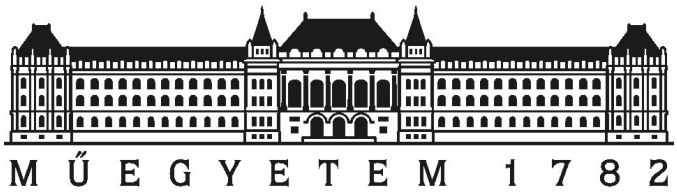


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Szélessávú Hírközlés és Villamosságtan Tanszék

SMOG-1 műhold kézi vevő fejlesztése

Szakdolgozat

Erdélyi Ádám

Konzulens – Dudás Levente

2017

BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
SZÉLESSÁVÚ HÍRKÖZLÉS ÉS VILLAMOSSÁGTAN
TANSZÉK

Szakdolgozat készítés

feladat

Erdélyi Ádám

szigorló villamosmérnök jelölt részére, melynek címe

SMOG-1 műhold kézi vevő fejlesztése

A BME VIK HVT-n készülő SMOG-1 hallgatói műhold fejlesztő csapatához csatlakozva készítsen kézi SMOG-1 műhold vevőkészüléket:

- Ismerkedjen meg a műhold felépítésével, működésével, fókuszálva a műholdfedélzeti kommunikációs rendszerre.
- Isemrje meg a SMOG-1 műhold földi állomásainak felépítését, működését, a műhold vételéhez és vezérléséhez szükséges paramétereket.
- Tervezze meg a SMOG-1 vételére optimalizált, kis méretű, kézi vevő kapcsolási rajzát: LNA, illesztő és szimmetrizáló hálózat, RF vevő IC, integrált mikrovezérlő, soros port - USB illesztő.
- Tervezze meg a SMOG-1 vevő nyomtatott huzalozású lemezének rajzolatát ügyelve az RF jelutak megfelelő minőségére.
- Készítse el a SMOG-1 vevő prototípusát: ültesse be, élessze fel az elkészült áramkört.
- Végezzen minősítő méréseket a vevőkészüléken és dokumentálja azokat: érzékenység, szelektivitás, bithiba és csomaghiba arány különböző sávszélesség és adatsebesség paraméterek mellett.
- Tervezze meg a vevőkészülékhez integrálható Yagi antennát: min. 6 elem, legalább 9 dBi nyereség, lineáris polarizáció.
- Szimulátor program használatával vizsgálja meg az antenna sugárzási tulajdonságait: nyereség, irányélességi szög, előre-hátra viszony.
- Készítse el és mérje meg az antennát.
- Végezzen minősítő méréseket az antenna + vevőkészülék elrendezésen és dokumentálja azokat.

A fejlesztés során ügyeljen arra, hogy kizárólag olyan tervező szoftvereket és fejlesztő eszközöket használjon, amelyek nyílt forráskódúak, vagy bárki által szabadon hozzáférhetőek (nem jogdíjas szoftverek), mint pl. KiCAD, MPLAB-X, Qt, Simplicity Studio, stb.

Infokommunikációs rendszerek specializáció
Nagyfrekvenciás rendszerek és alkalmazások ágazat - Szélessávú Hírközlés és Villamosságtan Tanszék

Záróvizsga tárgyak:

Nagyfrekvenicás rendszerek (BMEVIHVA342 Dr. Nagy Lajos, dr. Seller Rudolf)

A feladat benyújtásának határideje: 2017.12.08

Tanszéki konzulens: Dudás Levente, dr. Gschwindt András, dr. Seller Rudolf

Ipari konzulens: -

A tervezés bírálója:

Budapest, 2017.12.08

dr. Nagy Lajos
egyetemi docens
tanszékvezető

Konzultációk:

Időpont				
év	hó	nap	Észrevételek	Konzultációk

Ipari konzulens véleménye:

Tanszéki konzulens véleménye:

Bíráló véleménye:

HALLGATÓI NYILATKOZAT

Alulírott Erdélyi Ádám, 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, 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, 2017. december 8.

Erdélyi Ádám

Tartalomjegyzék

Kivonat

A Budapesti Műszaki és Gazdaságtudományi Egyetemen egy PocketQube osztályba sorolható kisműhold fejlesztése van folyamatban. A SMOG-1 a projekten több kar, és tanszék bevonásával, számos oktató, hallgató és külsős szakember dolgozik, és céljuk, egy Masat-1-hez hasonló sikereket elérni. A műhold elsődleges küldetése, egy Föld körüli pályán, az elektromágneses szmog mérése a DVB-T sávban. Ezen a 470 $MHz - 860$ MHz-es tartományban a földfelszíni digitális televíziós sugárzás található, és az űrbe kijutó teljesítmény kárba vész. Ez a feladat több kihívást is rejt magában, hiszen ilyen méréseket még senki nem tett közzé ebben a témában, és ilyen kis műhold sem működött még az űrben.

A feladatom, egy SMOG-1 vételére optimalizált, kis méretű, földi vevőt, és a hozzá tartozó antennát elkészíteni. Ez magába foglalja az áramköri és huzalozás szintű tervezést, az ültetést, és a szoftveres megvalósítást. A vevő a rádiófrekvenciás jeleket alakítja digitális adattá, így egy USB csatlakozó segítségével, a számítógép soros portján láthatjuk a műholdról vett csomagoknak a hexa megjelenítését.

Ez az egység népszerűsítési feladatokat is ellát, célja, hogy a műhold vétele közben minél több ember részt vehessen a SMOG projektben, ezzel terve őket az űrtudományok, a Műegyetem és a Villamosmérnöki Kar irányába. Remélhetőleg többen felfigyelnek erre a csapatra, és a későbbiekben is újabb támogatókat vonzhat. Ennek viszont az a feltétele, hogy a vevők és antennák készítése ne igényeljen nagy költségeket, ezáltal nagyobb darabszámban legyen készíthető. Ezért a vevő megalkotása során törekedtem az egyszerűségre, költséghatékonyságra és a későbbi újrafelhasználhatóságra.

Abstract

A PocketQube type satellite development is in progress in the Budapest University of Technology and Economics. The university's more facultys, departments and external experts are working on the SMOG-1 project, and their goal is to turn out as a success, like the Masat-1. The primary mission of the satellite to measure the electroagnetic smog around the Earth, in the frequency band of the DVB-T system. The terrestrial digital video broadcasting is using this 470 $MHz - 860$ MHz band, and the power that reaches the space, is lost. This task has more challenges, like this type of measure has been never done before, and no satellite with this size has been operated int the sapce.

My thesis is to design and create a small receiver that was optimized to receive the signals of the SMOG-1, and create an antenna for this function. This task contains a electronic level and circuit level design, planting, and software developing. The receiver convert the radio frequency signal to digital data, so we can see the packets as a hexa string, with the help of an USB connector and the serial ports of a personal computer.

This receiver's other purpose to popularize the SMOG-1 project. If they can be the part of this historical moment, mybe this will bring them closer to the spacesciences, the BME and the Faculty of Electrical Engineering and Informatics. Hopefully more people will know about this team, and in the future more sponsor can support the projects. This requires to lower the cost of the receiver and the antenna, so it can be produced in higher numbers. Beacuse of this, I have aimed the simplicity, the cost-effiency and the reusability.

1. fejezet

SMOG-1

1.1. Előzmény

A Masat-1 2012. február 13-án állt pályára. A Műegyetem által megvalósított projekt beírta magát a történelemkönyvekbe, hiszen megvalósították az első magyar űrobjektumot, ami a gyakorlatban is működőképes volt. A CubeSat osztályba tartozó műhold elsősorban egy technológiai kísérlet volt, hogy egyetemi körülmények között lehetséges-e egy űrminőségű eszközt létrehozni. A majd három éves, világűrben töltött időtartam bebizonyította, a BME képes a megbízhatóság kérdésében maximálisat alkotni. A Masat-1 2015. január 11-én visszatért a légkörbe, és örökre elhallgatott. Ez viszont nem jelentette a magyar műholdak végét.[1]

1.1. ábra. A Masat-1 logója

1.2. SMOG-1

A következő projekt már a PocketQube osztályba tartozik, és a SMOG-1 nevet kapta. Elsődleges küldetése a világűrbe jutó elektromágneses szmog mérése a DVB-T sávban ($470\ MHz - 860\ MHz$). Ennek eredete a földfelszíni digitális televíziós műsorszórás, az antennakarakterisztikákból adódó feleslegesen kisugárzott teljesítmény. Ezt a Föld körüli keringés során, a spektrum feltérképezésével viszi véghez. A kapott adatokat felhasználva, a jövőbeli antennaspecifikációkat hatásosabb energiafelhasználás irányába lehet terelni. A műhold fejlesztése oktatási irányt képvisel, tehát egyetemi hallgatók, oktatók és kutatók a megalkotói.[2]

A szakdolgozatom keretében, egy kisméretű vevőt – SMOG-1 RX – kell készítenem, ami a projekt későbbi népszerűsítésében is részt vesz. Ehhez mérten egy olyan egyszerű szerkezet megalkotása a cél, amit akár a Masat-1 esetében is használt kézi Yagi antennával is lehet használni. Elvárás még, hogy a megfelelő alkatrészek beszerzésével, akár házilag is elő lehessen állítani, ezzel is buzdítva az érdeklődőket szerte a bolygón, a SMOG-1 követésére. A rádióamatőröknek általában mind képesítésük, mind eszközük van egy műhold jelének vételére, de a lelkes amatőrök nem hagyatkozhatnak ezekre a dolgokra. Nekik szeretnénk segíteni azzal, hogy ezt a vevőt megrendelve, ők maguk is részesei lehessenek a küldetésnek.

1.2. ábra. A SMOG-1 logója

2. fejezet

A SMOG-1 RX felépítése

2.1. Alapkoncepció

A projekt alapötlete egy kifejezetten a SMOG-1 vételére alkalmas vevő építése. Ennek során egy házilag készített, mérőszalagból összeállított, és kézileg egyszerűen működtethető yagi antenna látná el az átalakító szerepét. Ez az éterben meglévő elektromágneses hullámokat a vevő bemenetére juttatja. Ezt a rádiófrekvenciás jelet a modul feldolgozza, és a kinyert információt már digitális formában küldi tovább. Egy USB csatlakozóba bedugva akár számítógép segítségével is folytathatjuk az adatok feldolgozását. Az USB interfész a vevő irányába való kommunikációra is lehetőséget biztosít, így később a felhasználó is képes lehet a programot módosítani, egyéb kiegészítéseket implementálni.

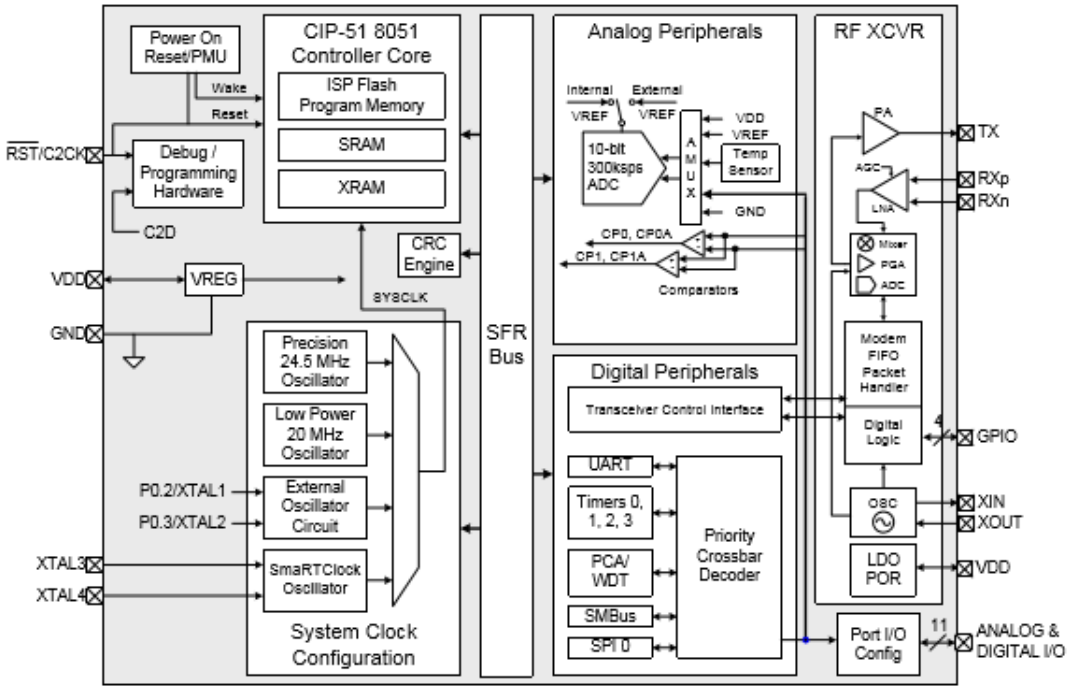
2.1. ábra. A SMOG-1 vevő alapkoncepciója

2.2. Részletes felépítés

Ez előbb ismertetett struktúra ránézésre nagyon egyszerűnek tűnhet, de ha jobban belegondolunk akkor ezeknek a feladatoknak, az egy áramköri elembe való integrálása nem triviális feladat. Megtehetnénk, hogy valamilyen FPGA vagy mikroszámítógép (például Raspberry Pi) segítségével valósítjuk meg a vevőt. Ez viszont jelentősen megnövelné az egység költségeit, és számos fölösleges kapacitást hordozna. Ez nem csak az energiaszükségletet emelné meg, hanem egy fizikailag is nagyobb vevőt jelentene, ami ellentétes lenne azzal a gondolattal, hogy alapvetően egy kézi egységet szeretnénk létrehozni. Az itt megemlítettekből érthető, hogy erre a feladatra a legmegfelelőbb vezérlő, egy mikrokontroller lenne.

2.2.1. SI1062

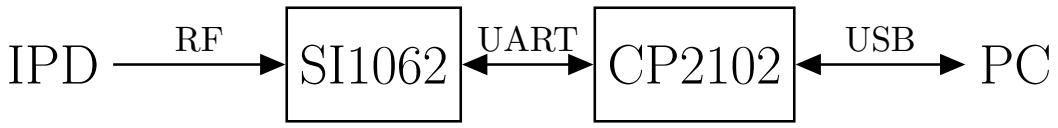
Mivel kommunikációról van szó, szükségszerű, hogy a két fél kompatibilis legyen egymással, így logikus megoldásnak tűnik, hogyha a vevőben, is pont ugyan az a rádiós IC található, mint az adóban. Ez a Silicon Labs C8051F930 [4] mikrovezérlője. Ezzel azonban számos esetben bonyolult lenne vezetéknélküli kommunikációt megvalósítani, így a gyártó ennek az MCU-nak a segítségével felépített egy rádiófrekvenciás mikrokontrollert, ami ultra-alacsony energiafelhasználásával, energiatakarékos üzemmódjaival egy műholdba is racionális választás. A vevőben ez a típus a SI1062 [3] lett. Ez azért más, mint a SMOG-ban levő SI1060-as, mivel más végfok van a két eszközben, így az általunk választott IC jelentősen kisebb energiafelhasználással üzemel. Ezen kívül a két modul megegyezik.



2.2. ábra. A SI1062 blokkdiagramja[3]

2.2.2. CP2102/9

A Silabsos IC azonban csak UART-tal (*Universal Asynchronous Receiver-Transmitter*) képes kommunikálni, nekünk pedig USB-s kapcsolatra lenne szükségünk, ezért egy UART/USB átalakító kell. Logikus lépés, ha a gyártót megtartjuk, hiszen joggal bízhatunk, hogy legalább a saját termékeik között zökkenőmentes kommunikáció tud megvalósulni. Egy másik előny, hogy ugyan azt a fejlesztőkörnyezetet tudjuk használni a fejlesztés során, így nem kell fölöslegesen több, programot telepíteni, és használatukat elsajátítani. Választásunk a CP2102/9 [7] típusszámú átalakítóra esett.
Erre az egységre még azért is szükségünk van, hisz alapvetően a SI IC-t a programozólábakon (C" Interface) kell felprogramozni. Ez azt jelenti, hogy az egyszeri elkészítés után vagy nem módosítunk a programon, vagy pedig a védőborításon rés ejtünk, ami a védelmet nagyban csökkenti, legyen az fólia, lakk vagy egyéb módszer. Viszont egy *bootloader* feltelepítésével akár a soros portokon is frissíthetjük a szoftvert, ezt pedig egy USB csatlakozóval egy számítógépről mindenféle segédprogram nélkül megoldható.



2.3. ábra. A SMOG-1 RX különböző interfészei

2.2.3. Elő-jelfeldolgozás

Az SI1062 IC-ben számos jelfeldolgozási algoritmust is megvalósíthatunk, mégis sokkal effektívebben nyerhetjük ki az információt a hordozóból, ha még az IC előtt végrehajtunk a rádiófrekvenciás jelen pár apróbb dolgot. Erre a kényelmi szempontok mellett az alkatrészek védelme szempontjából is szükségünk van, hiszen különböző szélsőértékek hatására a központi vezérlőegység is károsodhat.

IPD

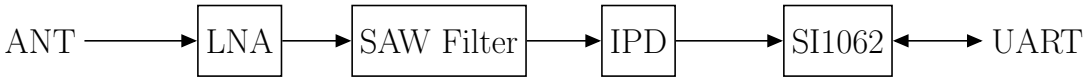
A rádiófrekvenciás alkalmazásoknál az egyik alap tétel az impedancia-illesztés. Az IC belső viszonyait a gyártó nem köti az orrunkra, de mindig ismerteti az adott frekvenciasávokhoz ajánlatos illesztéseket. Ez számos különböző paraméterű LC tag megfelelő elhelyezését jelentené, de szerencsére van egy másik megoldás. Ezeket a passzív elemeket egyes gyártók egy áramköri elemben megvalósítva árulják. Ezek az IPD-k (*Integrated Passive Device*) és ennek a csatoló egységnek a megvalósítását én is egy ilyennel végeztem, az IPD-0433BM41A0019-vel [8].

SAW szűrő

Hogy csak a számunkra releváns jel jusson a bemenetre, rádiófrekvenciás vevőknél illik szűrőt alkalmazni. Ez a jelen esetben egy 435 MHz középfrekvenciájú SAW MA09629 [9] sávszűrő. A SAW (*Surface Acoustic Wave*) szűrő működési eleve, hogy a beérkező elektromos jelet mechanikai hullámmá alakítja egy piezoelektromos kristály. A jel késik miközben keresztülhalad az eszközön, és amikor visszaalakítják elektromos jellé, akkor egy véges impulzus válaszú szűrő végzi el a munkát. A választás azért pont *saw* szűrőre esett, mert sokkal meredekebb karakterisztikát lehet kis helyen elérni, mint akár LC taggal, akár más egyszerűbb megoldással.

LNA

Vezetéknélküli jelek vételénél az antenna utáni első eszköz egy alacsony zajú erősítő (*Low Noise Amplifier*). Ennek a ADL5523 [10] LNA-nak az a feladata, hogy még a feldolgozás előtt emelje a jelszintet (jelen esetben 21dB-lel), és még a többi feldolgozás, szűrés rárakódó zaja előtt, emelje a jelteljesítményt, és az alacsony zajának köszönhetően, ezzel a jel-zaj viszonyt sem csökkenti jelentősen.



2.4. ábra. A SMOG-1 RX RF bemenet

2.2.4. Modulok

Látható, hogy a SMOG-1 RX építése során törekedtünk a modulokból való építkezésre, hiszen így jelentős mennyiségű idő spórolható, hogyha nem nekünk kell ezeket is egyesével az adott feladathoz megtervezni. A nagy-szériaszámú gyártmányoknak köszönhetően akár olcsóbban is kijöhetünk egy-egy elem rendeléséből, és számos esetben a minőség is fölötte van, mint ha mindennapi minőségű diszkrét áramköri alkatrészekből valósítottuk volna meg az egységet. Az alkatrész logikai kialakításának megbízhatóságára a gyártó vevőköre a biztosíték, és az adatlapból számos olyan karakterisztikát is megismerhetünk, amit csak körülményes módon, és hosszadalmas munkával lehetne meghatározni.

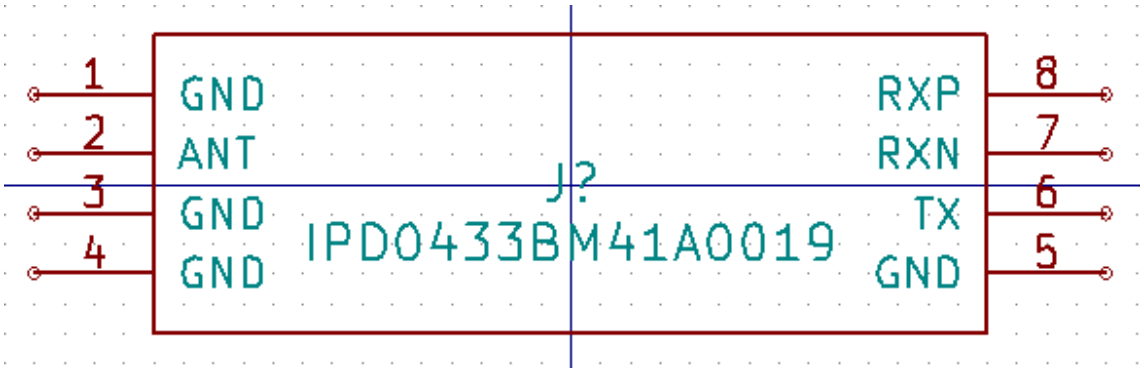
3. fejezet

A kapcsolási rajz

3.1. Fejlesztői környezet

Az áramkör megtervezéséhez a KiCad [11] programot használtam. Egyrészt ingyenesen hozzáférhető, és könnyen használható, másrészt *open-source*, tehát tetszőleges alkatrészekkel bővíthető a saját adatbázisa. Megvan tehát a lehetőség rá, hogy az internetről, a GitHubról[12] előre elkészített áramköri elemeket és *footprint*eket töltsünk le. Ezzel nemcsak időt tudunk spórolni, hanem a nagyszámú és aktív felhasználóbázisnak köszönhetően, nagy valószínűséggel hibamentes kivitelezéshez juthatunk.

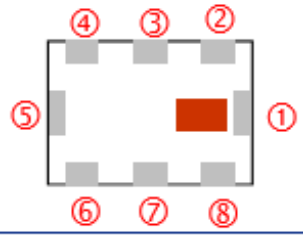
A vevő kacsolási rajzának elkészítése során törekedtem a beépített alkatrészek használatára, hiszen egy kis kreativitással ezek közel bármire használhatóak voltak. Például egyszerű tüskesorokkal meg lehetett oldani az USB csatlakozók modellezését, az IC programozólábainak egy galvanizált furatba történő kivezetését, vagy egy yagi antenna rögzítését is. Az ilyen barkácsmegoldások közben arra kell figyelni, hogy az alkatrésznek megfelelő számú kivezetéssel rendelkezzen, és a helyes alkatrészlábakkal legyen összekötve, hiszen az ebből az állományból létrehozott *netlist*tel építettem fel az ültetési rajzot.



3.1. ábra. Az IPD saját szerkesztésű modellje

Egyes esetekben azonban ez mégsem volt megfelelő megoldás, hiszen a kapcsolási rajznak egyértelműnek kell lennie. Az előző fejezetben említett egységeken kívül a kristályoszcillátort is a beépített alkatrésztervező használatának a segítségével én magam hoztam létre. Az átláthatóság érdekében törekedtem az egyszerű formákra, az alkatrész adatlapjában található lábelhelyezkedésekre, és a lábkiosztásra is. A pontos típusszámot az elem hátára írtam, így aki először látja a kapcsolást is sejtheti, hogy milyen egységről lehet szó.

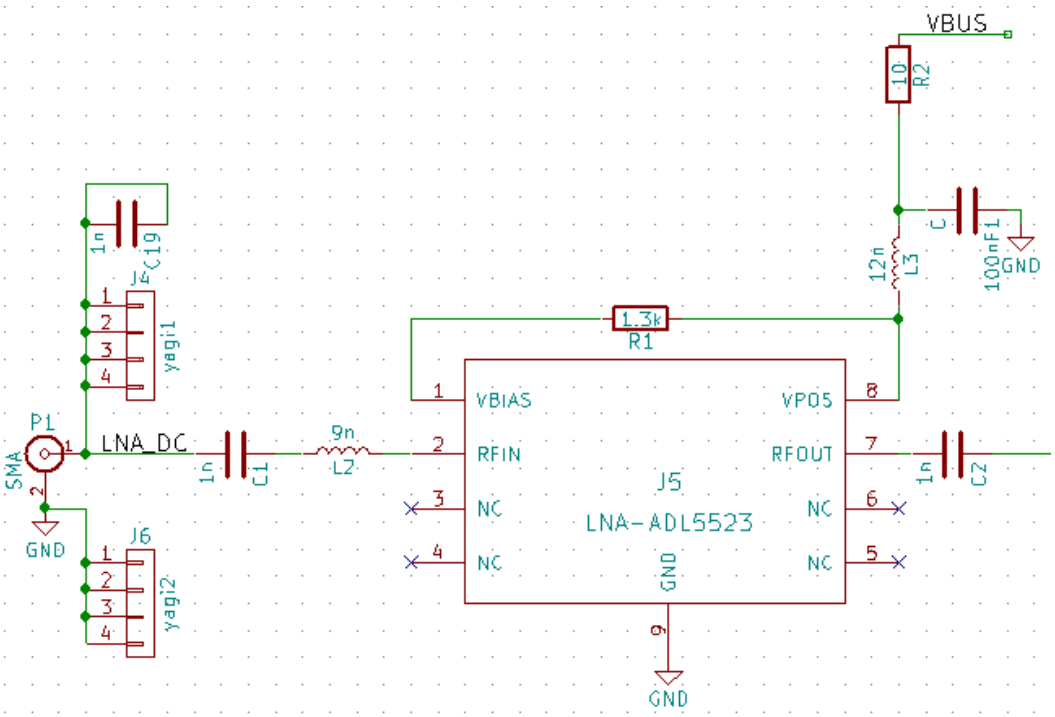
Terminal Configuration			
No.	Function	No.	Function
1	GND	5	GND
2	Ant	6	TX
3	GND	7	RXN
4	GND	8	RXP



3.2. ábra. Az adatlapi lábkiosztás[8]

3.2. Központi elemek és környezetük

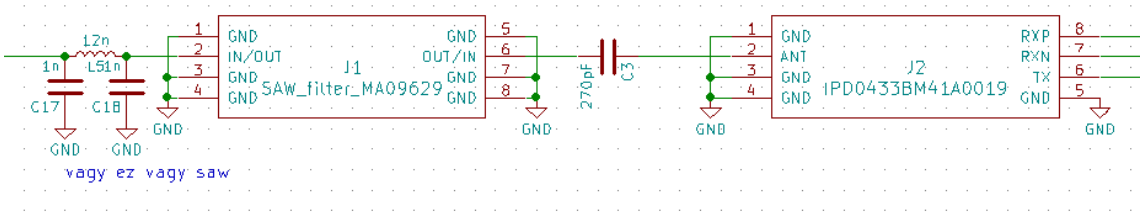
A modulokból történő építkezés velejárója, hogy nem ismerjük a gyártó által megvalósított áramkör pontos felépítését, így nagy mértékben rájuk vagyunk hagyatkoztatva. Ebből kifolyólag az egyetlen kiindulási pontunk az adatlap. Szerencsére itt mindig megtalálható egy példa kapcsolás, a lábkiosztás, és a portok fajtái is, így nem vagyunk teljesen egyedül. Én is számos esetben az adatlapi struktúrát valósítottam meg az ott megadott értékekkel, de több esetben is számos lehetőség közül választhattam, például frekvencia, vagy tápfeszültség függvényében. Viszont az egyedi kialakítású részeknél az alkatrészek értékét én magam választottam. Ilyen eset például a LED előtét-ellenállások, ahol A maximális terhelhetőség felére, 5 mA-ra választottam a vezérlő áram nagyságát.



3.3. ábra. Az SMA csatlakozó és az LNA

Az SMA csatlakozónál egyből láthatunk két csatlakozót meg egy rövidzárral kötött

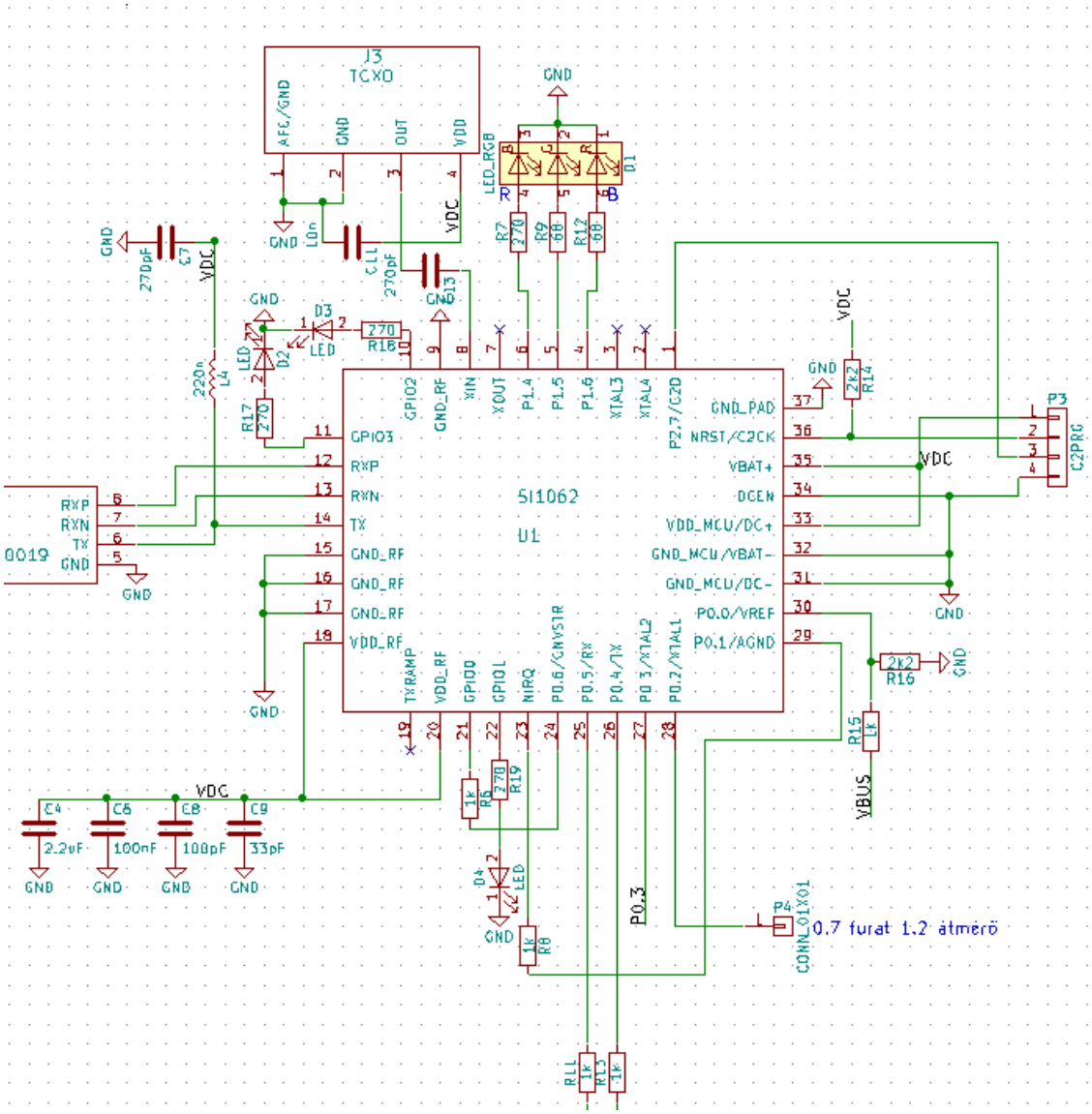
kondenzátort. Az előbbieknek a szerepe, hogy a mérőszalagból készült yagi antennák rögzítését is könnyen reprezentáljuk itt, és később az ültetési rajzolatra átvigyük (4 láb $\rightarrow$ 4 furat $\rightarrow$ 4 rögzítési pont). A rövidrezárt kondenzátor mint mechanikai gát, meggátolja a mérőszalag túlcsúszását, ezáltal nem kívánt galvanikus kapcsolat kialakulását. Azért eset ilyen kondenzátorra a választás, mert ebből az alkatrészből kell a legtöbbet használni, így fajlagosan ebből a legolcsóbb ezt a funkciót ellátni. Itt a vezetéken láthatunk egy *LNA_DC* feliratot, funkciójáról később fogunk szót ejteni. Ez egy úgynevezett címke, ami arra szolgál, hogy az kapcsolási rajz átláthatóbb legyen. Ilyen esetekben a program tudja, hogy az ugyan olyan nevű vezetékek össze vannak kötve, mégis jelentősen könnyíti a fejlesztést, hogy nincsenek zavaró, az egész rajzon átívelő vonalak. Ezek legtöbb esetben táp, vagy valamilyen vezérlést megvalósító vezetékek. Az LNA bemenetén egy LC tagot láthatunk, ami egy sáváteresztő szűrőként funkcionál, az áteresztési tartomány a SMOG-1 kommunikációs frekvenciája. A blokk fölött a tápellátásáért felelős kapcsolás, ami a *VBUS*ról, 5 *V*-ról van ellátva. Az áramkörön két különböző tápfeszültség érték található. Alapból az USB szabvány 5 *V*ja rendelkezésre álló érték, viszont ezt a CP2102 segítségével egy 3.3 *V*-tá alakítjuk, amit használ az SI1062 és a külső oszcillátor is. Az LNA ellátásához is lehetett volna ezt a kisebb tápfeszültség értéket használni, viszont a soros-porti átalakító csak korlátozott teljesítményt képes leadni, és az USB szabvány maximális áramterhelhetősége jóval nagyobb. A kapcsolási rajzon látható adatlapi megvalósítás, viszont a gyakorlatban beültetett konkrét értékek kicsit eltérhetnek tőle, a könnyebb beszerzés, és olcsóbb kivitelezés érdekében.



3.4. ábra. A SAW szűrő és az IPD

Ezután a SAW szűrő következik ami előtt egy LC tag található. Ennek célja, hogy a később mérések folyamán megvizsgálom, hogy megéri-e a szűrő alkalmazása, vagy egy egyszerű Π taggal is ugyan olyan jó sáváteresztést valósítunk meg, csak olcsóbban. A szűrő után az IPD következik, közöttük pedig a kondenzátornak egyenáramlecsatoló szerepe van.

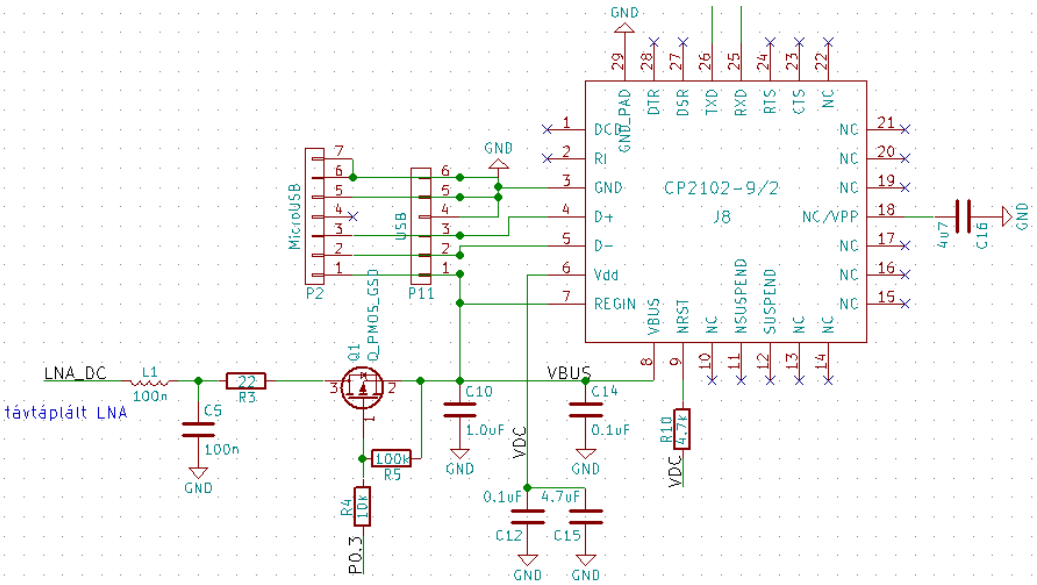
A SI1062-es körül számos diszkrét alkatrészt látunk. Egyrészt van három egyszínű, és egy RGB LED. Ezek szerepe, hogy a később működés során a felhasználó számár értékes információval szolgáljon a vevő működéséről. Például, hogy tápfeszültség alatt van-e, vagy van -e éppen rádiófrekvenciás vétel. Ezeket az eszközöket GPIO lábakra csatlakoztattam, így a legegyszerűbb vezérelni őket. A három piros LED a rádiós egységhez (SI 4460) csatlakozik, az RGB pedig közvetlenül C8051-es mikrokontrollerhez. Másrészt itt található egy külső hőmérséklet kompenzált kristály oszcillátor (TCXO) ami a pontos frekvenciát állítja elő a rádió keverőjéhez. A harmadik fontos részlet a 30-as lábak körül található csatlakozósor, ami a kontroller programozásához elengedhetetlen. Tervben van egy *bootloader* alkalmazása a későbbiek folyamán, de azt a programot is rá kell valahogy tölteni. Ennek a szerepe hogy a soros porton keresztül lehet programozni az eszközt, ami folyamán a későbbi felhasználó dönthet úgy, hogy ő jobb szoftvert tud rá írni. A P0.3-as kimenet (27-es láb) egy vezérlési funkciót lát el az áramkör egy másik pontján, később még visszatérek rá. A soros kommunikáció a P0.4-P0.5 porton keresztül történik, UART segítségével. Látunk még számos hidegítő kondenzátort amit a 3.3V-os *VDC*-re csatol-



3.5. ábra. A SI1062 és környezete

tam, ezek a tápfeszültség stabilitását javítják.

Az USB/UART átalakítónál két csatlakozó sor is látható, mivel az standard USB csatlakozó mellett egy MicroUSB is helyet kap. Ez a leendő felvasználó dolgát könnyítheti meg, hiszen a mai telefonok töltő/adatkapcsolati interfészei micro USB csatlakozóval vannak ellátva, így sokkal nagyobb a valószínűsége, hogy egy ilyen csatlakozóval rendelkező kábel található a háztartásban, mint egy USB hosszabbító. Az alkatrész legtöbb lábát nem kell kötni sehova, de mégis fontos funkciók vannak megvalósítva körülötte. A 25-26-os lábakon kommunikál a SI1062-vel, ezeken a jelutakon áramkorlátozó ellenállásokat lehet látni. A modul az USB szabványnak megfelelően kapott 5 V-os tápot átalakítja a többi egység által használt 3.3 V-tá. Itt térek rá a korábban többször említett LNA_DC és P0.3-as vezérlőlábra. Az ábrán is látható, hogy hogy a P0.3-as lábbal egy FET-et vezérlünk, hogy az LNA_DC-re kijusson az 5 V-os táp vagy ne. Ennek az a célja, hogy ha valaki nem az általam javasolt mérőszalag antennát alkalmazza, hanem valami komolyabb szerkezetet, akkor az SMA csatlakozón keresztül tápfeszültséget tudunk biztosítani a koax másik végén lévő erősítőknek.



3.6. ábra. A CP2102 és környezete

3.3. Kiegészítés

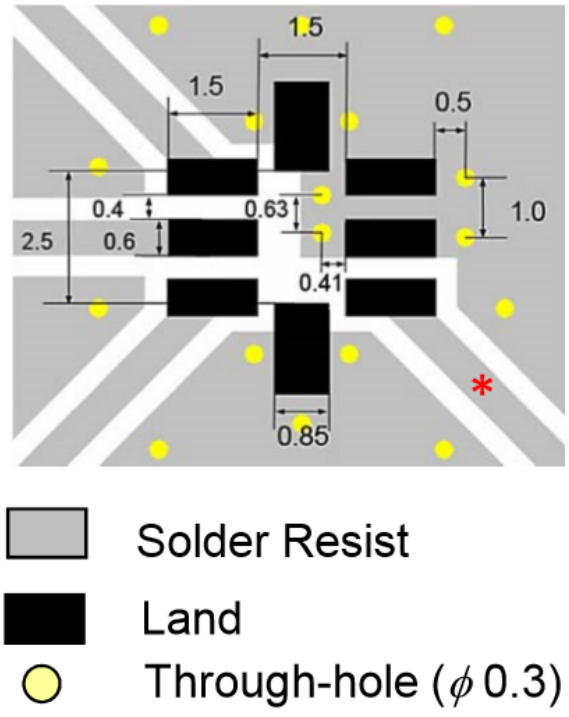
A kapcsolási rajzon (11.5) több komment látható. Ezek egyrészt a tervezés során hasznos kiegészítések, hiszen ha később is ránézek, akkor láthatom, hogy mit és miért csináltam. Másrészt az alkatrészlistát is itt gyűjtöttem össze, hisz így gyorsan ellenőrizhettem, hogy mindenből megfelelő mennyiségű és értékű lesz-e beszerezve.

4. fejezet

Ültetési rajz

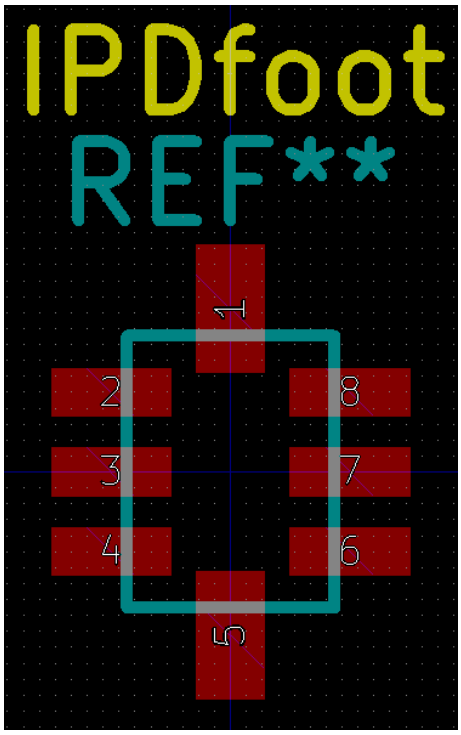
4.1. Rajzolat választás

Ahogyan már a kapcsolási rajzot is, úgy az ültetési rajzot is a KiCad segítségével készítettem el. Ennek első lépése az alkatrészekhez a megfelelő *footprint*-ek választása. A mindennapos daraboknál, hagyatkozhattam a beépített könyvtár adatbázisára, így ezekből az ellenállásokat, kondenzátorokat, és tekercseket meg tudtam oldani. Ezek választásánál elsődleges volt, hogy minél kisebbek legyenek, így a nyák méretéből adódóan lehet spórolni az összköltségen. Másik szempont, hogy még kézi forrasztás segítségével be lehessen ültetni az elemeket. Ezeket átgondolva a 0402-es méretcsaládra esett a választásom amit csak akkor módosítottam egyes esetekben, ha nem sikerült az adott méretben, könnyen beszerezhető olcsó alkatrészt találni. Ebből kifolyólag lett pár 0603-as induktivitás és kondenzátor is. Az alkatrészek beszerzéséhez alapértelmezetten a Farnell[13] és a Lomex[14] webáruházát használtam. A fő szempont ismét a gazdaságosság volt, de mivel nagyfrekvenciás alkalmazásról beszélünk, pár silány termék kiesett a választásból.



4.1. ábra. Az adatlapi rajzolat[8]

Egyes esetekben azonban a *footprint* nem állt a rendelkezésemre, így saját magamnak kellett megcsinálnom. Természetesen nem miliméterpapír és tolómérő segítségével oldottam meg a méretek levételét, hanem a gyári adatlapokban leírtak segítségével szerkesztettem meg a megfelelő lenyomatot. Figyelni kellett a pontos lábkiosztásra is, hiszen a program a kapcsolási rajzból legenerált *netlist*-ből szerzett információi alapján rendelte össze az összekötni kívánt alkatrészkivezetéseket.

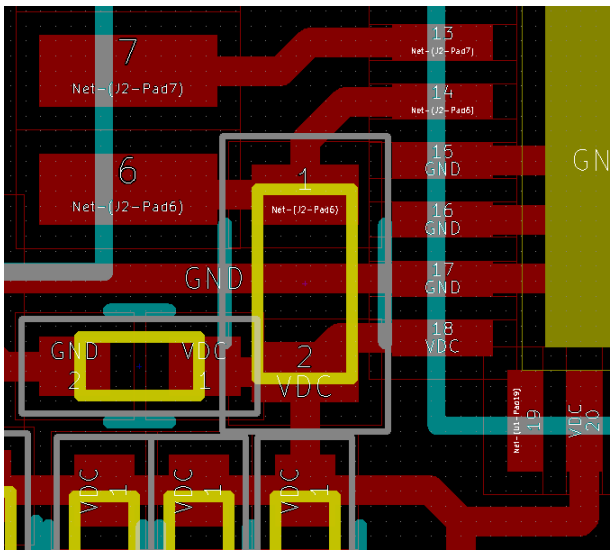


4.2. ábra. Az elkészített rajzolat

4.2. Alkatrészek elhelyezése

A kapcsolási rajzon látható, hogy ezt a hálózatot egy rétegen, SMD technológiával nem lehetne megvalósítani, ezért két rétegre volt szükségem, egy *top* és egy *bottom* oldalra. Alapvetően a *top* oldalt használom ültetési oldalnak, de a mérőszalag antennák rögzítésére szolgáló drótokat, a MicroUSB csatlakozót, és a túlcsúszás gátló kondenzátort a vevő aljára raktam. Az összeköttetések vezetését is alapvetően az ültetési oldalon valósítottam meg, de számos esetben szükség volt viák segítségével a túloldalra átvezetni azokat, hogy a megfelelő helyen ismét a *top* oldalra visszatérve, a megfelelő helyre csatlakozzanak. Az alkatrészek lehelyezésénél az elsődleges szempont a rádiófrekvenciás-jelút törésmentessége és zavartalansága volt, a többi elem elhelyezése pedig a minimális helyfoglalás.

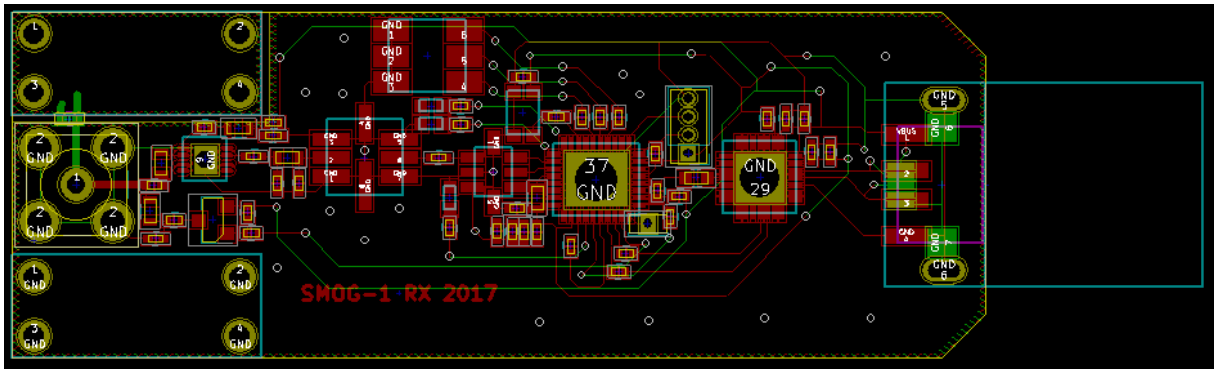
A vezető sávok szerkesztésénél kerültem a 45°-os irányváltásokat, helyettük tompaszögű megoldást választottam. Egy esetben (4.3 ábra) egy 0603-as alkatrész alatt vittem át egy sávot, ami példázza, hogy különböző méretű alkatrészek milyen jól ki tudják egészíteni egymást. A 4.4 ábrán láthatjuk, hogy a vezetékezés jelentős része a *top* oldalon fut, a *bot* oldali vezető sávokkal csak az elsődleges oldali kereszteződéseket kerültem el.



4.3. ábra. Alkatrész alatti átvezetés

4.3. Nagyfrekvenciás optimalizálás

Az tervezés befejeztével mind a két oldalon földkitöltést alkalmaztam (11.3 és 11.4 ábrák), és a két oldali földek között viákkal javítottam a nagyfrekvenciás viselkedést. A kisebb csillapítás érdekében az SMA csatlakozótól az LNA-ig futó meleg-ér szélességét növeltem. Ebben a frekvencia tartományban tápvonalelmélettel nem kellett foglalkoznom.



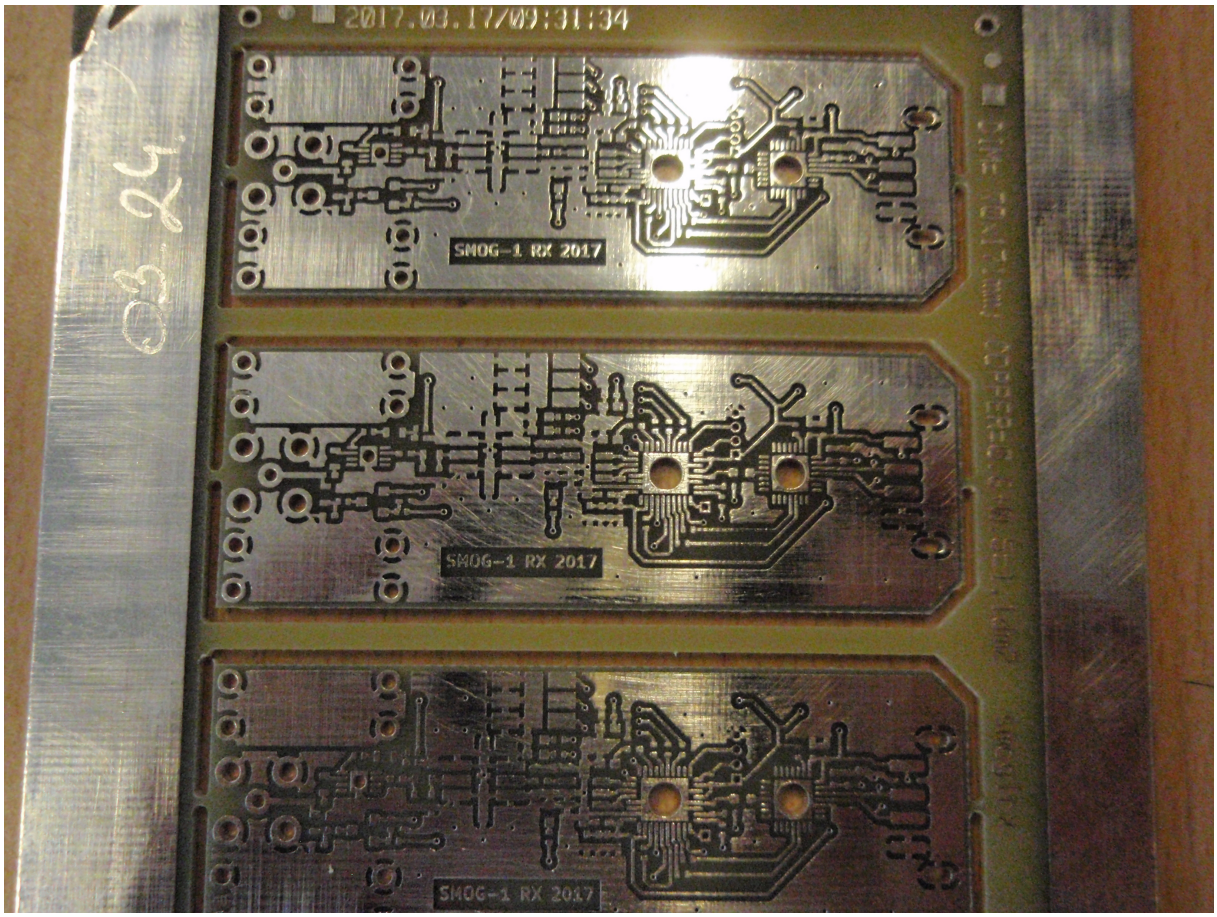
4.4. ábra. A kész ültetési rajz

5. fejezet

Építés és élesztés

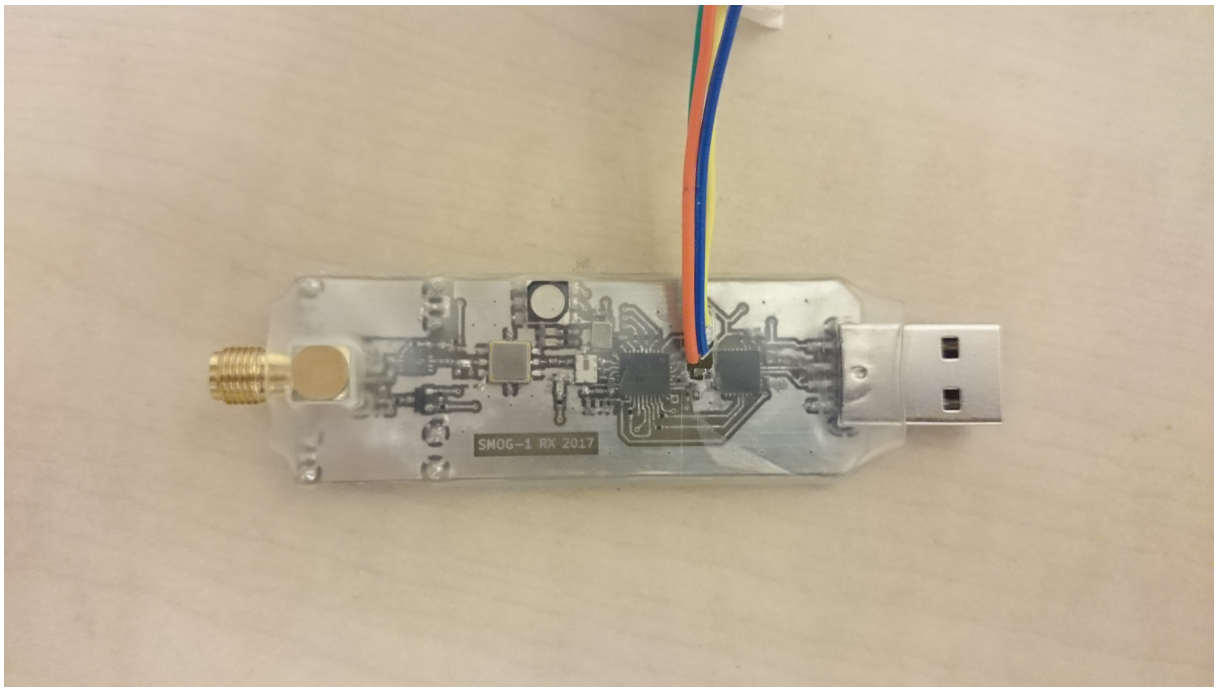
5.1. Alkatrészek beültetése

Az áramkör összeszerelését az V1504-es laborban végeztem. A 0402 alkatrészméretek következtében a forrasztópáka mellett szükségem volt egy mikroszkópra, csipeszre és $0,25\ mm$ átmérőjű forraszhuzalra is. Az ültetés során megtanultam SMD-t forrasztani, és a folyasztószer szükségességét is saját tapasztalataim bizonyítják. Az ültetés során figyelnem kellett az integrált alkatrészek helyes állására, amit az egyes láb jelölésének köszönhetően tudtam leellenőrizni. A beültetett LED-ek polaritását az adatlapban feltüntetett ábrák segítségével állapítottam meg.



5.1. ábra. Az elkészült NYHL top oldala

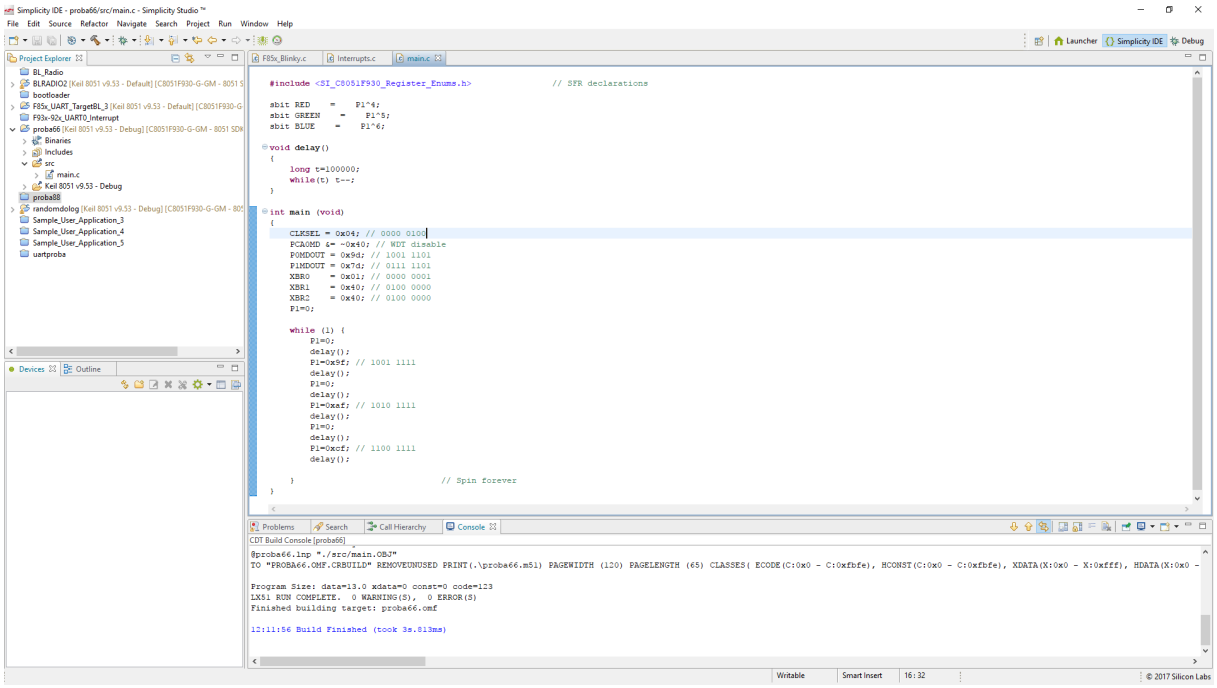
A munka során ért egy meglepetés, ugyanis a kristályoszcillátor forrasztási felületei nem egyeztek meg a honlapon található , így azt nem sikerült tökéletesen beforrasztani. Később ki is derült, hogy a forrasztás hibás, nincs tökéletes elektronikus kapcsolat, így az ki lett cserélve egy másik, 30 MHz-es kristályra. Azért erre esett a választás, mert ez volt megtalálgató a laborban, és a SMOG-on is ilyen található. Az áramkör elkészülte után, USB csatlakozóba bedugva, az RGB LED halvány világítása jelezte, hogy főbb rendszerekben nem esett kár, és a tápfeszültséget is megfelelő lábakról vettem le.
A beültetés elkészültével egy átlátszó zsugorcsőbe raktuk, aminek hő hatására csökken az átmérője és mintegy mechanikai védelmet nyújt az áramkörnek. Ez csak egy próba volt, hogy a megfelelő zsugorodási tényezőjű cső lett-e befejezve. A programozólábaknál kivágva, a furatokba vezetékek forrasztásával készen áll az SI IC a felprogramozásra.



5.2. ábra. Az összeszerelt vevő

5.2. Áramkör élesztése

Az áramkör alapfunkcióinak tesztelését egy egyszerű LED-villogtatós, a szakmában csak *blinky*-nek nevezett kód rátöltésével végeztem el. Ehhez az adatlapban található leírást, és a Silabs honlapról[6] letölthető Simplicity Studió 4-et használtam. Magát a programozást egy USB Debug Adapter végzi. A kód (5.3) elején *include*-oltam a kontroller specifikus fejlécet, amibe a vezérléshez szükséges beépített változók találhatóak. A *main* függvényben a kezdeti inicializálásokkal az órát egy induló értékre állítjuk, és kikapcsoljuk a *watchdog*-ot a megfelelő működés érdekében. A *PnMDOUT*-tal beállíthatjuk a használni kívánt kimeneteket, az *XBRn*-nel pedig annak típusait adhatjuk meg. Ezeknek a változóknak a szükséges értékeit az adatlapban bitenkénti bontásban találhatjuk meg. A kódtörzs maga egy végtelen ciklus, amiben az RGB LED fényeit egyenként felvillantjuk és kioltjuk. A kód célja annak ellenőrzése, hogy a tervezés és az ültetés során katasztrofális hibát nem követtem el, az IC megkapja a szükséges tápfeszültséget, órajelet és vezérlést.



5.3. ábra. A Simplicity Studio fejlesztő környezete

5.3. Beágyazott szoftverfejlesztés

A feladat innentől a beágyazott szoftverfejlesztés irányába megy át, hiszen az áramkör már kész, csak életre kell bírni. Itt az egyik legfontosabb feladat a regiszterek helyes beállítása, hisz akár egy nem/rosszul beállított érték, az egyébként logikailag helyes programot teheti működésképtelenné. A 5.4 ábrán látható XBR2-es regiszter XBARE bitjének 0-ban hagyása a kommunikációért felelős portokat is ellehetetleníti. Ilyenek miatt nagyon fontos a dokumentációk alapos tanulmányozása (például a [4]) és az ott lévő ábrák, táblázatok tanulmányozása. Ezekből a táblázatokból a regiszter funkcióján kívül, az egyes bitek olvashatóságáról, és a *Reset* állapot után felvett értékükről is tájékoztatást kaphatunk. A fejlesztőkörnyezeten belül rengeteg mintakód is található ami nagyban megkönnyíti az adott funkció megismerését és használatát.

SFR Definition 21.3. XBR2: Port I/O Crossbar Register 2

Bit	7	6	5	4	3	2	1	0
Name	WEAKPUD	XBARE						
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Page = 0x0; SFR Address = 0xE3

Bit	Name	Function
7	WEAKPUD	**Port I/O Weak Pullup Disable** 0: Weak Pullups enabled (except for Port I/O pins configured for analog mode).
6	XBARE	**Crossbar Enable** 0: Crossbar disabled. 1: Crossbar enabled.
5:0	Unused	**Unused.** Read = 000000b; Write = Don't Care.
Note: The Crossbar must be enabled (XBARE = 1) to use any Port pin as a digital output.		

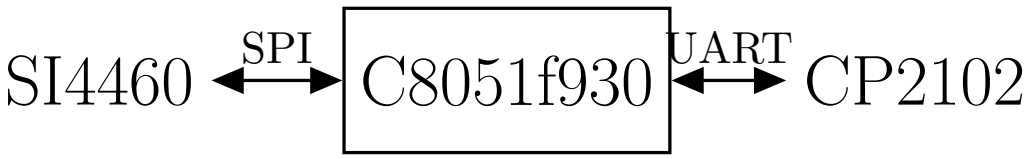
5.4. ábra. Egy az adatlapokban található táblázat[4]

6. fejezet

Vezérlési lánc

6.1. Kommunikációs interfészek

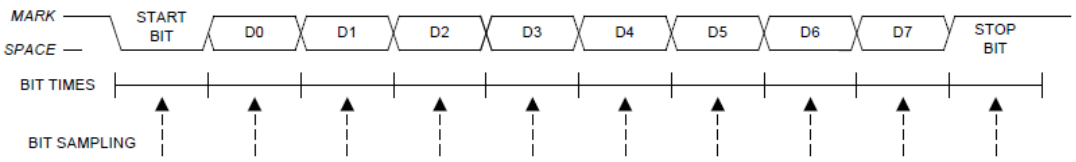
Az SI1062-ben lévő C8051F930-as mikrovezérlő több különböző kommunikációs protokollra is fel van készítve, többek között SMBus (*System Management Bus*) vagy I^2C. Számomra azonban csak a számítógéppel való kommunikációhoz egy UART, és a rádiós egységgel való kommunikációhoz egy SPI (*Serial Pheripheral Interface*) interfészre van szükségem. Ezekhez a protokollokhoz egy hardveres támogatás is jár, Tehát regiszter szinten megvannak a szükséges változók, és a megszakítás kezelőben is saját vektorok vannak nekik fenntartva. Viszont innentől kezdve a fejlesztőnek, vagyis nekem kell implementálnom, hogy mit akarok kezdeni az így kinyerhető információval.



6.1. ábra. A C8051f930 különböző kommunikációs interfészei

6.2. UART

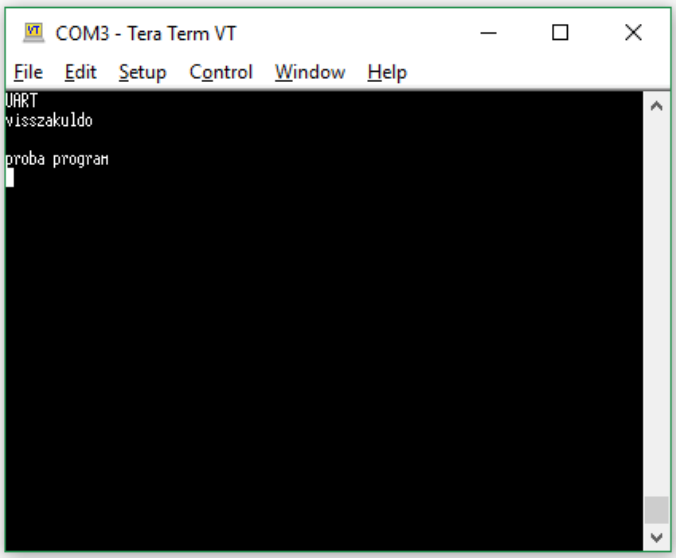
A C8051F930-ban megvalósított UART egy asszinkron, full duplex kommunikációs interfész. Ehhez mindössze két vezetéket vesz igénybe, egy TX (*Transciever*) és egy RX (*Receiver*) lábat. Viszont a vezérlő ezt elrejti előlünk, és az SBUF0 regiszterrel érjük el mindkét *buffer*t. Ha olvassuk ezt a regisztert, akkor a vételi tárolóból kapjuk meg az adatot, ha írunk bele, akkor pedig ez lesz a következő bitsorozat amit kiküld. A 11.6 ábrán az UART0 blokkdiagramja látható, hogy míg a megvalósítása eléggé összetett, addig a fejlesztőnek, lényegében csak az ott látható SCON és SBUF regiszterrel kell foglalkoznia. Az SCON regiszterben lehet beállítani, hogy 8 vagy 9 bites módot szeretnénk megvalósítani, és itt találhatóak az *interrupt flag*ek is. Az általam beállított értékek alapján az protokoll *Baud Rate*je 115200 bps, és egy 8N1-es adatszerkezetet valósít meg. Ez azt jelenti, hogy egy üzenet során 8 adatbitet küld, nem használ paritás bitet, és csak egy stop bitet alkalmaz.



6.2. ábra. Az UART0 idődiagramja[4]

Az UART szoftveres kezelése még azért is egy egyszerűbb feladat, hiszen a két kommunikáló fél egyenrangú, köztük semmilyen *slave-master* viszony nem áll fent az én esetemben. A kezelés mindkét esetben a megszakítás kezelő segítségével oldottam meg, hiszen így tudtam a leggyorsabban reagálni az eseményekre, nehogy lemaradjak és felülíródjanak a kapott üzenetek. Alapvetően két-két buffert használtam vételre és adásra. Ebből az egyiket a közvetlen UART *interrupt*on keresztül karakterenként töltődött vagy fogyott. A másikat pedig a főprogram és a megszakítás közötti kommunikációra, a vett parancsok és a küldeni kívánt adatok, átadására. Azért volt szükség ennek a két szintnek az elválasztására, mert így az éppen *interrupt* által végzett folyamatok, nem zavarják a főprogram futását, és csak akkor módosul a főprogram által használt tartalom, amikor abban már értelmes adat van.
A vételi oldalon (ÁBRA) sorzárásig, vagy a maximális hosszig gyűjtöm az érkezett karaktereket, majd ezt a másik tárolóba töltve, átadom a főprogramnak, és bebillentek egy változót, hogy szóljak, hogy üzenet jött.
Adásnál,(ÁBRA) a főprogram egy *string*et biztosít az *interrupt* számára, ami a karakterenkénti kiküldést már egyedül, a főprogramtól függetlenül végzi.

Ennek a funkciónak a leteszteléséré, egy egyszerű kis programot írtam, ami visszaküldi UARTon a kapott karaktereket.



6.3. ábra. Az UART0 interfész tesztje

6.3. SPI

A C80551F930 és az SI4460 között a kommunikációt egy SPI interfész oldja meg. Ez egy full-duplex szinkron soros busz. A szinkronitás ebben az esetben azt jelenti, hogy a vétel és az adás egyszerre történik, így elengedhetetlen, hogy a két eszköz között egy *master-slave* kapcsolat álljon fent. az adott felállásban egyértelmű, hogy a mikrokontroller lesz a *master*, mivel ezt tudjuk programozni, és a rádiós modul lesz a *slave*. A C8051F930 és az SI4460 két külön egység, viszont a Silabsa fejlesztők munkáját megkönnyítendő, egy tokban integrálták SI1062 néven, és számos egyéb logikus döntést hoztak. A mikrokontroller két SPI interfésze közül az egyiket, az IC-n belül összekötötték a rádiós egységgel és ez már csak dedikáltan erre használható. Viszont mi továbbra is a C8051-est programozzuk, így erre figyelni kell, hogy a regiszterekben a lábkiosztás nekünk kell beállítani.

MCU GPIO	Radio Control Interface
P0.7	SDN
P1.0/SCK	SCLK
P1.1/MISO	SDO
P1.2/MOSI	SDI
P1.3/NSS	nSEL

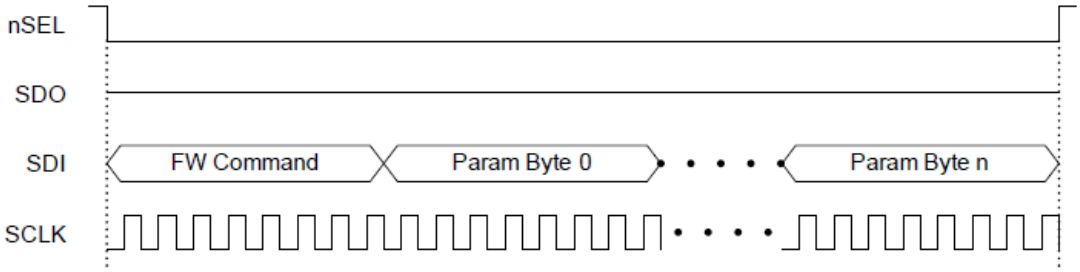
6.4. ábra. Az SI4460 és a C8051 lábösszerendelése[3]

Az itt látható lábakon kívül a rádiós IC rendelkezik még egy NIRQ lábbal, amivel hardveres megszakítást tud kérni a mikrokontrollertől, és négy GPIO lábbal, ami különböző állapotok jelzésére használható. Az SDN láb az SI4460 tápellátásáért felel, az SCK lábon pedig a kommunikáció során használt órajelet biztosítja a kontroller. A MISO és MOSI lábak a *Master/Slave* és *In/Out* rövidítéseiből állítható össze, ezeken keresztül folyik a tényleges adatfolyam. Az NSS egy *Slave Select*, amiben az N az alacsony aktív logikát jelöli. Ezt mindig szükséges alacsony logikai szintre húznunk, ha a rádiós egységgel szeretnénk kommunikálni.

A 11.7 ábrán is látható, hogy A Silabs itt is megvalósította az SPI kommunikációhoz szükséges alapokat, viszont ebben az esetben sokkal több regiszter értéket kell figyelnünk, és a kommunikáció is egy sokkal kötöttebb módon kell, hogy történjen. Az SPIDAT egy bufferként szolgál hasonlóan az UART esetéhez. Az SPICKR módosításával az SCK órajel frekvenciáját befolyásolhatjuk. A konfigurációs beállításokat a SPICFG és az SPICN regiszterben állíthatjuk, míg a számunkra hasznos *interrupt* bitek az SPICN bájtban találhatóak. Eredetileg az SPI kezelést is megszakítás segítségével oldottam meg, viszont később egyéb okokból ezt elvetettem, és küldés után egy arra a bitre vártam, ami azt jelzi, hogy az adatküldés megtörtént.

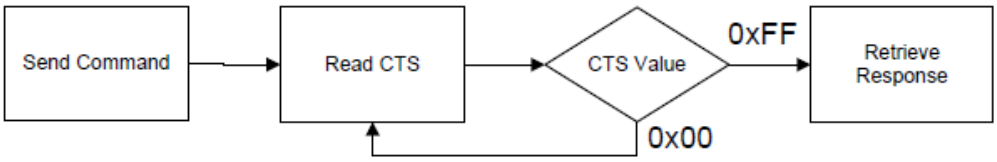
A kommunikáció fajtája egy pár esetre szűkíthető le. Egyrészt konfigurációs beállítások leküldése a rádiós egység számára. Erre nem várunk vissza reakciót, úgy vesszük, hogy mi sem rontottuk el, és a hardver is jól teljesítette a feladatot. Itt csak arra kell figyelnünk, hogy mielőtt küldjük az utasítást, győződjünk meg arról, hogy az előzőt már végrehajtotta, és készen áll az új fogadására.

Másik csoport a kérések leküldése amire választ várunk. Itt a leküldés után a *Slave Select*et elengedve, majd újra lehúzva egy úgynevezett CTS (*Clear To Send*) értéket mo-



6.5. ábra. Utasítás leküldése a rádió számára[5]

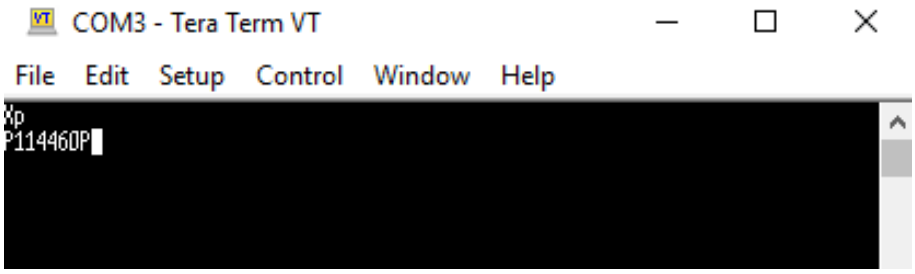
nitorozva várjuk, hogy az SI4460-as úgy jelezzen, hogy feldolgozta az utasítást, válaszra kész. Ezt egy $0xFF$ válasszal jelzi, és ilyenkor az ez után kapott bájtok a válasz értékek. Mivel, hogy ha olvasni akarunk, akkor küldeni is kell valamit, ilyenkor érdemes egyszerű hexa nullát küldeni, azzal elkerülhetőek a nem kívánt hatások.



6.6. ábra. A CTS monitorozása[5]

A harmadik eset, amikor a rádió jelez, hogy történt valami, és kezeljük már ezt az esetet. Ez lehet egy csomag érkezése, egy detektált *preamble* vagy akár csak egy hibás utasítás is. Ilyenkor az NIRQ lábon jelez a vezérlő fele, hogy esemény történt. A kontrollernél célszerű ezt a lábat egy saját alacsony aktív megszakítással társítani, így azonnal tudunk reagálni az esetre. Ilyenkor is szükséges nekünk kezdeményezni az SPI kommunikációt, és kideríteni, hogy pontosan mit okozta a megszakítást. Az *interrupt* kérés törlése után a lépések megegyeznek a második típusú kommunikációval.

Az SPI interfész tesztjét egy egyszerű *PartInfo* paranccsal végeztem el, ami a leküldés után visszaküldi a rádió típusszámát. A 6.7 ábrán látható, hogy a 4460-as érték jött vissza, tehát a kommunikáció megvalósult.



6.7. ábra. Az SPI1 interfész tesztje

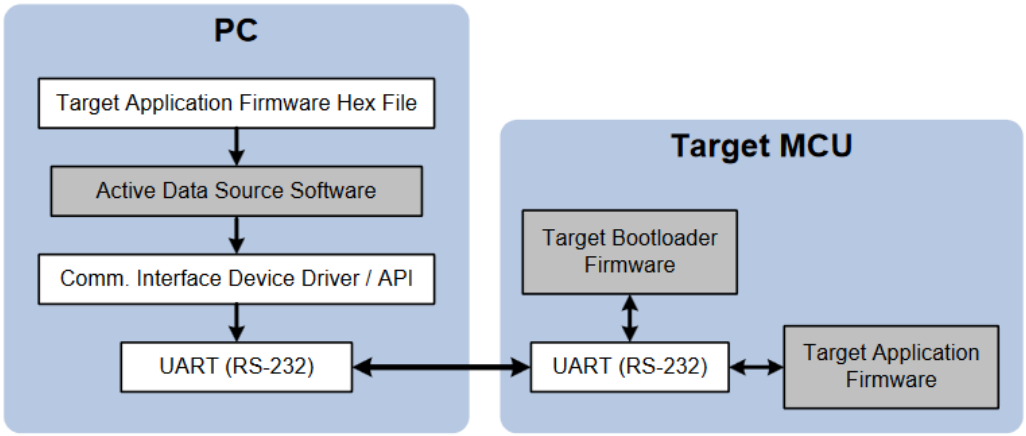
7. fejezet

Bootloader

7.1. Szoftver frissítés

Ahogy már korábban említettem, a szoftverben szerettük volna nyitva hagyni azt a lehetőséget, hogy a későbbiekben módosíthassuk a vezérlőben lévő programot. Ez egyrészt, lehet a működést segítő/optimalizáló változtatás, vagy akár csak egy *Easter-egg*, hogy a SMOG-1 egy speciális üzenetére egy adott sorozatú LED villogtatást végez. Ezen kívül a vállalkozóbb szellemű embereknek is szeretnénk meghagyni az esélyt, hogy akár ők maguk készíthessenek egy kódot, amivel venni tudják a műholdat. Egy *bootloader* implementálásához a Silicon Labs is nyújt egy kis segítséget, de majd később látható, hogy ez így is egy rengeteg buktatókkal rendelkező feladat.

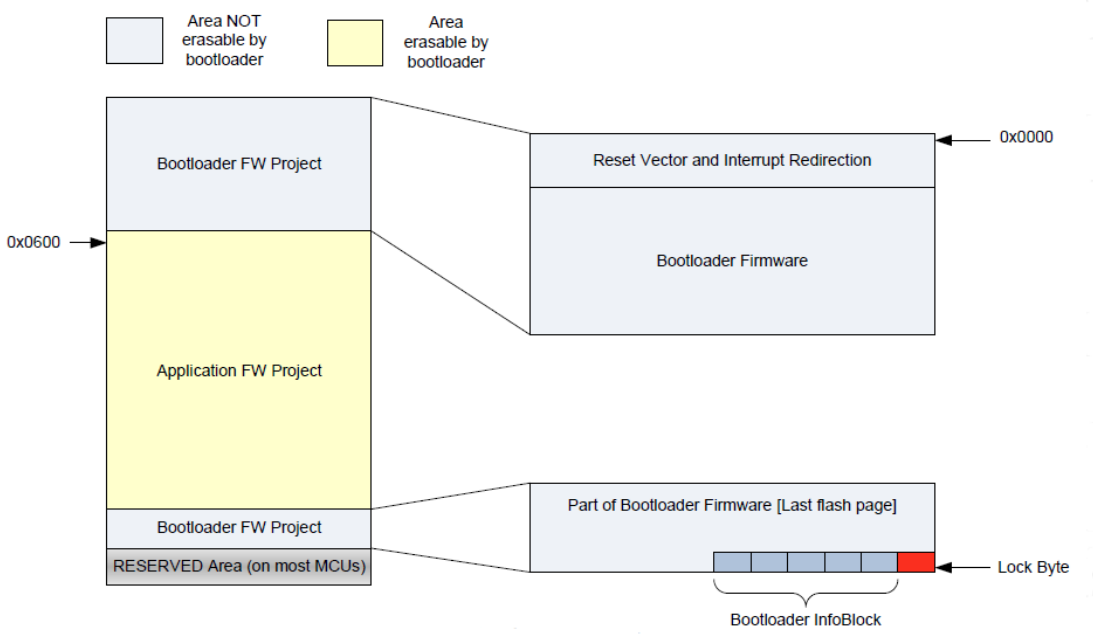
Egy mikrokontroller szoftverének a módosítására általában csak a programozói interfészen van lehetőség, ehhez pedig gyakran speciális programokra és programozó egységekre van szükség. Az nem várható el a lelkes amatőröktől, hogy rendelkezzenek ilyen eszközökkel, így egy másik lehetőség után kell nézni. A legtöbb kontrollerben meg van az a lehetőség, hogy direktbe memóriacímekre írjunk, így akár maga a rajta lévő programkód is módosítható. A vevő már amúgy is szerepel egy soros-porti csatlakozás, így ha meg tudnánk oldani, akkor ezen keresztül le tudjuk küldeni, hogy a program memóriában mit változtasson meg a vezérlő. Ehhez a módszerhez csak egy számítógépre és egy USB csatlakozóra van szükség, ezek többnyire pedig minden háztartásban megtalálhatóak.



7.1. ábra. A bootloader felépítése[16]

7.2. A bootloader felépítése

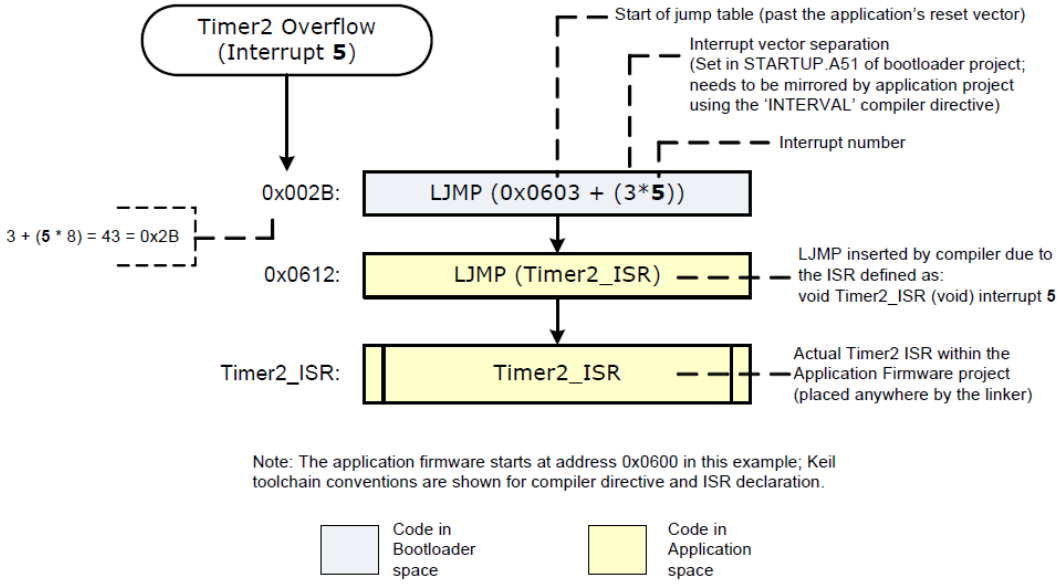
A bootloader egy megvalósítását láthatjuk a 7.1 ábrán. A három szürkével kiemelt rész a funkció elengedhetetlen részei. Egyrészt szükség van egy forrásra, ami biztosítja az adatokat, az adott memóriacímekhez tartozó bájtokat a mikrokontroller számára. Ezt a Silabs megvalósításában az *Active Data Source Software* valósítja meg, ehhez egy külön kis programot is készítettek, ami csak Windows operációs rendszer alatt működik. A vezérlő memóriájában két, egymástól független kód kap helyet. Az egyik magam a *bootloader* forráskódja, ami ezzel a *Data Source*-szal felveszi a kapcsolatot és és biztosítsa, hogy a hasznos kód jó helyre kerül. Ide érdemes egy védelmet alkalmazni, hogy nehogy véletlenül a *bootloader* felülírja magát, hisz futás közben omolhat össze az egész rendszer. Az egész legfontosabb eleme, pedig maga a program, amit ilyen módon akarunk frissíteni. Itt a frissítés csak képletes, hisz valójában az egész felhasználói programot újra kell fordítani, és a kontrollerre tölteni.



7.2. ábra. A mikrokontroller memória felosztása[15]

7.3. Memória elosztás

A C8051F930 64 kB kódmemóriával rendelkezik, ami azt jelenti, hogy a címtartomány 0x0000H-tól 0xFFFFH-ig terjed. Egy *Flash Page* 1kB, ami azért fontos, mert a *bootloader*, egyszerre egy oldalt tud kiírni. A *bootloader* programja a memóriában két részre van osztva. Egyrészt a memória elején található, hogy a tápfeszültség alá helyezett, és *boot*oló rendszer itt induljon el. Itt a 0x0000H-tól a 0x0400H-ig tartó területet foglalja el az én esetemben. Másrészt az utolsó használható oldalt használja fel. Itt található a *Lock Byte* ami a memóriaterület végét jelzi, és emiatt ezt a területet a mikrokontroller szoftveresen nem írhatja felül. Ez a 0xF800H-tól a 0xFBFFH-ig terjed. A ténylegesen legutolsó oldal, nem használható terület. E között a területek között helyezkedik el a felhasználói program.



7.3. ábra. A megszakítás átirányítás[15]

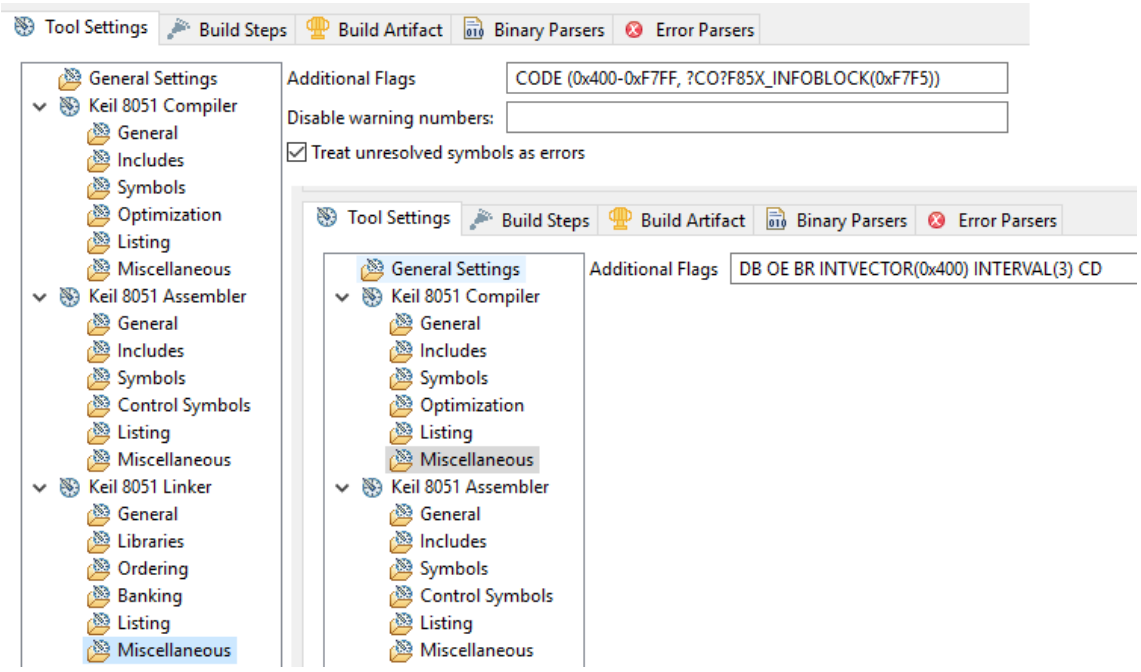
7.4. Megszakítás kezelés

A kódmemória területén az *interrupt* vektorok a 0x0000 címtől kezdődően helyezkednek el, a gyors elérés megkönnyítése érdekében. Ezeken a memóriaterületeken csak egy átirányítás található oda, ahova be lett fordítva a megírt megszakítás kezelés. Viszont a memória eleje a *bootloader* által kezelt terület így ezt nem tudjuk módosítani, pedig ha változik a megszakítás kezelésünk és az eredeti helyére nem fér el, akkor a fordító egy másik memória területre helyezi el. Ezért a mi helyzetünkben itt csak egy átirányításnak kell lennie a felhasználói területre, ahol tároljuk ezeket a vektorokat, amik megmondják, hogy az adott *interrupt*ot hogyan kezeljük.

7.5. Átportolás

A fentiekből az látható, hogy ez elméletileg egy elég jól definiált, és nem túl összetett feladat, pláne, ha a Silicon Labs még segítséget is nyújtott számunkra. Valójában A gyártó csak pár C8051 változatra készítette ezt el, az F930-asra nem. Így a verziószámban legközelebbi, F850-es verzióról kezdtem el átportolni a saját vezérlőmre. Ezt nehezítette, hogy azt a típust is csak részben ismertem, amivel én dolgoztam, a korábbit egyáltalán nem. Pár lényeges különbségre elég gyorsan rá lehetett jönni (például a 32kB-os kódmemória és a 512 bájtos *Flash Page*), viszont több dologra csak a nemműködés következtében derült fény.

Fordítás közben már szükséges tudni a kódok pontos elhelyezkedését, így a fejlesztő környezet számára is egyértelművé kellett tennünk, hogy milyen részeket hova szeretnénk befordítani. Ezeket a 7.4 ábrán látható módon tehetjük meg. A megszakítás átirányítás azonban még tartogatott meglepetéseket. Ugyanis a korábbi IC-ben csak kevesebb *interrupt* volt található, így a bootloaderben lévő JUMP tasítások csak ezekhez voltak elkészítve. Okozott pár érdekes pillanatot amikor a felhasználói programban lévő SPI megszakítás hívásakor a program hirtelen a *bootloader* módban kötött ki, egy fagyott meg. Azt hogy aktuálisan melyik kódrészben vagyok, a LED-ekkel jelzem. A *bootloa-*



7.4. ábra. A fordítási címtartomány kiválasztása

*der*es esetben a gyártó által biztosított *debugger* már nem volt használható, hiszen már nem pont az a kód volt a memóriában ami fordítva lett. Ilyenkor már csak a LED-ekre számíthattam a futás közben állapotok jelzésére, és különböző fordítási melléktermékeket bújhattam, hogy hova éppen milyen kód lett a fordító által befordítva, és mi okozhatja a hibás működést.

```
            0075H     001EH                *** GAP ***
CODE        0093H     0002H     ABSOLUTE
            0095H     006BH                *** GAP ***
CODE        0100H     00F9H     UNIT       ?PR?MAIN?FXXX_TARGETBL_MAIN
CODE        01F9H     0085H     UNIT       ?C_C51STARTUP
CODE        027EH     0041H     UNIT       ?PR?_TGT_WRITE_FLASH?FXXX_TARGETBL_COMMANDS
CODE        02BFH     003BH     UNIT       ?C?LIB_CODE
CODE        02FAH     0037H     UNIT       ?PR?_GET_BUF_CRC?F85X_CRC
CODE        0331H     002DH     UNIT       ?PR?_SRC_GET_PAGE?F85X_COMM_UART
CODE        035EH     0028H     UNIT       ?PR?_FLASH_MODIFY?F85X_FLASH
CODE        0386H     0024H     UNIT       ?PR?_UPDATE_CRC?F85X_CRC
CODE        03AAH     0023H     UNIT       ?PR?_UART_RECEIVE?F85X_COMM_UART
CODE        03CDH     001EH     UNIT       ?PR?SRC_GET_PAGE_INFO?F85X_COMM_UART
CODE        03EBH     0012H     UNIT       ?PR?F85X_COMM_UART
```

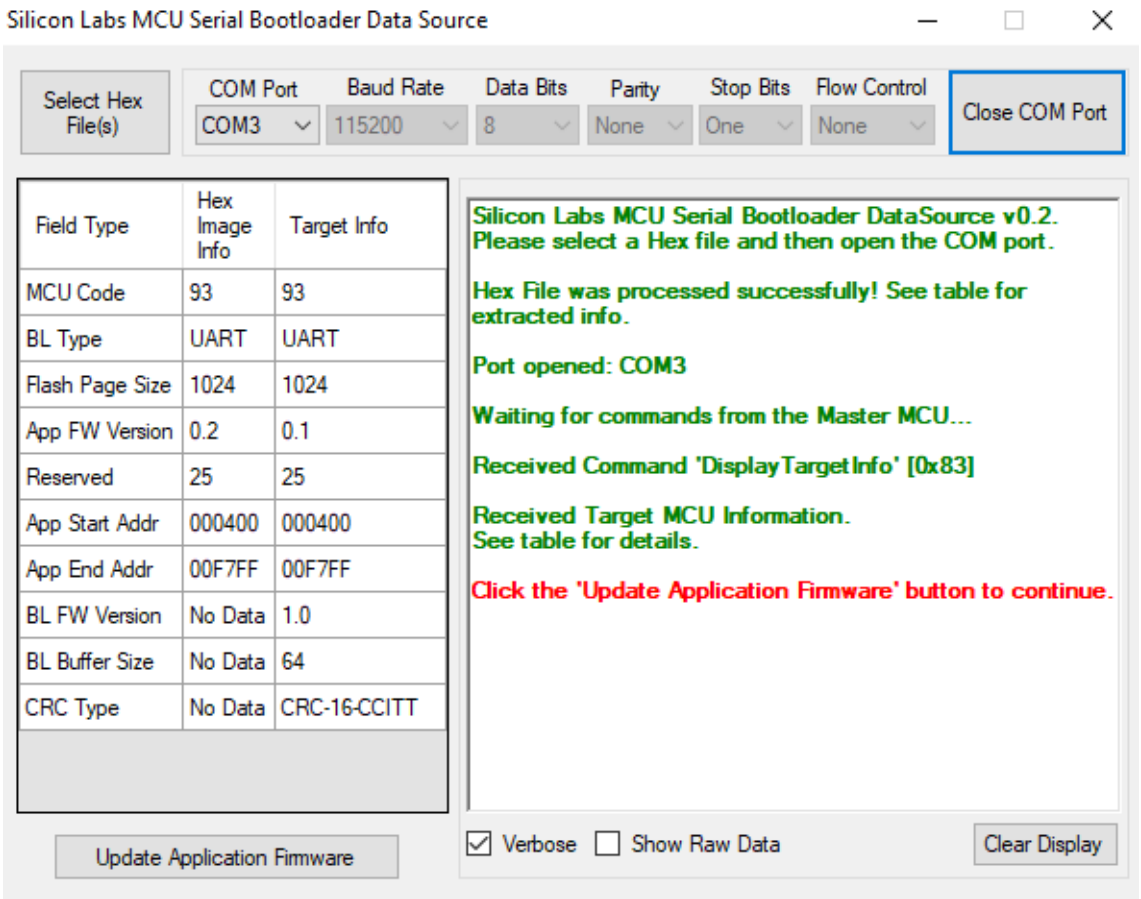
7.5. ábra. A fordítás memóriaképe

7.6. Frissítés menete

Az ilyen módon történő frissítés menete a következőből áll. A legelső és legfontosabb feladat, magának a *bootloader* programnak a vezérlőre juttatása. Ezt csak a hagyományos módszerrel, a programozólábakon lehet elvégezni. Viszont ez után, A Silabsos program segítségével már USB-n keresztül is rá lehet tölteni a felhasználói programot. Ehhez a fordítás után kapott *.hex* fájlt a programba be kell tölteni, és *bootload* módba helyezni a csatlakoztatott vezérlőt. Ez a mód több féleképpen érhető el.

Az eszköz alapból *bootload* üzemben indul, viszont ha a felhasználói memóriaterület végén, az *Info Block*ban megtalál egy bitsorozatot, akkor átvált felhasználói módra, és a 0x0400H címre irányítódik át. Viszont megmondhatjuk neki, hogy mégis csak *bootload* módban induljon. Ehhez én két különböző módszert is bevetettem. Egyrészt felhasználói szoftverből, ha a P0.2-es lábat a felhasználó fizikailag földre húzza, akkor egy memória túlcímzés miatt újraindul, és itt egy ERROR regiszterben látni fogja, hogy mi miatt indult újra. Viszont ha a szoftver meghalna egy keményebb módszert is alkalmaztam, mégpedig, hogy ha az induláskor szintén ez a láb, szintén földre van húzva, akkor is *bootload* módban indul. Ezekre azért volt szükség, mert az első frissítés után a felhasználói memóriaterület végébe bekerülnek azok a bitek, amik következtében, mindig ebbe a módba indulna, így nem lenne lehetőségünk többször módosítani a programon.

A frissítést a Silabsos programból csak akkor indíthatjuk el, ha a mikrokontroller *bootload* módban indult, és jelzett, hogy ő kész a műveletre (7.6 ábra). A bootloading a befejeztével (11.8 ábra) automatikusan a felhasználói kódra ugrik át, és már a mi programunk fut rajta.



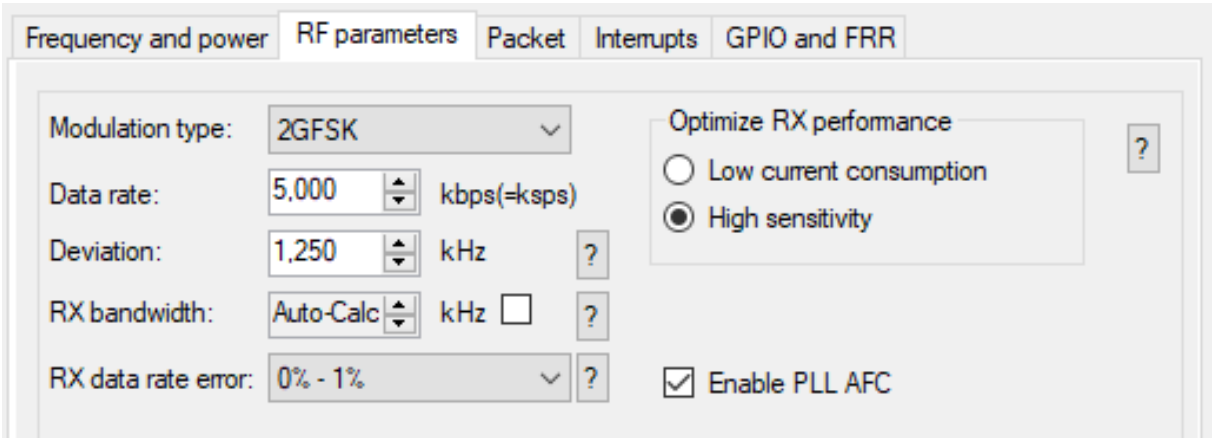
7.6. ábra. A bootload folyamat megkezdődhet

8. fejezet

Rádiós egység

8.1. A SMOG-1 kommunikációs rendszere

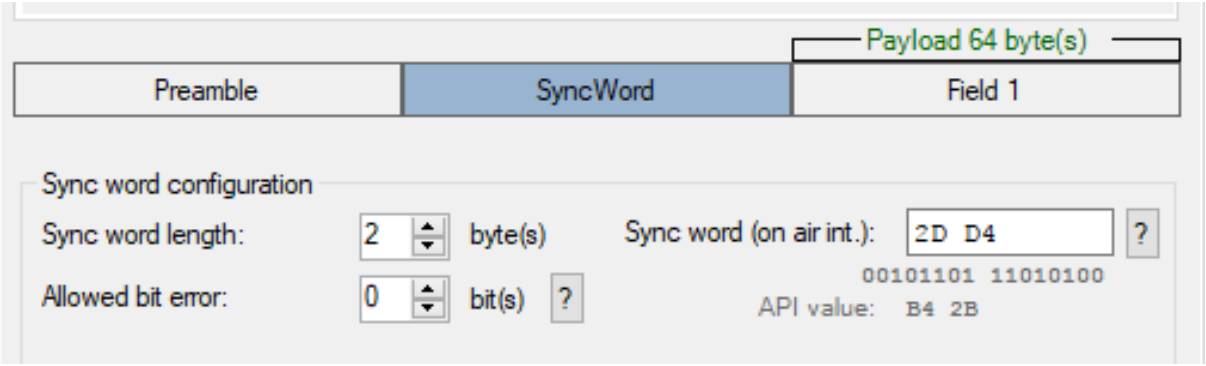
A SMOG-1 alapvetően kétirányú kommunikációt valósít meg, hisz az űrben mért adatokat valahogy el kell juttatnia a csapat számára, és a Földről való irányításra is szükségünk van. A lefele irányú kommunikáció $437,345\ MHz$-en valósul meg 2GFSK moduláció segítségével. Ez egy egyszerű FSK eljárás $1,250\ kHz$-es frekvencia lökettel. A rövidítésben A *G* a *Gaussian* tagért felel, ami a gyakorlatban az jelenti, hogy a jel egy Gauss-szűrőn halad keresztül, ami a sávszélességre lesz jótékony hatással. Az adatsebesség a vezérlő irányban mindig 5000 *bps* míg *downlink* irányban több különböző értéket vehet fel, például 1250 *bps*, 5000 *bps*, vagy 12500 *bps* értéket. Ezekre azért van szükség, mert a kis műholdnak csak korlátozott energiája áll rendelkezésre, ezért hanem a legjobbak a vételi viszonyok, akkor lejjebb tud váltani egy alacsonyabb adatsebességre, ezzel egy robusztusabb megoldásra. Az *uplink* irányban erre nincs szükség, hiszen az teljesítményben itt sokkal nagyobb tartalékok vannak a rendszerben. A nagyobb adatsebességekre pedig azért van szükség mert Magyarország felett csak korlátozott ideig fog tartózkodni, és jó lenne ez idő alatt minél több adatot eljuttatni a földi állomásra.



8.1. ábra. A WDS konfigurációs beállításai

A SMOG-1 a kommunikáció során csomagokat vesz és küld amire szintén vannak megkötések. Ezek a csomagok egy úgynevezett *Preamble* szakasszal kezdődnek, amik felváltott 1-es és 0-ás bitnek felelnek meg. Erre azért van szükség hogy a vevő megtalálja a helyes bitszinkront. Ezután egy két bájtos *SyncWord* következik, ami a bájtszinkron felismeréséért felel, és ez után jönnek a hasznos adatok. Itt pontosan 64 adatbájt érkezik,

amit egy CRC (*Cyclic Redundancy Check*) követ, ami az esetleges bithibák felismeréséért felel.



8.2. ábra. A SMOG-1 csomagszerkezete

8.2. A rádió felkonfigurálása

A fentebb vázolt konfiguráció rádióra való töltésének nagyon hosszadalmas és kockázatos módja lenne az adatlapból kikeresett értékek SPI interfészen való leküldése. Ha csak egy regisztert elfelejtünk a helyes értékre beállítani (a tudtunkon kívül) akkor a SMOG-1 vétele nem fog megtörténni. Éppen ezért a Silabs a rendelkezésünkre bocsájtott egy WDS (*Wireless Development Suit*[17]) nevű programot, ahol egy grafikus felületen "összekattingathatjuk" a számunkra megfelelő beállításokat, és ő ebből egy *header* fájlt generál a programunkhoz. Ezt sokkal egyszerűbb *include*-olni, és a megfelelően strukturált tömböt, szintén az SPI buszon leküldeni. Ennek a programnak a részletei láthatóak a 8.1 és a 8.2 ábrán.

8.3. Vétel

A rádiónak több állapota is van, például felkonfigurálás után *Ready State*-ben van, míg ha minimális energiafogyasztást szeretnénk, akkor ezt a *Sleep* módba helyezéssel elérhetjük. Számomra ezek most nem voltak szükségesek, Elég volt az *RX State*, amibe szerencsére bármilyen állapotból gyorsan áthelyezhetjük. Csak ebben a módban vesz a vevő, így ha szeretnénk venni a SMOG-1 jeleit akkor át kell ide manuális állítani. Ha ezt megtettük akkor a rádió elkezdi monitorozni az étert, hogy megtalálja-e a *Preamble*-t és az azt követő szinkron bájtokat. Ha es megtörtént, a következő bájtokat lementi az *RX FIFO* nevű tárolójába és egy belső regiszter jelezni fogja, hogy csomag jött. Ha szeretnénk, hogy nekünk kifele (a mikrokontroller fele) is szóljon, akkor még a konfigurációnál ki kell választani, hogy erről az esetről – csomag vétel – küldjön egy magszakítást kérést, amit a C8051 P0.1-es lábára kötöttem. A vezérlőnek itt egy külső megszakításkérést állítottam be, így nagyon gyorsan tudom lereagálni a vételt. Ezen kívül még számos *interrupt* is konfigurálható, például egy hibásan leküldött *command* esetét is le tudjuk így reagálni. Miután a rádió szólt, hogy történt valami, a *Master-Slalve* viszony miatt nekünk kell megkérdezni, hogy milyen esemény következett be. Itt a megfelelő regiszterek visszaolvasásával megtudhatjuk a megszakítás kérés okát, és a csomag vétel esetén tudjuk folytatni a szükséges lépéseket. Ilyen esetben az *RX FIFO* bufferből ki kell olvasni az adatokat, hisz ez csak 64 bájtot képes tárolni, így egy csomag után megtelik. Ha ez

megtörtént, akkor törölni kell a *fifo*-t és visszaállítani az *RX State*-et. Én csak ezután küldtem vissza a bájtokat soros porton, hiszen minél hosszabb ideig van a rádió vételi állapoton kívül, annál nagyobb a valószínűsége, hogy lemaradunk egy csomagról.

8.4. RSSI

A vétel során a csomagokon kívül még egy értékes információt tudunk kinyerni a rádióból, ezt pedig az RSSI (*Received Signal Strength Indicator*). Ebből kétféle található meg az SI4460-ban. Az egyiket úgy méri, hogy amikor megkérdezzük az értéket, akkor megméri az aktuális jelszintet a bemenetén, és azt adja vissza. Ezzel sok mindent nem tudunk kezdeni így feltételezhető, hogy a ténylegesen a vett jel erősségét a másikkal tudhatjuk meg. Azt az értéket, mindig a csomagvétel elején (például a szinkronbájtoknál) méri, így ezzel már a tényleges hasznos adatot lehet kinyerni. Ez azonban még nem egy dBm érték, ehhez át kell alakítani. Ezt az RSSI-t egy bájton tárolja, így egy 0-tól 255-ig terjed egész felbontásban. Ezt az értéket le kell osztani kettővel, hogy $0,5$ dBm-es felbontást kapjunk. Ilyenkor már 0-tól 127,5-ig ér a tartományunk. Ez azonban még nem megfelelő számunkra, hisz a vevő érzékenysége -126 dBm-ig terjed. Ezért egy ofszet ki kell vonni még belőle, és így kapjuk meg a vett jel erősséget dBm-ben.

9. fejezet

Antenna

9.1. Földi állomás

A műhold vételére és vezérlésére szolgáló elsődleges állomás, a Budapesti Műszaki és Gazdaságtudományi Egyetem E épületének a tetején található. Itt egy 3 és egy 4,5 méter átmérőjű parabolaantenna helyezkedik el. Ilyen méretekkel már megfelelő irányélességet, és antennanyereséget el lehet érni. Az új forgatónak köszönhetően, már folyamatos műhold követésre is képes. Könnyen megérthető, hogy egy ilyen nagyobb mennyiségben készített SMOG-1 vevőhöz valami kisebb, olcsóbb és egyszerűbben kezelhető antennát célszerű választani.

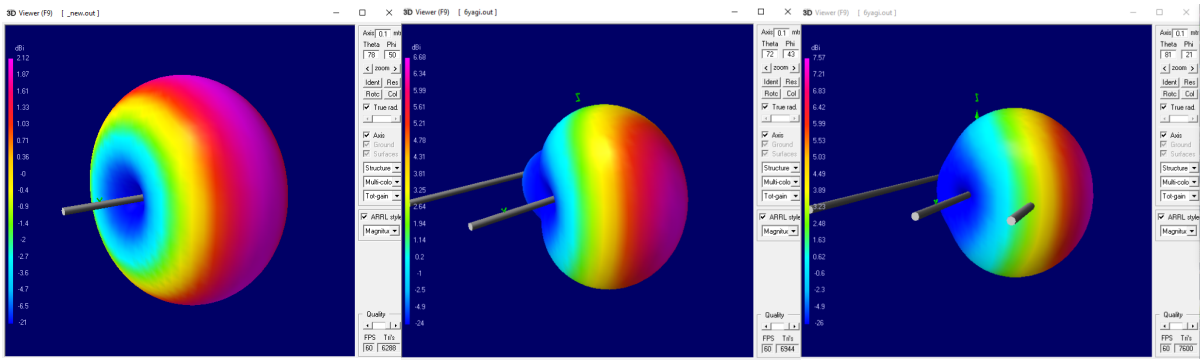
A Masat-1 esetén is hasonló volt az elvárás, tehát egy egyszerű, olcsó és könnyen kezelhető antennát kellett találni. Ez abban az esetben egy 6 elemes *yagi* antennát jelentett, aminek mérőszalag és colostok az alapanyaga. Ennek az előnye a fentebbiek felett még könnyen javítható, és akár otthon is könnyen elkészíthető. Mellesleg, egyszer már jól teljesített egy kisműhold vételénél, szóval okkal remélhető, hogy ismét sikeres lesz ez a dizájn.

9.2. Yagi antenna

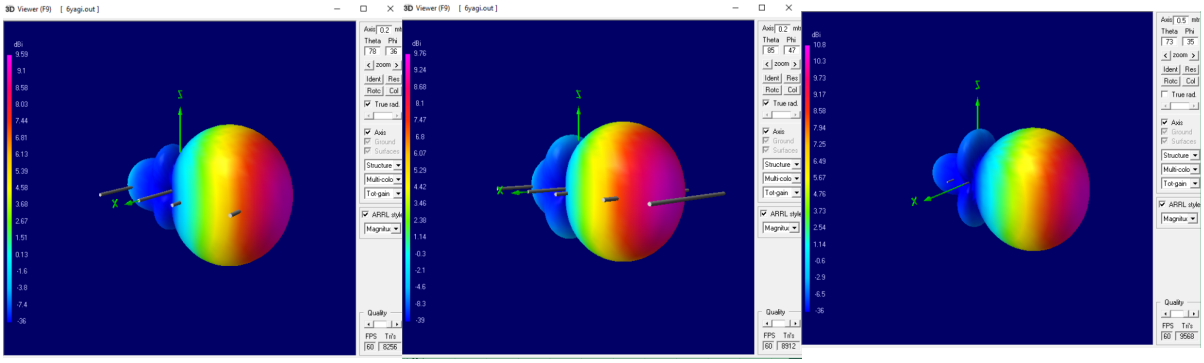
Az általam elkészített *yagi* antenna egy aktív, és öt passzív elemből áll. Az aktív elem a feszültséggel meghajtott elem (vevők esetében ennek a feszültségét figyeljük), míg passzív elemek, a kedvező iránykarakterisztikáért felelnek. A passzív elemek is két részre pontha-tóak, reflektorokra és direktorokra. A reflektor az aktív elem "mögött" míg a direktor az aktív elem "előtt" módosítja számunkra kedvezően a karakterisztikát. Az "előtt" és "mögött" kifejezést itt úgy kell érteni, hogy a kívánt irányítottsághoz képest. Tehát a meghajtott elemhez képest a direktor, abba az irányba helyezkedik el, amerre szeretnénk az antennával adni (vevő esetében amerről venni szeretnénk). Jelen esetben egy meghajtott elem, egy reflektor, és 4 direktorra volt szükség.

9.3. Tervezés

Az antenna tervezéséhez is szimulációjához a *4nec2*[18] programot használtam. Ez a háttérben differenciálegyenletek és peremfeltételek sokaságát számolja ki a szoftver, és ezekből kapja meg a végül megjelenített értékeket. A tervezés során egyszerre kellett figyelnem a 50 Ω-os impedanciára, a helyes reflexiótényezőre, és az antennának a nyereségére is.



(a) 1 elem (b) 2 elem (c) 3 elem

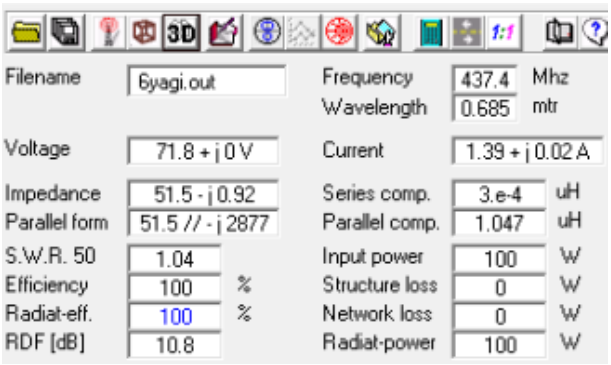


(d) 4 elem (e) 5 elem (f) 6 elem

9.1. ábra. A kapott iránykarakterisztika különböző elemszámok esetén

Először az aktív elemet vettem adtam hozzá az antennámhoz, és ehhez próbáltam a megfelelő hosszúságot megtalálni. Ideális esetben ez $\frac{\lambda}{4}(0.17149\ m)$ lenne, azonban a mérőszalag antenna miatt adott volt a vezető sugár, így ettől egy kicsit el kellett térnem. A többi elemnél az elem hosszát, és a táplált tagtól való módosítva lehetett hangolni az antennát. A 3D szimuláció során szemmel is megvizsgálhattam a kapott végeredményt. Ez az egyszerű dipol karakterisztikából lépésenként alakult át egy irányított antennává (9.1, 11.9).

A 11.9 ábrán jól látható, hogy a főirányban az antenna nyeresége 10.8 dB, az előre-hátra viszony 14 dB és az ehhez tartozó irányélességi szög 60°. Ez számunkra jó, hiszen nagyban megkönnyíti a SMOG-1 követését az égen. A 11.10 ábráról leolvasható, hogy az állóhullámarány lényegében 1, míg a reflexiós tényező meghaladja a -45 dB-t.



9.2. ábra. A szimulátor által számolt értékek

10. fejezet

Kvalifikációs mérések

10.1. Rádió mérése

Elsőnek a a rádiónak az alap funkcióját mértem meg, hogy képe-e csomagokat fogadni. Mivel addig módosítottam a kódomat, amíg ez nem sikerült, így következőnek minősítési méréseket végeztem. Ennek keretén belül különböző adatsebességek esetén mértem meg az érzékenységet, ezen a szinten a bit-, és csomaghiba arányokat, és a frekvencia szelektivitást.

10.1. ábra. A mérési elrendezés

A mérést 2017. 12. 01. napon végeztem a V1504 laborban. A vizsgálat során előre ismert teljesítménnyel ($-80\ dBm$), ismert karakterekből álló fix mennyiségű (49 db) csomagot küldtem a vevő SMA csatlakozójára, állítható csillapítókon keresztül. A csillapítók közül az egyiknek a felbontása 10 dB míg a másiké 1 dB. Az érzékenységet úgy állapítottam meg, hogy a teljes vett adatokra, 1%-nyi BER-t (*Bit Error Rate*) engedtem meg. Ez elsőre egy minimális hibaszintnek vélhető, viszont ha jobban belegondolunk, akkor könnyen kiszámítható, hogy a 64 bájtos csomagból már 5 bit meghibásodása szükséges, ami pedig már egy elég zavaró hiba a visszafejtésnél. Ezt az értéket az állítható csillapítók segítségével értem el. Utánna ezekből az adatokból megállapítottam a PER-t (*Packet Error Rate*). A csomagok elvesztése a szinkron bájtok meghibásodása miatt következik be. Ezt követően 10 dBm-mel nagyobb teljesítményen egy frekvenciaszelektivitást mértem. Azért alkalmaztam magasabb adóteljesítményt, hogy a bekövetkező hibákért, csak a frekvencia elállítás legyen felelős. Ezeket a méréseket 3 különböző adatsebességre végeztem el, ezek a következőek: 1250 *bps*, 5000 *bps* és 12500 *bps*.

A 10.1 táblázatban láthatóak a mérési eredmények. Az RSSI értéket az SI4460 bemenetén méri a rádió. Ezek az elvárt viselkedést tanúsították, azaz minden esetben a rádió érzékenységi szintjével ($-126\ dBm$) körüli értéket tapasztaltam. Az én értékeim kicsit magasabb értékek, mivel az adatlapi értékeket kisebb bitsebességnél adták meg. Az 1%-os bithiba arányt nem volt egyszerű pontosan meghatározni, hiszen a csillapítóim csak decibeles felbontásra voltak képesek, és felül látható, hogy a célként kitűzött értéket körülbelül 0.5%-kal sikerült körbelőnöm. Az adatsebesség növelésével az érzékenysége csökkent, hiszen kevesebb ideje volt a vevőnek meghatározni az analóg jelhez tartozó logikai értéket.

10.1. táblázat. A laborbeli mérések eredményei különböző bitsebességeknél

1250 *bps*					
Csillapító	Hibás bit	Összes bit	BER	PER	RSSI
59 dB	173	21504	$0,008$	$0,1429$	-118 dBm
60 dB	216	14336	$0,015$	$0,4286$	-120 dBm
50 dB	Doppler tűrő képesség				
-1490 Hz			$+940$ Hz		

5000 *bps*					
Csillapító	Hibás bit	Összes bit	BER	PER	RSSI
53 dB	63	20992	$0,003$	$0,1633$	-113 dBm
54 dB	260	14848	$0,018$	$0,4082$	-113 dBm
44 dB	Doppler tűrő képesség				
-3650 Hz			$+2960$ Hz		

12500 *bps*					
Csillapító	Hibás bit	Összes bit	BER	PER	RSSI
49 dB	175	22016	$0,008$	$0,1225$	-104 dBm
50 dB	220	15872	$0,014$	$0,3674$	-105 dBm
40 dB	Doppler tűrő képesség				
-6520 Hz			$+5990$ Hz		

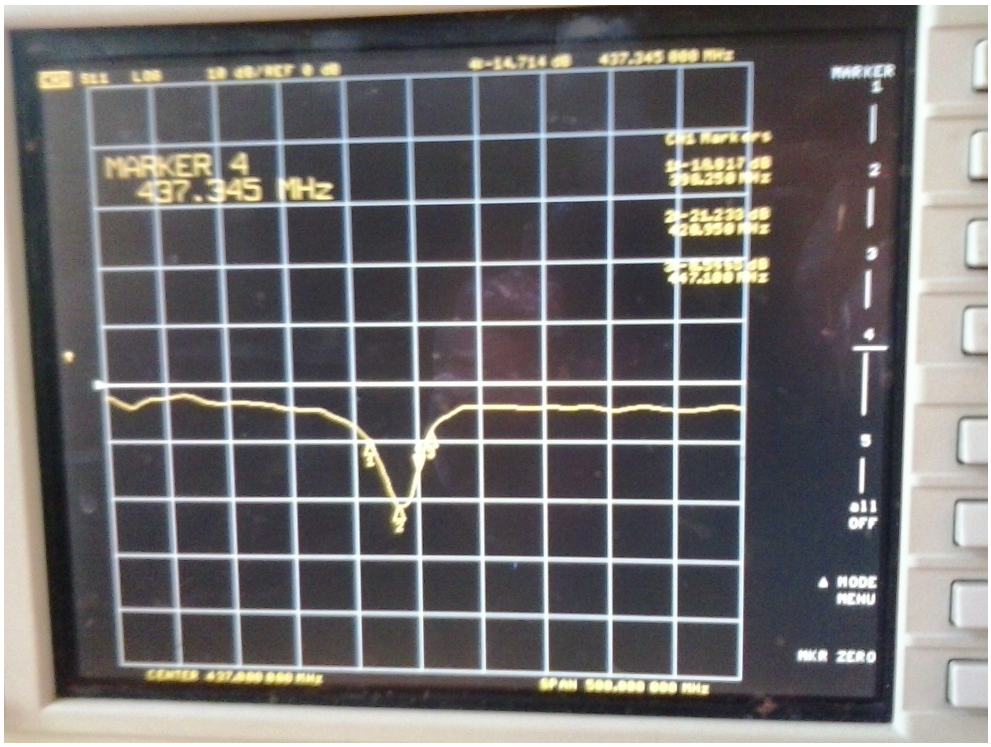
A csomagvesztéseknél láthatjuk, hogy nem független a bithiba aránytól, viszont annál jóval meredekebb, hisz ha a kezdeti szinkronbájtok, vagy a *preamble* meghibásodik akkor a rádió nem képes megtalálni a bit vagy a bájtszinkront. A frekvenciaszelektivitás értéke viszont az adatsebesség növelésével javul, ami annak tudható be, hogy a az adatsebesség növelésével a sávszélesség is nőni fog, így nagyobb sávszélességnél nagyobb frekvenciahiba szükséges a vétel elvesztéséhez.

A mérésnél a -80 dBm-es adóteljesítmény és $50-60$ dB csillapító következtében a vevő bemenetén (SMA csatlakozó) olyan $-130-140$ dBm jelszint volt mérhető, ez pedig kétségessé teszi a mért értékek valódiságát. Ha kiszámoljuk a termikus zajteljesítményszintet ($10 \cdot log(1,38 \cdot 10^{-23} \frac{J}{K} \cdot 300\ K \cdot 7500\ Hz) + 30$), akkor arra -135 dBm-t kapunk. GFSK jelnél pedig egy 6 dB-vel fölötte kell lennie a jelszintnek, hogy azt demodulálni lehessen. Így valószínű hogy az 12150 *bps*-es esetben a szabadtérben csatolódott át a jel. Ezt kiküszöbölni csak nagyobb távolságú mérés esetén lehet, ebben az esetben pedig a vevő, és az adó is a számítógépemhez volt csatlakoztatva USB-vel így korlátozott volt az adó és a vevő maximális távolsága.

10.2. Antenna

Az antenna a korábban említett alapanyagokból, colostokból és mérőszalagból épült. A 11.11 ábrán látható az elkészült darab. 2017. 12. 06-án a V1504-es laborban egy hálózatanalizátorra kapcsolva megmértem a reflexiós tényezőt (S11) paramétert. Az 10.2 ábrán a szimulált értékekhez hasonló 11.10 menetet láthatónk. A frekvenciában egy kicsit lejjebb van tolódva, és a mért érték is elmarad a szimulálthoz képest. A hibákat a nem milliméter pontos illesztések, és az esetleges mérési pontatlanságok mellett, a mérés környezetében lévő rengeteg fém okozta. De ettől függetlenül szépen látható, hogy a kívánt a frekven-

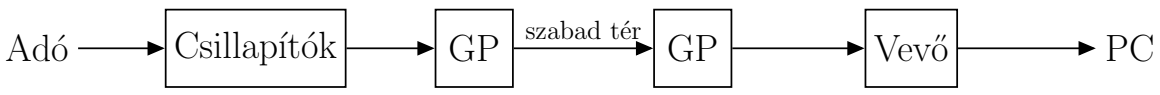
cián is -14 dB-s elnyomás van a rendszerben, 429 MHz–en pedig 21 dB. A megépített antenna nyereségét, irányélességi szögét, a keresztpolarizációs elnyomást, és az előre hátra viszont csak a terepi mérés során tudtam meghatározni.



10.2. ábra. A mért reflexiós tényező

10.3. Terep

A terepi mérést 2017. 12. 08-án, reggel végeztem a Gellért hegyen, míg a konzulensem a BME E épületének a tetején az adót kezelte (11.12 ábra). Az adót egy fix teljesítményű (0 dBm) SDR (*Software Defined Radio*), két változtatható csillapító (10 és 1 dB-s felbontás), és egy GP antennából áll. Ez az antenna körsugárzó, így a nyeresége az izotróp sugárzóhoz képes 2 dB. A szabadtéri szakaszcsillapítás az E tető és a Gellérthegy közötti $1,2$ km-en 87 dB.



10.3. ábra. A mérés blokkvázlata GP vevőantennával

A vételi oldalon első esetben én is egy GP antennával voltam, ehhez csatlakozott a vevő, ahhoz meg egy USB kábel segítségével a számítógépem. Ilyenkor az adó oldali csillapítás állításával megkerestük az 1%-os BER-hez tartozó RSSI értéket. Ez azért volt fontos, mert a következő mérésnél a GP antennát az általam tervezett yagi antennára cseréltem, és a csillapítók változatlanul hagyása mellett, az így vett, és az előző mérésnél kapott RSSI segítségével meghatározható a yagi antenna nyeresége. Ebben az esetben az antenna és a vevő között egy $1,5$ m-es kábel körülbelül plusz 1 dB csillapítást jelent.

Ez után a csillapítókat kiiktattuk az adóból, így rendelkezésemre állt elég teljesítmény, a következő mérésekhez, amiknél az vett RSSI abszolút értéke nem, csak a relatív változás volt fontos. Így mértem meg a 3 dB-hez tartozó irányélességi szöget, a keresztpolarizációs csillapítást, és az előre-hátra viszonyt.

10.4. ábra. A mérés blokkvázlata yagi vevőantennával

10.2. táblázat. A terepi mérés eredményei

Jelszint			
	Antennánál a levegőben	SMA csatlakozón	Si4460 bemenetén
GP	$-93\ dBm$	$-92\ dBm$	$-70\ dBm$
Yagi	$-123\ dBm$	$-110\ dBm$	$-92\ dBm$

3 dB-es irányélesség	70°
Keresztpolarizációs csillapítás	13 dB
Előre-hátra viszony	15 dB

A mért értékekkel elégedett vagyok, hiszen az antenna nyeresége a kitűzött célt, és az irányélessége és keresztpolarizációs csillapítása is hasonló a szimulációs értékekhez. Ezeknek az adatoknak a birtokában feltételezhető, hogy a SMOG-1-et is képes lesz venni, amikor az a Föld körül kering. A GP antenna esetében nem sikerült az érzékenységet jól kimérni, a mért érték jelentősen elmaradt a várakozásoktól. Ez a Gellért hegyen feltételezhetően nagy interferenciaszint, és a környezetben megtalálható sok fém hatása lehetett. Az érzékenységet így a yagi antennánál mértük meg, itt 36 dB plusz csillapítást volt képes elviselni 0 dBm-es adóteljesítménynél, 87 dB szakaszcsillapításnál.

10.4. Összefoglalás

A szakdolgozat keretén belül megterveztem egy SMOG-1 műhold vételére alkalmas vevő egységet kapcsolási és ültetési szinten, elkészítettem a működéshez szükséges kódot, és szimuláltam egy yagi antennát. Ezeket el is készítettem, és egy terepi mérés alkalmával megbizonyosodhattam felőle, hogy megfelelően működnek. A SMOG-1 adóteljesítménye 20 dBm lesz, így ez egy plusz tízszeres áthidalható távolságot jelent számunkra. Ezenkívül van még egy 36 dB-es tartalékunk a rendszerben a csillapítók következtében, ami egy plusz hatvanszoros plusz távolság áthidalást eredményez. És az adatsebességgel is nyerhetünk 6 dB-t ami még egy kétszeres nyereség. Így összességében egy 1440 km jön ki erősen interferált környezetben, ami egy 500 km-es pályához elég biztató. A horizonton (körülbelül 3600 km) valószínűleg nem lesz képes venni, de egy pár fokos eleváció esetében már igen. A jövőre nézve a szoftveren érdemes módosításokat végrehajtani, hogy a jelenleg készülő földi állomás, csak vételre képes programjával integrálni lehessen, így egy kényelmesebb, grafikus felületen szemlélhessük a vett adatokat. Ezenkívül hátra van még az éles alkalmazás, amikor a műhold fellövése után, megpróbáljuk ténylegesen a Föld körül keringő SMOG-1-et venni. A SMOG-1 fellövése, a jelenlegi tervekben 2018 Q2-ben várható.

Köszönetnyilvánítás

Szeretném megköszönni konzulensem, Dudás Levente segítségét, hogy bármilyen triviális kérdéssel is fordultam hozzá, mindig időt szakított rám és addig magyarázta amíg meg nem értettem. Hála neki megszerettem az addig távolról elkerült, CAD programok használatát, és lényegében megtanultam áramköri szinten tervezni. A kézi forrasztás rejtelmeibe beavatott, és nem hagyta, hogy úgy szerezzek villamosmérnöki diplomát, hogy nem forrasztottam QFN alkatrészt. Azt, hogy ha kellett, akkor miattam maradt bent tovább a laborban, és segített bármilyen témában.

Köszönöm Kristóf Timurnak, hogy mindig segített a laborban ha elakadtam valamivel, és kérés ajánlotta fel ezt a segítséget. Herman Tibornak, hogy a valamilyen Silabsos dologgal akadtam el, akkor bármikor írhattam neki egy e-mailt, és ő mindent elmagyarázott.

És végül a V1504-es labor közösségének, hogy mindig tartják egymásban a lelket, és mindig jó hangulat volt a laborban.

Rövidítés jegyzék

BER	Bit Error Rate
CRC	Cyclic Redundancy Check
DVB-T	Digital Video Broadcast - Terrestrial
GFSK	Gaussian Frequency Shift Keying
GMSK	Gaussian Minimum Shift Keying
GPIO	General-purpose input/output
IC	Integrated Circuit
IPD	Integrated Passive Device
LED	Light-Emitting Diode
LNA	Low-Noise Amplifier
PER	Packet Error Rate
RF	Radio frequency
RGB	Red Green Blue
RSSI	Received Signal Strength Indicator
SAW	Surface Acoustic Wave
SDR	Software Defined Radio
SMBus	System Management Bus
SPI	Serial Peripheral Interface Bus
TCXO	Temperature Compensated Crystal Oscillator
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
WDS	Wireless Development Suit

Irodalomjegyzék

[1] Masat-1 honlapjahttp://cubesat.bme.hu/ Elérés dátuma: 2017. december 8.

[2] SMOG-1 honlapja http://www.gnd.bme.hu/smog1/index.php Elérés dátuma: 2017. december 8.

[3] SI1062 adatlapja http://www.silabs.com/documents/public/data-sheets/Si106x-8x.pdf Elérés dátuma: 2017. december 8.

[4] C8051F930 adatlapja http://www.silabs.com/documents/public/data-sheets/C8051F93x-92x.pdf Elérés dátuma: 2017. december 8.

[5] SI4460 adatlapja https://www.silabs.com/documents/public/data-sheets/Si4464-63-61-60.pdf Elérés dátuma: 2017. december 8.

[6] Silicon Labs honlapja http://www.silabs.com Elérés dátuma: 2017. december 8.

[7] CP2102 adatlapja http://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf Elérés dátuma: 2017. december 8.

[8] IPD adatlapja https://www.silabs.com/documents/public/application-notes/AN904.pdf Elérés dátuma: 2017. december 8.

[9] SAW szűrő adatlapja http://www.golledge.com/pdf/products/specs/ma09629.pdf Elérés dátuma: 2017. december 8.

[10] LNA adatlapja http://www.analog.com/media/en/technical-documentation/evaluation-documentation/ADL5523.pdf Elérés dátuma: 2017. december 8.

[11] KiCad honlapja http://kicad-pcb.org/ Elérés dátuma: 2017. december 8.

[12] GitHub honlapja https://github.com/ Elérés dátuma: 2017. december 8.

[13] Farnell online áruház http://hu.farnell.com/ Elérés dátuma: 2017. december 8.

[14] Lomex online áruház http://lomex.hu/hu/homepage Elérés dátuma: 2017. december 8.

[15] Modular Bootloader Framework https://www.silabs.com/documents/public/application-notes/AN533.pdf Elérés dátuma: 2017. december 8.

[16] UART Bootloader https://www.silabs.com/documents/public/application-notes/AN778.pdf Elérés dátuma: 2017. december 8.

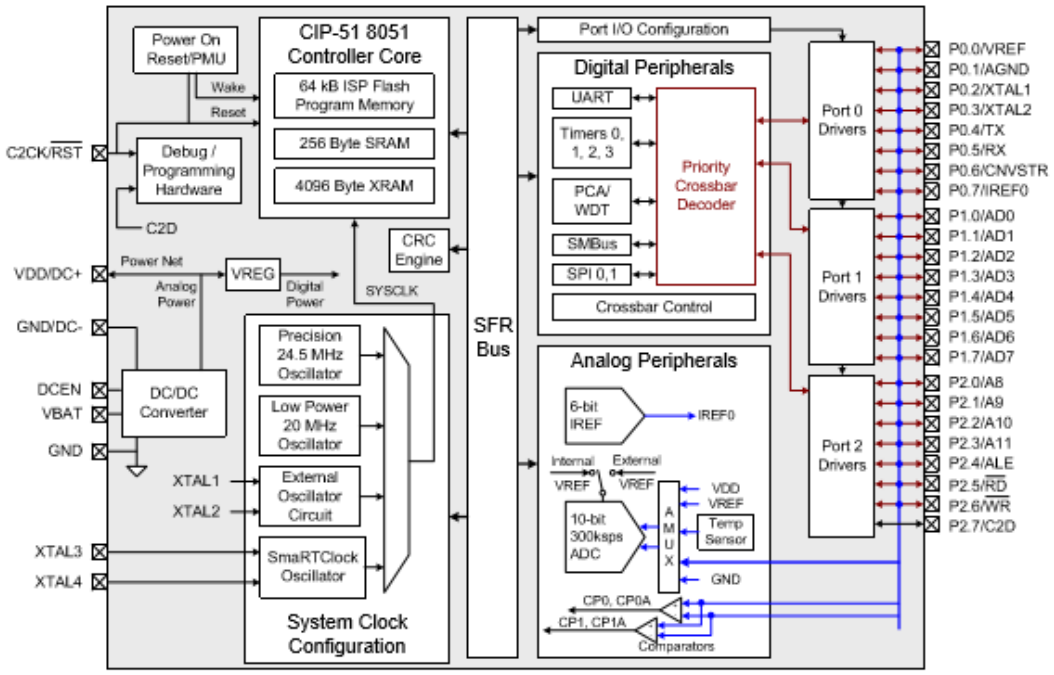
[17] WDS elérhetősége https://www.silabs.com/products/development-tools/software/wireless-development-suite Elérés dátuma: 2017. december 8.

[18] 4nec honlapja http://www.qsl.net/4nec2/ Elérés dátuma: 2017. december 8.

Ábrák jegyzéke

11. fejezet

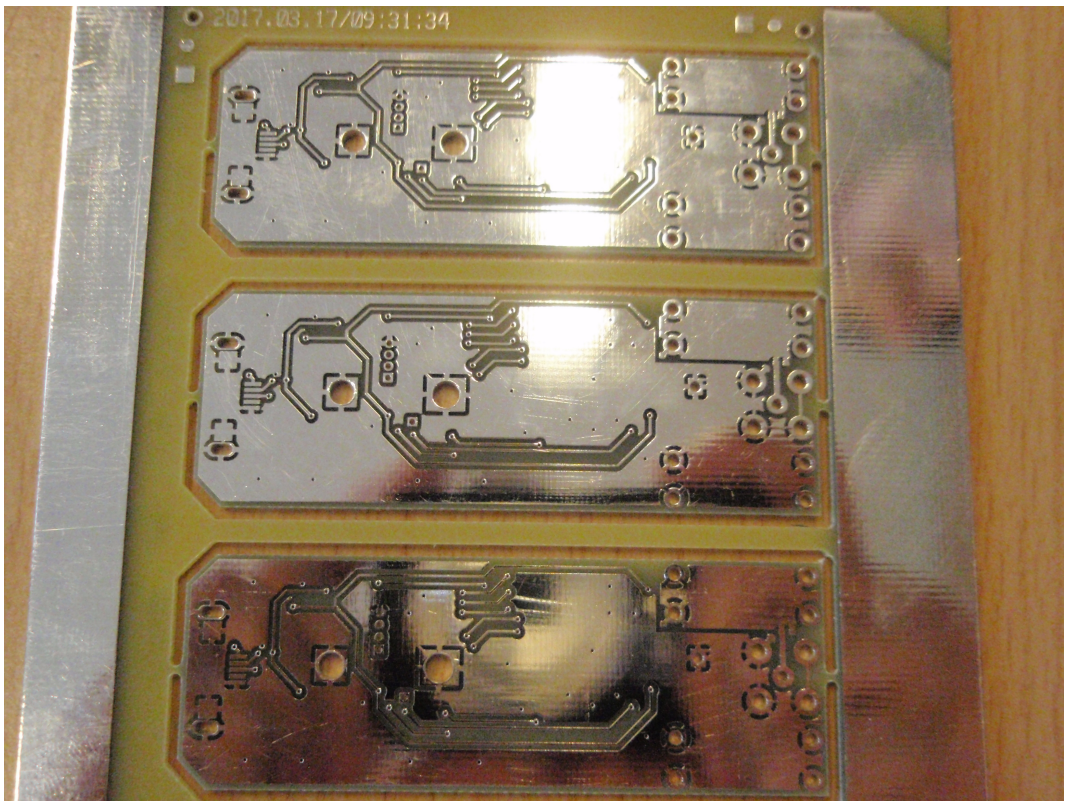
Függelék



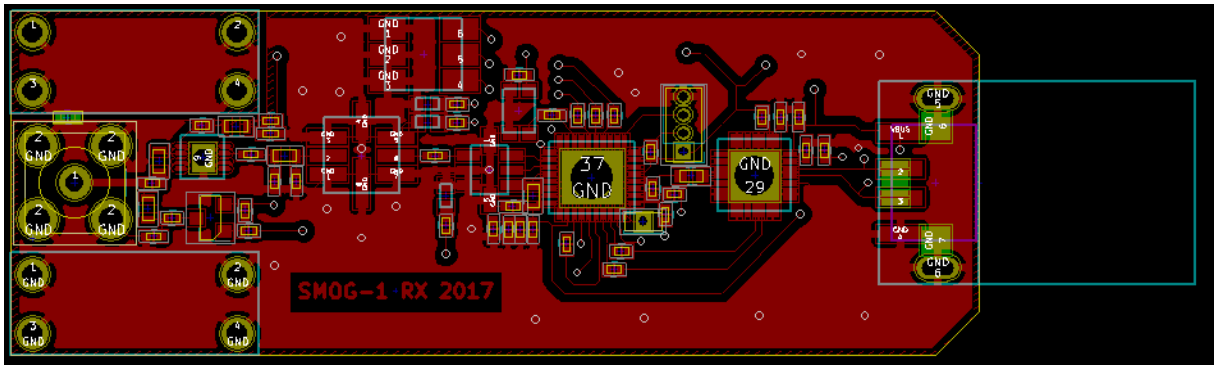
11.1. ábra. A C8051F930 blokkdiagramja[4]

Blinky programkódja

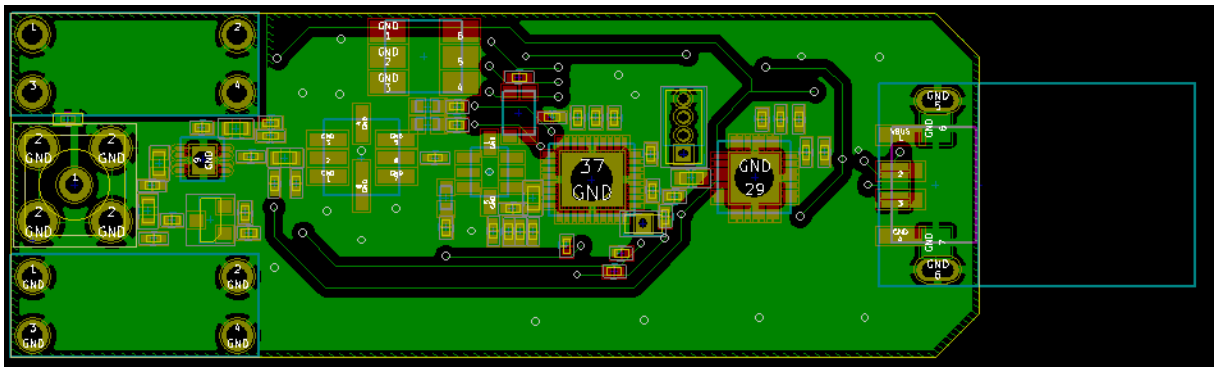
```
#include <SI_C8051F930_Register_Enums.h>                    // SFR declarations

sbit RED     =     P1^4;
sbit GREEN     =     P1^5;
sbit BLUE     =     P1^6;

void delay()
{
    long t=100000;
    while(t) t--;
}

int main (void)
{
    CLKSEL = 0x04; // 0000 0100
    PCA0MD &= ~0x40; // WDT disable
    P0MDOUT = 0x9d; // 1001 1101
    P1MDOUT = 0x7d; // 0111 1101
    XBR0     = 0x01; // 0000 0001
    XBR1     = 0x40; // 0100 0000
    XBR2     = 0x40; // 0100 0000
    P1=0;

    while (1) {
        P1=0;
        delay();
        P1=0x9f; // 1001 1111
        delay();
        P1=0;
        delay();
        P1=0xaf; // 1010 1111
        delay();
        P1=0;
        delay();
        P1=0xcf; // 1100 1111
        delay();

    }                                   // Spin forever
}
```



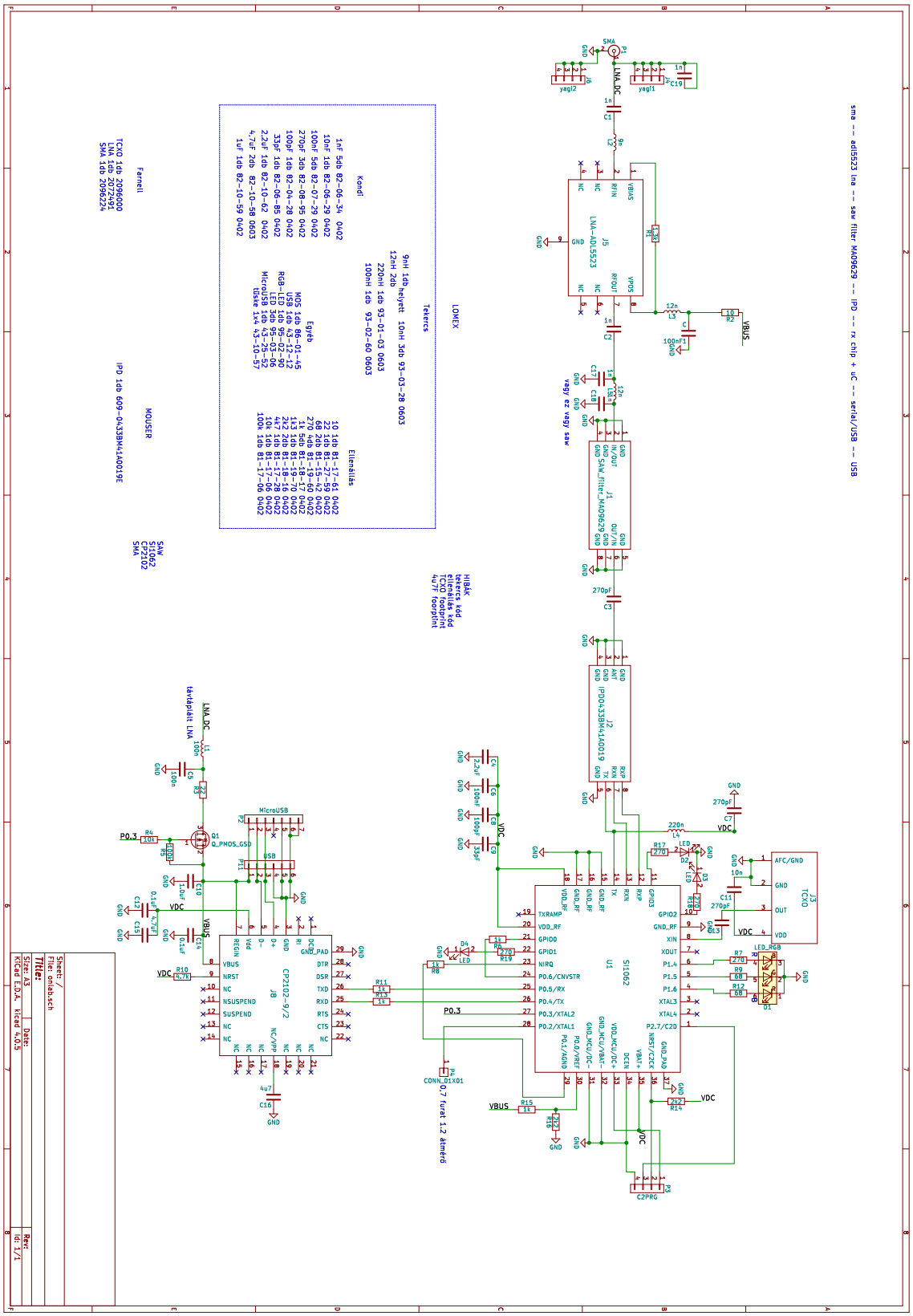
11.2. ábra. Az elkészült NYHL bot oldala



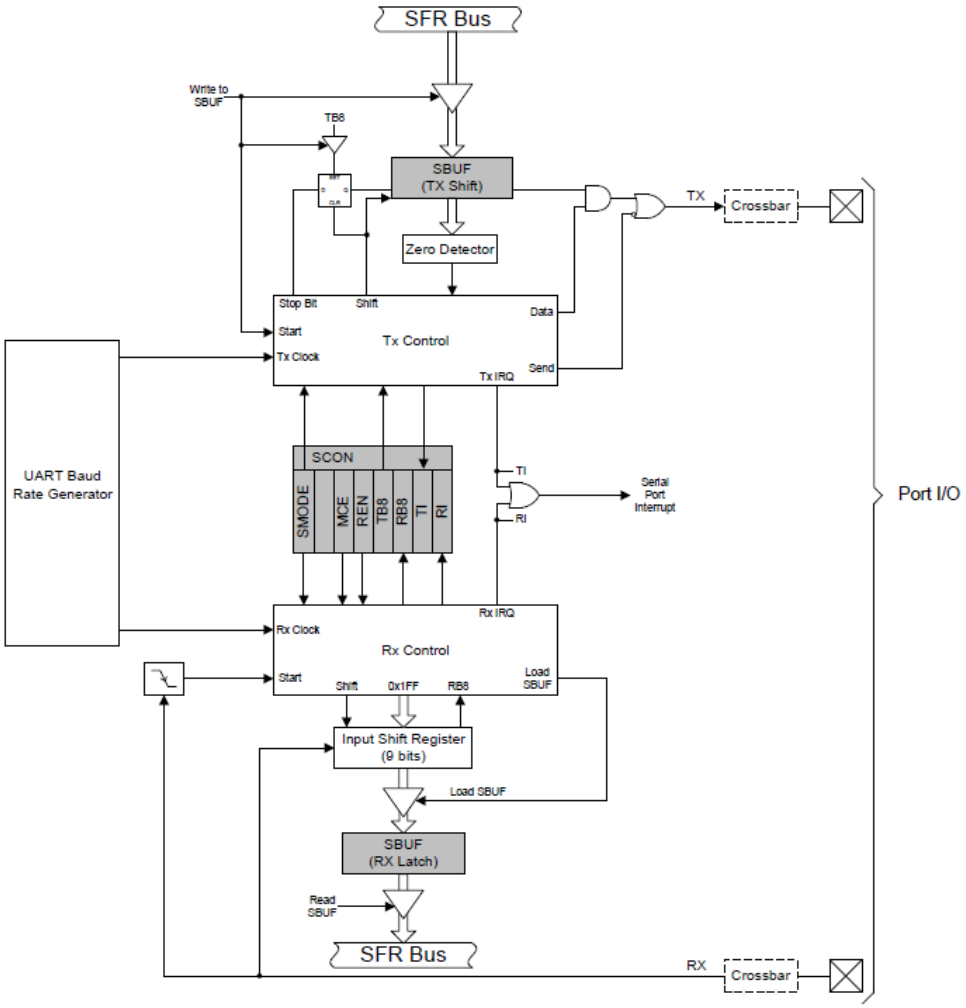
11.3. ábra. A top oldal huzalozása



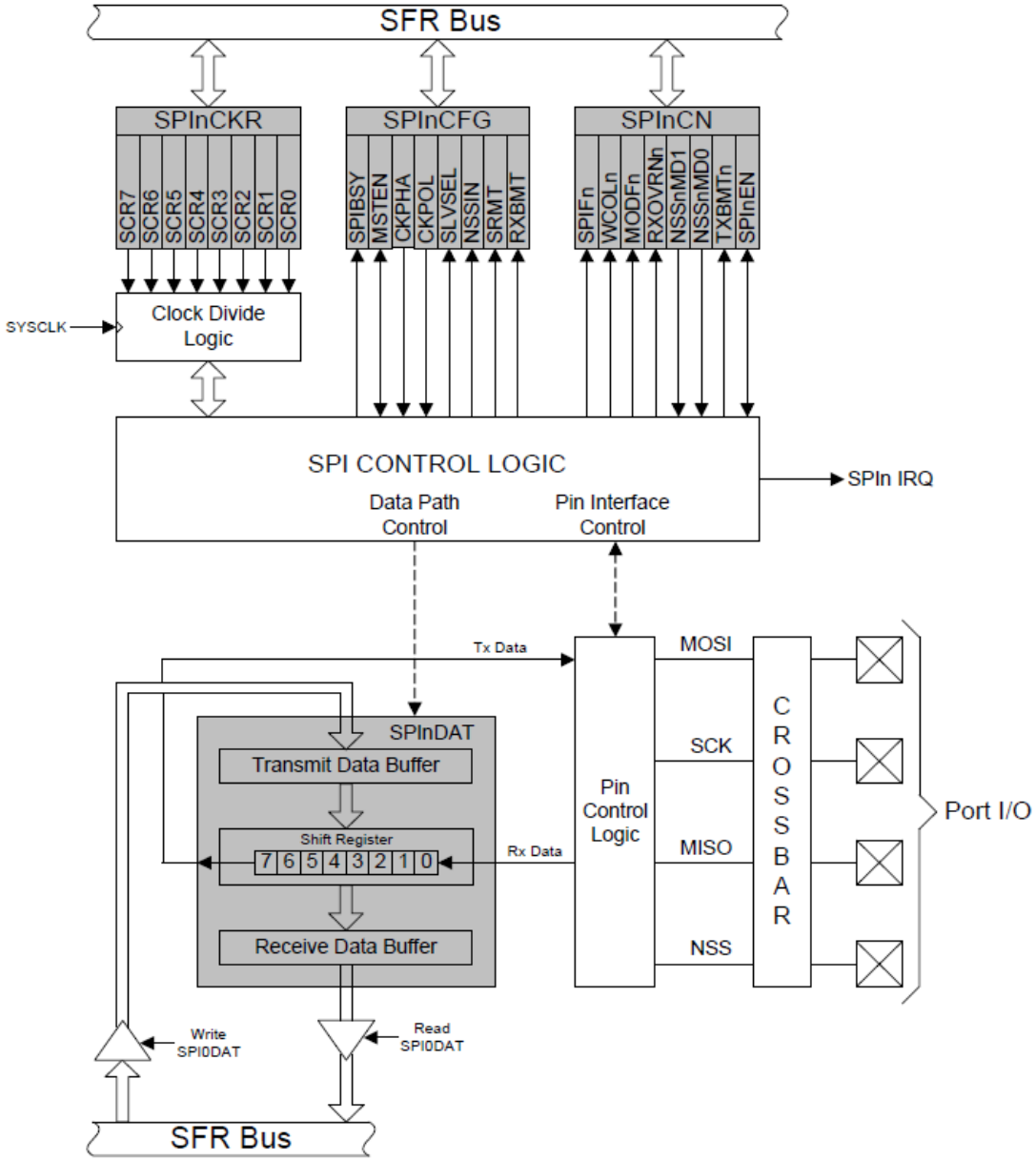
11.4. ábra. A bot oldal huzalozása



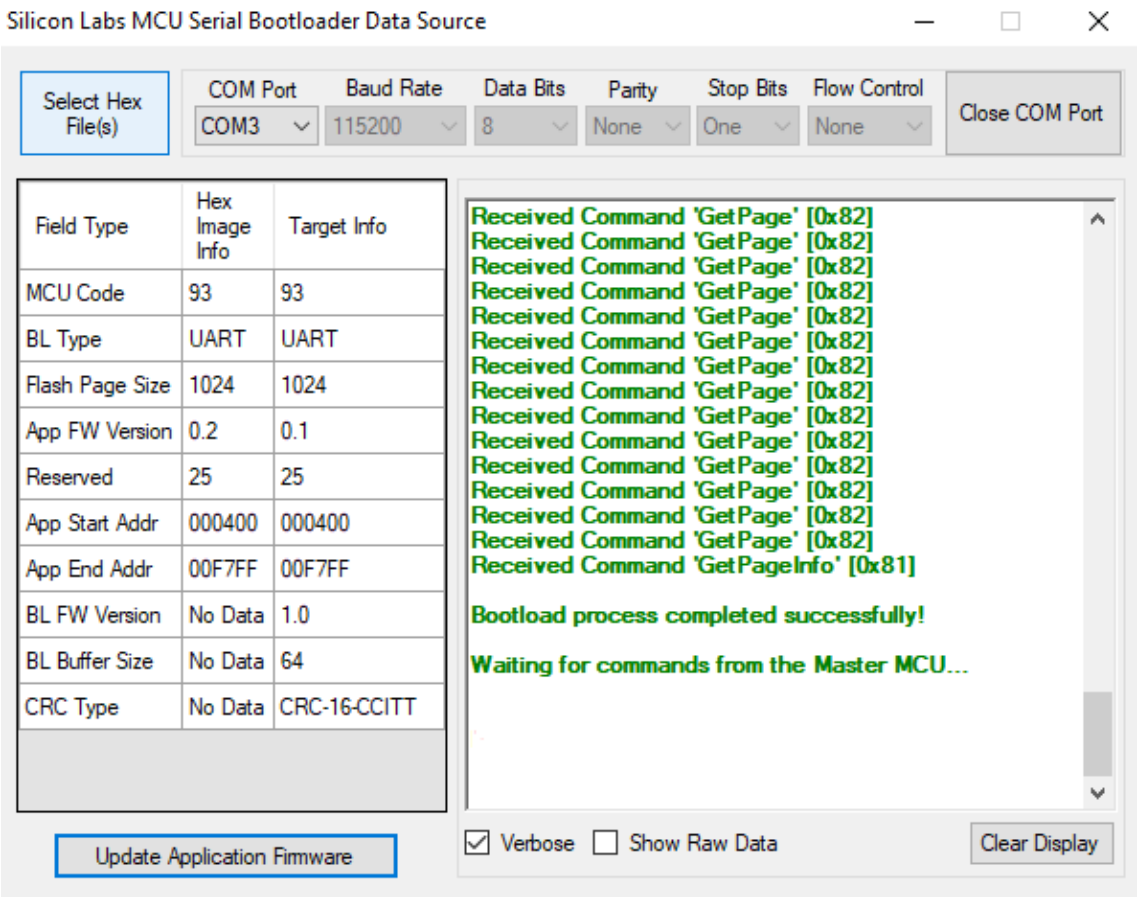
11.5. ábra. A kapcsolási rajz



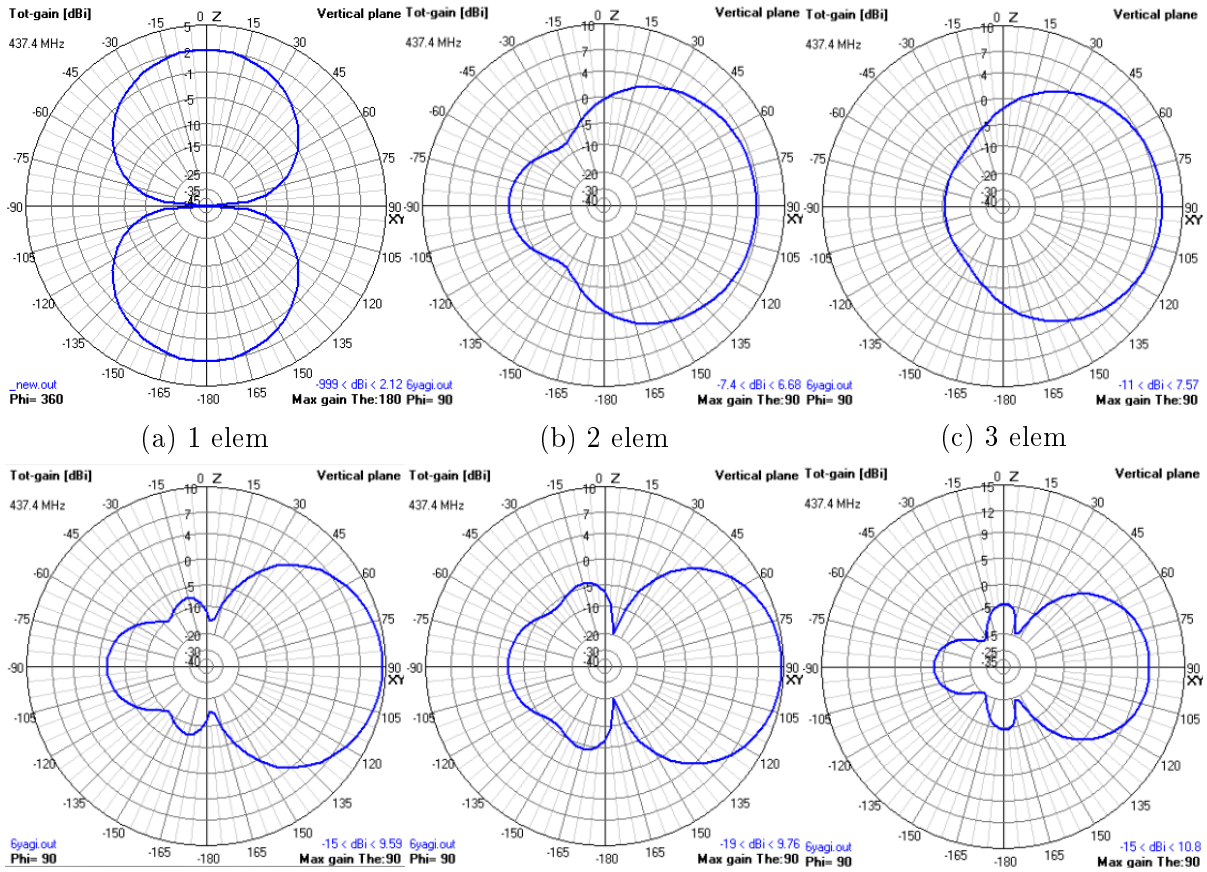
11.6. ábra. Az UART0 blokkdiagramja[4]



11.7. ábra. Az SPI blokkdiagramja[4]



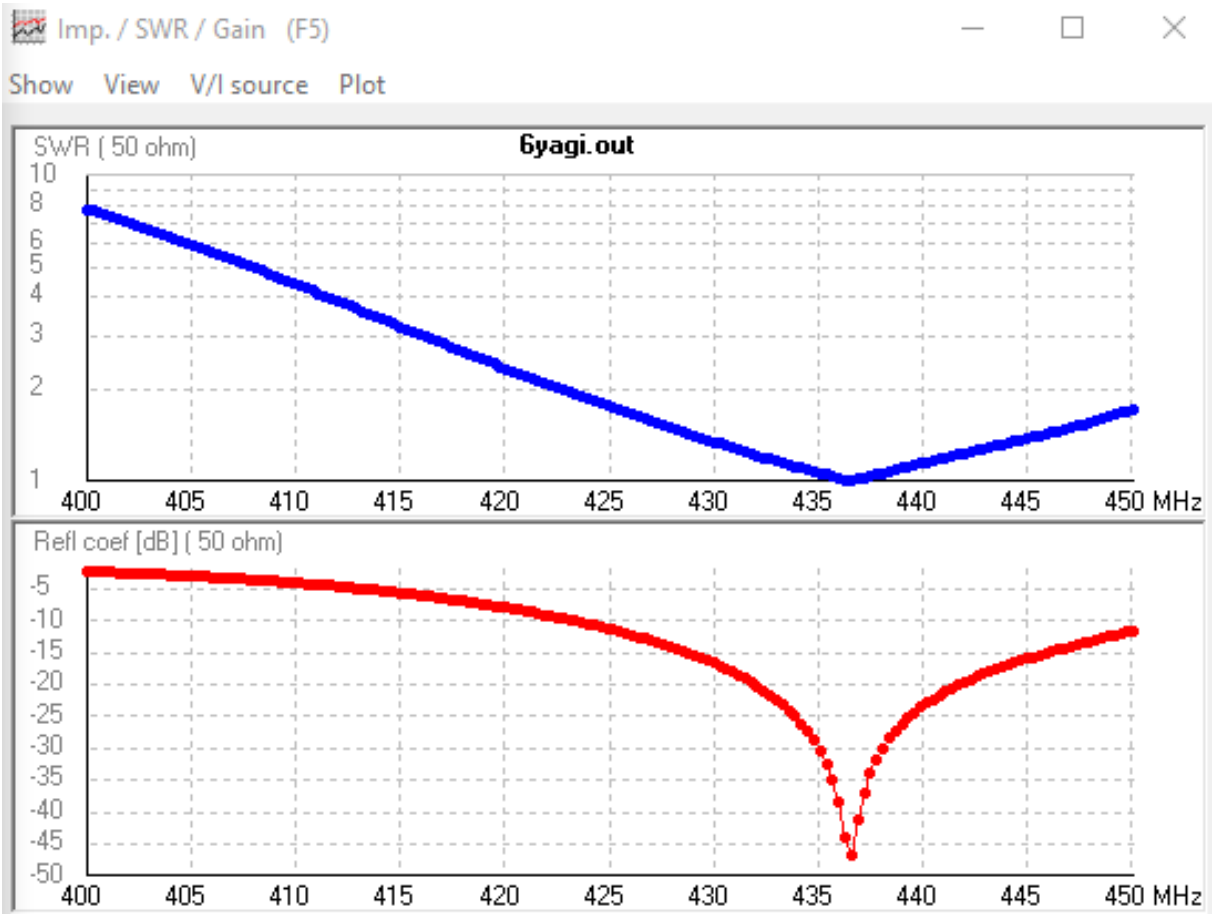
11.8. ábra. A bootload folyamat befejeződött



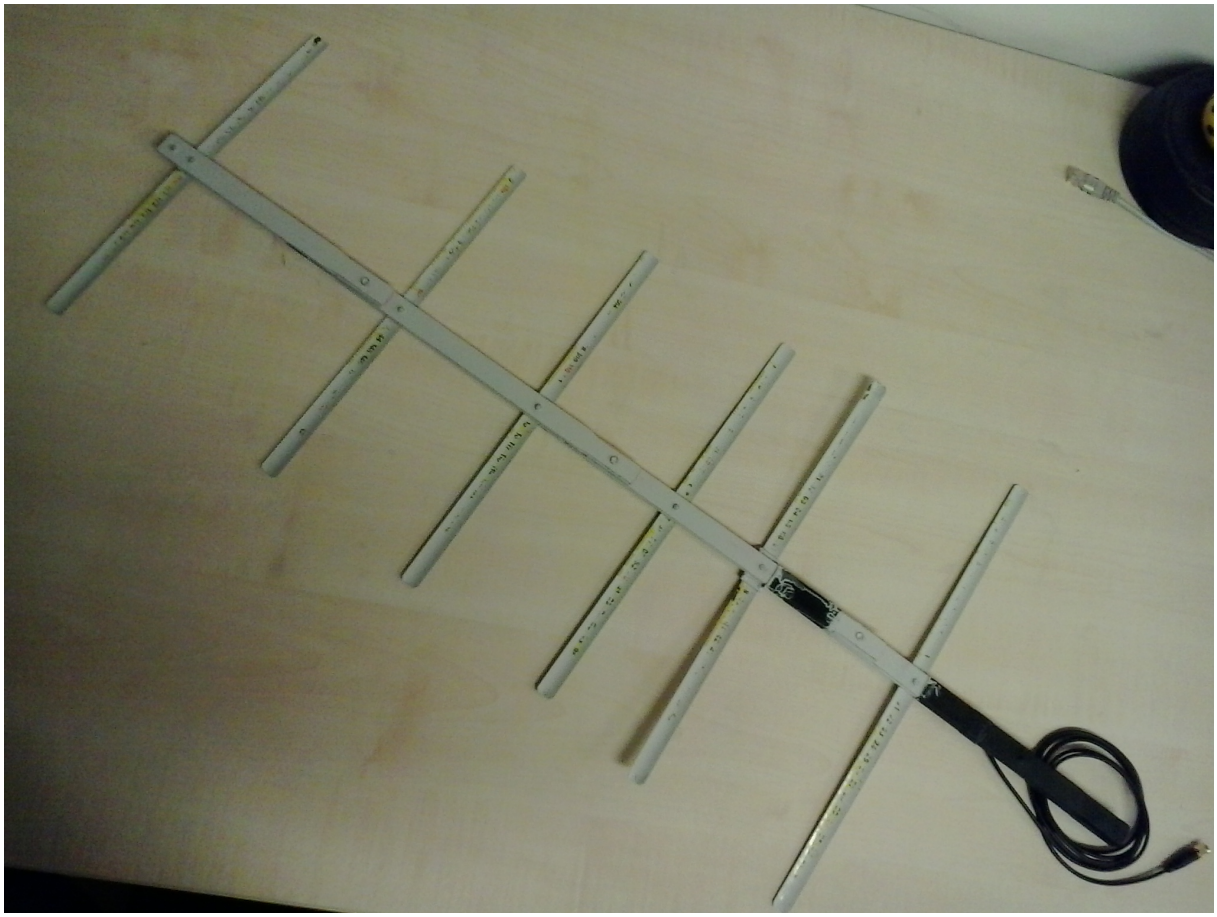
(a) 1 elem (b) 2 elem (c) 3 elem

(d) 4 elem (e) 5 elem (f) 6 elem

11.9. ábra. A kapott iránykarakterisztika különböző elemszámok esetén



11.10. ábra. Az állóhullámarány és a reflexiós tényező



11.11. ábra. Az elkészült Yagi antenna

11.12. ábra. A terepi mérés távolsága

12. fejezet

Forráskódok

A felhasználói program

```
//-------------------------------------------------------------------------
// F85x_Blinky.c
//-------------------------------------------------------------------------
// Copyright (C) 2014 Silicon Laboratories, Inc.
//
// AUTH: HF
// DATE: 04 FEB 2003
//
// This program flashes the green LED on the C8051F85x target board about
// five times a second using the interrupt handler for Timer2.
//
// Target: C8051F85x/86x
//
// Tool chain: KEIL C51
// INTVECTOR(0x000) INTERVAL(3)
//
// Release 1.0 / 22Sept2014 (SHY)
//    -Ported from F330
//-------------------------------------------------------------------------
// Includes
//-------------------------------------------------------------------------
#include <compiler_defs.h>
//#include <C8051F930_defs.h>
#include <SI_C8051F930_Defs.h>
#include "Radio_defs.h"
#include "radio_config_Si1060_5000.h"

//-------------------------------------------------------------------------
// Global CONSTANTS
//-------------------------------------------------------------------------

#define SYSCLK          24500000/8              // SYSCLK frequency in Hz

sbit RED            =     P1^4;
sbit GREEN          =     P1^5;
```

```
sbit BLUE      =     P1^6;                                  // LED='0' means ON
sbit BL        =     P0^2;
sbit TAP       =     P0^3;

sbit SS          = P1^3;
sbit SDN         = P0^7;
sbit NIRQ        = P0^1;

#define TGT_CMD_ENTER_BL_MODE        0x05
#define RESERVED_SPACE_ADDR          0xFFF0

#define UART_BUFFER_SIZE 8

U8 SERIAL_REPLY_DONE;
U8 SERIAL_COMMAND_CAME;
U8 Iterator;



U8 UART_RI_BUFFER[UART_BUFFER_SIZE];
U8 UART_RI_BUFFER_COUNTER;

U8 UART_TI_BUFFER[UART_BUFFER_SIZE];
U8 UART_TI_BUFFER_COUNTER;
U8 UART_TI_BUFFER_TI_SIZE;

U8 SERIAL_COMMAND[UART_BUFFER_SIZE];
U8 SERIAL_COMMAND_SIZE;

U8 SERIAL_REPLY[UART_BUFFER_SIZE];
U8 SERIAL_REPLY_SIZE;

xdata U8 FIFO[64];
U8 FIFO_SIZE;
U8 RSSI;

U8 Packet_Came;
U8 random;

//-----------------------------------------------------------------------------
// delay()
// ----------------------------------------------------------------------------
void delay ()
{
        U32 i;
        i=5000;
        while(i)
                i=i-1;
        return;
```

```
}

//-----------------------------------------------------------------------------
// Enter_BL_Mode
//-----------------------------------------------------------------------------
//
// Return Value : None
// Parameters   : None
//
// This function will cause a Flash Error Reset to enter BL mode.
//
//-----------------------------------------------------------------------------
U8 Enter_BL_Mode()
{
    return *(U8 code*)(RESERVED_SPACE_ADDR);
}

//-----------------------------------------------------------------------------
// Send_Back_Received_UART()
// ----------------------------------------------------------------------------
void Send_Back_Received_UART()
{
        SERIAL_COMMAND[SERIAL_COMMAND_SIZE] = '\n';
        SERIAL_COMMAND_SIZE++;

        SERIAL_REPLY_SIZE = SERIAL_COMMAND_SIZE;
        for(Iterator = 0; Iterator < SERIAL_COMMAND_SIZE; Iterator++)
        {
                SERIAL_REPLY[Iterator] = SERIAL_COMMAND[Iterator];
        }
        SCON0_TI = 1;
}

//-----------------------------------------------------------------------------
// Send_Byte_to_UART()
// ----------------------------------------------------------------------------
void Send_Byte_to_UART(U8 Byte)
{
        while(!SERIAL_REPLY_DONE);
        SERIAL_REPLY[0] = Byte;
        SERIAL_REPLY_SIZE = 1;
        SCON0_TI = 1;
}

//-----------------------------------------------------------------------------
// Bits_to_Hexa()
// ----------------------------------------------------------------------------
U8 Bits_to_Hexa(U8 Bits)
{
```

```
        if(Bits <= 9)
                Bits += '0';
        else
                Bits += ('A'-10);
        return Bits;
}

//-------------------------------------------------------------------------
// Send_Hexa_to_UART()
// ------------------------------------------------------------------------
void Send_Hexa_to_UART(U8 ascii)
{
        Send_Byte_to_UART(Bits_to_Hexa((ascii & 0xf0) >> 4 ));
        Send_Byte_to_UART(Bits_to_Hexa(ascii & 0x0f));
        return;
}

//-------------------------------------------------------------------------
// Send_Data_to_SPI()
// ------------------------------------------------------------------------
U8 Send_Data_to_SPI(U8 Data)
{
        SPI1DAT = Data;
        //while(!SPI_Done);
        while(!SPI1CN_SPIF);
        SPI1CN_SPIF = 0;
        //SPI_Done = 0;
        return SPI1DAT;
}

//-------------------------------------------------------------------------
// SPI_ClearToSend_Check()
// ------------------------------------------------------------------------
void SPI_ClearToSend_Check(U8 NSEL)
{
        SS = 0;
        delay();
        Send_Data_to_SPI(READ_CMD_BUFF);
        while(0xff != Send_Data_to_SPI(NOPE))
        {
                SS = 1;
                delay();
                SS = 0;
                delay();
                Send_Data_to_SPI(READ_CMD_BUFF);
        }
        if(NSEL)
                SS = 1;
        return;
```

```
}

//-------------------------------------------------------------------------
// Current_State()
// ------------------------------------------------------------------------
void Current_State()
{
        SS = 0;
        Send_Data_to_SPI(0x33);
        SS = 1;
        SPI_ClearToSend_Check(0);
        Send_Byte_to_UART('S');
        Send_Byte_to_UART(Send_Data_to_SPI(NOPE)+'0');
        Send_Byte_to_UART(Send_Data_to_SPI(NOPE)+'0');
        SS = 1;
        return;
}

//-------------------------------------------------------------------------
// Change_State()
// ------------------------------------------------------------------------
void Change_State(U8 State)
{
        //error handling
        SS = 0;
        Send_Data_to_SPI(0x34);
        Send_Data_to_SPI(State);
        SS = 1;
        SPI_ClearToSend_Check(1);
        return;
}

//-------------------------------------------------------------------------
// Part_Info()
// ------------------------------------------------------------------------
void Part_Info()
{
        SPI_ClearToSend_Check(1);

        SS = 0;
        Send_Byte_to_UART('P');
        Send_Data_to_SPI(PART_INFO);
        SS = 1;
        SPI_ClearToSend_Check(0);
        Send_Hexa_to_UART(Send_Data_to_SPI(NOPE));
        Send_Hexa_to_UART(Send_Data_to_SPI(NOPE));
        Send_Hexa_to_UART(Send_Data_to_SPI(NOPE));
        Send_Byte_to_UART('P');
        SS = 1;
```

```
}

//-----------------------------------------------------------------------------
// Radio_Config()
//-----------------------------------------------------------------------------
void Radio_Config()
{
        code U8 Config[] = RADIO_CONFIGURATION_DATA_ARRAY;
        U8 Iterator_Config;
        U8 Iterator_Command;

        for(Iterator_Config = 0; Config[Iterator_Config] != 0; )
        {
                SS = 0;
                for(Iterator_Command = Config[Iterator_Config], Iterator_Co
                {
                        Send_Data_to_SPI(Config[Iterator_Config]);
                }
                SS = 1;
                SPI_ClearToSend_Check(1);
        }
        return;
}

//-----------------------------------------------------------------------------
// Radio_Power_Up()
//-----------------------------------------------------------------------------
void Radio_Power_Up()
{
        SDN = 1;
        delay();
        SDN = 0;
        delay();
        SS = 0;
        random = SS;
        Send_Byte_to_UART('Y');
        Send_Byte_to_UART(random+'0');
        Send_Byte_to_UART('\r');
        Send_Byte_to_UART('\n');

        Radio_Config();
        SPI_ClearToSend_Check(1);
        return;
}

//-----------------------------------------------------------------------------
// Init()
//-----------------------------------------------------------------------------
void Init(void)
```

```
{
        PCA0MD &= ~0x40;
        OSCICN = 0x8F;
        while(!(OSCICN & 0x40));
        CLKSEL = 0;
        P0MDOUT    = 0x98;   // P0.4 push-pull(UART TX ), P0.5 open-drain(UAR
        P1MDOUT    = 0x7d;
        XBR0 = 0x01; // Enable UART0
        XBR1 = 0x40; // Enable SPI
        XBR2 = 0x40; // Enable crossbar
        TH1 = 0x96; //Baudrate 115200
        TL1 = 0x96;
        CKCON = 0x08; //T1M SYSCLK
        TMOD = 0x20; // T1M MODE2
        TCON |= 0x45;
        SCON0 |= 0x10;
        RSTSRC = 0x06;
        IE = 0x95;
        IP = 0x05;
        //EIE2 = 0x08;
        //EIP2 = 0x08;
        IT01CF = 0x12; //EXT INT //0x10
        SPI1CKR = 0x08;
        SPI1CFG |= 0x40;
        SPI1CN = 0x01;

        SERIAL_REPLY_DONE = 1;
        SERIAL_COMMAND_CAME = 0;

        UART_RI_BUFFER_COUNTER = 0;
        UART_TI_BUFFER_COUNTER = 0;
        UART_TI_BUFFER_TI_SIZE = 0;

        SERIAL_COMMAND_SIZE = 0;
        SERIAL_REPLY_SIZE = 0;

        FIFO_SIZE = 0;

        RED = 0;
        GREEN = 0;
        BLUE = 0;

        TAP = 1;

        SS = 1;//alacsony aktiv
        SDN = 1;//alacsony aktiv
        Packet_Came = 0;

        IE &= 0x7f;
```

```
        Radio_Power_Up();
        IE |= 0x80;

        SPI_ClearToSend_Check(1);
        SS = 0;
        Send_Data_to_SPI(GET_INT_STATUS);
        SS = 1;
        RED = 0;
}

//-------------------------------------------------------------------------
// MAIN Routine
//-------------------------------------------------------------------------
void main(void)
{

        Init();

        SPI_ClearToSend_Check(1);
        SS = 0;
        Send_Data_to_SPI(START_RX);
        Send_Data_to_SPI(0);
        Send_Data_to_SPI(0);
        Send_Data_to_SPI(0);
        Send_Data_to_SPI(64);
        Send_Data_to_SPI(0);
        Send_Data_to_SPI(0);
        Send_Data_to_SPI(0);
        SS = 1;

    while (1) {
          BLUE = 1;

          // spin forever


          if(SERIAL_COMMAND_CAME && SERIAL_REPLY_DONE)
          {
                SERIAL_COMMAND_CAME = 0;
                Send_Byte_to_UART('X');
                Send_Back_Received_UART();

                switch(SERIAL_COMMAND[0])
                {
                      case '\r':
                            break;
                      case 'b': RED = 1;
                            break;
                      case 'n': RED = 0;
```

```
                        break;
                case 'R': SPI_ClearToSend_Check(1);
                                SS = 0;
                                Send_Data_to_SPI(START_RX)
                                Send_Data_to_SPI(0);
                                Send_Data_to_SPI(0);
                                Send_Data_to_SPI(0);
                                Send_Data_to_SPI(64);
                                Send_Data_to_SPI(0);
                                Send_Data_to_SPI(0);
                                Send_Data_to_SPI(0);
                                SS = 1;
                        break;
                case 'c': Current_State();
                        break;
                case 'p': Part_Info();
                        break;
                case 'a': Change_State(0x03);
                        break;
                case 'x': SS = 0;
                        Send_Data_to_SPI(PART_INFO);
                        SS = 1;
                        SS = 0;
                        Send_Data_to_SPI(0x01);
                        Send_Data_to_SPI(0x01);
                        SS = 1;
                        Send_Byte_to_UART('X');
                        break;
                case 't': SS = 0;
                        Send_Data_to_SPI(GET_INT_STATUS);
                        SS = 1;
                        RED = 0;
                        break;
                default :
                        break;
                }
        }

        if(Packet_Came)
        {
                U8 Iterator;
                Packet_Came = 0;
                for(Iterator = 0; Iterator < 64; Iterator++)
                {
                        Send_Hexa_to_UART(FIFO[Iterator]);
                }
                Send_Hexa_to_UART(RSSI);
                Send_Byte_to_UART('\n');
                Send_Byte_to_UART('\r');
```

```
            }
    }
}
```

A megszakítás kezelések

```
//-----------------------------------------------------------------------
// src/Interrupts.c: generated by Hardware Configurator
//
// This file will be regenerated when saving a document.
// leave the sections inside the "$[...]" comment tags alone
// or they will be overwritten!
//-----------------------------------------------------------------------


// USER INCLUDES
//#include <SI_C8051F930_Register_Enums.h>
#include <compiler_defs.h>
#include "Radio_defs.h"
//#include <C8051F930_defs.h>
#include <SI_C8051F930_Defs.h>

#define UART_BUFFER_SIZE 8

sbit SS          = P1^3;
sbit GREEN         =    P1^5;
sbit RED = P1^4;

extern U8 Send_Data_to_SPI(U8 Data);
extern U8 Enter_BL_Mode();
extern void SPI_ClearToSend_Check(U8 NSEL);

extern U8 SERIAL_REPLY_DONE;
extern U8 SERIAL_COMMAND_CAME;

extern U8 UART_RI_BUFFER[UART_BUFFER_SIZE];
extern U8 UART_RI_BUFFER_COUNTER;

extern U8 UART_TI_BUFFER[UART_BUFFER_SIZE];
extern U8 UART_TI_BUFFER_COUNTER;
extern U8 UART_TI_BUFFER_TI_SIZE;

extern U8 SERIAL_COMMAND[UART_BUFFER_SIZE];
extern U8 SERIAL_COMMAND_SIZE;

extern U8 SERIAL_REPLY[UART_BUFFER_SIZE];
extern U8 SERIAL_REPLY_SIZE;

extern xdata U8 FIFO[64];
```

```
extern U8 FIFO_SIZE;
extern U8 RSSI;

extern U8 Packet_Came;

//-----------------------------------------------------------------------------
// UART0_ISR
//-----------------------------------------------------------------------------
//
// UART0 ISR Content goes here. Remember to clear flag bits:
// SCON0::RI (Receive Interrupt Flag)
// SCON0::TI (Transmit Interrupt Flag)
//
//-----------------------------------------------------------------------------
INTERRUPT (UART0_ISR, UART0_IRQn )
{
        U8 Iterator;
        if(SCON0_RI)
        {
                SCON0_RI = 0;
            if(UART_RI_BUFFER_COUNTER < UART_BUFFER_SIZE-1)
            {
                UART_RI_BUFFER[UART_RI_BUFFER_COUNTER] = SBUF0;

                UART_RI_BUFFER_COUNTER++;
            }

            if((UART_RI_BUFFER[UART_RI_BUFFER_COUNTER-1] == '\r') || (UART_
            {
                SERIAL_COMMAND_SIZE = UART_RI_BUFFER_COUNTER;
                for(Iterator = 0; Iterator < UART_RI_BUFFER_COUNTER; Iterat
                {
                        SERIAL_COMMAND[Iterator] = UART_RI_BUFFER[Iterator]
                }
                UART_RI_BUFFER_COUNTER = 0;
                SERIAL_COMMAND_CAME = 1;
            }
        }

        if(SCON0_TI)
        {
                SCON0_TI = 0;

                if(SERIAL_REPLY_DONE == 1)
                {
                        SERIAL_REPLY_DONE = 0;
                        UART_TI_BUFFER_TI_SIZE = SERIAL_REPLY_SIZE;
                        for(Iterator = 0; Iterator < SERIAL_REPLY_SIZE; Ite
                        {
```

```
                                        UART_TI_BUFFER[Iterator] = SERIAL_REPLY[Ite
                                }
                                UART_TI_BUFFER_COUNTER = 0;
                        }
                if(UART_TI_BUFFER_COUNTER < UART_TI_BUFFER_TI_SIZE)
                {
                        SBUF0 = UART_TI_BUFFER[UART_TI_BUFFER_COUNTER];

                        UART_TI_BUFFER_COUNTER++;

                }
                else
                {
                        SERIAL_REPLY_DONE = 1;
                }
        }
}


//-----------------------------------------------------------------------------
// SPI1_ISR
//-----------------------------------------------------------------------------
//
// SPI1 ISR Content goes here. Remember to clear flag bits:
// SPI1CN::MODF (Mode Fault Flag)
// SPI1CN::RXOVRN (Receive Overrun Flag)
// SPI1CN::SPIF (SPI# Interrupt Flag)
// SPI1CN::WCOL (Write Collision Flag)
//
//-----------------------------------------------------------------------------

INTERRUPT (INT0_ISR, INT0_IRQn)
{
        Enter_BL_Mode();
}

INTERRUPT (INT1_ISR, INT1_IRQn)
{
        U8 Interrupt;
        U8 PacketHandler;
        U8 Modem;
        U8 Chip;

        SPI_ClearToSend_Check(1);
        SS = 0;
        Send_Data_to_SPI(GET_INT_STATUS);
        SS = 1;

        SPI_ClearToSend_Check(0);
```

```
        Interrupt = Send_Data_to_SPI(NOPE);
        Send_Data_to_SPI(NOPE);
        PacketHandler = Send_Data_to_SPI(NOPE);
        Send_Data_to_SPI(NOPE);
        Modem = Send_Data_to_SPI(NOPE);
        Send_Data_to_SPI(NOPE);
        Chip = Send_Data_to_SPI(NOPE);
        Send_Data_to_SPI(NOPE);
        SS = 1;
        SPI_ClearToSend_Check(1);

        if(Interrupt != 0)
        {
                if((PacketHandler & 0x10) == 0x10)//packet RX
                {
                        U8 Iterator;
*/
                        SS = 0;
                        Send_Data_to_SPI(READ_RX_FIFO);
                        for(Iterator = 0; Iterator < 64; Iterator++)
                        {
                                FIFO[Iterator] = Send_Data_to_SPI(NOPE);
                        }
                        SS = 1;

                        Packet_Came = 1;

                        SPI_ClearToSend_Check(1);
                        SS = 0;
                        Send_Data_to_SPI(START_RX);
                         Send_Data_to_SPI(0);









                        SS = 1;
                        GREEN = 1;
                }
                if((PacketHandler & 0x08) == 0x08)//CRC Error
                {

                }
                if((Modem & 0x01) == 0x01)//Sync Detect
                {

                }
                if((Chip & 0x08) == 0x08)//CMD Error
```

```
                {

                }
        }

}
```

A rádió konfigurációs header

```
/*! @file radio_config.h
 * @brief This file contains the automatically generated
 * configurations.
 *
 * @n WDS GUI Version: 3.2.11.0
 * @n Device: Si1062 Rev.: A0
 *
 * @b COPYRIGHT
 * @n Silicon Laboratories Confidential
 * @n Copyright 2017 Silicon Laboratories, Inc.
 * @n http://www.silabs.com
 */

#ifndef RADIO_CONFIG_H_
#define RADIO_CONFIG_H_

// USER DEFINED PARAMETERS
// Define your own parameters here

// INPUT DATA
/*
// Crys_freq(Hz): 30000000    Crys_tol(ppm): 0    IF_mode: 2    High_perf_C
OSRtune: 0    Ch_Fil_Bw_AFC: 0    ANT_DIV: 0    PM_pattern: 0
// MOD_type: 3    Rsymb(sps): 5000    Fdev(Hz): 1250    RXBW(Hz): 150000
Manchester: 0    AFC_en: 1    Rsymb_error: 0.0    Chip-Version: 2
// RF Freq.(MHz): 437.345    API_TC: 29    fhst: 250000    inputBW: 0
BERT: 0    RAW_dout: 0    D_source: 0    Hi_pfm_div: 1
//
// # RX IF frequency is  -468750 Hz
// # WB filter 7 (BW =  7.56 kHz);  NB-filter 7 (BW = 7.56 kHz)
//
// Modulation index: 0,5
*/


// CONFIGURATION PARAMETERS
#define RADIO_CONFIGURATION_DATA_RADIO_XO_FREQ
30000000L
#define RADIO_CONFIGURATION_DATA_CHANNEL_NUMBER
0x00
#define RADIO_CONFIGURATION_DATA_RADIO_PACKET_LENGTH
0x40
```

```
#define RADIO_CONFIGURATION_DATA_RADIO_STATE_AFTER_POWER_UP
0x03
#define RADIO_CONFIGURATION_DATA_RADIO_DELAY_CNT_AFTER_RESET
0xF000
#define RADIO_CONFIGURATION_DATA_CUSTOM_PAYLOAD
{0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0x
0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC
0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC5, 0xC
0xC5, 0xC5, 0xC5, 0xC5}


// CONFIGURATION COMMANDS

/*
// Command:                    RF_POWER_UP
// Description:                Command to power-up the device and select the
*/
#define RF_POWER_UP 0x02, 0x01, 0x01, 0x01, 0xC9, 0xC3, 0x80

/*
// Command:                    RF_GPIO_PIN_CFG
// Description:                Configures the GPIO pins.
*/
#define RF_GPIO_PIN_CFG 0x13, 0x08, 0x21, 0x22, 0x18, 0x00, 0x00, 0x00

/*
// Set properties:             RF_GLOBAL_XO_TUNE_2
// Number of properties:       2
// Group ID:                   0x00
// Start ID:                   0x00
// Default values:             0x40, 0x00,
// Descriptions:
//   GLOBAL_XO_TUNE - Configure the internal capacitor frequency tuning bar
//   GLOBAL_CLK_CFG - Clock configuration options.
*/
#define RF_GLOBAL_XO_TUNE_2 0x11, 0x00, 0x02, 0x00, 0x00, 0x00

/*
// Set properties:             RF_GLOBAL_CONFIG_1
// Number of properties:       1
// Group ID:                   0x00
// Start ID:                   0x03
// Default values:             0x20,
// Descriptions:
//   GLOBAL_CONFIG - Global configuration settings.
*/
#define RF_GLOBAL_CONFIG_1 0x11, 0x00, 0x01, 0x03, 0x60

/*
```

```
// Set properties:            RF_INT_CTL_ENABLE_4
// Number of properties:      4
// Group ID:                  0x01
// Start ID:                  0x00
// Default values:            0x04, 0x00, 0x00, 0x04,
// Descriptions:
//   INT_CTL_ENABLE - This property provides for global enabling of the thr
//   INT_CTL_PH_ENABLE - Enable individual interrupt sources within the Pa
//   INT_CTL_MODEM_ENABLE - Enable individual interrupt sources within the
//   INT_CTL_CHIP_ENABLE - Enable individual interrupt sources within the C
*/
#define RF_INT_CTL_ENABLE_4 0x11, 0x01, 0x04, 0x00, 0x07, 0x18, 0x01, 0x08

/*
// Set properties:            RF_FRR_CTL_A_MODE_4
// Number of properties:      4
// Group ID:                  0x02
// Start ID:                  0x00
// Default values:            0x01, 0x02, 0x09, 0x00,
// Descriptions:
//   FRR_CTL_A_MODE - Fast Response Register A Configuration.
//   FRR_CTL_B_MODE - Fast Response Register B Configuration.
//   FRR_CTL_C_MODE - Fast Response Register C Configuration.
//   FRR_CTL_D_MODE - Fast Response Register D Configuration.
*/
#define RF_FRR_CTL_A_MODE_4 0x11, 0x02, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00

/*
// Set properties:            RF_PREAMBLE_TX_LENGTH_9
// Number of properties:      9
// Group ID:                  0x10
// Start ID:                  0x00
// Default values:            0x08, 0x14, 0x00, 0x0F, 0x21, 0x00, 0x00, 0x00
// Descriptions:
//   PREAMBLE_TX_LENGTH - Configure length of TX Preamble.
//   PREAMBLE_CONFIG_STD_1 - Configuration of reception of a packet with a
//   PREAMBLE_CONFIG_NSTD - Configuration of transmission/reception of a pa
//   PREAMBLE_CONFIG_STD_2 - Configuration of timeout periods during recept
//   PREAMBLE_CONFIG - General configuration bits for the Preamble field.
//   PREAMBLE_PATTERN_31_24 - Configuration of the bit values describing a
//   PREAMBLE_PATTERN_23_16 - Configuration of the bit values describing a
//   PREAMBLE_PATTERN_15_8 - Configuration of the bit values describing a N
//   PREAMBLE_PATTERN_7_0 - Configuration of the bit values describing a Na
*/
#define RF_PREAMBLE_TX_LENGTH_9 0x11, 0x10, 0x09, 0x00, 0x08, 0x14, 0x00, 0

/*
// Set properties:            RF_SYNC_CONFIG_5
// Number of properties:      5
```

```
// Group ID:                    0x11
// Start ID:                    0x00
// Default values:              0x01, 0x2D, 0xD4, 0x2D, 0xD4,
// Descriptions:
//   SYNC_CONFIG - Sync Word configuration bits.
//   SYNC_BITS_31_24 - Sync word.
//   SYNC_BITS_23_16 - Sync word.
//   SYNC_BITS_15_8 - Sync word.
//   SYNC_BITS_7_0 - Sync word.
*/
#define RF_SYNC_CONFIG_5 0x11, 0x11, 0x05, 0x00, 0x01, 0xB4, 0x2B, 0x00, 0x

/*
// Set properties:              RF_PKT_CRC_CONFIG_7
// Number of properties:        7
// Group ID:                    0x12
// Start ID:                    0x00
// Default values:              0x00, 0x01, 0x08, 0xFF, 0xFF, 0x00, 0x00,
// Descriptions:
//   PKT_CRC_CONFIG - Select a CRC polynomial and seed.
//   PKT_WHT_POLY_15_8 - 16-bit polynomial value for the PN Generator (e.g.
//   PKT_WHT_POLY_7_0 - 16-bit polynomial value for the PN Generator (e.g.,
//   PKT_WHT_SEED_15_8 - 16-bit seed value for the PN Generator (e.g., for
//   PKT_WHT_SEED_7_0 - 16-bit seed value for the PN Generator (e.g., for 1
//   PKT_WHT_BIT_NUM - Selects which bit of the LFSR (used to generate the
//   PKT_CONFIG1 - General configuration bits for transmission or reception
*/
#define RF_PKT_CRC_CONFIG_7 0x11, 0x12, 0x07, 0x00, 0x84, 0x01, 0x08, 0xFF,

/*
// Set properties:              RF_PKT_LEN_12
// Number of properties:        12
// Group ID:                    0x12
// Start ID:                    0x08
// Default values:              0x00, 0x00, 0x00, 0x30, 0x30, 0x00, 0x00, 0x00
// Descriptions:
//   PKT_LEN - Configuration bits for reception of a variable length packet
//   PKT_LEN_FIELD_SOURCE - Field number containing the received packet len
//   PKT_LEN_ADJUST - Provides for adjustment/offset of the received packet
//   PKT_TX_THRESHOLD - TX FIFO almost empty threshold.
//   PKT_RX_THRESHOLD - RX FIFO Almost Full threshold.
//   PKT_FIELD_1_LENGTH_12_8 - Unsigned 13-bit Field 1 length value.
//   PKT_FIELD_1_LENGTH_7_0 - Unsigned 13-bit Field 1 length value.
//   PKT_FIELD_1_CONFIG - General data processing and packet configuration
//   PKT_FIELD_1_CRC_CONFIG - Configuration of CRC control bits across Fiel
//   PKT_FIELD_2_LENGTH_12_8 - Unsigned 13-bit Field 2 length value.
//   PKT_FIELD_2_LENGTH_7_0 - Unsigned 13-bit Field 2 length value.
//   PKT_FIELD_2_CONFIG - General data processing and packet configuration
*/
```

```
#define RF_PKT_LEN_12 0x11, 0x12, 0x0C, 0x08, 0x00, 0x00, 0x00, 0x30, 0x30,

/*
// Set properties:              RF_PKT_FIELD_2_CRC_CONFIG_12
// Number of properties:        12
// Group ID:                    0x12
// Start ID:                    0x14
// Default values:              0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
// Descriptions:
//   PKT_FIELD_2_CRC_CONFIG - Configuration of CRC control bits across Fiel
//   PKT_FIELD_3_LENGTH_12_8 - Unsigned 13-bit Field 3 length value.
//   PKT_FIELD_3_LENGTH_7_0 - Unsigned 13-bit Field 3 length value.
//   PKT_FIELD_3_CONFIG - General data processing and packet configuration
//   PKT_FIELD_3_CRC_CONFIG - Configuration of CRC control bits across Fiel
//   PKT_FIELD_4_LENGTH_12_8 - Unsigned 13-bit Field 4 length value.
//   PKT_FIELD_4_LENGTH_7_0 - Unsigned 13-bit Field 4 length value.
//   PKT_FIELD_4_CONFIG - General data processing and packet configuration
//   PKT_FIELD_4_CRC_CONFIG - Configuration of CRC control bits across Fiel
//   PKT_FIELD_5_LENGTH_12_8 - Unsigned 13-bit Field 5 length value.
//   PKT_FIELD_5_LENGTH_7_0 - Unsigned 13-bit Field 5 length value.
//   PKT_FIELD_5_CONFIG - General data processing and packet configuration
*/
#define RF_PKT_FIELD_2_CRC_CONFIG_12 0x11, 0x12, 0x0C, 0x14, 0x00, 0x00, 0x

/*
// Set properties:              RF_PKT_FIELD_5_CRC_CONFIG_12
// Number of properties:        12
// Group ID:                    0x12
// Start ID:                    0x20
// Default values:              0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
// Descriptions:
//   PKT_FIELD_5_CRC_CONFIG - Configuration of CRC control bits across Fiel
//   PKT_RX_FIELD_1_LENGTH_12_8 - Unsigned 13-bit RX Field 1 length value.
//   PKT_RX_FIELD_1_LENGTH_7_0 - Unsigned 13-bit RX Field 1 length value.
//   PKT_RX_FIELD_1_CONFIG - General data processing and packet configurati
//   PKT_RX_FIELD_1_CRC_CONFIG - Configuration of CRC control bits across R
//   PKT_RX_FIELD_2_LENGTH_12_8 - Unsigned 13-bit RX Field 2 length value.
//   PKT_RX_FIELD_2_LENGTH_7_0 - Unsigned 13-bit RX Field 2 length value.
//   PKT_RX_FIELD_2_CONFIG - General data processing and packet configurati
//   PKT_RX_FIELD_2_CRC_CONFIG - Configuration of CRC control bits across R
//   PKT_RX_FIELD_3_LENGTH_12_8 - Unsigned 13-bit RX Field 3 length value.
//   PKT_RX_FIELD_3_LENGTH_7_0 - Unsigned 13-bit RX Field 3 length value.
//   PKT_RX_FIELD_3_CONFIG - General data processing and packet configurati
*/
#define RF_PKT_FIELD_5_CRC_CONFIG_12 0x11, 0x12, 0x0C, 0x20, 0x00, 0x00, 0x

/*
// Set properties:              RF_PKT_RX_FIELD_3_CRC_CONFIG_9
// Number of properties:        9
```

```
// Group ID:                    0x12
// Start ID:                    0x2C
// Default values:              0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
// Descriptions:
//   PKT_RX_FIELD_3_CRC_CONFIG - Configuration of CRC control bits across R
//   PKT_RX_FIELD_4_LENGTH_12_8 - Unsigned 13-bit RX Field 4 length value.
//   PKT_RX_FIELD_4_LENGTH_7_0 - Unsigned 13-bit RX Field 4 length value.
//   PKT_RX_FIELD_4_CONFIG - General data processing and packet configurati
//   PKT_RX_FIELD_4_CRC_CONFIG - Configuration of CRC control bits across R
//   PKT_RX_FIELD_5_LENGTH_12_8 - Unsigned 13-bit RX Field 5 length value.
//   PKT_RX_FIELD_5_LENGTH_7_0 - Unsigned 13-bit RX Field 5 length value.
//   PKT_RX_FIELD_5_CONFIG - General data processing and packet configurati
//   PKT_RX_FIELD_5_CRC_CONFIG - Configuration of CRC control bits across R
*/
#define RF_PKT_RX_FIELD_3_CRC_CONFIG_9 0x11, 0x12, 0x09, 0x2C, 0x00, 0x00,

/*
// Set properties:              RF_MODEM_MOD_TYPE_12
// Number of properties:        12
// Group ID:                    0x20
// Start ID:                    0x00
// Default values:              0x02, 0x80, 0x07, 0x0F, 0x42, 0x40, 0x01, 0xC9
// Descriptions:
//   MODEM_MOD_TYPE - Selects the type of modulation. In TX mode, additione
//   MODEM_MAP_CONTROL - Controls polarity and mapping of transmit and rece
//   MODEM_DSM_CTRL - Miscellaneous control bits for the Delta-Sigma Modul
//   MODEM_DATA_RATE_2 - Unsigned 24-bit value used to determine the TX dat
//   MODEM_DATA_RATE_1 - Unsigned 24-bit value used to determine the TX dat
//   MODEM_DATA_RATE_0 - Unsigned 24-bit value used to determine the TX dat
//   MODEM_TX_NCO_MODE_3 - TX Gaussian filter oversampling ratio and Byte 3
//   MODEM_TX_NCO_MODE_2 - TX Gaussian filter oversampling ratio and Byte 2
//   MODEM_TX_NCO_MODE_1 - TX Gaussian filter oversampling ratio and Byte 1
//   MODEM_TX_NCO_MODE_0 - TX Gaussian filter oversampling ratio and Byte 0
//   MODEM_FREQ_DEV_2 - 17-bit unsigned TX frequency deviation word.
//   MODEM_FREQ_DEV_1 - 17-bit unsigned TX frequency deviation word.
*/
#define RF_MODEM_MOD_TYPE_12 0x11, 0x20, 0x0C, 0x00, 0x03, 0x00, 0x07, 0x03

/*
// Set properties:              RF_MODEM_FREQ_DEV_0_1
// Number of properties:        1
// Group ID:                    0x20
// Start ID:                    0x0C
// Default values:              0xD3,
// Descriptions:
//   MODEM_FREQ_DEV_0 - 17-bit unsigned TX frequency deviation word.
*/
#define RF_MODEM_FREQ_DEV_0_1 0x11, 0x20, 0x01, 0x0C, 0x57
```

```
/*
// Set properties:            RF_MODEM_TX_RAMP_DELAY_8
// Number of properties:      8
// Group ID:                  0x20
// Start ID:                  0x18
// Default values:            0x01, 0x00, 0x08, 0x03, 0xC0, 0x00, 0x10, 0x20
// Descriptions:
//   MODEM_TX_RAMP_DELAY - TX ramp-down delay setting.
//   MODEM_MDM_CTRL - MDM control.
//   MODEM_IF_CONTROL - Selects Fixed-IF, Scaled-IF, or Zero-IF mode of RX
//   MODEM_IF_FREQ_2 - the IF frequency setting (an 18-bit signed number).
//   MODEM_IF_FREQ_1 - the IF frequency setting (an 18-bit signed number).
//   MODEM_IF_FREQ_0 - the IF frequency setting (an 18-bit signed number).
//   MODEM_DECIMATION_CFG1 - Specifies three decimator ratios for the Casco
//   MODEM_DECIMATION_CFG0 - Specifies miscellaneous parameters and decimat
*/
#define RF_MODEM_TX_RAMP_DELAY_8 0x11, 0x20, 0x08, 0x18, 0x01, 0x00, 0x08,

/*
// Set properties:            RF_MODEM_BCR_OSR_1_9
// Number of properties:      9
// Group ID:                  0x20
// Start ID:                  0x22
// Default values:            0x00, 0x4B, 0x06, 0xD3, 0xA0, 0x06, 0xD3, 0x02
// Descriptions:
//   MODEM_BCR_OSR_1 - RX BCR/Slicer oversampling rate (12-bit unsigned nur
//   MODEM_BCR_OSR_0 - RX BCR/Slicer oversampling rate (12-bit unsigned nur
//   MODEM_BCR_NCO_OFFSET_2 - RX BCR NCO offset value (an unsigned 22-bit r
//   MODEM_BCR_NCO_OFFSET_1 - RX BCR NCO offset value (an unsigned 22-bit r
//   MODEM_BCR_NCO_OFFSET_0 - RX BCR NCO offset value (an unsigned 22-bit r
//   MODEM_BCR_GAIN_1 - The unsigned 11-bit RX BCR loop gain value.
//   MODEM_BCR_GAIN_0 - The unsigned 11-bit RX BCR loop gain value.
//   MODEM_BCR_GEAR - RX BCR loop gear control.
//   MODEM_BCR_MISC1 - Miscellaneous control bits for the RX BCR loop.
*/
#define RF_MODEM_BCR_OSR_1_9 0x11, 0x20, 0x09, 0x22, 0x00, 0x5E, 0x05, 0x76

/*
// Set properties:            RF_MODEM_AFC_GEAR_7
// Number of properties:      7
// Group ID:                  0x20
// Start ID:                  0x2C
// Default values:            0x00, 0x23, 0x83, 0x69, 0x00, 0x40, 0xA0,
// Descriptions:
//   MODEM_AFC_GEAR - RX AFC loop gear control.
//   MODEM_AFC_WAIT - RX AFC loop wait time control.
//   MODEM_AFC_GAIN_1 - Sets the gain of the PLL-based AFC acquisition loop
//   MODEM_AFC_GAIN_0 - Sets the gain of the PLL-based AFC acquisition loop
//   MODEM_AFC_LIMITER_1 - Set the AFC limiter value.
```

```
//    MODEM_AFC_LIMITER_0 - Set the AFC limiter value.
//    MODEM_AFC_MISC - Specifies miscellaneous AFC control bits.
*/
#define RF_MODEM_AFC_GEAR_7 0x11, 0x20, 0x07, 0x2C, 0x00, 0x12, 0x80, 0x2C,

/*
// Set properties:              RF_MODEM_AGC_CONTROL_1
// Number of properties:        1
// Group ID:                    0x20
// Start ID:                    0x35
// Default values:              0xE0,
// Descriptions:
//    MODEM_AGC_CONTROL - Miscellaneous control bits for the Automatic Gain
*/
#define RF_MODEM_AGC_CONTROL_1 0x11, 0x20, 0x01, 0x35, 0xE2

/*
// Set properties:              RF_MODEM_AGC_WINDOW_SIZE_9
// Number of properties:        9
// Group ID:                    0x20
// Start ID:                    0x38
// Default values:              0x11, 0x10, 0x10, 0x0B, 0x1C, 0x40, 0x00, 0x00
// Descriptions:
//    MODEM_AGC_WINDOW_SIZE - Specifies the size of the measurement and sett
//    MODEM_AGC_RFPD_DECAY - Sets the decay time of the RF peak detectors.
//    MODEM_AGC_IFPD_DECAY - Sets the decay time of the IF peak detectors.
//    MODEM_FSK4_GAIN1 - Specifies the gain factor of the secondary branch i
//    MODEM_FSK4_GAIN0 - Specifies the gain factor of the primary branch in
//    MODEM_FSK4_TH1 - 16 bit 4(G)FSK slicer threshold.
//    MODEM_FSK4_TH0 - 16 bit 4(G)FSK slicer threshold.
//    MODEM_FSK4_MAP - 4(G)FSK symbol mapping code.
//    MODEM_OOK_PDTC - Configures the attack and decay times of the OOK Peak
*/
#define RF_MODEM_AGC_WINDOW_SIZE_9 0x11, 0x20, 0x09, 0x38, 0x11, 0x15, 0x15

/*
// Set properties:              RF_MODEM_OOK_CNT1_9
// Number of properties:        9
// Group ID:                    0x20
// Start ID:                    0x42
// Default values:              0xA4, 0x03, 0x56, 0x02, 0x00, 0xA3, 0x02, 0x80
// Descriptions:
//    MODEM_OOK_CNT1 - OOK control.
//    MODEM_OOK_MISC - Selects the detector(s) used for demodulation of an O
//    MODEM_RAW_SEARCH - Defines and controls the search period length for t
//    MODEM_RAW_CONTROL - Defines gain and enable controls for raw / nonstan
//    MODEM_RAW_EYE_1 - 11 bit eye-open detector threshold.
//    MODEM_RAW_EYE_0 - 11 bit eye-open detector threshold.
//    MODEM_ANT_DIV_MODE - Antenna diversity mode settings.
```

```
//    MODEM_ANT_DIV_CONTROL - Specifies controls for the Antenna Diversity c
//    MODEM_RSSI_THRESH - Configures the RSSI threshold.
*/
#define RF_MODEM_OOK_CNT1_9 0x11, 0x20, 0x09, 0x42, 0xA4, 0x03, 0xD6, 0x03,

/*
// Set properties:              RF_MODEM_RSSI_CONTROL_1
// Number of properties:        1
// Group ID:                    0x20
// Start ID:                    0x4C
// Default values:              0x01,
// Descriptions:
//    MODEM_RSSI_CONTROL - Control of the averaging modes and latching time
*/
#define RF_MODEM_RSSI_CONTROL_1 0x11, 0x20, 0x01, 0x4C, 0x02

/*
// Set properties:              RF_MODEM_RSSI_COMP_1
// Number of properties:        1
// Group ID:                    0x20
// Start ID:                    0x4E
// Default values:              0x32,
// Descriptions:
//    MODEM_RSSI_COMP - RSSI compensation value.
*/
#define RF_MODEM_RSSI_COMP_1 0x11, 0x20, 0x01, 0x4E, 0x40

/*
// Set properties:              RF_MODEM_CLKGEN_BAND_1
// Number of properties:        1
// Group ID:                    0x20
// Start ID:                    0x51
// Default values:              0x08,
// Descriptions:
//    MODEM_CLKGEN_BAND - Select PLL Synthesizer output divider ratio as a f
*/
#define RF_MODEM_CLKGEN_BAND_1 0x11, 0x20, 0x01, 0x51, 0x0A

/*
// Set properties:              RF_MODEM_CHFLT_RX1_CHFLT_COE13_7_0_12
// Number of properties:        12
// Group ID:                    0x21
// Start ID:                    0x00
// Default values:              0xFF, 0xBA, 0x0F, 0x51, 0xCF, 0xA9, 0xC9, 0xFC
// Descriptions:
//    MODEM_CHFLT_RX1_CHFLT_COE13_7_0 - Filter coefficients for the first se
//    MODEM_CHFLT_RX1_CHFLT_COE12_7_0 - Filter coefficients for the first se
//    MODEM_CHFLT_RX1_CHFLT_COE11_7_0 - Filter coefficients for the first se
//    MODEM_CHFLT_RX1_CHFLT_COE10_7_0 - Filter coefficients for the first se
```

```
//    MODEM_CHFLT_RX1_CHFLT_COE9_7_0 - Filter coefficients for the first set
//    MODEM_CHFLT_RX1_CHFLT_COE8_7_0 - Filter coefficients for the first set
//    MODEM_CHFLT_RX1_CHFLT_COE7_7_0 - Filter coefficients for the first set
//    MODEM_CHFLT_RX1_CHFLT_COE6_7_0 - Filter coefficients for the first set
//    MODEM_CHFLT_RX1_CHFLT_COE5_7_0 - Filter coefficients for the first set
//    MODEM_CHFLT_RX1_CHFLT_COE4_7_0 - Filter coefficients for the first set
//    MODEM_CHFLT_RX1_CHFLT_COE3_7_0 - Filter coefficients for the first set
//    MODEM_CHFLT_RX1_CHFLT_COE2_7_0 - Filter coefficients for the first set
*/
#define RF_MODEM_CHFLT_RX1_CHFLT_COE13_7_0_12 0x11, 0x21, 0x0C, 0x00, 0x39,

/*
// Set properties:            RF_MODEM_CHFLT_RX1_CHFLT_COE1_7_0_12
// Number of properties:      12
// Group ID:                  0x21
// Start ID:                  0x0C
// Default values:            0xFC, 0xFD, 0x15, 0xFF, 0x00, 0x0F, 0xFF, 0xC4
// Descriptions:
//    MODEM_CHFLT_RX1_CHFLT_COE1_7_0 - Filter coefficients for the first set
//    MODEM_CHFLT_RX1_CHFLT_COE0_7_0 - Filter coefficients for the first set
//    MODEM_CHFLT_RX1_CHFLT_COEM0 - Filter coefficients for the first set of
//    MODEM_CHFLT_RX1_CHFLT_COEM1 - Filter coefficients for the first set of
//    MODEM_CHFLT_RX1_CHFLT_COEM2 - Filter coefficients for the first set of
//    MODEM_CHFLT_RX1_CHFLT_COEM3 - Filter coefficients for the first set of
//    MODEM_CHFLT_RX2_CHFLT_COE13_7_0 - Filter coefficients for the second s
//    MODEM_CHFLT_RX2_CHFLT_COE12_7_0 - Filter coefficients for the second s
//    MODEM_CHFLT_RX2_CHFLT_COE11_7_0 - Filter coefficients for the second s
//    MODEM_CHFLT_RX2_CHFLT_COE10_7_0 - Filter coefficients for the second s
//    MODEM_CHFLT_RX2_CHFLT_COE9_7_0 - Filter coefficients for the second se
//    MODEM_CHFLT_RX2_CHFLT_COE8_7_0 - Filter coefficients for the second se
*/
#define RF_MODEM_CHFLT_RX1_CHFLT_COE1_7_0_12 0x11, 0x21, 0x0C, 0x0C, 0xF6,

/*
// Set properties:            RF_MODEM_CHFLT_RX2_CHFLT_COE7_7_0_12
// Number of properties:      12
// Group ID:                  0x21
// Start ID:                  0x18
// Default values:            0xB8, 0xDE, 0x05, 0x17, 0x16, 0x0C, 0x03, 0x00
// Descriptions:
//    MODEM_CHFLT_RX2_CHFLT_COE7_7_0 - Filter coefficients for the second se
//    MODEM_CHFLT_RX2_CHFLT_COE6_7_0 - Filter coefficients for the second se
//    MODEM_CHFLT_RX2_CHFLT_COE5_7_0 - Filter coefficients for the second se
//    MODEM_CHFLT_RX2_CHFLT_COE4_7_0 - Filter coefficients for the second se
//    MODEM_CHFLT_RX2_CHFLT_COE3_7_0 - Filter coefficients for the second se
//    MODEM_CHFLT_RX2_CHFLT_COE2_7_0 - Filter coefficients for the second se
//    MODEM_CHFLT_RX2_CHFLT_COE1_7_0 - Filter coefficients for the second se
//    MODEM_CHFLT_RX2_CHFLT_COE0_7_0 - Filter coefficients for the second se
//    MODEM_CHFLT_RX2_CHFLT_COEM0 - Filter coefficients for the second set o
```

```
//   MODEM_CHFLT_RX2_CHFLT_COEM1 - Filter coefficients for the second set o
//   MODEM_CHFLT_RX2_CHFLT_COEM2 - Filter coefficients for the second set o
//   MODEM_CHFLT_RX2_CHFLT_COEM3 - Filter coefficients for the second set o
*/
#define RF_MODEM_CHFLT_RX2_CHFLT_COE7_7_0_12 0x11, 0x21, 0x0C, 0x18, 0x0C,

/*
// Set properties:            RF_PA_TC_1
// Number of properties:      1
// Group ID:                  0x22
// Start ID:                  0x03
// Default values:            0x5D,
// Descriptions:
//   PA_TC - Configuration of PA ramping parameters.
*/
#define RF_PA_TC_1 0x11, 0x22, 0x01, 0x03, 0x3D

/*
// Set properties:            RF_SYNTH_PFDCP_CPFF_7
// Number of properties:      7
// Group ID:                  0x23
// Start ID:                  0x00
// Default values:            0x2C, 0x0E, 0x0B, 0x04, 0x0C, 0x73, 0x03,
// Descriptions:
//   SYNTH_PFDCP_CPFF - Feed forward charge pump current selection.
//   SYNTH_PFDCP_CPINT - Integration charge pump current selection.
//   SYNTH_VCO_KV - Gain scaling factors (Kv) for the VCO tuning varactors
//   SYNTH_LPFILT3 - Value of resistor R2 in feed-forward path of loop filt
//   SYNTH_LPFILT2 - Value of capacitor C2 in feed-forward path of loop fil
//   SYNTH_LPFILT1 - Value of capacitors C1 and C3 in feed-forward path of
//   SYNTH_LPFILT0 - Bias current of the active amplifier in the feed-forwa
*/
#define RF_SYNTH_PFDCP_CPFF_7 0x11, 0x23, 0x07, 0x00, 0x2C, 0x0E, 0x0B, 0x0

/*
// Set properties:            RF_MATCH_VALUE_1_12
// Number of properties:      12
// Group ID:                  0x30
// Start ID:                  0x00
// Default values:            0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
// Descriptions:
//   MATCH_VALUE_1 - Match value to be compared with the result of logicall
//   MATCH_MASK_1 - Mask value to be logically AND-ed (bit-wise) with the M
//   MATCH_CTRL_1 - Enable for Packet Match functionality, and configuratio
//   MATCH_VALUE_2 - Match value to be compared with the result of logicall
//   MATCH_MASK_2 - Mask value to be logically AND-ed (bit-wise) with the M
//   MATCH_CTRL_2 - Configuration of Match Byte 2.
//   MATCH_VALUE_3 - Match value to be compared with the result of logicall
//   MATCH_MASK_3 - Mask value to be logically AND-ed (bit-wise) with the M
```

```
//   MATCH_CTRL_3 - Configuration of Match Byte 3.
//   MATCH_VALUE_4 - Match value to be compared with the result of logicall
//   MATCH_MASK_4 - Mask value to be logically AND-ed (bit-wise) with the
//   MATCH_CTRL_4 - Configuration of Match Byte 4.
*/
#define RF_MATCH_VALUE_1_12 0x11, 0x30, 0x0C, 0x00, 0x00, 0x00, 0x00, 0x00,

/*
// Set properties:            RF_FREQ_CONTROL_INTE_8
// Number of properties:      8
// Group ID:                  0x40
// Start ID:                  0x00
// Default values:            0x3C, 0x08, 0x00, 0x00, 0x00, 0x00, 0x20, 0xFF
// Descriptions:
//   FREQ_CONTROL_INTE - Frac-N PLL Synthesizer integer divide number.
//   FREQ_CONTROL_FRAC_2 - Frac-N PLL fraction number.
//   FREQ_CONTROL_FRAC_1 - Frac-N PLL fraction number.
//   FREQ_CONTROL_FRAC_0 - Frac-N PLL fraction number.
//   FREQ_CONTROL_CHANNEL_STEP_SIZE_1 - EZ Frequency Programming channel st
//   FREQ_CONTROL_CHANNEL_STEP_SIZE_0 - EZ Frequency Programming channel st
//   FREQ_CONTROL_W_SIZE - Set window gating period (in number of crystal
//   FREQ_CONTROL_VCOCNT_RX_ADJ - Adjust target count for VCO calibration i
*/
#define RF_FREQ_CONTROL_INTE_8 0x11, 0x40, 0x08, 0x00, 0x39, 0x0A, 0x80, 0x


// AUTOMATICALLY GENERATED CODE!
// DO NOT EDIT/MODIFY BELOW THIS LINE!
// ---------------------------------------------------

#ifndef FIRMWARE_LOAD_COMPILE
#define RADIO_CONFIGURATION_DATA_ARRAY { \
        0x07, RF_POWER_UP, \
        0x08, RF_GPIO_PIN_CFG, \
        0x06, RF_GLOBAL_XO_TUNE_2, \
        0x05, RF_GLOBAL_CONFIG_1, \
        0x08, RF_INT_CTL_ENABLE_4, \
        0x08, RF_FRR_CTL_A_MODE_4, \
        0x0D, RF_PREAMBLE_TX_LENGTH_9, \
        0x09, RF_SYNC_CONFIG_5, \
        0x0B, RF_PKT_CRC_CONFIG_7, \
        0x10, RF_PKT_LEN_12, \
        0x10, RF_PKT_FIELD_2_CRC_CONFIG_12, \
        0x10, RF_PKT_FIELD_5_CRC_CONFIG_12, \
        0x0D, RF_PKT_RX_FIELD_3_CRC_CONFIG_9, \
        0x10, RF_MODEM_MOD_TYPE_12, \
        0x05, RF_MODEM_FREQ_DEV_0_1, \
        0x0C, RF_MODEM_TX_RAMP_DELAY_8, \
        0x0D, RF_MODEM_BCR_OSR_1_9, \
```

```
            0x0B, RF_MODEM_AFC_GEAR_7, \
            0x05, RF_MODEM_AGC_CONTROL_1, \
            0x0D, RF_MODEM_AGC_WINDOW_SIZE_9, \
            0x0D, RF_MODEM_OOK_CNT1_9, \
            0x05, RF_MODEM_RSSI_CONTROL_1, \
            0x05, RF_MODEM_RSSI_COMP_1, \
            0x05, RF_MODEM_CLKGEN_BAND_1, \
            0x10, RF_MODEM_CHFLT_RX1_CHFLT_COE13_7_0_12, \
            0x10, RF_MODEM_CHFLT_RX1_CHFLT_COE1_7_0_12, \
            0x10, RF_MODEM_CHFLT_RX2_CHFLT_COE7_7_0_12, \
            0x05, RF_PA_TC_1, \
            0x0B, RF_SYNTH_PFDCP_CPFF_7, \
            0x10, RF_MATCH_VALUE_1_12, \
            0x0C, RF_FREQ_CONTROL_INTE_8, \
            0x00 \
  }
#else
#define RADIO_CONFIGURATION_DATA_ARRAY { 0 }
#endif

// DEFAULT VALUES FOR CONFIGURATION PARAMETERS
#define RADIO_CONFIGURATION_DATA_RADIO_XO_FREQ_DEFAULT
30000000L
#define RADIO_CONFIGURATION_DATA_CHANNEL_NUMBER_DEFAULT
0x00
#define RADIO_CONFIGURATION_DATA_RADIO_PACKET_LENGTH_DEFAULT
0x10
#define RADIO_CONFIGURATION_DATA_RADIO_STATE_AFTER_POWER_UP_DEFAULT
0x01
#define RADIO_CONFIGURATION_DATA_RADIO_DELAY_CNT_AFTER_RESET_DEFAULT
0x1000
#define RADIO_CONFIGURATION_DATA_CUSTOM_PAYLOAD_DEFAULT
{0x42, 0x55, 0x54, 0x54, 0x4F, 0x4E, 0x31} // BUTTON1

#define RADIO_CONFIGURATION_DATA_RADIO_PATCH_INCLUDED
0x00
#define RADIO_CONFIGURATION_DATA_RADIO_PATCH_SIZE
0x00
#define RADIO_CONFIGURATION_DATA_RADIO_PATCH
{   }

#ifndef RADIO_CONFIGURATION_DATA_ARRAY
#error "This␣property␣must␣be␣defined!"
#endif

#ifndef RADIO_CONFIGURATION_DATA_RADIO_XO_FREQ
#define RADIO_CONFIGURATION_DATA_RADIO_XO_FREQ              RADIO_CONFIGURATION_
#endif
```

```
#ifndef RADIO_CONFIGURATION_DATA_CHANNEL_NUMBER
#define RADIO_CONFIGURATION_DATA_CHANNEL_NUMBER             RADIO_CONFIGURATION_
#endif

#ifndef RADIO_CONFIGURATION_DATA_RADIO_PACKET_LENGTH
#define RADIO_CONFIGURATION_DATA_RADIO_PACKET_LENGTH      RADIO_CONFIGURATION_
#endif

#ifndef RADIO_CONFIGURATION_DATA_RADIO_STATE_AFTER_POWER_UP
#define RADIO_CONFIGURATION_DATA_RADIO_STATE_AFTER_POWER_UP    RADIO_CONFIGUI
#endif

#ifndef RADIO_CONFIGURATION_DATA_RADIO_DELAY_CNT_AFTER_RESET
#define RADIO_CONFIGURATION_DATA_RADIO_DELAY_CNT_AFTER_RESET   RADIO_CONFIGUI
#endif

#ifndef RADIO_CONFIGURATION_DATA_CUSTOM_PAYLOAD
#define RADIO_CONFIGURATION_DATA_CUSTOM_PAYLOAD             RADIO_CONFIGURATION_
#endif

#define RADIO_CONFIGURATION_DATA {  \
                                    Radio_Configuration_Data_Array,
 \
                                    RADIO_CONFIGURATION_DATA_CHANNEL_NUMBER,
 \
                                    RADIO_CONFIGURATION_DATA_RADIO_PACKET_LENGTH,
 \
                                    RADIO_CONFIGURATION_DATA_RADIO_STATE_AFTER_POWE
 \
                                    RADIO_CONFIGURATION_DATA_RADIO_DELAY_CNT_AFTER_
 \
                                    RADIO_CONFIGURATION_DATA_CUSTOM_PAYLOAD
 \
                                    }

#endif /* RADIO_CONFIG_H_ */
```

Az antenna leírója

```
CM
CE
SY f=437345000
SY la=3e8/f
SY r=0.026/3.14/2          'sug
SY l=la/4
SY s=40
GW 1 s -l*0.90 0 0 l*0.90 0 0 r
GW 2 s -l*0.96 -0.11 0 l*0.96 -0.11 0 r
GW 3 s -l*0.84 0.08 0 l*0.84 0.084 0 r
GW 4 s -l*0.83 0.22 0 l*0.83 0.22 0 r
GW 5 s -l*0.755 0.315 0 l*0.755 0.315 0
```

```
GW      6       s       -1*0.735        0.49    0       1*0.735 0.49    0
GE      0
GN      -1
EK
EX      0       1       s/2     0       1       0

FR      0       0       0       0       437.345 0
EN
```

A bootloader főprogramja

```
//-----------------------------------------------------------------------------
// Fxxx_TargetBL_Main.c
//-----------------------------------------------------------------------------
// Copyright (C) 2014 Silicon Laboratories, Inc.
// http://www.silabs.com
//
// Program Description:
//
// The main routine for the Target Bootloader Firmware.
//
//
//
//
// Target:         C8051Fxxx (Any Silicon Labs Flash MCU)
// Tool chain:     Generic
// Command Line:   None
//
//
// Release 1.0 / 22Sept2014 (SHY)
//    -Ported from F330
//
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <compiler_defs.h>
#include "Fxxx_SFR_Defs_Wrapper.h"      // This header file will include the
// real MCU register definition file
#include "Fxxx_Target_Config.h"
#include "Fxxx_Target_Interface.h"
#include "Fxxx_TargetBL_Config.h"
#include "Fxxx_TargetBL_Interface.h"
#include "Fxxx_BL129_UART_Interface.h"
#include "F85x_CRC.h"
//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Global Variables
//-----------------------------------------------------------------------------
```

```
sbit BL       =    P0^2;
sbit RED      =    P1^4;
sbit GREEN    =    P1^5;// LED='1' means ON
sbit BLUE     =    P1^6;
SEGMENT_VARIABLE( Page_Buf[TGT_FLASH_PAGE_SIZE], U8, SEG_XDATA);

U8 data SRC_Response;
U16 data SRC_Page_CRC;
U8 data rx_buf[32];
U8 Last_Error = 0;

#define APP_MODE         0
#define BOOTLOADER_MODE 1

// Bit masks for the RSTSRC SFR
#define PORSF  0x02
#define FERROR 0x40

void SRC_Validate_Response(U8 response);
void SRC_Validate_PageCRC(U16 page_addr);


//=========================================================================
// Main Routine
//=========================================================================
void main(void)
{
    U8 device_mode = BOOTLOADER_MODE;
        U8 code* codeptr;
        U16 data page_addr = 0;
        U16 addr = 0;
        U8 packets_for_page = 1;

        // Initialize port I/O
            P0MDOUT   = 0x10;  // P0.4 push-pull(UART TX ), P0.5 open-drain
            P1MDOUT   = 0x71;  // P1.0 push-pull(LED),
            //P1SKIP     = 0x81;  // Skip pin for LED and SW
            XBR0 = 0x01; // Enable UART0
            XBR2 = 0x40; // Enable crossbar
            GREEN = 0;
            BLUE = 0;
            RED = 0;


    //-------------------------------------
    // Check the bootloader consition.
    //-------------------------------------
    codeptr = (U8 code*)(APP_FW_SIG3_ADDR);
   // The Signature (in Flash) should be valid to allow application FW exec
```

```
// This is written at the end of the bootloading process by the bootload
if((codeptr[0] == SIG_BYTE3) &&(codeptr[1] == SIG_BYTE2) &&(codeptr[2]
{
        device_mode = APP_MODE;
}

if (!BL || (((RSTSRC & PORSF) == 0) && (RSTSRC & FERROR)))
{
    device_mode = BOOTLOADER_MODE;
}

if (device_mode == APP_MODE)
{
    BLUE = 1;
    // If not in BL Override, jump to application
    START_APPLICATION();
}

//-----------------------------------------------------
// ** BL Mode ** Initialize MCU and Variables
//-----------------------------------------------------
Device_Init();
GREEN = 1;
//-----------------------------------------------------
// Main Loop
//-----------------------------------------------------
    while(1)
    {
        while(SRC_Disp_TGT_Info() != SRC_RSP_OK);
        SRC_Response = SRC_Get_Info();
    SRC_Validate_Response(SRC_Response);
    if (Last_Error != 0)
        goto error;
        Flash_Key0 = rx_buf[5];
        Flash_Key1 = rx_buf[6];
        while(1)
        {
            SRC_Response = SRC_Get_Page_Info();
            SRC_Page_CRC = rx_buf[4] | (rx_buf[5] << 8);
            page_addr = rx_buf[1] | (rx_buf[2] << 8);
        SRC_Validate_Response(SRC_Response);
        if (Last_Error != 0)
            break;
        // Exit this loop if no more pages are available from sourc
        if (SRC_Response == SRC_RSP_DATA_END)
            break;

            TGT_Erase_Page(page_addr);
            addr = page_addr;
```

```
            packets_for_page = TGT_BL_PACKETS_FOR_GET_PAGE;
            while(packets_for_page--) {
                SRC_Response = SRC_Get_Page(Page_Buf);
                SRC_Validate_Response(SRC_Response); // this will not check

                if (Last_Error != 0)
                        break;

                TGT_Write_Flash(Page_Buf, addr);
                                addr += SRC_CMD_GET_PAGE_RX_SZ;
            }
            SRC_Validate_PageCRC(page_addr); // this will check for CRC ma
            if (Last_Error != 0)
                    break;
                }
        // Set flash keys to 0
        Flash_Key0 = 0;
        Flash_Key1 = 0;
error:
        if (Last_Error != 0)
        {
            SRC_Response = SRC_Disp_Info_Code(Last_Error);
            Last_Error = 0;
        }
        RSTSRC = 0x12; // Initiate software reset with vdd monitor enabled
        }
}

//-----------------------------------------------------------------------------
// SRC_Validate_Response
//-----------------------------------------------------------------------------
//
// Return Value:  None
// Parameters:    None
//
//
//
//-----------------------------------------------------------------------------
void SRC_Validate_Response(U8 response)
{
    if ((response != SRC_RSP_OK) && (response != SRC_RSP_DATA_END))
    {
            Last_Error = ERR_SRC_UNEXPECTED_RSP;
            return;
    }
}

void SRC_Validate_PageCRC(U16 page_addr)
```

```
{
    U16 buf_crc;
    buf_crc = Get_Buf_CRC((U8 code *)page_addr, TGT_FLASH_PAGE_SIZE);
    if (buf_crc != SRC_Page_CRC)
    {
       Last_Error = ERR_SRC_CRC_MISMATCH;
    }
}


//-------------------------------------------------------------------------
// End Of File
//-------------------------------------------------------------------------
```

A méréseknél használt program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdint.h>
#include "ismert.h"
int main(int argc,char **argv)
{
        uint32_t n=0;
        uint8_t data[64];
        char str[128];
        int rssi=0;
        while(!feof(stdin)){
                int k=fscanf(stdin,"%s %d\r\n",str,&rssi);
                if(k>0){
                        int i;
                        int e=0;
                        for(i=0;i<64;i++){
                                uint8_t f,a;
                                f=str[2*i];
                                a=str[2*i+1];
                                if(f>='A') f=f-'A'+10; else f-='0';
                                if(a>='A') a=a-'A'+10; else a-='0';
                                data[i]=(((uint8_t)f&0x0f)<<4)|((uint8_t)a&
                                int j;
                                uint8_t x=data[i]^known[i];
                                for(j=0;j<8;j++){
                                        if((x&1)>0) e++;
                                        x>>=1;
                                }
                        }
                        printf("%d\t%d\t%d\r\n",n++,e,rssi);
                        fflush(stdout);
                }
                usleep(100);
```

```
        }
        return 0;
}
```