UNIVERSITÀ DEGLI STUDI DEL SANNIO

FACOLTÀ DI INGEGNERIA INFORMATICA

***A.A.** 2008/2009*

***E**laborazione dei **S**egnali e delle **I**nformazioni di **M**isura* **[ESIM]**

Prof. P. Daponte
Ing. L. De Vito

"**S**oftware **D**efined **R**adio [SDR]
per la Classificazione delle Modulazioni Analogiche"

Delli Veneri Luca
matr. 195/000529
Espresso Giovanni
matr. 195/001405
Facchiano Giambattista
matr. 195/000333
Gramazio Paolo
matr. 195/001164
Varricchio Giuseppe
matr. 195/000848

Versione 1.50 del 10/11/2009 00.16.49

Indice Generale

1. Introduzione

Lo scopo del progetto è stato quello di realizzare un *Classificatore di Modulazioni* come componente di una Software Defined Radio (SDR) il quale a partire da un segnale Analogico, dato in ingresso, ha il compito di identificare una:

- Modulazione di Ampiezza **AM -** *(Amplitude Modulation)*
- Modulazione di Fase **PM -** *(Phase Modulation)*
- Modulazione di Frequenza **FM -** *(Frequency Modulation)*

Per far ciò è stato necessario acquisire familiarità con:

- l'elaborazione numerica dei segnali e delle tecniche di filtraggio
- i principi alla base di una ***SDR***
- l'architettura di un Digital Signal Processor (***DSP***) nello specifico una scheda Texas Instrument (TI) TMS320C6711 DSK
- simulazioni del modello matematico risolutivo in MATLAB
- traduzione del corrispettivo linguaggio MATLAB in linguaggio C
- l'ambiente di sviluppo ed integrazione TI – Code Composer Studio v2

Lo sviluppo del progetto ha richiesto le seguenti fasi:

1. Analisi dei requisiti ed individuazione di una soluzione (modello matematico)
2. Implementazione in MATLAB di un M-File per la simulazione dell'algoritmo risolutivo (verifiche di corretto funzionamento)
3. Traduzione dell'algoritmo in linguaggio C sulla base di documentazione Texas Instrument, MathWorks ed ulteriore
4. Implementazione e testing sulla scheda DSP "fisica" utilizzando tra l'altro la gestione degli Interrupt Globali ed un generatore di Segnali per le Modulazioni

2. Conoscenze Matematiche fondamentali ed Analisi del problema

2.1 Software Defined Radio – SDR

La Software Defined Radio, in sigla SDR, è una tecnologia di comunicazione che permette di costruire ricevitori *Radio*[1] non più totalmente *hardware* ma in gran parte *software* che a differenza degli altri sono multistandard[2] nonché programmabili (es.: GSM, UMTS, WiMax, DVB, Satellitare, ecc.).

I vantaggi ad avere un ricevitore unico sono notevoli ma tra questi sicuramente citiamo i due più importanti:

- L'estensione della funzionalità del sistema ad un ampio intervallo di trasmissioni radio.
- La ri-configurabiltà che permette al sistema ri-programmabile di estendere le capacità del terminale senza la sostituzione di alcuna parte hardware. In questo modo più standard possono essere gestiti tranquillamente da un solo ricevitore.

Lo svantaggio principale è invece imposto dal Teorema del Campionamento[B1] (Nyquist-Shannon) visto che si richiederebbero convertitori ADC in grado di gestire una mole notevole di campioni, tuttavia sono possibili soluzioni tecnologiche di compromesso come ad esempio quella di de-modulare su frequenze più basse, non compromettendo la ricostruzione del segnale dopo il blocco di DAC.

Nel nostro caso di studio la SDR è composta da: una scheda DSP, convertitori, Analog Signal Generator; il nostro **gruppo di lavoro** ha realizzato un esempio di *Classificatore delle Modulazioni*.

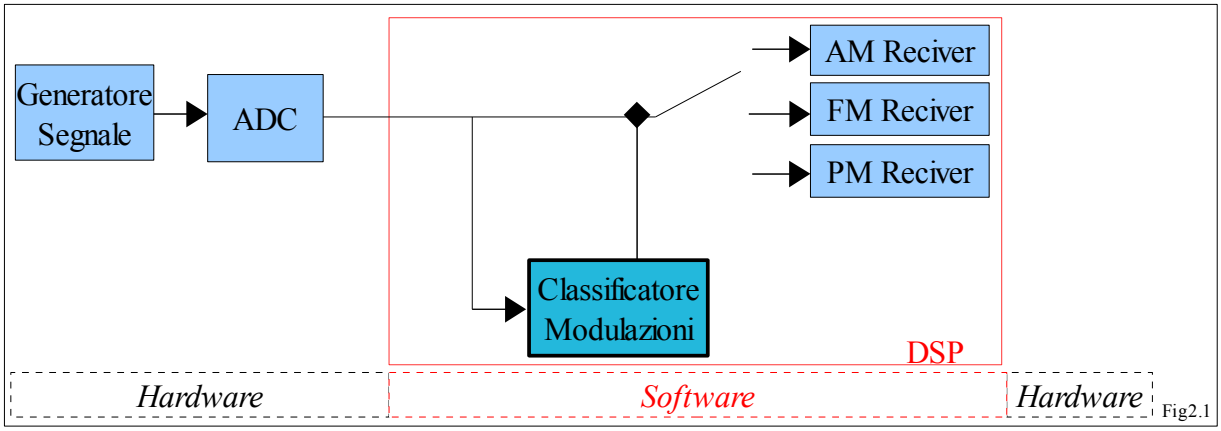


Fig2.1

E' realizzata in software,come si evince dalla Fig.2.1, gran parte della SDR compreso anche il nostro componente.

Casi di Studio avanzati, trascurati nel nostro ambito, possono essere la *Cognitive Radio (adattativa)* per ridurre le interferenze e/o ottimizzare i canali di trasmissione/ricezione, e così via.

1. apparecchio elettronico che permette di trasmettere e/o ricevere onde radio di natura elettromagnetica.
2. nel caso specifico di SDR, ricevitore che funziona con diversi standard perché il suo hardware è riprogrammabile attraverso software.

2.2 Classificatore Modulazioni

Per poter classificare le Modulazioni Analogiche, grazie ai concetti sull'elaborazione numerica dei segnali, abbiamo deciso di formulare un possibile algoritmo risolutivo sulla base di un modello matematico successivamente implementato in Matlab (per la simulazione e verifica della correttezza dello stesso). Il modello di riferimento utilizzato è quello classico dei segnali campionati, applicabile anche nelle tecnologie multimediali (musica ed immagini). Questo modello si fonda su diverse leggi matematiche, ma la più importante è il teorema di Shannon, che legando banda del segnale con la minima frequenza di campionamento (o la frequenza di campionamento con la massima banda campionabile), costituisce il vincolo tecnologico principale per l'applicabilità dei sistemi digitali nel mondo RF. Per il nostro progetto abbiamo preso in considerazione la banda **ULF - Ultra low frequency** che nella designazione ITU[3] indica la parte dello spettro delle onde radio compresa tra 300 e 3000 Hz. Potendo attraversare la terra, queste frequenze vengono utilizzate per le comunicazioni in miniera ed inoltre sono state utilizzate per **trasmissioni militari segrete**.

2.2.1 Strumenti Matematici: la rappresentazione analitica di un segnale

La **rappresentazione analitica**[B1] di una funzione o di un segnale a valori reali facilita molte manipolazioni matematiche su di essi. L'idea base è che le componenti di frequenza negativa della Trasformata di Fourier (o Spettro) di una funzione a valori reali sono superflue, grazie alla simmetria Hermitiana di tali spettri. Queste componenti di frequenza negativa possono essere scartate senza nessuna perdita di informazione, fornendo invece un modo di risoluzione con funzioni a valori complessi.

Questo rende alcuni attributi del segnale più accessibili e facilita la determinazione delle tecniche di modulazione e demodulazione. Finché la funzione manipolata non ha componenti di frequenza negativa la conversione dal complesso al reale è soltanto una sorta di eliminazione della parte immaginaria. La rappresentazione analitica è una generalizzazione del concetto di *Fasore*: mentre quest'ultimo è limitato all'ampiezza, fase e frequenza tempo-invariante, il primo tiene conto di parametri tempo-varianti.

Un segnale analitico può essere rappresentato in termine di coordinate complesse:

$$x_{\mathrm{a}}(t) = A(t)e^{j\phi(t)},$$

con A(t) il modulo e Φ(t) la fase

$$A(t) = |x_{\mathrm{a}}(t)| = \sqrt{x^2(t) + \hat{x}^2(t)}$$

$$\phi(t) = \arg\{x_{\mathrm{a}}(t)\}.$$

dove la funzione arg{} è la arcotangente del rapporto fra parte Reale ed Immaginaria di $x_a(t)$.

Queste funzioni rappresentano l'*inviluppo complesso* e la *fase istantanea* del segnale x(t). La derivata nel tempo della fase istantanea "unwrapped" è chiamata *frequenza istantanea:*

$$\omega(t) \stackrel{\text{def}}{=} \phi'(t) = \frac{d}{dt}\phi(t).$$

La funzione Ampiezza, Fase e Frequenza istantanea sono usate in molte applicazioni per misurare e determinare le caratteristiche locali dei segnali.

Quindi per la realizzazione del nostro *Classificatore*, dopo l'acquisizione e campionamento del segnale analogico, si è considerato un indice di variazione della ampiezza e della frequenza e fase istantanea.

3. Unione internazionale delle telecomunicazioni che si occupa di definire gli standard nelle telecomunicazioni e nell'uso delle onde radio.

2.2.2 Strumenti Matematici: Hilbert, Unwrap, Deviazione Standard STD

– Trasformata di Hilbert[B2]

Al fine di individuare ampiezza, fase e frequenza istantanea si è utilizzata la Trasformata di Hilbert (discreta), la quale è un operatore lineare che non cambia il dominio di appartenenza della funzione, ed è un utile strumento nell'analisi in Frequenza (tramite Fourier) fornendo un modo concreto di realizzare il coniugato di una data funzione. Il segnale analitico di cui sopra è definito come $x = x_r + i\, x_i$ dove la parte reale è il segnale di partenza e la parte immaginaria il suo coniugato (T. di Hilbert), con sfasamento di 90°.

L'algoritmo utilizzato in sostanza consta di 4 passi (step):

- calcolo della FFT del segnale in ingresso
- maschera per eliminare le frequenze negative (poste a 0), creando un vettore h(i) i cui elementi sono calcolati come segue:
  - 1 per i = 1, (n/2)+1
  - 2 per i = 2, 3, ... , (n/2)
  - 0 per i = (n/2)+2, ... , n
- prodotto scalare tra la FFT del segnale e la maschera h(i)
- calcolo della IFFT (inversa della FFT)

dove ***n*** è la dimensione/accuratezza (punti o campioni), e la **FFT** rappresenta la **DFT** (Discrete Fourier Transform) "veloce" ottimizzata nelle prestazioni per l'elaborazione numerica dei segnali.

– Unwrap della Fase ("salti di fase")

L'unwrap[B3] consente la corretta ricostruzione della fase originale del segnale e l'algoritmo risolutivo somma appropriati multipli di 2π per ogni fase in ingresso.

Il metodo utilizzato consta di 3 passi:

- pone k = 0
- controlla la presenza di un "salto di fase" tra elementi adiacenti:
  - se non c'è un "salto di fase", ovvero $(|u_{i+1} - u_i| \le |\alpha|)$, somma $2\pi k$ ad u_i e ripete il passo corrente
  - se c'è un "salto di fase", ovvero $(|u_{i+1} - u_i| > |\alpha|)$, somma $2\pi k$ ad u_i e procede al passo successivo
- Aggiorna k :
  - se $u_{i+1} < u_i$ (salto di fase negativo), incrementa k
  - se $u_{i+1} > u_i$ (salto di fase positivo), decrementa k

dove $\mathbf{u}_i$ è l'i-esimo elemento del vettore fase, **α** è la tolleranza scelta pari a *π, ed* il "salto di fase" infine rappresenta la differenza fra il campione corrente ed il successivo.

– STD - Deviazione Standard

La deviazione standard[B4] (o scarto tipo/scarto quadratico medio) è un indice di dispersione derivato direttamente dalla varianza, ha la stessa unità di misura dei valori osservati (mentre la varianza ha come unità di misura il quadrato dell'unità di misura dei valori di riferimento). La deviazione standard misura la dispersione dei dati intorno al valore atteso. Ed in statistica inferenziale si rappresenta:

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}}$$

dove $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ è la media aritmetica.

2.3 Modello matematico in sintesi

2.3.1 Modulazione AM

La *modulazione di ampiezza*[B5] è uno dei sistemi utilizzati per trasmettere informazioni utilizzando un segnale a radiofrequenza(RF). Consiste nel modulare l'ampiezza del segnale radio che si intende utilizzare per la trasmissione (detto portante) in maniera proporzionale all'ampiezza del segnale che si intende trasmettere (modulante). Il segnale modulato ha la stessa frequenza della portante. In formule:

$$v(t) = (V_p + K_a V_m \cos \omega_m t) \cos \omega_p t$$

con $\omega_p \gg \omega_m$,

dove $v_m(t) = V_m \cos(\omega_m t + \varphi)$ è la modulante e $v_p(t) = V_p \cos \omega_p t$ è la portante, ed $m_a = K_a \frac{V_m}{V_p}$ è l'indice o *profondità di modulazione (*tipicamente 40 - 50 %*)*.

2.3.2 Modulazione PM/FM

La *modulazione di fase*, è una tecnica di modulazione di un segnale e si ottiene variando la fase della portante rispetto al suo valore in assenza di modulazione, proporzionalmente al valore istantaneo dell'ampiezza del segnale.

La *modulazione di frequenza* consiste nel modulare la frequenza del segnale radio che si intende utilizzare per la trasmissione (detto portante) in maniera proporzionale all'ampiezza del segnale che si intende trasmettere. In formule:

$$v_{FM}(t) = V_p \cos(\omega_p t + m \operatorname{sen} \omega_m t)$$

con $\omega_p \gg \omega_m$,

dove $v_m(t) = V_m \cos(\omega_m t + \varphi)$ è la modulante e $v_p(t) = V_p \cos \omega_p t$ è la portante.

A causa,però, della similarità tra la modulazione di fase e frequenza si è scelto di trattarli come un unica classe di modulazione analogica (spesso identificate come modulazioni angolari).

2.3.3 Scelta della procedura risolutiva

Per poter classificare le modulazioni AM, PM/FM, raggruppate per comodità e similarità, si è proseguito come segue:

– acquisizione e campionamento del segnale
– normalizzazione del segnale campionato tramite la sua deviazione standard per renderlo di potenza unitaria
– calcolo della T. di Hilbert del segnale normalizzato
– rimozione delle code al fine di eliminare gli effetti deleteri della trasformata di Hilbert
– calcolo dell'*ampiezza istantanea* (inviluppo complesso)
– calcolo della *fase istantanea* e correzione tramite "Unwrap"
– calcolo della *frequenza istantanea* tramite derivata nel tempo della *fase istantanea*
– classificazione delle modulazioni

	AM	PM/FM	No Modulation
STD(ampiezza)	**≥ 0.3**	...	**Altrimenti**
STD(frequenza)	...	**≥ 0.04**	

Nota: per STD si intende la Deviazione standard degli attributi istantanei (ampiezza e frequenza) Tab2.1

Nella tabella Tab 2.1 è mostrata la logica del blocco *Classificatore* dove i valori soglia sono stati scelti in base a prove sperimentali tramite simulazioni in Matlab.

3. IMPLEMENTAZIONE IN MATLAB

L'implementazione in Matlab[B6] per valutare la correttezza dell'algoritmo risolutivo ha richiesto la simulazione di segnali modulati AM, PM/FM. L'accuratezza dei test, grazie al numero elevato di campioni (o punti) [1024] ha restituito un corretto risultato in gran parte delle prove di simulazione al variare delle tipologie dei segnali.

3.1 Codice M-file (Matlab)

```
%-- Segnale modulato in ampiezza --%
t=linspace(0,0.02048,1024);          % vettore dei tempi
xp=cos(2*pi*2000*t);                 % onda portante xp=vp*cos(2*pi*fp*t)
xm=0.5*cos(2*pi*200*t);              % onda modulante xm=vm*cos(2*pi*fm*t)
xam=(1+xm).*(cos(2*pi*2000*t));      % segnale modulato in ampiezza
                                     xam=(vp+xm)*cos(2*pi*fp*t) con fp>>fm
xam_norm = xam/std(xam);             % normalizzazione tramite STD
Xh1 = hilbert(xam_norm);             % Trasformata di Hilbert
t1 = linspace(0,0.012,600);          % vettore dei tempi (code omesse)
AI1 = abs(Xh1);                      % Ampiezza istantanea
AI1_r = AI1(213:812);                % rimozione code
FI1 = diff(unwrap(angle(Xh1)));      % Frequenza istantanea
FI1_r = FI1(213:811);                % rimozione code
stdAM = std (AI1_r)                  % Deviazione standard Amp. istantanea
stdFM = std (FI1_r)                  % Deviazione standard Freq. istantanea
%-- Segnale modulato in frequenza --%
xp=cos(2*pi*2000*t);                 % onda portante xp=vp*cos(2*pi*fp*t)
xm=0.5*cos(2*pi*200*t);              % onda modulante xm=vm*cos(2*pi*fm*t)
xfm=cos(2*pi*2000*t+0.5*sin(2*pi*    % segnale modulato in frequenza
200*t));                             xfm=vp*cos((2*pi*fp*t+m*sin(2*pi*fm*t)
                                     ) con wp>>wm
xfm_norm = xfm/std(xfm);             % normalizzazione tramite STD
Xh2 = hilbert(xfm_norm);             % Trasformata di Hilbert
AI2 = abs(Xh2);                      % Ampiezza istantanea
AI2_r = AI2(213:812);                % rimozione code
FI2 = diff(unwrap(angle(Xh2)));      % Frequenza istantanea
FI2_r = FI2(213:812);                % rimozione code
stdAM = std (AI2_r)                  % Deviazione standard Amp. istantanea
stdFM = std (FI2_r)                  % Deviazione standard Freq. istantanea
%-- Segnale modulato in fase --%
xp=cos(2*pi*2000*t);                 % onda portante xp=vp*cos(2*pi*fp*t)
xm=0.5*cos(2*pi*200*t);              % onda modulante xm=vm*cos(2*pi*fm*t)
xpm=cos(2*pi*2000*t+0.5*0.5*sin(2    % segnale modulato in fase
*pi*200*t));                         xpm=vp*cos((2*pi*fp*t+kp*vm*sin(2*pi*f
                                     m*t))
xpm_norm = xpm/std(xpm);             % normalizzazione tramite STD
Xh3 = hilbert(xpm_norm);             % Trasformata di Hilbert
AI3 = abs(Xh3);                      % Ampiezza istantanea
AI3_r = AI3(213:812);                % rimozione code
FI3 = diff(unwrap(angle(Xh3)));      % Frequenza istantanea
FI3_r = FI3(213:812);                % rimozione code
stdAM = std (AI3_r)                  % Deviazione standard Amp. istantanea
stdFM = std (FI3_r)                  % Deviazione standard Freq. istantanea
```

4. TRADUZIONE DEL CODICE MATLAB IN LINGUAGGIO C

In questa fase si è tradotto l'algoritmo risolutivo precedentemente implementato in Matlab in linguaggio C, compatibile con l'ambiente di sviluppo (Texas Instrument Code Composter Studio v2) per la DSP utilizzata[B7]. Questo processo ha richiesto la comprensione di alcuni algoritmi implementativi, citati nei precedenti capitoli, resi possibili dalle documentazioni della Mathworks e dalle user guide della Texas Instrument (es.: la *TMS320C67x DSP Library Programmer's Reference Guide).* Infine si è proceduto alla verifica di correttezza dei risultati ottenuti dal confronto tra l'M-File di Matlab e il codice C sviluppato.

4.1 Dettagli implementativi del codice C

E' stata realizzata un libreria (dsplib.h) con le funzionalità richieste tra cui citiamo per la realizzazione della funzione Hilbert :

- DSPF_sp_cfftr2_dit per il calcolo della FFT (dove sono di supporto le funzioni: *gen_w_r2* che genera i fattori di *twiddle*, *bit_rev* che effettua il bit-reserve e *divide*)
- DSPF_sp_icfftr2_dif per il calcolo della IFFT

dove FFT e IFFT sono in singola precisione, "cache" ottimizzate con radice pari a 2, e di complessità computazionale pari a O(n log(n)) secondo l'algoritmo di *Cooley-Tukey* o sue varianti; in sostanza si divide ricorsivamente una DFT di dimensione N qualsiasi in DFT più piccole di dimensioni N1, N2, insieme a O(n) moltiplicazioni per l'unità immaginaria, detti fattori twiddle.

4.2 Codice C

Il codice C di seguito è proposto come una libreria ***<dsplib.h>*** ed un ***<sdr01.c>*** che contiene la routine principale del software realizzato. La logica di funzionamento è real-time ed essendo gran parte del codice eseguito nella ISR – Interrupt Service Routine, si è gestita la *global_IRQ,* ovvero la Gestione globale delle Interruzioni. L'output è stato gestito con l'uso dei 3 LED[B8] per identificare le diverse modulazioni. Inoltre si fa notare che si effettua un controllo per scartare le interferenze tipo rumore sul canale di acquisizione in caso di mancanza di ingresso al convertitore ADC.

```
/*
 file: dsplib.h
* DSP Library for ESIM Project 2009 (Software Defined Radio)*
* Students:
* Delli Veneri Luca, Espresso Giovanni, Facchiano Giambattista, Gramazio Paolo, Varricchio Giuseppe
*/


#include <math.h>
#define M_PI          3.14159265358979323846
#define N 256
#define LB 53
#define B 1000
#define NN (N-2*LB)
#define STD_AM 0.3
#define STD_FM 0.04


//COMPLEX DEFINITION

struct complex {
            float re[N];
            float img[N];
};
```

```
//FUNCTION PROTOTYPES

struct complex hilbert(float p[], int dim);
void unwrap(float p[],float u[],int dim);
float std(float x[],int dim);
void diff(float p[],float u[],int dim);
void rmvq(float p[],float u[],int dim, int lb);
void angle(struct complex c,float u[],int dim);
void module(struct complex c,float u[],int dim);
int clas(float stdAM,float stdFM);
void norm_by_std(float p[],float u[], int dim);

/* How to use: FFT
gen_w_r2(w, N); // Generate coefficient table
bit_rev(w, N>>1); // Bit-reverse coefficient table
DSPF_sp_cfftr2_dit(x, w, N); //input in normal order, output in order bit-reversed coefficient table in bit-reversed order
bit_rev(x, N); // input in bit-reversed, output in normal order.
*/
void DSPF_sp_cfftr2_dit(float* x, float* w, short n);
void gen_w_r2(float* w, int n);
void bit_rev(float* x, int n);

/* How to use: IFFT
gen_w_r2(w, N); // Generate coefficient table
bit_rev(w, N>>1); // Bit-reverse coefficient table
bit_rev(x,N); // bit-reverse output
DSPF_sp_icfftr2_dif(x, w, N); // Inverse radix 2 FFT input in bit-reversed order, order output in normal coefficient table in bit-reverse order
divide(x, N); // scale inverse FFT output result is the same as original input
*/
void DSPF_sp_icfftr2_dif(float* x, float* w, short n);
void divide(float* x, int n);


//FUNCTION IMPLEMENTATIONS

/* Hilbert Transform - using FFT and IFFT by TI dsplib for DSP TMS320C6711 MathWorks method Algorithm */
struct complex hilbert(float p[],int dim) {
        struct complex c;
        float t_x1[2*N],h[2*N];
        float w[N];
        int i;
        for(i=0; i<dim; i++) {
                t_x1[2*i]=p[i];
                t_x1[2*i+1]=0.0;
        }
//1. Hilbert calculates the FFT of the input sequence, storing the result in a vector x.
        gen_w_r2(w, dim);
        bit_rev(w, dim>>1);
        DSPF_sp_cfftr2_dit(t_x1, w, dim);
        bit_rev(t_x1, dim);
//2. It creates a vector h(i) for maskig.
        H[0]=1;
        h[dim]=1;
        or(i=1; i<dim; i++) h[i]=2;
        for(i=(dim+1); i<2*dim; i++) h[i]=0;
//3. It calculates the element-wise product of x and h.
        for(i=0; i<2*dim; i++) t_x1[i] *= h[i];
//4. It calculates the inverse FFT of the sequence obtained in step 3 and returns the first n elements of the result.
        gen_w_r2(w, dim);
        bit_rev(w, dim>>1);
        bit_rev(t_x1, dim);
        DSPF_sp_icfftr2_dif(t_x1, w, dim);
        divide(t_x1, dim);
//5. create the Complex Struct type.
        for(i=0; i<dim; i++) {
                c.re[i]=t_x1[2*i];
                c.img[i]=t_x1[2*i+1];
        }
        return c;
}

/* Unwrap - Phase Angle using Mathworks algorithm method */
void unwrap(float p[],float u[],int dim) {
        int i, k=0;
        for (i=0;i<dim;i++) {
```

```
                        if(fabs(p[i+1]-p[i])<= M_PI) u[i]=p[i]+2*M_PI*k;
                        else {
                                u[i]=p[i]+2*M_PI*k;
                                if(p[i+1]<p[i]) k++;
                                else if(p[i+1]>p[i]) k--;
                                else ;
                        }
                }
}

/* STD - Standard Deviation - the Statistic inference type */
float std(float x[],int dim) {
                int i;
                float avg=0, std=0;
                for(i=0; i<dim; i++) avg+=x[i];
                avg/=dim;
                for(i=0;i<dim;i++) std+=pow((x[i]-avg),2);
                std/=(dim-1);
                std=sqrt(std);
                return std;
}

/* Diff function like MATLAB type (Mathworks) */
void diff(float p[],float u[],int dim) {
                int i;
                for (i=0;i<dim;i++) u[i]=p[i+1]-p[i];
}

/* Remove Queues - function for reduce and eliminate 2 queues from a Vector */
void rmvq(float p[],float u[],int dim, int lb) {
                int i;
                for(i=0;i<(dim-2*lb);i++) u[i]=p[lb+i];
}
/* Angle - Phase Angle calculated by ATAN2 function with complex (Vector) */
void angle(struct complex c,float u[],int dim) {
                int i;
                for(i=0; i<dim;i++) u[i] = atan2(c.img[i],c.re[i]);
}

/* Module - also ABS for Absolute Value of complex (Vector) */
void module(struct complex c,float u[],int dim) {
                int i;
                for(i=0; i<dim;i++) u[i] = sqrt(pow(c.re[i],2)+ pow(c.img[i],2));
}

/* Signal Classification - AM  or PM/FM, or No Modulation */
int clas(float stdAM,float stdFM) {
                if(stdAM >= STD_AM) return 1; //AM
                else if(stdFM >= STD_FM) return 2; //FM or PM
                else return 3; //No Modulation or error
}

/* Signal normalization by his STD (Standard Deviation) */
void norm_by_std(float p[],float u[], int dim) {
                int i;
                float t_std = std(p,dim);
                for(i=0;i<dim;i++) u[i]=p[i]/t_std;
}

/* Single-precision cache optimized radix-2 forward FFT with complex input  */
void DSPF_sp_cfftr2_dit(float* x, float* w, short n) {
                short n2, ie, ia, i, j, k, m;
                float rtemp, itemp, c, s;
                n2 = n;
                ie = 1;
                for(k=n; k > 1; k >>= 1) {
                        n2 >>= 1;
                        ia = 0;
                        for(j=0; j < ie; j++) {
                                c = w[2*j];
                                s = w[2*j+1];
                                for(i=0; i < n2; i++) {
                                        m = ia + n2;
                                        rtemp = c * x[2*m] + s * x[2*m+1];
                                        itemp = c * x[2*m+1] - s * x[2*m];
```

```
                                        x[2*m] = x[2*ia] – rtemp;
                                        x[2*m+1] = x[2*ia+1] – itemp;
                                        x[2*ia] = x[2*ia] + rtemp;
                                        x[2*ia+1] = x[2*ia+1] + itemp;
                                        ia++;
                                }
                                ia += n2;
                        }
                        ie <<= 1;
                }
}

/* generate real and imaginary twiddle table of size n/2 complex numbers */
void gen_w_r2(float* w, int n) {
                int i;
                float pi = 4.0*atan(1.0);
                float e = pi*2.0/n;
                for(i=0; i < ( n>>1 ); i++) {
                        w[2*i] = cos(i*e);
                        w[2*i+1] = sin(i*e);
                }
}

/* bit-reverse the output */
void bit_rev(float* x, int n) {
                int i, j, k;
                float rtemp, itemp;
                j = 0;
                for(i=1; i < (n-1); i++) {
                        k = n >> 1;
                        while(k <= j) {
                                j -= k;
                                k >>= 1;
                        }
                        j += k;
                        if(i < j) {
                                rtemp = x[j*2];
                                x[j*2] = x[i*2];
                                x[i*2] = rtemp;
                                itemp = x[j*2+1];
                                x[j*2+1] = x[i*2+1];
                                x[i*2+1] = itemp;
                        }
                }
}

/* Single-precision cache optimized radix-2 Inverse FFT with complex input  */
void DSPF_sp_icfftr2_dif(float* x, float* w, short n) {
                short n2, ie, ia, i, j, k, m;
                float rtemp, itemp, c, s;
                n2 = 1;
                ie = n;
                for(k=n; k > 1; k >>= 1) {
                        ie >>= 1;
                        ia = 0;
                        for(j=0; j < ie; j++) {
                                c = w[2*j];
                                s = w[2*j+1];
                                for(i=0; i < n2; i++) {
                                        m = ia + n2;
                                        rtemp = x[2*ia] – x[2*m];
                                        x[2*ia] = x[2*ia] + x[2*m];
                                        itemp = x[2*ia+1] – x[2*m+1];
                                        x[2*ia+1] = x[2*ia+1] + x[2*m+1];
                                        x[2*m] = c*rtemp – s*itemp;
                                        x[2*m+1] = c*itemp + s*rtemp;
                                        ia++;
                                }
                                ia += n2;
                        }
                        n2 <<= 1;
                }
}
```

```
/* divide each element of x by n */
void divide(float* x, int n) {
        int i;
        float inv = 1.0 / n;
        for(i=0; i < n; i++) {
                x[2*i] = inv * x[2*i];
                x[2*i+1] = inv * x[2*i+1];
        }
}
```

```
/*
file: sdr01.c
*/


#include "dsplib.h"
#include <stdio.h>
#include <c6x.h>                                //C6000 compiler definitions
#include <csl.h>                                //Chip - CSL headers
#include <csl_irq.h>
#include <csl_mcbsp.h>
#include <bsl.h>                                //Board - BSL headers
#include <bsl_ad535.h>


//PROTOTYPES
void codec_init(void);
void init_HWI(void);
void isr(void);


//GLOBAL VARIABLES
int i = 0;
int j = 0;
int k = 0;
int acq = 0;
float x[N];
float std_AM, std_FM;
struct complex c;
float y[N], z[NN];

// Codec AD535 Configuration and Handling
AD535_Handle hAD535;
AD535_Config my_AD535_Config = {
        AD535_LOOPBACK_DISABLE,
        AD535_MICGAIN_OFF,
        AD535_GAIN_0DB,
        AD535_GAIN_0DB
};


//MAIN FUNCTION
void main() {
        CSL_init();
        BSL_init();
        codec_init();
        init_HWI();

        while(1);
}

void codec_init() {
        hAD535 = AD535_open(AD535_localId);
        AD535_reset(hAD535);
        AD535_config(hAD535, &my_AD535_Config);
}

void init_HWI(void) {
        IRQ_globalEnable();                     //Enable ints globally
        IRQ_enable(IRQ_EVT_RINT0);              //Enable the recive INT0
}
```

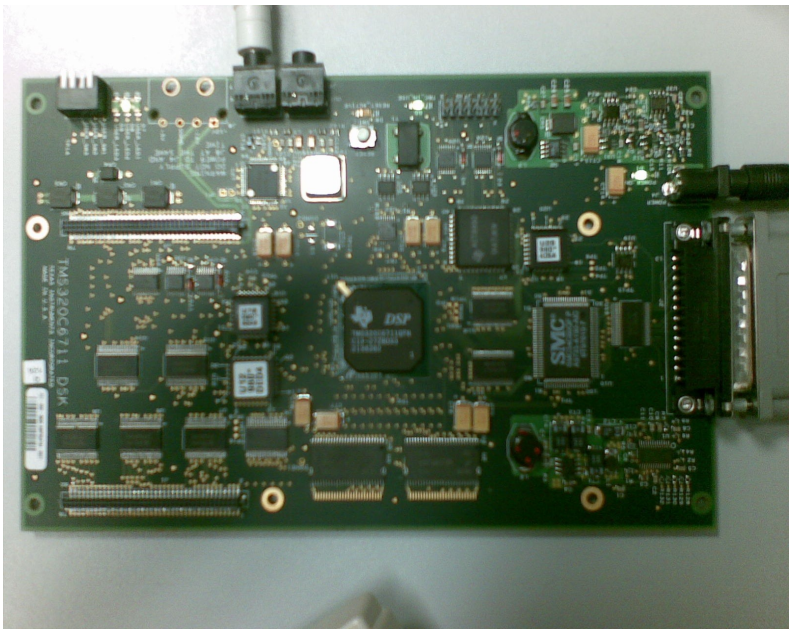
```
void isr(void) {
	int h;
	short sample = (short) AD535_read(hAD535);
	if(acq) {
		if(j < N) {
			x[j] = sample;
			j++;
		}
		else {
			k = 0;
			j = 0;
			acq = 0; //end of acquisition
			IRQ_globalDisable();
			LED_off(LED_ALL);
			if(std(x,N) < B) printf("\nsignal too low\n");
			else {
				norm_by_std(x,y,N);
				c=hilbert(y,N);
				module(c,x,N);
				rmvq(x,z,N,LB);
				std_AM=std(z,NN);
				angle(c,x,N);
				unwrap(x,y,N);
				diff(y,x,N);
				rmvq(x,z,N,LB);
				std_FM = std(z,NN);
				h = clas(std_AM,std_FM);
				if(h == 1) {
				//printf("\n AM\n"); //LED 1
				LED_on(LED_1);
				}
				else if(h == 2) {
				//printf("\n FM or PM\n"); //LED 2
				LED_on(LED_2);
				}
				else if(h == 3) {
				//printf("\n No modulation\n"); //LED 3
				LED_on(LED_3);
				}
				else {
				//printf("\n error \n"); //LED 0
				LED_on(LED_ALL);
				}
				for(i = 0; i < N; i++) {
					x[i] = 0;
				}
				IRQ_globalEnable();
			}
		}
	}
		else {
			acq = 1;
			x[0] = sample;
			j=1;
		}

}
```

5. IMPLEMENTAZIONE, PROVE SPERIMENTALI SU DSP E STRUMENTAZIONE

La fase finale del progetto ha richiesto l'implementazione e testing del software sviluppato per il processore DSP. I test sono stati effettuati presso il Laboratorio Polifunzionale nelle postazioni di sviluppo n° 03-04 tramite la seguente strumentazione a nostra disposizione.

5.1 Strumentazione Hardware

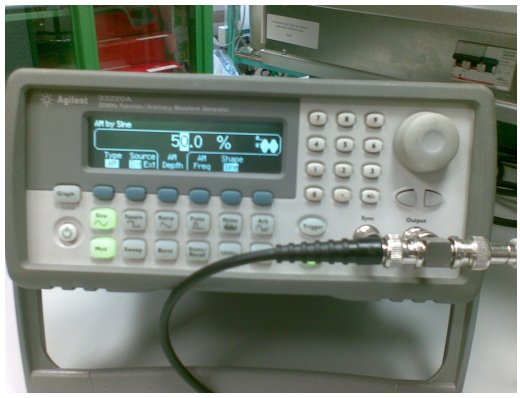
- **Texas Instrument** TMS320C6711 DSK (Scheda con processore DSP) *S/N 0247001347*



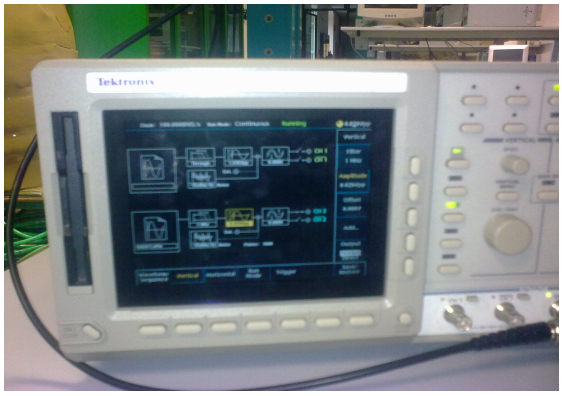
Caratteristiche principali della scheda DSK:

- processore DSP progettato per l'elaborazione real-time ed ottimizzato per diverse operazioni tra cui streaming dati, con 8 unità funzionali (2 moltiplicatori, 6 ALU) distribuite su 2 datapath
- frequenza di clock 150MHz
- TI 16 bit (ADC / DAC codec converter) AD535 con frequenza di campionamento a 8kHz e precisione 16bit
- LED's
- Daughter card expansion
- PowerSupply e Parallel Port Cable
- nessun supporto hardware per il multitastking
- 16 MB SDRAM
- 128 KB Flash ROM
- possibilità di accesso diretto alla memoria (DMA ed EDMA) 16 canali
- esegue fino ad 8 istruzioni in parallelo per ciclo
- esecuzione condizionale di tutte le istruzioni
- possibilità di calcoli a 40bit
- supporto hardware per istruzioni floating-point IEEE single-precision (32 bit) e double precision (64 bit)
- 1200 MFLOPS (Dual MACs – Multiply and Accumulate)
- Architettura di Harward (con 2 BUS) [Dati e Programmi]
- moltiplicazione intera 32x32 bit con risultato a 32 o 64 bit

- **Agilent** 33220A 20MHz Function/Arbitrary Waveform Generator (Generatore di Segnali Modulati) *S/N MY44000331*



- **Tektronix** TDS3012B (Oscilloscopio Digitale) e **Tektronix** AWG420 - Arbitrary Waveform Generator (Secondario)



5.2 Implementazione e Prove Sperimentali (Conclusione)

L'implementazione del software nel processore DSP è stata realizzata tramite ambiente di sviluppo TI Code Composer Studio v2 e le prove sperimentali sono state ottenute tramite l'utilizzo del Debug "run by step", e dei breakpoints.

Di seguito si riportano i test principali:

1) Segnale Modulato in Ampiezza **[*Risultato Soddisfacente*]**

Portante sinusoidale di f_p=2kHz ed ampiezza 2 V_{pp}
Modulante sinusoidale f_m=200Hz e profondità (depth) 40-50%

2) Segnale Modulato in Frequenza **[*Risultato Soddisfacente*]**

Portante sinusoidale di f_p=2kHz ed ampiezza 2 V_{pp}
Modulante sinusoidale f_m=200Hz e Freq. Deviation 100Hz

3) Segnale Modulato in Fase [***Risultato Soddisfacente***]

Portante sinusoidale di f_p=2kHz ed ampiezza 2 V_{pp}
Modulante sinusoidale con Phase Deviation 90° - 180°

LED_1 (on)	LED_2 (on)	LED_3 (on)	LED_ALL (off)
AM	PM/FM	No Modulation	Signal too Low

Tab 5.1

Nella Tab.5.1 è mostrata la gestione dell'output tramite l'utilizzo dei 3 LED[B8] usando la notazione presente nella documentazione BSL[1].

1. Board Support Library – insieme di Interfacce di Programmazione per Applicazioni (API) usato per configurare e controllare tutti i dispositivi della scheda DSK.

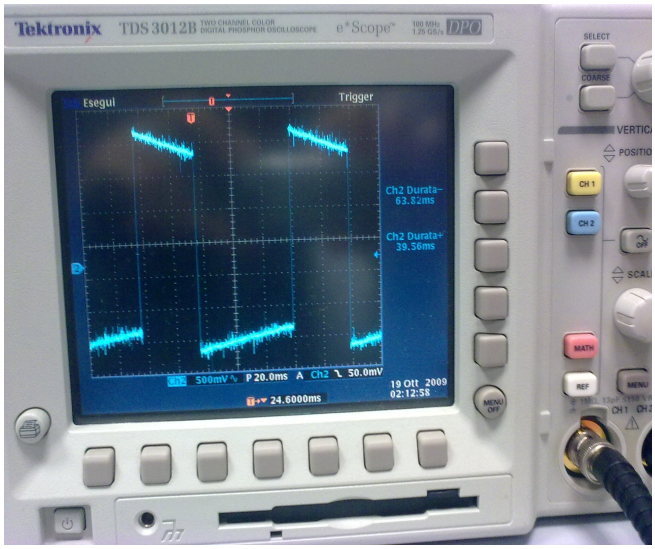
Ulteriori test hanno mostrato le seguenti limitazioni per il corretto funzionamento: frequenza portante nell'intervallo ULF [300Hz – 3kHz], con ampiezza del segnale almeno [1.3 - 2 V_{pp}], AM Depth [40 - 50%], PM Deviation [90 - 180°]. In aggiunta per verificare la robustezza del *Classificatore* ad un eventuale rumore additivo sono state effettuate altre prove aggiungendo al segnale generato con l'**Agilent** 33220A sia un rumore Uniforme/Gaussiano che uno di tipo sinusoidale sovrapposto all'originario, utilizzando il **Tektronix** AWG420.

Test di Robustezza al Rumore additivo		
Tipo Rumore	***Classificazione***	***Range****
Uniforme	Corretta	0 – 2 V
Gaussiano	Corretta	0 – 2 V
Sinusoidale	Errata (solo per il segnale Non Modulato)	≥ 1.64 V

Nota: * Dove per Range si intende l'intervallo di ampiezza del rumore additivo sul segnale generato. Tab 5.2

Dalla Tab.5.2 si evince complessivamente una robustezza al Rumore additivo del *Classificatore* sopratutto quando questo è Gaussiano oppure Uniforme, nell'intervallo 0 – 2 V (intervallo consentito dal **Tektronix** AWG420) anche se risulta compromessa ed alterata la classificazione nella sola tipologia di Non Modulazione quando il rumore accoppiato è di tipo Sinusoidale con ampiezza pari o superiore ad 1.64 V.

Infine per assicurare una corretta velocità di elaborazione si è scelto di utilizzare un numero di campioni pari a 512, visto che l'implementazione della funzione Hilbert implica un notevole utilizzo di memoria e complessità computazionale, nonostante sia stato effettuato un primo livello di ottimizzazione del codice sviluppato.



Inoltre sono stati effettuati analisi prestazionali sul tempo di esecuzione ed acquisizione, rispettivamente 40ms e 64ms (si ricorda AD535 con frequenza di campionamento a 8kHz e precisione 16bit), ottenuti osservando il tempo di On/Off dei LED tramite oscilloscopio digitale collegato alla DSK.

In conclusione, dopo aver individuato il range di funzionamento ULF ed effettuati diversi "test", si evince un'individuazione corretta e soddisfacente delle Modulazioni applicate arbitrariamente in ingresso.

NOTE BIBLIOGRAFICHE

[B1]

- Ernesto Conte: Lezioni di Teoria dei Segnali, Liguori editore.
- J. Proakis, M. Salehi: Communication systems engineering.
- Sistemi di controllo Digitale .Bonivento-Melchiorri-Zanasi. Progetto Leonardo –Bologna.

[B2]

- Claerbout, J.F., Fundamentals of Geophysical Data Processing, McGraw-Hill, 1976, pp.59-62.
- Marple, S.L., "Computing the discrete-time analytic signal via FFT," IEEE Transactions on Signal Processing, Vol. 47, No.9 (September 1999), pp.2600-2603.
- Oppenheim, A.V., and R.W. Schafer, Discrete-Time Signal Processing, 2nded., Prentice-Hall, 1998.

[B3]

- Unwrap Method © 1994-2009 The MathWorks, Inc.

[B4]

-Pearson, Karl (1894). "On the dissection of asymmetrical frequency curves". *Phil. Trans. Roy. Soc. London, Series A* **185**: 719-810.

[B5]

- J. Proakis M. Salehi, Communication Systems Engineering, Prentice Hall.

[B6]

-*Cavallo* / *Setola* / *Vasca* Liguori Editore, 2002. MATLAB - Guida all'uso
-Matlab User Guide © 1994-2009 The MathWorks, Inc.

[B7]

- Texas Instruments *TMS320C67x DSP Library Programmer's Reference Guide*
- "Il Linguaggio C" - Kerningham, Ritchie

[B8]

- Texas Instruments TMS320C6000 DSK Board Support Library API User's Guide