

**PROGETTO FINALIZZATO
SISTEMI INFORMATICI E CALCOLO PARALLELO**

SOTTOPROGETTO 7
Sistemi di Supporto al Lavoro Intellettuale
Coordinatore Cesare Maioli

Isabella Pighi[^]

**NOTE TECNICHE DEL MODULO
TIMBRELAB**

N. M/29

10/92

Rapporto Tecnico Serie MUSIC

[^] LIM - Laboratorio di Informatica Musicale
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano.

INDICE

Capitolo 1 Introduzione

- 1.1 Generalità sull'applicazione TimbreLab
- 1.2 Requisiti hardware per il corretto funzionamento
- 1.3 Requisiti software per il corretto funzionamento
- 1.4 Requisiti firmware per il corretto funzionamento
- 1.5 Note sulla fase di testing
- 1.6 Note per l'installazione

Capitolo 2 Strutture dati

- 2.1 Strutture dati interne
 - 2.1.1 LabVIEW2.2
 - 2.1.2 Code Interface Node
- 2.2 Strutture dati esterne
 - 2.2.1 SDI
 - 2.2.2 SDII

Capitolo 3 Struttura dello Strumento

- 3.1 Fase di computazione
- 3.2 La sessione di lavoro
- 3.3 Rete di calcolo
 - 3.3.1 Generatori
 - 3.3.2 Operandi
- 3.4 Strutture di alcuni moduli implementati
 - 3.4.1 DrawerLab
 - 3.4.2 ReaderLab
 - 3.4.3 BandPassLab

Capitolo 4 Algoritmi

- 4.1 Filtri
- 4.2 Rumori
- 4.3 Grafici

Appendice A

Appendice B

1 Introduzione

1.1 Generalità sull'applicazione TimbreLab

TimbreLab è stata sviluppata da Isabella Pighi, presso il LIM (Laboratorio di Informatica Musicale), dipartimento di Scienze dell'Informazione, Università degli Studi di Milano. L'applicazione (eseguibile in ambiente Macintosh) consiste in un sistema ad architettura aperta per la programmazione timbrica con interfaccia iconica.

1.2 Requisiti hardware per il corretto funzionamento

- Gli appartenenti alla famiglia Macintosh II
- Digidesign Sound Accelerator Card
- Digidesign Audiomedia

1.3 Requisiti software per il corretto funzionamento

- System versione T1-6.0.8, Finder versione T1-6.1.7
- LabVIEW versione 2.2

1.4 Requisiti firmware per il corretto funzionamento

- Tutte le ROM dalla versione a 128 kbyte in poi.

1.5 Note sulla fase di testing

Configurazione usata:

- Macintosh IICx
- Digidesign Audiomedia
- System T1-6.0.8
- Finder T1-6.1.7
- LabVIEW 2.2

Sono stati eseguiti cinque strumenti diversi con grado di complessità crescente, nella tabella sottostante vengono riportati i tempi d'esecuzione in relazione al numero di campioni prodotti:

LA: un generatore sinusoidale.

10.000 campioni	30"
44.000 campioni	2' 05"
100 000 campioni	4' 50"

MFrequenza: tre oscillatori sinusoidali in cascata con inviluppo triangolare.

10.000 campioni	41"
44.000 campioni	3' 15"
100 000 campioni	6' 59"

Arpeggio: somma di quattro generatori sinusoidali ritardati con moduli di delay.

10.000 campioni	48"
44.000 campioni	3' 40"
100 000 campioni	8' 20"

Rumore+Seno: rumore scuro sommato ad un generatore sinusoidale.

10.000 campioni	59"
44.000 campioni	4' 10"
100 000 campioni	9' 12"

Seno+Filtro+lettore file: generatore sinusoidale filtrato da un filtro passabanda.

10.000 campioni	2' 40"
44.000 campioni	4' 40"
100 000 campioni	9' 40"

Dalla tabella si può intuire che:

- a) il numero di sottostrumenti influisce in modo approssimativamente lineare sul tempo di esecuzione;
- b) esistono degli strumenti più costosi nei termini di tempo, questi sono i filtri e i rumori, i quali hanno come supporto particolari strutture dati;
- c) i lettori di file in generale sono più lenti poiché necessitano di continui accessi a disco.

Nella fase di testing é risultato equivalente l'uso di strumenti a coefficienti costanti o variabili ed é stato riscontrato che la presenza di altri task rallenta gravemente il tempo di esecuzione.

1.6 Note per l'installazione

L'applicazione consiste in un'estensione della libreria di LabVIEW, le funzioni sono contenute in una cartella di nome **TimbreLab**, formata dalle seguenti 5 sottocartelle :

Envelope contenente i seguenti sottostrumenti di LabVIEW:

EnvSin:	generatore di inviluppo sinusoidale a coefficienti variabili;
EnvRamp:	generatore di inviluppo rampa a coefficienti variabili;
EnvTriangle:	generatore di inviluppo triangolare a coefficienti variabili;

EnvSquare:	generatore di inviluppo quadro a coefficienti variabili;
EnvImpulse:	generatore di inviluppo impulso a coefficienti variabili;
kEnvSin:	generatore di inviluppo sinusoidale a coefficienti costanti;
kEnvRamp:	generatore di inviluppo rampa a coefficienti costanti;
kEnvTriangle:	generatore di inviluppo triangolare a coefficienti costanti;
kEnvSquare:	generatore di inviluppo quadro a coefficienti costanti;
kEnvImpulse:	generatore di inviluppo impulso a coefficienti costanti.

Noise contenente i seguenti sottostrumenti di LabVIEW:

WhiteNoise:	generatore di rumore bianco a coefficienti variabili;
PinkNoise:	generatore di rumore rosa a coefficienti variabili;
BrownNoise:	generatore di rumore scuro a coefficienti variabili;
kWhiteNoise:	generatore di rumore bianco a coefficienti costanti;
kPinkNoise:	generatore di rumore rosa a coefficienti costanti;

kBrownNoise: generatore di rumore scuro a coefficienti costanti.

Oscillator contenente i seguenti sottostrumenti di LabVIEW:

OscillSin: generatore d'onde sinusoidali a coefficienti variabili;

OscillRamp: generatore d'onde rampa a coefficienti variabili;

OscillTriangle: generatore d'onde triangolari a coefficienti variabili;

OscillSquare: generatore d'onde quadre a coefficienti variabili;

OscillImpulse: generatore d'onde impulso a coefficienti variabili;

kOscillSin: generatore d'onde sinusoidali a coefficienti costanti;

kOscillRamp: generatore d'onde rampa a coefficienti costanti;

kOscillTriangle: generatore d'onde triangolari a coefficienti costanti;

kOscillSquare: generatore d'onde quadre a coefficienti costanti;

kOscillImpulse: generatore d'onde impulso a coefficienti costanti.

TimbreLabTools contenente i seguenti VI di LabVIEW:

InitLab:	modulo di inizializzazione necessario in tutte le sessioni di sintesi;
StoreLab:	modulo di ricovero e analisi del segnale prodotto necessario in ogni sessione di sintesi;
DelayLab:	modulo generatore di linee di ritardo;
StopLab:	modulo di interruzione;
DrawerLab:	sottostrumento attraverso il quale é possibile introdurre funzioni affini per intervalli;
MultiDrawerLab:	sottostrumento attraverso il quale é possibile introdurre un insieme di funzioni affini per intervalli;
ReaderLab:	sottostrumento attraverso il quale é possibile leggere file formato SoundDesignerI, SoundDesignerII mono con parametri a coefficienti variabili;
kReaderLab:	sottostrumento attraverso il quale é possibile leggere file formato SoundDesignerI, SoundDesignerII mono con parametri a coefficienti costanti;
BandPassLab:	filtro passa banda a coefficienti variabili;
kBandPassLab:	filtro passa banda a coefficienti costanti;
BandStopLab:	filtro taglia banda a coefficienti variabili;
kBandStopLab:	filtro taglia banda a coefficienti costanti;
GeneralFilterLab:	filtro generale a coefficienti variabili;

kGeneralFilterLab: filtro generale a coefficienti costanti.

Il corretto funzionamento di TimbreLab é assicurato dalla presenza di una seconda cartella chiamata **Supporto** la quale é formata da 5 sottocartelle:

varglobali contenente le variabili globali dell'applicazione:

gvNumeroCampioni:	numero di campioni del segnale;
gvsamplingfrequency:	frequenza di campionamento del segnale;
gvfase:	campione corrente;
gvhandle:	indirizzo della zona di memoria dove ricoverare i campioni prodotti;
gvrefs:	array contenente i reference number dei file aperti da ReaderLab e kReaderLab;
gvIsFirstTime:	semaforo della zona di inizializzazione delle variabili globali;
gvlast:	struttura contenete i risultati parziali prodotti dai generatori di rumore;
gvzoomsi:	semaforo della zona di zoom;
gvartrasf:	array coinvolto nel calcolo dello spettro di potenza;
gvrefStereo:	reference number del file creato formato SDII stereo;
gvSemCanS:	componente del sistema di semafori della di salvataggio su file formato SDII stereo;
gvSemStereo:	componente del sistema di semafori della di salvataggio su file formato SDII stereo;

gvSemCanD:	componente del sistema di semafori della di salvataggio su file formato SDII stereo;
gvar3D:	array coinvolto nel calcolo dello spettro 3D;
gvMatH_in:	indirizzo di memoria della zona di calcolo dedicata ai filtri;
gvMatH_out:	indirizzo di memoria della zona di calcolo dedicata ai filtri;
gvNFiltri:	numero di filtri.

Strumenti contenente i seguenti strumenti di LabVIEW:

InterfaceLab:	sottostrumento di StoreLab, mediante esso è possibile ascoltare, analizzare e salvare il segnale prodotto;
Spect3DLab:	sottostrumento di InterfaceLab, realizza lo spettro 3D;
Spect2DLab:	sottostrumento di InterfaceLab, realizza lo spettro 2D;
AllocateLab:	sottostrumento di InitLab, alloca una porzione di RAM utilizzata come area di parcheggio per il segnale;
TrasArrayLab:	sottostrumento di Spect2DLab, prepara un array di campioni per il calcolo della trasformata;
PotenzaLab:	sottostrumento di Spect2DLab e Spect3DLab, calcola la funzione spettro di potenza;

RubberLab:	pulisce lo schermo dai diagrammi generati da routine esterne a LabVIEW;
BankLab:	sottostrumento di StoreLab, ricovera nella zona di memoria allocata i campioni prodotti nel ciclo di sintesi;
MatriceLab:	sottostrumento di InitLab, prepara le strutture utilizzate dai filtri;
ClarMathLab:	sottostrumento di pulitura della memoria;
3DrawLab:	sottostrumento di Spect3DLab, realizza il diagramma dello spettro 3D;
SvegliaLab:	sottostrumento di ReaderLab e di kReaderLab, legge da file esterno il campione corrente;
SalvoLab:	sottostrumento di InterfaceLab, salva su disco i campioni del segnale prodotto;
DisposeLab:	pulisce la memoria alla fine del ciclo di sintesi e alla fine dell'applicazione;
CreateSDStereo:	sottostrumento di SalvoLab, crea un file di formato SDII stereo;
CanaleDestroLab:	sottostrumento di SalvoLab, salva i campioni prodotti sul canale destro del file formato SDII stereo;
CanaleSinistroLab:	sottostrumento di SalvoLab, salva i campioni prodotti sul canale sinistro del file formato SDII stereo;
EndWorkLab:	sottostrumento di SalvoLab, chiude la sessione di salvataggio dei campioni in un file di formato SDII stereo.

DocumentiCin contenente gli header, i codici, i progetti e gli eseguibili sviluppati in THINK C dei seguenti Code Interface Node:

Allocate:	alloca la memoria temporanea necessaria per il ricovero dei campioni prodotti;
Bank:	ricovera nella zona di memoria allocata i campioni durante la produzione;
Controller:	controlla il formato durante la lettura di un file SDI esterno;
Dispose:	pulisce la memoria alla fine di ogni esecuzione;
Normalize:	normalizza i campioni prodotti;
Resto:	preleva parte dei campioni prodotti per il calcolo dello spettro 2D;
Resto3D:	preleva i campioni prodotti dall'area di memoria allocata in porzione dell'ordine della trasformata per la realizzazione dello spettro 3D;
salvo:	salva i campioni prodotti in un file di formato SDI;
Sveglia:	legge un campione per volta da un file di cui si é specificato il reference number.

(esempio: Allocate.h Allocate.c Allocate.prj Allocate.r)

Spectre contenente gli header, i codici, i make, gli oggetti e gli eseguibili dei seguenti Code Interface Node sviluppati in MPW:

Brown:	parte del codice del generatore di rumore scuro;
ControllerSD2:	controlla il formato durante la lettura di un file SD2 esterno;
FCancella:	cancella i diagrammi generate esternamente a LabVIEW;
Fdraw3D:	realizza il diagramma dello spettro 3D;
Fdrawzoom:	realizza il diagramma del segnale;
Pink:	parte del codice del generatore di rumore rosa;
salvo2canale:	salva i campioni prodotti su un file di formato SDII stereo;
salvo2mono:	salva i campioni prodotti su un file di formato SDII mono;
salvo2stereo:	crea un file di formato SDII stereo;
Fzoomspetr:	realizza il diagramma dello spettro 2D;
Filtro:	realizza la equazione alle differenze;
Matrice:	alloca la memoria dedicata alla computazione dell'equazione alle differenze;
ClearMatH:	pulisce la memoria occupata da Matrice.

(esempio: Brown.h Brown.c Brown.make Brown.c.o Brown)

inoltre é presente anche la libreria CINLib.o e lo header LabTypes.h condiviso da alcuni codici.

Play contenente gli strumenti di LabVIEW:

Play2SA: permette l'ascolto del segnale ripetutamente;

OPlay2S: permette l'ascolto del segnale una sola volta.

Inoltre contiene gli header, i codici, i make, gli oggetti e gli eseguibili dei seguenti Code Interface Node sviluppati in MPW:

Play2;
OPlay2.

Per installare la libreria occorre inserire Supporto nella cartella contenente LabVIEW2.2, TimbreLab nella sottocartella VI.lib e mediante ResEdit aggiungere la risorsa SCD (ID=101) a LabVIEW2.2

2 Strutture dati

2.1 Strutture dati interne

Vengono classificate in base all'ambiente di appartenenza.

2.1.1 LabVIEW2.2

Sono elencate nel capitolo precedente tutte le variabili globali usate da TimbreLab. La maggior parte di esse sono situate nei due moduli principali `InitLab` e `StoreLab` e generalmente si intuisce la loro struttura e la loro funzione.

Intendiamo fare luce sulle seguenti:

Strutture dati per i lettori di file

gvRefs : array dinamico di interi a 16 bit contenente i reference number dei file aperti.

La struttura è inizializzata dal modulo controllore come un array contenente solo 0 di dimensione pari al parametro di ingresso `readers number`.

Durante il primo ciclo di calcolo le varie istanze del modulo lettore aprono i propri file e ricoverano nella struttura i relativi reference number.

Strutture per i generatori di rumore

gvLast: array dinamico di float a 32 bit contenente i risultati parziali.

La struttura è inizializzata dal modulo controllore come un array di dimensione pari al parametro di ingresso `noises number` contenente tutti 0.

Durante il calcolo del segnale le istanze dei generatori di rumore scuro e rosa memorizzeranno all'interno dell'array i risultati parziali da reimpiegare nella computazione.

Le strutture introdotte sono state concepite poiché tutte le istanze di uno strumento durante un processo condividono le medesime strutture dati, questo ha portato a gestire i parametri di ingresso e i risultati parziali con array.

L'identificatore di strumento è l'indice 'i' che determina la posizione dedicata alla singola istanza all'interno dell'array, perciò 'i' ha le caratteristiche seguenti:

- a) range (0 .. numero totale istanze - 1);
- b) due istanze non possono avere lo stesso identificatore.

Per quanto riguarda i parametri il discorso è analogo, a meno di non specificare tutti gli ingressi da diagramma come succede per gli strumenti a coefficienti variabili, l'unico modo che permette di introdurre i dati da pannello e di aver due istanze di uno stesso strumento con coefficienti diversi è quello di costruire un array di cluster con i parametri come campi.

L'ingresso `choice` altro non è che l'indice determinante il cluster scelto all'interno dell'array `Configurations`.

2.1.2 Code Interface Node

Hanno tutti in comune un'unica struttura di lavoro.

Nel modulo di `InitLab` si trova un sottostrumento chiamato `AllocateLab`, questo ricevendo in ingresso il numero di campioni da produrre alloca mediante un CIN una quantità di memoria utile a contenere una struttura così definita:

```
typedef struct {
    int32          DimSize;
    float32        arg1(numero di campioni)
}                spazio di memoria;
```

Fatto ciò restituisce all'esterno un intero a 32 bit contenente l'indirizzo del puntatore all'area di memoria riservata alla struttura.

`BankLab` (sottostrumento di `StoreLab`) è il responsabile della scrittura del campione corrente nel record, riceve in ingresso l'indirizzo al puntatore custodito in `gvhandle` e il risultato dell'i-esima computazione.

Al termine della fase di produzione del segnale, il CIN `Normalize` normalizza l'array dei campioni trasformandolo in interi a 16 bit e libera la memoria in eccesso senza cambiare l'indirizzo contenuto in `gvHandle`.

Da questo punto in poi il record è trasformato in:

```
typedef struct {
    int32          DimSize;
    int16          arg1(numero di campioni);
}                spazio di memoria;
```

Lavoreranno su di essa di altri CIN:

- a) Salvo che salverà i campioni in un file formato SDI;
- b) SalvoSDIImono che salverà i campioni in un file SDII mono;
- c) SalvoSDIIStero, SalvoCanaleDestro, SalvoCanaleSinistro che contribuiscono alla formazione di un file SDII stereo;
- d) Play e Oplay che inviano alla scheda sonora i campioni prodotti;
- e) FdrawZoom che produce il diagramma del segnale.

Esiste un'ulteriore zona di memoria allocata dal modulo controllore, quest'ultimo, ricevendo in ingresso il numero di filtri presenti nel diagramma totale, realizza due strutture C schematizzate come segue:

```
typedef struct {
    int32          dimSize;
    float32        arg1[];
}                TD1;

typedef TD1        **TD1Hdl;

typedef struct {
    int32          Nfiltri;
    TD1Hdl         tab[];
}                Matrice;

typedef Matrice    **MatHdl;
```

Durante la computazione sono coinvolte dallo strumento che realizza un filtro digitale:

$$\sum_{i=0}^K a_i y(n-i) = \sum_{j=0}^M b_j x(n-j)$$

Nella prima struttura sono memorizzate le (K+1)-uple (y(n) .. y(n-K)) di tutte le istanze mentre nella seconda le (M+1)-uple (x(n) .. x(n-M)).

Ogni istanza di filtro accede alla zona di calcolo privata mediante l'identificatore di strumento 'i', infatti gli handle contenuti nei campi `tab[i]` permettono ad ogni istanza di recuperare durante il calcolo del singolo campione i valori precedentemente memorizzati.

2.2 Strutture esterne

TimbreLab permette di eseguire il salvataggio del segnale prodotto in diversi formati.

2.2.1 SDI

Il formato Sound Designer I è il primo formato sonoro originale Digidesign realizzato nel 1985. E' largamente usato e supportato, per esempio molti dischi CD ROM hanno effetti sonori immagazzinati in questo formato.

E' principalmente usato per memorizzare campioni a 16 bit di suoni brevi.

FileType: 'SFIL'

Resource Fork: Non vengono usate risorse

Data Fork: i primi 1336 bytes del data fork contengono lo header del segnale seguito dai suoi campioni.

La struttura dello header é riportata nell'Appendice B

2.2.2 SDII

Sound Designer II immagazzina tutti i campioni nel data fork e tutti i parametri del suono nel resource fork. Questo è estremamente conveniente poiché è possibile modificare i parametri senza chiamare in causa il data fork che può essere di dimensioni notevoli.

File Type: 'Sd2f'

Resource fork: sono utilizzati diversi tipi di risorse che specificano dalla frequenza di campionamento alle caratteristiche del diagramma.

Molte sono usate esclusivamente da Sound Design II.

Le più importanti sono le seguenti:

Type: 'STR ' **ID:** 1000 **Name:** 'sample-size'

Contenuto: Stringa che specifica il numero di byte per campione.

Type: 'STR ' **ID:** 1001 **Name:** 'sample-rate'

Contenuto: Stringa che specifica la frequenza di campionamento.

Type: 'STR ' **ID:** 1002 **Name:** 'channels'

Contenuto: Stringa che specifica il numero di canali.

Solo questi tre parametri Sound Designer II è in grado di leggere, rappresentare e convertire in analogico il segnale digitale contenuto nel file.

Nell'ultima risorsa la stringa '1' da come risultato file mono mentre '2' identifica file stereo. Le risorse rimanenti sono illustrate nell'Appendice B.

Data Fork: il primo byte del data fork è il primo byte del primo campione.

I dati sono organizzati alternando (se ci sono più canali) i campioni in questo modo:

```
Left  channel sample 1
Right channel sample 1
Left  channel sample 2
Right channel sample 2
Left  channel sample 3
Right channel sample 3
```

e così via.

In caso di più tracce è consigliabile salvarle individualmente su file mono.

3 Struttura dello strumento

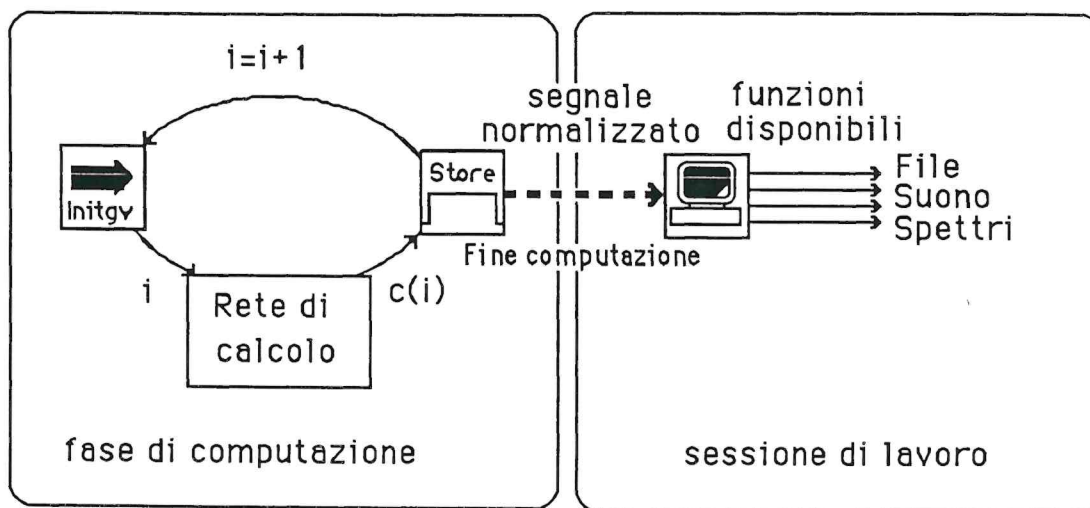
TimbreLab permette di costruire un insieme di applicazioni in grado di produrre segnali digitali (*strumenti*); ogni diagramma segue un comune schema di realizzazione articolato su tre funzioni:

Temporizzatore: formato da *InitLab*, responsabile della sincronizzazione degli strumenti e delle caratteristiche del segnale prodotto.

Magazzino: cui appartiene il modulo di *StoreLab* in grado di memorizzare il campione prodotto dalla rete di calcolo.

Rete di calcolo: che raggruppa tutti i sottostrumenti che elaborano i segnali.

I moduli principali colloquiano come rappresentato in figura:



3.1 Fase di computazione

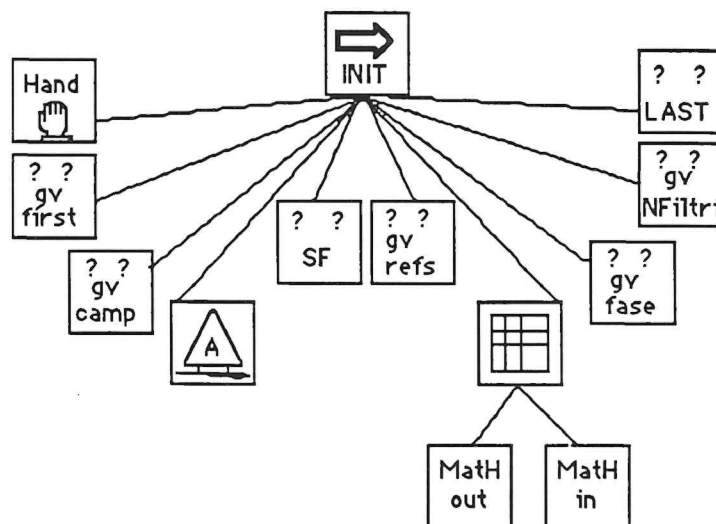
Si suddivide in:

- a) inizializzazione delle variabili globali;
- b) ciclo di computazione.

Una caratteristica peculiare di LabVIEW è quella di non imporre un ordine di esecuzione dei vari moduli software a meno di una esplicita indicazione del programmatore

InitLab ha il compito di gestire la sincronizzazione degli strumenti ed attivarli nel momento opportuno.

Mostriamo qui sotto l'albero relativo agli strumenti e moduli richiamati direttamente da InitLab.



All'interno di InitLab risiedono le variabili globali della applicazione, la zona dove esse sono situate è controllata da un semaforo realizzato con una variabile booleana di nome `isFirstTime` con valore di default pari a FALSE.

Le variabili globali che regolano il funzionamento totale del processo sono le seguenti:

?? gv camp	<code>gvNumeroCampioni:</code>	numero di campioni del segnale;
?? SF	<code>gvsampligfrequency:</code>	frequenza di campionamento del segnale;
?? gv fase	<code>gvfase:</code>	campione corrente;
Hand [Icona di una mano]	<code>gvhandle:</code>	indirizzo della zona di memoria dove ricoverare i campioni prodotti;



gvrefs:

array contenente i reference number dei file aperti da ReaderLab e kReaderLab;



gvIsFirstTime:

semaforo della zona di inizializzazione delle variabili globali;



gvlast:

struttura contenete i risultati parziali prodotti dai generatori di rumore;



gvMatH_in:

indirizzo di memoria della zona di calcolo dedicata ai filtri;



gvMatH_out:

indirizzo di memoria della zona di calcolo dedicata ai filtri;



gvNFilteri:

numero di filtri.

Oltre a queste esistono due ulteriori sottostrumenti virtuali di `InitLab` che vengono coinvolti durante la fase di inizializzazione di cui specifichiamo il nome e l'utilizzo:



AllocateLab:

sottostrumento di `InitLab`, alloca una porzione di RAM utilizzata come area di parcheggio per il segnale il cui valore è memorizzato in `gvHandle`;



MatriceLab:

sottostrumento di `InitLab`, prepara le strutture utilizzate dai filtri i quali indirizzi di memoria sono memorizzati in `gvMatH_in` e `gvMatH_out`.

Dopo l'inizializzazione di tutte le variabili globali e di tutte le strutture dati il valore del semaforo diventa `TRUE` vietando all'applicazione di accedere alla zona fino al termine dell'esecuzione.

Dopo di che è eseguita la computazione del segnale digitale, durante la quale `InitLab` viene dirottato dal semaforo nel ramo `ELSE` dell'algoritmo, dove legge l'indice del campione corrente (memorizzato in `gvfase`) e lo spedisce alla rete di calcolo, che lo elabora e restituisce al modulo di `StoreLab`, il quale lo ricovera in una zona privata di memoria e incrementa l'indice del campione corrente, che verrà letto da `InitLab` e così via fino ad esaurimento dei campioni da produrre.

La fase descritta può essere così espressa in pseudocodice:

```
i=0
WHILE i < numero_campioni
  Begin
    InitLab spedisce i alla rete di calcolo
    Rete di calcolo restituisce c(i)
    StoreLab ricovera c(i) in spazio, incrementa i
  End
```

3.2 La sessione di lavoro

La quale si suddivide in :

- a) fase di normalizzazione;
- b) rappresentazione del segnale;
- c) consumo del segnale.

Al termine della computazione del segnale il controllo del processo è nelle mani `StoreLab` che fino a questo istante si è limitato a memorizzare i campioni che saranno utilizzati come:

- 1) input per la scheda sonora;
- 2) rappresentati con il modulo di visualizzazione;
- 3) memorizzati su disco.

Il formato richiesto dalle precedenti operazioni è intero a 16 bit e non il formato float a 32 bit usato durante il ricovero.

I campioni perciò vengono convertiti con il seguente algoritmo:

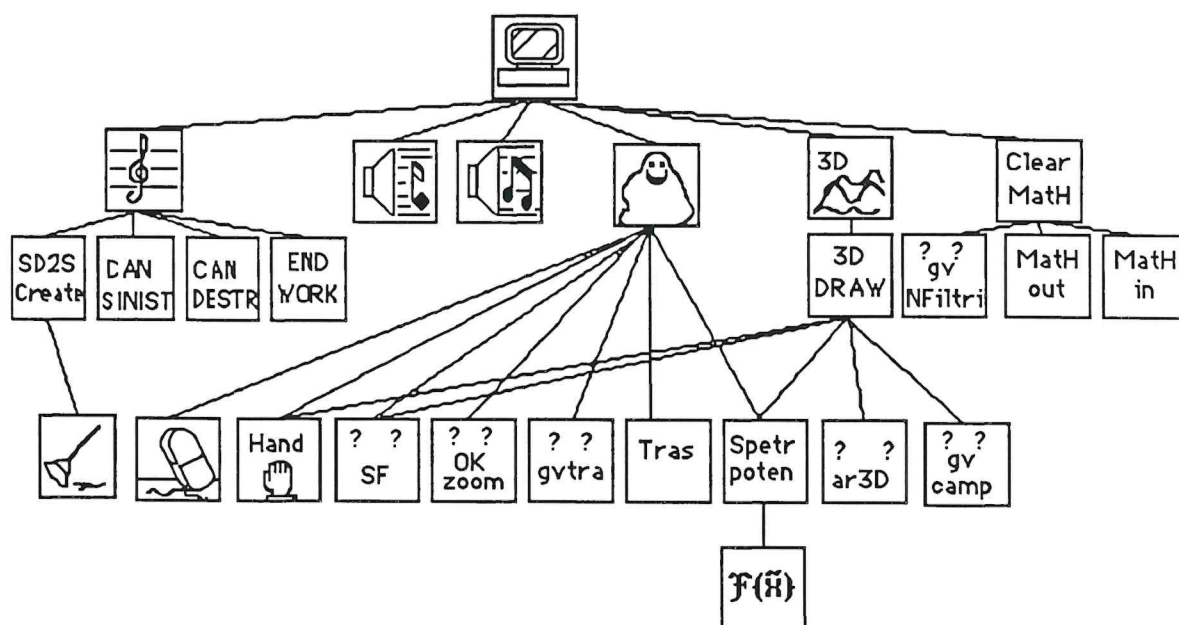
$$\text{campione}_{\text{int16}}(i) = (\text{int16})(\text{campione}_{\text{float32}}(i) \times 32767) : \text{maxcamp}$$

dove `maxcamp` è il valore massimo assoluto assunto dai campioni.

Il range dei campioni normalizzati è $(-32767, 32767)$, in questo modo la perdita di informazione dovuta al passaggio dal formato a 32 bit a quello a 16 bit viene minimizzata.

Dopo di che il segnale è pronto per essere rappresentato e utilizzato.

Tutto è svolto all'interno di InterfaceLab che costituisce il modulo principale di StoreLab e quello più complesso realizzato durante lo sviluppo di TimbreLab, di cui riportiamo lo schema ad albero.



InterfaceLab è formato da sei sottostrumenti virtuali principali che implementano le funzioni messe a disposizione dalla applicazione:



Spect3D Lab: realizza lo spettro 3D;



Spect2D Lab: realizza lo spettro 2D;



Clear Math Lab: sottostrumento di pulizia della memoria;



Salvo Lab: salva su disco il segnale prodotto;



Play2SA:

permette l'ascolto del segnale ripetutamente;



OPlay2S:

permette l'ascolto del segnale una sola volta.

Ogni VI funziona grazie ad un sistema privato di semafori implementati con variabili globali, il quale impone la giusta successione di eventi all'interno della singola funzione, ad esempio `SalvoLab` è formato dai seguenti quattro sottostrumenti:



CreateSDStereo:

crea un file di formato SDII stereo;



CanaleDestroLab:

salva i campioni prodotti sul canale destro;



CanaleSinistroLab:

salva i campioni sul canale sinistro;



EndWorkLab:

chiude le sessione di salvataggio.

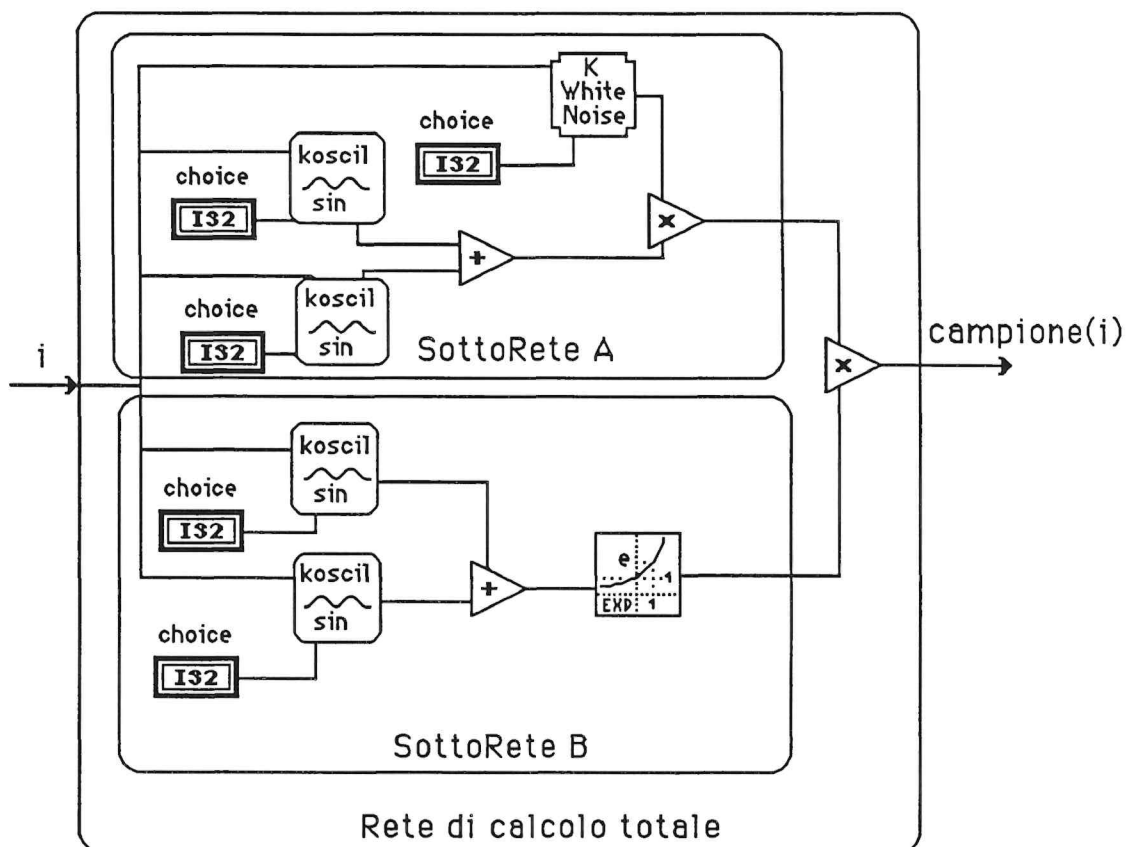
La corretta successione dei precedenti è necessaria affinché l'esecuzione giunga al termine (palesamente: non si può salvare prima di creare il file), questo è assicurato dalla presenza del sistema di semafori costituito da `gvSemCanS`, `gvSemCanD`, `gvSemStereo`

La seguente struttura espressa in pseudocodice descrive il funzionamento generale di `InterfaceLab`:

```
WHILE continue or close = false
  Begin
    if salvo = true      then salva nel formato scelto
    if once = true       then suona il segnale
    if loop = true       then suona finché il tasto è
                              premuto
    if spectr2D = true   then rappresenta lo spettro 2D
    if spectr3D = true   then rappresenta lo spettro 3D
    if continue = true   then libera la memoria, torna a
                              InitLab
    if close = true      then libera la memoria, fine
  End
```

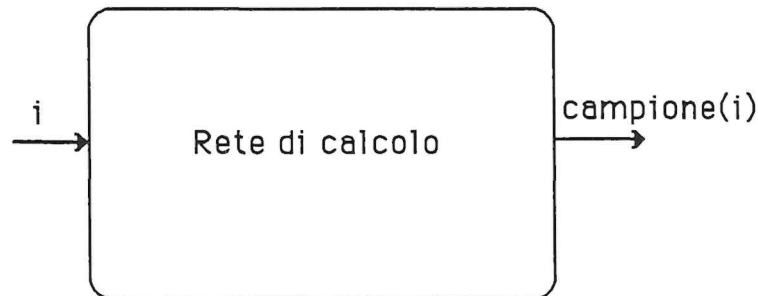
3.3 La rete di calcolo

La struttura della rete di calcolo è analoga a quella di una rete logica: i fili che collegano gli strumenti rappresentano le connessioni e gli strumenti sono i blocchi logici, inoltre entrambe elaborano informazioni rappresentate da segnali digitali.



Entrambe trasformano un segnale in ingresso in un segnale di uscita mantenendo un comportamento unidirezionale: ovvero è impossibile alterare l'informazione in ingresso agendo sulla porta d'uscita.

E' possibile comporre reti di calcolo o individuare reti di calcolo più semplici interconnesse fra loro, a patto che venga rispettato il seguente criterio:



Un solo ingresso per il temporizzatore una sola uscita per il magazzino. Esiste una differenza basilare tra le due reti in quanto è possibile classificare ogni rete logica come rete sincrona o come rete asincrona. Nelle reti asincrone viene rappresentata la funzione in forma ridondante al fine di evitare il fenomeno del transitorio.

A differenza della rete logica, la rete di calcolo non conosce il fenomeno del transitorio, infatti ogni strumento, prima di dar luogo alla computazione, attende che ogni ingresso sia pronto.

Questo aspetto, tipico di LabVIEW, permette di evitare l'uso di semafori per il controllo della sincronizzazione degli strumenti e di dover rispettare alcun vincolo sulla forma della funzione, ma fa della rete di calcolo uno strumento lento.

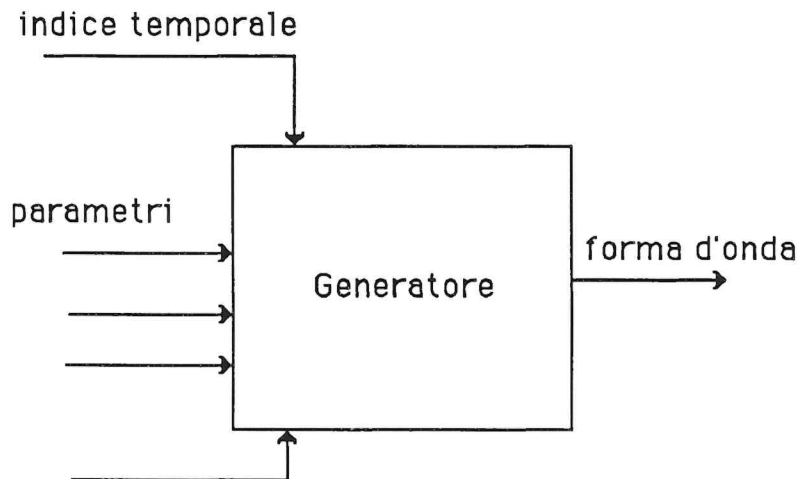
Fanno parte della rete di calcolo tutti i sottostrumenti a carattere funzionale.

Tali sottostrumenti dividono in due categorie:

- a)Generatori di onde;
- b)Operandi.

3.3.1 Generatori

I generatori sono schematizzati come segue:



L' indice temporale, un intero a 32 bit, proviene dal temporizzatore mentre il campione in uscita è un extended a 96 bit, come il resto degli strumenti di LabVIEW.

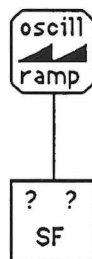
Le funzioni implementate dai generatori sono :

- a) sinusoidali;
- b) triangolari;
- c) rampa;
- d) rettangolari;
- e) impulso;
- f) rumore;

Esiste in aggiunta la possibilità di determinare la funzione del generatore in modo arbitrario: questo è possibile "disegnando" una funzione per punti oppure specificando il file contenente i valori della funzione desiderata.

Ogni generatore dialoga con il modulo di inizializzazione mediante le variabili globali che contengono i dati caratteristici della forma d' onda da computare.

Vediamo come esempio l'albero di moduli e sottostrumenti chiamati direttamente dall'oscillatore di onde sinusoidali:



Tale generatore interroga la variabile globale contenente la frequenza di campionamento necessaria durante il calcolo del campione corrente, tale prassi evita la comunicazione tra moduli mediante i fili che rende il diagramma disordinato e poco comprensibile, ma impone ogni volta la chiamata ad una subroutine rallentando così l'esecuzione.

Data la natura dell'indice temporale (che come indice non ha unità di misura) i legami con lo spazio dei tempi sono rappresentati dalla frequenza di campionamento e dal numero di campioni del segnale. Infatti:

$$\text{Intervallo di campionamento} = \frac{1}{\text{frequenza di campionamento}}$$

$$\text{Tempo}(i) = \frac{i}{\text{frequenza di campionamento}}$$

$$\text{Durata del segnale} = \frac{\text{N}^\circ \text{campioni}}{\text{frequenza di campionamento}}$$

possono essere facilmente ricavate mediante le due grandezze citate.

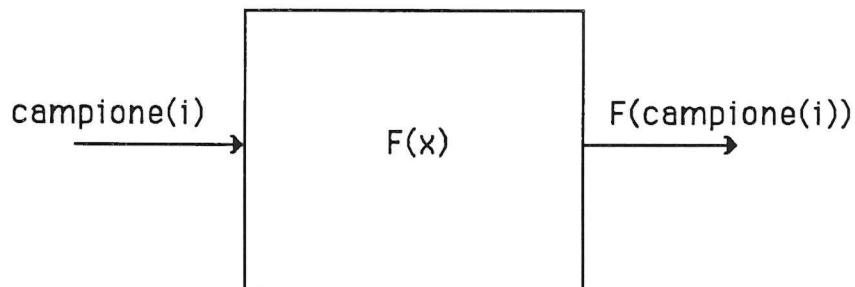
3.3.2 Operandi

Tutti gli strumenti di LabVIEW agenti su extended a 96 bit possono essere impiegati nella rete di calcolo.

La libreria matematica di LabVIEW comprende:

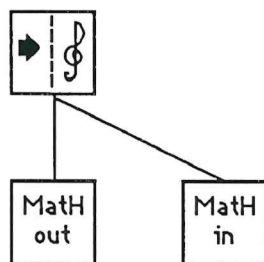
- a) operazioni aritmetiche;
- b) funzioni trigonometriche esponenziali logaritmiche;
- c) costanti.

Sono stati sviluppati in aggiunta alcuni strumenti ad hoc così schematizzati:



dove $\text{campione}(i)$ proviene dai generatori e con $F(\text{campione}(i))$ è un extended a 96 bit.

Fanno parte di questa categoria i filtri seriali, gli interruttori e i generatori delle linee di ritardo, anch'essi dialogano con il modulo di inizializzazione mediante le variabili, riportiamo come esempio l'albero di moduli e sottostrumenti del filtro generale.



Tale modulo necessita degli indirizzi delle zone di memoria riservate come aree di calcolo, queste sono state inizializzate da InitLab che, in un secondo tempo, memorizza l'indirizzo nelle due variabili globali.

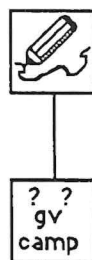
Durante lo sviluppo dello strumento quindi è possibile impiegare tutti gli strumenti messi a disposizione ricordando che nessuno strumento può ricevere o restituire array, durante l'esecuzione, infatti ogni strumento della rete di calcolo elabora un campione per volta

3.4 Struttura di alcuni moduli implementati

3.4.1 DrawerLab

Il primo modulo sul quale intendiamo far luce è il modulo lettore di grafici, esso permette di introdurre nel modello timbrico funzioni definite per punti totalmente arbitrarie.

La figura seguente illustra l'albero dei moduli ad esso associato:



Solitamente funzioni di questo genere vengono impiegate durante la determinazione di involuppi.

Le modalità di utilizzo differiscono da ogni altro VI appartenente al sistema ad architettura aperta: i punti costituenti le funzioni vengono determinati disegnando con opportuni tool sul pannello di DrawerLab.

LabVIEW mette a disposizione un controllo di tipo GRAPH, esso oltre a permettere la definizione di una generica spezzata con un qualsiasi numero di punti, mette a disposizione due cursori per il controllo dei valori, la possibilità di scroll sia orizzontale che verticale.

LabVIEW interpreta i dati introdotti mediante il controllo GRAPH come un array di cluster, dove ogni cluster è formato da una coppia di valori rappresentanti le coordinate del singolo punto.

DrawerLab ha un singolo sottostrumento:



gvNumeroCampioni: numero di campioni del segnale.

Durante il calcolo del valore corrente, il contenuto della variabile globale viene utilizzato per la determinazione della distanza fra i punti all'interno del segnale.

Ad esempio:

Lunghezza segnale: 1000 campioni

Numero di punti della spezzata: 10

Intervallo fra due punti adiacenti: 100 campioni

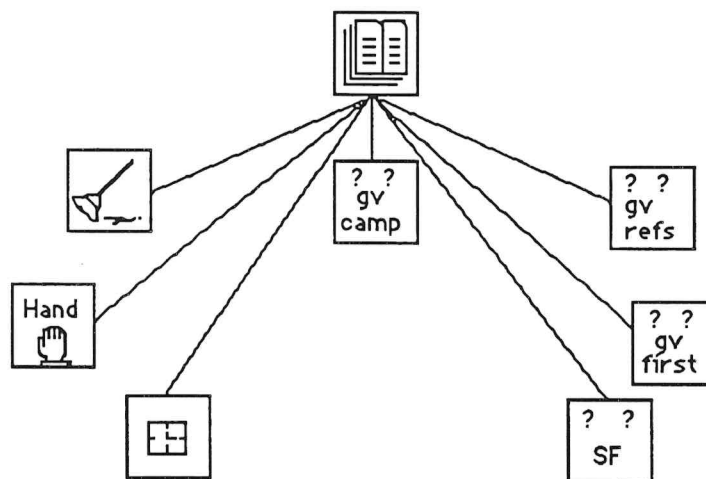
Il lettore di grafici è così in grado di calcolare la funzione formata da tanti segmenti di retta con lunghezza pari a 100 campioni.

3.8.2 ReaderLab

Appartiene alla classe dei generatori, teoricamente scandisce i valori contenuti in una tabella sincronicamente rispetto ai rimanenti moduli del modello timbrico.

Esso permette di introdurre nella computazione funzioni con valori arbitrari senza specificarne la rappresentazione matematica.

Seguirà la struttura ad albero con i dovuti commenti:



ReaderLab è costituito da sette sottostrumenti virtuali che collaborano durante il calcolo del campione corrente.

L'algoritmo realizzato è semplicissimo e strutturato in due fasi:

a) *fase di controllo*: durante la quale sono eseguiti determinati controlli sul formato del file);

b) *fase di calcolo*: finchè la lunghezza del file è minore al numero di campioni da produrre, il campione letto viene moltiplicato per una costante di volume e spedito in uscita.

Illustriamo come i sottostrumenti di ReaderLab partecipano alla computazione:



SvegliaLab:

sottostrumento destinato alla lettura del campione corrente;



DisposeLab:

strumento di pulitura della memoria, viene richiamato da ReaderLab ogni qual volta si presentano delle situazioni d'errore, dopo di che interrompe l'esecuzione;



gvIsFirstTime:

semaforo della zona di inizializzazione delle variabili globali, viene riportato al valore di TRUE prima di uscire dall'applicazione;



gvNumeroCampioni:

numero di campioni del segnale;



gvsamplefrequency:

frequenza di campionamento del segnale, utilizzata durante la fase di controllo;



gvhandle:

indirizzo della zona di memoria dove ricoverare i campioni prodotti, esso costituisce l'input per lo strumento di pulitura;



gvrefs:

array contenente i reference number dei file aperti da ReaderLab e kReaderLab, alla fine della fase di controllo, stabilita

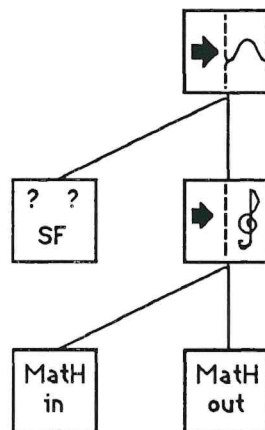
l'idoneità del formato, il reference number è memorizzato nella struttura in modo che possa essere recuperato nel prossimo ciclo di calcolo.

3.8.3 BandPassLab

Tutti i filtri sono stati implementati con una comune strategia di programmazione.

Un filtro digitale è definito mediante un'equazione alle differenze, la quale viene realizzata da `GeneralFilterLab`, che, oltre ad essere un modulo operando indipendente, costituisce il sottostrumento principale di tutti i filtri presenti nel sistema ad architettura aperta.

L'albero di moduli sottostante relativo a `BanPassLab` è formato da quattro sottostrumenti:



Oltre al filtro generale sono presenti le variabili globali `gvMath_in` e `gvMath_out`, che ricoverano l'indirizzo della zona di memoria di calcolo dedicata al filtro, e `gvsamplingfrequency`, il quale contiene il valore della frequenza di campionamento necessaria durante la computazione del campione corrente.

4 Algoritmi

4.1 Filtri

Un filtro digitale è definito dall'equazione alle differenze:

$$\sum_{i=0}^K a_i y(n-i) = \sum_{j=0}^M b_j x(n-j)$$

dove $x(n)$ e $y(n)$ sono l'ingresso e l'uscita del filtro al tempo n , $(a_0..a_{n-K})$ e $(b_0..b_{n-M})$ sono i coefficienti d'ingresso e di uscita.

Nel filtro generale l'utente determina il grado e tutti i coefficienti di volta in volta necessari mentre nel filtro passabanda viene implementata la seguente relazione:

$$y(n) = G(x(n) - Rx(n-2)) + b_1 y(n-1) + b_2 y(n-2)$$

dove:

$$R = e^{-\pi \frac{B}{S}}$$

$$G = 1 - R$$

$$b_1 = 2R \cos\left[2\pi \frac{f}{S}\right]$$

$$b_2 = -R^2$$

f = frequenza centrale

S = frequenza di campionamento

B = larghezza di banda

4.2 Rumori

Sono realizzati tre tipi di generatori di rumore, dei quali riportiamo l'algoritmo in linguaggio C e opportuni commenti.

Rumore Bianco

Viene definito come un processo stocastico con le frequenze risultanti non correlate tra loro (statisticamente indipendenti).

La funzione `Random(-a,a)` restituisce un extended a 96 bit con funzione di distribuzione uniforme con estremi $(-a,a)$.

```
For (i=0; i<lunghezza del segnale; i++)
{
    campione(i)=Random(-1,1);
}
```

Rumore Rosa

E' definito come un processo stocastico caratterizzato da una densità spettrale del tipo $1:f$.

```
lunghezza del segnale=2N;
range=1/N;
last=lunghezza del segnale -1;
For(n=0 ; n < lunghezza del segnale; n++ )
{
  For(R=i=0 ; i < N ; i++ )
  {
    If( (( 1<< i ) & n) != (( 1<< i ) & last) )
      r(i)=Random(-range,range);
    R=r(i)
  }
  campione(n)=R;
  last=R;
}
```

Rumore Scuro

Processo stocastico caratterizzato da una densità spettrale del tipo $1:f^2$.

```
R=0;
Rlast=0;
For(i=0 ; i< lunghezza del segnale ; i++ )
{
  Do
  {
    R=Rlast + Random(-1,1)
  }
  While(R > 1 || R < -1);
  campione(i)=R;
  Rlast=R;
}
```

Il rumore bianco ha il maggior comportamento random e mostra cambiamenti rapidi e non correlati. Il rumore scuro è il più correlato fra quelli implementati e presenta cambiamenti lenti. Il rumore rosa identifica un situazione intermedia

4.3 Grafici

Per ragioni di lentezza e di occupazione di memoria si è stabilito di realizzare i diagrammi dell'applicazione prodotta da TimbreLab mediante Code Interface Node.

Malgrado alcune differenze dovute al tipo funzione da rappresentare, Fdraw3D, Fdrawzoom e Fzoomspetr implementano il seguente algoritmo semplificato.

Ogni segnale è formato da punti, principalmente si calcola il valore massimo assoluto e con esso il valore dY rappresentato da ogni pixel, dopo di che mediante le coordinate della finestra contenente il grafico si calcola il numero di punti per colonna.

```
if( due o più punti per colonna di pixel )
{
  alloca un array dove memorizzare i punti della colonna
  for(i=0;i<=colonne da disegnare;i++)
  {
    carica nell'array i punti dell'i-esima colonna
    trova il valore max e min dell'array
    stabilisci le due ordinate dei punti estremi della
    linea verticale
    if( se non è la prima linea vert. che traccio )
    {
      se la corrente linea è interamente sopra in
      senso verticale a quella precedente, devo
      congiungere il punto più alto della linea
      precedente a quello più basso della linea
      attuale
      altrimenti se la corrente linea è tutta più
      bassa di quella precedente, eseguo il
      raccordo dal pto più basso della linea
      precedente a quello più alto della linea
      attuale
      memorizzo Ymin e Ymax
      traccio la linea attuale dopo aver effettuato
      i raccordi
    }
  }
}
```

```
else ( ho meno di 2 punti per colonna di pixel)
{
  carica tutta l'onda in un buffer
  determina le coordinate di ogni punto, mi muovo nelle
  vecchie coordinate traccio una linea fino alle nuove
  coordinate
  memorizzo coordinate X, Y
}
```

L'algoritmo adottato presenta numerosi vantaggi: esiste un'assoluta indipendenza dalle coordinate della finestra, è veloce e non occupa memoria.

Appendice A

Strumenti matematici utilizzati dai generatori di TimbreLab:

$$\text{involuppo sinusoidale}(i) = \sin\left(2\pi \frac{f}{F} i + \text{phase}\right) \quad F = \text{sample rate}$$

$$\text{involuppo triangolare}(i) = \frac{(1 - ((i \text{ dt}) - \text{delay}))}{\text{width}} \text{amplitude}$$

$$\text{involuppo rampa}(i) = \frac{(\text{final value} - \text{init value})}{(\text{sample number} - 1)} i + \text{init value}$$

$$\text{involuppo impulso}(i) = \begin{cases} \text{amplitude} & i = \text{index} \\ 0 & \text{altrimenti} \end{cases}$$

$$\text{involuppo rettangolare}(i) = \begin{cases} \text{amplitude} & \text{delay} \leq i < \text{delay} + \text{width} \\ 0 & \text{altrimenti} \end{cases}$$

Si definisca la funzione :

$$\text{Mod}_n^0 x : \mathbf{N} \times \mathbf{R} \rightarrow \mathbf{R}$$

che associa a (n, r) un numero reale r' tale che:

$$r' = \text{Mod}_n [\text{int}(r)] + \text{dec}(r)$$

dove $\text{int}(r)$ e $\text{dec}(r)$ indicano rispettivamente la parte intera e la parte decimale del numero reale r , mentre Mod_n indica l'operazione di modulo definita per gli interi.

$$\text{oscillatore sinusoidale}(i) = \sin\left(2\pi \frac{f}{F} i + \text{phase}\right) \quad F = \text{sample rate}$$

$$\text{oscillatore triangolare (i)} = \frac{\left(1 - \left(\left| \text{Mod}_i^{\partial} \left(\frac{F}{f} \right) \right| dt\right) - \text{delay} \right)}{\text{width}} \text{amplitude}$$

$$\text{oscillatore rampa (i)} = \frac{(\text{final value} - \text{init value})}{(\text{samples number} - 1)} \text{Mod}_i^{\partial} \left(\frac{F}{f} \right) + \text{init value}$$

$$\text{oscillatore impulso (i)} = \begin{cases} \text{amplitude} & \text{Mod}_i^{\partial} \left(\frac{F}{f} \right) = \text{index} \\ 0 & \text{altrimenti} \end{cases}$$

$$\text{oscillatore rettangolare (i)} = \begin{cases} \text{amplitude} & \text{delay} \leq \text{Mod}_i^{\partial} \left(\frac{F}{f} \right) < \text{delay} + \text{width} \\ 0 & \text{altrimenti} \end{cases}$$

Appendice B

Vengono riportate le strutture dati impiegate nei formati prodotti da TimbreLab.

Formato Sound Designer I

I primi 1336 bytes del data fork contengono l'header del segnale che può essere inteso come la seguente struttura C :

```
typedef struct {
    Boolean      Free;   (True se il Marker é libero)
    int32        Position; (posizione nel file)
    Str32        Name;   (nome del Marker)
    }           MarkerType;   (tipo di Marker)

typedef enum    { leftSide=0, rightSide }      SideType;

typedef struct {
    int32        HiAddr;
    int32        LoAddr;
    SideType     ExtendSide;
    }           EditRecord;

typedef enum    {
    Chrono=0,
    SampleNumber,
    HexSampNum,
    Volts,
    Percent,
    dbm,
    User
    }           ScaleNames;

typedef enum    {
    Select=0,
    Draw,
    ZoomSelect
    }           ModeType;

typedef struct {
    int16        v;           (Scala verticale)
    int32        h;           (Scala orizzontale)
    }           ZoomType;     (tipo di zoom)
```

```
typedef struct {
    int32          VFactor;
    ScaleNames     VType;
    Str32          VString;
    int32          HFactor;
    ScaleNames     HType;
    Str32          HString;
    }              ScaleType;  (Unità di misura sugli assi)
```

```
typedef struct {
    int16          HeaderSize;      (N° bytes
                                     dell'header)

    int16          Version;
    Boolean        Preview;
    WindowPtr      WPtr;
    WindowPeek     WPeek;
    int32          HInxPage,HInxLine;
    int32          VInxPage,VInxLine;
    int32          HCtlPage,HCtlLine;
    int32          VCtlPage,VCtlLine;
    int32          VOffset,HOffset;
    int16          VOffConst;
    ZoomType       ZF;
    ScaleType       SF;
    int16          VScrUpdate;
    int32          *BufPtr;
    Size           BufBytes;
    int32          BufOffset;
    RgnHandle       WaveRgn,ClipArea,ScaleArea;
    Rect           CtlWidth;
    ControlHandle   VScroll,HScroll;
    Size           FileSize;        (N° bytes del
                                     segnale)

    Str64          BUName;
    Str64          FileName; (Nome del file)
    int16          BURefNum;
    int16          refNum;   (reference nuber del
                             file)

    int16          vRefNum;
    Boolean        BufChanged;
    Boolean        FileChanged;
    Boolean        NoBackup;
    ModeType       Mode;
    EditRecord     Edit;
    int32          CursorPos;
    RgnHandle       CursorRgn;
```

```

MarkerType      MarkerData[10];(lista di
                  marker)

int32            MarkerOffset;
int32            LoopStart;
int32            LoopEnd;
Boolean          ZeroLineOn;
Boolean          CursorOn;
Boolean          ScalesOn;
Str255           Comment;
int32            SampRate; (freq di
                           campionamento)

int32            SampPeriod;
int16            SampSize; (N° di bytes del
                           campione)

Str32            CodeType;
Str255           UserStr1;
Size            BufSize;
int32            Loop2Start;
int32            Loop2End;
SignedByte       Loop1Type;
SignedByte       Loop2Type;
int16            User4;

                } WaveRec;

```

La maggior parte dei parametri indicati sono usati esclusivamente dalla applicazione Sound Designer II specialmente quelli riguardanti la rappresentazione del grafico del segnale.

Formato Sound Designer II

Le seguenti risorse memorizzano informazioni usate da Sound Designer II durante le applicazioni avanzate:

Type: 'sdDD' **ID:** 1000

Uso: Memorizza informazioni generali: unità di misura sugli assi, tipo di zoom, offset durante il runtime, versione del file.

Struttura: la seguente struttura C

```

typedef enum      { leftSide=0,rightSide } SideType;

typedef struct    {
    int32          HiAddr;
    int32          LoAddr;
    SideType       ExtendSide;
} EditRecord;

```

```

typedef enum    {
    Chrono=0,
    SampleNumber,
    HexSampNum,
    Volts,
    Percent,
    dbm,
    User
    }      ScaleNames;

typedef enum    {Select=0,Draw,ZoomSelect} ModeType;

typedef struct {
    int16          v;
    int32          h;
    }      ZoomType;

typedef struct {
    int32          VFactor;
    ScaleNames     VType;
    Str32          VString;
    int32          HFactor;
    ScaleNames     HType;
    Str32          HString;
    }      ScaleType;

typedef struct {
    int16          Version;
    int32          BufOffset;
    int32          MarkerOffset;
    Str255         Comment;
    int16          HDPlayBufMultiple;
    int32          SMPTEStartTime;
    int16          FramePerSec;
    int16          FilmSize;
    int32          StartBarNum;
    int32          StartBeatNum;
    int32          StartFrame;
    Fixed          Tempo;
    Fixed          TimeSignature;
    int32          CursorPos;
    Boolean        CursOn;
    ZoomType       Zoom;
    ScaleType       Scale;
    EditRecord     Edit;
    int32          VOffset;
    int32          HOffset;

```

```

Rect                CtlWidth;
Point               ZoomIndex;
int16               SelectMode;
} DocumentDataRecord;

```

Type: 'sdML' **ID:** 1000

Uso: Memorizza una lista Marker

Struttura: la seguente struttura C

```

typedef struct {
    int16                Markertype;
    int32                Position;
    int32                Text;
    int16                CursorID;
    int16                MarkerID;
    int32                TextLength;
} Marker;

typedef struct {
    int16                Version;
    int32                MarkerOffset;
    int16                NumMarkers;
    Marker                list[NumMarkers];
} MarkersRecord;

```

Type: 'sdLL' **ID:** 1000

Uso: Memorizza una lista di cicli audio

Struttura: la seguente struttura C

```

typedef struct {
    int32                LoopStart;
    int32                LoopEnd;
    int16                LoopIndex;
    int16                LoopSense;
    int16                Channel;
} Loop;

```

```

typedef struct {
    int16      Version;
    int16      HScale;
    int16      VScale;
    int16      NumLoops;
    Loop       array[NumLoops];
} LoopsRecord;

```

Le due risorse precedenti ricoverano informazioni private dell'applicazione Sound Designer II.

