Sentinel 2: sistema per il monitoraggio delle radio-interferenze

C. Bortolotti, M. Cecchi, S. Montebugnoli, M. Roma

Rapporto interno IRA 294/00
Istituto di Radioastronomia – C.N.R. Bologna

Indice

Premessa.

Il problema delle radiointerferenze (RFI) diventa di giorno in giorno sempre piu' pressante per la radioastronomia sia per il continuo aumentare delle stazioni interferenti sia per il continuo aumento delle sensibilita` dei ricevitori radioastronomici. Nell' ambito del programma europeo "*Enhancing the European VLBI Network of Radio Astronomy*" (contratto FMGE-CT98-0101) si e' sviluppato un sistema di monitoraggio continuativo, a larga banda. In questo rapporto interno, preceduto da una introduzione di tipo teorico / pratico relativa alle problematiche in oggetto, verra' descritto il sistema da un punto di vista hardware e software. La filosofia di funzionamento si basa su di un approccio fondamentalmente diverso da quello degli altri sistemi di monitoraggio. Questo e' stato reso possibile dallo stato dell' arte dei processori di ultima generazione che permettono il calcolo della trasformata di Fourier, di grandi quantita` di dati in tempi estremamente brevi, fino a poco tempo fa dominio incontrastato dei grandi computer a processori paralleli.

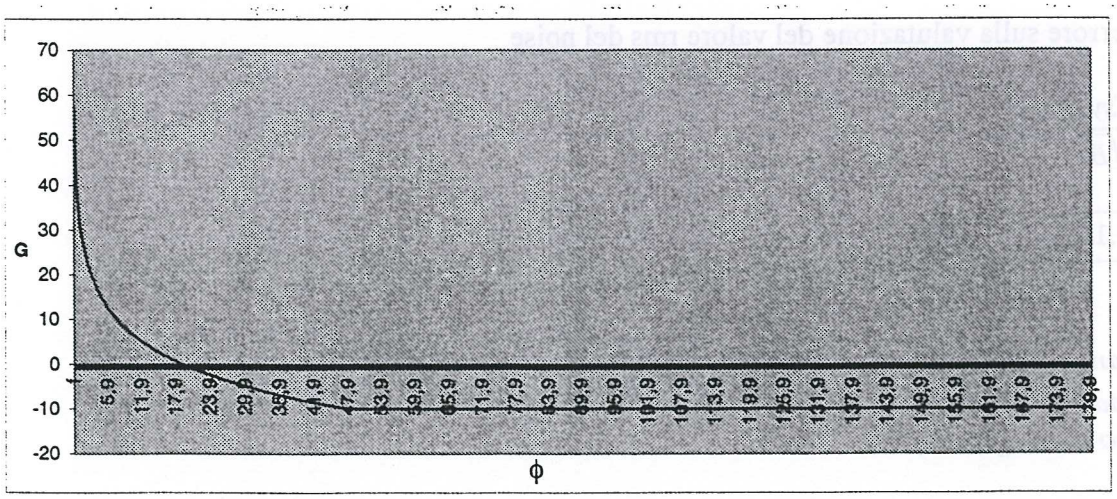
La radioastronomia e le radio interferenze.

In radioastronomia i segnali, oggetto della misura, sono estremamente deboli ed il rapporto segnale/rumore (S/N dello stadio RF o IF) generalmente va da –20 dB a –60 dB, contro il S/N>0 che si ha nel campo delle telecomunicazioni. A complicare le cose si aggiunge il fatto che questi segnali, per loro natura, hanno una probabilita' di distribuzione in ampiezza di tipo Gaussiano, per cui tendono a confondersi con il *noise* di origine artificiale (ricevitore, terra, atmosfera etc...) ed inoltre, nella stragrande maggioranza dei casi, non hanno nessun tipo di modulazione che possa aiutare a distinguerli da quest'ultimo. Si possono raggiungere i livelli di S/N sopra riportati solo per il fatto che i segnali, in uscita dagli stadi IF, vengono rivelati e integrati per periodi che possono raggiungere anche le decine di ore, introducendo una drastica riduzione del valore *rms* del *noise*.
Le osservazioni radioastronomiche vengono effettuate con antenne di dimensioni ragguardevoli. Queste assicurano risoluzioni angolari (beamwidth) che vanno da alcuni gradi a frazioni di secondi d'arco per cui la probabilità che un segnale interferente entri attraverso il *beam* principale e' molto ridotta (a meno che non si osservi a bassissime declinazioni). Questo porta a considerare solo le interferenze che entrano attraverso i lobi secondari dell'antenna. Un modello dell'inviluppo del guadagno dei lobi secondari di un paraboloide da almeno 100λ di diametro operante nella banda 2-30 GHz e` descritto dalla relazione e dal grafico seguenti:

$G = (32 - 25\log\phi)$ dBi, $1^0 < \phi < 47.8^0$

$G = -10$ dBi, $47.8^0 < \phi < 180^0$

dove ϕ rappresenta l'angolo dall'asse del beam principale (Fig.1).



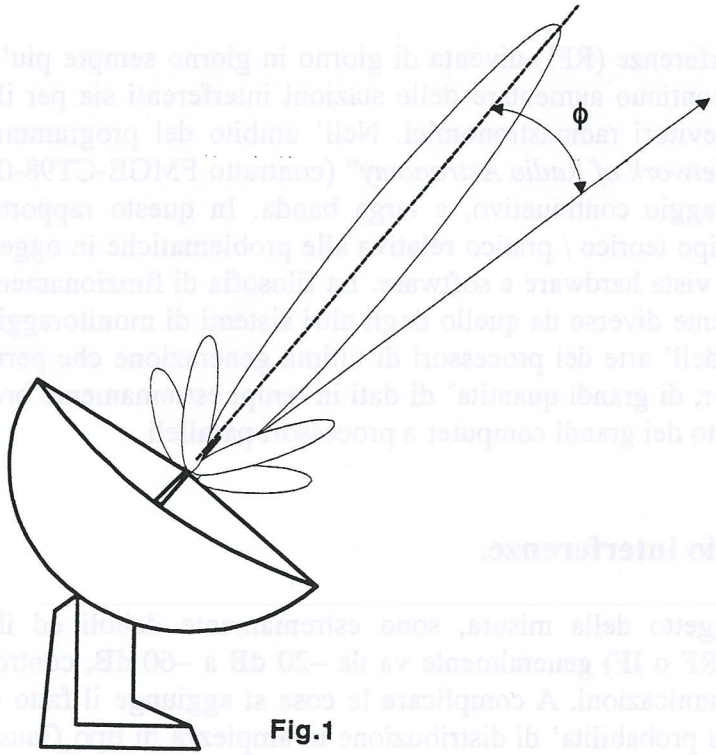


Fig.1

Il primo lobo con livello G=0 dBi, che rappresenta il guadagno di un radiatore isotropico, si incontra a

$$G = (32 - 25\log\phi) = 0$$

$$32 - 25\log\phi = 0$$

$$\phi = 10^{1.26} = 19^{o}$$

In radioastronomia si assume di valutare i possibili effetti delle interferenze in questo punto (G=0 dBi) del diagramma di antenna (nel caso di antenna paraboliche), considerando "pericoloso" quel livello di potenza del segnale interferente in ingresso che fa aumentare del 10% il valore rms della potenza di sistema in uscita. Una valutazione di questo "*Harmful Level*" della potenza di ingresso ΔP_h (W), si può ottenere tenendo conto della *densità spettrale di potenza*

$$\Delta P_s = k\Delta T \ (\mathrm{W/Hz^2}) \qquad (1)$$

e dell' errore sulla valutazione del valore rms del noise

$$\Delta T = \frac{Tsys}{\sqrt{Bt}} \ (\mathrm{K}): \qquad (2)$$

$$\boxed{\Delta P_h = 0.1\Delta P_S B \quad (\mathrm{W})} \qquad (3)$$

con:

k= *Costante di Boltzmann* 1.3810^{-23} [J/K]

B= *Banda*

T= *tempo di integrazione.*

Le interferenze possono anche essere espresse (vedi ITU-R V.574 Recommendation) in termini di *densità di flusso* $S_H B$ (W/m^2) incidente sull' antenna su tutta la banda e la *densità di flusso dello spettro di potenza* S_H $(\mathrm{W/m^2}Hz)$. Siccome i livelli interferenti sopra discussi sono dati per una antenna che presenta un guadagno nella direzione di arrivo dell' interferenza pari a quello di una antenna isotropica, nella determinazione della $S_H B$ e S_H si deve tenere conto dell' area efficace del radiatore isotropico $c^2/(4\pi f^2)$ (m^2). Il calcolo di $S_H B$ diventa quindi:

$$S_H B = \Delta P_H / Aeff_{rad.isotropico} \quad (W/m^2) \qquad (4)$$

Considerando da ora in poi la notazione logaritmica (10 log xxx dB), si ottiene:

$$S_H B = 10\log(\Delta P_H \frac{4\pi f^2}{c^2}) = 10\log\Delta P_H - 10\log c^2 + 20\log f + 10\log 4\pi$$
$$= 10\log\Delta P_H + 20\log f - 158.5 \qquad [\mathrm{dB\,(W/m^2)}] \qquad (5)$$

mentre il calcolo della S_H risulta essere:

$$S_H = S_H B - 10\log B \qquad [\mathrm{dB\,(W/m^2}Hz)] \qquad (6)$$

Applicando la (1),(2),(3), (4), (5), e (6) e` possibile calcolare gli *Harmful Level,* a 2000 secondi di integrazione, per la parabola di Medicina alle frequenze operative fino a 10 GHz.

f (cent. bw)	B	Ta	Tr	ΔT	ΔPs	ΔPh	ShΔf	Sh
MHz	MHz	K	K	mK	dB(W/Hz)	db(W)	dB(W/m^2)	DB(W/m^{2}H)
1420	80	5	50	0.14	-267	-198	-174	-252
1665	80	5	60	0.16	-266	-197	-172	-251
2280	160	5	45	0.8	-259	-197	-168	-250
4900	400	6	40	0.05	-256	-195	-160	-246
8580	800	10	30	0.02	-274	-195	-154	-243

Dove:

f	=	Centro Banda
B	=	Larghezza di Banda
Ta	=	Temperatura di antenna
Tr	=	Temperatura del Ricevitore
ΔT	=	System Sensibility (noise fluctuation)
ΔPs	=	System Sensibility (Power spectral Density
ΔPh	=	Input Power
ShΔf	=	Power Flux Density
Sh	=	Spectral Power Flux Density

Sentinel 2: sistema per il monitoraggio delle radio-interferenze.

Introduzione

Negli ultimi anni l'inquinamento elettromagnetico è aumentato vertiginosamente a causa di un utilizzo sempre più diffuso di tecniche radio che vanno dalle trasmissioni dati alle telecomunicazioni. Nonostante le osservazioni astronomiche vengano compiute in determinate zone dello spettro di frequenza riservate allo scopo, la cui tutela è garantita da organi di controllo mondiali, sono ormai sempre più frequenti i casi di interferenze che vanno a ricadere in queste bande denominate 'radioastronomiche' provocando seri disturbi alle osservazioni. Si è stimato che se il trend di aumento del traffico radio rimarrà di questo passo, entro quindici anni risulterà praticamente impossibile effettuare osservazioni astronomiche per la maggior parte dei radiotelescopi.

Sensibile all'importanza di questo problema, la comunità europea del VLBI ha deciso di intraprendere un progetto orientato allo studio delle radiointerferenze (RFI) e mirato allo sviluppo di nuove tecnologie e strategie di osservazione. A questo programma europeo, denominato in codice RTD ERBFMGECT 980101, partecipano diverse nazioni facenti parte della CE, ciascuna delle quali ha l'incarico di svolgere un determinato compito (Fig 0.1). Il programma è articolato in quattro sottoprogetti principali.

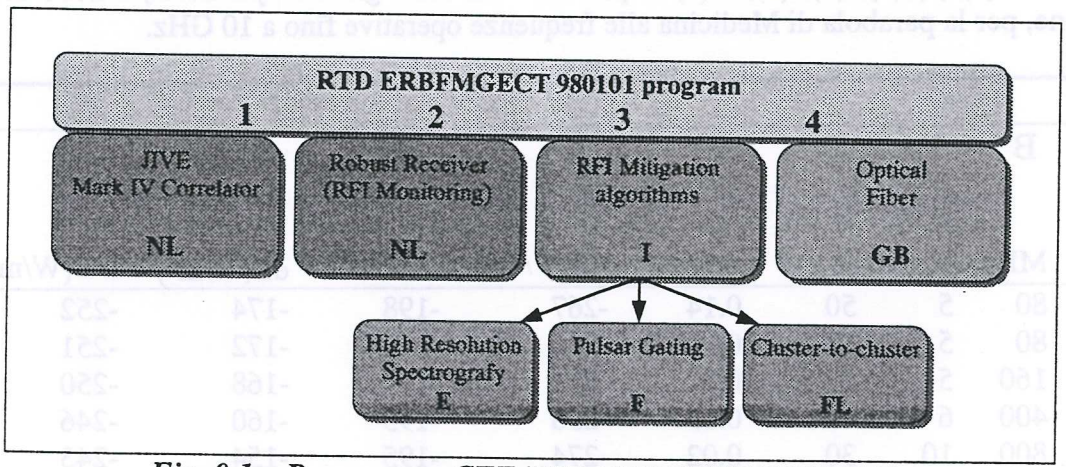


Fig. 0.1 – Programma CEE RTD ERBFMGECT 980101.

Questi sono le tematiche affrontate dai sottoprogetti:

1. Sviluppo di metodologie osservative che prevedono l'uso del moderno correlatore denominato *Mark IV*;
2. Sviluppo di ricevitori "robusti" alle radiointerferenze;
3. Studio di algoritmi per la mitigazione degli effetti delle RFI sulle osservazioni radioastronomiche, è a sua volta suddiviso in 3 sottogruppi:

    3a. Tecniche osservative di spettroscopia ad alta risoluzione;
    3b. Sviluppo di tecniche di osservazione delle pulsar con la rete VLBI (*pulsar gating*);
    3c. Sviluppo di tecniche osservative *cluster-cluster*;

4. Studio di connessioni tra apparecchiature radioastronomiche per mezzo di fibre ottiche.

All'Istituto di Radioastronomia è stata assegnato il sottoprogramma n°3 relativo allo studio di algoritmi di mitigazione delle RFI. Per eseguire nel migliore dei modi questo compito è necessario effettuare osservazioni sistematiche e continuative dello spettro radio al fine di analizzare, catalogare e classificare l'attuale panorama delle RFI.

Sentinel 2 è un sistema di monitoraggio per RFI che, in questo senso, ci permette di effettuare una completa survey sulla natura delle interferenze nella stazione radioastronomica di Medicina.

I dati provenienti delle suddette attività di monitoraggio risulteranno inoltre utili per la costruzione del 'robust receiver'; Sentinel 2 apporterà quindi un sostanziale contributo anche al sottoprogramma n°2 nel quale l'Istituto di Radioastronomia collabora già con la stazione di Noto.

Gli obbiettivi principali del progetto Sentinel 2 sono così riassumibili:

1. Studiare la natura ed il panorama delle interferenze nella stazione radiostronomica di Medicina;
2. Apportare una proficua collaborazione al sottoprogramma n°2 del progetto RTD;
3. Permettere un miglior approccio al progetto degli algoritmi di mitigazione di RFI, nell'ambito del sottoprogramma n°3.
4. Supportare i locali "cacciatori" di RFI nelle loro quotidiane attività;

Prima della costruzione di Sentinel 2 il monitoraggio automatico delle interferenze a Medicina veniva effettuato mediante l'impiego del Serendip IV e del Sentinel 1. S IV è lo spettrometro che opera in *piggy-back* sull'antenna parabolica; un sistema siffatto riesce a rilevare solo interferenze di banda strettissima eventualmente presenti alla stessa frequenza e puntamento in cui si svolgono le osservazioni radioastronomiche previste. Sentinel 1 era invece un sistema prototipale con 10 MHz di banda e prestazioni e funzionalità inferiori a quelle del Sentinel 2.

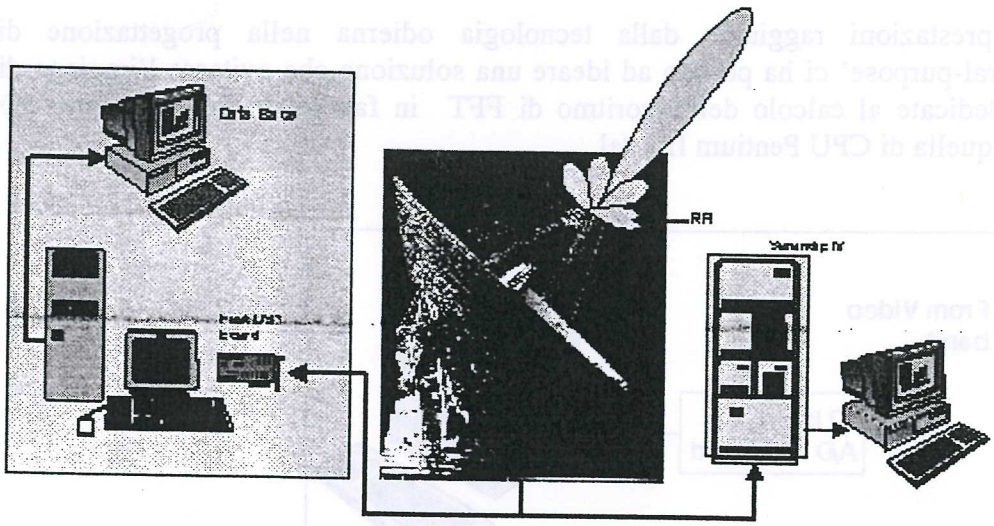


Fig. 0.2 – Precedenti sistemi di monitoraggio RFI (Sentinel 1 e Serendip IV)

Capitolo 1.
Descrizione funzionale del sistema Sentinel 2.

1.1 Architettura e specifiche progettuali

L'esperienza acquisita durante la progettazione e lo sviluppo del sistema "Sentinel 1" ci ha portato a concepire una nuova architettura che ne rappresenta un'evoluzione ed un raffinamento. La filosofia che sta alla base dei due sistemi è la medesima ma i sistemi hardware e software sono differenti.

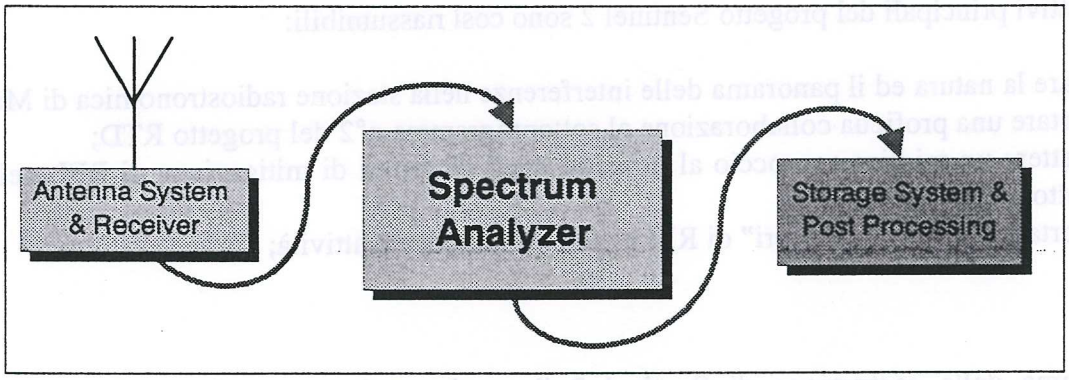


Fig 1.1. – Schema a blocchi della struttura dei sistemi Sentinel 1 e 2.

Lo schema di Fig. 1.1 mostra la struttura generale del sistema. Come si vede, l'applicazione è costituita dallo spettrometro che analizza i segnali provenienti da uno stadio di ricezione. Gli spettri così elaborati vengono quindi trasferiti all'applicazione di visualizzazione che funziona *off-line*.

Le crescenti prestazioni raggiunte dalla tecnologia odierna nella progettazione di microprocessori 'general-purpose' ci ha portato ad ideare una soluzione che evitasse l'impiego di costose schede DSP dedicate al calcolo dell'algoritmo di FFT in favore di un'architettura più scalare e diffusa come quella di CPU Pentium II Intel.

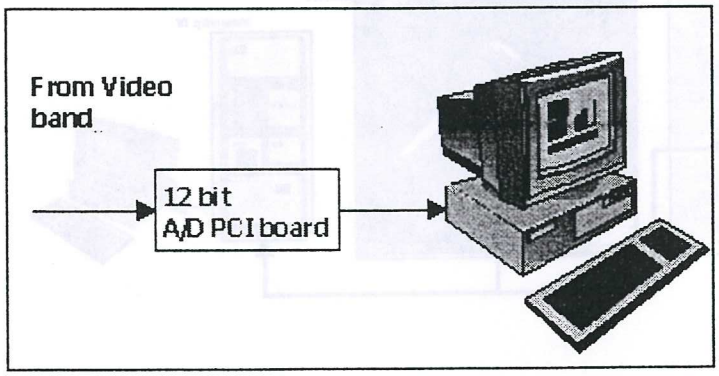


Fig 1.2 – Sentinel 2 lato hardware

La strumentazione necessaria per la costruzione del sistema radio-frequenza, denominato *front-end*, comprende le antenne, il ricevitore e in genere tutto ciò che necessita per captare un

segnale radio e per convertirlo dalla frequenza originale in banda base; è sicuramente la parte più costosa.
La strumentazione Hardware e tutto il Software necessario per l'elaborazione del segnale digitalizzato costituiscono il *back-end*. Un convertitore A/D, ospitato da un PC su bus PCI, campiona il segnale d'ingresso in banda base che proviene dalla parte RF; i dati sono trasferiti in blocchi discreti dalla scheda A/D alla memoria del PC, da qui vengono calcolati gli spettri di potenza per ogni serie di dati di ogni blocco (Fig. 1.2).
Il back-end rappresenta il cuore dell'intero sistema e rimane praticamente lo stesso indipendentemente dalle specifiche di progetto della parte RF; la soluzione proposta, essendo composta da elementi facilmente reperibili sul mercato in quanto progettati per scopi generici e non specificatamente per questo particolare campo di applicazione, costituisce indubbiamente una soluzione flessibile, modulare e soprattutto economica.

Il sistema RFI Sentinel2 realizzato alla stazione radioastronomica di Medicina è composto, nella fattispecie, dal seguente hardware di back-end:

- n.1 convertitore A/D modello ADDA1280 della UltraView da 40Ms/s, 12-bit con 4Mb di memoria a bordo. I 12 bit permettono di arrivare fino a circa 72dB di dinamica.

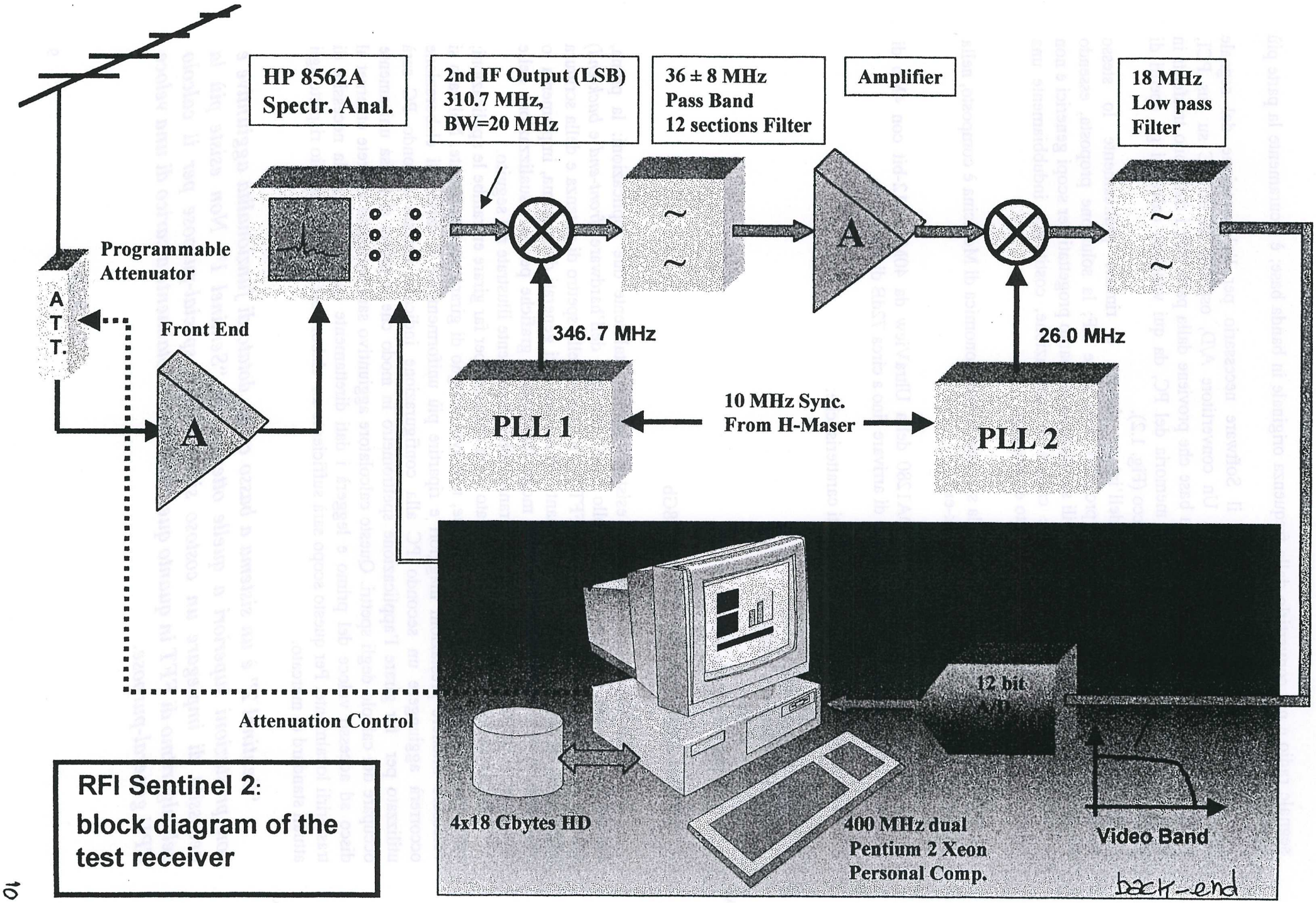
- n.1 Personal computer con le seguenti caratteristiche:

  scheda madre Intel Xeon
  n. 2 Pentium II a 400MHz
  128Mb di Ram
  n. 1 disco rigido da 5.0 GB

- n.1 Hard-Disk ultra-wide SCSI 2 da 18Gb

Dal punto di vista del software, esistono fondamentalmente due applicazioni: la prima, funzionante *on-line*, si occupa del controllo e del pilotaggio dell'hardware (*front-end* e *back-end*) nonché dell'elaborazione dell'algoritmo FFT per il calcolo dello spettro di potenza e della scrittura degli stessi su disco, la seconda, funzionante *off-line* sui dati generati dalla prima, implementa lo spettrometro vero e proprio e provvede molte funzionalità grafiche per la visualizzazione delle interferenze. Entrambe le applicazioni verranno abbondantemente illustrate nel seguito.
In questa configurazione il PC indicato verrà utilizzato per far girare entrambe le applicazioni; visto che la seconda, che funziona *off-line*, non avrà bisogno di girare continuamente. Nel caso si volessero ottenere prestazioni maggiori e ripartire più uniformemente il carico di elaborazione occorrerà aggiungere un secondo PC alla configurazione indicata. Questo secondo PC sarà utilizzato per far girare l'applicazione spettrometro in modo che il primo si possa unicamente occupare del calcolo degli spettri. Questo calcolatore aggiuntivo sarà collegato via rete ethernet al disco ad accesso veloce del primo e leggerà i dati direttamente da esso, senza la necessità di trasferirli localmente. Per questo scopo sarà sufficiente un elaboratore di livello medio rispetto agli attuali standard di mercato.

"Sentinel 2" è un sistema a basso costo, dotato di funzionalità aggiuntive e con prestazioni superiori a quelle ottenute da "Sentinel 1". Non esiste più la necessità di impiegare un costoso sistema DSP special-purpose per il calcolo dell'algoritmo di FFT in quanto questo compito è totalmente a carico di una veloce CPU general-purpose.



RFI Sentinel 2:
block diagram of the test receiver

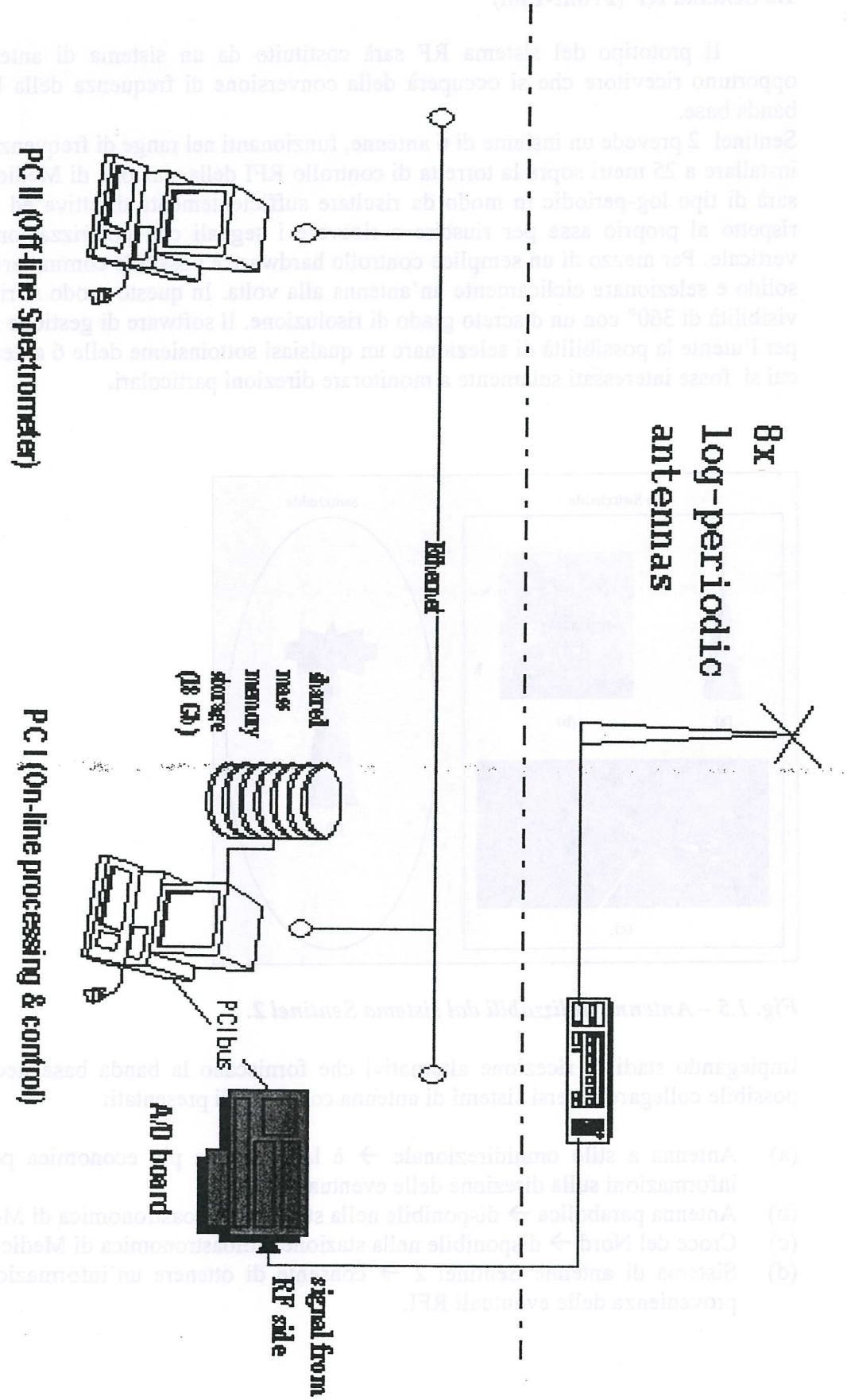


Fig 1.4 Diagramma a blocchi del back - end

1.2 Sistema RF (Front-End)

Il prototipo del sistema RF sarà costituito da un sistema di antenne affiancato da un opportuno ricevitore che si occuperà della conversione di frequenza della banda analizzata nella banda base.
Sentinel 2 prevede un insieme di 6 antenne, funzionanti nel range di frequenze 300MHz- 2 GHz, da installare a 25 metri sopra la torretta di controllo RFI della stazione di Medicina. Ciascuna antenna sarà di tipo log-periodic in modo da risultare sufficientemente direttiva ed andrà inclinata di 60° rispetto al proprio asse per riuscire a ricevere i segnali con polarizzazione sia orizzontale che verticale. Per mezzo di un semplice controllo hardware è possibile commutare uno switch allo stato solido e selezionare ciclicamente un'antenna alla volta. In questo modo si riuscirà ad ottenere una visibilità di 360° con un discreto grado di risoluzione. Il software di gestione del Sentinel 2 prevede per l'utente la possibilità di selezionare un qualsiasi sottoinsieme delle 6 antenne nell'eventualità in cui si fosse interessati solamente a monitorare direzioni particolari.

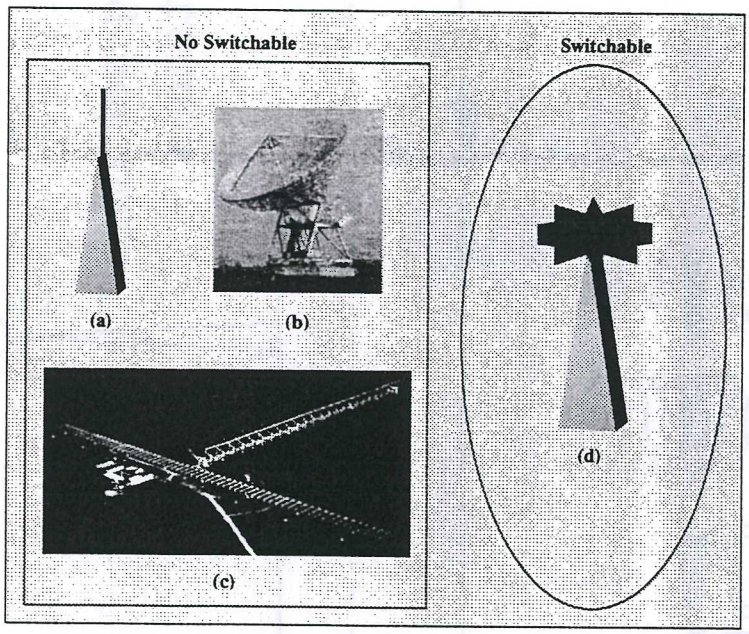


Fig. 1.5 – Antenne utilizzabili dal sistema Sentinel 2.

Impiegando stadi di ricezione alternativi che forniscano la banda base necessaria al Sentinel, è possibile collegare diversi sistemi di antenna come quelli presentati:

(a) Antenna a stilo omnidirezionale → è la soluzione più economica possibile, non fornisce informazioni sulla direzione delle eventuali RFI;
(b) Antenna parabolica → disponibile nella stazione radioastronomica di Medicina;
(c) Croce del Nord → disponibile nella stazione radioastronomica di Medicina;
(d) Sistema di antenne Sentinel 2 → consente di ottenere un'informazione sulla direzione di provenienza delle eventuali RFI.

1.3 Conversione in banda base

Lo stadio di ricezione deve sostanzialmente convertire il segnale a radiofrequenza in banda base, per renderlo acquisibile dalla scheda A/D. In attesa di poter installare il prototipo le cui specifiche progettuali sono state esposte al paragrafo precedente, la banda RF da monitorare viene attualmente fornita dal locale ***centro di ascolto interferenze*** e proviene dal sistema ricevente installato sulla torre, in particolare dal blocco a larga banda 1-2.5 GHz costituito da:

- Antenna parabolica 1.2 mt. con polarizzazione 45° (+14dB ca.)
- Sistema amplificazione/filtraggio e cavo di discesa (+36dB ca.).

Il segnale risulta inoltre attenuato in modo rilevante dal cavo coassiale che lo trasmette al locale dove sono state effettuate le prove.

Per fornire il segnale in banda base al convertitore A/D (massima banda campionabile 20 MHz) si è realizzato un prototipo di conversione della banda RF, già opportunamente amplificata e filtrata, che prevede l'utilizzo di un Analizzatore di Spettro con ***second IF out*** a 310.7 MHz (BW min. 20 MHz) di cui vi era già disponibilità.
Questo strumento, un **HP 8562A**, utilizzato nella funzione ***zero span,*** fornisce una conversione "ribaltata" centrata a 310.7 MHz, indipendente dal settaggio ***Resolution BW*** ed ***Amplitude*** ma sensibile ai parametri ***Frequency*** ed ***Input attenuation***; esso, attraverso link HP-IB, viene comandato dal PC (Sentinel2) che ne seleziona la frequenza centrale.

Tale uscita, opportunamente filtrata, viene convertita e "ribaltata" (LSB) con frequenza centrale a 36 MHz (BW utile 15 MHz) e di nuovo filtrata e convertita (USB) in banda base: banda utile 2.5-17.5 MHz.
Il centro frequenza dell'analizzatore di spettro viene così riportato a 10 MHz e la banda è disponibile nel "verso" corretto.

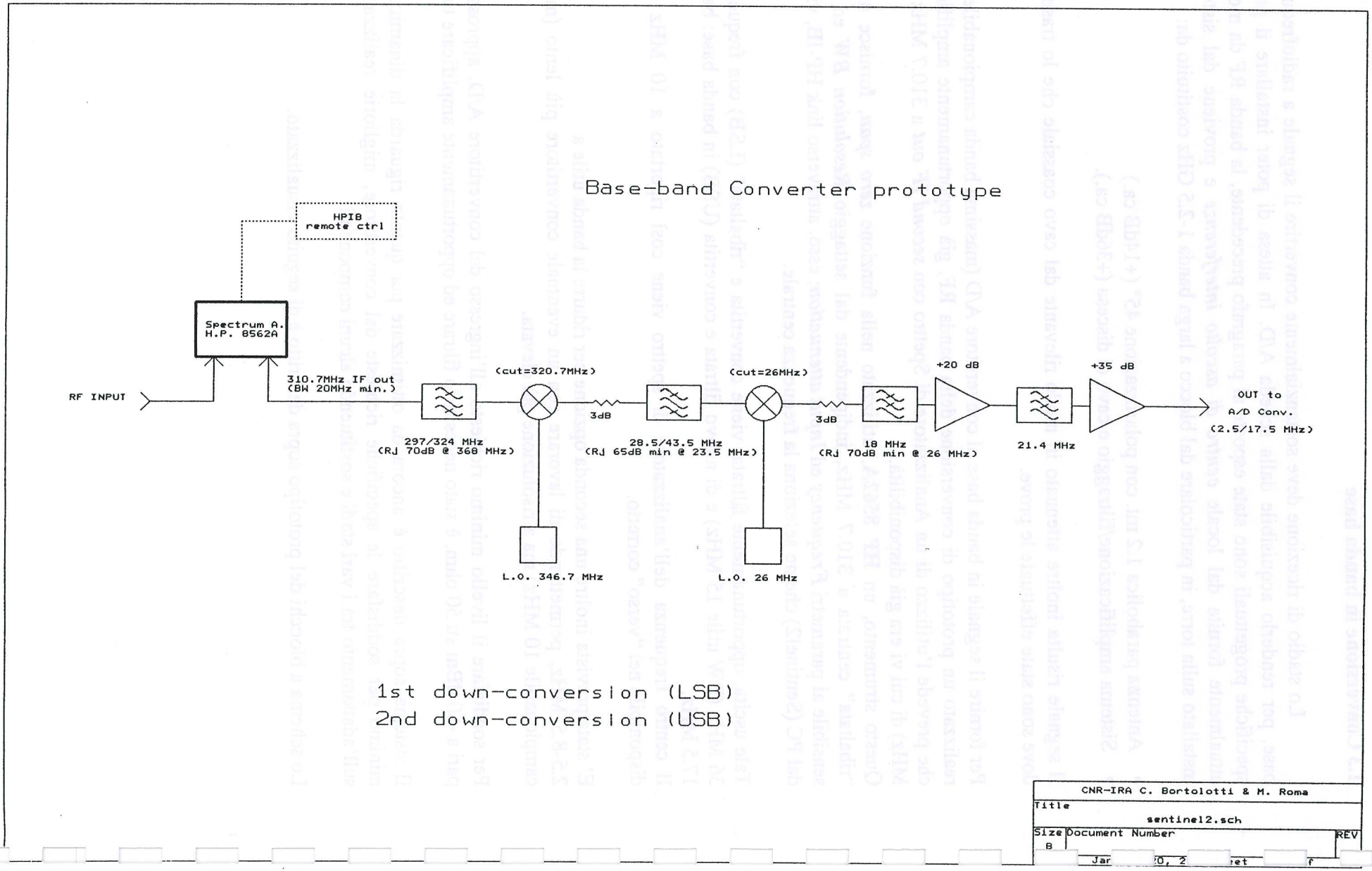
E' stata prevista inoltre una seconda opzione per ridurre la banda utile a
2.5-8.5 MHz, permettendo di lavorare con un eventuale convertitore più lento (massima banda campionabile 10 MHz) ma a risoluzione più elevata.

Per soddisfare il livello minimo richiesto all'ingresso del convertitore A/D, approssimativamente pari a -50 dBm su 50 ohm, è stato necessario filtrare ed opportunamente amplificare tale segnale.

Il sistema sopra descritto è ancora da ottimizzare per quanto riguarda la dinamica ed il livello minimo per soddisfare le specifiche richieste dal convertitore, migliorie realizzabili lavorando sull'adattamento tra i vari stadi e sostituendo alcuni componenti.

Lo schema a blocchi del prototipo sopra descritto è di seguito visualizzato.

Fig. 1.4 - Schema della fase di conversione del segnale RF.



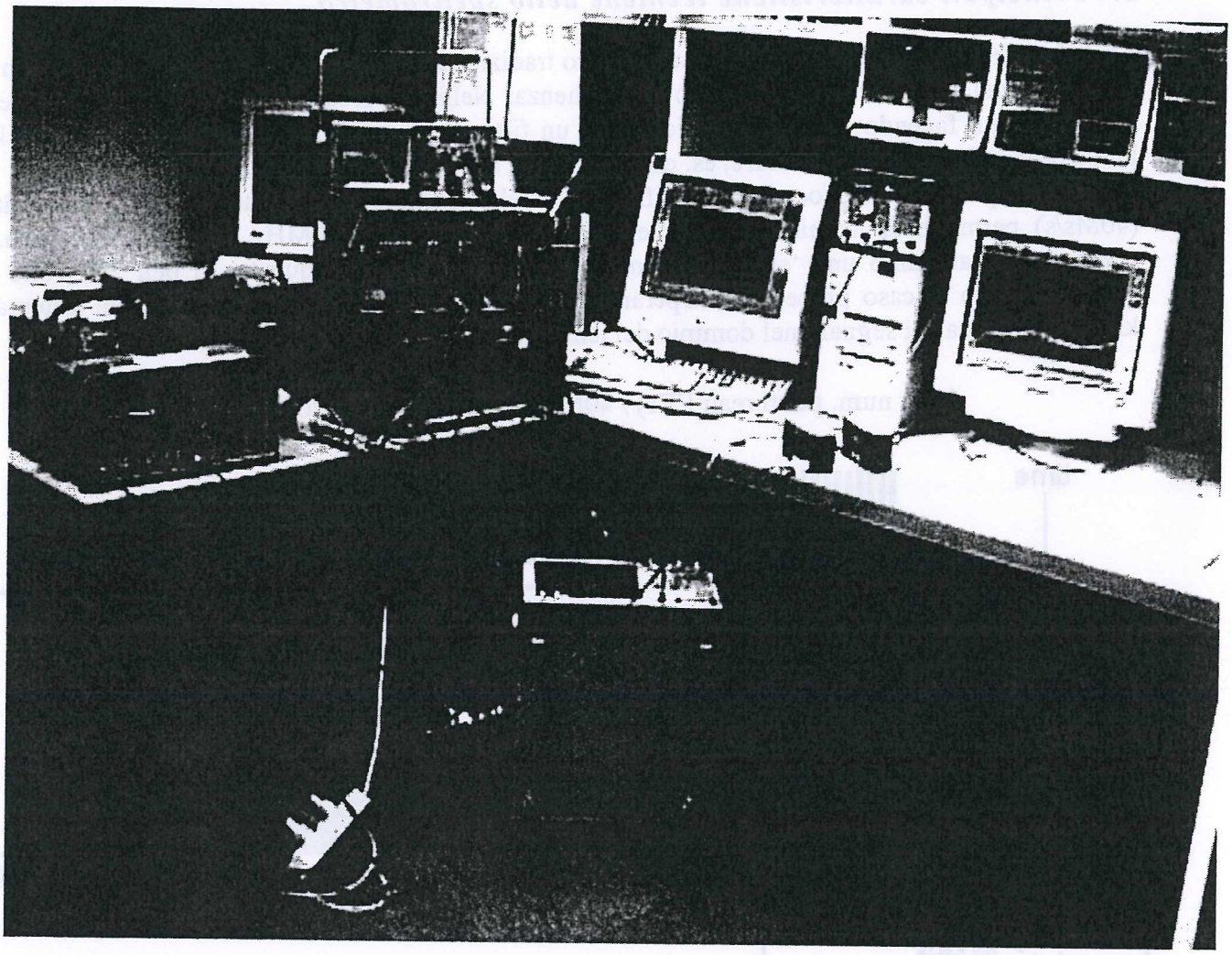


Fig.1.4bis - La strumentazione utilizzata per la conversione del segnale il banda base (Medicina, stanza del ricevitore)

1.4 Principali caratteristiche tecniche dello spettrometro

La differenza fondamentale tra uno spettrometro tradizionale ed il Sentinel consiste nel modo in cui il segnale nel tempo viene convertito in frequenza. Nelle apparecchiature tradizionali lo spettro viene ricavato facendo ciclare periodicamente un filtro della risoluzione dell'ordine dei 100 kHz lungo tutta la banda da monitorare, es. come in figura da 100Mb a 2,3 GHz.
Sentinel 2 opera con un procedimento differente. L'alta sampling rate di cui è dotato il convertitore (40Ms/s) permette di acquisire in maniera pressoché istantanea 20MHz di banda alla volta. In questo modo abbiamo una 'snapshot', una fotografia del segnale aquisito, di ben 200 volte più grande rispetto al caso precedente. Operando con la massima risoluzione prevista dal sistema (5 kHz), ogni 'fetta' di segnale nel dominio del tempo verrà campionata, al massimo, in:

(max num. punti reali FFT) / 40MHz = 8192*25 ns = 205 μs

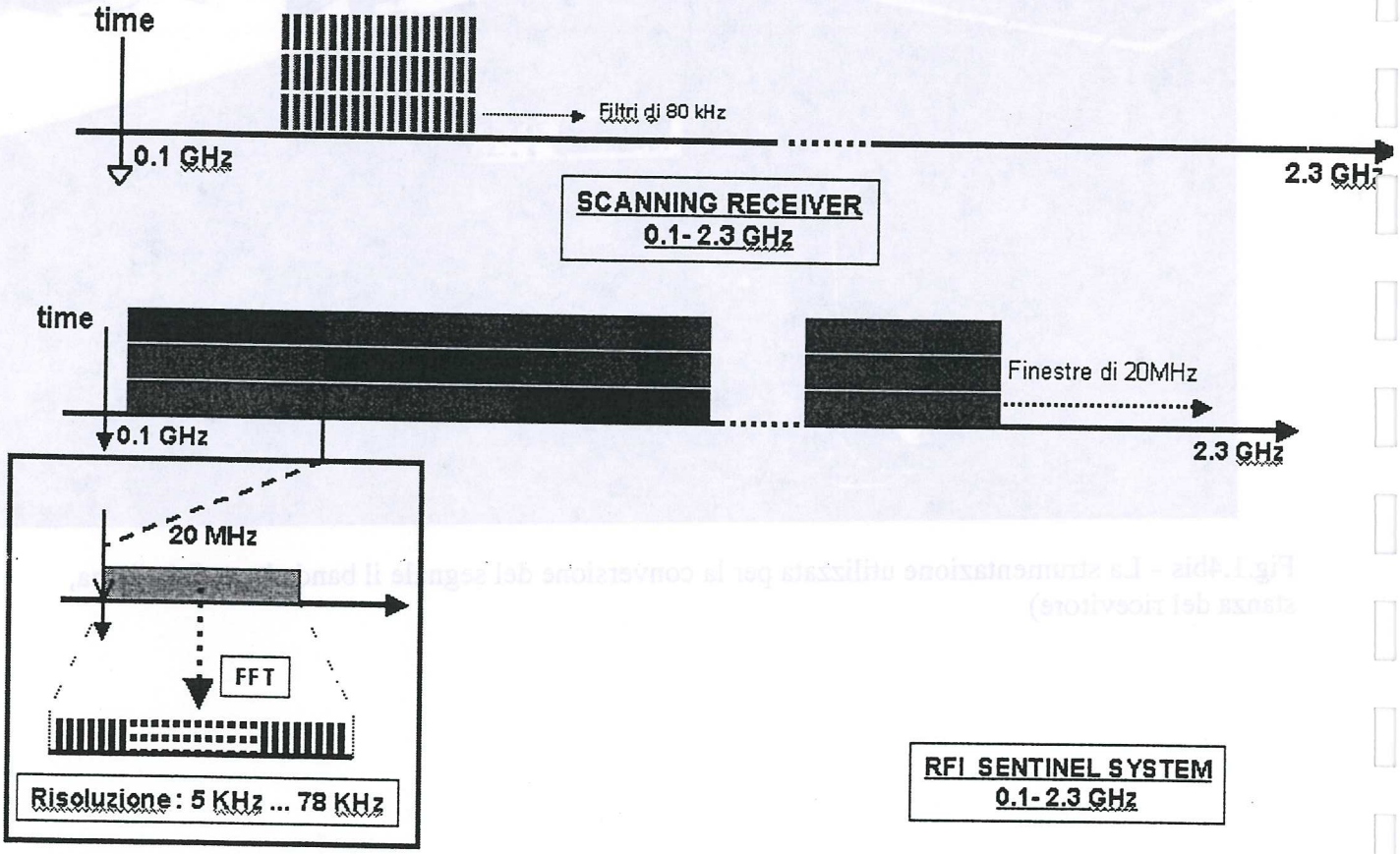


Fig. 1.6 Differenza tra il funzionamento di uno scanning receiver e Sentinel 2.

Ovviamente l'analisi spettrale di una banda della larghezza di 20MHz richiede un certo periodo di tempo, che varia con la risoluzione adottata, per il calcolo dell'algoritmo. Il sistema non funziona quindi in *real-time* in quanto durante il tempo richiesto per il calcolo dello spettro non verranno effettuati campionamenti del segnale; questo 'buco' introdurrà quindi un *duty-cycle*. Non è comunque necessario che il sistema funzioni in *real-time*, ciò che importa è che durante il periodo di elaborazione (*work-cycle*) il segnale da esaminare venga acquisito in un tempo brevissimo in modo da avere una fotografia il più possibile istantanea del segnale al momento del campionamento; in Sentinel 2 questa caratteristica è abbondantemente garantita dalla velocità di campionamento del convertitore.

Ogni *work-cycle* è composto dalle seguenti operazioni:

1. **acquisizione dati**
2. **trasferimento in blocchi dalla memoria della scheda A/D alla memoria del PC**
3. **calcolo dello spettro complesso (algoritmo FFT radix-2)**
4. **calcolo potenze e medie**
5. **scrittura dati su HD**
6. **pilotaggio frequenza dell'analizzatore di spettro via IEEE 488 ed eventuale selezione antenna**
7. **visualizzazione e gestione grafica**

La tabella che segue mostra, al variare della risoluzione e del numero di spettri da elaborare e mediare in blocco, i tempi necessari per l'esecuzione di un singolo work-cycle e per l'esecuzione di un ciclo completo di monitoraggio con antenna monodirezionale nel range di frequenza da 100MHz a 1.91 GHz.

Canali	Risoluzione		Numero di spettri mediati		
			256	128	64
256	78 kHz	1 work-cycle (msecs)	2,57	3,91	6,34
		Sweep time (secs)	66	52	37
512	39 kHz	1spectrum (msecs)	3,84	5,25	7,81
		Sweep time (secs)	96	68	53
1024	19.5 kHz	1spectrum (msecs)	6,71	7,94	10,5
		Sweep time (secs)	162	98	68
2048	10 kHz	1spectrum (msecs)	13,79	14,28	17,09
		Sweep time (secs)	325	172	106
4096	4.9 kHz	1spectrum (msecs)	X	25,88	28,56
		Sweep time (secs)	X	305	172

Come è possibile notare le prestazioni migliori per singolo *work-cycle* si ottengono con un alto numero di spettri; questo si verifica perché il throughput del sistema è ottimizzato quando la memoria del convertitore è sfruttata al massimo visto che i tempi necessari per il calcolo del post-processing di visualizzazione, per la scrittura su HD, per il pilotaggio delle frequenze del ricevitore e per la programmazione delle direzioni delle antenne sono costanti e non variano con il numero di spettri da calcolare ad ogni *work-cycle*.

Tutti i dati immagazzinati sono strutturati all'interno di un database su cui è possibile operare *off-line*, al fine di produrre un'accurata mappa delle interferenze radio rilevate.
Le risorse hardware e software impiegate per l'intero progetto sono raccolte nella Sentinel 2 prevede due maniere di monitoraggio del dominio delle frequenze:

(a) Scansione continua dell'intero dominio (range) di frequenze;
(b) Scansione locale attorno alle bande di interesse.

Nel caso (a) si ottengono informazioni complete su ciascuna frequenza ma si impiega molto tempo. Il caso (b) permette di ottenere meno informazioni ma consente una gestione più efficiente del tempo-macchina.

Il sistema è in grado di rilevare tutti i tipi di interferenza conosciuti con un intensità (all'uscita della catena del ricevitore) compresa tra –14dBm e 51 dBm. Attualmente i casi limite, dettati dall'esperienza, sembrano essere i seguenti:

- interferenze deboli → è necessario un lungo tempo di integrazione per la loro rilevazione;
- interferenze di tipo *spread-spectrum* → si tratta generalmente di comunicazioni militari ad alta sicurezza; non solo non occorre integrazione per rilevarle ma occorre far funzionare lo spettrometro nella configurazione più rapida possibile.

Il sistema Sentinel 2, riassumendo, rappresenta una soluzione flessibile e completa con le seguenti principali caratteristiche:

- larghezza di banda di ogni 'finestra': 20MHz
- dinamica di 65 dB (-14dBm, 51 dBm);
- monitoraggio di tutte le frequenze da 100MHz a 2.35 GHz con una buona efficienza in tempo;
- software dedicato per l'analisi e la visualizzazione grafica degli spettri funzionante con integrazione off-line.
- è un sistema modulale, scalabile e facilmente aggiornabile;
- ha un costo ragionevole facendo uso di componenti Hw general-purpose e ambineti di sviluppo Sw molto diffusi sul mercato (vedi tabella che segue).

Hardware	Software
1 Personal computer biprocessore	Windows NT 4.0 + service pack 5.0
1 Hard-Disk ultra-SCSI II da 18GB	MS Visual Studio 6.0
1 Personal computer monoprocessore (opzionale)	
1 A/D converter UltraD 1280 40MHz	

Tab. 1.2 – Risorse h/w e s/w richieste dal back_end del sistema Sentinel 2.

Capitolo 2

Descrizione del software per la gestione ed il controllo.

2.1 Architettura del software

Il software necessario per la realizzazione dello spettrometro RFI Sentinel 2 è stato concepito e sviluppato completamente *ex-novo,* utilizzando tecniche di programmazione avanzata al fine di ottenere elevate prestazioni in termini di velocità di elaborazione e di rendere più agevole possibile per l'utente la programmazione della apparecchiature controllate. La progettazione di questo SW rappresenta indubbiamente una componente su cui è stato compiuto un lavoro approfondito e complesso.

Il S/W è composto da due applicazioni:

- *"on-line processing & control"*
- *"off-line spectrometer"*

La prima applicazione funziona on-line, ossia è continuamente in esecuzione ed è parte integrante del ciclo operativo del sistema. Si occupa fondamentalmente di coordinare e controllare il front-end ed il back-end sistema di acquisire i segnali, ricavarne lo spettro di potenza e salvarli su disco.
Questo software funziona su sistema operativo Windows NT 4.0 ed e' stato sviluppato in C++ utilizzando l'ambiente di sviluppo Microsoft Visual C++ 6.0. L'interfaccia grafica permette all'utente di impostare rapidamente i parametri di configurazione e di tenere monitorata l'attività del sistema.

La seconda applicazione implementa una spettrometro grafico per l'analisi e la visualizzazione delle interferenze e, al contrario della prima funziona *off-line*, ossia va eseguita solamente quando si voglia consultare l'archivio spettri che viene generato dalla prima. Funziona su sistema operativo Windows 95/98 o NT 4.0 ed e' stata sviluppata in Microsoft Visual Basic 6.0 impiegando la libreria Olectra Chart 6.0 per quanto riguarda la gestione della visualizzazione grafica degli spettri.

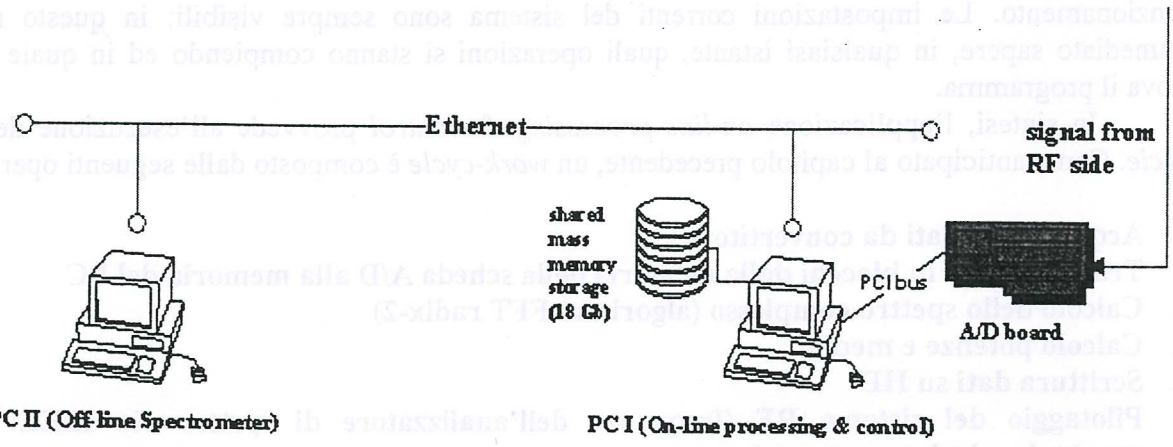


Figura 2.1 Dettaglio dell'architettura di back-end che illustra una possibile distribuzione delle applicazioni

2.2 L'applicazione "*on-line processing & control*"

All'avvio del programma, l'operatore si troverà di fronte ad una videata del tipo di Fig. 2.2

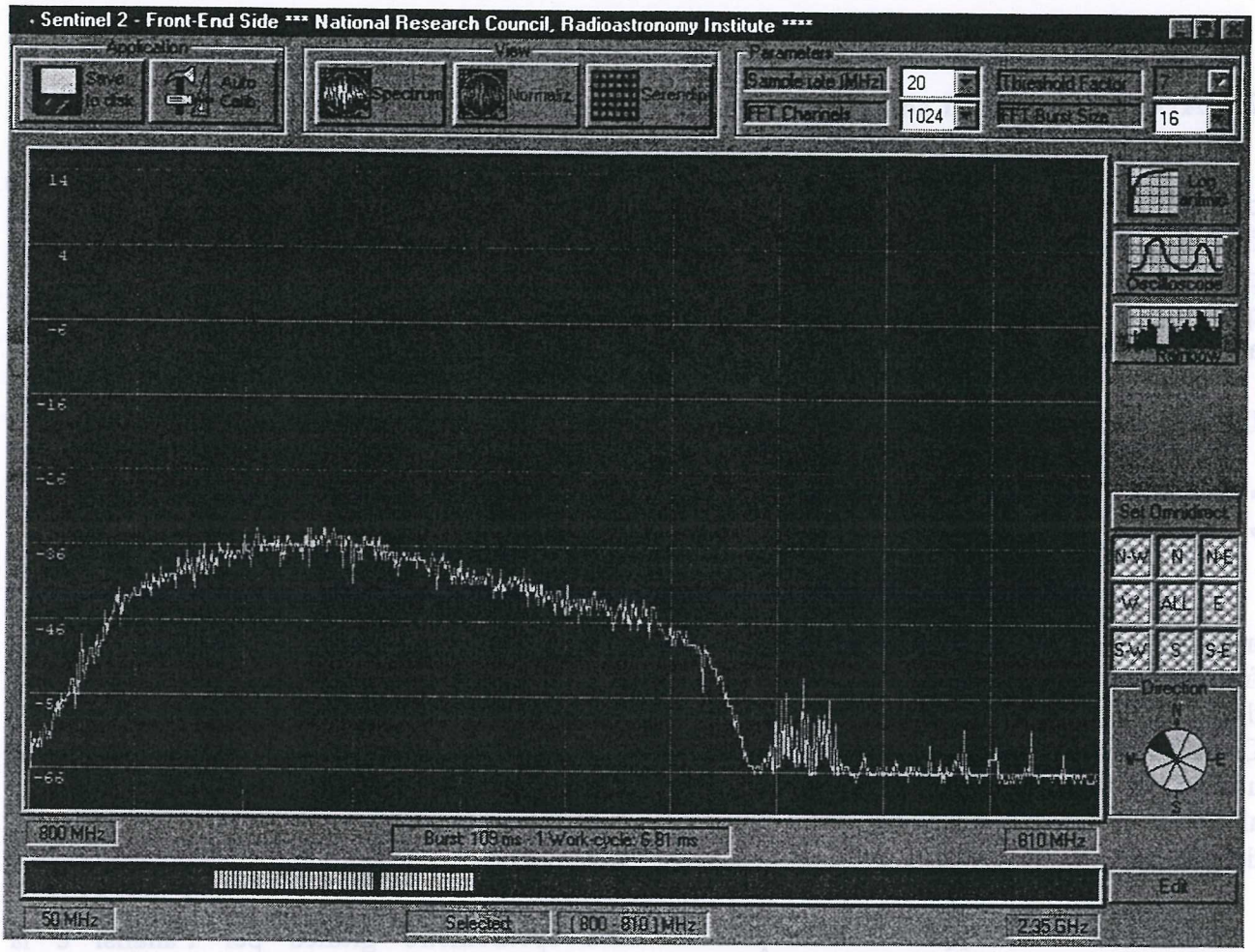


Fig.2.2 – Interfaccia utente dell'applicazione 'on-line processing & control'.

I comandi sono disposti lungo la parte superiore e la parte destra dello schermo e risultano raggruppati in base alla loro funzione. Nella zona centrale viene visualizzato lo spettro del segnale acquisito e nella parte bassa vi sono altre informazioni che specificano la situazione attuale di funzionamento. Le impostazioni correnti del sistema sono sempre visibili; in questo modo è immediato sapere, in qualsiasi istante, quali operazioni si stanno compiendo ed in quale stato si trova il programma.

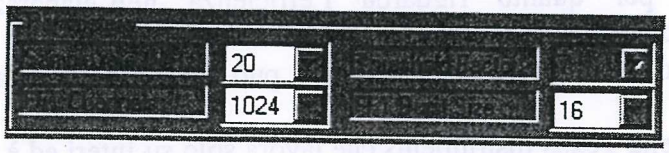
In sintesi, l'applicazione *on-line processing & control* provvede all'esecuzione del *work-cycle*. Come anticipato al capitolo precedente, un *work-cycle* è composto dalle seguenti operazioni:

1. **Acquisizione dati da convertitore**
2. **Trasferimento in blocchi dalla memoria della scheda A/D alla memoria del PC**
3. **Calcolo dello spettro complesso (algoritmo FFT radix-2)**
4. **Calcolo potenze e medie**
5. **Scrittura dati su HD**
6. **Pilotaggio del sistema RF (frequenza dell'analizzatore di spettro via IEEE 488 ed eventuale selezione antenna)**
7. **Algoritmi di visualizzazione e dell'interfaccia grafica**

Passiamo ora ad esaminare nel dettaglio queste sette fasi:

2.2.1 Acquisizione dati da convertitore

I dati possono venire acquisiti dalla scheda in differenti modalità programmabili dall'utente tramite le *combo-box* presenti nella parte alta dell'interfaccia di controllo.



Sample rate definisce la frequenza di campionamento del segnale che viene acquisito.

FFT Channels: permette di scegliere il numero di canali, e quindi la risoluzione, che la Fast Fourier Transform calcola per ogni spettro. Le opzioni possibili sono 256, 512, 1024, 2048 o 4096 coerenti con l'algoritmo radix-2 impiegato.

FFT Burst Size: determina la dimensione del *burst*, ossia il numero di spettri da trasferire ad ogni ciclo. Un valore alto di questo fattore produce un miglioramento nelle performance in quanto permette ai tempi fissi necessari per la visualizzazione, la gestione della parte RF ecc.. di essere ammortizzati. Il *burst* è una quantità discreta dipendente dal numero di canali adottato per il calcolo della FFT come verrà illustrato al punto successivo.

2.2.2 Trasferimento in blocchi dalla memoria della scheda A/D alla memoria del PC

L'operazione di trasferimento dalla RAM del convertitore alla RAM del PC avviene tramite bus PCI ed ha quindi un transfer rate di 130Mb/s.
La scheda dispone di una memoria RAM di 4Mb, ciò permette di acquisire un alto numero di spettri contigui per ogni work-cycle che dipende comunque dalla risoluzione impiegata.
La tabella a seguito mostra le possibili configurazioni previste dall'applicazione.

Risol. (kHz)	*Canali*	*Max burst size*	*Min burst size*
78	256	2048	32
39	512	1024	16
20	1024	512	8
10	2048	256	4
5	4096	128	2

2.2.3 Calcolo dello spettro complesso (algoritmo FFT radix-2)

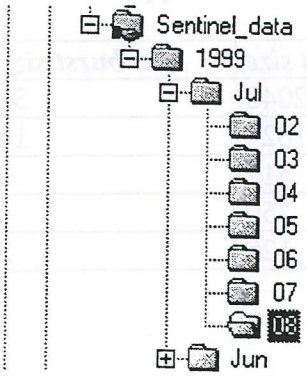
L'insieme dei campioni digitalizzati nel dominio del tempo viene elaborato da un'algoritmo di FFT per ricavare lo spettro di potenza. Dal punto di vista delle prestazioni globali del sistema questo algoritmo costituisce un nodo fondamentale. E' l'operazione con la complessità computazionale più onerosa dell'intero *work-cycle* ed occupa circa il 70% del tempo di esecuzione del ciclo intero. Risulta quindi oltremodo importante adottare un algoritmo ottimizzato per le caratteristiche dell'hardware adottato, in particolare per quanto riguarda l'efficienza nell'elaborazione dell'algoritmo e la precisione nella ricostruzione dello spettro.
L'applicazione implementa al suo interno due differenti algoritmi e può essere compilata per impiegare l'uno o l'altro.
Il primo algoritmo è l'algoritmo Radix-2 della Texas Instruments che lavora solo su interi ed è stato progettato per i DSP (è l'algoritmo utilizzato anche nel sistema Sentinel 1). Ha una precisione molto buona ed una discreta efficienza in tempo anche se eseguito su un Pentium II e non su di un DSP TI.
Il secondo algoritmo porta il nome del suo ideatore, il giapponese "Ooura". Ooura FFT utilizza i floating point, e può arrivare fino a 1M canali, memoria permettendo. E' circa il 40% più veloce della FFT Texas Instrument, ma è leggermente meno preciso.

2.2.4 Calcolo potenze e medie

Ad ogni work-cycle viene calcolato uno spettro di potenza ottenuto sommando i quadrati di parte reale e parte immaginaria dello spettro complesso. Per la visualizzazione si calcola uno spettro risultato della media di ognuno degli spettri di potenza componenti il *burst*.

2.2.5 Scrittura dati su HD

La codifica utilizzata per il salvataggio degli spettri è il codice ASCII.
A partire da una directory principale, il cui nome è impostabile dall'utente nel file di configurazione "Sentinel.ini", gli spettri vengono salvati seguendo una struttura ordinata di directory catalogate per anno, mese e giorno correnti.
Si veda l'esempio in figura:

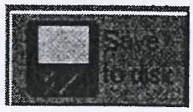


All'interno dell'ultima directory, quella relativa al giorno, sono presenti le informazioni relative agli spettri suddivise a loro volta in files identificati univocamente dal nome che cambia a seconda della direzione dell'antenna e della frequenza.

La codifica ASCII facilita la lettura e la decodifica dei dati agevolando la condivisione delle informazioni con i membri della comunità scientifica ed uniformandosi in tal modo alle specifiche di progetto dettate dal CRAF (Committe on RadioAstronomical Frequencies), l'organo di protezione delle bande radioastronomiche. Occorre comunque dire che metodo rende più lenta la

fase di recupero delle informazioni. In Sentinel 1 veniva utilizzato un database formato Access che, nonostante fosse più veloce nel recupero dei dati, aveva il grosso limite che un file non poteva superare la dimensione di 1 Gb; questo metodo, inoltre, richiedeva la necessità di possedere la licenza del S\W Microsoft Access.

Esistono due differenti modalità di salvataggio degli spettri. Nella prima ogni singolo spettro appartenente al *burst* dati viene memorizzato su un disco fisso ultra-wide SCSI 2 ad accesso veloce. Questa modalità viene avviata manualmente premendo il pulsante:



Il salvataggio viene arrestato premendo nuovamente lo stesso bottone che, nel frattempo, avrà cambiato dicitura.

Operando in questa maniera verrà generata una grande quantità di dati, Questo permetterà tuttavia di poter scovare anche le interferenze più rapide; l'eventuale integrazione degli spettri verrà compiuta *a posteriori* dallo spettrometro *off-line* nella maniera che l'utente riterrà più opportuna.

La seconda modalità prevede l'attivazione automatica del salvataggio dati. Lo scopo di questo metodo consiste nel permettere delle sessioni automatiche di monitoraggio a scopo di controllo senza generare un'eccessiva mole di dati. In questo senso è stata prevista anche la possibilità di salvare spettri già integrati *on-line* a differenza del primo caso. Ad esempio sarà possibile scegliere di salvare giornalmente uno spettro di 1024 canali, risultato della media di 512 spettri singoli e contigui, ogni 10 minuti dalle ore 20:00 alle ore 24:00. La schedulazione del salvataggio dati si programma tramite un'apposita interfaccia che agisce sulla configurazione dell'applicazione *"on-line processing & control"* e che permette di specificare una sessione di monitoraggio giornaliera con la granularità minima del minuto. La schedulazione si imposta specificando uno o più periodi (dalla ora alla ora) e l'intervallo di tempo (in minuti) con il quale salvare i dati nei periodi precisati. E' anche possibile impostare il numero di canali, la frequenza di campionamento, il numero di spettri del *burst* ed il numero di spettri da mediare all'interno del *burst*.

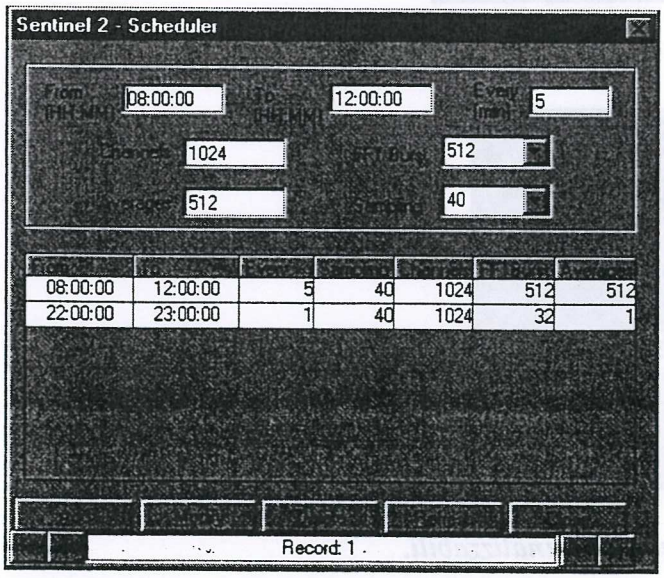


Fig. 2.3 ***Interfaccia per la schedulazione del salvataggio degli spettri***

2.2.6 Pilotaggio frequenza dell'analizzatore di spettro via IEEE 488 e selezione dell'antenna

Nella zona inferiore della finestra principale dell'applicazione si trova una zona rettangolare composta da tante piccole barrette gialle e da barretta una rossa (Fig. 2.4) che chiameremo '*Sweeping Bar*' e che visualizza la sintonia all'interno della banda da analizzare.

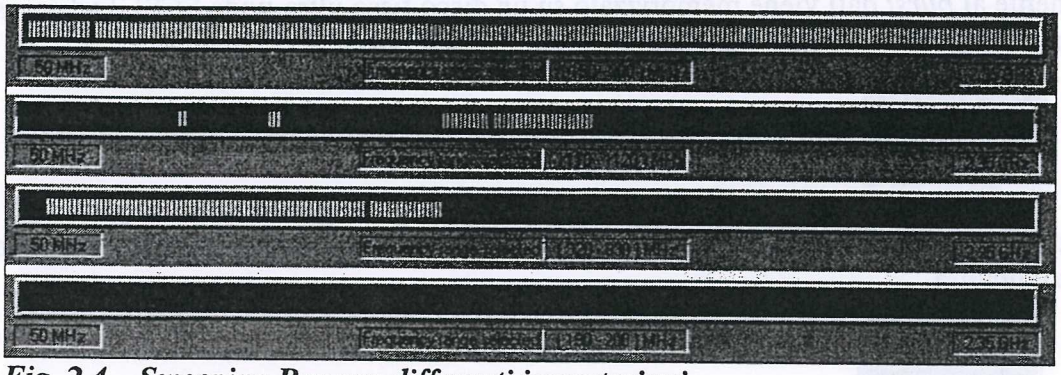
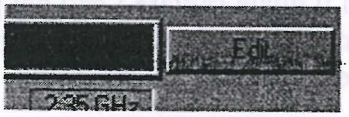


Fig. 2.4 – Sweeping Bar con differenti impostazioni.

All'interno della *Sweeping Bar* si visualizzano con piccole barre gialle le bande di frequenza che si è scelto di analizzare e si indica con il colore rosso la banda attualmente acquisita dal sistema. Ogni barretta individua una finestra di massimo 20MHz come imposto dalla sampling rate del convertitore. Agli estremi sono indicati i valori limite mentre al centro viene mostrata la corrente finestra analizzata.
Questa maniera di presentare le informazioni è stata progettata per avere facilmente il controllo della funzionamento del sistema.
Attualmente l'applicazione è predisposta per l'analisi delle frequenze radio comprese tra 50MHz e 2.35GHz in virtù delle specifiche dettate dalla parte di radiofrequenza. E' stata prevista la possibilità di personalizzare le bande da analizzare permettendo all'utente di specificare fino a tre differenti programmi di scansione, è inoltre prevista la possibilità di soffermarsi su di una finestra fissa di frequenza.
Il programma di scansione si seleziona dalla finestra mostrata in Fig. 2.5 che appare alla pressione del tasto "Edit"  sito proprio di fianco alla *Sweeping Bar*.

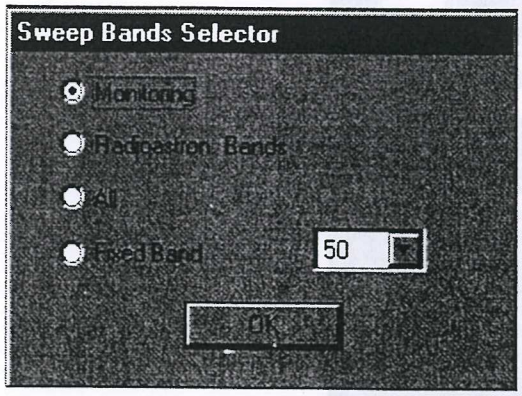


Fig. 2.5 – Controllo delle bande analizzabili.

L'impostazione dei tre programmi di scansione personalizzabili avviene invece nel file "Sentinel.ini" e va quindi compiuta prima che l'applicazione venga avviata perché le modifiche abbiano effetto. Per ogni programma si possono specificare fino a cinque differenti aree di scansione. Segue un esempio relativo alle impostazioni mostrate in Fig. 2.4.

```
[Sweep Bands A]
Label=Sequence Complete
Begin1=50
End1=2350
Begin2=0
End2=0
Begin3=0
End3=0
Begin4=0
End4=0
Begin5=0
End5=0

[Sweep Bands B]
Label=Radioastron. Bands
Begin1=400
End1=420
Begin2=600
End2=630
Begin3=1000
End3=1350
Begin4=0
End4=0
Begin5=0
End5=0

[Sweep Bands C]
Label=Hessdalen Bands
Begin1=100
End1=1000
Begin2=0
End2=0
Begin3=0
End3=0
Begin4=0
End4=0
Begin5=0
End5=0
```

Oltre a selezionare la frequenza di osservazione, il sistema ha la possibilità di gestire la direzione di ricezione del segnale radio. Questa funzione viene eseguita utilizzando la porta parallela (LPT1) del PC I. Il dato trasmesso verrà poi impiegato dall' *Antennas Control* per selezionare una delle 8 antenne previste nel sistema ricevente.
Il controllo del sistema di antenne contiene relè allo stato solido a commutazione pressochè istantanea: pertanto dopo l'invio del codice non è necessario introdurre ulteriori stati di *wait*.
Dal punto di vista dell'utente la selezione delle direzioni desiderate si compie in maniera molto semplice; occorre semplicemente attivare o disattivare i pulsanti relativi alle direzioni volute (si osservi la Fig. 2.6).

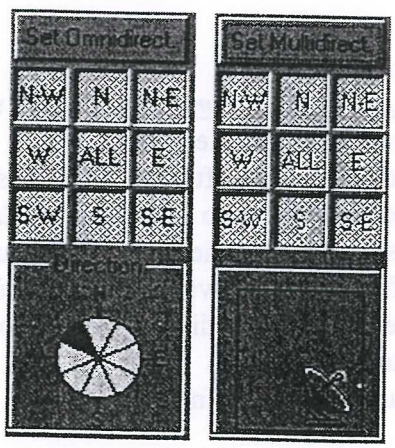
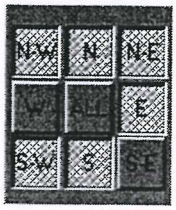


Fig.2.6 – Controllo direzioni.

Al momento si sono previste otto direzioni, ciascuna associata ad un'antenna che deve essere posizionata coerentemente; esse sono rappresentate da un cerchio giallo diviso in settori; il settore in blu identifica la direzione attualmente analizzata. Il programma, dopo aver sintonizzato l'oscillatore locale, commuterà in sequenza le antenne attivando solo quelle abilitate dai relativi pulsanti. Per esempio, la configurazione del tipo mostrato a lato, seleziona le antenne nella sequenza: N-NE-E-S-SW-NW.



Come si nota nella Fig. 2.6, un ulteriore pulsante centrale *Set Omnidirect. / Set Multidirect.* permette, nel caso ve ne fosse necessità, di predisporre il sistema all'impiego di un'antenna singola (ad esempio una qualsiasi antenna omnidirezionale) in alternativa al gruppo di antenne direttive commutabili.
Nella stazione di Radioastronomia di Medicina, si utilizza questa opzione per analizzare il segnale captato dalla Croce del Nord o dall'antenna parabolica VLBI (dopo una necessaria conversione in banda base) e testare così il funzionamento del sistema Sentinel 2.

2.2.7 Algoritmi di visualizzazione e gestione dell'interfaccia grafica

L'applicazione "*On-line Processing and control*" mette a disposizione dell'utente altre funzionalità aggiuntive non strettamente inerenti all'esecuzione del *work-cycle*.

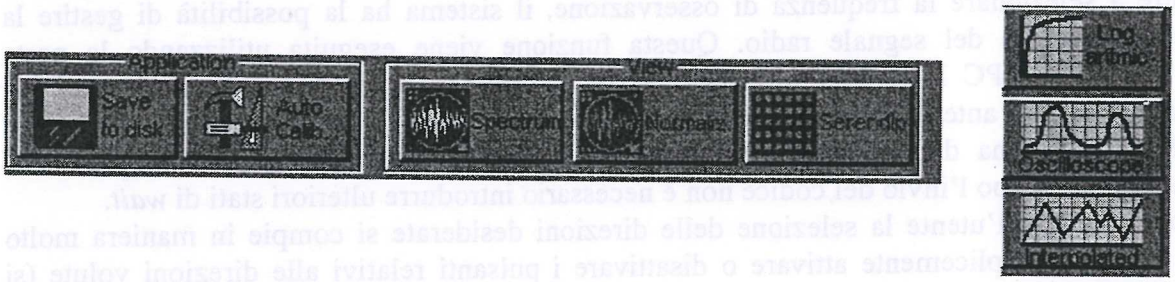


Fig. 2.7 – Comandi principali.

Un'utile funzione disponibile tra i comandi principali è l'*Auto Calibrazione* che effettua una taratura del sistema permettendo di filtrare il rumore di fondo proveniente dall'elettronica.
L'introduzione di questo comando è avvenuta durante la fase di test del software quando si è constatato che, tra i diversi segnali che venivano rilevati, alcuni di questi erano autogenerati dalla scheda A/D. Anche se i livelli di tali segnali sono bassi risultano sufficienti ad introdurre informazioni spurie negli spettri di potenza. Poichè tali interferenze si trovano per lo più a frequenze fisse, si è allora pensato di campionarle in assenza del segnale utile e di tenerne poi conto nelle misure successive, compensando quei canali dove comparivano.
Dopo aver avviato l'autocalibrazione l'applicazione mostra la seguente popup window:

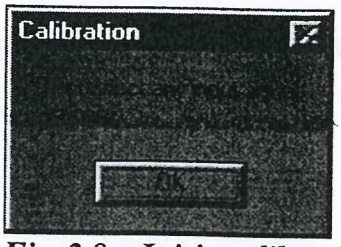


Fig. 2.8 – Inizio calibrazione.

Dopo aver disconnesso il segnale in ingresso, il sistema comincia ad acquisire una serie di spettri che conterranno unicamente i contributi "spuri" derivanti dall'elettronica della scheda. Sulla suddetta serie verrà calcolata la media in modo da annullare il rumore di fondo ed evidenziare i segnali autogenerati. L'indicazione di fine calibrazione è la seguente:

Fig. 2.9 – Messaggio di fine calibrazione.

Nelle successive misure l'offset così calcolato verrà sottratto agli spettri di potenza. La calibrazione va effettuata dopo avere avviato l'applicazione e ogni qual volta si commuta il numero di canali.

Nella zona inferiore della finestra principale è presente una cornice contenente informazioni relative alle prestazioni del sistema per quanto riguarda i tempi (espressi in millisecondi) necessari all'esecuzione dell'intero burst dati acquisito e di un singolo *work-cycle*.

Fig. 2.11 – Informazioni generali.

I comandi presenti alla destra del pannello centrale sono inerenti alle modalità di visualizzazione degli spettri.

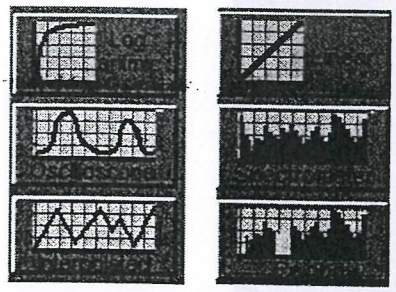


Fig. 2.12 – Modalità di visualizzazione

Logaritmic / Linear seleziona il tipo di scala con cui si desidera visualizzare gli spettri. In Fig. 2.13 si riporta un esempio di uno spettro rappresentato nelle due scale.

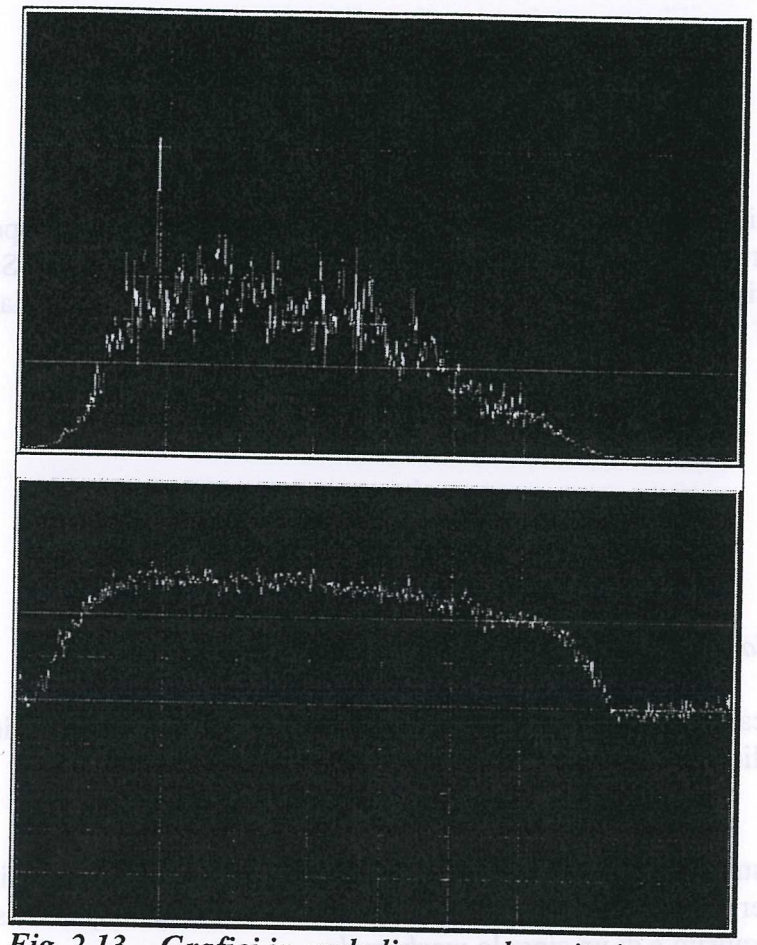
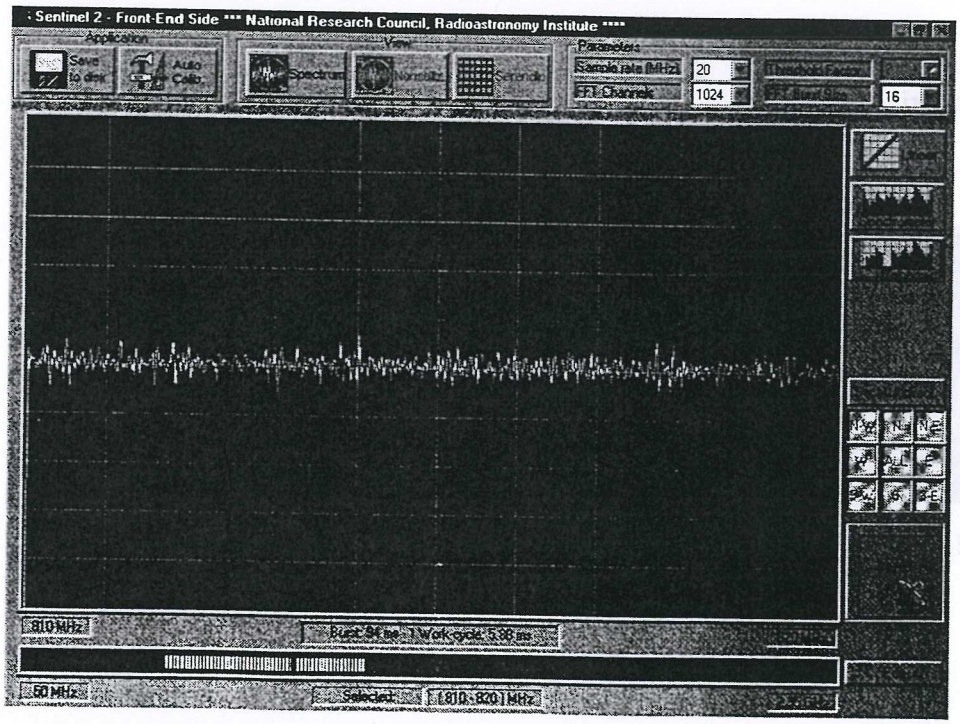


Fig. 2.13 – Grafici in scala lineare e logaritmica.

Oscilloscope / Spectrometer seleziona la modalità di funzionamento del Sentinel 2 che può agire come spettrometro o anche più semplicemente come oscilloscopio permettendo di esaminare il segnale campionato nel tempo, senza effettuare l'analisi di Fourier.



Interpolated / Rainbow seleziona come determinare la modalità di tracciamento dello spettro. Nel primo caso il valore assunto da un canale è interpolato linearmente con il successivo, mentre nel secondo caso il valore assunto da ogni canale è rappresentato ha una barra che ne indica l'ampiezza; il colore della barra è associato alla frequenza all'interno della finestra analizzata e va dal rosso al violetto di concerto con la sequenza dello spettro della radiazione elettromagnetica visibile.

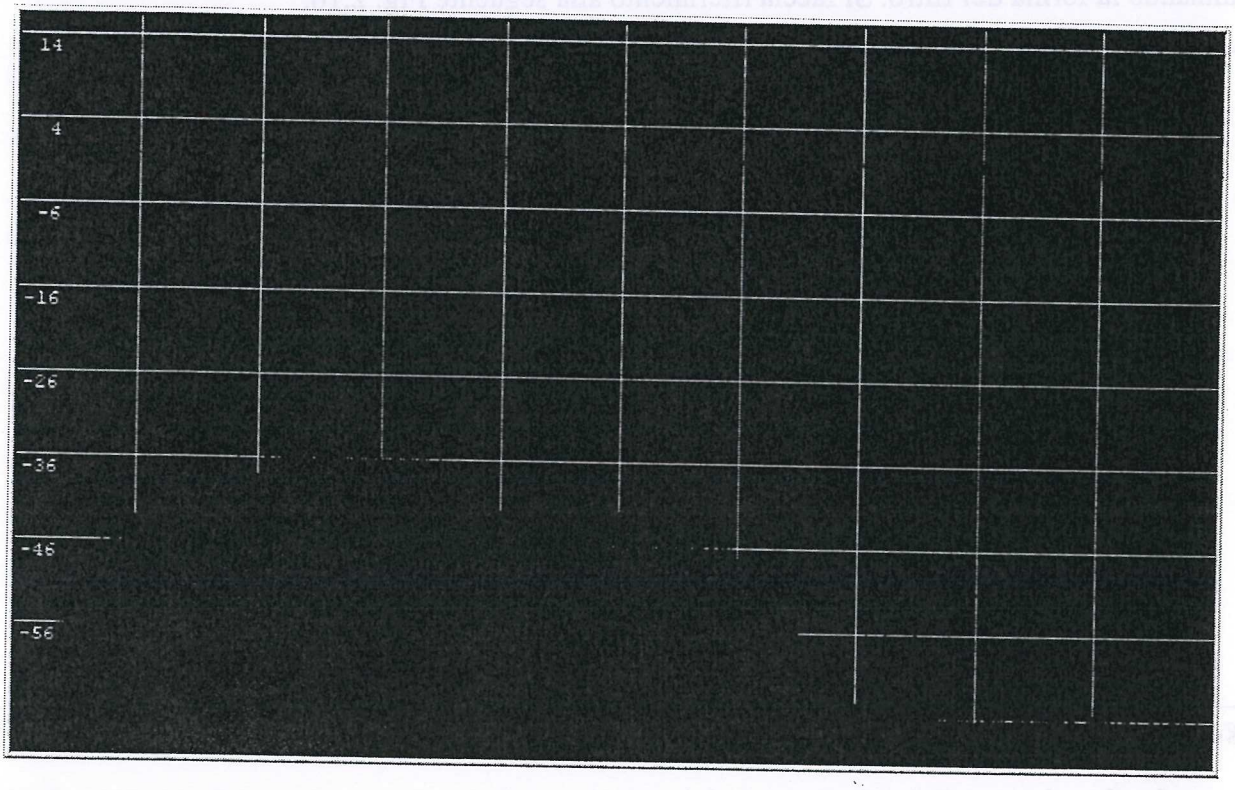


Fig. 2.14 – Rainbow Mode.

Il pannello View è relativo anch'esso alla visualizzazione degli spettri e permette di selezionare tra differenti algoritmi di *post-processing*. In particolare, sono previste tre possibilità: Spectrum, Normalized e Serendip. Si tiene a ribadire che questi algoritmi di elaborazione del segnale vengono applicati sono nella fase relativa alla visualizzazione al fine di agevolare l'utente nell'identificazione in prima istanza (per l'esame approfondito è stata appositamente studiata la seconda applicazione di cui al par 2.3) di un possibile segnale 'interessante' ai fini della ricerca di interferenze. Gli spettri continueranno comunque ad essere memorizzati così come sono senza l'applicazione di queste particolari tecniche di elaborazione.

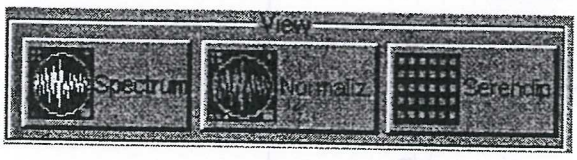


Fig. 2.15 – View Mode.

Spectrum

Visualizza la media degli spettri contenuti nel *burst* acquisito. Questa classica rappresentazione è comune a tutti gli analizzatori di spettro che si trovano in commercio. Lungo

l'asse delle ordinate si indica l'intensità di potenza mentre sulle ascisse vengono rappresentate le frequenze.

Normalized

Questa modalità di visualizzazione consente di evidenziare la parte utile del segnale analizzato eliminando la forma del filtro. Si faccia riferimento alla seguente Fig. 2.16.

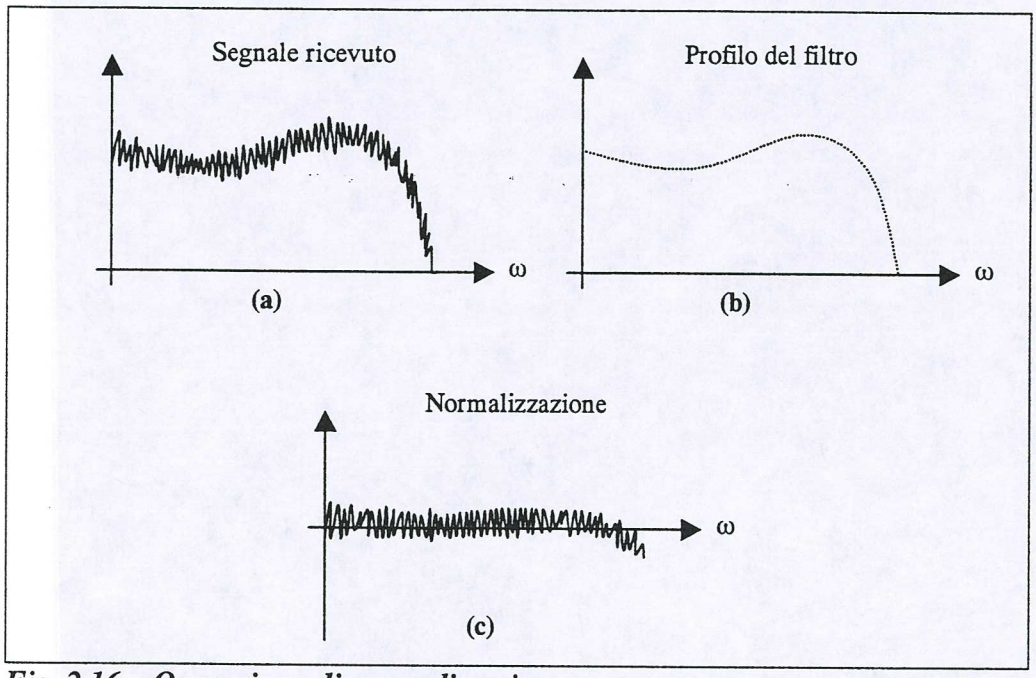


Fig. 2.16 – Operazione di normalizzazione.

Il segnale che giunge all'analizzatore è tipicamente costituito da una certa informazione (RMS) sovrapposta ad un contributo fisso (offset). La componente che interessa è contenuta nell'RMS (a) mentre la parte costante è costituita dal profilo (*shape*) del sistema di ricezione (b) sito a monte dell'analizzatore di spettro.

Poiché questo profilo si mantiene costante essendo determinato dalle caratteristiche del sistema ricevente, è inopportuno utilizzare dei bit per descriverne l'andamento andando in questo modo a perdere precisione nella descrizione del segnale utile ai nostri scopi. A tal fine, la normalizzazione calcola il profilo del filtro suddetto effettuando una ***Boxcar Integration*** di 16-esimo ordine e lo sottrae, a *run-time*, ad ogni spettro mediato prima di visualizzarlo (c).

L'operazione che si esegue è mostrata in Fig. 2..

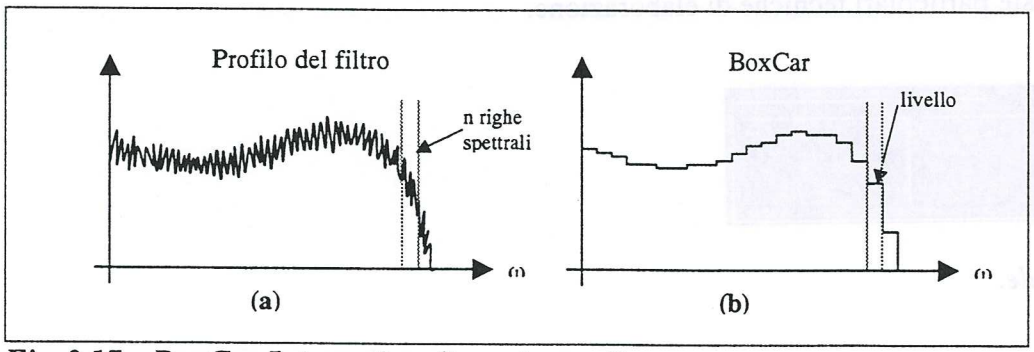


Fig. 2.17 – BoxCar Integration di n-esimo ordine.

Per ogni n canali dello spettro si calcola la media aritmetica che viene assunta come valore unico per tutti i canali della serie considerata. Ciò costituisce un livello della BoxCar, il numero di livelli calcolati sarà quindi pari a: ceil(*n°canali **div** n)*.
Al termine del procedimento lo spettro si presenta come in Fig. 2.16(c). Dopo la sottrazione del profilo del filtro tutti i bit di cui si dispone sono impiegati per descrivere le informazioni associate all'RMS.

Serendip

Serendip è l'acronimo di *Search for Extraterrestrial Radio Emissions from Nearby Developed Intelligent Populations*.
Questo particolare metodo di post-processing è proprio dello spettrometro realizzato dall' Università di Berkeley nell'ambito del progetto SETI; si è deciso di implementarlo in quanto risulta particolarmente utile nell'individuare interferenze *narrow-band*, ossia a banda stretta.
L'algoritmo Serendip permette di evidenziare tutti e solo quei canali che sono caratterizzati da una sufficiente intensità di potenza. Per fare questo viene calcolata una soglia e si valuta quali canali la superano. La soglia da considerare è un multiplo intero del profilo ricavato elaborando lo spettro con la tecnica della *moving average,* ossia mediando i valori di intensità dei canali su una finestra mobile anziché su di una finestra fissa come avviene nel metodo *boxcar*.

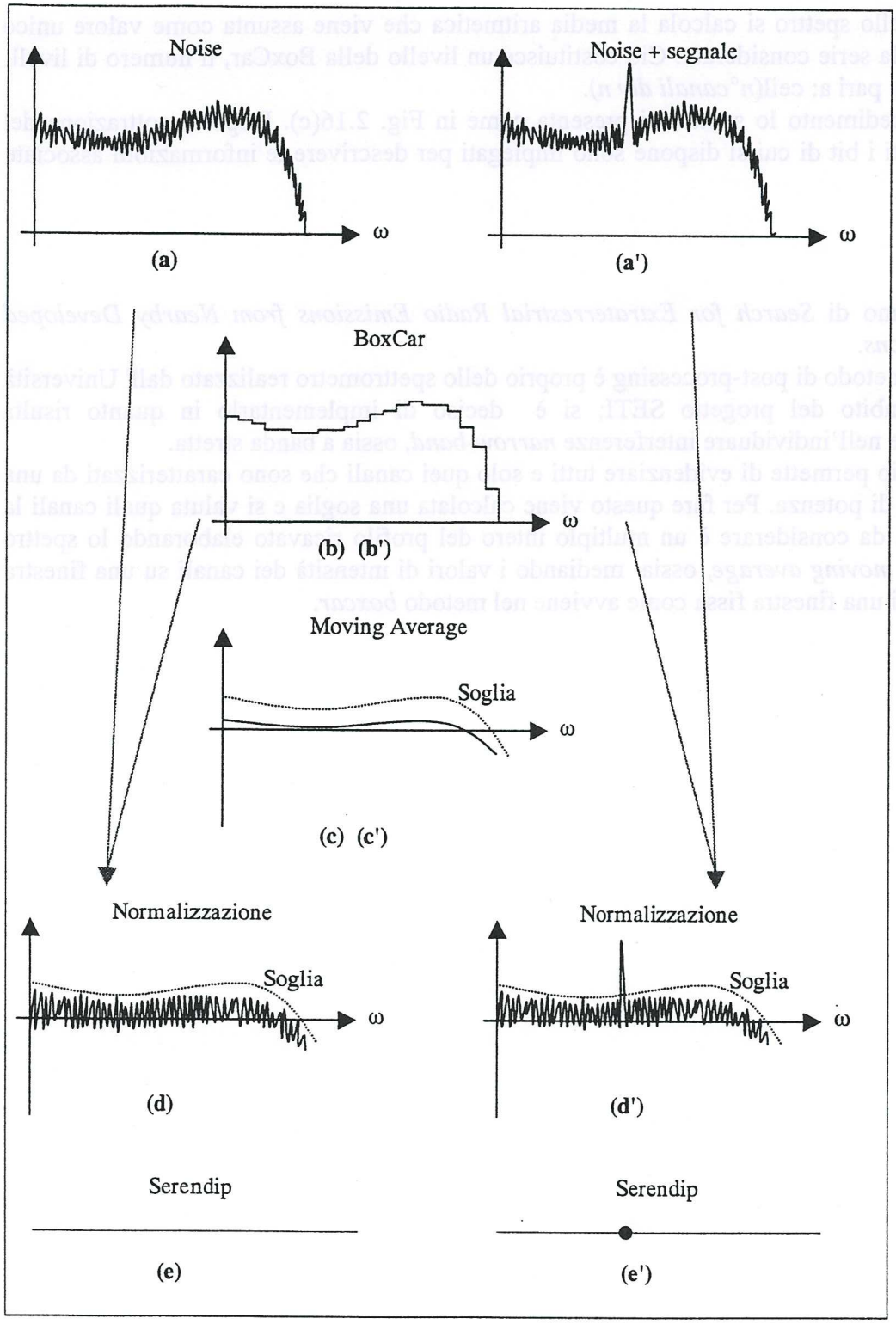
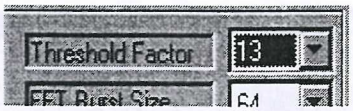


Fig.2.19 – Passaggi operativi della modalità Serendip.

La soglia calcolata nel modo illustrato alla Fig. 2.19 può essere lasciata invariata oppure può essere moltiplicata per un fattore (*Threshold Factor*) in maniera tale da filtrare maggiormente lo spettro elaborato.
La combo box *Threshold Factor* presente nel frame *parameters* permette appunto di cambiare il livello di soglia (infatti è attiva solamente nella modalità Serendip). Il suddetto fattore moltiplicativo è un numero intero compreso tra 1 e 20. Tipicamente una visualizzazione sufficientemente filtrata dal *noise* di fondo si ottiene con valori attorno a 13, quindi con una soglia sufficientemente elevata.



La modalità di visualizzazione è visibile in Fig. 2.20

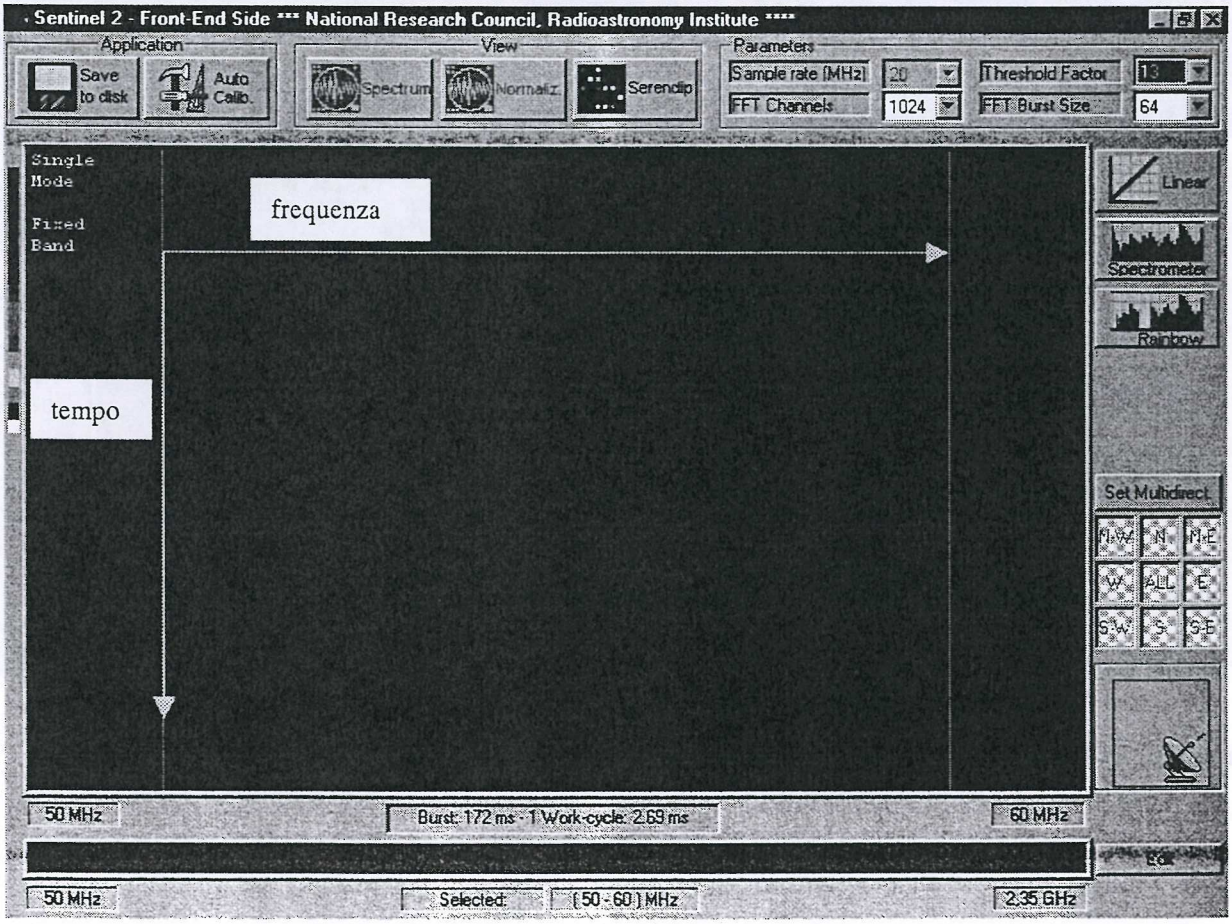


Fig.2.20 – Modalità Serendip con antenna omnidirezionale.

Lo schermo evidenzia un punto acceso nei canali che superano la soglia. Il diagramma è composto da righe parallele, ognuna delle quali rappresenta uno spettro. L'asse y indica quindi il tempo, mentre l'asse x continua a rappresentare il dominio delle frequenze come nei casi precedenti. Il colore di ciascun punto del canale è un'indicazione di quante volte l'intensità del canale ha superato la soglia. La didascalia presente nel lato sinistro del grafico illustra la scala dei colori adottati.

L'implementazione realizzata supporta anche la scansione delle direzioni. Un esempio viene presentato in Fig. 2.21

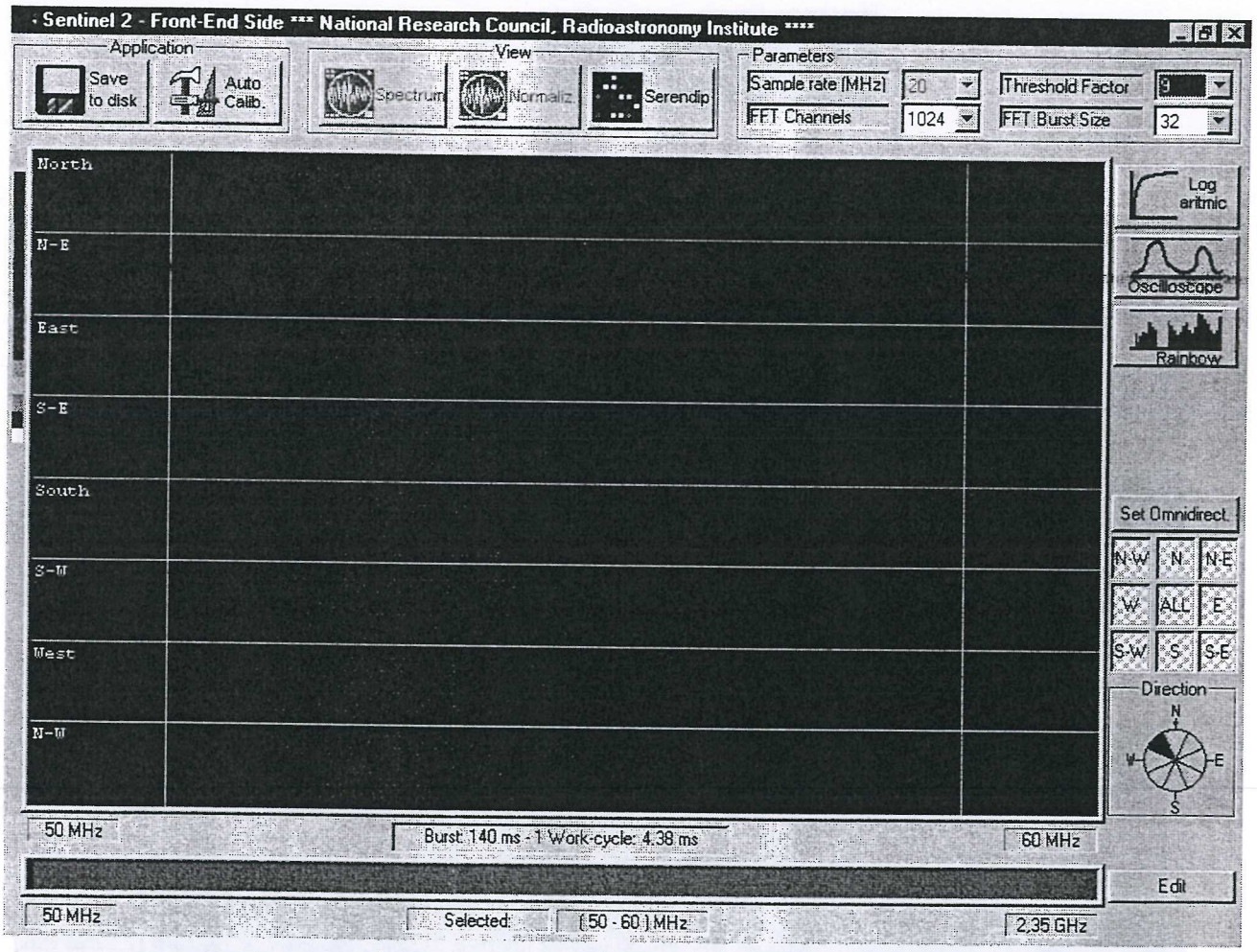


Fig.2.21 – Modalità Serendip con gestione delle otto antenne.

2.3 L'applicazione "*off-line spectrometer*"

Come già ribadito la presente applicazione implementa uno spettrometro funzionante *off-line* sui dati dell'archivio spettri generato dalla prima applicazione che deve sempre essere attiva durante il funzionamento del sistema.. Gli spettri vengono salvati in formato ASCII dalla quale le presente applicazione va a leggere per la visualizzazione. Gli spettri possono essere recuperati in differenti modalità concepite soprattutto per agevolare il cacciatore di interferenze delle sue ricerche.

All'avvio del programma, l'operatore si troverà di fronte ad una videata come quella qui riportata

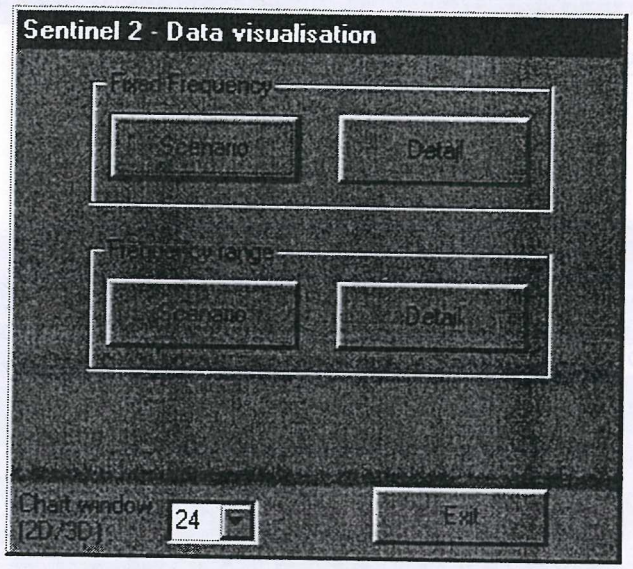


Fig 2.22 – La schermata principale dell'applicazione di visualizzazione degli spettri.

L'applicazione permette di accedere a quattro differenti modalità, divise per '*Fixed Frequency*' o '*Frequency Range*' e per '*Scenario*' o '*Detail*'. La modalità '*Fixed frequency*' visualizza gli spettri acquisiti prendendo esclusivamente la banda acquisita al momento dell'elaborazione; con l'opzione '*Frequency Range*' è invece possibile concatenare più bande per poter visualizzare un 'panorama' dell'intera banda di frequenza da monitorare. A questo punto i dati possono essere recuperati per giorno, mese, ora, direzione e frequenza.

Una volta definiti gli estremi per la ricerca, le modalità di visualizzazione offerte sono 3:

Grafico Tempo-Frequenza-Ampiezza (x,y,z) in 3-D
Grafico Tempo-Frequenza-Ampiezza in 2-D dove l'ampiezza è rappresentata dal colore
Grafico Frequenza-Ampiezza relativo ad uno spettro per volta.

Un'altra utile funzionalità messa a disposizione dal programma permette di integrare gli spettri selezionati tra di loro, in modo da poter visualizzare una serie di n spettri sommati m alla volta (m<n).

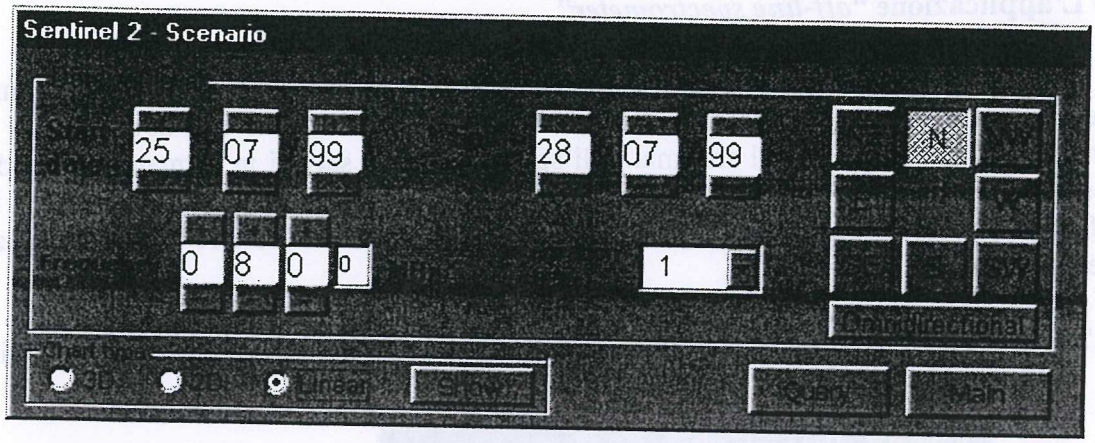


Fig. 2.23 – Schermata Fixed Frequency Scenario

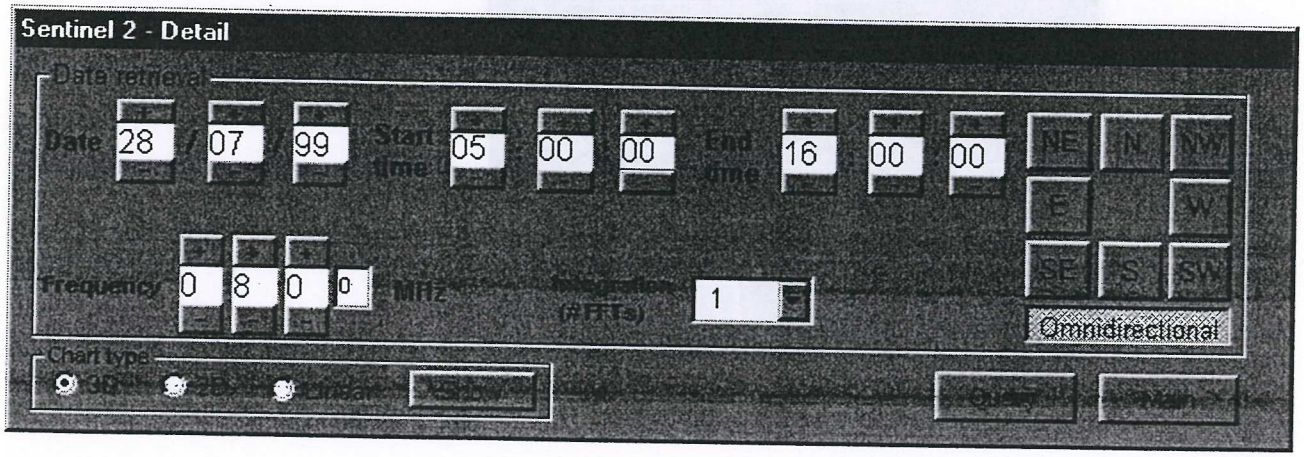


Fig. 2.24 – Schermata Fixed Frequency Detail

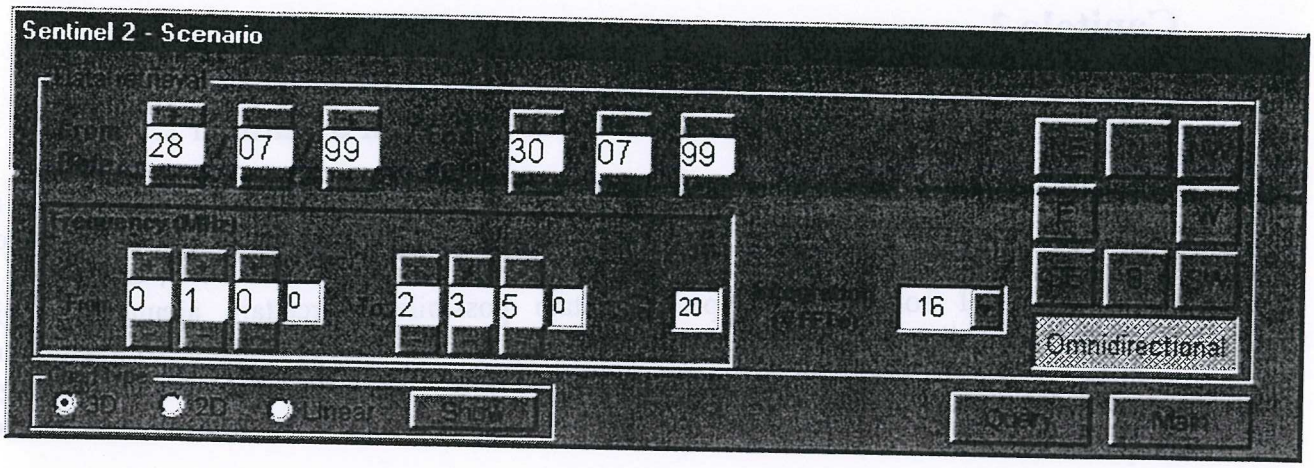


Fig. 2.25 – Schermata Frequency Range – Scenario

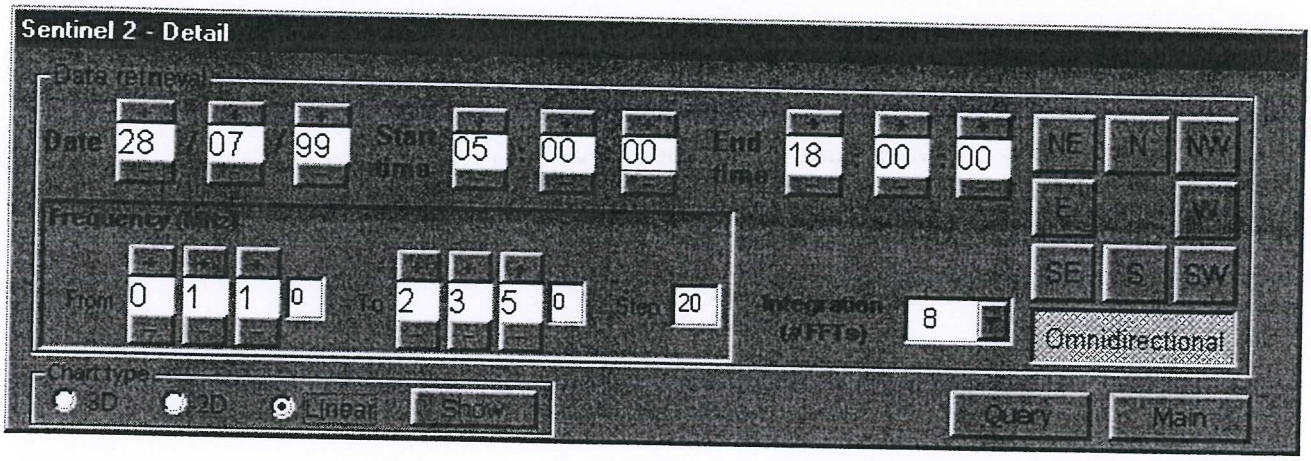


Fig. 2.26 – Schermata Frequency Range - Detail

Capitolo 3

Risultati di *'RFI monitoring sessions'* effettuate con il Sentinel 2.

Il sistema Sentinel 2 è stato impiegato, utilizzando la strumentazione di test descritta al capitolo 1, per effettuare delle sessioni di monitoraggio all'interno del programma europeo RTD. Durante queste prove, che sono inoltre servite come banco di test, debugging e affinamento del software sviluppato, è stato possibile intercettare alcune tipologie interferenze, quali le spread-spectrum di origine militare e l'iridium per le comunicazioni cellulari satellitari, altrimenti molto difficili da rilevare con strumenti 'canonici' quali gli analizzatori di spettro commerciali a causa delle elevate frequenze con cui variano nel tempo.
Le illustrazioni che seguono rappresentano delle 'snapshot' generate con l'applicazione *'Off-line spectrometer'* nelle quali sono visualizzati i risultati di alcune delle sessioni di monitoraggio.

Trasmissione militare "Spread-spectrum" attorno a 1828 MHz.
1 secondo di acquisizione, 1 spettro (di 2048 punti reali) ogni 10 ms

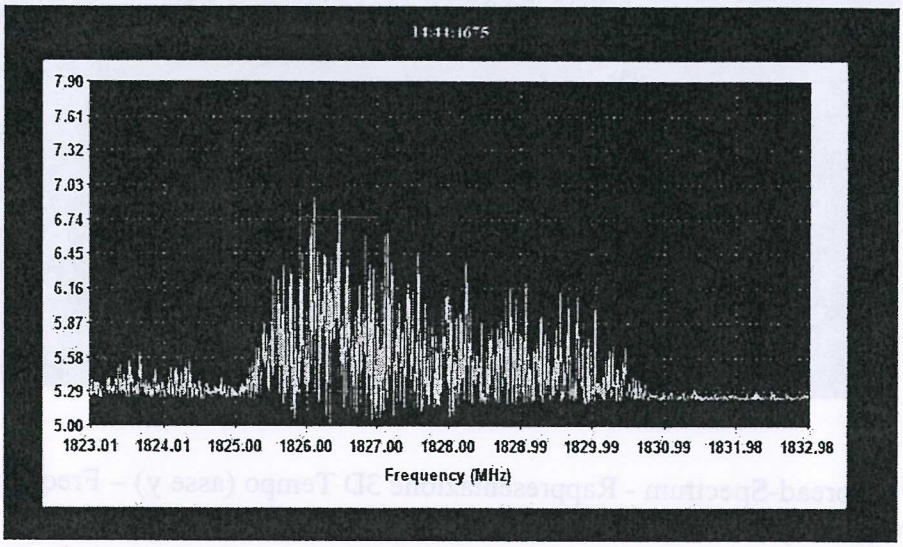


Fig. 2.27 – Spread-Spectrum - Rappresentazione Frequenza-Ampiezza. La linea gialla rappresenta uno spettro appartenente alla serie, quella rossa lo spettro 'peak-hold' ottenuto prendendo il valore massimo su ogni canale.

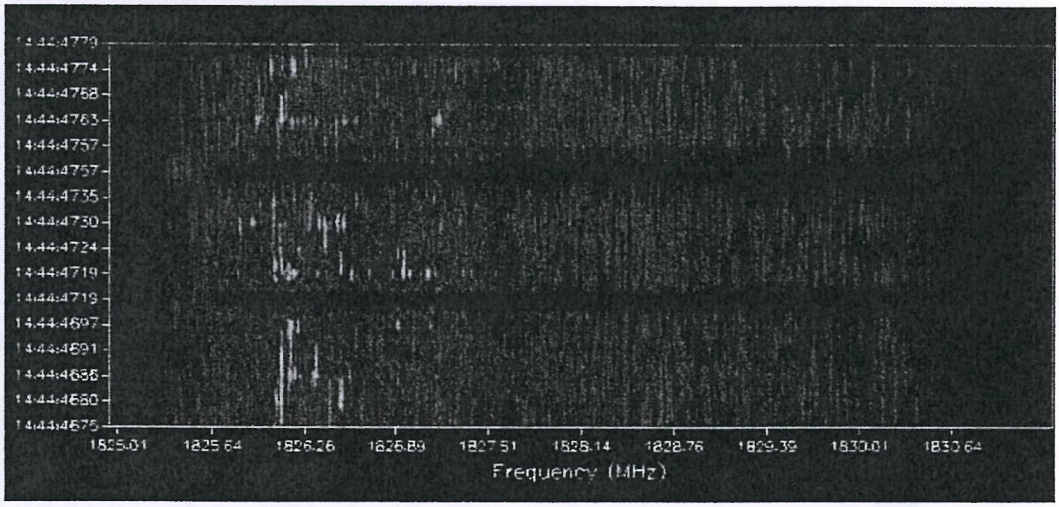


Fig. 2.28 – Spread-Spectrum - Rappresentazione 2D Tempo (asse y) – Frequenza (asse x) – Ampiezza (colore).

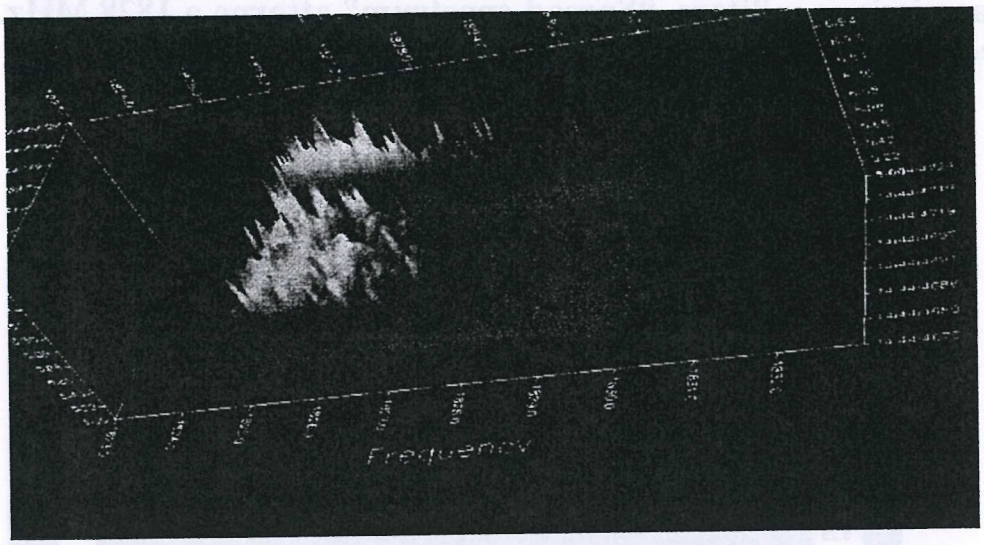


Fig. 2.29 – Spread-Spectrum - Rappresentazione 3D Tempo (asse y) – Frequenza (asse x) – Ampiezza (asse z).

Trasmissione satellitare "Iridium" per comunicazioni cellulari attorno 1825MHz.

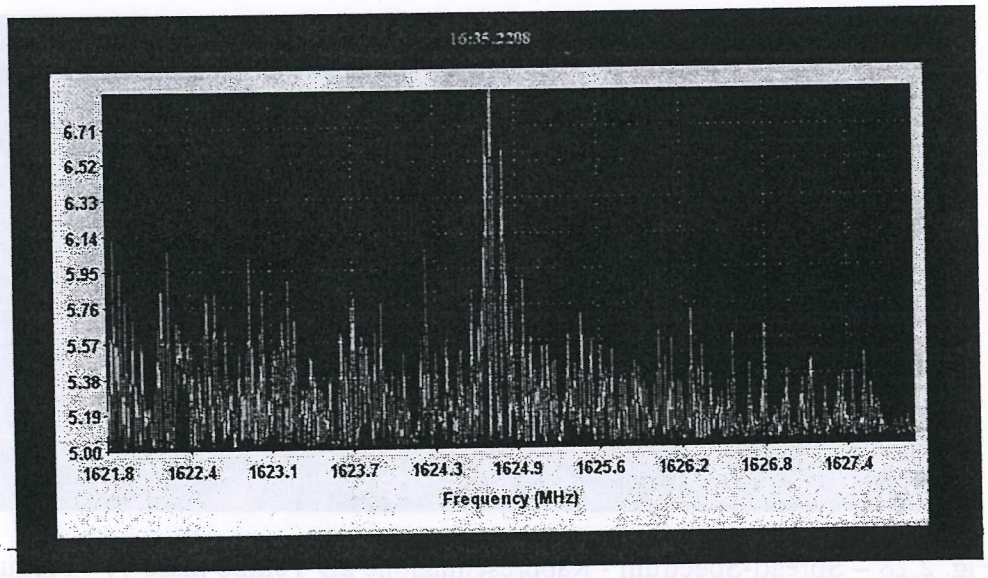


Fig. 2.30 – Iridium - Rappresentazione Frequenza-Ampiezza.

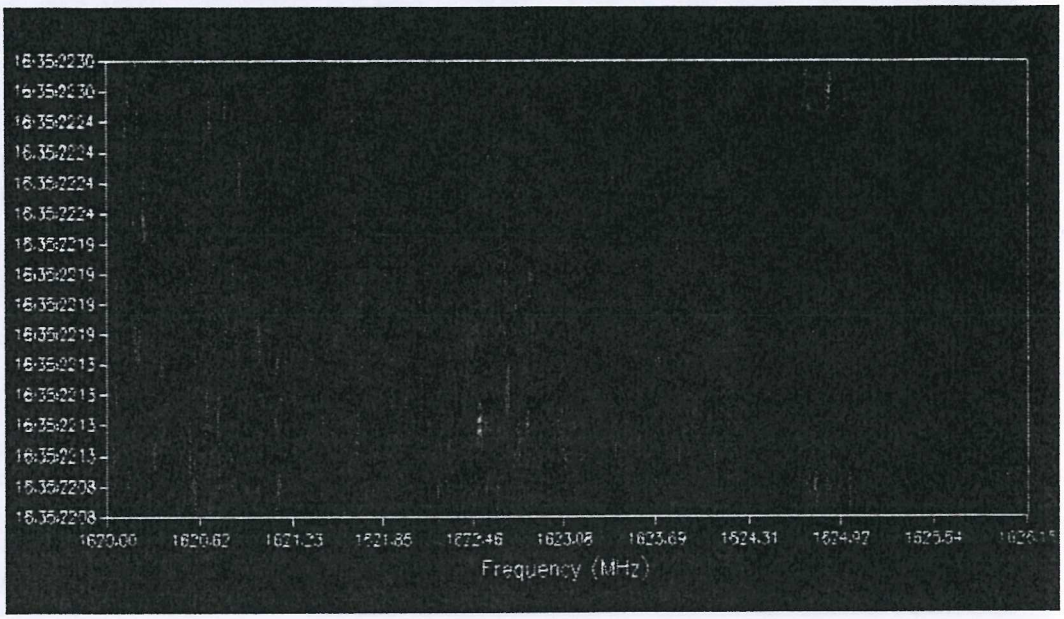


Fig. 2.31 – Spread-Spectrum - Rappresentazione 2D Tempo (asse y) – Frequenza (asse x) – Ampiezza (colore).

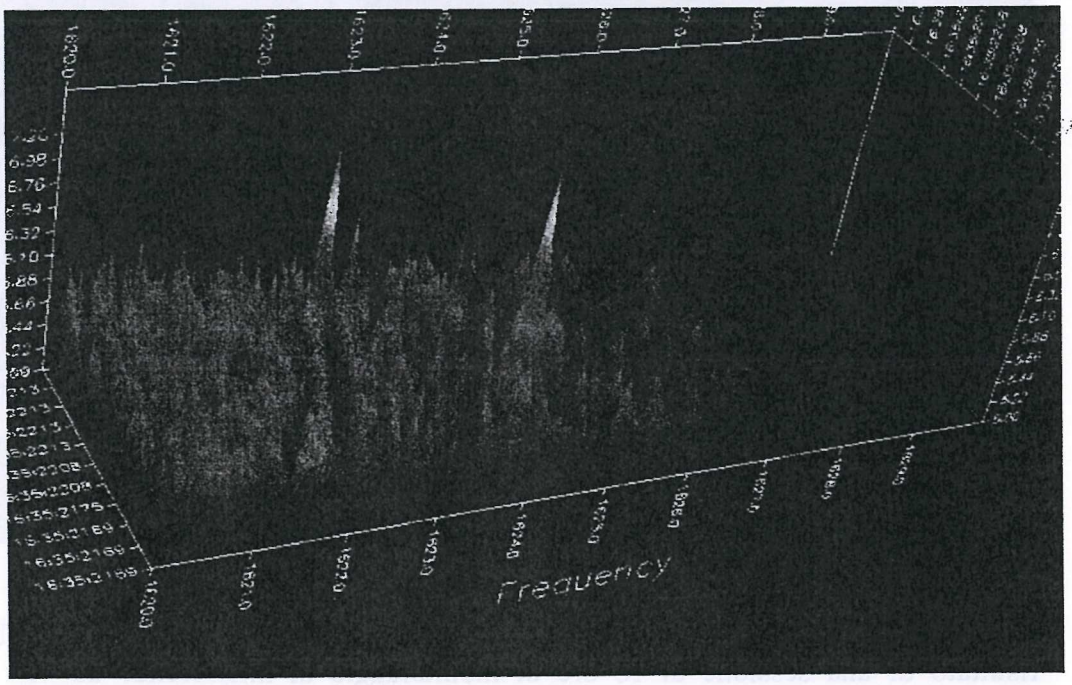
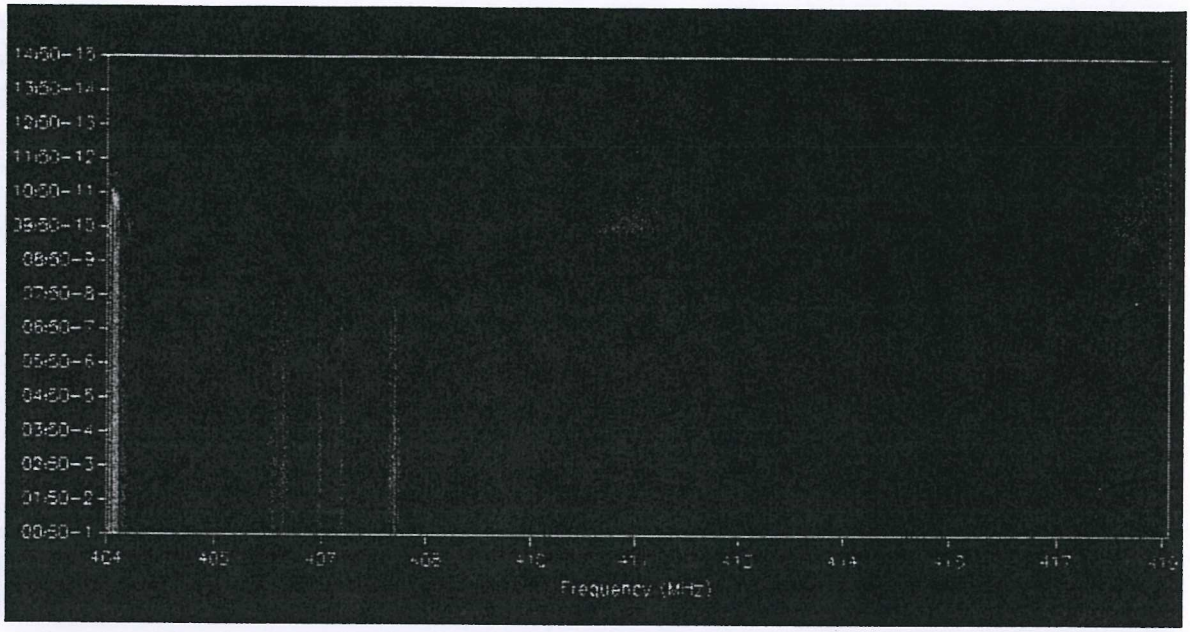


Fig. 2.32 – Spread-Spectrum - Rappresentazione 3D Tempo (asse y) – Frequenza (asse x) – Ampiezza (asse z).



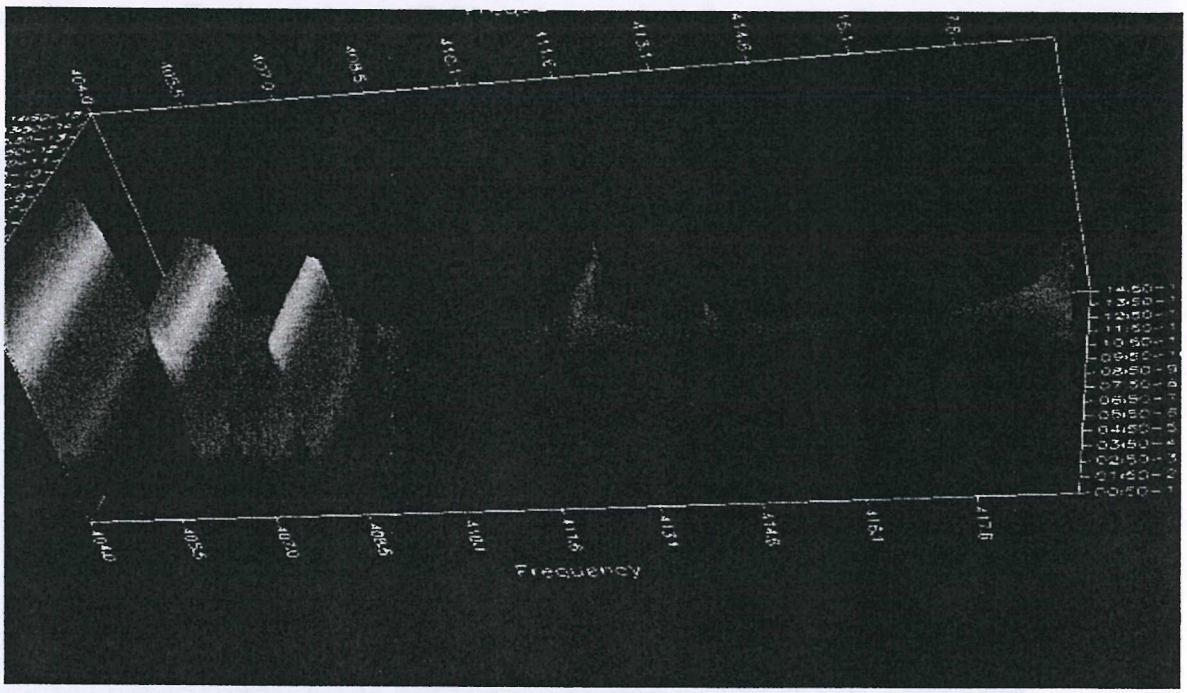


Fig. 2.33 – 2.34 – RFI monitoring session - In queste due immagini è possibile vedere il risultato di una sessione di 16 ore di monitoraggio di interferenze da 404 a 419 MHz compiuta nell'Agosto 1999. Gli spettri acquisiti (uno ogni 10 minuti) sono stati integrati *off-line* in modo da ottenere in visualizzazione un singolo spettro per ora.

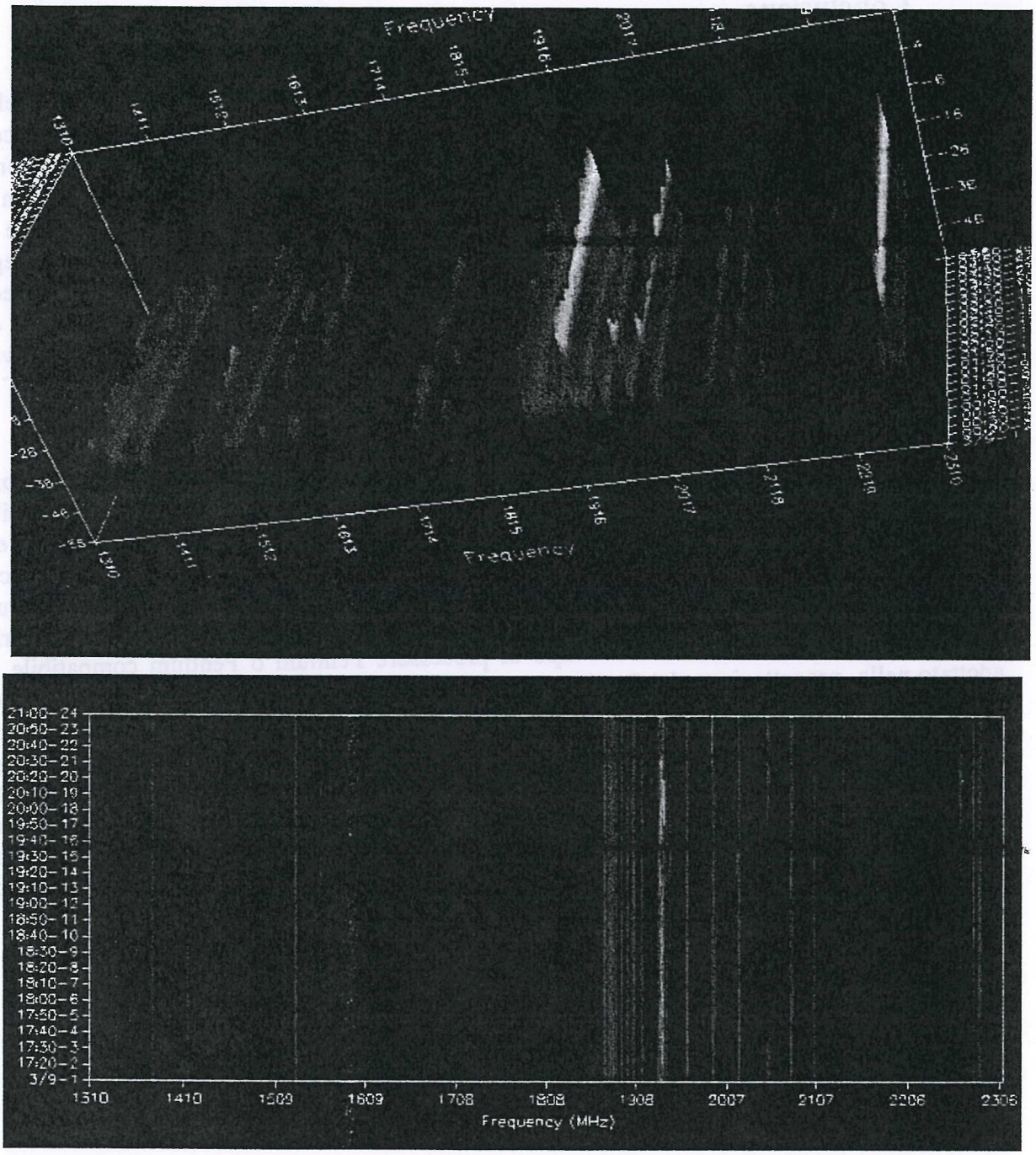


Fig. 2.35 – 2.36 – Agosto 1999- RFI monitoring session - In queste due immagini è possibile vedere il risultato di una sessione di 16 ore di monitoraggio di interferenze da 1310 a 2310 MHz ottenuta concatenando una ad una finestre di 20 MHz di banda. Gli spettri acquisiti (uno ogni 10 minuti) sono stati integrati *off-line* in modo da ottenere in visualizzazione un singolo spettro per ora.

Conclusione.

"Sentinel 2" è un sistema versatile a basso costo per il monitoring di interferenze nella banda 300MHz – 2GHz che evita di impiegare un costoso sistema DSP special-purpose per il calcolo degli spettri di potenza utilizzando una veloce CPU general-purpose. Le prestazioni ottenute sono molto più elevate di quelle raggiunte dal sistema "Sentinel 1". Non essendo Windows NT un sistema operativo real-time, la gestione dell'acquisizione e dell'elaborazione del calcolo delle FFT, il cosiddetto *work-cycle*, non possono essere garantite ad intervalli di tempo esattamente regolari in quanto dipendono dal carico di lavoro del sistema operativo al momento dell'esecuzione. Ovviamente questo inconveniente non si presenterebbe utilizzando una scheda processore dedicata (DSP), ma, anche nel caso nostro, può essere risolto acquistando delle estensioni software, che modificano il kernel del sistema operativo e sono in grado di far funzionare Windows NT in *real-time*.

Il software scritto, essendo stato sviluppato in due moduli distinti ed in un ambiente di sviluppo molto diffuso (Microsoft Visual Studio 6.0) risulta altamente flessibile ed estensibile per eventuali aggiornamenti futuri dell'hardware ed anche per far svolgere all'hardware già esistente compiti differenti da quelli strettamente concepiti per le finalità del Sentinel, ossia la caccia alle interferenze. La crescente evoluzione delle architetture delle CPU e le sempre più alte frequenze di clock raggiunte fanno sì che processori sempre più potenti siano velocemente prodotti e rilasciati sul mercato. Nel caso che un nuovo tipo di processore Pentium o Pentium compatibile venisse adottato nella progettazione del Sentinel 2 non sarebbe necessario alcun cambiamento nel software già scritto; nel caso invece in cui un nuovo convertitore A/D fosse disponibile sul mercato (con una maggiore dinamica od una più alta sampling rate) sarebbe sufficiente modificare i driver di gestione nel modulo *"on-line processing & control"* per aggiornare l'applicazione; il modulo *"Off-line Spectrometer"* rimarrebbe completamente invariato. Modificando l'attuale software è inoltre stato possibile, già da ora, sviluppare un sistema (chiamato Sentinel SIV) in grado di monitorare una banda radio con una risoluzione di circa 500.000 punti ottenendo in questo modo un'applicazione spettrometro ad elevatissima risoluzione.

Appendice A

L'analizzatore di spettro digitale e la Fast Fourier Transform

A.1 Definizioni e caratteristiche

Il funzionamento degli spettrometri digitali è basato sull'applicazione della trasformata di Fourier discreta (DFT) ad un insieme di dati digitali che rappresentano la forma d'onda di un segnale nel dominio del tempo. Il calcolo della DFT può essere effettuato in modo veloce ed efficiente tramite particolari algoritmi (detti di Fast Fourier Transform, FFT) eseguiti da circuiti integrati dedicati.
Lo schema di principio di uno spettrometro di questo tipo è riportato in
Fig. A.1 e comprende un circuito di campionamento e digitalizzazione del segnale, un blocco per il calcolo della FFT ed una unità *terminal* per la rappresentazione dei risultati. La filosofia di questo tipo di progetto è quella di riuscire a realizzare un sistema molto flessibile in grado di sfruttare la potenza di calcolo di un'architettura bi-processore e la semplicità di gestione ed elaborazione dati propria del calcolatore elettronico e degli strumenti informatici odierni.

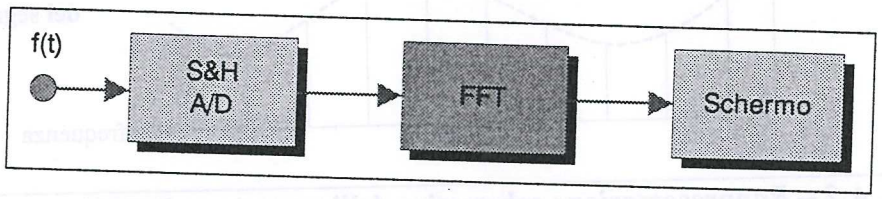


Fig. A.1 – Schema a blocchi semplificato di un normale analizzatore di spettro digitale.

L'algoritmo di FFT agisce su di un intero insieme di N campioni equidistanti nel dominio del tempo, chiamato *Time Record* e di durata pari a:

$$T_R = N\tau \qquad \text{(A.1)}$$

dove τ indica la spaziatura temporale (corrispondente al periodo di campionamento). L'operazione di campionamento e discretizzazione è effettuata dal convertitore A/D. Se vengono prelevati N campioni di un certo segnale, il teorema di Shannon garantisce una corretta trasformazione in un massimo di $R = N/2$ righe equispaziate di $1/\tau$ nel dominio delle frequenze.
L'operazione di FFT su di un blocco completo di valori campionati costituisce una caratteristica fondamentale di un analizzatore di spettro. Questo significa che uno spettro di frequenze sarà disponibile solo dopo avere acquisito ed elaborato un intero *Time Record.*

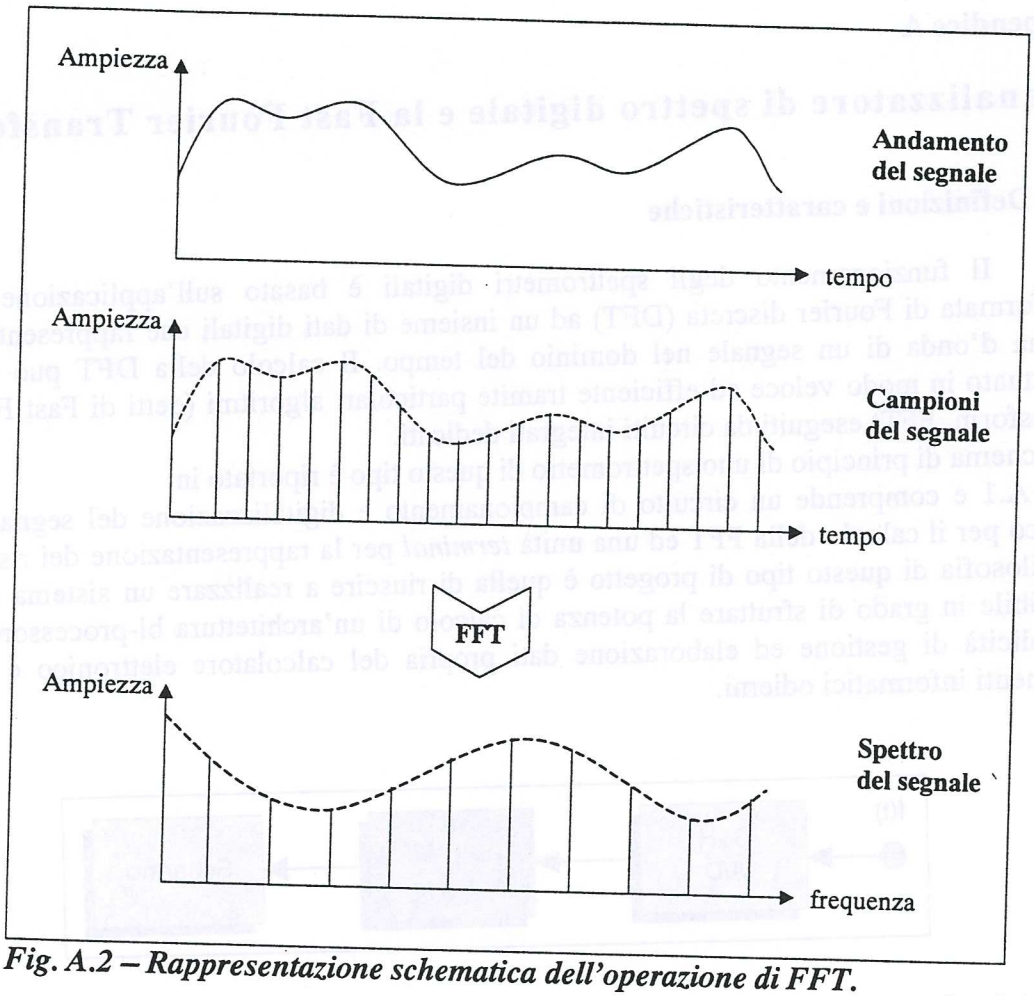


Fig. A.2 – Rappresentazione schematica dell'operazione di FFT.

A.2 La trasformata di Fourier discreta (DFT)

L'esecuzione della trasformata di Fourier per mezzo di un sistema di calcolo automatico è limitato dalle caratteristiche intrinseche di tale sistema: non è infatti possibile memorizzare un numero infinito di termini e quindi di valori. Per questo motivo l'operazione di calcolo della FFT sarà necessariamente un'approssimazione più o meno buona della *F*-trasformata continua.
Come noto, la trasformata di Fourier di una funzione continua è definita da:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-j\omega t} dt \qquad \text{(A.2)}$$

e rappresenta anch'essa una funzione continua.
La *Discrete Fourier Transform* o DFT si differenzia leggermente, in quanto elabora una serie temporale e fornisce come risultato una serie di punti discreti, coincidenti con le R righe dello spettro risultante.

Definendo con Δf la spaziatura delle righe, la (A.2) diventa:

$$F(r\Delta f) = \int_{-\infty}^{\infty} f(t) \cdot e^{-j2\pi r\Delta f t} dt \quad r = 0,1,\ldots,R-1 \qquad \text{(A.3)}$$

A questo punto l'integrazione numerica della (A.3) avviene grazie al metodo di integrazione rettangolare (Fig A.3).

Definendo con Δt la base di ogni rettangolo (coincidente con il periodo di campionamento), si ottiene:

$$F(r\Delta f) \cong \Delta t \sum_{-\infty}^{\infty} f(n\Delta t) \cdot e^{-j2\pi r\Delta f n\Delta t} \qquad \text{(A.4)}$$

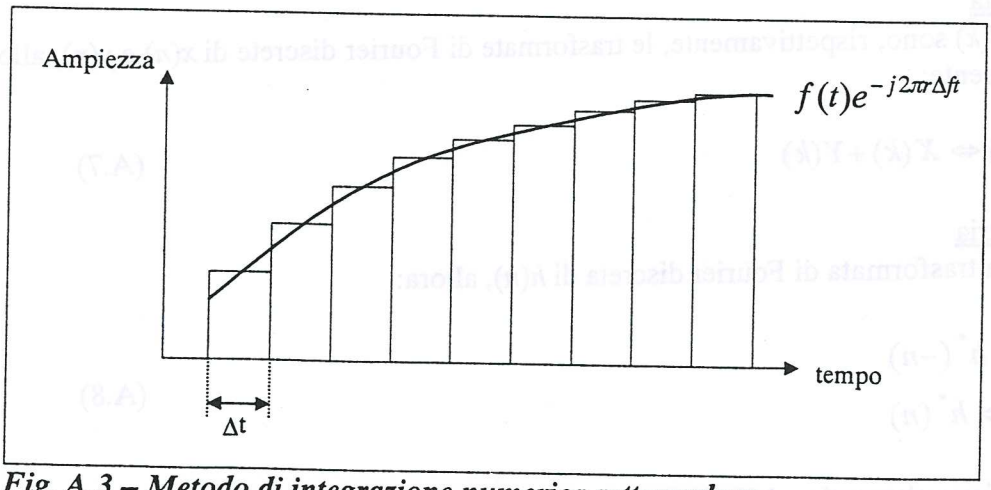


Fig. A.3 – Metodo di integrazione numerica rettangolare.

Siccome il numero di intervalli temporali sui quali viene calcolata la trasformata di Fourier non può essere infinito, la (A.3) diventa:

$$F(r\Delta f) \cong \Delta t \sum_{n=0}^{N-1} f(n\Delta t) \cdot e^{-j2\pi r \Delta f n \Delta t} \qquad \text{(A.4)}$$

Tenendo conto che la spaziatura tra due righe è:

$$\Delta f = \frac{1}{T_R}$$

e che la spaziatura tra due campioni vale:

$$\Delta t = \frac{T_R}{N}$$

si ricava in conclusione la formulazione della DFT, espressa da:

$$F(r\Delta f) \cong \frac{T_R}{N} \sum_{n=0}^{N-1} f(n\frac{T_R}{N}) \cdot e^{-j2\pi \frac{nr}{N}} \qquad \text{r = 0,1,..., R -1} \qquad \text{(A.5)}$$

Per completezza viene di seguito riportata la funzione inversa della DFT, denominata IDFT (*Inverse* DFT) che vale:

$$f(n\frac{T_r}{N}) = \frac{1}{N} \sum_{r=0}^{R-1} F(r\Delta f) \cdot e^{j2\pi \frac{rn}{N}} \qquad \text{(A.6)}$$

Proprietà della trasformata di Fourier discreta

Le proprietà più interessanti della DFT, utili per ricavare la Fast Fourier Transform sono:

- Linearità

Se $X(k)$ e $Y(k)$ sono, rispettivamente, le trasformate di Fourier discrete di $x(n)$ e $y(n)$, allora vale la seguente:

$$x(n) + y(n) \Leftrightarrow X(k) + Y(k) \qquad \text{(A.7)}$$

- Simmetria

Se $H(k)$ è la trasformata di Fourier discreta di $h(n)$, allora:

$$H^*(k) \Leftrightarrow h^*(-n)$$
$$H^*(-k) \Leftrightarrow h^*(n) \qquad \text{(A.8)}$$

- Espressione alternativa della IDFT

La DFT inversa, definita dalla (2.2.6), può anche essere scritta nella forma:

$$h(n) = \frac{1}{N}\left[\sum_{n=0}^{N-1} H^*(k) e^{-j2\pi nk/N}\right]^* \qquad \text{(A.9)}$$

dove il carattere '*' indica il complesso coniugato.
L'espressione tra parentesi altro non è che la DFT di $H^*(k)$; questa forma permette di calcolare la trasformata inversa sfruttando quella diretta: basta coniugare la funzione in ingresso alla DFT e successivamente coniugare il risultato. Ciò significa che è possibile, nella pratica implementativa, sfruttare lo stesso codice di programma per calcolare sia la trasformata diretta che quella inversa.

- Funzioni complesse

Se $h(n)$ è una funzione complessa, cioè se $h(n) = h_r(n) + jh_i(n)$ (con $h_r(n)$ e $h_i(n)$, rispettivamente, parte reale e parte immaginaria), allora la sua trasformata di Fourier discreta può essere scritta come:

$$\begin{aligned} H(k) &= \sum_{n=0}^{N-1}\left(h_r(n)\cos(\tfrac{2\pi nk}{N}) + h_i(n)sin(\tfrac{2\pi nk}{N})\right) - \\ &- j\sum_{n=0}^{N-1}\left(h_r(n)sin(\tfrac{2\pi nk}{N}) - h_i(n)\cos(\tfrac{2\pi nk}{N})\right) \end{aligned} \qquad \text{(A.10)}$$

A questo punto è semplice ricavare le proprietà legate alle classi di funzioni che a noi interessano maggiormente.

- Funzioni reali

Se $h(n)$ è puramente reale, allora $h_r(n) = h(n)$ e $h_i(n) = 0$, e dalla (A.10) si ottiene:

$$H(k) = \sum_{n=0}^{N-1} h(n)\cos(\tfrac{2\pi nk}{N}) - j\sum_{n=0}^{N-1} h(n)sin(\tfrac{2\pi nk}{N}) \qquad \text{(A.11)}$$

Siccome il coseno è una funzione pari, mentre il seno è una funzione dispari, allora la parte reale della DFT di una funzione reale è una funzione pari, mentre la parte immaginaria è una funzione dispari.

- Funzioni reali pari

Se $h_e(n)$ è una funzione reale pari, cioè se $h_e(n) = h_e(-n)$ ($e \rightarrow even$), allora la sua trasformta di Fourier discreta è una funzione pari, reale e vale:

$$h_e(n) \Leftrightarrow H_e(k) = \sum_{n=0}^{N-1} h_e(n)\cos(\tfrac{2\pi nk}{N}) \qquad \text{(A.12)}$$

Ciò si trova sostituendo $h_e(n)$ nella (2.2.11) ed osservando che la parte immaginaria si annulla, essendo una sommatoria su un numero pari di cicli di una funzione dispari. Il risultato è quindi la (2.2.12).

- Funzioni reali dispari

Se $h_o(n)$ è una funzione reale dispari, cioè se $h_o(n) = -h_o(-n)$ ($o \rightarrow odd$), allora la sua trasformta di Fourier discreta è una funzione dispari, immaginaria e vale:

$$h_o(n) \Leftrightarrow jH_o(k) = -j\sum_{n=0}^{N-1} h_o(n)sin(\frac{2\pi nk}{N}) \qquad \text{(A.13)}$$

Ciò si trova sostituendo $h_o(n)$ nella (A.11) ed osservando, in questo caso, che è la parte reale che si annulla. Il risultato è quindi la (A.13).

A.3 La Fast Fourier Transform (FFT)

La DFT derivata nel paragrafo precedente, permette il calcolo numerico dello spettro di un segnale $f(t)$. Il tempo necessario per calcolare lo spettro mediante la (A.5) dipende dal numero delle righe R richieste e dal numero di campioni N del segnale $f(t)$. Infatti, per ogni riga dello spettro è necessario risolvere la (A.5) per tutti gli N campioni del segnale da analizzare. Questa operazione può essere velocizzata usando apposite tabelle in cui sono memorizzati i valori di:

$$e^{-j2\pi\frac{nr}{N}} = W_N^{nr} \qquad \text{(A.14)}$$

detti *Twiddle Factors*, ed effettuando una moltiplicazione per ogni valore di N. Il numero C di operazioni di ricerca tabulare e di moltiplicazioni diventa pari a: $C = R \cdot N$.
Appare subito evidente che, con questo tipo di proporzionalità, il tempo di calcolo diviene presto molto grande non appena N assume valori dell'ordine di 10^3, e impedisce l'uso diretto della DFT in tutti quelle applicazioni che richiedono analisi rapide, se non addirittura il *real time*.
Per lungo tempo, pur sfruttando la simmetria della DFT e la periodicità dei coefficienti, non si riuscì a ridurre la complessità di calcolo dell'algoritmo se non di un fattore 2 o poco più, finchè nel 1965 J.W. Cooley e J.W. Tukey pubblicarono un algoritmo per il calcolo efficiente della trasformata di Fourier discreta, che poteva essere utilizzato quando N era un numero composto, cioè era il prodotto di due o più interi, e che rappresentava un reale miglioramento rispetto a tutti i lavori precedenti.
Le idee di Cooley e Tukey, contenute in quell'articolo, diedero una nuova spinta alle ricerche e da allora sono fiorite tante nuove applicazioni della DFT all'elaborazione dei segnali che hanno condotto, tra l'altro, allo sviluppo di numerosi algoritmi sempre più efficienti: gli algoritmi di Fast Fourier Transform (FFT) appunto.
Il principio fondamentale su cui si basano questi algoritmi è la scomposizione del calcolo della trasformata di Fourier discreta di una sequenza lunga N nel calcolo di trasformate discrete di dimensioni via via più piccole; i modi secondo i quali viene applicato questo principio sono molteplici e danno luogo ad una varietà di algoritmi caratterizzati da miglioramenti nella velocità di calcolo.
In base alla posizione ove si attua la decomposizione, possiamo distinguere gli algoritmi di FFT in due classi fondamentali. Nella prima, chiamata a ***decimazione in tempo*** (DIT), il processo di divisione del calcolo in trasformate via via più piccole avviene scomponendo la sequenza di ingresso x_n, e devono il loro nome al fatto che spesso l'indice n è associato al dominio del tempo. Nella seconda classe invece, è la sequenza dei coefficienti X_k della DFT che viene scomposta, da cui prendono il nome di algoritmi a ***decimazione in frequenza.***
È importante osservare che, indipendentemente dagli algoritmi impiegati, la FFT non introduce errori nel calcolo della DFT, in quanto rappresenta solamente una maniera diversa di calcolare lo stesso oggetto matematico. Eventuali errori rispetto alla trasformata continua, sono introdotti dalla quantizzazione e dalla codifica del segnale tempo-continuo originario eseguite per renderlo elaborabile da un sistema di calcolo digitale e dalle operazioni di arrotondamento compiute ad ogni operazione di somma e moltiplicazione presenti negli algoritmi.

2.4 Algoritmi per implementare la FFT

Come già detto in precedenza, la trasformata di Fourier discreta (DFT) è la funzione tempo-discreta equivalente alla trasformata di Fourier nel dominio delle funzioni tempo-continue. Mentre la trasformata continua opera con segnali x(t) continui nel tempo, la DFT opera sui campioni del segnale x_n e li trasforma in sequenze di campioni nel dominio delle frequenze X_k secondo la formula:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot W_N^{nr} \qquad k = 0,1,.....,N-1 \tag{A.15}$$

e dove i *Twiddle Factors* valgono:

$$W_N^{nr} = e^{-j2\pi\frac{nr}{N}} \tag{A.16}$$

A questa rappresentazione della DFT è associata una complessità di calcolo pari a N^2; il tempo di calcolo cresce allora rapidamente con il numero N di campioni ricevuti in ingresso e per ovviare a ciò si devono percorrere strade che conducano ad una formulazione migliore della trasformata.
Nel seguito ci occuperemo solamente degli algoritmi a decimazione in tempo, non solo perché sono quelli più frequentemente usati ma anche per il fatto che vengono implementati proprio dal chipset della scheda M62 utilizzato nella presente tesi. Ulteriori algoritmi si possono trovare consultando il riferimento bibliografico.

Gli algoritmi di tipo Radix-n

Nell'algoritmo che viene presentato, oltre ad utilizzare la possibilità di dividere la sequenza dei campioni in ingresso, si sfrutta la particolare simmetria e periodicità dei coefficienti W_N^{nk}, al fine di ottenere una ulteriore diminuzione del carico computazionale.
Questo algoritmo consente di calcolare la trasformata di Fourier di una serie temporale che sia formata da un numero di campioni pari ad una potenza intera di 2. In questa maniera esiste la possibilità di separare l'equazione (A.15) in due parti, contenenti rispettivamente gli elementi di indice pari e quelli di indice dispari:

$$X_k = \sum_{n\,\text{pari}} x_n \cdot W_N^{nk} + \sum_{n\,\text{dispari}} x_n \cdot W_N^{nk} = \tag{A.17}$$
$$= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \cdot W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} \cdot W_N^{(2n+1)k}$$

Dall'equazione (A.15) risulta che:

$$W_N^2 = e^{-2j\frac{2\pi}{N}} = e^{-j\frac{2\pi}{N/2}} = W_{N/2} \tag{A.18}$$

e sfruttando tale uguaglianza nella (A.16) si ottiene:

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \cdot W_{N/2}^{nk} + W_N^k \cdot \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} \cdot W_{N/2}^{nk} \tag{A.19}$$

Se ora si definiscono due sequenze di N/2 punti date dagli elementi pari, $g_n = x_{2n}$ e dispari $h_n = x_{2n+1}$, si perviene alla seguente formulazione:

$$\begin{aligned} X_k &= \sum_{n=0}^{\frac{N}{2}-1} g_n \cdot W_{N/2}^{nk} + W_N^k \cdot \sum_{n=0}^{\frac{N}{2}-1} h_n \cdot W_{N/2}^{nk} = \\ &= G_k + W_N^k \cdot H_k \end{aligned} \tag{A.20}$$

Questo è interpretabile definendo la trasformata di Fourier della serie temporale di lunghezza N come una opportuna combinazione delle trasformate di Fourier delle due serie temporali di lunghezza $N/2$ costituite dai termini pari e dispari della serie originaria. Questa formulazione della DFT porta ad un numero di moltiplicazioni complesse da svolgere pari a $(N^2/2 + N)$; questo significa che per valori di N molto grandi, si ottiene già un risparmio sul tempo di calcolo fino al 50% rispetto all'uso della formula (A.14). Dal momento che per gli elementi X_k il pedice può andare da 0 ad N-1, mentre per G_k e H_k i pedici variano solo da 0 a $N/2$-1 è necessario, a rigore, utilizzare una notazione che descriva la (2.4.6) quando $k \geq N/2$. Si sfrutta a tal fine la periodicità della DFT, ricordando che G_k e H_k hanno periodo pari a $N/2$; ciò permette di scrivere:

$$X_k = \begin{cases} G_k + W_N^k \cdot H_k & k = 0,1,\text{K}, \frac{N}{2}-1 \\ G_{k-\frac{N}{2}} + W_N^k \cdot H_{k-\frac{N}{2}} & k = \frac{N}{2},\text{K}, N-1 \end{cases} \tag{A.21}$$

Ipotizzando N=8, X_0 si ottiene moltiplicando H_0 per W_8^0 e sommando il prodotto con G_0; analogamente X_1 si ricava sommando G_1 al prodotto tra H_1 e W_8^1. Per quanto riguarda X_4, dovremo ottenerlo da $G_4 + W_8^0 H_4$; però, siccome G_k e H_k hanno in questo caso periodo pari a 4, risulta G_4=G_0 e H_4=H_0 e dunque ricaviamo X_4 moltiplicando H_0 per W_8^4 e sommando al prodotto G_0. Si osservi poi che, essendo $W_N^{k+\frac{N}{2}} = -W_N^k$, è possibile riscrivere la (A.21) nella forma:

$$X_k = \begin{cases} G_k + W_N^k \cdot H_k & k = 0,1,\mathrm{K}\,,\dfrac{N}{2}-1 \\ G_{k-\frac{N}{2}} - W_N^{k-\frac{N}{2}} \cdot H_{k-\frac{N}{2}} & k = \dfrac{N}{2},\mathrm{K}\,,N-1 \end{cases} \qquad \text{(A.22)}$$

Il modo di operare fin qui descritto può essere efficacemente rappresentato dalla Fig.A.4 (sempre nell'ipotesi che sia $N = 8$) che mostra con immediatezza la procedura da utilizzare.

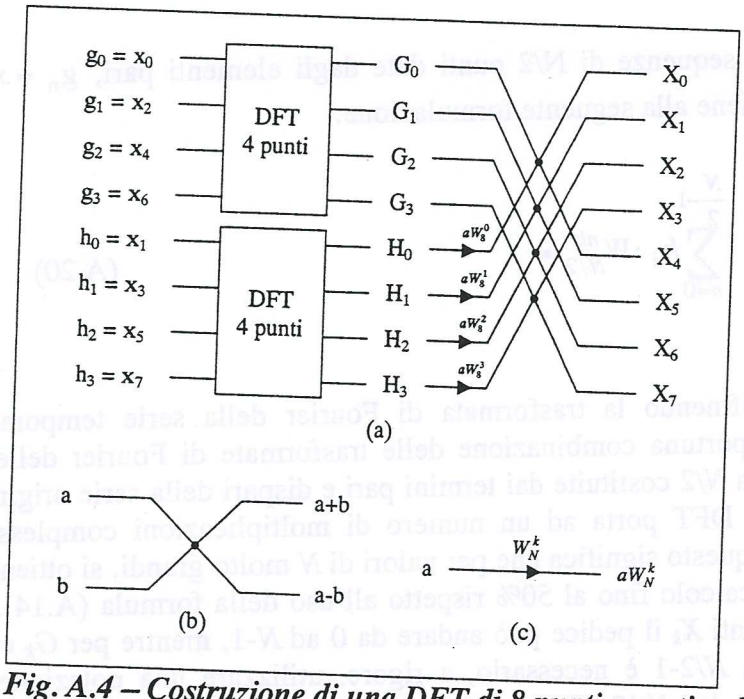


Fig. A.4 – Costruzione di una DFT di 8 punti a partire da due DFT di 4 punti (a). Notazione utilizzata (b), (c).

Questo procedimento può essere iterato dividendo a loro volta le due serie temporali G_k e H_k in coppie di serie composte da $N/4$ elementi ciascuna e così via fino ad ottenere tante serie temporali di due soli elementi.
Si arriva così ad ottenere una costruzione dell'algoritmo come quella presentata nella Fig. A. in cui si devono effettuare delle trasformate di due soli elementi alla volta, ottenendo un risparmio considerevole nel numero delle moltiplicazioni. Come si vede, vengono individuate $\log_2 N$ batterie di operazioni elementari e la complessità dell'algoritmo risulta pari a $N \cdot \log_2 N$ (come ottenuto da Cooley e Tukey).
Questo permette un grande risparmio in termini di tempo di calcolo e rende la trasformata di Fourier adatta anche per applicazioni più critiche, come quelle che richiedono il calcolo in *real time* di spettri di potenza oppure l'analisi di immagini.

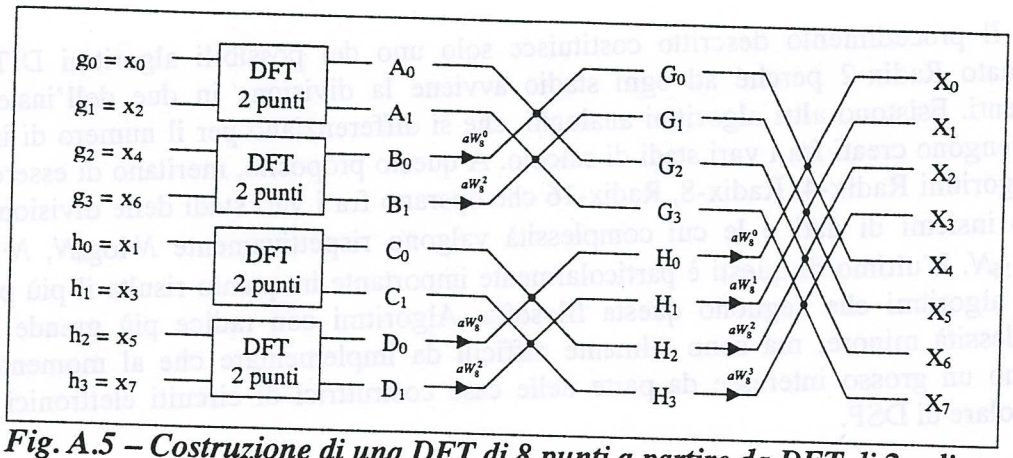


Fig. A.5 – Costruzione di una DFT di 8 punti a partire da DFT di 2 soli punti.

In questo tipo di algoritmo si utilizza un'operazione ricorrente che costituisce, tra le altre cose, la base delle più efficienti architetture di calcolo: l'**operazione Butterfly**, nella quale i due ingressi A e B sono combinati in modo da produrre le seguenti relazioni:

$$X = A + W_N^k \cdot B$$
$$Y = A - W_N^k \cdot B \qquad \text{(A.23)}$$

Per meglio chiarire questo discorso è utile fare riferimento alla Fig. A., che ne riporta una rappresentazione schematica e ne evidenzia la particolare denominazione.

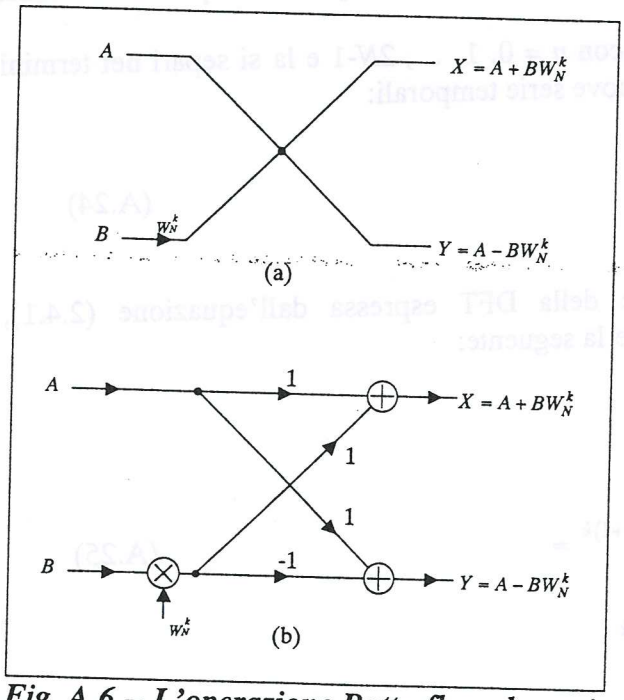


Fig. A.6 – L'operazione Butterfly: schematizzazione (a) ed evidenziazione delle singole operazioni (b).

Il procedimento descritto costituisce solo uno dei possibili algoritmi DIT ed è chiamato Radix-2 perché ad ogni stadio avviene la divisione in due dell'insieme di elementi. Esistono altri algoritmi analoghi, che si differenziano per il numero di insiemi che vengono creati fra i vari stadi di calcolo. A questo proposito, meritano di essere citati gli algoritmi Radix-4, Radix-8, Radix-16 che operano fra i vari stadi delle divisioni in 4, 8, 16 insiemi di dati e le cui complessità valgono rispettivamente $N \cdot \log_4 N$, $N \cdot \log_8 N$, $N \cdot \log_{16} N$. L'ultimo di questi è particolarmente importante in quanto risulta il più potente degli algoritmi che seguono questa filosofia. Algoritmi con radice più grande hanno complessità minore, ma sono talmente difficili da implementare che al momento non destano un grosso interesse da parte delle case costruttrici di circuiti elettronici ed in particolare di DSP.

Un algoritmo FFT per dati reali

Finora, per mantenere una certa generalità di trattazione, si sono considerate complesse tutte le funzioni, sia del tempo che della frequenza. Spesso accade che la FFT debba operare su dati che, al contrario, sono reali. In situazioni di questo tipo risulta intuitivo utilizzare lo stesso algoritmo, con l'accortezza di annullare la parte immaginaria dei valori da processare. In realtà un tale approccio è altamente inefficiente poichè verrebbero eseguite molte operazioni inutilmente.
Di seguito viene riportata una soluzione espressamente sviluppata per ottimizzare una situazione di questo tipo che, in particolare, permette di calcolare la DFT di $2N$ campioni reali fornendo in uscita N valori (che rappresentano la parte di spettro a frequenze positive, in genere sufficiente).
Si consideri una serie temporale $\{x_n\}$ con $n = 0, 1, \ldots, 2N-1$ e la si separi nei termini di indice pari e dispari, costruendo due nuove serie temporali:

$$\left.\begin{aligned} h_n &= x_{2n} \\ g_n &= x_{2n+1} \end{aligned}\right\} \qquad n = 0,1,\ldots,N-1 \tag{A.24}$$

Facendo riferimento alla definizione della DFT espressa dall'equazione (2.4.1), e sostituendovi la (2.4.10), si può scrivere la seguente:

$$\begin{aligned} X_k &= \sum_{n=0}^{2N-1} x_n \cdot W_{2N}^{nk} = \\ &= \sum_{n=0}^{N-1} x_{2n} \cdot W_{2N}^{2nk} + \sum_{n=0}^{N-1} x_{2n+1} \cdot W_{2N}^{(2n+1)k} = \\ &= \sum_{n=0}^{N-1} h_n \cdot W_N^{nk} + W_{2N}^{k} \cdot \sum_{n=0}^{N-1} g_n \cdot W_N^{nk} \end{aligned} \tag{A.25}$$

Anche in questo caso le sommatorie dei termini pari e dispari possono interpretarsi come due DFT su N punti, ossia:

$$X_k = H_k + W_{2N}^{k} \cdot G_k \tag{A.26}$$

Definiamo ora una funzione complessa che abbia come parte reale gli elementi pari e come parte immaginaria quelli dispari:

$$y_n = x_n + jg_n \quad n = 0,1,\ldots,N-1 \tag{A.27}$$

Calcolandone la DFT, si ottiene:

$$\begin{aligned} Y_k &= \sum_{n=0}^{N-1} y_n \cdot W_N^{nk} = \\ &= \sum_{n=0}^{N-1} h_n \cdot W_N^{nk} + j\sum_{n=0}^{N-1} g_n \cdot W_N^{nk} = \\ &= H_k + jG_k \end{aligned} \tag{A.28}$$

Se ora si separano le parti reali ed immaginarie di H_k e G_k, si ottiene:

$$\begin{aligned} Y_k &= [\mathrm{Re}(H_k) + j\,\mathrm{Im}(H_k)] + j[\mathrm{Re}(G_k) + j\,\mathrm{Im}(G_k)] = \\ &= [\mathrm{Re}(H_k) - \mathrm{Im}(G_k)] + j[\mathrm{Im}(H_k) + \mathrm{Re}(G_k)] \end{aligned} \tag{A.29}$$

È possibile inoltre calcolare la seguente:

$$\begin{aligned} Y_{N-k} &= \sum_{n=0}^{N-1} y_n \cdot W_N^{n(N-k)} = \\ &= \sum_{n=0}^{N-1} y_n \cdot W_N^{-nk} = \\ &= \sum_{n=0}^{N-1} h_n \cdot W_N^{-nk} + j\sum_{n=0}^{N-1} g_n \cdot W_N^{-nk} = \\ &= H_k^* + jG_k^* \end{aligned} \tag{A.30}$$

Analogamente alla (2.4.15) risulta corretto scrivere:

$$Y_{N-k} = [\mathrm{Re}(H_k) + \mathrm{Im}(H_k)] + j[\mathrm{Re}(G_k) - \mathrm{Im}(H_k)] \tag{A.31}$$

A questo punto, ricordando la definizione dei *Twiddle Factors*:

$$W_{2N}^k = e^{-j\frac{\pi k}{N}} = \cos\left(\frac{\pi k}{N}\right) - j\sin\left(\frac{\pi k}{N}\right) \tag{A.32}$$

è possibile combinare le equazioni (2.4.12), (2.4.15) e (2.4.17) per poter determinare:

$$\begin{aligned}\mathrm{Re}(X_k) &= \frac{1}{2}\mathrm{Re}(Y_k + Y_{N-k}) + \frac{1}{2}\cos\left(\frac{\pi k}{N}\right)\mathrm{Im}(Y_k + Y_{N-k}) - \\ &\quad - \frac{1}{2}sin\left(\frac{\pi k}{N}\right)\mathrm{Re}(Y_k - Y_{N-k}) \\ \mathrm{Im}(X_k) &= \frac{1}{2}\mathrm{Im}(Y_k - Y_{N-k}) + \frac{1}{2}sin\left(\frac{\pi k}{N}\right)\mathrm{Im}(Y_k + Y_{N-k}) - \\ &\quad - \frac{1}{2}\cos\left(\frac{\pi k}{N}\right)\mathrm{Re}(Y_k - Y_{N-k})\end{aligned} \tag{0.19}$$

Le (2.4.19) sono, rispettivamente, la parte reale e la parte immaginaria della trasformata di Fourier discreta X_k.
Grazie a questo algoritmo si riesce a risparmiare tempo nel calcolo della DFT e a dimezzare la quantità di memoria necessaria a contenere i risultati intermedi fra i vari stadi della FFT.
Prima di concludere occorre fare un'ultima considerazione: come già detto all'inizio del paragrafo, si ottengono in uscita solo N valori e non $2N$ come ci si sarebbe aspettato. Questo tutto sommato è sufficiente, nel senso che lo spettro di un segnale reale è speculare e presenta le seguenti caratteristiche:

$$\begin{aligned}&|F(\omega)| = |F(-\omega)| \\ &\arg(F(\omega)) = -\arg(F(-\omega)) \qquad \omega \geq 0\end{aligned} \tag{0.20}$$

A questo punto, se si vuole ottenere lo spettro completo, basta operare la ricostruzione della parte mancante sfruttando la simmetria dello spettro di ampiezza rispetto all'asse verticale e quella dello spettro di fase rispetto all'origine. Si consideri poi che, per molte applicazioni, è necessario conoscere solamente lo spettro di potenza monolatero del segnale in ingresso e pertanto in tali condizioni questo algoritmo consente un notevole risparmio di risorse hardware e software.
La Tab. 2.1 mette a confronto l'algoritmo per dati reali e quello generale. Supponendo di impiegare un medesimo DSP capace di elaborare un dato ogni μs, si evince l'effettivo miglioramento che il primo algoritmo offre rispetto al secondo: sia il numero di operazioni da eseguire che la durata complessiva del calcolo della FFT risultano inferiori.

	Algoritmo generale			**Algoritmo per dati reali**		
N° punti	**N° passi**	**Punti elaborati per passo**	**Tempo in msec**	**N° passi**	**Punti elaborati per passo**	**Tempo in msec**
256	2	256	0.512	3	128	0.384
1024	3	1024 (1K)	3.072	4	512	2.048
4096	3	4096 (4K)	12.288	4	2048 (2K)	8.196
16384	4	16384 (16K)	65.536	5	8196 (8K)	40.960
65536	4	65536 (64K)	262.144	6	32768 (32K)	196.608
262144	5	262144 (256K)	1310.720	6	131072 (128K)	786.432
1048576	5	1048576 (1M)	5242.880	7	524288 (512K)	3670.016

Tab. A.1 – Confronto tra l'algoritmo per dati reali e quello generale.

Appendice B

Sorgenti dell'applicazione Sentinel 2 – Front- End.
Ambiente di sviluppo Microsoft Visual Studio 6.0

Modulo Sentinelmain.h

```
#include <time.h>
#include <sys\timeb.h>
#include <string.h>
#include "stdio.h"
#include <fstream.h>
#include <limits.h>
#include <direct.h>
#include <conio.h>
#include <windows.h>
#include <winuser.h>
#include <commctrl.h>
#include <float.h>
#include <math.h>

#include "ultrad_defines.h"
#include "afxres.h"
#include "resource.h"
#include "ieee_32.h"                    // PC<>GPIB      under Win32 (link ieee_32m.lib for Microsoft compiler)

typedef struct sched_t {
        short   onoff;                  // se la schedulazione c'e' o no al dato minuto
        short   samplerate;
        short   numchannels;
        short   numffts;
        short   nummedie;
} sched_t;

#define         PI                              3.141592653589793       // definition of pi

//#define       AD_BOARD_SIMULATE
//#define       INTRO_IMAGE

// FFT Texas Instr. radix 2 or Ooura FFT
//#define       TEXASFFT

// configurazione più veloce (no grafica)
//#define       FASTEST

// Single antenna or multiple antenna
#define         SINGLE_MULTI_SUPPORT

#ifdef          TEXASFFT
        typedef __int64         INT64;
        #define         MAX_CHANNELS            (8192)
#else
        #define         MAX_CHANNELS            (8192)
#endif

#define         MAX_FFTS                (1024)
#define         NMAXSQRT                75      // 2+sqrt(MAX_CHANNELS/2)

// Power data representation
#define         SPECTRE_MAX_Y           UINT_MAX
#define         SPECTRE_MIN_Y           1

// Frequency range
#define         FREQ_MIN                50                      // 50  MHz
#define         FREQ_MAX                2350                    // 2350 MHz
```

```
// Filter shape
#define     MOV_AVRG_POINT          70
#define     BOX_POINTS              4
#define     LOG2_BOX_POINTS         2                   // power of 2

#define     TYPE_NORMAL             0
#define     TYPE_SERENDIP   1

#ifdef TEXASFFT
        #define     DB0YVALUE               100.0
#else
        #define     DB0YVALUE               100.0
#endif

#define     LOGARITMIC(x)   (20.0*log10((double)x/DB0YVALUE))

#define     DIR_ALL                 10

#define     SPC                     2
#define     ORG_X                   16          // label distance from spectrum screen
#define     ORG_Y                   75          // coord. Spectrum origin
#define     LEN_X                   690
#define     LEN_Y                   420         // Spectrum Dimension (multiply of 10)
#define     SER_LEN_X               512         // Spectrum Dimension (multiply of 10)
#define     BAR_LEN_X               690         // Serendip mode width
#define     BAR_LEN_Y               17
#define     BAR_X                   16
#define     BAR_Y                   532         // coord. SweepingBar origin
#define     CLK_X                   ORG_X
#define     CLK_Y                   ORG_Y+LEN_Y+22
#define     TIME_X                  CLK_X+247
#define     TIME_Y                  CLK_Y
#define     DIR_X                   ORG_X+LEN_X+19
#define     DIR_Y                   ORG_Y+LEN_Y-72      // coord. DirectionIcon origin

// Color map
#define     MY_BLACK                RGB( 0, 0, 0)
#define     MY_GRAY                 RGB(62,67,61)
#define     MY_BROWN                RGB(124, 60, 1)
#define     MY_REDMAGENTA   RGB(255, 0,170)
#define     MY_MAGENT               RGB(255, 0,255)
#define     MY_MAGENTBLUE   RGB(170, 0,255)  // viola
#define     MY_BLUE                 RGB( 0, 0,255)
#define     MY_BLUECYAN             RGB( 0,170,255)
#define     MY_CYAN                 RGB( 0,255,255)
#define     MY_CYANGREEN    RGB( 0,255,170)
#define     MY_GREEN                RGB( 0,255, 0)
#define     MY_GREENYELLOW  RGB(170,255, 0)
#define     MY_YELLOW               RGB(255,255, 0)
#define     MY_ORANGE               RGB(255,170, 0)
#define     MY_RED                  RGB(255, 0, 0)
#define     MY_WHITE                RGB(255,255,255)
#define     MY_LIGHTGRAY    RGB(192,192,192)
```

Modulo Acquire.c

```
/*-- acquire.c

This file contains the functions that handle reading data from the board
as it acquires data. --*/

#include <windows.h>
#include <stdio.h>
#include <malloc.h>

#include "dumppci.h"
#include "ultrapi.h"
#include "ultrad_defines.h"

HANDLE                          ultrad_board_handle;
int                             giBoardNumber;

/*
acquire_entire_ultrad_memory_for_file() copies the contents of the entire device
memory to a user buffer. It does this after the board has acquired data
to fill up it memory.
topic instruction
 setup_ultrad_io(board_handle, ULTRAD_READING_FROM_BOARD, ULTRAD_INTERACTIVE_IO);
*/
int acquire_entire_ultrad_memory( HANDLE board_handle,USHORT board_speed, DWORD *dataBuf)
{
        DWORD bytes_read;
        DWORD bytes_to_read_from_board;

        /*
        Calculate how many bytes to read. A chunk is how much device memory is
        filled each time an interrupt is issued. chunks_to_write is how many such chunks
        are in the device' memory.
        */
        bytes_to_read_from_board = 4*1024*1024;

        // Make sure the function parameters are valid.
        if ((board_handle == 0) || (board_speed == 0))
                return FALSE;

        // Setup the A/D speed mode.
        setup_ultrad_io(board_handle, ULTRAD_TWO_X, 0);

        // Setup the A/D_D/A speed mode.
        setup_ultrad_io(board_handle, ULTRAD_AD_DA, 0);

        // Set the transfer rate for the board's acquire.
        set_ultrad_board_register(board_handle, ULTRAD_SAMPLE_RATE,
                (USHORT) board_speed);

        // Setup the driver for the IO.
        setup_ultrad_io(board_handle, ULTRAD_READING_FROM_BOARD, ULTRAD_INTERACTIVE_IO);

        // Start the acquire.
        start_ultrad_io(board_handle);

        // Stop the acquire. Ther driver will stop the acquire once the device memory is
        // filled.
        end_ultrad_io(board_handle);

        // Read the board's device memory.
        bytes_read = read_from_ultrad_board(board_handle, (LPVOID)dataBuf,
                bytes_to_read_from_board);

        if(bytes_read==4*1024*1024)
                return TRUE;
        else
                return FALSE;
```

```
}

int acquire_ultrad_memory( HANDLE board_handle,DWORD words_to_read_from_board, USHORT board_speed, DWORD
*dataBuf)
{
        DWORD bytes_read,
                        bytes_to_read_from_board;

        bytes_to_read_from_board=4*words_to_read_from_board;

        // Make sure the function parameters are valid.
        if ((board_handle == 0) || (board_speed == 0))
                return FALSE;

        // Setup the A/D speed mode.
        setup_ultrad_io(board_handle, ULTRAD_TWO_X, 0);

        // Setup the A/D_D/A speed mode.
        setup_ultrad_io(board_handle, ULTRAD_AD_DA, 0);

        // Set the transfer rate for the board's acquire.
        set_ultrad_board_register(board_handle, ULTRAD_SAMPLE_RATE,
                (USHORT) board_speed);

        // Setup the driver for the IO.
        setup_ultrad_io(board_handle, ULTRAD_READING_FROM_BOARD, ULTRAD_INTERACTIVE_IO);

        // Start the acquire.
        start_ultrad_io(board_handle);

        // Stop the acquire. Ther driver will stop the acquire once the device memory is
        // filled.
        end_ultrad_io(board_handle);

        // Read the board's device memory.
        bytes_read = read_from_ultrad_board(board_handle, (LPVOID)dataBuf,
                bytes_to_read_from_board);

        if(bytes_read==bytes_to_read_from_board)
                return TRUE;
        else
                return FALSE;
}
```

Modulo FFT.c

```
#include  <math.h>
#include  "SentinelMain.h"

#ifdef TEXASFFT

        void                            split1(short , INT64 *, short *, short *, INT64 *);
        void                            radix2(INT64 *xy, short n, short *w);
        void                            R2DigitRevIndexTableGen(int n, int *count, unsigned short *IIndex, unsigned short
*JIndex);
        void                            digit_reverse(INT64 *yx, unsigned short *JIndex,unsigned short *IIndex, int count);

        void radix2(INT64 *xy,short n,short *w)
        {
                int             n1,n2,ie,ia,i,j,k,l;
                INT64           xt,yt,c,s;
                INT64   tmp0;

                        n2 = n;
                        ie = 1;
                        for (k=n; k > 1; k = (k >> 1) )
                        {
                                n1 = n2;
                                n2 = n2/2;
```

```
                ia = 0;
                for (j=0; j < n2; j++)
                {
                 c = w[2*ia];
                 s = w[2*ia+1];
                 ia = ia + ie;
                 for (i=j; i < n; i += n1)
                 {
                        l = i + n2;
                        xt                      = xy[2*l] - xy[2*i];
                        xy[2*i]   = xy[2*i] + xy[2*l];
                        yt                      = xy[2*l+1] - xy[2*i+1];
                        xy[2*i+1]           = xy[2*i+1] + xy[2*l+1];

                        tmp0=(INT64)c * (INT64)xt+(INT64)s * (INT64)yt;
                        xy[2*l]                 = (INT64)(tmp0>>15);
                        tmp0=(INT64)c * (INT64)yt-(INT64)s * (INT64)xt;
                        xy[2*l+1]           = (INT64)(tmp0>>15);
                 }
                }
        ie = ie*2;
        }

}

void split1(short N, INT64 *X, short *A, short *B, INT64 *G)
{

        int             k;
        INT64   Tr, Ti;

        for (k=0; k<N; k++)
        {
                Tr = X[2*k] * (INT64)A[2*k] - X[2*k+1] * (INT64)A[2*k+1] +
                        X[2*(N-k)] * (INT64)B[2*k] + X[2*(N-k)+1] * (INT64)B[2*k+1];
                G[2*k] = Tr>>15;

                Ti = X[2*k+1] * (INT64)A[2*k] + X[2*k] * (INT64)A[2*k+1] +
                        X[2*(N-k)] * (INT64)B[2*k+1] - X[2*(N-k)+1] * (INT64)B[2*k];
                G[2*k+1] = Ti>>15;
        }
}

void R2DigitRevIndexTableGen(int n, int *count, unsigned short *IIndex, unsigned short *JIndex)
{
        short j, n1, k, i;
        j = 1;
        n1 = n - 1;
        *count = 0;

        for(i=1; i<=n1; i++)
        {
                if(i < j)
                {
                        IIndex[*count] = (unsigned short)(i-1);
                        JIndex[*count] = (unsigned short)(j-1);
                        *count = *count + 1;
                }

                k = n >> 1;

                while(k < j)
                {
                        j = j - k;
                        k = k >> 1;
                }
                j = j + k;
        }
```

```
}

// int mette insieme i due short della parte reale ed
// immaginaria per ogni canale
void digit_reverse(INT64 *yx, unsigned short *JIndex, unsigned short *IIndex, int count)
{

        int i;
        unsigned short I, J;
        INT64 YXIr, YXJr,YXIi, YXJi;

        for (i = 0; i<count; i++)
        {
                I = IIndex[i];
                J = JIndex[i];
                YXIr = yx[2*I];
                YXIi = yx[2*I+1];
                YXJr = yx[2*J];
                YXJi = yx[2*J+1];
                yx[2*J] = YXIr;
                yx[2*J+1] = YXIi;
                yx[2*I] = YXJr;
                yx[2*I+1] = YXJi;
        }

}

#else

        void                            rdft(int, int, float *, int *, float *); // Ooura FFT

        /*
        -------- Real DFT / Inverse of Real DFT --------
                [definition]
                        <case1> RDFT
                                R[k] = sum_j=0^n-1 a[j]*cos(2*pi*j*k/n), 0<=k<=n/2
                                I[k] = sum_j=0^n-1 a[j]*sin(2*pi*j*k/n), 0<k<n/2
                        <case2> IRDFT (excluding scale)
                                a[k] = R[0]/2 + R[n/2]/2 +
                                        sum_j=1^n/2-1 R[j]*cos(2*pi*j*k/n) +
                                        sum_j=1^n/2-1 I[j]*sin(2*pi*j*k/n), 0<=k<n
                [usage]
                        <case1>
                                ip[0] = 0; // first time only
                                rdft(n, 1, a, ip, w);
                        <case2>
                                ip[0] = 0; // first time only
                                rdft(n, -1, a, ip, w);
                [parameters]
                        n               :data length (int)
                                                        n >= 2, n = power of 2
                        a[0...n-1]      :input/output data (float *)
                                                        <case1>
                                                                output data
                                                                        a[2*k] = R[k], 0<=k<n/2
                                                                        a[2*k+1] = I[k], 0<k<n/2
                                                                        a[1] = R[n/2]
                                                        <case2>
                                                                input data
                                                                        a[2*j] = R[j], 0<=j<n/2
                                                                        a[2*j+1] = I[j], 0<j<n/2
                                                                        a[1] = R[n/2]
                        ip[0...*]       :work area for bit reversal (int *)
                                                        length of ip >= 2+sqrt(n/2); if n % 4 == 2
                                                                                2+sqrt(n/4); otherwise
                                                        ip[0],ip[1] are pointers of the cos/sin table.
                        w[0...n/2-1]    :cos/sin table (float *)
                                                        w[],ip[] are initialized if ip[0] == 0.
                [remark]
```

```
                        Inverse of
                                rdft(n, 1, a, ip, w);
                        is
                                rdft(n, -1, a, ip, w);
                                for (j = 0; j <= n - 1; j++) {
                                        a[j] *= 2.0 / n;
                                }
*/

void rdft(int n, int isgn, float *a, int *ip, float *w)
{
        void makewt(int nw, int *ip, float *w);
        void makect(int nc, int *ip, float *c);
        void bitrv2(int n, int *ip, float *a);
        void cftsub(int n, float *a, float *w);
        void rftsub(int n, float *a, int nc, float *c);
        int j, nw, nc;
        float xi;

        nw = ip[0];
        if (n > (nw << 2)) {
                nw = n >> 2;
                makewt(nw, ip, w);
        }
        nc = ip[1];
        if (n > (nc << 2)) {
                nc = n >> 2;
                makect(nc, ip, w + nw);
        }
        if (isgn < 0) {
                a[1] = (float)0.5 * (a[1] - a[0]);
                a[0] += a[1];
                for (j = 3; j <= n - 1; j += 2) {
                        a[j] = -a[j];
                }
                if (n > 4) {
                        rftsub(n, a, nc, w + nw);
                        bitrv2(n, ip + 2, a);
                }
                cftsub(n, a, w);
                for (j = 1; j <= n - 1; j += 2) {
                        a[j] = -a[j];
                }
        } else {
                if (n > 4) {
                        bitrv2(n, ip + 2, a);
                }
                cftsub(n, a, w);
                if (n > 4) {
                        rftsub(n, a, nc, w + nw);
                }
                xi = a[0] - a[1];
                a[0] += a[1];
                a[1] = xi;
        }
}


/* -------- initializing routines -------- */

void makewt(int nw, int *ip, float *w)
{
        void bitrv2(int n, int *ip, float *a);
        int nwh, j;
        float delta, x, y;

        ip[0] = nw;
        ip[1] = 1;
```

```
        if (nw > 2) {
                nwh = nw >> 1;
                delta = (float)atan(1.0) / nwh;
                w[0] = 1;
                w[1] = 0;
                w[nwh] = (float)cos(delta * nwh);
                w[nwh + 1] = w[nwh];
                for (j = 2; j <= nwh - 2; j += 2) {
                        x = (float)cos(delta * j);
                        y = (float)sin(delta * j);
                        w[j] = x;
                        w[j + 1] = y;
                        w[nw - j] = y;
                        w[nw - j + 1] = x;
                }
                bitrv2(nw, ip + 2, w);
        }
}


void makect(int nc, int *ip, float *c)
{
        int nch, j;
        float delta;

        ip[1] = nc;
        if (nc > 1) {
                nch = nc >> 1;
                delta = (float)atan(1.0) / nch;
                c[0] = 0.5;
                c[nch] = (float)0.5 * (float)cos(delta * nch);
                for (j = 1; j <= nch - 1; j++) {
                        c[j] = (float)0.5 * (float)cos(delta * j);
                        c[nc - j] = (float)0.5 * (float)sin(delta * j);
                }
        }
}


/* -------- child routines -------- */


void bitrv2(int n, int *ip, float *a)
{
        int j, j1, k, k1, l, m, m2;
        float xr, xi;

        ip[0] = 0;
        l = n;
        m = 1;
        while ((m << 2) < l) {
                l >>= 1;
                for (j = 0; j <= m - 1; j++) {
                        ip[m + j] = ip[j] + l;
                }
                m <<= 1;
        }
        if ((m << 2) > l) {
                for (k = 1; k <= m - 1; k++) {
                        for (j = 0; j <= k - 1; j++) {
                                j1 = (j << 1) + ip[k];
                                k1 = (k << 1) + ip[j];
                                xr = a[j1];
                                xi = a[j1 + 1];
                                a[j1] = a[k1];
                                a[j1 + 1] = a[k1 + 1];
                                a[k1] = xr;
                                a[k1 + 1] = xi;
                        }
```

```
                }
        } else {
                m2 = m << 1;
                for (k = 1; k <= m - 1; k++) {
                        for (j = 0; j <= k - 1; j++) {
                                j1 = (j << 1) + ip[k];
                                k1 = (k << 1) + ip[j];
                                xr = a[j1];
                                xi = a[j1 + 1];
                                a[j1] = a[k1];
                                a[j1 + 1] = a[k1 + 1];
                                a[k1] = xr;
                                a[k1 + 1] = xi;
                                j1 += m2;
                                k1 += m2;
                                xr = a[j1];
                                xi = a[j1 + 1];
                                a[j1] = a[k1];
                                a[j1 + 1] = a[k1 + 1];
                                a[k1] = xr;
                                a[k1 + 1] = xi;
                        }
                }
        }
}


void cftsub(int n, float *a, float *w)
{
        int j, j1, j2, j3, k, k1, ks, l, m;
        float wk1r, wk1i, wk2r, wk2i, wk3r, wk3i;
        float x0r, x0i, x1r, x1i, x2r, x2i, x3r, x3i;

        l = 2;
        while ((l << 1) < n) {
                m = l << 2;
                for (j = 0; j <= l - 2; j += 2) {
                        j1 = j + l;
                        j2 = j1 + l;
                        j3 = j2 + l;
                        x0r = a[j] + a[j1];
                        x0i = a[j + 1] + a[j1 + 1];
                        x1r = a[j] - a[j1];
                        x1i = a[j + 1] - a[j1 + 1];
                        x2r = a[j2] + a[j3];
                        x2i = a[j2 + 1] + a[j3 + 1];
                        x3r = a[j2] - a[j3];
                        x3i = a[j2 + 1] - a[j3 + 1];
                        a[j] = x0r + x2r;
                        a[j + 1] = x0i + x2i;
                        a[j2] = x0r - x2r;
                        a[j2 + 1] = x0i - x2i;
                        a[j1] = x1r - x3i;
                        a[j1 + 1] = x1i + x3r;
                        a[j3] = x1r + x3i;
                        a[j3 + 1] = x1i - x3r;
                }
                if (m < n) {
                        wk1r = w[2];
                        for (j = m; j <= l + m - 2; j += 2) {
                                j1 = j + l;
                                j2 = j1 + l;
                                j3 = j2 + l;
                                x0r = a[j] + a[j1];
                                x0i = a[j + 1] + a[j1 + 1];
                                x1r = a[j] - a[j1];
                                x1i = a[j + 1] - a[j1 + 1];
                                x2r = a[j2] + a[j3];
                                x2i = a[j2 + 1] + a[j3 + 1];
```

```
                x3r = a[j2] - a[j3];
                x3i = a[j2 + 1] - a[j3 + 1];
                a[j] = x0r + x2r;
                a[j + 1] = x0i + x2i;
                a[j2] = x2i - x0i;
                a[j2 + 1] = x0r - x2r;
                x0r = x1r - x3i;
                x0i = x1i + x3r;
                a[j1] = wk1r * (x0r - x0i);
                a[j1 + 1] = wk1r * (x0r + x0i);
                x0r = x3i + x1r;
                x0i = x3r - x1i;
                a[j3] = wk1r * (x0i - x0r);
                a[j3 + 1] = wk1r * (x0i + x0r);
            }
            k1 = 1;
            ks = -1;
            for (k = (m << 1); k <= n - m; k += m) {
                k1++;
                ks = -ks;
                wk1r = w[k1 << 1];
                wk1i = w[(k1 << 1) + 1];
                wk2r = ks * w[k1];
                wk2i = w[k1 + ks];
                wk3r = wk1r - 2 * wk2i * wk1i;
                wk3i = 2 * wk2i * wk1r - wk1i;
                for (j = k; j <= l + k - 2; j += 2) {
                    j1 = j + l;
                    j2 = j1 + l;
                    j3 = j2 + l;
                    x0r = a[j] + a[j1];
                    x0i = a[j + 1] + a[j1 + 1];
                    x1r = a[j] - a[j1];
                    x1i = a[j + 1] - a[j1 + 1];
                    x2r = a[j2] + a[j3];
                    x2i = a[j2 + 1] + a[j3 + 1];
                    x3r = a[j2] - a[j3];
                    x3i = a[j2 + 1] - a[j3 + 1];
                    a[j] = x0r + x2r;
                    a[j + 1] = x0i + x2i;
                    x0r -= x2r;
                    x0i -= x2i;
                    a[j2] = wk2r * x0r - wk2i * x0i;
                    a[j2 + 1] = wk2r * x0i + wk2i * x0r;
                    x0r = x1r - x3i;
                    x0i = x1i + x3r;
                    a[j1] = wk1r * x0r - wk1i * x0i;
                    a[j1 + 1] = wk1r * x0i + wk1i * x0r;
                    x0r = x1r + x3i;
                    x0i = x1i - x3r;
                    a[j3] = wk3r * x0r - wk3i * x0i;
                    a[j3 + 1] = wk3r * x0i + wk3i * x0r;
                }
            }
        }
        l = m;
    }
    if (l < n) {
        for (j = 0; j <= l - 2; j += 2) {
            j1 = j + l;
            x0r = a[j] - a[j1];
            x0i = a[j + 1] - a[j1 + 1];
            a[j] += a[j1];
            a[j + 1] += a[j1 + 1];
            a[j1] = x0r;
            a[j1 + 1] = x0i;
        }
    }
}
```

```
void rftsub(int n, float *a, int nc, float *c)
{
        int j, k, kk, ks;
        float wkr, wki, xr, xi, yr, yi;

        ks = (nc << 2) / n;
        kk = 0;
        for (k = (n >> 1) - 2; k >= 2; k -= 2) {
                j = n - k;
                kk += ks;
                wkr = (float)0.5 - c[kk];
                wki = c[nc - kk];
                xr = a[k] - a[j];
                xi = a[k + 1] + a[j + 1];
                yr = wkr * xr - wki * xi;
                yi = wkr * xi + wki * xr;
                a[k] -= yr;
                a[k + 1] -= yi;
                a[j] += yr;
                a[j + 1] -= yi;
        }
}


void dctsub(int n, float *a, int nc, float *c)
{
        int j, k, kk, ks, m;
        float wkr, wki, xr;

        ks = nc / n;
        kk = ks;
        m = n >> 1;
        for (k = 1; k <= m - 1; k++) {
                j = n - k;
                wkr = c[kk] - c[nc - kk];
                wki = c[kk] + c[nc - kk];
                kk += ks;
                xr = wki * a[k] - wkr * a[j];
                a[k] = wkr * a[k] + wki * a[j];
                a[j] = xr;
        }
        a[m] *= 2 * c[kk];
}


void dstsub(int n, float *a, int nc, float *c)
{
        int j, k, kk, ks, m;
        float wkr, wki, xr;

        ks = nc / n;
        kk = ks;
        m = n >> 1;
        for (k = 1; k <= m - 1; k++) {
                j = n - k;
                wkr = c[kk] - c[nc - kk];
                wki = c[kk] + c[nc - kk];
                kk += ks;
                xr = wki * a[j] - wkr * a[k];
                a[j] = wkr * a[j] + wki * a[k];
                a[k] = xr;
        }
        a[m] *= 2 * c[kk];
}

#endif
```

Modulo Install.c

```
/*++

Module Name:

  install.c

Abstract:

  Win32 routines to dynamically load and unload a Windows NT kernel-mode
  driver using the Service Control Manager APIs.

Environment:

  User mode only

Author:

  Steve Dziok

Revision History:

  01-Jul-1996:  Created

--*/

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "dumppci.h"


BOOLEAN
InstallDriver(
  IN SC_HANDLE  SchSCManager,
  IN LPCTSTR    DriverName,
  IN LPCTSTR    ServiceExe
  );

BOOLEAN
ManageDriver(
  IN LPCTSTR  DriverName,
  IN LPCTSTR  ServiceName,
  IN USHORT   Function
  );

BOOLEAN
RemoveDriver(
  IN SC_HANDLE  SchSCManager,
  IN LPCTSTR    DriverName
  );

BOOLEAN
StartDriver(
  IN SC_HANDLE  SchSCManager,
  IN LPCTSTR    DriverName
  );

BOOLEAN
StopDriver(
  IN SC_HANDLE  SchSCManager,
  IN LPCTSTR    DriverName
  );
```

```
BOOLEAN
InstallDriver(
    IN SC_HANDLE  SchSCManager,
    IN LPCTSTR    DriverName,
    IN LPCTSTR    ServiceExe
    )
/*++

Routine Description:

Arguments:

Return Value:

--*/
{
  SC_HANDLE   schService;
  DWORD       err;

  //
  // NOTE: This creates an entry for a standalone driver. If this
  //       is modified for use with a driver that requires a Tag,
  //       Group, and/or Dependencies, it may be necessary to
  //       query the registry for existing driver information
  //       (in order to determine a unique Tag, etc.).
  //

  //
  // Create a new a service object.
  //

  schService = CreateService(SchSCManager,          // handle of service control manager database
                    DriverName,             // address of name of service to start
                    DriverName,             // address of display name
                    SERVICE_ALL_ACCESS,     // type of access to service
                    SERVICE_KERNEL_DRIVER,  // type of service
                    SERVICE_DEMAND_START,   // when to start service
                    SERVICE_ERROR_NORMAL,   // severity if service fails to start
                    ServiceExe,             // address of name of binary file
                    NULL,                   // service does not belong to a group
                    NULL,                   // no tag requested
                    NULL,                   // no dependency names
                    NULL,                   // use LocalSystem account
                    NULL                    // no password for service account
                    );

  if (schService == NULL) {

    err = GetLastError();

    if (err == ERROR_SERVICE_EXISTS) {

      //
      // Ignore this error.
      //

      return TRUE;

    } else {

      printf("CreateService failed! Error = %d \n", err );

      //
      // Indicate an error.
      //

      return FALSE;
    }
  }
```

```
    //
    // Close the service object.
    //

    if (schService) {

      CloseServiceHandle(schService);
    }

    //
    // Indicate success.
    //

    return TRUE;

  } // InstallDriver

  BOOLEAN
  ManageDriver(
    IN LPCTSTR  DriverName,
    IN LPCTSTR  ServiceName,
    IN USHORT   Function
    )
  {

    SC_HANDLE  schSCManager;

    BOOLEAN rCode = TRUE;

    //
    // Insure (somewhat) that the driver and service names are valid.
    //

    if (!DriverName || !ServiceName) {

      printf("Invalid Driver or Service provided to ManageDriver() \n");

      return FALSE;
    }

    //
    // Connect to the Service Control Manager and open the Services database.
    //

    schSCManager = OpenSCManager(NULL,                  // local machine
                                 NULL,                  // local database
                                 SC_MANAGER_ALL_ACCESS  // access required
                                 );

    if (!schSCManager) {

      printf("Open SC Manager failed! Error = %d \n", GetLastError());

      return FALSE;
    }

    //
    // Do the requested function.
    //

    switch( Function ) {

      case DRIVER_FUNC_INSTALL:

        //
        // Install the driver service.
        //
```

```
            if (InstallDriver(schSCManager,
                              DriverName,
                              ServiceName
                              )) {

              //
              // Start the driver service (i.e. start the driver).
              //

              rCode = StartDriver(schSCManager,
                                  DriverName
                                  );

            } else {

              //
              // Indicate an error.
              //

              rCode = FALSE;
            }

            break;

        case DRIVER_FUNC_REMOVE:

#ifdef MULTIPLE_BOARD_SUPPORT
            //
            // Stop the driver.
            //

            StopDriver(schSCManager,
                       DriverName
                       );

            //
            // Remove the driver service.
            //

            RemoveDriver(schSCManager,
                         DriverName
                         );

            //
            // Ignore all errors.
            //
#endif
            rCode = TRUE;

            break;

        default:

            printf("Unknown ManageDriver() function. \n");

            rCode = FALSE;

            break;
    }

    //
    // Close handle to service control manager.
    //

    if (schSCManager) {

        CloseServiceHandle(schSCManager);
    }
```

```
    return rCode;

} // ManageDriver


BOOLEAN
RemoveDriver(
  IN SC_HANDLE   SchSCManager,
  IN LPCTSTR     DriverName
  )
{
  SC_HANDLE  schService;
  BOOLEAN    rCode;

  //
  // Open the handle to the existing service.
  //
  schService = OpenService(SchSCManager,
                           DriverName,
                           SERVICE_ALL_ACCESS
                           );

  if (schService == NULL) {

    printf("OpenService failed!  Error = %d \n", GetLastError());

    //
    // Indicate error.
    //

    return FALSE;
  }

  //
  // Mark the service for deletion from the service control manager database.
  //

  if (DeleteService(schService)) {

    //
    // Indicate success.
    //

    rCode = TRUE;

  } else {

    printf("DeleteService failed!  Error = %d \n", GetLastError());

    //
    // Indicate failure.  Fall through to properly close the service handle.
    //

    rCode = FALSE;
  }

  //
  // Close the service object.
  //

  if (schService) {

    CloseServiceHandle(schService);
  }

  return rCode;

} // RemoveDriver
```

```
BOOLEAN
StartDriver(
  IN SC_HANDLE  SchSCManager,
  IN LPCTSTR    DriverName
  )
{
  SC_HANDLE  schService;
  DWORD      err;

  //
  // Open the handle to the existing service.
  //

  schService = OpenService(SchSCManager,
                           DriverName,
                           SERVICE_ALL_ACCESS
                           );

  if (schService == NULL) {

    printf("OpenService failed!  Error = %d \n", GetLastError());

    //
    // Indicate failure.
    //

    return FALSE;
  }

  //
  // Start the execution of the service (i.e. start the driver).
  //

  if (!StartService(schService,    // service identifier
                    0,             // number of arguments
                    NULL           // pointer to arguments
                    )) {

    err = GetLastError();

    if (err == ERROR_SERVICE_ALREADY_RUNNING) {

      //
      // Ignore this error.
      //

      return TRUE;

    } else {

      printf("StartService failure! Error = %d \n", err );

      //
      // Indicate failure.  Fall through to properly close the service handle.
      //

      return FALSE;
    }

  }

  //
  // Close the service object.
  //

  if (schService) {
```

```
        CloseServiceHandle(schService);
    }
        return TRUE;

} // StartDriver


BOOLEAN
StopDriver(
  IN SC_HANDLE   SchSCManager,
  IN LPCTSTR     DriverName
  )
{
  BOOLEAN       rCode = TRUE;
  SC_HANDLE     schService;
  SERVICE_STATUS  serviceStatus;

  //
  // Open the handle to the existing service.
  //

  schService = OpenService(SchSCManager,
                DriverName,
                SERVICE_ALL_ACCESS
                );

  if (schService == NULL) {

    printf("OpenService failed!  Error = %d \n", GetLastError());

    return FALSE;
  }

  //
  // Request that the service stop.
  //

  if (ControlService(schService,
            SERVICE_CONTROL_STOP,
            &serviceStatus
            )) {

    //
    // Indicate success.
    //

    rCode = TRUE;

  } else {

    printf("ControlService failed!  Error = %d \n", GetLastError() );

    //
    // Indicate failure.  Fall through to properly close the service handle.
    //

    rCode = FALSE;
}

//
// Close the service object.
//

if (schService) {

  CloseServiceHandle (schService);
}
```

```
    return rCode;

  } // StopDriver

  BOOLEAN
  SetupDriverName(
    PUCHAR DriverLocation
    )
  {
    HANDLE fileHandle;

    DWORD driverLocLen = 0;

    //
    // Get the current directory.
    //

    driverLocLen = GetCurrentDirectory(MAX_PATH,
                                       DriverLocation
                                       );

    if (!driverLocLen) {

      printf("GetCurrentDirectory failed!  Error = %d \n", GetLastError());

      return FALSE;
    }

    //
    // Setup path name to driver file.
    //

    strcat(DriverLocation, "\\");
    strcat(DriverLocation, PCIDUMPR_DRIVER_NAME);
    strcat(DriverLocation, ".sys");

    //
    // Insure driver file is in the current directory.
    //

    if ((fileHandle = CreateFile(DriverLocation,
                                 GENERIC_READ,
                                 0,
                                 NULL,
                                 OPEN_EXISTING,
                                 FILE_ATTRIBUTE_NORMAL,
                                 NULL
                                 )) == INVALID_HANDLE_VALUE) {

      printf("The file %s does not exist. \n", DriverLocation);

      //
      // Indicate failure.
      //

      return FALSE;
    }

    //
    // Close open file handle.
    //

    if (fileHandle) {

      CloseHandle(fileHandle);
    }

  //
```

```
    // Indicate success.
    //

    return TRUE;


} // SetupDriverName
```

Modulo ultrapi.c

```
/*++

 ultrapi.c

 This file contains the "helper" functions, or API, for connecting to and
reading and writing from and to the ULTRAD board.

--*/

#include <windows.h>
#include "devioctl.h"
#include <stdio.h>
#include <malloc.h>

#include "dumppci.h"
#include "ultrapi.h"
#include "ultrad_defines.h"

extern int giBoardNumber;

/*
open_ultrad_board() loads the ULTRAD device driver, which then connects to the
board. open_ultrad_board() then returns a handle to the board. This handle is then
used for the various operations on the board.

*/
HANDLE open_ultrad_board()
{


   HANDLE fileHandle;
   UCHAR serviceName[MAX_PATH];
#ifdef _WIN32_WINNT
        UCHAR l_cBoardNumber[2];
#endif

   strcpy(serviceName, "\\\\.\\");
   strcat(serviceName, PCIDUMPR_DRIVER_NAME);

#ifndef _WIN32_WINNT
   strcat(serviceName, ".VXD");

        fileHandle = CreateFile(
                serviceName,
                GENERIC_READ | GENERIC_WRITE,
                0,
                NULL,
                OPEN_EXISTING,
                FILE_FLAG_DELETE_ON_CLOSE,//FILE_ATTRIBUTE_NORMAL,
                NULL
                );
#else
   //
   // Try to connect to driver that is already loaded.
        // If this fails, try to load the driver
   // dynamically.
   //
```

```
            sprintf(l_cBoardNumber,"%d",giBoardNumber);
            strcat(serviceName, l_cBoardNumber);
//          printf("\nSelected Borad :%s\n",serviceName);
    if ((fileHandle = CreateFile(serviceName,
                        GENERIC_READ | GENERIC_WRITE,
                        0,
                        NULL,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        NULL
                        )) == INVALID_HANDLE_VALUE)
            {
                        DWORD errNum = 0;
                        UCHAR driverLocation[MAX_PATH];

      errNum = GetLastError();

                        if (errNum != ERROR_FILE_NOT_FOUND)
                        {

       printf("The driver couldn't be connected to.  Error = %d\n", errNum);

       //
       // Indicate failure.
       //

       return INVALID_HANDLE_VALUE;
     }

     //
     // Setup full path to driver name.
     //

     if (!SetupDriverName(driverLocation))
                    {

       //
       // Indicate failure.
       //

       return INVALID_HANDLE_VALUE;
     }

     //
     // Install driver.
     //
     if (!ManageDriver(PCIDUMPR_DRIVER_NAME,
                driverLocation,
                DRIVER_FUNC_INSTALL
                ))
                    {
      printf("Unable to load driver.\n\
Make sure the Ultrad driver and board exist and are in proper places\n\
and that you have permission to load the driver.");

      //
      // Error - remove driver.
      //

      ManageDriver(PCIDUMPR_DRIVER_NAME,
            driverLocation,
            DRIVER_FUNC_REMOVE
            );

      //
      // Indicate error.
      //
```

```
        return INVALID_HANDLE_VALUE;
    }

    //
    // Try to open the newly installed driver.
    //
    if ((fileHandle = CreateFile(serviceName,
                        GENERIC_READ | GENERIC_WRITE,
                        0,
                        NULL,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        NULL
                        )) == INVALID_HANDLE_VALUE)
                    {

        printf("Starting driver failed!  Error = %d\n", GetLastError() );
        printf("Unable to start driver.\n\
Make sure Ultrad driver and board exist and are in proper places\n");

        //
        // Indicate failure.
        //

        return INVALID_HANDLE_VALUE;
    }
        }
#endif // _WIN32_WINNT

        return fileHandle;
}

/*
close_ultrad_board() closes the connection to the board and
unloads the device driver.
*/

BOOL close_ultrad_board(IN HANDLE fileHandle)
{
#ifdef _WIN32_WINNT
        UCHAR driverLocation[MAX_PATH];
#endif

// Close the connection to the board.

  if (fileHandle)
        {

    CloseHandle(fileHandle);
  }
#ifdef _WIN32_WINNT
        // Get the driver's name.
  if (!SetupDriverName(driverLocation))
        {
                // Indicate failure.
                return 0;
        }
        else
        {

                //
                // Unload the driver.  Ignore any errors.
                //

                ManageDriver(PCIDUMPR_DRIVER_NAME,
                                        driverLocation,
                                        DRIVER_FUNC_REMOVE
                                        );
        }
```

```
#endif
   return TRUE;

}

/*
write_to_ultrad_board() copies a user buffer to the boards device memory.
It returns the number of bytes it copied.
*/
DWORD write_to_ultrad_board(IN HANDLE fileHandle, IN LPCVOID data, IN DWORD buffer_size)
{
        DWORD number_of_bytes_written = 0;
        BOOL result;

        // Make sure there's a connection to the driver.

        if (!fileHandle)
        {
                return FALSE;
        }
#ifndef _WIN32_WINNT
        // Write the buffer to the board.
        result = DeviceIoControl( fileHandle, IOCTL_ULTRAD_WRITE, (LPVOID)data,
                        buffer_size, NULL, 0, &number_of_bytes_written, NULL);
#else
        // Write the buffer to the board.
        result = WriteFile(fileHandle, data, buffer_size, &number_of_bytes_written, NULL);
#endif
        return number_of_bytes_written;

}

/*
read_from_ultrad_board() copies part or all of the board's device memory to
a user buffer. It returns the status of the copy.
*/
BOOL read_from_ultrad_board(IN HANDLE fileHandle, LPVOID data, IN DWORD buffer_size)
{
        DWORD number_of_bytes_read = 0;
        BOOLEAN result;

        //Make sure there's a connection to the driver.
        if (!fileHandle)
                return FALSE;

#ifndef _WIN32_WINNT
        // Copy the device memory to the user buffer.
        result = DeviceIoControl( fileHandle, IOCTL_ULTRAD_READ, NULL,
                        0, data, buffer_size, &number_of_bytes_read, NULL);
#else
        // Copy the device memory to the user buffer.
        result = ReadFile(fileHandle, data, buffer_size, &number_of_bytes_read, NULL);
#endif
        return number_of_bytes_read;

}

/*
set_ultrad_board_register() changes a field in the board's register to
the desired setting. It returns the status of the change.
*/
BOOL set_ultrad_board_register(IN HANDLE fileHandle, IN WORD which_parameter, IN WORD setting)
{
        BOOL    result;
        DWORD control_code, hold;

// Make sure the new setting is allowed.

        result = check_register_parameters(which_parameter, setting);
```

```
        if (result == FALSE)
        {
                return FALSE;
        }

/*
Combine the identifier for the setting and the setting
itself into a single number. This is needed so that DeviceIoControl() can
be used to send the number to the device driver.
*/

        control_code = MAKELONG(setting, which_parameter);

// Send the register setting to the driver.

        result = DeviceIoControl( fileHandle, IOCTL_ULTRAD_SET_REGISTER, &control_code,
                        sizeof(ULONG), NULL, 0, &hold, NULL);

        return result;

}

/*
setup_ultrad_io() prepares the device driver for the input or output to follow.
The driver must know whether an 'acquire' or a 'spitout' operation follows.
It also must know whether one large chunks of memory is going to copied after
the board runs or whether memory will be copied in chinks as the board is running.
*/
BOOL setup_ultrad_io(IN HANDLE fileHandle, IN WORD io_direction,
                                        IN WORD io_type)
{
        BOOL    result;
        DWORD control_code, hold;

// Make sure the type of IO specified is valid.

        if ((io_type != ULTRAD_FILE_IO) &&
                        (io_type != ULTRAD_INTERACTIVE_IO))
        {
                        return FALSE;
        }
// Make sure the direction of the IO, 'acquire' or 'spitout'. is valid.

        if ((io_direction != ULTRAD_WRITING_TO_BOARD) &&
                        (io_direction != ULTRAD_READING_FROM_BOARD))
        {
                        return FALSE;
        }

// Combine the IO type and direction into a single number that can be sent
// to the board via DeviceIoControl()

        control_code = MAKELONG(io_direction, io_type);

// Send the IO type and direction that will follow to the board.

        result = DeviceIoControl( fileHandle, IOCTL_ULTRAD_SETUP_IO, &control_code,
                        sizeof(ULONG), NULL, 0, &hold, NULL);

                return result;
}

/*
start_ultrad_io() start the board's acquiring and spitting out of data.
*/

BOOL start_ultrad_io(IN HANDLE fileHandle)
{
```

```
        BOOL    result;
        DWORD hold;

// Tell the driver, through DeviceIoControl(), to start the board.

        result = DeviceIoControl( fileHandle, IOCTL_ULTRAD_START_IO, NULL, 0,
                        NULL, 0, &hold, NULL);

                return result;
}

/*
check_register_parameters() make sure that valid paremeters are used
when trying to reset the board's control register.
*/

BOOL check_register_parameters(IN WORD which_parameter, IN BOOL parameter_setting)
{

// Certain parameters must be set to either TRUE or FALSE.

        switch (which_parameter)
        {
                case ULTRAD_CLEAR_CONTROL_REGISTER:

                break;
                case ULTRAD_SOFTWARE_RUN:
                case ULTRAD_BUFFER_WRAP:
                case ULTRAD_TWO_X:
                case ULTRAD_AD_DA:
                case ULTRAD_AD_MODE:
                case ULTRAD_RSSTB:
                case ULTRAD_RSCLK:
//              case ULTRAD_TIMESTAMP_TEST:

                        if ((parameter_setting != TRUE) && (parameter_setting != FALSE))
                        {
                                return FALSE;
                        }
                break;

// Other parameters must be set to a two-bit number.

                case ULTRAD_BUFFER_INTERRUPT_SIZE:
                case ULTRAD_INTERRUPT_LEVEL:

                        if (parameter_setting > 3)
                        {
                                return FALSE;
                        }
                        break;

// The Sample rate for the board must be set to a number between 1 and 255

                case ULTRAD_SAMPLE_RATE:

                        if (parameter_setting > 255)
                        {
                                return FALSE;
                        }
                break;

                default:

                        return FALSE;
        }
        return TRUE;
}
```

```
/*
end_ultrad_io() tells the board toi stop either acquring or spiting out data.
*/

BOOL end_ultrad_io(IN HANDLE fileHandle)
{
        BOOL    result;
        ULONG hold;

// Use DeviceIoControl() to tell the board to stop acquring/spitting out.

        result = DeviceIoControl( fileHandle, IOCTL_ULTRAD_END_IO, NULL,
                        0, NULL, 0, &hold, NULL);

                return result;
}

/*
open_data_file_for_writing() opens a file to write ULTRAD data into.
It returns the Handle for the file.
*/

HANDLE open_data_file_for_writing(char *data_file_name)
{
        HANDLE file_handle;

// If a file already exists, delete it.

        DeleteFile(data_file_name);

// Open the file for writing.

        file_handle = CreateFile(data_file_name, GENERIC_WRITE, 0, NULL,
                CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
        return file_handle;
}

/*
open_data_file_for_reading() opens a file to read ULTRAD data from.
It returns the Handle for the file.
*/

HANDLE open_data_file_for_reading(char *data_file_name)
{
        HANDLE file_handle;

// Open the file for reading.

                file_handle = CreateFile(data_file_name, GENERIC_READ, 0, NULL,
                        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

                return file_handle;
}

/*
write_data_to_file() writes a buffer of data into a file.
It returns its success or failure.
*/

BOOL write_data_to_file(PVOID dataBuf, HANDLE data_file_handle, ULONG bytes_to_write)
{
        DWORD bytes_written;
        BOOL result;

        result = WriteFile(data_file_handle, dataBuf, bytes_to_write,
                &bytes_written, NULL);

        return result;
}
```

```
/*
read_data_to_file() reads data from a file into a buffer.
It returns whether it was successful.
*/

BOOL read_data_from_file(PVOID dataBuf, HANDLE data_file_handle, ULONG bytes_to_read)
{
        DWORD bytes_read;
        BOOL result;

// Read the data from the file into a buffer.

        result = ReadFile(data_file_handle, dataBuf, bytes_to_read,
                &bytes_read, NULL);

// Check to make sure whether all the bytes asked for were read.

        if (bytes_read != bytes_to_read)
        {
                return FALSE;
        }
        return TRUE;
}
/*
get_device_memory_length() returns the size of the ULTRAD device memory in bytes.
*/

DWORD get_device_memory_length(IN HANDLE fileHandle)
{
        BOOL    result;
        DWORD device_memory_length, bytes_returned;


        result = DeviceIoControl( fileHandle, IOCTL_ULTRAD_GET_DEVICE_LENGTH, NULL,
                        0, &device_memory_length, sizeof(DWORD), &bytes_returned, NULL);

        return bytes_returned;

}
```

Modulo SentinelMain.c

```
#include "SentinelMain.h"
#include "FFT.h"
#include "UltraAD.h"

// Global Variables
DWORD                   *dataBuf;                       // A/D data buffer address

int                             ScreenMaxX,
                                ScreenMaxY;                     // max window dimensions

#ifdef TEXASFFT

        INT64                   g[2*MAX_CHANNELS];
        INT64                   G[2*MAX_CHANNELS];
        short                   W[2*MAX_CHANNELS]; // sin/cos tables
        short                   A[2*MAX_CHANNELS];
        short                   B[2*MAX_CHANNELS];
        unsigned short  IIndex[MAX_CHANNELS];
        unsigned short  JIndex[MAX_CHANNELS];
        int                             R4IndexCount;

#else

        int ip[NMAXSQRT + 2];
        float *g,*W;
```

```
#endif

int                     SpectrumCalib[MAX_CHANNELS];
double                  Spectrum[MAX_FFTS][MAX_CHANNELS];
int                     SpectrumData[MAX_CHANNELS];

int                     SpectrumMedie[MAX_FFTS][MAX_CHANNELS];   // per wrt file

BOOL                    SingleMode = TRUE;

short                   SampleRate          =   20,                 //
SAMPLE_RATE (MHz)
                        NumChannels         =   1024,               //
NUM_CHANNELS
                        SelNumChannels  =   0,
                        SelNumChsOld    =   SelNumChannels,
                        NumFfts             =   256,                //
FFT bank size (512 FFT calcolate da 2048 punti reali)
                        NumMedie            =   1;                  //
come mediare le FFT del bank size

short
                        SaveSampleRate,                             // per
riprendere i valori
                        SaveNumFfts,                                //
originari prima della schedula
                        SaveNumChannels;

double                  SpectrumMaxY,
                        SpectrumStepY;

int                     ZeroScale = ORG_Y+LEN_Y;

// MODE_SERENDIP
int                     ThresholdFactor =   7;
short                   SelThreshold    =   ThresholdFactor-1;
int                     SerendipLine    =   0;
char                    Map[MAX_CHANNELS];
int                     MapHeight = LEN_Y/8-6;
short                   SerendipCycle = 0;
BOOL                    SerendipFixedBandView = FALSE;
COLORREF                SerendipColor[16] =
                                        {
        MY_BLACK,MY_GRAY,MY_BROWN,MY_REDMAGENTA,MY_MAGENT,MY_MAGENTBLUE,  \

        MY_BLUE,MY_BLUECYAN,MY_CYAN,MY_CYANGREEN,MY_GREEN,MY_GREENYELLOW,       \
                                        MY_YELLOW,MY_ORANGE,MY_RED,MY_WHITE  };
int                     SerendipLevel[16] =
                                        {       1, 4, 16, 64, 256, 1024, 4096, 16384, 65536,
262144, 1048576, \
                                        4194304, 16777216, 67108864, 268435456,
1073741824 };

BOOL                    ScaleLog = FALSE;
//BOOL                  Autoscale = FALSE;
BOOL                    TurnComplete = FALSE;

// Save Files
char                    InitFileName[] = "Sentinel.ini";
char                    CurrWorkDir[MAX_PATH];
char                    strSaveDrive[3];
char                    DataType = TYPE_NORMAL;
BOOL                    SaveToDisk = FALSE;

// Timing
DWORD                   TickStart, TickEnd;
int                     TransferTime;

// Calibration
```

```
BOOL            CalibrationCycle  =  FALSE;
BOOL            ToggleCalibView   =  FALSE;
BOOL            SentinelCalibrated =  FALSE;

enum            t_SpectrumMode
                    { MODE_SPECTRUM, MODE_NORMALIZED, MODE_SERENDIP };
t_SpectrumMode  SpectrumMode = MODE_SPECTRUM;

// Frequencies
char            BandsSection[3][14] = { "Sweep Bands A", "Sweep Bands B", "Sweep bands C" };
char            BeginLabel[5][7] = { "Begin1", "Begin2", "Begin3", "Begin4", "Begin5" };
char            EndLabel[5][5] = { "End1", "End2", "End3", "End4", "End5" };
char            Label[3][21];
short           BandsSelected;
int                 BandFixed = FREQ_MIN;
short           SelBandFixed = 0;

BOOL            FreqWidth[FREQ_MAX/10];
short           BeginFreq[3][5];
short           EndFreq[3][5];
short           FreqOffset = FREQ_MIN;
short           FreqStep=10;

short           Direction = 0;
BOOL            DirEnable[8];

short           SpectrumPalette[5][3];
short           Oscilloscope; // modalità: Oscilloscopio o
                                                    // Spettrometro
short           PaintRainbowSpectrum; // modalità di visualizz.
                                                    // dello spettro (classica o
arcobaleno)

sched_t         SchedulingMap[1440]; // un record per ogni
                                                    // minuto della giornata
short           FftC_10dB_Tarature[5]; // la taratura logaritmica di 10 db
                                                    // a tacca cambia ad ogni risoluzione
long            SpectrumMaxYY[5];
short           SpectrumTopOffset[5];

short           SintPresent=TRUE;

// Global Handles
HINSTANCE       g_hInst;
HWND            g_hWnd, g_hDlg0, g_hDlg1, g_hDlg2, g_hDlg3, g_hDlg6;
HDC                 MemMaskDC, MemDirDC;
HICON           hIconSavetodisk0, hIconSavetodisk1;
HBRUSH
    g_hBrushBlack,g_hBrushGreen,g_hBrushMagent,g_hBrushCyan,g_hBrushRed,g_hBrushLightGray,g_hBrushGray,g_hBr
ushDarkGray;
HPEN            g_spectrum,g_hPenBlack, g_hPenRed, g_hPenBlue, g_hPenGreen, g_hPenCyan,
g_hPenYellow, g_hPenGray, g_hPenLightGray, g_hPenBrown;
RECT            AllRect,SpectrumRect,InfoLeftRect,SweepingBarRect,DirectionRect;
TEXTMETRIC      tmCourier;

// External var definitions
extern  "C"     HANDLE      ultrad_board_handle;
extern  "C"     int         giBoardNumber;

// Function & procedures prototypes
BOOL            InitApplication(HANDLE);
BOOL            InitInstance(HANDLE, int);
long APIENTRY       MainWndProc(HWND, UINT, UINT, LONG);
BOOL APIENTRY       DialogIntro(HWND,UINT,WPARAM,LPARAM);
BOOL APIENTRY       DialogFunc1(HWND,UINT,WPARAM,LPARAM);
BOOL APIENTRY       DialogFunc2(HWND,UINT,WPARAM,LPARAM);

int                     UltraD_ISR(void);
```

```
//void                          AutoscaleXY(int *data);
int                             MinValue(int *data);

void                            SaveDataFiles(char *year,char *month,char *day,char *timeline);
BOOL                            SaveProfileData(short bands);

void                            Calibration(void);
int                             TargetStart(void);
void                            TargetStop(void);
BOOL                            CheckFiles();
BOOL                            InitIEEE488();
void                            Sintonize(void);
void                            FftInit(int num_channels);

void                            PaintSpectrum();
void                            PaintExtra(void);
void                            MapStart(void);
void                            ReadScheduling(void);
void                            InitLogTarature(void);
double                          log2(double x);
double                          mymin(double x,double y);

//*************************************************************************************
// WinMain()
//*************************************************************************************
//
// FUNCTION:    WinMain(HANDLE, HANDLE, LPSTR, int)
//
// PURPOSE:     Entry point for the application.
// COMMENTS:    this function initializes the application and processes
//                              the     message loop.
int APIENTRY WinMain (  HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                LPSTR lpCmdLine, int nCmdShow )
{
        MSG msg;
        // Check if another application already exists
        CreateMutex(NULL,TRUE,"DUMMY");
        if ( GetLastError() == ERROR_ALREADY_EXISTS )
        {
                MessageBox(HWND_DESKTOP,"Sentinel 1 already started!","ERROR",MB_ICONSTOP | MB_OK);
                return FALSE;
        }

        // Check Target Programs & Configuration Files
        if ( !CheckFiles() )
                return FALSE;

        InitIEEE488();

        FftInit(NumChannels);

        #ifndef TEXASFFT
                g=(float *)malloc(sizeof(float)*2*MAX_CHANNELS);
                W=(float *)malloc(sizeof(float)*2*(long)(((float)MAX_CHANNELS * 5.0) / 4.0));
        #endif

        // scheduling file read
        ReadScheduling();
        PaintRainbowSpectrum    =       FALSE;
        Oscilloscope                    =       FALSE;
        SpectrumMaxY                    =       SpectrumMaxYY[0];

        InitLogTarature();

        // Get Screen Coordinates
        ScreenMaxX = GetSystemMetrics(SM_CXSCREEN);
        ScreenMaxY = GetSystemMetrics(SM_CYSCREEN);

        if (!InitApplication(hInstance))
```

```
                return FALSE;

        // Create the main window.
        if (!InitInstance(hInstance, nCmdShow))
                return FALSE;

        // Starting ULTRAD
        if (TargetStart()==-1)
        {
                MessageBox(g_hWnd,"Can't access A/D converter","ERROR",MB_ICONSTOP | MB_OK);
                exit(-1);
        }

        // Acquire and dispatch messages until a WM_QUIT message is received.
    while ( GetMessage(&msg,NULL,0,0) )
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);    // restituisce il controllo a Windows
        }

        return (msg.wParam);                    // returns the value from PostQuitMessage.
}

// FUNCTION:    InitApplication(HANDLE)
//
// PURPOSE:     Initializes window data and registers window class
//
// COMMENTS:
//
//     In this function, we initialize a window class by filling out a data
//     structure of type WNDCLASS and calling the Windows RegisterClass()
//     function.
BOOL InitApplication(HANDLE hInstance)
{
  WNDCLASS  wc;

  // Register the window class for my window.
        wc.style                =0;                                     */
        wc.lpfnWndProc          = (WNDPROC)MainWndProc;         // Class style
        wc.cbClsExtra           = 0;                            // Window procedure for this class.
wc.cbWndExtra           = 0;                                    // No per-class extra data.
wc.hInstance            = (HINSTANCE)hInstance;                 // No per-window extra data.
        wc.hIcon                = LoadIcon((HINSTANCE)hInstance,"MY_ICON");     // Application that owns the class.
        static HCURSOR g_hCurNorm       = LoadCursor(NULL,IDC_ARROW); // My Icon
        static HCURSOR g_hCurWE                 = LoadCursor(NULL,IDC_SIZEWE); // Cursor style
        static HCURSOR g_hCurNS         = LoadCursor(NULL,IDC_SIZENS);
        wc.hCursor                      = g_hCurNorm;
        wc.hbrBackground = g_hBrushLightGray;
        wc.lpszMenuName         = NULL;                                 // no menu
        wc.lpszClassName = "MainWin";                           // Name of window class

        return (RegisterClass(&wc));
}

// FUNCTION: InitInstance(HANDLE, int)
//
// PURPOSE: Saves instance handle and creates main window
//
// COMMENTS:
//
//     In this function, we save the instance handle in a global variable and
//     create and display the main program window.
BOOL InitInstance( HANDLE hInstance, int nCmdShow)
{
        // Save off the handle to the current instance.
        g_hInst = (HINSTANCE)hInstance;
        // Create a main window for this application instance.
        g_hWnd = CreateWindow(
```

```
                "MainWin",                                              // Name of window class.
                "Sentinel 2 - Front-End Side *** National Research Council, Radioastronomy Institute ****",     // Window title.
                WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN,// Window style.
                CW_USEDEFAULT,CW_USEDEFAULT,            // Window position
                CW_USEDEFAULT,CW_USEDEFAULT,            // Window diensions
                NULL,                                                   // Overlapped windows have no
parent.
                NULL,                                                   // Use the window class menu.
                (HINSTANCE)hInstance,                                   // This instance owns this
window.
                NULL                                                    // Pointer not needed.
        );

        // If window could not be created, return "failure".
        if (!g_hWnd)
    return FALSE;

        // Make the window visible; update its client area; and return "success".
        ShowWindow(g_hWnd,SW_HIDE);
        // Draw Intro Dialog
        #ifdef INTRO_IMAGE
                g_hDlg0 = CreateDialog((HINSTANCE)hInstance,"INTRO",HWND_DESKTOP,(DLGPROC)DialogIntro);
        #else
                ShowWindow(g_hWnd,SW_SHOWMAXIMIZED);
        #endif
        // Transparent dialog window
        g_hDlg1 = CreateDialog((HINSTANCE)hInstance,"DIALOG1",g_hWnd,(DLGPROC)DialogFunc1);
        InvalidateRect(g_hDlg1,NULL,0);
        InvalidateRect(g_hWnd,NULL,0);

        return (TRUE);
}

// FUNCTION: MainWndProc(hWnd, unsigned, WORD, LONG)
//
// PURPOSE:  Processes messages for the main window.
LONG APIENTRY MainWndProc( HWND hWnd, UINT message, UINT wParam, LONG lParam)
{
        HDC     hdc;
        LOGBRUSH lplb;
        int kx;

        switch(message)
        {

                case WM_CREATE:

                        // Create a compatible imagine into memory device
                        hdc = GetDC(hWnd);
                        MemMaskDC = CreateCompatibleDC(hdc);
                        static HBITMAP g_hBitMask = CreateCompatibleBitmap(hdc,ScreenMaxX,ScreenMaxY);
                        SelectObject(MemMaskDC,g_hBitMask);
                        MemDirDC = CreateCompatibleDC(hdc);
                        static HBITMAP hBitDirection = LoadBitmap(g_hInst,"DIRECTION");
                        SelectObject(MemDirDC,hBitDirection);
                        ReleaseDC(hWnd,hdc);

                        // Create Brushes
                        lplb.lbStyle = BS_SOLID;
                        lplb.lbColor = MY_CYAN;
                        g_hBrushCyan            = CreateBrushIndirect(&lplb);
                        lplb.lbColor = MY_RED;
                        g_hBrushRed             = CreateBrushIndirect(&lplb);
                        lplb.lbColor = MY_GREEN;
                        g_hBrushGreen           = CreateBrushIndirect(&lplb);
                        lplb.lbColor = MY_MAGENT;

                        g_hBrushMagent  = CreateBrushIndirect(&lplb);
                        g_hBrushBlack           = (HBRUSH)GetStockObject(BLACK_BRUSH);
```

```
                        g_hBrushDarkGray        = (HBRUSH)GetStockObject(DKGRAY_BRUSH);
                        g_hBrushGray            = (HBRUSH)GetStockObject(GRAY_BRUSH);
                        g_hBrushLightGray       = (HBRUSH)GetStockObject(LTGRAY_BRUSH);

                        // Create Pens
                        g_hPenBlack                     = CreatePen(PS_SOLID,1,MY_BLACK);
                        g_hPenBlue                      = CreatePen(PS_SOLID,2,MY_BLUE);
                        g_hPenRed                       = CreatePen(PS_SOLID,1,MY_RED);
                        g_hPenGreen                     = CreatePen(PS_SOLID,1,MY_GREEN);
                        g_hPenCyan                      = CreatePen(PS_SOLID,1,MY_CYAN);
                        g_hPenYellow            = CreatePen(PS_SOLID,1,MY_YELLOW);
                        g_hPenGray                      = CreatePen(PS_SOLID,1,MY_GRAY);
                        g_hPenLightGray         = CreatePen(PS_SOLID,1,MY_LIGHTGRAY);
                        g_hPenBrown                     = CreatePen(PS_SOLID,1,MY_BROWN);

                        // Create Fonts
                        static HFONT hFontCourier = CreateFont(14,0,0,0,FW_NORMAL,0,0,0, \
        ANSI_CHARSET,OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS, \
                                                                DEFAULT_QUALITY,DEFAULT_PITCH |
    FF_DONTCARE,"Courier New");

                        // Set Text Attributes
                        SetBkMode(MemMaskDC,TRANSPARENT);
                        SelectObject(MemMaskDC,hFontCourier);
                        GetTextMetrics(MemMaskDC,&tmCourier);

                        // Define Visualization Area
                        AllRect.left = 0;
                        AllRect.top = ORG_Y;
                        AllRect.right = ScreenMaxX;
                        AllRect.bottom = ScreenMaxY;
                        SpectrumRect.left = ORG_X;
                        SpectrumRect.top = ORG_Y;
                        SpectrumRect.right = ORG_X+1+LEN_X;
                        SpectrumRect.bottom = ORG_Y+1+LEN_Y;
                        InfoLeftRect.left = 0;
                        InfoLeftRect.top = ORG_Y+11;
                        InfoLeftRect.right = ORG_X-6;
                        InfoLeftRect.bottom = 1+ORG_Y+10+16*11;
                        SweepingBarRect.left = BAR_X-1;
                        SweepingBarRect.top = BAR_Y;
                        SweepingBarRect.right = 1+BAR_X+BAR_LEN_X;
                        SweepingBarRect.bottom = 1+BAR_Y+BAR_LEN_Y;
                        DirectionRect.left = DIR_X;
                        DirectionRect.top = DIR_Y;
                        DirectionRect.right = DIR_X+1+63;
                        DirectionRect.bottom = DIR_Y+1+71;
                        FillRect(MemMaskDC,&AllRect,g_hBrushLightGray);
                        for ( kx=0; kx<8; kx++ )
                                DirEnable[kx] = TRUE;
                        break;

                case WM_TIMER:
                        if(UltraD_ISR() == -1)
                        {
                                MessageBox(g_hWnd,"Timer Interrupt- A/D converter error","ERROR",MB_ICONSTOP |
MB_OK);
                                exit(-1);
                        }
                        break;
                case WM_PAINT: // UPDATING SCREEN
                        PAINTSTRUCT    paintstruct;

                        // Copy memory image onto screen
                        if ( hdc = BeginPaint(hWnd,&paintstruct) )// get device context
                        {
                                BitBlt(hdc,0,ORG_Y,ScreenMaxX+1,ScreenMaxY-
ORG_Y,MemMaskDC,0,ORG_Y,SRCCOPY);
```

```
                        EndPaint(hWnd,&paintstruct);                    // release device context
                    }
                    break;

                case WM_CLOSE:// TERMINATE PROGRAM: window being destroyed
                    if ( MessageBox(g_hWnd,"Are you
sure?","EXIT",MB_ICONEXCLAMATION|MB_YESNO|MB_TOPMOST) == IDYES )
                    {
                        DestroyWindow(g_hDlg1);

                        DeleteObject(g_hPenBlack);      DeleteObject(g_hPenRed);
                        DeleteObject(g_hPenGreen);      DeleteObject(g_hPenCyan);
                        DeleteObject(g_hPenYellow);         DeleteObject(g_hPenGray);
                        DeleteObject(g_hPenLightGray);  DeleteObject(g_hPenBrown);
                        DeleteObject(g_hBrushBlack);    DeleteObject(g_hBrushGreen);
                        DeleteObject(g_hBrushMagent);   DeleteObject(g_hBrushCyan);
                        DeleteObject(g_hBrushRed);      DeleteObject(g_hBrushLightGray);
                        DeleteObject(g_hBrushGray);         DeleteObject(g_hBrushDarkGray);
                        DeleteDC((HDC)hFontCourier);    DeleteObject(g_hPenBlue);
                        DeleteObject(g_hBitMask);
                        DeleteObject(hBitDirection);
                        DeleteDC(MemMaskDC);
                        DeleteDC(MemDirDC);
                        TargetStop();
                        KillTimer(g_hWnd,1);
                        PostQuitMessage(0);
                    }
                    break;

                default:    // let Win95 elaborate any message not specified in switch
                    return ( DefWindowProc(hWnd,message,wParam,lParam) );
        }

        return (0);
}

//*****************************************************************
// Dialog window: INTRO
//*****************************************************************
BOOL APIENTRY DialogIntro (HWND hdwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
        switch(message)
        {
                case WM_COMMAND:
                        switch(LOWORD(wParam))
                        {
                                case IDOK:
                                        ShowWindow(g_hWnd,SW_SHOWMAXIMIZED);
                                        DestroyWindow(g_hDlg0);
                                        break;
                        }
        }

        return 0;
}

//*****************************************************************
// Dialog window: OPTIONS
//*****************************************************************
BOOL APIENTRY DialogFunc1 (HWND hdwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
        char str[80];
        int     kx,i;
        switch(message)
        {

                case WM_INITDIALOG:
                        // Initialize BUTTONS
                        static HBITMAP hBitOn  = LoadBitmap(g_hInst,"ON");
```

```
                        static HBITMAP hBitOff = LoadBitmap(g_hInst,"OFF");

                        static HBITMAP hBitSaveToDisk1 = LoadBitmap(g_hInst,"SAVE_TO_DISK1");
                        static HBITMAP hBitSaveToDisk0 = LoadBitmap(g_hInst,"SAVE_TO_DISK0");

                        if ( SaveToDisk )
    SendDlgItemMessage(hdwnd,ID_ACQUIRE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hB
itSaveToDisk1);
                        else

    SendDlgItemMessage(hdwnd,ID_ACQUIRE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hB
itSaveToDisk0);

                        static HBITMAP hBitCalibration = LoadBitmap(g_hInst,"CALIBRATION");
    SendDlgItemMessage(hdwnd,ID_CALIBRATION,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HAND
LE)hBitCalibration);
                        static HBITMAP hBitExit = LoadBitmap(g_hInst,"EXIT");

                        static HBITMAP hBitSpectrum1 = LoadBitmap(g_hInst,"SPECTRUM1");
                        static HBITMAP hBitSpectrum0 = LoadBitmap(g_hInst,"SPECTRUM0");
                        if ( SpectrumMode == MODE_SPECTRUM )
    SendDlgItemMessage(hdwnd,IDC_SPECTRUM_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(H
ANDLE)hBitSpectrum1);
                        else
    SendDlgItemMessage(hdwnd,IDC_SPECTRUM_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(H
ANDLE)hBitSpectrum0);
                        static HBITMAP hBitNormalized1 = LoadBitmap(g_hInst,"NORMALIZED1");
                        static HBITMAP hBitNormalized0 = LoadBitmap(g_hInst,"NORMALIZED0");

                        if ( SpectrumMode == MODE_NORMALIZED )
    SendDlgItemMessage(hdwnd,IDC_NORMALIZED_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM
)(HANDLE)hBitNormalized1);
                        else

    SendDlgItemMessage(hdwnd,IDC_NORMALIZED_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM
)(HANDLE)hBitNormalized0);

                        static HBITMAP hBitSerendip1 = LoadBitmap(g_hInst,"SERENDIP1");
                        static HBITMAP hBitSerendip0 = LoadBitmap(g_hInst,"SERENDIP0");

                        if ( SpectrumMode == MODE_SERENDIP )
    SendDlgItemMessage(hdwnd,IDC_SERENDIP_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(H
ANDLE)hBitSerendip1);
                        else

    SendDlgItemMessage(hdwnd,IDC_SERENDIP_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(H
ANDLE)hBitSerendip0);
                        EnableWindow(GetDlgItem(hdwnd,ID_COMBO_THRESHOLD),(SpectrumMode ==
MODE_SERENDIP));

                        static HBITMAP hBitUp   = LoadBitmap(g_hInst,"UP");
                        static HBITMAP hBitDown = LoadBitmap(g_hInst,"DOWN");
                        //static HBITMAP hBitAutoscale = LoadBitmap(g_hInst,"AUTOSCALE");
                        static HBITMAP hBitLinear    = LoadBitmap(g_hInst,"LINEAR");
                        static HBITMAP hBitLogaritmic = LoadBitmap(g_hInst,"LOGARITMIC");
                        static HBITMAP hBitOscill    = LoadBitmap(g_hInst,"OSCILL");
                        static HBITMAP hBitSpectrom          = LoadBitmap(g_hInst,"SPECTROM");

                        static HBITMAP hBitSingle = LoadBitmap(g_hInst,"SINGLE");
                        static HBITMAP hBitMulti  = LoadBitmap(g_hInst,"MULTI");
                        static HBITMAP hBitRainbowSpectrum = LoadBitmap(g_hInst,"RAINBOWSPECTRUM");
                        static HBITMAP hBitNormalSpectrum = LoadBitmap(g_hInst,"NORMALSPECTRUM");
                        if (PaintRainbowSpectrum)
```

```
        SendDlgItemMessage(hdwnd,ID_RAINBOW_SPECTRUM,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM
)(HANDLE)hBitNormalSpectrum);
                        else

        SendDlgItemMessage(hdwnd,ID_RAINBOW_SPECTRUM,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM
)(HANDLE)hBitRainbowSpectrum);

                        SendDlgItemMessage(hdwnd,IDC_SWEEP_BANDS_1,BM_SETCHECK,TRUE,0);


        //SendDlgItemMessage(hdwnd,ID_Y_ZOOM_PLUS,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HAN
DLE)hBitUp);

        //SendDlgItemMessage(hdwnd,ID_Y_ZOOM_MINUS,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HA
NDLE)hBitDown);
        //SendDlgItemMessage(hdwnd,ID_AUTOSCALE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDL
E)hBitAutoscale);


        SendDlgItemMessage(hdwnd,ID_LIN_LOG,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)(Sc
aleLog?hBitLogaritmic:hBitLinear));
        SendDlgItemMessage(hdwnd,ID_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)(Oscill
oscope?hBitSpectrom:hBitOscill));
                        #ifdef SINGLE_MULTI_SUPPORT
                                SetDlgItemText(hdwnd,IDC_SINGLE_MULTI,SingleMode?"Set Multidirect.":"Set
Omnidirect.");
                        #else
                                EnableWindow(GetDlgItem(hdwnd,IDC_SINGLE_MULTI),FALSE);
                        #endif

                        _chdrive(2); // da mettere nell'ini
                        chdir("\\");
                        mkdir("Sentinel_data");
                        chdir("Sentinel_data");
                        for (i=0;i<5;i++)

        SendDlgItemMessage(hdwnd,ID_COMBO_CHS,CB_ADDSTRING,0,(LPARAM)itoa(256*(int)pow(2.0,(double)i),str,10)
);

                        SendDlgItemMessage(hdwnd,ID_COMBO_CHS,CB_SETCURSEL,2,0);

                        SelNumChannels = (short)SendDlgItemMessage(hdwnd,ID_COMBO_CHS,CB_GETCURSEL,2,0L);
                        SendDlgItemMessage(hdwnd,ID_COMBO_CHS,CB_GETLBTEXT,2,(LPARAM)str);
                        NumChannels = atoi(str);

                        for (i=1;i<128;i*=2)
                        {
                                if (ULTRAD_MEMORY_SIZE/(2*i*NumChannels)<=MAX_FFTS)

        SendDlgItemMessage(hdwnd,ID_COMBO_BURST,CB_ADDSTRING,0,(LPARAM)itoa(ULTRAD_MEMORY_SIZE/(2
*i*NumChannels),str,10));
                        }
                        SendDlgItemMessage(hdwnd,ID_COMBO_BURST,CB_SETCURSEL,0,0);

        SendDlgItemMessage(hdwnd,ID_COMBO_CHS,CB_GETLBTEXT,SendDlgItemMessage(hdwnd,ID_COMBO_BURST,
CB_GETCURSEL,0,0),(LPARAM)str);
                        NumFfts=atoi(str);

                        SendDlgItemMessage(hdwnd,ID_COMBO_SAMPLE_RATE,CB_ADDSTRING,0,(LPARAM)"5");
                        SendDlgItemMessage(hdwnd,ID_COMBO_SAMPLE_RATE,CB_ADDSTRING,0,(LPARAM)"10");
                        SendDlgItemMessage(hdwnd,ID_COMBO_SAMPLE_RATE,CB_ADDSTRING,0,(LPARAM)"15");
                        SendDlgItemMessage(hdwnd,ID_COMBO_SAMPLE_RATE,CB_ADDSTRING,0,(LPARAM)"20");
                        SendDlgItemMessage(hdwnd,ID_COMBO_SAMPLE_RATE,CB_ADDSTRING,0,(LPARAM)"40");
```

```
                        SendDlgItemMessage(hdwnd,ID_COMBO_SAMPLE_RATE,CB_SETCURSEL,3,0);

                        FreqStep=10;

                        for (i=1;i<31;i++)

        SendDlgItemMessage(hdwnd,ID_COMBO_THRESHOLD,CB_ADDSTRING,0,(LPARAM)itoa(i,str,10));
                        SendDlgItemMessage(hdwnd,ID_COMBO_THRESHOLD,CB_SETCURSEL,SelThreshold,0);

                        SetDlgItemText(hdwnd,IDC_SWEEP_BANDS_1,Label[0]);
                        SetDlgItemText(hdwnd,IDC_SWEEP_BANDS_2,Label[1]);
                        SetDlgItemText(hdwnd,IDC_SWEEP_BANDS_3,Label[2]);

                        SetDlgItemText(hdwnd,IDC_STARTFREQ,"");
                        SetDlgItemText(hdwnd,IDC_ENDFREQ,"");
                        SetDlgItemText(hdwnd,IDC_DATE,"");
                        SetDlgItemText(hdwnd,IDC_TIME,"");
                        SetDlgItemText(hdwnd,IDC_TIMING,"");
                        SetDlgItemText(hdwnd,IDC_MINFREQ,"50 MHz");
                        SetDlgItemText(hdwnd,IDC_MAXFREQ,"2,35 GHz");

                        for (i=0;i<9;i++)
                                SendDlgItemMessage(hdwnd,IDC_DIR0+i,BM_SETCHECK,TRUE,0);

                        break;
                case WM_COMMAND:
                        switch( LOWORD(wParam) )
                        {
                                case ID_ACQUIRE:
                                        SaveToDisk ^= 1;
                                        if ( SaveToDisk )

        SendDlgItemMessage(hdwnd,ID_ACQUIRE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hB
itSaveToDisk1);
                                        else

        SendDlgItemMessage(hdwnd,ID_ACQUIRE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hB
itSaveToDisk0);
                                        break;
                                case ID_RAINBOW_SPECTRUM:
                                        PaintRainbowSpectrum ^= 1;
                                        if (PaintRainbowSpectrum)

        SendDlgItemMessage(hdwnd,ID_RAINBOW_SPECTRUM,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM
)(HANDLE)hBitNormalSpectrum);
                                        else

        SendDlgItemMessage(hdwnd,ID_RAINBOW_SPECTRUM,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM
)(HANDLE)hBitRainbowSpectrum);
                                        break;
                                case ID_COMBO_CHS:
                                        SelNumChsOld = SelNumChannels;
                                        SelNumChannels =
(short)SendDlgItemMessage(hdwnd,ID_COMBO_CHS,CB_GETCURSEL,0,0L);

        SendDlgItemMessage(hdwnd,ID_COMBO_CHS,CB_GETLBTEXT,SelNumChannels,(LPARAM)str);
                                        NumChannels = atoi(str);

                                        SendDlgItemMessage(hdwnd,ID_COMBO_BURST,CB_RESETCONTENT,0,0);
                                        for (i=1;i<128;i*=2)
                                                if (ULTRAD_MEMORY_SIZE/(2*i*NumChannels)<=MAX_FFTS)

        SendDlgItemMessage(hdwnd,ID_COMBO_BURST,CB_ADDSTRING,0,(LPARAM)itoa(ULTRAD_MEMORY_SIZE/(2
*i*NumChannels),str,10));

                                        SendDlgItemMessage(hdwnd,ID_COMBO_BURST,CB_SETCURSEL,0,0);

        SendDlgItemMessage(hdwnd,ID_COMBO_BURST,CB_GETLBTEXT,SendDlgItemMessage(hdwnd,ID_COMBO_BUR
ST,CB_GETCURSEL,0,0),(LPARAM)str);
```

```
                                        NumFfts = atoi(str);
                                        // info waiting
                                        if(SelNumChsOld != SelNumChannels)
                                        {
                                                g_hDlg3 = CreateDialog(g_hInst,"DIALOG3",g_hDlg1,NULL);
                                                EnableWindow(g_hWnd,FALSE);
                                                //Autoscale = TRUE;
                                                FftInit(NumChannels);

                                                EnableWindow(g_hWnd,TRUE);
                                                DestroyWindow(g_hDlg3);
                                        }
                                        break;
                                case ID_COMBO_BURST:

        SendDlgItemMessage(hdwnd,ID_COMBO_BURST,CB_GETLBTEXT,SendDlgItemMessage(hdwnd,ID_COMBO_BUR
ST,CB_GETCURSEL,0,0L),(LPARAM)str);
                                        NumFfts=atoi(str);
                                        break;
                                case ID_COMBO_SAMPLE_RATE:

        SendDlgItemMessage(hdwnd,ID_COMBO_SAMPLE_RATE,CB_GETLBTEXT,(short)SendDlgItemMessage(hdwnd,ID_
COMBO_SAMPLE_RATE,CB_GETCURSEL,0,0L),(LPARAM)str);
                                        SampleRate = atoi(str);
                                        FreqStep=SampleRate/2;
                                        break;
                                case ID_COMBO_THRESHOLD:
                                        SelThreshold =
(short)SendDlgItemMessage(hdwnd,ID_COMBO_THRESHOLD,CB_GETCURSEL,0,0L);

        SendDlgItemMessage(hdwnd,ID_COMBO_THRESHOLD,CB_GETLBTEXT,SelThreshold,(LPARAM)str);
                                        ThresholdFactor = (int)atoi(str);
                                        break;
                                case IDC_SPECTRUM_MODE:
                                        SpectrumMode = MODE_SPECTRUM;

        SendDlgItemMessage(hdwnd,IDC_SPECTRUM_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(H
ANDLE)hBitSpectrum1);

        SendDlgItemMessage(hdwnd,IDC_NORMALIZED_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM
)(HANDLE)hBitNormalized0);

        SendDlgItemMessage(hdwnd,IDC_SERENDIP_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(H
ANDLE)hBitSerendip0);
                                        EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_THRESHOLD),FALSE);
                                        EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_SAMPLE_RATE),TRUE);
                                        MapStart();
                                        break;
                                case IDC_NORMALIZED_MODE:
                                        SpectrumMode = MODE_NORMALIZED;

        SendDlgItemMessage(hdwnd,IDC_SPECTRUM_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(H
ANDLE)hBitSpectrum0);

        SendDlgItemMessage(hdwnd,IDC_NORMALIZED_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM
)(HANDLE)hBitNormalized1);

        SendDlgItemMessage(hdwnd,IDC_SERENDIP_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(H
ANDLE)hBitSerendip0);
                                        EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_THRESHOLD),FALSE);
                                        EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_SAMPLE_RATE),TRUE);
                                        MapStart();
                                        break;
                                        break;
                                case IDC_SERENDIP_MODE:
                                        SpectrumMode = MODE_SERENDIP;
```

```
SendDlgItemMessage(hdwnd,IDC_SPECTRUM_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hBitSpectrum0);

SendDlgItemMessage(hdwnd,IDC_NORMALIZED_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hBitNormalized0);

SendDlgItemMessage(hdwnd,IDC_SERENDIP_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hBitSerendip1);
                                        EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_THRESHOLD),TRUE);
                                        EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_SAMPLE_RATE),FALSE);
                                        MapStart();
                                        break;
                                case IDC_DIR0:
                                        SendDlgItemMessage(hdwnd,IDC_DIRALL,BM_SETCHECK,FALSE,0);
                                        DirEnable[0] = (IsDlgButtonChecked(hdwnd,IDC_DIR0) == BST_CHECKED );
                                        MapStart();
                                        break;
                                case IDC_DIR1:
                                        SendDlgItemMessage(hdwnd,IDC_DIRALL,BM_SETCHECK,FALSE,0);
                                        DirEnable[1] = (IsDlgButtonChecked(hdwnd,IDC_DIR1) == BST_CHECKED );
                                        MapStart();
                                        break;
                                case IDC_DIR2:
                                        SendDlgItemMessage(hdwnd,IDC_DIRALL,BM_SETCHECK,FALSE,0);
                                        DirEnable[2] = (IsDlgButtonChecked(hdwnd,IDC_DIR2) == BST_CHECKED );
                                        MapStart();
                                        break;
                                case IDC_DIR3:
                                        SendDlgItemMessage(hdwnd,IDC_DIRALL,BM_SETCHECK,FALSE,0);
                                        DirEnable[3] = (IsDlgButtonChecked(hdwnd,IDC_DIR3) == BST_CHECKED );
                                        MapStart();
                                        break;
                                case IDC_DIR4:
                                        SendDlgItemMessage(hdwnd,IDC_DIRALL,BM_SETCHECK,FALSE,0);
                                        DirEnable[4] = (IsDlgButtonChecked(hdwnd,IDC_DIR4) == BST_CHECKED );
                                        MapStart();
                                        break;
                                case IDC_DIR5:
                                        SendDlgItemMessage(hdwnd,IDC_DIRALL,BM_SETCHECK,FALSE,0);
                                        DirEnable[5] = (IsDlgButtonChecked(hdwnd,IDC_DIR5) == BST_CHECKED );
                                        MapStart();
                                        break;
                                case IDC_DIR6:
                                        SendDlgItemMessage(hdwnd,IDC_DIRALL,BM_SETCHECK,FALSE,0);
                                        DirEnable[6] = (IsDlgButtonChecked(hdwnd,IDC_DIR6) == BST_CHECKED );
                                        MapStart();
                                        break;
                                case IDC_DIR7:
                                        SendDlgItemMessage(hdwnd,IDC_DIRALL,BM_SETCHECK,FALSE,0);
                                        DirEnable[7] = (IsDlgButtonChecked(hdwnd,IDC_DIR7) == BST_CHECKED );
                                        MapStart();
                                        break;
                                case IDC_DIRALL:
                                        if ( IsDlgButtonChecked(hdwnd,IDC_DIRALL) )
                                                for ( kx=0; kx<8; kx++)
                                                {
                                                        DirEnable[kx] = TRUE;
SendDlgItemMessage(hdwnd,IDC_DIR0+kx,BM_SETCHECK,TRUE,0);
                                                }
                                        else
                                                for ( kx=0; kx<8; kx++)
                                                {
                                                        DirEnable[kx] = FALSE;
SendDlgItemMessage(hdwnd,IDC_DIR0+kx,BM_SETCHECK,FALSE,0);
                                                }
                                        MapStart();
```

```
                                        break;
                                case IDC_SWEEP_BANDS_EDIT:
                                        if ( g_hDlg2 == NULL )
                                        {
                                                g_hDlg2 =
CreateDialog(g_hInst,"DIALOG2",g_hDlg1,(DLGPROC)DialogFunc2);
                                                EnableWindow(g_hWnd,FALSE);
                                        }
                                        break;
                                case ID_CALIBRATION:
                                        TargetStop();
                                        if ( MessageBox(g_hWnd,"Disconnect any input
signal.\nWaiting...","Calibration",MB_OK) == IDOK )
                                        {
                                                for (int kx=0; kx<NumChannels; kx++)
                                                        SpectrumCalib[kx] = 0;
                                                CalibrationCycle = TRUE;
                                                if (TargetStart()==-1)
                                                {
                                                        MessageBox(g_hWnd,"Can't access A/D
converter","ERROR",MB_ICONSTOP | MB_OK);
                                                        exit(-1);
                                                }
                                        }
                                        break;
                                case ID_MODE:
                                        Oscilloscope ^= 1;
                                        if ( Oscilloscope )

        SendDlgItemMessage(hdwnd,ID_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hBitSp
ectrom);
                                        else

        SendDlgItemMessage(hdwnd,ID_MODE,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hBitO
scill);
                                        break;
                                case ID_LIN_LOG:
                                        ScaleLog ^= 1;
                                        //Autoscale = TRUE;
                                        if ( ScaleLog )

        SendDlgItemMessage(hdwnd,ID_LIN_LOG,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hBi
tLogaritmic);
                                        else

        SendDlgItemMessage(hdwnd,ID_LIN_LOG,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)(HANDLE)hBi
tLinear);
                                        break;
                                /*
                                case ID_AUTOSCALE:
                                        Autoscale = TRUE;
                                        break;
                                case ID_Y_ZOOM_MINUS:
                                        SpectrumMaxY += SpectrumStepY;
                                        if (SpectrumMaxY > SPECTRE_MAX_Y)
                                                SpectrumMaxY = SPECTRE_MAX_Y;
                                        break;
                                case ID_Y_ZOOM_PLUS:
                                        SpectrumMaxY -= SpectrumStepY;
                                        if (SpectrumMaxY < SPECTRE_MIN_Y)
                                                SpectrumMaxY = SPECTRE_MIN_Y;
                                        break;
                                */
                                case IDC_SINGLE_MULTI:
                                        #ifdef SINGLE_MULTI_SUPPORT
                                                SingleMode ^= 1;
                                                if ( SingleMode )
                                                {
                                                        Direction=8;
```

```
Multidirect.");                                         SetDlgItemText(hdwnd,IDC_SINGLE_MULTI,"Set

                                                        SetDlgItemText(hdwnd,IDC_DIRECTION,"");
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR0),FALSE);

                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR1),FALSE);
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR2),FALSE);

                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR3),FALSE);
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR4),FALSE);

                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR5),FALSE);
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR6),FALSE);

                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR7),FALSE);
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIRALL),FALSE);
                                                        MapStart();
                                                }
                                                else
                                                {

                                                        Direction=0;
                                                        SetDlgItemText(hdwnd,IDC_SINGLE_MULTI,"Set
Omnidirect.");

                                                        SetDlgItemText(hdwnd,IDC_DIRECTION,"Direction");
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR0),TRUE);

                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR1),TRUE);
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR2),TRUE);

                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR3),TRUE);
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR4),TRUE);

                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR5),TRUE);
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR6),TRUE);

                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIR7),TRUE);
                                                        EnableWindow(GetDlgItem(g_hDlg1,IDC_DIRALL),TRUE);
                                                        MapStart();
                                                }
                                        #endif
                                        break;
                                }
                                return (1);
                        }
                        return (0);
}

//*************************************************************
// Dialog window: SWEEP BANDS EDIT
//*************************************************************
BOOL APIENTRY DialogFunc2 (HWND hdwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
        int x,i;
        char str[20];
        switch(message)
        {
                case WM_INITDIALOG:

                        EnableWindow(GetDlgItem(hdwnd,ID_COMBO_FIXED),TRUE);
                        for (i=FREQ_MIN;i<FREQ_MAX;i+=10)

        SendDlgItemMessage(hdwnd,ID_COMBO_FIXED,CB_ADDSTRING,0,(LPARAM)itoa(i,str,10));
                        SendDlgItemMessage(hdwnd,ID_COMBO_FIXED,CB_SETCURSEL,0,0);

                        SetDlgItemText(hdwnd,IDC_SWEEP_BANDS_1,Label[0]);
                        SetDlgItemText(hdwnd,IDC_SWEEP_BANDS_2,Label[1]);
                        SetDlgItemText(hdwnd,IDC_SWEEP_BANDS_3,Label[2]);

                        if ( IsDlgButtonChecked(hdwnd,IDC_SWEEP_BANDS_1) )
```

```
                BandsSelected = 0;
        else if ( IsDlgButtonChecked(hdwnd,IDC_SWEEP_BANDS_2) )
                BandsSelected = 1;
        else if ( IsDlgButtonChecked(hdwnd,IDC_SWEEP_BANDS_3) )
                BandsSelected = 2;
        else
                BandsSelected = 0;

        break;
case WM_COMMAND:
        switch( LOWORD(wParam) )
        {
                case IDOK:

                        EnableWindow(g_hWnd,TRUE);
                        DestroyWindow(g_hDlg2);
                        g_hDlg2 = NULL;
                        PaintExtra();
                        break;
                case IDC_SWEEP_BANDS_1:

                        SerendipFixedBandView = FALSE;
                        for ( i=FREQ_MIN/FreqStep; i<FREQ_MAX/FreqStep; i++ )
                                FreqWidth[i] = FALSE;
                        for ( x=0; x<5; x++ )
                                for ( i=BeginFreq[0][x]/FreqStep; i<EndFreq[0][x]/FreqStep; i++ )
                                FreqWidth[i] = TRUE;
                        do
                        {
                                FreqOffset += FreqStep;
                                if ( FreqOffset >= FREQ_MAX )
                                        FreqOffset = FREQ_MIN;
                        }
                        while ( !FreqWidth[FreqOffset/FreqStep] );
                        PaintExtra();
                        MapStart();
                        break;
                case IDC_SWEEP_BANDS_2:
                        SerendipFixedBandView = FALSE;
                        for ( i=FREQ_MIN/FreqStep; i<FREQ_MAX/FreqStep; i++ )
                                FreqWidth[i] = FALSE;

                        for ( x=0; x<5; x++ )
                                for ( i=BeginFreq[1][x]/FreqStep; i<EndFreq[1][x]/FreqStep; i++ )
                                FreqWidth[i] = TRUE;
                        do
                        {
                                FreqOffset += FreqStep;
                                if ( FreqOffset >= FREQ_MAX )
                                        FreqOffset = FREQ_MIN;
                        }
                        while ( !FreqWidth[FreqOffset/FreqStep] );
                        PaintExtra();
                        MapStart();
                        break;
                case IDC_SWEEP_BANDS_3:

                        SerendipFixedBandView = FALSE;
                        for ( i=FREQ_MIN/FreqStep; i<FREQ_MAX/FreqStep; i++ )
                                FreqWidth[i] = FALSE;
                        for ( x=0; x<5; x++ )
                                for ( i=BeginFreq[2][x]/FreqStep; i<EndFreq[2][x]/FreqStep; i++ )
                                FreqWidth[i] = TRUE;
                        do
                        {
                                FreqOffset += FreqStep;
                                if ( FreqOffset >= FREQ_MAX )
                                        FreqOffset = FREQ_MIN;
                        }
```

```
                                        while ( !FreqWidth[FreqOffset/FreqStep] );
                                        PaintExtra();
                                        MapStart();
                                        break;
                                case IDC_SWEEP_BANDS_FIXED:

                                        SerendipFixedBandView = TRUE;
                                        for ( i=FREQ_MIN/FreqStep; i<FREQ_MAX/FreqStep; i++ )
                                                FreqWidth[i] = FALSE;
                                        FreqWidth[BandFixed/FreqStep] = TRUE;
                                        FreqOffset = BandFixed;
                                        PaintExtra();
                                        MapStart();
                                        break;
                                case ID_COMBO_FIXED:

                                        SelBandFixed =
(short)SendDlgItemMessage(hdwnd,ID_COMBO_FIXED,CB_GETCURSEL,0,0L);

        SendDlgItemMessage(hdwnd,ID_COMBO_FIXED,CB_GETLBTEXT,SelBandFixed,(LPARAM)str);
                                        BandFixed = (int)atoi(str);
                                        for ( i=FREQ_MIN/FreqStep; i<FREQ_MAX/FreqStep; i++ )
                                                FreqWidth[i] = FALSE;
                                        FreqWidth[BandFixed/FreqStep] = TRUE;
                                        FreqOffset = BandFixed;
                                        PaintExtra();
                                        break;
                        }
        }

        return (0);
}

int UltraD_ISR(void)
{
        static short        ScheduleInProgress=0,
                                dayminuteprec=0,
                                ScheduleAlreadyDone=0,
                                StartFreqOffset=0;

        USHORT                  BoardSpeed;     // = 1/(25ns * BS) --- BS=1 FreqRate=40MHz
        int                     result,h,i,k;
        double                  p1,p2;
        short                   dayminute, // ordinale del minuto della giornata
                                hh,
                                mm;
        char                    timestr[70],
                                timeline[9],
                                year[5],
                                month[4],
                                day[3],
                                dhour[3],
                                dmin[3];
        char                    str[10];
        time_t                  ltime;


        time( &ltime );
        sprintf(timestr,"%s",ctime( &ltime ));

        timeline[0]=timestr[11];timeline[1]=timestr[12];timeline[2]=timestr[13];timeline[3]=timestr[14];timeline[4]=timestr[15];ti
meline[5]=timestr[16];timeline[6]=timestr[17];timeline[7]=timestr[18];timeline[8]=0;
        dhour[0]=timestr[11];dhour[1]=timestr[12];dhour[2]=0;
        dmin[0]=timestr[14];dmin[1]=timestr[15];dmin[2]=0;
        year[0]=timestr[20];year[1]=timestr[21];year[2]=timestr[22];year[3]=timestr[23];year[4]=0;
        month[0]=timestr[4];month[1]=timestr[5];month[2]=timestr[6];month[3]=timestr[7];month[4]=0;
        day[0]=timestr[8];day[1]=timestr[9];day[2]=0;

        hh=atoi(dhour);if(hh==24) hh=0;
```

```
mm=atoi(dmin);
dayminute=hh*60+mm;

if (ScheduleInProgress)
{
        if((!SingleMode && Direction==7 || SingleMode) && StartFreqOffset == FreqOffset)
        {       // giro completo direzioni e frequenze
                // termine della schedula relativa al minuto interessato
                // (anche se è durata di più)
                ScheduleInProgress=0;
                EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_SAMPLE_RATE),TRUE);
                EnableWindow(GetDlgItem(g_hDlg1,ID_ACQUIRE),TRUE);
                EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_CHS),TRUE);
                EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_BURST),TRUE);
                SampleRate  = SaveSampleRate;
                NumFfts     = SaveNumFfts;
                if (SaveNumChannels != NumChannels)
                {
                        NumChannels = SaveNumChannels;
                        FftInit(NumChannels);
                }
        }
}
else
{
        if (SchedulingMap[dayminute].onoff==1)
        {  // inizio schedulazione
                // controllo per fare solo una schedula nello stesso minuto
                if(dayminuteprec == dayminute)
                        ScheduleAlreadyDone=1;
                else
                        ScheduleAlreadyDone=0;

                if (!ScheduleAlreadyDone)
                {
                        ScheduleInProgress=1;
                        dayminuteprec=dayminute;

                        EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_SAMPLE_RATE),FALSE);
                        EnableWindow(GetDlgItem(g_hDlg1,ID_ACQUIRE),FALSE);
                        EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_CHS),FALSE);
                        EnableWindow(GetDlgItem(g_hDlg1,ID_COMBO_BURST),FALSE);

                        SaveSampleRate  = SampleRate;
                        SaveNumFfts     = NumFfts;
                        SaveNumChannels = NumChannels;

                        SampleRate      = SchedulingMap[dayminute].samplerate;
                        FreqStep = SampleRate/2;
                        NumFfts         = SchedulingMap[dayminute].numffts;
                        NumMedie        = SchedulingMap[dayminute].nummedie;
                        NumChannels     = SchedulingMap[dayminute].numchannels;
                        if (SaveNumChannels != NumChannels)
                                FftInit(NumChannels);

                        // dobbiamo sincronizzarci in modo che
                        // la direzione prossima sia la prima
                        // perchè si inizia a registrare quando siamo in
                        // ScheduleInProgress che viene attivato al timer
                        // interrupt successivo
                        if (!SingleMode)
                                Direction=7;
                        else
                                Direction=8;

                        StartFreqOffset = FreqOffset;
                }
        }
}
```

```
        if ( !CalibrationCycle )
        {
                // update direction...
                if ( SingleMode )
                {
                        Direction = 8;
                        TurnComplete = TRUE;
                }
                else
                {
                        do
                        {
                                Direction++;
                                Direction %= 8;
                                if ( Direction == 0)
                                        TurnComplete = TRUE;
                        }
                        while ( !DirEnable[Direction] );

                        // update antennas control -> out to parallel port
                        // _outp(0x378,(int)Direction); // LPT 1
                }

                // update sweeping frequency
                if ( TurnComplete )
                {
                        do
                        {
                                FreqOffset += FreqStep;
                                if ( FreqOffset >= FREQ_MAX )
                                        FreqOffset = FREQ_MIN;
                        } while ( !FreqWidth[FreqOffset/FreqStep] );

                        Sintonize();
                        TurnComplete = FALSE;
                }
        }

        // board acquisition
        #ifndef AD_BOARD_SIMULATE

                switch(SampleRate)
                {
                        case 40:
                                BoardSpeed=1;
                                break;
                        case 20:
                                BoardSpeed=2;
                                break;
                        case 15:
                                BoardSpeed=3;
                                break;
                        case 10:
                                BoardSpeed=4;
                                break;
                        case 5:
                                BoardSpeed=8;
                                break;
                }

                result = acquire_ultrad_memory(
ultrad_board_handle,2*NumChannels*NumFfts,(USHORT)BoardSpeed,dataBuf);        // acquisisce solo la memoria richiesta
//              result = acquire_entire_ultrad_memory( ultrad_board_handle,(USHORT)BoardSpeed,dataBuf);
                        // acquisisce il blocco intero di 1Mword (4Mb)
                if (result==FALSE)
                        return (-1);
        #endif
```

```
		// spectra calculation
		for (k=0; k<NumChannels; k++)
			SpectrumData[k] = 0; // annullo l'offset

		for(h=0;h<NumFfts;h++) // per ogni banco di fft (burst)
		{
			for(i=0;i<2*NumChannels;i++) // 2*NumChannels reali
			{
				#ifndef AD_BOARD_SIMULATE
					// dataBuf è l'indirizzo, h è il banco di FFT, i è l'offset
					g[i]= (float)(*(dataBuf+(h*(2*NumChannels))+i) & 0xFFF);
				#else
					g[i]=2048+(1040.0*sin(((1+Direction)*.18+.24)*i));
				#endif

				// Windowing
//				g[i]=((double)i/(double)NumChannels)*g[i]; // triangular
//				g[i]=g[i]*(0.54+0.46*(double)cos(PI/2+PI*(double)i/(double)NumChannels));
//
	g[i]=g[i]*(0.402174+0.49703*cos(PI/2+PI*(double)i/(double)NumChannels)+0.09392*cos(PI/2+2*PI*(double)i/(double)
NumChannels)+0.00183*cos(PI/2+3*PI*(double)i/(double)NumChannels)); // blackman 4 termini
			}

			// SpectrumData è mediato su NumFfts
			if(!Oscilloscope)
			{
				#ifdef TEXASFFT
					radix2(g,(short)NumChannels,W);
					digit_reverse((INT64 *)g, IIndex, JIndex, R4IndexCount);
					split1((short)NumChannels,g,A,B,G);
				#else
				  rdft(2*NumChannels, 1, g, ip, W);
				#endif

				// Square and average
				for (k=0; k<NumChannels; k++)
				{
					#ifdef TEXASFFT
						p1 = (double)G[2*k]*(double)G[2*k];
						p2 = (double)G[2*k+1]*(double)G[2*k+1];
					#else
						p1 = g[2*k]*g[2*k];
						p2 = g[2*k+1]*g[2*k+1];
					#endif

					Spectrum[h][k] = (double)sqrt(p1+p2);

					if ( SentinelCalibrated ) // calibrazione
						// reale, non in visualizzazione
						Spectrum[h][k] -= SpectrumCalib[k];

					#ifndef FASTEST
						SpectrumData[k] = SpectrumData[k]+(int)Spectrum[h][k]; // annullo l'offset
					#endif
				}

				Spectrum[h][0] = Spectrum[h][1]; // annullo l'offset
			}

		}

		if(Oscilloscope)
			for (k=0; k<NumChannels; k++)
				SpectrumData[k] = (int)g[k];
		else
		{
			#ifndef FASTEST
				SpectrumData[0] = 0; // annullo l'offset
				for (k=1; k<NumChannels; k++)
```

```
                                        SpectrumData[k] = SpectrumData[k] / NumFfts; // visualizza lo spettro mediato
                #else
                        ;
                #endif
        }

        // time performance
        TickEnd = GetTickCount();
        TransferTime = (int)(TickEnd - TickStart);
        TickStart = GetTickCount();

        if ( CalibrationCycle )
                Calibration();
        else
        {
                char szTiming[100];

                if ( ScheduleInProgress )
                {
                        // registriamo gli spettri una schedulazione è in atto
                        SaveDataFiles(year,month,day,timeline);
                        sprintf(szTiming,"Schedule running... (min: %d)",hh*60+mm);
                }
                else if ( SaveToDisk )
                {
                        // registriamo gli spettri se salvattaggio manuale
                        SaveDataFiles(year,month,day,timeline);
                        sprintf(szTiming,"Burst: %d ms - 1 Work-cycle: %.2f ms
writing...",TransferTime,(float)TransferTime/(float)NumFfts);
                }
                else
                        sprintf(szTiming,"Burst: %d ms - 1 Work-cycle: %.2f
ms",TransferTime,(float)TransferTime/(float)NumFfts);

                SetDlgItemText(g_hDlg1,IDC_TIMING,szTiming);

                #ifndef FASTEST
                        PaintSpectrum();
                        PaintExtra();
                #endif

                sprintf(str,"%d MHz",FreqOffset);
                SetDlgItemText(g_hDlg1,IDC_STARTFREQ,str);
                sprintf(str,"%d MHz",FreqOffset+FreqStep);
                SetDlgItemText(g_hDlg1,IDC_ENDFREQ,str);


        }

        return(0);
}

/*
void AutoscaleXY(int *data)
{

        int MaxVal = data[0];

        for (int kx=1; kx<NumChannels; kx++)
        {
                int Level = data[kx];
                if ( Level > MaxVal )
                        MaxVal = Level;
        }
        if ( MaxVal == 0 )
                SpectrumMaxY = (int)1;
        else
                if ( SpectrumMode == MODE_SPECTRUM )
                        SpectrumMaxY = (int)( MaxVal*1.2 );
```

```
                else
                        SpectrumMaxY = (int)( MaxVal*2.1 );

        SpectrumStepY = SpectrumMaxY/8;

}
*/

int     MinValue(int *data)
{
        int MinVal = data[0];

        for (int kx=1; kx<NumChannels; kx++)
        {
                int Level = data[kx];
                if ( Level < MinVal )
                        MinVal = Level;
        }
        return MinVal;
}

void SaveDataFiles(char *year,char *month,char *day,char *timeline)
{
        char    name[11];
        int             medie_count;

        mkdir((char *)year);
        chdir((char *)year);

        mkdir((char *)month);
        chdir((char *)month);

        mkdir((char *)day);
        chdir((char *)day);

        // formato nome del file: Dir_StFreq_EndFreq
        sprintf(name,"%01d_%04d_%04d",Direction,FreqOffset,FreqOffset+FreqStep);

        // apertura file dati in append
        ofstream f1(name,ios::app,filebuf::sh_none);

        if ( !f1 )
        {
                char ErrorMsg[80];
                sprintf(ErrorMsg,"Impossible to open %s",name);
                MessageBox(g_hWnd,ErrorMsg,"ERROR",MB_ICONSTOP | MB_OK);
        }

        // Write Data
        f1 << (char *)timeline << ' ' << NumChannels << ' ' << NumFfts << ' ' << NumMedie << endl;

        medie_count=0;
        for (int num=0; num<NumFfts; num++)
        {
                if( NumMedie == 1 )
                {
                        for (int kx=0; kx<NumChannels; kx++)
                                f1 << '-' << (int)Spectrum[num][kx];

                        f1 << endl;
                }
                else
                {
                        if (medie_count==0)
                                for (int kx=0; kx<NumChannels; kx++)

        SpectrumMedie[num/NumMedie][kx]=(int)(Spectrum[num][kx]/(double)NumMedie);
                        else
                                for (int kx=0; kx<NumChannels; kx++)
```

```
                SpectrumMedie[num/NumMedie][kx]=(int)(SpectrumMedie[num/NumMedie][kx]+Spectrum[num][kx]/(double)NumMedi
    e);

                                medie_count++;
                                if( NumMedie==medie_count )
                                {
                                        medie_count=0;
                                        for (int kx=0; kx<NumChannels; kx++)
                                                f1 << '-' << SpectrumMedie[num/NumMedie][kx];

                                        f1 << endl;
                                }
                        }
                }

                f1.close();

                chdir("..");chdir("..");chdir("..");

        }

        BOOL SaveProfileData(short sel)
        {
                char buffer[6];
                WritePrivateProfileString( BandsSection[sel], "Label", Label[sel], InitFileName );
                for ( int x=0; x<5; x++ )
                        {
                                WritePrivateProfileString( BandsSection[sel], BeginLabel[x], itoa(BeginFreq[sel][x],buffer,10),
        InitFileName );
                                WritePrivateProfileString( BandsSection[sel], EndLabel[x],  itoa(EndFreq[sel][x],  buffer,10),
        InitFileName );
                        }
                return TRUE;
        }

        void Calibration()
        {
                short i;

                for(i=0;i<NumChannels;i++)
                        SpectrumCalib[i] = SpectrumData[i];
                TargetStop();

                CalibrationCycle = FALSE;
                SentinelCalibrated = TRUE;
                MessageBox(g_hWnd,"Calibration done!","Calibration",MB_OK);

                if (TargetStart()==-1)
                {
                        MessageBox(g_hWnd,"Can't access A/D converter","ERROR",MB_ICONSTOP | MB_OK);
                        exit(-1);
                }
        }

        int TargetStart()
        {

                #ifndef AD_BOARD_SIMULATE

                        // Load the ULTRAD driver and open a connection to the ULTRAD board
                        ultrad_board_handle = open_ultrad_board();

                        // If the driver can't be found or the driver loaded, then exit.
                        if (ultrad_board_handle == INVALID_HANDLE_VALUE)
                                return (-1);

                        // Malloc a user buffer to copy the device memory into.
```

```
                if (!(dataBuf = (DWORD *)malloc(ULTRAD_MEMORY_SIZE*sizeof(DWORD *))))
                        return (-1);
        #endif

        SetTimer(g_hWnd,1,3,NULL); // minimo ogni 3 ms

        return(0);
}

void TargetStop()
{
        KillTimer(g_hWnd,1);

        #ifndef AD_BOARD_SIMULATE

                // Unload the driver and cut the connection to the board.
                if( close_ultrad_board(ultrad_board_handle)==0)
                { // errore in chiusura
                        ;
                }

        #endif

        free(dataBuf);

}

BOOL CheckFiles()
{

        // Process Profile File
        GetPrivateProfileString( "Settings", "Save_drive", "C:", strSaveDrive, 3, InitFileName );
  //WritePrivateProfileString( "Settings", "Save_drive", strSaveDrive, InitFileName );

        BandsSelected = GetPrivateProfileInt( "Settings", "Bands_selected", 0, InitFileName );

        for ( int n=0; n<3; n++ )
        {
                GetPrivateProfileString( BandsSection[n], "Label", "Sequence", Label[n], 21, InitFileName );
                for ( int x=0; x<5; x++ )
                {
                        BeginFreq[n][x] = GetPrivateProfileInt( BandsSection[n], BeginLabel[x], FREQ_MIN, InitFileName );
                        EndFreq[n][x]   = GetPrivateProfileInt( BandsSection[n], EndLabel[x]  , FREQ_MAX, InitFileName );
                }
        }

        for ( int i=FREQ_MIN/FreqStep; i<FREQ_MAX/FreqStep; i++ )
                FreqWidth[i] = FALSE;
        for ( int x=0; x<5; x++ )
                for ( int i=BeginFreq[BandsSelected][x]/FreqStep; i<EndFreq[BandsSelected][x]/FreqStep; i++ )
                        FreqWidth[i] = TRUE;

        return (TRUE);
}

BOOL InitIEEE488()
{
        int status=0;
        char str[100];

        initialize (21,0);

        // comandi per HP 8561A spectrum analyzer
        sprintf(str,"SP %dMHZ;",16); // Span frequency
        send(17,str,&status);

        sprintf(str,"SNGLS;"); // single sweep
        send(17,str,&status);
```

```
        if ( status != 0 )
        {
                MessageBox(HWND_DESKTOP,"Sintonizer not found at given address ...\nRun
anyway...","ERROR",MB_ICONSTOP | MB_OK);
                SintPresent=FALSE;
                return FALSE;
        }

        Sleep(80);
        return TRUE;

}

void Sintonize()
{
        static int LastSint=0;
        int status;
        char str[100];

        if(SintPresent)
        {
                if(LastSint!=FreqOffset+FreqStep/2)
                {
                        sprintf(str,"CF %dMHZ;",FreqOffset+FreqStep/2); // center frequency
                        send(17,str,&status);
                        if ( status != 0 )
                                MessageBox(HWND_DESKTOP,"Sintonizer not found at given address ...\nRun
anyway...","ERROR",MB_ICONSTOP | MB_OK);

                        Sleep(80);
                        LastSint=FreqOffset+FreqStep/2;
                }
        }
}

void FftInit(int num_channels)
{
        int num,k;

        #ifdef TEXASFFT

                for (k=0; k<num_channels; k++)
                {
                        // calcola la tabella coseni/seni
                W[2*k]  = (short)(32767.0*(-cos((double)((double)2.0*k*PI/num_channels))));
                        W[2*k+1]= (short)(32767.0*(-sin((double)((double)2.0*k*PI/num_channels))));

                        A[2*k] = (short)(16383.0*(1.0 - sin(2*PI/(double)(2.0*num_channels)*(double)k)));
                        A[2*k+1] = (short)(16383.0*(-cos(2*PI/(double)(2.0*num_channels)*(double)k)));
                        B[2*k] = (short)(16383.0*(1.0 + sin(2*PI/(double)(2.0*num_channels)*(double)k)));
                        B[2*k+1] = (short)(16383.0*(cos(2*PI/(double)(2.0*num_channels)*(double)k)));
                }

                R2DigitRevIndexTableGen(num_channels, &R4IndexCount, IIndex, JIndex);

        #endif

        for (num=0; num<MAX_FFTS; num++)
                for (k=0; k<NumChannels; k++)
                {
                        Spectrum[num][k] = 0;
                        SpectrumMedie[num][k] = 0;
                }

}

void PaintSpectrum()
```

```
{
        int             BoxCar[MAX_CHANNELS>>LOG2_BOX_POINTS];   // 2 is min BOX_POINTS
        int             Normalized[MAX_CHANNELS];
        int             MovingAverage[MAX_CHANNELS];
        int             kx, oldX;
        double  ratio, oldY, y;
        POINT   PtsVect[2];

        if ( SpectrumMode != MODE_SERENDIP )
        { // visualizzazioni di base
                // Draw Black Screen
                FillRect(MemMaskDC,&SpectrumRect,(HBRUSH)g_hPenGray);
                SelectObject(MemMaskDC,g_hPenLightGray);
                SetTextColor(MemMaskDC,MY_YELLOW);

                // Draw Orizontal Grid
                if ( ScaleLog && !Oscilloscope )
                {
                        short pixel10db, topPixel, TopdBScale;
                        char dBstr[4];

                        kx=0;
                        TopdBScale=14;
                        pixel10db = FftC_10dB_Tarature[(int)log2(NumChannels/256)];
                        while (SpectrumTopOffset[(int)log2(NumChannels/256)]+kx*pixel10db<LEN_Y)
                        {
                                topPixel=SpectrumTopOffset[(int)log2(NumChannels/256)];

                                MoveToEx(MemMaskDC,ORG_X,ORG_Y+topPixel+kx*pixel10db,NULL);
                                LineTo(MemMaskDC,ORG_X+LEN_X+1,ORG_Y+topPixel+kx*pixel10db);
                                if ( SpectrumMode == MODE_SPECTRUM )
                                {
                                        sprintf(dBstr,"%3d",TopdBScale);
                                        TextOut(MemMaskDC,ORG_X+3,ORG_Y+topPixel+kx*pixel10db,dBstr,3);
                                        TopdBScale = TopdBScale-10;
                                }

                                kx++;
                        }
                }
                else
                {
                        for (kx=1; kx<10; kx++)
                        {
                                MoveToEx(MemMaskDC,ORG_X,ORG_Y+kx*LEN_Y/10,NULL);
                                LineTo(MemMaskDC,ORG_X+LEN_X+1,ORG_Y+kx*LEN_Y/10);
                        }
                }

                // Draw Vertical Grid
                for (kx=1; kx<10; kx++)
                {
                        MoveToEx(MemMaskDC,ORG_X+kx*LEN_X/10,ORG_Y,NULL);
                        LineTo(MemMaskDC,ORG_X+kx*LEN_X/10,ORG_Y+LEN_Y+1);
                }
        }

        if ( SpectrumMode == MODE_NORMALIZED )
        {
                // BoxCar shaping
                for (kx=0; kx<NumChannels>>LOG2_BOX_POINTS; kx++)
                {
                        int parziale = 0;
                        for (int j=0; j<BOX_POINTS; j++)
                                parziale += SpectrumData[kx*BOX_POINTS+j];
                        BoxCar[kx] = parziale>>LOG2_BOX_POINTS;
                }

                // Normalization: subtract BoxCar from FFT shape
```

```
                for (kx=0; kx<NumChannels>>LOG2_BOX_POINTS; kx++)
                        for (int j=0; j<BOX_POINTS; j++)
                                Normalized[kx*BOX_POINTS+j] = SpectrumData[kx*BOX_POINTS+j] - BoxCar[kx];


                // Moving Average
                for (kx=0; kx<NumChannels-MOV_AVRG_POINT+1; kx++)
                {
                        int parziale = 0;

                        for (int j=0; j<MOV_AVRG_POINT; j++)
                                parziale += Normalized[kx+j];
                        MovingAverage[kx] = parziale/MOV_AVRG_POINT;
                }
                // sugli ultimi punti le medie vanno "pesate"
                for (kx=MOV_AVRG_POINT-1; kx>0; kx--)
                {
                        int parziale = 0;

                        for (int j=0; j<kx; j++)
                                parziale += Normalized[NumChannels-kx+j];
                        MovingAverage[NumChannels-kx] = parziale/kx;
                }
        }

        if ( SpectrumMode == MODE_SERENDIP )
        {
                // BoxCar shaping
                for (kx=0; kx<NumChannels>>LOG2_BOX_POINTS; kx++)
                {
                        int parziale = 0;
                        for (int j=0; j<BOX_POINTS; j++)
                                parziale += SpectrumData[kx*BOX_POINTS+j];
                        BoxCar[kx] = parziale>>LOG2_BOX_POINTS;
                }

                // Normalization: subtract BoxCar from FFT shape
                for (kx=0; kx<NumChannels>>LOG2_BOX_POINTS; kx++)
                        for (int j=0; j<BOX_POINTS; j++)
                                Normalized[kx*BOX_POINTS+j] = SpectrumData[kx*BOX_POINTS+j] - BoxCar[kx];


                int MinVal = MinValue(Normalized);
                int div_factor = (int)(1+19.0*sqrt((double)NumFfts)/11.3); // NumFfts:1-128 -> div_factor:1-20
                for (kx=0; kx<NumChannels; kx++)
                        Normalized[kx] -= MinVal/15;

                // Moving Average
                for (kx=0; kx<NumChannels-MOV_AVRG_POINT+1; kx++)
                {
                        int parziale = 0;

                        for (int j=0; j<MOV_AVRG_POINT; j++)
                                parziale += Normalized[kx+j];
                        MovingAverage[kx] = parziale/MOV_AVRG_POINT;
                }
                // sugli ultimi punti le medie vanno "pesate"
                for (kx=MOV_AVRG_POINT-1; kx>0; kx--)
                {
                        int parziale = 0;

                        for (int j=0; j<kx; j++)
                                parziale += Normalized[NumChannels-kx+j];
                        MovingAverage[NumChannels-kx] = parziale/kx;
                }

                for (kx=0; kx<NumChannels; kx++)
                {
                        int Level   = Normalized[kx];
```

```
                        int Threshold = ThresholdFactor * MovingAverage[kx];

                        if ( Level <= Threshold )
                                Map[kx] = (char)0;
                        else
                        {
                                for (int j=15; j>0; j--)
                                        if ( Level > SerendipLevel[j] )
                                        {
                                                Map[kx] = (char)(j);
                                                break;
                                        }
                        }
                }
        }

        // Linear scale
        if (Oscilloscope)
                SpectrumMaxY=4100;
        else
                SpectrumMaxY=SpectrumMaxYY[(int)log2(NumChannels/256)];

        // Logaritmic scale
        if ( ScaleLog && (SpectrumMode != MODE_SERENDIP) )
        {
                if (Oscilloscope)
                        SpectrumMaxY=LOGARITMIC(4100);
                else
                        SpectrumMaxY=LOGARITMIC(SpectrumMaxYY[(int)log2(NumChannels/256)]);

                if ( SpectrumMode == MODE_SPECTRUM )
                {
                        for (kx=0; kx<NumChannels; kx++)
                        {
                                if ( SpectrumData[kx] > 0 )
                                        SpectrumData[kx] = (int)LOGARITMIC(SpectrumData[kx]);
                                else
                                        SpectrumData[kx] = 1;
                        }
                }
                if ( SpectrumMode == MODE_NORMALIZED )
                {
                        for (kx=0; kx<NumChannels; kx++)
                        {
                                int Level = MovingAverage[kx];
                                if ( Level > 1 )
                                        MovingAverage[kx] = (int)LOGARITMIC(Level);
                                if ( Level < -1 )
                                        MovingAverage[kx] = -(int)LOGARITMIC(abs(Level));
                        }
                }
        }

        // Draw Graph shape
        if ( SpectrumMode != MODE_SERENDIP )
        {
                SpectrumPalette [0][0]=128;SpectrumPalette [0][1]=0;  SpectrumPalette [0][2]=0;
                SpectrumPalette [1][0]=128;SpectrumPalette [1][1]=128;SpectrumPalette [1][2]=0;
                SpectrumPalette [2][0]=0;  SpectrumPalette [2][1]=128;SpectrumPalette [2][2]=0;
                SpectrumPalette [3][0]=0;  SpectrumPalette [3][1]=128;SpectrumPalette [3][2]=128;
                SpectrumPalette [4][0]=0;  SpectrumPalette [4][1]=0;  SpectrumPalette [4][2]=128;

                /*
                if ( Autoscale )
                {
                        int *pData;
                        switch( SpectrumMode )
                        {
                                case MODE_SPECTRUM:                     pData = SpectrumData;       break;
```

```
                                        case MODE_NORMALIZED:                   pData = Normalized;
        break;
                                }

                                AutoscaleXY(pData);
                                Autoscale = FALSE;
                        }
                        */

                        if(!PaintRainbowSpectrum)
                                MoveToEx(MemMaskDC,ORG_X,ZeroScale,NULL);

                        ratio   = (double)LEN_X/(double)NumChannels;
                        oldX    = ORG_X;
                        oldY    = ORG_Y+LEN_Y;

                        for (kx=0;kx<NumChannels;kx++)
                        {

                                if (SpectrumMode == MODE_SPECTRUM)
                                        y = ZeroScale -((SpectrumData[kx]*(double)LEN_Y)/SpectrumMaxY);
                                if (SpectrumMode == MODE_NORMALIZED)
                                        y = ( ZeroScale - ((double)(ZeroScale-ORG_Y)*(double)Normalized[kx])/SpectrumMaxY );

                                PtsVect[0].x = (long)ORG_X+(long)((double)kx*ratio);
                                if(PtsVect[0].x == oldX)
                                        // il canale ricade sullo stesso pixel del precedente
                                        oldY = min(y,oldY); // perchè il valore max è negativo
                                                                                // in quanto l'asse y è rovesciato
                                else
                                {
                                        y = min(y,oldY);
                                        PtsVect[1].x = PtsVect[0].x;
                                        PtsVect[1].y = (long)y;

                                        if(PaintRainbowSpectrum)
                                        {
                                                // si disegna il punto precedente
                                                if (SpectrumMode == MODE_SPECTRUM)
                                                        PtsVect[0].y = (LEN_Y+ORG_Y);
                                                if (SpectrumMode == MODE_NORMALIZED)
                                                        PtsVect[0].y = (LEN_Y/2+ORG_Y);

                                                short   tmp     = (short)(kx*5)/NumChannels;
                                                double  tmp1 = (double)((kx*5)%NumChannels)/(float)NumChannels;

        g_spectrum=CreatePen(PS_SOLID,1,RGB(SpectrumPalette[tmp][0]+(SpectrumPalette[tmp+1][0]-
SpectrumPalette[tmp][0])*tmp1,SpectrumPalette[tmp][1]+(SpectrumPalette[tmp+1][1]-
SpectrumPalette[tmp][1])*tmp1,SpectrumPalette[tmp][2]+(SpectrumPalette[tmp+1][2]-SpectrumPalette[tmp][2])*tmp1));
                                                SelectObject(MemMaskDC,g_spectrum);
                                                Polyline(MemMaskDC,(CONST POINT *)PtsVect,2);
                                        }
                                        else
                                        {
                                                g_spectrum=CreatePen(PS_SOLID,1,MY_YELLOW);
                                                SelectObject(MemMaskDC,g_spectrum);
                                                LineTo(MemMaskDC,PtsVect[1].x,PtsVect[1].y);
                                        }

                                        DeleteObject(g_spectrum);

                                        oldX = PtsVect[0].x;
                                        oldY = ORG_Y+LEN_Y;
                                }

                        }

                }
                else // SERENDIP MODE
```

```
        {
                register char       width = NumChannels/SER_LEN_X;
                if ( SerendipFixedBandView )
                {
                        int CoordX = ORG_X+(LEN_X-SER_LEN_X)/2;

                        SerendipLine = MapHeight;

                        int     CoordY;
                        if ( SingleMode )
                                CoordY = ORG_Y + 2 + SerendipLine;
                        else
                                CoordY = ORG_Y + 2 + Direction*LEN_Y/8 + SerendipLine;

                        BitBlt(MemMaskDC,CoordX,CoordY-
MapHeight,SER_LEN_X,MapHeight+1,MemMaskDC,CoordX,CoordY+2-MapHeight,SRCCOPY);

                        // draw 1 line
                        if (width>0)
                        {
                                for (kx=0; kx<SER_LEN_X; kx++)
                                {
                                        register char maxVal = Map[kx*width];

                                        for (int j=0; j<width; j++)
                                        {
                                                register char ch = Map[kx*width+j];
                                                if ( ch > maxVal )
                                                        maxVal = ch;
                                        }

                                        SetPixel(MemMaskDC,CoordX+kx,CoordY,SerendipColor[maxVal]);
                                }
                        }
                        else
                        {
                                ratio=(double)SER_LEN_X/(double)NumChannels;
                                for (kx=0; kx<NumChannels; kx++)
                                        SetPixel(MemMaskDC,CoordX+(int)((double)kx*ratio),
                                        CoordY,SerendipColor[Map[kx]]);
                        }

                }
        }

        InvalidateRect(g_hWnd,&SpectrumRect,0);

}

void PaintExtra()
{
        char str[80];

        // Draw Coordinates
        SetTextColor(MemMaskDC,MY_BLACK);

        // INFOs on the LEFT of the panel
        if ( SpectrumMode != MODE_SERENDIP )
        {

                if (ScaleLog)
                        sprintf(str,"10dBmxsect");
                else
                        sprintf(str,"0");

        }
```

```
		// Draw SweepingBar
		FillRect(MemMaskDC,&SweepingBarRect,g_hBrushGray);
		SelectObject(MemMaskDC,g_hPenYellow);
		for (int kx=0; kx<(FREQ_MAX-FREQ_MIN)/FreqStep; kx++)
			if ( FreqWidth[kx+FREQ_MIN/FreqStep] )
			{
				MoveToEx(MemMaskDC,BAR_X+3*kx+1,BAR_Y+3,NULL);
				LineTo(MemMaskDC,BAR_X+3*kx+1,BAR_Y+BAR_LEN_Y-3);
				MoveToEx(MemMaskDC,BAR_X+3*kx+2,BAR_Y+3,NULL);
				LineTo(MemMaskDC,BAR_X+3*kx+2,BAR_Y+BAR_LEN_Y-3);
			}

		SelectObject(MemMaskDC,g_hPenBlue);
		Rectangle(MemMaskDC,BAR_X+3*(FreqOffset-FREQ_MIN)/FreqStep,BAR_Y,BAR_X+3*(FreqOffset-
FREQ_MIN)/FreqStep+3,1+BAR_Y+BAR_LEN_Y);
		sprintf(str,"( %d - %d ) MHz",FreqOffset,FreqOffset+FreqStep);
		SetDlgItemText(g_hDlg1,IDC_FREQWINDOW,str);

		// Draw Direction Bitmap
		if ( SingleMode )
			BitBlt(MemMaskDC,DIR_X,DIR_Y,63,71,MemDirDC,0,71*8,SRCCOPY);
		else
			BitBlt(MemMaskDC,DIR_X,DIR_Y,63,71,MemDirDC,0,71*Direction,SRCCOPY);

		// Repaint screen
		InvalidateRect(g_hWnd,&SweepingBarRect,0);
		InvalidateRect(g_hWnd,&DirectionRect,0);
}

void MapStart()
{
		FillRect(MemMaskDC,&InfoLeftRect,g_hBrushLightGray);

		if ( SpectrumMode == MODE_SPECTRUM )
			ZeroScale = ORG_Y + LEN_Y;
		else
			ZeroScale = ORG_Y + LEN_Y/2;

		if ( SingleMode )
			MapHeight = LEN_Y-4;
		else
			MapHeight = LEN_Y/8-6;

		if ( SpectrumMode == MODE_SERENDIP )
		{
			SerendipLine = 0;
			// Draw Black Screen
			FillRect(MemMaskDC,&SpectrumRect,g_hBrushBlack);
			SelectObject(MemMaskDC,g_hPenLightGray);
			// Traccia estremi
			MoveToEx(MemMaskDC,ORG_X+(LEN_X-SER_LEN_X)/2-2,ORG_Y,NULL);
			LineTo(MemMaskDC,ORG_X+(LEN_X-SER_LEN_X)/2-2,ORG_Y+LEN_Y+1);
			MoveToEx(MemMaskDC,ORG_X+(LEN_X-SER_LEN_X)/2+SER_LEN_X+2,ORG_Y,NULL);
			LineTo(MemMaskDC,ORG_X+(LEN_X-SER_LEN_X)/2+SER_LEN_X+2,ORG_Y+LEN_Y+1);

			SetTextColor(MemMaskDC,MY_WHITE);
			if ( !SerendipFixedBandView )
			{
				TextOut(MemMaskDC,ORG_X+SPC+100,ORG_Y+3*tmCourier.tmHeight,"No view possible!",17);
				TextOut(MemMaskDC,ORG_X+SPC+100,ORG_Y+4*tmCourier.tmHeight,"Must be in fixed
band.",22);
			}
			else
			{
				if ( SingleMode )
				{
					TextOut(MemMaskDC,ORG_X+SPC,ORG_Y,"Single",6);
```

```
                                TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+tmCourier.tmHeight,"Mode",4);
                                TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+3*tmCourier.tmHeight,"Fixed",5);
                                TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+4*tmCourier.tmHeight,"Band",4);

                        }
                        else
                        {
                                for (int ky=1;ky<8;ky++)
                                {
                                        MoveToEx(MemMaskDC,ORG_X,ORG_Y-1+ky*LEN_Y/8,NULL);
                                        LineTo(MemMaskDC,ORG_X+LEN_X+1,ORG_Y-1+ky*LEN_Y/8);
                                }
                                TextOut(MemMaskDC,ORG_X+SPC,ORG_Y,"North",5);
TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+LEN_Y/8,"N-E",3);
                                TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+2*LEN_Y/8,"East",4);
TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+3*LEN_Y/8,"S-E",3);
                                TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+4*LEN_Y/8,"South",5);
TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+5*LEN_Y/8,"S-W",3);
                                TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+6*LEN_Y/8,"West",4);
TextOut(MemMaskDC,ORG_X+SPC,ORG_Y+7*LEN_Y/8,"N-W",3);
                        }
                }

                // Draw Serendip color scale
                int len_y = 11;
                int init_x = 3;
                int init_y = ORG_Y+len_y+2;

                for (int n=0;n<16;n++)
                        for (int ky=0;ky<len_y;ky++)
                                for (int kx=0;kx<8;kx++)
                                        SetPixel(MemMaskDC,init_x+kx,init_y+len_y*n+ky,SerendipColor[n]);

        }
        //else  Autoscale = TRUE;

        InvalidateRect(g_hWnd,&InfoLeftRect,0);
        InvalidateRect(g_hWnd,&SpectrumRect,0);
        PaintExtra();
}

void ReadScheduling(void)
{
        int             i,dummy;
        FILE    *fin;

        // apertura file dati in append
        fin = fopen("\\schedule","r");
        if (fin==NULL)
        {
                MessageBox(g_hWnd,"Can't find scheduling file","ERROR",MB_OK);
        }
        else
        {
                i=0;
                while (i<1440)
                {
                        fscanf(fin,"%d",&dummy);
                        if(dummy==1)
                        {
                                SchedulingMap[i].onoff                  =   1;
                                fscanf(fin,"%d",&dummy);
                                SchedulingMap[i].samplerate             =  dummy;
                                fscanf(fin,"%d",&dummy);
                                SchedulingMap[i].numchannels    =  dummy;
                                fscanf(fin,"%d",&dummy);
                                SchedulingMap[i].numffts                =  dummy;
                                fscanf(fin,"%d",&dummy);
                                SchedulingMap[i].nummedie               =  dummy;
```

```
                        }
                        else
                                SchedulingMap[i].onoff = 0;
                        i++;
                }

                fclose(fin);
        }

}

void InitLogTarature(void)
{
        #ifdef TEXASFFT

                FftC_10dB_Tarature[0]=36; // 256
                FftC_10dB_Tarature[1]=35; // 512
                FftC_10dB_Tarature[2]=34; // 1024
                FftC_10dB_Tarature[3]=33; // 2048
                FftC_10dB_Tarature[4]=32; // 4096

                SpectrumMaxYY[0]=485000;
                SpectrumMaxYY[1]=1200000;
                SpectrumMaxYY[2]=2000000;
                SpectrumMaxYY[3]=4900000;
                SpectrumMaxYY[4]=7800000;

                SpectrumTopOffset[0]=3;
                SpectrumTopOffset[1]=3;
                SpectrumTopOffset[2]=3;
                SpectrumTopOffset[3]=3;
                SpectrumTopOffset[4]=3;

        #else

                FftC_10dB_Tarature[0]=52; // 256
                FftC_10dB_Tarature[1]=48; // 512
                FftC_10dB_Tarature[2]=48; // 1024
                FftC_10dB_Tarature[3]=43; // 2048
                FftC_10dB_Tarature[4]=40; // 4096

                SpectrumMaxYY[0]=470000;
                SpectrumMaxYY[1]=1100000;
                SpectrumMaxYY[2]=2070000;
                SpectrumMaxYY[3]=4900000;
                SpectrumMaxYY[4]=7800000;

                SpectrumTopOffset[0]=5;
                SpectrumTopOffset[1]=3;
                SpectrumTopOffset[2]=10;
                SpectrumTopOffset[3]=7;
                SpectrumTopOffset[4]=8;

        #endif

}

double log2(double x)
{
        return(log10(x)/log10(2));
}

double mymin(double x,double y)
{
        if(x<y)
                return(x);
```

```
        else
                return(y);
}
```