**UNIVERSITÀ DEGLI STUDI DI TRIESTE**

**FACOLTA' DI INGEGNERIA**
**DIPARTIMENTO DI ELETTROTECNICA ELETTRONICA ED INFORMATICA**

**CORSO DI LAUREA IN INGEGNERIA DELLE TELECOMUNICAZIONI**

TESI DI LAUREA IN

*ELETTRONICA 3: DSP E MICROCONTROLLORI*

# SVILUPPO ED IMPLEMENTAZIONE SU DSP DI UN DEMODULATORE NUMERICO IN BANDA BASE

CANDIDATO
Stefano de Fabris

RELATORE
Sergio Carrato

Anno Accademico 2004/2005

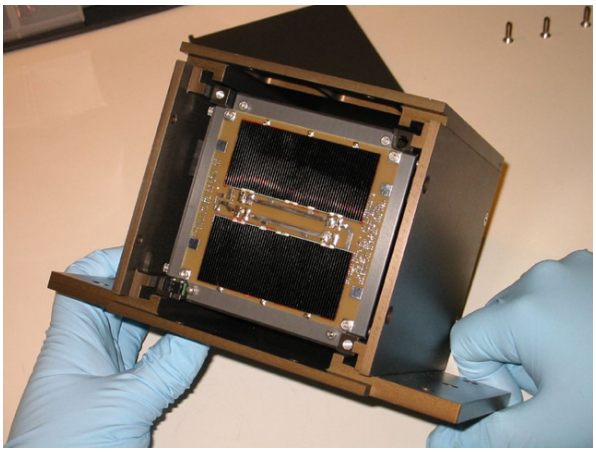
# CAPITOLO 1: introduzione

La tesi ha come obiettivo quello di sviluppare un software da applicare ad un DSP (digital signal processor) che provveda alla modulazione e demodulazione in banda base.

I requisiti di progetto sono:

- Buone prestazioni, in particolare in caso di basso rapporto segnale/rumore della portante.
- Velocità di almeno 1200baud.
- Ampiezza di banda utilizzata inferiore ai 15 kHz.
- Consumo energetico minore possibile.

Il punto relativo alle prestazioni, ovvero una bassa probabilità di errore (Pe) anche per segnali deboli è vincolato all'applicazione del progetto, ovvero quello di ricetrasmettitore per il satellite AtmoCube, in fase di progettazione da parte dell'Università. Considerando che la massa totale del satellite dovrebbe essere intorno al chilogrammo, una delle conseguenze è che la sua produzione energetica da parte dei pannelli solari estremamente limitata. Questo ha vincolato la scelta del DSP ad un modello a basso consumo.



CubeSat della California Polytechnic State University

La velocità di trasmissione è stata calcolata stimando la mole di dati da trasmettere per ogni "passaggio" del satellite sopra l'orizzonte. Infatti, dal momento che il satellite viaggia in un'orbita bassa (sotto i 1000Km di quota) esso non è visibile in un punto fisso del cielo come i satelliti geostazionari, ma è necessario "inseguirlo" lungo l'orbita percorsa. Nel corso della tesi si cercherà di dare una soluzione al problema rispettando tutti i vincoli progettuali qui elencati.

# CAPITOLO 2: definizione dello schema realizzativo

In questo capitolo verranno studiate le possibili soluzioni al problema, le quali sono legate tra loro in uno stretto equilibrio. In particolare verranno studiati:

- La modulazione da adottare in funzione dei vincoli di velocità imposti
- L'algoritmo che implementa la modulazione e la demodulazione secondo un determinato schema
- La complessità computazionale dell'algoritmo e relativa scelta del DSP da utilizzare
- La verifica dei risultati ottenuti con le applicazioni "commerciali" equivalenti.

Per quanto riguarda la velocità di trasmissione è parso subito chiaro che non sarebbe stato un problema vincolare l'ampiezza di banda richiesta per il trasferimento dati con l'ampiezza di banda del complesso ricevitore. Pertanto la scelta possibile si riduce a tre possibili modulazioni:

- FSK, variazione della frequenza della portante in funzione del dato trasmesso
- ASK, variazione dell'ampiezza della portante in funzione del dato trasmesso
- PSK, variazione della fase della portante in funzione del dato trasmesso

Nel caso di una modulazione binaria (ovvero trasmettendo un bit per volta) la 2-ASK e la 2-PSK coincidono, portando la scelta a sole due modulazioni possibili.
Nonostante la modulazione di frequenza sia molto utilizzata per la diffusione (broadcast) del servizio radiofonico a motivo della sua elevata resistenza al rumore, questa modulazione si presta poco alla trasmissione di dati a causa dell'elevata energia richiesta per la trasmissione (cfr. Appunti del corso di Trasmissione Numerica). Tuttavia la modulazione di frequenza presenta indubbi vantaggi riguardo la resistenza a variazioni di ampiezza, molto comuni in qualunque trasmissione radio. C'è da notare inoltre che il passaggio del segnale attraverso la ionosfera aumenta questo fenomeno in quanto va ad attenuare in modo variabile e non prevedibile il segnale (fenomeno del fading). Per questo motivo la modulazione d'ampiezza non viene utilizzata in nessuna trasmissione radio a lunga distanza. Un ulteriore problema della comunicazione con mezzi mobili è quello della variazione di fase, dovuta all'effetto doppler che si verifica tra qualunque mezzo in movimento. Appurato che il satellite si muove a grande velocità rispetto all'osservatore e che questa velocità non è per nulla costante, il rischio è che un effetto doppler marcato renda impossibile una trasmissione di tipo PSK, a meno di non adottare una codifica di tipo differenziale che però fa

degradare le prestazioni di 3 dB circa (valore teorico, cfr. Appunti del corso di Trasmissione Numerica).

Si sceglie così, anche per "similitudine" con i già esistenti satelliti Cubesat, di implementare una modulazione FSK binaria.

A questo punto del progetto si è reso necessario operare una ricerca sull'algoritmo realizzativo per il modulatore ed il demodulatore. Per quanto riguarda il modulatore si è scelto lo schema a doppia frequenza: 1200-2400Hz.

A questo punto si è dovuto operare una scelta relativa all'algoritmo di demodulazione, in particolare sono stati presi in considerazione due algoritmi commerciali:

- Demodulazione I/Q
- Demodulazione asincrona

Lo schema a blocchi di un demodulatore I/Q si trova in figura 1:

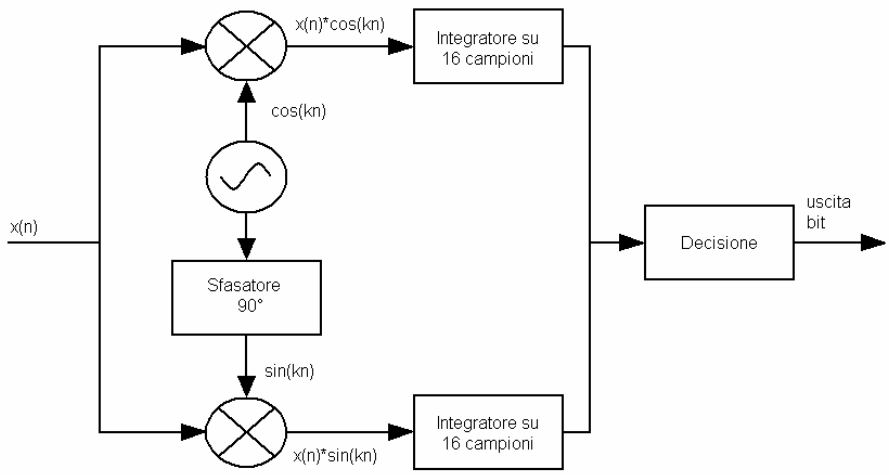



**Figura 1**

Schema a blocchi di un demodulatore I/Q

Il demodulatore si basa sul principio che moltiplicando la portante per la frequenza intermedia in fase o in quadratura ed integrando sul periodo di segnalazione è possibile discriminare la frequenza del segnale trasmesso.

Lo schema a blocchi di un demodulatore asincrono è raffigurato in figura 2:

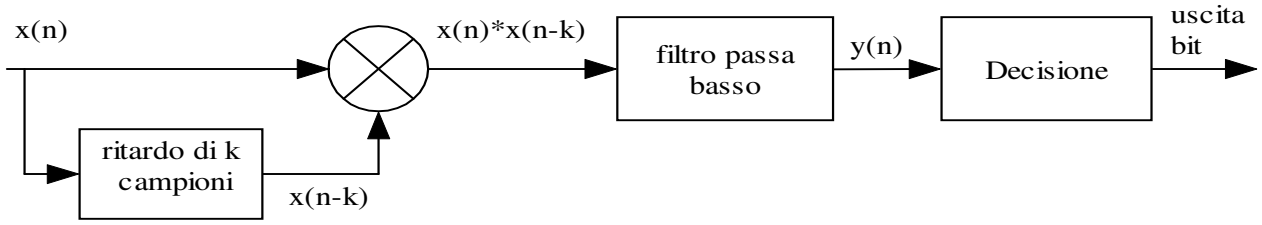



**Figura 2**

Schema a blocchi di un demodulatore asincrono

Dimostriamo il principio di demodulazione di quest'ultimo. Sia

$$x(t) = A\sin(2\pi ft)$$

dove A è l'ampiezza del segnale ricevuto ed f è la frequenza che nel nostro caso può essere 1200 o 2400 Hz. Da questa ricaviamo il segnale campionato sostituendo t con $nT_s$:

$$x(n) = A\sin(2\pi fnT_s)$$

Dove $n \in N$ è il numero del campione e $T_s$ è il periodo di campionamento.

Da questa si ricava facilmente che:

$$x(n-k) = A\sin(2\pi f(n-k)T_s)$$

Faccio il prodotto dei due segnali:

$$x(n)x(n-k) = A^2\sin(2\pi fnT_s)\sin(2\pi f(n-k)T_s) =$$

Per le formule di Werner:

$$= (A^2/2)\cos(2\pi fkT_s) - (A^2/2)\cos(4\pi fnT_s - 2\pi fkT_s)$$

Il primo termine dell'equazione è funzione della sola frequenza f, del ritardo k, dell'ampiezza A e non dipende da n. Il secondo termine ha una frequenza doppia del primo e può essere eliminato mediante un filtro passa-basso.

All'uscita del filtro si ha quindi, in prima approssimazione:

$$y(n) = (A^2/2)\cos(2\pi fkT_s) \qquad (1)$$

Con k e $T_s$ fissi e f che può assumere soltanto due valori distinti.

Posto $f_0$=1200Hz e $f_1$=2400Hz si sceglie k in modo da massimizzare la differenza:

$$\cos(2\pi f_0 kT_s) - \cos(2\pi f_1 kT_s)$$

A questo punto è stato necessario operare una scelta sul valore di k che appare chiaro essere funzione anche di $T_s$. Poiché nella demodulazione I/Q si è scelto, data la gamma di frequenze di campionamento dei convertitori A/D e D/A utilizzati, un valore di finestra di 16 campioni e si vuole mantenere la compatibilità con i due sistemi di demodulazione si sceglie anche qui una frequenza di

campionamento $f_s$ pari al prodotto della frequenza di simbolo per il numero di campioni per simbolo: $f_s = 1200 baud * 16 campioni = 19200 Hz$ da cui ricaviamo $T_s = 1/f_c$ .

Per trovare il massimo è sufficiente fare la derivata rispetto a k della formula precedente e porla uguale a zero:

$$-\sin(2\pi f_0 k T_s) + \sin(2\pi f_1 k T_s) = 0$$

$$-\sin(\frac{\pi}{8}k) + \sin(\frac{\pi}{4}k) = 0$$

Ottengo che l'equazione si annulla per k=0,8,16. Devo discriminare quali siano i massimi ed i minimi sostituendo nell'equazione (1). Risultano essere minimi i valori 0 e 16, mentre risulta essere massimo il valore 8. La differenza risulta essere uguale a 2, la massima possibile.

# CAPITOLO 3: modellizzazione in Matlab

Si rende indispensabile, vista la complessità del problema, andare a sperimentare l'algoritmo di demodulazione in un ambiente di sviluppo prima di implementarlo. Questa fase metterà in luce eventuali problemi fin qui ignorati che potrebbero presentarsi in fase realizzativa.

## Descrizione dell'ambiente di sviluppo e prima soluzione del problema

Il software MATLAB versione 6.1 è un ambiente di sviluppo che permette di modellizzare un problema con l'ausilio di innumerevoli librerie già pronte che semplificano notevolmente il lavoro del programmatore. Il linguaggio di programmazione è proprietario, ma si nota che deriva essenzialmente dal C e presenta molte analogie dal punto di vista sintattico.

Per una soluzione più semplice e mirata dei diversi problemi che lo sviluppo del demodulatore comporta si decide di scomporre il progetto in più parti, ognuna delle quali può essere sviluppata separatamente:

- Generatore di bit casuali
- Modulatore
- Simulatore di canale (aggiunta rumore ed effetto doppler)
- Demodulatore
- Controllo dei risultati ottenuti e degli errori commessi

Si decide di iniziare a sviluppare il problema in modo da semplificare a massimo il programma ponendo i seguenti vincoli che verranno in seguito rimossi:

- I bit sono veramente casuali ($P_b$=0.5)
- Rumore gaussiano bianco, privo di effetto doppler
- Il segnale in ingresso è perfettamente sincronizzato, ossia il primo campione in ingresso al demodulatore corrisponde al primo campione emesso dal modulatore.

In appendice A si trova il listato dei quattro programmi: datagen.m, modulator.m(1), noisegen.m e demodulator(1).m

Il programma datagen.m, molto semplice, crea una array di lunghezza len di numeri casuali creati con distribuzione uniforme che poi vengono convertiti in dei bit casuali mediante confronto con il valore mediano 0,5.

Il programma modulator.m(1) effettua la modulazione vera e propria dei dati andando a generare per ogni bit sedici

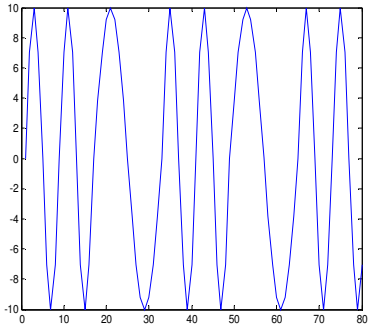



**Figura 3**

Portante modulata

campioni di una sinusoide a frequenza variabile: dato fondamentale è che tutti i campioni con indice multiplo di 16 siano zero. In questo modo si ha un segnale in uscita che non presenta discontinuità (in figura 3 il grafico dei campioni in uscita).

Il programma noisegen.h chiede all'utente quale sia il rapporto potenza di segnale su potenza di rumore desiderato in scala lineare. Dopodiché genera un vettore noise della stessa lunghezza della portante precedentemente generata mediante la funzione Matlab randn che genera numeri casuali con una distribuzione normale aventi media nulla. Quindi viene calcolata la potenza della portante (port) con quella del rumore appena generato mediante la funzione:

$$P = \lim_{N \to \infty} \frac{1}{2N+1} \sum_{n=-N}^{N} \left| x^2(n) \right|$$

Tuttavia dal momento che la lunghezza del segnale è finita ed N è noto ricaviamo che:

$$P = \frac{1}{N+1} \sum_{n-N}^{N} \left| x^2(n) \right|$$

Ma poiché questo vale sia per la portante che per il rumore e dobbiamo calcolare il loro rapporto, possiamo scrivere:

$$SNR = \frac{P_S}{P_N} = \frac{\frac{1}{N+1} \sum_{n=0}^{N} \left| x_S^2(n) \right|}{\frac{1}{N+1} \sum_{n=0}^{N} \left| x_N^2(n) \right|} = \frac{\sum_{n=0}^{N} \left| x_S^2(n) \right|}{\sum_{n=0}^{N} \left| x_N^2(n) \right|} = q$$

Il programma implementa questa funzione ottenendo così una variabile q con questo rapporto. Da notare che poiché vengono usati vettori reali non è necessario calcolare il modulo del valore al quadrato ma soltanto il valore al quadrato. A questo punto è necessario effettuare uno "scalaggio" del rumore in modo da generare una nuova portante con le caratteristiche di SNR volute. Per fare ciò è necessario ottenere due segnali con la stessa potenza di rumore e quindi dividere questa potenza per il valore scelto, moltiplicando ogni termine del vettore noise per un numero che sia la radice quadrata del rapporto trovato:

$$\frac{\sum_{n=0}^{N} x_S^2(n)}{\sum_{n=0}^{N} \left( x_N(n) * \sqrt{q} \right)^2} = 1$$

Se moltiplico entrambi i membri dell'equazione precedente per il SNR (SNR') voluto ottengo:

$$SNR' = \frac{\sum_{n=0}^{N} x_S^2(n)}{\sum_{n=0}^{N} \left(x_N(n) * \sqrt{q}\right)^2} \quad SNR' = \frac{\sum_{n=0}^{N} x_S^2(n)}{\sum_{n=0}^{N} \left(x_N(n) * \sqrt{q/SNR'}\right)^2}$$

A questo punto è sufficiente sommare i due vettori port e noise per ottenere il segnale desiderato.

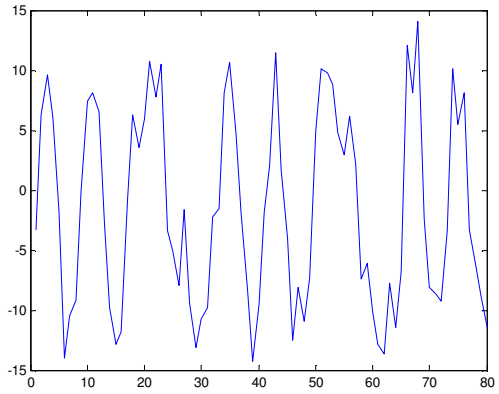


**Figura 4**

Segnale con SNR = 10dB

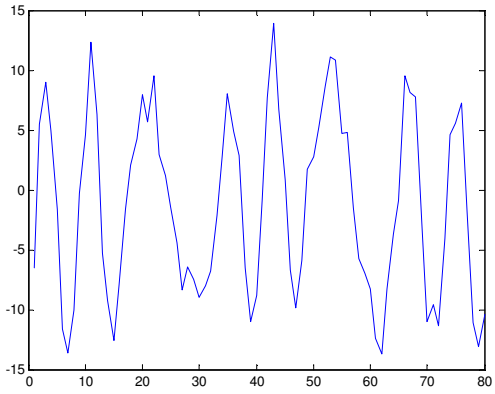


**Figura 5**

Segnale con SNR = 6dB

**In figura 4 e 5 possiamo vedere i grafici del segnale alla fine dell'elaborazione del segnale di figura 2 ottenuta con un SNR rispettivamente di 10 e 6dB.**

Il programma demodulator.m (2) implementa un demodulatore I/Q andando a moltiplicare l'ingresso per il valore della portante, integrandolo e confrontandolo con una soglia. Nel caso ideale (segnale in assenza di rumore) si ottengono, avendo trasmesso il simbolo corrispondente al valore 1, int1= 47,2760 e int2= 31,5888. Se invece il simbolo ricevuto corrisponde al valore 0, allora ottengo int1= -34,4324 e int2= -23,0070. Viene posta come soglia il valore mediano tra la somma dei due segnali, ovvero il valore 10,7.

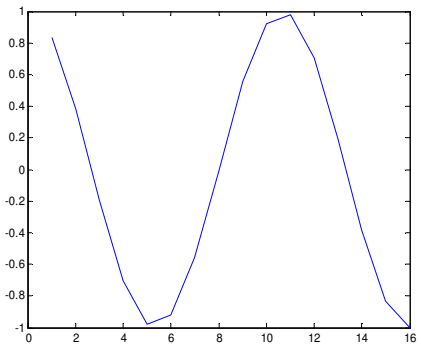


**Figura 6**

Segnale generato in fase

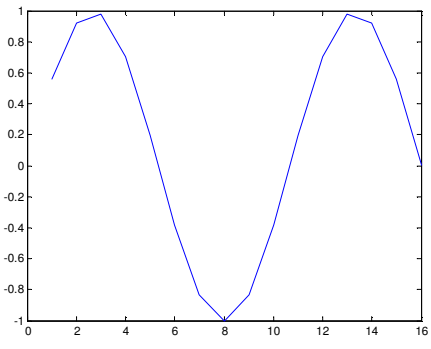


**Figura 7**

Segnale generato in quadratura

Nelle figure 6 e 7 il grafico della portante di demodulazione in fase e in quadratura usato dal demodulatore.

## Test dei risultati ottenuti

Per testare la bontà dell'algoritmo utilizzato si decide di implementare un programma di test il cui listato è in appendice 1 sotto il nome di Graph.m.

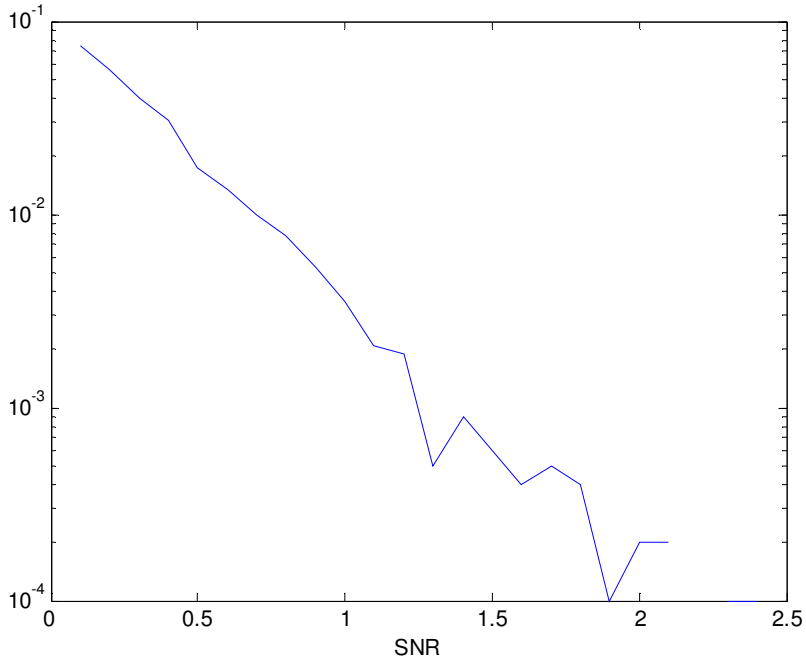



**Figura 8**

Prestazioni del demodulatore

Il programma Graph.h genera un grafico sulle prestazioni dell'algoritmo di demodulazione (come questo in figura 8); questo esegue una serie di prove partendo da un SNR = 0.1 per arrivare ad un rapporto SNR di 3 (da -10dB a 4,7dB). Il test è molto pesante dal punto di vista computazionale, infatti il test che ha portato al grafico qui riportato è stato fatto su un dato lungo 10.000 bit calcolato nell'intervallo di valori da 0.1 a 4 (da -10dB a 6dB) con step di 0,1 richiede circa 25 minuti su di un computer dotato di processore a 1,4Ghz.

Comunque si può vedere chiaramente che è possibile raggiungere un tasso d'errore sul bit nell'ordine di $10^{-3}$ (uguale a quello massimo utilizzabile dai sistemi GSM) con soli 1,3 dB circa di rapporto segnale-rumore.

A questo punto viene creato un nuovo file Demodulator.m (2) con l'implementazione del demodulatore asincrono. Dal listato si nota che il programma moltiplica un campione per quello otto posizioni precedenti. Quindi vengono calcolati i valori per un filtro di Butterworth con risposta in frequenza come da grafico (figura 11) e viene applicato il filtro all'uscita precedentemente creata (caso ideale senza rumore) per una sequenza in ingresso data=[1 1 1 1 0 0 1 1 0 1]:

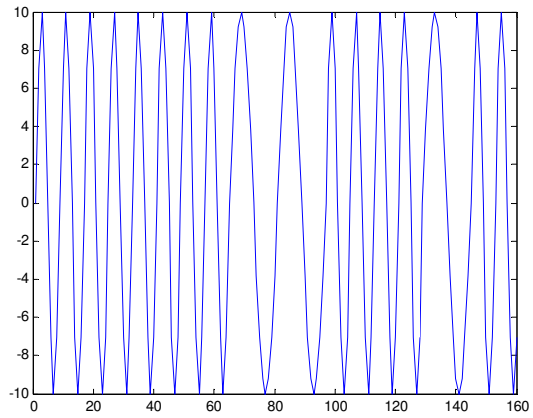



**Figura 9**

Portante generata dal modulatore

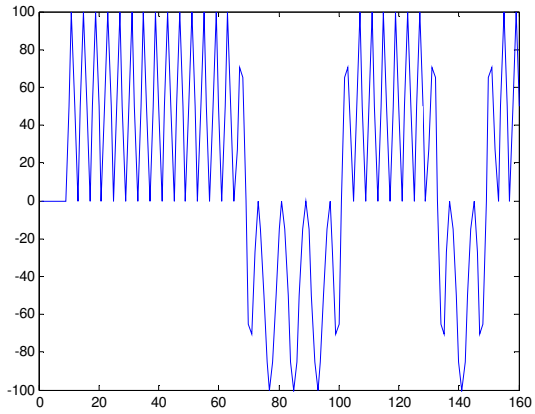



**Figura 10**

Segnale dopo il primo stadio di demodulazione

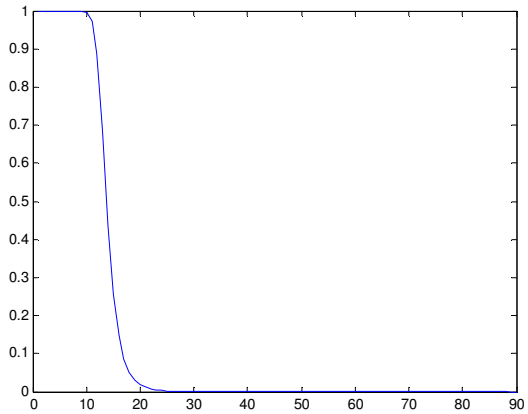



**Figura 11**

Risposta in frequenza del filtro di Butterworth

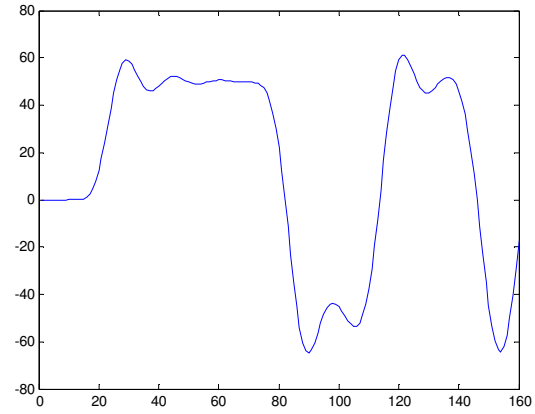



**Figura 12**

Risultato ottenuto dopo il filtraggio

A questo punto si apre un problema: quale algoritmo utilizzare per operare una decisione sul bit trasmesso. Se si osserva il grafico si capisce subito che nel caso in cui vengano trasmessi molti bit uguali consecutivi è molto difficile discriminare eventuali massimi locali della curva (in realtà il quarto massimo locale è visibile andando ad ingrandire molto il grafico). Non è possibile neppure associare il numero dei bit alla "distanza" tra due passaggi per lo zero in quanto, poiché il filtro ha

una fase non nulla, questa distanza non è proporzionale al numero di bit trasmessi. Infine il tutto viene disturbato dal rumore (qui non presente) e dal fatto che l'ampiezza del segnale varia nel tempo a causa del fading del canale radio. A questo punto sono possibili più soluzioni:

- Adottare una codifica di canale adeguata, ad esempio una codifica 4B5B oppure una 8B10B come quelle usate nello standard Ethernet (IEEE 802.3) che prevedono un numero massimo di simboli uguali consecutivi e un sistema di rilevazione degli errori di trasmissione.
- Utilizzare un sistema ibrido: la demodulazione asincrona può essere usata per recuperare il sincronismo e i dati vengono poi ricavati mediante demodulazione I/Q.

## Sviluppo di un algoritmo che ovvi alle limitazioni imposte

Si sceglie la seconda strada, in modo da abbinare gli ottimi risultati teorici ottenuti con la demodulazione I/Q ad un meccanismo di sincronizzazione. Si sceglie come trailer la sequenza di bit [1 0 1 0 1 0 1 1]. Al momento si continua a utilizzare un segnale non distorto dall'effetto doppler.

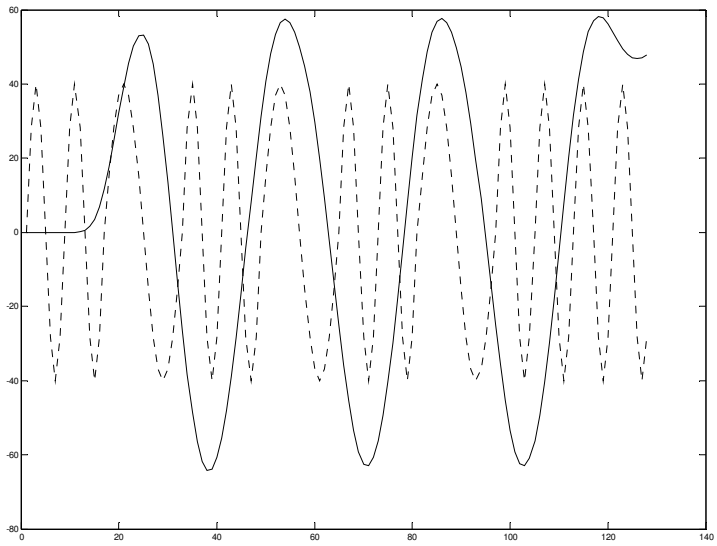



**Figura 13**

Sovrapposizione del grafico della portante e del segnale dopo il filtro:

Come si può vedere dal grafico in figura 13, i massimi locali del segnale demodulato (linea continua) si trovano, nei bit intermedi, 22 campioni dopo l'inizio del simbolo corrispondente sulla portante (linea tratteggiata). Viene quindi riscritto il programma modulatore, con l'aggiunta della possibilità di poter ritardare l'inizio della trasmissione di un numero casuale di campioni e con l'inserimento della sequenza trailer prima dei dati, in appendice sotto il nome di modulator.h (2).

Viene quindi riscritto il programma demodulatore, ricavato dalla fusione dei due programmi preesistenti, in appendice sotto il nome di demodulator.h (3).
Il programma esegue la demodulazione della portante modulata in tre passi fondamentali:

- Demodulazione asincrona della portante
- Ricerca della sequenza trailer
- Demodulazione I/Q dei dati utili

La demodulazione asincrona sfrutta l'algoritmo già implementato in precedenza.
La ricerca della sequenza trailer si basa sulla posizione media degli estremi locali. Per semplificare l'algoritmo viene per prima cosa calcolato il valore assoluto del segnale in uscita dal demodulatore asincrono. Quindi vengono ricercati i primi otto estremi locali e dalla loro posizione si ricava la posizione media d'inizio della sequenza. Prendiamo l'algoritmo:

```
pos=pos+(p-1)-(max*16)
```

La posizione del primo campione utile è data dalla posizione attuale all'interno dei valori campionati (p) meno 16 campioni per ogni estremo locale rilevato (max*16).
Una volta calcolato questo valore è sufficiente dividerlo per 8 e sottrarre il numero di campioni di ritardo fissi dovuti alla fase del filtro (da prove sperimentale si è ricavato il valore 8) che ha portato alla scrittura della riga

```
fi=round(pos/8)-8;
```

A questo punto viene riportato il codice relativo al demodulatore I/Q con l'aggiunta dello sfasamento. Ad esempio la linea:

```
s1=port(i*16+j)*cos((3/16)*pi*j);
```

Viene sostituita con

```
s1=port(i*16+j+fi)*cos((3/16)*pi*j);
```

## La soluzione all'effetto doppler

Poiché il demodulatore deve essere in grado di ricevere dati provenienti da un satellite, è prevedibile che il segnale sia soggetto ad un effetto doppler variabile nel tempo, ma che può essere considerato costante nell'intervallo di trasmissione di un pacchetto (nell'arco di pochi secondi la velocità relativa del satellite può essere ragionevolmente considerata costante).
Si procede quindi ad una stima dell'effetto doppler in Matlab, assumendo che in sua assenza la distanza tra gli estremi locali del segnale in uscita dal demodulatore asincrono sia costante. L'algoritmo si propone di fare alcune considerazioni sull'ultimo bit della sequenza trailer, in particolare sul passaggio per lo zero della portante.

In condizioni ideali il campione alla posizione fi+128 della sequenza precedente è uguale a zero; in caso sia di rumore, sia di effetto doppler il valore sarà diverso. In caso di rumore (a meno che non sia particolarmente forte) l'andamento della curva è ancora riconducibile a quello originario, ma in caso di effetto doppler l'andamento vedrà uno spostamento dell'intera sequenza di campioni. L'algoritmo qui implementato cerca di dare una soluzione al problema:

```
p=fi+125;
while (~(port(p-1)<0 & port(p)>=0)),
   p=p+1;
end;
p=(p-fi-128)/8;
l=floor((length(port)-fi)/16);
```

Mediante questo algoritmo possiamo trovare che ogni 128 campioni c'e' uno sfasamento di p-fi-128 campioni per effetto doppler. Dovrò quindi scalare fi di un fattore (p-fi-128)/128 per ogni simbolo, cioe' (p-fi-128)/8 per ogni finestra di integrazione. Quindi nel punto in cui si procede all'integrazione delle componenti I e Q del segnale le righe di codice interessate vengono modificate con:

```
s1=port(i*16+j+fi+round(p*i))*cos((3/16)*pi*j);
```

# CAPITOLO 4: sviluppo dell'algoritmo su DSP

L'algoritmo sin qui sviluppato in ambiente Matlab è pronto ad essere trasportato su DSP per un utilizzo pratico dello stesso. Viene scelto pertanto un processore commerciale prodotto dalla Texas Instrument, in particolare il modello C5402, appartenente alla famiglia di DSP a basso consumo, per rientrare nelle specifiche di progetto. Il processore è montato su una scheda dimostrativa denominata TMS320C5402DSK la quale permette di realizzare e testare l'algoritmo di demodulazione. L'ambiente di sviluppo utilizzato è Code Composer Studio 2.1 prodotto dalla Texas Instruments per i propri DSP prodotti. Il linguaggio di programmazione utilizzato è l'ANSI C con aggiunte specifiche per l'elaborazione di segnali fatte dalla Texas Instruments: si trovano ad esempio già implementate funzioni per il calcolo della FFT (trasformata veloce di Fourier) o della convoluzione.

Caratteristiche della scheda TMS320C5402DSK:

- Processore C5402 a basso consumo con unità logico aritmetica (ALU) in virgola fissa
- Convertitori analogico/digitale e digitale/analogico con frequenza massima di campionamento di 8000Hz
- Porta seriale standard (RS-232)
- 2 interruttori utilizzabili come ingresso, 3 led utilizzabili come uscita
- Slot di espansione per eventuali periferiche esterne

Inizialmente si è pensato di utilizzare un convertitore A/D molto veloce: l'ADS8361. In questo modo si sarebbe potuto risparmiare la sezione audio in banda base, lavorando direttamente in media frequenza. Tuttavia questo si è dimostrato inadeguato per due motivi:

- Il suo elevato consumo energetico (circa ½ Watt)
- La grossa potenza di calcolo necessaria a sopportare la mole di dati prodotta (10 Mbps) avrebbe richiesto una programmazione in linguaggio assembler di buona parte del codice, non avendo comunque garanzia del risultato. Si pensi che solo per leggere i dati dal campionatore nel programma scritto in C che la Texas Instrument porta come esempio di utilizzo si va ad usare la maggior parte delle risorse della CPU.

Poiché la frequenza di campionamento del convertitore A/D e D/A a bordo della scheda TMS320 è in ogni caso troppo bassa (per la bitrate richiesta è necessaria una frequenza di campionamento di almeno 19200Hz) si decide di collegare sullo slot di espansione la PCM3003 Audio Daughter Card, una scheda che monta una coppia di convertitori A/D e D/A operanti fino alla frequenza massima di 44100Hz.

Come punto di partenza per i programmi di modulazione e demodulazione che si andranno a scrivere si prende l'esempio scritto dal prof. Richard Sikora che contiene già il software che permette di interfacciare la PCM3003 alla scheda DSK.

## Scrittura del protocollo di comunicazione seriale

Per facilitare il lavoro della coppia modulatore/demodulatore che si trovano a condividere il canale di comunicazione è necessario che la comunicazione avvenga a pacchetti.

Come base viene scelto il protocollo di trasmissione seriale SLIP (serial line IP, RFC 1055) il quale, benché oramai obsoleto e non più utilizzato, è estremamente semplice e più che adeguato allo scopo.

Il protocollo SLIP viene ulteriormente modificato in modo da semplificare la fase di debug dei programmi modulatore e demodulatore. Alla fine il protocollo viene così definito:

- Un pacchetto inizia con il carattere speciale '{' (codice ASCII 123); eventuali caratteri inviati prima di questo dovranno venire ignorati
- Se in un pacchetto ricompare il carattere ASCII 123 esso dovrà essere considerato come carattere valido e quindi trasmesso con la sequenza binaria corrispondente.
- Il carattere di termine del pacchetto è '}' (codice ASCII 125) seguito da un carattere diverso. Se il messaggio originario contiene il carattere ASCII 125, in fase di trasmissione deve venire trasmesso due volte. In questo modo è possibile rilevare che il carattere trasmesso non indica la fine del pacchetto.

Esempio: la sequenza "123{abcd}}ef}ghi" in ingresso alla porta seriale del DSP, dopo le procedure di modulazione e demodulazione, viene ricevuta come "{abcd}}ef}g". Il terminale che riceve questa sequenza la dovrà interpretare come "abcd}ef".

## Problema del filtraggio

Il problema più grosso nello sviluppo di tutto l'algoritmo di modulazione/demodulazione è senz'altro la realizzazione del filtro di Butterworth su di un processore in virgola fissa. Mentre nell'ambiente di sviluppo Matlab non avevamo problemi di velocità di elaborazione, su DSP il filtraggio deve essere svolto in tempo reale e, dal momento che il processore utilizzato dispone soltanto di ALU in virgola fissa, non è possibile utilizzare l'emulazione software della virgola mobile essendo questa troppo pesante dal punto di vista computazionale. Pertanto viene dapprima sviluppato separatamente un filtro digitale che verrà poi inserito nel programma definitivo.

Un generico filtro IIR (infinite input response) è calcolato mediante la seguente formula:

a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb) - a(2)*y(n-1) - ... - a(na+1)*y(n-na)

Nel caso del filtro che dobbiamo implementare il coefficienti di a e di b sono:

a = 1.0000 -3.6476 5.4605 -4.1694 1.6181 -0.2547

b = 0.0002 0.0011 0.0022 0.0022 0.0011 0.0002

Se questi coefficienti vanno bene per un calcolo in virgola mobile, è impensabile utilizzare le approssimazioni all'unità per il calcolo in virgola fissa.

Si può ricorrere pertanto ad un accorgimento, ovvero si può andare a modificare opportunamente i coefficienti dell'equazione per ottenere il risultato voluto.

La prima idea è quella di moltiplicare tutti i coefficienti per lo stesso numero, molto alto, per risolvere il problema. Se per esempio moltiplicassimo per $10^6$ tutti i coefficienti otterremmo:

a = 1000000 -3647600 5460500 -4169400 1618100 -254700

b = 218.7 1093.5 2187.0 2187.0 1093.5 218.7

Questi coefficienti non vanno bene per due motivi:

- I coefficienti a sono troppo grandi: -4169400 da solo è in grado di mandare la variabile di accumulazione in l'overflow se moltiplicato per un numero maggiore di 510 circa, essendo l'intervallo ammesso per i numeri interi a 32 bit (±2147483648).
- a[0] non è una potenza di 2, pertanto devo effettuare una divisione la quale risulta essere pesantissima dal punto di vista computazionale, in certi casi più pesante di tutto il resto del programma.

Soluzioni:

- scegliere come coefficiente di moltiplicazione una potenza di due, in modo da poter effettuare la divisione con un semplice shifting dei bit (che viene effettuato in poche battute di clock).
- Moltiplicare i vettori dei coefficienti a e b per delle costanti diverse, in questo modo.

Prendiamo la formula di partenza:

a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb) - a(2)*y(n-1) - ... - a(na+1)*y(n-na)

Moltiplico tutti gli addendi dell'equazione per una costante k*w:

K*w*a(1)*y(n)= k*w*b(1)*x(n) +.. + k*w*b(nb+1)*x(n-nb) – k*w*a(2)*y(n-1) -...– k*w*a(na+1)*y(n-na)

Pongo A(n)=w*a(n), B(n)=k*w*b(n) e Y(n)=k*x(n):

A(1)*Y(n) = B(1)*x(n) + B(2)*x(n-1) + ... + B(nb+1)*x(n-nb) – A(2)*Y(n-1) - ... – A(na+1)*Y(n-na)

In questo modo ottengo che i coefficienti b possono essere moltiplicati per una costante k*w a discapito di ottenere i valori in uscita moltiplicati per la stessa. Inoltre anche i coefficienti a vengono moltiplicati per una costante w. Il fatto di poter moltiplicare i due vettori di coefficienti per costanti diverse è di fondamentale importanza per poter scalare diversamente le costanti del filtro.

Se pongo k=w=1024 e arrotondando i coefficienti all'unità più vicina ottengo i seguenti coefficienti:

$$A = 1024 \quad -3735 \quad 5592 \quad -4269 \quad 1657 \quad -260$$

$$B = 229 \quad 1147 \quad 2293 \quad 1147 \quad 229$$

Consideriamo l'ingresso limitato a ±256. Dal momento che il filtro non amplifica l'uscita avrà un valore massimo di ±256. Il vettore temporaneo Y avrà come valori numeri uguali all'uscita moltiplicati per k=1024, per cui sono compresi in un intervallo compreso tra $-2^{18}$ e $+2^{18}$. Poiché il processore lavora a 32 bit ho ancora 13 bit più un bit di segno con cui posso lavorare. Poiché $2^{13}$ = 8192 riesco a rimanere abbondantemente sotto la cifra critica dei coefficienti. Avere un ingresso a 9 bit in certi casi può essere penalizzante, però eventuali cifre significative che devono venir eliminate per poter restare dentro l'intervallo di ingresso consentito possono essere recuperate con un minore shifting all'uscita.

In appendice B si trova il listato in C++ del filtro IIR risultante. Per scrivere il programma completo è necessario leggere un valore dal campionatore, scalarlo in modo che non superi in modulo il valore di 2048, inserirlo nel vettore in alla posizione 5. Il vettore res deve essere dichiarato come static, ovvero i valori in esso contenuti non devono venire cancellati tra le diverse chiamate del sottoprogramma filtro. In appendice B si trovano anche i listati in C++ dei programmi definitivi.

# Capitolo 5: conclusioni e bibliografia

Al termine della stesura del codice eseguibile si è proceduto a testare i due programmi ottenuti, il modulatore ed il demodulatore, separatamente mediante interfaccia con la scheda audio del PC. Si è proceduto a generare con il programma Matlab modulator.m (2) un segnale contenente dati correttamente formattati (compatibili con il protocollo di comunicazione) per poi essere riprodotti mediante il comando soundsc. Questo segnale viene poi fatto arrivare al DSP e da qui demodulato. Subito si è notato un grosso problema: le frequenze di campionamento dei diversi convertitori D/A e A/D della scheda sonora da una parte e DSP dall'altra sembrano molto diverse (circa 10%) rendendo estremamente difficoltosi i primi test in assenza di effetto doppler. Dopo molteplici prove tuttavia si è riusciti a ricavare delle prestazioni sostanzialmente in linea con il codice scritto in Matlab. Stesso discorso è valso una volta invertiti i ruoli, ovvero generando un segnale mediante il DSP e demodulandolo con Matlab. Come ultima prova è stato fatto uso del PC come mero "registratore", ovvero andando a registrare un segnale modulato dal DSP, riprodotto con aggiunta di rumore ed effetto doppler e quindi demodulato dal DSP per verifica.

Le prestazioni ottenute sono state abbastanza difficili da registrare, nel complesso si può dire che esiste un limite di SNR che si aggira sui 6dB sotto il quale è molto raro che venga addirittura agganciata la portante, ma basta un miglioramento di qualche decimo di dB che è improbabile che il segnale venga demodulato in maniera errata. Questo avviene per una ragione molto semplice: a risentire maggiormente della presenza del rumore non è tanto il demodulatore I/Q, quanto del meccanismo di aggancio di fase. Pertanto, una volta superata la soglia dove quest'ultimo opera in maniera corretta, la demodulazione avviene con ottimi risultati.

Resta in sospeso la fusione dei due programmi in un unico eseguibile da caricare su DSP. Se dal punto di vista computazionale ciò è possibile, in quanto entrambi gli algoritmi usati impiegano meno del 50% della potenza del DSP, il problema è la comunicazione mediante porta seriale. La comunicazione in uscita (demodulazione) avviene mediante scrittura su buffer, ed essendo la velocità della seriale molto più alta di quella della comunicazione (9600 contro 1500 baud) si può affermare con ragionevole approssimazione che questo buffer sia sempre vuoto. Discorso molto più complesso avviene per la comunicazione in ingresso (modulazione) dove devo andare a leggere sul buffer in ingresso. Purtroppo l'unica funzione di libreria che permette di fare questa operazione è bloccante, ovvero una volta invocata blocca l'esecuzione del programma finché non viene letto perlomeno un byte dalla seriale. Si rende necessario pertanto la scrittura o di una funzione non bloccante oppure di una funzione che effettui un controllo sulla presenza di dati nel buffer. La

realizzazione di quest'ultima funzione è stata tentata senza successo, in quanto la documentazione fornita dal produttore, la Texas Instruments, è scarsa e parzialmente incompleta.

## Bibliografia:

THE DSP HANDBOOK - Algorithms, Applications and Design Techniques
A. Bateman, I. Paterson-Stephens
Prentice Hall - 2002

APPLICATION NOTE SPRA80 della Texas Instruments:
Implementation of an FSK Modem Using the TMS320C17
http://www-s.ti.com/sc/psheets/spra080/spra080.pdf

Teaching DSP Through the Practical Case Study of an FSK Modem
http://www-s.ti.com/sc/psheets/spra347/spra347.pdf

Tesina per il corso di Elaborazione Digitale di Segnali e Immagini
Realizzazione di un demodulatore FSK con un sistema di Digital Signal Processing
Davide Cicuta, Alessandro Vascotto

Audio Daughter Card with TMS320C5402 DSK.
Template for a TMS320C5402 project. Revision: 1.00
Author : Richard Sikora
30th April 2002.

# Appendice A: listati Matlab

```
len=input('length of the data=');
data=rand(1,len);
for x=1:len,
    if data(x)>0.5
        data(x)=1;
    else data(x)=0;
    end
end
clear x;
clear len;
```

datagen.m

---

```
len=length(data);
for i=0:len-1,
  for j=0:15,
   port((i*16)+j+1)=10*sin((12+12*data(i+1))*pi*j/96);
  end
end
clear len;
clear i;
clear j;
```

modulator.m

---

```
snr=input('SNR=');
oldport=port;
l=length(port);
noise=randn(1,l);
ps=0; %signal power
pn=0; %noise power
for i=1:l,
    ps=ps+(port(i)^2);
    pn=pn+(noise(i)^2);
end
q=(ps/pn)/snr;    % snr=ps/pn*k
noise=noise*sqrt(q) ;
%E=sum((x*k)^2)
%for i=1:l,
%    pn=pn+(noise(i)^2);
%end
port=port+noise;

clear l;
clear i;
```

```
clear q;
```

noisegen.m

```
l=length(port)/16;
for i=0:l-1,
  int1=0;
  int2=0;
  for j=1:16,
    s1=port(i*16+j)*cos((3/16)*pi*j);
    int1=int1+s1;
    s2=port(i*16+j)*sin((3/16)*pi*j);
    int2=int2+s2;
  end
  if (int1 + int2) > 10.7 out(i+1)=1;
  else out(i+1)=0;
  end  %fine if
end

clear i;
clear j;
clear l;
clear int1;
clear int2;
clear s1;
clear s2;
```

demodulator.m (1)

```
len=input('length of the data=');
data=rand(1,len);
for x=1:len,
    if data(x)>0.5
        data(x)=1;
    else data(x)=0;
    end
end
modulator;
oldport=port;
for gg=1:40,
    snr=gg/10; % define snr
    port=oldport;
    clear noise;
    clear out
    noisegen;
    demodulator;
```

```
    w=data-out;
    err(gg)=0;
    for x=1:len,
        err(gg)=err(gg)+(w(x));
    end
    err(gg)=err(gg)/len;
end
semilogy(err);

clear w;
clear len;
clear x;
```

graph.h

```
l=length(port);
buff=(0 0 0 0 0 0 0 0);
for x=1:l,
    prod(x)=buff(1)*port(x);
    for j=1:7,
        buff(j)=buff(j+1);
    end % fine for j
    buff(8)=port(x);
end % fine for x
(N, Wn) = buttord(.13, .2, 3, 30);
(a,b)=butter(N,Wn);
out1=filter(a,b,prod);
clear a;
clear b;
clear N;
clear Wn;
```

demodulator.m (2)

```
fi=input('delay=');
l=length(data);
for i=1:fi,
    port(i)=0;
end
%generiamo la portante
for i=0:l-1,
    for j=0:15,
        port((i*16)+j+1+fi)=10*sin((12+12*data(i+1))*pi*j/96);
    end % fine for j
end  % fine for i
```

```
port(16*l+fi+1)=0;
port(16*l+fi+2)=0;
```

modulator.m (2)

```
l=length(port);
buff=(0 0 0 0 0 0 0 0);
for x=1:l,
   prod(x)=buff(1)*port(x);
   for j=1:7,
      buff(j)=buff(j+1);
   end % fine for j
   buff(8)=port(x);
end % fine for x
(N, Wn) = BUTTORD(.13, .2, 3, 30);
(a,b)=butter(N,Wn);
out1=filter(a,b,prod);
% decisione sui termini
l=length(out1);
out1a=abs(out1);
max=1;
pos=0;
p=1;
while (max<9),
   if ((out1a(p)<out1a(p+1) & out1a(p+1)>out1a(p+2)) & out1a(p+1)>30)
        pos=pos+(p-1)-(max*16);
      max=max+1;
   end  %fine if
   p=p+1;
end     %fine while
fi=round(pos/8)-8;
% ho trovato la fase del segnale

l=(length(port)-fi)/16;
for i=0:l-1,
int1=0;
int2=0;
   for j=1:16,
      s1=port(i*16+j+fi)*cos((3/16)*pi*j);  %prodotto
      int1=int1+s1;                  %integro
      s2=port(i*16+j+fi)*sin((3/16)*pi*j);  %prodotto
      int2=int2+s2;                  %integro
   end

   if (int1 + int2) > 0 out(i+1)=1;
   else out(i+1)=0;
   end  %fine if
end
```

demodulator.m (3)

# Appendice B: listati C++

```
//i vettori a e b sono già dati
//in è il vettore di dati in ingresso, in(5) è il più nuovo
//res è il vettore temporaneo di dati in uscita
//out è l'uscita vera e propria

long temp;      //intero con segno a 32 bit
int i;
temp=b(0)*in(5);

for (i=1;i<6;i++) {
        temp-=(a(i)*res(5-i));
        temp+=(b(i)*in(5-i));
}

res(5)=(temp>>10);     // a(0)=1024=2^10
for (i=0;i<5;i++) res(i)=res(i+1);          // spostamento dei valori in uscita

// salvataggio dei valori in uscita
out=(res(5)>>10);
```

Listato filtro IIR (Butterworth)

---

```
/*****************************************************************************
*
* DESCRIPTION
*   Project template for use with TMS320C5402 DSK and Audio Daughter Card.
*
* REVISION
*   Revision: 1.00
*   Author  : Stefano de Fabris
*----------------------------------------------------------------------------
*
* HISTORY
*   Revision 1.00
*   19th August 2004.
*
*****************************************************************************/

#include <stdio.h>
#include <uart.h>
#include <board.h>

/* Default UART settings */
UartBaud        baud = UART_BAUD_9600;
UartWordLen     wordLength = UART_WORD8;
UartStopBits    stopBits = UART_STOP1;
UartParity      parity = UART_EVEN_PARITY;
UartFifoControl  fifoControl = UART_FIFO_DISABLE;
UartLoop        loopEnable = UART_NO_LOOPBACK;

/*****************************************************************************/
/* main()                                                                    */
```

```
/*****************************************************************************/

void main(void)
{
  int inpos=0;
  char fifoin(400);
  short transmit=0, loading=0;
  int dato;
  short alpha=20,posizione=10,part=0,parziale=0;
  //int i=0;

  brd_init(50);  /* Necessary. Initialise board for 100 MHz operation */

  // uart
  if(uart_init() == ERROR)
  {
     printf("Error initializing UART for Demo\n");
  }

  MCBSP1_init();  /* Initialise buffered serial port */

  fifoin(0)=213;  /* Initialise output buffer */

  printf("\nTMS320C5402 DSK with Audio Daughter Card. Initialized\n");

  for ( ; ;  )
   {
          /* Read sample from and write back to handset codec
      data = *(volatile u16*)DRR1_ADDR(HANDSET_CODEC);
      *(volatile u16*)DXR1_ADDR(HANDSET_CODEC) = data; */
      int userInput=EOF;
      int output;
            //double temp1, temp2;
            //char filedata(3);

      if (transmit!=2) userInput = uart_fgetc(); // se non sta trasmettendo attendi dati dalla seriale
      //userInput = 0;
      if (userInput != EOF) {
          /* Clear upper byte */
         userInput &= 0x00ff;
          if (loading==1) {                                          // se sta leggendo dati
             if (transmit==1) {                  // se e'gia arrivato un primo carattere di possibile chiusura
                            if (userInput==125) {               // se ne arriva un altro uguale
                                    transmit=0;                                     // vuol dire che e'un carattere
                                    inpos++;                                        // e lo scrive nel buffer di invio
                            fifoin(inpos)=125;
                            userInput=-1;              // per evitare problemi dovuti alle istruzioni successive
                            }
                            else {
                            transmit=2;                         // altrimenti inizia a trasmettere
                            inpos++;
                            fifoin(inpos)=128;
                            }
                    }
             if (userInput==125) {
                    transmit=1;      // se arriva un possibile comando di chiusura
                                     // (ASCII 125 seguito da carattere diverso) poni transmit a 1
                    inpos++;
                    fifoin(inpos)=125;
```

```
                }
            else {                                  // altrimenti e' arrivato un carattere qualsiasi
                if (userInput>=0) {
                        inpos++;                                        // incrementa di uno il contatore
                        fifoin(inpos)=userInput;        // scrivi il dato nel buffer di scrittura
                        if (inpos>=397) {
                                transmit=2; // se il buffer e'pieno manda ignorando i dati successivi
                                                inpos++;
                                                fifoin(inpos)=125;
                                                inpos++;
                                                fifoin(inpos)=128;
                                                }
                                        }
                }
            }
            else {                                  // se non e'ancora iniziata la sequenza di invio dei dati
                if (userInput==123) {   // controlla che arrivi un carattere di inizializzazione (ASCII 123)
                        loading=1;                                      /* inizializza la lettura dei dati */
                        inpos=0;                        // inizializza la posizione iniziale del buffer a zero
                }
           }
        }
        //output=(s16)(fifoin(inpos)*100);
        //*(volatile u16*)DXR1_ADDR(HANDSET_CODEC) = output;
        if (loading==1) brd_led_enable(BRD_LED0);       // accendi il LED 0 se stai ricevendo dati validi dalla seriale
           else brd_led_disable(BRD_LED0);                      // altrimenti tienilo spento
        if  (transmit>=1) brd_led_enable(BRD_LED1);     // accendi il LED 1 se aspetti la conferma per trasmettere
dati
            else brd_led_disable(BRD_LED1);                     // spegnilo altrimenti
        if  (transmit==2) brd_led_enable(BRD_LED2);     // accendi il LED 2 se stai trasmettendo dati
            else brd_led_disable(BRD_LED2);                     // spegnilo altrimenti

                if (transmit==2) {                              // se sono in modalita' trasmissione
                        alpha++;                                // aumenta di una unita' l'angolo
                        if (alpha>=16) {                        // se l'angolo e' maggiore del dovuto
                                alpha=0;                        // riportalo a zero
                                posizione++;    // aumenta di uno la posizione all'interno del byte
                                if (posizione>=8) {                                     // se e'finito il byte
                                        posizione=0;                    // riporta la posizione a zero
                                        if (part<inpos) {       // se non hai finito di trasmettere il buffer
                                                parziale=fifoin(part);  // leggi il valore da trasmettere
                                                part++;         // incrementa di uno il puntatore al
                                                                // valore che devi trasmettere
                                        }
                                        else {
                                                inpos=0;        // resetta il buffer
                                                transmit=0;     // ritorna nello stato di attesa di trasmissione
                                                part=0;
                                                loading=0;
                                                alpha=20;
                                                posizione=9;
                                                brd_led_disable(BRD_LED2);
                                                brd_led_disable(BRD_LED1);
                                                brd_led_disable(BRD_LED0);
                                        }
                                }
                                // devo ricavare un bit all'interno di un byte
                                // esempio: parto da    01001011
                                // ricavo il primo bit      00000001 e lo metto nella variabile dato
```

```
                                        // quindi faccio un shift  00100101
                                        dato = parziale;
                                        dato &= 0x0001;
                                        //parziale=parziale & 0xfffe;
                                        parziale = parziale >> 1;
                                        //temp1=parziale/2.0;
                                        //dato=(short)(2.0*modf(temp1,&temp2));
                                        //parziale=(char)(temp2);
                                        //dato=dato;
                                }
                                output=(sen((1+dato)*alpha));
                                //output=0;
                                MCBSP1_write(output);   /* Write to first channel */
                                MCBSP1_write(output);   /* Write to first channel */
                        } // fine if transmit*/
                        else {
                                MCBSP1_write(0);   /* Write to first channel */
                                MCBSP1_write(0);   /* Write to first channel */
                        }
  }
}

/* Funzione di calcolo veloce della funzione seno */
int sen(int angle)
{
        int ang1;
        int val;
        if (angle>15) angle = angle - 16;
        if (angle>=8) ang1=angle-8;
           else ang1=angle;
        switch (ang1) {
                case 0: val=0;
                        break;
                case 1: val=6258;
                        break;
                case 2: val=11583;
                        break;
                case 3: val=15122;
                        break;
                case 4: val=16384;
                        break;
                case 5: val=15122;
                        break;
                case 6: val=11583;
                        break;
                case 7: val=6258;
                        break;
                default:  val=0;
        }
        val=val>>1;
        if (angle>=8) val=(-val);
        return val;
}
```

Listato del programma principale del trasmettitore

```
/*****************************************************************************
```

```
*
* FILENAME
*         main.c
*
* REVISION
*   Revision: 1.00
*   Author  : de Fabris Stefano
*--------------------------------------------------------------------------
*
*****************************************************************************/

#include <stdio.h>
#include <board.h>
#include <uart.h>

long prod(6);
long res(6);
long out(8),outabs(8);
int buff(8);

void reset_filter();

int segno(float x);

/* Default UART settings */
UartBaud        baud = UART_BAUD_9600;
UartWordLen     wordLength = UART_WORD8;
UartStopBits    stopBits = UART_STOP1;
UartParity      parity = UART_EVEN_PARITY;
UartFifoControl  fifoControl = UART_FIFO_DISABLE;
UartLoop        loopEnable = UART_NO_LOOPBACK;

/*****************************************************************************/
/* main()                                                                    */
/*****************************************************************************/

void main(void)
{
        int data,data1,dataOut=0,i=1,j,k=-1,pos=0,distance=200,sign=0,leng=0;
        int max=0, pos1=0;
        float doppler=0,jump=0;
        long sen(24), integ=0;
        long temp=0;
        long a(6),b(6);
        int initialized=0;

        brd_init(50);  /* Necessary. Initialise board for 100 MHz operation */

        MCBSP1_init();  /* Initialise buffered serial port */

        // inizializzazione variabili
        // coefficienti filtro Buttworth
   a(0)= 1024;          b(0)=22;
   a(1)=-3735;          b(1)=109;
   a(2)= 5591;          b(2)=219;
   a(3)=-4268;          b(3)=219;
   a(4)= 1656;          b(4)=109;
   a(5)= -261;          b(5)=22;
```

```
// valori del seno (usati dal demodulatore IQ)
sen(0)=831;          sen(1)=383;          sen(2)=-195;
sen(3)=-707; sen(4)=-981;    sen(5)=-924;
sen(6)=-556; sen(7)=0;               sen(8)=556;
sen(9)=924;          sen(10)=981;    sen(11)=707;
sen(12)=195;         sen(13)=-383;   sen(14)=-831;
sen(15)=-1000;       sen(16)=-831;   sen(17)=-383;
sen(18)=195;         sen(19)=707;    sen(20)=981;
sen(21)=924;         sen(22)=556;    sen(23)=0;

reset_filter();

// inizializzazione seriale
if(uart_init() == ERROR)
{
   printf("Error initializing UART for Demo\n");
}

      printf("\nTMS320C5402 DSK with Audio Daughter Card. Initialized\n");
      for ( ; ;  )
      {
      data = MCBSP1_read();         // Read first channel
   data1 = MCBSP1_read();// Read second channel
   data1=(data+data1)>>5;          //faccio la media dei due valori e divido per 32
                                   //ottengo cosi' un numero minore di 2^12=4096

      //per debug
            /*test(o)=data1;  //data1 / out(7)
      o++;
      if (o>=300)
            o=0;*/

      /* In questo punto cerco di agganciare la fase */
      prod(0)=prod(1);        // salvo i prodotti ottenuti precedentemente
      prod(1)=prod(2);    // mi servono all'ingresso del filtro
      prod(2)=prod(3);
      prod(3)=prod(4);
      prod(4)=prod(5);
         prod(5)=buff(0)*data1;
      prod(5)=prod(5)>>10;
      for (j=0;j<7;j++) buff(j)=buff(j+1); // shifting dei dati in ingresso
      buff(7)=data1;                                  // salvo il nuovo campione in ingresso
      if (max<8) {
            // NOTA: i coefficienti sono scalati di un fattore s=1/b(0)
            // a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
            //                   - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
            temp=b(0)*prod(5);  // calcolo i valori del filtro
               temp-=(a(1)*res(4));
      temp+=(b(1)*prod(4));
            temp-=(a(2)*res(3));
         temp+=(b(2)*prod(3));
      temp-=(a(3)*res(2));
            temp+=(b(3)*prod(2));
         temp-=(a(4)*res(1));
      temp+=(b(4)*prod(1));
               temp-=(a(5)*res(0));
      temp+=(b(5)*prod(0));

                        res(5)=(temp>>10);     // a(0)=1024=2^10
```

```
res(0)=res(1);              // spostamento dei valori in uscita
res(1)=res(2);
res(2)=res(3);
res(3)=res(4);
res(4)=res(5);
// per debug accendo i led a seconda dell'ampiezza del segnale in ingresso
if (abs(out(7))>260) brd_led_enable(BRD_LED0);
        else brd_led_disable(BRD_LED0);
if (abs(out(7))>500) brd_led_enable(BRD_LED1);
        else brd_led_disable(BRD_LED1);

// salvataggio dei valori in uscita
for (j=0;j<7;j++) out(j)=out(j+1);
out(7)=(res(5)>>3);
outabs(7)=abs(out(7));
outabs(6)=abs(out(6));
outabs(5)=abs(out(5));
outabs(4)=abs(out(4));

// controlla che vi sia un minimo di distanza tra un pacchetto ed il successivo
if (distance<200) distance++;
doppler++;
// se c'e' un massimo/minimo potrebbe trattarsi di un carattere d'inizializzazione
if ((outabs(5)<=outabs(6) & outabs(6)>outabs(7)) & (outabs(6)>300 & distance>=100)) {
                if (max==0) doppler=0;
                max++;
                if (out(6)>0) dataOut=dataOut+i;
        i=i<<1;
                distance=100;
        }
        if (distance>130) {  //se rilevo un picco isolato vuol dire che è solo rumore
                max=0;
                dataOut=0;
                i=1;
        }
}
else {
        if (max==8) {
                if (dataOut!=213) { //il carattere d'inizializzazione e' stato
                        max=0;                  //ricevuto errato: annulla tutto
                        dataOut=0;
                        i=1;
                        }
                else {              // carattere inizializzazione ricevuto correttamente
                        doppler=(doppler-107)*16/107;  // stima dell'effetto doppler
                        integ= sen(0)*buff(7) + sen(8)*buff(7);
                        integ+=sen(1)*buff(6) + sen(9)*buff(6);
                        if (buff(4)>0) sign=1;  //controlla il "segno" del segnale
                                else sign=-1;
                        uart_fputc(123);        //scrivi il carattere d'inizio in uscita
                        max=9;                          //resetto alcune variabili
                        pos1=2;
                        pos=0;
                        dataOut=0;
                        i=1;
                        initialized=2;
                        leng=0;
                        jump=0;
                }
```

```
                        } //fine if max==8
                        else {
                                if (pos1>=16) { //se e' finita la finestra di simbolo
                                        if ((integ*sign)>0) k=0; //DECISIONE SUL SIMBOLO
                                                else k=1;
                                        //uart_fputc(k+100);
                                        integ=0;
                                        jump=jump+(16*doppler);    //funzione per il calcolo periodico
                                        if (abs(jump)>1) {                      //dell'effetto doppler
                                                pos1=-segno(jump);     //per ogni finestra d'integrazione
                                                jump=jump-segno(jump); //stimo se allungarla o meno
                                                }
                                          else pos1=0;
                                        dataOut=dataOut+(i*k); //scrivi il valore nella memoria
                                i=i<<1;
                                pos++;
                                if (pos>=8) { //se e' finito il byte (8 bit) scrivili sulla seriale
                                        leng++;
                                        if (leng>=400) { //pacchetto di lunghezza massima
                                                initialized=1;
                                                dataOut=0;
                                        }
                                        if (initialized==2 & dataOut!=125)  //carattere normale
                                                uart_fputc(dataOut);
                                        if (initialized==1) {  //potenziale carattere di fine pacchetto
                                                if (dataOut==125) { //caso negativo
                                                        uart_fputc(dataOut);
                                                        dataOut=0;
                                                        initialized=2;
                                                }
                                                else { //arrivato carattere terminazione del pacchetto
                                                        uart_fputc(113);
                                                        reset_filter(); //risetto il filtro
                                                        max=0;                          //azzero le variabili
                                                        leng=0;
                                                        distance=-100;
                                                        k=-1;
                                                        sign=0;
                                                        //dataOut=0;
                                                        initialized=0;
                                                }
                                        }
                                        if (dataOut==125 & initialized==2) {
                                                initialized=1;
                                                uart_fputc(125);
                                                }
                                        pos=0;
                                        dataOut=0;
                                        i=1;
                                }  //fine if (pos>=8)

                                } //fine if pos1>=16
                                integ+=sen(pos1)*data1+sen(pos1+8)*data1; //integrale parziale
                                pos1++;
                        }
                }
  }
}
```

```
// procedura per la cancellazione dei valori del filtro di butworth
void reset_filter() {
        int j;

   prod(0)=0;          res(0)=0;                   out(0)=0;
   prod(1)=0;          res(1)=0;                   out(1)=0;
   prod(2)=0;          res(2)=0;                   out(2)=0;
   prod(3)=0;          res(3)=0;
   prod(4)=0;          res(4)=0;
   prod(5)=0;          res(5)=0;

        for (j=0;j<8;j++) buff(j)=0;
}

//funzione per il calcolo del segno di un numero
int segno(float x) {
        if (x>0) return 1;
        if (x<0) return (-1);
        return 0;
}
```

Listato del programma principale del ricevitore