Consiglio Nazionale delle Ricerche

ISTITUTO DI
ELABORAZIONE DELLA
INFORMAZIONE

Pisa

MULTIC25, un sistema multi-DSP con schede LeonardC25

G. Bertini, D. Fabbri

Nota Interna B4 - 14
Maggio 1993

PROGETTO FINALIZZATO

SISTEMI INFORMATICI E CALCOLO PARALLELO

SOTTOPROGETTO 2

Coordinatore Prof. Franco Denoth
Processori dedicati

G.Bertini[1], D.Fabbri[2]

MULTIC25, UN SISTEMA MULTI-DSP CON SCHEDE LEONARDC25.

N. R/ 2 / 108 **Aprile 1993**

Rapporto Interno

1- Istituto di Elaborazione dell'Informazione Consiglio Nazionale delle Ricerche, Via Santa Maria 46, 56126 Pisa

2- Collaboratore esterno IEI/CNR

INDICE

Ringraziamenti

Si desidera ringraziare M. Marani e A. Di Bari (collaboratori esterni, Studio di progettazione SEED, Massa) per il contributo dato allo sviluppo del progetto e ai test del sistema; A. Ribolini e C. Ori per il supporto generale; A. Landucci, M. Moretto e C.A. Giorgi per la realizzazione di prototipi hardware.

MULTIC25, UN SISTEMA MULTI-DSP CON SCHEDE LEONARDC25.

G.Bertini[1], D.Fabbri[2]

1- Istituto di Elaborazione dell'Informazione Consiglio Nazionale delle Ricerche, Via Santa Maria 46, 56126 Pisa

2- Collaboratore esterno IEI/CNR

Sommario

Nel lavoro viene descritto il progetto e la realizzazione di un prototipo del sistema multi-processore MULTIC25 per l'elaborazione e la sintesi di segnali in banda acustica.

Il sistema è basato su un Personal Computer PC IBM compatibile e alcune schede DSP Leonard C25 (fino ad un massimo di 8). Tali schede, sviluppate in collaborazione con il Centro Leonardo S.p.a. (Massa) e descritte in [1][2], utilizzano il microprocessore TMS320C25 e permettono l'implementazione di algoritmi per il DSP in tempo reale per applicazioni nei settori biomedico, industriale e audio digitale. Inoltre consentono l'assemblaggio di apparati multi-DSP modulari, utilizzabili per l'esecuzione parallela di determinate classi di algoritmi (come quelli di sintesi e di elaborazione di segnali audio musicali con elevato grado di polifonia e complessità timbrica). I moduli DSP sono connessi tramite la loro porta seriale; inoltre il primo e l'ultimo sono connessi ad un modulo esterno di conversione A/D - D/A.

E' stato realizzato il sistema operativo della stazione di lavoro che permette il controllo da Host dell'apparato multi-scheda e la comunicazione fra i vari moduli. Le funzioni principali consistono nella distribuzione dei processi sui moduli installati, la sincronizzazione dei dati ed il controllo del flusso dei parametri degli algoritmi a run-time, per mezzo di comandi provenienti dall'elaboratore ospite.

MULTIC25 è un sistema modulare con il quale è possibile approntare delle stazioni economiche per sviluppare e testare algoritmi di DSP in applicazioni nel settore dei segnali audio musicali.

Introduzione

Grazie all'evoluzione della tecnologia VLSI, attualmente sono disponibili sistemi per l'elaborazione numerica in tempo reale di segnali in diverse aree applicative, basati su calcolatori delle classe PC-IBM, MacIntosh, ecc. equipaggiati con moduli di calcolo che utilizzano microprocessori per DSP con funzioni di co-processori programmabili.

Ultimamente alcuni tipi di questi moduli, forniti di circuiterie di conversione adatte alle caratteristiche richieste nella banda audio, sono stati proposti per l'impiego nel settore audio professionale e per applicazioni di ricerca e produzione musicale assistita da calcolatore. I sistemi che ne derivano consentono un maggior grado di libertà grazie alla possibilità di implementare qualsiasi tecnica di sintesi e di trattamento del suono, difficilmente riscontrabile in macchine dedicate e consentono, fra l'altro, un decisivo allungamento dei tempi di obsolescenza delle apparecchiature stesse, unitamente ad una maggiore consapevolezza dei fenomeni acustico-musicali.

Una tecnica di sintesi o di trattamento digitale è rappresentabile in generale con un modello matematico realizzabile con un algoritmo che, eseguito iterativamente, ne implementa la funzionalità; il risultato prodotto è di solito costituito da una sequenza di numeri (campioni) corrispondenti ai valori istantanei del segnale audio desiderato, secondo una prestabilita frequenza di campionamento.

Un sistema per la generazione in tempo reale di segnali musicali deve essere in grado di eseguire un elevato numero di questi algoritmi in un tempo inferiore al periodo di campionamento, che a seconda delle applicazioni può variare fra 100 e 20 μsec. Per avere un sufficiente grado di polifonicità e ricchezza timbrica dei suoni, sono richieste potenze di calcolo dell'ordine di varie decine di MIPS.

Per raggiungere tali livelli di performance, non è sufficiente, e comunque non conveniente, l'utilizzo di un solo processore DSP anche molto potente; in generale è più vantaggiosa la soluzione di realizzare sistemi multi-processor basati su processori di medie prestazioni.

L'IEI/CNR e il CNUCE/CNR sono attivi da tempo nel settore con proposte originali di sistemi polifonici per il lavoro in tempo reale [3] ed hanno sviluppato negli anni passati progetti di apparati multi-DSP con architettura orientata all'elaborazione ed alla sintesi polifonica in tempo reale [4][5], per applicazioni nella ricerca, nella didattica e nell'audio professionale.

Questi lavori, seppur validi da un punto di vista progettuale propositivo, prevedono soluzioni multi-DSP basate su bus specializzati, la cui ingegnerizzazione avrebbe richiesto relativamente alti costi di sviluppo.

Parallelamente un aspetto importante che è stato affrontato è quello dello sviluppo del software per la gestione di tali sistemi e dei tools software di interfaccia utente. I problemi incontrati riguardano principalmente lo scheduling dei compiti per i microprocessori e la loro gestione a run-time, problemi che, d'altra parte, si ritrovano generalmente anche in sistemi che usano processori più potenti, come ad esempio la stazione di lavoro NeXt, sviluppata all'Ircam di Parigi [6], dal costo di varie decine di milioni di lire, oppure il sistema MARS [7][8], in sviluppo all'IRIS di Paliano (Frosinone).

Per poter verificare alcune soluzioni proposte in [9][10] ed effettuare delle sperimentazioni nel settore di nostro interesse riducendo al minimo lo sforzo realizzativo dell'hardware, è stato deciso di assemblare un sistema che prevede un'architettura molto semplice, basata essenzialmente sul bus PC IBM e che usa come componente modulare una scheda DSP commerciale di basso costo, la Leonard C25. Tale scheda, sviluppata nell'ambito di una collaborazione tra l'IEI di Pisa e il Centro Leonardo S.p.a. di Massa, è capace di effettuare l'elaborazione numerica in tempo reale di segnali in campo industriale, biomedico e nel settore audio vocale e musicale.

Il sistema, denominato MULTIC25, consente di fare lavorare in parallelo più processori tramite una semplice soluzione di connessione, offerta dalle schede Leonard, che sfrutta il protocollo seriale DSP della Texas Instruments. Il sistema può essere configurato in modo scalabile, da uno fino ad un massimo di otto moduli, per ottenere il desiderato grado di polifonicità e complessità timbrica richiesto nei diversi ambiti di utilizzo (ricerca, didattica e produzione musicale, effettistica ecc.).

1 - Architettura del sistema

L'architettura di MULTIC25 è basata su più moduli di calcolo Leonard C25 gestiti direttamente dall'elaboratore ospite, un Personal Computer IBM compatibile. In figura 1 è illustrato lo schema dell'architettura della sistema.

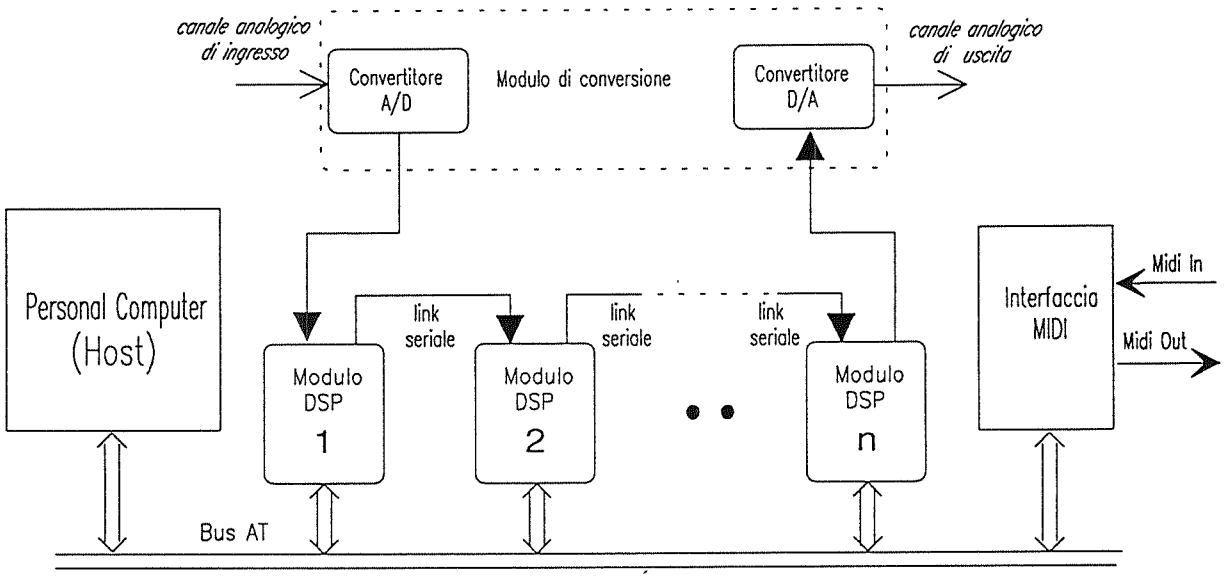


Fig.1 Architettura del sistema MULTIC25

I moduli DSP sono collegati fra loro mediante l'interfaccia seriale ad alta velocità di cui essi dispongono, senza necessità di logica aggiuntiva, in modo da formare una 'catena' nella quale l'uscita del primo modulo è connessa all'ingresso del secondo, l'uscita del secondo a quella del terzo e così via. L'ingresso del primo modulo è poi connesso all'uscita di un convertitore A/D e l'uscita dell'ultimo modulo all'ingresso di un convertitore D/A.

Questa scelta progettuale ha semplificato la messa a punto del sistema ed ha rispettato le esigenze di economicità desiderate, senza limitare troppo le potenzialità applicative.

Le operazioni di conversione dei segnali analogici in ingresso e in uscita dal sistema sono eseguite da un apposito modulo hardware di conversione comprendente dispositivi integrati adatti a trattare segnali in banda audio.

Per l'interfaccia MIDI viene utilizzato un modulo commerciale, facilmente reperibile sul mercato, che ha il compito di rendere possibile il colloquio con le macchine che usano tale standard.

L'Host avrà il compito di assegnare i compiti elaborativi ai vari moduli di calcolo, e di gestire l'interazione fra l'utente e gli algoritmi. Questa interazione potrà avvenire mediante le interfacce standard dell'elaboratore ospite come la tastiera e il mouse, oppure tramite l'interfaccia MIDI e quindi mediante controllers dotati della stessa interfaccia (sequencers, Master keyboards, drum pads, ecc.).

2 - Il modulo Leonard C25

Come elemento modulare per il sistema MULTIC25 è stata scelta la scheda Leonard C25 principalmente per le particolari caratteristiche di versatilità, di flessibilità d'uso e non ultimo per il suo costo estremamente contenuto (.5 ML).

In figura 2 è mostrata l'architettura della scheda utilizzata. Per i dettagli funzionali e realizzativi della scheda rimandiamo a [11]. Possiamo distinguere le seguenti parti principali:

- un microprocessore DSP TMS320C25 con frequenza di clock a 40 MHz, a 16 bit, in grado di eseguire un ampio set di istruzioni tipiche del DSP, fra cui citiamo la moltiplicazione/somma di due operandi (multiply/accumulate) che viene eseguita in 100 ns. Il software sviluppato per il TMS320C25 può essere impiegato eventualmente anche da alcuni processori più veloci della generazione successiva (TMS320C50), di recente apparizione sul mercato.
- un banco di memoria RAM da 28 kwords e un banco di memoria EPROM da 4 kwords, allocati nello spazio di memoria programma del microprocessore. Nella memoria EPROM si trovano le routines per il bootstrap, per la gestione a basso livello della scheda da parte dell'Host e un vettore di 2048 campioni di un periodo della funzione seno, rappresentati su 16 bit in complemento a 2.
- un'interfaccia parallela di comunicazione bidirezionale con l'elaboratore ospite. L'interfaccia è inoltre in grado di inviare richieste di interrupt al controllore di interruzioni dell'elaboratore ospite.

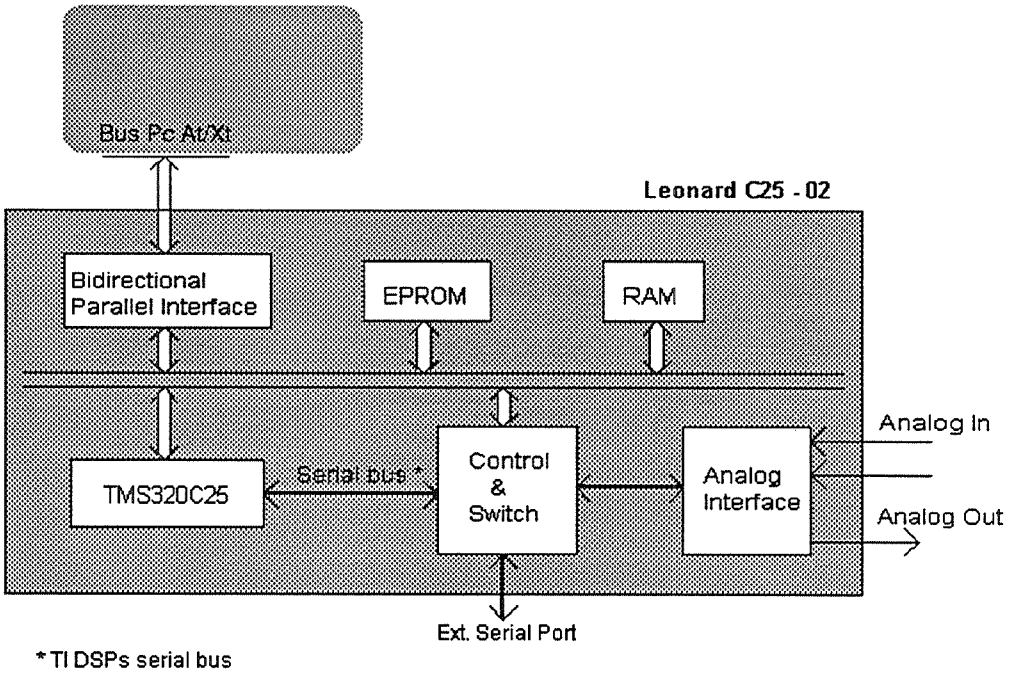


Fig. 2 Architettura del modulo Leonard C25

- un apposita rete (Control & Switch) che effettua la commutazione della porta seriale del TMS sul convertitore interno o verso la seriale esterna.
- un'interfaccia verso il mondo dei segnali analogici costituita da un convertitore integrato A/D-D/A e da una sezione di condizionamento per i segnali di ingresso. Questa parte del modulo non è utilizzata dal sistema descritto in questo lavoro, ma necessaria per il funzionamento autonomo della scheda.
- una porta seriale esterna: il modulo può comunicare con dispositivi esterni (altri moduli Leonard C25, convertitori esterni ecc.) tramite una comunicazione seriale full-duplex (bidirezionale) con velocità massima di 5 Mbit/s. Il protocollo di comunicazione seriale è quello implementato dal TMS320C25 mediante vari registri e segnali di controllo: i principali sono il registro DXR, che contiene il dato da trasmettere e il DRR che contiene il dato ricevuto. Ad ognuno delle due sezioni (trasmissione e ricezione) è associato una linea per la transito dei dati in forma seriale (DX, DR), un segnale di clock per il sincronismo di bit (CLKX, CLKR) ed un segnale per il sincronismo di frame (/FSX, /FSR) per la comunicazione a pacchetti di 8 o 16 bit. Le operazioni di I/O su tale porta possono essere gestite dal programmatore tramite le due interruzioni interne al processore XINT e RINT che notificano rispettivamente il completamento della trasmissione e della ricezione di un frame. La porta è accessibile esternamente tramite un connettore Cannon a 15 poli.

3 - Il modulo di conversione A/D - D/A

L'altro componente hardware del sistema è un modulo di conversione dei segnali analogici in ingresso e in uscita, il cui schema circuitale è riportato in figura 3.

Il modulo è basato su un dispositivo integrato TLC32044, prodotto dalla Texas Instruments, che implementa al suo interno le due operazioni di conversione A/D - D/A. E' provvisto di un'interfaccia digitale che usa il protocollo di comunicazione seriale bidirezionale utilizzato dalla famiglia di processori TMS320CXX e quindi può lavorare direttamente con la scheda Leonard C25, collegandosi a questa mediante l'interfaccia seriale esterna, tramite il connettore Cannon posto sul retro della stessa.

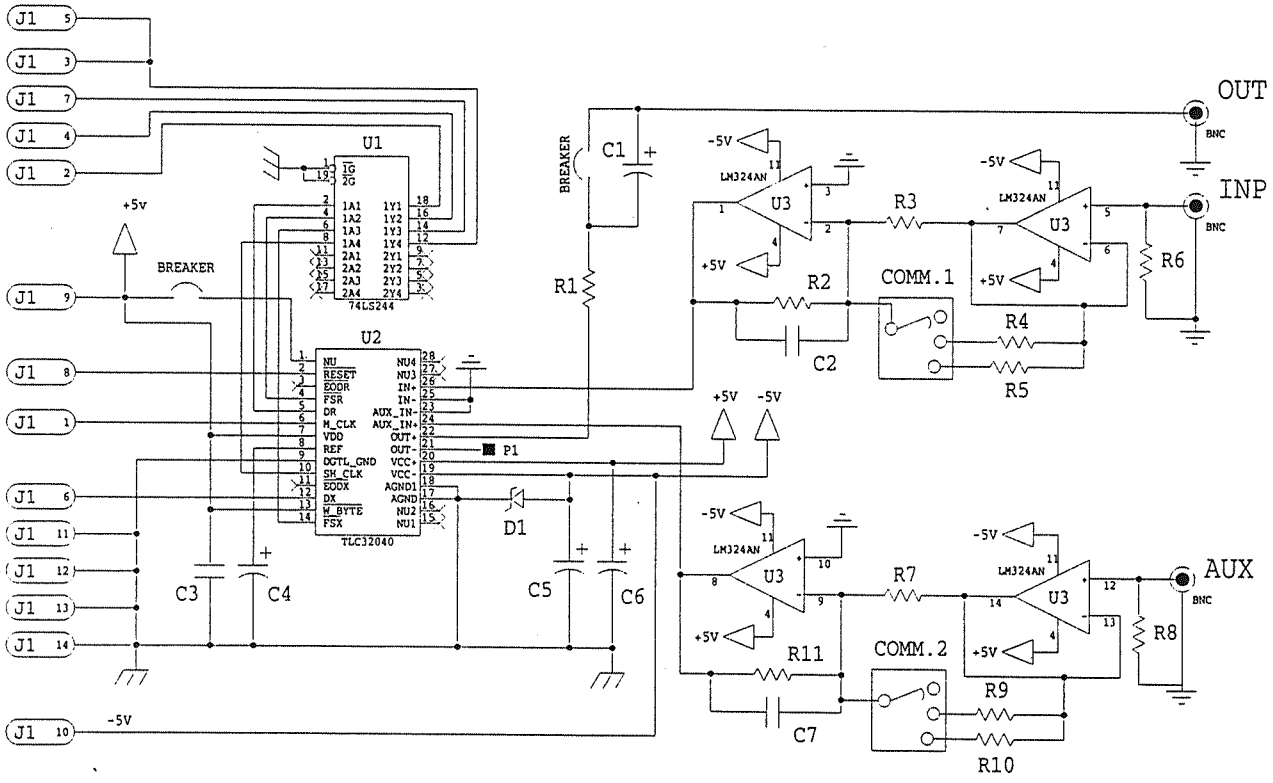


Fig. 3 Il modulo di conversione

I segnali digitali in uscita dal TLC (DR,/FSR,/FSX e SHIFTCLK) sono bufferizzati prima di essere portati al connettore mediante un 74LS244 in modo da aumentarne il fan out.

Il segnale di MASTER CLOCK per il funzionamento di questo integrato è fornito dal modulo Leonard tramite il segnale CLOCKOUT1 (a 10 MHz). Il segnale di SHIFTCLK generato dal TLC e ottenuto mediante divisione per 4 del MASTER CLOCK, è utilizzato per il sincronismo di bit nei trasferimenti seriali. La velocità della comunicazione seriale viene ad essere quindi di 2,5 Mbit/sec che comporta un tempo di 6,8 μsec. per il trasferimento di una parola a 16 bit.

Le due sezioni di conversione del TLC possono essere programmate, tramite tale interfaccia seriale, per lavorare in molte configurazioni operative: è possibile ad esempio impostare la frequenza di conversione per ciascuna sezione, oppure disattivare il filtro passa alto che si trova in ingresso alla sezione A/D. Da prove effettuate si è trovato che il dispositivo lavora egregiamente fino a frequenze intorno a 25 KHz, consentendo di trattare segnali a 12 KHz senza introdurre apprezzabili distorsioni.

Per quanto riguarda la sezione analogica è stata inserita una circuito di condizionamento dei segnali di ingresso al convertitore TLC32044, del tutto simile a quello presente sulla scheda Leonard. Si può variare il guadagno di ciascuna sezione di ingresso (IN e AUX) utilizzando un commutatore che permette di scegliere fra tre possibili valori adatti alle diverse tipologie di segnali da trattare.

4- Le funzioni del sistema operativo

Il software per la gestione del sistema MULTIC25 deve svolgere le seguenti funzioni principali:

1) consentire la distribuzione, sui vari moduli di calcolo, dei processi applicativi eseguibili dal sistema, costituiti dagli algoritmi di sintesi e di elaborazione;

2) gestire le comunicazioni e la sincronizzazione tra le varie componenti del sistema;

3) mantenere ininterrotto il flusso di output dei campioni del segnale, all'interno dei singoli suoni, ad una frequenza prefissata, per evitare l'introduzione di rumore;

4) minimizzare i tempi di risposta alla modifica dei parametri di controllo delle elaborazioni al fine di mantenerli entro le tolleranze prefissate.

Ogni applicazione può essere definita come uno spazio di **n** processi che si evolvono nel tempo; ogni processo è composto da almeno tre sottoprocessi temporalmente distinti [10]:

- *generazione dei campioni*, cioè il calcolo dei campioni dei segnali ottenuti mediante l'esecuzione di algoritmi di generazione. L'intervallo di tempo fra due esecuzioni di questo processo è il periodo di campionamento che può variare tra 100 μsec. e 20 μsec., con scarti consentiti di alcuni decimi di μsec. Degli **n** processi di generazione dei campioni eseguibili globalmente dal sistema, un sottoinsieme **m** sarà costituito da processi di sintesi ed un sottoinsieme **n-m** da processi di elaborazione; il numero **m** dei processi di sintesi corrisponderà alla capacità polifonica del sistema.

- *generazione degli inviluppi*, cioè la lettura dei valori dalle tabelle degli inviluppi o il loro calcolo, da utilizzare come parametri per gli algoritmi. Il tasso di aggiornamento dei valori è dell'ordine del msec. con scarti locali consentiti di frazioni di msec.

- *gestione delle strutture di controllo*, cioè la gestione degli eventi ad alto livello (come ad esempio una sequenza di note musicali) sincronizzati su host ed inviati, codificati opportunamente, all'apparato multi-DSP; gli intervalli di tempo minimi sono dell'ordine dei decimi di secondo con scarti locali consentiti del msec.

La concorrenza tra questi sottoprocessi viene realizzata su ciascun modulo mediante l'uso del meccanismo di interruzione disponibile sul processore TMS320C25, come descritto in dettaglio nei paragrafi successivi.

L'intera applicazione che il sistema deve trattare viene 'spezzata' in sezioni, ciascuna da allocare ad uno degli **N** moduli di calcolo; ogni modulo elabora una di queste sezioni tenendo conto dei risultati provenienti dalla sezione precedente ed inviando i risultati al modulo successivo: si ottiene, in questo modo, un'esecuzione in *pipeline*.

Facendo seguito alla proposta suggerita in [10] è stato accettato che ogni modulo possa ospitare al più 4 degli **n** algoritmi di generazione che l'intero sistema deve eseguire. In effetti, considerando il tempo disponibile per il calcolo degli algoritmi, la complessità media degli stessi, la velocità di elaborazione del processore TMS320C25 e le dimensioni delle memorie programma e dati, si conclude che ben difficilmente potrebbero essere eseguiti più di quattro algoritmi.

Nella rara eventualità che la complessità computazionale di un algoritmo di generazione da trattare superi la capacità di calcolo di un singolo modulo, è demandata all'utente la possibilità di implementare l'algoritmo stesso distribuendolo su più moduli.

In merito alla gestione dello scambio dei dati fra i moduli, si può osservare che, per le richieste di polifonicità e politimbricità, i vincoli temporali fra i vari processi di generazione sono dell'ordine degli scarti consentiti per la gestione delle strutture di controllo, cioè dell'ordine del msec. In questo modo è possibile il passaggio del risultato di un processo da un modulo di calcolo all'altro con frequenza uguale a quella di campionamento, senza che questo comprometta le prestazioni del sistema, tenendo conto che per altre considerazioni (lunghezza di parola, numero di alloggiamenti nei bus, costi ecc.), il numero massimo dei moduli di calcolo è stato fissato a otto.

5 - I collegamenti fra i moduli

I collegamenti tra i moduli di MULTIC25 sono effettuati mediante l'interfaccia seriale di cui le schede Leonard C25 dispongono, senza necessità di logica aggiuntiva.

I collegamenti delle N schede fra loro e con il modulo di conversione sono riportati in fig.4. Come si può vedere i segnali di sincronismo di frame e di bit sono ricavati per tutti i moduli di calcolo dall' /FSX e dallo SHIFTCLK del modulo di conversione, sia per la in trasmissione che per la ricezione. La filatura completa del cavo che realizza il bus seriale è riportata in appendice A.

Il TLC32044, per il corretto utilizzo di questo sistema, deve essere configurato per lavorare in modo sincrono: il periodo di campionamento Tc è unico per la conversione A/D e per quella D/A. Inoltre con cadenza temporale Tc tutti i moduli eseguono, in modo contemporaneo, il trasferimento del dato prodotto al modulo successivo della catena: il convertitore A/D trasferisce il suo dato al primo modulo, il primo al secondo e così via fino all'ultimo modulo che trasferisce il suo dato al convertitore D/A.

Naturalmente ogni modulo deve produrre il dato da trasferire, oltre che in funzione delle proprie elaborazioni, tenendo conto opportunamente del dato ricevuto: in questo modo al convertitore D/A giunge, ogni Tc, il risultato complessivo dell'intera elaborazione.

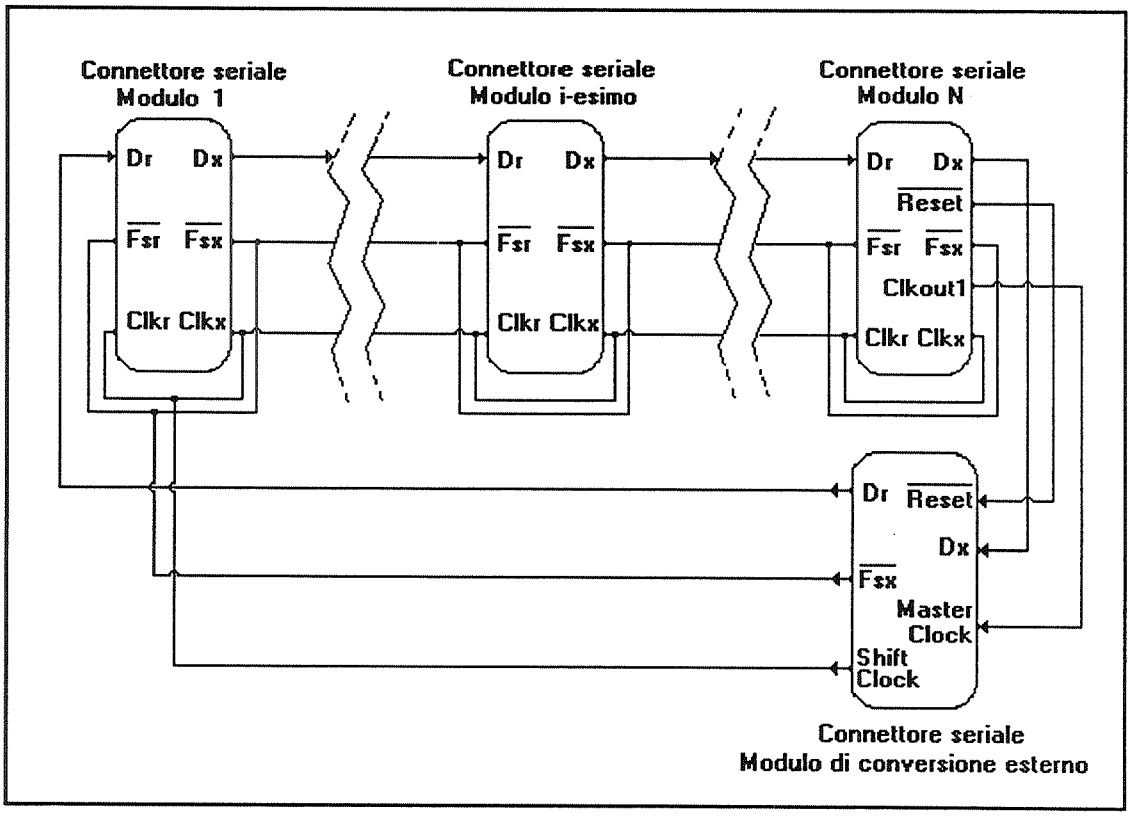


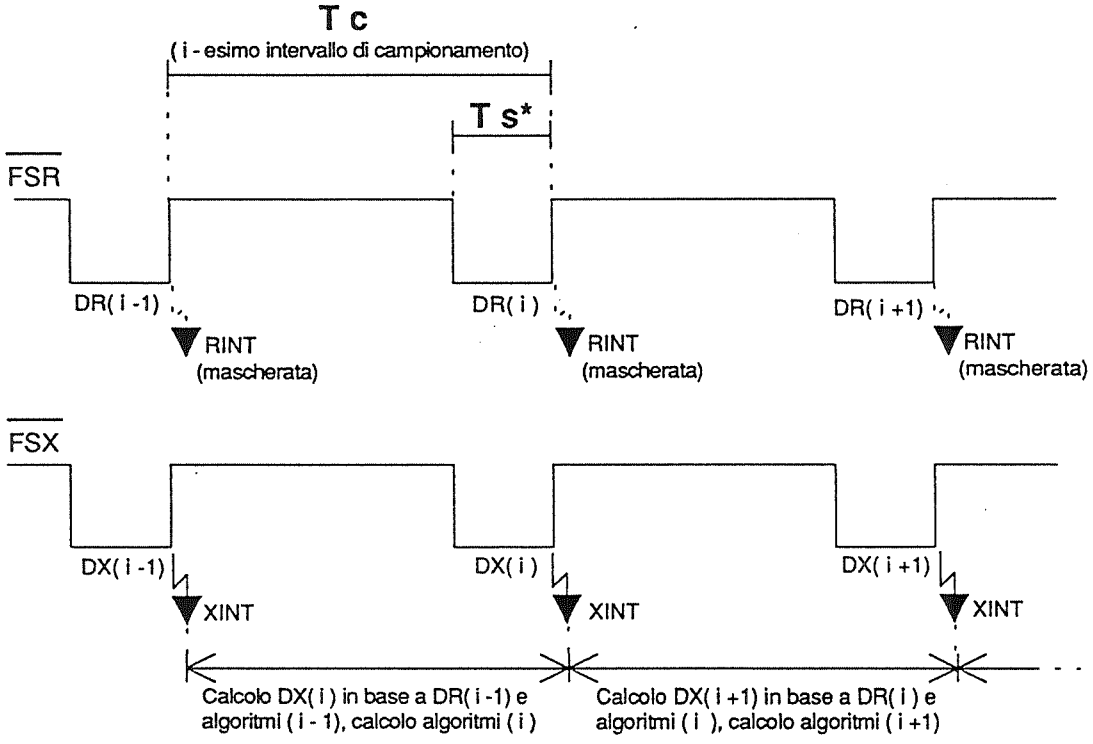
Fig. 4 Connessioni sulle porte seriali delle schede all'interno del sistema.

I moduli non devono invece tenere conto dei diversi ritardi con cui i dati fluiscono nella catena, in quanto questi rientrano ampiamente nello scarto temporale consentito dell'ordine del msec.

In figura 5 è riportato l' andamento operazioni di trasferimento per uno dei moduli. In questa schematizzazione la proporzionalità fra il periodo di campionamento e il tempo di trasferimento di un dato è puramente qualitativa, considerando i possibili valori assegnabili alla frequenza di campionamento.

Come scelta di progetto, imponiamo che la frequenza di campionamento per il sistema sia fissa: configureremo il TLC per lavorare alla frequenza di 19,841 KHz, quindi con Tc di circa 50 μsec, realizzando un compromesso fra la necessità di disporre del maggior tempo possibile per il calcolo algoritmico e quella di poter trattare segnali audio a larga banda.

Si può notare inoltre che il processo che esegue in sequenza calcolo del campione da trasmettere ed il calcolo degli algoritmi per il periodo successivo, deve essere attivato in seguito all'interruzione interna XINT oppure RINT. Dato il sincronismo delle operazioni, le due interruzioni sono concettualmente contemporanee, per cui l'una o l'altra hanno come contenuto informativo sia l'avvenuta trasmissione che l'avvenuta ricezione di un dato. Come vedremo successivamente la scelta progettuale è stata quella di gestire l'interruzione XINT, tenendo mascherata la RINT. Niente avrebbe impedito di effettuare la scelta contraria.



* T s = Tempo di trasferimento del dato = (1 / CLKX) * (16 + 1) = 6,8 μsec.

Fig. 5 Operazioni di trasferimento per un modulo

6 - La gestione dell'interruzione a frequenza di campionamento

In risposta all'interruzione di fine trasmissione XINT, su ciascun modulo di calcolo si attiva, quindi, il processo di generazione dei campioni cioè l'esecuzione degli algoritmi di generazione, preceduta dal calcolo del dato da trasmettere al modulo successivo, in modo tale che alla successiva attivazione del frame di sincronismo venga trasmesso questo dato. I risultati degli algoritmi di generazione saranno memorizzati in opportune locazioni di sistema (**RIS**i con i = 1 .. 4), in modo da essere utilizzati nel periodo successivo.

Per il calcolo del dato da trasmettere, per prima cosa si devono sommare assieme i 4 risultati che gli algoritmi hanno prodotto nel periodo precedente. Come abbiamo già detto il formato dati del TMS320C25 è di 16 bit, mentre l'accumulatore è a 32 bit. Poiché il risultato della somma deve essere ancora rappresentabile su 16 bit, dovremo dividere per 4 il contenuto dell'accumulatore, dove è stata fatta la somma, o meglio effettuare uno shift di 2 posizioni a sinistra.

Il codice in assembler per queste operazioni è il seguente:

```
LAC     RIS1
ADD     RIS2
ADD     RIS3
ADD     RIS4
SFR
SFR
```

Poi questo risultato deve essere combinato con quello ricevuto dal modulo precedente e che può essere letto nel registro DRR.

In generale ogni modulo avrà due parametri da usare come coefficienti moltiplicativi, uno per il risultato della propria elaborazione (**VOLPROPRIO**) ed uno per il dato ricevuto dal modulo precedente (**VOLRICEVUTO**), in modo da ottenere, sommando i risultati delle moltiplicazioni, la combinazione opportuna dei due dati, su 16 bit. Il risultato va poi quantizzato su 14 bit che è il formato dei dati del convertitore D/A. Il codice di queste operazioni è il seguente:

```
SSXM          ; setta il bit per l'estensione di segno
SPM   1         ; setta lo shift per l'uscita del reg. P
SACL  TMP       ; TMP <-- (RIS1+RIS2+RIS3+RIS4)/4
LT    VOLPROPRIO
MPY   TMP       ; reg. P <-- TMP * VOLPROPRIO
ZAC
LT    VOLRICEVUTO
LDPK  DRR
MPYA  DRR; ACC <-- reg. P (con 1 bit di segno)
                      ; reg. P <-- DRR * VOLRICEVUTO
APAC     ; ACC <-- ACC + reg. P (con 1 bit di segno)
LDPK  TMP
SACH  TMP   ;TMP <-- risultato a 16 bit con 1 bit di segno
LAC   TMP
ANDK  0FFFCH      ; quantizzazione a 14 bit del risultato
LDPK  DXR
```

```
SACL   DXR          ; invio al modulo successivo
SPM    0          ; ripristina 0 shift per l'uscita del reg. P
```

Il valore di **VOLPROPRIO** e di **VOLRICEVUTO** dipenderà dal numero dei moduli **N** e dal tipo di applicazione da eseguire. Come applicazioni possono essere riconosciute due classi di algoritmi: algoritmi di sintesi, con i quali vengono generati direttamente dei segnali audio implementando una determinata tecnica, che può anche prevedere l'utilizzo di dati precedentemente memorizzati in strutture adeguate; algoritmi di elaborazione con i quali invece si effettua il trattamento di segnali prelevati da sorgenti fisiche ed acquisiti con convertitori A/D, oppure l'elaborazione dei risultati di altri algoritmi (di sintesi o di elaborazione), prima di inviarli in output (effetti di eco, riverbero, ecc...).

Nel seguito faremo riferimento ad entrambe le classi di algoritmi con il termine di *tecniche di generazione digitale.*

Se tutti i moduli devono eseguire algoritmi di 'sintesi', al convertitore D/A dovrà giungere la combinazione di tutti i risultati, indipendente dalla posizione che il modulo occupa nella catena, cioè :

$$\frac{1}{N}\sum_{j=1}^{N}\frac{\sum_{i=1}^{4}RISi}{4}$$

Per ottenere questo risultato ciascun modulo dovrà trasmettere al successivo il seguente valore:

$$\frac{\sum_{i=1}^{4}RISi}{4}\frac{1}{j}+\frac{j-1}{j}DRR$$

dove j rappresenta la posizione del modulo nella catena. Quindi in questo caso è **VOLPROPRIO** = 1/j e **VOLRICEVUTO** = (j-1)/j, rappresentati normalizzati su 16 bit, con 1 bit di segno e 15 bit di parte decimale.

Supponiamo invece che tutti i moduli eseguano algoritmi di 'elaborazione': il primo modulo elaborerà il dato acquisito dal convertitore A/D, il secondo modulo elaborerà il risultato prodotto dal primo e così via, fino al convertitore D/A che riceverà l'elaborazione dell'ultimo modulo. In questo caso è quindi **VOLPROPRIO** = 1 e **VOLRICEVUTO** = 0 per tutti i moduli: saranno gli algoritmi di elaborazione stessi a tenere conto del dato ricevuto dalla scheda precedente e non la procedura di calcolo del campione da trasmettere.

Il programma residente su Host che consentirà di gestire MULTIC25 dovrà permette di configurare ciascuna scheda per eseguire compiti di 'sintesi' oppure di 'elaborazione'. Si possono ottenere quindi configurazioni complessive intermedie fra i due casi esaminati. In generale si può pensare che ogni modulo che

fa 'elaborazione' separi due sottocatene di moduli che fanno 'sintesi'. Un esempio è riportato in figura 6.

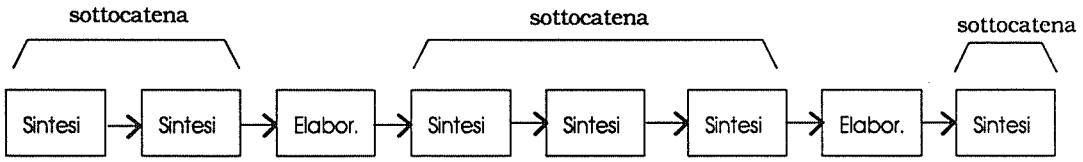


Fig. 6 Un esempio di configurazione della catena

Ogni modulo di 'elaborazione' opera sul risultato della precedente sottocatena di moduli che fanno 'sintesi': per esso vale sempre **VOLPROPRIO** = 1 e **VOLRICEVUTO** = 0.

Per ogni modulo di 'sintesi' appartenente alla sottocatena di cui il primo modulo è anche il primo modulo di tutta la catena vale ancora **VOLPROPRIO** = 1/j e **VOLRICEVUTO** = (j-1)/j.

Per ogni modulo di 'sintesi' appartenente invece alle altre sottocatene, per comprendere nel calcolo anche il risultato del precedente modulo di 'elaborazione', deve essere **VOLPROPRIO** = 1/(j +1) e **VOLRICEVUTO** = j / (j + 1), dove questa volta j è la posizione del modulo nella sottocatena.

Sarà compito del programma di gestione impostare, per ogni scheda, i valori dei due parametri, in base alla configurazione operativa scelta dall'utente.

7 - Gestione dei moduli di calcolo

Oltre al processo di generazione dei campioni, su ciascun modulo devono essere eseguiti altri processi fra cui quello di generazione degli inviluppi e quello di gestione dei comandi da Host.

Al processo di generazione dei campioni deve essere assegnata la priorità maggiore, in modo da mantenere ininterrotto il flusso di output dei campioni: il driver di gestione della XINT deve essere quindi eseguito dal processore completamente ad interruzioni disabilitate, mentre gli altri processi non devono tenere le interruzioni disabilitate per un periodo superiore a quello di campionamento.

Il processo di generazione degli inviluppi, come abbiamo già detto, deve essere eseguito con una cadenza dell'ordine del msec. con scarti locali consentiti di frazioni di msec. Potremmo utilizzare per la sua temporizzazione il timer interno al TMS, oppure sincronizzarlo con il periodo di campionamento di 50 μsec. La soluzione che adottiamo è la seconda.

Il processo è eseguito in *background*, ad interruzioni sempre abilitate. Il driver di gestione della XINT, prima di eseguire il ritorno da interrupt decrementa una variabile (CLOCK), alla quale all'inizializzazione viene assegnato il valore 20. Al termine del calcolo degli inviluppi, il processo in background cicla attendendo che CLOCK sia arrivato a 0. Quando questa condizione è soddisfatta, ricomincia la sua esecuzione dopo aver assegnato nuovamente a CLOCK il valore 20.

In questo modo si ottiene che il processo di generazioni degli inviluppi viene eseguito con periodicità di 1 msec. (20* 50 μsec.).

Riportiamo il codice relativo alla procedura di terminazione del processo:

```
LIMCLOCK  .SET    20

ATTESA: LDPK  CLOCK
        LAC   CLOCK
        BGZ   ATTESA   ; if CLOCK > 0 va ad ATTESA
        LACK  LIMCLOCK ; else ricarica clock al valore LIMCLOCK
        SACL  CLOCK
        B     BACKGR     ; riesegue il processo
```

Naturalmente il driver di gestione della XINT esegue opportunamente il salvataggio e il ripristino dello stato del processore.

Inoltre, perché il processo in background possa essere eseguito correttamente entro 1 msec., il processo di generazione dei campioni dovrà lasciare libero un intervallo di tempo all'interno del periodo di campionamento, per il calcolo degli inviluppi. Per dare una valutazione quantitativa di tale intervallo occorre fare delle ipotesi: supponiamo che i quattro algoritmi di generazione siano quattro algoritmi di sintesi con quattro oscillatori e quindi gli inviluppi da gestire, due per ogni oscillatore, siano 32.

Da valutazioni effettuate, si è trovato che il tempo massimo necessario per eseguire un algoritmo di aggiornamento di un singolo inviluppo può essere ritenuto 5 μsec. Il tempo massimo per gestire tutti gli inviluppi sarà quindi 32 * 5 = 160 μsec., da distribuire in 20 periodi di campionamento (1 msec. / 50 μsec.); per ogni

Tc (di 50 µsec.) si dovrà quindi eseguire al massimo 160 / 20 = 8 µsec. di processo in background. Quindi il tempo riservato al processo di generazione dei campioni all'interno del periodo di campionamento non dovrà superare i 42 µsec., tempo normalmente sufficiente per eseguire quattro algoritmi di sintesi con quattro oscillatori ciascuno.

Per quanto riguarda i comandi da Host, una parte di questi sono gestiti dal software residente su EPROM, per la gestione di base del modulo, come la scrittura e la lettura di blocchi di memoria, dei registri del TMS e così via. Queste procedure, attivate dall'interruzione INT0, per poter svolgere correttamente i loro compiti, sono eseguite dal processore ad interruzioni disabilitate: possono essere utilizzate per il caricamento in memoria dei programmi oppure per il debugging dell'attività del TMS, ma non possono essere utilizzate in concorrenza con il processo di generazione dei campioni.

Per questo motivo è stato assemblato un altro insieme di routines per il controllo a *run-time* degli algoritmi che si occupa di gestire il modulo più ad alto livello (comandi di attivazione e disattivazione delle note musicali ecc.), da eseguire ad interruzioni abilitate, attivabili mediante un'altra sorgente di interruzione, la INT1. L'insieme di queste routines sarà descritto in dettaglio nel prossimo paragrafo.

Il processo di interpretazione e di gestione di questi comandi (che abbiamo chiamato 'interprete dei comandi'), si trova quindi al livello di priorità intermedia fra quello di generazione dei campioni e quello in background.

In figura 7 è illustrato lo schema temporale dei tre processi in esecuzione.

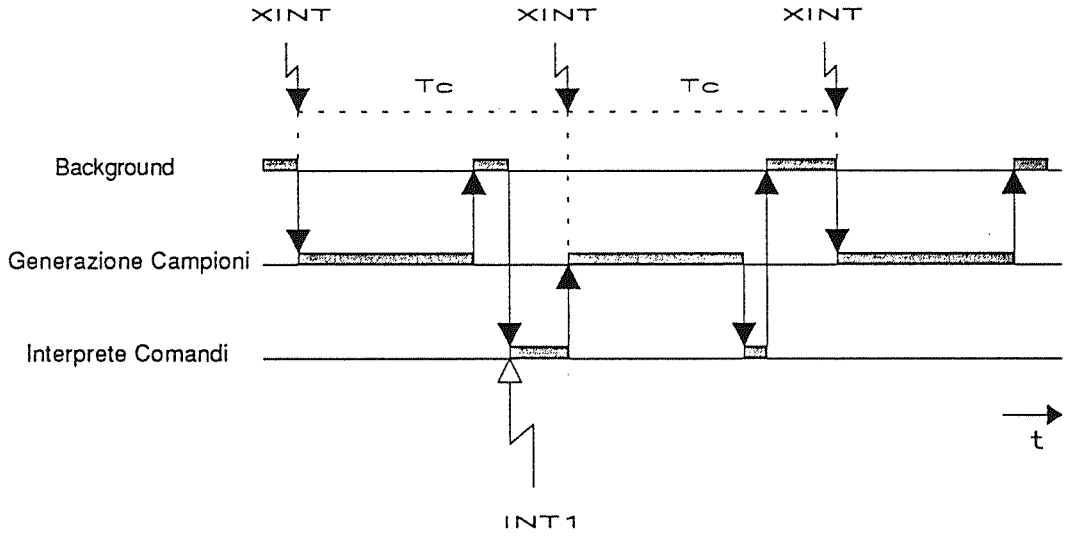


Fig. 7 Schema temporale dei processi in esecuzione su un modulo

Si può notare che l'interprete comandi può interrompere il processo in background e può a sua volta essere interrotto dal processo di generazione dei campioni.

Questa scelta rispetta i requisiti temporali dei tre processi. Infatti, il processo di generazione dei campioni non può subire alcun ritardo. Il processo di interpretazione dei comandi potrà essere ritardato al massimo di qualche periodo di campionamento dal processo di generazione, ritardo più che accettabile considerando gli scarti temporali che si possono avere sulla gestione dell'invio dei comandi da parte dell'Host. Il processo in background può essere ritardato dall'interprete dei comandi al massimo di qualche frazione di msec., scarto temporale conforme con il tasso di aggiornamento.

Per consentire una eventuale temporizzazione di sequenze di eventi musicali sull'elaboratore ospite, su uno dei moduli dovrà essere eseguito un ulteriore processo. Questo processo viene attivato dall'interruzione del timer di cui il TMS dispone, programmato per generare interruzioni ogni 5 msec. ed esegue solo una operazione di OUT alla porta **PA7** in modo da provocare una richiesta di interruzione al PC:

```
TIMER:  OUT   TMP,PA7 ; scrittura fittizia : manda IRQ a PC
        EINT          ; si ritorna al processo interrotto
        RET           ; riabilitando le interruzioni
```

Come si vede non sono necessari salvataggi di stato e quindi la durata del processo, di 4 cicli macchina cioè di 400 nsec. non condiziona minimamente l'esecuzione degli altri processi. A sua volta il processo di timer può subire un ritardo dovuto al processo di generazione dei campioni al massimo di un periodo di campionamento, tempo assolutamente trascurabile nella gestione di eventi musicali.

Tutte le routines di sistema sono state raccolte in un file riportato in appendice (Sys.asm).

L'insieme totale delle procedure relative ai processi di calcolo di ciascun modulo, può essere strutturato come un programma eseguibile per il processore TMS320C25. Le sezioni di programma relative alle specifiche degli algoritmi di generazione e di inviluppo, devono essere specificate dall'utente. Uno dei compiti del programma sull'elaboratore ospite che dovrà occuparsi della gestione della stazione di lavoro, sarà appunto quello di consentire all'utente l'inserimento di queste specifiche.

In figura 8 è mostrata la mappa completa degli spazi di memoria programma e dati di ciascun modulo di calcolo.

Per quanto riguarda la gestione della memoria, per ciascuno dei quattro algoritmi di generazione previsti (che per semplicità possiamo chiamare *voci*) viene riservata una zona di memoria dati *locale* composta da 64 variabili. Un ulteriore

blocco di memoria di 192 variabili costituisce invece una zona *globale*, utilizzabile da tutte le voci.

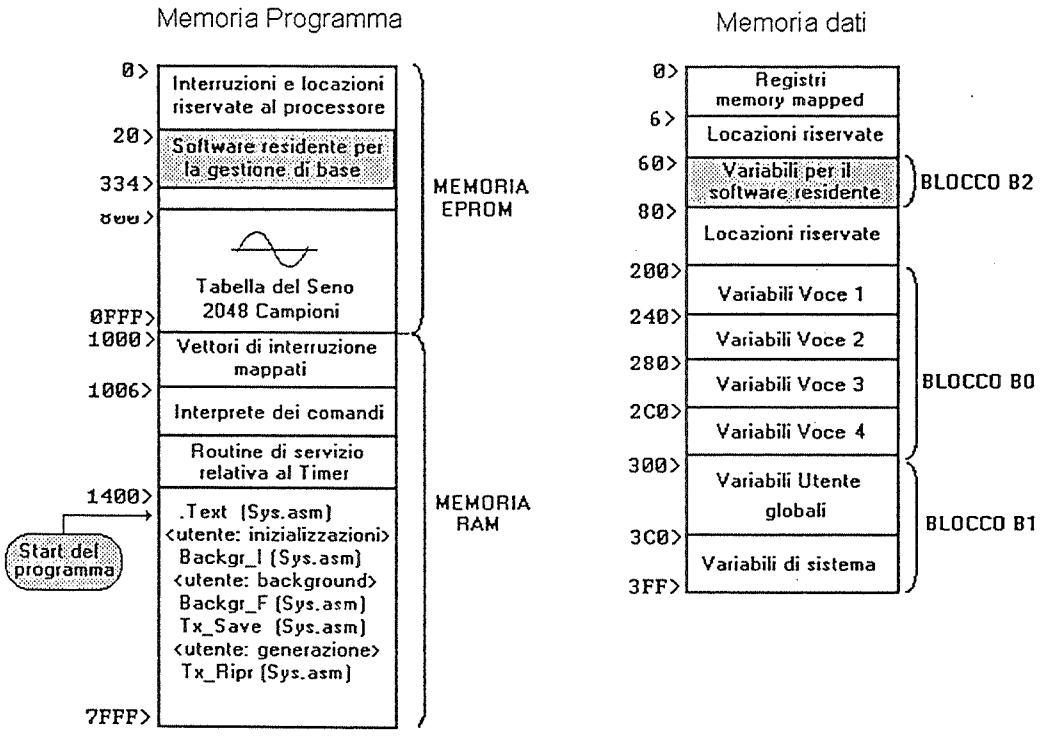


Fig. 8 Mappa delle memorie programma e dati

8- L'interprete dei comandi

Il codice assembler completo relativo all'interprete dei comandi si trova in appendice, nel file Sys.asm, insieme alle routines per la comunicazione fra i moduli.

Allo stato attuale questo insieme di routines comprende solo tre comandi: Note_On (attivazione di nota), Note_Off (disattivazione di nota) e Set_Var (assegnamento di un valore ad una variabile locale di una voce). Sviluppi futuri del sistema potranno prevedere fino a 128 tipi di comandi diversi.

I tre comandi sono implementati attraverso l'invio di due word da parte dell'Host dopo l'invio della richiesta di interrupt INT1. Il formato di queste word è riportato in figura 9.

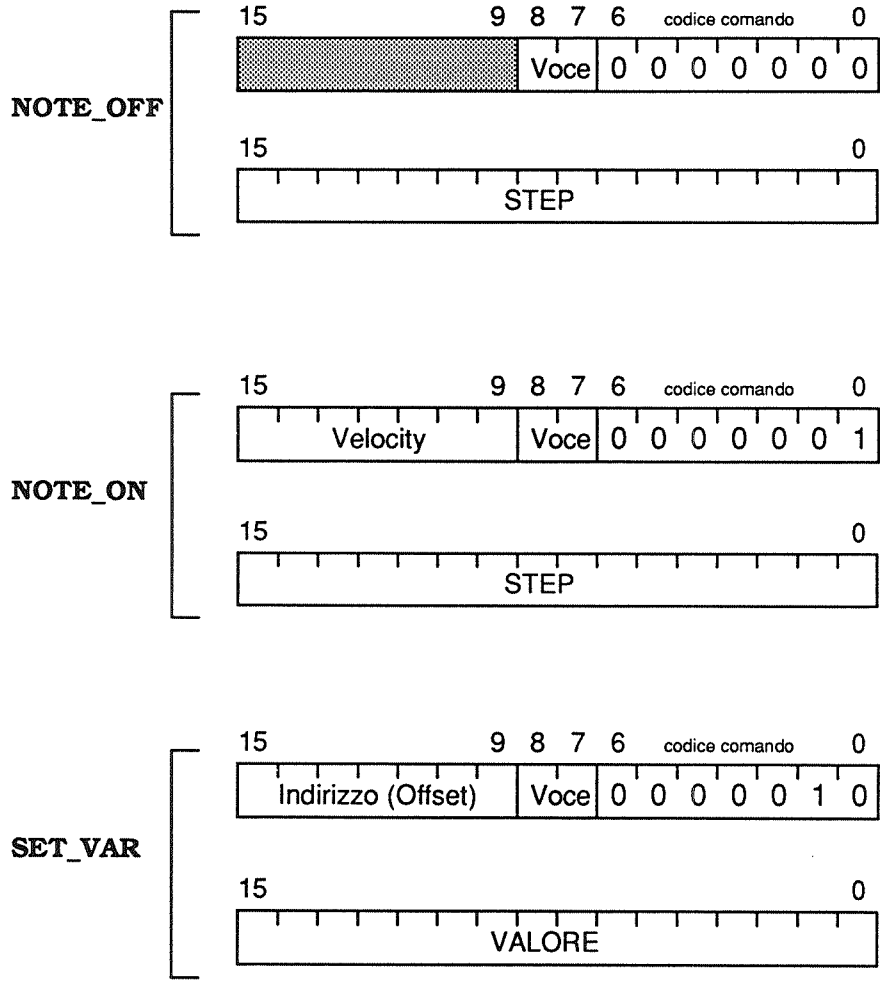


Fig. 9 Formato dei comandi

I comandi di Note_Off e Note_On sono stati implementati ad imitazione dei relativi comandi del protocollo MIDI (Musical Instrument Digital Interface), per consentirne un'agevole traduzione. L'interprete scrive i valori ricevuti in apposite

locazioni di sistema per ognuno dei quattro algoritmi: **STEP1**, **STEP2**, **STEP3**, **STEP4** per quanto riguarda il valore frequenziale relativo alla nota e **VELO1**, **VELO2**, **VELO3**, **VELO4** per quanto riguarda la pressione con cui è stato premuto il tasto (quest'ultima solo nel caso di Note_On).

Inoltre per differenziare lo stato (On, Off) vengono settati o resettati due bit per ogni voce della locazione di sistema **NOTE**. Il primo di questi due bit viene chiamato convenzionalmente bit di nota e il secondo bit di stato, come si può vedere dalla figura 10.

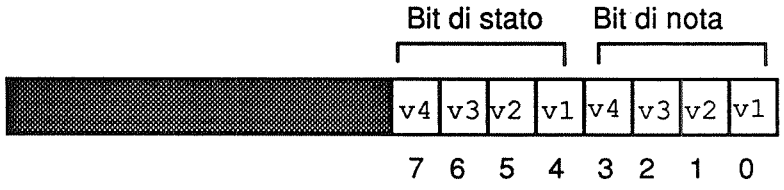


Fig. 10 La parola NOTE

9- Considerazioni Conclusive e Sviluppi Futuri.

Nella realizzazione del sistema MULTIC25 un primo obiettivo raggiunto è stato quello di dotare il sistema di un software di base che consenta all'utente di gestire le risorse hardware nella maniera più trasparente possibile, svincolandosi dalle problematiche tipiche del basso livello.

Come applicazione è stato sviluppata la prima versione di un programma denominato MuSt C25 (MusicStation C25, le cui funzionalità sono descritte in un apposita nota interna del P.F. [12]), operante in ambiente Windows 3.1, che consente di configurare e gestire a run-time fino a otto schede in applicazioni di sintesi e di trattamento di segnali musicali a spettro complesso e con medio grado di polifonia

Questo programma si configura come un vero e proprio ambiente di lavoro che consente, al livello attuale, la specifica degli algoritmi di generazione in codice assembler corredato da una serie di nuove istruzioni, definite appositamente. Inoltre consente di controllare ad alto livello la generazione dei segnali mediante l'utilizzo di strumenti grafici e di un sequencer integrato.

Al livello di sviluppo a cui è stato condotto, la stazione di lavoro corredata dal programma MuSt C25 offre già molte possibilità allo sperimentatore di algoritmi per la sintesi e l'elaborazione digitale di segnali musicali in tempo reale.

Il sistema, d'altra parte, potrà essere ulteriormente ampliato per espanderne le funzionalità.

Una prima necessità che si impone è quella di effettuare l'integrazione nella stazione di lavoro di un editore grafico per la specifica degli algoritmi, già realizzato presso il CNUCE/CNR per computer MacIntosh, e di implementarne le procedure grafiche per gestire l'interazione fra le procedure di generazione degli algoritmi e gli strumenti di controllo di cui l'utente dispone.

Dal punto di vista hardware, presso i laboratori dell'IEI, è stato sviluppato un modulo di conversione (ASI-16/23) che utilizza circuiti integrati basati sulla sigma-delta modulation. Integrando questo dispositivo nel sistema (eventualmente in alternativa al modulo di conversione attualmente utilizzato), la stazione di lavoro potrà presentare caratteristiche audio professionali, consentendo il trattamento di segnali stereofonici ad alta fedeltà. Inoltre è in previsione la realizzazione di un sistema per un livello di prestazioni superiori che utilizzi, come compontenti modulari, schede DSP con microprocessori floating-point (TMS320C30 o TMS320C31).

10 - Bibliografia

[1] G. Bertini, L.M. Del Duca, M.Marani
The Leonard C25 System for real time Digital Signal Processing
Proceed's Int'l Workshop on Man-Machine Interaction
Pisa, Italy 1991, pag. 107- 118

[2] G. Bertini
Progetto, funzionamento e prestazioni dei moduli LeonardC25-02.
Contratto di collaborazione tecnico-scientifica Leonardo Spa - IEI / CNR, nota interna IEI B4-57, novembre 1992

[3] G. Bertini, M.Chimenti, F.Denoth
TAU2: Un Terminale Audio per Esperimenti di "Computer Music" Alta Frequenza, vol. 12 dicembre 1977

[4] L. Tarabella, G. Bertini
Un sistema DSP ad elevate prestazioni controllato da Personal Computer.
Quaderni di Musica Realtà n. 14, Ed. Unicopli, Milano 1987, pag 330- 335

[5] L. Tarabella, G. Bertini
A DSP System and Graphic Editor for Systhesis Algorithms.
Int'l Computer Music Conference, Proceed's, Columbus Ohio, USA 1989, pag. 312- 315

[6] D.A. Jaffe
Efficient dynamic resource management on multiple DSPs, as implemented in the NeXT Music Kit.
Proceedings of the ICMC, pp. 188 - 190, 1990

[7] S. Cavaliere, G. Di Giugno, E. Guarino
MARS: The X20 device and the SM1000 board.
Proceedings of the ICMC, pp. 348 - 351, 1992

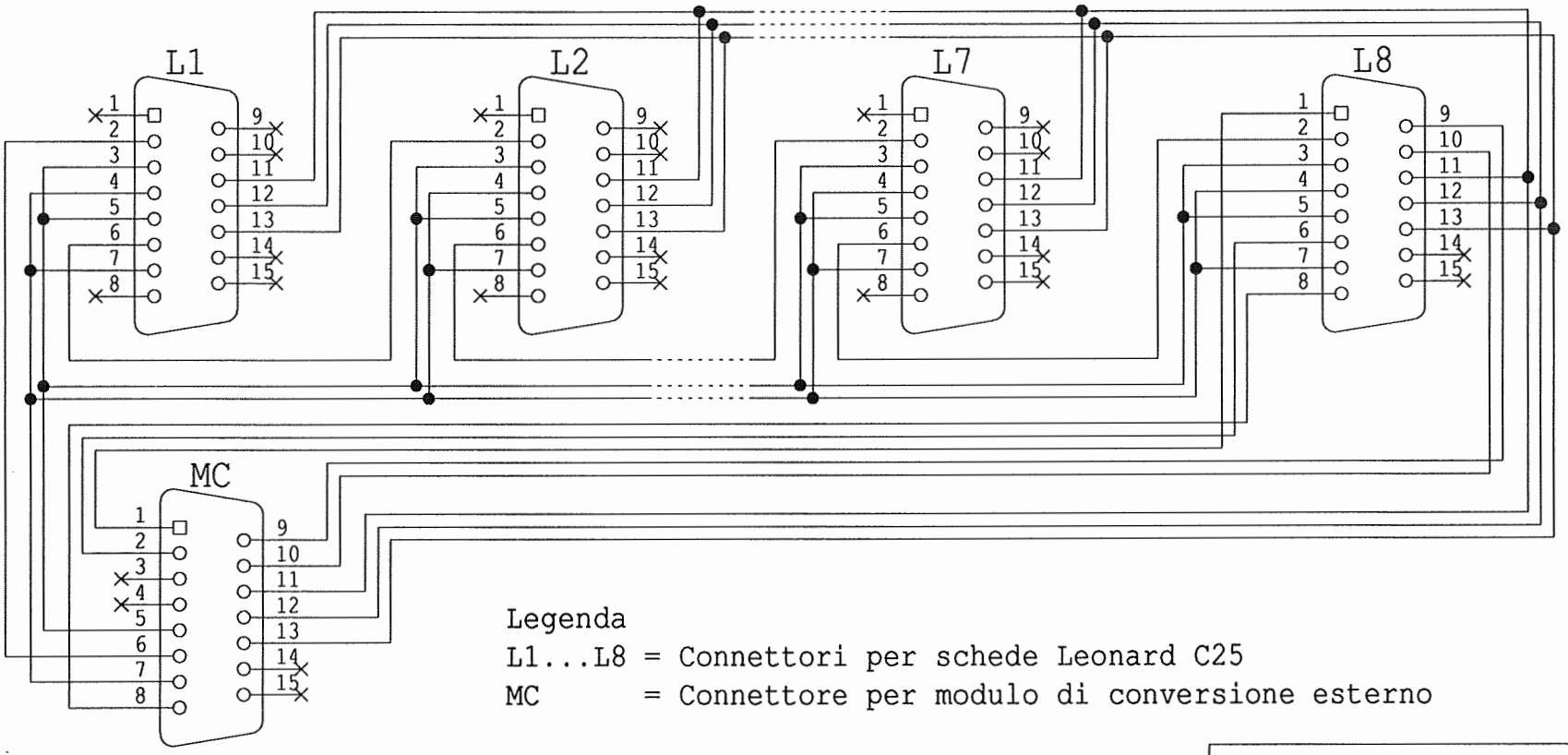
[8] F. Armani, L. Bizzarri, E. Favreau, A. Paladin
MARS: DSP Environment and Application.
Proceedings of the ICMC, pp. 334 - 347, 1992

[9] G. Bertini, P.Chiodaroli, L. Tarabella, S. Toni
Tecniche per la gestione in tempo reale del sintetizzatore audio digitale MP3A.
Nota interna IEI/CNR, dicembre 1988

[10] G. Bertini, L.Tarabella, G.Bacchiocchi, M.Balestrieri
Tecniche di gestione delle risorse in sistemi multi-DSP per la sintesi e l'elaborazione in tempo reale di segnali audio
Internal Report P.F. "Sistemi Informatici e Calcolo parallelo"
CNR/IEI PI2 - R/2/55 agosto 1991

[11] G. Bertini, A. Landucci, M.Moretto, A.Moretto
Scheda LeonardC25-02. Descrizione della realizzazione.
Contratto di collaborazione tecnico-scientifica Leonardo Spa - IEI / CNR, nota tecnica IEI B4-58, novembre 1992

[12] G. Bertini, D. Fabbri, L. Tarabella
MuStC25: una stazione di lavoro musicale con il sistema MULTIC25. Descrizione funzionale e manuale operativo.
Internal Report P.F. "Sistemi Informatici e Calcolo parallelo"
CNR/IEI PI2 - R/2/109 maggio 1993



Legenda

L1...L8 = Connettori per schede Leonard C25

MC = Connettore per modulo di conversione esterno

NB In assenza di una scheda connettere il pin 6 con il pin 2 del relativo connettore

Per il funzionamento del sistema deve essere sempre inserita una scheda sul connettore L8

IEI DEL CNR PISA	
CONNETTORI PORTE SERIALI MultiC25	
CNT SCH	M. Moretto

APPENDICE B : Sys.asm

```
*************************************************************************
*                Modulo   di  sistema                                   *
*************************************************************************



*************************************************************************
*             Dichiarazione di    simboli globali
          *
*************************************************************************

      .GLOBAL   IMR, DXR, DRR
      .GLOBAL   SENOTAB
      .GLOBAL   BEGIN
      .GLOBAL   STEP1, STEP2, STEP3, STEP4, RIS1, RIS2, RIS3, RIS4,
      .GLOBAL   VELO1, VELO2, VELO3, VELO4, NOTE



*************************************************************************
*                 Assegnazione di  Simboli                              *
*************************************************************************

SENOTAB      .SET      2048    ; indirizzo tabella del seno



*************************************************************************
* Sezione Registri mappati in memoria in data memory a indirizzo 0H   *
*************************************************************************

       .SECT    "MMR"
DRR:   .SPACE  16
DXR:   .SPACE  16
TIM:   .SPACE  16
PRD:   .WORD   0C350H;    periodo del timer di 5 millesimi di sec.
IMR:   .SPACE  16
GREG:  .SPACE  16



*************************************************************************
*      Sezione Vettore di interruzione mappato in                       *
*         program  memory a indirizzo 1000H                             *
*************************************************************************

       .SECT   "VINT"
RINVIO:     .SPACE  2*16
ISR1:       B       INTERPRETE
ISR2:       .SPACE  2*16
TISR:       B       TIMER
RISR:       .SPACE  2*16
XISR:       B       TX
```

```
TRSR:       .SPACE  2*16

*************************************************************************
*     Sezione    variabili   per il salvataggio dei registri di        *
*     stato:  in data memory ad indirizzo  7Ch        *
*    (Fine Blocco  B2, pag. 0)         *
*************************************************************************


STATO0      .USECT      "STATOVAR",1
STATO1      .USECT      "STATOVAR",1
STATOAG0    .USECT      "STATOVAR",1
STATOAG1    .USECT      "STATOVAR",1


*************************************************************************
*       Sezione    variabili   di   sistema:  in data memory   ad      *
*       indirizzo  3C0h  (Fine Blocco  B1, pag. 7)                      *
*************************************************************************

      .BSS          TEMP, 1
      .BSS          TMP, 1
      .BSS          DUMMY, 1
      .BSS          UTI, 1
      .BSS          RACCL , 1
      .BSS          RACCH , 1
      .BSS          RAR0SYN, 1
      .BSS          RAR1SYN, 1
      .BSS          RARAGG, 1
      .BSS          RIS1, 1
      .BSS          RIS2, 1
      .BSS          RIS3, 1
      .BSS          RIS4, 1
      .BSS          VELO1, 1
      .BSS          VELO2, 1
      .BSS          VELO3, 1
      .BSS          VELO4, 1
      .BSS          STEP1, 1
      .BSS          STEP2, 1
      .BSS          STEP3, 1
      .BSS          STEP4, 1
      .BSS          IND_RITORNO, 1
      .BSS          ACCL_AGG, 1
      .BSS          ACCH_AGG, 1
      .BSS          RPH, 1
      .BSS          RPL, 1
      .BSS          RT, 1
      .BSS          CLOCK, 1


*************************************************************************
*       Sezione    variabili   di   sistema 2:  in data memory  ad     *
*            indirizzo  3FDh  (Fine Blocco  B1, pag.7)                  *
*************************************************************************

NOTE        .USECT    "SYSVAR2",1    ;
VOLPROPRIO  .USECT    "SYSVAR2",1    ; inizializzate al caricamento
VOLRICEVUTO .USECT    "SYSVAR2",1    ;
```

```
;*********************************************************************
;*                    MACRO  PER  INT1          *
;*********************************************************************

DELAY       $MACRO
            TBLR    DUMMY
            $ENDM


BEGRX       $MACRO
            DELAY
            BIOZ PRENDI?    ;  if BIO = 1 esce
            B         VAI?
            DINT
PRENDI?     IN    DUMMY,PA5    ; else fa una lettura -> BIO :=1
            EINT
VAI?
            $ENDM


RECEIV      $MACRO  L
            DELAY
CICLA?      BIOZ        RICEVI?        ; if BIO = 1 cicla
            B           CICLA?
            DINT
RICEVI?     IN      :L:,PA5       ; else legge il dato
            EINT
            $ENDM


*********************************************************************
*     Sezione Driver di interruzione della INT1 :                   *
*      Interprete   di comandi.                                                *
*  in program memory dopo il vettore di interruzione  mappato       *
*********************************************************************

       .SECT    "INT1"
* All'interruzione Int1 deve far seguito l'invio di un comando,
* opportunamente interpretato.
* I primi due Comandi (0 e 1) sono quelli di  Note_Off  e di Note_On
* che hanno lo scopo rispettivamente di disattivare e di attivare
* una nota. Il Comando 2 serve per assegnare un valore ad una delle
* variabili locali delle voci.
* Le funzionalità sono espandibili mediante l'introduzione di nuovi
* comandi (al massimo 128).
* Il codice è interrompibile nella maggior parte del suo
* svolgimento. Poiché a sua volta può aver interrotto il
* processo in background, effettua un salvataggio dello stato prima
* di modificarlo.


INTERPRETE: SST       STATOAG0    ; Salva i registrI di stato
            SST1      STATOAG1; in una loc. di B2
            LDPK      TEMP
            SACL      ACCL_AGG; salva l'accumulatore
            SACH      ACCH_AGG
```

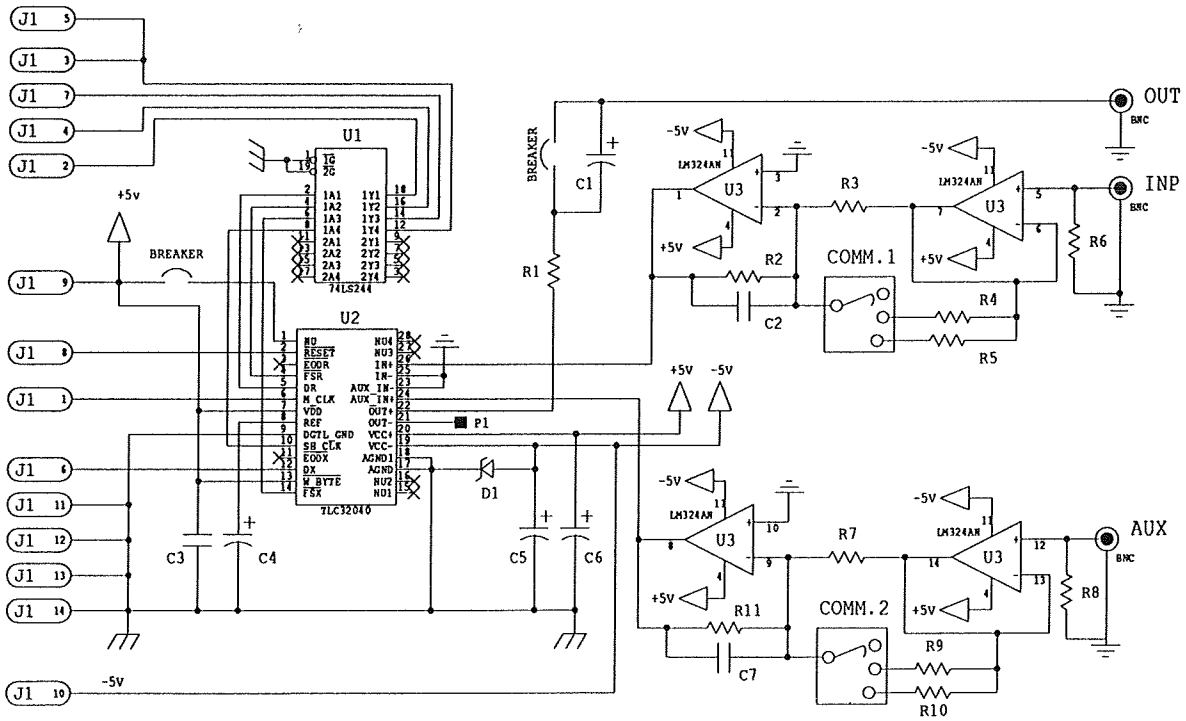
```
            LDPK    IMR
            LAC     IMR
            ANDK    0FFFDH          ; maschera int1
            SACL    IMR
            LDPK    TEMP
            POPD    IND_RITORNO
            EINT                ; ABILITA interruzioni
            SAR     AR0, RARAGG     ; salva AR0 prima di sporcarlo

            BEGRX
RIFARE:     LDPK    TEMP
            RECEIV TEMP         ; DataMem[TEMP] <- (codice comando)
            LAC      TEMP
            ANDK    007FH
            SFL                 ; ACC <- 2 * (codice comando)
            ADLK    INIZIO_TRAT ; ACC = ACC + INIZIO_TRAT
            BACC                ; PC <- ACC

INIZIO_TRAT: B      NOTE_OFF    ;   comando = 0
            B       NOTE_ON     ;   comando = 1
            B       SET_VAR     ;   comando = 2
            .SPACE  22 * (2*16)
            B       RITORNA     ;   comando =25 (per uscire dalla INT1)


RITORNA:    LAR     AR0, RARAGG ; ripristina  reg. ausiliario  sporcato

            DINT
            LDPK    IMR
            LAC     IMR
            ORK     02H             ;  smaschera INT1
            SACL    IMR
            LDPK    TEMP
            LAC     ACCH_AGG,16       ; ripristina l'accumulatore
            ADDS    ACCL_AGG
            PSHD    IND_RITORNO
            LDPK    0
            LST1    STATOAG1        ; ripristina registro di stato ST1
            LST     STATOAG0       ; ripristina registro di stato ST0

            EINT
            RET


; ************************  NOTE_ON  *******************************

NOTE_ON:    LAC     TEMP,  9
            SACH    TEMP
            LAC     TEMP
            ANDK    0003H         ;  ACC  <-   numero voce (0..3)
            SACL    DUMMY
            ZAC
            SC                    ;  setta bit di carry
            RPT     DUMMY
            ROL
            SACL    UTI
            OR      NOTE
            SACL    NOTE            ; mette 1 nel BITvoce(0..3) di NOTE
            LAC     UTI, 4
            OR      NOTE
            SACL    NOTE        ; mette 1 nel BITvoce(0..3)+4 di NOTE
```

```
                LAC     DUMMY
                ADLK    VELO1       ; ACC <-indirizzo di VELOvoce (1..4)
                SACL    DUMMY
                LARP    0
                LAR     AR0, DUMMY      ;  AR0  <-  indirizzo di VELOvoce
                LAC     TEMP, 6
                ANDK    7F00H   ;ACC <- Velocity(nei bit più significativi)
                SACL    *
                ADRK    4     ; AR0  <-  indirizzo di STEPvoce

                RECEIV  TEMP        ;  riceve lo step
                LAC     TEMP
                SACL    *
                B       RITORNA     ; esce subito



;*****************************    NOTE_OFF    ***********************

NOTE_OFF:       LAC     TEMP,  9
                SACH    TEMP
                LAC     TEMP
                ANDK    0003H         ;  ACC  <-   numero voce  (0..3)
                SACL    DUMMY
                ZAC
                SC                    ;  setta bit di carry
                RPT     DUMMY
                ROL
                SACL    UTI
                CMPL
                AND     NOTE
                SACL    NOTE          ; mette 0 nel BITvoce(0..3)   di NOTE
                LAC     UTI, 4
                CMPL
                AND     NOTE
                SACL    NOTE        ; mette 0 nel BITvoce(0..3)+4 di NOTE
                LAC     DUMMY
                ADLK    STEP1      ; ACC <- indirizzo di STEPvoce (1 .. 4)
                SACL    DUMMY
                LARP    0
                LAR     AR0, DUMMY    ;  AR0  <-  indirizzo di STEPvoce

                RECEIV  TEMP     ;  riceve lo step
                LAC     TEMP
                SACL    *
                B       RITORNA         ; esce subito

;*****************************    SET_VAR  ****************************

SET_VAR:        LAC     TEMP,  9
                SACH    TEMP
                LAC     TEMP
                ANDK    00FFH ; ACC <-- indirizzo relativo della variabile
                ADLK    200H   ; ACC <-- indirizzo assoluto della variabile
                SACL    UTI
                LARP    0
                LAR     AR0, UTI  ; AR0 <-- indirizzo della variabile

                RECEIV  TEMP  ; TEMP <-- valore da assegnare alla variabile
                LAC     TEMP
```

```
            SACL    *
            B       RITORNA          ; esce subito

*************************************************************************
*     Sezione Drivers di interruzione :  interruzioni  XINT e TINT    *
*************************************************************************

      .SECT   "TX_SAVE"

*;  IL Processo per la generazione del campione da inviare
*;  all'interfaccia seriale ogni periodo di campionamento,
*;  è attivato dall'interruzione di fine trasmissione (XINT).
*;  Questo processo ha la priorità sul processo in background
*;  e su quello di interpretazione comandi, perciò  contrariamente
*;  a questi, esegue completamente ad interruzioni disabilitate
*;  Quindi per prima cosa deve salvare il contesto.
*;  Poi esegue il calcolo del campione in base ai risultati
*;  dell'elaborazione precedente, al dato ricevuto dalla seriale
*;  e alla configurazione della catena



TX:   SST0    STATO0      ; salva i registri di stato in due locazioni
      SST1    STATO1      ; nel blocco B2 , a pagina 0
      LDPK    TMP
      SACL    RACCL       ; salva l'accumulatore ...
      SACH    RACCH
      SAR     AR0, RAR0SYN;  i registri AR0 ed AR1
      SAR     AR1, RAR1SYN    ;  usati dai processi interrotti   ...
      SPH     RPH               ;  il registro P  ...
      SPL     RPL
      LACK    1
      SACL    TMP
      MPY     TMP
      SPL     RT                ; ed il registro T .
      SSXM                    ; setta il bit per l'estensione di segno
      SPM     1               ; setta lo shift per l'uscita del reg. P
      LAC     RIS1
      ADD     RIS2
      ADD     RIS3
      ADD     RIS4
      SFR
      SFR                     ;  diviso 4
      SACL    TMP             ; TMP <-- (RIS1+RIS2+RIS3+RIS4)/4
      LT      VOLPROPRIO
      MPY     TMP             ; reg. P <-- TMP * VOLPROPRIO
      ZAC
      LT      VOLRICEVUTO
      LDPK    DRR
      MPYA    DRR      ; ACC<--P(con 1 bit di segno), P<--DRR*VOLRICEVUTO
      APAC              ; ACC <-- ACC + reg. P (con 1 bit di segno)
      LDPK    TMP
      SACH    TMP      ; TMP <-- risultato a 16 bit con 1 bit di segno
      LAC     TMP
      ANDK    0FFFCH     ; quantizzazione a 14 bit del risultato
      LDPK    DXR
      SACL    DXR          ; invio al modulo successivo
      SPM     0          ; ripristina 0 shift per l'uscita del reg. P
```

```
      .SECT "TX_RIPR"

*;  Al termine del processo di calcolo degli algoritmi si deve eseguire
*;  il decremento del clock per la sincronizzazione del processo in
*;  background e il ripristino dello stato interrotto


      LDPK  CLOCK
      LAC   CLOCK
      SUBK  1
      SACL  CLOCK
      LAR   AR0, RAR0SYN      ; ripristino reg. ausiliari
      LAR   AR1, RAR1SYN
      LT    RPL                 ; ripristino P register..
      MPYK  1
      LPH   RPH
      LT    RT                  ; ..  T register
      LAC   RACCH,16          ; .. accumulatore
      ADDS  RACCL
      LDPK  0
      LST1  STATO1          ; e registri di stato
      LST   STATO0
      EINT                ; si ritorna al processo interrotto
      RET               ; riabilitando le interruzioni



      .SECT "TIMER"
*;   IL Processo di Timer fa solo una scrittura fittizia sulla porta 7
*;    ogni 5 millisecondi che  ha come effetto  quello di provocare una
*;   richiesta di interruzione al PC.
*;   L'interruzione può essere utilizzata dal PC per sincronizzare
*;   sequenze di eventi musicali
*;   L' invio della richiesta di interruzione può subire un ritardo
*;    dovuto al processo di generazione dei campioni(non interrompibile),
*;   al massimo di un periodo di campionamento,   quindi assolutamente
*;   trascurabile.


TIMER:      OUT     TMP,PA7   ; scrittura fittizia : manda IRQ a PC
            EINT              ; si ritorna al processo interrotto
            RET               ; riabilitando le interruzioni



*************************************************************************
*    Sezione   .Text :     inizializzazioni   ( in prog memory a 1400)    *
*************************************************************************

          .TEXT
          DINT
          SSXM
          LDPK       NOTE
          LAC        NOTE
          BNZ        SCH_1
          BZ         SCH_N
```

```
SCH_1:      LACK        2BH      ; abilitate XINT, TINT,  INT1, INT0
            LDPK        IMR
            SACL        IMR
        B       VAI
SCH_N:  LACK    23H             ; abilitate XINT,  INT1, INT0
        LDPK    IMR
        SACL    IMR
VAI:    ZAC
        SACL    DXR             ; inizializza la seriale..
        LDPK    RIS1            ; .. e le variabili di sistema
        SACL    RIS1
        SACL    RIS2
        SACL    RIS3
        SACL    RIS4
        SACL    STEP1
        SACL    STEP2
        SACL    STEP3
        SACL    STEP4
        SACL    VELO1
        SACL    VELO2
        SACL    VELO3
        SACL    VELO4
        SACL    NOTE
        LACK    LIMCLOCK        ; carica il clock con LIMCLOCK
        SACL    CLOCK
        SPM     0


*************************************************************************
*  Sezione BackGround : in prog memory dopo la sezione .TEXT            *
*************************************************************************

*; E' la sezione per la generazione degli inviluppi o per qualsiasi
*; altro processo da eseguire in background rispetto agli altri.
*; Viene attivato ogni msec.


LIMCLOCK    .SET    20    ; valore per il clock di sincronizzazione
                ;  20 * 50 µsec.(periodo campionamento) = 1 msec.


*************************************************************************
      .SECT    "BACKGR_I"
*; Intestazione per gli algoritmi definiti dall'utente

          EINT      ; abilitazione delle interruzioni
BACKGR:

*************************************************************************
      .SECT    "BACKGR_F"
;* Al termine del processo di background si attende che il periodo
;* di 1 msec. sia terminato (CLOCK = 0). Quindi si esegue nuovamente
;* il processo.

ATTESA: LDPK   CLOCK
        LAC    CLOCK
        BGZ    ATTESA       ; if CLOCK > 0 va ad ATTESA
        LACK   LIMCLOCK     ; else ricarica clock al valore LIMCLOCK
        SACL   CLOCK
        B      BACKGR ; riesegue il processo

       .END
```


J1 5
J1 3
J1 7
J1 4
J1 2
J1 9
J1 8
J1 1
J1 6
J1 11
J1 12
J1 13
J1 14
J1 10
+5v
BREAKER
U1
74LS244
U2
TLC32040
C1
C2
C3
C4
C5
C6
C7
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
D1
P1
U3
LM324AN
-5V
+5V
COMM. 1
COMM. 2
OUT
INP
AUX
BNC