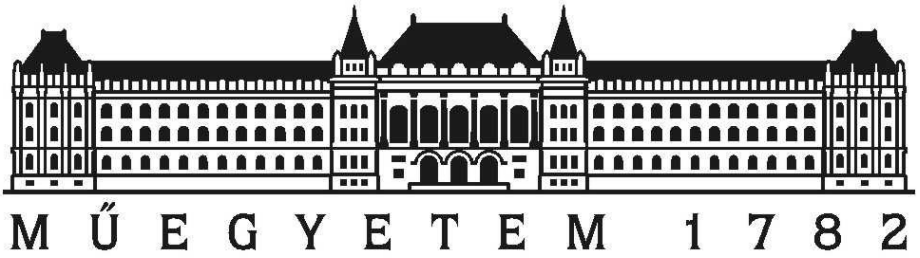
MÉRÉSI ÚTMUTATÓ

Rádióátviteli mérések laboratórium 2

ADS-B

V1 épület 504
Mikrohullámú Távérzékelés Laboratórium



BUDAPESTI MŰSZAKI és GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI és INFORMATIKAI KAR

Szélessávú Hírközlés és Villamosságtan Tanszék
1111 Budapest, Egry József utca 18, V1 épület
Tel.: (+36 1) 463-1559, Fax: (+36 1) 463-3289

Bevezetés

A mérés célja a szoftverrádiók, szoftveres jelfeldolgozási technikák, valamint a kooperatív módon működő szekunder radarok szabványos üzenetváltásának módjával való megismerkedés.
A mérés során Linux operációs rendszer használata szükséges. A leírás Debian alapú disztribúció (Linux Mint, Ubuntu, Debian, stb...) használatát feltételezi.

Munkakörnyezet kialakítása

Amennyiben rendelkezésünkre áll telepített Linux operációs rendszer, elegendő az alább látható csomagok telepítése, ha még nem állnak rendelkezésre.
Ellenkező esetben töltsünk le egy szimpatikus disztrót, kezdőknek ajánlott a Linux Mint Cinnamon grafikus felülettel, vagy választhatunk a különböző Ubuntu kiadások (Ubuntu LTS ajánlott) közül is. Ha sikerült letölteni az ISO képfájlt, akkor több lehetőségünk van:

- bootolható pendrive-ot készítünk belőle (Windows alatt pl Rufus nevű programmal), majd telepítjük a meglévő operációs rendszer mellé
- virtuális gépbe (pl VirtualBox, VMware, stb…) telepítjük; ebben az esetben telepítés után általában szükség van a virtuális gépen futó Linuxban telepíteni egy integrációs csomagot, ez minden VM szoftver esetében máshogy kell megtenni
- nem ajánlott, de lehetséges hogy a bootolható pendrive-ról indított „Live” rendszert használjuk, ebben az esetben vigyázni kell, hogy az itt elvégzett műveleteket (pl szoftver telepítések) kikapcsolás után elfelejti a rendszer; a fájlokat megfelelő helyre gyakran mentsük!

A méréshez az alábbi programcsomagokat telepítsük (a megadott parancsok Debian alapú rendszereken működnek):

```
sudo apt install gcc
sudo apt install build-essential
sudo apt install octave
```

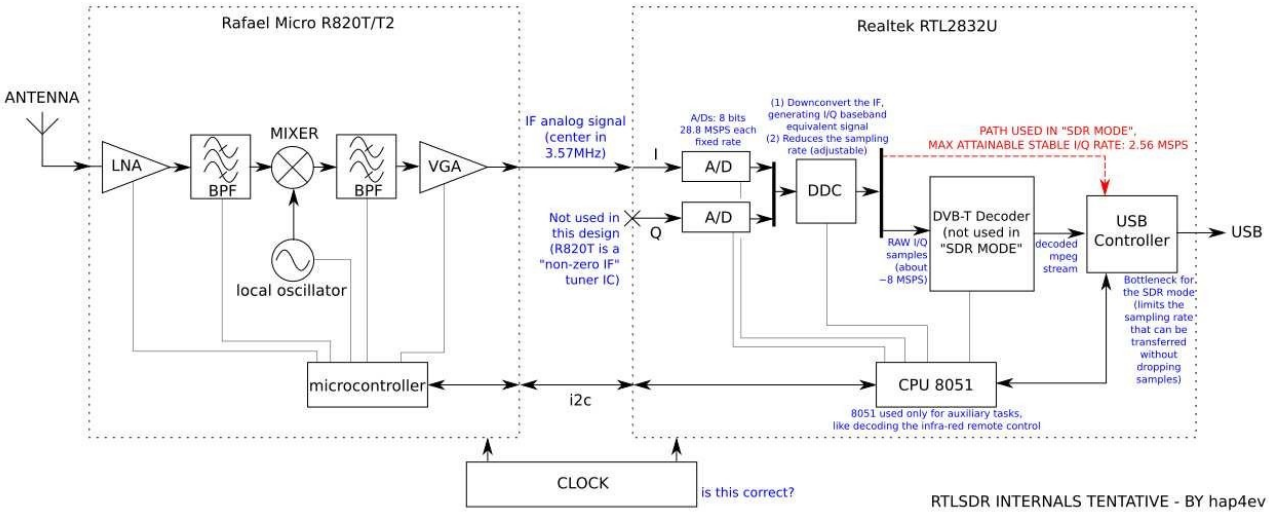
A gcc fordító valószínűleg már gyárilag telepítve van, ne lepődjünk meg, ha ezt jelzi a rendszer. Az octave egy Matlab kompatibilis szoftverkörnyezet.
Ha grafikus környezetben szeretnénk majd a .C kódot szerkeszteni akkor telepítsük pl. a Gedit szerkesztőt:

```
sudo apt install gedit
```

Szoftverrádió, RTL-SDR

A szoftverrádiók (SDR – Software Defined Radio) működési alapja, hogy vevő oldalon az antenna után minél hamarabb áttérjünk digitális feldolgozásra, valamint adó oldalon minél kevesebb analóg fokozat legyen a digitál-analóg átalakító és az antenna között. Ideális esetben az ADC és a DAC közvetlenül az antennára csatlakozik; jelenleg ez a megoldás még nem elterjedt a szükséges alkatrészek magas ára miatt.
Közelítő megoldás, ha beiktatunk egy tunert (nagyon leegyszerűsítve keverő) az antenna és a digitális rész közé. Ebben az esetben a szimultán feldolgozható sávszélesség jóval kisebb lesz, de maga a hardver nagyon olcsó lesz.

A fenti ábrán egy DVB-T televízió vevő látható, mely nagyon olcsón (néhány ezer forint) beszerezhető a különböző webes áruházakból. A működését évekkel ezelőtt sikerült visszafejteni, és azóta nagyon sokan szoftverrádió vevőként használják különféle projekteben. Maga az eszköz adásra nem alkalmas, de a vevőegysége 30MHz-től kezdve, jóval 1 GHz feletti tartományban képes venni.

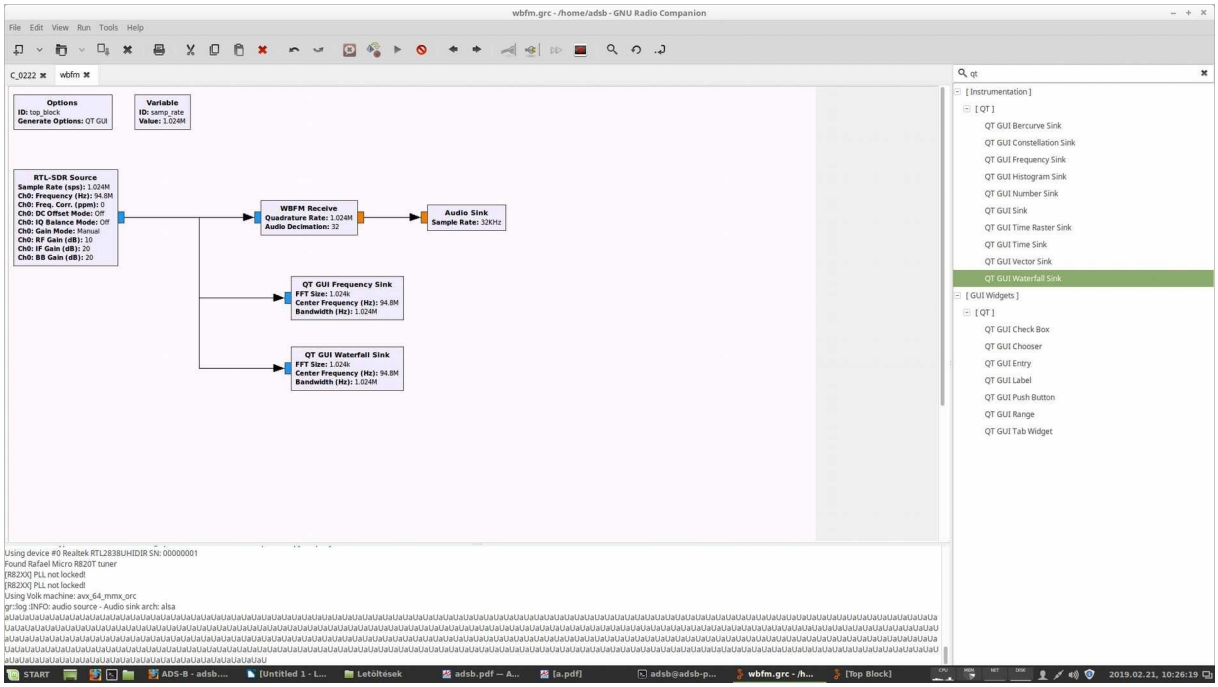


Az RTL-SDR vevő blokkvázlata a fenti ábrán látható, az elnevezése a digitalizációt és az USB alapú adatátvitelt megvalósító Realtek IC nevének rövidítéséből ered.

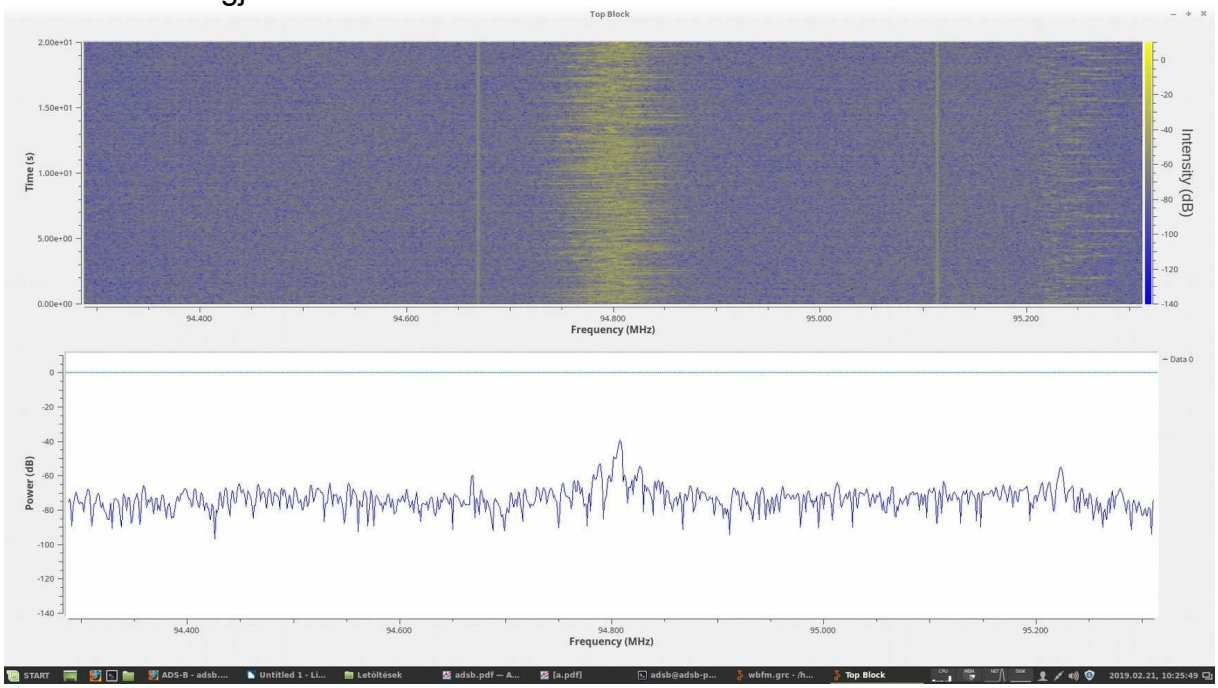
Egy-két nagyságrenddel nagyobb összegből már adásra is alkalmas SDR vásárolható (pl ADALM-PLUTO vagy az Ettus különféle számozású USRP-i)

GNU Radio

A gnuradio egy grafikus programozói környezet digitális jelfeldolgozási láncok létrehozására. Az élőszereplős mérések során az első mérési pont ebben a környezetben egy egyszerű FM rádió vevő létrehozása szokott lenni. A távoktatási rendszerben ennek a mérési pontnak az elvégzésétől eltekintünk, helyette bemutató jelleggel álljon itt egy létrehozott FM vevő blokkvázlata:

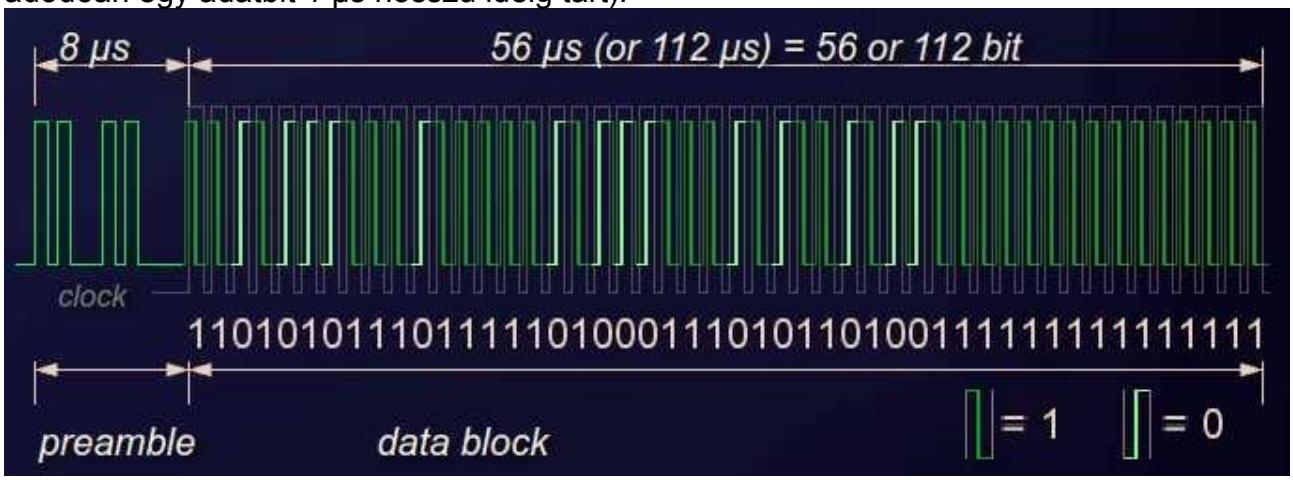


Az egyes „dobozok” a jobb oldali listából lettek kiválasztva. Az „RTL-SDR Source” blokk konfigurálja a megfelelő vételi és mintavételi frekvenciákra az RTL-SDR eszközt, majd az elvett I-Q mintákat a jobb oldalán lévő (kék színű) porton keresztül kiadja. Az FM demodulációt a „WBFM Receive” modul végzi el, a kimenetén már valós audio mintákat ad ki, mely az „Audio Sink” modul segítségével a hangkáryta kimenetére kiadásra kerül.
A „QT_GUI_Frequency_Sink” és „QT_GUI_Waterfall_Sink” modulok az alapsávi I-Q jelek spektrumának megjelenítését teszik lehetővé:



Szoftveres ADS-B vevő implementálása – bevezetés

Az ADS-B (Automatic dependent surveillance–broadcast) a Mode S típusú szekunder radar jelzésátviteli rendszeren alapuló automatikus helyzet és állapotjelzési protokoll.
A csomagok a Mode S 1090 MHz-es downlink frekvenciáján vehetőek, OOK demoduláció segítségével. Az alábbi ábrán a Mode S adatcsomagok felépítése látható. A fix mintázatú preamble-t egy 56 vagy 112 adatbitet tartalmazó rész követi. Az adatbitek Manchester kódolásúak. A preamble és a kódolt bitek 0,5 µs hosszú impulzusokból állnak (ebből adódóan egy adatbit 1 µs hosszú ideig tart).



A jel vételhez az RT-SDR 2 MSPS mintavételi sebességgel szolgáltatja az I-Q mintákat, ezt az alábbi paranccsal tudnánk elindítani, ha rendelkezésre állna egy eszköz:

```
rtl_sdr -f 1090000000 -s 2000000 -
```

Az ***-f*** kapcsolóval a venni kívánt RF frekvenciát, az ***-s*** kapcsolóval a mintavételi frekvenciát tudjuk megadni. A parancs végén önmagában álló kötőjel azt jelzi az *rtl_sdr* programnak, hogy a mintavett 8 bites I és Q mintákat standard outputján (**stdout**) adja ki.
Fontos megemlíteni, hogy a „linuxos" (és minden POSIX kompatibilis operációs rendszer alatti) programoknak van többek között egy **stdin** bemenetük és egy **stdout** kimenetük; a bemeneten érkező adatokat feldolgozzák, a feldolgozás eredményét pedig a kimeneten adják vissza. Ha önmagában futtatunk egy ilyen programot, akkor a bemenetre a billentyűzet által bevitt karakterek kerülnek, a kimenet pedig a terminálban/konzolban megjelenik a képernyő. Megtehetjük ugyanakkor azt is, hogy egy program kimeneti adatfolyama egy másik szoftver bemenetére kerül, ezt legegyszerűbben a **|** pipe szimbólummal tehetjük meg. Tehát ha például az általunk elkészítendő vevő szoftver neve demod lesz, akkor az RTL-SDR-ből jövő mintákat így tudnánk átadni:

```
rtl_sdr -f 1090000000 -s 2000000 - | ./demod
```

(A demod előtt álló **./** szimbólum azt jelzi hogy az OS a **demod** futtatható állományt az aktuális könyvtárban keresse; természetesen csak a megfelelő könyvtárban fogjuk tudni lefuttatni. Az **rtl_sdr** egy telepített alkalmazás, ezért azt bármely könyvtárból tudnánk futtatni.)
A mérés során valószínűsíthetően nem áll rendelkezésre RTL-SDR vevő, ehelyett egy előre rögzített mintafájlt fogunk használni (konzerv/iq.dat). Ezzel nagyon egyszerűen tudjuk helyettesíteni az **rtl_sdr** vevőszoftvert:

```
cat konzerv/iq.dat | ./demod
```

(A saját Neptun kódodnak megfelelő fájlt töltsd le, és nevezd át iq.dat-ra !)
A **cat** program a paraméterként megadott fájlt olvassa és a tartalmát kiadja az **stdout**-ján.
A vevőszoftver fejlesztése során többször is az Octave segítségével fogjuk ellenőrizni a **demod** kimenetét, ezért ahelyett, hogy a képernyőn jelenne meg a feldolgozás eredménye, azt egy (test.txt nevű) fájlba fogjuk átirányítani:

```
cat konzerv/iq.dat | ./demod > test.tx
```

Ha megnézzük a az ADS-B mérés fájljai között a **run** szkriptet, akkor a fenti parancsot fogjuk benne találni, így ennek begépelése helyett egyszerűen elegendő lesz a szkriptet futtatni:

```
bash run
```

Kezdő állapotban még nem létezik a **demod** futtatható állomány, ezt minden C programkód módosítás után fordítással fogjuk megkapni; ezt a **build** szkript futtatásával tehetjük meg:

```
bash build
```

(Ha megnézzük a szkript tartalmát, láthatjuk hogy a gcc fordító végzi el a fordítást.)

Szoftveres ADS-B vevő implementálása – programkód váz

Ha megnyitjuk szerkesztésre az adsb.c fájlt, láthatjuk, hogy nem a nulláról kell elkezdenünk a kód megírását. A különböző változók és konstansok definiálása után eljutunk a fő feldolgozó ciklushoz (amely egy do – while ciklus), amely láthatóan mindaddig fut, ameddig az **stdin** bemeneten kap mintákat. Az I-Q minták „elvételét” az **fread** függvény végzi el, egyszerre **BUF_SIZE** számú bájtot olvas be és helyez el a *buffer* tömbben. Az rtl_sdr szoftver kimenetén az egyes minták 8 bitesek, tehát egy bájtosak, az I és Q minták egymást felváltva követik egymást (IQIQIQIQ…). E két fontos információ alapján kikövetkeztethetjük, hogy egy **fread** beolvasás megfelel **BUF_SIZE / 2** számú I-Q mintapár elvételének. Lehetséges lenne, hogy a **BUF_SIZE** értéke 2 legyen, ekkor minden ciklusban egy komplex mintát olvasna be, azonban hatékonyabb ha egyszerre több mintát veszünk át (az **fread** függvény hívásának viszonylag nagy a költsége: rendszerhívás/syscall, kontextusváltások, stb...).
A beolvasás után látható egy belső **for ciklus**, amelyre igazából csak a fentebb jelzettek miatt van szükség, vagyis ez a ciklus egyesével fogja feldolgozni a tömbösítve átvett komplex mintákat (látható, hogy a *bix* futóváltozó kettesével növekszik az I-Q bájtpárok miatt).
A **for ciklus**ban látható kommentek jelzik, hogy a jelfeldolgozáshoz milyen lépéseket kell megvalósítani; majd pedig egy **printf** függvény a teszteléshez szükséges adatokat kiadja az **stdout** kimenetre (és ezek a **run** szkript szerint kiírásra kerülnek a test.txt fájlba).

Szoftveres ADS-B vevő implementálása – első fordítás

Nyissunk meg egy terminált (Ctrl + Alt + T) és navigáljunk el a programkódot és szkripteket tartalmazó könyvtárba,

VAGY

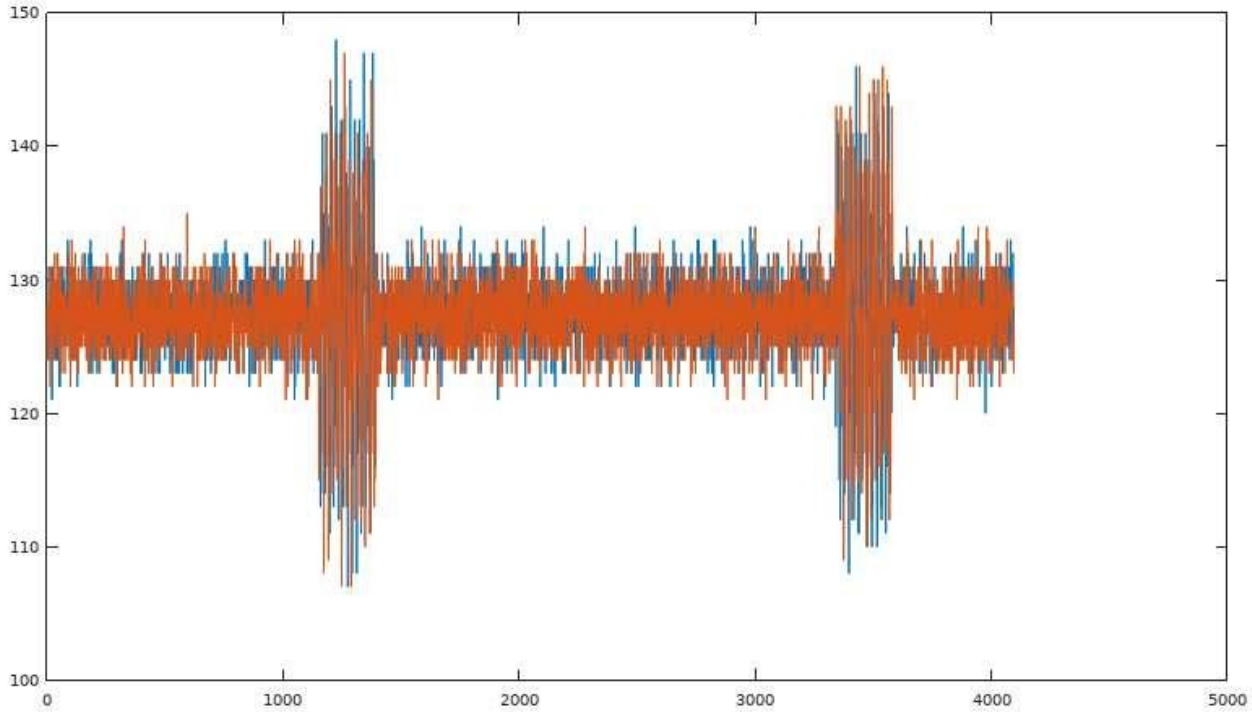
a grafikus fájlkezelőben menjünk a fájlokat tartalmazó könyvtárba, és ott jobb klikkel válasszuk a „Megnyitás terminálban” (vagy hasonló nevű) opciót.
Első lépésben mindenféle módosítás nélkül fordítsuk le a kódot, majd futtassuk, hogy megkapjuk a test.txt fájlt.
Nyissunk egy újabb terminált, amelyben navigáljunk a könyvtárunkba, majd indítsuk el az Octave környezetet az

```
octave
```

paranccsal. Az Octave-ban először töltsük be a test.txt fájl, majd rajzoljunk belőle grafikont:

```
>> load test.txt
>> plot(test)
```

Eredményül egy hasonló grafikont kell kapnunk (mindenkinél egyedi delay-el):



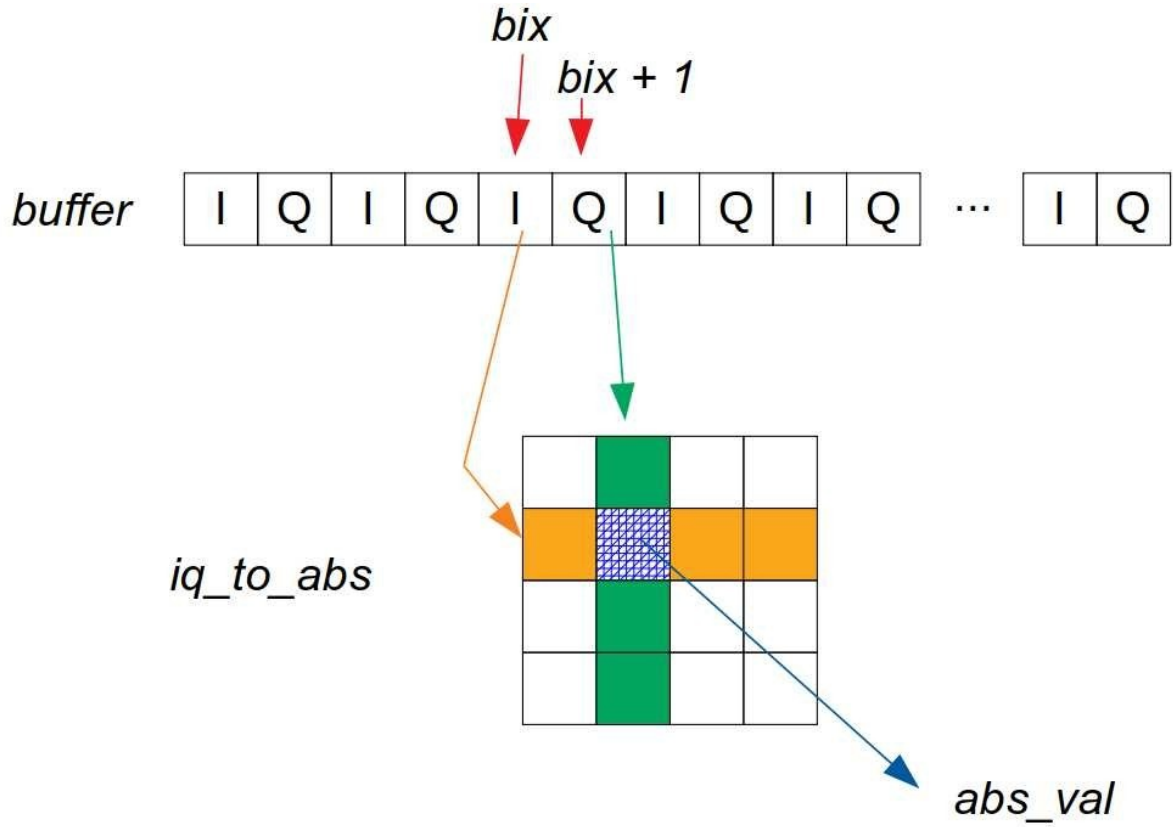
Látható, hogy az I és Q csatorna középértéke 128, ez a 8 bites teljes kivezérlési tartomány 256 szintjének közepe, ez azonos a jel DC = 0V-jával.

Szoftveres ADS-B vevő implementálása – abszolút érték képzés

Ahogy korábban említve volt, a Mode S jelzésátvitel OOK modulációt alkalmaz, tehát a szoftverünkben amplitúdó demodulációt kell végrehajtanunk. Ehhez első lépésben a komplex mintákból abszolút értéket kell képezni. A **for ciklus**ban mindig egy komplex mintát fogunk feldolgozni, amelynek I és Q értékei a *buffer* tömbben találhatóak, a *bix* futóváltozóval indexelve:

buffer[bix] == a komplex minta I bájtja
buffer[bix + 1] == a komplex minta Q bájtja.

Hogyan számítsuk ki a komplex minta magnitúdóját az I és Q értékből? Definíció szerint a két érték négyzeteinek összegének gyöke adja meg az amplitúdót.
A négyzetgyökvonás azonban meglehetősen számításigényes művelet, ehelyett egy Lookup Table (LUT) segítségével néhány memóriaolvasás segítségével megkaphatjuk a kívánt eredményt. A kódban látható *iq_to_abs* kétdimenziós tömb valósítja meg ezt a LUT-ot, amely az // "iq_to_abs" array initialization kommnent után látható **for ciklus**ban kerül feltöltésre.
Láthattuk, hogy az I és Q értékek nem közvetlenül állnak rendelkezésre, hanem egy tömb indexelés által, így az abszolút értéket a LUT miatt kettős indirekcióval kapjuk meg, az így megkapott értéket az *abs_val* változóban tároljuk el:

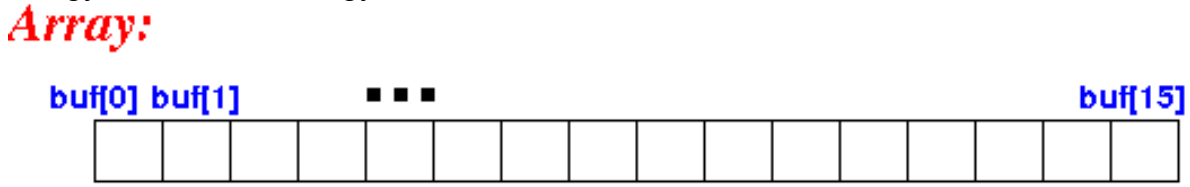


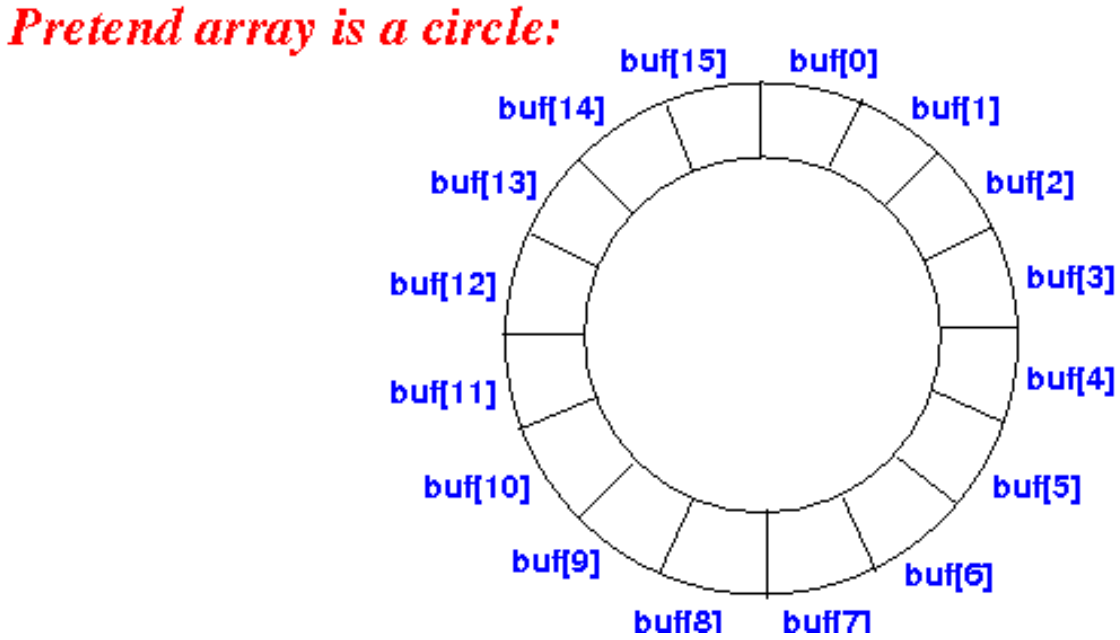
Ezután módosítsuk a ciklus végén található **printf** függvényt úgy, hogy harmadik paraméterként a kiszámolt abszolút értéket írja ki; ne felejtsük el az elválasztó tabulátort ("\t"). Fordítsuk le, majd futtassuk a kódot, az Octave-ban töltsük be újra a kimenetet és rajzoljuk ki a grafikont. Ha mindent jól csináltunk, a grafikon alján látható lesz az abszolút érték. Erről készítsünk képernyőmentést a jegyzőkönyvbe.

Szoftveres ADS-B vevő implementálása – OOK demoduláció / döntési szint

Most, hogy rendelkezésre áll az aktuális komplex minta abszolút értéke, a következő lépés az egyes bitek értékének meghatározása. Láthatjuk, hogy a mintavételi frekvencia megfelel a kódolt bitek 0,5 µs hosszának, tehát minden egyes mintát egy-egy bitre kell „átalakítani", azaz döntést kell végrehajtani. A döntéshez azonban először szükségünk van egy döntési szintre. A kérdés az, hogy ez mekkora legyen? Nyilván egy konstans érték nem lesz jó, mivel egy közeli és egy távoli transzponder jeleinek amplitúdója között nagyságrendi különbségek vannak, tehát mindenféleképpen egy adaptív döntési szintet kell létrehoznunk.
A Mode S csomag felépítését mutató ábrán láthatjuk, hogy a Manchester kódolás miatt az adatblokkban az 1-es és 0-s kódolt bitek száma megegyezik, tehát ha egy átlagot veszünk, akkor az egy jó döntési szint lehet. Ahhoz hogy mindez adaptív is legyen, alkalmazzunk egy megfelelően nagy, a megadott **FIR_LEN** minta hosszú mozgó átlagot!
A mozgó átlag kiszámításához az utolsó N minta ismerete szükséges. Emlékezzünk vissza, hogy a feldolgozó ciklusban mindig csak egy, az éppen aktuális minta áll rendelkezésre. Emiatt a szükséges számú, korábban meghatározott amplitúdóértéket el kell tárolnunk. Ebből a tárolóból mindig a legrégebbi minta fog kiesni, vagyis logikailag egy FIFO (First In, First Out) adattároló struktúrát kell megvalósítani. Láthatjuk a kódban, hogy már definiálva van egy *fifo* nevű tömb, ebben pontosan **FIR_LEN** értéket tudunk eltárolni (beleértve az aktuálisan kiszámolt amplitúdó értéket is).

Hogyan valósítsuk meg ezt a FIFO struktúrát? A legegyszerűbb az lenne, ha a mintákat shiftelnénk, és az elejére mindig az új értéket írnánk be. Ez azonban hosszú FIFO esetén nem lenne hatékony, ehelyett egy úgynevezett gyűrűs buffert kell megvalósítanunk, mely mellőzi az adatok mozgatását.
Példa egy 16 elem hosszú gyűrűs bufferre:





Látható, hogy a FIFO-t megvalósító tömböt képzeletben feltekertük, a végét és az elejét „összeragasztottuk". Szükségünk van még egy mutatóra (a kódban *fptr*-ként van definiálva), amely megmondja, hogy a tömb melyik elemére írhatjuk az új értéket (és éppen az itt felülírt érték lesz a legrégebbi amely egyébként a logikai FIFO-ból kiesne). Csupán arra kell vigyázni, hogy a mutató értékét növelve, az ne mutasson ki a tömbből, vagyis a fenti ábrán ha a mutató értéke 15, akkor a növelés után 0-nak kell lennie.
A kódunkban a FIFO (vagyis a *fifo nevű tömb*) **FIR_LEN** hosszúságú, tehát az *fptr* mutató értéke csak a 0 … (**FIR_LEN** – 1) értékeket veheti fel. Ezt maradékos osztással tudjuk megvalósítani, azaz a

fptr = fptr + 1;

növelés után maradékosan el kell osztani **FIR_LEN** hosszúsággal:

fptr = fptr % **FIR_LEN**;

(Megjegyzés: az új érték beírása után növeljük a mutató!t)

Ha megvalósítottuk a FIFO-t, akkor már minden egyes ciklus iterációnál rendelkezésre állnak a mozgóátlag képzéshez szükséges korábbi minták. Ezeket az átlagképzéshez már csak össze kell adni, és elosztani az elemszámmal. Legkevesebb munkával így tudnánk megvalósítani az átlag számítását, azonban ez megint csak nem lenne hatékony, különösen nagy elemszámú FIFO esetén.
A hatékony megoldáshoz csupán egy nulla értékre inicializált akkumulátor változóra (a kódban *accumulator*), valamint egy összeadás és egy kivonás műveletre van szükség; továbbá feltesszük, hogy a *fifo* tömb nullákkal van inicializálva (ahogy a kódban is látható).

Hogy is működik ez a megoldás?

- Kezdő állapotban az akkumulátor és a FIFO elemei is mind nulla, tehát teljesül, hogy a FIFO elemeinek összege megegyezik az akkumulátor értékével
- Kezdjük el a mintákat „beshiftelni” a FIFO-ba és ezeket mindig adjuk is hozzá az akkumulátorhoz. Amikor a legelső minta éppen elért a FIFO végére, álljunk meg a gondolatkísérletünkkel egy pillanatra, és lássuk be, hogy az akkumulátor értéke meg fog egyezni a FIFO-ban lévő eleme összegével.
- Ezután hajtsunk végre még egy ilyen műveletet, ekkor már láthatjuk, hogy az akkumulátor értéke nagyobb lesz mint az elemek összege, viszont pont annyi lesz az eltérés, mint a FIFO végéről kieső elem. Tehát ha annak értékét levonjuk az akkumulátorból, akkor helyreáll a rend, az akkumulátor értéke meg fog egyezni a FIFO aktuális elemeinek összegével.

Tehát a kódunkban minden egyes új abszolút értéket hozzá kell adni az akkumulátorból, és a FIFO-ból kieső értéket pedig le kell vonnunk belőle. Vigyázzunk arra, hogy a kieső elem a FIFO-ban abban a pozícióban található, ahová beírjuk az új értéket, tehát az akkumulátorból történő levonást még a FIFO elemének felülírása előtt kell megtennünk!

Ezután módosítsuk a ciklus végén található **printf** függvényt úgy, hogy negyedik paraméterként az mozgó átlag értéket írja ki; ne felejtsük el az elválasztó tabulátort (”\t”), valamint ara is emlékezzünk, hogy az akkumulátor az elemek összegét tartalmazza, nem az átlagot, tehát a kiíráshoz az értékét le kell osztanunk az elemek számával. Fordítsuk le, majd futtassuk a kódot, az Octave-ban töltsük be újra a kimenetet és rajzoljuk ki a grafikont. Ha mindent jól csináltunk, a grafikon alján látható lesz az abszolút érték, valamint meg kell jelennie a mozgóátlag értékének, nagyjából a „fűcsomók” közepén. Erről készítsünk képernyőmentést a jegyzőkönyvbe.
Ezek után az Octave-ot bezárhatjuk, a továbbiakban már nem lesz rá szükségünk. A kódunkban kommenteljük ki a teszteléshez használt **printf** függvényt, valamint a **while ciklus** vége előtt található **break;** utasítást is!

Szoftveres ADS-B vevő implementálása – OOK demoduláció / döntés

A következő lépés a döntési szint felhasználása a logikai 1 és 0-ák meghatározásához. Meg kell említeni egy, a szűrőkre általánosságban jellemező tulajdonságot, a késleltetést. Ennek következtében az éppen meghatározott döntési szint nem az aktuális mintára fog vonatkozni, hanem a szűrő válaszának késése miatt egy korábbi mintára. A mozgó átlag ablakfüggvényéből belátható, hogy ez a szűrő az ablak hosszának felével késleltet, hiszen az ablak közepén lévő mintához képest ugyanannyi újabb minta van mint korábbi (most tekintsünk el attól, hogy páros számú mintánál ez nem pontosan jön ki).
Tehát a döntéshez (**FIR_LEN** / 2)-vel ezelőtti abszolút értéket kell összehasonlítani az éppen kiszámolt átlaggal (*accumulator* / **FIR_LEN**). Amennyiben a minta a döntési szint felett van, logikai 1-re, ellenkező esetben 0-ra döntünk (*bit* változó).
A korábbi mintát megtaláljuk a FIFO-ban, ehhez a *fptr* mutató által mutatott hely felől „visszalépkedünk”. A korábbi mintára történő mutató értékének kiszámításakor ne feledkezzünk meg annak kordában tartásáról (maradékos osztás) ...

Szoftveres ADS-B vevő implementálása – ADS-B csomag keresése

Jelenleg minden egyes feldolgozási ciklusban rendelkezésre áll egy darab döntött bit. Ebből a bitstream-ből kell kihalászni az ADS-B csomagokat. A csomagok keresését a preamble alapján kell elvégezni, ez egy fix mintázat, mely előre definiálva van a C kódban (*adsb_preamble*). Az egyszerű megoldás itt is egy FIFO-ban történő eltárolás lenne, ahol minden egyes ciklusban a teljes preamble mintázatot össze kellene hasonlítani a FIFO tartalmával.
Ehelyett állapotgépes (state machine) keresést fogunk megvalósítani, ez nem igényli a korábbi bitek eltárolását, a memóriát egy darab állapotgép változó (*stm*) fogja megvalósítani:

- kezdésként *stm* == 0
- amennyiben az aktuális *bit* == *adsb_preamble*[*stm*], növeljük *stm* értékét, ellenkező esetben nullázzuk
- amennyiben a bitfolyam utolsó 16 bitje egy adott pillanatban megegyezik a preamble mintázattal, az *stm* értéke 16-ig fog növekedni (vigyázzunk, arra, hogy *stm* > 15 esetén ne indexeljük az *adsb_preamble* tömböt!)

A preamble keresés akkor sikeres ha az *stm* értéke elérte a 16-ot. Ezután megkezdhetjük a következőkben jövő 224 kódolt bit (hosszú Mode-S csomag, 112 Manchester kódolt adatbit) leszámolását, ehhez használjuk az *stm* változót (16 ... 239). Amikor leszámoltuk az utolsó kódolt bitet, nullázuk az állapotgép változó értékét, így a későbbiekben a keresés tovább folytatódhat.

Szoftveres ADS-B vevő implementálása – Manchester dekódolás

A Mode-S csomag felépítését mutató ábrán látható, hogy a Manchester kódolás az 1-es adatbitből 10 bitpárosra, 0-s adatbitből pedig 01 bitpárosra kódol. Ebből látható, hogy a legegyszerűbben úgy tudjuk dekódolni a Manchester kódolást, ha kizárólag az *stm* páros értékeinél lévő biteket vesszük figyelembe, ekkor azok meg fognak egyezni az adatbitekkel; természetesen ez csak *stm* >= 16 esetén igaz…
Az adatbiteket fűzzük össze bájtokká, ehhez használjuk a *hex* változót. A bitek az „MSb first” szabály szerint érkeznek, tehát a biteket az alsó helyiérték felől kell beshiftelnük a *hex* változóba:

```
hex = hex << 1;
if (bit == 1) hex = hex | 1;
```

Szoftveres ADS-B vevő implementálása – kimenet generálása

Elérkeztünk az utolsó lépéshez, a bemeneten (**stdin**) kapott minták jelfeldolgozását már megírtuk, most már csak a kimenetet (**stdout**) kell megvalósítani. A kimenetnek az alábi szabályoknak kell megfelelnie:

- minden dekódolt Mode-S csomag új sorba kerül
- a soroknak csillaggal (*) kell kezdődnie
- a csillag után a 112 adatbitnek kell következnie hexadecimális formában (112 / 4 = 28 karakter)
- a sor végét egy pontosvesszővel, majd pedig CR LF karakterekkel kell zárni

Néhány jó tanács az implementáláshoz:

- a csillag karaktert a preamble megtalálásakor célszerű kiírni (**printf("*");**)

- az összefűzött adatbiteket bájtosával írjuk ki (**printf("%02X", hex);**), ügyeljünk arra, hogy mely *stm* értékeknél kell ezt megtennünk
- a sor végét lezáró karaktereket (**printf(";\r\n");**) az állapotgép utolsó lépésben célszerű kiírni

Szoftveres ADS-B vevő tesztelése

Helyes működés esetén (ne feledkezünk meg a fordításról) a **run** szkript futtatása után a test.txt fájlban többségében az alábbi sorokat kellene látni (bithibákból adódó apróbb eltérések előfordulhatnak):

```
*904748061051831583 5820efe698;
*904748061051831583 5820efe698;
```

Helyes működés esetén az első demodulált bejegyzés egy egyedi kód, amelynek utolsó négy „karaktere” a letöltött konzerv fájl fájlnevében is megtalálható ellenőrzés céljából.
RTL-SDR vevő hiányában a további teszteléstől eltekintünk...

Jegyzőkönyv

A jegyzőkönyvbe a korábban jelzett képernyőmentések, az elkészült program forráskódja, valamint néhány demodulált csomag kerüljön bele (az egyedi kód mindenféleképpen).