"Un sistema multi-DSP per la sintesi e l'elaborazione di segnali audio con TMS320C25."

G.Bertini, S.Toni, L.Tarabella

nota interna B4 - 25

Maggio 1989

INDICE

Abstract.

Viene descritto un sistema per l'elaborazione digitale di segnali in banda audio, progettato con il microprocessore TMS320C25 (10mips) della Texas Instruments, da utilizzare prevalentemente in attività di ricerca acustica e musicale.

Il progetto, argomento di una Tesi di Laurea alla Facoltà di Ingegneria di Pisa [0], è stato svolto nell'ambito della collaborazione tra il reparto di Elaborazione di Segnali e Immagini dell' IEI di Pisa e quello di Informatica Musicale del CNUCE, riguardo lo studio di sistemi modulari per l'elaborazione digitale del suono.

L'architettura è composta da un microprocessore *master* che controlla la comunicazione con un Host Computer, con i convertitori D/A ed A/D, ed un vettore di processori *slave*. Ogni processore *slave*, essendo programmabile, agisce come un coprocessore altamente sofisticato; la comunicazione tra processore *master* e il singolo processore *slave* è risolta attraverso l'uso di *ram* di tipo *dual port*. Inoltre i processori *slave* possono comunicare tra loro utilizzando una linea seriale ad anello capace di operare fino a 5Mbit/sec.

Il sistema è stato progettato per permettere l'implementazione della maggior parte degli algoritmi di sintesi e di filtraggio digitale per segnali audio (ad es. Karplus-Strong, FM, etc...), la sperimentazione di tecniche di campionamento, di elaborazione dei segnali reali o sintetizzati e di tecniche miste.

E' stato realizzato un prototipo ed un ambiente di sviluppo, in via di ampliamento, per cercare di verificare le prestazioni dell'architettura proposta e valutare i problemi relativi all'uso dei nuovi componenti VLSI introdotti nel progetto (TMS320C25, Dual Port, PLD, ...).

[1] - Introduzione.

Questo sistema a processori multipli per l'elaborazione digitale di segnali (multi-DSP) per la sua particolare architettura offre all'utente la potenza di calcolo idonea per un numero elevato di complesse elaborazioni in tempo reale di segnali in banda audio oltre ad una agile interfaccia con un Personal Computer (PC). L'apparato è nato dall'esigenza di risolvere problemi tipici del settore dell'Informatica Musicale quali l'implementazione in tempo reale di algoritmi per la sintesi, il filtraggio e l'estrazione dei parametri del segnale. Comunque l'architettura disegnata è tale da adattarsi agilmente anche ad altre applicazioni tipiche del *Digital Signal Processing.*

Il presente lavoro si inserisce nell'ambito della ricerca di un'architettura ottimale per questa classe di algoritmi.

Nel campo della sintesi audio i sistemi disponibili sul mercato offrono prestazioni sempre più sofisticate sia nel trattamento della timbrica sia per le possibilità di "performance" polifonica e di elaborazione del segnale. Questi risultati sono stati conseguiti principalmente con l'introduzione di circuiti integrati *custom.* da parte dei maggiori costruttori del settore, dedicati a svolgere funzioni specifiche come ad esempio la sintesi digitale del suono con algoritmo di modulazione di frequenza o con altre tecniche, unità di riverberazione e di effetti speciali, eccetera. Tali macchine tuttavia non soddisfano le esigenze tipiche della ricerca (acustica, psicoacustica) e della produzione musicale professionale poichè non permettono la completa programmabilità sia per quanto riguarda gli algoritmi di sintesi che di elaborazione dei parametri del segnale. Tali caratteristiche sono riscontrabili solo in alcune macchine speciali molto complesse sviluppate ed utilizzate presso quei pochi laboratori specializzati ed università come IRCAM (Parigi), CCRMA (Stanford Ca,USA), University of Illinois (Urbana, USA), CSC dell'Università di Padova.

Solo in questi ultimi anni la disponibilità di microprocessori per l'elaborazione digitale dei segnali con elevata potenza di calcolo, e di dispositivi VLSI di nuova concezione che permettono di semplificare la progettazione e la realizzazione di apparati, ha permesso di pensare nuove architetture compatte e versatili, che in unione ad un host computer della classe "personal" (PC IBM, McIntosh, MSX, Archimedes, ecc...) costituiscano delle stazioni di lavoro adatte sia alla sperimentazione e alla ricerca in campo audio, sia alla produzione di Computer Music (CM).

Il sistema proposto è stato disegnato dopo un'analisi iniziale dei meccanismi software in tempo reale da implementare in particolar modo per applicazioni di Sintesi di Segnali Audio Polifonica, oltre che per la gestione dell'intero sistema, tenendo presenti anche i risultati ottenuti in precedenti lavori sullo stesso argomento [1]. Una attenta valutazione delle precedenti proposte architetturali ha infatti evidenziato l'importanza di una nuova impostazione progettuale sia a livello hardware che software dell'intero sistema. A tal proposito sono state introdotte e sperimentate nuove tecniche di approccio alla sintesi del segnale audio [2]. La nuova impostazione permette inoltre una agevole realizzazione e manutenzione del software di controllo dell'intero sistema, e nella sua semplicità di fondo, si presta ad evoluzioni e varianti a seconda delle esigenze dell'utente. L'architettura è interamente basata sull'utilizzo del microprocessore TMS320C25 (DSP) a 16/32 bit prodotto dalla Texas Instruments [3] capace di eseguire la maggior parte delle sue istruzioni in un tempo di 100*nsec* per una potenza di calcolo globale di circa 10*mips*. Questo microprocessore è dotato di un set di istruzioni e di una architettura tali da consentirne l'uso anche in applicazioni *general purpose*, caratteristica, questa, non comune ad altri microprocessori per DSP della stessa classe (o anche superiore) progettati per assolvere in maniera molto efficiente poche specifiche funzioni quali ad esempio operazioni di FFT, e di filtraggio [4,5,6]. Pertanto questo microprocessore offre anche una alternativa economica alla progettazione di dispositivi *custom* VLSI. Il TMS320C25 è in grado di gestire 256kbyte di memoria veloce, ma attraverso l'uso di una logica di *wait-state* offre anche la possibiltà di interfacciare dispositivi esterni più lenti; possiede inoltre un timer interno ed una interfaccia seriale con canali di ingresso e di uscita separati capace di operare fino a 5Mbit/sec con varie modalità di trasmissione sincronizzata, e sei sorgenti di interruzioni mascherabili con diversa priorità.

[2] - Architettura del sistema.

L'architettura di base del sistema consiste di una struttura multi-DSP, nella quale è presente un microprocessore *master* ed un *array* di processori *slave*; il sistema è stato concepito per essere controllato da un Personal Computer (Host) sul quale vengono sviluppate applicazioni specifiche. In fig.1 è mostrato lo schema a blocchi dell'intero sistema.

L'unità *master* gestisce tutte le risorse del sistema svolgendo principalmente i seguenti compiti: 1) interfacciamento con l'host; 2) controllo delle unità *slave* di elaborazione; 3) sincronizzazione delle elaborazioni di tutto il sistema; 4)

aquisizione di segnali campionati; 5) invio dei segnali elaborati ai convertitori da digitale ad analogico.

I moduli *slave,* in pratica svolgono la funzione di *coprocessori del master* eseguendo sotto il suo controllo le elaborazioni relative ai programmi caricati. Il collegamento *master - slave* avviene mediante una memoria *dual port.*, particolare componente questo che permette a due dispositivi indipendenti di avere accessi in lettura e scrittura anche simultanei nella stessa memoria, velocizzando e semplificando la comunicazione tra le due unità eliminando i conflitti tipici di un *bus* comune. Con questo tipo di collegamento la potenza di calcolo resa disponibile dalla macchina per le classi di algoritmi che si intendono implementare, è funzione lineare del numero di unità *slave* che si attivano, nei limiti delle possibilità di gestione del *master.*

Tramite i canali seriali del TMS320C25 i moduli *slave* possono poi comunicare tra loro senza l'intervento del *master* permettendo di "spezzare" elaborazioni particolarmente complesse in più fasi ed in cascata. E così se nella normale configurazione parallela, ogni microprocessore, in modo del tutto indipendente dagli altri, esegue elaborazioni complete su un certo numero di campioni, i risultati delle quali vengono poi raccolti dal *master*, invece nella configurazione serie ogni micro elabora una singola fase durante la quale è previsto che riceva i risultati parziali della fase precedente e che produca i dati per la fase successiva (esecuzione in *Pipeline*).

Le applicazioni di interesse prevedono in generale una o piu' temporizzazioni delle loro attività, risolte normalmente con l'uso di meccanismi di interruzione [2]; in questa apparecchiatura le sorgenti comuni di interruzione sono ottenute con un timer programmabile (8254-2) [7] presente sul modulo *master* che si vanno ad aggiungere all'ulteriore possibilità di adoperare il timer interno dei microprocessori della Texas.

Per il collegamento del *master* con l'*host* sono state realizzate due interfacce: la prima è una interfaccia parallela direttamente collegata al bus dell'*host* (*IBM* compatibile [8]) che semplifica il controllo e la messa a punto dell'hardware attraverso uno scambio di dati negli spazi di I/O delle due sezioni; l'altra, realizzata mediante l'UART 68B50 [9], permette il controllo del sistema da un qualunque "personal" dotato di interfaccia seriale standard MIDI (Musical Instruments Digital Interface) a 31,25Kbit/sec; si è preferito in questo caso dotare il master di una interfaccia seriale di questo tipo poichè di facile e più completa gestione software rispetto ad una ottenibile per adattamento con logica esterna dei canali seriali già presenti sullo stesso microprocessore, che

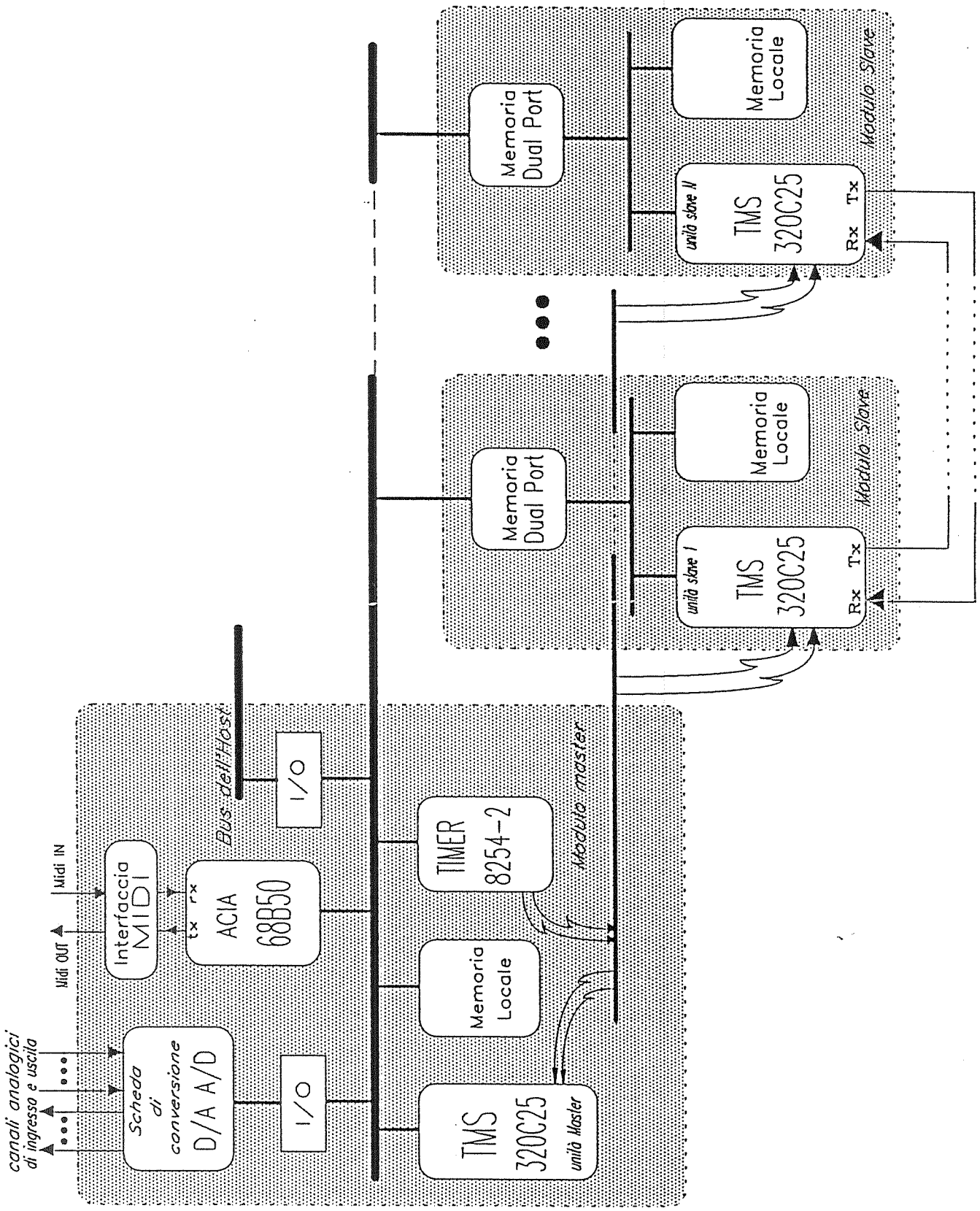


Fig.1 schema a blocchi dell'architettura del sistema

inoltre per quanto riguarda le sezioni slave si intendono adoperare per altri scopi.

Per contenere il consumo dell'intero apparato senza rinunciare alla logica veloce di tipo FAST [10], necessaria per queste velocità di elaborazione, si sono utilizzati per la logica sparsa del sistema, dove possibile, componenti di tipo FAST CMOS (FCT) [11].

È stata completamente realizzata una scheda prototipo che consente lo sviluppo del software, e quindi la valutazione delle prestazioni del sistema e la sperimentazione delle varie configurazioni possibili.

Tale prototipo è costituito dal modulo master ed un modulo slave, capaci di comunicare tra loro sia attraverso il link seriale che tramite le memorie Dual Port. Il sistema viene inizializzato caricando i programmi da eseguire da un PC IBM sul quale è stato implementato "sotto" MS-DOS un ambiente di sviluppo attorno ai pacchetti Macroassembler, Linker, Simulator [12,13] della Texas Instruments basato su utility scritte direttamente in linguaggio Pascal dal quale sono visibili le porte di I/O in cui è allocata l'interfaccia realizzata. Per comodità si è dotata tale interfaccia di microswitch che ne consentono l'allocazione da parte dell'utente nello spazio più idoneo.

[3] Problemi connessi con l'implementazione di algoritmi per la sintesi e l'elaborazione di segnali audio.

Abbiamo precedentemente descritto sommariamente l'architettura dell' apparato disegnato. Per meglio chiarire le funzionalità delle varie parti, è necessaria una breve premessa delle problematiche da risolvere.

Come si è già detto, l'architettura in esame è stata studiata e progettata per soddisfare le esigenze tipiche della sintesi polifonica dei campioni di segnali audio e pertanto faremo ora riferimento alla particolare classe di algoritmi coinvolti in questa applicazione per giustificare la soluzione progettuale implementata. Premettiamo che non è nostra intenzione fare nè una trattazione, nè un riassunto della teoria dell'elaborazione e della sintesi di segnali in banda audio, ma solo portare un esempio completo di una applicazione che metta in rilievo le problematiche e che giustifichi le soluzioni adottate.

È evidente che trattandosi di un'applicazione in tempo reale, un parametro di valutazione delle possibilità della macchina, può essere (per quanto discutibile), il numero di istruzioni per l'elaborazione dei segnali eseguibili nell'unità di tempo che essa mette a disposizione dell'utente. Naturalmente, come si è

precisato, questo è solo un aspetto, poichè anche il formato dei dati trattato (a 16, 24, 32 bit) e le istruzioni disponibili (moltiplicazioni, addizioni, tipi di indirizzamento...) sono certamente altrettanto importanti. Nel campo della sintesi audio esistono dei classici "benchmark" cui si usa fare riferimento per un confronto delle apparecchiature per la sintesi digitale del suono. Uno di questi è il cosìdetto *"oscillatore virtuale"*, che è alla base dell'implementazione della maggior parte degli algoritmi più complessi. Si tratta, in sostanza di un generatore di onda sinusoidale, ottenuta per lettura di una tabella nella quale viene memorizzata la funzione "seno" campionata su un numero di punti pari ad una potenza del due. Esistono in proposito varie tecniche di implementazione dell'algoritmo tese ad ottimizzare sia il tempo dell'esecuzione, sia la lunghezza della tabella (esigenze purtroppo contrastanti). Un semplice metodo per ottenere la generazione della sinusoide a frequenza **F** con buona accuratezza e bassa distorsione **[14]** su un ampio range di frequenze, è quello del *"table look up"*, nel quale si crea una tabella campionando la funzione "sin(x)" in **N** punti presi ad intervalli di 2¶/**N**; quindi si legge la tabella, vista come circolare, con uno STEP costante di scansione (che rappresenta l'incremento di fase della sinusoide) soddisfacendo la condizione STEP<=**N**/2; cioè che si abbiano almeno due campioni per ogni periodo per soddisfare il criterio di Nyquist. Si ha che la frequenza del segnale così generato è pari a **F**=STEP/(**N**x**T**) (Hz); dove **T** è espresso in secondi e rappresenta l'intervallo di campionamento.

In Assembler TMS320 una possibile implementazione di questo metodo è riportata nel seguente listato:

```
OSCILL   $MACRO     FASE, STEP, TAB, AMPLI, RES

* la sinusoide è definita dalla fase, dalla frequenza, dall'ampiezza, dalla
* tabella dei punti; il valore calcolato viene posto nella cella RES

ZALS :  STEP.S:        * Acc:=incremento di fase
ADD     :FASE.S:       * Acc:=Acc+fase vecchia
SACL    :FASE.S:       * Fase:=Fase+Step
LAC     :FASE.S:,9     * calcola la parte intera
SACH    TEMP   * salva
LAC     MSK_1FF        * ottiene i 9 bit del puntatore
AND     TEMP   * alla tabella e ci
ADD     :TAB.S: *  somma l'indrizzo base
TBLR    TABVAL         * Tabval:=sin(Fase)
LT      :AMPLI.S:      * Tr:=ampiezza
MPY     TABVAL         * Pr:=Ampli x sin(Fase)
PAC             * Acc:=PR
```

```
SACH  :RES.S:,1      * RES:=Acc;
$END
```

In totale sono 15 cicli di istruzione cui corrisponde un tempo di esecuzione di 1,5µsec (l'algoritmo non è ottimizzato).

Un secondo "benchmark" fondamentale è rappresentato dal filtro IIR del secondo ordine:

$$Y(n)=A*X(n)+B*Y(n-1)+C*Y(n-2); \quad (eq.1)$$

che per particolari valori dei coefficienti A, B e C può essere utilizzato in condizione di "risuonatore digitale" **[15 par5.7]**. Potremmo implementarlo nel modo che segue:

```
IIR2      $MACRO     XN, YN1, YN2, A, B2, C

* occorrono 6 celle di memoria per memorizzare i valori dei
* coefficienti e per X(N), Y(N-1) ed Y(N-2)

  ZAC             * Acc:=0;
  LT      :XN.S:  * T:=X(N);
  MPY     :A.S:   * Pr:=T*A; [ A*X(N) ]
  LTA     :YN2.S: * Acc:=Pr; T:=Y(N-2);
  MPY     :C.S:   * Pr:=T*C;  [ C*Y(N-2) ]
  LTD     :YN1.S:     * Acc:=Acc+Pr;Y(N-2):=Y(N-1);
                  * T:=Y(N-1);
                  * [Acc:=A*X(N)+C*Y(N-2) ]
  MPLY  B2        * Pr:=B/2*Y(N-1)
  APAC            * Acc:=Acc+B/2*Y(N-1)
  APAC            * Acc:=Acc+B/2*Y(N-1)
  SACH  :YN1.S:,1     * salva in YN1 (uscita del filtro)
  $END
```

che impegna circa un microsecondo di tempo di elaborazione del TMS320C25.

L'oscillatore virtuale viene solitamente rappresentato con uno schema a blocchi come quello di figura 2 (l'operatore $\mathbf{Z^{-1}}$ rappresenta un ritardo di un intervallo di campionamento, cioè il puntatore calcolato per la lettura del campione precedente) che noi compatteremo come illustrato sempre nella stessa figura.

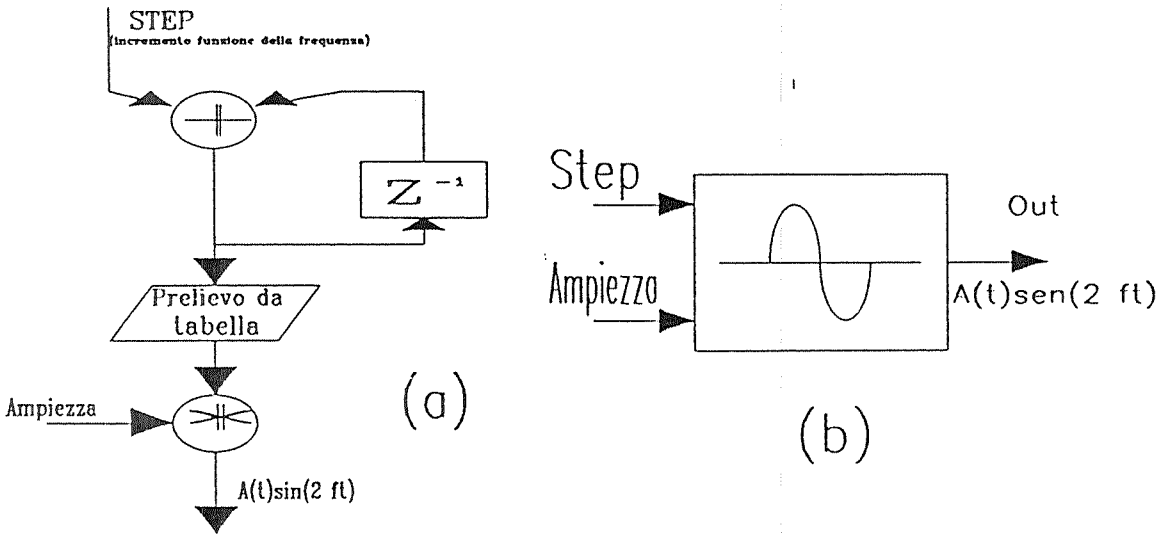


Fig.2 Schema a blocchi dell'oscillatore "virtuale" in due delle possibili rappresentazioni

Con l'uso di questo blocco fondamentale si possono costruire algoritmi molto articolati, e anche se non è nostra intenzione addentrarci nei particolari è bene avere presente almeno un esempio completo di generazione di un suono di buona qualità.

Faremo pertanto riferimento alla generazione del suono con l'algoritmo in modulazione di frequenza (FM) di J.Chowning **[16]** (recentemente adottato da una nota famiglia di sintetizzatori commerciali) che per la sua semplicità e per le notevoli possibilità di soluzioni timbriche, rappresenta un esempio molto interessante e completo. L'algoritmo si basa sulla tecnica della modulazione di frequenza usata nel campo delle telecomunicazioni **[17]**, ma con la sostanziale differenza che nel caso del segnale musicale, le frequenze delle portanti e delle modulanti cadono *entrambe* nella banda acustica. Un esempio di questo metodo è stato riportato e in **[2]**; per avere un'idea della tecnica portiamo l'esempio di una implementazione di un algoritmo per la generazione di un suono S(t) la cui formulazione analitica è la seguente :

$$S(t)=A_p(t)*sen[(at)+A_1(t)*sen(bt)+A_2(t)*sen(ct)+A_3(t)*sen(dt)] \quad (eq.2).$$

Pur non essendo questo algoritmo dei più complessi (si usano normalmente anche 8 generatori sinusoidali variamente interconnessi), comporta già la

generazione di 4 oscillatori virtuali (a frequenza a,b,c,d), ed il controllo di 4 variazioni di ampiezze istantanee (A_p,A_1,A_2,A_3); chiameremo generatore di *inviluppo* il meccanismo che controlla la variazione della ampiezza. Tipicamente il rinnovo del valore degli inviluppi deve avvenire in un tempo dell'ordine di 1msec e con una legge che può essere espressa da una funzione, o da una tabella (come per il "table look up", o per interpolazione lineare).

Nasce dunque l'esigenza di elaborare processi con cadenze temporali diverse, uno alla frequenza di campionamento (da 20 a 40 Khz) l'altro dell'ordine di 1Khz. I due processi vanno quindi sincronizzati su differenti frequenze, ed un metodo possibile per un'implementazione efficiente dell'algoritmo può essere basato sull'utilizzo del meccanismo di interruzione del microprocessore [2]. Inoltre le tabelle dei valori degli inviluppi e dei generatori è bene che siano mantenute distinte, e per problemi di precisione aritmetica [14] non è possibile scendere sotto i 512 valori per tabella. Per il nostro algoritmo questo comporta già un ingombro di 4Kword (considerando separate le tabelle delle funzioni seno, che possono cosí assumere una diversa forma desiderata dall'utente al fine di aggiungere ulteriore contenuto spettrale).

Interessanti passi nella sperimentazione di questi algoritmi vengono introdotti facendo intervenire direttamente nella sintesi, segnali dal mondo esterno aquisiti mediante l'uso di convertitori A/D con almeno 12bit di precisione (S/N>70dB) e successivamente elaborati con tecniche digitali. Ad esempio si può pensare di controllare l'inviluppo di uno dei generatori, con l'inviluppo di un segnale esterno in banda audio (una voce umana, un suono di uno strumento musicale, ecc...), in tal caso è necessario disporre di almeno un canale di aquisizione nel sistema, filtrare digitalmente il segnale con un passa basso opportuno (con f_L dell'ordine di pochi Khz) per prelevarne l'inviluppo con il quale controllare l'ampiezza dell'oscillatore. Infine va prevista la possibilità di aggiungere un trattamento finale del segnale (riverberazione [18], equalizzazione) che porta alla produzione di più segnali "di uscita" da inviare ad un certo numero di convertitori D/A per il controllo di un apparato di amplificazione (ad esempio quadrifonico); ciò al fine di riprodurre anche l'effetto di provenienza spaziale di un sorgente sonora od il suo movimento nello spazio [19] (ad esempio sono possibili tecniche di implementazione dell'effetto Doppler).

Abbiamo così individuato almeno le esigenze fondamentali per il nostro apparato, e cioè: una unità veloce di calcolo con una precisione di almeno 16bit, i convertitori D/A e A/D per l'interfacciamento col "mondo" analogico, delle sorgenti di temporizzazione, e una adeguata capacità di memoria. A

quest'ultimo proposito osserviamo che saranno necessari un minimo di 16k per ospitare programmi e tabelle delle funzioni, mentre per l'implementazione delle linee di ritardo (necessarie per l'implementazione di effetti eco, riverberi e di altro tipo) occorre una quantità di ram che dipende dalla frequenza di campionamento adottata e dal ritardo massimo che si vuole ottenere secondo la semplice relazione:

$$\text{n.° word}_{RAM}=(\text{sample\_freq.})*\text{Delay;} \qquad \text{(eq.3)}$$

dove "sample_freq" va espresso in Hz e "Delay" in secondi. Ad esempio per ottenere un ritardo di 2sec ad una frequenza di campionamento di 30KHz, occorrono circa 60Kword di ram.

Resta infine da tener anche conto dell'esigenza di un agile controllo del sistema, cioè come interfacciare il sistema ad un *host* da cui controllare con agilità i parametri delle elaborazioni.

[4] Il microprocessore adottato.

L'architettura interna del TMS320C25 è di tipo Harvard modificata a singolo accumulatore, in cui la memoria dati e la memoria programma risiedono in spazi di indirizzi separati. Questo permette un completo parallelismo tra fase di fetch di una istruzione e fase di esecuzione **[3 cap.3]**, inoltre il trasferimento dei dati tra i due spazi è tuttavia possibile mediante l'uso di apposite istruzioni. La maggior parte delle istruzioni viene eseguita in soli 100nsec. (ciclo di istruzione), permettendo di disporre di una potenza di calcolo effettiva di circa 10*mips*, se si opera alla frequenza di clock di 40 MHz. Anche l'operazione di moltiplicazione con accumulazione tra operandi a 16 bit (1 word) e risultato su 32 bit, viene eseguita in un solo ciclo. Queste performance sono state raggiunte grazie all'impiego della tecnologia CMOS a 1.8 micron con la quale anche il consumo tipico del chip risulta contenuto a soli 0,5W. Esternamente, gli spazi di memoria progamma e memoria dati sono multiplessati nello stesso bus per minimizzare il numero dei piedini del dispositivo e allo stesso tempo massimizzare lo spazio di indirizzamento che è per entrambi gli spazi, pari a 64kword. Questa capacità di indirizzamento, può ritenersi soddisfacente per le elaborazioni numeriche dei segnali in banda audio senza dover ricorrere ad espansioni della memoria attraverso metodi di paginazione o di generazione

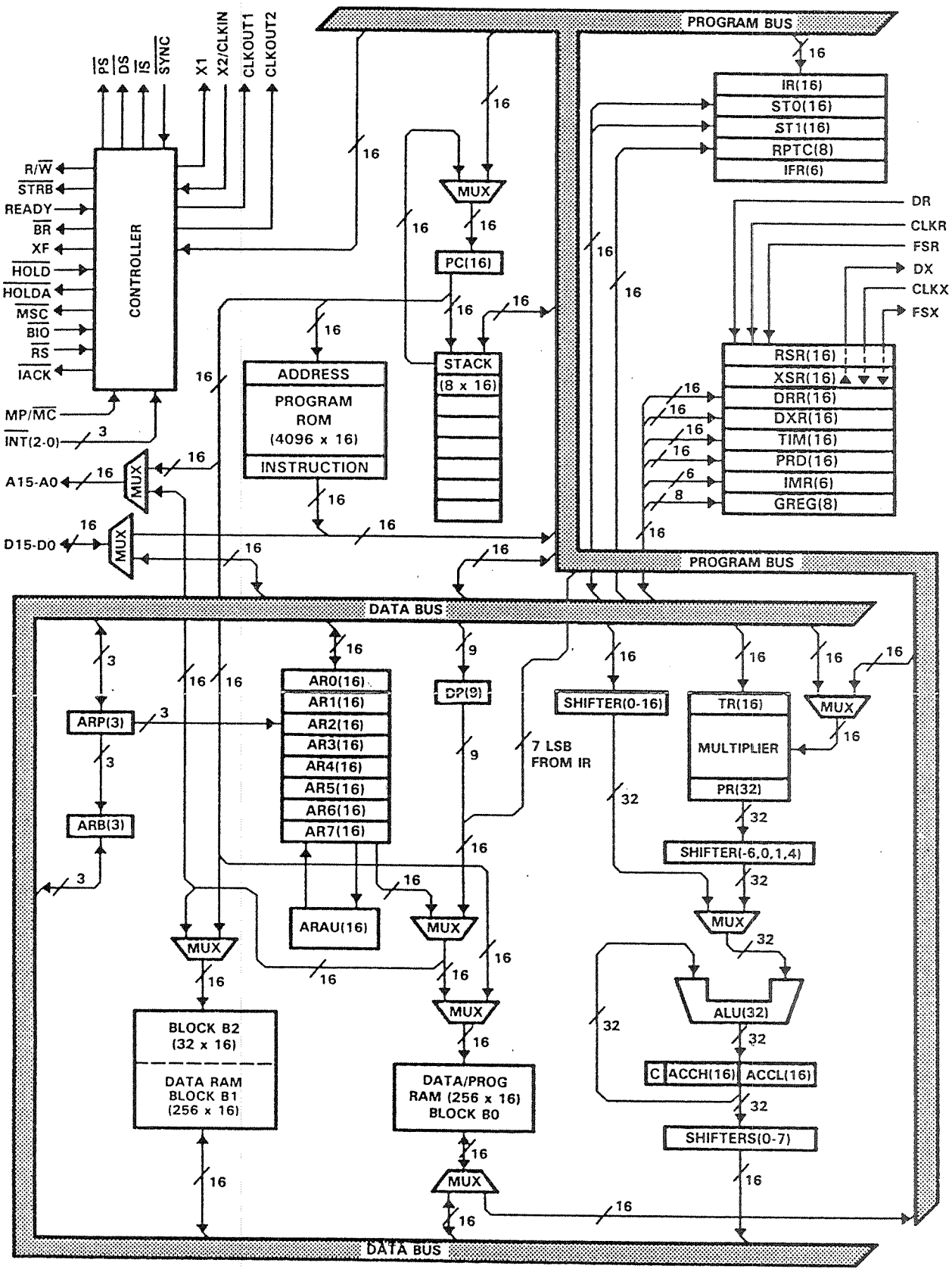


Fig.3 Schema a blocchi dell'architettura del TMS320C25

automatica degli indrizzi ottenibili con dispositivi esterni quali ad esempio le memorie fifo, tipicamente utilizzate nell'acquisizione ed elaborazione di segnali digitalizzati [**20 e 11 cap.14, pag.1/8**]. Internamente l'architettura del TMS mantiene due bus separati per Dati e Programma per ottenere la massima velocità di elaborazone. Il meccanismo di pipelining Fetch-Decode-Execute dell'istruzione è essenzialmente trasparente all'utente salvo in quei casi in cui deve essere interrotto (come per le istruzioni di Branch) e se ne deve tenere conto solo per quanto riguarda la variazione nella durata dell'esecuzione dell' istruzione. Inoltre tale procedimento è stato modificato rispetto a quello del TMS32020 per ottenere una maggior efficienza e per poter veramente eseguire la maggior parte delle istruzioni in un solo ciclo macchina (nel caso in cui si utilizzino memorie esterne opportunamente veloci). È comunque possibile realizzare l'interfacciamento con memorie e periferiche esterne più lente secondo un protocollo del tipo asincrono sincronizzato che rende le istruzioni "multiciclo" ed adattabili ai tempi di risposta dei dispositivi più lenti.

In fig.3 è mostrato il diagramma funzionale a blocchi del TMS320C25, con le principali connessioni logiche interne al microprocessore e tutte le sue linee di interfaccia con il mondo esterno. Viene evidenziato il fatto che tutta l'architettura è costruita attorno a due bus principali: il bus programma ed il bus dati. Il bus programma porta il codice dell'istruzione e gli operandi immediati dalla memoria programma. Il bus dati connette vari elementi alla memoria dati come la CALU (Central Arithmetic Logic Unit) e l'insieme degli otto "auxiliary registers" (AR0-AR7). L'accumulatore a 32 bit è il registro principale della CALU, e riceve il risultato dei calcoli fatti in questa unità anch'essa a 32 bit, costituita da alcuni shifters ed un moltiplicatore. Il caricamento degli operandi dalla memoria all'accumulatore avviene passando da un *barrel shifter*, ossia uno shifter particolarmente efficiente, con il quale si può fare l'operazione di shift aritmetico verso sinistra di un numero di posizioni compreso tra 0 e 15. Anche il trasferimento del contenuto dell'accumulatore verso la memoria avviene attraverso uno shifter che consente traslazioni a sinistra da 0 a 7 posizioni. L'accumulatore possiede un bit di carry che permette una efficiente computazione in aritmetica estesa a differenza del TMS32020 e del TMS32010 che ne erano sprovvisti. Sempre nella CALU è compreso il moltiplicatore, che può operare con segno o senza segno. Uno dei due operandi deve essere caricato preventivamente in un apposito registro (TR) mentre l'altro o è in memoria, o è un immediato a 13 bit. Il risultato è ottenuto nel registro PR a 32 bit e da qui può essere trasferito nell'accumulatore con uno shift di 1,4 o nessuna posizioni a sinistra e di 6 a destra, inoltre si può anche sommare o

sottrarre il nuovo risultato, così shiftato, al precedente contenuto dell'accumulatore. Il dato può essere fornito al moltiplicatore sia dal bus dati che da quello programma e da entrambe le memorie esterna ed interna, il che, in unione ad esempio alla istruzione REPEAT permette di realizzare l'operazione di moltiplicazione ed accumulazione in un solo ciclo di istruzione reperendo entrambi gli operandi simultaneamente sui due bus separati (attraverso le due istruzioni MAC e MACD). L'istruzione REPEAT permette infatti di ripetere un'istruzione un numero fissato di volte ed è particolarmente utile per ottimizzare i tempi di esecuzione di alcuni algoritmi come verrà mostrato nei listati in appendice. Gli otto registri ausiliari (AR0-AR7) servono prevalentemente come indici per l'indirizzamento indiretto, ma si possono anche usare come registri di appoggio. Questi sono connessi ad un'unità dedicata di calcolo (ARAU) che supporta la manipolazione degli indrizzi in parallelo ad altre operazioni e che può servire come unità aritmetica general purpose, poichè i registri ausiliari comunicano direttamente con la memoria dati. Un altro registro, a 9 bit, viene usato per puntare alla pagina di memoria dati corrente (Data Page Pointer, DP) per accedere a questa in maniera diretta anzichè attraverso i registri ausiliari. Questo registro assieme ad altri numerosi bit di stato della CPU sono in realtà parte dei due registri di stato del microprocessore chiamati ST0 ed ST1. Il TMS320C25 offre la possibilità di memorizzare e ricaricare questi due registri, in modo da implementare un completo salvataggio e ripristino dello stato della CPU nei drivers per la gestione delle interruzioni o nella chiamata a subroutines. Il campo ARP compreso in ST0 è il puntatore al registro ausiliario corrente ed è supportato da un buffer (ARB), che fa parte di ST1, essenziale per un corretto ripristino dello stato precedente un'interruzione. Infatti, tutte le volte che si modifica ARP, il valore precedente viene salvato in ARB e quando si ricarica ST1 automaticamente il valore di ARB viene ricopiato in ARP, ripristinando il valore precedente. Si noti inoltre che il caricamento di ST0 mediante l'apposita istruzione LST, non modifica il bit del modo di interrupt (INTM) che controlla l'abilitazione degli interrupt, accessibile solo con le istruzioni DINT ed EINT. Il TMS320C25 ha anche uno stack interno di celle a 16 bit con una profondità di otto livelli che fa parte dell'hardware che implementa il meccanismo di esecuzione delle istruzioni in pipeline **[4 par.3.4]**, ed è in pratica usato principalmente per il salvataggio del program counter (PC) o della parte bassa dell'accumulatore. Ad ogni modo tramite apposite istruzioni è possibile espandere lo stack nella memoria dati.

Il meccanismo di pipelining è di interesse in unione alle configurazioni di memoria possibili, in quanto determina la durata dell'esecuzione di una istruzione. Si noti comunque che lo stack interno non ha uno stack pointer vero e proprio, poichè mettendo nove dati in pila non si ha il cosidetto stack overflow, ma si perde invece la possibilità di recuperare il primo dei nove dati. Se invece si tenta di prendere più dati di quanti ve ne erano stati messi, si continuerà ad ottenere l'ultimo dato correttamente recuperato, ossia il primo che vi si era posto.

Lo spazio di I/O del TMS320C25 consiste di 16 porte di ingresso e di 16 porte di uscita che implementano un'interfaccia di I/O a 16 bit attraverso il bus dei dati del microprocessore. I registri relativi al timer interno e all'interfaccia seriale sono invece mappati in memoria dati.

[5] Interfacce e mappe di memoria.

Entrambe le sezioni Master e Slave sono costruite attorno al TMS320C25 ed alcuni altri componenti di uso comune, pertanto ne descriviamo le caratteristiche nello stesso paragrafo. Per gli schemi elettrici e per la descrizione dettagliata della logica di controllo (che per lo slave è stata integrata in un dispositivo PAL) si rimanda alla Tesi di Laurea già citata [0]. Il collegamento tra le due unità è come già detto ottenuto sia mediante memoria Dual Port sia (nel prototipo) mediante interfaccia seriale.

Nello spazio di memoria dati della sezione master sono allocate anche le memorie dual port per il collegamento con le unità slave di cui riparleremo più avanti. Complessivamente per la sezione del master si ha la mappa di memoria riportata in figura 4, dove il significato del blocco B0 e della data ram interna è già stato chiarito precedentemente. La mappa della memoria della sezione slave si ricava da questa considerando però che la memoria dual port occupa solo lo spazio compreso tra >4000 e >43FF, mentre il rimanente spazio può essere usato liberamente per la memoria dati.

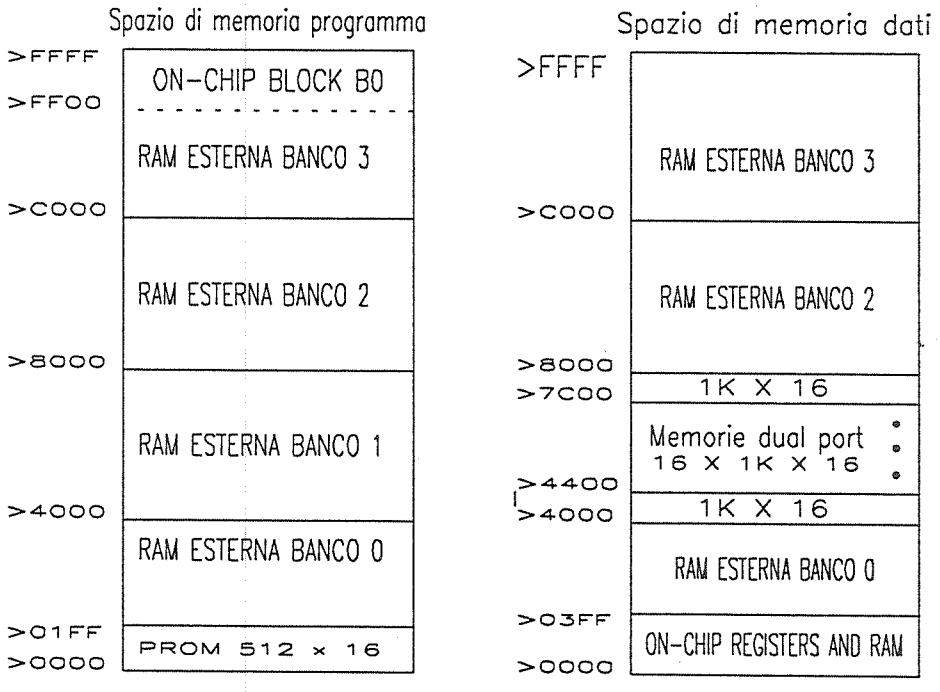


Fig.4 mappa della memoria del modulo master

La rete realizzata per l'introduzione di stati di attesa nel ciclo di istruzione del TMS320C25 è stata realizzata come suggerito dallo stesso manuale del processore. Tale rete permette di gestire fino a dieci spazi di indirizzi, ed associare ad ognuno un dato numero di cicli di attesa. Comunque, se ritenuto necessario in una successiva versione del sistema, il numero degli ingressi può essere aumentato in maniera molto semplice utilizzando al posto delle porte nand a 4 ingressi (74F20) quelle a 8 ingressi (74F30). Nel nostro caso abbiamo tre ingressi per dispositivi che non hanno bisogno di stati di attesa (0 wait-state), tre per dispositivi che necessitano l'introduzione di un ciclo di attesa e altri quattro per ottenere due stati di attesa. La rete è composta da due flip-flop di tipo JK contenuti nell'integrato 74F112 con *clear e preset* e tre porte nand a 4 ingressi (74F20).

Sostanzialmente la funzione della rete è quella di ritardare l'attivazione del segnale di READY verso il TMS di un numero di cicli che dipende dal dispositivo interessato dall'operazione di bus.

Lo spazio di I/O.

Dal punto di vista delle temporizzazioni dei segnali, il TMS320C25 tratta gli accessi alle porte di I/O allo stesso modo della memoria. Lo spazio di I/O viene selezionato mediante la linea di indirizzamento IS, attiva allo stato basso. IS viene attivato all'inizio del ciclo di I/O. Tutti gli altri segnali di controllo ed i parametri di temporizzazione sono gli stessi di quelli visti per interfacciare la memoria dati o programma.

Le istruzioni di I/O del TMS possono accedere 16 porte in lettura e 16 in scrittura (il formato dati è sempre su 16 bit). Queste vengono specificate mediante i quattro bit meno significativi del bus indrizzi, e dal R/W. L'esecuzione di una operazione di ingresso o di uscita prende due cicli di istruzione (uno solo se usata con l'opzione repeat-counter o nella configurazione PI/DI). Queste istruzioni sono la IN e la OUT che possono assumere entrambi i formati di indirizzamento diretto ed indiretto.

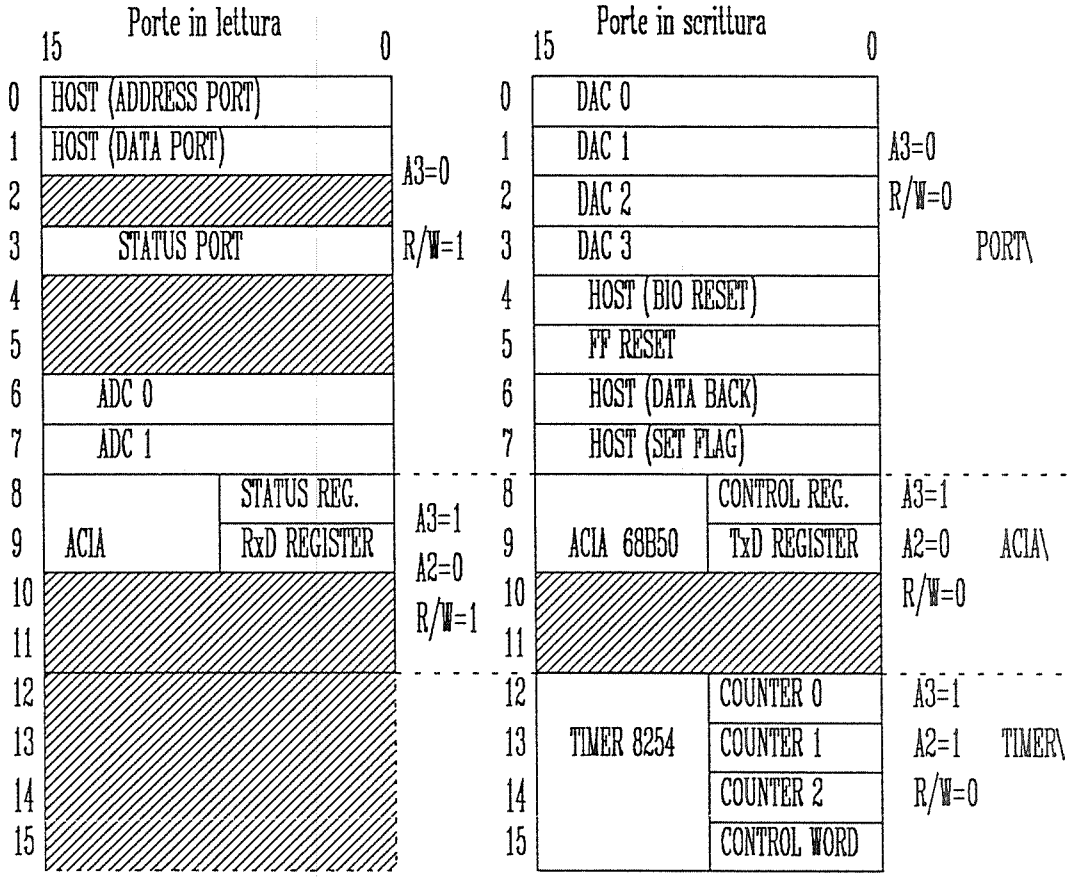


Fig.6 mappa dello spazio di I/O del modulo master

Nel modulo master lo spazio di I/O è utilizzato per l'interfaccia con l'host, il timer 8254, l'ACIA 68B50, l'interfaccia con la sezione analogica, ed alcune porte di controllo e di stato di cui viene spiegata in seguito la funzione. Nei moduli slave non vi sono dispositivi nello spazio di I/O. La mappa completa dello spazio di I/O del modulo master è riportata in fig.6.

Le porte che abbiamo associato allo spazio PORT sono realizzate con logica sparsa (74FCT373, 74F244, 74F74, 74F374) vengono selezionate mediante una rete comprendente principalmente due decodificatori 74FCT138, abilitati uno nel caso di lettura, l'altro nel caso di scrittura. In quest'ultimo caso, è il segnale di scrittura (W\) ottenuto dall'operazione tramite l'OR negato del R/W e dello STRB\ ad abilitare l'integrato 74FCT138.

L'ACIA (68B50) è stato interfacciato con due cicli di wait, il che garantisce una durata minima del ciclo di I/O completo di circa 300nsec. Inoltre poichè il tempo di disabilitazione dell'ACIA può essere dell'ordine di alcune decine di nsec. occorre impiegare un buffer veloce (74F245) sul bus dati per evitare

possibili conflitti nel ciclo successivo. Il buffer viene abilitato dallo stesso segnale ACIA\ e la direzione è controllata direttamente dal R/W.
Per il Timer-Counter 8254 non sono stati necessari adattamenti particolari dei segnali del bus. La corretta durata degll'impulso di scrittura (WR\) è ottenuta tramite l'inserzione di un ciclo di ritardo. Il timer è accessibile solo in scrittura poichè in questa configurazione del sistema non si è ritenuto utile poter leggere il contenuto dei contatori interni, e in tal modo si evitano problemi di conflitto dovuti alla lentezza della disabilitazione dei buffer di uscita nel caso di una lettura.

Dal punto di vista software l'ACIA è costituita da 4 registri, sovrapposti due in scrittura e due in lettura; lo schema logico del componente ed il significato dei bit dei registri interni vengono riportati in fig.7. I registri in sola lettura sono quelli di STATO e quello di ricezione dati. Quelli in sola scrittura sono quello di CONTROLLO e quello di trasmissione dati. È importante notare che non sono possibili due accessi consecutivi ai registri dell'ACIA; tra un accesso e il successivo si deve osservare un'attesa di almeno 500nsec come è mostrato neli relativi diagrammi di temporizzazione [9].

L'inizializzazione dell'ACIA si ottiene mediante un "master reset" scrivendo un "1" nei bit CR0 e CR1. Quindi si passa a settare il registro di controllo specificando il divisore del clock di trasmissione (CR0, CR1), il formato di trasmissione (CR2, CR3, CR4), e abilitando o meno la generazione di interrupt in ricezione e trasmissione (CR5, CR6, CR7). A questo punto si può iniziare la trasmissione con le modalità programmate per la quale si rimanda ai listati in appendice.

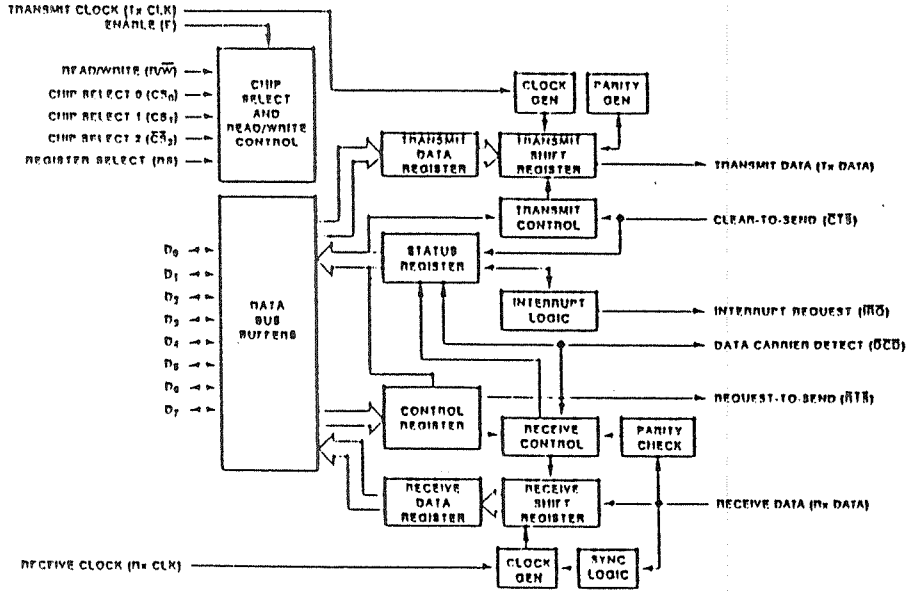


Fig.7 schema a blocchi del 68B50

L'8254 è un Timer-Counter programmabile organizzato con tre contatori a 16 bit indipendenti, ognuno con una velocità di conteggio fino a 8MHz (10MHz nella versione più veloce: 8254-2). Dispone di ben sei modalità operative programmabili via software ed i suoi registri di conteggio sono accessibili in lettura e scrittura. Lo schema a blocchi del dispositivo è riportato in figura 8, assieme al significato dei bit del registro di controllo del timer (Control Word) ed alle forme d'onda che interessano le modalità utilizzate.

Nel nostro sistema si usa questo componente per generare segnali di interruzione (SAMPLE, ed ENV) verso i TMS e per generare il baud rate programmabile (SCLK) per le linee seriali "incorporate" nei microprocessori. E' stato utilizzato il 74LS90 per dividere per 10 la frequenza dell'oscillatore "ibrido" TTL a 8Mhz (OSC) per il clock CLK2 del contatore 2 (OSC/10), mentre agli altri due contatori viene direttamente mandato il clock a 8Mhz (CLK0, CLK1).

Gli ingressi GATE di tutti e tre i contatori sono stati messi al livello logico alto per non inibirne il funzionamento. Come per l'ACIA, anche per l'8254 si ha un "command recovery time" da rispettare nella programmazione che in questo caso è di 200nsec [7].

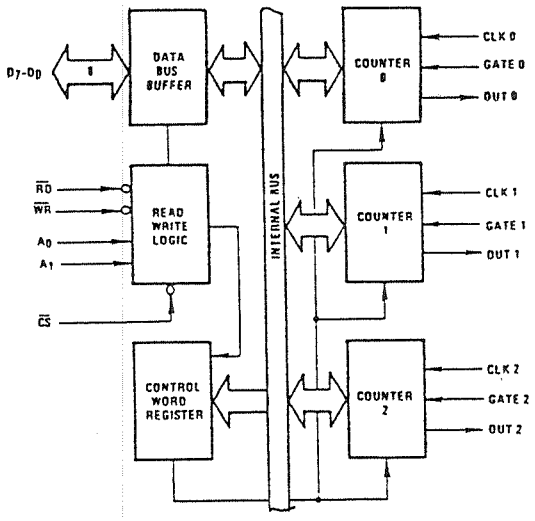


Fig.8 Schema a blocchi del Timer 8254

I contatori 1 e 2 vengono infatti usati nella modalità "rate generator" (MODE_2), ossia in modo da generare un impulso "in negativo", della durata

di un periodo del clock di ingresso, tutte le volte che il conteggio ha termine senza bisogno di dover ricaricare il contatore via software. Pertanto con il registro 1 si possono ottenere interruzioni (SAMPLE) con frequenza compresa tra 122,1 hz e 4Mhz, mentre col registro 2 (ENV) tra 12,21 hz e 400Khz. Il registro 0 è usato in modalità "square wave mode" (MODE_3) e per generare un baud rate (SCLK) simmetrico deve essere inizializzato con un numero pari. Si può ottenere un baud rate di 4Mbit/sec., 2Mbit/sec, eccetera.

La programmazione dei contatori avviene in due fasi successive: la prima che consiste nello scrivere nel registro di controllo quale contatore si vuole programmare (SC0, SC1), con quale modalità (M0, M1, M2) e con quale aritmetica deve essere eseguito il conteggio (BCD); la seconda consiste nell'inviare i due byte (LSB, MSB) corrispondenti al valore da scrivere nel registro di conteggio indicato nella Control Word, rispettando la sequenza che si è specificata nei bit RW1 ed RW0 della stessa Control Word. Per ulteriori chiarimenti si possono consultare i listati riportati in appendice.

Diamo ora le specifiche per l'interfacciamento con la scheda di aquisizione e conversione che si intende realizzare per questo sistema ma che al momento non è ancora implementata.

Abbiamo visto nella figura 6 quali sono le porte riservate per l'interfacciamento con i dispositivi di conversione A/D e D/A. Comunque, come appare dalla stessa figura sono ancora a disposizione altri spazi qualora queste locazioni fossero insufficienti per il numero dei dispositivi da interfacciare.

Dal TMS320C25 i convertitori, sia in ingresso che in uscita, vengono "visti" come semplici registri in lettura (nel caso degli A/D) o scrittura (nel caso dei D/A), senza porte di controllo o di stato associate.

Si intende implementare un interfaccia per la quale ci sia da attenersi solo al seguente tipo di convenzione : per i convertitori D/A (A/D) non si deve inviare un dato prima che sia passato il tempo necessario per la conversione del dato "precedentemente inviato" ("attivata in seguito alla precedente lettura").

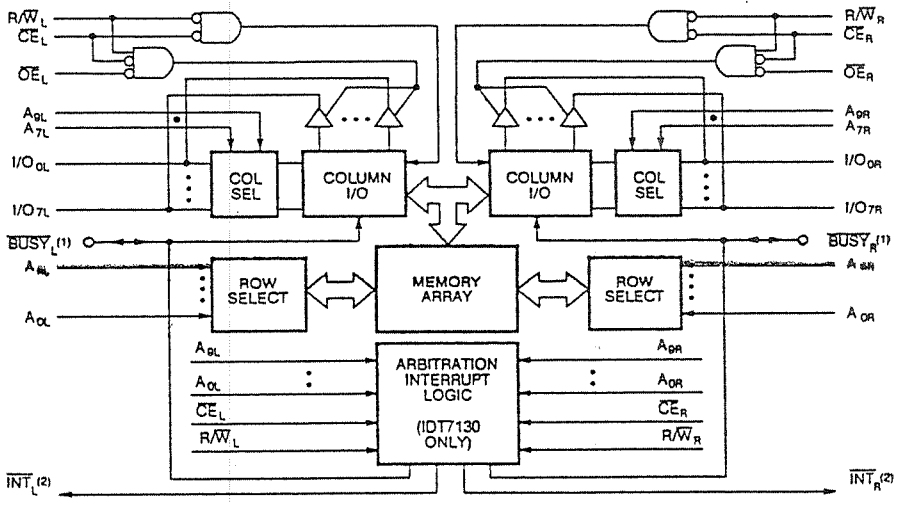
In particolare per le applicazioni in banda audio possono essere utilizzati i convertitori D/A PCM54 a 16bit prodotti dalla Burr Brown che presentano un tempo di conversione di circa 3µsec [22]. Il dato a 16 bit viene convertito semplicemente applicando la stringa di bit da convertire agli ingressi dei piedini D0-D15. Pertanto l'interfaccia è semplicemente costituita da una coppia di registri "positive D-edge triggered", per ogni convertitore PCM54 che si vuole impiegare.Maggiori dettagli sulla rete, completa dei i filtri passa basso

necessari per eliminare le ripetizioni spettrali del segnale all'uscita dal convertitore, possono essere trovati in [1].

Come convertitore A/D può essere utilizzato per esempio l'ADC 80H a 12 bit, anch'esso prodotto dalla Burr Brown [22], che presenta un tempo di conversione minimo di 16μsec se viene pilotato da una sorgente di clock esterna.

Le memorie dual port.

Le ram statiche di tipo dual port utilizzate sono le IDT7130 ed IDT7140 da 1k x 8 bit, prodotte con tecnologia CEMOS (Complementary Enhanced MOS) dalla Integrated Device Technology, Inc. **[11 cap.5,14]**. L'uso di questi dispositivi semplifica l'interfacciamento tra due unità che funzionano in modo indipendente poichè permettono accessi in lettura e scrittura simultanei nella stessa memoria. Infatti sono dotate di due bus completi e distinti (indrizzi, dati, e controllo) internamente arbitrati per permettere l'accesso allo stesso insieme di celle di memoria senza conflitto. L'integrato utilizzato è contenuto in un package DIP a 48 piedini di cui riportiamo lo schema a blocchi in figura 9.



NOTES:

1. IDT7130 (MASTER): $\overline{BUSY}$ is open drain output and requires pullup resistor.
   IDT7140 (SLAVE): $\overline{BUSY}$ is input.
2. Open drain output: requires pullup resistor.

Fig.9 schema logico della memoria dual port

La versione utilizzata nel progetto possiede anche una logica di controllo per facilitare la segnalazione tra due microprocessori tramite due linee particolari chiamate INTR (per la porta destra) ed INTL (per la porta sinistra); queste vengono gestite da un meccanismo hardware/software molto semplice : le due celle fittizie di memoria agli indrizzi più alti della RAM servono per il controllo di queste due linee, da ambo "i lati" (o porte) della memoria. Se la CPU del lato sinistro scrive nella cella >3FF attiva la linea INTR verso la CPU di destra. Questa linea viene disattivata quando la CPU di destra legge nella cella >3FF. Analogamente la CPU di destra può attivare la linea INTL verso la CPU di sinistra scrivendo nella cella >3FE, mentre una lettura della CPU di sinistra nella stessa cella disattiva la linea corrispondente.

La logica di arbitraggio sopra citata risolve il caso "critico" in cui si hanno accessi contemporanei in scrittura/lettura sulla stessa cella di memoria (vedi **[0]** e **[11]** per ulteriori dettagli sul problema). Le funzioni che essa svolge sono essenzialmente tre : 1) fornisce un segnale di occupato (linea di BUSY) alla porta arrivata per ultima a selezionare la stesa cella di memoria; 2) inibisce la scrittura da parte della porta che ha avuto la peggio nella richiesta di accesso alla memoria; 3) prende una decisione (casuale) in favore di una delle due porte quando gli indrizzi arrivano nello stesso istante. In quest'ultimo caso l'arbitraggio è casuale.

Il tempo massimo per la "soluzione" di arbitraggio si ha nel caso di conflitto, e si riflette sul tempo di assestamento della linea di BUSY (TBAC), va da 35nsec per la versione più veloce fino a 55nsec per la versione a 100nsec. La porta che vede attivarsi la propria linea di occupato deve attendere fino a che la porta che ha vinto l'accesso non terminerà l'operazione in memoria. Solo allora comincerà di diritto il suo accesso. La logica esterna del bus deve quindi poter "trattare" la linea di BUSY se si vuole rendere trasparente al programmatore la gestione degli accessi alla memoria dual port.

Le memorie dual port sono state usate per formare dei banchi "accessibili alla parola" da 1k x 16 bit, per ottenere questo formato occorre impiegare una memoria di tipo master (IDT7130) ed una di tipo slave (IDT7140, che non possiede la logica di arbitraggio interna), e collegare le linee di BUSY di destra e di sinistra come mostrato in figura 10 (le resistenze di pull-up sono dovute all'uscita di tipo open-drain). Infatti non solo non avrebbe significato adoperare due logiche di arbitraggio per la stessa cella di memoria, ma sarebbe anche scorretto, come è facile capire se si pensa al caso di arbitraggio casuale discusso precedentemente.

Le memorie utilizzate sono nella versione con tempi di accesso in lettura e scrittura di 100nsec, e risultano disponibili a basso costo permettendo così di contenere il costo globale del sistema. Di conseguenza però è necessario introdurre un ciclo di attesa nel ciclo di accesso alla memoria dual port.

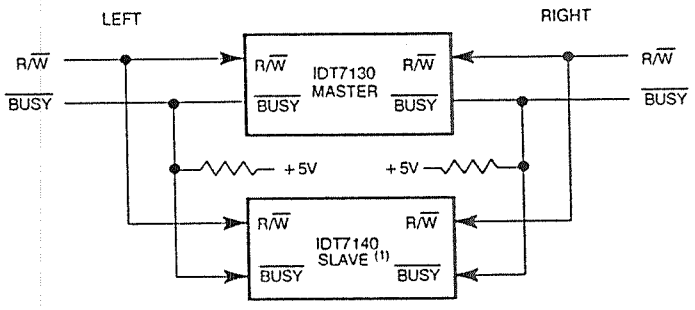


Fig.10 Collegamento delle dual port master (IDT7130) e slave (IDT7140)

Una delle due linee di INTR del lato collegato al TMS320C25 slave viene collegata al piedino BIO del microprocessore. Sull'altro lato invece la linea INTL viene portata alla porta di stato del master. In questo modo si è aggiunta la possibilità di implementare un handshake tra master e slave tramite le due linee di interrupt della dual port usate come flag testabili via software. Ad esempio si può utilizzare questo flag per segnalare la consistenza dei dati che una porta vuole comunicare all'altra. Inoltre questo collegamento viene sfruttato per ottenere un semplice colloquio in fase di inizializzazione dello slave.

Si noti infine che la logica di controllo dell'unità slave, è stata integrata in una PAL 16L8 la cui descrizione dettagliata è riportata tra i listati.

[6] Ambiente di sviluppo.

Accenniamo brevemente ad alcune caratteristiche del pacchetto di sviluppo per il TMS320C25 prodotto dalla Texas Instruments che è stato ampiamente sfruttato durante la fase di scrittura di simulazione e di "debugging" del software. Esso comprende un Macro Assemblatore, un Link Editor ed un Simulatore molto versatili. L'assemblatore può produrre codici oggetto rilocabili dal Linker od in formato assoluto ed è fornito di un macro linguaggio sofisticato (assemblaggio condizionale con espansione delle Macro dipendentemente dai parametri passati o da i loro attributi, eccetera) e di un altrettanto ampio insieme di utili direttive. Non ritenendo utile elencare qui tutte le possibilità offerte dai programmi si rimanda per una approfondita trattazione ai manuali di sistema **[3, 12]** dove possono ritrovarsi eventuali tavole esplicative della notazione dell'assembler TMS320 che può agevolare la comprensione dei listati peraltro facilitata dall'uso di Macro per migliorarne la leggibilità. Le variabili usate e definite nelle Macro sono *strettamente locali,* ossia sono disponibili solo per la Macro che le definisce; i parametri della macro possono essere specificati usando i cosìdetti *qualificatori* (S, A, V, L), che permettono di riferirsi ad un particolare attributo del parametro. Ad esempio dato il parametro X, avremo che X.S indica la rappresentazione simbolica del parametro, ovvero la stringa del nome del parametro: 'X'; X.V indica di sostituire il valore numerico corrispondente al nome del parametro; X.L indica la lunghezza della stringa del nome del parametro attuale. Dai listati riportati piu' avanti si può avere qualche esempio dei vari casi possibili di impiego degli attributi.

In figura 11 infine è visibile lo schema a blocchi dell'ambiente di sviluppo per il prototipo nel quale sono compresi i programmi di caricamento e gestione realizzati, precedentemente descritti, il simulatore e per completezza vi è indicato anche l'emulatore del TMS320C25; in effetti quest'ultimo non è disponibile attualmente nel laboratorio dell'IEI. In considerazione dell'elevato costo è in fase di valutazione in base ad indrizzi di progetti futuri, l'eventualità di un suo acquisto. Comunque è stato usato occasionalmente nell'ambito di una collaborazione con la società SIM di Roma portando la scheda "sotto test" presso il laboratorio Texas, sede di Milano.

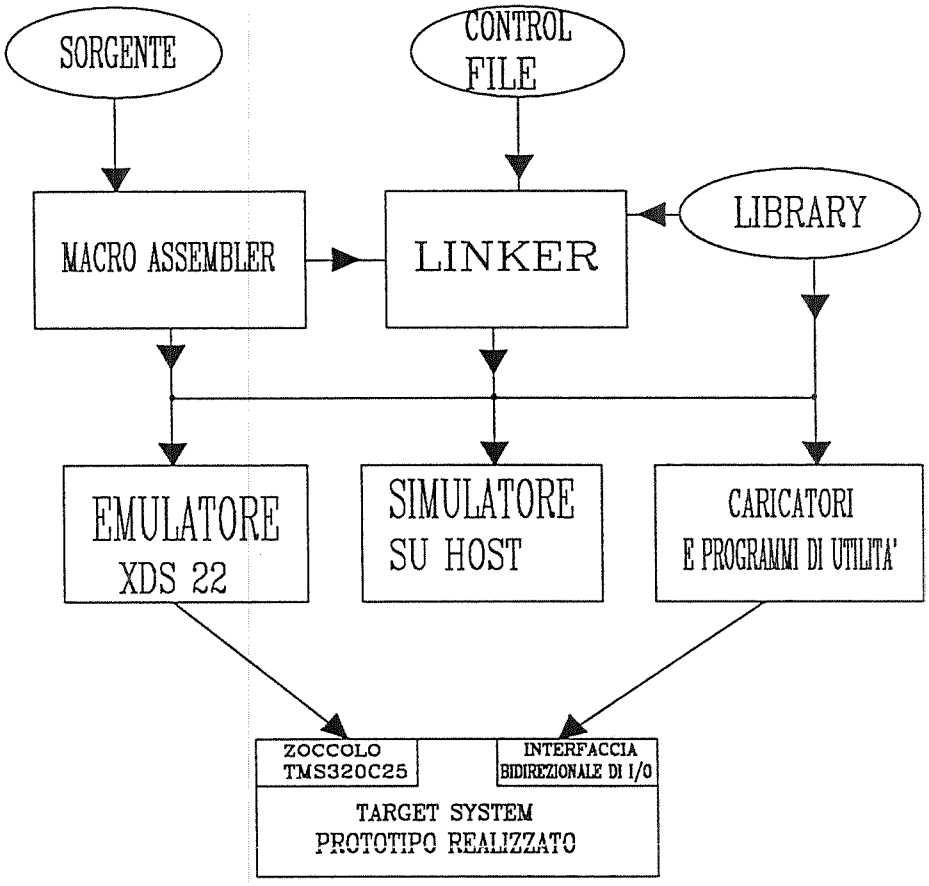


Fig.11 Schema a blocchi dell'ambiente di sviluppo per il prototipo

L'emulatore utilizzato porta la sigla XDS/22 ed è prodotto dalla stessa Texas Instruments per il TMS32020, compatibile hardware con il TMS320C25, non essendo ancora disponibile la versione per il secondo modello. L'emulatore permette l'emulazione completa in tempo reale del microprocessore, permette di operare sia in modo "debug", cioè ispezionando e modificando il contenuto della memoria e dei registri interni ed esterni ed eseguendo le istruzioni "step by step", che in modo "running", con introduzione di "breakpoint" sul verificarsi di particolari condizioni. E' provvisto di un disassemblatore che facilita il debugging. Il sistema è collegato all'host per mezzo di una interfaccia RS232 e da questo gestibile con apposito pacchetto software, che tra l'altro permette di sfruttare i file prodotti dal pacchetto Macroassembler - Linker precedentemente citato.

La piena disponibilità dell'emulatore si rivela essenziale nel debugging dell'hardware, specie nella fase iniziale, dove facilita l'individuazione degli errori di connessione del "wire wrap", la verifica delle temporizzazioni del bus

e delle abilitazioni dei vari componenti senza bisogno di programmi di inizializzazione in ROM.

Un altro strumento utilizzato per il test della scheda è il logic analyzer. Il tipo a disposizione è della Tektronix, un vecchio modello basato su un oscilloscopio 7633 ed un cassetto logic analyzer tipo 7D01 con soli 16 canali. E' chiaro invece che per la messa a punto di schede con microprocessori a 16 bit occorrono logic analyzer con molti più canali. In effetti i logic analyzer presenti in commercio hanno anche altre utili prestazioni. Fra tutte citiamo solo quella di poter avere la rappresentazione delle configurazioni di bit nelle linee sotto test, addirittura in forma disassemblata (mnemonica) per i microprocessori più popolari.

Comunque, una volta ottenuto il corretto funzionamento dell'hardware e dell'interfacciamento con l'host, è stato organizzato il sistema di caricamento dei programmi mediante i moduli descritti piu' avanti. In questa fase lo strumento più importante per la messa a punto del software è senza dubbio il simulatore contenuto nel pacchetto della Texas, che fornisce una simulazione completa di tutte le funzioni della CPU.

Simulatore

La figura 12 descrive le funzioni dei principali comandi del simulatore. Si possono associare stati di attesa differenti agli spazi di memoria dati, programma e di I/O per effettuare un accurato controllo della durata delle routine di calcolo dipendentemente dalla configurazione della propria scheda. Le interruzioni possono essere simulate per generazione a cadenza periodica, e si può provare anche il funzionamento del timer e dell'interfaccia seriale del TMS. Ad ogni dispositivo di I/O può essere associato un file ASCII su disco, il che ha consentito di simulare completamente ad esempio i programmi di bootstrap. Si possono preparare dei file contenenti sequenze di comandi di uso frequente e richiamarli per l'esecuzione dal simulatore onde accelerare le sessioni di simulazione.

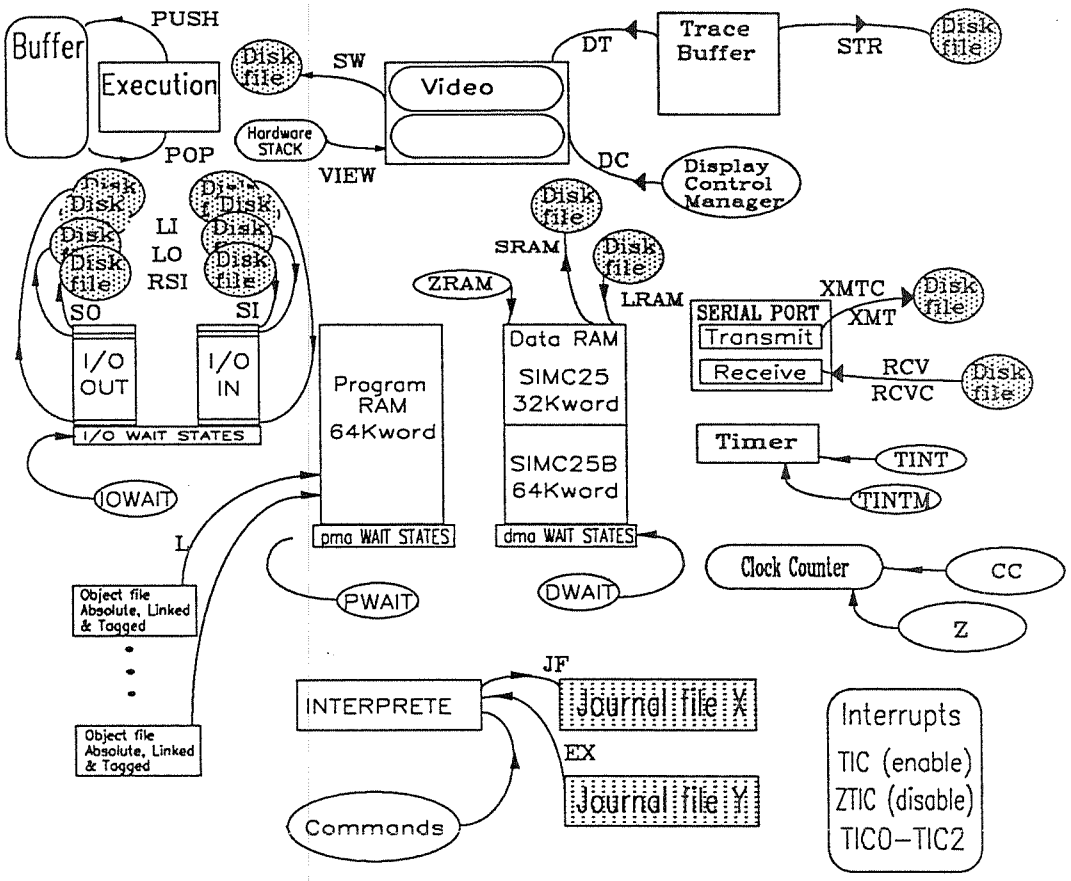


Fig.12 schema delle funzioni offerte dal simulatore del TMS320C25

In particolare l'uso del simulatore ha permesso la rilevazione di errori tipici quali la mancata inizializzazione di variabili, il mancato controllo sui loop, e shift indesiderati nel caricamento dei dati nell'accumulatore o dal registro PR in Acc. Capita infatti, data la particolare struttura dell'Assembler TMS320, che il programmatore abituato all'Assembler dei microprocessori general purpose, commetta grossolani errori di semantica, dovuti alla particolare architettura di questi dispositivi per DSP.

Note sullo sviluppo degli schemi elettrici.

L'uso di un appropriato software per la progettazione dei circuiti elettrici è importante non solo per le ovvie facilitazioni di stampa e di disegno che ne derivano, ma soprattutto per la disponibilità di strumenti integrati al programma di "editing" dello schema elettrico, che consentono di "aiutare" il progettista in tutte le fasi dello sviluppo del progetto, e cioè di disegno, simulazione e layout su circuito stampato.

Per il disegno degli schemi elettrici si è adoperato l'editor grafico SDT (Schematic Design Tools) della OrCAD System Corporation corredato di varie utility per il controllo delle connessioni elettriche, per la produzione di netlist, eccetera. Il disegno dello schema della piastra del prototipo, è stato suddiviso in un insieme di fogli di lavoro strutturati in forma "gerarchica", che semplifica e velocizza la fase di "editing" al terminale. Così, partendo da un unico foglio di lavoro, chiamato "project root", è possibile accedere agilmente a tutte le parti dello schema "visitando" una sorta di struttura "ad albero" in cui ad esempio, è possibile associare ad ogni nodo (che è un foglio di lavoro) una precisa funzione. I segnali possono essere esportati od importati dal nodo, attraverso l'uso delle "module ports", ovvero di "freccette" la cui direzione ha solo lo scopo di indicare l'esportazione o importazione dal foglio, da non confondere con la direzione logica del segnale. Nello stesso foglio si possono dichiarare connessioni implicite assegnando nomi uguali a segnali uguali.

Le regole di visibilità dei segnali nella struttura ricordano da vicino quelle dei linguaggi di programmazione strutturata ad alto livello, per cui il foglio piu' interno "vede" solo i segnali esterni che vengono importati con le "module ports", ed i segnali dichiarati localmente al foglio non sono visibili all'esterno.

Lo schema a blocchi che si riporta in figura 13 è significativo e riassuntivo delle varie fasi di editing e di correzione del progetto elettronico dell'apparato.

Facciamo notare ad esempio l'importanza di disporre di una "netlist" dello schema disegnato, risiede soprattutto nelle possibilità di "interfacciamento" con altri CAD per circuiti elettrici o con programmi specializzati nella simulazione delle reti digitali ed analogiche, oltre a risultare utili come ulteriore verifica delle connessioni tra i componenti, e per la fase di filatura. Dal programma "Netlist" si possono infatti ottenere file in diversi formati; ad esempio è possibile produrre netlist in formato PDIF (P-CAD Database Interchange Format), che sono esportabili all'ambiente PCB (Printed Circuit Board) del Cad prodotto dalla P-CAD, ed attualmente in uso in IEI, attraverso i programmi di interfaccia PDIFIN, PDIFOUT, per passare alla fase di layout.

I disegni degli schemi elettrici sono riuniti al completo in [0] dove sono descritti i dettagli realizzativi della scheda prototipo.

Molti dei componenti utilizzati nel progetto sono di recente produzione, pertanto non essendo disponibili nelle pur ampie librerie di sistema sono stati definiti e compilati nella fase iniziale del disegno e sono ora disponibili nel reparto.

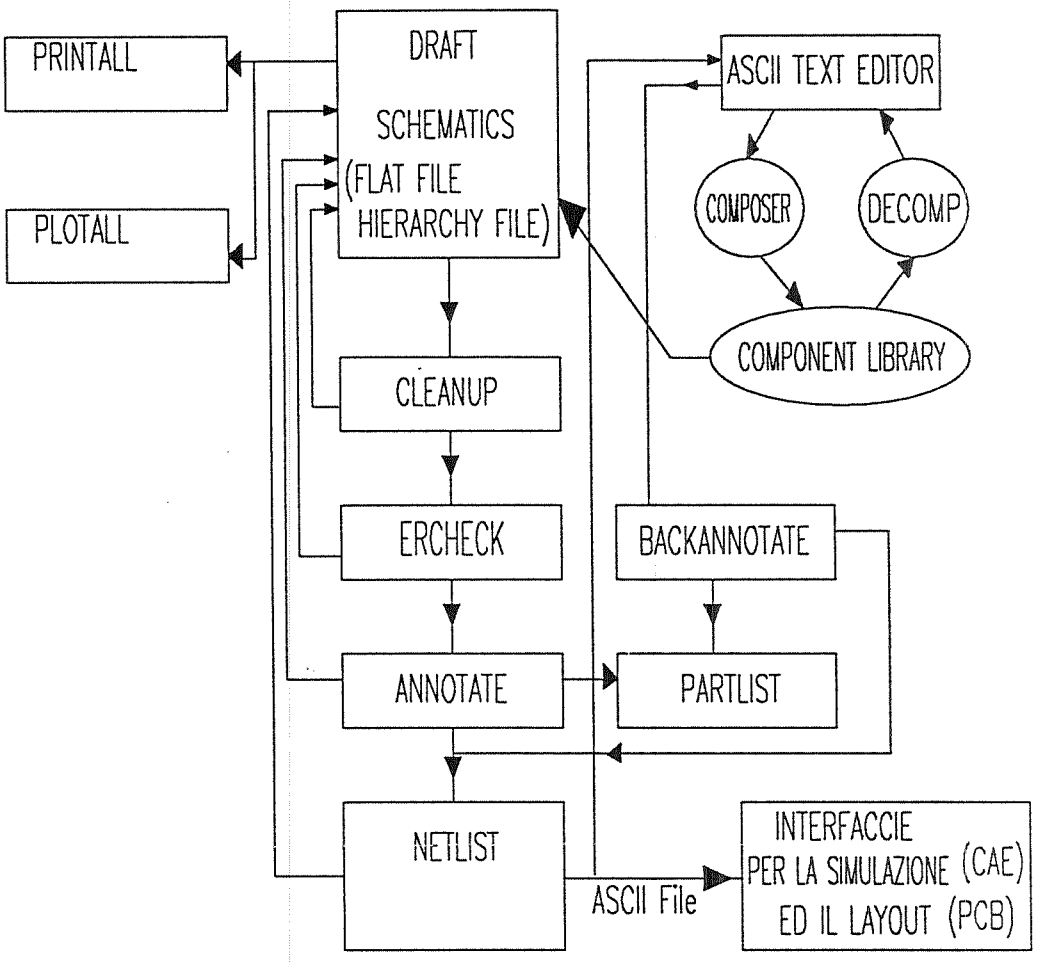


Fig.13 schema a blocchi delle fasi di sviluppo del disegno

Una citazione particolare va fatta a proposito dell'implementazione delle reti logiche del modulo slave.

Implementazione PAL16L8B

I vantaggi dell'uso dei PLD sono molti anche in fase di realizzazione di un prototipo, e non solo nella realizzazione di un layout di dimensioni contenute. Infatti il PLD permette di realizzare una funzione logica, a prescindere dal grado di complessità, con un tempo di ritardo massimo assicurato (che è quello specificato dal costruttore), inoltre riducendo il numero dei chip presenti sulla scheda, cala anche la probabilità di errori di filatura, ed aumenta la velocità dell'esecuzione dei test di controllo.

Le reti logiche del modulo slave sono state integrate in una PAL combinatoria 16L8B [26]. La compilazione del file contenente le equazioni logiche e la nomenclatura dei pin di ingresso ed uscita del dispositivo è stata

effettuata con il compilatore logico CUPL (della P-CAD Systems Inc.) [24] ed i relativi listings sono riportati più avanti. Questo sofisticato pacchetto software è dedicato allo sviluppo e l'integrazione di reti combinatorie e sequenziali in dispositivi PLD (Programmable Logic Devices); la rete da integrare può essere descritta in un linguaggio che presenta le caratteristiche tipiche dei linguaggi ad alto livello.

Il CUPL fornisce anche la possibilità di simulare la rete "compilata" prima di passare alla "bruciatura" del dispositivo, e di inserire automaticamente i "vettori di test" nel file da inviare al programmatore che può essere prodotto in standard JEDEC, compatibile con la maggior parte dei dispositivi attualmente presenti sul mercato. Dall'ambiente CAE del P-CAD è possibile anche passare automaticamente dallo schema disegnato del circuito alla compilazione del dispositivo PLD equivalente.

Attualmente i dispositivi programmabili sono già disponibili con tempi di propagazione inferiori ai 15nsec, però realizzati in tecnologia bipolare (come per la 16L8B) e quindi con consumi abbastanza elevati. Tuttavia ci sono promettenti progressi della tecnologia CMOS anche in questo settore, e molto recentemente sono stati annunciati PLD sotto i 20nsec con la caratteristica di essere anche "erasable" come le normali EPROM ("Windowed").

Una PAL è costituita da una matrice programmabile realizzata mediante fusibili. Le funzioni logiche vengono realizzate in base alla proprietà che qualunque rete logica combinatoria può essere rappresentata mediante una *somma di prodotti*. Inoltre si possono trovare integrati nello stesso chip dei registri d-edge triggered che permettono la realizzazione di macchine a stati finiti.

[7] Conclusioni e sviluppi

La soluzione progettuale proposta porta alla realizzazione di un sistema di costo ragionevole in proporzione alle prestazioni ottenibili, completamente programmabile e dall'architettura "aperta" a futuri miglioramenti e potenziamenti. Infatti è possibile, sempre attraverso il collegamento con memorie dual port, l'interfacciamento del master con altri dispositivi che amplino le prestazioni del sistema. Per esempio un ampliamento notevole delle possibilità di utilizzo della scheda, si avrebbe usando la memoria dual port per l'interfacciamento con l'host.

Il completamento dell'Hardware consiste della realizzazione di una particolare sezione di conversione D/A ed A/D la cui realizzazione pratica non rientrava negli scopi di questa prima parte di progetto/valutazione delle possibilita' dell'architettura, essendo assai note le problematiche connesse invece a tale sezione.

Dal punto di vista Software i risultati ottenuti per simulazione delle tecniche di gestione brevemente descritte possono essere ritenuti indicativi per futuri sviluppi della sperimentazione. Inoltre la capacità di memoria programma gestibile dal TMS320C25 permette di realizzare un sistema operativo residente ad esempio su Eprom che consenta di sveltire le varie fasi di inizializzazione e di sviluppo delle applicazioni. Ulteriori considerazioni sul software di comunicazione tra le unità slave e l'unità master, tuttora in fase di completamento verranno illustrate in una nota successiva.

Si fa notare infine che l'apparato nella versione completa pur progettato come sistema di ricerca e sviluppo nel settore audio, è dotato anche di interfaccia MIDI; ciò consente un adeguamento agli standard degli apparecchi commerciali utilizzati ed un eventuale utilizzo del sistema per la produzione di Computer Music; vedere a tale riguardo la proposta di utilizzo contenuta in [27].

[8] Bibliografia.

[0] Progetto e realizzazione di un sistema multi-DSP per la sintesi e l'elaborazione di segnali audio controllata da PC - S.Toni [Tesi di Laurea alla Facoltà di Ingegneria dell'Università di Pisa A.A. 1987/1988

[1] Progetto e Realizzazione di un Sistema a Microprocessori per la Sintesi del Suono - G.Chini, P.Chiodaroli [Tesi di Laurea alla Facoltà di Ingegneria Elettronica dell'Università di Pisa AA.1985/1986]

[2] Tecniche per la Gestione in Tempo Reale del Sintetizzatore Audio Digitale MP3A - G,Bertini, P.Chiodaroli, L.Tarabella, S.Toni. Nota interna IEI B4-64 Dicembre 1988

[3] TMS320C25 User's Guide - Digital Signal Processor Products-Preliminary [Texas Instruments U.S.A. (July 1986)] ed anche TMS320C25 Manual Update [Texas Instruments U.S.A. (1987)]

[4] The Versatility of Digital Signal Processing Chips - A.Aliphas (DSP Associates), J.A.Feldman (Kurzweil Applied Intelligence)

[5] Computer Music Journal, Volume II Number I - Spring 1987; pag.103/106

[6] The ADSP-2100 DSP Microprocessor - IEEE Micro, Dicember 1986

[7] Microprocessor and Pheriperal Handbook [Intel Corporation (1983)] pag.6-150/6-165

[8] AT Technical Reference [International Business Machines Corporation 1984] pag.1-15/29

[9] Fairchild Microprocessor Data Book [FAIRCHILD 1982] pag.5-207/5-217, ed anche: Motorola Microprocessor Handbook [pag.3-494/3-502]

[10] FAST "Fairchild Advanced Shottky TTL" [Fairchild Camera add Instruments Corporation, Digital Products Division (1984)]

[11] High Performance CMOS 1988 Data Book [Integrated Device Technology,Inc. (1988)] cap.4,5,10,14

[12] TMS320 Family Simulator User's Guide [Texas Instruments 1986]

[13] TMS320 Family Development Support Reference Guide [Texas Instruments (1987)]

[14] Precision Digital Sine-Wave Generation with the TMS32010 - D.Garcia [Digital signal Processing - Semiconductor Group Texas Instruments]

[15] Musica Digitale, Sintesi, Analisi e Filtraggio Digitale - L.M.Del Duca [Franco Muzzio Editore, 1987]

[16] The Synthesis of Complex Audio Spectra by Means of Frequency Modulation - J.M.Chowning [CMJ, April 1977]

[17] Lezioni di Comunicazioni Elettriche - M.Mancianti [ETS Editrice, 1985]

[18] Natural Sounding Artificial Reverberation - M.R.Schroeder [Journal of The Audio Engineering Society, Vol.10, N.3]

[19] The Simulation of Moving Sound Sources - J.M.Chowning [CMJ June 1978]

[20] MV61902-61903 (1k x 9 Dipstick and Parity FIFO) - Preliminary Information [Plessey Semiconductors (August 1986)]

[21] Hardware Interfacing to the TMS32020 - J.Borninski, J.Bradley, C.Crowell, D.Garcia [Digital Signal Processing - Semiconductor Group Texas Instruments]

[22] Integrated Circuit Data Book [Burr Brown, Tucson Arizona (1986)] pag.5-64/5-71; 6-180/6-187

[23] OrCAD/SDT, Schematic Design Tools [OrCAD Systems Corporation, September 1986] e P-CAD System Manuals [Personal CAD System Inc. 1987]

[24] CUPL 2.1 "The Universal Compiler for Programmable Logic" - User's Manual [Assisted Technology (Personal CAD Systems Inc.)]

[25] Digital Signal Processing Applications with the TMS320 Family - Digital Signal Processing Semiconductor Group [Texas Instruments 1986] e TMS32010 User's Guide - Digital Signal Processor Products [Texas Instruments U.S.A. (1983)] e TMS32020 User's Guide Preliminary - Digital Signal Processor Products [Texas Instruments U.S.A. (1985)]

[26] Programmable Logic - Handbook/Databook 1986-1987 [Advanced Micro Devices, Inc. (1986)]

[27] Un sistema grafico per progettare e realizzare algoritmi di sintesi e filtri digitali in banda audio - G.Bertini, A.Lombardini, L.Tarabella [rapporto interno CNUCE, c - 88 n.°54]

[9] - Listati.

Vengono di seguito riportati i listati dei programmi di "bootstrap" contenuti nelle PROM della scheda realizzata, i programmi di caricamento dall'host e di inizializzazione del sistema, il cui funzionamento è stato discusso in precedenza. Non si sono invece proposti i numerosi ma banali programmi di test realizzati per provare il prototipo. Viene inoltre per primo riportato il listato della semplice PAL impiegata per la logica sparsa del modulo slave.

```
CUPL  Version 2.11b Serial# 5-00001-577
Copyright (C) 1983,1986 Personal CAD Systems, Inc.
CREATED Mon Jun 06 12:07:11 1988
LISTING FOR LOGIC DESCRIPTION FILE: SLAVE1.pld
 1:      NAME   SLAVE1;
 2:      PARTNO  SLV01;
 3:      REVISION      01;
 4:      DATE   2/6/88;
 5:      DESIGNER      S.TONI;
 6:      COMPANY IEI-CNR;
 7:      LOCATION      NONE;
 8:      ASSEMBLY      SLAVE_CARD;
 9:      FORMAT  J;
10:      DEVICE  P16L8;
11:
12:/*  PAL  per la logica di controllo e di selezione sulla sezione "SLAVE" del
13:   prototipo realizzato
14:*/
15:/*************** inputs *********************/
16:PIN 1 = RW ;
17:PIN 2 = !STRB ;
18:PIN 3 = !PS ;
19:PIN 4 = !DS ;
20:PIN 5 = A15 ;
21:PIN 6 = A14 ;
22:PIN 7 = !MSC ;
23:PIN 8 = !RESET ;
24:PIN 9 = !SWRS2 ;
25:/****************** outputs **************/
26:PIN 12 = !DRAMH ;
27:PIN 13 = !DRAML ;
28:PIN 14 = !OER ;
29:PIN 15 = !PROM ;
30:PIN 16 = !PRAM ;
31:PIN 17 = !RS ;
32:PIN 18 = READY ;
33:PIN 19 = !W ;
34:/*************** EQUAZIONI *************/
35:
36:!W = RW # !STRB ;
37:READY = !MSC # !DS ;
38:!RS = !RESET & !SWRS2 ;
39:!PRAM = !A14 # !PS ;
40:!PROM = A14 # !PS ;
```

```
41:!OER = !RW ;
42:!DRAML = !DS # A15 ;
43:!DRAMH = !DS#!A15;
```

Expanded Product Terms

```
PRAM =>   PS & A14
PROM =>   PS & !A14
OER =>   RW
RS =>   RESET  # SWRS2
DRAMH =>   DS & A15
DRAML =>   DS & !A15
READY =>   MSC & DS
W =>   STRB & !RW
PRAM.oe  =>   1
PROM.oe  =>   1
OER.oe  =>   1
RS.oe  =>   1
DRAMH.oe  =>   1
DRAML.oe  =>   1
READY.oe  =>   1
W.oe  => 1
```

Symbol Table

Pin Pol	Variable Name	Ext	Pin	Type	Pterms Used	Max Pterms	Min Level
!	PRAM	16	V	1	7	1	
!	STRB	2	V	-	-	-	
!	PROM	15	V	1	7	1	
!	MSC 7	V	-	-	-		
!	OER 14	V	1	7	1		
!	DS 4	V	-	-	-		
!	PS 3	V	-	-	-		
!	RS 17	V	2	7	1		
	A14 6	V	-	-	-		
	A15 5	V	-	-	-		
	RW 1	V	-	-	-		
!	DRAMH	12	V	1	7	1	
!	DRAML	13	V	1	7	1	
	READY	18	V	1	7	1	
!	W 19	V	1	7	1		
!	SWRS2	9	V	-	-	-	
!	RESET	8	V	-	-	-	
	PRAM	oe	16	D	1	1	0
	PROM	oe	15	D	1	1	0
	OER	oe	14	D	1	1	0
	RS	oe	17	D	1	1	0
	DRAMH	oe	12	D	1	1	0
	DRAML	oe	13	D	1	1	0
	READY	oe	18	D	1	1	0
	W	oe	19	D	1	1	0

LEGEND F : field D : default variable M : extended node
N : node I : intermediate variable T : function
V : variable X : extended variable U : undefined

*

```
* Programma di bootstrap per il modulo Master
* CPU: TMS320C25                    V.1.1
*    PROM di 512WORD (74S472)  0 Wait State
*-------------------------------------------------------
*
* Questo programma viene eseguito tutte le volte
* che si ha il RESET verso il TMS320C25 del modulo
* Master. Il segnale di Reset (RS\) e' anche l'
* unica sorgente di interrupt NON-MASCHERABILE
* per il TMS320C25.
* Si deve tenere presente, che la ricezione di RS\
* comporta le seguenti inizializzazioni interne al
* processore:
* 1)-tutti gli interrupt vengono DISABILITATI
* settando il bit INTM a "1".
* 2)-tutta la RAM INTERNA al microprocessore viene
* configurata come DATA RAM (CNF=0 nel registro di
* stato ST1, inoltre anche OV=0, XF=1, FO=0, TXM=0).
* 3)-il registro GREG viene azzerato per impostare
* tutta la DATA RAM come memoria locale.
*
* Per altri dettagli non implicati nella stesura di
* questo programma si rimanda al manuale.
*
* L'esecuzione ha inizio dalla locazione 0 della
* PROGRAM MEMORY ESTERNA.
*
* Il bootstrap permette di caricare qualunque
* programma nei 64KWORD di PROGRAM MEMORY ESTERNA.
*
        TITL    'BOOTSTRAP'
        OPTION      XREF
        IDT     'PROM25'
*
*       PORTE DI I/O UTILIZZATE
*
TMS1    EQU     >0      * porta per passare l'INDIRIZZO
TMS2    EQU     >1      * porta per passare il VALORE
BIORES          EQU     >4      * porta di un bit per resettare il
*                               flag di dati validi
BACK    EQU     >6      * porta di ritorno a 16-bit verso
*                               l'HOST
SETFLG          EQU     >7      * porta di un bit per segnalare il
*                               dato all' HOST
TMS3    EQU     >3      * segnale di test per l'HW
*
*       DICHIARAZIONI
*
TABRAM          EQU     >4000   * viene definita una tabella all'
*                               inizio dei 16Kword di memoria prog.
*                               del prototipo, per redirigere gli
*                               interrupt che hanno il vettore
*                               fissato nelle prime celle della PROM.
```

```
STEP   EQU   10      * costante
ID     EQU   25      * identificatore di scheda
*
DRV1   EQU   TABRAM      * per ogni interrupt vengono lasciate
DRV2   EQU   TABRAM+STEP      * 10 parole sufficienti a contenere un certo
DRV3   EQU   TABRAM+(STEP*2)   * numero di istruzioni (ad es.EINT,RET nel
DRV4   EQU   TABRAM+(STEP*3)   * caso non si desideri intraprendere
DRV5   EQU   TABRAM+(STEP*4)   * alcuna azione al ricevimento dell'
DRV6   EQU   TABRAM+(STEP*5)   * interrupt relativo o per eseguire una
DRV7   EQU   TABRAM+(STEP*6)   * breve routine di controllo semaforico)
*
*      LOCAZIONI DELLA DATA RAM UTILIZZATE
*
       AORG   0      * viene utilizzata la data ram interna
*                           in pagina 4, corrispondente al blocco B0
*                     pero' qui viene definito solo l'offset
*                     l' indrizzo completo e' calcolato con DP.
POINT  BSS   1
VALUE  BSS   1
DUMMY  BSS   1
TEMP   BSS   1
*
*      PROGRAMMA PER LA PROM
*
       AORG   >0
*
INIT   B     COLD  * si salta alla locazione dove comincia
*                  il programma di inizializzazione per
*                  lasciare spazio alla tabella dei vettori
*                  di interrupt
*
*      TABELLA DEI VETTORI DI INTERRUPT
*
INT0   B     DRV1  * INTERRUPT ESTERNO 0
INT1   B     DRV2  * INTERRUPT ESTERNO 1
INT2   B     DRV3  * INTERRUPT ESTERNO 2
*
*                  si devono saltare le locazioni 8-23 che
*                  sono riservate per future espansioni
*
       B     INIT  * per avere la prom corretta
       B     INIT  * si occupa tale spazio con
       B     INIT  * questa sequenza di istruzioni
       B     INIT
       B     INIT
       B     INIT
       B     INIT
       B     INIT
*
TINT   B     DRV4  * INT.INTERNO DEL TIMER
RINT   B     DRV5  * INT.INTERNO RICEZIONE SERIALE
XINT   B     DRV6  * INT.INTERNO TRASMISSIONE SERIALE
USER   B     DRV7  * INDRIZZO DELL' ISTRUZIONE TRAP
```

```
*
*       Programma caricatore e Test
*
COLD  ROVM          * disabilita Overflow Mode
      LARP  0       * si utilizza il registro ausilirio AR0
      LDPK  4       * si punta alla prima Word della DATA RAM
*                   INTERNA tramite il registro DP. Tale
*                   locazione e' infatti corrispondente alla
*                   pagina 4 della DATA MEMORY ed e' nel
*                   blocco B0
      OUT   ID,BACK       * scrive l'identificatore di scheda nella
*                   porta di uscita
      OUT   DUMMY,SETFLG  * e segnala all' Host che e' pronto per
*                   colloquiare
WAIT  BIOZ  INPUT * se ci sono dati validi vai a prenderli
      IN    DUMMY,TMS3        * altrimenti genera un segnale per il
      NOP         * test dell' hardware
      IN    DUMMY,TMS3        * [PIN12 74FCT138 U30]
      B     WAIT  * e ritorna ad attendere
*
INPUT IN    POINT,TMS1   * preleva indirizzo
      IN    VALUE,TMS2   * preleva valore
      OUT   VALUE,BACK         * e lo spedisce all' Host per verifica
      OUT   DUMMY,BIORES       * resetta il flag
      LAC   POINT              * carica nell'accumulatore l'indrizzo
      BZ    START    * se l'indrizzo e' zero significa
*                    che si deve saltare ad eseguire il
*                    programma caricato
      TBLW  VALUE              * altrimenti scrive in memoria prog.
*                    il valore al corrispondente indrizzo
      B     WAIT     * e st torna ad attendere un' altra
*                    coppia di valori
STARTLAC    VALUE          * se l'indrizzo e' zero in VALUE e'
*                    contenuto l'indrizzo della routine
      CALA           * a cui saltare e si esegue una CALL
*                    calcolata a tale locazione
      DINT           * se si torna da tale routine con una
      B     INIT     * RET si riesegue il programma di
*                    caricamento ad interruzioni disabilitate.
      END


*  ____________________________________________
* Programma di bootstrap per il modulo Slave
*              CPU: TMS320C25
*   PROM di 512WORD (74S472)  0 Wait State
*  ____________________________________________
*
* Si deve tenere presente, che la ricezione di RS\
* comporta le seguenti inizializzazioni interne al
* processore:
* 1)-tutti gli interrupt vengono DISABILITATI
```

```
* settando il bit INTM a "1".
* 2)-tutta la RAM INTERNA al microprocessore viene
* configurata come DATA RAM (CNF=0 nel registro di
* stato ST1, inoltre anche OV=0, XF=1, FO=0, TXM=0).
* 3)-il registro GREG viene azzerato per impostare
* tutta la DATA RAM come memoria locale.
*
* Per altri dettagli non implicati nella stesura di
* questo programma si rimanda al manuale.
*
* L'esecuzione ha inizio dalla locazione 0 della
* PROGRAM MEMORY ESTERNA.
*
* Il programma permette di caricare codice o dati
* nella PROGRAM MEMORY dello SLAVE, attraverso la
* DUAL PORT RAM. L'Handshake e' realizzato per mezzo
* delle linee INTR ed INTL, rispettivamente collegate
* col BIO dello SLAVE e l'INT1 del MASTER.
*
* Lo SLAVE e' collegato "a DESTRA" il MASTER "a SINISTRA".
*
        TITL   'BOOTSTRAP'
        OPTION      XREF
        IDT    'SLVPRM'
*
*       DICHIARAZIONI
*
TABRAM      EQU   >4000  * viene definita una tabella all'
*                        inizio dei 16Kword di memoria prog.
*                        del prototipo, per redirigere gli
*                        interrupt che hanno il vettore
*                        fissato nelle prime celle della PROM.
STEP   EQU   10        * costante
*
DRV1   EQU   TABRAM       * per ogni interrupt vengono lasciate
DRV2   EQU   TABRAM+STEP  * 10 WORD sufficienti a contenere un certo
DRV3   EQU   TABRAM+(STEP*2) * numero di istruzioni (ad es.EINT,RET nel
DRV4   EQU   TABRAM+(STEP*3) * caso non si desideri intraprendere
DRV5   EQU   TABRAM+(STEP*4) * alcuna azione al ricevimento dell'
DRV6   EQU   TABRAM+(STEP*5) * interrupt relativo o per eseguire una
DRV7   EQU   TABRAM+(STEP*6) * breve routine di controllo semaforico)
*
BASE   EQU   >400A * BUFFER di trasferimento master>slave
BIORES       EQU   >43FF  * lo slave >LEGGENDO< nella cella >43FF
*                        * resetta la linea di BIO (INTR)
UNO    EQU   >1
SIGNAL       EQU   >43FE  * lo slave >SCRIVENDO< nella cella >43FE
*                        * attiva la linea di INTL verso il MASTER
*
*       LOCAZIONI DELLA DATA RAM UTILIZZATE
*
        AORG   0       * viene utilizzata per data ram la memoria
*                        dual port (pagina 128) che corrisponde all'
```

```
*                       indrizzo >4000
*                       (qui viene definito solo l'offset
*                       l' indrizzo completo e' calcolato con DP).
ADDR BSS    1
NUM  BSS    1
*
*       PROGRAMMA PER LA PROM
*
        AORG   >0
*
INIT    B       START* si salta alla locazione dove comincia
*                       il programma di inizializzazione per
*                       lasciare spazio alla tabella dei vettori
*                       di interrupt
*
*       TABELLA DEI VETTORI DI INTERRUPT
*
INT0    B       DRV1  * INTERRUPT ESTERNO 0
INT1    B       DRV2  * INTERRUPT ESTERNO 1
INT2    B       DRV3  * INTERRUPT ESTERNO 2
*
*                       si devono saltare le locazioni 8-23 che
*                       sono riservate per future espansioni
*
        B       INIT  * per avere la prom corretta
        B       INIT  * si occupa tale spazio con
        B       INIT  * questa sequenza di istruzioni
        B       INIT
        B       INIT
        B       INIT
        B       INIT
        B       INIT
*
TINT    B       DRV4  * INT.INTERNO DEL TIMER
RINT    B       DRV5  * INT.INTERNO RICEZIONE SERIALE
XINT    B       DRV6  * INT.INTERNO TRASMISSIONE SERIALE
USER    B       DRV7  * INDRIZZO DELL' ISTRUZIONE TRAP
*
*       INIZIALIZZAZIONE
*
STARTROVM             * disabilita Overflow Mode
        RSXM          * disabilita sign-extension mode
        CNFD          * configura la ram interna come data ram
        LDPK  128     * DP punta alla pagina 129 (Dual Port)
        RXF           * segnala la presenza al MASTER
        NOP
        NOP
LOOP    BIOZ  NEXT
        B     LOOP
*
*       ALGORITMO DI CARICAMENTO BLOCCHI
*
NEXT    ZALS  NUM     * ACC:=dma(>4001);
```

```
        BZ      EXECUT          * if NUM=0 then EXECUTE;
        LARP    AR0     * (ARP):=0;
        LAR     AR0,NUM         * AR0:=NUM;
        SBRK    UNO     * NUM:=NUM-1;
        ZALS    ADDR    * ACC:=dma(>4000)
        LARP    AR1     * (ARP):=1
        LRLK    AR1,BASE        * AR1:=>400A
CICLO   TBLW    *+,AR0* pma(ADDR):=dma(BASE)
*                       * AR1:=AR1+1; {BASE:=BASE+1}
*                       * ARP:=AR0;
        ADDK    UNO     * ADDR:=ADDR+1;
        BANZ    CICLO,*-,AR1 * if AR0=0 then begin
*                       * NUM:=NUM-1;
*                       * ARP:=AR1      end
*                       *       else begin
*                       * NUM:=NUM-1;
*                       * ARP:=AR1;
*                       * ciclo     end;
* e' finito il trasferimento del blocco:
*
        LRLK    AR1,BIORES
        LAC     *       * resetta INTR della DUAL PORT
        LRLK    AR1,SIGNAL
        SACL    *       * attiva l'interrupt verso il MASTER per
*                       * segnalare il rilascio della DUAL PORT
        B       LOOP    * torna ad attendere istruzioni dal MASTER
*
EXECUT  LRLK    AR1,BIORES
        LAC     *       * anche in questo caso bisogna resettare
*                       * la linea di INTR della DUAL PORT
        ZALS    ADDR    * carica l'indrizzo a cui saltare in ACC
        CALA            * salta a tale indrizzo con una CALL
*
        DINT            * un ritorno da tale routine riporta lo SLAVE
        B       LOOP    * in colloquio con il MASTER
*
        END


{
                LOAD.PAS
}
PROGRAM CARICATORE_Per_TMS320;
{
il nome del file da caricare va passato sulla linea di comando.
l'uscita e' fatta scrivendo nelle porte dello spazio di I/O attraverso
le quali si vede la scheda TMS320
}
const
    TMSADDRLO=$280;
    TMSADDRHI=$281;
    TMSDATALO=$282;
    TMSDATAHI=$283;
```

```
    TMSBACKLO=$284;
     TMSBACKHI=$285;
    BIOSET=$286;
    STATUS=$289;
    FLAGRES=$288;

begin
{
```

Nota : il programma non viene riportato perchè, eccettuate alcune modifiche dovute al nuovo hardware impolementato, è sostanzialmente lo stesso programma di caricamento sviluppato per l'MP3A che trovasi in [2].

```
}
end.

* ---------------------------------------------------------
* Routine di "PONTE" per l'unità MASTER   V.1.1
* ---------------------------------------------------------
* questa routine una volta "lanciata" permette di modificare
* la memoria dati direttamente dall'host (master in trasparenza)
* e quindi di colloquiare con lo SLAVE attraverso la Dual Port
*
        TITL   'PONTE'
        OPTION      XREF
        IDT    'BRIDGE'
*
*       PORTE DI I/O UTILIZZATE
*
TMS1  EQU   >0      * porta per passare l'INDIRIZZO
TMS2  EQU   >1      * porta per passare il VALORE
BIORES      EQU   >4      * porta di un bit per resettare il
*                   flag di dati validi
BACK  EQU   >6      * porta di ritorno a 16-bit verso
*                   l'HOST
SETFLG      EQU   >7      * porta di un bit per segnalare il
*                   dato all' HOST
TMS3  EQU   >3      * segnale di test per l'HW
*
*       DICHIARAZIONI
*
STEP  EQU   10      * costante
ID    EQU   25      * identificatore di scheda
UNO   EQU   1
*
*
*       LOCAZIONI DELLA DATA RAM UTILIZZATE
*
        AORG   0    * viene utilizzata la data ram interna
*             in pagina 6, corrispondente al blocco B1
*             pero' qui viene definito solo l'offset
*             l' indrizzo completo e' calcolato con DP.
INDR  BSS    1
VAL   BSS    1
DUMMY       BSS    1
```

```
TEMP  BSS   1
IDEN  BSS   1
*
*       PROGRAMMA
*
        AORG  >4100
*
*
        ROVM          * disabilita Overflow Mode
        LARP  0       * si utilizza il registro ausilirio AR0
        LDPK  6       * si punta alla prima Word della DATA RAM
*                     INTERNA tramite il registro DP.
        LACK  ID
        SACL  IDEN
        OUT   IDEN,BACK    * scrive l'identificatore di scheda nella
        NOP                * porta di uscita
        NOP
        OUT   DUMMY,SETFLG      * e segnala all' Host che e' partito
        NOP
        NOP
*
* inizializza la memoria dual port da >4000 a >43F0 con il valore >0000
*
        LARP  AR1           * arp:=ar1
        LRLK  AR1,>3F0              * ar1:=>3F0 {contatore}
        SBRK  UNO           * ar1:=ar1-1
        ZAC                 * acc:=0
        LARP  AR0           * arp:=ar0
        LRLK  AR0,>4000             * ar0:=>4000
CICLO SACL  *+              * dmem[ar[arp]]:=acc*2^0;
        NOP
        LARP  AR1           * arp:=ar1
        NOP
        BANZ  CICLO,*-,AR0          * if ar1<>0 then begin
*                           ar1:=ar1-1; arp:=ar0;
*                           ciclo else begin
*                           ar1:=ar1-1; arp:=ar0 end;
LOOP  BIOZ  AV
        B     LOOP
*
AV      IN    INDR,TMS1     * preleva indirizzo
        IN    VAL,TMS2      * preleva valore
        ZALS  INDR
        BZ    EXIT          * se l'indrizzo e' zero significa
*                           che si deve tornare al bootstrap
        LAR   AR0,INDR
        LAC   *             * prima di scrivere il nuovo
        SACL  TEMP          * valore nella cella della dual
        OUT   TEMP,BACK             * port, scrive in BACK il vecchio
*                           contenuto
        LAC   VAL
        SACL  *             * ora ci scrive
        OUT   DUMMY,BIORES      * e avvisa l'host che ha finito
```

```
        B       LOOP            * torna al loop
*
EXIT    OUT     DUMMY,BIORES    * uscita dal "ponte"
        OUT     DUMMY,SETFLG    * avvertimento finale
        RET
*
        END


*_____________________________________________________________
*  PROGRAMMAZIONE TIMER/COUNTER 8254 (INTEL) SU MODULO
*  MASTER
*_____________________________________________________________
* Il contatore 1 genera le interruzioni a frequenza di campionamento
* (linea SAMPLE\)
* il contatore 2 genera interruzioni a frequenza di rinnovo del valore
* degli inviluppi (linea ENV\)
* il contatore 0 fa da divisore della frequenza di clock fornita
* dall'oscillatore (linea Osc) per generare il clock di trasmissione
* dell'interfaccia seriale del TMS320C25 (linea SCLK)
*
        TITL    '82C54-12 INIT'
        IDT     'TIMER'
        OPTION          XREF
*
TIME    $MACRO          VAL,REG
        LACK  :VAL.V:
        SACL TIMER
        OUT TIMER,:REG.S:
        $END
*
DELAY           $MACRO
        NOP
        NOP
        NOP
        NOP
        NOP
        $END
*
*       PORTE DI I/O (registri 8254)
*
TCTL    EQU     15
TCN0    EQU     12
TCN1    EQU     13
TCN2    EQU     14
*
        AORG 0

*       variabili

TIMER BSS       1
*
        AORG >4100
```

```
*
        TIME   >74,TCTL

* scrive il byte:
*               01 11 010 0
* nella porta di controllo del timer/counter 8254.
* Si usa infatti il registro di conteggio 1, in modalità RATE GENERATOR
* (MODE 2), e conteggio in BINary.

        DELAY

* perdita di tempo di precauzione dovuta al
* Command Recovery Time del dispositivo

        TIME   44,TCN1

* il contenuto del registro viene inizializzato con >013C
* cioe' 300

        DELAY
*
        TIME   1,TCN1
*
        DELAY
*
        TIME   >B4,TCTL
*
* scrive il byte:
*               10 11 010 0
* nella porta di controllo per programmare il registro di conteggio 2
* con la stessa modalità del precedente

        DELAY
*
        TIME   232,TCN2

* il contenuto del registro viene inizializzato con >03E8
* ossia 1000

        DELAY
*
        TIME   3,TCN2
*
        DELAY
*
        TIME   >36,TCTL

* scrive il byte:
*               00 11 011 0
* nella porta di controllo per programmare il registro 0 in modalità
* SQUARE WAVE (mode 3)
*
        DELAY
```

```
*
        TIME   2,TCN0

* il contenuto del registro viene inizializzato con >0002
*
        DELAY
*
        TIME   0,TCN0
*
        DELAY
*
        RET             * torna al caricatore della prom
*
        END


*
* INIZIALIZZAZIONE  ACIA (68b50) SU MODULO MASTER
*
        TITL   '68B50 INITIALIZE'
        IDT    'ACIA'
        OPTION         XREF
*
SCRIVI          $MACRO       VAL,REG
        LACK  :VAL.V:
        SACL ACIA
        OUT ACIA,:REG.S:
        $END
*
LEGGI $MACRO       MEM,REG
        IN      ACIA,:REG.S:
        $END

*
DELAY           $MACRO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        $END
*
*       PORTE
SCR     EQU    8
RW      EQU    9
BIORES          EQU    4
BACK EQU    6
*
        AORG 0
*
ACIA   BSS     1
DUMMY          BSS     1
```

```
        AORG >4100

        SCRIVI      3,SCR
*
* scrivendo il byte:
*           0 00 000 11
* nel registro di controllo dell'ACIA si fa il "Master Reset" del
* dispositivo
*
        DELAY
*
* è sempre necessario rispettare il "Command Recovery Time" dell'ACIA
* pertanto si introduce un ritardo
*
        SCRIVI  22,SCR
*
* scriviamo il byte:
*           0 00 101 10
* nel registro di controllo, con il quale si hanno i seguenti "settaggi":
* - trasmetti alla freq. del clock di ingresso divisa per 64 (31,25Khz)
* - formato di trasmissione 1 bit di start, un dato a 8 bit, 1 bit di stop
* - interrupt disabilitati
*
        DELAY
*
* trasmissione di prova: viene trasmesso il byte 'A' fino a che
* da Host non viene attivata la linea di BIO per segnalare di tornare
* caricatore
*
LOOP    SCRIVI      >A,RW

        DELAY

TEST    LEGGI ACIA,SCR      * controlla il bit TDRE del registro
*                           stato (Transmit Data Register Empty)
        DELAY
        BIT     ACIA,14             * se è 0 si deve attendere che abbia
*                           terminato la trasmissione prima di
*                           spedire il byte successivo
        BBZ     TEST

        DELAY

        BIOZ    EXIT        * se l'host attiva il BIO si torna al
*                           programma di caricamento
        B       LOOP

EXIT    OUT     DUMMY,BIORES

        DELAY

        NOP
```

```
        END

*_____________________________________________
* "LOOP DI CONTROLLO" del MASTER  v.2.2
*_____________________________________________
*
* Simulazione della gestione delle risorse in una applicazione in tempo reale
* da parte del microprocessore Master (CPU TMS320C25).
*
        TITL    'SIMULAZIONE MASTER'
        IDT     'OS v10'
        OPTION          XREF
*
* Convenzioni:
* 1) i campioni elaborati dallo slave i-esimo si trovano nella relativa
* DUAL PORT RAM a partire dall' indrizzo >0 da sommarsi a BASE+OFFSET*SLAVE[i];
* 2) questi sono disponibili al MASTER dopo un tempo DELTA dall' interrupt
* di sincronismo;
* 3) eventuali variazioni dei parametri degli algoritmi sono possibili
* modificando su controllo da host i valori delle variabili di "interfaccia"
* nelle memorie dual port degli slave
*
* Mappa del sistema :
*
*       PORTE DI I/O
*
*       REGISTRI DI INTERFACCIA E DI CONTROLLO
TMS1    EQU     0       * IN
TMS2    EQU     1       * IN
STATUS          EQU     3       * IN
BIORES          EQU     4       * OUT
INTRES          EQU     5       * OUT
BACK    EQU     6       * OUT
FLAG    EQU     7       * OUT
*       CONVERTITORI PCM54
DAC1    EQU     0       * OUT
DAC2    EQU     1       * OUT
DAC3    EQU     2       * OUT
DAC4    EQU     3       * OUT
*       ACIA 68B50
SCR     EQU     8       * IN/OUT
RW      EQU     9       * IN/OUT
*       TIMER 82C54-12
TCTL    EQU     15      * OUT
TCN0    EQU     12      * OUT
TCN1    EQU     13      * OUT
TCN2    EQU     14      * OUT
*
* SLAVE DEVICES mappati in DATA RAM ogni KWORD a partire da >8000
*
BASE    EQU     >8000
OFFSET          EQU     >400    * offset di 1K
*
```

```
TABRAM      EQU   >4000  * TABELLA DEI VETTORI DI INTERRUPT IN RAM
STEP  EQU   >000A
DRV1  EQU   TABRAM
DRV2  EQU   TABRAM+STEP
DRV3  EQU   TABRAM+(STEP*2)
DRV4  EQU   TABRAM+(STEP*3)
DRV5  EQU   TABRAM+(STEP*4)
DRV6  EQU   TABRAM+(STEP*5)
DRV7  EQU   TABRAM+(STEP*6)
*****************************************************************************
**
* PARAMETRI DELLA SIMULAZIONE
*
SLAVES      EQU   16      *numero di microprocessori slave visti dal master
ELEMEN      EQU   4       *numero di elementi elaborati e restituiti al master
*                         ad ogni intervallo di campionamento
TRUNC       EQU   4       * parte intera arrotondata per eccesso del logaritmo
*                         in base 2 del numero di slaves (troncamento)
*****************************************************************************
**
PAGE6 EQU   0
*
STACK       EQU   >A000
*
* Data Ram usata per le variabili
*
        AORG PAGE6
*
CAMP BSS    ELEMEN      * array per memorizzare elaborazioni intermedie
PARNUM      BSS   1
PARVAL      BSS   1
DUMMY       BSS   1
BUFFER      BSS   1
*

CALC  $MACRO      NUM,LOC,DAC              * MACRO per fare la somma degli
elementi
        LRLK  AR6,BASE+:NUM.V:  * contenuti nei vettori elaborati
        ZAC           * azzera l'accumulatore prima di fare le somme
        RPTK  SLAVES      * con soli 2 fetch si eseguono SLAVES+1 operazioni
        ADD   *0+     * in "FULL-SPEED"
        SHIFT
        SACL  :LOC.S:
        OUT   :LOC.S:,:DAC.S:
        $END
*
SHIFT $MACRO              * MACRO per ottenere uno shift di 4 bit a destra
        RPTK  TRUNC       * con soli 2 fetch
        SFR
        $END
*
        AORG DRV1
*
```

```
        B       INT0    * "ON SAMPLE DO INT0;"
*
* ************************
*    loop di controllo
* ************************

        AORG >4100
*
* Driver di INTerruzione 0 (MAX.PRIORITA')
* - salva lo stato
* - eventuale ricezione dall' Host di nuovi parametri
* - legge i campioni elaborati all' interruzione precedente dagli
*   slave, completa l' elaborazione (in doppia precisione), tronca
*   i valori finali a 16 bit e li invia ai DAC
* - restaura lo stato
* - riabilita l'interruzione e ritorna
*
INT0    LRLK  AR7,STACK
        LARP  AR7     * AR7 e' lo STACK POINTER
        PUSH          * SALVA LO STATO DELLA CPU
        SST1  *-
        SST   *-
        LDPK  6
        BIOZ  INPUT * input parametri da HOST
GO      LRLK  AR0,OFFSET  * tramite AR0 si realizza l'indiciato
        RC                * resetta il bit di carry
        SSXM              * setta l'estensione di segno
        LARP  AR6     * AR6 e' il puntatore alla RAM DUAL PORT
*                           e viene inizializzato in CALC
        CALC  0,CAMP,DAC1
        CALC  1,CAMP+1,DAC2
        CALC  2,CAMP+2,DAC3
        CALC  3,CAMP+3,DAC3

*
        POP           * RESTAURA LO STATO DELLA CPU
        LARP  AR7
        LST   *+
        LST1  *+
        EINT
        RET
INPUT   IN    PARNUM,TMS1       * legge nuovo parametro
        IN    PARVAL,TMS2
        OUT   DUMMY,BIORES
        LAR   1,PARNUM
        LAC   PARVAL
        SACL  *       * lo manda direttamente in DUAL PORT RAM
*                     allo SLAVE e segnala tramite la relativa
*                     cella >3FF collegata al BIO dello stesso
        B     GO
        END
```

```
Program Timer8254;

const
    Mask_Address=$280;
    Timer_Sel=$1E80;  { 7*$400+mask_address }
    Tim_Count_0=Timer_sel;
    Tim_Count_1=$1E81;      { Timer_sel+1; }
    Tim_Count_2=$1E82;      { Timer_sel+2; }
    Tim_Control=$1E83;      { Timer_sel+3; }

{ Il Timer e' mappato al posto della Card_7 }

begin
   clrscr;
      { Il timer viene usato per pilotare il campionamento sulla  scheda di sintesi del suono
basata  sul microprocessore TMS32010  ed usata in fase di simulazione. L'interruzione viene
generata con il contatore 0 usato in  modalita "RATE GENERATOR". Inoltre con il
contatore 2 si generano interruzioni sul  canale IRQ3 del PC alla freq. di 1khz. Questa
sorgente di temporizzazione puo' essere sfruttata per sincronizzare le  applicazioni sul PC. La
risoluzione del conteggio in questo caso si ottiene modificando sia il  contenuto del registro
di conteggio 1 (che genera il clock per il registro 2), che quello  del contatore 2 }

    { La frequenza in ingresso a Tim_Count_0 e Tim_Count_1 e' ricavata dividendo per 2 il
segnale di riferimento del BUS IBM  nominato OSC che e' a 14.31818 MHz  (70nsec. di
periodo) ottenendo  un onda quadra di periodo pari a 140nsec }

    port[Tim_Control]:=$34;  { modalità RATE GENERATOR }
    port[Tim_Count_0]:=223;  {  140nsec. * 223=31220nsec. f=32030 }
    port[Tim_Count_0]:=0;
    writeln;
    writeln('Timer_0 abilitato a generare interrupt a 32KHz');
    writeln;

    port[Tim_Control]:=$76;  { modalita' "SQUARE WAVE" }
    port[Tim_Count_1]:=$CA;  { periodo di 140nsec. * 714=0.1msec. }
    port[Tim_Count_1]:=$02;
    WRITELN('Timer_1 genera onda quadra a 10Khz');
    writeln;

    port[Tim_Control]:=$B4;  { modalita' "RATE GENERATOR" }
    port[Tim_Count_2]:=$0a;  { 0.1msec. * 10=1msec. }
    port[Tim_Count_2]:=$00;
    writeln('Timer_2 manda interrupt al PC a 1KHz');
    writeln;

end.
```