SVILUPPO DI UNO STRUMENTO VIRTUALE REAL-TIME PER LA GENERAZIONE, ANALISI E ACQUISIZIONE DI SEGNALI.

di

Alfredo Accattatis

Tesi presentata per la discussione
del diploma di laurea in

Ingegneria Informatica

Università di Roma - Tor Vergata

A. A. 2005/2006

Relatore:
prof. *Salvatore Tucci*

Correlatore:
prof. *Marcello Salmeri*

Sommario

Indice delle figure

RINGRAZIAMENTI

Giunto oramai alla conclusione di questa mia nuova avventura Universitaria, ho avuto la fortuna di incontrare due persone senza le quali il mio desiderio di arrivare alla fine sarebbe stata solo una velleità. Per questo non smetterò mai di essere grato al prof. Salvatore Tucci, che ha dimostrato una competenza scientifica ed una umanità senza eguali, ed al contempo con quella umiltà tipica di chi "è" e non ha bisogno di "apparire". E parimenti al prof. Marcello Salmeri, mio vecchio amico e collega, che con la sua costante pazienza e capacità professionale ha saputo infondermi fiducia e rendermi piacevole il cammino.

Capitolo 1

Un circuito universale

1.0 Introduzione

Il termine "Computer" dal latino "cum" e "putare" con l'ovvio significato di contare, calcolare. Un altro termine correntemente usato è quello di "Elaboratore elettronico", cui si affianca il termine "Calcolatore". Tutti questi termini, ed altri correntemente usati, tendono ad identificare un tipo di macchina che sta dominando la scena tecnologica di questi ultimi decenni, e che può a ragione vedersi come una sorta di "macchina universale" ossia un dispositivo che, a seconda del programma eseguito si trasforma in un dispositivo dedicato alla risoluzione di una particolare classe di problemi. In termini ancora diversi un moderno elaboratore elettronico a programma memorizzato può vedersi come un insieme di componenti elettronici le cui connessioni, e quindi il circuito realizzato, sono determinate da un programma di elaborazione o "software". L'enorme varietà degli elaboratori elettronici esistenti di fatto non cambia il concetto appena esposto, che pertanto risulta valido sia per un potente "Mainframe" che per un semplice "Personal computer". Inoltre, i moderni sistemi operativi consentono l'esecuzione parallela o pseudo-parallela di più programmi, trasformando l'elaboratore in più "macchine dedicate" in funzionamento parallelo (per esempio un PC può eseguire musica mentre si scrive una lettera e si stampa un documento,

emulando di fatto tre differenti macchine: un lettore di mp3, una macchina per scrivere ed una stampante tipografica).

La categoria dei personal computer nasce ufficialmente nel 1977 con la messa in produzione del famosissimo Apple II. Prima di quel momento la scena era dominata dai microcomputer ed home computer che erano rivolti per lo più ad una ristretta cerchia di appassionati di elettronica e normalmente incompatibili tra loro, non semplici da usare e non provvisti "di serie" di periferiche di ingresso/uscita. Il personal computer era invece caratterizzato dall'essere contenuto in una scatola compatta e di formato standard, semplice da usare e provvisto di monitor e tastiera, ed ovviamente dotato di un sistema operativo. Pochi anni dopo questa importante rivoluzione vede la luce (nel 1981) il personal computer 5150 di IBM, meglio noto come PC-IBM. A fronte di prestazioni tutto sommato modeste, la caratteristica vincente di questa tipologia di macchine era che IBM forniva assieme al PC anche gli schemi elettrici e logici. Ed anche il listato del sistema operativo (dos) era facilmente ottenibile. A questo si affiancava la validissima rete di assistenza IBM, il prodotto era solido ed affidabile, e cosa forse più importante di tutte, usava un'architettura espandibile (la famosissima architettura a "schede" tuttora utilizzata) e basata su componenti commerciali non sviluppati "ad-hoc". Tutto questo ne decretò un notevole successo sia in ambito aziendale che nell'uso privato. Attirando l'attenzione dei produttori asiatici che invasero il mercato con i loro prodotti a basso costo, ossia i famosi PC IBM-compatibili, che hanno dato l'avvio ad una continua "corsa alle prestazioni" tuttora in atto. La compatibilità hardware/software verso i modelli più vecchi (compatibilità "verso il basso") ne ha accelerato ulteriormente la definitiva adozione su scala planetaria.

In tempi estremamente ridotti si è giunti ad avere un mercato in cui a costi praticamente irrisori si può oggi disporre di macchine estremamente potenti e

versatili, che talvolta fanno concorrenza a sistemi ben più costosi ed "antichi" quali i Mainframe; e che hanno invaso moltissimi altri campi, persino quelli sino a poco tempo fa appannaggio esclusivo di classi di microprocessori dedicati, quali ad esempio i DSP (Digital Signal Processor). La notevole disponibilità di schede aggiuntive sempre più economiche e performanti, quali schede sonore, schede video e supporti di memorizzazione veloci e affidabili (hard disk, dvd, cd-rom) fa si che i moderni personal computer possono praticamente trasformarsi in qualsiasi cosa, ossia siano dei "circuiti universali" assemblati tramite un software appositamente progettato.

La diffusione capillare in ogni azienda e quasi in ogni casa, in sinergia con lo sviluppo di sistemi operativi economici e sempre più affidabili (Windows XP, Linux) fa si che il limite delle possibilità di utilizzo sia dato solo dalla fantasia dei programmatori.

1.1 Sistemi operativi

Come detto, la diffusione capillare del personal computer va di pari passo con lo sviluppo di sistemi operativi capaci di rendere l'uso della macchina semplice ed efficiente. E che siano particolarmente economici. Il mercato attuale vede essenzialmente due vincitori: il sistema operativo "Windows" di Microsoft e l'ottimo (e open source) "Linux" nelle sue mille distribuzioni. La vittoria di questi due sistemi operativi è naturalmente dettata da fattori assai diversi dalla loro effettiva valenza tecnica, ma questo è un discorso ininfluente ai fini della presente trattazione e che lasciamo ad altri lavori.

Windows, sin dalla versione 95/98, implementa una versione di multitasking (preempitive) che comprende anche il supporto del Multi-Threading, caratteristica di cui faremo un uso intensivo nel corso del presente lavoro. La

versione più recente di Windows XP ha notevolmente migliorato l'implementazione di queste caratteristiche, al punto da rendere industrialmente utilizzabili i personal computer anche per applicazioni di tipo "real time", ossia per tutte quelle applicazioni per cui il tempo di esecuzione è un fattore critico. Questa categoria di programmi non era originariamente prevista dai progettisti di casa Microsoft. Grazie ad hardware sempre più potenti e paralleli, e tramite l'utilizzo del Multi-threading è attualmente possibile scrivere programmi che sono a tutti gli effetti in "tempo reale"; e quindi si allarga di fatto il campo di utilizzo dei moderni personal computer ad una ennesima categoria di problemi oltretutto assai vasta, che sconfina abbondantemente nel campo dell'elaborazione numerica dei segnali e del controllo di processi così come nel settore delle macchine utensili a controllo numerico.

Come conseguenza e come già accennato, molti programmi un tempo appannaggio esclusivo di sistemi hardware/software dedicati possono essere tranquillamente e più comodamente/economicamente implementati tramite un semplice programma per PC.

Per esempio programmi per l'analisi di segnali in tempo reale, per il filtraggio digitale e per la sintesi di musica elettronica. Od ancora programmi per il riconoscimento vocale ed il riconoscimento dei caratteri o programmi per l'editing ed il montaggio video (sebbene non di rado questi ultimi si appoggino a periferiche aggiuntive o necessitino di potenziare quelle in dotazione standard, per esempio la scheda video).

1.2 Visual Analyser – overview

Oggetto di questa tesi è la scrittura di un programma, denominato ***Visual Analyser,*** che dimostri come sia possibile utilizzare un personal computer per simulare un completo set di strumenti per il laboratorio dell'Ingegnere Elettronico e per i tecnici elettronici in genere, che usi esclusivamente la dotazione hardware di base ed il sistema operativo in dotazione. In altre parole, senza praticamente alcun hardware aggiuntivo (che al massimo si limita all'uso di un set di cavi ed opzionalmente un partitore resistivo) e nessuna modifica al sistema operativo (Windows o Linux). Dimostrando così come un semplice personal computer sia effettivamente una macchina universale adatta agli impieghi più diversi e complessi.

Gli strumenti di misura ed analisi dei segnali elettrici, normalmente utilizzati nei laboratori di elettronica, prelevano il segnale elettrico generalmente tramite delle "sonde" o "puntali" li amplificano e li elaborano tramite appositi circuiti (es. amplificatori) e li visualizzano su un adatto strumento di visualizzazione, che può essere un semplice galvanometro od uno schermo a raggi catodici o persino una semplice barra di diodi led. Per esempio un multimetro è generalmente usato per misurare valori di tensione e corrente sia continua che alternata (in questo secondo caso ne misurano il valore RMS) o per la misura di valori di resistenza, e fa generalmente uso di un galvanometro opportunamente graduato (vedi figura 1) mentre un oscilloscopio è usato per visualizzare in tempo reale le forme d'onda presenti in un circuito tramite uno schermo su cui si visualizza il segnale (vedi figura 2). Ancora, un analizzatore di spettro serve a scomporre un segnale nelle sue componenti armoniche in ampiezza e fase.

Figura 1: multimetro analogico

Figura 2: oscilloscopio

In generale gli strumenti di misura si dividono grossolanamente in due grandi categorie: analogici e digitali. I primi si limitano ad acquisire il segnale direttamente, lo amplificano od attenuano con degli amplificatori analogici, e li visualizzano sempre in maniera analogica (vedi il classico multimetro analogico con strumento a bobina mobile o l'oscilloscopio con tubo a raggi catodici). I secondi effettuano prima una conversione da segnale analogico (ossia tempo continuo) a segnale digitale in maniera da poter utilizzare per

l’analisi ed elaborazione dei circuiti a microprocessori, ottenendo prestazioni in genere nettamente superiori. Questa seconda modalità, sfrutta un teorema fondamentale detto “Teorema del campionamento” o “Teorema di Shannon-Nyquist” tramite il quale è possibile trasformare ogni segnale elettrico analogico (ossia un segnale la cui variabile temporale è un numero reale) in un segnale discreto (la variabile temporale è un numero intero). Successivamente il segnale così ottenuto viene “digitalizzato”, ossia ad ogni campione viene associato un numero (intero) trasformando di fatto dei segnali elettrici tempo continuo in una sequenza di numeri “comprensibili” ad un normale elaboratore (per esempio anche un PC).

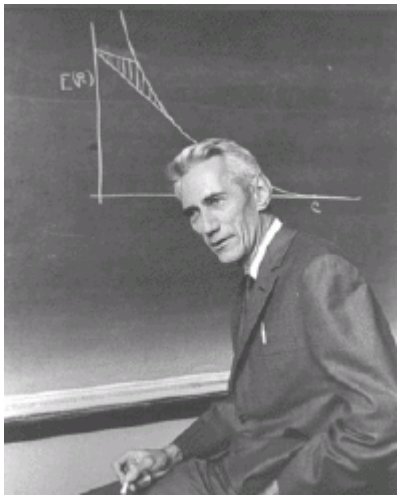


Figura 3: Claude Shannon

Ogni personal computer, quasi senza eccezione, è dotato di una scheda sonora, e quindi dei necessari circuiti di amplificazione, campionamento e digitalizzazione dei segnali, più altri particolarmente utili quali ingressi multipli, mixer e filtri, e talune persino preamplificatori particolari (es. preamplificatori con curva di equalizzazione RIAA per dischi in Vinile). E con la possibilità di acquisire via software applicativo i campioni numerici (tramite

API di sistema), accumularli in ram sotto forma di array di campioni ed elaborali opportunamente.

Il programma oggetto di questo lavoro fa uso della scheda sonora come strumento di acquisizione dei segnali elettrici; esso pertanto ricade nella categoria degli strumenti di tipo digitale, ed utilizza come strumento di visualizzazione il monitor del PC. E' facile immaginare le enormi potenzialità di un software di questo tipo, i cui limiti sono dettati esclusivamente dalla qualità della scheda sonora e dalla potenza di calcolo dell'hardware del PC stesso. Il PC diventa così terreno fertile per la pratica applicazione di tutti i concetti della materia nota come "*elaborazione numerica dei segnali*". Infatti, la disponibilità di un segnale numerico in memoria ed un hardware veloce, in sinergia con l'utilizzo di algoritmi sofisticati, consentono di simulare un numero enorme di "circuiti virtuali" che rendono l'elaboratore ancora più flessibile ed adatto a simulare la maggior parte degli strumenti di misura. Tra questi è notevole la famosissima trasformata di Fourier (tramite la quale effettuare analisi armonica), od i filtri digitali (FIR o IIR tramite i quali implementare filtri le cui prestazioni sono impensabili nel mondo analogico) e molti altri ancora. Ed infine con la possibilità di automatizzare l'uso dei vari strumenti (per esempio si può generare un segnale di test, inviarlo all'ingresso di un device da analizzare ad intervalli di tempo predefiniti, contemporaneamente si effettua la lettura, ed infine confronto e visualizzazione su schermo dei risultati).

Purtuttavia è da notare come le moderne schede sonore siano comunque limitate al campo delle basse frequenze: nelle più sofisticate si arriva a frequenze di campionamento dell'ordine di 192 kHz, e quindi a bande passanti pari a circa 96 kHz e che generalmente non comprendono la componente continua. E' da notare altresì che un numero consistente di circuiti elettronici tratta segnali esclusivamente ad audio frequenza, e che in ogni modo Visual Analyser è predisposto per l'uso di schede di acquisizione dati dedicate (e

quindi con frequenze di campionamento paragonabili a strumenti professionali). Inoltre, l'uso del paradigma Object Oriented rende facile la modifica del programma in funzione di schede progettate "ad hoc" per specifiche applicazioni.

1.3 Hardware e software

Le scelte di base effettuate si sintetizzano nella lista che segue:

1. Sistema operativo : Windows, dalla versione 95 in poi con possibilità di porting su Linux od utilizzo diretto su Linux tramite Wine
2. Hardware : PC Ibm compatibile con processore a partire da pentium II con min. 64 Mb di ram e minimo una *scheda sonora*
3. Linguaggio di programmazione : Borland C++ builder versione 6.0

In dettaglio:

(1) La scelta di Windows è stata effettuata perché è di fatto il sistema operativo più diffuso e relativamente economico. Il programma è stato sviluppato originariamente su Windows 2000 e successivamente su Windows XP. Esso è tuttavia perfettamente compatibile con Windows 98 e persino Windows 95. L'uso del compilatore Borland rende possibile un porting relativamente semplice su Linux tramite il compilatore aggiuntivo "Borland Kylix 3.0", che si affianca al BC++ Builder 6.0 ed è in grado di ricompilare il codice scritto per Windows (e viceversa) a patto di utilizzare esclusivamente le librerie di oggetti "cross-platform" appositamente definite in entrambi i compilatori. In altre parole, qualsiasi chiamata ad API di sistema deve essere "mediata" da una classe definita con identica interfaccia (ossia gli stessi metodi) per entrambi i compilatori ma che evidentemente al suo interno effettua

le chiamate alle API opportune definite nei rispettivi sistemi operativi. Nella attuale versione del programma è possibile compilare per Linux solo una versione con limitate funzionalità. Esigenze di elevata ottimizzazione rendono più conveniente la chiamata diretta di API di sistema e talvolta persino a system call e routine in linguaggio macchina. Tuttavia, tramite l'uso di un pacchetto aggiuntivo in Linux è generalmente possibile utilizzare molte delle applicazioni scritte per Windows direttamente in Linux. Il pacchetto Wine (www.winehq.org) è un software open source che consente di emulare il comportamento di moltissime API di Windows. L'uso di questo pacchetto ha reso possibile utilizzare il programma anche in Linux direttamente e senza bisogno di ricompilazioni.

(2) Questo punto è diretta conseguenza del precedente; l'uso di codice altamente ottimizzato ha permesso di far girare il programma su macchine molto meno performanti di quelle su cui il programma è stato effettivamente sviluppato (tipicamente pentium IV 1.6 Ghz), arrivando a rilevare prestazioni accettabili persino su normali Pentium MMX di vecchissima generazione e con quantità irrisorie di memoria (32 Mbyte). Risultati non sempre identici tra macchine apparentemente simili hanno consigliato di indicare come requisito minimo l'uso di un pentium II con almeno 64 Mb di ram. La scheda sonora è un prerequisito essenziale: i segnali vengono acquisiti tramite essa e parimenti generati tramite i suoi circuiti. La mancanza della scheda sonora determina l'impossibilità di eseguire il programma. Nella pratica, non è possibile trovare un PC sprovvisto di scheda sonora. Nel peggiore dei casi essa è infatti direttamente integrata nella scheda madre. Il programma è inoltre in grado di gestire schede sonore multiple, senza limitazione di numero. In pratica, se il programma è usato sia per la generazione di segnali (funzione generatore di funzioni) che per l'acquisizione, è possibile utilizzare due schede sonore

differenti, una per ciascuna funzione. Invero, la quasi totalità delle schede sonore attualmente in commercio è di fatto full-duplex, ossia consente di generare ed acquisire i segnali contemporaneamente, pertanto la necessità di poter utilizzare due schede sonore differenti è certamente meno necessaria.

(3) Il linguaggio di programmazione scelto è il BC++ Builder 6.0, con un uso relativamente limitato delle VCL a corredo dello stesso. Infatti quasi ogni routine è invero stata scritta ex-novo data la natura strettamente real-time del problema, non facendo pertanto uso di oggetti (e VCL) presenti a corredo della pur nutrita libreria del compilatore di casa Borland. Il codice scritto è perfettamente ricompilabile con il più attuale Borland Developer Studio, che non mantiene tuttavia la completa compatibilità con Kylix e fa dunque venir meno la possibilità di effettuare il porting verso il sistema operativo Linux. Per questo motivo ed altri meno eclatanti si è preferito continuare ad usare la versione 6.0 del compilatore BC++ Builder della Borland. Scendendo in maggiore dettaglio, vediamo di dare ulteriori nozioni che facciano comprendere la scelta di questo compilatore. Nel variegato ed in continua evoluzione mondo dei compilatori C++, spiccano essenzialmente due grandi antagonisti: il mondo Microsoft, con il suo onnipresente Visual C++ e il mondo Borland con il su C++ Builder. Una delle grosse differenze è a vantaggio della Borland è relativa alla libreria di oggetti a corredo del compilatore, che in massima parte dipende da terze parti. In pratica, il Visual C++ fornisce una libreria che deriva direttamente dalle MFC (Microsoft Foundation Classes). Essa è costituita essenzialmente da una completissima lista di oggetti che incapsulano la maggior parte delle API di Windows più molti altri che forniscono già pronte le routine di uso più frequente (per esempio oggetti che implementano editor di testi, lettori di mp3 etc). La risposta Borland è l'altrettanto famosa OWL (Object Windows Library) che per

lo più fornisce le stesse potenzialità della libreria di casa Microsoft. Borland fornisce poi un secondo tipo di libreria, detta VCL (Visual Custom Library) , che tramite una semplice estensione del linguaggio C++ (sono state aggiunte alcune nuove parole chiave) consente di definire un nuovo tipo di libreria, completamente visuale e che si integra rapidamente nell' IDE del compilatore. In altre parole, è possibile standardizzare i propri oggetti e compilarli in maniera da creare degli oggetti "visuali" che possono essere rapidamente integrati nel proprio progetto tramite una operazione di "drag & drop". La possibilità di distribuire le librerie così sviluppate anche senza il codice sorgente, ossia pre-compilate, apre al mondo dei produttori esterni, un po' come l'architettura IBM modulare del PC ha fatto per l'hardware. Inoltre, proprie librerie di uso frequente diventano realmente riutilizzabili in molti progetti, mantenendo finalmente la promessa di "riusabilità" spesso disattesa dal modello Object Oriented . Invero, la natura essenzialmente real-time del programma oggetto di questo lavoro, ha sconsigliato l'uso delle VCL per le sezioni più critiche, e persino sconsigliato l'uso di semplici librerie (anche ottimizzate) che semplificano l'uso delle API più comuni. Si è infatti optato per l'uso diretto di API ed in molti casi si è preferito riscrivere ex-novo alcune librerie di oggetti che incapsulano in maniera molto più efficiente (sebbene meno elegantemente) le API maggiormente utilizzate (per esempio per il disegno a video, per la gestione dei semafori, e per la gestione della scheda sonora in genere). In ogni caso, per le funzioni meno critiche e taluni aspetti dell'interfaccia utente sono stati utilizzati oggetti VCL a corredo ottenendo un notevole risparmio di tempo e certezza di stabilità di funzionamento (per esempio nelle finestre di acquisizione spettro e oscilloscopio).

Il programma realizzato è frutto di oltre due anni di lavoro "continuativo" distribuiti su 5 anni effettivi e consta di oltre 35000 linee di codice C++ divisi in più di 100 files, tra file "c" e file "h" oltre ai file necessari per la definizione delle varie "form" (i file .dfm che rappresentano la descrizione visuale delle finestre utilizzate a "design time"). Esso è interamente Object Oriented, sebbene esigenze di velocità di esecuzione abbiano talvolta costretto a violare le regole di una corretta programmazione Object Oriented. Come detto, lo scopo del programma è quello di simulare un set di strumenti di misura per l'uso nel laboratorio elettronico, senza alcun hardware aggiuntivo. Il prerequisito essenziale è ovviamente la presenza di almeno una scheda sonora (due sono preferibili, come già accennato) essendo essa utilizzata come hardware per l'acquisizione dei segnali esterni o per la sintesi degli stessi (funzione "generatore di funzioni").

1.4 Gli strumenti simulati

Di seguito la lista degli strumenti che Visual Analyser consente di simulare:

1) Oscilloscopio
2) Analizzatore di spettro
3) Generatore di funzioni (senza aliasing)
4) Frequenzimetro
5) Voltmetro AC
6) Filtri digitali
7) Cattura segnali nel dominio del tempo e frequenza con stampa e salvataggio
8) Cattura dei segnali con threshold e pre-acquisizione
9) Distorsiometro (THD, THD+noise)

10) Rilevazione automatica della risposta in frequenza, tramite uso automatico di (1), (2) e (3)

Tutte questi strumenti sono stati simulati utilizzando sette thread all'interno del più generale processo "Visual Analyser" di cui due sono in esecuzione permanente e cinque in esecuzione "on demand", nel senso che sono creati solo al momento in cui è richiesta una specifica funzione (per esempio si invoca la funzione "generatore di funzioni") e cessano di esistere con la chiusura di essa (esempio messa in "off" del generatore di funzioni).
Ognuno degli strumenti simulati effettua l'analisi di un segnale esterno (oppure due in contemporanea se la scheda è stereo), eccetto il generatore di funzioni che al contrario ne genera uno: per acquisire o generare un segnale viene utilizzata la scheda sonora ed in particolare rispettivamente gli ingressi "line-in" "mic" o "aux" e le uscite "speaker out" o "line out" (cfr sezione scheda sonora). Da notare che la stragrande maggioranza delle schede è stereo (eccetto per l'ingresso microfonico che generalmente è uno solo) e quindi ogni strumento simulato è doppio: per esempio l'oscilloscopio è "doppia traccia" così come l'analizzatore di spettro e tutti i restanti strumenti.

1.5 Il programma in pratica

Il programma in esecuzione si presenta come in figura 4; alla prima esecuzione compare la finestra principale che contiene la finestra dell'oscilloscopio con i suoi comandi essenziali, la finestra analizzatore di spettro anch'essa con i suoi comandi più usati ed una "barra dei comandi" posta sotto l'estremo superiore della finestra (la "barra" del titolo). La "barra comandi" è in particolare sempre visibile anche nella modalità "flottante", caratteristica di cui parleremo diffusamente nel seguito. Nella barra dei comandi è possibile invocare praticamente tutte le funzioni disponibili nel programma, compresa una

particolare finestra, detta di “Settings”, dove sono racchiusi tutti i comandi, opzioni e preferenze che è possibile impartire al programma.

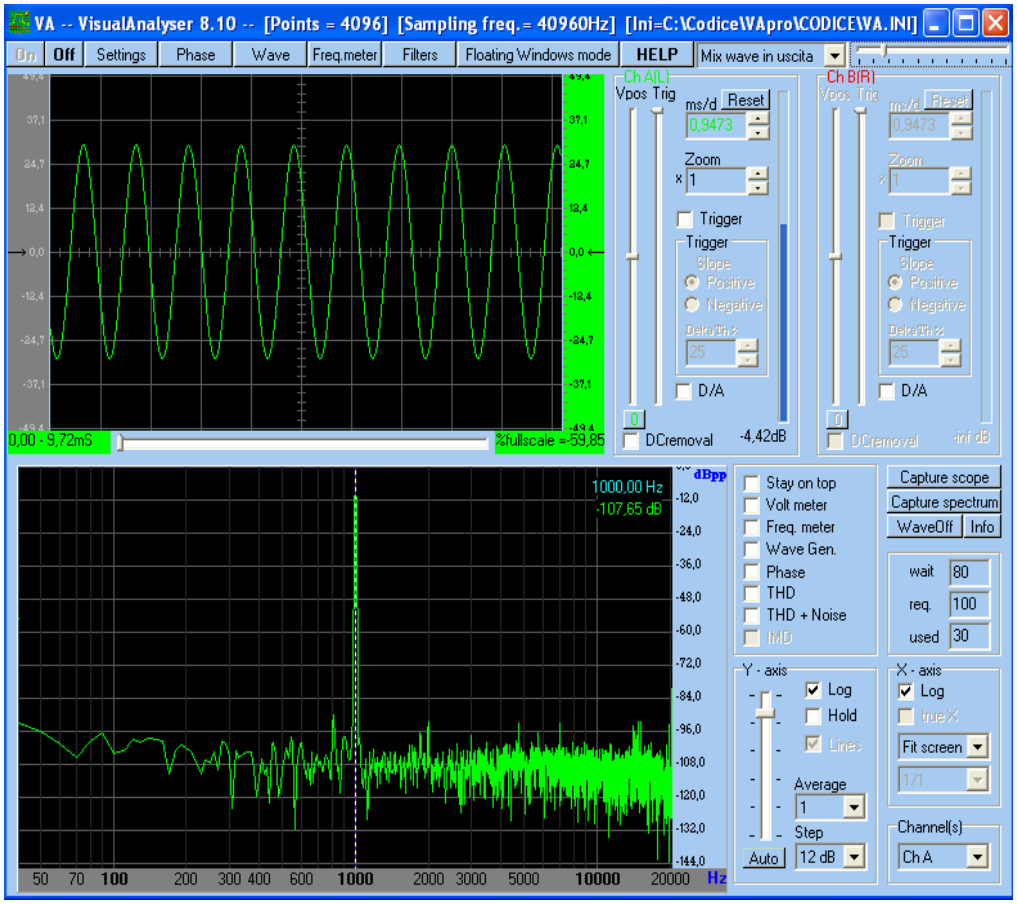


Figura 4 : Visual Analyser in configurazione standard

La barra dei comandi contiene dunque la possibilità di invocare le varie funzioni del programma semplicemente cliccando sul bottone corrispondente. In particolare il bottone “ON” serve a mandare in esecuzione le routine che acquisiscono i segnali dalla scheda sonora, poi elaborati secondo quanto richiesto dalle varie funzioni (esempio calcolo dello spettro per la funzione analizzatore di spettro, calcolo della frequenza per la funzione frequenzimetro eccetera) . Il bottone “OFF” serve ad arrestare il meccanismo di acquisizione. Tutte le funzioni non comprese nella finestra principale provocano la comparsa di finestre aggiuntive; la configurazione prescelta è memorizzabile su un file con estensione .ini ed alla successiva partenza ripristinata. Se si desidera

“specializzare” il programma per una o più funzioni (per esempio si è interessati al solo oscilloscopio, oppure al solo generatore di funzioni ed analizzatore di spettro) è prevista l’opzione “finestre flottanti”. Con essa, invocabile tramite la pressione del tasto “Floating Windows mode” situato nella barra dei comandi, la finestra principale si riduce alla sola barra dei comandi, ed ogni funzione selezionata (per esempio l’oscilloscopio) provoca la comparsa di una finestra dedicata, completamente disgiunta dalla finestra principale e liberamente dimensionabile. Anche in questo caso è possibile salvare la configurazione che alla successiva ripartenza verrà ripristinata. In tal modo è possibile salvare più configurazioni (per esempio una con solo oscilloscopio, oppure con solo analizzatore di spettro od ancora con oscilloscopio e frequenzimetro) che potranno essere ripristinate in funzione di applicazioni diverse, risparmiando tempo e risorse di elaborazione.

1.6 Prestazioni

Le prestazioni di VA saranno ovviamente dipendenti dalla macchina utilizzata, ossia dal processore, dalla quantità di memoria e dalla bontà della scheda madre. Per quest’ultimo parametro si fa riferimento alla bontà di “accoppiamento” tra i vari elementi della scheda madre e qualità dei componenti elettronici in se, oltre alle differenti filosofie adottate dai diversi costruttori (memorie cache, bus, etc). E, in misura preponderante, dalle caratteristiche della scheda sonora.

Le moderne schede sonore hanno raggiunto prestazioni notevoli persino nei modelli “entry level”. Non è infrequente trovare rapporti S/N dell’ordine dei 96 .. 144 dB (teorici) e frequenze di campionamento che possono arrivare a 192 kHz con un numero di bit utilizzati sino a 24 (32 in modelli professionali non

molto diffusi). Per dare un'idea numerica del grado di efficienza raggiunto, si faccia riferimento ad una configurazione siffatta:

- Frequenza di campionamento 40960 Hz pari ad una banda passante di 20480 Hz
- Buffer di acquisizione 4096 valori
- Abilitato un solo canale (per esempio il sinistro)
- Hardware Pentium IV 1.6 Ghz 512 Mb Ram scheda video NVIDIA Vanta 128 Mb ram
- Dimensioni finestra oscilloscopio e analizzatore di spettro "standard"
- Nessuna altra funzione attivata

Con questi parametri il tempo "fisico" necessario per acquisire un intero buffer di campioni è pari esattamente a 100 millisecondi. Infatti in ogni secondo vengono acquisiti 40960 campioni, ossia 40960/4096 = 10 cioè in un decimo di secondo (100 mS) vengono acquisiti i 4096 punti. Questo dato è indipendente dalla macchina, ma è un fatto fisico dettato dai parametri scelti per la configurazione delle scheda sonora. Il tempo impiegato per calcolare la trasformata di Fourier, disegnare a video la stessa più il disegno della funzione oscilloscopio ed altre operazioni minori è pari, per una macchina configurata come sopra, un tempo approssimativo oscillante tra i 10 e 15 millisecondi (la variabilità è data dal tempo di volta in volta "rubato" dal sistema operativo "preempitive" al processo Visual Analyser, che non è ovviamente l'unico in esecuzione). Questo tempo cresce sino a triplicare se è attiva la funzione conversione D/A. Qualora per vari motivi il programma non riesca a stare entro i tempi disponibili (100 mS in questo caso) viene segnalata la "perdita di dati" ma il programma continua a girare correttamente. Per consentire al programma di sopperire a momentanee fluttuazioni nella disponibilità delle risorse si è implementato un semplice sistema a coda circolare che consente di accumulare "n" buffer (tipicamente 10) . In altre parole, se per esempio il sistema operativo decide di interrompere l'esecuzione di Visual Analyser in favore di un altro

processo a maggiore priorità (per esempio un processo di sistema) i dati accumulati nei buffer interni vengono poi rapidamente svuotati non appena il controllo passa nuovamente al programma (NOTA: i buffer sono regioni di memoria allocate in RAM, passati come puntatori alla scheda sonora: essi sono riempiti senza l'intervento delle risorse elaborative del PC, quindi parallelamente ad esso; in parole diverse se la CPU viene "tolta" al processo Visual Analyser i buffer continuano ad essere riempiti). In pratica (vedi sezione successiva) il thread principale di acquisizione dati testa ogni volta il numero di buffer "pronti" per essere utilizzati; qualora in numero maggiore di uno esso tenta di utilizzarli tutti nel rispetto del tempo massimo a sua disposizione, altrimenti continua nel giro successivo. Ossia, sempre relativamente all'esempio attuale, entro i 100 millisecondi a disposizione. In questo caso, ammettendo di impiegare 10 millisecondi per "servire" un buffer, rimane un margine di 90 mS ossia la possibilità di servire altri 9 buffer eventualmente accumulati.

L'uso quotidiano con macchine di questa potenza (ed a maggior ragione superiori) dimostra che assai di rado si arriva alla perdita di dati, confermando la perfetta esecuzione "real time" del programma anche in un sistema operativo come Windows (XP, 9X, 2000, Nt).

1.7 Applicazioni

Le possibili applicazioni di un simile prodotto sono ovviamente tutte quelle previste per un laboratorio elettronico dotato di tutti questi strumenti, ossia infinite. A titolo di esempio possiamo citare la misura della risposta in frequenza di un amplificatore audio. Essa può essere condotta in varie modalità, tra le quali la più interessante è quella che fa uso del generatore di

funzioni interno e del calcolo della funzione di trasferimento, metodologia che consente di mantenere una relativa indipendenza della misura dalle caratteristiche proprie di banda passante e distorsione della scheda sonora. Infatti, per misurare la risposta in frequenza dell'amplificatore utilizziamo (anche) un altro amplificatore (quello interno alla scheda sonora). La cui qualità non è necessariamente migliore di quello sotto analisi: calcolando la funzione di trasferimento come rapporto tra il segnale d'ingresso e quello in uscita, ossia, equivalentemente, come differenza in dB tra il segnale applicato all'uscita ed all'ingresso dell'amplificatore sotto misura, otterremmo in teoria una perfetta compensazione della banda passante dell'amplificatore audio interno alla scheda (cfr capitolo settimo).

Un'altra interessante applicazione è la generazione di forme d'onda arbitrarie, che possono essere impostate armonica per armonica, e con l'ausilio di una finestra di "preview", che mostra in tempo reale la forma d'onda ottenuta man mano che vengono aggiunte le varie armoniche (in termini di ampiezza, fase e frequenza). Molti strumenti sia software che hardware non prevedono questa possibilità e talvolta, pur prevedendo una qualche forma di generazione arbitraria, vanno incontro all'insidioso fenomeno noto come "aliasing" che in Visual Analyser non è presente nemmeno per le forme d'onda "predefinite".

1.8 La conversione analogico digitale

A conclusione di questa introduzione descriviamo una caratteristica peculiare di Visual Analyser che verrà sviluppata dettagliatamente nel capitolo quarto. Ogni programma simile a Visual Analyser utilizza i campioni numerici prelevati dai buffer interni della scheda sonora; essi sono limitati in banda (da un filtro anti-aliasing presente nella scheda stessa) e campionati ad una determinata frequenza stabilita dall'utente. I campioni così ottenuti

rappresentano completamente il segnale acquisito, e vengono pertanto utilizzati per rappresentare a video il segnale (per esempio) sulla finestra oscilloscopio. La tecnica utilizzata normalmente consiste nel disegnare a video i punti acquisiti raccordandoli con un segmento di retta, ottenendo una rappresentazione veloce da disegnare e relativamente "simile" al segnale analogico da essi rappresentato. Invero, questo va bene per frequenze fondamentali del segnale molto minori della massima frequenza ammessa; tipicamente minore di 1/5 .. 1/6 della frequenza di Nyquist. Per fissare le idee, un segnale campionato a 10 kHz avrà una banda passante che si estende sino a 5 kHz; una sinusoide a 1000 Hz sarà pertanto rappresentata da 10 punti a ciclo (=10 kHz /1000Hz). Questo significa che per ogni ciclo del segnale avremo 10 punti raccordati da segmenti di retta. Ossia un disegno la cui approssimazione del segnale reale analogico (composto da infiniti punti) è relativamente avvertibile ma tollerabile (figura 5). Se visualizziamo un segnale a 100Hz la situazione sarà ancora migliore (figura 6) avendo a disposizione la bellezza di 100 punti per ciclo (in questo caso, vista la risoluzione "discreta" di uno schermo di una scheda grafica praticamente indistinguibile dal segnale originale).

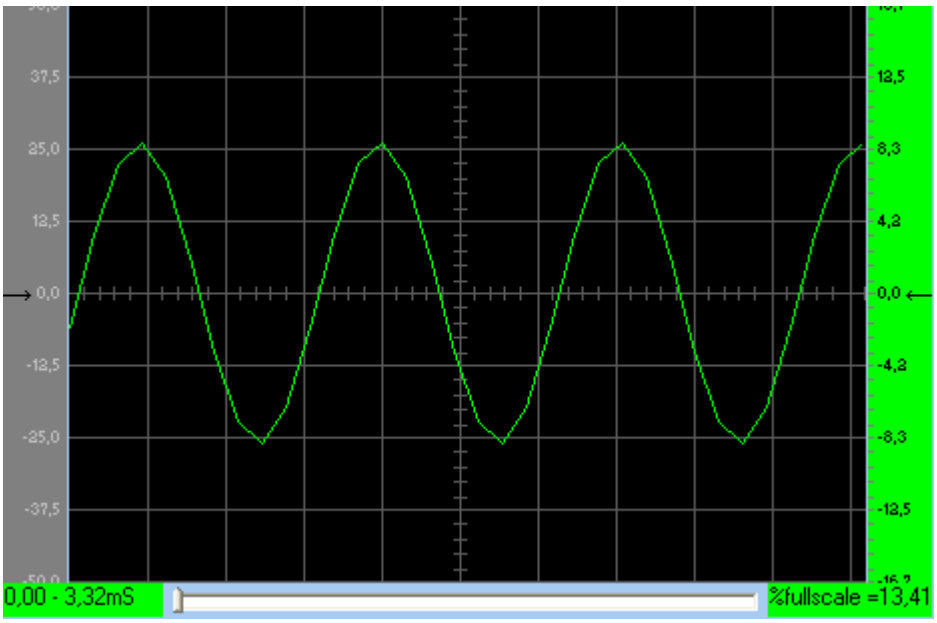


Figura 5: sinusoide 1000 Hz campionata a 10Khz

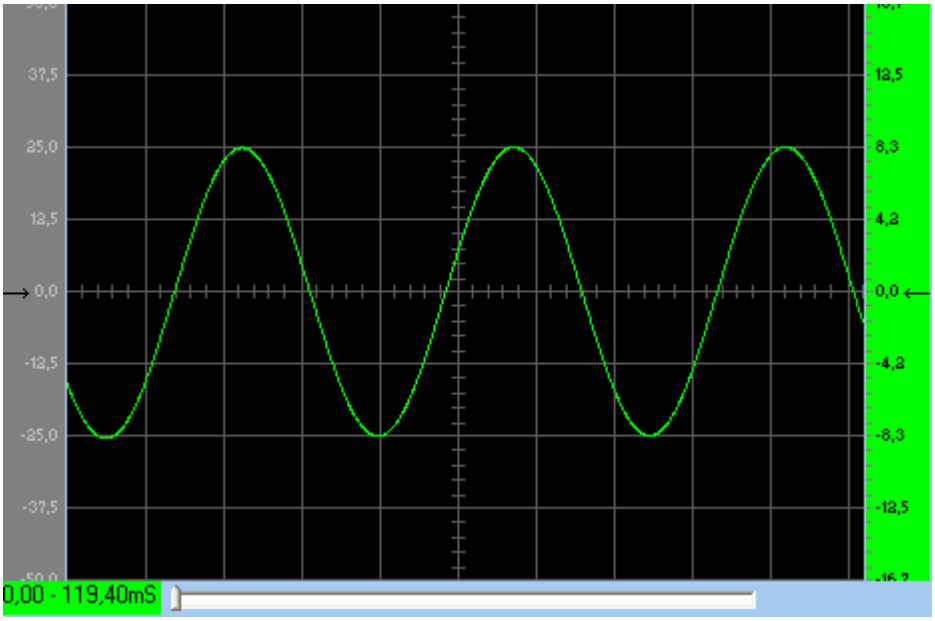


Figura 6: sinusoide 100Hz campionata a 10 kHz

La situazione cambia drasticamente all'approssimarsi degli estremi superiori della banda passante. Consideriamo il caso limite di un segnale a 5000 Hz: esso sarà disegnato avendo a disposizione solo due punti per ciclo, del tutto insufficienti per poter disegnare direttamente il segnale a video. Esso apparirà infatti come un'onda triangolare (figura 7). La situazione migliora drasticamente spuntando la checkbox "D/A" relativa al canale corrispondente (in questo caso il sinistro). A fronte di una maggiore complessità computazionale, il segnale originale viene ricostruito esaustivamente tramite l'algoritmo descritto nel capitolo settimo (figura 8).

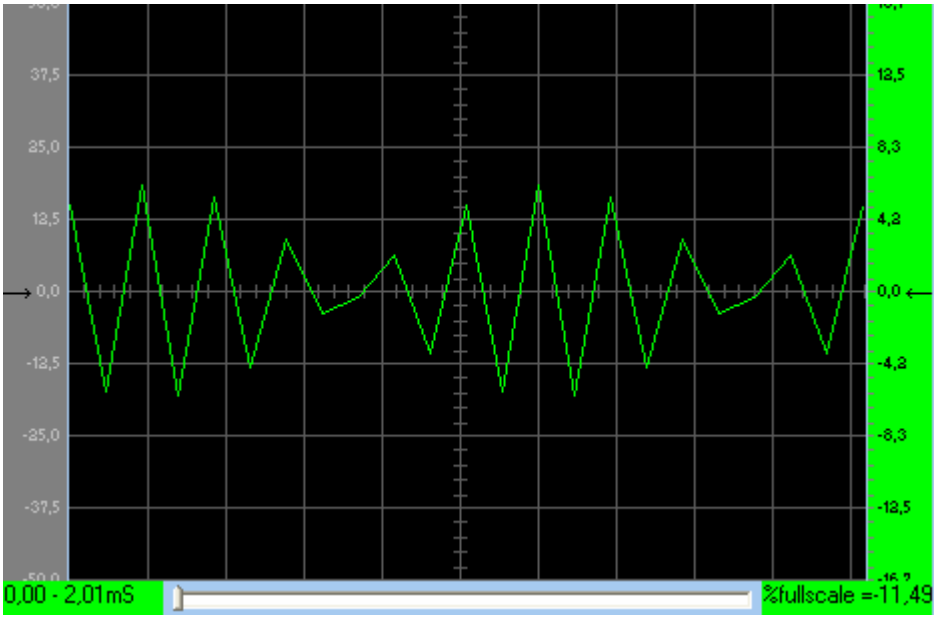


Figura 7: sinusoide a 5000 Hz campionata a 10 kHz

Se da un lato il teorema del campionamento ci insegna che le informazioni per ricostruire completamente il segnale analogico ci sono tutte, è ovvio che per far ciò bisogna applicare correttamente e completamente l'algoritmo. Cosa teoricamente fattibile, ma cosa praticamente troppo onerosa computazionalmente. Visual Analyser riesce tuttavia nell'intento.

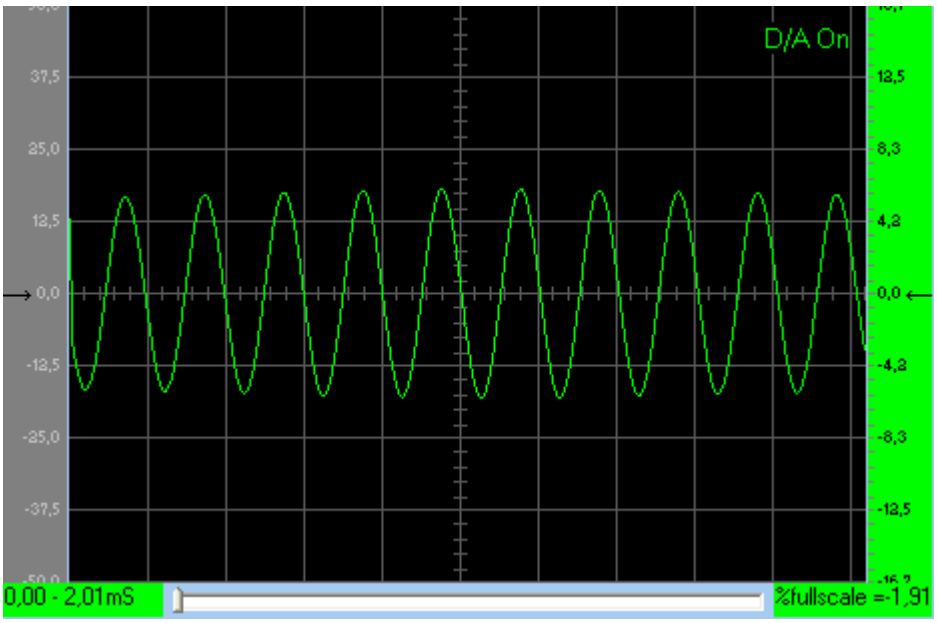


Figura 8: sinusoide a 5000 Hz campionata a 10Khz con conversione D/A

Un simile risultato, rarissimo in questa categoria di programmi, è stato ottenuto tramite l'aggiunta di un thread dedicato e l'ottimizzazione (ed approssimazione) delle formule necessarie per il calcolo del segnale originale (ossia il teorema del campionamento), mantenendo un accettabile grado qualità del segnale ed esecuzione in tempo reale. Come detto i dettagli sono ampiamente descritti nel capitolo quarto ed il codice è visibile nel capitolo terzo, oltre all'appendice con i listati delle classi più significative.

Per consentire la massima flessibilità d'uso è possibile abilitare manualmente la funzione "conversione D/A" direttamente dal pannello principale posto alla destra dell'oscilloscopio, consentendo, laddove non necessario (per esempio nel caso dei 100 Hz campionati a 10 kHz di figura 6) di risparmiare una notevole dose di risorse di calcolo da lasciare così disponibili a beneficio di altre funzioni eventualmente attivate. Invero, come avremo modo di vedere nei paragrafi successivi, anche quando la funzione D/A è attivata Visual Analyser cerca di non usarla se un apposito algoritmo di controllo si rende conto che il risultato a video non cambierebbe. In taluni casi tuttavia, se non si vuole correre il rischio di avere perdita di dati, è opportuno disattivare del tutto questa opzione.

Capitolo 2

Architettura software

2.0 Introduzione

In questa sezione descriveremo l' architettura software del programma e la filosofia generale di progetto. Iniziamo descrivendo la tecnica usata per l'acquisizione dei dati. Come spiegato esaustivamente nella sezione dedicata alla scheda sonora, i segnali applicati ad essa vengono nell'ordine amplificati, miscelati (se richiesto) e convertiti in segnali numerici secondo le modalità stabilite dall'utente (in termini di frequenza di campionamento, numero di bit, codifica, numero di canali e stato del mixer interno). Successivamente, essi vengono impacchettati a blocchi di "n" campioni, le cui dimensioni sono stabilite anch'esse dall'utente (ossia definibili come impostazioni del programma).

Due sono gli elementi software che caratterizzano Visual Analyser; la gestione della scheda audio (inizializzazione e prelevamento dei campioni) ed i thread che ne sfruttano le informazioni acquisite per simulare i vari strumenti.

2.1 Inizializzazione della scheda audio

Il settaggio dei parametri desiderati per la scheda sonora avviene tramite il passaggio della struttura predefinita di tipo "WaveFormat" i cui campi vendono riempiti dal costruttore "format" nella seguente maniera:

WaveFormat format(nChannels, SamplesPerSec, bitsPerSample); (2.f.1)

dove:

nChannels = numero dei canali (es. 2);
cSamplesPerSec = numero di campioni per secondo (es. 44100);
bitsPerSample = numero di bit utilizzati per rappresentare un campione;

La struttura viene passata alla API :

waveInOpen ( ) (2.f.2)

che inizializza ("apre") il dispositivo audio passandogli i parametri definiti nella struttura di tipo "WaveFormat". In effetti, la struttura originale prevista da Windows si chiama *WAVEFORMATEXTENSIBLE*, per motivi di praticità è stata incapsulata nella classe WaveFormat (vedi sezione listati). Ancora è stata definita ex-novo una ulteriore classe denominata "WaveInDevice" che incapsula in maniera efficiente le principali API relative alla gestione della scheda sonora e riguardanti la sezione d'ingresso. Pertanto la chiamata è la seguente:

waveInDevice.Open (device, format, event); (2.f.3)

Dove:

waveInDevice = istanza della classe WaveInDevice;
format = variabile di tipo WaveFormat;
event = semaforo bloccante (v. avanti);
device = scheda sonora selezionata.

Non appena la scheda è stata inizializzata, bisogna attivare il meccanismo di trasferimento dei dati acquisiti. Il pacchetto di dati viene "rilasciato" dall'hardware (e firmware) della scheda sonora e dal suo "device driver" in regioni di memoria predefinite, previo passaggio del relativo puntatore ad esse. Ossia, da programma si comunica alle routine di gestione della scheda sonora l'indirizzo di ram in cui trasferire i campioni acquisiti a pacchetti di dimensioni prefissate (tramite API di sistema). Inoltre si comunica anche il *numero* di buffer da rilasciare in regioni consecutive di memoria (nota: le regioni di memoria identificate dal puntatore passato devono tassativamente essere allocate dinamicamente da programma, pena il "crash" disastroso del programma e talvolta di tutto il sistema). In questo modi si implementa automaticamente una coda di tipo circolare, gestita in perfetto parallelismo hardware (rispetto alla CPU) dalla stessa scheda sonora.

Per comunicare alla scheda sonora la regione di memoria cui scaricare i campioni acquisiti, si deve riempire una struttura di tipo *WAVEHDR* (prevista da Windows) così definita:

```
typedef struct
 {
   LPSTR  lpData;
   DWORD  dwBufferLength;
   DWORD  dwBytesRecorded;
   DWORD  dwUser;
   DWORD  dwFlags;
   DWORD  dwLoops;
```

```
   struct wavehdr_tag * lpNext;
   DWORD  reserved;
} WAVEHDR;
```

Listato 1

Invero, una apposita classe che "eredita" dalla struttura originale di Windows è stata definita ma per semplicità di trattazione eviteremo di fare ad essa riferimento (vedere appendice listati). Dovendo poi definire un set di buffer, per poter implementare la coda circolare, definiremo un vettore "header[]" che contiene NUM_BUF strutture di tipo WAVEHDR. Il codice che inizializza e utilizza il vettore *header* è il seguente e si trova nella classe *Recorder*:

```
   int cbBuf = cSamples * nChannels * bitsPerSample/8;
   int pBuf = new char [cbBuf * NUM_BUF * nChannels];

   for ( int i = 0; i < NUM_BUF; i++ )
   {
      header[i].lpData = &pBuf [i * cbBuf];
      header[i].dwBufferLength = cbBuf;
      header[i].dwFlags = 0;
      header[i].dwLoops = 0;
      waveInDevice.Prepare (&header[i]);
      waveInDevice.SendBuffer (&header [i]);
   }
   waveInDevice.Start();
```

Listato 2

Dove si può notare la chiamata al metodo "Prepare" della classe *WaveInDevice* che chiama l'API di sistema:

(2.f.4) waveInPrepareHeader(handle, pHeader, sizeof(WAVEHDR));

e la successiva *SendBuffer* che chiama:

(2.f.5) waveInAddBuffer (handle, pHeader, sizeof(WAVEHDR));

dove pHeader è un puntatore al vettore header[], "handle" è l'handle della scheda Sonora selezionata.

Esse servono a predisporre le variabili del caso e comunicare alla scheda sonora i dati riguardanti la regione di memoria in ram in cui depositare i campioni prelevati.

La filosofia generale del programma è quindi la seguente: i dati vengono prelevati a gruppi di "n" campioni ed elaborati nel tempo che intercorre tra il prelevamento di due buffer successivi, dove obbligatoriamente il numero "n" deve essere una potenza di due. Questo per le esigenze dell'algoritmo utilizzato per il calcolo della trasformata di Fourier, che, come descritto nel capitolo quarto, è la ben nota FFT (Fast Fourier Transform). In altre parole ancora, Visual Analyser preleva i buffer dei campioni non appena disponibili (se tutto va bene il prelievo avviene un buffer alla volta) e nel tempo "libero" che rimane tra l'acquisizione di due buffer consecutivi vengono effettuate tutte le restanti operazioni necessarie per le funzioni richieste (calcolo della FFT, disegno a video, calcoli di funzioni aggiuntive quali frequenzimetro, voltmetro, calcolo di distorsione etc.). Nel caso della funzione generatore di funzioni tutto avviene al contrario, ma essenzialmente nella medesima maniera: i buffer di campioni vengono in questo caso generati dal programma, e passati alla scheda sonora sempre tramite il meccanismo dei buffer. La scheda sonora interpreterà questi campioni secondo quanto indicato in fase di setup; essi saranno considerati come dati ottenuti da un campionamento di una forma d'onda

analogica (a banda limitata). Verranno quindi passati all'apposito circuito di conversione Digitale/Analogico, successivamente al mixer ed infine all'amplificatore finale, alle cui uscite avremo disponibile la forma d'onda generata. Invero, le schede sonore presentano due tipi di uscite, come descritto nel capitolo XX. Una amplificata in potenza, con bassissima impedenza d'uscita (tipicamente 4..8 ohm) e destinata a pilotare gli altoparlanti. L'altra solo preamplificata, e normalmente destinata a pilotare sistemi alta fedeltà o registratori audio o comunque sistemi già amplificati in potenza.

2.2 I thread utilizzati

Per implementare il meccanismo descritto sommariamente nelle righe precedenti si è sfruttato il modello di programmazione a "multithread" (cfr. capitolo quinto), definendo i seguenti sette thread :

- Principale (main vcl thread) (paragrafo 2.2.1)
- Acquisizione dati (paragrafo 2.2.1)
- Calcolo D/A canale sinistro e destro (due thread) (paragrafo 2.2.2)
- Frequenzimetro (paragrafo 2.2.3)
- Generatore di funzioni (paragrafo 2.2.4)
- Calcolo conversione D/A in finestra capture (paragrafo 2.2.5)

2.2.1 Thread acquisizione dati e main vcl thread

Il thread di acquisizione dei dati non viene istanziato sino alla messa in "on" del programma. A meno che non siano state attivate funzioni come generatore di funzioni e frequenzimetro , sino a che il programma non verrà avviato tramite tasto di "on" l'unico thread in esecuzione effettiva è il "main vcl

thread" che è in pratica il thread automaticamente generato alla partenza del programma e serve a gestire essenzialmente l'interfaccia utente. Esso provvede a gestire la risposta ai vari "controlli" presenti sulla finestra principale (per esempio i "bottoni", le checkbox, listbox eccetera). Non appena il programma viene messo in "on" viene istanziato il thread di tipo "*Getdata*" che per semplicità nel disegno abbiamo definito come "GET" e successivamente posto nello stato di "esecuzione". (disegno da fare….)

Prima di descrivere esaustivamente il funzionamento del metodo "execute" del thread *Getdata* accenniamo ad altre due classi, fondamentali per la corretta comprensione di Visual Analyser in genere, ed in particolare di questo thread. La prima di esse è chiamata "*Recorder*" e serve ad acquisire fisicamente i campioni del segnale, che dalla scheda sonora vengono trasferiti nelle regioni di memoria ram prestabilite. Ad essa in pratica viene passato il puntatore alla sezione di ram pre-allocata allo scopo. Un ulteriore classe, definita "*SampleIter*" serve a gestire in maniera universale i campioni acquisiti; ad essa in pratica viene passato il puntatore all'oggetto di tipo *Recorder.* La classe *SampleIter* è usata per definire dei metodi "universali" indipendentemente dalla classe *Recorder* effettivamente istanziata. Quest'ultima infatti può appartenere a differenti classi, tutte discendenti dall'oggetto base di tipo "*Recorder*":

- RecorderM8 (acquisisce campioni mono a 8 bit)
- RecorderS8 (acquisisce campioni stereo a 8 bit)
- RecorderM16 (acquisisce campioni mono a 16 bit)
- RecorderS16 (acquisisce campioni strereo a 16 bit)
- RecorderM24 (acquisisce campioni mono a 24 bit)
- RecorderS24 (acquisisce campioni stereo a 24 bit)
- RecorderM32 (acquisisce campioni mono a 32 bit)
- RecorderS32 ((acquisisce campioni stereo a 32 bit)

Essi in pratica differiscono tra loro esclusivamente nei metodi che aquisicono i campioni dai buffer in ram (perché campioni a 8 bit sono disposti in maniera diversa rispetto a quelli a 16 ed ancora differenti se stereo o mono). L'idea è quindi quella di definire tali metodi come virtuali; l'oggetto *Recorder(8,16,M,S....)* verrà passato come puntatore all'oggetto *SampleIter* che sarà dotato di analoghi metodi di acquisizione dati (ossia con gli stessi nomi) ma che richiameranno a loro volta quelli dell'oggetto *Recorder* passato in fase di allocazione. Ma essendo questi virtuali, saranno richiamati quelli dell'oggetto *Recorder(8,16,M,S....)* effettivamente istanziato, ossia indipendentemente se di tipo *RecorderM16* o *RecorderS24* eccetera. Questo per definizione di ereditarietà e metodi virtuali (il tipo del parametro di passaggio è un puntatore alla classe base *Recorder*, che quindi consente di passare anche i puntatori a classi derivate)

Per fissare le idee, ecco a titolo esemplificativo la definizione dei metodi della classe *Recorder* "base" relativamente all'acquisizione del canale destro e sinistro:

(2.f.6)

```
virtual int __fastcall  GetSampleL (char *pBuf, int i) const = 0;
virtual int __fastcall  GetSampleR (char *pBuf, int i) const = 0;
```

I relativi oggetti discendenti avranno ridefiniti gli stessi metodi con diversa implementazione (per esempio per l'acquisizione di campioni a 16 o 24 bit nel caso rispettivamente di RecorderS16 e RecorderS24). La definizione degli analoghi metodi nell'oggetto *SampleIter* sarà:

(2.f.7)

```
Int __fastcall GetSampleL () const
        { return recorder -> GetSampleL(pBuffer, iCur);}
```

int **__fastcall** GetSampleR () const { **return** recorder -> GetSampleR(pBuffer, iCur);}

Quindi, chiamando il metodo "*GetSampleL*" verrà chiamato il metodo "*GetSampleL*" dell'oggetto *Recorder(8,16,M,S....).* In questo modo verrà usato sempre lo stesso oggetto di tipo *SampleIter* quale che sia l'oggetto *Recorder* usato.

Adesso abbiamo sufficienti informazioni per poter descrivere il funzionamento del thread *GetData* che costituisce il motore principale di tutto il programma e meglio ne definisce la filosofia di base.

Il metodo "*Execute*" del thread è costituito dal seguente codice (opportunamente semplificato e commentato per comodità di trattazione):

```
void __fastcall GetData::Execute()
{
  Counter = 0;
  //numero massimo di buffer previsti
  int NUM_BUF = GetSample -> GetNumBuf();

  if ( IsStarted )
  try
  {
       for (;;)
       {
         START_TIME_WAIT = GetTickCount();
         if (Terminated) break;

// aspetta che la scheda sonora abbia finito di riempire almeno  un buffer
         event.Wait();
```

```
        TIME_PLOT_START = GetTickCount();
        TIME_WAIT = TIME_PLOT_START - START_TIME_WAIT;

   // itera tra i tutti i NUM_BUF  buffer previsti nella coda circolare
        for (int iCurr  =  0; iCurr  <  NUM_BUF; iCurr++)
        {
  // controlla che il buffer iCurr sia arrivato completamente, altrimenti
  // salta un giro
          if (GetSample -> IsBufferDone(iCurr))
          {
 // viene chiamato UNPREPARE;.il buffer è pronto per essere copiato
// rapidamente
           GetSample -> BufferCanRead(iCurr);

 //  passa all'oggetto "iter" il puntatore all'oggetto recorder
 // l'oggetto iter sarà successivamente utilizzato dall'oggetto FFT
 // per copiare i campioni nei suoi buffer interni
           Main -> iter.SetRec(GetSample);  //ibuf settato a iCurr

 // viene ora acquisito il lock
 // previene accesso contemporaneo con altri thread
           Lock -> Acquire();
            try
            {
 //Copia negli oggetti che eseguono l'FFT  i campioni acquisiti
              switch(Channel)
              {
                case 0:  //A
                  Main -> FFTleft  -> CopyIn( Main -> Iter );
                break;
```

```
                case 1:  //B
                  Main -> FFTright -> CopyIn( Main -> Iter );
                break;
                --
                --
               }
 // effettua la  trasformata di fourier  e disegna a video
                Main -> Plot();
// tramite questo metodo viene effettuato il “prepare” del buffer appena
// copiato, rendendolo nuovamente disponibile
// per la copia di nuovi dati
                 GetSample -> BufferDone(iCurr);
               }
               __finally
               {
                Lock -> Release();
               }
          }  //IsBufferdone
          else
          {
           if (Main -> GetCount() < -(NUM_BUF * 3) )
           {
             Terminate();
             MessageBox (0, "Error”, "Visual Analyser", MB_OK);
           }
           else
             Main -> ResetCount();

             if  ( Terminated )  break;
          }
          TIME_OVER = GetTickCount();
```

```
        if ((TIME_OVER - TIME_PLOT_START)
                        > (Main -> TEMPO_RICHIESTO << 1) break;
        }
       } // for iCurr
       Synchronize(&PlotTime);
       Application -> ProcessMessages();
      }
  }
  __finally
  {
   // operazioni conclusive….
  }
}
```

Listato 3

Allo scopo di ridurre quanto più possibile l'overhead dovuto alla commutazione di contesto tra thread (per definizione di thread già piuttosto ridotto), si è cercato di limitare quanto più possibile il proliferare di questi. Generalmente tutti i programmi simili a Visual Analyser utilizzano un thread per l'acquisizione dei campioni, un thread per il disegno a video ed un thread per il calcolo dei dati. In Visual analyser si è sperimentalmente verificato che l'uso di due soli thread offre prestazioni generalmente migliori. Analizzando il codice sorgente del metodo *Execute*, si osserva come esso sia costituito da un ciclo infinito all'interno del quale è presente un ulteriore ciclo for che itera tra i NUM_BUF buffer previsti. In assenza di particolari colli di bottiglia, sarà presente al più un solo buffer. Ad ogni giro viene verificata la presenza del buffer corrente (iCurr), ed in caso affermativo viene chiamato il metodo *BufferCanRead* che effettua l'UNPREPARE del buffer iCurr. In tal modo esso è dichiarato come non più di proprietà della scheda sonora e può essere al

limite liberato e/o utilizzato. Infatti, né viene rapidamente copiato il contenuto nelle strutture interne degli oggetti di tipo Fft (FFTleft e FFTright) che contengono la maggior parte delle routine per il calcolo della trasformata di Fourier, filtri digitali, analisi in terzi d'ottava e finestre di smooth. Subito prima di copiare i dati viene passato all'oggetto *iter* di tipo *SampleIter* il puntatore all'oggetto di tipo *Recorder*. Effettuata la copia negli oggetti *FFTleft* e *FFTright* viene finalmente chiamato il metodo *plot* dell'oggetto *Main*, che costituisce in pratica il "core" del "main vcl thread". Esso effettua la trasformata di Fourier, ed il disegno a video dei dati, sia dell'oscilloscopio che dell'analizzatore di spettro, oltre al distorsiometro e voltmetro. Ed altre importantissime funzioni descritte nel paragrafo 2.2.1.1.

A questo punto il ciclo ricomincia, sino all'esaurimento di tutti i buffer potenziali. Esaurito lo "scanning" dei vari buffer il ciclo principale "infinito" che caratterizza il thread *GetData* ricomincia daccapo. Si noti come in testa al programma esista la chiamata:

event.Wait() *(2.f.8)*

event è una istanza della classe *Event* che serve ad implementare un semaforo binario ed un meccanismo di attesa su di esso capace di bloccare il thread che lo sta eseguendo. Questo meccanismo è usato per comunicare al thread *GetData* l'avvenuto completamento di almeno un buffer di dati da parte della scheda sonora. Esso utilizza direttamente API di sistema, il cui codice è il seguente:

```
class Event
{
public:
   // costruttore
   Event ()
```

```
    {
       // parte settato a rosso
       _handle = CreateEvent (0, FALSE, FALSE, 0);
      // per default è posto a infinito, ma è possible a runtime ridefinire
      // la politica tramite il metodo SetTimeOut
       _TimeOut = INFINITE;
    }
    // distruttore
   ~Event () {CloseHandle (_handle);}

   void SetTimeOut (__int64 TimeOut) {_TimeOut = TimeOut;}

   // mette a verde
   void Release () { SetEvent (_handle); }

   void Wait ()
   {
      // Aspetta sino a che diventa verde
      // con modalità definite dalla variabile TimeOut
      // se TimeOut = INFINITE aspetta a oltranza sino a
     //che non diventa verde
      WaitForSingleObject(_handle, _TimeOut);
   }
   operator HANDLE () { return _handle; }
private:
   HANDLE _handle;
   __int64    _TimeOut;
};
```

Listato 4

Il funzionamento si basa sull'API "*CreateEvent*" e "*WaitForSingleObject*". La prima crea un semaforo binario che restituisce un handle successivamente passato come argomento alla seconda, più una variabile "TimeOut" che definisce il tempo massimo cui attendere il verificarsi dell'evento (al limite infinito). Lo stesso handle viene passato nelle variabili di inizializzazione della scheda sonora, per la precisione nella (2.f.3) qui riportata per comodità:

waveInDevice.Open (device, format, event); (2.f.3)

In tal modo la scheda sonora segnala tramite il semaforo binario *event* l'avvenuto completamento di un buffer di dati. Il thread che esegue la event.wait è nel frattempo bloccato sul semaforo (è in stato di wait) e libera risorse di calcolo. Il timeout è settato ad un valore minore di "infinito" per evitare blocchi del programma in caso di failure della scheda sonora.

2.2.1.1 Il metodo plot

Il metodo plot fa parte del "main vcl thread", ed è praticamente una delle routine fondamentali per la simulazione degli strumenti, sia perché effettua a video il disegno dei dati e principalmente perché invoca (se richieste) la maggior parte delle routine di trasformazione ed analisi. Il codice della routine è particolarmente lungo, pertanto è opportuno limitarsi alla descrizione dei passi eseguiti e rimandare all'appendice un eventuale riscontro con il codice.

La prima considerazione da fare è relativa al fatto che, come si può evincere facilmente e come chiaramente descritto nelle sezioni precedenti, questa routine viene chiamata direttamente dal thread di acquisizione dati. Questo potrebbe sembrare un controsenso perché è una routine apparentemente "time-consuming" e che sarebbe conveniente tenere fuori da cicli che per loro natura dovrebbero essere estremamente veloci (come appunto quello del thread *Getdata* da cui è invocata). Ma, data la natura delle modalità di scambio dati, si

è preferito non aggiungere un ulteriore thread solo per gestire il metodo *plot*. Infatti, i campioni arrivano a intervalli di tempo ben definiti, e comunque bufferizzati in una coda circolare autonoma. L'unico intervallo di tempo in cui possono essere utilizzati è quello tra un buffer e l'altro, in cui il thread Getdata sarebbe comunque fermo. Anche se i campioni arrivassero prima del tempo necessario per effettuare calcoli & disegno essi sarebbero comunque (alla fine) persi. E con un problema di sincronizzazione in più. Inoltre, i tempi di calcolo e disegno sono sempre molto minori del tempo che intercorre tra due buffer successivi, infatti l'uso pratico ha dimostrato che non si verifica mai perdita di dati anche su macchine "lente".

Chiariti a grandi linee i motivi di queste scelte architetturali, passiamo alla descrizione del metodo *plot*. Esso effettua essenzialmente le seguenti operazioni, premettendo che ogni operazione svolta è preceduta da una selezione (ossia numero dei canali e modalità di disegno, per esempio se a linee, barre o somma di canali eccetera) ed è comunque relativa ad un solo buffer:

1. calcolo dei livelli in dB per le barre indicatrici. Il calcolo è effettuato individuando il campione a maggior ampiezza, applicando la relazione dBr = 20*Log(max/zero) dove per zero si è scelto il massimo numero rappresentabile con il numero intero utilizzato (es. 16 bit = 65536, 8 bit = 256 etc.);
2. memorizzazione del buffer di pre-inizializzazione per la conversione Digitale/analogico (v. capitolo quarto);
3. se il voltmetro è attivato, visualizzazione della tensione su finestra separata (rappresentazione in volt del campione a massima ampiezza, vedere sezione"calibrazione");
4. chiama il metodo "*Transform*" degli oggetti FFT per effettuare la trasformata di Fourier del canale selezionato (o di entrambi);

5. disegno a video, invocando i metodi dell'oggetto "*ViewWave*" appartenente alla classe "*View*" (vedi appendice) che fornisce, definiti "in proprio" tutti i metodi e le classi necessarie per il disegno a video nelle varie modalità (lineare, logaritmico, linee, barre, spettro, oscilloscopio eccetera). Inoltre contiene la parte real-time della conversione digitale analogico dell'oscilloscopio. Ancora, si occupa del disegno del reticolo, delle barre di indicazione dB e dell'operazione automatica di trascinamento della scale dell'analizzatore di spettro (con mouse) . NOTA: sono richiamate, previo incapsulamento in semplici classi, direttamente API di sistema;
6. se richiesta effettua l'operazione di media aritmetica su "n" buffer precedentemente acquisiti (funzione "average");
7. se la funzione è abilitata invoca i metodi dell'oggetto *PeakFreq* della classe *TPeakFreq*, per visualizzare la frequenza (a massimo valore di ampiezza) del segnale in finestra separata (v capitolo sesto e appendice per listato);
8. invoca il metodo GetTHD() e GetTHDNoise() per il calcolo della THD, se richiesta. Sono metodi forniti dallo stesso oggetto FFT;
9. se abilitata la funzione "capture scope" richiama le complesse routine per la cattura dei campioni dell'oscilloscopio secondo le modalità definite dalla finestra di Settings/Capture ;
10. se abilitate (default) invoca i metodi dell'oggetto FFT per il calcolo delle finestre di smooth;
11. se abilitaitainvoca i metodi dell'oggetto FFT relativi ai filtri digitali (vedi sezione finestra di settings in capitolo sesto);

2.2.2 Thread conversione analogico-digitale

Come ampiamente descritto nella sezione dedicata al teorema di Shannon ad alla conversione digitale-analogico, il thread dedicato alla funzione di ricostruzione dei campioni originali serve a calcolare "una-tantum", al verificarsi di eventi poco frequenti, i coefficienti relativi alla funzione sinc(x) = sin(x)/x che vengono convenientemente memorizzati in una matrice per poter essere rapidamente acceduti durante le funzioni di conversione in tempo reale. In pratica la conversione digitale-analogico avviene all'interno del thread *GetData*, descritto nella sezione 2.2.1, ad opera del metodo *RestoreTimeDomain* il quale però si basa sulla matrice precalcolata dal thread *TDAprecompute* di cui normalmente esistono due istanze, una per ogni canale. Esso si incarica di calcolare una matrice il cui tempo di calcolo normalmente rischia di non rientrare nell'intervallo di tempo a disposizione tra l'arrivo di due buffer successivi. L'idea è quindi quella di demandare questo compito oneroso ad un task a bassa priorità, che comunque esegua i calcoli in tempi dell'ordine di poche centinaia di millisecondi od al più qualche secondo. Durante tale periodo la forma d'onda verrà visualizzata normalmente ma senza la conversione digitale-analogico. Gli eventi che costringono ad effettuare il ricalcolo della matrice (chiamata SINC) sono:

1. cambio di frequenza di campionamento;
2. variazioni delle dimensioni della finestra dell'oscilloscopio, quindi aumento o diminuzione del numero di punti da precalcolare;
3. variazione del parametro "time-division" : la variazione della "risoluzione" sull'asse delle X dell'oscilloscopio costringe a ricalcolare i punti;
4. prima messa in "on" del programma;

Il meccanismo che consente di attivare il ricalcolo si basa su una variabile booleana ed è inserito nel ciclo di disegno a video dell'oscilloscopio, ossia contenuto nel metodo "plot" (cfr 2.2.1) ed è il seguente:

```
if ((PreComputeA) && (Form1 -> Acquisisci) && (PreTaskA -> Computed))
{
    // passa i parametri al Thread
    PreTaskA -> PreComputeSINC( &fft,
                                Plott,
                                from, 1, -t_d,
                               &SINCA, &XallocA, &PreA,
                               &prePointsA);
    // …lo risveglia
     PreTaskA -> Resume();
 // setta a false per indicare che non c'è bisogno di precalcolo al prossimo giro
     PreComputeA = false;
 }
// Se il task ha effettuato il ricalcolo può effettuare la conversione effettiva
if (PreTaskA -> Computed)
     RestoreTimeDomain( fft, Plott,
                        from, 1,
                       -t_d, &SINCA,
                       &PreA, prePointsA );
```

Listato 5

Al verificarsi degli eventi di cui ai punti 1,2,3,4 la variabile *PreComputeA* verrà settata al valore true, innescando il precalcolo ad opera del thread *TDAprecompute* e non appena completata (variabile "Computed" posta al valore true) verrà effettivamente innescata la conversione digitale analogica ad opera del metodo *RestoreTimeDomain.*

Il thread *TDAprecompute* è definito nella seguente maniera:

```
class TDAprecompute : public TThread
{
private:
  Fft    *Sample;
  int    To;
  int    from;
  int    start_trig;
  int    t_d;
  float  *(**SINC);
  int    *Xalloc;
  int    **Pre;
  int    *prePoints;
  void    PreComp();
protected:
  void __fastcall Execute();
public:
  void __fastcall TDAprecompute(bool CreateSuspended);
  void PreComputeSINC(….);
  void DePreComputeSINC();
  bool   Computed;
};
```

Listato 6

Come già messo in evidenza dai commenti del listato 5, il metodo *PreComputeSINC* (di cui nel listato 6 sono stati omessi i parametri per motivi di impaginazione e perché già presenti nel listato 5) serve a comunicare i parametri fondamentali al thread; il "resume" del task manderà in esecuzione il metodo "Execute" del thread che essenzialmente si riduce all'esecuzione del

metodo privato *PreComp()* (previa esecuzione di alcuni controlli) che consiste nel seguente codice:

```
void TDAprecompute::PreComp()
{

 float x,A,B,C,D;
 long double tt = 0;
 int  Local;

 // To = punti da visualizzare sullo schermo (larghezza schermo)
 // Points = tutti i punti acquisiti

 double SR      =  Sample->GetSampleRate();
 A              =  PI*SR;
 double T       =  1.0/SR;
 double PASS  =  T / ((double)t_d);
 B              = 0;

 if (Sample->Points() < To)
 {
  Local = Sample->Points();
  *prePoints = Sample->Points();
 }
 else
 {
  Local = To;
  *prePoints = To;
 }

 Local += *prePoints;
```

```
To    += *prePoints;


DePreComputeSINC();

try
{
 *SINC = new float* [To + 1];
 for (int i = 0; i < To + 1; i++) (*SINC)[i] = new float [Local];
 *Xalloc = To + 1;
 *Pre = new int[*prePoints];

 for (int t = 0; t < *prePoints; t++ ) (*Pre)[t] = 0;
}
catch(...)
{

  MessageBox (0, "Cannot allocate memory for FULL D/A conversion.",
                      "Visual Analyser",
                       MB_OK |
                       MB_ICONWARNING |
                       MB_SYSTEMMODAL);

  *SINC = 0;
  *Xalloc = 0;
  *Pre = 0;
}


for (int t = 1; t <= To; t++ )    // punti da graficare sullo schermo
{
```

```
    for (int k = 0; k < Local; k++ )
    // punti usati per ricostruire la forma d'onda
    {
      B = (float)k * T;
      D = A * ( tt - B );
      if (D==0)
        (*SINC)[t][k] = 1;
      else
        (*SINC)[t][k] = sin(D)/D;
    }
  tt += PASS;
  if  ( Terminated )  break;
 }
}
```

Listato 7

Infine, ecco riportato il codice del metodo *RestoreTimeDomain* :

```
void ViewWave::RestoreTimeDomain(Fft const &Sample,
                                 int To, int from,
                                 int start_trig, int t_d,
                                 float *(**SINC),
                                 int *Pre[], int prePoints)
{

 if (!Main -> Acquisisci) return;
 float x = 0;
 int *temp = new int[To];
 int  Local;

 // To = punti da visualizzare sullo schermo (larghezza schermo)
```

```
// Points = tutti i punti acquisiti

if (Points < To)
  Local = Points;
else
  Local = To;



// per velocizzare il ciclo
int incr = from + start_trig;
int prePincr = -1 - prePoints;

for (int t = 1 + prePoints; t <= To + prePoints; t++ )
 // punti da graficare sullo schermo
{
   for (int k = 0; k < prePoints ; k++ )
   {
     x += (float)((*Pre)[k]) * ((*SINC)[t][k]);
   }
   // punti usati per ricostruire la forma d'onda
   for (int k = 0; k < Local; k++ )
   {
     x += (float)(Sample.tape[k + incr]) * ((*SINC)[t][k + prePoints]);
   }
        temp[t + prePincr] = (int)x;
        x = 0;
}

int diff = prePoints - incr;
if (incr < prePoints)
{
```

```
   // parte precedente
   memmove (*Pre, &Sample.tape[Points - diff ], diff * sizeof(int));
   // parte visibile
   memmove (&(*Pre)[diff], Sample.tape, incr * sizeof(int));
  }
  else
   memmove (*Pre, &Sample.tape[-diff], prePoints * sizeof(int));

  memmove (&Sample.tape[incr], temp, To * sizeof(int));
  Application -> ProcessMessages();
}
```

Listato 8

2.3 Il generatore di funzioni

Il generatore di funzioni è l'unico tra gli strumenti simulati che produce un segnale invece di acquisirlo. Ciononostante vengono impiegarti meccanismi analoghi, sebbene la direzione del flusso dei dati proceda in senso inverso. Inoltre esso è ovviamente escluso dal ciclo principale, ossia non ha bisogno del thread *Getdata* che può essere anche non istanziato (ossia, in parole povere, Visual Analyser può essere nello stato di "off") e nemmeno del metodo *plot*. Parimenti, il generatore di funzioni è provvisto di un suo thread dedicato ("*Generator*" della classe "*FuncGen*") che svolge la maggior parte delle funzioni, oltre ad una finestra specifica che gestisce l'interfaccia utente (oggetto "*GenFun*" della classe "*TGenFun*"). A tutti gli effetti il generatore di funzioni è un programma separato, che potrebbe tranquillamente vivere di vita propria. Tra le varie cose, avere un generatore di funzioni separato potrebbe introdurre un certo grado di vantaggio in termini di sfruttamento delle risorse; se da un lato si attiverebbe un processo ulteriore (mentre quando attivato

"dall'interno" si limiterebbe ad un thread aggiuntivo, quindi meno oneroso computazionalmente) dall'altro la gestione del multitasking di Windows sembra capace di gestire meglio questa situazione: si è rilevato sperimentalmente che, lanciando il generatore di funzioni come processo separato da Visual Analyser, si spuntano migliori tempi di esecuzione per entrambi i programmi.

Oltre ad essere un programma "autonomo" presenta anche due modalità di funzionamento assai diverse tra loro, sebbene una soffra di qualche limitazione pratica, a fronte di tempi di esecuzione enormemente migliori. In particolare abbiamo due modalità di funzionamento:

- In tempo reale
- Loop infinito

La prima è quella che consente la variazione "al volo" di ogni parametro che caratterizza la forma d'onda, persino quella relativa alle forme d'onda "custom". In altre parole, è possibile variare la frequenza generata, il tipo di forma d'onda, la fase, il numero dei canali utilizzati, il tutto in tempo reale (persino le singole componenti armoniche di una forma d'onda custom), solo con una leggera latenza dovuta al meccanismo della produzione dei dati "a blocchi" (ossia i dati vengono generati a gruppi di "n" byte che una volta passati alla scheda sonora sono "ininterrompibili"). Pertanto, se si varia la frequenza od un altro qualsiasi parametro, esso verrà applicato al buffer successivo, ossia dopo un tempo *al più* pari al tempo di esecuzione di un buffer. Le dimensioni del buffer sono comunque definibili in funzione della reattività desiderata e delle risorse di calcolo a disposizione. E' ovvio che buffer più piccoli implicano certamente una minore latenza ma richiedono macchine più reattive, pena introduzione di armoniche indesiderate (distorsione).

La seconda modalità è quella spesso sfruttata dalla stragrande maggioranza dei generatori di funzione che è possibile reperire in rete; i dati vengono preparati "fuori ciclo" una tantum, passati alla scheda sonora ed eseguiti da questa in un loop infinito. A questo punto le risorse di calcolo del PC saranno completamente svincolate e la scheda sonora opererà in perfetto parallelismo hardware. Il rovescio della medaglia è costituito da due problemi: non sarà possibile variare nessun parametro in tempo reale, eccetto il livello d'uscita; per variare i restanti parametri si dovrà spegnere il generatore, effettuare le variazioni, e riaccendere il generatore. Il secondo problema è relativo alle difficoltà di calcolo: la forma d'onda eseguita in loop deve raccordarsi perfettamente tra la fine di un buffer e l'inizio del successivo (per esempio partire da inizio ciclo e finire esattamente a fine ciclo). Altrimenti la forma d'onda complessiva risulterà distorta. Questo comporta una difficoltà ulteriore, se si ha l'esigenza di generare forme d'onda in contemporanea su due canali e di diversa frequenza. Infatti le schede sonore hanno come vincolo che i buffer dei due canali DEVONO essere delle medesime dimensioni. Diventa pertanto ancora più difficile far collimare inizio e fine di due buffer consecutivi: per esempio, una sinusoide a 100Hz di quanti punti necessita perché un dente di sega a 326,8 Hz ci stia dentro un numero intero di volte (e quindi inizino e finiscano entrambi per esempio con zero)? Nella maggior parte dei casi il problema è insolubile, e si è pertanto optato per impedire la generazione di forme d'onde differenti nei due canali se abilitata la modalità "loop". Ossia in pratica, sebbene prodotte indipendentemente da entrambi i canali, nella modalità loop la generazione è di tipo rigorosamente "monofonico".

L'utilità di questa modalità ridotta è altresì evidente quando si necessita di una forma d'onda che sia costante per buona parte del tempo (per esempio generazione di rumore bianco mentre si testa un device), avvantaggiandosi in maniera consistente del mancato consumo di risorse. L'esperienza di laboratorio ha portato alla constatazione che è di fatto la modalità usata più di frequente, pur se certamente meno flessibile

2.2.3 La modalità in tempo reale

La modalità in tempo reale fa uso del thread citato nel paragrafo 2.3 (classe "*FuncGen*") la cui istanza utilizzata nel programma si chiama "*Generator*". L'idea alla base di questo thread è relativamente semplice. Per generare i campioni e consentire una qual certa reattività in tempo "quasi reale" nella risposta alla variazione dei parametri (ed ovviare pertanto a tutti i problemi di calcolo "statico" descritti nel paragrafo 2.3) si è optato per un calcolo dei buffer "al volo" in un ciclo infinito. Esso è realizzato con il thread di tipo "*FuncGen*" il cui metodo *Execute* è il seguente:

```
void __fastcall FuncGen::Execute()
{

 LedOff();
 IsTerminated = false;
 if (ERR)
 {
   IsTerminated = true;
   return;
 }
 for (;;)
 {
   if (IsBufferDone())
   {
     LedOn();
     waveOutUnprepareHeader(WaveHandle,
                            &Header [iBuf],
                            sizeof(WaveHeader));
     GenerateFunction(DatiGeneratore.WaveFunctL,
```

```
                        DatiGeneratore.WaveFunctR);
      BufferDone();
    }

   if (Terminated) break;

  }
  waveOutReset(WaveHandle);
  waveOutClose(WaveHandle);
  delete [] pBuf;
   IsTerminated = true;
}
```

Listato 9

Il funzionamento è relativamente semplice; in fase di inizializzazione (nel costruttore del thread) vengono preparati *NBUFF* buffer (in numero definibile da finestra di settaggio del generatore di funzioni, vedi capitolo sesto) nella seguente maniera:

```
   for (int i = 0; i < NBUFF; i++ )
   {
      iBuf = i;
      GenerateFunction(DatiGeneratore.WaveFunctL,
                        DatiGeneratore.WaveFunctR);
      Header[i].lpData = &pBuf [i * Count];
      Header[i].dwBufferLength = Count;
      Header[i].dwFlags = 0;
      Header[i].dwLoops = 0;
      res = waveOutPrepareHeader( WaveHandle,
                                  &Header [i],
                                  sizeof(WaveHeader));
```

```
// controllo errore su variabile res
      res = waveOutWrite(WaveHandle, &Header [i], sizeof(WaveHeader));
// controllo errore su variabile res
   }
```

Listato 10

Come si vede l'inizializzazione avviene con le medesime modalità utilizzate per la lettura dei buffer (negli strumenti di acquisizione); ossia è presente un "pool di buffer" di tipo predefinito (*WAVEHDR*) contenuti nel vettore *Header[]*. La dimensione dei buffer (*Count*) è anch'essa predefinita nella finestra di "setup" del generatore di funzioni. La forma d'onda viene generata (tramite *GenerateFunction(…)*), il puntatore memorizzato nella posizione corrispondente del vettore *header[]*, e viene invocata l'API *waveOutPrepareHeader* con il medesimo significato della omologa *waveInPrepareHeader* descritta nella sezione 2.1 relazione (2.f.4). Infine, la chiamata alla routine *waveOutWrite* da inizio al processo di generazione della forma d'onda.

Il metodo Execute funziona dunque nella seguente maniera: dopo qualche semplice operazione di inizializzazione, si entra nel ciclo infinito di tipo "for" che :

1. testa se un buffer è stato utilizzato (*IsBufferDone*);
2. se si, lo libera (*unprepare*) altrimenti passa al giro successivo (salta a (1));
3. ne genera uno nuovo (*GenerateFunction*);
4. lo invia al buffer circolare impostato nella scheda;
5. salta al passo (1).

Si osservi l'uso del metodo "*LedOn*"; esso produce il lampeggio del riquadro verde che circonda il bottone "On" del generatore di funzioni. La rispettiva funzione "*LedOff*" è chiamata nella *BufferDone*, ottenendo così un indicatore che monitorizza attivamente lo "stato di salute" del thread generatore di funzioni.

2.2.4 La modalità Loop

Questa seconda opzione di funzionamento non fa uso di un thread dedicato, e si trova anzi direttamente inclusa (come metodo) nell'oggetto "*GenFun*" che costituisce la finestra principale del generatore di funzioni. Come detto, il calcolo della forma d'onda è eseguito "one shot" utilizzando la stessa routine di generazione "*GenerateFunction*" riempiendo un solo vettore di campioni ed in modalità "monofonica" non potendosi determinare univocamente due forme d'onda a differenti frequenze con buffer di identiche dimensioni. Quest'ultimo fatto sembra infatti essere un vincolo irrinunciabile per la maggior parte delle schede sonore. Eseguite le suddette operazioni, e settata la modalità "loop infinito" della scheda sonora, viene iniziata la generazione, eseguita in perfetto parallelismo hardware rispetto alla CPU (ad esclusione delle risorse spese per la gestione della finestra "generatore di funzioni" , comunque del tutto passiva e che può essere ridotta a icona o perfino chiusa).

2.4 Il frequenzimetro

La funzione frequenzimetro fa uso di un thread aggiuntivo, appartenente alla classe *FreqTask* la cui istanza utilizzata è stata denominata *FFTpad* per motivi che appariranno presto chiari. La più semplice implementazione di un

frequenzimetro, in un programma che già esegue una analisi spettrale tramite trasformata di Fourier, è ovviamente quella di ricavare il valore di ampiezza dell'armonica a maggiore intensità (la "fondamentale"): la sua frequenza sarà il valore che cerchiamo. Quindi con una relativa facilità; basta scorrere la lista delle armoniche, individuare quella ad ampiezza più elevata, e presentare a video il valore ricavato. Invero è quello che viene fatto, sebbene solo per l'opzione "base" a minore qualità. Infatti, i normali parametri con cui viene normalmente configurato l'analizzatore di spettro non portano ad una risoluzione soddisfacente. Per esempio, si consideri il caso (molto frequente) in cui si usi una frequenza di campionamento pari a 44100 Hz ed un buffer di 4096 punti. In tal caso la risoluzione è pari a 44100/4096 = 10.76 Hz. Questo significa che potremo determinare la frequenza di una forma d'onda a "passi" di oltre 10Hz, con l'imprecisione che ne consegue. Per aumentare la risoluzione, abbiamo due (ovvie) possibilità :

- Aumentare le dimensioni del buffer
- Diminuire la frequenza di campionamento

Nel primo caso andremo incontro ad un notevole incremento dei tempi di calcolo nel ciclo principale, oltre ad una latenza di acquisizione sempre più marcata (i buffer arrivano a intervalli temporali proporzionalmente più elevati). Questo potrebbe essere di nessuna utilità per la funzione analizzatore di spettro e parimenti per oscilloscopio, e costituire dunque un ingiustificato appesantimento generale e grosso spreco di risorse

Nel secondo caso limiteremmo la banda passante in maniera notevole, cosa che potrebbe non essere accettabile per l'applicazione in corso.

Per evitare i due inconvenienti dei punti precedenti, si è operato nella seguente maniera. Intanto si parte dalla considerazione che è accettabile, per un frequenzimetro, una frequenza di refresh con tempi dell'ordine delle centinaia

di millisecondi e finanche di secondi (sono i tempi tipici di un frequenzimetro "reale"). Allora, per svincolarsi completamente dal calcolo della FFT effettuato per l'analizzatore di spettro, è stato definito un thread dedicato che opera a priorità *tpNormal* (il thread *GetData* opera a *tpTimeCritical*) e calcola una sua FFT interna. Per ottenere le risoluzioni volute sarà comunque necessario operare su buffer di dimensioni elevate. Ma, essendo la finalità di questa solo di determinare la frequenza dell'armonica a maggior ampiezza, si è utilizzato un comodo artificio. I punti "reali" acquisiti dal thread principale *GetData* vengono passati al thread *FFTpad* ; i punti aggiuntivi vengono posti a zero sfruttando la tecnica nota come "zero padding" (da qui il nome *FFTpad* = FFT con zero padding). In più, per evitare sprechi di memoria, non si è allocata ulteriore memoria per i punti aggiuntivi (che sono tutti nulli) ma semplicemente creata una versione "modificata" della routine che esegue l'FFT. Usando la tecnica di "zero padding" possono essere aggiunti un numero qualsivoglia di punti aggiuntivi (sempre rispettando il vincolo di essere globalmente una potenza di due) aumentando a piacere la risoluzione. I tempi di calcolo aumentano (in misura di N*log(N)) ma vengono eseguiti nei tempi morti, senza gravare sul ciclo principale che è notoriamente "time critical". In pratica, ad ogni giro del ciclo principale (thread *GetData*) e quindi ad ogni nuovo buffer pervenuto dalla scheda sonora, il thread FFTpad viene "risvegliato" (resume) dopo avergli passato i punti del buffer. Sino a che i calcoli non vengono ultimati non viene passato nessun buffer ulteriore; effettuati i calcoli e presentati a video il task viene sospeso in attesa di un nuovo buffer ed essere nuovamente posto in stato di "esecuzione". In questo modo, per risoluzioni relativamente spinte, non si ottiene il calcolo su ogni buffer acquisito ma su un campione certamente rappresentativo (parliamo di funzioni periodiche).

Il risultato ottenuto è notevole; i tempi di esecuzione del ciclo principale sono rimasti sostanzialmente invariati, ed i tempi di calcolo della frequenza rimangono estremamente contenuti. Le prestazioni rilevate, usando una frequenza di campionamento di 40960 Hz e 4096 punti di buffer (10 Hz di

risoluzione “naturale”), su Pentium IV 1.6 Ghz 512 MB ram e con dimensioni delle finestre “di default” sono le seguenti:

- risoluzione 10Hz (ossia thread non allocato): tempo “nullo” (non misurabile);
- 5 Hz (pari a 4096 p.ti reali + 4096 zeri = 15 mS;
- 2.5 Hz (pari a 4096 p.ti reali + 3*4096 zeri = 28 mS;
- 1.25 Hz (pari a 4096 p.ti reali + 7*4096 zeri = 72 mS;
- 0.63 Hz (pari a 4096 p.ti reali + 15*4096 zeri = 215 mS;
- 0.32 Hz (pari a 4096 p.ti reali + 31*4096 zeri = 505 mS;
- 0.16 Hz (pari a 4096 p.ti reali + 63*4096 zeri = 1115 mS.

Il primo tempo è stato dichiarato come non misurabile, perché non involve l’istanziazione di alcun thread aggiuntivo, e quindi non misurabile come tempo impiegato dal thread. Esso è tuttavia causa di un aggravio piccolo ma finito, perché deve pur sempre effettuare un ciclo sul buffer (4096 punti in questo esempio) identificare l’armonica a maggior ampiezza e infine effettuare il disegno a video (tempi comunque molto inferiori al mS e quindi non misurabili con i metodi usati).

Infatti, la misurazione del tempo è effettuata all’interno del metodo Execute del thread *FFTpad* usando l’API di sistema *GetTickCount()* che restituisce una variabile intera il cui passo minimo rappresenta un millisecondo.

Riportiamo infine il codice relativo al metodo Execute della classe FreqTask:

```
void __fastcall FreqTask::Execute()
{

// variabile usata per il test
```

```
  TimeTest = 0;

  while (!Terminated)
  {

   __int64 TIME_START = GetTickCount();

   int step = 1;

// pre calcoli per la trasformata FFT
   for (level = 1; level <= _logPoints; level++)
   {
      int increm = step * 2;
      for (int j = 0; j < step; j++)
      {
         Complex U = _W [level][j];
         for (int i = j; i < Points; i += increm)
         {
            // butterfly
            Complex TT = U;
            TT *= X [i+step];
            X [i+step] = X[i];
            X [i+step] -= TT;
            X [i] += TT;

         }
      if (Terminated) break;
      }
      step *= 2;
      if (Terminated) break;
   }
```

```
   MAX = 0;
   iMAX = 0;
   if (!Terminated)
   {
    for (int i = 1; i <= Points/2; i++)
    {
     double temp = X[i].Mod();
     if (temp > MAX)
     {
      MAX = temp;
      iMAX = i;
     }
    }

// provvisori per il calcolo delle prestazioni
     TimeTest = GetTickCount() - TIME_START;

     Synchronize(&UpdateCaption);
     Synchronize(&UpdateLedOn);
    }

   if (Terminated)
    break;
   else
    Suspend();
  }


delete []BitReverse;
```

```
for (int l = 1; l <= _logPoints; l++) delete []_W[l];

delete [] _W;
}
```

Listato 11

2.5 Cattura dei segnali nel dominio del tempo

Visual Analyser è provvisto di due utilissime funzioni "offline", ossia eseguite non strettamente in tempo reale, che sono le uniche a far uso di alcuni oggetti "preconfezionati" di terze parti per la visualizzazione di grafici e diagrammi, le cui potenzialità non sono nemmeno state sfruttate completamente. La funzione di cattura di Visual Analyser è estremamente potente, sia nella fase di acquisizione (con modalità che consentono l'impostazione di soglie d'intevento variamente impostabili) che nella fase successiva, ossia dopo l'acquisizione è possibile gestire in vario modo il segnale acquisito (effettuare la conversione digitale/analogico, l'analisi armonica, la stampa ed il salvataggio). Lasciamo al capitolo sesto i dettagli operativi e concentriamoci su aspetti architetturali.

La cattura dei segnali nel dominio del tempo, ossia dei campioni visualizzati dall'oscilloscopio, si effettua solo con Visual Analyser in esecuzione e cliccando sul bottone "capture scope" presente sulla finestra principale o equivalentemente nella finestra di Settings (non è in questo caso presente un bottone nella barra dei comandi). La caratteristica saliente del meccanismo di acquisizione è che essa avviene su un numero di buffer a partire da un minimo di uno sino ad un massimo limitato solo dalla memoria effettivamente presente nel sistema. L'impostazione del numero di buffer avviene dalla finestra di Settings, sottocartella "Capture Scope/Spectrum", ed avviene in termini di "tempo" di acquisizione. Esso è ovviamente calcolato come un multiplo del

buffer di acquisizione, più una porzione rimanente; la risoluzione temporale dipende ovviamente (al solito) dalla frequenza di campionamento, ossia di quanto sono distanziati temporalmente due campioni successivi. In tal senso, l'operazione di acquisizione fisica dei campioni è ancora in tempo reale, sebbene avvenga a finestra non ancora resa visibile. Tutte le operazioni effettuate dalla finestra non sono in tempo reale e avvengono a priorità più bassa del ciclo principale.

La finestra compare subito dopo il trasferimento dei campioni nelle strutture interne della finestra di cattura, presentando a video una porzione degli stessi (che possono essere comunque "zoomati", selezionati e fatti "scorrere" nelle varie direzioni). Le funzioni che è possibile applicare ai campioni acquisiti sono:

- effettuare la conversione digitale analogico dei campioni visibili nella finestra, separatamente per ogni canale;
- effettuare la trasformata di Fourier (sempre tramite FFT) e visualizzarla in contemporanea al segnale in due finestre separate;
- stampare e salvare su file in formato testo o grafico wmf;

La conversione digitale analogico è in questo caso *completa*, ossia non approssimata come le esigenze di calcolo in tempo reale impongono. In tal senso, viene applicato l'algoritmo su tutti i punti acquisiti. Essa avviene in realtà tramite un thread "implicito", ossia tramite un ciclo che preleva i campioni dai vettori interni e itera dando ad ogni giro la possibilità di servire la coda dei messaggi associata al processo tramite la primitiva delle librerie Borland "Processmessages()" (che al suo interno richiama le classiche API "GetMessage", "TranslateMessage()" e "DispatchMessage") consentendo di fatto di eseguire la routine e contemporaneamente mantenere attiva l'interfaccia utente della finestra. In questo caso si è optato per questa soluzione invece dell'uso di un ennesimo thread per motivi di semplicità.

Naturalmente il “costo” di una conversione operata utilizzando tutti i punti acquisiti diviene ben presto notevole; un meccanismo di stima dei tempi necessari avvisa l’utente della situazione. E’ comunque possibile interrompere il calcolo in qualsiasi momento senza perdere i punti calcolati che vengono comunque presentati a video.

Le modalità di acquisizione comprendono:

- senza threshold;
- con threshold, ed acquisizione dopo il primo superamento del livello di threshold;
- con threshold, ed acquisizione dopo un certo tempo continuativo di superamento della soglia.

Esse verranno descritte in maggiore dettaglio nel capitolo sesto.

2.6 Cattura dei segnali nel dominio della frequenza

Questa finestra è analoga a quella definita nel paragrafo precedente, e consente di acquisire lo spettro già calcolato dalle routine dell’analizzatore di spettro. Concettualmente più semplice, non ha soglie ed operazioni sofisticate al suo interno (sebbene una prossima espansione potrebbe, per simmetria con la analoga funzionalità nel dominio del tempo, prevedere delle soglie da impostare sul livello di una o più armoniche). Consente numerose operazioni di manipolazione grafica dello spettro, e di effettuare stampe, salvataggi ed acquisizioni in clipboard. La modalità di acquisizione è invece leggermente differente; nel caso del paragrafo 2.5 l’acquisizione dei campioni avviene nel ciclo principale tramite semplice operazione di copia. In questo caso

l'operazione è la medesima, con l'aggiunta di una operazione di media aritmetica su un certo numero di buffer definibili nella finestra di Setting. L'operazione di media avviene ad opera di un metodo già previsto negli oggetti che effettuano la trasformata di Fourier (metodo "average" della classe FFT).

Da non confondere dunque la media aritmetica effettuata in tempo reale sullo spettro visualizzato nella finestra dell'analizzatore di spettro, da quella relativa a questa finestra. Invero, l'esistenza di questa doppia possibilità non implica sovrapposizioni. Infatti, settando un certo fattore di media per la finestra in tempo reale esso non sarà considerato se si attiva la cattura dello spettro, e viceversa. Ossia le funzioni sono completamente svincolate tra loro pur essendo sostanzialmente identiche. Da un lato questo può sembrare un controsenso, ma è stato introdotto dapprima casualmente (le funzioni sono state aggiunte negli anni) eppoi è stato volutamente lasciato per fornire un grado di libertà ulteriore. In tal modo si può visualizzare un segnale mediato ed acquisire al contempo il segnale standard; e viceversa.

2.7 I filtri digitali

Visual Analyser consente di inserire nel percorso del segnale una serie predefinita di filtri digitali che consentono di filtrare il segnale inviato agli strumenti; come discusso in altre sezioni, esso non è il segnale che poi viene inviato alla scheda sonora e quindi ai diffusori acustici. In parole povere, non è possibile ascoltare il segnale filtrato, ma solo visualizzarlo. Questo è logico perché lo scopo di Visual Analyser è quello di implementare degli strumenti di misura e non un programma di elaborazione di segnali audio. In ogni modo questa era una opzione presente nelle precedenti versioni del programma, attualmente disabilitata perché ancora non del tutto stabile.

I filtri implementati consentono di effettuare dei filtraggi in tempo reale sul segnale in ingresso agli strumenti, e possono essere utilizzati per isolare il segnale "utile" da componenti armoniche indesiderate e rumore. Essi sono fisicamente implementati da metodi della classe FFT; quando (vedi listato 3) il segnale viene copiato nell'oggetto di tipo FFT tramite il metodo CopyIn, esso viene anche moltiplicato per i coefficienti del filtro, ossia viene effettuata la "convoluzione" con la funzione di trasferimento del filtro. Ecco a titolo di esempio uno spezzone di codice interno al metodo CopyIn che effettua la convoluzione o il calcolo del filtraggio di tipo IIR:

```
            for (int i = 0; i < cSample; i++, iter.Advance())
            {
              temp = GetS();
              if  (DCREMOVAL)
              {
                DCremoval(temp);
                dataprev = GetS();
              }
              if  (DIODE && (temp < 0)) temp = 0;
              tape[i] = temp;
            }

            _aTape[0] = 0;
            _aTape[1] = 0;
            _aTape[2] = 0;
// Se il filtro è di tipo FIR effettua la convoluzione
// …altrimenti applica il filtro IIR
            if (Filter.ItemIndex<5)
              Convolve(_aTape, tape);
            else
              IIRfilter(_aTape, tape);
// pre calcoli per la trasformata di Fourier
```

```
        for (int i = 0; i < cSample; i++)
        {
         temp = _aTape[i];
         _X [_aBitRev[i]] = Complex (temp);
         tape[i] = temp;
        }
      }
```

Da esso si evince con una certa facilità che il metodo che effettua la convoluzione con il segnale è il metodo "*Convolve*".

I coefficienti dei filtri vengono calcolati dal metodo "*SetFilter*" ogni qual volta vengono selezionati nuovi filtri o cambiato qualche parametro fondamentale , come frequenza di campionamento e dimensioni del buffer. La *SetFilter* usa a sua volta le procedure "LowPass, HiPass, BandReject, BandPass" per i filtri FIR e le procedure "IIRnotch, IIRinvNotch" per i filtri IIR. Per i filtri FIR viene essenzialmente riempito il vettore "*Kernel*", per i filtri IIR calcolati i coefficienti "a" e "b". Si noti che tutti i filtri FIR sono ricavati a partire dall'unica procedura "*LowPass*", potendosi da questa ricavare i coefficienti del filtro passa alto semplicemente invertendo il segno dei coefficienti del corrispondente passa basso; e combinando questi ultimi due ottenere i restanti filtri.

Capitolo 3

La scheda sonora

3.0 Introduzione

Si è ritenuto utile dedicare un capitolo a parte per discutere, seppur brevemente, di uno degli elementi che maggiormente caratterizzano l'esistenza e la fattibilità di questo lavoro: la scheda sonora, ossia l'hardware di acquisizione e generazione dei segnali. I primi PC interagivano con l'utente attraverso un piccolo altoparlante posto all'interno del "case" del PC, capace di emettere semplici beep predefiniti e presto anche singole note (sostanzialmente un'onda quadra la cui frequenza si poteva variare da programma). Ben presto furono create le prime schede sonore, ossia dispositivi hardware che si potevano inserire nel bus del PC (come una normale scheda di espansione), capaci di gestire l'acquisizione e la generazione dei segnali audio, e che interagivano con il PC tramite il bus della motherboard. Dapprima dotate di una elettronica piuttosto semplice, consentivano di emettere comunque suoni ben più complessi di quelli possibili con l'altoparlante interno.

Le primissime schede venivano spesso direttamente pilotate dai programmi senza appoggiarsi a driver particolarmente sofisticati (per esempio interagendo direttamente con i registri interni della scheda sonora, passandogli dati e comandi).

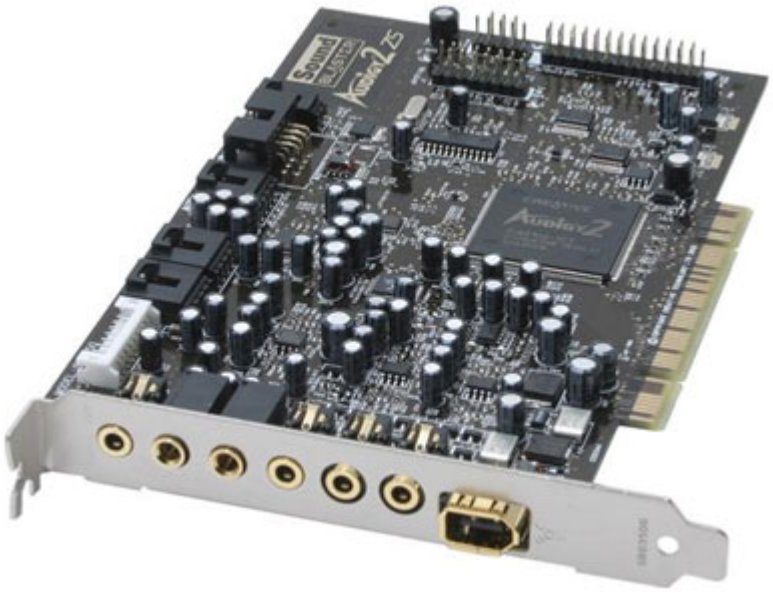


Figura 9 : una tipica scheda audio di tipo interno

Con l'avvento di Windows e delle schede madri dotate di "chipset", la gestione della scheda audio viene sempre più affidata ai driver scritti dal costruttore ed utilizzati dal software applicativo tramite le API di sistema. Le schede audio avevano la capacità di acquisire i segnali ad esse applicati convertendoli in segnali digitali (tramite un convertitore analogico/digitale) e viceversa in uscita (tramite un convertitore digitale/analogico). I segnali acquisisti potevano essere poi utilizzati dal software applicativo, per esempio per implementare un simulatore di strumenti come Visual Analyser; parimenti i segnali generati dal software applicativo venivano inviati alla scheda sonora, convertiti in analogico ed applicati ad un altoparlante (dopo opportuna amplificazione). In tal modo un PC rendeva possibile l'ascolto di musica tramite brani memorizzati direttamente sull'hard disk, o generare suoni complessi via software ,eccetera.

3.1 L'evoluzione della specie

Ben presto l'evoluzione della tecnologia con l'introduzione, verso la fine degli anni 80 e l'inizio degli anni '90, dei processori DSP apre nuovi orizzonti in

moltissimi settori. Non ultimo quello delle schede sonore, che vedono realizzarsi la possibilità di implementare funzioni complesse, tipiche della materia nota come "Elaborazione Numerica dei segnali", sgravando completamente la CPU da qualsiasi onere computazionale. In tal modo, la scheda sonora diviene generatore musicale, filtro digitale e persino interfaccia per lo standard MIDI. Ben presto, il PC diviene uno strumento capace di eseguire spartiti, comporre musica, ascoltare brani memorizzati su supporti di memorizzazione di massa e molto altro ancora, tutto in contemporanea con altre attività.

Ancora, l'introduzione dello standard USB e Firewire (ieee1394) come standard di comunicazione seriale veloce, ha ulteriormente ampliato le possibilità di espansione dei PC, schede sonore comprese. Le schede sonore, come moltissimi altri dispositivi, possono essere finalmente svincolate persino dall'hardware del PC (ossia dalla particolare tecnologia utilizzata dalla scheda madre), ed interfacciarsi ad esso con un semplice cavo USB o Firewire; grazie a degli appositi strati software vengono rese visibili al sistema operativo come delle normali schede di espansione direttamente inserite nel bus di pertinenza (per esempio PCI nel caso di schede sonore, o IDE nel caso di hard disk).

Figura 10: scheda sonora esterna

3.2 Il mondo dei DSP

I DSP sono dei processori dedicati al trattamento in tempo reale di segnali numerici. Tipicamente i segnali digitali sono ottenuti tramite campionamento di segnali analogici (quindi tramite i convertitori analogico/digitale). Il DSP può così elaborare i segnali digitali in tempo reale, applicando delle trasformazioni matematiche ed algoritmi che consentono di realizzare operazioni impensabili nel mondo esclusivamente analogico, o comunque con caratteristiche molto migliorate (ad esempio un filtro digitale presenta pendenze e livelli di attenuazione nella "stopband" diversi ordini di grandezza più elevati del corrispondente filtro analogico). Dopo aver effettuato una elaborazione di tipo numerico, il segnale viene dunque riconvertito in segnale analogico tramite un convertitore digitale/analogico, ed il segnale analogico risultante è a sua volta un segnale "elaborato". In pratica, la catena [conversione digitale] – [elaborazione con DSP] – [conversione analogica] si comporta globalmente come un dispositivo analogico che effettua una qualche trasformazione sul segnale applicato ai suoi ingressi. Con la fondamentale differenza che la trasformazione effettuata *dipende da un programma di calcolo* e quindi, oltre a fornire prestazioni notevoli é anche estremamente flessibile. Per esempio, applicando gli opportuni algoritmi è possibile implementare digitalmente un filtro passa basso o passa alto, o effettuare un filtraggio del rumore od eseguire una qualche equalizzazione che segue curve complesse (per esempio secondo la curva RIAA per i dischi in vinile). Od anche funzioni di alterazione della voce (scrambling) o per esempio effetti speciali in ambito musicale (eco, riverbero etc). La figura 11 sintetizza il meccanismo descritto.

Figura 11 : la catena completa

3.3 Il mondo dei DSP ed i PC

Ben presto il vertiginoso incremento della potenza di calcolo dei moderni PC ha reso possibile l'utilizzo di molti degli algoritmi un tempo appannaggio esclusivo dei DSP. Per esempio l'algoritmo noto come FFT (vedi capitolo quarto) consente il calcolo dello spettro di un segnale; esso era generalmente utilizzato in costosi strumenti, detti analizzatori di spettro, dotati di potenti DSP appositamente programmati. Attualmente è possibile una implementazione dello stesso algoritmo in un programma per PC, sfruttando come strumento di acquisizione dei segnali la scheda sonora oppure un hardware dedicato, costruito secondo gli standard PCI o USB. Anche nel mondo video si assiste alla medesima rivoluzione; l'uso di PC mediamente potenti in abbinamento con schede video sempre più performanti (dotate di processori autonomi e memoria locale) consente di implementare via software programmi di editing video una volta disponibili solo come costosissime apparecchiature hardware/software dedicate (e quindi alla portata di pochi laboratori professionali).

3.4 Schema di principio di una scheda sonora e volume di Windows

L'architettura interna di una scheda sonora moderna è riportata in figura 12. Essa prevede, come ampiamente accennato nei paragrafi precedenti, l'uso di convertitori A/D e D/A, di un DSP, e amplificatori-buffer. Più altri componenti,

alcuni dei quali ininfluenti per lo scopo di questo lavoro ed altri utili che sarà conveniente descrivere con un certo dettaglio. In particolare, si osservi cosa accade al percorso del segnale in ingresso. Esso viene amplificato da un amplificatore audio e successivamente inviato ad un mixer; il mixer è di tipo analogico e viene direttamente controllato dal tanto famoso quanto incompreso mixer di Windows. Il "Mixer Volume Control" di Windows altro non è che un software che, tramite l'uso di complicatissime API di sistema si interfaccia al mixer hardware (analogico) della scheda sonora, e ne gestisce le varie funzionalità. Esse comprendono il livelli di registrazione, il bilanciamento dei canali ed in modelli di schede più sofisticate prevede anche dei semplici controlli di tono. Da notare che stiamo parlando di livello di registrazione e non riproduzione. Infatti, se lanciamo l'eseguibile SndVol32.exe in Windows XP (ma anche in 9x, 2000 ed NT), cosa che possiamo fare direttamente dalla sua icona standard (normalmente presente sulla barra inferiore di Windows - per intenderci dove c'è l'orologio) o, se non presente, direttamente da finestra dos o da pannello di controllo, manderemo in esecuzione il "controllo di volume di windows" o "Mixer volume control". Esso parte però in modalità "riproduzione", simmetrica di quella di "registrazione" di cui stiamo discutendo. Infatti, quest'ultima pilota il mixer visibile in figura 12, mentre quella relativa alla riproduzione pilota un mixer analogico posto subito dopo il convertitore D/A (non riportato in fugura, si troverebbe a fianco di quello già presente), che è praticamente ininfluente per la maggior parte degli strumenti simulati da Visual Analyser, eccetto uno. Visual Analyser infatti preleva il segnale subito dopo il convertitore A/D. Si noti che, allo scopo di semplificare la gestione dei livelli d'ingresso, Visual Analyser prevede una gestione autonoma del controllo di volume in registrazione che è descritta nel capitolo sesto (così si evita di dover essere dipendenti da un software esterno). Qualora si utilizzi invece il generatore di funzioni, allora il controllo riproduzione è ovviamente di una qualche utilità (con l'unica funzione di variare l'ampiezza del segnale generato).

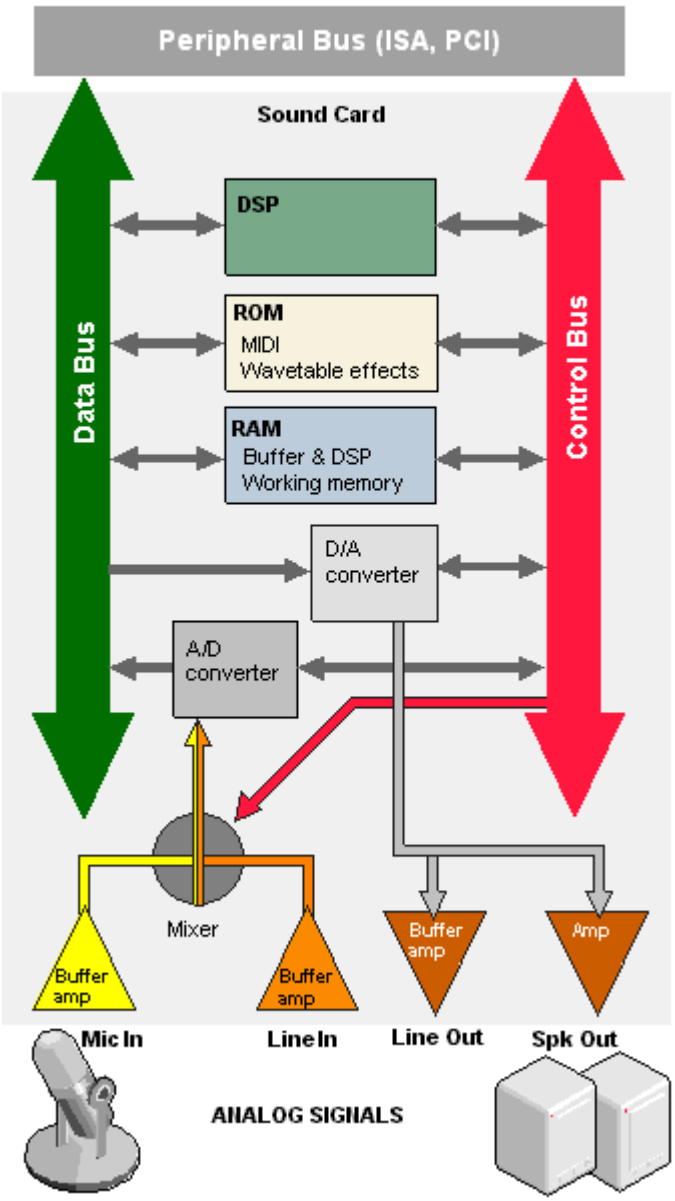


Figura 12 : architettura di una scheda sonora

Volendo comunque usare il Controllo di volume di Windows per tutte le funzioni escluso il generatore di funzioni, si deve:

1. lanciare il programma SndVol32.exe;

2. andare in Opzioni/Proprietà;
3. selezionare registrazione;
4. se necessario selezionare i controlli di interesse (Line-in, mic, etc);
5. cliccare sul tasto OK.

3.5 Sensibilità d'ingresso e impedenza

In particolare, indipendentemente dall'uso che si fa di Visual Analyser, sorgono due problemi. Ossia: quale ingresso usare? E qual è la sensibilità d'ingresso di ciascuno? La prima domanda ha una duplice risposta; dipende dall'ampiezza del segnale ma, in generale, anche dalla banda passante richiesta. Generalmente l'ingresso Line-in o Aux presenta una sensibilità d'ingresso che varia, in funzione del modello, da 300 mV a 2 V; e presenta una risposta in frequenza generalmente estesa a tutta la banda disponibile. Viceversa un ingresso microfonico ha una sensibilità più elevata, che parte da 10 mV sino a 300/500 mV, ma spesso limitato superiormente in banda, tipicamente sui 10 kHz. Sul web è possibile trovare informazioni, sebbene alle volte in disaccordo tra loro. L'esperienza pratica porta a consigliare l'utilizzo dell'ingresso Line-in o Aux, anche perché tipicamente l'impedenza d'ingresso è considerevolmente più elevata e può arrivare sino a 100 Kohm, contro i 10..47 Kohm dell'ingresso microfonico. Il fattore impedenza d'ingresso è cruciale; intanto, è evidente che, indipendentemente dall'ingresso scelto, assai difficilmente avremo a che fare con segnali di ampiezza nota, ed in generale potrebbero essere di ampiezza troppo elevata o troppo ridotta. Nel primo caso è facile porre rimedio: basterà costruire un attenuatore costituito da una semplice resitenza in serie al segnale, il cui dimensionamento può essere fatto in funzione del rapporto di attenuazione desiderato. Il secondo caso è anch'esso di facile soluzione: basta costruire un semplice preamplificatore che adatti il segnale alla sensibilità

d'ingresso disponibile. Un altro fattore importante è l'impedenza d'ingresso; un'impedenza d'ingresso elevata è senza dubbio desiderabile per due motivi:

- non sovraccarica il circuito sotto esame;
- consente di utilizzare "probe" di oscilloscopi commerciali.

Il secondo punto è vero purtroppo in parte; un classico probe di oscilloscopio con rapporto di attenuazione 10x ha in serie una resistenza di circa 10 Megaohm se l'impedenza dell'oscilloscopio è di 1 Mohm (che normalmente è un valore standard); utilizzando ingressi microfonici con impedenza troppo bassa (10 Kohm) avremo una attenuazione proporzionalmente più elevata e probabilmente eccessiva (1:1000). Attenuazione utile solo in casi molto particolari, e comunque possibile fonte di ulteriori disturbi.

Attenzione, è sempre meglio in prima battuta esagerare con i fattori di attenuazione, perché un segnale d'ingresso di ampiezza troppo elevata potrebbe irreparabilmente danneggiare la scheda sonora. La messa a punto può arrivare successivamente e per gradi

Sebbene esuli dagli scopi di questo lavoro, si fanno le seguenti considerazioni. La soluzione ai problemi di sensibilità/impedenza è semplice; a mio avviso sarebbe opportuno dotarsi di un attenuatore "variabile", per esempio un commutatore che consente di selezionare tra un set di resistenze, od addirittura anche un semplice potenziometro di con valore di qualche decina di Mohm. Ed utilizzare l'ingresso Line-in. Qualora si avessero problemi di bassa sensibilità d'ingresso (ossia si devono visualizzare segnali di bassissimo livello), allora si può aggiungere un semplice preamplificatore d'ingresso. E' possibile con costi praticamente irrisori costruire ottimi preamplificatori a componenti discreti od a circuiti integrati con ottimi rapporti S/N e banda passante elevatissime, e persino elevate impedenze d'ingresso (p. es. amplificatori con stadi d'ingresso a FET). La rete abbonda di progetti simili.

Le considerazioni pratiche relative ai possibili schemi di attenuazione e/o circuiti di protezione aggiuntivi anch'esse esulano dagli scopi del presente lavoro, ma sono altresì di enorme interesse per chi voglia praticamente utilizzare questo programma; è tuttavia possibile ricavare suggerimenti in tal senso sul web e/o contattandomi direttamente (a.accattatis@libero.it).

Capitolo 4

Algoritmi

4.0 Introduzione

Visual Analyser rappresenta un esempio di programmazione concorrente, un software in tempo reale e anche un tipico esempio di applicazione degli algoritmi più diffusi della materia nota come "Elaborazione numerica dei segnali". Come accennato in più punti del presente lavoro, verso la fine degli anni '80 e inizio degli anni '90 l'evoluzione dell'hardware dei PC ha fatto si che algoritmi una volta appannaggio esclusivo del mondo dei DSP divenissero finalmente utilizzabili anche su semplici personal computer. E talvolta in maniera così efficiente da preferire l'uso esclusivo dei PC al posto di macchine dedicate e poco flessibili.

Visual Analyser fa uso dei più noti algoritmi utilizzati normalmente negli strumenti di misura digitali, con l'aggiunta di soluzioni architetturali e approssimazioni create "ad hoc" per la risoluzione di particolari problemi pratici. Ad esempio, nell'usare il teorema di Shannon/Nyquist si è ovviamente dovuto utilizzare qualche semplice artificio per riuscire ad effettuare i calcoli nei tempi disponibili (vedi paragrafo 1.4 e paragrafo 2.2.2 oltre al presente capitolo); Oppure, per il calcolo dello spettro del segnale si è dovuto operare su insiemi di punti che rappresentano parte di un segnale in realtà molto più

esteso, dovendo pertanto fronteggiare una distorsione indotta, che avrebbe potuto tradursi in uno spettro non stabile e con armoniche aggiuntive.

In questo capitolo passeremo in rassegna gli algoritmi utilizzati da Visual Analyser, senza addentrarci troppo in dettagli teorici e dimostrazioni, per le quali rimandiamo ai numerosissimi testi esistenti e la copiosa quantità di informazioni reperibile in rete; porremo l'accento, laddove necessario, sulle soluzioni particolari e semplificazioni adottate in Visual Analyser.

4.1 Gli algoritmi utilizzati

In ordine di importanza e frequenza d'uso, ecco gli algoritmi principali utilizzati da Visual Analyser:

- Teorema di Shannon Nyquist (o teorema del campionamento) [4.2]
- Trasformata di Fourier veloce (FFT, Fast Fourier Transform) [4.3]
- Finestre di smoothing [4.4]
- Sviluppo in serie di Fourier (per generatore di funzioni) [4.5]
- Filtri digitali (FIR, IIR) [4.6]
- Calcolo THD (Total Harmonic Distortion = distorsione armonica totale) [4.7]

4.2 Il teorema del campionamento

Di esso abbiamo diffusamente parlato nel presente lavoro, ed in particolare abbiamo dato un valido accenno nel capitolo 1 (paragrafo 1.4) e nel capitolo 2 (paragrafo 2.2.2). Come prima considerazione facciamo osservare che esso è

fondamentale per la stessa esistenza di questo programma ma in generale per la pratica applicazione della materia nota come "Elaborazione Numerica dei segnali", consentendo di trasformare "operativamente" ogni segnale analogico in segnale digitale e viceversa. Più in generale ogni apparecchiatura elettronica, sia esso uno strumento di misura od un elettrodomestico fa inevitabilmente uso (indirettamente o direttamente) del teorema del campionamento. Per esempio un lettore CD, un telefonino, un videoregistratore DVD ma anche una autoradio, dove frequentemente è presente un DSP, praticamente sino ad arrivare persino alle batterie al litio, tanto per citare alcuni esempi tra i più eclatanti. Inoltre, ogni segnale telefonico viaggia nelle reti sotto forma di segnale numerico (almeno per la stragrande maggioranza delle tratte) sebbene esso in origine sia un segnale elettrico fornito da un microfono (la cornetta del telefono) e quindi strettamente analogico: le telecomunicazioni come le conosciamo oggi sono pertanto praticamente basate sul teorema di Shannon.

Bisogna però fare una distinzione importante, peculiare al presente lavoro. Visual Analyser usa il teorema del campionamento "implicitamente" tramite i convertitori presenti nella scheda audio; ed in maniera molto più esplicita, utilizzando l'algoritmo all'interno del suo codice per la ricostruzione del segnale analogico originale, ossia quello applicato agli ingressi della scheda sonora. Questo si è reso necessario per poter visualizzare il segnale sullo schermo dell'oscilloscopio, che è per sua natura uno strumento che visualizza su assi cartesiani un segnale strettamente analogico.

Ciò premesso veniamo all'enunciazione rigorosa del teorema di Shannon, per poi discuterla in termini pratici:

Ipotesi:

- sia X(t) un segnale a *banda limitata*, ossia la trasformata di Fourier X(f) sia uguale a zero per $| f | > B$ (dove B è la banda del segnale). Ossia, sia

X(t) un segnale il cui contenuto armonico vari da zero a un massimo (per esempio da 0 a 20000 Hz)

- sia la *frequenza di campionamento* maggiore od almeno uguale al doppio di B, ossia si prelevino *campioni* del segnale con una frequenza almeno doppia di quella della massima frequenza presente nel segnale (ossia B).

Allora (tesi):

- il segnale X(t) è rappresentato completamente dai suoi campioni
- il segnale può essere ricostruito con un filtro passa-basso avente frequenza di taglio Ft tale che B<Ft
- il segnale X(t) può essere ricostruito a partire dai suoi campioni con lo sviluppo in serie definito dalla seguente formula:

$$x(t) = \sum_n x(nT_c) \cdot \mathrm{sinc}\left(\pi \frac{t - nT_c}{T_c} \right)$$

(f.4.0)

Teorema la cui dimostrazione è lasciata ai testi di matematica. Detto in altre parole, il teorema asserisce che se da un segnale analogico preleviamo alcuni "campioni" (ossia il valore istantaneo del segnale in un determinato valore di tempo t) con una determinata cadenza, essi saranno sufficienti a ricostruire il segnale in maniera completa.

Il segnale così ottenuto non è ancora comunque un segnale digitale; esso è solo un segnale che ricade nella categoria dei cosiddetti "segnali tempo discreti". La digitalizzazione avviene se facciamo un passo ulteriore, ossia associamo un

numero ad ognuno di questi campioni (quantizzazione). In altri termini, dato un numero intero (per esempio a 16 bit) associamo ad ogni campione acquisito un numero tra tutti quelli possibili (che nel caso dei 16 bit sono 65536). Ossia *approssimiamo* il livello reale del campione con il numero intero ad esso più vicino. Così facendo abbiamo commesso un errore, tanto più piccolo quanto più la profondità di bit del numero intero è elevata (ossia la "grana" del nostro numero intero è più fine). Chiariamo ulteriormente questi concetti con dei grafici. La figura 13 rappresenta il processo di campionare un segnale analogico, immaginandolo come ottenuto dalla moltiplicazione di un treno periodico di impulsi per il segnale da campionare.

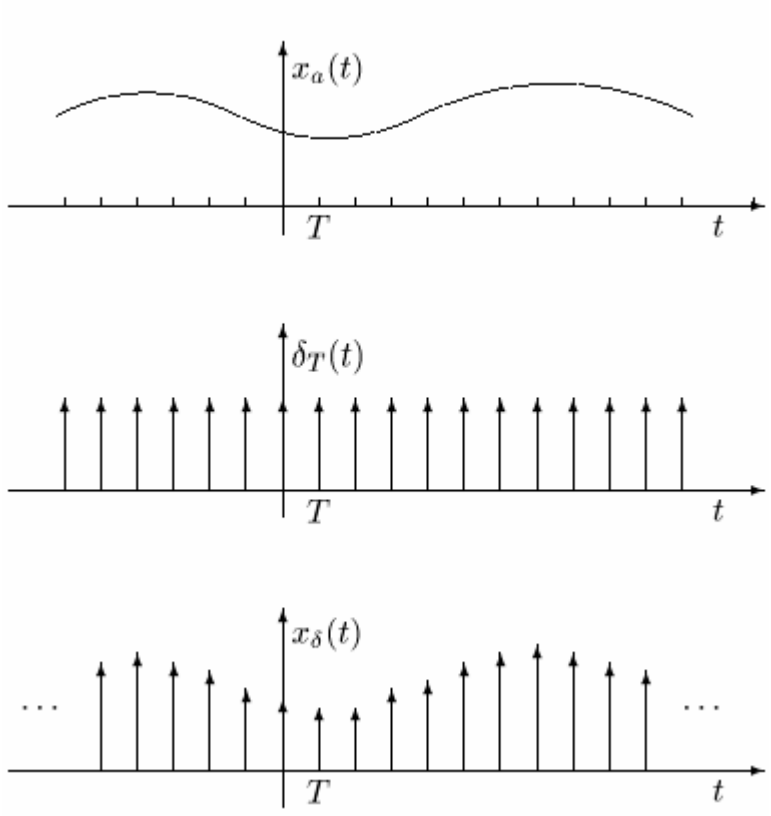


Figura 13

Nella prima parte della figura è rappresentato un generico segnale analogico x(t); nella parte immediatamente inferiore il treno di impulsi e nell'ultima parte il segnale ottenuto moltiplicando i due, ossia il segnale campionato. Ricordiamo che l'impulso è un segnale che vale "1" in un solo punto e zero altrove. Il segnale ottenuto è ancora un treno di impulsi il cui inviluppo segue il segnale originale tempo continuo. A partire da questo segnale sarà possibile effettuare una quantizzazione, e quindi ottenere

un segnale digitale, oppure applicare la (f.4.0) e ricostruire il segnale. Ancora, a partire dal segnale digitalizzato, ricostruire la sequenza di impulsi e applicare la (f.4.0) tramite la quale riottenere ancora il segnale originale. Ossia esattamente quello che faremo con Visual Analyser, operando delle semplificazioni aggiuntive (vedi paragrafo 4.1.1.1). Riportiamo una ulteriore figura (14) che serve a far meglio comprendere il processo di "campionare" (sample) un segnale analogico introducendo anche un'altra caratteristica, che è quella di "mantenere" (hold) il segnale sino al campione successivo, caratteristica preziosa per quei dispositivi hardware che poi dovranno effettuare la "quantizzazione".

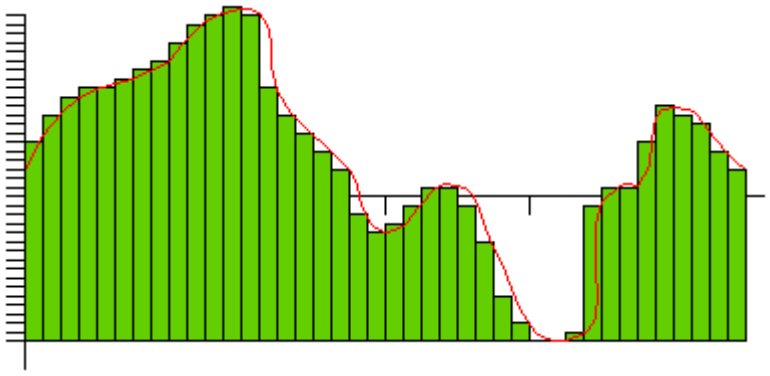


Figura 14: campionamento e mantenimento di un segnale

Il teorema appena enunciato è fondamentalmente alla base del funzionamento dei convertitori presenti nella scheda sonora, il cui compito è di operare la conversione di segnali analogici in segnali digitali, ossia rappresentati da sequenze di numeri e viceversa. In altre parole ancora, rendere "comprensibile" un segnale analogico ad un elaboratore (per esempio un PC).

Visual Analyser si avvale della (f.4.0) nelle modalità descritte nel paragrafo successivo.

E' importante fare delle considerazioni finali, relative al concetto di "aliasing". Il processo di campionamento da origine ad un segnale impulsivo che, come abbiamo appena visto, può servire a ricostruire completamente il segnale

originario tramite la (f.4.0) o equivalentemente applicando un filtro passa basso con frequenza di taglio pari alla frequenza di Nyquist. Questo può comprendersi meglio facendo notare che il segnale campionato ha ovviamente uno spettro molto più esteso dell'originale (sono impulsi, che per definizione hanno un contenuto armonico estesissimo) ma presenta la caratteristica di essere identico allo spettro originale ma "periodicizzato"; ossia è lo stesso spettro ripetuto all'infinito, come può vedersi nella figura 15. Pertanto, il filtro passa basso ha il solo scopo di eliminare gli spettri "aggiunti" e lasciare solo la porzione originale.

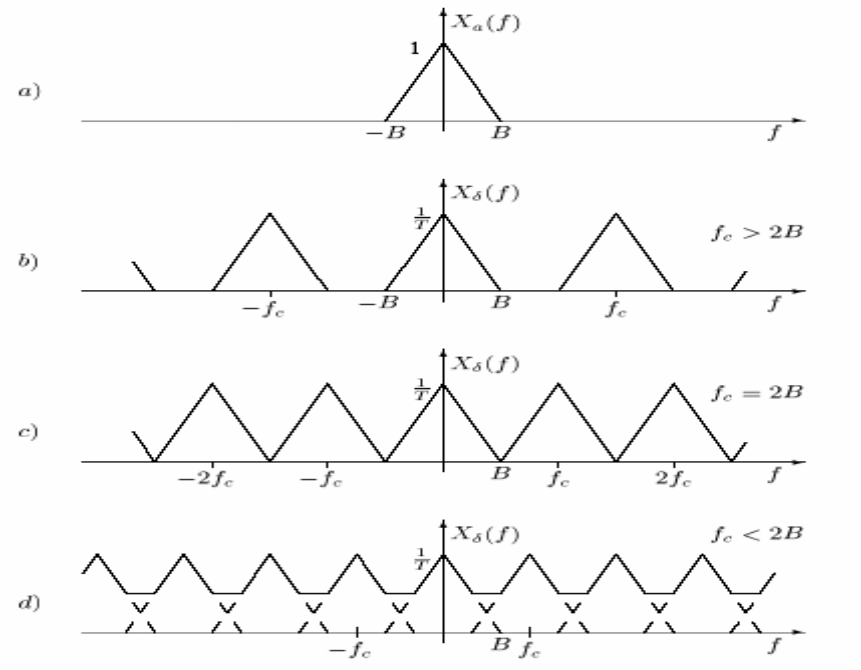


Figura 15: aliasing

Nella figura 15 abbiamo: in (a) lo spettro originale; in (b) lo spettro periodico ottenuto campionando con fc (frequenza di campionamento) addirittura maggiore della massima frequenza contenuta nel segnale (lo spettro è periodico ma ben separato); in (c) usando una fc strettamente uguale alla massima frequenza (lo spettro è periodico con periodi perfettamente adiacenti); in (d) un campionamento insufficiente, che porterà al fenomeno noto come *aliasing*,

ossia gli spettri periodici si “sovrappongono” rendendo impossibile la corretta ricostruzione del segnale.

4.2.1 La conversione digitale analogica in Visual Analyser

Visual analyser deve operare una conversione dei campioni digitali che gli vengono passati dalla scheda sonora; per le esigenze citate più volte in questo capitolo ed nel paragrafo 1.4; e con le modalità software riportate in 2.2.2 che qui verranno descritte in maggiore dettaglio.

I punti disponibili per la conversione digitale-analogica sono quelli relativi ad un buffer, e sono rigorosamente una potenza di due, per esigenze di altri algoritmi (FFT). La dimensione tipica di un buffer è di 4096 punti (ma definibile a piacere tra un set di valori), e la dimensione standard della finestra oscilloscopio è dell’ordine dei 400 pixel (sebbene possa arrivare sino alla massima risoluzione orizzontale consentita dalla scheda video). La formula da applicare è la (f.4.0), qui ripetuta per comodità, che implica una sommatoria (al limite infinita) estesa comunque *almeno* a tutti i punti del segnale disponibile e ripetuta per ogni punto del segnale da ricostruire.

$$x(t) = \sum_{n} x(nT_c) \cdot \mathrm{sinc}\left(\pi \frac{t - nT_c}{T_c} \right)$$

La prima considerazione da fare è che bisogna ripetere la sommatoria per almeno 400 volte su 4096 punti. Ossia avremmo 4096*400 = 1.638.400 iterazioni che comprendono:

- 1.638.400 divisioni;
- 4.915.200 moltiplicazioni;
- 1.638.400 operazioni di seno (la funzione “sinc(x)” è una abbreviazione di sin(x)/x);
- 1.638.400 sottrazioni.

Le operazioni di estrazione di seno e operazione di divisione sono comunque le più onerose a livello di cicli di CPU; sommando tutte le precedenti otteniamo la bellezza di 9.830.400 operazioni complessive da compiere in (molto) meno di 100 millisecondi. Infatti, tolti i 10..20 millisecondi necessari per le restanti operazioni (calcolo FFT, disegno a video eccetera), e considerando ragionevole riservarsi un margine di almeno 10 mS per fluttuazioni nella disponibilità delle risorse (es. CPU assegnata ad altri processi), il tempo rimanente si aggira intorno ai 60 millisecondi. Le prove sperimentali hanno fornito tempi che si aggiravano proprio intorno a tale valore, cosa che dipende ovviamente fortemente dalla macchina usata e dallo stato di carico. Il risultato è che il sistema diventava troppo critico per poter essere utilizzato praticamente. A questo si aggiunga il fatto che dovendo usare il generatore di funzioni o peggio la funzione doppia traccia, si sarebbe avuto un ulteriore consumo di risorse “nel ciclo” cosa che rendeva ancor più improponibile l’uso di un simile algoritmo.

La possibile soluzione arriva dalle seguenti considerazioni. Intanto, fissata una determinata frequenza di campionamento il termine Tc diviene una costante, e lo stesso dicasi per i termini n*Tc, dato che ad ogni buffer il contatore del ciclo della ricostruzione (n) riparte da zero. Lo stesso dicasi per la variabile t, che rappresenta (in Visual Analyser) il tempo “relativo” da inizio a fine schermata, ossia ad ogni schermata riparte da zero.

L’idea è quindi questa: “precalcolare” le quantità che non variano ad ogni nuovo buffer e per quel dato valore di Tc, ossia di frequenza di campionamento. In particolare la quantità:

$$\mathrm{sinc}\left(\pi \frac{t - nT_c}{T_c}\right)$$

viene precalcolata tramite un thread a bassa priorità (una tantum) e posta nella matrice SINC[t][k] il cui significato delle due variabili indipendenti è il seguente.

- t = è il tempo t di cui si vuole sapere il valore del punto del segnale analogico ricostruito
- k = è il parametro su cui iterare per applicare la f.4.0 (che può arrivare ad essere compresa tra zero e tutti i punti del buffer)

Precalcolando queste quantità, le operazioni da compiere in tempo reale si limitano a 4096*400 operazioni per il disegno di una schermata, ossia 9.830.400 – 1.638.400 = 8.192.000 in meno ad ogni giro!

Prove sperimentali sulla (solita) macchina (composta da un pentium IV 1.6 Ghz con 512 Mb ram, 40960 Hz di frequenza di campionamento e 4096 punti di buffer) hanno rilevato un tempo aggiuntivo di calcolo nel ciclo principale pari a 5..10 millisecondi, rendendo l'algoritmo praticamente utilizzabile.

Valgono le seguenti considerazioni aggiuntive. Intanto le variazioni delle dimensioni del buffer possono influire significativamente sulle considerazioni sinora effettuate, in special modo laddove le dimensioni del buffer siano comparabili con le dimensioni della finestra oscilloscopio. Inoltre, una finestra grafica di dimensioni maggiori (per esempio di 1024 punti) aumenta significativamente il tempo di calcolo. Ancora, l'uso della funzione trigger può portare alla necessità di effettuare l'estensione del ciclo di calcolo della conversione digitale analogico a parte dei punti del buffer precedente

aumentando ulteriormente il carico computazionale. Per tali ragioni la conversione interna può essere disabilitata "manualmente" e persino quando abilitata, è automaticamente disinserita da Visual Analyser in talune particolari condizioni; per esempio, se si rilevano configurazioni tali per cui la sua applicazione non comporta miglioramenti apprezzabili nel disegno del segnale a video.

4.3 La trasformata di Fourier veloce

La spettro di un segnale è una possibile rappresentazione del segnale; quella considerata standard è quella detta nel *dominio del tempo* ossia il segnale viene rappresentato graficamente su assi cartesiani, in cui il tempo è riportato in asse X e l'ampiezza sull'asse delle ordinate. Per intenderci è la classica rappresentazione che usiamo in strumenti tipo oscilloscopio, ossia quella cui siamo più abituati. Un'altra rappresentazione dei segnali è quella che fa uso di una differente variabile indipendente : la frequenza. I segnali sono così rappresentati dalla frequenza in asse X e ancora dalle ampiezze in asse Y. I segnale è sempre lo stesso, cambia solo il modo di "vederlo"; questa seconda modalità è quella per esempio usata dagli analizzatori di spettro. Il fatto di analizzare uno stesso segnale da differenti punti di vista è cosa particolarmente utile giacchè talvolta consente di affrontare meglio particolari classi di problemi. Ad esempio, per testare la linearità in frequenza di un dispositivo elettronico (per esempio un filtro) è assai più agevole effettuare una analisi spettrale (ossia per esempio tracciare lo "spettro" di un segnale in uscita dal filtro tramite un analizzatore di spettro) per valutarne rapidamente la banda passante. Cosa per esempio molto meno agevole a farsi osservando lo stesso segnale nel dominio del tempo, ossia per esempio con un oscilloscopio.

Il passaggio da una rappresentazione all'altra, ossia per esempio da quella nel dominio del tempo a quella nel dominio della frequenza, è ottenuta tramite una operazione di trasformazione, detta *trasformata di Fourier*; essa nella sua forma più generale comprende segnali tempo continuo, ma ne esiste una versione specializzata per segnali tempo discreto .

Quest'ultima classe di segnali è quella per noi di maggiore interesse, visto che Visual Analyser tratta esclusivamente di questi (vedi paragrafo precedente). Per essi è definita quella che si chiama DFT (Discrete Fourier Transform) definisce una corrispondenza tra due sequenze discrete, di cui una (nel nostro caso) rappresenta la sequenza d'ingresso, ossia i campioni del segnale nel dominio del tempo, e l'altra è la sequenza d'uscita, ossia la voluta rappresentazione nel dominio della frequenza. Essa è espressa dalla relazione:

$$X_q = \mathcal{F}_d(x_n) = \sum_{k=0}^{N-1} x_k e^{-j\frac{2\pi}{N}kq}$$

(f.4.1)

Dove q e k variano tra 0..N-1 ed N è il numero di punti. Questo algoritmo presenta tuttavia una complessità computazionale proporzionale al quadrato dei punti, ossia come si suole dire è di "ordine" N*N ossia O(N*N).

Una versione senza dubbio più "economica" in termini di complessità di calcolo è la versione detta Trasformata di Fourier veloce (Fast Fourier Transform = FFT), dato che è accreditata di una complessità pari a O(N*log(N)), quindi particolarmente vantaggiosa al crescere del numero di punti.

L'algoritmo FFT più diffuso è l'algoritmo di Cooley-Tukey. Questo algoritmo si basa sul principio di *dividi e conquista* (dividi et impera), e spezza

ricorsivamente una DFT di qualsiasi dimensione N (con N numero composto tale che $N=N_1N_2$) in DFT più piccole di dimensioni N_1 e N_2. La (prima) versione più famosa fu sviluppata nel 1965 ad opera di J. W. Cooley e J. W. Tukey. Più tardi si scoprì che in effetti essa era stata già scoperta ad opera di Carl Friedrich Gauss nel 1805.

Il drastico miglioramento prestazionale della FFT rispetto alla DFT si comprende tramite i ragionamenti che seguono. La (f.4.1) viene suddivisa in due parti, una relativa al calcolo dei coefficienti di ordine pari e l'altra di quelli dispari, suddividendo di fatto il calcolo della DFT in due DFT più piccole (di dimensioni N/2). La divisione viene nuovamente ripetuta per ogni DFT (ottenendo complessivamente quattro DFT separate di dimensioni N/4) e così via sino a giungere a DFT che sono composte da due soli punti (decimazione nel tempo). Traduciamo il discorso qualitativo in formule; riscriviamo per comodità la (f.4.1) nella seguente maniera

$$V[k] = \Sigma_{n=0..N-1} W_N^{kn} v[n] \qquad \text{(f.4.2)}$$

Dove per semplicità di notazione abbiamo posto $W_N = e^{-2\pi j/N}$

Possiamo dunque scrivere:

$$V[k] = \Sigma_{n\ even} W_N^{kn} v[n] + \Sigma_{n\ odd} W_N^{kn} v[n] = \Sigma_{r=0..N/2-1} W_N^{k(2r)} v[2r] + \Sigma_{r=0..N/2-1} W_N^{k(2r+1)} v[2r+1]$$

$$= \Sigma_{r=0..N/2-1} W_N^{k(2r)} v[2r] + \Sigma_{r=0..N/2-1} W_N^{k(2r)} W_N^{k} v[2r+1]$$

$$= \Sigma_{r=0..N/2-1} W_N^{k(2r)} v[2r] + W_N^{k} \Sigma_{r=0..N/2-1} W_N^{k(2r)} v[2r+1]$$

$$= (\Sigma_{r=0..N/2-1} W_{N/2}^{kr} v[2r]) + W_N^{k} (\Sigma_{r=0..N/2-1} W_{N/2}^{kr} v[2r+1])$$

Dove si è posto:

$$W_N^{k(2r)} = e^{-2\pi i*2kr/N} = e^{-2\pi i*kr/(N/2)} = W_{N/2}^{kr}$$

I termini con k maggiore o uguale ad N/2 possono essere ulteriormente semplificati facendo uso di un'altra identità::

$$W_{N/2}^{m+N/2} = W_{N/2}^{m} W_{N/2}^{N/2} = W_{N/2}^{m}$$

Che è verificata, dato che $W_m^m = e^{-2\pi i} = \cos(-2\pi) + i \sin(-2\pi) = 1$.

Se usiamo un numero di punti N che è una potenza di due, questa suddivisione può continuare sino ad arrivare a DFT "elementari" di due punti.

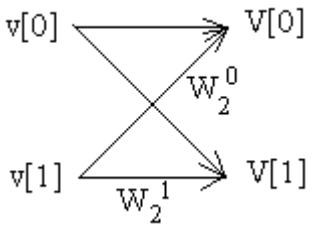
A questo punto proviamo a partire in senso contrario, ossia a considerare una DFT composta da soli due punti, ossia

$$V[k] = W_2^{0*k} v[0] + W_2^{1*k} v[1], \quad k=0,1$$

Le due componenti per esteso sono:

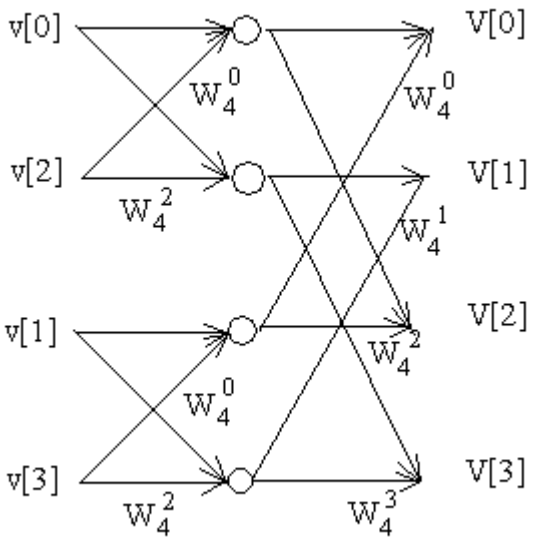
$$V[0] = W_2^0 v[0] + W_2^0 v[1] = v[0] + W_2^0 v[1]$$
$$V[1] = W_2^0 v[0] + W_2^1 v[1] = v[0] + W_2^1 v[1]$$

Che è possibile rappresentare con un grafo del tipo



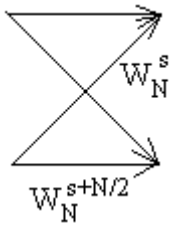
Dove in generale si assuma la convenzione che i rami senza "peso" esplicitamente indicato hanno valore unitario. Il grafo di cui sopra è equivalente alla relazione, nel senso che le variabili dei nodi d'uscita si calcolano moltiplicando le variabili dei nodi intermedi per i pesi del ramo.

Per esempio, ecco il grafo di una DFT su quattro punti:



Ottenuto notando che $W_{N/2}^{n} = W_N^{2n}$

E' evidente che l'elemento "atomico" che possiamo identificare è il seguente:

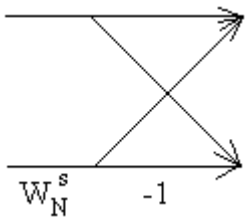


Ed utilizzando la relazione

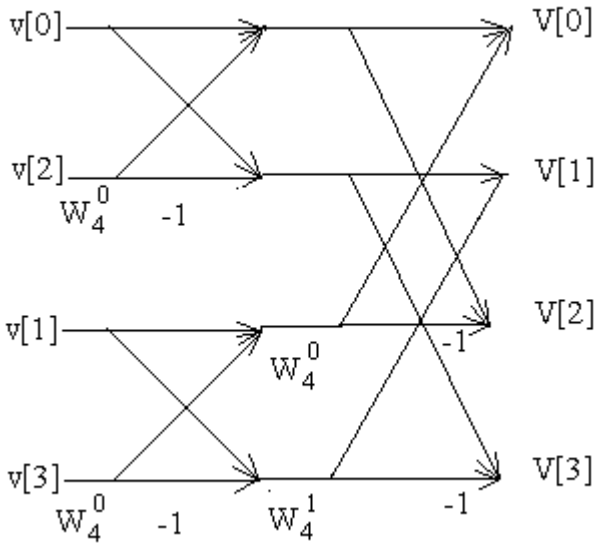
$$W_N^{s+N/2} = W_N^s \, W_N^{N/2} = -W_N^s$$

(vera perché $W_N^{N/2} = e^{-2\pi i(N/2)/N} = e^{-\pi i} = \cos(-\pi) + i\sin(-\pi) = -1$)

Otteniamo finalmente la versione più semplice (detta “butterfly” = farfalla):



Che ci consente di ridisegnare il grafodella DFT su 4 punti nella seguente maniera:



Che costituisce l’essenza della FFT. Il “trucco” principale è non calcolare ogni elemento della DFT separatamente. Piuttosto, i calcoli vanno fatti a “stadi”. Ad ogni stadio si inizia con N punti (in generale complessi) e si applica la “butterfly”per ottenere un nuovo set di N numeri. Numeri che a loro volta diverranno l’ingresso per lo stadio successivo. Per esempio nel caso di DFT su quattro punti, gli N numeri d’ingresso saranno i campioni originali, i quattro d’uscita i valori della trasformata di Fourier.

Si noti che ogni stadio comprende N/2 moltiplicazioni complesse, N/2 inversioni di segno (ottenute moltiplicando per -1) ed N addizioni complesse. Cosicché ogni stadio ha una complessità di ordine N; Il numero di stadi è log2(N), la complessità totale è pertanto Nlog2(N).

Ultima caratteristica da considerare è la possibilità di fare i "calcoli sul posto". Ponendo i campioni d'ingresso in un particolare ordine, che viene naturale effettuando una decimazione nel tempo, otteniamo l'importantissimo vantaggio di poter utilizzare le stesse locazioni di memoria per i dati in ingresso ed in uscita; ossia il vettore che passiamo in ingresso con i campioni da trasformare conterrà in uscita direttamente i campioni trasformati. Per esempio, consideriamo N=8 campioni di ingresso, rappresentati dall'indice (temporale) 0,1,2,3,4,5,6,7 che possiamo riscrivere in binario come:

000, 001, 010, 011, 100, 101, 110, 111

Effettuiamo la prima divisione in pari dispari:

[pari] (000, 010, 100, 110), [dispari] (001, 011, 101, 111)

Procedendo con lo stesso criterio:

((000, 100), (010, 110)), (001, 101), (011, 111))

Alla fine quindi abbiamo:

000, 100, 010, 110, 001, 101, 011, 111

Ossia, come è facile rendersi conto, abbiamo la stessa sequenza di partenza se invertiamo il posto dei singoli bit (000 = 000, 100 = 001 etc). E' facile rendersi conto che se utilizziamo un vettore dei dati in ingresso ordinato preventivamente in tal guisa otterremo l'indubbio vantaggio di poter effettuare in calcoli "sul posto" ossia utilizzando (per i campioni d'uscita) le stesse

locazioni di memoria appena prima appartenute ai campioni d'entrata (evitando così spreco di spazio e tempo).

Ora abbiamo tutti gli elementi per definire l'algoritmo FFT:

1. Selezionare un numero N potenza di due
2. Acquisire N campioni
3. Ordinare i campioni in ordine "bit reversed" e metterli in un buffer di N punti complessi (parte immaginaria = 0); è facile ottenere questo tramite un vettore intermedio di "traduzione" precalcolato.
4. Applicare il primo stadio di butterfly usando coppie adiacenti di campioni nel buffer
5. Applicare il secondo stadio di butterfly usando coppie separate da 2
6. applicare il terzo stadio di butterfly usando coppie separate da 4
7. Continuare applicando il butterfly nel buffer sino a che si ottiene una separazione di N/2
8. Il buffer contiene la trasformata di Fourier

4.4 Le finestre di smoothing

Il calcolo della trasformata di Fourier avviene elaborando un buffer alla volta, ed esso è acquisito direttamente dalla scheda sonora. Il problema che sorge utilizzando una simile architettura è relativo a quello che accade "agli estremi" del segnale. Immaginiamo per esempio una sinusoide ad una frequenza qualsiasi, e supponiamo di prelevare i campioni della sinusoide a blocchi di N campioni. E' piuttosto evidente che assai difficilmente preleveremo un numero esatto di periodi, e men che meno a partire (e finire) nel passaggio per lo zero. Ossia, data una sinusoide, Essa verrà campionata ed acquisita a blocchi di N punti; ecco la porzione di segnale acquisito effettivamente:

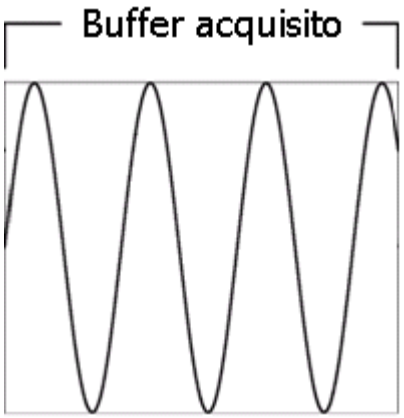


Figura 16: sinusoide campionata

Ossia il segnale NON sarà più una sinusoide, ma una sinusoide moltiplicata per due gradini, ossia una sinusoide che inizia e finisce "bruscamente", introducendo di fatto molte armoniche aggiuntive. Ecco lo spettro di un segnale del genere (figura 17) e lo spettro che dovrebbe avere (figura 18):

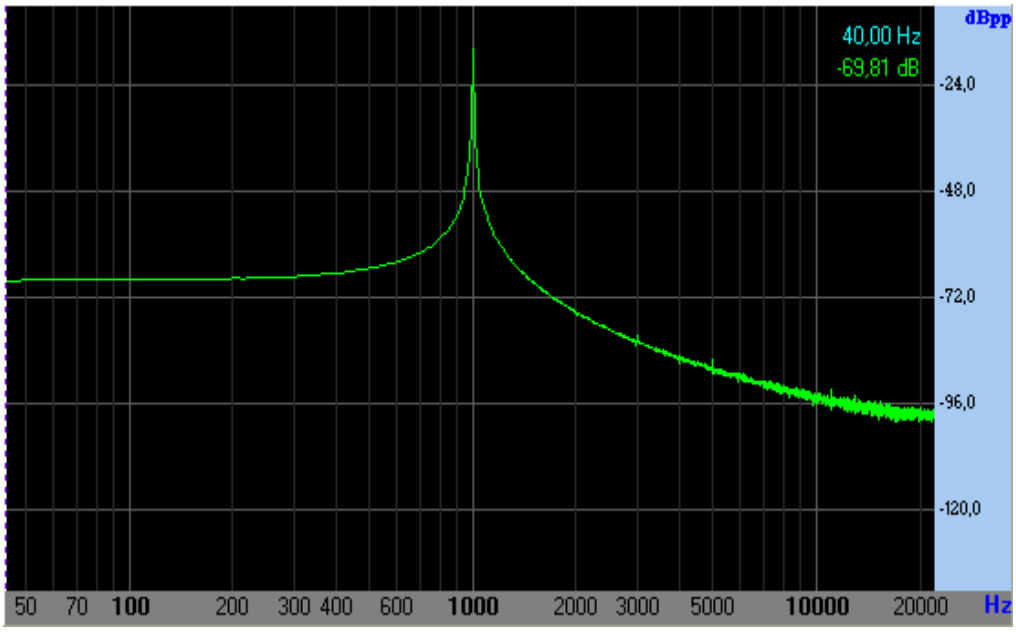


Figura 17: lo spettro del buffer acquisito

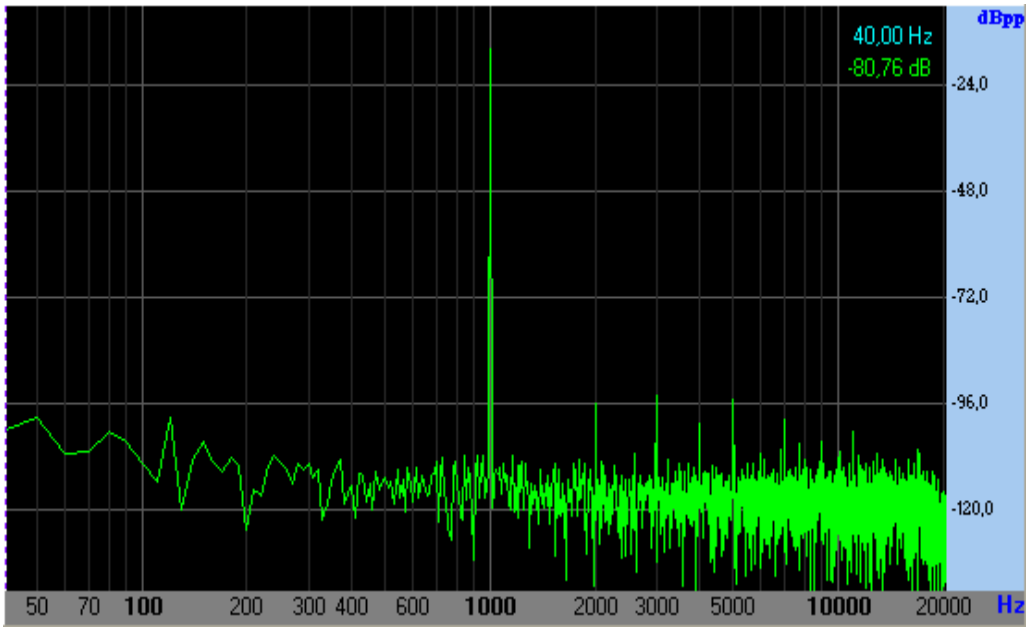


Figura 18: lo spettro teorico

Il problema non sussisterebbe se il segnale acquisito fosse costituito da un numero esatto di periodi, il cui primo periodo inizia esattamente dove l'ultimo finisce, per esempio quando interseca l'asse X (ossia per lo zero). Nella realtà non è possibile ricondursi a questo caso, e bisogna ricorrere ad altri sistemi.

La metodologia più seguita in questi casi è quella di usare le cosiddette finestre di "smoothing". L'idea è quella di attenuare il contributo dei segnale agli estremi (distorto), esaltando quello verso la parte centrale (non distorto). Una siffatta caratteristica si ottiene moltiplicando il segnale originale campionato per una curva (la finestra di smoothing) che abbia le caratteristiche descritte. Ecco un esempio qualitativo (volutamente esagerato):

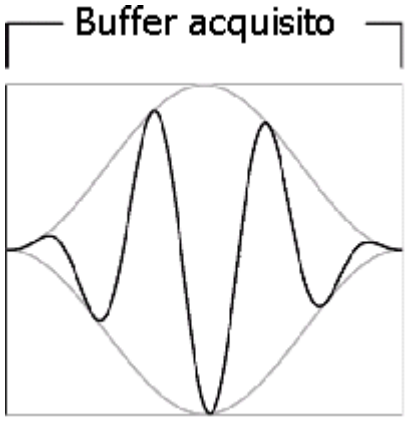


Figura 19: la finestra di smoothing

Come si vede, la curva utilizzata fa si che il segnale fornisca la maggior parte del contributo verso la parte centrale, che è poi quella a minore distorsione. In tal senso si opera anche una "periodizzazione" del segnale, cosa che può tornare utile per segnali poco periodici.

Esistono una notevole quantità di finestre di smoothing, e Visual Analyser consente di implementarne un certo numero tra le più diffuse; ed al limite di non utilizzarne nessuna (cosa spesso non permessa nemmeno in programmi commerciali). Le finestre implementate in Visual Analyser sono le seguenti:

a. Hanning
b. Blackman
c. Blackman optimized
d. Blackman Exact
e. Bartlett
f. Hamming
g. FlatTop

L'uso dell'una piuttosto dell'altra dipende da numerosi fattori, ma più tipicamente dalle caratteristiche del segnale che si sta analizzando e dall'applicazione. Ecco in rapida sintesi i vari campi d'uso delle diverse finestre:

Finestra	**Risoluzione in frequenza**	**Risoluzione in Ampiezza**	**Leakage Suppression**	**Applicazione**
Bartlett	Sufficiente	Sufficiente	Moderato	
Blackman	Sufficiente	Buono	Eccellente	Misure di distorsione
Flattop	Povero	Eccellente	Moderato	Misure di precisione

Hamming	Sufficiente	Sufficiente	Sufficiente	
Hanning	Sufficiente	Eccellente	Eccellente	Misure di Precisione, Misure di rumore
Triangolare	Sufficiente	Sufficiente	Povero	
Nessuna	Eccellente	Povero	Povero	Misure in frequenza ad alta risoluzione, Misure di risposta impulsiva

Le relazioni utilizzate per il calcolo delle varie finestre sono le seguenti:

Hanning:

w[i] = 0.5 * (1 – cos(2 * Pi * i / (n-1))

Hamming:

w[i] = 0.54 – 0.46 * cos (2* Pi * i/ (n – 1))

0 <= i <= n - 1

Blackman:

w[n] = (alpha + 1)/2 – 0.5 * cos (2 * Pi * n / (n -1))

– (alpha/2) * cos(4*Pi*n / (n – 1))

alpha = - 0.16 standard

alpha = -0.25 per grandi n

0 <= n < M - 1

Blackman Optimal:

w[i] = (alpha + 1) / 2 - 0.5 * cos (2* Pi * n / (n – 1))

- (alpha/2) * cos(4 * Pi * n / (n – 1))

alpha = - (sin (Pi / (n -1)) / sin (2 * Pi / (n – 1)))^2

Blackman Exact :

w[n] = 0.42659071 – 0.49656062 * cos (2 * Pi * n / (n – 1))
+ 0.07684867 * cos(4 * Pi * n / (n - 1))

0 <= n <= M – 1

FlatTop:

w[i] = 0.2810639 – 0.5208972 * cos (2 * Pi * i / (n - 1))
+ 0.1980399 * cos (4 * Pi * i / (n - 1))

0 < i < n – 1

Bartlett:

w [n] = 2*n / (M - 1)	0 <= n <= (M - 1) / 2
w [n] = 2 – 2*n / (M - 1)	(M – 1) / 2 <= n <= M - 1
w [n] = 0	per tutti gli altri valori di n

Da notare che l'applicazione delle varie finestre comporta una attenuazione del segnale che deve essere compensata con dei fattori correttivi dipendenti dalla finestra usata e dal tipo di visualizzazione (se lineare o logaritmica). I fattori sono stati individuati sperimentalmente applicando una sinusoide a 1000 Hz di ampiezza nota. Essi sono:

- Hanning: 6.02 dB logaritmica 2 unità lineare;
- Blackman: 7.53 dB logaritmica 2.375 unità lineare;
- Blackman optimal: 8.52 dB logaritmica 2.662 unità lineare;
- Bartlett : 6.02 dB logaritmica 2 unità lineare;
- Hamming: 5.36 dB logaritmica 11.50 unità lineare;

- FlatTop: 11.03 dB logaritmica 4.554 unità lineare;
- Blackman exact: 7.4 dB logaritmica 1.995 unità lineare.

4.5 Sviluppo in serie di Fourier

Lo sviluppo in serie di Fourier è una relazione che consente di rappresentare i segnali periodici come una sommatoria infinita di segnali elementari, tipicamente sinusoidi e cosinusoidi. La relazione finale per segnali reali e periodici di periodo due pi-greco, è la seguente:

$$x(t) = a_0 + \sum_{n=1}^{+\infty}\left[a_n \cos\left(\frac{2\pi}{T} nT\right) + b_n sen\left(\frac{2\pi}{T} nT\right)\right]$$

(f.4.3)

Come si vede essa richiede il calcolo dei coefficienti "a" e "b" che possono essere ricavati tramite le relazioni:

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} dx\, f(x)\cos(nx) \quad \text{e} \quad b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} dx\, f(x)\sin(nx)$$

(f.4.4)

La rappresentazione di una funzione tramite lo sviluppo in serie di Fourier è stata utilizzata in Visual Analyser per la sintesi delle forme d'onda nella funzione "generatore di funzioni". La generazione di forme avviene tramite la sintesi interna dei campioni che vengono successivamente passati alla scheda sonora. Essa li interpreta come campioni ottenuti da un processo di conversione analogico/digitale (a partire da un segnale analogico a banda limitata), e li passa al suo convertitore digitale analogico per ricostruire in uscita nuovamente il segnale analogico.

Generalmente si trascura il termine “a banda limitata”. E’ infatti opinione comune che per generare una forma d’onda arbitraria sia sufficiente:

- disegnare la forma d’onda arbitraria;
- prelevare un “tot” di campioni in accordo con una certa frequenza di campionamento;
- passarli alla scheda sonora.

In realtà agendo in siffatta maniera si commette un gravissimo errore; i campioni sono si apparentemente corretti, ma *non* sono relativi ad una forma d’onda a banda limitata, così come richiesto dal teorema del campionamento. Il problema d’altra parte è come fare ad individuare un algoritmo che a partire da una forma d’onda del tutto arbitraria (e quindi a contenuto armonico potenzialmente infinito, si pensi ad un impulso) ne ricavi la corrispondente rappresentazione a banda limitata. Né è possibile applicarvi un filtro digitale, giacchè esso suppone a sua volta una forma d’onda correttamente campionata (se si filtra una forma d’onda erroneamente campionata, si otterrà un filtraggio a sua volta erroneo).

La soluzione a questo problema e sintetizzare le forme d’onda volute a partire dal loro sviluppo in serie, ossia considerando le sole armoniche effettivamente comprese nella “banda limitata” che deve avere il segnale per effetto del campionamento prescelto. Ad esempio, per ricavare un’onda triangolare campionata a 44.100 kHz genereremo un segnale le cui componenti armoniche saranno ricavate dalle relazioni (f.4.3) e (f.4.4) e si arresteranno alla prima componente armonica la cui frequenza sarà maggiore di 22.050 Hz. In altre parole, ogni punto delle forme d’onda generate da Visual Analyser sarà costruita tramite una sommatoria che aggiunge un contributo per ogni seno e coseno coinvolti nello sviluppo in serie di Fourier. In questo modo eviteremo

l'insorgere di un fenomeno che abbiamo già preso in considerazione nel paragrafo 4.2, ossia *l'aliasing*

Allo scopo di semplificare le relazioni, quando possibile saranno ricercate particolari simmetrie che consentono di semplificare considerevolmente la (f.4.3). Per esempio nel caso di funzioni pari, ossia simmetriche rispetto all'asse delle ordinate, è possibile in molti casi ricondurre lo sviluppo in serie a somma di sole sinusoidi.

La sintesi delle varie forme d'onda è affidata ad una classe appositamente costruita che prende il nome di "*TGENDATA*"

4.6 Filtri digitali

I filtri digitali di tipo FIR (= Finite Impulsive Response) sono ottenuti per convoluzione del Kernel del filtro con il segnale appena copiato nei buffer interni dell'oggetto FFT. Come accennato nella sezione 2.7 essi sono costruiti usando un solo tipo di filtro di partenza poi opportunamente manipolato per ottenere gli altri: il filtro passa basso. Il kernel del filtro passa basso è calcolato a partire dalla relazione seguente:

$$h[i]=K\frac{\sin(2\pi fc(i-M/2)}{i-M/2}[0.42-0.5\cos(\frac{2\pi i}{M})+0.08\cos(\frac{4\pi i}{M})]$$

Tramite questa relazione viene calcolato il kernel "generale" che verrà usato per implementare:

- Passa basso
- Passa alto

- Passa banda
- Elimina banda

I coefficienti del filtro IIR notch è invece calcolato tramite le seguenti relazioni:

a0 = 1 – K
a1 = 2*(K – R)*cos(2*pi*f)
a2 = R*R – K
b1 = 2*R*cos(2*pi*f)
b2 = - R*R

mentre per il filtro notch inverso si ha:

a0 = K
a1 = -2*K*cos(2*pi*f)
a2 = K
b1 = 2*R*cos(2*pi*f)
b2 = - R*R

dove per entrambi vale:

$$K = \frac{1 - 2R\cos(2\pi f) + R^2}{2 - 2\cos(2\pi f)}$$

R = 1 – 3BW

BW = banda passante espressa come frazione della frequenza di campionamento, ossia variabile tra 0 e 0.5.

4.7 THD

Il calcolo della distorsione armonica totale (THD = Total Harmonic Distortion) è eseguito con una relativa semplicità tramite la sua definizione:

$$THD(\%) = 100 * SQRT[(V_2^2 + V_3^2 + V_4^2 + ... + V_n^2)] / V_t$$

Dove il generico termine V_n rappresenta il valore RMS dell'armonica ennesima, mentre il valore V_t rappresenta il valore RMS della somma di tutte le armoniche (comprensiva della fondamentale). Il calcolo della grandezza THD + noise, ossia della distorsione totale comprensiva del "rumore", è ottenuto nella medesima maniera ma aggiungendo al calcolo complessivo anche il contributo del rumore.

Capitolo 5

Parallelismo e programmazione concorrente

5.0 Introduzione

Si è usato come titolo di questo capitolo il termine volutamente generico "parallelismo" per indicare quello che alle volte, altrettanto impropriamente, è indicato come multitasking. Ponendoci come obiettivo di fare chiarezza nel corso di questo capitolo sulle terminologie usate e sulle implicazioni pratiche che esse comportano, vogliamo far notare come Visual Analyser sia un programma il cui modello programmativo predominante è quello concorrente, tramite uso intensivo del multithreading secondo le "regole" dettate dal sistema operativo "Windows" .

Accenniamo solo brevemente ai concetti principali che un campo così vasto ed interessante offre, solo per poter validamente introdurre le soluzioni adottate in Visual Analyser; ben coscienti, tra le varie cose, che i lettori di questo lavoro saranno certamente ben ferrati sulla materia, e che questo lavoro non vuole essere un trattato sui sistemi operativi ma piuttosto un esempio applicativo di determinatio concetti.

Eseguire più attività in parallelo è una delle caratteristiche che contraddistingue la maggior parte degli elaboratori moderni, e da sempre oggetto di studio dell'informatica. Abbiamo fondamentalmente due tipi di parallelismo:

- il parallelismo hardware;
- lo pseudo-parallelismo.

Il primo è quello più evidente; per esempio si potrebbe avere a che fare con una macchina composta da più CPU o per esempio una CPU e molte schede periferiche specializzate per adempiere a funzioni particolari (per esempio una scheda sonora che segue in loop un buffer di campioni audio, od un processore dedicato per operazioni di I/O). Un programma potrebbe pertanto essere eseguito da una CPU, un secondo dall'altra, mentre un terzo sfrutta la scheda sonora per produrre un segnale audio.

Il secondo tipo è quello che alterna l'uso di una stessa risorsa per compiti differenti in rapida sequenza, o su tempi più lunghi, per esempio dedica completamente la risorsa ad un solo compito (magari si sta scrivendo con Word, e non si sta usando nessun altro programma) salvo poi riprendere ad usare più programmi in rapida successione (si usa Word ma si è lanciato un programma di compressione audio mentre e si continua a digitare testo). In altre parole, il commutare l'uso della risorsa può avvenire su scale temporali diverse (millisecondi o minuti), ma sempre secondo il principio che solo un programma è effettivamente in esecuzione ad ogni istante. In definitiva lo pseudoparallelismo è dunque un parallelismo apparente, sebbene utile al pari del primo tipo. Con questa strategia si possono avere più processi che "competono" per l'uso concorrente di una stessa risorsa (la CPU), ed in effetti ad ogni istante solo un processo è effettivamente servito dalla risorsa. In definitiva questa strategia porta ad avere una macchina "virtuale" dedicata ad ogni singolo processo che generalmente dovrebbe apparire all'utente finale come (ovviamente) più lenta della macchina reale. In ogni modo date le alte

velocità in gioco, e data la natura di molti programmi, è una situazione che spesso non viene nemmeno percepita dall'utente finale, il quale ha come vantaggio la possibilità di avere in esecuzione contemporanea più programmi, pur avendo fisicamente disponibile una sola macchina.

L'elemento principale che consente di sfruttare al meglio il parallelismo (hardware o logico) da parte dei software applicativi è il sistema operativo; è evidente che scrivere un programma per una macchina potenzialmente parallela che non abbia un sistema operativo è in teoria certamente possibile, ma probabilmente troppo oneroso, e forse non giustificato di questi tempi. I differenti sistemi operativi consentono diversi tipi di strategie; Per esempio Windows XP consente di implementare un multitasking di tipo "preempitive" mentre il vecchio Windows 3.11 era di tipo "cooperative"; alcuni sistemi operativi permettono il "multithreading" ed altri i "processi leggeri", che sono caratteristiche ulteriori che si affiancano al generico concetto di multitasking (vedi prossimo paragrafo).

5.1 definizioni e terminologie

Sebbene, come detto nel paragrafo precedente, non si ritiene utile affrontare in questa sede uno studio sistematico della materia, diamo comunque un minimo di terminologia e discutiamo alcuni dei principi fondamentali.

Scopo del multitasking è quello di aumentare il throughput del sistema, ossia il *numero dei programmi eseguiti per unità di tempo.* Per far questo vengono caricati in memoria più programmi simultaneamente e ne vengono eseguiti alcuni in parallelo, utilizzando i diversi processori a disposizione; una configurazione tipica comune a molti personal computer è quella di avere una CPU ed un processore di I/O, che ancora consente una esecuzione parallela:

infatti l'esecuzione di un programma è normalmente un susseguirsi di operazioni che implicano l'uso della CPU e fasi in cui si eseguono operazioni di I/O che quindi lasciano inattiva la CPU. E viceversa. Nella fasi di attesa è dunque possibile mandare in esecuzione altri programmi. Così, un programma può proseguire la sua esecuzione stampando un documento e l'altro fare dei calcoli tramite la CPU.

Distinguiamo diversi tipi di parallelismo:

1. *Multitasking*: esecuzione di programmi indipendenti su CPU ed unità di I/O;
2. *Multiprogrammazione*: multitasking con l'aggiunta di tecniche di protezione della memoria che assicurano che un programma in esecuzione non possa accedere alle aree di memoria assegnate agli altri programmi;
3. *Multiprocessing*: multiprogrammazione estesa ad elaboratori dotati di più CPU e processori di I/O;
4. *Pseudo parallelismo*: con questo termine intendiamo il parallelismo apparente ottenuto commutando la CPU tra vari programmi dedicando a ognuno un "quanto" di tempo generalmente compreso tra le decine e le centinaia di millisecondi.

Cruciale è a questo punto la definizione di *Processo*: si definisce processo o "processo sequenziale" un programma in esecuzione, che include il valore del contatore di programma, registri e variabili. Il concetto di processo è necessario per distinguere *l'attività svolta da un processore dalla esecuzione di un programma;* infatti un processore in un determinato intervallo di tempo può alternativamente eseguire sequenze di istruzioni che appartengono a programmi diversi, mentre un determinato programma può essere sospeso e ripreso più volte. Quindi, il processo è associato alla effettiva esecuzione di un determinato programma; il programma in se è la somma di tutte le sue istruzioni ed

operandi, il processo è associato all'avanzamento di quel determinato programma.

Un processo può essere in esecuzione o meno: il compito del sistema operativo diventa quello di controllare lo stato di avanzamento di un gruppo di processi con due obiettivi: sfruttare al meglio i differenti processori e soddisfare le richieste degli utenti.

Il processo può essere in tre differenti stati:

- *In esecuzione*: quando un processore sta eseguendo le istruzioni del programma associato a quel processo
- *Pronto* : è fermo in attesa di un processore libero
- *Bloccato* sull'evento E: sta attendendo, per poter proceder, il verificarsi di un determinato evento

Diamo ora un cenno alle due grandi modalità di gestione dei processi che riguardano il mondo Windows; esso è di tipo "preempitivo" sin dalla versione 95 (sebbene non completamente). Un sistema operativo è preempitivo se la commutazione di risorse assegnate ai processi avviene "d'ufficio" per azione del sistema operativo. Ossia, non c'è possibilità da parte dei programmi applicativi di ottenere una accesso esclusivo ad una risorsa (per esempio la CPU) e quindi un programma non è in grado di bloccare la macchina. Viceversa, la modalità utilizzata in Windows 3.x era di tipo "cooperative", ossia i programmi dovevano prevedere espressamente il rilascio della risorsa, per consentire ad altri processi di avanzare. Se una applicazione di questo tipo andava in crash, molto probabilmente il sistema doveva essere fatto ripartire.

Invero, una gestione completamente preempitiva non era stata correttamente implementata in Windows 95 che doveva pur sempre mantenere la compatibilità verso le "vecchie" applicazioni del modo a 16 bit facenti capo a

Windows 3.11, e quindi in definitiva al DOS. Ossia, un'applicazione a 16 bit veniva gestita in maniera "cooperativa" e poteva in definitiva ancora bloccare il sistema.

Un'ultima osservazione circa vantaggi e svantaggi. E' pur vero che il modello preempitive è certamente più vantaggioso sotto quasi ogni profilo; eccezion fatta per l'overhead che inevitabilmente una gestione così sofisticata introduce. In tutta probabilità, due o tre applicazioni "ben scritte" che girano su di una macchina "cooperativa" spuntano certamente tempi di esecuzione medi molto migliori.

5.2 programmazione concorrente

I programmi che sono in grado di sfruttare i diversi gradi di parallelismo sommariamente descritti nel paragrafo precedente sono detti "programmi concorrenti". Il supporto offerto dal sistema operativo alla programmazione concorrente è ovviamente diverso da caso a caso (esempio Windows ammette il multithread e Linux i processi leggeri).

I programmi sono pertanto genericamente divisi in due grandi categorie:

- Sequenziali
- Concorrenti

I primi danno luogo alla esecuzione di un solo processo, i secondi a più di un processo. Da osservare che è possibile realizzare tutto quello che si può realizzare con un programma concorrente con un programma sequenziale. La differenza sostanziale nell'usare il modello concorrente è ovviamente nell'efficienza di sfruttamento delle risorse, e nella maggiore facilità "logica"

nella scrittura di un programma. Talune classi di problemi sono per loro natura portate ad essere spontaneamente suddivise in una serie di attività parallele. Si pensi per esempio a Visual Analyser; è logico pensare ai singoli strumenti simulati dal programma come attività a se stanti; oppure, scendendo di livello di astrazione, di pensare alle attività di acquisizione dati, gestione interfaccia utente e disegno a video come attività distinte e capaci di sfruttare differenti risorse del sistema in parallelo.

Un'altra caratteristica importante da considerare oltre al concetto di processo è la *sincronizzazione* tra essi; in generale è vero che i singoli processi vivono autonomamente, ma nella maggior parte dei casi essi concorrono alla realizzazione di un obiettivo comune, e devono pertanto in qualche misura interagire tra loro e scambiarsi dati; la misura dell'interazione prevede la possibilità che un processo aspetti che un altro processo abbia finito di effettuare determinate azioni (e magari produrre determinati dati) prima di portare avanti le proprie. Per esempio aspettare che i dati letti dalla scheda sonora siano pronti per essere elaborati e successivamente stampati a video. Oppure, se completamente svincolati tra loro, devono cercare almeno di non ostacolarsi a vicenda.

Nei sistemi operativi più classici i singoli processi sono provvisti di un loro proprio "address space", normalmente non accessibile da altri processi. Inoltre, ogni processo ha un singolo "filo" (thread) di esecuzione, ossia al suo interno si sta eseguendo un'attività strettamente sequenziale. La sincronizzazione tra processi è altresì fattibile, anche con scambio di dati, ma facendo uso di primitive messe a disposizione dal sistema operativo stesso. Un'altra importantissima variante, comprende l'uso di processi che al loro interno consentono l'esecuzione di più thread di esecuzione parallela (vedi paragrafo successivo).

5.2.1 I thread

Un altro modo di vedere i processi è che essi costituiscono un modo di raggruppare una serie di risorse tra loro relazionate. Un processo ha un "address space" che contiene il codice del programma ed i dati, al pari di altre risorse; esse possono essere file aperti, processi figli, informazioni di account etc. Raggruppandoli sotto forma di processo essi possono essere gestiti molto più facilmente. Un processo ha anche quello che si definisce "thread" di esecuzione. Il thread ha un program counter che mantiene traccia della prossima istruzione da esguire; ha dei registri che contengono le variabili di lavoro; ha uno stack, che contiene la "storia" dell'esecuzione con la traccia delle procedure chiamate e non ancora rientrate. I tal senso il thread può essere visto come un concetto separato dal resto: i processi sono usati per raggruppare delle risorse; i thread sono le entità schedulate dalla CPU per essere eseguite.

L'idea è quindi chiara; consentire all'interno di un processo l'esecuzione di più thread di esecuzione. Ossia, in parole molto semplici e imprecise, consentire ancora l'esecuzione di più programmi paralleli all'interno di un processo. Invero, la differenza fondamentale tra le due cose è che mentre un processo ha un suo address space e tutta una serie di risorse associate, i thread *condividono* tutto tra loro eccetto il program counter, uno stack dedicato (che in Windows sono due) e l'esecuzione.

In altre parole ancora, tramite i thread è possibile scrivere un programma che da vita ad un processo, il quale al suo interno da origine a più attività parallele. Che è esattamente quello che fa Visual Analyser. Il vantaggio nell'uso di questo approccio è notevole; intanto la comunicazione e sincronizzazione è molto più agevole, visto che i thread condividono lo stesso address space e quindi, in definitiva, le stesse aree di memoria. La *commutazione di contesto* è particolarmente "veloce" nel senso che le operazioni che il sistema operativo deve fare per assegnare la CPU da un thread all'altro sono in numero

considerevolmente minore rispetto ad quelle necessarie per effettuare la stessa operazione tra due processi. Il rovescio della medaglia è che un thread "può fare danni" nel senso che può effettuare delle azioni che pregiudicano la corretta esecuzione degli altri thread. Del resto si suppone che tutti i thread siano stati scritti per il raggiungimento di un fine comune, e quindi in tal senso è nell'interesse di tutti cooperare costruttivamente. Mentre i processi appartengono generalmente a programmi diversi che nemmeno sanno dell'esistenza gli uni degli altri (si pensi ad un editor di testi in esecuzione contemporanea a Visual Analyser) e che pertanto, anche involontariamente, potrebbero arrecarsi danni vicendevolmente; ecco il perché di tanta cura nel "separare" logicamente i due ambienti.

Un thread può trovarsi nelle medesime condizioni di un processo, ossia può essere nello stato di "pronto", "bloccato" ed in "esecuzione". Esso può essere gestito a livello di Kernel, come un normale processo, oppure gestito a livello "user". In quest'ultimo caso il sistema operativo è invero non direttamente coinvolto nella gestione dei thread. In tal senso la gestione dei thread in modalità "user" consente di utilizzare dei sistemi operativi non multithread per implementare comunque il multithread (tramite l'uso di programmi aggiuntivi e compilatori scritti "ad hoc").

5.3 Windows e multithread

Veniamo ora al paragrafo principale del capitolo, dove parleremo brevemente dell'implementazione Windows del multithreading. Esso è stato implementato per la prima volta in Windows NT, e ben presto in windows 95 e tutte le versioni successive sino all'attuale XP e prossimo Vista.

L'implementazione Windows comprende la gestione del multithread in Kernel mode ed anche in User mode. Windows dalla versione 95 in poi (ed NT) è un sistema operativo di tipo "preempitive".

In Windows un processo comprende:

- un address space privato dove il codice del processo ed i dati sono memorizzati;
- un "access token" sul quale Windows fa dei controlli di sicurezza;
- risorse di sistema, quali file e finestre (rappresentati da "handle");
- almeno un thread per eseguire il codice.

Mentre un thread comprende:

- uno "stato del processo" che include il puntatore all'istruzione corrente;
- uno stack da usare quando gira in user mode;
- uno stack da usare quando gira in kernel mode.

Dal momento che i processi posseggono l'access token, gli handle delle risorse e l'address space, i thread di converso non li hanno; perciò tutti i thread condividono la stessa memoria, gli access token e le risorse di sistema del processo in cui sono generati. E' lasciato pertanto al programmatore un uso corretto di queste risorse condivise. Per gestire al meglio questa condivisione di risorse, è necessario fornire delle primitive di sincronizzazione e dei meccanismi che consentano l'accesso "esclusivo" a regioni di memoria condivise, per evitare conflitti di accesso contemporaneo.

Le primitive di sincronizzazione e gli "oggetti" che consentono di regolamentare l'accesso alle regioni di memoria di accesso comune (le famose "critical section") sono gestite da programma tramite API di sistema e strutture dati predefinite. In particolare, la maggior parte dei linguaggi di

programmazione moderni, generalmente di tipo “obiect oriented”, forniscono una libreria di classi che standardizza l’uso di queste API.

Per esempio, in Borland C++Builder, esistono differenti classi predefinite che risolvono problemi di sincronizzazione e arbitrano gli accessi ad aree di memoria condivise; per quest’ultima funzione si può usare la classe “*TCriticalSection*”, di cui Visual Analyser fa uso per proteggere una “sezione critica” (vedi listato 3 paragrafo 2.2.1) . L’uso è relativamente semplice ed efficace; prima di accedere alla sezione di memoria il thread deve obbligatoriamente effettuare la chiamata al metodo “*Aquire*”. Qualsiasi altro thread che voglia accedere alle medesime regioni dovrà effettuare la stessa chiamata; qualora la regione fosse già in uso, esso verrà sospeso e messo in una coda d’attesa (perché possono essere presenti più thread che hanno già effettuato un tentativo d’accesso). Alla fine, ogni thread che ha finito di utilizzare la regione di memoria dovrà “liberarla” effettuando una chiamata al metodo “Release”. Come detto a più riprese, la classe “*TCriticalSection*” incapsula gli strumenti forniti come API dal sistema operativo. L’altro oggetto utilizzato da Visual Analyser per scopi di sincronizzazione è un oggetto di tipo “Event”. Per questo tipo di oggetto si è optato per un uso quasi diretto delle API relative.
In tal senso, è stata definita autonomamente una classe chiamata “*TEvent*”; essa è stata usata nel thread principale “*GetData*” che è stato analizzato sempre nel paragrafo 2.2.1, listato 3; essa semplifica l’uso delle API “*CreateEvent*” e “*WaitForSingleObject*”.
Per implementare i thread di Visual Analyser è stata usata la classe predefinita “*TThread*” che, come al solito, utilizza le principali API di sistema usate per allocare i Thread (*CreateThread*, *TerminateThread* etc).

5.4 Programmi in tempo reale

In definitiva quindi, Visual Analyser fa uso di Thread, con un vincolo aggiuntivo: il programma ricade nella categoria dei programmi in "tempo reale".

Un programma in tempo reale è *un programma concorrente che deve soddisfare dei vincoli relativi ai tempi d'esecuzione di almeno uno dei sottoprogrammi che lo compongono.*.

In particolare, esistono sistemi operativi appositamente progettati per lo sviluppo ed esecuzione di programmi in tempo reale. Ad esempio il sistema operativo OSE utilizzato su moltissimi microprocessori e DSP della Texas Instruments; oppure, per rimanere in ambiti più noti, sino a non molto tempo fa era frequente sentire parlare di "Dos real time". Ancora, un noto sistema operativo per "Sistemi embedded" è il famosissimo VXworks. Tutti questi sistemi operativi consentono una programmazione di tipo concorrente particolarmente efficiente ed orientata alla scrittura di programmi in tempo reale con vincoli temporali strettissimi ed affidabilità molto elevata.

Una tipica ed eclatante applicazione dei concetti di "tempo reale" la si ha per esempio nel caso di "software avionici", ossia quella particolarissima categoria di software embedded che viene utilizzato per il governo degli aerei militari e di linea. E' noto infatti che un moderno aereo da caccia militare è progettato per avere delle prestazioni tali che è possibile ottenere solo facendo lavorare l'aereo in un "punto di lavoro" aerodinamico con costanti di tempo particolarmente ridotte; ossia che costringono all'uso di elaboratori dedicati per il mantenimento di un corretto assetto di volo. Questi debbono pertanto funzionare con vincoli temporali estremamente ridotti; essere pesantemente ridondati per assicurare continuità di funzionamento anche in caso di failure; gestire numerose altre funzioni anch'esse in tempo reale, quali GPS (o DMG,

Digital Map Generator) o strumenti per gestire formati di navigazione (esempio ADI o HSI).

Windows di per se non nasce come sistema operativo orientato al “real time”, sebbene negli ultimi anni la crescente richiesta di multimedialità ha fatto si che “de facto” il PC debba gestire delle applicazioni che tipicamente girano in tempo reale, come per esempio l’esecuzione di musica, filmati e riconoscimento vocale. In tal senso, Windows (e Linux) che sono dei sistemi operativi general pur pose, diventano adatti anche a far girare applicazioni in tempo reale spinto. Visual Analyser è la dimostrazione di come un sistema operativo come Windows possa comunque essere utilizzato con successo con programmi real time che vanno ben oltre al semplice compito di ascoltare un brano musicale e visualizzare un filmato.

Capitolo 6

Gli strumenti simulati

6.0 Introduzione

Gli strumenti simulati da Visual Analyser sono:

- Oscilloscopio (par. 6.2);
- Analizzatore di spettro (par. 6.3);
- Generatore di funzioni *senza aliasing* (par. 6.4);
- Frequenzimetro (par. 6.5);
- Voltmetro (par. 6.6);
- Filtri digitali (par. 6.7);
- Cattura segnali nel dominio del tempo e frequenza (par. 6.8);
- Distorsiometro (THD, THD+noise) ;
- Rilevazione automatica della risposta in frequenza;

In questo capitolo cercheremo di descrivere dettagliatamente le singole funzioni implementate da un punto di vista operativo, ossia di chi deve usare il programma, cercando di identificare i limiti e le possibili future estensioni del programma. Visual Analyser si installa tramite un programma di installazione

realizzato con la versione di InstallShield a corredo di tutte le versioni dei compilatori di casa Borland. Una volta installato esso risulta costituito da un unico file eseguibile comprensivo di ogni libreria utilizzata (ossia autosufficiente) più un set di file aggiuntivi che comprendono i vari file di help, i file di esempio di calibrazione ed i file di esempio per il generatore di funzioni. Il percorso di installazione è definibile dall'utente; viene comunque proposto come default il percorso c:\programmi\SillanumSoft. All'occorrenza, il file VA.exe installato può essere copiato, utilizzato e ridistribuito senza bisogno di alcuna installazione.

Visual Analyser è utilizzabile in due grandi modalità: la prima, detta "standard", è quella di default e comprende una finestra principale (ridimensionabile a partire da un minimo) che contiene:

- Oscilloscopio;
- Comandi principali oscilloscopio;
- Analizzatore di spettro;
- Comandi principali analizzatore di spettro;
- Comandi vari;
- Barra dei comandi.

L'ultimo punto, ossia la barra dei comandi, è quella caratteristica che rimane sempre presente anche se si decide di utilizzare la seconda modalità permessa, ossia la "floating windows" (a finestre flottanti). In questa seconda modalità la finestra principale si riduce alla sola barra dei comandi (v fig. 20), e tutte le restanti funzioni compaiono come finestre separate e svincolate dalla finestra principale (se espressamente invocate). Per commutare nella modalità "floating windows" è sufficiente cliccare sul bottone "floating windows mode" nella barra dei comandi.

Figura 20: la barra dei comandi

Prima di procedere oltre nella descrizione delle varie funzionalità, è importante descrivere la finestra di "Settings" che costituisce, dopo la finestra principale, quella di maggior importanza per un corretto e agevole uso del programma.

6.1 La finestra di Settings

Essa può essere invocata cliccando sul bottone "Settings" nella barra dei comandi, ed appare come nella figura 21.

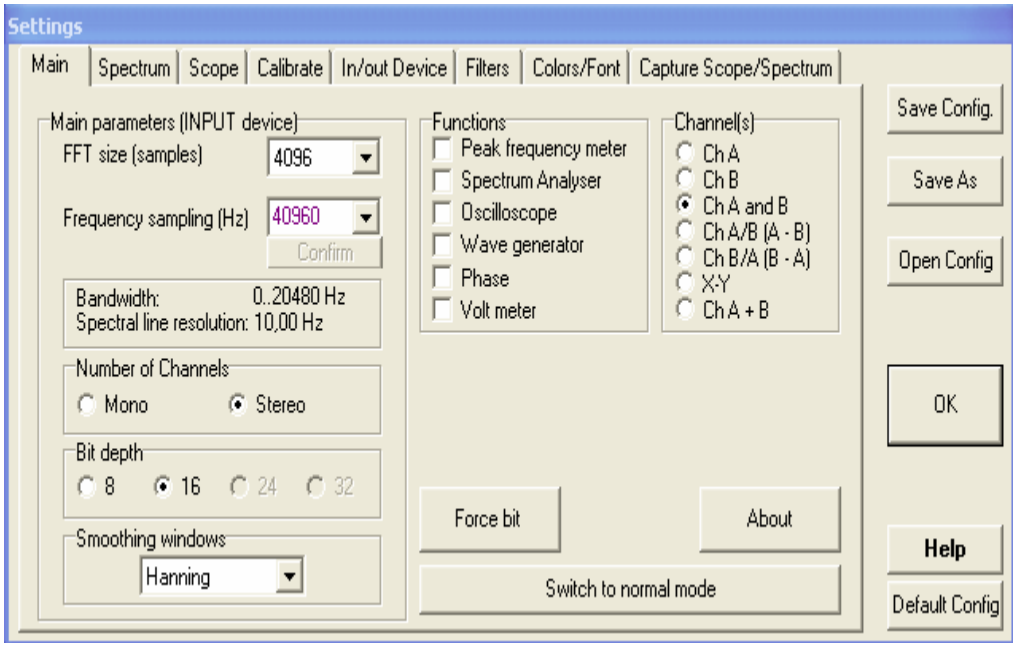


Figura 21 : la finestra di settings

Invero, risulta costituita di otto sottocartelle opportunamente raggruppate tramite un "tabbed notebook" che consente di risparmiare una considerevole dose di spazio e raggruppare utilmente i comandi per categoria. Essa consente di accedere a *tutti* i comandi e opzioni di configurazione di Visual Analyser;

risulta invocabile dalla barra dei comandi ma anche dalle singole finestre separate (quando in modalità a finestre flottanti).

La prima cartella (Main), visualizzata in figura 21, consente di:

1. [Main parameters] definisce i parametri principali e generali del programma, ossia le dimensioni del buffer di acquisizione, la frequenza di campionamento, il numero di canali usati e la dimensione in bit usata per convertire i campioni.
2. [Functions] consente di invocare le varie funzionalità del programma.
3. [Channels] consente di definire quali canali usare. In particolare
    - solo canale sinistro (A);
    - solo canale destro (B);
    - canale destro e sinistro (A and B);
    - canale sinistro meno destro (in dB) = funzione di trasferimento;
    - lo stesso del punto precedente con i canali invertiti;
    - modalità XY;
    - A+B somma dei due canali.
4. [Switch to normal mode] bottone che consente di commutare tra le due modalità “standard” e “finestre flottanti”.

6.1.1 La sottocartella Spectrum

La seconda cartella presente nella finestra di settings è relativa ai settaggi peculiari della funzione analizzatore di spettro. Come più volte ripetuto, essi sono parzialmente replicati direttamente nella finestra principale, per comodità d’uso. Inoltre esse sono in parte presenti anche nella versione a finestra flottante(v. sezione 6.3).

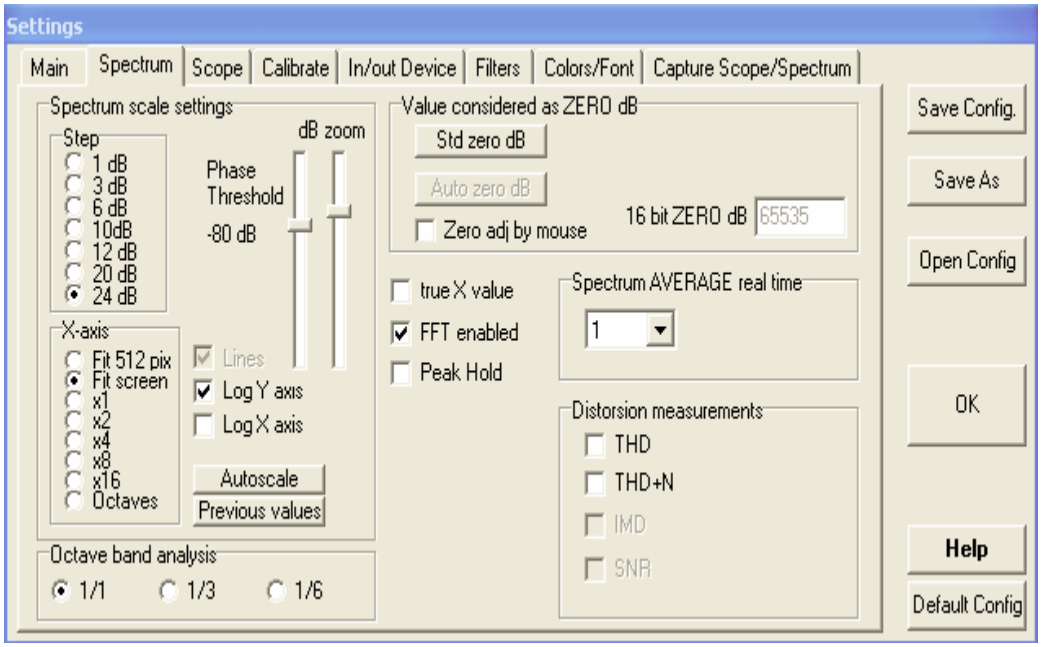


Figura 22 : la cartella Spectrum

Essa consente di:

1. [Spectrum scale settings] definisce parametri relativi al "passo" in dBr, alle modalità di rappresentazioni degli assi (lineari/logaritmici), ed al livello di zoom. In particolare può attivarsi la funzione di "autoscale" e la funzione di ripristino del valore precedente di scala. Inoltre, si può definire la modalità di rappresentazione dell'asse X come descritto nella sezione 6.3.
2. [Octave band analysis] seleziona la modalità di rappresentazione quando attivata la funzione "Octaves".
3. [Value considered as ZERO dB] questa importantissima funzione, poco usata in programmi similari, consente di ridefinire il livello di zero dB, anche dinamicamente tramite il mouse (trascinando direttamente il mouse sulla scala delle Y dell'analizzatore di spettro tenendo il bottone sinistro premuto). All'attivazione di questa funzione la scala delle Y cambia colore.

4. [true X value] consente di visualizzare sull'asse X il vero valore rappresentato e non una sua approssimazione.
5. [FFT enabled] normalmente sempre attivata consente di disattivare il calcolo della FFT se non utilizzata (liberando una notevole quantità di carico). Da notare che altre funzioni correlate cesseranno di funzionare (ossia quelle funzioni che si basano sul calcolo dello spettro, come ad esempio il frequenzimetro).
6. [Spectrum AVERAGE real time] consente di abilitare una media matematica sugli ultimi "n" buffer, funzione molto importante quando si vuole ridurre il tasso di rumore del segnale visualizzato. Da non confondere con l'analoga funzione implementata per la finestra di cattura spettro (vedi sezione cattura spettro).
7. [Distorsion measurements] Serve ad attivare la misura del valore di THD e THD+noise, visualizzata in alto a destra della finestra analizzatore di spettro. Le funzioni IMD (Inter Modulation Distortion) e SNR (Signal to Noise Ratio) sono relative a funzioni in fase di implementazione (all'atto della scrittura del presente testo Visual Analyser è alla versione 8.10).

6.1.2 La sottocartella Scope

La terza sottocartella è relativa alle opzioni dell'oscilloscopio. Come per le altre sottocartelle le opzioni in essa presente sono parzialmente replicate nella finestra principale per motivi di praticità.

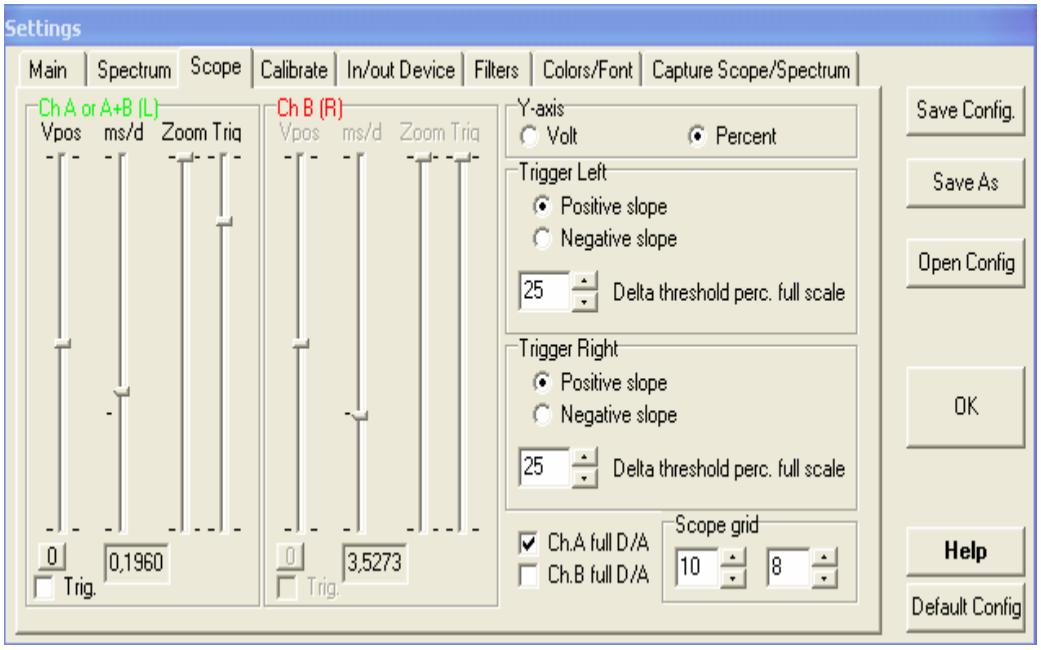


Figura 23 : la cartella Scope

Le opzioni disponibili in questa cartella sono le medesime che è possibile trovare nella finestra principale con qualche piccola funzione aggiuntiva di uso molto meno frequente; da osservare che talune opzioni sono fruibili tramite trackbar, mentre nella finestra principale tramite caselle di edit coadiuvate da spinbutton. Invero questo deriva da una fase iniziale di trasformazione, poi non portata a termine perché si è constatato che in talune occasioni era più utile avvalersi di una trackbar e talvolta di una casella di testo. Lasciando le due modalità si consente di operare al meglio in funzione delle esigenze.

Le opzioni che si trovano in questa sottocartella sono quasi tutte duplicate (per la presenza di due canali separati); ad eccezione di [Y-axis] e [scope grid] che restano valide per entrambi i canali. Le opzioni presenti dunque sono le seguenti:

1. [Vpos] è la posizione verticale del grafico

2. [ms/d] è la base dei tempi; essa è relativa alla divisione orizzontale, ossia dell'asse delle X. Il valore indicato dalla casella di testo alla base della trackbar indica il valore del tempo compreso nell'intervallo. Purtroppo, nella maggior parte delle configurazioni questo non potrà essere un comodo valore intero. Questo è anche logico e relativamente inevitabile perché abbiamo una risoluzione orizzontale con un numero di pixel discreti, ed ognuno è generalmente relativo ad un campione, che a sua volta dipende dalla frequenza di campionamento scelta. Inoltre, una "divisione" orizzontale da un tempo generalmente pari alla somma di "n" valori decimali, a sua volta decimale. Per esempio, scegliendo una frequenza di campionamento pari a 44100 Hz avremo che tra due pixel il tempo intercorso è di 1/44100 secondi, quindi decimale. Invero, settando una frequenza di campionamento pari a 40960 Hz, ed aggiustando opportunamente le dimensioni della finestra (il numero di divisioni orizzontali resta lo stesso, ergo aumenta il numero di pixel per divisione orizzontale) si arriva rapidamente ad avere una scala di time division a passi di un millisecondo.
3. [Zoom] è il fattore di zoom applicato alla scala delle Y; essa non amplifica il segnale ma lo "dilata" lungo l'asse Y variando di conseguenza le scale.
4. [Trig] varia la soglia di intervento del trigger; il trigger implementato in Visual Analyser prevede l'individuazione del ripetersi di una particolare soglia in una forma d'onda periodica. Abilitando il trigger tramite la spunta della checkbox "Trig", compare una barra orizzontale tratteggiata che è possibile spostare verticalmente tramite la trackbar del trigger. Essa individua la soglia di intervento; ogni qual volta la forma d'onda del buffer corrente sorpassa il valore di soglia, essa viene effettivamente disegnata a video a partire da quel preciso punto. Per evitare falsi riconoscimenti di soglia, è possibile definire una "finestra" d'intervento, ossia impostare un range di intervento composto di una serie di valori consecutivi (entro un intervallo definito) che determini

l'itervento del trigger solo dopo che la soglia è stata superata per un "tot" di campioni (o equivalentemente per un certo intervallo di tempo). Questi parametri vengono impostati nel riquadro "Trigger left" e "Trigger right", dove è anche possibile definire se intervenire sul fronte ascendente e discendente della forma d'onda.

5. [Y-axis] definisce se usare la scala in Volt o percentuale fondo scala.
6. [Trigger left/ Right] vedi punto 4.
7. [Ch A full D/A] abilita la conversione digitale analogica interna di Visual Analyser relativamente al vanale sinistro .
8. [Ch B full D/A] abilita la conversione digitale analogica interna di Visual Analyser relativamente al vanale destro.
9. [Scope Grid] consente di variare il reticolo dell'oscilloscopio, per esempio per rendere il time division un numero intero.

6.1.3 La sottocartella Calibrate

Questa sottocartella è relativa ad una delle funzioni più importanti di Visual Analyser in quanto consente di tarare direttamente in Volt tutte le scale dei molteplici strumenti simulati dal programma (sebbene in generale sarebbe possibile tararla in qualsiasi unità di misura, visto che il segnale elettrico ottenuto è in definitiva un segnale ottenuto da un trasduttore che potenzialmente può leggere un qualsiasi tipo di grandezza).

La scala "di base" (di default) è quella relativa ai "dB relativi" e/o percentuale fondo scala. Essa parte dalle considerazione che i campioni manipolati da Visual Analyser sono in realtà rappresentati da grandezze intere ben definite; tipicamente a 16 bit ma potenzialmente a 8, 16 e 24 bit. In ogni caso sono delle grandezze intere in base due il cui numero massimo di possibili valori rappresentati è due (la base) elevato al numero di bit.

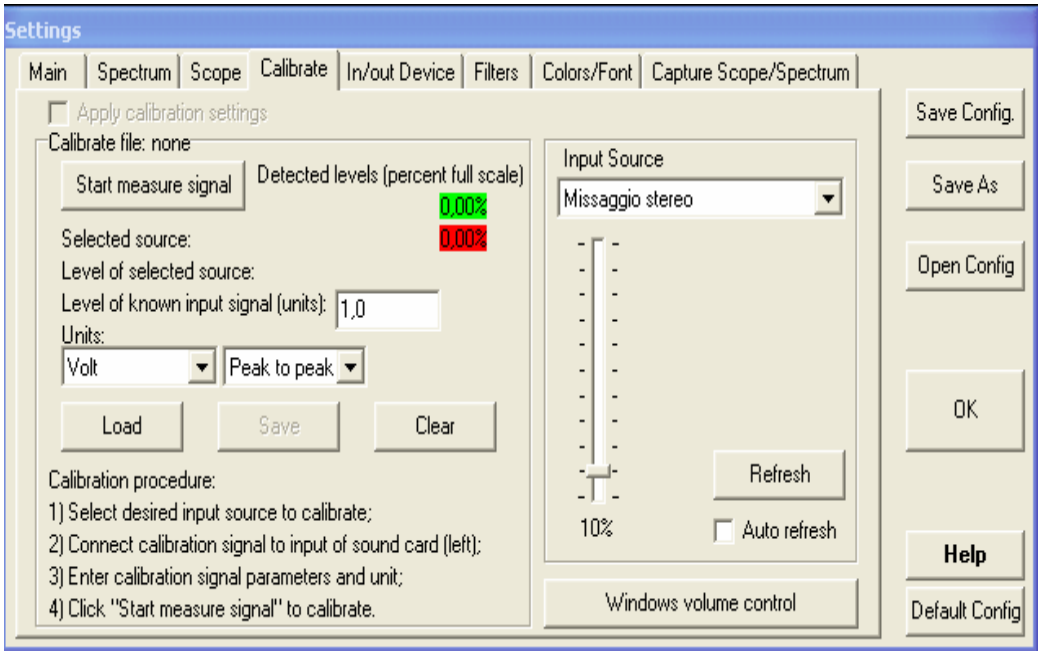


Figura 24 : la sottocartella Calibrate

Così, usando il valore di 16 bit, abbiamo che il massimo numero di valori che possiamo rappresentare è 65536. In tal modo è semplice rappresentare le scale in termini di percentuale fondo scala; avremo per esempio il 100% del segnale quando i campioni raggiungeranno il valore limite di 65536. Lo stesso dicasi per il livello in dB relativi: fissato un valore inteso come zero dB (per esempio il massimo valore) tutto il resto sarà rappresentato di conseguenza (da zero a meno infinito).

Una simile strategia è talvolta sufficiente, ma in taluni casi inadeguata; inoltre la simulazione completa di taluni strumenti (per esempio l'oscilloscopio) non può non comprendere una taratura (almeno) in Volt.

Il problema è che la conversione operata dalla scheda attraverso il convertitore analogico digitale è fatta dopo aver arbitrariamente amplificato il segnale. Quasi nessuna scheda fornisce indicazioni a riguardo, e talvolta nemmeno sull'impedenza d'ingresso (vedi capitolo scheda sonora). A complicare le cose, esiste sempre la possibilità di intervenire sul "Mixer volume control" di

Windows e quindi variare ancora arbitrariamente il coefficiente di amplificazione dell'amplificatore interno. Per finire è dare il colpo di grazia, per molte applicazioni pratiche sarà necessario anteporre un partitore resistivo a monte dell'amplificatore (i segnali in gioco potrebbero essere di ampiezza eccessiva per la sensibilità d'ingresso della scheda sonora).

Dalle considerazioni precedenti appare chiaro che è necessario ideare un metodo di tipo generale che consenta, nella maniera più automatica possibile, di tarare le scale in Volt e che impedisca qualsiasi tipo di ambiguità procedurale. L'idea è piuttosto semplice; applicare all'ingresso della scheda sonora una tensione di valore noto (di tipo sinusoidale); settare i parametri relativi nella sottocartella calibrate (ossia dire a Visual Analyser "ti sto applicando a tale ingresso la tale ampiezza…") e lanciare una procedura che consenta di memorizzare ed applicare la taratura. Ed alla fine, effettuare una memorizzazione dei parametri calcolati da riutilizzare in occasioni successive o per altre macchine che utilizzano la stessa scheda sonora.

Ecco generalmente come procedere in Visual Analyser:

- selezionare un ingresso (ad esempio line-in; può essere fatto direttamente dalla listbox "input source");
- settare il livello di amplificazione (per esempio il 50%, direttamente dalla trackbar presente in finestra);
- introdurre nella list box "units" le unità desiderate (per esempio Volt);
- indicare se l'ampiezza del segnale che indicheremo nella casella "level of known input level" è RMS o picco picco (per esempio RMS);
- scrivere nella casella "level of known input level" il valore della tensione nota che stiamo applicando all'ingresso;
- premere il tasto "start measure signal" ed attendere qualche secondo
- la calibrazione è finita, per "applicarla" effettivamente spuntare la checkbox "Apply calibration settings";

- se desiderato salvare la configurazione premendo il tasto “save”.

La procedura sembra macchinosa ma in realtà è molto semplice e veloce. Si noti comunque che una volta resa operativa la taratura tramite la spunta della checkbox “apply calibration settings” verrà disabilitata la possibilità di variare il volume d’ingresso tramite la replica del controllo di volume di Windows, presente sia in questa sottocartella, che nella barra dei comandi di Windows; ed in qualcuna delle finestre flottanti. Questo per evitare di perdere la calibrazione, che è stata effettuata per quel determinato livello di sensibilità di ingresso. Invero, una attenta analisi delle caratteristiche del controllo di volume di Windows potrebbero condurre all’eliminazione di questo vincolo; sarebbe sufficiente conoscere la legge di variazione del controllo. Essa sembrerebbe essere di tipo non lineare (logaritmica) ma prove effettuate in tal senso non hanno fornito risultati incoraggianti e per il momento non applicate.

Un altro problema potrebbe essere quello relativo alla disponibilità di un segnale sinusoidale (a bassa distorsione) di valore noto. Una possibile soluzione potrebbe essere quella di misurare una tensione alternata tramite un economico multimetro digitale che generalmente fornisce una lettura piuttosto precisa del valore RMS. La tensione potrebbe essere direttamente quella di rete, opportunamente ridotta tramite un trasformatore ed un generoso partitore resistivo.

6.1.4 La sottocartella In/Out device

Consente di definire la scheda audio da usare come ingresso e come uscita; alla partenza del programma la medesima routine di identificazione scheda invocabile da questa finestra è automaticamente eseguita per identificare (e scegliere) le schede audio presenti nel sistema. Qualora venga aggiunta

"dinamicamente" una nuova scheda audio (per esempio di tipo USB) è possibile forzare un nuovo riconoscimento premendo il tasto "Detect", e successivamente selezionare la nuova scheda desiderata.

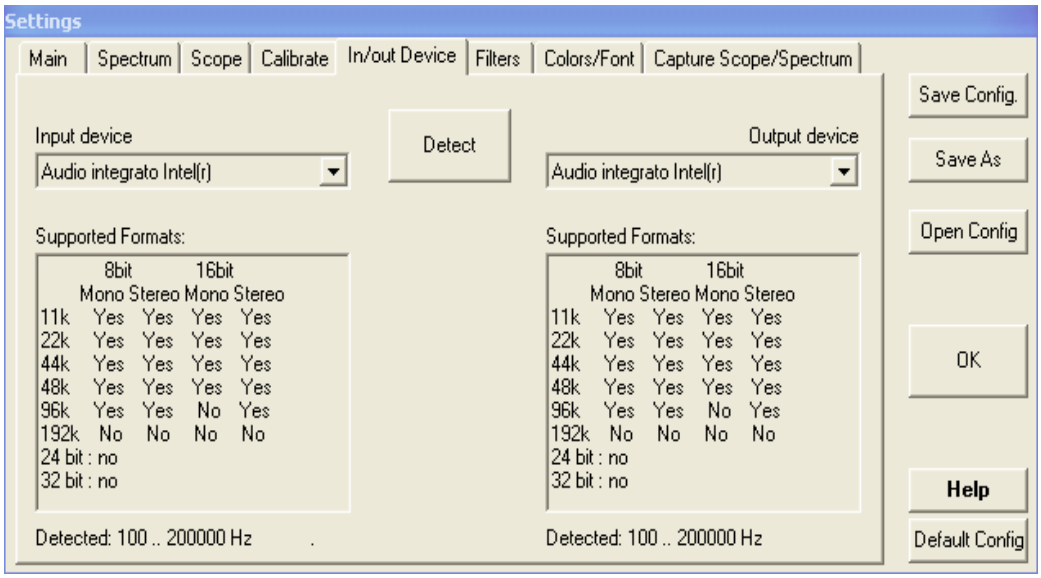


Figura 25: la sottocartella In/Out device

6.1.5 La sottocartella Filters

La sottocartella "Filters" permette di abilitare, per ogni canale, un filtro digitale in *tempo reale* con frequenza di taglio definibile. Come è possibile chiaramente osservare nella figura 26, esiste anche la possibilità ulteriore di inserire un "diodo" durante il percorso del segnale, ed un filtro per la soppressione della componente continua.

Il filtro "diodo" si limita ad applicare in maniera estremamente semplice il concetto di diodo ideale; semplicemente le componenti del segnale che hanno componente negativa sono eliminate. Il verso del diodo indica chiaramente che verranno eliminate le componenti negative. I filtri vengono applicati al segnale interno agli strumenti; in altre parole, il segnale è modificato localmente agli strumenti, e l'azione dei filtri è visibile pertanto solo sui display degli strumenti. Invero, versioni precedenti del programma consentivano anche di

inviare il segnale filtrato alla scheda sonora, e poter così "ascoltare" gli effetti dei filtri direttamente attraverso i diffusori acustici.

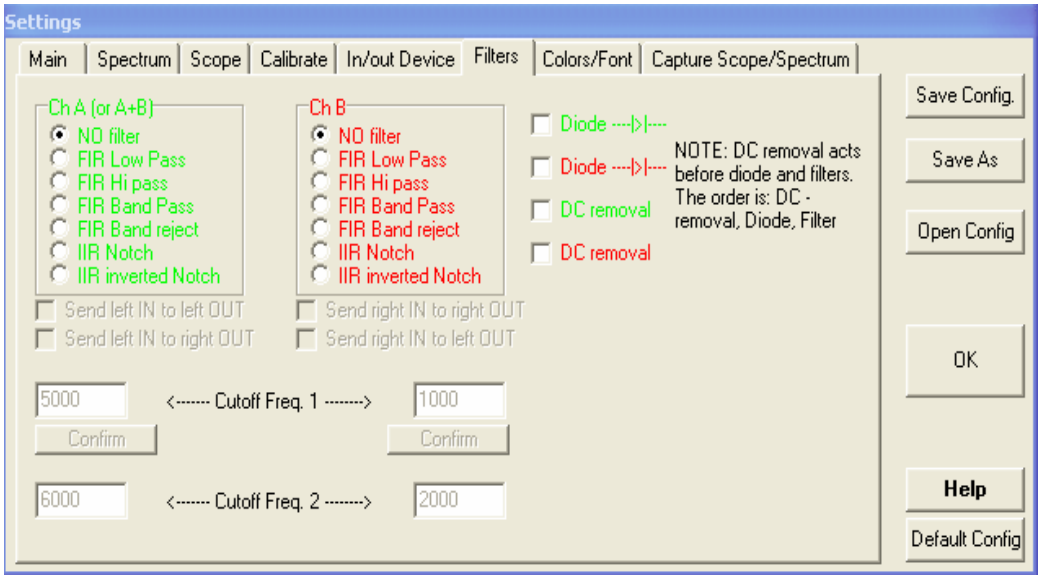


Figura 26: la sottocartella filters

Essendo l'operazione eseguita esclusivamente via software, essa era caratterizzata da una latenza di un tempo pari ad almeno un buffer. In versioni successive questa opzione è stata temporaneamente eliminata (che compare a video disabilitata) a causa di una mutata architettura del programma, e di qualche instabilità che si era rilevata con talune versioni del sistema operativo.

I filtri che sono stati implementati sono praticamente tutti quelli standard; la maggior parte sono stati realizzati come FIR (Finite Impulsive Response) e sono del seguente tipo:

- passa basso;
- passa alto;
- passa banda;
- elimina banda.

Mentre sono stati realizzati in versione IIR (Infinite Impulsive Response) i seguenti ulteriori filtri:

- notch
- inverted Notch.

Si rimanda al capitolo relativo agli algoritmi per i dettagli algoritmici e prestazionali. Si vuole porre invece l'attenzione sulle motivazioni che hanno indotto all'aggiunta di questa particolare funzionalità. Essa nasce dalla considerazione che in molti casi i segnali in analisi erano completamente "sepolti" da notevoli quantità di rumore o armoniche indesiderate. Per esempio molti segnali di ampiezza relativamente modesta erano affetti dalla presenza di un segnale sinusoidale a 50 Hz (disturbo dovuto alla rete elettrica a 220 volt) che rendevano difficile la misura; in tal caso basta inserire un filtro di "notch" con frequenza di taglio centrata sui 50 Hz. Oppure un analogo filtro passa alto se non interessano le componenti al di sotto di tale frequenza. Ancora, taluni segnali presentano una notevole componente continua, che può essere agevolmente eliminata tramite l'apposito filtro dedicato. Alcuni segnali presentavano armoniche indesiderate in determinate regioni dello spettro, ed in questo caso è stato sufficiente utilizzare un filtro elimina banda.

L'uso del "diodo" è invero nato per scopi puramente didattici, in particolare per estrarre il segnale modulante da un segnale modulato. Infatti, l'uso del filtro "diodo" ed in successione un filtro passa basso (equivalente ad un condensatore in parallelo al segnale) sono l'esatto equivalente di un circuito rivelatore.

6.1.6 La sottocartella Colors/Font

Questa sottocartella serve a personalizzare i colori assegnati ai vari grafici; inoltre c'è la possibilità di cambiare il simbolo di separazione decimale, cosa

che potrebbe tornare utile nel caso di installazioni di Windows di tipo americano (in cui il punto prende il posto della virgola e viceversa).

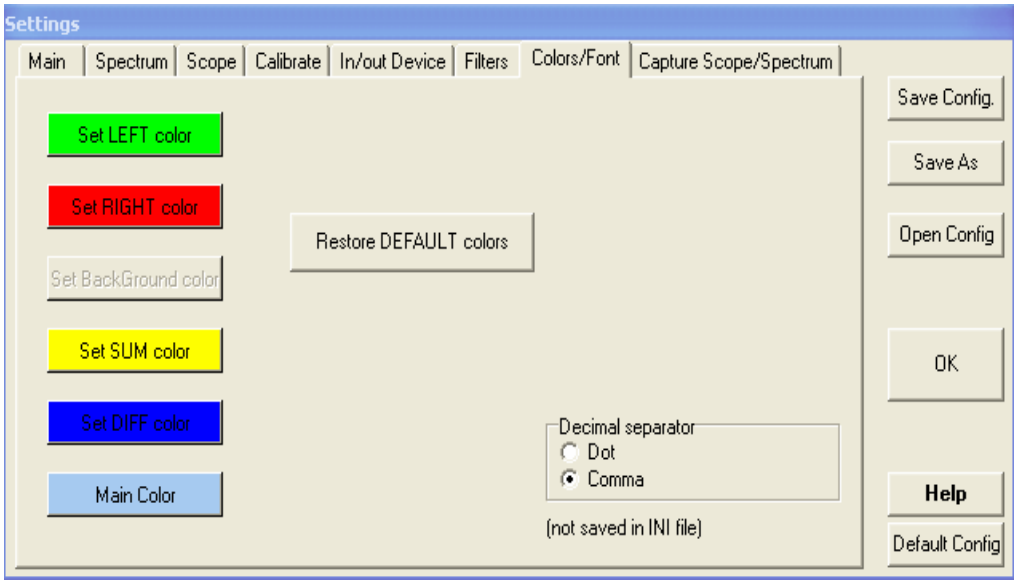


Figura 27: la sottocartella Colors/Font

6.1.7 La sottocartella Capture scope/spectrum

In questa sottocartella è possibile definire le opzioni relative ad una delle funzionalità più importanti di Visual Analyser: la cattura del segnale nel dominio del tempo e della frequenza.

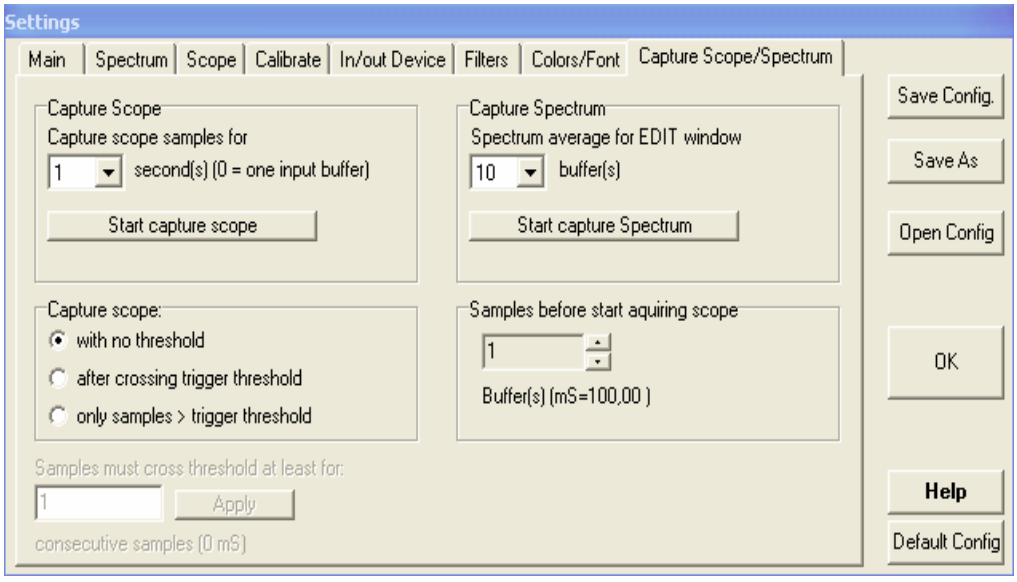


Figura 28: la sottocartella Capture Scope/Spectrum

Le opzioni sono relative dunque alle due famiglie “cattura oscilloscopio”, ossia segnali nel dominio del tempo (i campioni visualizzati dall’oscilloscopio) e “cattura spettro” ossia i segnali nel dominio della frequenza (lo spettro visualizzato nella finestra analizzatore di spettro).

Per quanto riguarda le opzioni della prima categoria, esse sono le seguenti:
Riquadro “Capture scope” (è doppio) + riquadro “samples before…”

1. [capture scope samples for…] è una listbox che serve a definire il numero di secondi di acquisizione del segnale
2. [Start capture scope] è il bottone che serve a dare l’avvio al processo di cattura dei campioni
3. [with no threshold] i campioni vengono acquisiti immediatamente
4. [after crossino trigger threshold] i campioni vengono acquisiti solamente dopo il superamento (almeno una volta) della soglia impostabile con la trackbar del trigger: in questo caso la soglia del trigger assume una doppia funzione
5. [only samples > trigger threshold] vengono aquisiti solo i campioni che superano la soglia di threshold: in pratica si ottiene un effetto simile al diodo.
6. [samples before start acquiring scope] consente di definire un buffer di pre-acquisizione. Il segnale è catturato comprendendo un certo numero di campioni prima che l’evento si sia verificato; questa opzione è particolarmente utile per non perdere parte del segnale che ha consentito l’inizio del processo di acquisizione.

La selezione delle opzioni (4) e (5) abilita una ulteriore opzione: quella di definire il numero di campioni minimo che devono superare la soglia impostata per far scattare il processo di acquisizione; essa è definibile tramite la casella di testo [samples must cross….]. In altre parole, la soglia deve essere oltrepassata almeno per un numero di campioni pari a quelli impostati. Questa opzione è

utile quando si vuole assicurare che la soglia sia effettivamente oltrepassata "stabilmente" e non a causa di un'isolato campione dovuto magari a rumore.
Per finire, le opzioni relative alla seconda categoria (frequenza) sono:

1. [Spectrum average for edit window] consente di impostare il numero di buffer su cui effettuare la media. Questa funzione è virtualmente identica a quella "real time", impostabile da finestra principale e finestra di setting.
2. [Start capture spectrum] consente di iniziare il processo di acquisizione. Non è per ora prevista nessuna soglia, sebbene è in lavorazione un sistema di soglie basato sulle ampiezze delle singole armoniche.

6.2 L'oscilloscopio

L'oscilloscopio è lo strumento più diffuso, dopo il multimetro, nei laboratori di elettronica di tutto il mondo. Esso consente di visualizzare su di un display il segnale elettrico applicato ai suoi puntali (il display è in genere un tubo a raggi catodici o un LCD a schermo piatto). In pratica si tratta, per la maggior parte delle applicazioni, di visualizzare un grafico in tempo reale rappresentato su di un sistema di assi cartesiani con il tempo sull'asse X e l'ampiezza sull'asse Y. Invero esiste un'altra modalità possibile solo negli oscilloscopi a doppia traccia, detta "XY"; il segnale di un canale "muove" le coordinate dell'asse X e l'altro quelle dell'asse Y. In tal modo si ottengono le cosiddette figure di Lissajoux, utili per effettuare misure di fase tra due segnali e la frequenza. In questa modalità il tempo non è direttamente rappresentato. Questa modalità è presente in Visual Analyser.

Il costo di un oscilloscopio è considerevole, e per taluni utilizzi è del tutto sufficiente un semplice strumento come quello proposto; questo ovviamente vale nel caso in cui si lavori essenzialmente su frequenze che non superino le

capacità di campionamento di una scheda sonora, e dunque limitatamente al campo audio o poco più.

Visual Analyser, nella modalità di esecuzione standard, simula l'oscilloscopio nella finestra superiore sinistra contenuta nella finestra principale, riportata per comodità nella figura sottostante insieme alla barra dei comandi ed alle opzioni direttamente accessibili da finestra principale e pertinenti lo stesso oscilloscopio. In questo particolare caso è stata abilitata la funzione doppia traccia, e sono stati applicati due segnali differenti ai due ingressi a scopo puramente dimostrativo.

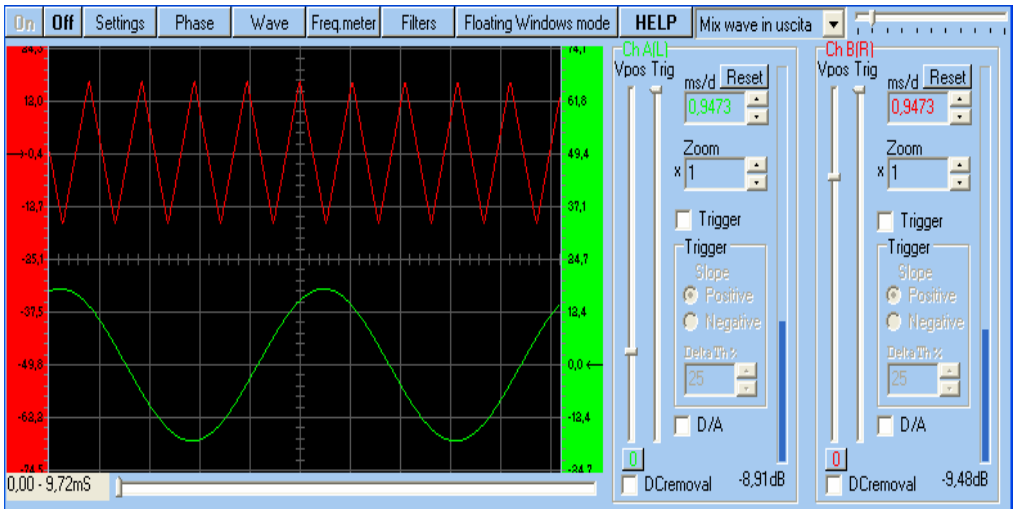


Figura 29 : la finestra oscilloscopio

Come filosofia generale Visual Analyser utilizza colori diversi per rappresentare i differenti canali, e per talune funzioni, come l'analizzatore di spettro, ulteriori colori per visualizzare segnali somma/differenza. Nel caso dell'oscilloscopio viene utilizzato solo un terzo colore per rappresentare il segnale somma dei due canali. Dall'osservazione della finestra, si possono evidenziare un certo numero di caratteristiche. Intanto che ogni canale ha una sua scala differente, i cui valori di zoom sono impostabili tramite la casella "Zoom" presente per ogni canale (esistono due tipi di scale: in percentuale di fondo scala e in volt, vedi avanti). Lo stesso dicasi per il "Time division"

(casella “ms/d”) il Trigger (e parametri correlati). Inoltre è possibile attivare un comodo filtro per l’eliminazione della componente continua e variare con continuità il livello di trigger e posizione verticale del grafico. Inoltre è presente, per ciascun canale, la possibilità di attivare o meno la conversione digitale analogica interna di Visual Analyser, con le modalità descritte nel capitolo settimo. In ultimo da notare che per ogni canale è presente un comodo indicatore di livello in dBr (dB relativi).

Per il livello in dBr del segnale, così come per le scale degli assi Y, le cose sono state definite tenendo presente la profondità di bit usati per il convertitore analogico digitale della scheda sonora (NOTA: è cosa diversa dalla conversione Digitale Analogica “algoritmica” implementata da VA). Infatti, se per esempio sono stati utilizzati 16 bit essi consentono la rappresentazione di 65536 differenti valori, che in termini di numeri relativi (con segno) si riduce ad un range che varia tra il valore -32768 ed il valore +32767. Si è stabilito di fissare come livello di zero dB il valore massimo (65536) e nella modalità “percentuale fondo scala” degli assy Y fissare come 100% il medesimo valore. Ossia, per gli assi Y dell’oscilloscopio la scala varia tra -50% = -32768 e 50% = 32767. Gli stessi ragionamenti possono applicarsi per 8, 24 e 32 bit. Per l’indicatore di livello la scala varia tra zero dB (massimo valore) e -999 (simblo usato per indicare -infinito). Si osservi comunque che per l’oscilloscopio la misura in dB viene applicata solo per le barre indicatrici di livello, mentre per gli assi Y può usarsi la “percentuale fondo scala” e la scala in Volt. Per l’analizzatore di spettro invece verrà utilizzata la scala direttamente tarata in dBr (picco-picco) oppure in Volt.
Facendo riferimento a quanto descritto nel paragrafo precedente, si osservi che l’oscilloscopio può visualizzare i due canali secondo le seguenti quattro modalità (molte di più invece per l’analizzatore di spettro):

- solo canale sinistro;
- solo canale destro;

- sinistro e destro;
- sinistro + destro.

Invero, selezionando la modalità “funzione di trasferimento” ossia la sinistro – destro o viceversa l’oscilloscopio continuerà ancora a visualizzare contemporaneamente il canale sinistro e destro, per esigenze che appariranno chiare nel capitolo dedicato alle applicazioni pratiche.

Per selezionare queste quattro modalità si può usare la finestra di settings oppure più comodamente i comandi riportati alla destra dell’analizzatore di spettro in finestra principale, come da figura 30.

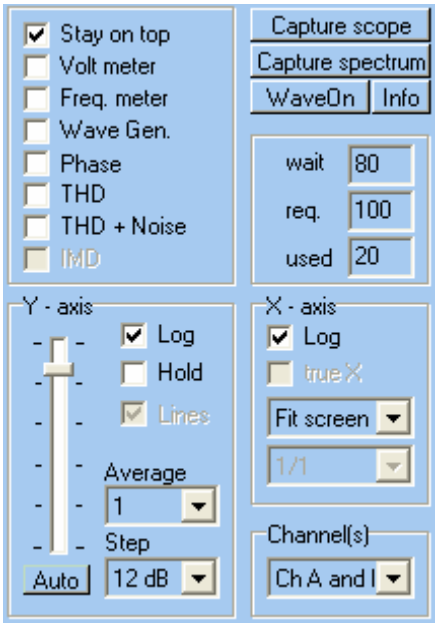


Figura 30: i comandi principali

Essi sono relativi per lo più all’analizzatore di spettro o fanno riferimento a caratteristiche generali; La listbox “Channels” fornisce la possibilità di cambiare i canali d’ingresso per entrambi gli strumenti. Si osservi come anche in questa sezione sono presenti le checkbox che permettono la selezione rapida delle varie funzioni (Frequenzimetro, Fase, THD eccetera). Le sezioni relative agli assi X e Y sono dedicate esclusivamente all’analizzatore di spettro; per l’oscilloscopio non è prevista nessuna visualizzazione diversa da quella lineare sia in asse X che Y. Si osservi alla base della finestra oscilloscopio la presenza di una trackbar affiancata ad una scritta che riporta un range in millisecondi. Infatti, il buffer di punti normalmente eccede la dimensione orizzontale in pixel della stessa finestra oscilloscopio. Allora, per evitare di perdere informazioni tagliando il numero di punti o dover raggruppare tutti i punti su di un numero ristretto di pixel (e

quindi in definitiva perdere comunque informazioni) si permette, tramite la trackbar, lo scorrimento della scala dei tempi su tutto il buffer di punti disponibili. I tempi in millisecondi indicano il range dei tempi relativamente al segnale effettivamente visualizzato.

Menzione a parte va fatta per il bottone "Reset" presente nei comandi dedicati all'oscilloscopio sopra la casella relativa al time-division. Attivando la modalità conversione Digitale/Analogica si è rilevato che sino a determinati valori di time-division si aveva comunque una corrispondenza di almeno un punto per ogni pixel, dunque l'attivazione della routine di conversione analogico digitale aveva come unico effetto quella di consumare risorse di elaborazione senza apportare nessun miglioramento alla qualità del segnale visualizzato. Allora, si è attivato un meccanismo automatico che disinserisce automaticamente la conversione D/A quando non necessaria, e la attiva per quei valori di time-division che renderebbero la forma d'onda povera di punti ossia simile ad una spezzata. Il tasto reset porta il valore di time-division a quel valore a confine tra le due modalità, che normalmente consiste nel valore a corrispondenza diretta 1 punto = 1 pixel. Esso ha senso anche se la modalità D/A non è stata attivata: in tal caso segna il confine tra una corrispondenza punto-pixel di almeno uno a uno a uno a molti, ossia da forma d'onda continua a spezzata (talvolta una spezzata è sufficiente per determinate applicazioni). E' interessante notare che visualizzando una forma d'onda con un time-division tale da renderla una spezzata e successivamente attivando la conversione D/A si ottiene immediatamente la scomparsa dell'effetto spezzata a parità di time-division (cfr fig. XX in sezione introduzione del capitolo primo).

Infine, la barra dei comandi contiene i bottoni per la selezione rapida delle varie funzioni, ed anche una caratteristica comune a tutte le funzioni di Visual Analyser che è la selezione dell'ingresso e il suo controllo di volume. Questa "utilità" replica parte dei comandi già normalmente presenti nel "controllo di volume di windows" di cui discuteremo ampiamente. Si osservi che il codice relativo per la sua corretta gestione è quanto di più ostico e complesso (e mal documentato) che si può trovare nel mondo Windows.

6.2.1 Funzioni aggiuntive dell'oscilloscopio (frequenzimetro)

Una funzione che presenta la notevole caratteristica di far risparmiare tempo prezioso durante l'uso pratico, è quella relativa alla determinazione dell'ampiezza e della frequenza tramite mouse. Nella finestra oscilloscopio è possibile (cliccando con il tasto sinistro e mantenendo la pressione mentre si muove il mouse) selezionare una regione rettangolare dello schermo che consente il calcolo della frequenze e dell'ampiezza picco-picco. La frequenza è calcolata tenendo conto della time division effettiva, per cui il valore calcolato è quello che avrebbe una forma d'onda se un periodo entrasse esattamente nella regione selezionata. In termini pratici, individuata una forma d'onda periodica di cui interessa conoscere (per esempio) la frequenza, si tracci un rettangolo che comprenda (esclusivamente lato asse X) un periodo della stessa. La frequenza riportata in prossimità del cursore è quella desiderata. Questo è un metodo alternativo e più veloce all'uso del frequenzimetro inteso come finestra a se stante, sebbene di minore precisione e con l'inconveniente di richiedere l'uso della mano e una forma d'onda "ferma" sullo schermo (che tuttavia si può rendere comunque tale grazie al trigger).

6.3 L'analizzatore di spettro

L'analizzatore di spettro è uno degli strumenti normalmente più costosi, la cui implementazione simulata è probabilmente la più richiesta dai cultori dell'audio e della musica elettronica, sia a livello amatoriale che professionale. L'elevata qualità delle schede sonore, normalmente già presenti in questi laboratori, rende un analizzatore di spettro simulato su PC una scelta estremamente conveniente e del tutto sufficiente a coprire ogni esigenza

L'analizzatore di spettro, tramite un algoritmo noto come trasformata di Fourier, fornisce una rappresentazione "nel dominio della frequenza" di un segnale elettrico (o conseguentemente di un qualsiasi segnale "mediato" da un segnale elettrico). Lo spettro di un segnale è costituito generalmente da due grafici, uno relativo alla decomposizione armonica del segnale e l'altro alla fase. In entrambi i grafici la variabile indipendente (asse delle ascisse) è la frequenza, mentre in ordinata abbiamo in uno l'ampiezza in dB delle armoniche (diagramma delle ampiezze) e nell'altro il valore della fase (in gradi o radianti, ed è il diagramma delle fasi). In genere il diagramma delle ampiezze è quello di maggiore utilità e per questo motivo Visual Analyser fornisce la possibilità di visualizzare il diagramma delle fasi solo in una finestra separata, e non direttamente nella finestra principale.

Si osservi come gli strumenti analogici di primissima generazione, effettuavano l'analisi armonica del segnale tramite dei filtri (analogici) che isolavano le varie componenti armoniche del segnale. I moderni strumenti digitali utilizzano invece, oltre alla necessaria circuteria analogica ad elevata banda passante (per l'amplificazione del segnale), un microprocessore (spesso di tipo DSP = Digital Signal Processor) operando sostanzialmente come Visual Analyser.

Come discusso nel capitolo quarto l'algoritmo scelto in Visual Analyser è la versione "veloce" della trasformata di Fourier, chiamata FFT ossia "Fast Fourier Transform" (trasformata di Fourier veloce). Rispetto all'algoritmo standard presenta una velocità di esecuzione proporzionale a N*logN dove N il numero di punti campione. L'algoritmo standard prevede tempi d'esecuzione proporzionali a N*N quindi nettamente superiori (in pratica è inutilizzabile per applicazioni in tempo reale con le macchine attuali). La limitazione insita nella FFT è che il numero di punti deve essere necessariamente una potenza di due. Questo è il motivo per cui nella finestra di Settings non è possibile definire un buffer arbitrario ma è necessario utilizzare dei valori predefiniti e pari appunto a potenze di due. Peraltro sarebbe comunque possibile ovviare all'inconveniente effettuando una operazione di "zero padding", cosa che in

effetti è stata utilizzata per il frequenzimetro, ma nel caso dell'analizzatore di spettro non si è ravvisata la necessità di un simile artificio.

6.3.1 Il diagramma delle ampiezze

Il diagramma delle ampiezze è di fatto la caratteristica più utile dello spettro di un segnale. Per tale motivo il diagramma delle ampiezze è presente per default nella finestra principale di Visual Analyser, ed è dotato di numerose opzioni.

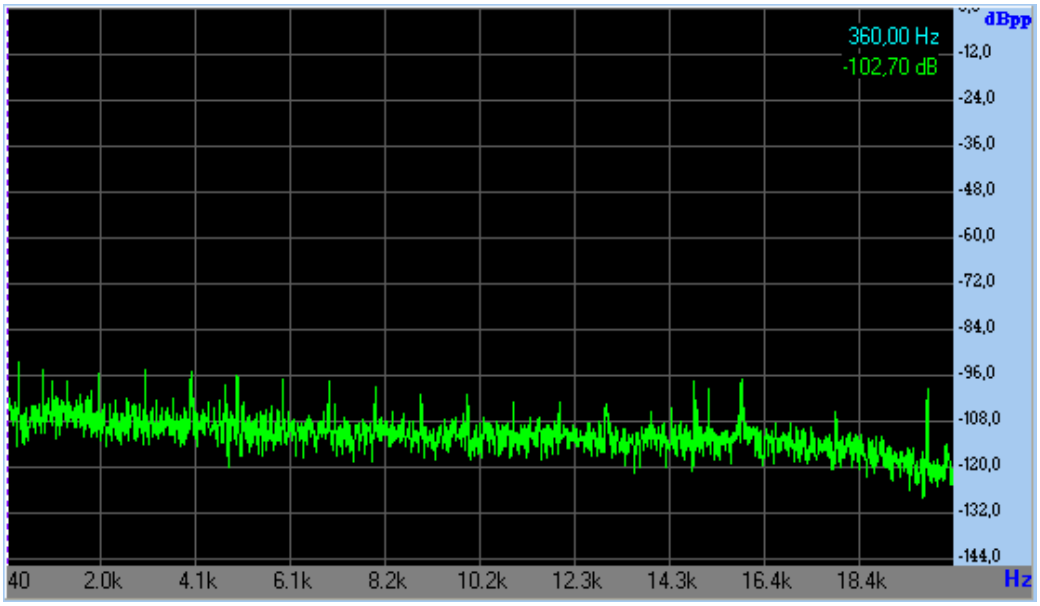


Figura 31 : digramma delle ampiezze

Facendo per ora riferimento alla figura 30 (pag151) queste sono le opzioni disponibili:

1. [Y-axis] in questo gruppo di opzioni abbiamo la possibilità di effettuare uno zoom manuale od uno zoom automatico (bottone "auto"); definire un livello di average, ossia stabilire il numero di buffer su cui effettuare la media aritmetica (cfr. paragrafo 6.3); settare

la modalità di visualizzazione dell'asse Y (logaritmica o lineare); infine, il "passo" del reticolo.

2. [X-axis] sono le opzioni relative all'asse X; anche in questo caso si può scegliere tra rappresentazione lineare o logaritmica; si può definire se visualizzare un valore arrotondato o reale (true X value); infine definire la modalità di rappresentazione dell'asse X. Per questa importantissima opzione sono previste le seguenti scelte:

    - Fit screen: significa che i valori dell'asse X sono adattati alla larghezza della finestra; è comunque mantenuta la corrispondenza pixel-armonica (v sezione sotto)
    - x1,x2,x4,x8,x16 (le armoniche sono disegnate ogni "n" pixel, serve a zoomare, ed è possibile scorrere l'asse X tramite una trackbar);
    - Octaves: rappresentazione a ottave, terzi d'ottava e sesti d'ottava, selezionabile tramite la listbox al di sotto della listbox per queste selezioni

12. [Channels] è la stessa di quella descritta per l'oscilloscopio. In questo caso le opzioni relative alla differenza dei canali implicano il disegno della effettiva differenza, ma in dB; inoltre il valore di zero dB è ora differente da quello delle altre opzioni (e comunque impostabile da settaggi) ed è inteso come segnale differenza nullo. Un valore positivo o negativo in dB indica che un segnale è inferiore o superiore all'altro di un "tot" di volte.

Per quanto riguarda l'opzione "Fit Screen" e tutte le successive si deve fare la seguente (importantissima) considerazione. Si è volutamente mantenuta una strettissima corrispondenza pixel-armonica, quale che sia il la modalità di

rappresentazione scelta per l'asse X. Infatti, il numero di armoniche calcolato dall'algoritmo è in numero finito e pari alla metà dei punti del buffer (più la componente continua). Per avere sempre disponibile tutti i dati calcolati, pur avendo generalmente un numero di pixel nettamente inferiore si è operato nella seguente maniera. Nella modalità fit screen è ovvio che un singolo pixel è rappresentativo di più armoniche; allora, per consentire di sapere esattamente quali armoniche sono rappresentate da quel singolo elemento del disegno è sufficiente cliccare sullo schermo (tenendo premuto) con il tasto sinistro del mouse; una finestra affiancata al cursore indicherà frequenza ed ampiezza dell'armonica (o delle armoniche) rappresentate da quel pixel. Questa caratteristica si mantiene per tutte le possibili rappresentazioni dell'asse X. NOTA: nella stragrande maggioranza dei programmi di analisi spettrale il valore visualizzato è uno solo, spesso il valore medio di tutte le armoniche "concentrate" nel punto o peggio, uno solo.

La scelta dell'opzione A-B o B-A relativamente al punto (3) significa che il grafico risultante è la differenza in dB dei due canali, nell'ordine scelto. Per esempio, scegliendo l'opzione A – B ossia canale destro meno sinistro, significa che:

(dB)destro – (dB)sinistro = 20*log(D/zero) – 20*log(S/zero) = A – B

Dove :

D = ampiezza canale destro
S = ampiezza canale sinistro
Zero = valore zero dB

Ma si può scrivere anche come (applicando le proprietà dei logaritmi):

A – B = 20*log (S/D)

Se immaginiamo S essere il segnale d'uscita di un amplificatore e D quello in ingresso allora si ha che il segnale (in dB) ottenuto dalla differenza A – B è esattamente la *funzione di trasferimento* dell'amplificatore. Esso è inoltre indipendente dalle caratteristiche intrinseche dell'amplificatore proprio della scheda sonora, perché esse si elidono con il rapporto (nei limiti delle approssimazioni di calcolo).

Se Visual Analyser gira nella modalità a finestre flottanti, il diagramma delle ampiezze è rappresentato in una finestra separata e liberamente dimensionabile. Essa presenta alcune opzioni di uso più comune direttamente accessibili da una sorta di "barra comandi" locale, più un menù a tendina richiamabile tramite mouse invocato "cliccando" sull'area di disegno con il tasto destro. Le opzioni presenti nella barra dei comandi locali comprendono:

- Tasto "S" per effettuare l'autoscale
- Tasto "U" per ritornare al valore precedente della scala
- Listbox "Average" per settare la media in tempo reale
- Checkbox "logy" per settare la scala dell'asse Y logaritmica/lineare
- Check box "logx" come sopra ma per l'asse X
- Checkbox "lines" per settare la rappresentazione delle ampiezze a barre o linee (quando consentito)
- Bottone "Settings" per invocare la finestra di settings
- Spin button "Zoom" per effettuare lo zoom dell'asse Y
- Spin button "start" per settare la posizione del reticolo

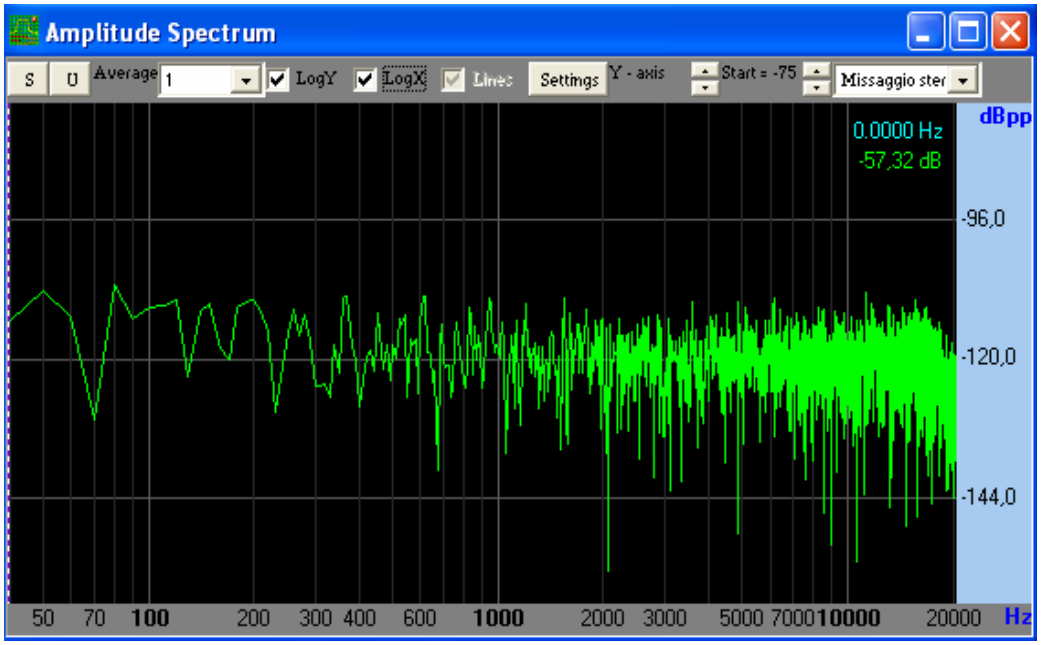


Figura 32 : il diagramma delle ampiezze a finestre flottanti

6.3.2 Il diagramma delle fasi

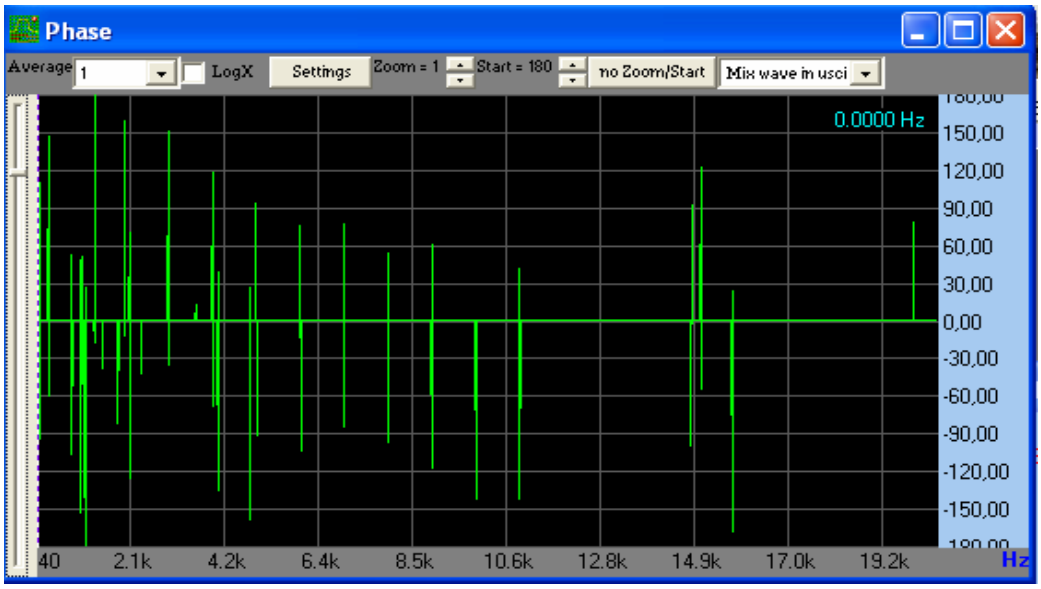


Figura 33: diagramma delle fasi

Per accedere a questa seconda parte, meno significativa, dello spettro del segnale, sarà sufficiente cliccare sull'opzione "phase" nella finestra principale

o, equivalentemente, sul bottone “phase” della barra dei comandi. La finestra del diagramma delle fasi presenta le stesse caratteristiche ed opzioni di quella dello spettro, eccetto il fatto che essa esiste solo nella modalità a finestre flottanti. Come tale è dotata di una barra dei comandi locale che consente di accedere rapidamente alle opzioni più comuni. Data l’estrema variabilità della fase di ogni singola componente armonica, è stata introdotta una trackbar capace di impostare con continuità una “soglia di intervento” basandosi sul diagramma delle ampiezze. In altre parole, è possibile escludere dalla visualizzazione della fase tutte quelle armoniche che hanno un’ampiezza in dB al di sotto di un certo valore. Questa opzione si rivela particolarmente utile perché il contributo in fase di armoniche di ridottissima ampiezza (per esempio del rumore) non consentirebbe una agevole valutazione di quelle ad ampiezza più significativa.

6.4 Il generatore di funzioni

Per lanciare il generatore di funzioni si può, al solito, spuntare la checkbox sulla finestra principale, od equivalentemente cliccare sul corrispondente bottone contenuto nella barra dei comandi (bottone “Wave”). Come si può notare da una prima analisi della finestra, essa è dotata di numerose opzioni; inoltre, è una finestra organizzata in sottocartelle (anch’esse numerose) e dotate di molte caratteristiche che val la pena di descrivere. Dedicheremo un paragrafo per ogni sottocartella, intanto definiamo le caratteristiche generali. Il generatore di funzioni, alla versione 8.10, è in grado di generare:

- Onde sinusoidali
- Onde quadre
- Sweep di sinusoide (settaggi in sottocartella “set sweep” paragrafo 6.4.4)

- Rumore bianco
- Rumore rosa
- Forme d'onda completamente definibili dall'utente (Custom) tramite la sottocartella "Custom function" (v. paragrafo Custom Function)
- Impulsi (settaggi in cartella Set Pulse, v. paragrafo 6.4.3)
- Onde Triangolari (settaggi in sottocartella Triangle/Sawtooth, v paragrafo 6.4.5)
- Onde a dente di sega (con pendenza completamente definibile, settaggi in sottocartella Triangle/Sawtooth, v paragrafo 6.4.5)

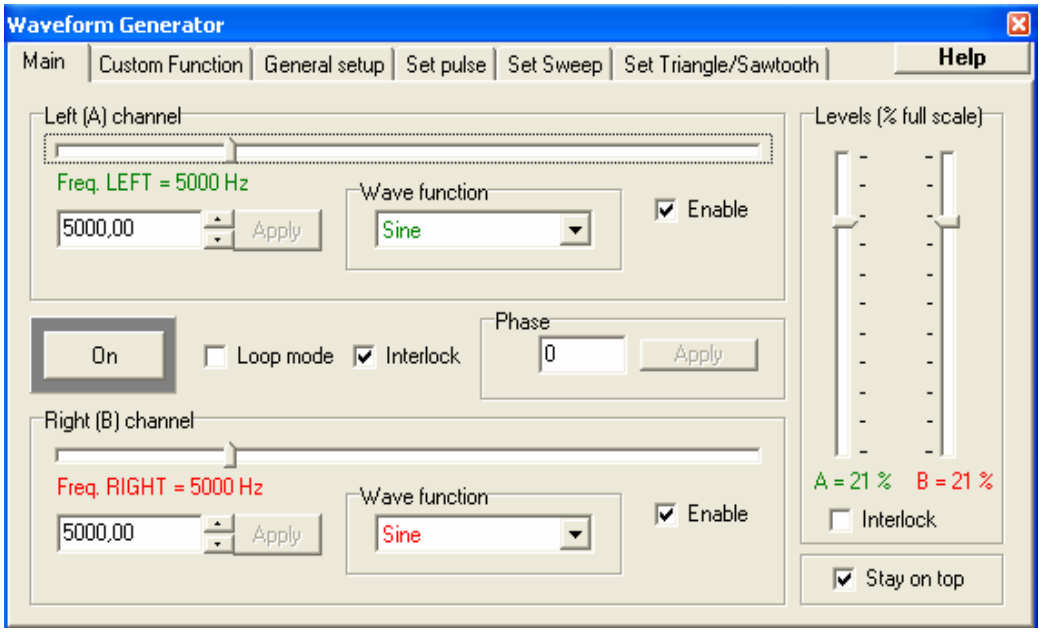


Figura 34: il generatore di funzioni

Le forme d'onda possono essere generate in due modalità, ossia nella modalità "Loop" od in "tempo reale" e per far questo è sufficiente spuntare la checkbox "Loop mode". Giova ricordare che nella modalità in "loop" il generatore di funzioni calcola la forma d'onda e la invia alla scheda sonora, la quale la esegue in un loop infinito; nel caso "tempo reale" i campioni vengono forniti di volta in volta alla scheda sonora, a blocchi di un buffer alla volta. La differenza sostanziale tra queste due modalità consiste nella possibilità, per la seconda, di

variare i parametri in tempo reale ed avere la possibilità di generare forme d'onda differenti per ciascun canale. Nel caso "loop" i parametri saranno modificabili solo dopo aver momentaneamente "spento" il generatore e successiva "riaccensione". Si rimanda per i dettagli implementativi ed architetturali al capitolo II. Indipendentemente dalla modalità prescelta, i comandi relativi a questa prima finestra sono i medesimi. In particolare la selezione della forma d'onda avviene tramite la listbox "WaveFunction"; la frequenza si imposta con una trackbar, e per ottenere una maggiore precisione si può usare la casella di testo situata subito al di sotto delle stessa trackbar per input "manuale" con numeri decimali. La casella "Phase" è relativa alla fase del secondo canale rispetto al primo, ed è espressa in gradi; infine, per iniziare la generazione della forma d'onda è sufficiente premere il tasto "ON". Si noti che se selezionata la modalità "tempo reale", ossia la casella "loop" è deselezionata, durante la generazione delle forme d'onda la cornice verde attorno al bottone di "on" inizierà a lampeggiare; essa è direttamente pilotata dal task che genera i campioni, ed indica che l'algoritmo sta effettivamente girando (ossia, ogni commutazione da verde a grigio e viceversa significa che un buffer è stato prodotto ed inviato alla coda). Viceversa, nella modalità loop l'indicatore resta costantemente verde (ad indicare che nessun task sta effettivamente girando, e tutto è stato semplicemente demandato alla scheda sonora).

Menzione a parte va fatta per i controlli di volume, individuati dal riquadro "Levels". Essi NON pilotano il mixer (in riproduzione) di Windows, ma indicano l'ampiezza relativa del segnale generato. In altre parole, determinano l'ampiezza locale (di picco) del segnale generato, in termini di percentuale fondo scala. Per esempio, se si usano 16 bit, e si seleziona il 100% di volume, verrà generata una forma d'onda con un ampiezza picco-picco di 65536 unità.

6.4.1 Sottocartella "Custom Function"

La sottocartella “Custom Function” permette di configurare le opzioni per la definizione della forma d’onda “arbitraria”. Infatti, sebbene per la maggior parte degli usi le forme d’onda predefinite siano del tutto sufficienti, si è comunque preferito implementare un meccanismo che consenta di definire una forma d’onda potenzialmente arbitraria. Essa viene costruita sintetizzandola come somma di segnali elementari ricavati tramite lo sviluppo in serie di Fourier (v. paragrafo 4.5). Qualora la difficoltà del calcolo dei coefficienti sia elevata (esistono invero in rete un notevole numero di programmi adatti allo scopo), è possibile utilizzare un tool grafico compreso in Visual Analyser (bottone Visual Tool) che consente di visualizzare il risultato ottenuto in tempo reale (v. paragrafo 6.4.1.1).

La generazione di una forma d’onda arbitraria potrebbe sembrare un’operazione relativamente semplice, dovendo in fondo definire un certo numero di punti che idealmente “campionano” la forma d’onda desiderata; come operazione successiva sarebbe poi sufficiente trasmettere i campioni ottenuti alla scheda sonora. Molti software esistenti fanno in effetti proprio questo: tramite una semplice routine grafica si disegna la forma d’onda ed in maniera automatica se ne ricavano i campioni. Il problema che affligge questo tipo di approccio è la probabile presenza di aliasing; ossia i punti “numerici” ottenuti non appartengono assolutamente ad una forma d’onda a banda limitata (con estremo superiore pari alla frequenza di Nyquist), come rigorosamente richiesto dal teorema del campionamento. Il problema è risolvibile solo generando il segnale voluto come somma delle sue componenti armoniche (ossia sinusoidali pure), che è poi la strategia adottata da Visual Analyser.

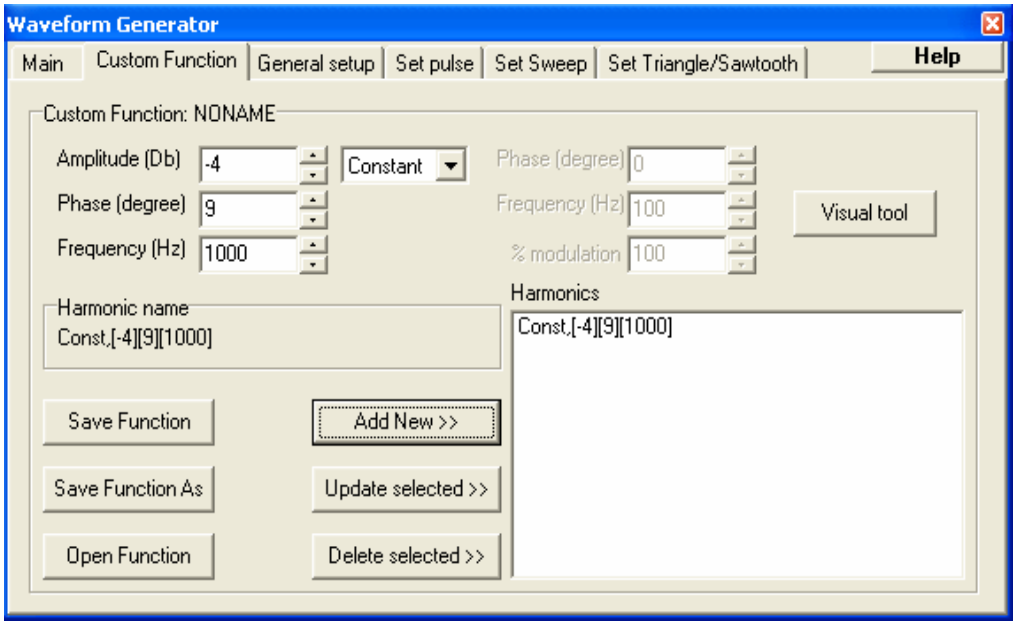


Figura 35: la cartella Custom Function

Per ogni armonica è pertanto possibile definire:

1. Ampiezza in dB (Amplitude)
2. Fase (Phase) in gradi
3. Frequenza (Frequency) in Hz

L'ampiezza presenta una ulteriore opzione (invero "ridondante"); ossia la possibilità di "modulare" l'ampiezza secondo una delle forme d'onda predefinite (quadra, triangolare, sinusoidale). Ridondante perché sarebbe cosa già fattibile semplicemente con le opzioni (1) (2) e (3), giacchè una forma d'onda modulata in ampiezza altro non è che un semplice segnale, calcolabile con lo sviluppo in serie di Fourier. Peraltro tramite questa opzione si semplifica di molto il calcolo dello stesso; è sufficiente calcolare lo sviluppo in serie della forma d'onda non modulata e poi applicare la modulazione. Da notare che il default è una modulazione di tipo "Constant" ossia assenza di modulazione. Qualora si opti per l'aggiunta di una modulazione diversa da "constant" essa permette di definire per la forma d'onda modulante i seguenti parametri:

1. Fase (Phase) in gradi
2. Ampiezza (Amplitude) in dB
3. Percentuale di modulazione in %

Una volta definita l'armonica, essa potrà essere aggiunta alla lista (riquadro "Harmonics") tramite il pulsante "Add new", laddove il pulsante "Update selected" consente la modifica di una già esistente e "Delete selected" la cancellazione. Il riquadro "Harmonic name" costruisce una stringa che servirà da identificatore univoco dell'armonica definita, secondo le seguenti convenzioni:

[tipo di modulazione],[ampiezza][fase][frequenza]

Se presente la modulazione diversa da "constant" si aggiungono le tre sequenze:

[fase modulazione][Frequenza modulazione][percentuale modulazione]

La funzione definita è memorizzabile su file con estensione .fun, tramite i bottoni "Save Function" e "Save Function as", ed eventualmente richiamabile tramite il bottone "Open Function" od automaticamente alla prima esecuzione di Visual Analyser

6.4.1.1 Il Tool Visuale ("Visual tool")

Alla versione 8.10 questo tool è invero ancora in via di sviluppo, sebbene perfettamente funzionante (con qualche limitazione). Esso consente di abilitare tramite checkbox l'immissione di armoniche tra massimo una ventina già predisposte in appositi riquadri, la cui ampiezza, fase e frequenza possono

rapidamente essere inserite tramite trackbar (e comunque “rifinite” tramite caselle di testo decimali), e contemporaneamente visualizzate su un piccolo display locale. Finita la costruzione della forma d’onda, essa può essere “accettata” tramite il tasto OK o Apply. Cliccando OK la forma d’onda viene “accettata” e la finestra chiusa. Tramite “Apply” semplicemente “validata” rimanendo ancora nella finestra. In entrambi i casi i parametri vengono trasmessi alla sottocartella “custom function” e codificati secondo lo schema del paragrafo precedente. Si noti che la finestra è di tipo *modale* ossia non è possibile uscire da essa sino alla chiusura (contrariamente alla totalità delle rimanenti finestre di Visual Analyser).

6.4.2 Sottocartella “General Setup”

In questa finestra si possono settare i parametri fondamentali del generatore di funzioni; essi sono relativi all’architettura utilizzata per la modalità “tempo reale”. Per i dettagli sull’architettura si veda il capitolo due. I parametri che è possibile settare sono:

1. [Buffer (samples)] esso rappresenta la dimensione in numero di campioni del pool di buffer con cui la scheda sonora viene “alimentata”;
2. [Number of Buffer] rappresenta il numero di buffer mantenuti nella coda che alimenta la scheda sonora; ovviamente il minimo è uno;
3. [Frequency sampling] è la frequenza di campionamento con la quale vogliamo che siano prodotti i campioni. Si osservi che essa è svincolata dalla frequenza di campionamento con la quale leggiamo i campioni dagli altri strumenti; a dimostrazione ulteriore di come la funzione generatore di funzioni sia in effetti svincolata completamente dalle altre;

4. [Bit per Sample] è la profondità di bit con cui vengono rappresentati i campioni; anche questo parametro è svincolato da quello degli altri strumenti

6.4.3 Sottocartella “Set Pulse”

La sottocartella Set Pulse consente di definire i parametri fondamentali per la funzione “impulso”; essa è piuttosto semplice ed intuitiva, potendosi infatti rapidamente desumere dal disegno ivi riportato le funzioni che i vari parametri sono portati a rappresentare. In particolare “pulse width” e “pulse repetition” non necessitano di ulteriori spiegazioni; l’opzione “pulses” consta di tre scelte mutuamente esclusive:

- [Single shot Pulses] ossia l’impulso verrà ripetuto solo una volta dopo la messa in “on” del generatore di funzione;
- [Infinite] l’impulso definito verrà ripetuto all’infinito, ossia sino allo spegnimento del generatore di funzioni; si noti come in questo caso definendo un impulso “simmetrico” si ricade nel caso “onda quadra”;
- [User defined] consente di ripetere la struttura impostata un numero finito di volte che si può definire nella casella “Number of Pulses” abilitata solo nel caso di uso di questa opzione.

6.4.4 Sottocartella “Set Sweep”

La sottocartella “setsweep” consente (per ora) di definire solo tre parametri fondamentali; la frequenza di partenza, di arrivo ed il tempo necessario a compiere la “spazzolata”.
Si osservi che la funzione sweep sarebbe facilmente estendibile a tutte le forme d’onda predefinite di Visual Analyser; altre caratteristiche semplici da aggiungere sono la possibilità di effettuare uno spazzolamento bidirezionale, ed estendere anche a forme d’onda arbitrarie. Attualmente la “spazzolata” in frequenza è effettuata sono in una direzione (dalla più bassa alla più alta) .

6.4.5 Sottocartella “Set triangle/sawtooth”

La sottocartella “Set Triangle …” consente di definire visivamente i parametri dell’onda triangolare, che può così trasformarsi in un’onda a dente di sega con parametri variabili.

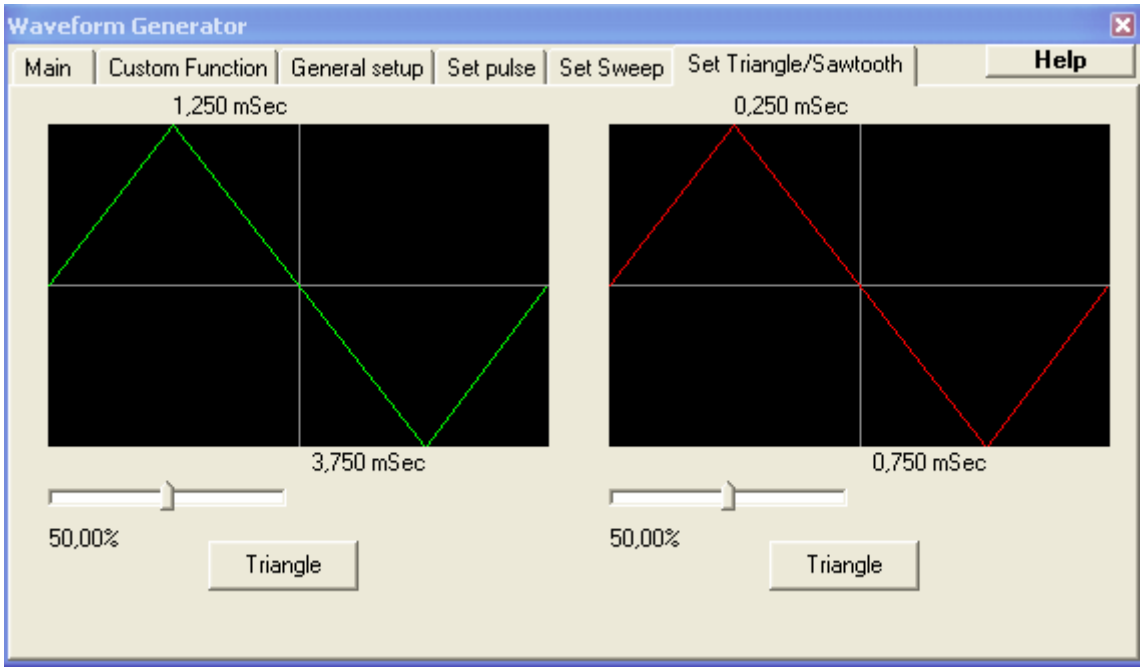


Figura 36 : la sottocartella set Triangle/sawtooth

Si osservi che anche per questa funzione è stato previsto il calcolo esplicito dello sviluppo in serie di Fourier, che viene modificato in tempo reale mentre la

forma d'onda viene modificata con le trackbar. Il bottone "Triangle" consente di effettuare una sorta di "reset" rapido per consentire di partire dalla funzione di default triangolare.

6.5 Il frequenzimetro

La funzione frequenzimetro da origine ad una finestra relativamente semplice, ma in effetti essa è alimentata da un thread dedicato che opera in maniera relativamente complessa, come descritto ampiamente nel capitolo secondo. Limitiamoci a fare qualche eventuale richiamo e dare qualche dettaglio operativo.

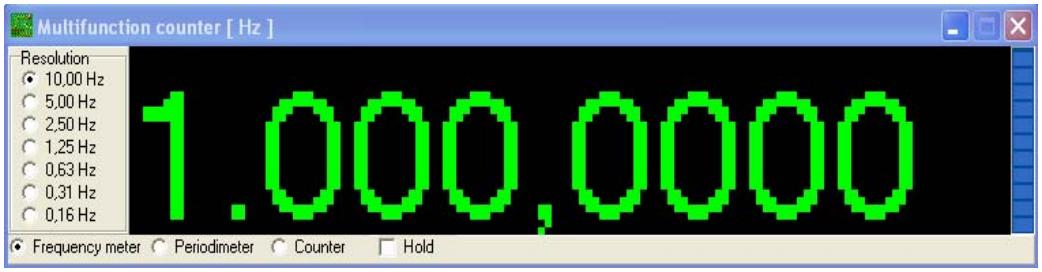


Figura 37 : il frequenzimetro

Questa finestra si può richiamare, come di consueto, dalla barra dei comandi cliccando sul bottone "Freq.Meter" oppure in maniera del tutto equivalente tramite la checkbox "freq. Meter" della finestra principale.

Il frequenzimetro presenta una scala di "risoluzioni" che potrebbero apparire incomprensibili; la prima a cominciare dall'alto (10Hz nella figura 37) è quella "naturale" definita nei settaggi generali di Visual Analyser. Usando questa prima opzione non si attiva nessun thread aggiuntivo, e si sfrutta la FFT calcolata per la funzione "Analizzatore di spettro". Questa però può essere insufficiente (per esempio dieci Hertz di risoluzione si ottengono con 40960 Hz di frequenza di campionamento e 4096 punti di buffer = 40960/4096). Per alzare la risoluzione si potrebbe allora optare per un aumento delle dimensioni

del buffer od una diminuzione della frequenza di campionamento. Ma in entrambi i modi si andrebbe ad impattare pesantemente sul carico computazionale (aumento delle dimensioni del buffer) o sulla banda passante generale (diminuzione frequenza di campionamento). Per ottenere accettabili prestazioni e non interferire con i settaggi propri dell'analizzatore di spettro e dell'oscilloscopio, il thread interno opera su un buffer di dimensioni proporzionalmente maggiori, ma usando una priorità più bassa rispetto al thread che incorpora il calcolo della FFT dell'analizzatore di spettro. E che quindi viene eseguito "quando c'è tempo", spuntando tempi di esecuzione generalmente comparabili a quelli di un frequenzimetro reale (si può arrivare a tempi dell'ordine del secondo tra due letture successive). Infatti, un "led" lampeggiante (non visibile in figura 37) indica quando il thread ha finito i suoi calcoli e presenta la nuova lettura a video (esattamente come succede per un frequenzimetro reale). Si osservi la figura 37; la prima risoluzione è (come detto) quella relativa all'analizzatore di spettro, quindi 40960/4096=10 Hz, mentre le altre si ottengono:

- 5 Hz = 40960/(4096*2)
- 2.5 Hz = 40960/(4096*4)
- 1.25 Hz = 40960/(4096*8)
- 0.63 Hz = 40960/(4096*16)
- 0.31 Hz = 40960/(4096*32)
- 0.16 Hz = 40960/(4096*64)

Le altre funzioni fornite dalla finestra frequenzimetro sono selezionabili nella barra inferiore della finestra, e consentono in semplici "variazioni sul tema", ossia comprendono un periodimetro (viene indicata la frequenza in termini del suo periodo temporale in mS) ed un contatore (opzione "Counter"). Quest'ultima variante consente di contare il numero di periodi di una funzione periodica, cosa che si ottiene incrementando il contatore quando il livello del segnale supera una determinata soglia preimpostata; a tal proposito, la

selezione di questa opzione comporta la comparsa di una trackbar che si affianca al misuratore di livello locale presente sull'estremo destro della finestra; con essa è possibile impostare la soglia di intervento per il conteggio dei periodi. In ultimo, la checkbox "Hold" consente, quando attivata, di "mantenere" il valore corrente a video indefinitamente. NOTA: con questa opzione attivata le letture continuano ma il video NON viene aggiornato.

6.6 Il Voltmetro

Il voltmetro è nell'apparenza simile al frequenzimetro, ma rispetto ad esso enormemente più semplice algoritmicamente parlando. Esso infatti si limita a disegnare a video il valore in Volt (rms o di picco) dell'ampiezza massima del segnale, già in effetti rilevato nel ciclo principale (vedi metodo plot del capitolo due).

E' possibile selezionare (come accennato) la visualizzazione del valore efficace (RMS) e del valore di picco. E' possibile inoltre attivare la funzione Hold, ossia il mantenimento ad oltranza della lettura presente a video al momento dell'inserimento della funzione "hold"

6.7 I filtri digitali

Questa opzione rimanda direttamente ad una sottocartella della finestra di settings, pertanto si rimanda alla sezione 6.1.5 ad essa relativa ed alla corrispondente sezione del capitolo secondo per ulteriori approfondimenti (2.7); si veda anche il paragrafo 4.6 per gli algoritmi usati.

6.8 La cattura dei segnali

Visual Analyser è in grado di catturare sia i campioni acquisiti dalla scheda sonora che vengono direttamente inviati all'oscilloscopio, e sia lo spettro calcolato. La cattura si può effettuare cliccando sul bottone corrispondente nella finestra principale (bottone "capture scope" e bottone "capture Spectrum"). La prima funzione che analizzeremo è quella relativa alla cattura dei campioni dell'oscilloscopio, che è poi quella più ricca di opzioni. Una denominazione alternativa per queste due modalità di cattura è quella già usata altrove in questo testo, e che definisce la cattura dei campioni dell'oscilloscopio come "cattura del segnale nel dominio del tempo" e l'altra "cattura del segnale nel dominio della frequenza", senza dubbio più rigorosa e tramite la quale definiamo i due sottoparagrafi successivi

6.8.1 La cattura del segnale nel dominio del tempo

Per poter usare la funzione di cattura dei segnali nel dominio del tempo, invocabile dalla finestra principale tramite il bottone "Capture scope", si dovrà preventivamente "accendere" Visual Analyser tramite il bottone "on". Questo è ovviamente indispensabile perché altrimenti non vi sarebbero campioni da catturare. La finestra che compare a video dopo il tempo fisico necessario all'acquisizione dei campioni (secondo le opzioni descritte in 6.1.7) visualizza i campioni acquisiti e consente di effettuare numerose ed interessanti opzioni su di essi. Per esempio è possibile selezionare una parte del grafico visualizzato, trascinarlo in tutte le direzioni, zoomarlo e de-zoomarlo. Il tutto può essere realizzato tramite mouse nella seguente maniera:

- Zoom in: evidenziare la zona da ingrandire tramite il tasto sinistro del mouse; cliccare e mantenere premuto con il tasto destro e

contemporaneamente tracciare l'area da ingrandire definendo un rettangolo che parte dall'alto verso il basso;

- Zoom out: per eliminare lo zoom, usare la stessa procedura del passo precedente ma tracciando un rettangolo dal basso verso l'alto; in questo caso la dimensione del rettangolo non conta e serve solo a eliminare qualsiasi fattore di zoom applicato
- Spostamento del grafico: per "muovere" il grafico in qualsiasi direzione usare il tasto destro del mouse; premere in un qualsiasi punto del disegno, mantenere premuto e muovere il mouse

In maniera perfettamente equivalente è possibile utilizzare le trackbar presenti lungo gli assi del grafico e i bottoni presenti nei riquadri "Time e Spectrum".
Da notare che , come vedremo nelle righe successive, è possibile effettuare la trasformata di Fourier dei campioni selezionati (zoomati) ed anche la conversione digitale analogica. In tal modo anche in questa finestra sarà possibile, in tempo non reale, avere lo spettro del segnale (ma di una ben determinata porzione) ed il segnale ricostruito applicando l'algoritmo senza semplificazioni.

Il menu presente nella finestra è composto dalle seguenti opzioni:

[menu "file"]

- Open = consente di aprire un file precedentemente salvato
- Save = consente di salvare un grafico
- Save as text = salva un grafico come file testo (per uso con altri programmi)
- Save to clipboard = salva il grafico in clipboard
- Print white background = stampa con sfondo bianco (per risparmiare inchiostro)
- Print time = stampa il grafico dei campioni
- Print spectrum = stampa lo spettro

- Close = chiude la finestra

[menu D/A]

- D/A ch A = esegue la conversione digitale analogico dei campioni del canale A
- D/A ch B = esegue la conversione digitale analogico dei campioni del canale B

[menu X-axis]

- Marks time = nel grafico dei campioni nel dominio del tempo abilita i marks
- Logarithmic time = scala logaritmica dei campioni nel tempo
- Marks spectrum = nel grafico dei campioni nel dominio della frequenza abilita i marks
- Logarithmic spectrum = scala logaritmica dei campioni nella frequenza

[menu View]

- Spectrum = consente di visualizzare lo spettro calcolato a partire dai campioni effettivamente visibili sullo schermo
- Time = consente di visualizzare i campioni acquisiti

Da notare che i campioni catturati e lo spettro eventualmente calcolato tramite l'opzione "Spectrum" del menu "View" possono essere visualizzati contemporaneamente, come visibile in figura 38

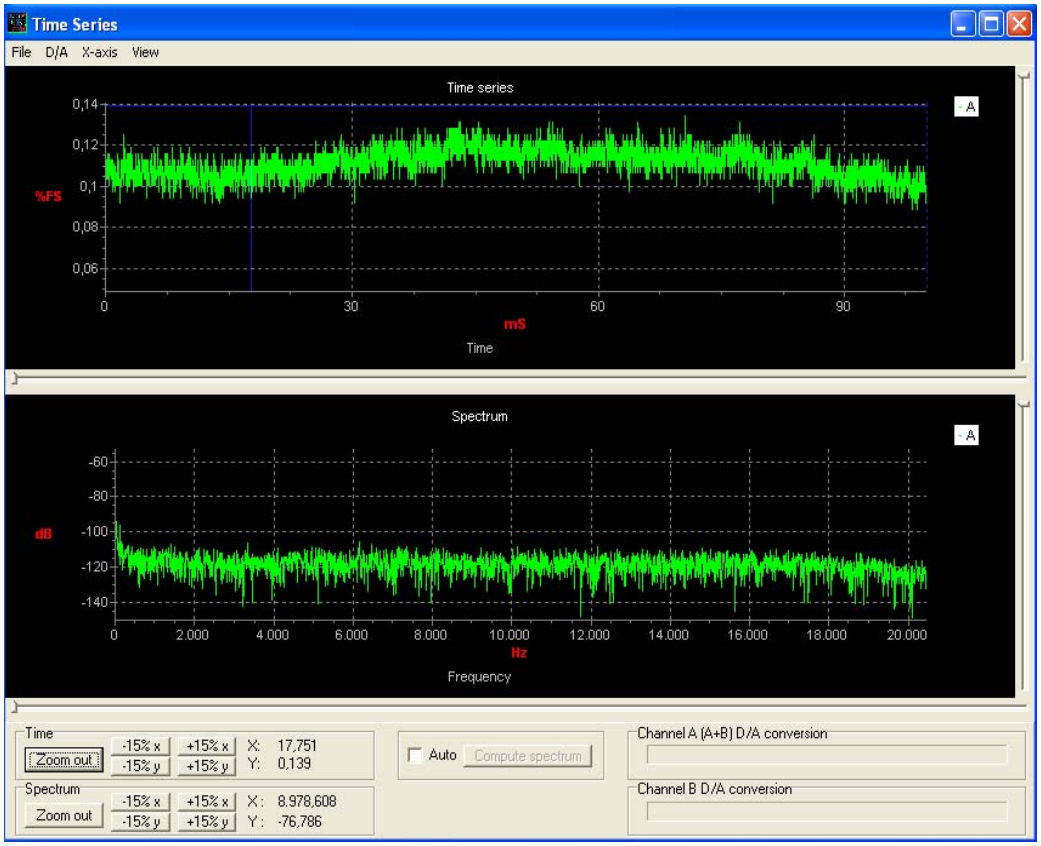


Figura 38 : la finestra di cattura dei segnali nel dominio del tempo

6.8.2 La cattura del segnale nel dominio della frequenza

Anche per questa funzione sarà indispensabile aver preventivamente mandato in "esecuzione" le routine di acquisizione del segnale dalla scheda sonora. Per acquisire i campioni sarà sufficiente cliccare sul bottone "capture spectrum" in finestra principale; la finestra è quella di figura 39. In essa sarà possibile effettuare tutte le operazione di zoom e spostamento grafico descritte nella sezione precedente; Il menù "File" e "X-axis" saranno parimenti dotate delle medesime opzioni.

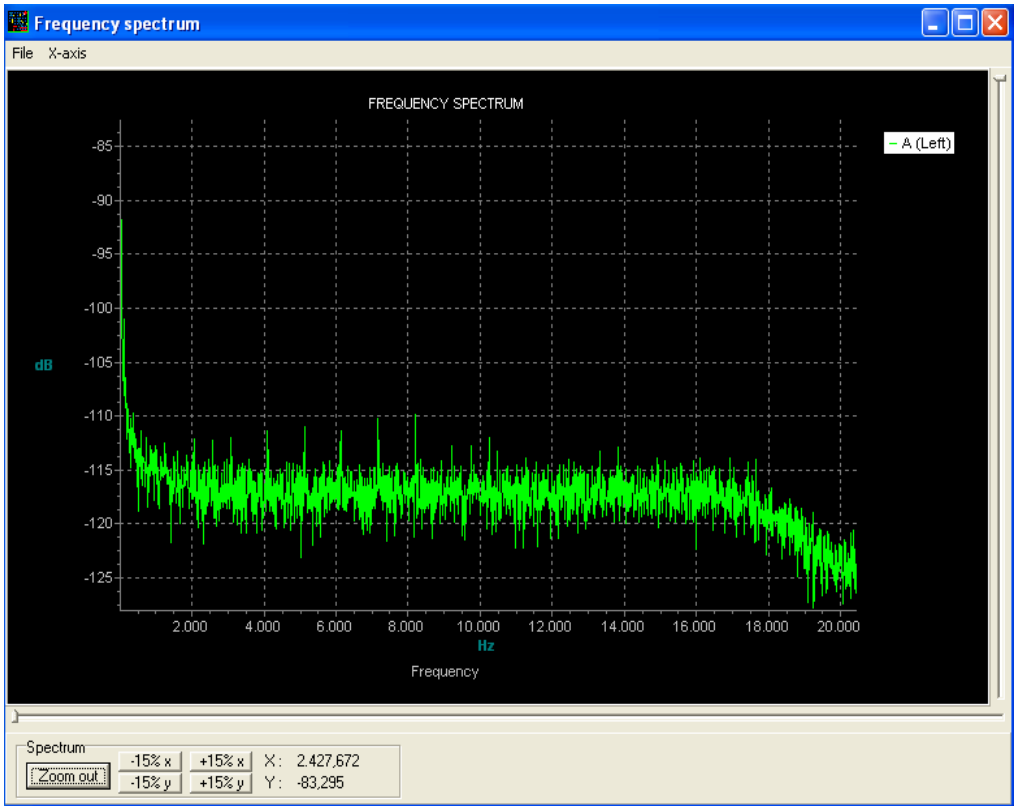


Figura 39 : la finestra di cattura dei segnali nel dominio della frequenza

Capitolo 7

Esempi d'uso

7.0 Introduzione

Visual Analyser è un programma che ha riscosso un notevole successo di pubblico grazie alla diffusione (gratuita) ottenuta tramite Internet; Un considerevole "feedback" da parte di numerosissimi utilizzatori sparsi ovunque nel mondo, ha consentito una messa a punto particolarmente efficace, ed ha permesso di aggiungere continuamente nuove funzioni sulla base di esigenze individuate nell'uso quotidiano da una molteplicità di utenti diversi. Inoltre, l'uso su macchine con differenti versioni del sistema operativo e configurazioni hardware, ha permesso di collaudare il programma anche dal punto di vista della compatibilità verso le macchine reali.

La disponibilità in laboratorio di strumenti quali il generatore di funzioni, l'oscilloscopio e l'analizzatore di spettro consente di effettuare una delle misure più classiche e parimenti più utili che si conoscano: la misura della risposta in frequenza di un dispositivo (per esempio di un amplificatore). Essa costituisce il nucleo di questo capitolo. Successivamente affronteremo la sintesi di una forma d'onda arbitraria a partire dal suo sviluppo in serie, per finire con un esempio pratico di calibrazione.

Gli usi di laboratorio cui Visual Analyser può essere destinato sono ovviamente infiniti, sebbene limitati al campo delle frequenze che la scheda sonora adottata

può trattare, e che normalmente arrivano sino ai 96 kHz di banda passante, corrispondenti a 192 kHz di frequenza di campionamento.
Varie aziende, per lo operanti in ambito musicale (studi di registrazione, piccoli produttori di dispositivi audio e aziende che si occupano di formazione) hanno praticamente adottato Visual Analyser come strumento di base, e nei loro siti è possibile trovare una precisa indicazione delle misure che con esso hanno effettuato (vedi "conclusioni" e "bibliografia").

7.1 La misura della risposta in frequenza

Generalmente la risposta in frequenza di un dispositivo elettronico (ma è un concetto che ha senso anche per altre realtà fisiche) tende ad identificarsi con il diagramma delle ampiezze dello spettro del segnale in uscita. In altre parole, descrive il comportamento del dispositivo al variare della frequenza del segnale in ingresso; in taluni casi esso è desiderato completamente "piatto", ossia si cerca di ottenere che esso amplifichi allo stesso modo tutte le frequenze della banda d'interesse (per esempio un amplificatore audio alta fedeltà) oppure che abbia un comportamento marcatamente selettivo (per esempio un filtro passa basso). In taluni casi il dispositivo può avere una risposta in frequenza impostabile tramite determinati controlli (hardware o software), che devono avere una efficacia la cui validità può essere parimenti testata con una misura di risposta in frequenza. Ancora, la misura della risposta in frequenza può essere relativa ad un ambiente, quale per esempio una stanza od una sala per concerti. In questo caso il discorso resta sostanzialmente simile, sebbene con delle variazioni: il segnale in ingresso sarà generato da un altoparlante (segnale d'ingresso) e rilevato da un microfono (segnale d'uscita); ammesso che la risposta propria della catena altoparlante-microfono-amplificatore sia completamente "piatta" o resa tale tramite vari accorgimenti, la risposta rilevata sarà quella relativa all'ambiente.

Esistono generalmente molteplici strategie adottabili per la rilevazione della risposta in frequenza di un dispositivo elettronico; la più classica è quella che prevede l'uso di un generatore di segnale sinusoidale, un oscilloscopio e un pezzo di carta; oppure, se l'oscilloscopio non è disponibile, è sufficiente anche un indicatore di livello rms, purchè dotato della necessaria banda passante. La strategia è semplice: si applica il segnale all'ingresso del dispositivo, e se ne rileva l'ampiezza all'uscita; questo per vari "punti", corrispondenti a diverse frequenze. Per ogni frequenza usata si traccia un punto su di un grafico che ha in ascissa la frequenza ed in ordinata l'ampiezza, generalmente calcolata in dB. Così facendo, ed usando un numero significativo di punti, si ottiene il diagramma delle ampiezze, ossia la voluta risposta in frequenza dell'amplificatore. Va da se che il segnale in ingresso dovrebbe essere "lineare", ossia avere la stessa ampiezza per ogni frequenza generata. Una versione appena più sofisticata della metodologia descritta, dovrebbe prevedere il calcolo del rapporto tra segnale d'ingresso e d'uscita, in maniera da rendere la misura relativamente indipendente dalle non linearità proprie del generatore di funzioni.

Per rendere la procedura più veloce e ricca di punti, è possibile usare un generatore di sweep sinusoidale, ossia un generatore che usi un segnale che varia con continuità tra due estremi della gamma audio, per esempio tra 20 e 20.000 Hertz. Versioni più professionali (di qualche tempo fa) utilizzavano un generatore di sweep, che forniva anche un segnale a dente di sega utilizzato per sincronizzare direttamente la base dei tempi dell'oscilloscopio con la "spazzolata" in frequenza, e permetteva di ottenere direttamente il disegno della risposta in frequenza sullo schermo dell'oscilloscopio. In pratica il dente di sega inizia quando lo sweep inizia a generare la sua frequenza più bassa e finisce quando si è raggiunta quella più elevata; per poi ricominciare daccapo. In tal modo, pilotando l'asse X dell'oscilloscopio con il dente di sega, e l'asse Y con le ampiezze d'uscita delle armoniche, si ottiene direttamente a video il disegno della risposta in frequenza.

Un'altra possibile strategia, che è quella che useremo per la nostra prova, prevede l'uso di un segnale a banda larga, detto rumore bianco, caratterizzato dall'avere uno spettro "piatto" nella banda di interesse. Invero il rumore bianco "puro" dovrebbe avere un contenuto spettrale virtualmente infinito, altrimenti si ricade in una delle categorie di rumori "colorati" che hanno ben differenti caratteristiche di distribuzione spettrale (per esempio il rumore rosa). Anche se, nella realtà, non esiste un rumore perfettamente bianco, per le inevitabili caratteristiche fisiche del mondo reale. In questa sede considereremo come rumore bianco un segnale che copra uniformemente tutto lo spettro del nostro "universo", che in questo caso è il campo prettamente audio, ossia il range compreso tra i 20 Hz ed i 20.000 Hz. Useremo una schema pratico di principio quale quello indicato nella figura 40; in questo schema si è identificato un arbitrario "generatore di segnale" come generatore del segnale in ingresso (rumore bianco) ma in effetti quello che si può tranquillamente utilizzare (ed è effettivamente utilizzato) è quello interno a Visual Analyser, opportunamente configurato.

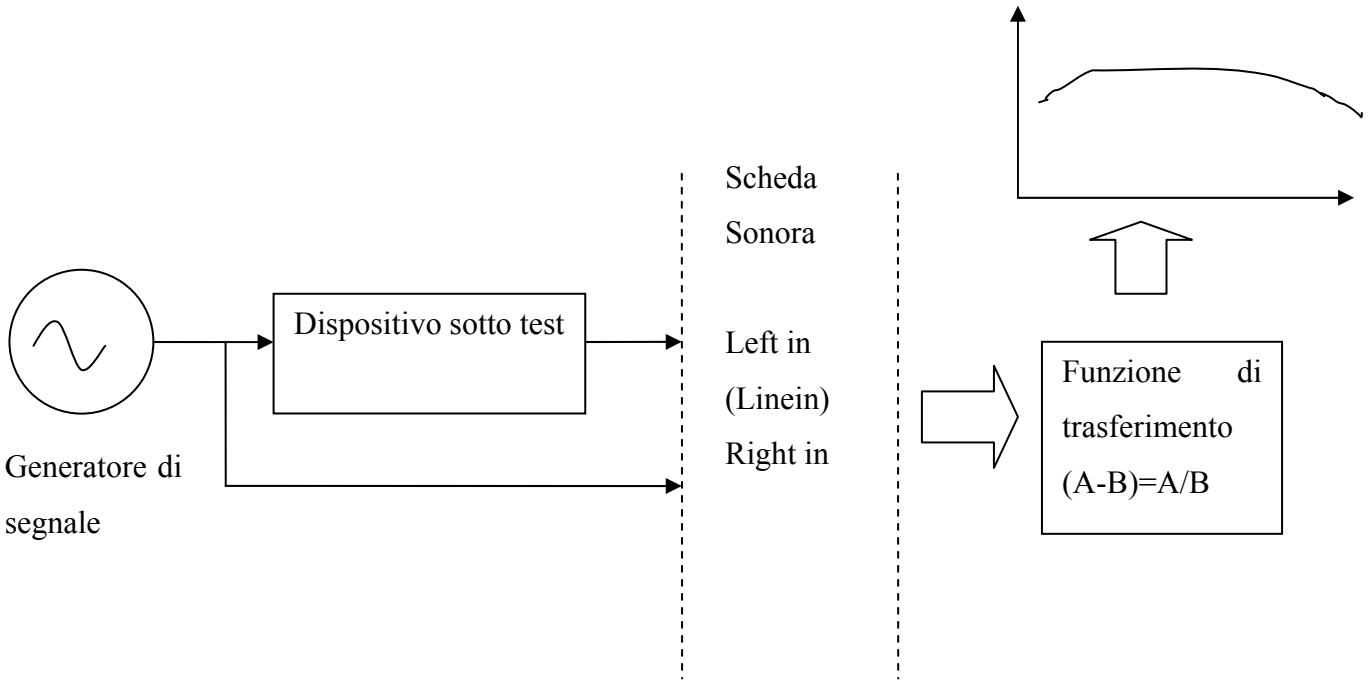


Figura 40: lo schema di principio per la misura

Per effettuare la misura praticamente ed avere la massima flessibilità possibile, si è scelto di usare un "equalizzatore grafico", ossia un dispositivo che al suo interno contiene:

- un semplice amplificatore
- un set di filtri

Tramite esso sarà infatti possibile tracciare la risposta in frequenza dell'amplificatore con l'aggiunta di poter variare a piacimento la risposta in frequenza del dispositivo, dimostrando la validità della procedura e la capacità di Visual Analyser di effettuare misure in tempo reale.

L'equalizzatore grafico è un dispositivo che si introduce nella catena di amplificazione di un segnale audio per poterne variare a piacimento la banda passante, entro i limiti delle capacità del dispositivo; esso serve per esempio a correggere la risposta in frequenza tipica di un ambiente (per esempio che assorbe eccessivamente le alte frequenze) o le carenze di un sistema di diffusori, od un qualsiasi elemento delle catena; od anche, per assecondare i gusti dell'ascoltatore. Esso è normalmente composto da un set di filtri, centrati su un certo numero di frequenze (sette nel nostro caso) che agiscono su una "banda" di frequenze che si estende sino al valore del controllo immediatamente precedente e successivo; e la cui capacità d'intervento è modificabile tramite un semplice potenziometro a scorrimento (a "slide"). Tramite essi è possibile, con continuità, esaltare od attenuare una singola banda di frequenze. Si chiama equalizzatore grafico (anche) perché la curva che si delinea posizionando i singoli potenziometri ("l'inviluppo"), segue quella ideale che dovrebbe ottenersi come risposta in frequenza (vedi figura 41). Noi quindi testeremo:

- La risposta in frequenza in posizione “flat”, nella quale tutti i controlli sono posizionati nella posizione centrale;
- La risposta in frequenza ottenuta variando uno o più dei controlli; in particolare varieremo l’amplificazione della banda centrata sui 1000 Hz esaltandola al suo valore massimo e successivamente attenuandola al suo valore minimo

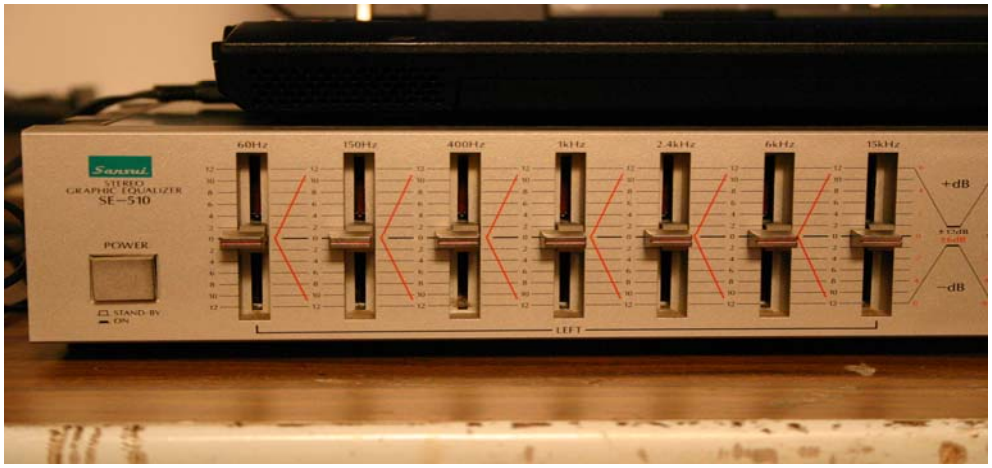


Figura 41: l’equalizzatore grafico in posizione “flat”

Le bande di frequenza dell’equalizzatore grafico (Sansui SE-510) sono le seguenti (in Hz):

- 60
- 150
- 400
- 1000
- 2400
- 6000
- 15000

Ossia con un andamento evidentemente di tipo logaritmico. Visual Analyser è stato pertanto impostato secondo le seguenti opzioni (sono indicate solo le opzioni modificate rispetto alle opzioni di default):

- Finestra principale a dimensioni massime
- Riquadro Y-axis in finestra principale, listbox "step" selezionato il valore 3dB
- Riquadro X-axis spuntata opzione "Log" (ossia asse delle X logaritmica in analizzatore di spettro)
- Riquadro Y-axis in finestra principale, average settato al valore 40
- Opzione "Channels" settata a Ch B/A = B – A ossia canale destro meno sinistro

Il generatore di funzioni è stato impostato nella modalità "loop", selezionando la forma d'onda "white noise"; le ampiezze relative del segnale generato sono state poste al valore di 30%; è stato usato lo schema di principio di figura 40, collegando direttamente i cavi senza nessun resistore di attenuazione e senza diodi di protezione. In particolare è stato usato il canale sinistro per monitorare il segnale in ingresso (A), ed il destro per il segnale d'uscita (B).

7.2 esecuzione pratica della misura

Effettuati i settaggi appena indicati, si può procedere all'accensione del generatore e mettere in "on" Visual Analyser. Posizionando i controlli dell'equalizzatore grafico in posizione "flat" otteniamo la curva indicata nella figura 42:

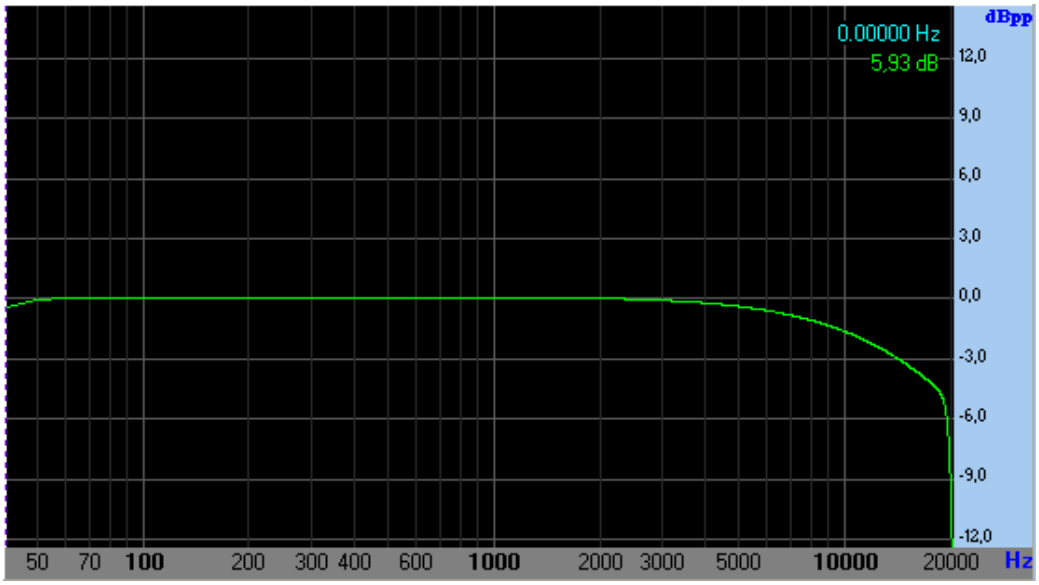


Figura 42: risposta con i controlli in posizione neutra

Si può osservare che il semplice amplificatore presente all'interno dell'equalizzatore grafico presenta una risposta in frequenza che si mantiene relativamente piatta sino ai 15 kHz (ossia si mantiene entro i +/- 3 dB) con marcata tendenza ad attenuare oltre questo valore. Data la classe dello strumento è un risultato accettabile. Posizionando il controllo della banda dei mille Hz nella posizione di massimo guadagno (indicata sulla scala come +12 dB) otteniamo il grafico di figura 43.

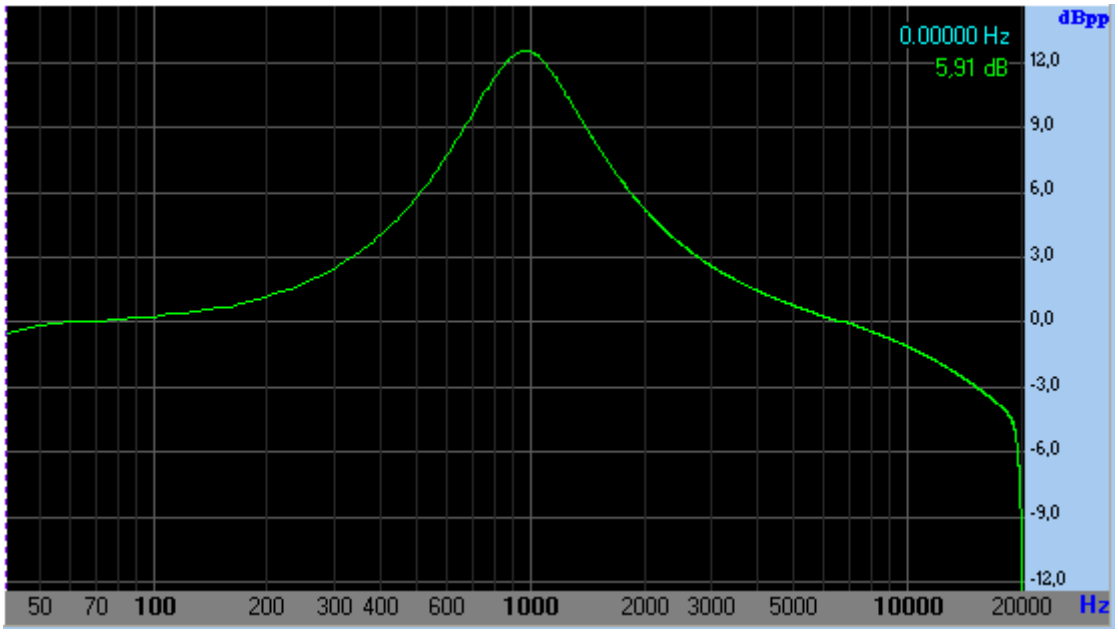


Figura 43 : con lo slide 1000Hz in posizione di massimo guadagno

Si osserva una perfetta coincidenza con quanto indicato dai controlli, ossia una esaltazione di 12 dB; nella posizione opposta otteniamo quanto riportato in figura 44 che continua a confermare la validità del filtro.

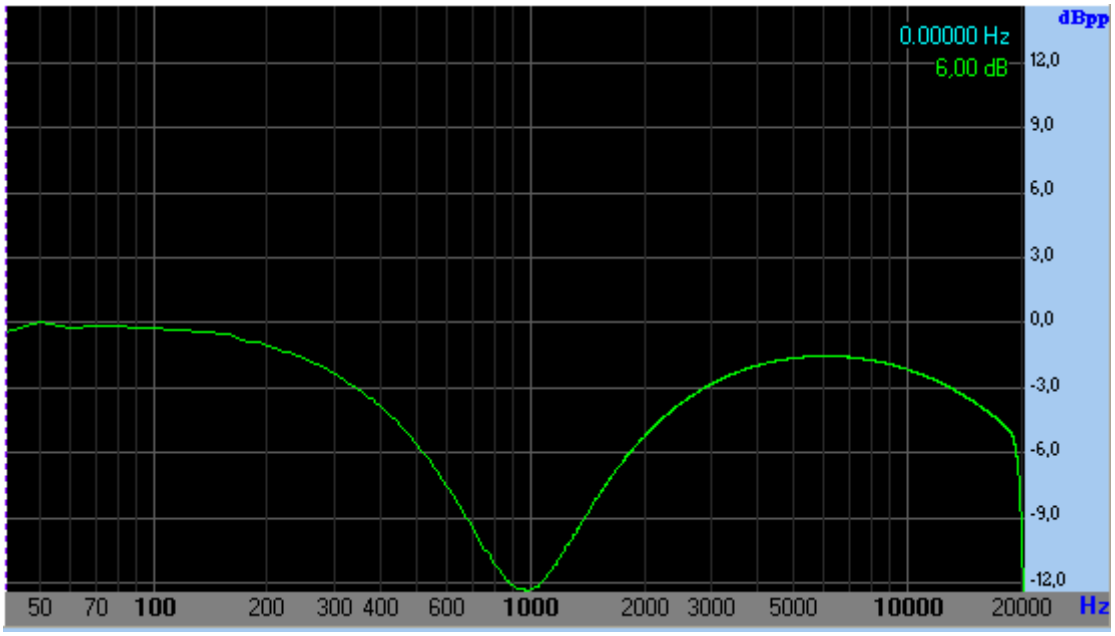


Figura 44: con lo slide 1000 Hz in posizione di minimo guadagno

Si riporta, in ultima battuta, l'immagine reale della prova effettuata; le precedenti schermate sono state ottenute "catturando" lo schermo mentre veniva effettuata la misura; l'immagine di figura 45 è stata ottenuta fotografando otticamente le apparecchiature durante la stessa esecuzione della misura.

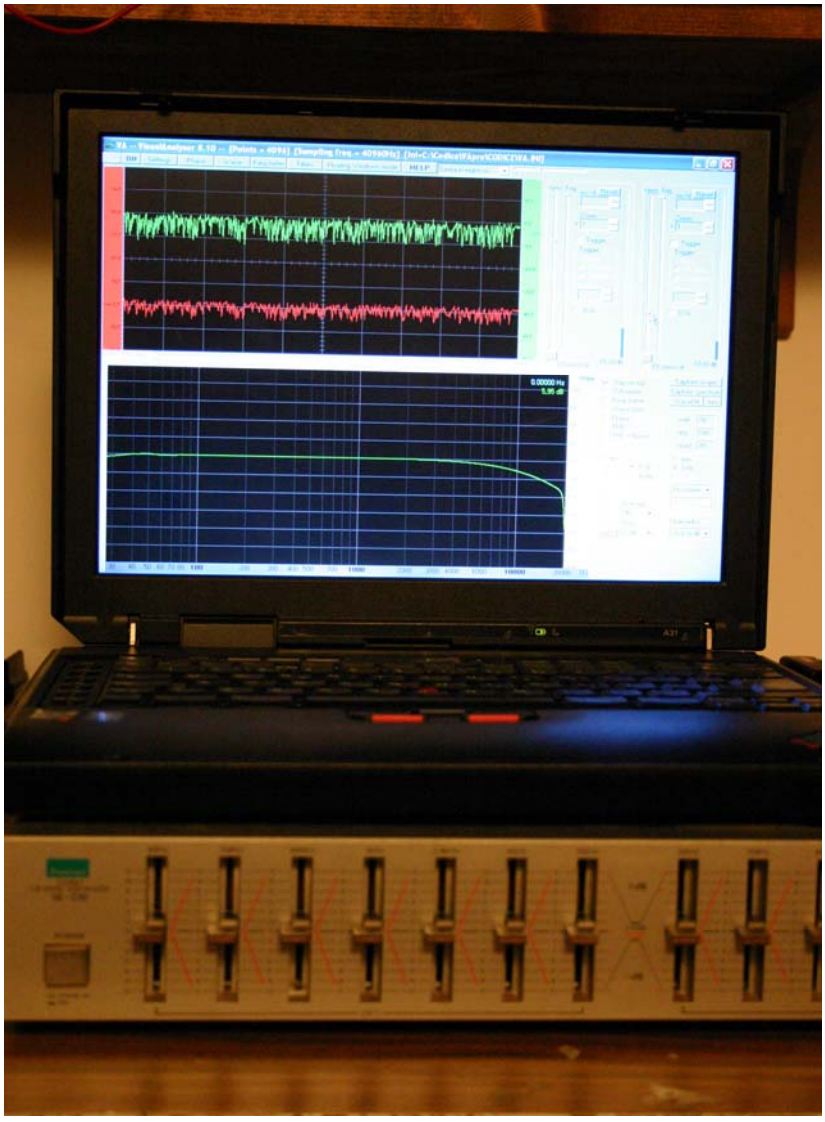


Figura 45: il banco di misura reale, controlli "flat"

Si osservi la figura; l'oscilloscopio traccia il rumore bianco in entrata al dispositivo (grafico verde) e quello in uscita (grafico rosso); i controlli dell'equalizzatore sono posti in posizione "neutra". Per finire, si riportano le

immagini “reali” con il controllo della banda dei 1000 Hz in posizione +12dB e -12dB.

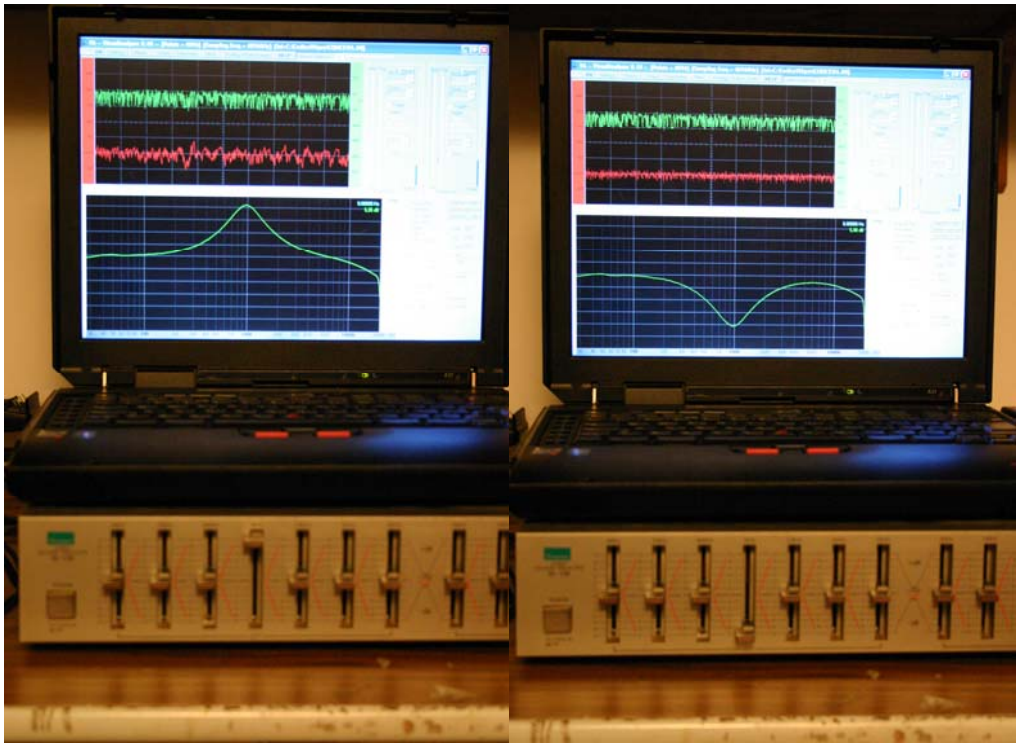


Figura 46: il banco di misura reale, controlli +12dB e -12dB

Si osservi che a questo punto sarebbe altresì possibile:

- Catturare i segnali con il tasto “Capture Spectrum” entrando nella finestra descritta in 6.8;
- Stampare il grafico ottenuto sfruttando le opzioni presenti nella finestra;
- Salvarlo come immagine;
- Salvarlo come file testo;
- Ingrandirlo, rimpicciolirlo e cambiare le scale ed i colori

Il tutto mentre Visual Analyser sta effettuando altre misure, in perfetto parallelismo.

Appendice

Le classi principali

In questa sezione si riporta la definizione delle classi usate da Visual Analyser e l'implementazione dei metodi più significativi; in particolare:

- Paragrafo A, classe Fft : effettua la trasformata di Fourier tramite algoritmo FFT; filtri digitali; finestre di smooth; analisi in terzi d'ottava; metodi per il calcolo delle grandezze in dB; calcolo della funzione average;
-

A. Classe Fft

```
class Fft
{
public:

   Fft (int Points, long sampleRate,
        int maxpoints, int SmWin,
        bool Qp, int DP, FILT Filt);
```

```
    Fft (int Points, long sampleRate, int SmWin);
    ~Fft ();

    inline  void Redefine (int Points, long sampleRate);
    int      Points () const { return _Points; }
    void    Transform (void);
    void    Transform (Complex *X);

    // Per capture time
    int      GetIter ();
    void    Transform  (int level, int step);
    void    Precompute (int level, int _2_l);
    void    UnTransform (void);
    void    CopyIn (SampleIter& iter, short ch,
                    bool nofilt, bool DIODE,
                    bool DCREMOVAL);
    void    CopyRaw (double Sample, int i);

// Filtri FIR ----------------------------------------
    void LowPass(double CutFreq);
    void HiPass(double CutFreq);
    void BandPass(double CutFreq1, double CutFreq2);
    void BandReject(double CutFreq1, double CutFreq2);
    void Convolve(double* SignalOut, int* SignalIn);
// ---------------------------------------------------


// Filtri IIR ----------------------------------------
    void Notch(double CutFreq1);
    void INVNotch(double CutFreq1);
    void IIRfilter(double* SignalOut, int* SignalIn);
```

```
// ------------------------------------------------

    AnsiString GetTHDNoise(void);
    void SetSampleRate (long sampleRate);
    AnsiString GetTHD();

    int GetNumPoints(void)
    void ComputeSmooth (int SmoothWin, bool CorrX4p);
    void ComputeFilter (void);
    void SetFilter(int ItemIndex, double f1, double f2);
    double Power(double A);
    int GetSmoothWin(void);
    double GetConversionFactor(int P);

    // ampiezza dell'armonica ì-esima
    double  GetIntensity (int i) const

    // ampiezza dell'armonica  ì-esima a gruppi di j
    double  GetIntensity (int i, int j) const
    void SetVzero(int ZeroDB);

    int GetVzero(void) const

    // fase della riga ì-esima in GRADI
    double  GetPHASE (int i) const

    // ampiezza della riga ì-esima in DB average
    double  GetDB (int i) const


    // ampiezza delle riga da i a i+j in DB  average
```

```
double  GetDB (int i, int j) const

// ampiezza della barra i-esima
double  GetDBOCTAVES (int i, double *From, double *To) const


// restituisce la posizione della barra a MASSIMA ampiezza
int  GetPosMaxAMPOCTAVES (double *From,
                          double *To,
                          int Bar,
                          int Start) const

// restituisce la posizione della barra a MINIMA ampiezza
int  GetPosMinAMPOCTAVES (double *From,
                          double *To,
                          int Bar,
                          int Start) const


// ampiezza della riga ì-esima in DB mediata
double  GetDBAverage (int i) const



// ampiezza delle riga da i a i+j in DB mediata
double  GetDBAverage (int i, int j) const


// restituisce la riga spettrale a massima ampiezza
float  GetMaxFreq (void) const
```

// restituisce la riga spettrale a minima ampiezza

float GetMinFreq (**void**) **const**

// restituisce la posizione della riga spettrale a massima ampiezza

int GetPosMaxFreq (int Start, int Stop) const

// restituisce la posizione della riga spettrale a minima ampiezza

int GetPosMinFreq (**int** Start, **int** Stop) const

void SetStart(**int** st)

int GetStart(**void**)

// Restituisce la media effettuata su tutti i campioni

double GetAverageAmp (**void**) **const**

double GetMaxAmp (**void**) **const**

// massima ampiezza delle armoniche a gruppi di M

double GetMaxAmp (**int** M) **const**

double GetFrequency (**int** point) **const**

int HzToPoint (**double** freq) **const**

double PointToHz (**int** Point) **const**

int MaxFreq() **const** { **return** _sampleRate; }

int Tape (**int** i) **const**

Complex __X(**int** i) **const**

```
    int    SetPhaseAmplitudeThreshold (double threshold)
    int    GetSampleRate (void);
    int    SetPoolAverage (int dim);
    int    SetPoolAverageNOLOCK (int dim);

    void ZeroAverage(void);
    bool AddToAverage(int NA);
    void Average(int n);
    int GetCountAverage(void);
    double GetPhaseCorr(int Index);



private:

     int             *tape;
    double           *_aTape;
    double           *TempTape;
    double           SmoothCorrection;
    double           SmoothLinear;
    double           *Smooth;
    int              SmoothWin;
    bool             QP;
    int                  _Points;
    int                  halfPoints;
    int              MAXPOINTS;
    long         _sampleRate;
    int                  _logPoints;
    double               _sqrtPoints;
    double               _sqrtPointsSQR;
    int               *_aBitRev;
```

```
Complex         *_X;
Complex         *_XT;
double          *Av;
Complex         **_W;
double          **PoolFFT;
int             PosPool;
int             _Average;
double          Inv_Average;
int             Vzero;
double          Inv_Vzero;
double          Log_Vzero;
int             StT;
double          AmpFatt;
int             CountAverage;
bool            FirstFill;
int             DimPool;
double          *Kernel;
double          *SignalMEMO;
double          *SigTemp;
double          *PhaseCorr;
int             LenK;
FILT            Filter;
double          a0,a1,a2,b1,b2,xn_1,xn_2,yn_1,yn_2;
double          DCa0, DCa1, DCb1, DCyn_1, dataprev;
double          xnp1,xnp2,ynp1,ynp2;
double          PhaseAmplitudeThreshold;
int             CounterPhase;
long double     Tb, Fi, ResFFT;
long double     Residuo;
double          Inv_Points;
```

```
    double   SmoothOctaves;   void PutAt ( int i, double val );
    inline    void DCremoval (double &data);

};
```

Conclusioni

Visual Analyser è un programma che dimostra come i moderni PC siano dei circuiti universali che tramite un software si trasformano in dispositivi dedicati alla soluzione di una determinata classe di problemi; è un programma concorrente ed è anche un ottimo esempio di programma in tempo reale e di applicazione della materia nota come "Elaborazione Numerica dei Segnali". Ed in ultimo, è un programma che costituisce un anello di congiunzione tra il mondo dell'elettronica e dell'informatica, oramai sempre più strettamente legate l'una all'altra. E, cosa più importante di tutte, ha regalato a me ed a tutti i miei "collaboratori virtuali", momenti di vero e proprio godimento intellettuale in ogni fase della sua creazione.

Visual Analyser ha richiesto un faraonico impegno che è iniziato sin dall'anno 2000 (ancora in corso d'opera), ed è composto da oltre 35.000 linee di codice C++ suddivise in circa 140 files. La realizzazione pratica è andata di pari passo con la sua diffusione su internet, ottenuta tramite siti web dai quali è possibile effettuare gratuitamente il download. Grazie ad un numero di utilizzatori enorme, praticamente situati in ogni angolo del mondo, ed un numero di download complessivi pari a circa 20.000, si è potuto mettere a punto in qualche anno un programma che, data la sua complessità, avrebbe richiesto tempi ben maggiori. I feedback inviati tramite posta elettronica hanno consentito oltre ad un potente debug, anche di utilizzare idee e suggerimenti per aggiungere nuove funzionalità, e quindi per ricominciare ogni volta daccapo. Visual Analyser è stato oggetto di articoli su riviste come "Elettronica In" e "Fare elettronica" (vedi Bibliografia), così come di recensioni su siti quali www.electronics-lab.com e citato in numerosissimi forum. Esso è stato

adottato come software da studi di registrazione come “the audio lab” (www.theaudiolab.com) e aziende di dispositivi audio (http://microphonium.blogspot.com)

.

Bibliografia

- Daniel Pierre Bovet: appunti dal corso di "Sistemi Operativi", Università di Tor Vergata
- A.V. Oppenheim, R.W. Schafer "Elaborazione numerica dei segnali", Franco Angeli editore
- Andrew S. Tanembaum "Operating Systems"
- Steven W. Smith "The scientist and Engineer's Guide to Digital Signal Processing" Analog Device
- Jarrod Hollingworth, Bob Swart, Mark Cashman, Paul Gustavson "Borland C++ Developer's guide", ed. Sams
- Sommerville "Software Engineering" ed. Addison Wesley
- Alfredo Accattatis "Oscilloscopio ed Analizzatore di spettro su PC", rivista "Elettronica In" del 12/2004
- Alfredo Accattatis "Visual Analyser: un programma Windows per la simulazione di strumenti di misura e generazione di forme d'onda" rivista "Fare Elettronica" del 07/2006
- Galeazzo costantini, Ugo Guernelli "Strumentazione e misure elettroniche", ed. Zanichelli