy PIC32MM0064GPL036 alacsony fogyasztású mikrovezérlő állítja elő.

A szintézer bemeneteinek differenciális I és Q jelre van szükségük. Ezt nyolc GPIO láb és egy feszültségbeállító ellenállás hálózat segítségével állítja elő a mikrovezérlő.

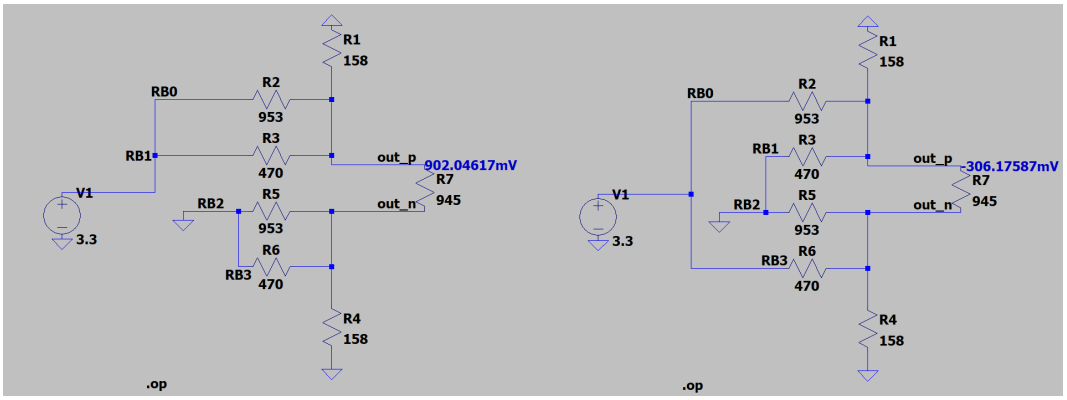
Az ellenállás hálózat értékeit MatLab segítségével számítottam és az eredményeket LT-Spice programban validáltam3.2.

3.1.3. Realizáció

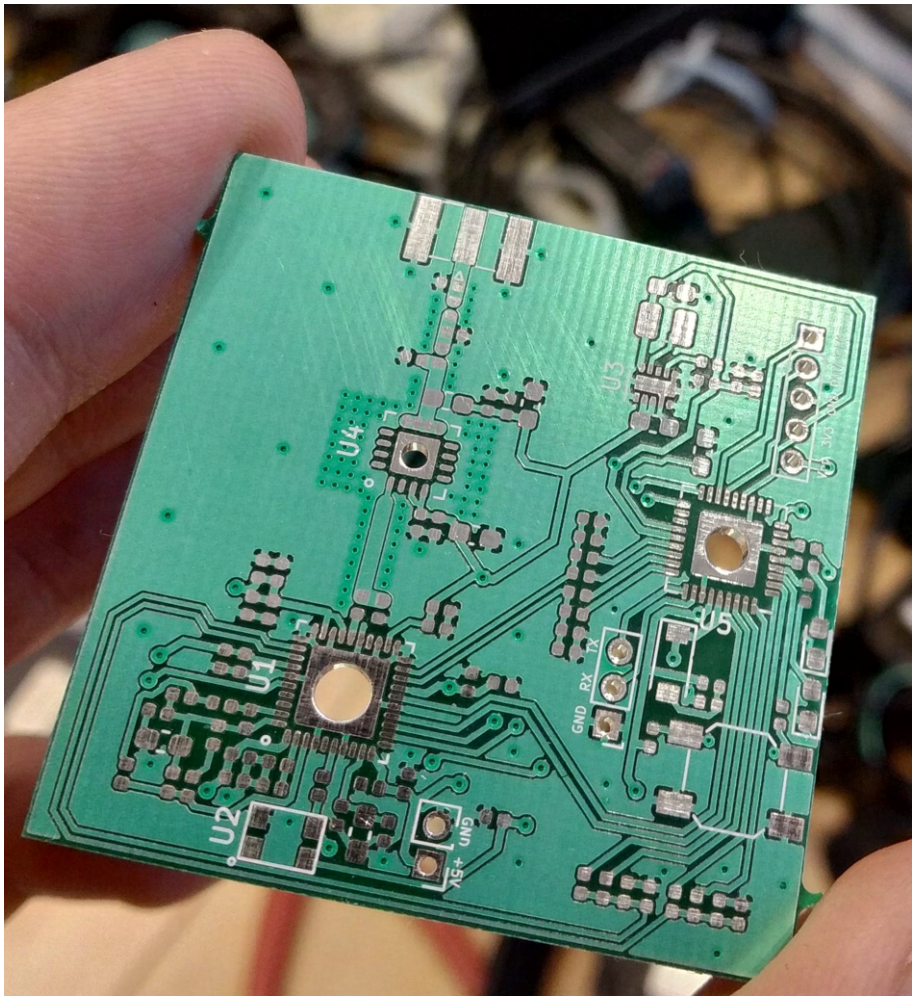
A tárgyalt áramkör legyártva 3.3. Az alkatrészek, ahelyett hogy egyszerre lettek volna beforrasztva, egyenként be lettek ültetve és működésük le lett tesztelve.

A beültetés és tesztelés

1. Először a mikrovezérlő tápellátását szolgáltató TPS62177 3.3V DC-DC konverter lett beforrasztva, multiméterrel le lett mérve a feszültsége. Majd egy 18 Ω terheléssel szintén ellenőrizve lett hogy tartja e az adatlapban specifikált paramétereket



3.2. ábra. A feszültségbeállító hálózat LTSpice-ban



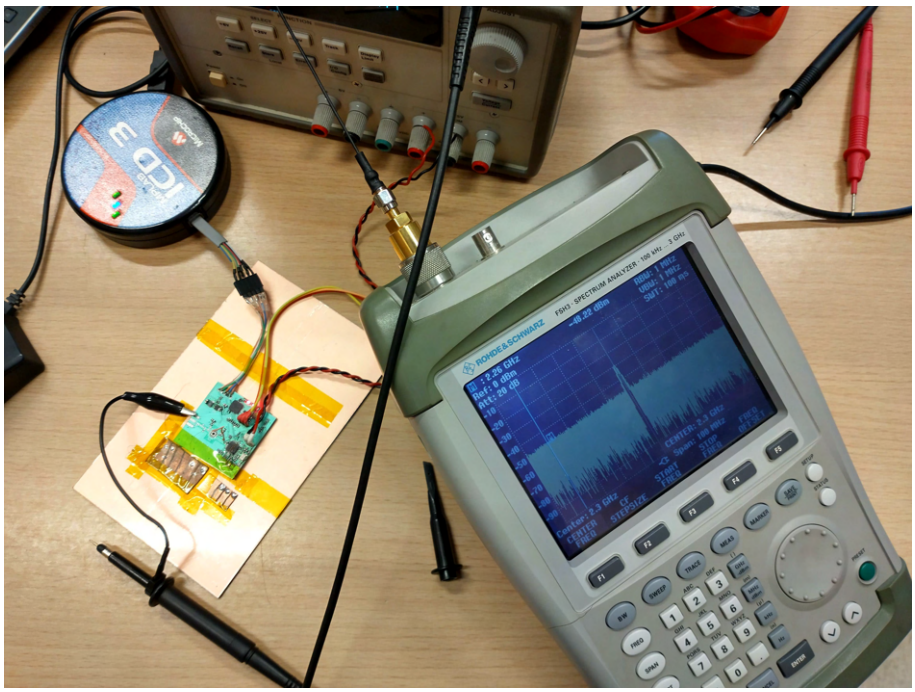
3.3. ábra. Az áramkör megvalósítása nyomtatott huzalozású lemezen

2. Miután a tápellátása le lett tesztelve, a PIC32MM0064GPL típusú mikrovezérlő lett beforrasztva. Először a programozó interfész(ICSP) lett ellenőrizve egy ICD 3 típusú programozó perifériával. Majd a soros interfész(RX,TX lábak) működése is ki lett próbálva egy USB-TTL átalakító segítségével. Az SPI interfész jele oszcilloszkóppal lett ellenőrizve a SCK, SDO és LE lábakon(ezek szükségesek a szintézer programozásához).

3. A következő lépés az LFTCXO075797 típusú hőmérséklet-kompenzált 20MHz-es

oszcillátor beültetése, jelének lemérése oszcilloszkóppal volt

4. Miután sikerült úgy beállítani a mikrovezérlőt, hogy a külső oszcillátor jeléről fusson(ellenőrzés a rendszer órajel kivezetésével) a következő lépés az ADRF6703 típusú IQ szintézer beültetése volt. Rengeteg munka után sikerült a chip regisztereit megfelelően konfigurálni. A szintézer működésének gyors ellenőrzése egy Rhode & Schwarz FSH3 hordozható spektrum analizátorral történt a képen 3.4 látható módon (az áramkör bemenete labortápról ellátva 5V feszültséggel és 300mA áramkorláttal). A chip kimenetére egy vezeték lett forrasztva antennaként.



3.4. ábra. A szintézer ellenőrzése a chip kimenetére forrasztott méretezett vezetékkel antennaként

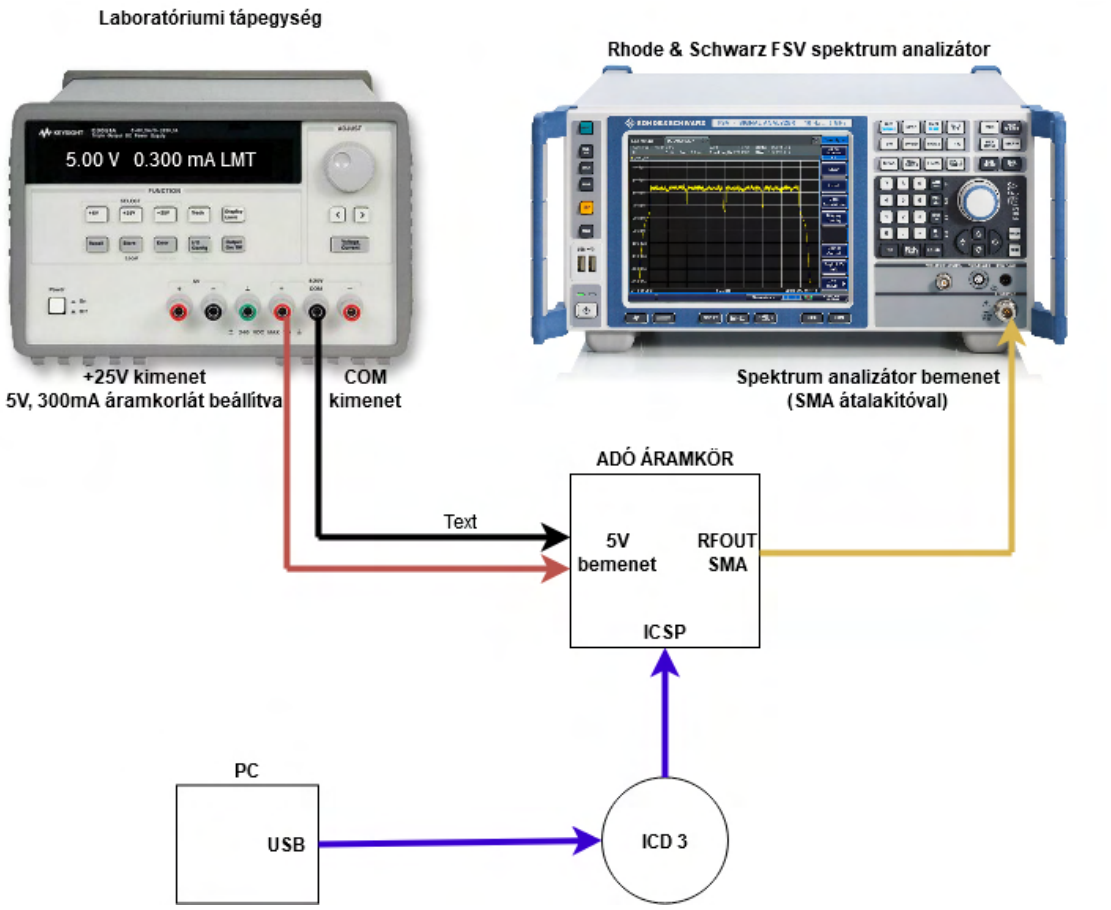
5. Az utolsó beültetendő alkatrész az ADL5606 típusú erősítő

3.1.4. Az adó jelének kimérése

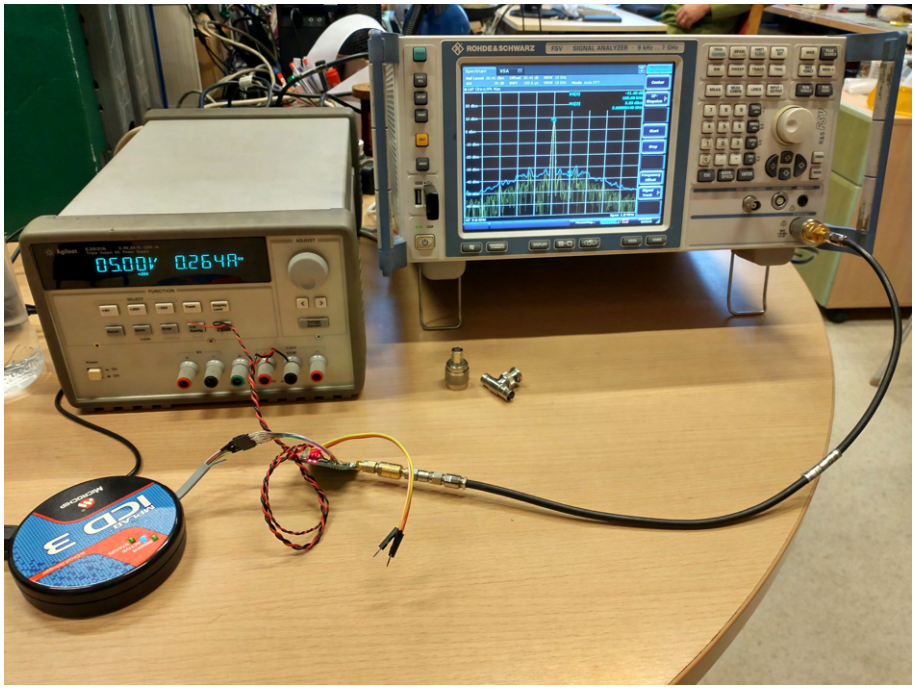
Az előző szekcióban a kimenő jel csak felületesen lett megmérve. Ebben a szekcióban, a szintézer kimeneti jelén, egy Rhode & Schwarz FSV típusú spektrum analizátorral folytatott mérés jegyzőkönyve következik. A mérés célja az adó megfelelő működésének ellenőrzése volt.

Az szintézert a mikrovezérlőn futó 4.14 program konfigurálja és küldi tovább neki a QPSK modulált adatsort.

Az adó három frekvencián lett kimérve háromféle bitsebességgel. Az áramkör képes adni 2100MHz-től 2600MHz-ig terjedő tartományon, ehhez három reprezentatív adófrekvencia: 2140MHz, 2340MHz, 2600MHz. Ebből a dolgozatba csak a 2340MHz-es mérési eredmények lettek berakva, mivel reprezentálják a többi frekvencián mért adatokat is.

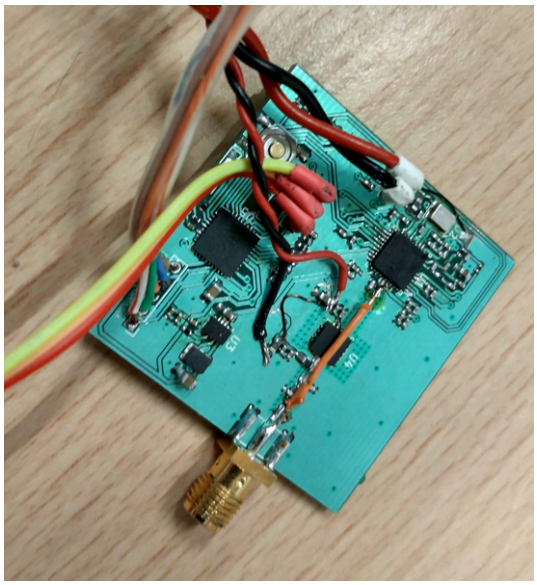


3.5. ábra. A mérési összeállítás blokkvázlata



3.6. ábra. A mérési összeállítás

A háromféle szimbólumsebesség : 10kS/s, 100kS/s és 166kS/s (kS/s: kilosample per second). Mivel QPSK modulációt használ az áramkör, minden szimbólum két bitnyi in-



3.7. ábra. Az áramkör felkészítve a mérésre

formációt tartalmaz, ezért a bitsebességek a következők: 20kbit/s, 200kbit/s és 332kbit/s.

A tesztelő adatsor egy álvéletlen, 512 byte-ból álló, byte sorozat. Ez ismétlődik ciklikusan a kimeneten, biztosítva az egyenlő szimbólumeloszlást, szimulálva egy valóságos adatátvitelt, a reális mérési eredmények érdekében.

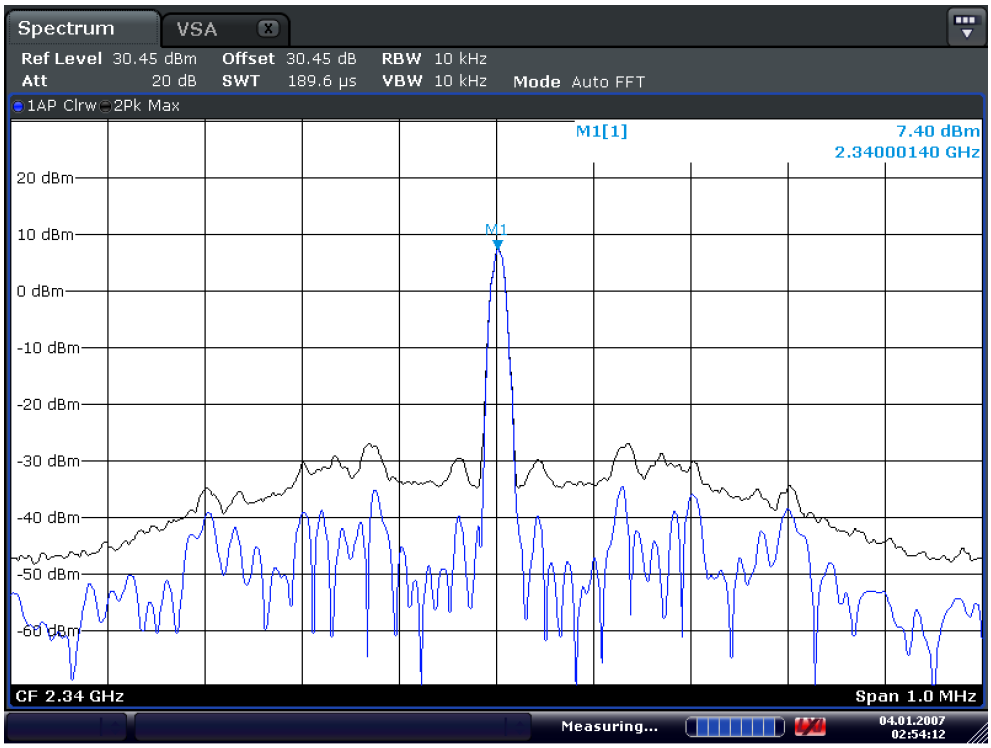
A mérési összeállításban szerepel az ICD 3 programozó periféria is. Ezzel lett beállítva az adófrekvencia és a szimbólumsebesség. Ezek a paraméterek a jövőben a mikrokontroller valamelyik soros interfészén(UART,I2C,SPI) keresztül lesznek programozhatóak működés közben.

A konstellációs mérési eredmények a spektrum analizátor Vector Signal Analysis funkciójának segítségével lettek megmérve. Az additív gaussi zaj pedig az SMA konnektor széthúzásával adódott a kommunikációhoz.

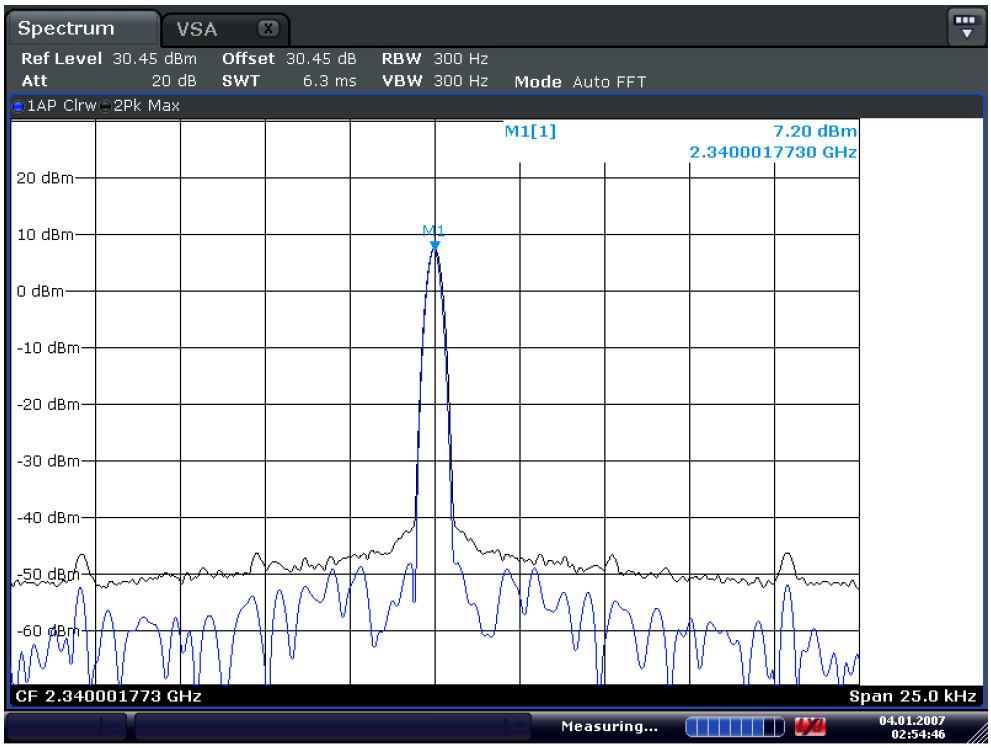
Mérések 2.34 GHz adófrekvencián

Szintézer stabilitásának a mérése

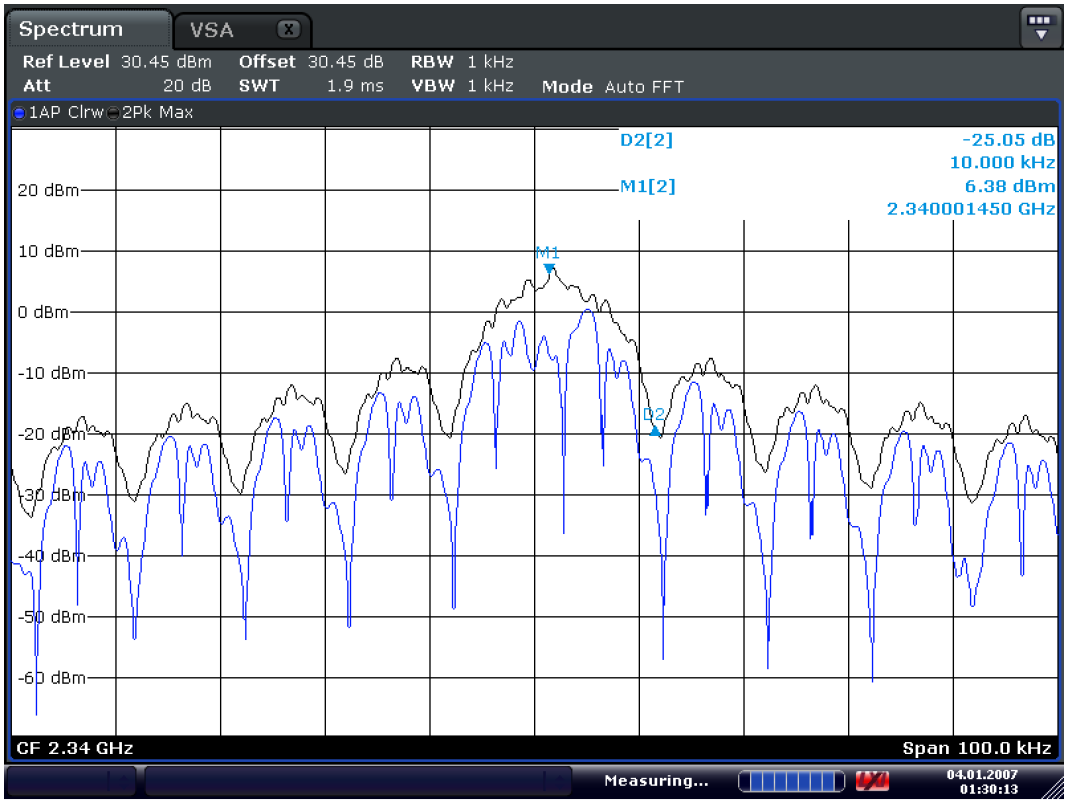
A szintézer spektruma ki lett mérve, úgy hogy a tápfeszültség 0.1 V lépésekben csökkentettem. Az utolsó szint amin még volt kimeneti RF jel a 3.3 V volt 3.17.



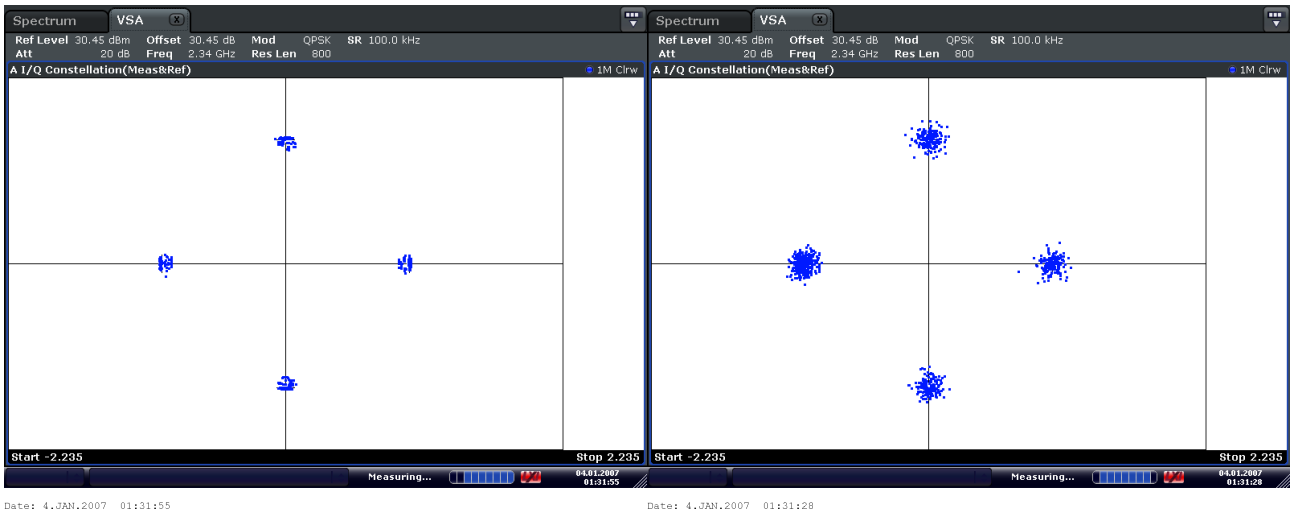
3.8. ábra. A vivőjel spektruma 2.34 GHz frekvencián, 1MHz span



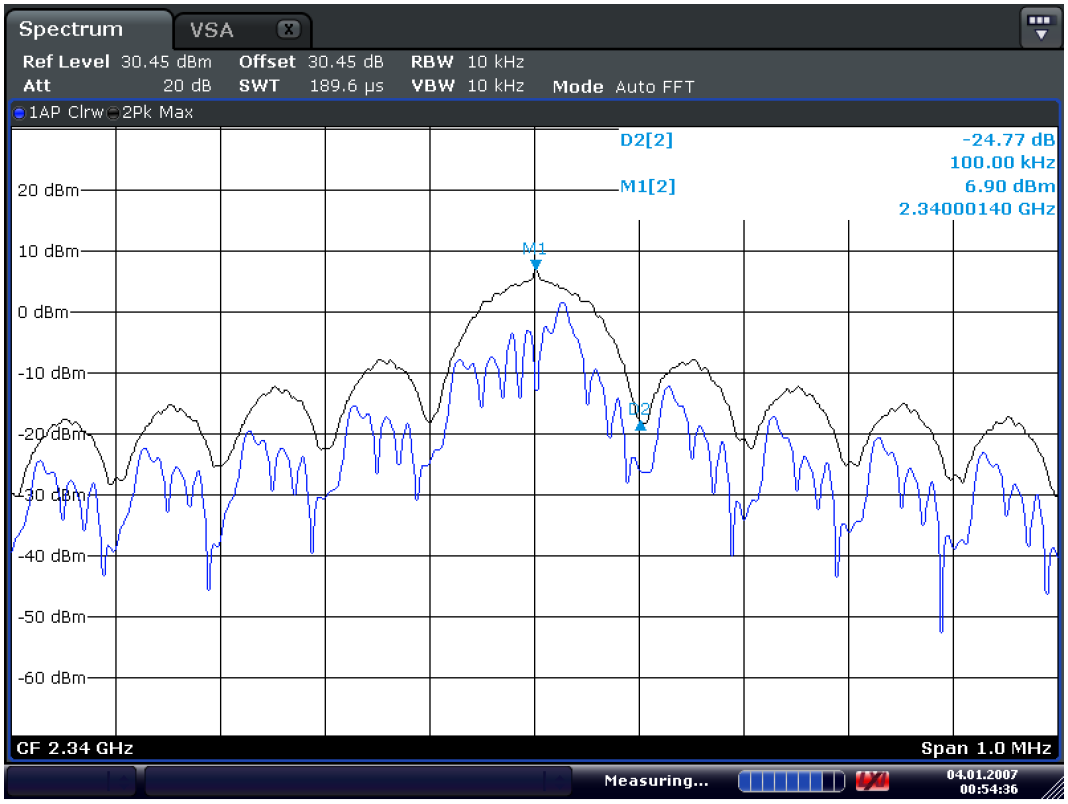
3.9. ábra. A vivőjel spektruma 2.34 GHz frekvencián, 25kHz span



3.10. ábra. A modulált jel spektruma 2.34 GHz frekvencián, 10kS/s, 100 kHz span

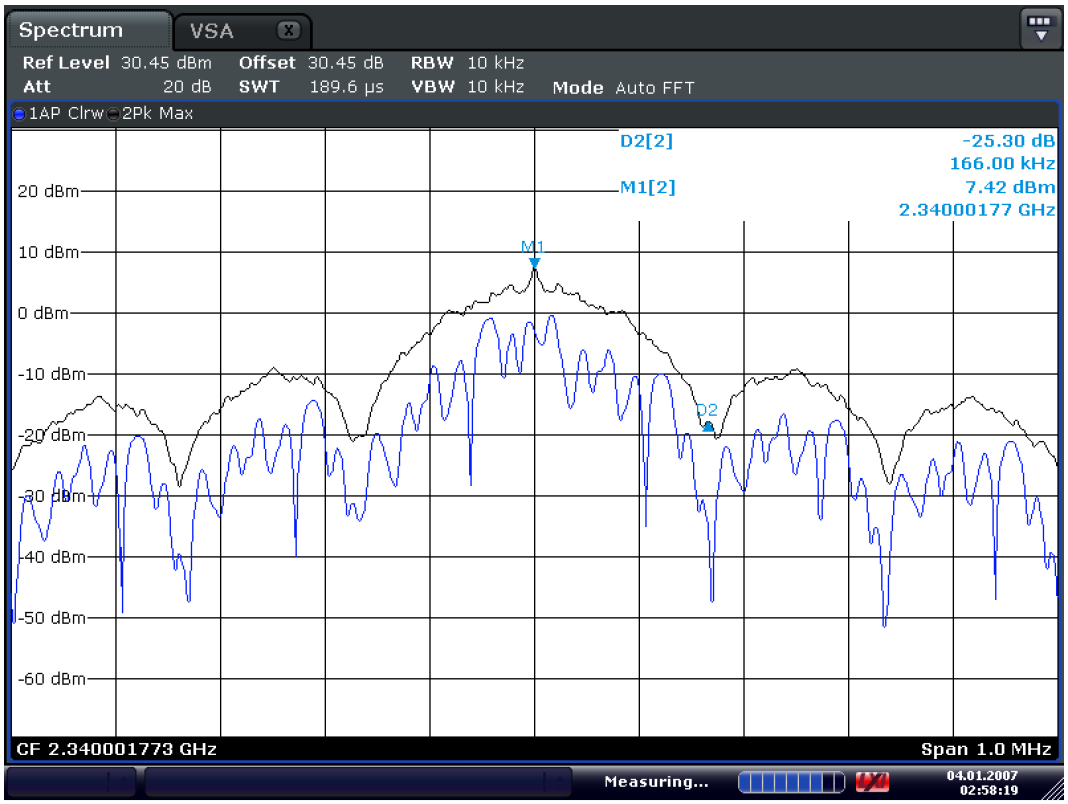


3.11. ábra. A modulált jel konstellációja additív zaj nélkül és additív gaussi zajjal 2.34 GHz frekvencián, 10kS/s

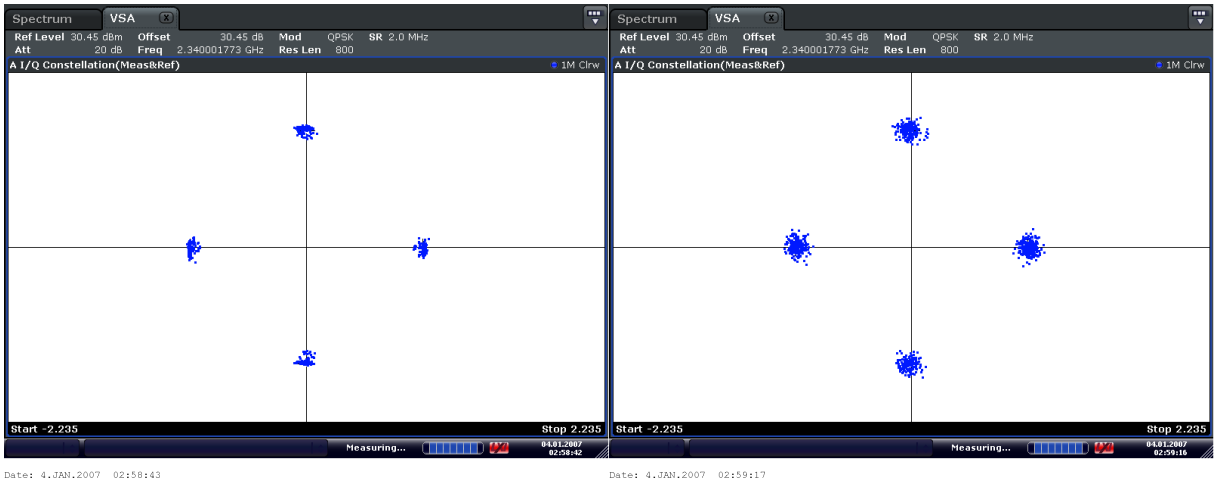


3.12. ábra. A modulált jel spektruma 2.34 GHz frekvencián, 100kS/s, 1 MHz span

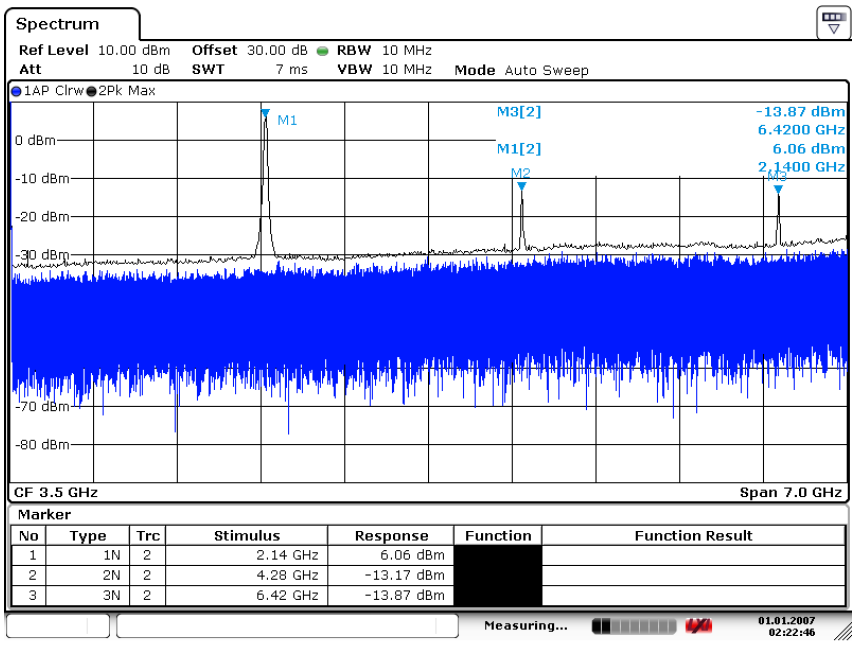
3.13. ábra. A modulált jel konstellációja additív zaj nélkül és additív gaussi zajjal 2.34 GHz frekvencián, 100kS/s



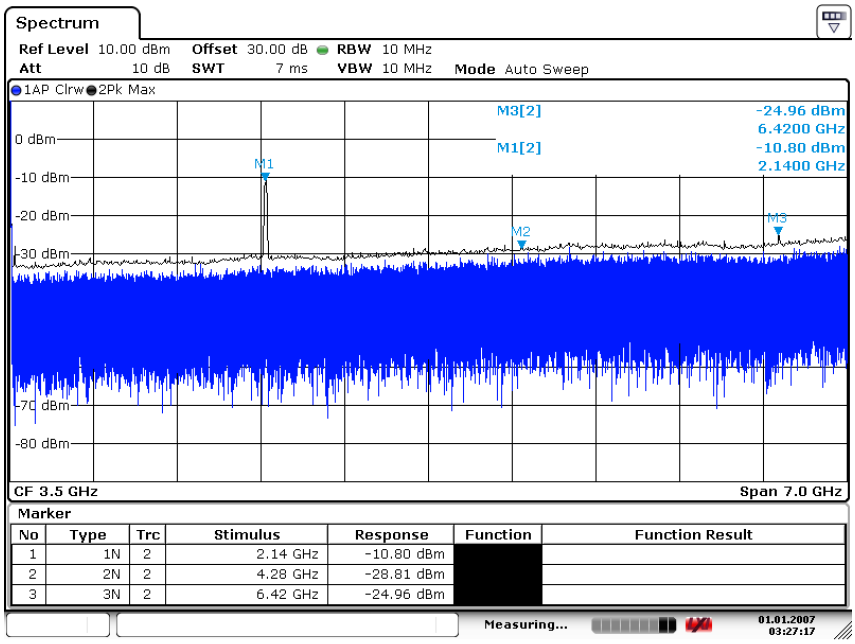
3.14. ábra. A modulált jel spektruma 2.34 GHz frekvencián, 166kS/s, 1 MHz span



3.15. ábra. A modulált jel konstellációja additív zaj nélkül és additív gaussi zajjal 2.34 GHz frekvencián, 166kS/s



3.16. ábra. A szintézer harmonikusai



3.17. ábra. A szintézer spektruma 3.3V tápfeszültségen

Mérési eredmények kiértékelése

A vivőjelek minden mért frekvencián megfelelőek, a modulált jelek spektruma minden sebességnél az elvárt

$$\sin(x)/x$$

formát hozza, első leszívási pont a moduláló jel frekvenciájánál. A vivőfrekvenciák értékei sosem pont a beállított értékekre estek, viszont ez betudható a referencia oszcillátor hibahatárának.

A jel teljesítmény mindenhol meghaladja az adatlapban specifikált 4-5 dBm értéket. Ennek az oka hogy a VCO amplitúdót állító regiszter a legnagyobb értéken volt, a moduláló jel amplitúdója szintén az ajánlott értékek felső határán helyezkedett el. Ez meglátszik a fogyasztásban is: az adó áramfelvétele 250-270 mA között mozgott, általában 265 mA (a mérési ábrán is pont 264 mA látszik 3.6) értéket vett fel. Ebből levonva a mikrovezérlő 8 mA fogyasztását 257 mA jön ki, mely 17 mA-rel több mint az adatlapi átlagérték. Ez az átlagérték viszont 4 dBm teljesítményre 25 C°-on lett kimérve tehát még a státusz LEDek fogyasztását és az IC melegedését elhanyagolva is megfelelő érték a 257 mA-es fogyasztás.

A konstellációs diagramokból viszont látszik, hogy a szimbólumok "el vannak kenődve", tehát fáziszajos a modulált jel, a relatíve nagy jelteljesítmény ellenére is.

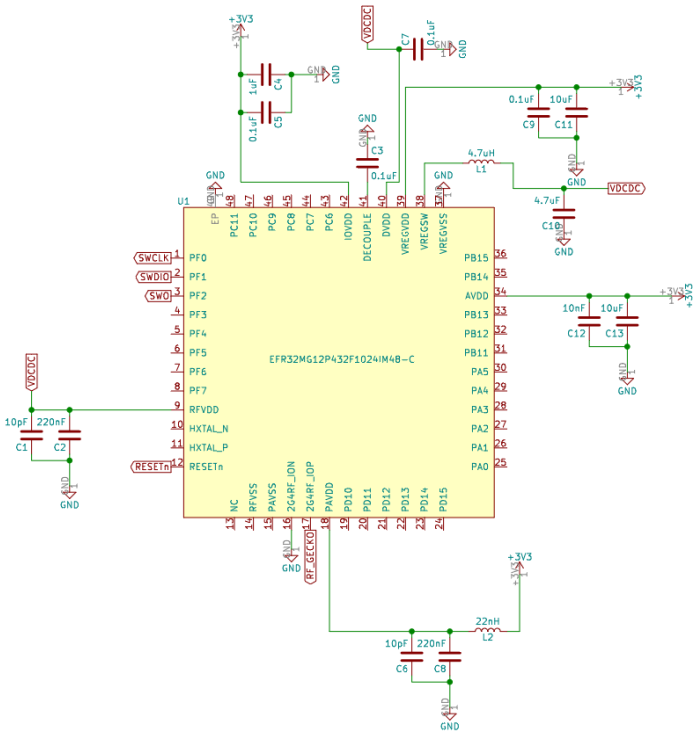
Az erősítő méréséről későbbi szekcióban lesz szó.

3.2. Integrált szintézeres áramkör

3.3. SiliconLabs rádiós IC

A tervezett frekvencia melyen a műhold kommunikálni fog 2 GHz és 2.4 GHz közé esett a pontos specifikációk előtt. A 2.4 GHz-es ISM sávokban zajló kommunikáció viszont jelentős interferenciát okozott volna a csatornán. Ezért minél távolabb kerül az aktuális csatorna a jelentős zajteljesítménnyel telített sávoktól, annál hatékonyabb lesz a kommunikáció. Viszont azok a kereskedelmi forgalomban lévő szintézer IC-k amik lefedik ezt a 2.4 GHz alatti frekvenciasávot, és képesek saját rádiós protokoll megvalósítására (nem pedig előre kódolt protokollokra mint a BLE, Zigbee stb.) nem túl magas hatásfokúak és általában 5 V (vagy több) tápfeszültségről üzemelnek.

Azok az IC-k melyek képesek 3.3 voltról üzemelni és magas hatásfokúak is mind az ISM sávokat lefedő adó-vevő eszközök. Ezek a chipek általában egy rádiós modulból és egy mikrovezérlő modulból állnak(egy tokban), gyakorlatilag a rádiós kommunikáció teljes hardveres része egy szilíciumra integrálva(alapsávi jelgenerálás, felkeverés és erősítés plusz szűrők). Mivel az ISM sávokban üzemelő IC-kre nagy a piac, ezért olcsók és optimalizáltak. De az említett eszközök paraméterei között nem található meg általában az, hogy el tudják-e érni a 2245 MHz-es frekvenciát(amin az adó üzemelni fog). Ennek több oka is lehet: fizikailag nem képes a szintézer lemenni eddig a frekvenciáig, le tud menni de nem lehet elérni ezt a beállítást csak speciális eszközökkel, nem írták bele az adatlapba mivel az átlag felhasználó úgyis csak az ISM sávban fogja használni stb. Emiatt sok időt töltöttem a különböző chipgyártók ügyfélszolgálatával való levelezéssel, hogy megtaláljam melyik ilyen eszköz képes mégis lefedni a megfelelő frekvenciasávot. Az első amit találtam az a Silicon Labs EFR32MG12 chip volt. Ennek a szintézere 2360 MHz-ig volt képes jelet

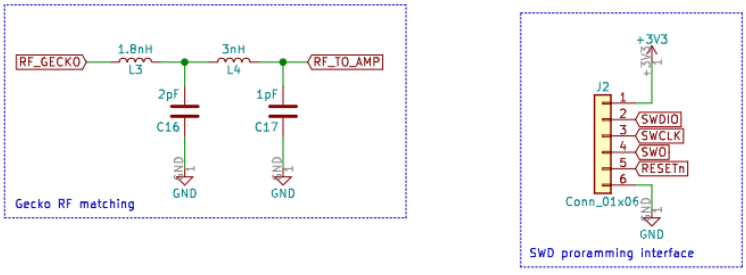


3.18. ábra. EFR32MG12 adó-vevő integrált áramkör, 20 dBm adóteljesítményre optimalizált kapcsolásban

előállítani(adatlapon nem jegyzett), ekkor még nem volt tisztázva a pontos adófrekvencia ezért terveztem ezzel az eszközzel egy kapcsolást 3.18.

Ez a chip elvileg 20 dBm adóteljesítményre képes[10]. A megvalósított kapcsolás erre a teljesítményre optimalizált, emellett igénybe veszi az IC belső DC-DC konverterét amely minimálisan több alkatrészt jelent viszont hatékonyabb működést[11].

A kapcsolásban az illesztés is az említett adóteljesítményre történt[12]. Sajnos ez az IC nem tudja kezelni a végleges frekvencia sávot, ezért nem lett használva, és a kapcsolás is befejezetlen maradt amikor áprilisban megtudtam a hírt, hogy nem tudok vele tovább haladni.

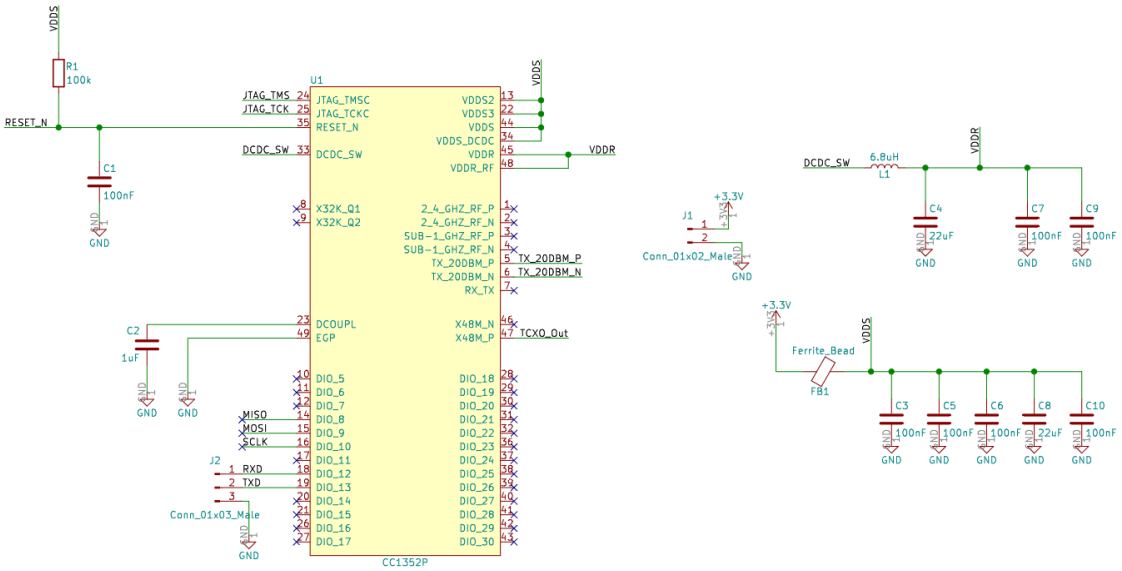


3.19. ábra. EFR32MG12 illesztő áramköre és programozó interfésze

3.4. Texas Instruments rádiós IC

Miután fixálódott a működési frekvenciasáv újra elkezdtem keresni olyan integrált áramkört amely megfelel az elvárásoknak. Mivel még távolabb került a frekvencia az ISM sá-

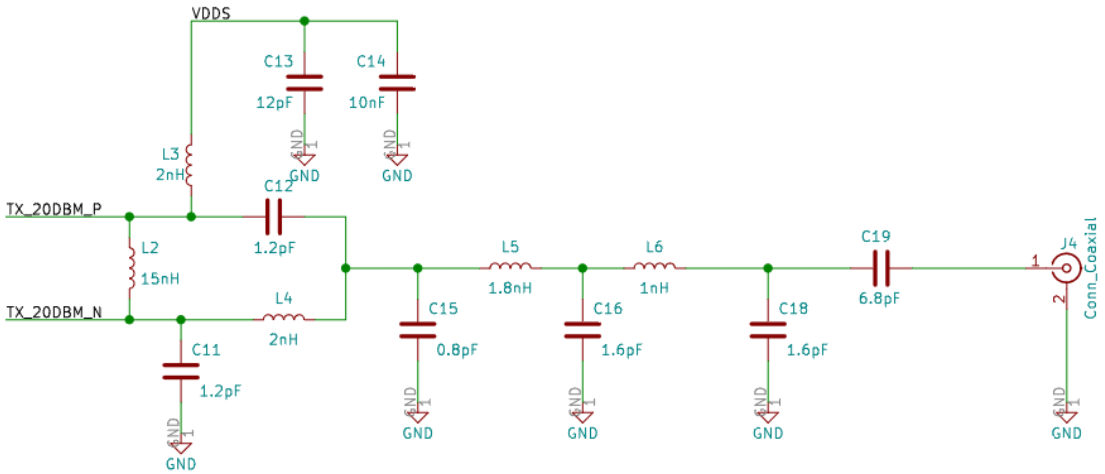
voktól nehezebb feladat lett találni ilyen IC-t. De ennek ellenére megtaláltam a Texas Instruments katalógusában a CC1352p típusú chipet, melynek a szintézere a vevőszolgálat szerint képes 2153 MHz-ig lemenni és lehet rajta implementálni saját rádiós protokollokat is. Ezért ehhez az eszközhöz terveztem egy kapcsolást. A CC1352p is 20dBm kime-



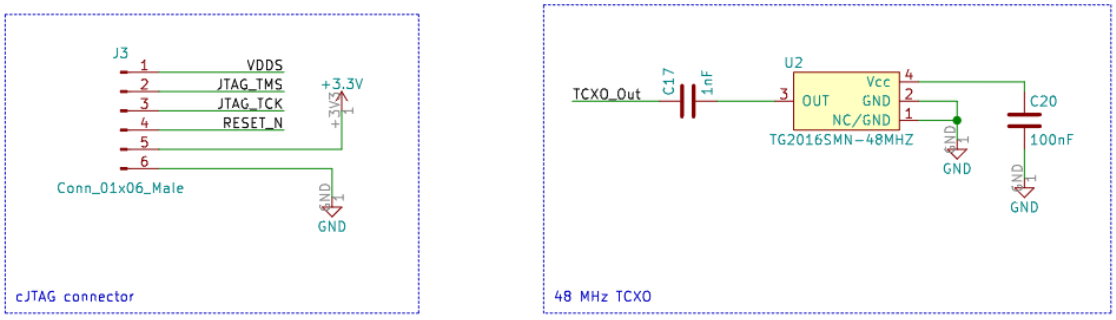
3.20. ábra. A CC1352p adó-vevő integrált áramkör alapvető kapcsolása

nő teljesítményt tud 2.4 GHz-en,ezt tudja 3 V, 85 mA fogyasztás mellett[13]. Amely hogyha pontos adat, akkor kitűnő hatásfokot jelent. A megvalósított kapcsolás erre a teljesítményre optimalizált, emellett igénybe veszi az IC belső DC-DC konverterét amely minimálisan több alkatrészt jelent viszont hatékonyabb működést[14]. A kapcsolásban az illesztés is az említett adóteljesítményre történt[15]. Később lesz szó a végfok erősítőről, de fontos kiemelni, hogy az koaxiális csatlakozó helyére fog majd kerülni a kapcsolásban. A hőmérséklet eltolódásból adódó frekvencia- és fázishiba csökkentésének érdekében egy TCXO (Temperature Compensated Oscillator) szolgáltatja az órajelet a rádió modulnak.

Ez az eszköz csak Gaussian FSK modulációra képes, melyet lehet MSK-ként konfigurálni tehát az gyakorlatilag Offset QPSK.



3.21. ábra. A CC1352p adó-vevő IC illesztő kapcsolása 20 dBm teljesítményen



3.22. ábra. A TXCO és a programozó cJTAG interfész

3.5. Végfok erősítők

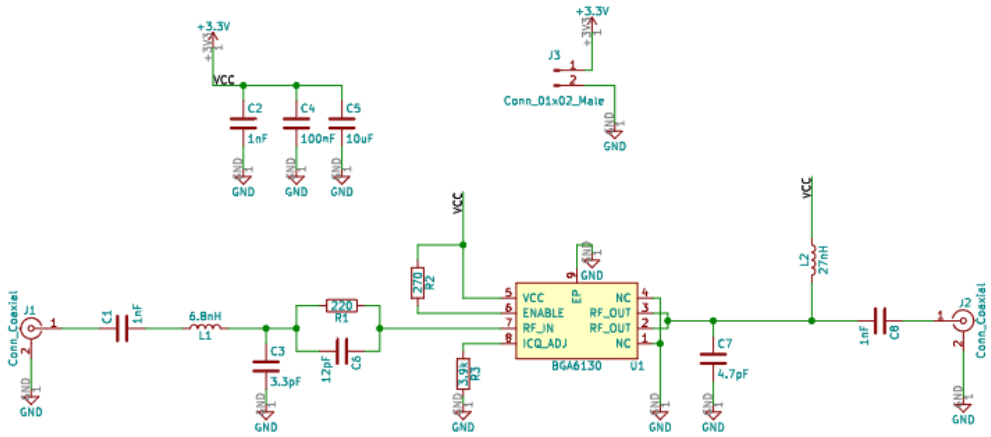
Az erősítő választást kevésbé limitálta a frekvenciasáv; ezzel szemben a 3 V tápfeszültség kitétel igen. Az elvárt (legalább 27 dBm) teljesítményt elérő IC-k nagy része 5 V vagy magasabb tápfeszültségről üzemel. Ha viszont egy rádióhullámú tranzisztorról van szó, általában nincs megadva az adatlapon, hogy a nominális feszültség alatt, hogy teljesít az eszköz. Azt majdnem biztosan állítom, hogy nem létezik 3.3 V nominális drain-source feszültségről üzemelő, 27 dBm kimenő teljesítménnyel rendelkező tranzisztor. Ezért ezeket a paramétereket szükséges kimérni.

A következő szekciókban méréshez tervezett teszt áramkörök fognak következni. Az ezekhez tartózó mérés úgy fog zajlani, hogy egy megfelelő teljesítményű jelforrást kapcsolunk a bemeneti SMA csatlakozóra ,például egy SDR-t. A kimeneti SMA csatlakozóra pedig egy spektrumanalizátort csatlakoztatunk amelyen figyeljük a kiadott teljesítményét. A tápfeszültséget egy labortáp szolgáltatja.

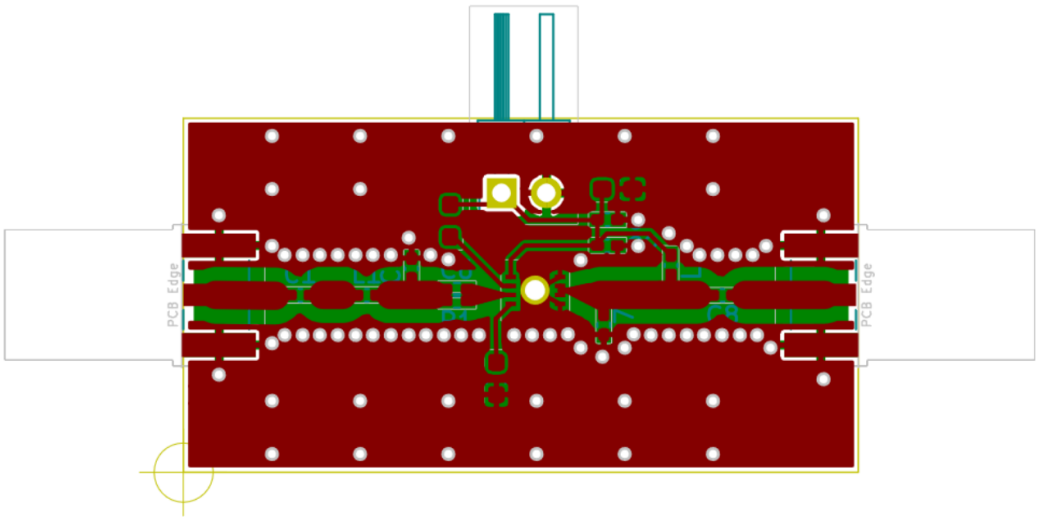
Ez alól kivételt képez az ADL5606 típusú IC, mely az ADRF6073 szintézer ICt tartalmazó áramkörrel volt kimérve.

3.5.1. BGA6130

Ez az integrált áramkör nem egy diszkrét tranzisztor és 3 voltról is üzemel, viszont szükséges a mérése mivel a elvárt frekvenciához közeli mérési adatokat nem tartalmaz az adatlapja. Ezzel szemben azt állítja, hogy felmegy 2700 MHz-ig a sávszélessége[17]. Az erősítő olyan kiemelkedően jó hatásfokkal (PAE:55 százalék) büszkélkedik[17], hogy a kimérése ennek az ellenőrzéséhez is szükséges. A mérési áramkör magát az IC-t, a hozzá tartozó impedancia illesztő hálózatot, tápellátást és bemeneti plusz kimeneti SMA csatlakozókat tartalmaz.



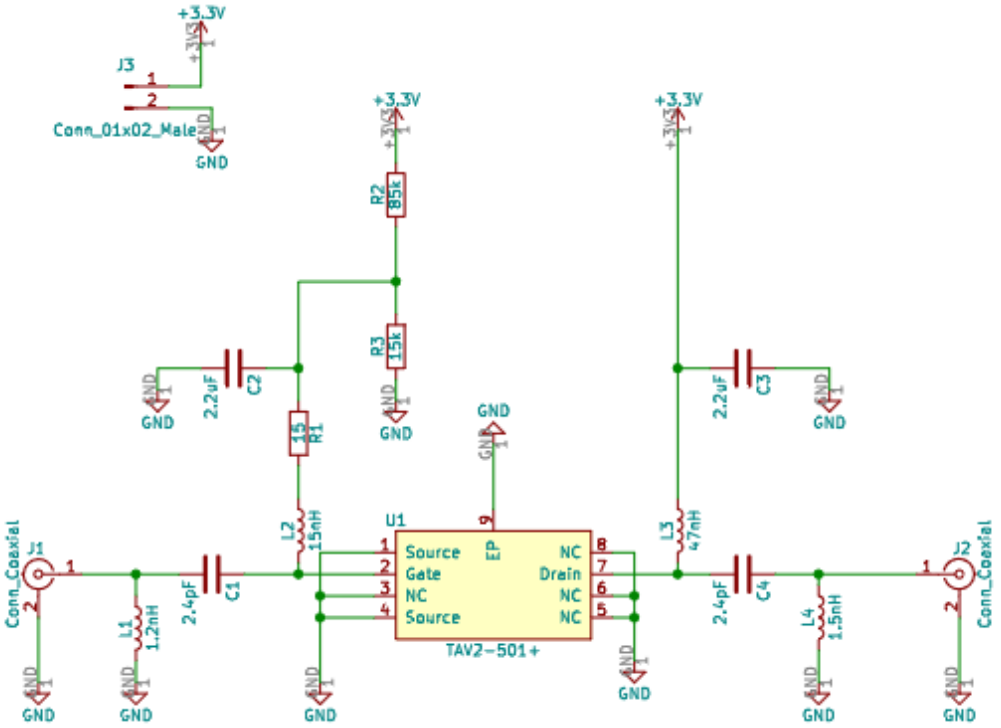
3.23. ábra. BGA6130 erősítő mérési kapcsolás



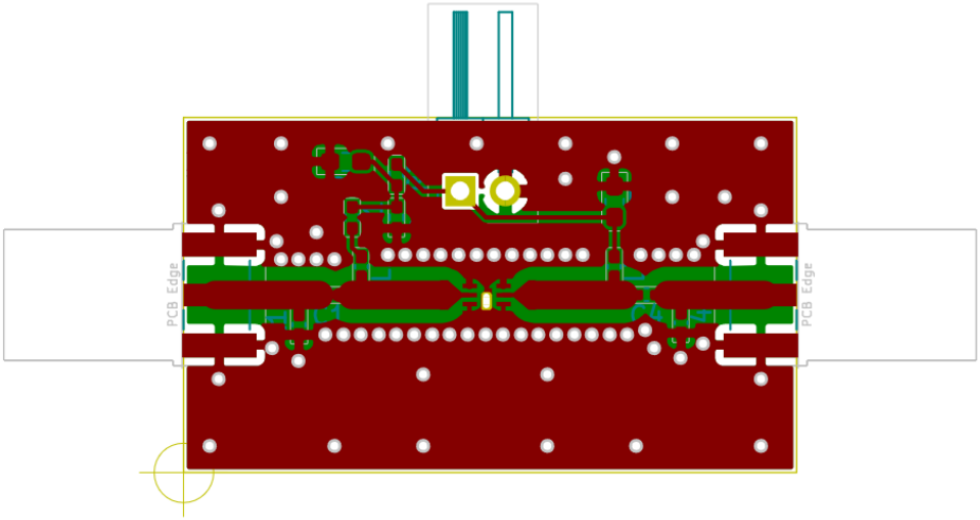
3.24. ábra. BGA6130 erősítő mérési panel

3.5.2. TAV2-501+

Ez a tranzisztor megfelelő kimenő teljesítménnyel és sávszélességgel rendelkezik viszont csak 4.5 V drain-source feszültségen mért adatai vannak [16]. Ezért, hogy ki tudjuk mérni 3.3 volton, ehhez is terveztem egy erősítő áramkört. Ez tartalmazza a referencia erősítő kapcsolást [16], gate-source feszültség beállító hálózatot, impedancia illesztést és bemeneti plusz kimeneti SMA csatlakozókat.



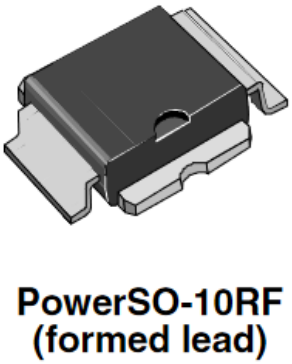
3.25. ábra. TAV2-501+ erősítő mérési kapcsolás



3.26. ábra. TAV2-501+ erősítő mérési panel

3.5.3. PD20010-E

Ez a tranzisztor sokkal nagyobb kimeneti teljesítménnyel és nominális feszültséggel rendelkezik mint ami tervezett [18]. De a MASAT-1 küldetés során is egy ehhez nagyon hasonló, csak más sávszélességű modellel sikereket értek el a laboratóriumban. Ezért alacsonyabb feszültség- és teljesítményszintekkel ki kell mérni, mivel nagy előnye az előzőekkel szemben, hogy a tokozása sokkal jobban hűthető.

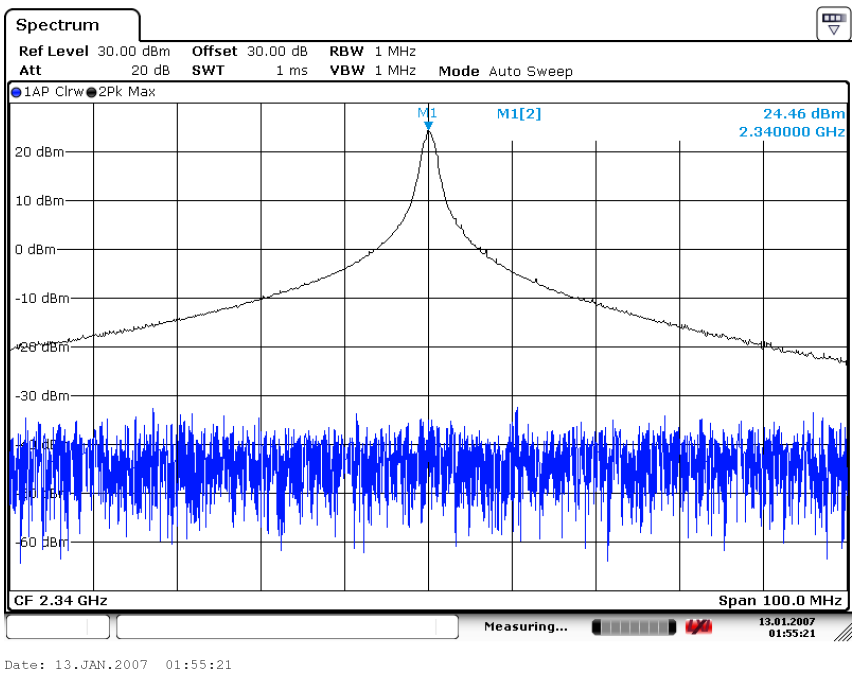


PowerSO-10RF
(formed lead)

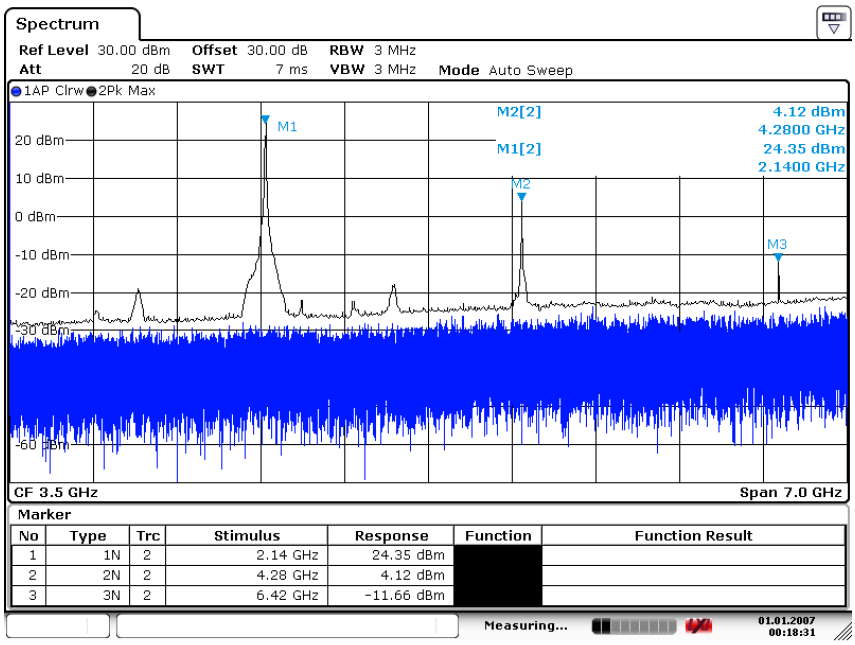
3.27. ábra. A PD20010-E tokozása

3.5.4. ADL5606

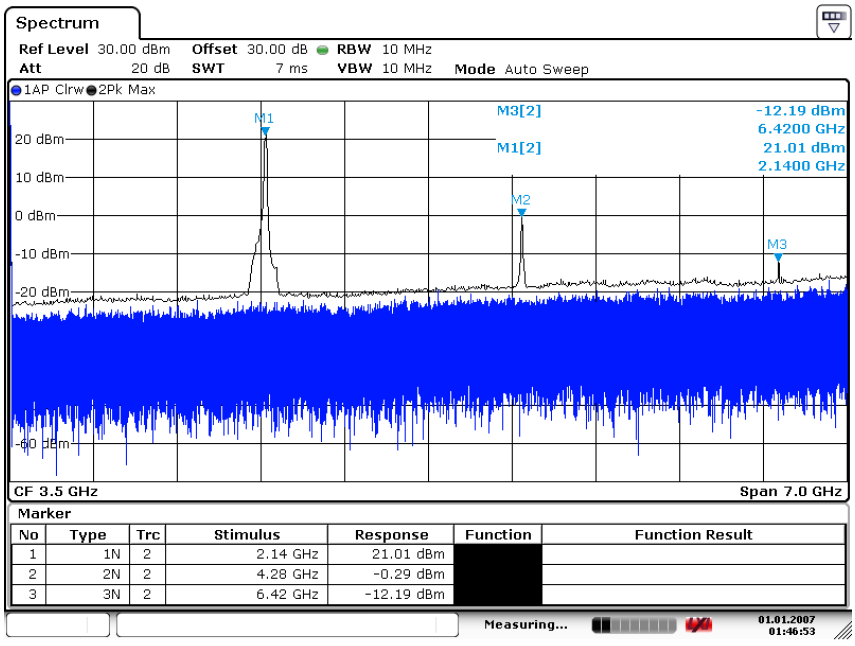
Ez az IC 5V -os feszültségről üzemel és előnyös adatlapi paraméterekkel rendelkezett. Viszont, mivel könnyen gerjedt és nem tudta hozni az elvárt teljesítményt 3V környékén 3.30 nem ez lesz a megfelelő erősítő.



3.28. ábra. Végfok kimenetén végzett mérés



3.29. ábra. A végfok harmonikusai



3.30. ábra. A végfok spektruma 3.4V tápfeszültségen

4. fejezet

Összegzés

E dokumentumban leírt feladatok végrehajtásánál tapasztalatot szereztem nagyfrekvenciás kapcsolások és áramköri lemezek tervezésében, megvalósításában és ezek kiméréséhez használt eszközök használatában. Megismertem a PIC és TI mikrovezérlő családot és tapasztalatot szereztem programozásukban.

Elmélyedtem a digitális modulációk elméletében, ezek után megismerkedtem a GNU-Radio környezettel és használtam többféle szoftverrádiót is.

A szimuláció írás közben elmélyítettem a C és Python tudásom, megismertem a Shell script nyelvet és rengeteg tapasztalatot szereztem a Linux operációs rendszerrel kapcsolatban.

Az utóbbi heteket a CC1352P chip programozásával töltöttem és nagy reményeket mutat arra, hogy a repülő példányon szerepeljen. Az erősítők mindegyik nagyon ígéretes és a következő hónapokban ki fog derülni melyik lesz majd alkalmazva végfok erősítőként.

A mérnöki példány előállítás sürgető, de én személy szerint úgy látom kész lesz időben.

Köszönetnyilvánítás

Köszönet Dr. Dudás Leventének a rengeteg segítségért és rá szánt időért, még így a járvány idején is.

Irodalomjegyzék

[1] Miklós Barnabás: Műholdfedélzeti QPSK adó tervezése és megvalósítása

[2] Miklós Barnabás: Műholdfedélzeti QPSK adó fejlesztése

[3] http://hvt.bme.hu

[4] https://www.unilim.fr/pages_perso/vahid/notes/ber_awgn.pdf

[5] Roland Best - Costas Loops Theory, Design, and Simulation

[6] https://www.analog.com/media/en/technical-documentation/data-sheets/ADRF6703.pdf

[7] http://ww1.microchip.com/downloads/en/DeviceDoc/60001324b.pdf

[8] https://www.analog.com/media/en/technical-documentation/data-sheets/ADL5606.pdf

[9] http://www.ti.com/lit/ds/symlink/tps62177.pdf

[10] https://www.silabs.com/documents/public/data-sheets/efr32mg12-datasheet.pdf

[11] https://www.silabs.com/documents/public/application-notes/an0002.1-efr32-efm32-series-1-hardware-design-considerations.pdf

[12] https://www.silabs.com/documents/public/application-notes/an930.1-efr32-series-1.pdf

[13] https://www.ti.com/lit/ds/symlink/cc1352p.pdf?ts=1620844625885

[14] https://www.ti.com/lit/an/swra640e/swra640e.pdf

[15] https://www.ti.com/lit/zip/SWRC363

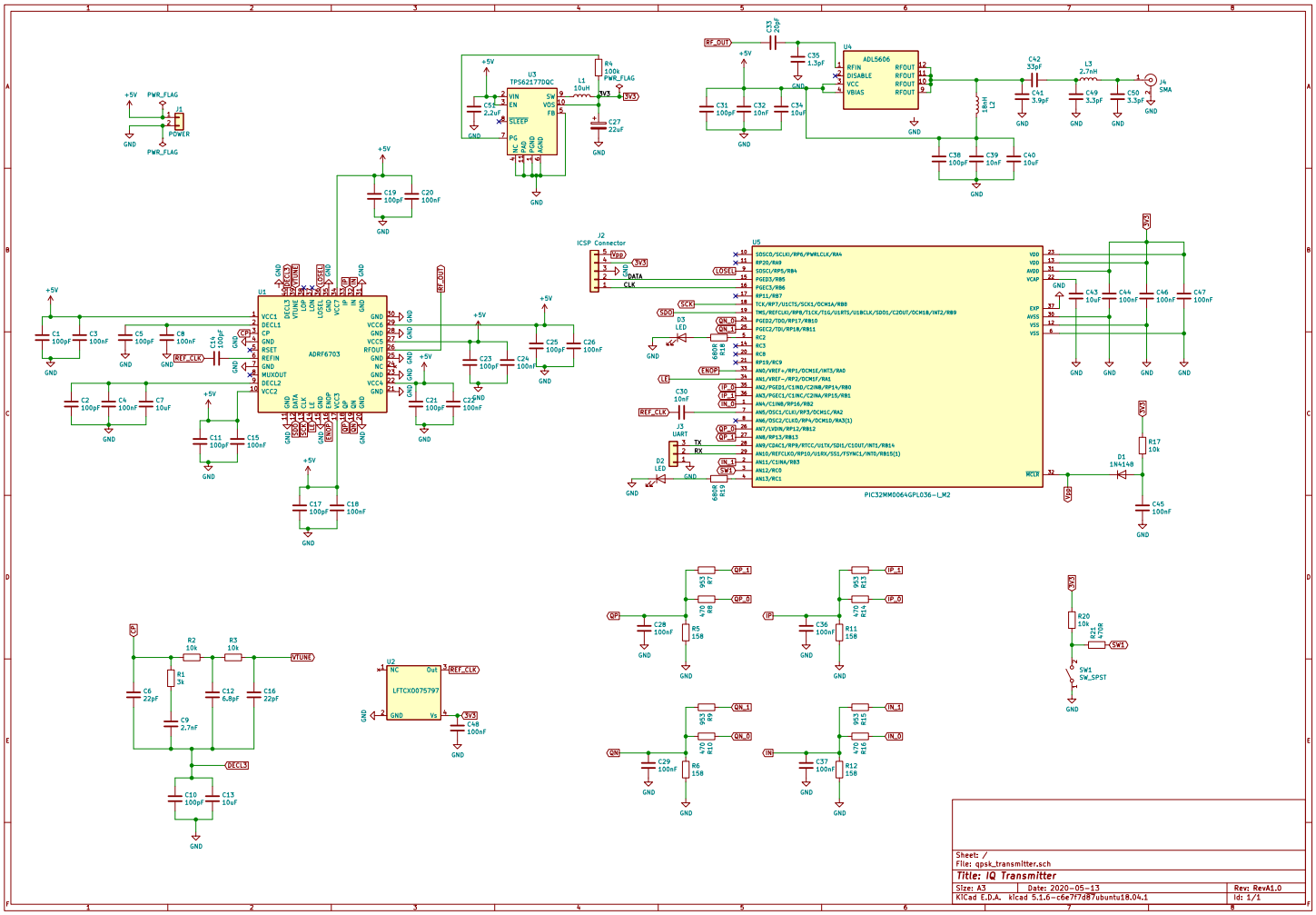
[16] https://www.minicircuits.com/pdfs/TAV2-501+.pdf

[17] https://www.nxp.com/docs/en/data-sheet/BGA6130.pdf

[18] https://www.st.com/resource/en/datasheet/pd20010-e.pdf

Ábrák jegyzéke

Függelék



4.1. ábra. ADRF6703 adó kapcsolási rajz

4.1. Listing. randombytes.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdint.h>

int main(int argc, char **argv)
{

        unsigned long n=128;
        if(argc>1) n=atol(argv[1]);
        unsigned long i;
        int j=0;


        srand(time(NULL));

        for(i=0;i<n;i++){
                uint8_t random=rand()&0xff;

                printf("%c",random);

        }
        return 0;
}
```

4.2. Listing. byte2symbols.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <complex.h>
#include <stdint.h>
#include <math.h>

int main(int argc, char **argv)
{
        uint8_t in[1];
        uint8_t sym_buffer = 0;
        FILE * bytes = fopen("bytes.txt","w");
        FILE * onesandzeros= fopen("onesandzeros.txt","w");

        float one_per_sqrt_two=1/sqrt(2);
        int k;
        int i;
        int j = 0;

        complex float out[1];

        while(1){
```

```
k=fread(in,1,1,stdin);
if(feof(stdin)) break;
if(k>0){
        for(i=0;i<4;i++){
                sym_buffer = in[0] & (0b11000000
                    >>(2*i));

                sym_buffer = sym_buffer << (2*i)
                    ;
                //sym_buffer = 0;
                switch (sym_buffer)
                {
                case 0b00000000:
                        out[0] = -
                            one_per_sqrt_two -
                            one_per_sqrt_two*I;
                        fprintf(onesandzeros,"00
                            ");
                        break;
                case 0b01000000:
                        out[0] = -
                            one_per_sqrt_two +
                            one_per_sqrt_two*I;
                        fprintf(onesandzeros,"01
                            ");
                        break;
                case 0b10000000:
                        out[0] =
                            one_per_sqrt_two -
                            one_per_sqrt_two*I;
                        fprintf(onesandzeros,"10
                            ");
                        break;
                case 0b11000000:
                        out[0] =
                            one_per_sqrt_two +
                            one_per_sqrt_two*I;
                        fprintf(onesandzeros,"11
                            ");
                        break;
                default:
                        out[0] = 404;
                }

                        fwrite(out,sizeof(
                            complex float),1,
                            stdout);
```

```
                        }


                }
                else{
                        usleep(100);
                }
        }
        return 0;
}
```

4.3. Listing. agwn.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <complex.h>
#include <stdint.h>
#include <math.h>
#include <time.h>



double rnd() { return rand()/(1.0+RAND_MAX); }

complex double gwn()
{
        double x,y,a,rr;
        do{
                x=rnd()*2-1;
                y=rnd()*2-1;
                rr=x*x+y*y;
        } while(rr>=1.0||rr==0.0);
        a=sqrt(-log(rr)/rr);
        return a*x+a*y*I;
}


int main(int argc, char **argv){

        double sigma;
        double SNR = 10;
        if(argc>1) SNR = atof(argv[1]);
        sigma = pow(10.0,(-SNR)/20);



    complex float in[1];
        int k;
```

```
        int i=0;

        complex float out[1];
        while(1){
                k=fread(in,sizeof(complex float),1,stdin);
                if(feof(stdin)) break;
                if(k>0){
                                complex double noise= gwn();
                                    out[0]=in[0] + 1/sqrt(2)*noise*
                                       sigma;
                                    i++;
                fwrite(out,sizeof(complex float),1,stdout);
                }
                else{
                        usleep(100);
                }
        }

        return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <complex.h>
#include <stdint.h>
#include <math.h>
#include <time.h>

int main(int argc, char **argv)
{

        uint32_t multiplier = 10;



        if(argc>1) multiplier = atol(argv[1]);


    FILE * tmpfile = fopen( "incremented.txt", "w");

        int k;
        float i=0;
        complex float out[1];
        complex float in[1];
        complex float integrator1=0;
        complex float derivator1_out=0;
        complex float derivator1=0;
        unsigned long int samples_out=0;
        int j=10;
        while(1){
                k=fread(in,sizeof(complex float),1,stdin);
                if(feof(stdin)) break;
                if(k>0){

                        derivator1_out=in[0] -  derivator1;
                        derivator1=in[0];
                        j=0;
                        integrator1=integrator1 + derivator1_out
                            ;


                        for(j=0;j<multiplier;j++){
                        out[0]=integrator1;

                        fwrite(out,sizeof(complex float),1,
                            stdout);
                        fprintf(tmpfile,"%f,%f,%ld\n",creal(out
                            [0])
```

```
                        ,cimag(out[0]),samples_out++ );

                        }

                }
                else{
                        usleep(100);
                }
        }
    fclose(tmpfile);
        return 0;
}
```

4.5. Listing. cnco.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <complex.h>
#include <stdint.h>
#include <math.h>
#include <time.h>



int main(int argc, char **argv)
{

    //uint16_t T = 273;
    uint32_t f_sample = 500;
    uint32_t f_mix = 1;



        if(argc>1) f_sample = atol(argv[1]);
    if(argc>2) f_mix = atol(argv[2]);


    FILE * tmpfile = fopen( "cnco.txt", "w");// txt to plot

        int j=0;
    int k=0;
    long unsigned int samplenum=0;
        long double i=0;

    complex float * phasor = malloc(f_sample*sizeof(complex
       float));
    complex float in[1];
        complex float out[1]={0};
```

```
    for(i=0;i<=2*M_PI;i=i+(2*M_PI/f_sample)){

        phasor[j]=cexp(i*I);
        j++;


    }

    j=0;

        while(1){
                k=fread(in,sizeof(complex float),1,stdin);
                if(feof(stdin)) break;
                if(k>0){
            out[0]= creal(phasor[j])*creal(in[0])
            +cimag(phasor[j])*cimag(in[0])*I;
            j=j+f_mix;
            if(j>=f_sample) j=0;

            fprintf(tmpfile,"%.12f, %.12f, %ld\n",
            creal(out[0]),cimag(out[0]),samplenum++);

            fwrite(out,sizeof(complex float),1,stdout);

                }
                else{
                        usleep(100);
                }
        }
    fclose(tmpfile);
    free(phasor);
        return 0;
}
```

4.6. Listing. cnco2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <complex.h>
#include <stdint.h>
#include <math.h>
#include <time.h>



int main(int argc, char **argv)
{
```

```
uint32_t f_sample = 500;
uint32_t f_mix = 1;



    if(argc>1) f_sample = atol(argv[1]);
if(argc>2) f_mix = atol(argv[2]);

FILE * tmpfile = fopen( "cnco2.txt", "w");// txt to plot

    int32_t j=0;
uint32_t k=0;
long unsigned int samplenum=0;
    long double i=0;

complex float * phasor = malloc(f_sample*sizeof(complex
   float));
complex float in[1]={0};
    complex float out[1]={0};
//printf("cnco2 started\n");

//j=f_sample-1;
for(i=0;i<=2*M_PI;i=i+(2*M_PI/f_sample)){

    phasor[j]=cexp(i*I);

    j++;


}


j=0;

    while(1){
            k=fread(in,sizeof(complex float),1,stdin);
    //printf("in %f,    ",creal(in[0]));
            if(feof(stdin)) break;
            if(k>0){
     //printf(" j= %d\n",j);
     out[0]= conj(phasor[j])*in[0];
     j=j+f_mix;
     j=j % f_sample;
     fprintf(tmpfile,"%.12f, %.12f, %ld\n",creal(out[0]),
        cimag(out[0]),samplenum++);
     fwrite(out,sizeof(complex float),1,stdout);


            }
```

```
                else{
                        usleep(100);
                }
        }
    fclose(tmpfile);
    free(phasor);
        return 0;
}
```

4.7. Listing. decrementbinary.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <complex.h>
#include <stdint.h>
#include <math.h>
#include <time.h>
#include <string.h>

int main(int argc, char **argv)
{

        uint32_t multiplier = 10;

    char file_name[64]="decremented.bin";


        if(argc>1) multiplier = atol(argv[1]);
    if(argc>2) strcpy(file_name,argv[2]);

   //printf("filename : %s\n\n",file_name);
    FILE * tmpfile = fopen( file_name, "w");


        int k;
        int i=0;
    complex float in[1];
        complex float out[1]={0};
        complex float integrator1=0;
        complex float integrator1_out=0;
        complex float derivator1=0;
    int j=0;

        while(1){
                k=fread(in,sizeof(complex float),1,stdin);
                //printf("in : %f\n",creal(in[0]));
                if(feof(stdin)) break;
                if(k>0){
```

```
                        integrator1 = integrator1 + in[0];
                        j++;

                        if(j==multiplier){

                        out[0]=integrator1-derivator1;
                        derivator1=integrator1;
                        out[0]=out[0]/multiplier;



                        fwrite(out,sizeof(complex float),1,
                            tmpfile);
                        fwrite(out,sizeof(complex float),1,
                            stdout);

                        j=0;
                        }


                }
                else{
                        usleep(100);
                }
        }
    fclose(tmpfile);
        return 0;
}
```

4.8. Listing. lpf.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <complex.h>
#include <stdint.h>
#include <math.h>
#include <time.h>

#define FILTER_SIZE 20


int main(int argc, char **argv)
{
    fflush(stderr);

    FILE * tmpfile = fopen( "lpf_out2.txt", "w");// txt to plot

    long unsigned int samplenum=0;
```

```
    uint32_t f_sample = 500;
    uint32_t f_cutoff = 1;

    if(argc>1) f_sample = atol(argv[1]);
    if(argc>2) f_cutoff = atol(argv[2]);

    complex float in[1];
        complex float out[1]={0};

        complex double x[2]={0,0};
    double alpha=0.5;
    int k=0;
    x[1]=0;
        while(1){
                k=fread(in,sizeof(complex float),1,stdin);
                if(feof(stdin)) break;
                if(k>0){
            x[0]=in[0];
            out[0]=x[0]+x[1]*(1-alpha);
            x[1]=out[0];
            fprintf(tmpfile,"%.12f, %ld\n",creal(out[0]),
               samplenum++);

            out[0]=0;

                }
                else{
                        usleep(100);
                }
        }
    fflush(stderr);
    fclose(tmpfile);
        return 0;
}
```

4.9. Listing. phasediff.c

```
#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <unistd.h>
#include <math.h>

int main(int argc, char **argv)
{

        complex float in1[1];
        complex float in2[1];
    complex float out[1];
        long unsigned int samplenum=0;
```

```
    FILE * output = fopen( "phase_diff.txt", "w");// txt to plot

    FILE * f_input_normal = fopen( "decremented.bin", "r");
    FILE * f_input_limited= fopen( "decrementedlimited.bin", "r"
       );

        //printf("\n\n");
        while(1){
                int k=fread(in1,sizeof(complex float),1,
                   f_input_normal);
                int j=fread(in2,sizeof(complex float),1,
                   f_input_limited);
                if(feof(f_input_normal)) break;
                if(k>0){

            out[0]=creal(in1[0])*cimag(in2[0]) - creal(in2[0])*
               cimag(in1[0]);
                        fwrite(out,sizeof(complex float),1,
                           stdout);
            fprintf(output,"%.12f, %ld\n",creal(out[0]),
               samplenum++);

                }
                else{
                        usleep(10);
                }
        }
        fflush(stderr);
        fclose(output);
        return 0;
}
```

4.10. Listing. iqlevels.m

```
Us=3.3
R_Load=945
Un=0.4
Up=0.7

R=optimvar('R',3) % optimalizalasi problemakent viszem be
eq1=(1-Us)/R(2)+(1-Us)/R(3)+(1-0.1)/R_Load+1/R(1)==0; %
    csomoponti egyenletek
eq2=(0.1-1)/R_Load+0.1/R(2)+0.1/R(3)+ 0.1/R(1)==0;%a vart fesz.
    ertekekkel
eq3=(Up-Un)/R_Load+(Up-Us)/R(3)+Up/R(2)+Up/R(1)==0;
eq4=(Un-Up)/R_Load+(Un-Us)/R(2)+Un/R(3)+Un/R(1) == 0;


prob = eqnproblem;
```

```
prob.Equations.eq1 = eq1;
prob.Equations.eq2 = eq2;
prob.Equations.eq3 = eq3;
prob.Equations.eq4 = eq4;
show(prob)
R0.R=[100 100 100];
[sol,fval,exitflag] = solve(prob,R0)
disp(sol.R)
```

4.11. Listing. costas.c

```
#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <unistd.h>
#include <math.h>
#include <stdint.h>

complex float limit(complex float symbol){

        float one_per_sqrt_two = 0.70710678118;
        complex float out = 0;

        if((creal(symbol)<0)&& (cimag(symbol)<=0))    {
                out=-one_per_sqrt_two - one_per_sqrt_two*I;;
                return out;

        }else if ((creal(symbol)<=0) && (cimag(symbol)>0) ){
                out=-one_per_sqrt_two + one_per_sqrt_two*I;
            return out;


        }else if( (creal(symbol)>=0) && (cimag(symbol)>0)){
                out=one_per_sqrt_two + one_per_sqrt_two*I;
                return out;

        }else if( (creal(symbol)>0) && (cimag(symbol)<=0)){
                out=one_per_sqrt_two - one_per_sqrt_two*I;
                return out;
        }

}

int main(int argc, char **argv)
{
        float p_accumulator=0;
        float p_error=0;

        uint32_t f_sample = 1000000;
        long int osc_index=0;
```

```
    FILE * LUT = fopen( "../../output/bin/cnco_lut.bin", "r"
       );// txt to plot

    FILE * PREV_LOOP_DATA = fopen( "../../output/bin/
       prev_loop_data.bin", "r");

    if(PREV_LOOP_DATA!=NULL){
            long int buffer[1]={0};
            fprintf(stderr,"succesfull file read\n");
            fread(&osc_index,sizeof(long int),1,
               PREV_LOOP_DATA);
            fread(&p_accumulator,sizeof(float),1,
               PREV_LOOP_DATA);
            fprintf(stderr,"prev osc index: %ld  , prev
               phase accumulator: %lf\n",osc_index,
               p_accumulator);

    }else{
            fprintf(stderr,"cannot read file\n");

    }



    uint32_t k=0;
    long double i=0;

    if(argc>1) f_sample = atol(argv[1]);


complex float * phasor = malloc(f_sample*sizeof(complex
   float));
complex float in[1];
    complex float out[1]={0};
    long int temp_index=0;

    fflush(stderr);
for(i=0;i<=f_sample;i++){

    fread(in,sizeof(complex float),1,LUT);
    ///printf("%lf\n", creal(in[0]));
    phasor[temp_index]=in[0];
    temp_index++;


}
```

```
//if(argc>2) osc_index = atol(argv[2]);
fprintf(stderr,"----------new set----------\n");

//printf("\n\n");

while(1){
        int k=fread(in,sizeof(complex float),1,stdin);
        if(feof(stdin)) break;
        if(k>0){
                out[0]=in[0]*phasor[osc_index];

    p_error=creal(out[0])*cimag(limit(out[0])) - creal(
       limit(out[0]))*cimag(out[0]);
                p_accumulator+=p_error;
                if(p_accumulator<0) p_accumulator+=
                   f_sample;
                osc_index =(osc_index+(long)
                   p_accumulator)%f_sample;

                //fwrite(out,sizeof(complex float),1,
                   stdout);//  forwarding bits
                fprintf(stderr,"%f\n",p_accumulator);
                fwrite(out,sizeof(complex float),1,
                   stdout);

        }
        else{
                usleep(10);
        }
}


fprintf(stderr,"opening for writing\n");
FILE * LOOP_DATA = fopen( "../../output/bin/
   prev_loop_data.bin", "w");

if(LOOP_DATA==NULL){
        fprintf(stderr,"cannot open file for writing\n")
           ;
}else{
        fprintf(stderr,"successfully opened file for
           writing\n");
        fwrite(&osc_index,sizeof(long int),1,LOOP_DATA);
        fwrite(&p_accumulator,sizeof(float),1,LOOP_DATA)
           ;

}
fclose(LUT);
```

```
        fclose(PREV_LOOP_DATA);
        fclose(LOOP_DATA);
        fflush(stderr);
        return 0;
}
```

4.12. Listing. sim.sh

```
#!/bin/bash

#---PATHS---#
BLD="../../build"
OUT="../../output/bin"
PYS="../python"

#---PARAMS---#
n=2
SNR=10

rotation=-45
backrotation=45
phaseerror=20

bit_per_sec=1000
sym_per_sec=bit_per_sec/2

multiplier=1
fs=1000000
fm=100000
fm_b=99000

costas_prev_index=0

fc=10000


#---SIMULATION---#
export i=0
#mkfifo $OUT/symbols_pipe
#mkfifo $OUT/cnco_tx_pipe
#mkfifo $OUT/cnco_rx_pipe

$BLD/generate_cnco_lut.out $fs
python3 $PYS/complex_plot_try.py &
#python3 $PYS/rt_plot_process.py &

rm $OUT/prev_loop_data.bin

start=`date +%s%N`
```

```
for i in {0..100}
do
    $BLD/randombytes.out $n | \
    #tee $OUT/bytes.bin | \
    $BLD/byte2symbol.out | \
    tee $OUT/symbols.bin | \
    #$BLD/ftee.out $OUT/symbols_pipe | \
    #$BLD/ftee.out $OUT/cnco_rx_pipe | \
    #$BLD/agwn.out $SNR | \
    #tee /dev/stderr | \
    #./rotate.out $phaseerror | \
    #$BLD/agwn.out $SNR | \
    #$BLD/cf2reim2.out | \
    $BLD/increment.out $multiplier | \
    tee $OUT/incremented.bin | \
    $BLD/agwn.out $SNR | \
    $BLD/cnco.out $fs $fm | \
    tee $OUT/cnco_tx.bin | \
    #$BLD/ftee.out $OUT/cnco_tx_pipe | \
    #./rotate.out $phaseerror | \
    #            |
    #         CHANNEL
    #            |
    $BLD/cnco_conj.out $fs $fm_b | \
    $BLD/costas.out $fs |\
    tee $OUT/cnco_rx.bin | \
    #$BLD/agwn.out $SNR | \
    tee > $OUT/noisy.bin
    #./agwn.out $SNR | \
    #./rotate.out $backrotation | \
    #$BLD/decrement_binary.out $multiplier | \
    #$BLD/limiter.out | \
    #tee > $OUT/decrementedlimited.bin
    #$BLD/phase_diff.out | \
    #$BLD/derivator.out | \
    #$BLD/lpf.out $fs $fc
    sleep 0.5
    #echo $i" iter"
    #sleep 1

    #./symbols2bits.out | \
    # tee > decoded.bin
    # ./berr.out $SNR | \
    # tee -a bercurve.txt
done
end=`date +%s%N`;

#echo `expr $end - $start`
#---PLOTTING---#
```

```
    #python3 $PYS/plot.py --n $n
    #python3 $PYS/scatterplot.py
    #python3 $PYS/plot_co.py
    #python3 $PYS/plot_costas.py
```

4.13. Listing. rtplot.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import argparse
import collections
import os
import pdb
import math
from itertools import count

ix=os.environ["i"]

class Signal:
    output_path='../../output/bin/'
    t_window_length = 160
    id_count = count(0)
    signal_count = [0]

    def __init__(self,signal_file):
        self.path = self.output_path + signal_file
        self.buffer = collections.deque(maxlen = self.
            t_window_length)
        self.prev_modified_date = os.stat(self.path).st_mtime_ns
        for i in range(self.t_window_length):
            self.buffer.append(np.complex64(0.0 + 0.0j))
        self.id = next(self.id_count)
        self.signal_count[0] = self.id + 1
        self.name = signal_file

    def append(self,value):
        self.buffer.append(value)

    @classmethod
    def set_output_path(cls,path):
        cls.output_path = path

    @classmethod
    def set_t_window_length(cls,window_length):
        cls.t_window_length = window_length

    @classmethod
    def get_signal_count(cls):
```

```
        return cls.signal_count[0]

class Plotter:

    signal_list = []
    signal_plot_types = []
    plot_list = []
    def __init__(self,signal_count):
        self.fig, self.ax = plt.subplots(signal_count)

    def add_signal(self,plot_type,signal,gui):
        self.signal_plot_types.append(plot_type)
        self.gui = gui
        self.signal_list.append(signal)


    def init_plot(self):
        for signal in self.signal_list:
            if self.signal_plot_types[signal.id] =="line":
                #line_plot = self.ax[signal.id].plot(np.real(
                    signal.buffer),'-')
                #self.plot_list.append(line_plot[0])
                self.plot_list.append(self.ax[signal.id].plot(np
                    .real(signal.buffer),'-',label='real'))
                self.ax[signal.id].set_ylim([-1.5,1.5])
                self.ax[signal.id].set_ylabel("real values")
                self.ax[signal.id].set_xlabel("samples")
                self.ax[signal.id].set_title(signal.name)
            elif self.signal_plot_types[signal.id] =="scatter":
                #scatter_plot = self.ax[signal.id].plot(np.real(
                    signal.buffer),np.imag(signal.buffer),'go',
                    markersize=1)
                #self.plot_list.append(scatter_plot[0])
                self.plot_list.append(self.ax[signal.id].plot(np
                    .real(signal.buffer),np.imag(signal.buffer),'
                    go',markersize=1))
                self.ax[signal.id].grid(b=True,)
                self.ax[signal.id].set_xlabel('Re')
                self.ax[signal.id].set_ylabel('Im')
                self.ax[signal.id].set_title(signal.name)
                self.ax[signal.id].axis('square')
                self.ax[signal.id].set_ylim([-2,2])
                self.ax[signal.id].set_xlim([-2,2])


    def update_plot(self):
        for signal in self.signal_list:
            stat = os.stat(signal.path) # load metadata
```

```
                modified_date = stat.st_mtime_ns
                graph = None
                if modified_date > signal.prev_modified_date: #if
                    the frame data got updated
                    graph = np.fromfile(signal.path,dtype=np.
                        complex64) # load the data from the file

                    for i in graph:
                        signal.append(i) # append new data

                    if self.signal_plot_types[signal.id] =="line":
                        self.plot_list[signal.id][0].set_ydata(np.
                            real(signal.buffer))
                    elif self.signal_plot_types[signal.id] ==
                        "scatter":
                        self.plot_list[signal.id][0].set_data(np.
                            real(signal.buffer),np.imag(signal.buffer
                            ))


                signal.prev_modified_date = modified_date

            return self.plot_list

window_length=320


Signal.set_output_path('../../output/bin/')
Signal.set_t_window_length(320)


plotter = Plotter(4)
plotter.add_signal("line",Signal("symbols.bin"),0)
plotter.add_signal("line",Signal("cnco_tx.bin"),0)
plotter.add_signal("line",Signal("cnco_rx.bin"),0)
plotter.add_signal("scatter",Signal("noisy.bin"),0)
plotter.init_plot()


def animate(i):
    return plotter.update_plot()

ani = animation.FuncAnimation(
    plotter.fig, animate, interval=500,blit=False,save_count=50)
plt.autoscale(False)
plt.tight_layout()
plt.show()
```

4.14. Listing. picmain.c

```
/**
  Generated main.c file from MPLAB Code Configurator

  @Company
    Microchip Technology Inc.

  @File Name
    main.c

  @Summary
    This is the generated main.c using PIC24 / dsPIC33 / PIC32MM
        MCUs.

  @Description
    This source file provides main entry point for system
        initialization and application code development.
    Generation Information :
        Product Revision  :  PIC24 / dsPIC33 / PIC32MM MCUs -
            1.169.0
        Device            :  PIC32MM0064GPL036
    The generated drivers are tested against the following:
        Compiler          :  XC16 v1.50
        MPLAB             :  MPLAB X v5.40
*/

/*
    (c) 2020 Microchip Technology Inc. and its subsidiaries. You
        may use this
    software and any derivatives exclusively with Microchip
        products.

    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO
        WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE,
        INCLUDING ANY IMPLIED
    WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS
        FOR A
    PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP
        PRODUCTS, COMBINATION
    WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

    IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT,
        SPECIAL, PUNITIVE,
    INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF
        ANY KIND
    WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF
        MICROCHIP HAS
    BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE
        FORESEEABLE. TO THE
```

```
    FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY
        ON ALL CLAIMS IN
    ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT
        OF FEES, IF ANY,
    THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

    MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR
        ACCEPTANCE OF THESE
    TERMS.
*/

/**
  Section: Included Files
*/
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/mcc.h"
#include <stdio.h>
#include <inttypes.h>
#include <stdbool.h>
#include "random_bytes.h"
/*
                         Main application
 */

#define NUM_CHARS 256
#define QP_0    (1<<12)
#define QP_1    (1<<13)
#define QN_0    (1<<10)
#define QN_1    (1<<11)
#define IP_0    (1<<0)
#define IP_1    (1<<1)
#define IN_0    (1<<2)
#define IN_1    (1<<3)

#define IQ_TX_BUFFER_LENGTH 256
#define I_BITMASK 1
#define Q_BITMASK 0

#define TEST_ARRAY_S 8

#define PFD_FREQ 40000000 //40 MHz
#define STEP_FREQ  100000 // 100 kHz
#define MOD PFD_FREQ/STEP_FREQ

const uint32_t reg0_default = 0x0001C0;
const uint32_t reg1_default = 0x003001;
const uint32_t reg2_default = 0x001802;
```

```
bool isButtonReady = true;
uint32_t period = 31250*2;
uint16_t tx_buffer[IQ_TX_BUFFER_LENGTH];
uint8_t tx_buffer_index = 0;
uint8_t tx_buffer_send_index = 0;
bool iq_tx_ready = true;

typedef enum {
    LOW = 0,
    HIGH = 1
}out_state_t;

typedef struct{
    out_state_t i_state;
    out_state_t q_state;
}iq_out_t;

iq_out_t iq_current_state = {HIGH,HIGH};

void SetQHigh(void){
    QP_1_SetHigh();
    QP_0_SetHigh();
    QN_0_SetLow();
    QN_1_SetLow();
}
void SetQLow(){
    QN_1_SetHigh();
    QN_0_SetHigh();
    QP_0_SetLow();
    QP_1_SetLow();


}

void SetILow(){
    IN_1_SetHigh();
    IN_0_SetHigh();
    IP_0_SetLow();
    IP_1_SetLow();
}
void SetIHigh(void){
    IP_1_SetHigh();
    IP_0_SetHigh();
    IN_0_SetLow();
    IN_1_SetLow();
}
void SetIOff(){
    IP_1_SetLow();
    IP_0_SetLow();
```

```
    IN_0_SetLow();
    IN_1_SetLow();
}
void SetQOff(){
    QP_1_SetLow();
    QP_0_SetLow();
    QN_0_SetLow();
    QN_1_SetLow();
}
void Delay_us(int us){ //shitty delay function
    uint32_t start_time = CORETIMER_CountGet();
    while((CORETIMER_CountGet() - start_time) < 10*us){};

}

void UART1_SendString(char * string){

    uint16_t stringIterator = 0;
    uint8_t characterToSend = string[stringIterator];

    LED1_Toggle();
    do{

        if(UART1_IsTxReady()){
            UART1_Write(characterToSend);
            stringIterator++;
            characterToSend = string[stringIterator];
        }

    }while((characterToSend!='\0'));

    UART1_Write(characterToSend);

    LED1_Toggle();



}
void SW1_CallBack(void){
    if(isButtonReady){
        isButtonReady=0;
        //LED1_Toggle();

        if((MCCP1_TMR_Period32BitGet())== period)
        {
            MCCP1_TMR_Start();
        }
    }
```

```
}
void Handle_IQ_LATCH(iq_out_t iq_output){
    tx_buffer[tx_buffer_index] = 0;

    if(iq_current_state.q_state != iq_output.q_state){
        tx_buffer[tx_buffer_index] |= (QP_0|QP_1|QN_0|QN_1);
        iq_current_state.q_state = iq_output.q_state;
    }
    if(iq_current_state.i_state != iq_output.i_state){
        tx_buffer[tx_buffer_index] |= (IP_0|IP_1|IN_0|IN_1);
        iq_current_state.i_state = iq_output.i_state;
    }
    tx_buffer_index++;
}
void IQ_TX(uint8_t byte){
    uint8_t i = 0;
    uint8_t sym_buffer = 0;
    iq_out_t iq_output;
    for(i=0;i<4;i++){
        sym_buffer = byte & (0b11000000>>(2*i)); // masking out
            the actual bits

        sym_buffer = sym_buffer << (2*i); // shifting the bits
            to msb
        //sym_buffer = 0;
        switch (sym_buffer){
        case 0b00000000:
            iq_output.i_state = LOW;
            iq_output.q_state = LOW;
            break;
        case 0b01000000:
            iq_output.i_state = LOW;
            iq_output.q_state = HIGH;
            break;
        case 0b10000000:
            iq_output.i_state = HIGH;
            iq_output.q_state = LOW;
            break;
        case 0b11000000:
            iq_output.i_state = HIGH;
            iq_output.q_state = HIGH;
            break;
        default:
            iq_output.i_state = LOW;
            iq_output.q_state = LOW;
        }
        Handle_IQ_LATCH(iq_output);
    }
}
```

```
ADRF6703_SetRegister(uint8_t byte0, uint8_t byte1, uint8_t byte2
    )
{
    LE_SetLow();
    SPI2_Exchange8bit(byte0);
    SPI2_Exchange8bit(byte1);
    SPI2_Exchange8bit(byte2);
    Nop();
    LE_SetHigh();
    Nop();
    LE_SetLow();
    Nop();
}
void ADRF6703_SetFrequency(uint32_t freq_hz){

    uint8_t int_reg = 0;
    int_reg = freq_hz/PFD_FREQ;

    uint16_t frac_reg = 0;
    frac_reg= (freq_hz %PFD_FREQ)/ STEP_FREQ;
    uint16_t mod_reg = MOD;
    uint8_t reg_to_set[3] = {0,0,0};
    uint32_t reg_value_buffer = 0;

    ENOP_SetLow();

    uint32_t frac_reg_mask = 0x000002;
    reg_value_buffer =frac_reg_mask | (frac_reg<<3);
    reg_to_set[2] =(uint8_t)reg_value_buffer;
    reg_to_set[1] =(uint8_t)(reg_value_buffer>>8);
    reg_to_set[0] =(uint8_t)(reg_value_buffer>>16);

    ADRF6703_SetRegister(reg_to_set[0],reg_to_set[1],reg_to_set
        [2]);

    uint32_t mod_reg_mask = 0x000001;
    reg_value_buffer =mod_reg_mask | (mod_reg<<3);
    reg_to_set[2] =(uint8_t)reg_value_buffer;
    reg_to_set[1] =(uint8_t)(reg_value_buffer>>8);
    reg_to_set[0] =(uint8_t)(reg_value_buffer>>16);

    ADRF6703_SetRegister(reg_to_set[0],reg_to_set[1],reg_to_set
        [2]);

    uint32_t int_reg_mask = 0x000000;
    reg_value_buffer =int_reg_mask | (int_reg<<3);
    reg_to_set[2] =(uint8_t)reg_value_buffer;
    reg_to_set[1] =(uint8_t)(reg_value_buffer>>8);
    reg_to_set[0] =(uint8_t)(reg_value_buffer>>16);
```

```
    ADRF6703_SetRegister(reg_to_set[0],reg_to_set[1],reg_to_set
        [2]);

    ENOP_SetHigh();
}
void ADRF6703_Init(void){


    uint8_t reg0[3] = {0x00,0x01,0xc0};
    uint8_t reg1[3] = {0x00,0x0c,0x81};
    uint8_t reg2[3] = {0x00,0x06,0x42};
    uint8_t reg3[3] = {0x70, 0x00, 0x0b};// dither control
        default
    uint8_t reg4[3] = {0x02,0xa7,0xa4};
    uint8_t reg5[3] = {0x00, 0x00, 0xe5};// LO output disabled,
        modulator enabled          0b00000000, 0b00000000, 0
        b11010101
    uint8_t reg6[3] = {0x1e, 0xfd, 0x06};
    uint8_t reg7[3] = {0x00, 0x00, 0x07};// external VCO
        disabled                          0b00000000, 0b00000000,
         0b00000111

    LOSEL_SetLow();
    ENOP_SetLow();
    ADRF6703_SetRegister(reg7[0],reg7[1],reg7[2]);
    ADRF6703_SetRegister(reg6[0],reg6[1],reg6[2]);
    ADRF6703_SetRegister(reg5[0],reg5[1],reg5[2]);
    ADRF6703_SetRegister(reg4[0],reg4[1],reg4[2]);
    ADRF6703_SetRegister(reg3[0],reg3[1],reg3[2]);
    ADRF6703_SetRegister(reg2[0],reg2[1],reg2[2]);
    ADRF6703_SetRegister(reg1[0],reg1[1],reg1[2]);
    ADRF6703_SetRegister(reg0[0],reg0[1],reg0[2]);
    ENOP_SetHigh();
    LED2_SetHigh();
}
void TMR1_CallBack(void){
    LATBINV = tx_buffer[tx_buffer_send_index++];
}
void TMR1_CallBack_Empty(void){

}
int main(void)
{
    bool statusTimer1;
    uint32_t dummyNumber=0xFFFFFFFF;
    uint8_t prev_buff_index = 0;

    SYSTEM_Initialize();
```

```
SetIHigh();
SetQHigh();
iq_current_state.q_state = HIGH;
iq_current_state.i_state = HIGH;
SW1_SetInterruptHandler(&SW1_CallBack);

TMR1_SetInterruptHandler(&TMR1_CallBack);
IEC0bits.T1IE = false;

MCCP1_TMR_Initialize();
MCCP1_TMR_Period32BitSet(period);

LOSEL_SetLow();
ENOP_SetLow();

ADRF6703_Init();
uint32_t freq = 2600000000UL;
ADRF6703_SetFrequency(freq);

LED1_SetHigh();
LED2_SetHigh();

while (1)
{
    MCCP1_TMR_Timer32Tasks();

    if( ((statusTimer1 = MCCP1_TMR_Timer32ElapsedThenClear()
       ) && !(isButtonReady)) == true)
    {
        MCCP1_TMR_Stop();
        LED2_Toggle();
        int i= 0;
        for(i=0;i<=1000;i++){
            uint16_t message_index = 0;
            uint16_t sent_bytes = 0;

            IQ_TX(message[message_index++]);
            IQ_TX(message[message_index++]);
            IEC0bits.T1IE = true;
            while(sent_bytes <= MESSAGE_S){

                sent_bytes+= tx_buffer_send_index -
                   prev_buff_index;
                prev_buff_index = tx_buffer_send_index;

                if(tx_buffer_index==(tx_buffer_send_index-1)
                   ){
                    Nop();
```

```
                    }
                    else{
                        IQ_TX(message[message_index++]);
                    }
                }
                IEC0bits.T1IE = false;
                LED1_Toggle();
            }
            LED2_Toggle();
            isButtonReady=1;
        }//

        // Add your application code
    }

    return 1;
}
/**
 End of File
*/
```