SDR basics

Giacinto Gelli

Gennaio 2019

Sommario

In questo documento si descrivono in ordine sparso alcuni concetti relativi alla programmazione SDR con USRP, RTL-SDR e GNURadio. In alcuni casi si tratta di un backup in pulito di note scritte su quaderni e/o block notes.

Capitolo 1

Configurazione USRP

1.1 Configurazione di rete di USRP

La USRP di default ha l'indirizzo 192.168.10.2 (vedi manuale). Per collegarla è necessario impostare la rete: è necessario che l'host che deve controllare USRP abbia l'indirizzo di rete 192.168.10.1 con subnet mask 255.255.255.0 (non è necessario settare il gateway ed il DNS). Fare un check con ping da riga di comando per verificare che la USRP sia vista

```
ping 192.168.10.2
```

NB: su Windows è possibile utilizzare programmi come NetSetMan per commutare facilmente tra più configurazioni, su Linux Ubuntu basta aggiungere un nuovo profilo nelle configurazioni di rete.

Se sono stati installati i driver UHD ed il software di supporto, si può verificare il funzionamento di USRP anche con i comandi

```
uhd_find_devices
```

oppure

```
uhd_usrp_probe
```

1.2 Thread priority

Il programma di probe `uhd_usrp_probe` segnala un warning relativo alla possibilità di attivare la priorità dei thread (Ubuntu). Per sistemare:

1. Verificare che esista il gruppo `usrp` con

`cat /etc/group`

2. Aggiungere l'utente corrente `boss` al gruppo `usrp`

   `sudo usermod -a -G usrp boss`

3. Aggiungere la riga seguente al file `/etc/security/limits.conf`

   `@usrp    - rtprio    99`

4. Entrare e uscire dall'account per aggiornare le modifiche

5. Lanciare nuovamente `uhd_usrp_probe` per verificare che il warning sia scomparso

Source: `https://files.ettus.com/manual/page_general.html`

1.3 Aggiornamento firmware

1. Utilizzare l'utility `uhd_image_downloader` per scaricare il firmware (meglio con privilegi di root):

   `sudo uhd_images_downloader` (scarica tutti)

   `sudo uhd_images_downloader --types=usrp2` (scarica solo i firmware per USRP2)

   Tutti i firmware vengono memorizzati nella cartella `/usr/share/uhd/images`

2. Lanciare da riga di comando:

   `uhd_image_loader --args="type=usrp2,addr=192.168.10.2,reset"`

   che installa il firmware di default (scaricato in precedenza) ed effettua il reset della scheda. I file che vengono installati (gennaio 2019) sono

   `usrp_n210_fw.bin` (firmware image)

   `usrp_n210_r4_fpga.bin` (FPGA image)

3. Verificare dopo il reset che sia tutto OK con il comando `uhd_usrp_probe`

1.4 Antenna

L'antenna fornita con USRP2 è il prodotto VERT 400, Tri-band vertical antenna. Copre tre bande: 144 MHz, 400 MHz, 1200 MHz

I dati sono i seguenti da `https://www.ettus.com/product/details/VERT400`:

```
Tri-band 2M/70cm/1200MHz HT Antenna
Extended receive range: 118-160MHz, 250-290MHz,
```

```
360-390MHz, 420-470MHz, 820-960MHz, 1260-1300MHz
Gain & Wave:
146MHz 0dBi 1/4 wave
446MHz 0dBi 1/4 wave
1200MHz 3.4dBi 5/8 wave
Max Power: 10 watts
Length: 17 cm (6.5 inches)
Connector: SMA
```

1.5 Setting di USRP per Matlab

La USRP NI-2920 in realtà è la ETTUS N201 con daughterboard WBF incorporata. Di fabbrica usa i driver UHD compatibili con Labview e GNURadio. Per farla funzionare con Matlab è necessario flashare il firmware come documentato anche da Matlab (USRP2).

1. Flashare il nuovo firmware; da Matlab si può fare con il comando:

   `sdruload('Device', 'n210_r4')`

2. Al termine verificare con il comando `findsdru`

Compare un messaggio di warning sulla dimensione della MTU; possibile modifica con Windows (registro di sistema, vedi dopo) ma probabilmente funziona ugualmente.

NB: il firmware per Matlab è diverso da quello per GNURadio, per cui è necessario ripetere il flash del firmware ogni volta che si passa da Matlab a GNURadio (vedere meglio istruzioni per il flash del vecchio firmware).

1.6 Settaggio USRP sotto Linux/Ubuntu

1. Il firmware di Linux è incompatibile con quello che usa Matlab sotto Windows; ho provato anche con Matlab sotto Linux ed il firmware è incompatibile

2. Provo a caricare il firmware con

   `sudo uhd_images_downloader`

   che scarica il firmware ma non sono sicuro se sia compatibile cone Matlab oppure con GNURadio (credo la seconda)

3. Provo a flashare il firmware scaricato con il comando

   `sudo uhd_image_loader --args="type usrp2, addr = 192.168.10.2"`

   e va a buon fine; un successivo `findsdru` da Matlab non funziona; confermando che il firmware per Matlab è differente

Un successivo

`uhd_usrp_probe` non va a buon fine, quindi non è stato flashato correttamente (come si risove?)

4. Provo ad aggiornare il firmware da Matlab, avendo già installato il supporto (toolbox) USRP lancio il comando

   ```
   sdruload('Device','n210_r4')
   ```

   Matlab installa il firmware che preleva da una sua directory interna; questa procedura di aggiornamento del firmware è la setssa che si effettua sotto linux: al termine `findsdru` fornisce un messaggio di successo; interessante che anche `uhd_usrp_probe` va a buon fine (da riga di comando) per cui con Matlab riesco a settare correttamente la USRP anche per l'uso con GNURadio (credo!)

1.7 Frequenze di campionamento di USRP

La frequenza base di USRP è 100 MHz. Al suo ingresso posso avere frequenze

$$\frac{100}{N}$$

Trovo su Internet che USRP può funzionare con fattori di interpolazione/decimazione che seguono questa legge

da 1-128

da 128-256, N pari

da 256-512, N divisibile per quattro

Questi parametri possono essere settati in Matlab attraverso le proprietà di SDRu System Object

La divisibilità per quattro del fattore di interpolazione è richiamato anche in altri post ed è legato alla particolare architettura dell'USRP (CIC filters).

1.8 Frequenze di campionamento di RTL-SDR

C'è un limitato numero di frequenze, ma la granularità è abbastanza alta. Sono tutte sotto-multiple (o legate razionalmente) al clock a 28.8 MHz. Ho trovato un documento (RTL2832 sample rates.pdf) che descrive TUTTE le possibili frequenze utilizzabili:

le principali sono SOTTOMULTIPLI di 28.8, ad esempio gqrx usa le seguenti:
240 000 fattore 120
300 000 fattore 96
960 000 fattore 30
1 152 000 fattore 25

1.9 Installazione RTL-SDR sotto Linux

Seguire passo passo le istruzioni del documento "RTLSDR for Linux Quick Start Guide".

In alternativa è possibile installarla sotto Ubuntu ricorrendo ai pacchetti:

```
sudo apt install rtl-sdr
```

Nota: con Ubuntu 18 è già installata (o forse è installata con GNURadio). Per provare, basta inserire la chiavetta ed utilizzare il comando:

```
rtl_test -t
```

Una ulteriore prova che si può effettuare è attraverso il comando

```
rtl_fm <opzioni>
```

Per il supporto di RTL-SDR su GNURadio bisogna installare il pacchetto `gr-osmosdr` con il classico

```
sudo apt install gr-osmosdr
```

Per verificare l'installazione, da riga di comando c'è un comando molto semplice per fare la fft ed utilizzarla:

```
osmocom_fft
```

Conviene anche installare `gqrx` per avere un ambiente grafico per fare semplici esplorazioni dello spettro:

```
sudo apt install gqrx-sdr
```

Lanciare il programma con:

```
gqrx
```

1.10 Uso USRP sotto Linux

Comandi per verificare il device:

```
uhd_usrp_probe
```

```
uhd_find_devices
```

Con il comando:

```
uhd_fft -f100e6
```

si lancia una GUI che calcola la FFT ad una singola frequenza

Con il comando

```
uhd_siggen_gui
```

si generano vari segnali da riga di comando che possono essere utilizzati per il testing della scheda

1.11 GNURadio per Ubuntu

L'installazione più conveniente di GNURadio per Ubuntu è la seguente:

```
sudo apt update
```

```
sudo apt upgrade
```

```
sudo apt install gnuradio
```

Se si prova a lanciare `gnuradio-companion` da riga di comando si ha un errore (Ubuntu 18). Bisogna installare manualmente un pacchetto:

```
sudo apt install python-gtk2
```

Probabilmente sarà corretta nelle prossime release di GNURadio e/o della distribuzione di Ubuntu.[1]

Dopo l'instllazione lanciare gnuradio-companion

```
$ gnuradio-companion
```

[1] Nell'installazione su Ubuntu 18.04 LTS (giugno 2019) il problema non si è verificato, per cui penso che sia stato corretto.

Se si lancia un qualunque flowgraph, lancia un warning "The xterm executable is missing". To fix it, biosgna modificare il file grc.conf che si trova in

`/etc/gnuradio/conf.d`

con i privilegi di root, aprire il file grc.conf ed aggiungere la linea

`xterm_executable=gnome-terminal`

Issue: jack server

Ci sono vari warning di GNURadio. Uno di questi è relativo al jack server che è un driver a bassa latenza per la scheda audio. Provo a risolverlo installandolo.

`sudo apt install jackd`

`jack_control start`

Bisogna modificare alcune cose nel file `limits.conf` (vedere su Internet) prima che il tutto funzioni.

1.12 GNURadio per Windows

Ci sono dei binari già compilati disponibili `http://www.gcndevelopment.com/gnuradio/downloads.htm`

1.13 Impiego della RTL-SDR per GPS (Windows)

Si parla della chiavetta argentata acquistata sul sito RTL-SDR

1. La chiavetta monta i soliti driver per i quali è necessario effettuare la solita installazione com Zadig
2. Per ricevere un segnale GPS è necessario collegare un'antenna attiva (quella del Teseo va bene); l'antenna va alimentata dalla chiavetta. La chiavetta contiene un dispositivo bias-tee che serve per l'alimentazione ed ha un controllo software; attraverso un file scaricabile è possibile utilizzare i comandi

   `bias_tee_on`

   `bias_tee_off`

   per far funzionare la chiavetta. Il problema è che il segnale non è visualizzabile sullo schermo (ad esempio con SDRSharp) per il livello basso di potenza; per questo motivo per elaborarlo bisogna acquisirlo (vedi dopo) ed eventualmente cross-correlarlo con i codici di Gold

3. per testare il funzionamento della chiavetta è possibile utilizzare il software GNSS + SDRLIB + RTKLIB come documentato sul sito `www.rtl-sdr.com`

1.14 FM stereo

Alcune considerazioni sull'implementazione di un ricevitore FM stereo:

1. Per la demodulazione del segnale composito (multiplex) a valle del discriminatore FM ho trovato più conveniente utilizzare segnali complessi, cioè lavorare con il segnale analitico (si utilizza a tale scopo per la generazione un blocco `hilbert`). Questo consente ad esempio di demodulare il segnale L-R in maniera più semplice (nella moltiplicazione per la portante locale non si generano termini a frequenza doppia). Inoltre si adatta meglio al funzionamento del blocco PLL di GNURadio che per sua natura utilizza segnale complessi.

2. Nella demodulazione del multiplex stereo assume una particolare importanza il progetto dei filtri. Infatti è importante anzitutto che i filtri LPF e BPF che devono estrarre le componenti L+R ed L-R siano perfettamente simmetrici (stesso numero di taps, stesso profilo in frequenza). Altrimenti si generano degli sbilanciamenti per cui non si recuperano più correttamente i segnali L ed R (ho fatto delle prove con segnali solo L oppure solo R). Per fare questo ho utilizzato un solo filtro LPF che agisce su L+R e su L-R riportato in bassa frequenza.

3. Un altro aspetto importante è che il filtro per l'estrazione della pilota a 19 kHz introduce un ritardo nel segnale che va adeguatamente compensato sui due rami L+R ed L-R, altrimenti si introduce un errore di fase nella demodulazione. Ho introdotto un blocco per questo che calcola la lunghezza del filtro per il recupero della pilota e setta correttamente il ritardo (pari a $(N-1)/2$ campioni per filtri FIR a fase lineare con N taps).

4. Ancora, un altro aspetto è che la normativa ITU-R prevede una certa relazione di fase tra la pilota a 19 kHz e la sottoportante a 38 kHz. Secondo i miei calcoli questo equivale ad introdurre dopo il moltiplicatore x 2 di frequenza uno sfasatore a 90 gradi. Per maggior flessibilità ho reso tale sfasamento un parametro di ingresso dei blocchi modulatore e demodulatore. Tuttavia le verifiche che ho effettuato confermano tale interpretazione: settando ad esempio a 0 tale valore nei flowgraph, il segnale L-R è quasi nullo e non si ha una ricezione audio stereo.

5. Dai test audio con una catena TX-RX ho riscontrato che il segnale presenta una distorsione significativa in presenza di suoni sibilanti (la s). Su Internet ho trovato un articolo che spiega i motivi (vedi articolo di Lourens). La spiegazione è una sovramodulazione dell'onda FM che è peggiorata dalla preenfasi. Una soluzione proposta (che andava realizzata comunque per evitare che il segnale FM presenti deviazioni di frequenza eccessive) è effettuare un limiting del segnale DOPO la preenfasi. Ho usato il blocco `rail` di GNUradio settando i limiti a -1 e +1. Un test mostra un deciso miglioramento in

corrispondenza delle sibilanti utilizzando tale accorgimento. Ovviamente il limiting può introdurre delle distorsioni nel segnale da trasmettere (da valutare).

6. Ispirato dall'esame dei blocchi GNURadio ho implementato anche una versione dei demodulatori che utilizzano un PLL per effettuare la demodulazione FM. Il settaggio dei parametri è abbastanza complicato: bisogna ricordare che il loop filter deve presentare una banda sufficientemente ampia per lasciar passare il segnale modulante. Ad esempio, per un `samp_rate` tipico di 240 kHz la banda numerica del loop filter sarà

   ```
   loop_bw = 2*math.pi*15.0/240.0
   ```

   ovvero

   ```
   loop_bw = 2*math.pi/5.0
   ```

   che risulta MOLTO diversa da quella consigliata da GNURadio (ed anche da quella apparentemente implementata nel blocco `wfm_rcv_pll.py`, pari a

   ```
   loop_bw = 2*math.pi/100.0
   ```

1.14.1 Audio choppy e lunghezza dei filtri FIR

Spostando il mio ricevitore FM dal PC Dell (Windows) ad un PC meno potente (Linux) i ricevitori mono/stereo hanno iniziato a funzionare con audio choppy (a singhiozzo), con lunghe stringhe di "OaU' da GRC. In un primo momento ho pensato che fosse un problema legato al driver della scheda audio (alsa), ma dopo un serie di prove in cui ho considerato un flowgraph ridotto all'osso e ho confrontato linea per linea i codici Python di due programmi (uno funzionante e l'altro no), ho trovato il problema:

nell'implementazione del primo filtro FIR a valle del RTL-SDR avevo settato una frequenza di taglio di 180e3 ed un banda di transizione di 2e3 molto piccola, il che genera un filtro FIR molto lungo (quasi 3000 taps, verificare con `len(taps)` in un blocco `Variable`). Con il comando `top` di Linux il processo `python2` occupa più del 100 % di CPU, ed il processo `pulseaudio` non riesce ad elaborare l'audio in tempo reale. Aumentando la banda di transizione a 5e3 la lunghezza del filtro FIR scende a circa 1000 taps, la CPU occupata scende intorno al 77 % e l'audio funziona. In effetti anche valori più grandi della banda di transizione si possono considerare.

1.15 Confronto mie implementazioni con blocchi GNURadio

In GNURadio ci sono tre blocchi che implementano TX/RX FM:

`WBFM Receive`: implementa un ricevitore FM mono (usa `wfm_rcv.py`)

`WBFM Receive PLL`: implementa un ricevitore FM stereo, recupera la portante a 19 kHz ed utilizza un PLL per rigenerarla (usa `wfm_rcv_pll.py`)

`WBFM Transmit`: implementa un trasmettitore FM mono (usa `wfm_tx.py`)

Inoltre è presente nella distro GNURadio anche `wfm_rcv_fmdet.py` che però non è tra i blocchi GRC.

E' necessaria un'analisi più approfondita del codice (mi sembra che siano scritti in Python, quindi non dovrebbe essere complicato). Comunque, ho effettuato qualche prova confrontando la mia implementazione con quella GNURadio.

Nel caso mono si hanno risultati confrontabili, anche se va indagato meglio come il RX predefinito effettua il filtraggio (la forma dello spettro audio è abbastanza differente nei due casi) (vedi dopo).

Nel caso stereo il RX predefinito funziona male; da un esame preliminare del codice (da approfondire) sembra che non si ripristini correttamente la separazione stereo (infatti i canali L ed R sembrano uguali). Inoltre ho il sospetto che la preenfasi venga effettuata su tutto il segnale (sullo stereo multiplex) il che poi costringe a strane compensazioni nei filtri (secondo me questo aumenta il livello di rumore, oltre a potenzialmente creare sbilanciamenti tra L ed R). Devo però analizzare meglio i codici utilizzati da GNURadio in particolare:

`analog.wfm_rcv` per il ricevitore MONO

`analog.wfm_rcv_pll` per il ricevitore STEREO

L'analisi del trasmettitore MONO non l'ho ancora effettuata.

NB: si può anche fare un flowgraph che implementa i due ricevitori in parallelo per vedere gli output nella stessa GUI (fatto per il caso mono)

1.16 Quadrature demodulator

Codice C `quadrature_demod_cf_impl.cc` (directory GitHub `gnuradio/gr-analog/lib/`)

Dall'esame del codice esso ha un parametro di ingresso `gain`, ed agisce su un flusso complesso di ingresso generando un flusso float in uscita. Detto $x(n)$ il segnale di ingresso, $y(n)$ il segnale di uscita, e g il guadagno (gain), il modello è il seguente:

1. Effettua la moltiplicazione tra $x(n)$ ed $x^*(n-1)$, lo memorizza in una variabile di appoggio `tmp`

2. Calcola la fase del risultato utilizzando la funzione `atan2` che opera sulla parte reale ed immaginaria di `tmp`

3. Restituisce il risultato in uscita moltiplicando per g (variabile `gain`)

Quindi il modello i-u dovrebbe essere:

$$y(n) = g \arg\{x(n)x^*(n-1)\}$$

Se $x(n)$ contiene i campioni I/Q di un segnale FM, deve risultare

$$x(n) = A\, e^{j(2\pi k_f \sum_{k=-\infty}^{n} m(k)T_s + \phi_0)}$$

dove il termine $\sum_{k=-\infty}^{n} m(k)T_s$ deriva dalla discretizzazione dell'integrale $\int_{-\infty}^{t} m(\tau)d\tau$. Con facili passaggi, si ha:

$$x(n)x^*(n-1) = A^2\, e^{j2\pi k_f \left(\sum_{k=-\infty}^{n} m(k)T_s - \sum_{k=-\infty}^{n-1} m(k)\right)} = A^2\, e^{j2\pi k_f m(n)T_s}$$

da cui:

$$y(n) = g \arg\{x(n)x^*(n-1)\} = g\, 2\pi k_f T_s m(n)$$

Pertanto il corretto valore del guadagno g per ripristinare il segnale $m(n)$ è

$$g = \frac{1}{2\pi k_f T_s}$$

dove k_f rappresenta la massima deviazione di frequenza (assumendo che $|m(t)| \leq 1$).

1.17 Funzione RDS

Ho esplorato il pacchetto `gr-rds` scritto da Bastian Boessl per l'implementazione GNURadio di un ricevitore/trasmettitore RDS. Con la versione di GNURadio disponibile su Windows, il pacchetto è nella versione WX GUI (una versione QT GUI è disponibile su `github` nel ramo `next` ma non è semplice integrarla in GNURadio, in quanto utilizza Qt5).

Il pacchetto contiene una serie di blocchi per il lato TX/RX e qualche flowgraph di esempio (uno dei quali contiene un ricevtore FM radio, da testare). Sono partito da esso per modificarlo in modo da integrarlo nel mio ricevitore FM stereo. Dopo aver studiato attentamente il formato del segnale RDS, ho impostato i parametri del ricevitore in modo da aver un numero di bit/simbolo intero (80). Tale valore nasce considerando che il bit-rate del sistema SPS è 1187.5 bps, e considerando anche la modulazione bifase, per cui si ha:

$$f_c = 1187.5 \times 2 \times 80 = 190\text{kHz}$$

Questo comporta l'impiego di una frequenza di campionamento nel ricevitore differente da quella utilizzata e pari a 190 kHz. L'impiego di tale frequenza ha costretto ad inserire nel ricevitore FM un blocco di resampling.

Ho leggermente modificato la struttura del ricevitore rispetto al file di esempio in modo da sfruttare tutta l'energia del segnale e non metà come fatto da Boessl. Lato trasmittente ho fatto qualche modifica ma l'ultimo flograph non funziona.

Da fare: andrebbe sviluppato un blocco python con QT GUI per sostituire il blocco `rdspanel` che utilizza WX, in modo da avere anche le informazioni RDS sulla GUI. AL momento questo è possibile solo con la versione di `rdspanel` che utilizza WX.

1.18 Analisi ricevitori FM GNURadio

Ricevitore `wfm_rcv_py`

Utilizza come ingressi `quad_rate` e `audio_decimation`, definisce

`audio_rate = quad_rate | audio_decimation`

Tipici valori sono `audio_rate` = 48 kHz e `quad_rate` = 240 kHz pari a `audio_decimation` = 5.

1. Utilizza `quadrature_demod_cf` (vedi sezione descrittiva) con guadagno di demodulazione fissato a

   `quad_rate / (2*math.pi*max_dev)`

   `max_dev = 75e3`

   Questi valori dall'analisi che ho fatto sono quelli corretti per riscalare il segnale modulante alla dinamica originaria

2. Effettua un filtraggio del segnale risultante con un filtro LPF per la demodulazione i cui tap sono definiti come segue:

   `firdes.low_pass(1.0, quad_rate, audio_rate/2 - width_of_transition_band, widt`

   dove

   `width_of_transition_band = audio_rate/32`

   La frequenza di taglio per `audio_rate` = 48 kHz viene pari a 22.5 kHz mentre la banda di transizione è 48/32 = 1.5 kHz. Un po' larga la banda....(non sarebbe stato più appropriato 15 kHz?)

3. Effettua la deenfasi (non è possibile settare `tau` per cui dovrebbe utilizzare quello di default paria 75 μs (USA)

In definitiva l'unico problema grave è quello di non poter settare correttamente la pre-enfasi; la banda del filtro LPF sembra un po' larga ma questo va investigato oltre.

Ricevitore `wfm_rcv_fmdet.py`

Notiamo che questo ricevitore non ha uno schema a blocchi GRC.

Il ricevitore è stereo, quindi è più complicato da descrivere.

Utilizza come ingressi `demod_rate` e `audio_decimation`, definisce:

`low_freq = -125e3/ demod_rate`

`high_freq = 125e3/ demod_rate`

```
audio_rate = demod_rate / audio_decimation
```

1. Effettua la demodulazione FM utilizzando `fmdet_cf` in luogo di `quadrature_demod_cf` (non è chiaro il motivo).

2. Analizzo `fmdet_cf_impl.cc` (spostare a dopo). Non è chiarissimo cosa fa (il codice è più complicato. Secondo il manuale Doxygen implementa un I/Q slope detector (con limiting). E' un demodulatore basato su una tecnica analogica, non si capisce il vantaggio rispetto a quello precedente (anzi dovrebbe essere peggio). Vediamo i passaggi successivi, dando per buono che questo blocco recupera (a meno di costanti moltiplicative) il multiplex stereo $m(n)$

3. Setta i parametri del filtro audio. Simile al progetto precedente, ma qui la frequenza di taglio è fissata a 15 kHz

4. Setta i parametri di un filtro per il recupero della portante a 19 KHz. Utilizza `firdes.complex_band_pass` con guadagno 10, frequenze inferiore -19020 e superiore -18980, una banda di transisione pari a `audio_rate/32`

5. Setta i parametri di un filtro per il recupero della componente L-R. Utilizza `firdes.complex_band_pass` con guadagno 20, frequenze inferiore $38000 - 15000/2$ e $38000 + 15000/2$, stessa banda di transizione di prima

6. I guadagni dei filtri sono strani, servono a compensare il fatto che (credo) faccia la deenfasi sul multiplex stereo (sbagliato). Inoltre le frequenze di taglio del filtro per L-R sembrano anch'esse sbagliate.

7. Setta anche i paremtri di un filtro per recuperare il segnale RDS (ma non viene poi elaborato, finisce in un null sink). In ogni caso i parametri sono: guadagno 30, frequenze 57000 - 1500, 57000 + 1500, solita banda di transizione.

8. Sospetto che mantenga sempre uguale la banda di transizione per avere tutti filtri della stessa lunghezza.

9. Definisce i parametri di un PLL, come segue:

   ```
   loop_bw = 2 * math.pi/100.0
   max_freq = -2.0*math.pi*18990/audio_rate
   min_freq = -2.0*math.pi*19010/audio_rate
   ```

   Lavora ad `audio_rate`?

10. Per il resto si può vedere lo schema a blocchi (da mettere in pulito). Ha un senso anche se è poco elegante. Inoltre c'è il problema che la portante a 38 kHz non ha la corretta relazione di fase con quella a 19 kHz.

Ricevitore `wfm_rcv_pll.py`

Anche questo ricevitore è stereo, ed è quello implementato nel blocco GRC. In questo caso utilizza due PLL: uno per la demodulazione FM, l'altro per il rcupero della pilota a 19 KHz e successive eleborazioni. Il progetto è molto simile a quello precedente, ma invece dello slope detector utilizza `pll_freqdet_cf` con i seguenti parametri:

```
loop_bw = 2*math.pi/100.0
```

```
max_freq = 2.0*math.pi*90e3/demod_rate
```

Per il resto è esattamente lo stesso.

Implementando la mia versione del ricevitore con `pll_freqdet_cf` ho riscontrato che la banda da impostare nel blocco deve essere abbastanza più grande rispetto ai valori suggeriti da GNURadio. Un valore di `2 pi/10` è adeguato, inoltre il fattore di scala dell'uscita dev'essere 0.5 (non si capisce perché, bisogna vedere meglio il codice).

1.19 Studio del PLL di GNURadio

1.19.1 `control_loop.cc`

Ingresso: `loop_bw max_freq min_freq`

Dal codice: `d_damping = sqrtf(2.0f)/2.0f;` (damping factor)

```
float denom = (1.0 + 2.0*d_damping*d_loop_bw + d_loop_bw*d_loop_bw);
d_alpha = (4*d_damping*d_loop_bw) / denom;
d_beta = (4*d_loop_bw*d_loop_bw) / denom;
```

```
void
    control_loop::phase_wrap()
    {
      while(d_phase>M_TWOPI)
        d_phase -= M_TWOPI;
      while(d_phase<-M_TWOPI)
        d_phase += M_TWOPI;
}
```

```
void
    control_loop::frequency_limit()
    {
      if(d_freq > d_max_freq)
        d_freq = d_max_freq;
      else if(d_freq < d_min_freq)
```

```
        d_freq = d_min_freq;
}


void
    control_loop::set_frequency(float freq)
    {
      if(freq > d_max_freq)
        d_freq = d_min_freq;
      else if(freq < d_min_freq)
        d_freq = d_max_freq;
      else
        d_freq = freq;
}
```

QUesta sembra strana!!!

```
void
    control_loop::advance_loop(float error)
    {
      d_freq = d_freq + d_beta * error;
      d_phase = d_phase + d_freq + d_alpha * error;
}
```

1.19.2 pll_freqdet_cf_impl.cc

```
float
    pll_freqdet_cf_impl::phase_detector(gr_complex sample, float ref_phase)
    {
      float sample_phase;
      sample_phase = gr::fast_atan2f(sample.imag(), sample.real());
      return mod_2pi(sample_phase - ref_phase);
}


int
    pll_freqdet_cf_impl::work(int noutput_items,
      gr_vector_const_void_star &input_items,
      gr_vector_void_star &output_items)
    {
      const gr_complex *iptr = (gr_complex*)input_items[0];
      float *optr = (float*)output_items[0];

      float error;
```

```
      int size = noutput_items;

      while(size-- > 0) {
   *optr++ = d_freq;

   error = phase_detector(*iptr++, d_phase);

   advance_loop(error);
   phase_wrap();
   frequency_limit();
      }
      return noutput_items;
}
```

Vedere meglio implemnentazione che fa Michael Rice e le note di Rondeau.

1.19.3 pll_refout_cc_impl.cc

I blocchi prima sono gli stessi.

```
int
    pll_refout_cc_impl::work(int noutput_items,
     gr_vector_const_void_star &input_items,
     gr_vector_void_star &output_items)
    {
      const gr_complex *iptr = (gr_complex*)input_items[0];
      gr_complex *optr = (gr_complex*)output_items[0];

      float error;
      float t_imag, t_real;
      int size = noutput_items;

      while(size-- > 0) {
gr::sincosf(d_phase,&t_imag,&t_real);
*optr++ = gr_complex(t_real,t_imag);

error = phase_detector(*iptr++,d_phase);

advance_loop(error);
phase_wrap();
frequency_limit();
      }
      return noutput_items;
```

```
}
```

pll_carriertracking_cc_impl.cc

```
int
    pll_carriertracking_cc_impl::work(int noutput_items,
      gr_vector_const_void_star &input_items,
      gr_vector_void_star &output_items)
    {
      const gr_complex *iptr = (gr_complex*)input_items[0];
      gr_complex *optr = (gr_complex*)output_items[0];

      float error;
      float t_imag, t_real;

      for(int i = 0; i < noutput_items; i++) {
gr::sincosf(d_phase, &t_imag, &t_real);
optr[i] = iptr[i] * gr_complex(t_real, -t_imag);

error = phase_detector(iptr[i], d_phase);

advance_loop(error);
phase_wrap();
frequency_limit();

d_locksig = d_locksig * (1.0 - d_alpha) + \
  d_alpha*(iptr[i].real() * t_real + iptr[i].imag() * t_imag);

if((d_squelch_enable) && !lock_detector())
  optr[i] = 0;
      }
      return noutput_items;
}
```

1.20 Throttle con blocchi audio

Ho riscontrato un differente comportamento di GNURadio con riferimento allo schema semplice

Wav file source -> Throttle -> Audio Sink

GNURadio lancia un warning che il blocco throttle non ci vuole. Tuttavia sotto Windows se non lo metto, la riproduzione avviene ad una velocità sbagliata. Se la metto, è choppy. Sotto Linux/Ubunt questo non succede /versione di GNURadio companion 3.7.13.4)

1.21 Trasmissione di file di testo

Ho costruito dei flowgraph che trasmettono/ricevono utilizzando modulazione GFSK.

Per ricevere da un UDP Sink aprire (Ubuntu) una finestra di terminale e lanciare

```
nc -ul 1234
```

Al posto di nc posso usare ncat (della suite nmap, per WIndows) che ha una sintassi leggermente diversa:

```
ncat -lu -p 4321
```

L'esperimento funziona (con qualche tuning) sia trasmettendo dal DELL con la USRP, sia ricevendo. Bisogna indagare meglio come parametrizzare in termini di bit rate, samples per symbol, etc.

NB: cambiare porta, a volte si imballa con una porta e non funziona più...

Devo capire bene come organizzare le cartelle per evitare duplicazioni selvagge di file tra trasmettitore e ricevitore.

1.22 Alcuni comandi per audio/video streaming

`ffmpeg -i input.wav -b:a 128000 output.ts` per settare il bit rate nella conversione

`ffplay -i udp://localhost:1234` per mandare in streaming (audio/video) da una connessione UDP. A volte bisogna far partire prima `ffplay` altrimenti non riesce a sincronizzarsi con lo streaming

Con vlc (Ubuntu) lo streaming funziona ma occorrono un paio di tweak:

1. L'indirizzo da immettere deve essere nella forma `udp://@:1234` (se si usa `udp://127.0.0.1:1234` non funziona
2. Bisogna settare la dimensione dei pacchetti UDP più piccola (sembra che il massimo sia 1316), anche sotto Windows

Per lanciare uno streaming si usa `ffmpeg` con varie opzioni:

`ffmpeg -i Crosseyed_and_Painless.wav -tune zerolatency -b:a 64k -f mpegts udp://127.0.0.1:1234`

Il prblema è che al momento con Windows non riesco a tempificare la trasmissione, finisce tutto troppo presto (non ho verificato con Ubuntu che cosa succede). Anche i loopback funzionano male. Il problema delo streaming da/verso GNURadio è ancora tutto da risolvere.

1.23 Packet vs. unpacked byte

Let's take a moment to explore the concept of a packed or unpacked byte. An unpacked byte indicates its value in the least-significant bit. All other bits are ignored. On the other hand, in a packed byte, each bit is significant and represents an individual bit. For instance, the bit stream `1 0 1` could be expressed in the following two ways.

Unpacked: `00000001  00000000  00000001`

Packed: `00000101`

Nota che queste sono rappresentazioni LSB. Invece GNURadio utilizza anche rappresentazioni MSB. ...

1.24 Pulse shaping in GNURadio

Nella realizzazione di filtri pulse shaping con `firdes.root_raised_cosine` ho verificato che esistono combinazioni tra `sps` e `span` (da approfondire meglio, magari confrontando con Matlab) che portano a filtri strani (che funzionano malissimo, come se non rispettassero la condizione di Nyquist). Mi pare di ricordare che qualcosa del genere accadesse anche sotto Matlab, ma non sono riuscito a trovare una mia documentazione.

Ad esempio (con un loopback):

`sps = 8, span = 16` va bene

`sps = 8, span = 18` è catastrofico

In generale le combinazioni con `span` multiplo di `sps` sembrano più robuste.

Attenzione che non si vede dalla costellazione, ma guardando i dati demodulati. Potrebbe esserci uno shift dei bit/byte? Cioè potrebbe solo essere un problema di impacchettamento dei bit e non di forme d'onda [SOLVED].

NB: ho verificato introducendo un ritardo nel flowgraph (vedi `Packet_QPSK_simple_vector_v1.grc` Sembra essere un problema di impacchettamento dei bit nei byte. Evidentemente i bit di differenti byte vengono inseriti dal ricevitore. E'chiaramente un problema di framing che deve

essere risolto per una trasmissione affidabile introducendo degli header, altrimenti l'unica soluzione è aggiustare con un ritardo a mano (cosa che ho previsto nel flowgraph).

NB2: in ogni caso, alcuni valori di `span` (in particolare dispari) vanno evitati, in questo caso ho un problema a livello di forma d'onda.

E' interessante fare un confronto (utilizzando magari direttamente python) tra i risultati di firdes e quelli che si possono ottenere mediante Matlab (rcosdesign).

1.25 Impiego di una libreria custom con blocchi gerarchici

Per realizzare dei blocchi custom (assemblando blocchi GNU Radio) bisogna costruire dei flowgraph indicando nel blocco Options la categoria Hier Block ed indicando una categoria tra parentesi quadre (es. [CW library]). Poi basta seguire i seguenti passi:

1. aprire con GNU Radio Companion i blocchi creati
2. compilarli (Generate)
3. fare un refresh della libreria dei blocchi (tasto Reload Blocks)
4. deve comparire nella finestra di destra la libreria [CW library] con i blocchi creati che possono essere utilizzati nei propri flowgraph

Capitolo 2

Esperimenti con Raspberry

2.0.1 Installazione RTL-SDR

Ho già configurato Raspberry, posso accedere da remoto utilizzando Putty (Raspberry si collega via wireless e posso individuarne l'indirizzo acccedendo alla pagina di controllo del modem/router).

Nelle versioni più recenti di raspbian (il sistema operativo di Raspberry basato su Debian) è semplice installare rtl-sdr come package

```
sudo apt install rtl-sdr
```

In versioni più vecchie bisogna compilarlo dalla sorgente. La nuova installazione dovrebbe anche creare il blacklist del driver DVB-T originale (vedi istruzioni per installare RTL-SDR sotto Linux).

Dopo l'installazione fare il reboot con

```
sudo reboot
```

per abilitare il blacklist del driver DVB-T (credo).

Inserisco la chiavetta e lancio per la verifica il comando

```
rtl_test
```

che restituisce un output significativo se la chiavetta viene rilevata.

2.0.2 Installazione gqrx

Per avere un output grafico posso utilizzare `VNV Viewer` che ho già configurato.

Provo ad installare gqrx in Raspberry. Anche in questo caso posso utilizzare la versione packaged

```
sudo apt install gqrx-sdr
```

Installa molta roba (oltre 400 MB) quindi ci vuole pazienza. Al sito `https://drwxr.org/2017/04/setting-up-rtl-sdr-on-raspberry-pi-3/` è descritta l'installazione in dettaglio ed una serie di trucchi per la configurazione. In realtà installa gnuradio per intero (l'unica cosa che non installa, credo, è GRC companion).

In realtà da un errore. Allora seguo la procedura descritta qui `http://gqrx.dk/download/gqrx-sdr-for-the-raspberry-pi`.

Funziona. Se collego una cuffia all'uscita audio del Raspberry riesco anche a sentire la radio FM (anche se l'audio è choppy).

Per settare una risoluzione dello schermo ragionevole, procedere modificando il file /boot/config.txt

settando

```
framebuffer_width=1920
```

```
framebuffer_height=1280
```

Oppure quello che si ritiene.

2.0.3 Installazione osmosdr

Nell'installazione di gqrx credo installi anche gnuradio companion (compare addirittura nella lista delle applicazioni). Tuttavia per avere anche i blocchi sink RTL-SDR biogna installare osmosdr:

```
sudo apt install gr-osmosdr
```

2.0.4 Prova

Un semplice flowgraph di prova funziona,senza neanche appesantire troppo la CPU Nell'avvio GNUradio da una diagnostica che però non ne preclude il funzionamento

```
*** Error in ‘/usr/bin/python’: corrupted double-linked list: 0x01233ad8 ***
```

Invece se lancio da linea di comando

```
osmocom_fft
```

L'errore precedente si presenta e ne preclude il funzionamento. In realtà sembra solo un problema di prestazioni, lanciando più volte ilcomando dà errori sempre divdersi (a volte anche segmentation fault) ed infine sono riuscito a farlo partire, anche se freeeza

```
osmocom_fft -s 1.2e6
```

2.0.5 Osservazioni

Ho provato a reinstallare tutto, ed adessoil flowgraph non funziona più. Sembra ci sia qualche incompatibilità. In realtà è un problema di CPU run-time, se si riduce ad esempio la frequenza di campionamento il sistema sembra funzionare. In ogni caso, fornisce degli errori abbastanza random.

Per un riferimento si veda qui `https://www.rs-online.com/designspark/taking-the-raspberry-pi-2-for-a`

Appendice A

Configurazione base di Raspberry PI

A.1 Installazione sistema operativo

Nel seguito si fa riferimento ad un'installazione senza monitor o tastiera (headless) che può essere un po' più complicata. SI assume che il RPI si colleghi ad un router con DHCP abilitato (altrimenti è più complicato nel caso si debba settare un IP statico).

1. Scaricare da `http://www.rasperrypi.org` il SO raspbian (formato zip -> img)
2. Utilizzare (su Windows) Win32DiskImager per copiare il SO sulla SD card (NB: la partizione creata è grande solo per contenere il sistema operativo) -> non è così, superata dalle nuove versioni)
3. Inserire la SD Card nel RPI
4. Collegare il cavo Ethernet ed il cavo di alimentazione ed accendere RPI
5. Installare zenmap (un porting di NMAP per windows) per cercare il device sulla rete; in realtà è possibile anche entrare nella pagina web del router ed individuare l'ndirizzo IP che è stato assegnato -> 192.168.1.98 -> flaggando lì'opzione "aggiungi dispoditivonel DHCP statico" faccio in modo che nelle successive comnnessioni l'indirizzo non cambia

   con nmap -> utilizzare il comando `nmap -T4 -F 192.168.1.*`
6. Lanciare putty per accedere a RPI via SSH

   Nota: come documentato su Internet, a partire da novembre 2016 RPI ha SSH disabilitatoùdi default -> per abilitarlo è necessario copiare un file ssh (vuoto) nella directory di boot della SD -> funziona
7. I parametri di Putty sono indirizzo IP, porto 22, SSH, username: pi, password: raspberry -> OK funziona

8. Lanciare raspi-config per configurare una serie di cose
   sudo raspi-config
9. Spegnere la board sempre da software
   sudo reboot per reboot
   sudo shutdown -h per shutdown

A.2 Abilitare accesso remoto con desktop

1. Aprire raspi-config
2. Abilitare "Interfacing options" -> P3 VNC (?)
3. Effettuare un reboot -> sudo reboot
4. Successivamente è possibile connettersi al desktop da remoto utilizzando VNC Viewer (o simili) e le credenziali

Bibliografia

[1] https://archive.fosdem.org/2014/schedule/track/software_defined_radio/

[2] https://archive.fosdem.org/2015/schedule/track/software_defined_radio/

[3] https://archive.fosdem.org/2016/schedule/track/software_defined_radio/

[4] https://archive.fosdem.org/2017/schedule/track/software_defined_radio/

[5] https://archive.fosdem.org/2017/schedule/track/software_defined_radio/

[6] https://archive.fosdem.org/2018/schedule/track/software_defined_radio/

[7] https://fosdem.org/2019/schedule/track/free_software_radio/

[8] http://www.trondeau.com/

[9] https://www.gnuradio.org/grcon/grcon18/

[10] https://www.youtube.com/channel/UCceoapZVEDCQ4s8y16M7Fng

[11] NI USRP - 29xx - Getting Started Guide