2

Il processore SHARC ADSP-21262

Il processore ADSP-21262, utilizzato nel presente lavoro, appartiene ad una vasta famiglia di Digital Signal Processors della Analog Devices, denominata SHARC®, particolarmente indicata per il controllo di sistemi automatici, elaborazione di segnali audio e immagini. Attualmente la Analog Devices, azienda leader nella produzione di processori DSP, propone sul mercato la sua 3ª generazione di processori SHARC, la quale ha come modello base il processore ADSP-21261, con frequenza di clock pari a 150MHz e capace di effettuare 900 MFLOPS (Mega Floating-Point Operations) e 300 MMACS (Mega Multiply, Accumulate and Sum), e come modello di punta il processore ADSP-21369, con frequenza di clock pari a 400MHz, 2400 MFLOPS e 800 MMACS.

Tale processore, pur avendo tali caratteristiche, ha un costo molto contenuto (l'ADSP-21262 ha un prezzo sul mercato USA di 10$), e viene fornito dalla Analog Devices in package LQPF e MiniBGA, in modo tale da poter essere direttamente inserito su schede già programmate, oppure bundled in un sistema di sviluppo chiamato EZKIT, il quale verrà descritto nei paragrafi successivi.

Inoltre, la Analog Devices fornisce un sistema di sviluppo per Microsoft Windows®, VisualDSP++, il quale permette di scrivere e testare i programmi da eseguire sul DSP, sia in modalità "simulazione", ovvero simulando via software il comportamento del processore SHARC, sia sul processore DSP bundled sulla board EZKIT. Con il VisualDSP++ viene fornito anche un nucleo real-time per la gestione

ottimizzata dei thread e dell'I/O, chiamato VDK (VisualDSP Kernel), descritto nei capitoli seguenti.

In questo capitolo, verrà dato ampio spazio all'architettura del DSP e del sistema EZKIT, descrivendo nel dettaglio tutte le caratteristiche hardware utilizzate per realizzare l'algoritmo di misurazione adottato nel presente lavoro.

2.1 Il core

Lo schema di principio del processore SHARC è riportato in figura 2.1.

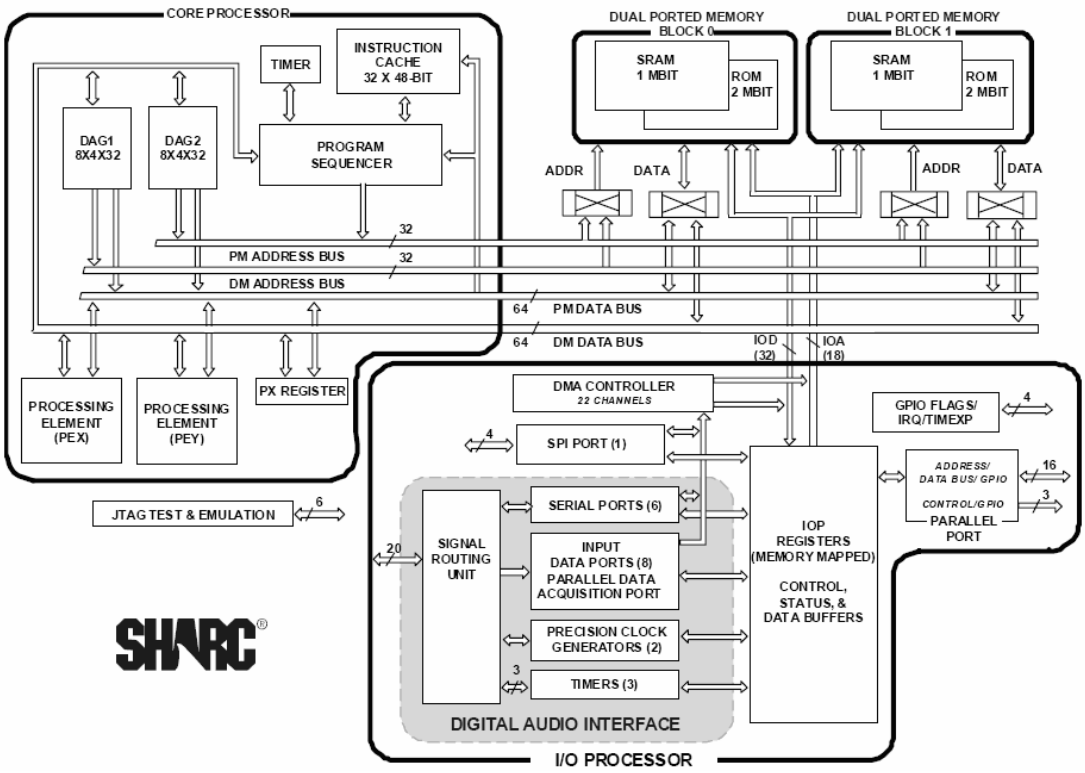

Fig. 2.1 – Core

E’ possibile notare che il core è diviso in tre blocchi, ognuno dedicato alla gestione di un aspetto del DSP. In particolare, sono presenti:

- due unità dedicate alla sezione di calcolo, chiamate PEx e PEy, ognuna contenente un moltiplicatore, una ALU, uno shift register e una pila di registri accumulatori, denominata “data register file”;
- un Program Sequencer, il cui compito è gestire gli accessi in memoria per le istruzioni;
- due blocchi SRAM da 1Mbit ognuno;
- due DAG (Data Address Generator), utilizzati per effettuare gli accessi in memoria;
- una sezione di I/O contenente porte seriali, parallele, SPI e un controllore DMA;
- tre bus per il collegamento dei blocchi, denominati PM (Program Memory), DM (Data Memory) e I/O.

Nei paragrafi successivi verranno descritti singolarmente tutti i singoli blocchi.

2.2 I blocchi di calcolo PEx e PEy

I blocchi di calcolo PEx e PEy rappresentano il cuore computazionale del DSP. La figura 2.2 mostra le loro parti fondamentali.

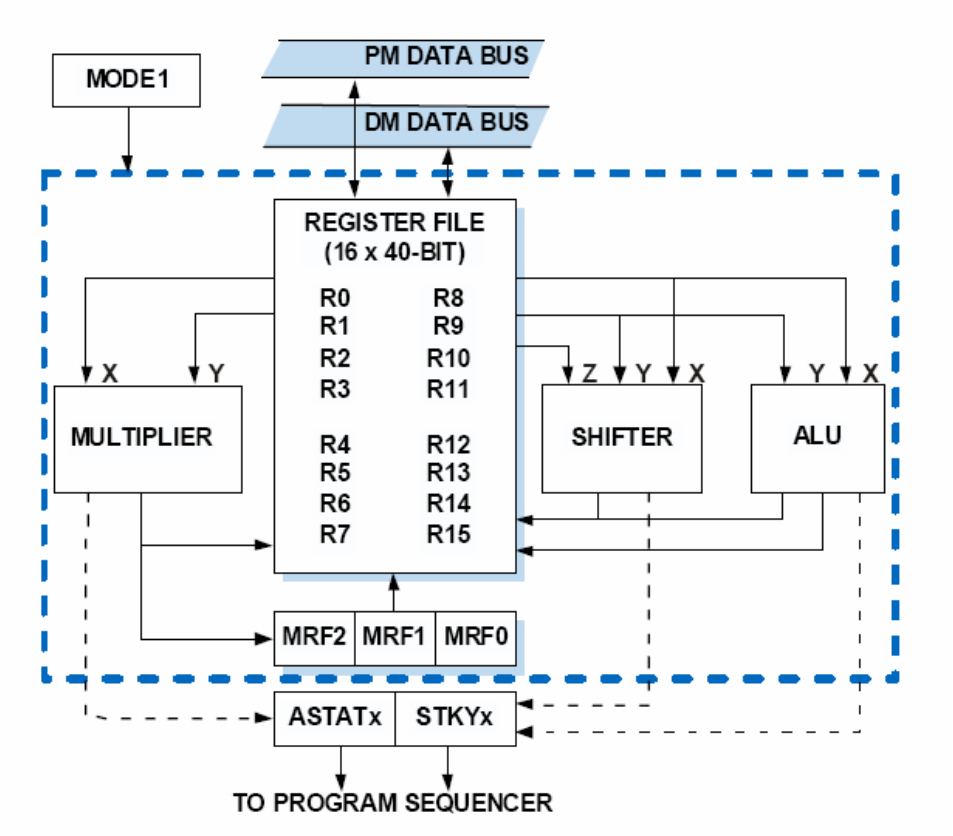

Fig. 2.2 – Il blocco PE

Dalla figura è possibile notare la pila di registri accumulatori R0-R15, il moltiplicatore, lo shift register, l'ALU, tre registri aggiuntivi, MRF2, MRF1 e MRF0, per conservare il risultato temporaneo di una moltiplicazione e due registri di stato, ASTATx e STKYx. Il blocco illustrato in figura rappresenta l'elemento PEx. L'elemento PEy è costruito allo stesso modo, con l'unica differenza che i registri accumulatori vengono denominati S0-S15, i tre registri aggiuntivi MSF2, MSF1 e MSF0, e i due registri di stato ASTATy e STKYy.

Procedendo dall'alto, è possibile notare la presenza di un ulteriore registro, chiamato MODE1. Questo registro contiene bit utili alla configurazione del

processore, e verrà descritto dettagliatamente in seguito. Per quanto riguarda il suo uso con gli elementi PE, esso permette di abilitare l'elemento PEy. Difatti, per default lo SHARC non usa mai la sezione PEy, e i calcoli vengono effettuati utilizzando unicamente PEx. In tal modo, ogni singola istruzione lavora solo sull'operando specificato, realizzando la classica modalità SISD (Single Instruction Single Data). Abilitando il bit PEYEN nel registro MODE1, viene attivato il modulo PEy, entrando così in modalità SIMD (Single Instruction Multiple Data). In tal modo, un'istruzione può lavorare su insiemi di registri contigui in memoria, dimezzando di fatto il tempo necessario all'elaborazione rispetto al tempo impegnato in modalità SISD.

Due sono i bus utili per portare gli operandi all'interno del data register file, PM e DM, e possono essere usati in un solo ciclo, realizzando una "dual data fetch", ovvero una "quad-data fetch" in modalità SIMD.

Ogni registro Rx è lungo 40-bit, e può accogliere numeri interi e a virgola mobile, regolata secondo il formato IEEE 754. Tale standard prevede 32-bit, di cui 24 per la mantissa e 8 per l'esponente. Lo SHARC prevede la possibilità di "allargare" la mantissa con altri 8 bit, aumentando in tal modo la precisione durante l'elaborazione, tenendo ben presente però che il risultato definitivo in memoria viene in ogni caso depositato secondo la norma IEEE 754. Un bit specifico nel registro MODE1 indica se gli 8 bit aggiuntivi devo essere trascurati durante la memorizzazione o possono essere usati per arrotondare il risultato. I registri MR invece sono lunghi 80-bit, e vengono utilizzati solo per le moltiplicazioni.

Il data register file è diviso in quattro sezioni, R0-R3, R4-R7, R8-R11, R12-R15. Questa divisione permette di utilizzare in parallelo il moltiplicatore e la ALU per realizzare una MACS (Multiply, Accumulate and Sum). Le sezioni R0-R3 e R4-R7 contengono i fattori di una moltiplicazione, R8-R11 e R12-R15 gli addendi per una somma. Se a tale caratteristica si aggiunge la "dual-data-fetch", è possibile capire la grande capacità computazionale dello SHARC, che in un solo ciclo permette due accessi in memoria, un prodotto e una somma. Inoltre, è possibile aggiungere anche una differenza in parallelo, purche tale sottrazione agisca sugli stessi addendi

impiegati nella somma. Le potenzialità offerte dal “dual data fetch” e dalla modalità SIMD sono state enormemente sfruttata nello sviluppo dell’algoritmo di misura, per minimizzare il tempo di calcolo.

2.2.1 La modalità SIMD

La modalità Single Instruction Multiple Data è una delle tante tecniche di ottimizzazione del calcolo prevista dalla teoria dei Calcolatori Elettronici. Essa è una delle quattro modalità generali conosciute:

- SISD (Single Instruction Single Data);
- SIMD
- MISD (Multiple Instruction Single Data);
- MIMD (Multiple Instruction Multiple Data);

La modalità MISD non ha implementazioni pratiche, poiché non fornisce vantaggi computazionali, ed esiste solo per completezza teorica. Le altre modalità vengono implementate in diversi sistemi di calcolo, in particolare la MIMD è sfruttata per elaborazioni grafiche di alte prestazioni.

Ogni processore implementa le modalità di accesso ai dati e alle istruzioni nel modo che più si adatta all’architettura scelta. Nel caso dei processori SHARC, per ogni istruzione vengono prelevati (o memorizzati) due dati contigui in memoria, dei quali uno viene inviato al PEx e l’altro al PEy, e viene eseguita la stessa operazione su entrambi gli elementi PE. E’ immediato verificare che tale modalità è particolarmente indicata per l’elaborazione di dati contenuti in vettori.

Particolare attenzione va quindi prestata alla progettazione dell’algoritmo, in quanto non tutti possono essere implementati in maniera agevole in modalità SIMD; questo è vero in particolar modo quella classe di problemi che prevedono esecuzioni di istruzioni diverse a seconda dei valori assunti dai dati d’ingresso (come sarà nel caso in esame per la sezione di unwrap).

La tecnologia SIMD implementata sul processore SHARC ha anche due limitazioni:

- gli indirizzi di memoria dei vettori devono essere numeri pari, altrimenti non viene realizzata una corretta lettura degli operandi;
- l'attivazione e la disattivazione di tale modalità prevede la perdita di quattro cicli (due per la commutazione del bit PEYEN, più due di attesa), rendendola impraticabile per vettori di lunghezza inferiori a 6

2.3 Il Program Sequencer

Il Program Sequencer rappresenta la sezione dedicata alla gestione degli accessi in memoria per il caricamento delle istruzioni. Il suo compito primario è quindi quello di gestire il flusso del codice, e le strutture classiche utilizzate nella programmazione, come cicli, salti, interruzioni e chiamate a subroutine. In figura 2.3 è riportato il suo schema. Il Program Sequencer ha una struttura notevolmente complessa, dovuta principalmente al fatto che particolare cura è stata dedicata nel rendere minimo l'impatto negativo dei salti sulla pipeline; il suo obiettivo è comunque estremamente semplice, ovvero quello di impostare nel registro Program Counter, lungo 24 bit, l'indirizzo della prossima istruzione da eseguire.

Lo SHARC è dotato di una pipeline lunga tre stadi, la quale viene riempita con le fasi di *fetch* (caricamento dell'istruzione), *decode* (decodifica dell'istruzione) ed *execute* (esecuzione). Quindi viene eseguita un'istruzione per ogni ciclo, fintantochè la pipeline non viene distrutta. La distruzione può essere causata da un salto (condizionato o incondizionato), il quale azzera le sezioni di *decode* e *execute*, in quanto le due istruzioni caricate nei due cicli successivi al salto non devono più essere eseguite. Per riempire nuovamente gli stadi della pipeline, devono essere eseguite due istruzioni di attesa (NOP), quindi è immediato capire che i salti possono degradare notevolmente le prestazioni del processore.

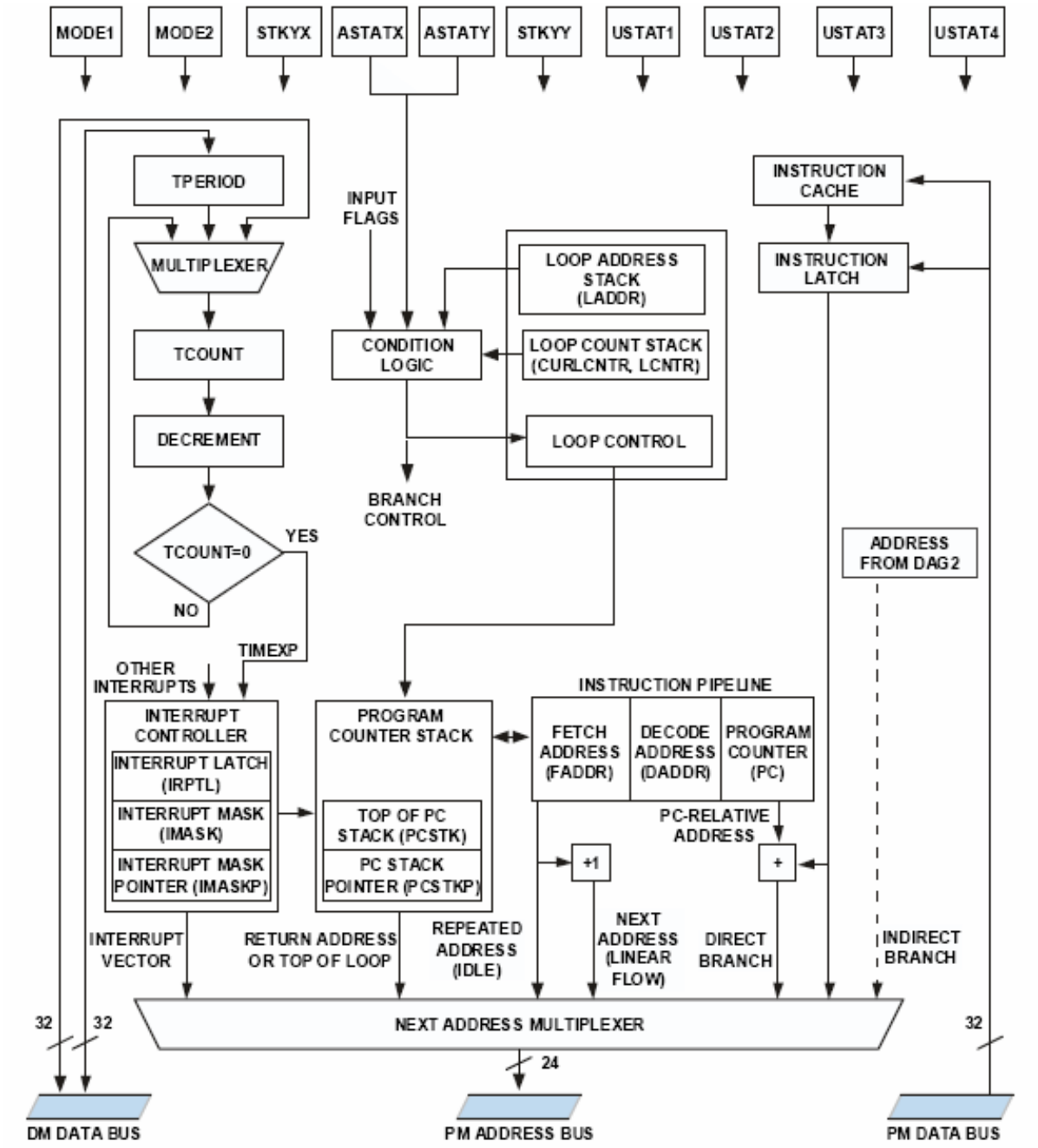

Fig. 2.3 – Program Sequencer

Per quanto riguarda i loop, la gestione della pipeline è automatica, e il test della condizione di uscita (o di decremento del contatore) viene effettuato due cicli prima della fine del loop, in modo tale da non svuotare mai la pipeline. Invece nulla può essere fatto nel caso dei salti o di chiamata a subroutine, in quanto non è possibile prevedere e/o alterare in automatico la sequenza delle istruzioni per evitare lo svuotamento. Tuttavia, lo SHARC permette di realizzare il cosidetto *Delayed Branch*. L'idea di fondo è molto semplice: se, prima di effettuare un salto, devono essere effettuate due operazioni che non alterano il comportamento del salto (tipo accessi in memoria), è possibile indicare al Program Sequencer di effettuare un salto ritardato (con un preciso codice operativo) e posticipare all'istruzione di salto le due istruzioni da effettuare in ogni caso. Così facendo, la pipeline è sempre piena e non vengono persi cicli per eseguire istruzioni NOP.

Una ulteriore struttura utilizzata nel presente metodo è un timer integrato, il quale può essere programmato secondo un multiplo del tempo di ciclo, generando una interruzione ad intervalli di tempo regolari.

Come in qualsiasi altra architettura, è prevista una sezione dedicata alla gestione delle interruzioni, con la possibilità di mascherare le singole richieste, di disabilitare/abilitare l'intera sezione, e di prevedere le interruzioni innestate con priorità, ovvero l'esecuzione di interruzioni all'interno di ISR (Interrupt Service Routine) aventi priorità più alta di quella che viene servita all'interno della routine dedicata alla sua gestione. Una utile istruzione, particolarmente usata in questo lavoro, è l'istruzione IDLE, la quale consente di portare il processore in modalità "basso consumo" di potenza qualora l'esecuzione di una particolare parte di codice sia condizionata al presentarsi di un evento sul segnale di interrupt.

2.4 Data Address Generator

I DAG contenuti nel processore SHARC sono due dispositivi che si occupano di gestire i registri dedicati agli accessi nei due blocchi di memoria SRAM. Sono presenti due DAG, uno per la memoria DM e uno per la memoria PM, ognuno contenente quattro pile di otto registri (figura 2.5):

- i registri I, contenenti l'indirizzo di memoria sul quale effettuare l'accesso;
- i registri M, contenenti il valore da sommare all'indirizzo per ottenere l'indirizzo fisico da inviare alla memoria;
- i registri B, che contengono l'indirizzo base sul quale impostare una modalità di accesso circolare ai dati;
- i registri L, che contengono la lunghezza del buffer circolare su cui si sta effettuando l'accesso.

La modalità di accesso per la gestione dei buffer circolari viene attivata se e solo se il registro L interessato all'accesso ha un valore diverso di zero, e se è alzato il bit CBUFEN nel registro MODE1. In tal caso, ad ogni accesso il registro I viene modificato con il valore del registro M, e confrontato con l'indirizzo ottenuto sommando i registri B e L. Se tale indirizzo supera quest'ultimo valore, in I viene riportato il valore del registro B, in modo tale da ripetere dal primo indirizzo l'accesso in memoria.

Tutto questo viene realizzato in modalità trasparente dal DAG, il quale prevede anche due modalità di modifica dell'indirizzo durante un accesso:

- con pre-modifica senza alterazione, ovvero viene sommato il valore di M ad I, viene effettuato l'accesso con l'indirizzo ottenuto e il valore di I viene lasciato inalterato;
- con post-modifica e alterazione, ovvero viene effettuato un accesso con il valore di I, e subito dopo aver trasferito il dato viene modificato il valore di I sommandogli il valore di M (figura 2.4).

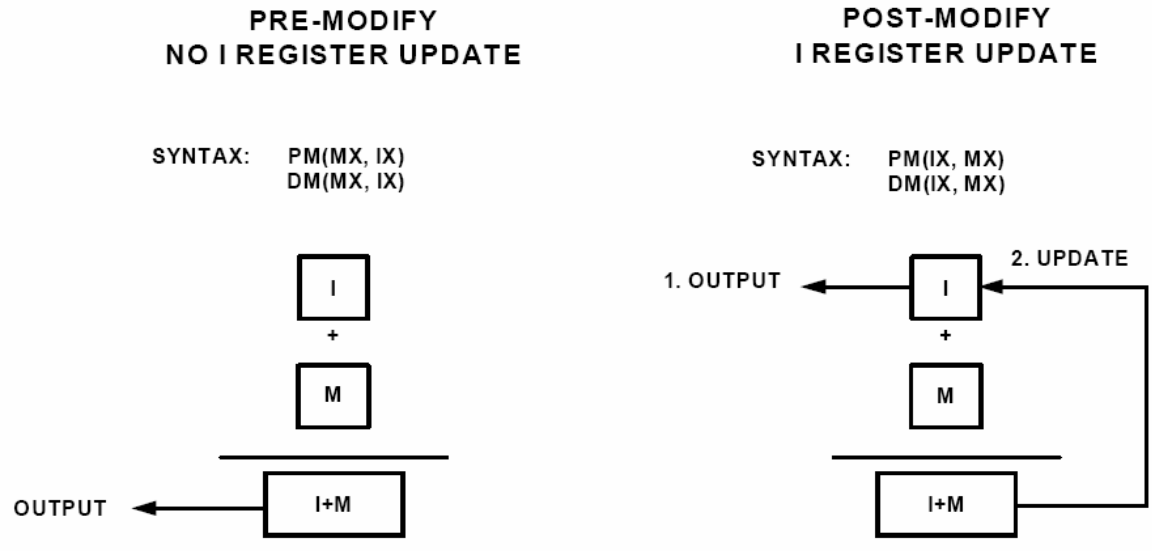

Fig. 2. 4 – Pre-modifica e post-modifica

I due generatori prevedono altre caratteristiche, come il broadcasting degli indirizzi nei registri I, e la possibilità di utilizzare gli indirizzi rovesciando i bit contenuti sempre in tali registri. Tuttavia, queste caratteristiche non sono state utilizzate nel presente metodo.

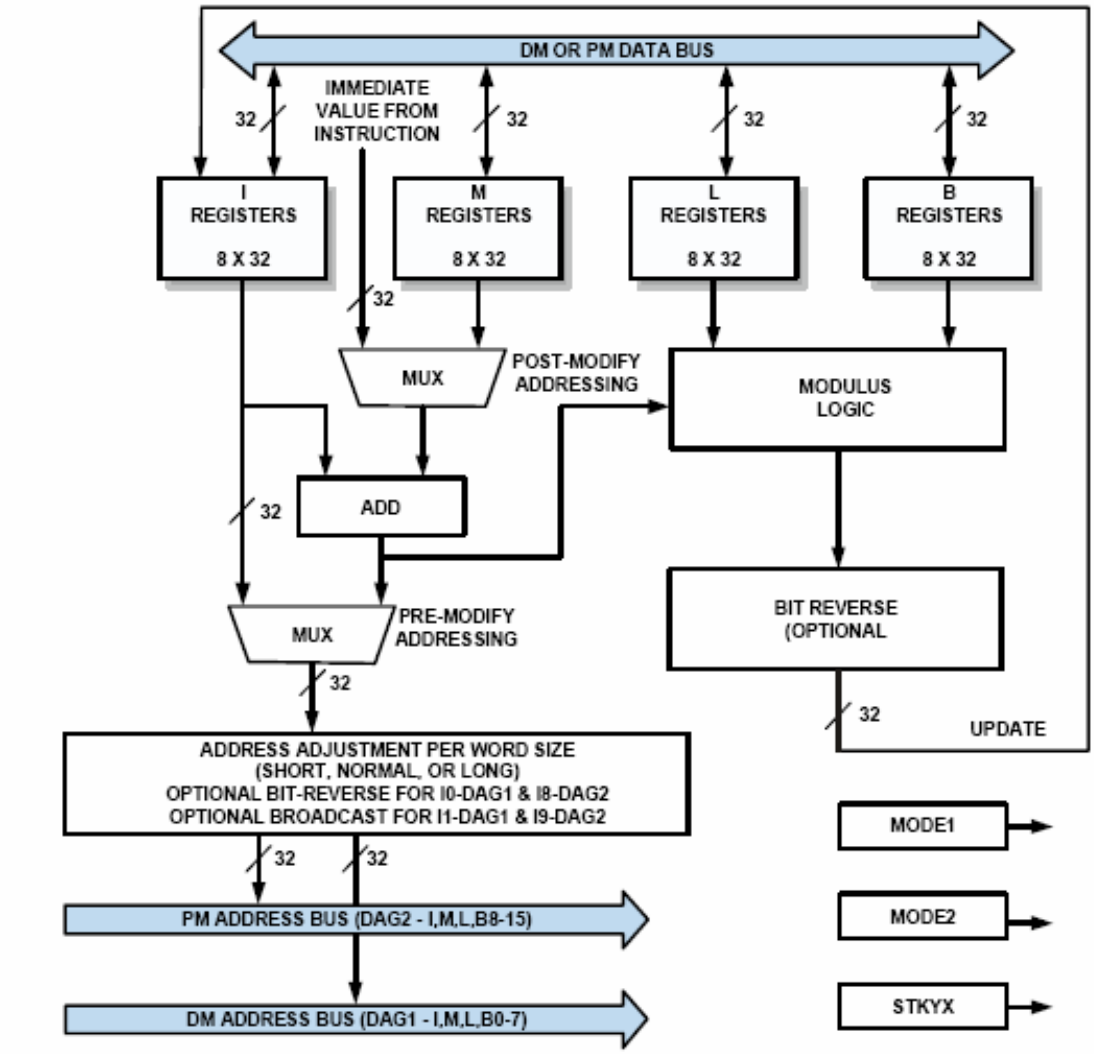

Fig. 2.5 – Data Address Generator

2.5 La memoria

La figura 2.6 mostra la struttura della memoria presente sia sul DSP, sia sulla board di sviluppo EZKIT. Sul processore SHARC, oltre ai 2Mbit di memoria SRAM già indicati in precedenza, vi sono 4Mbit di Flash ROM programmabile, che permettono di operare il bootstrap del codice all'accensione del DSP, e 512Kbyte di memoria SRAM esterna, alla quale si può accedere tramite la sezione I/O, utilizzando la porta parallela o il DMA.

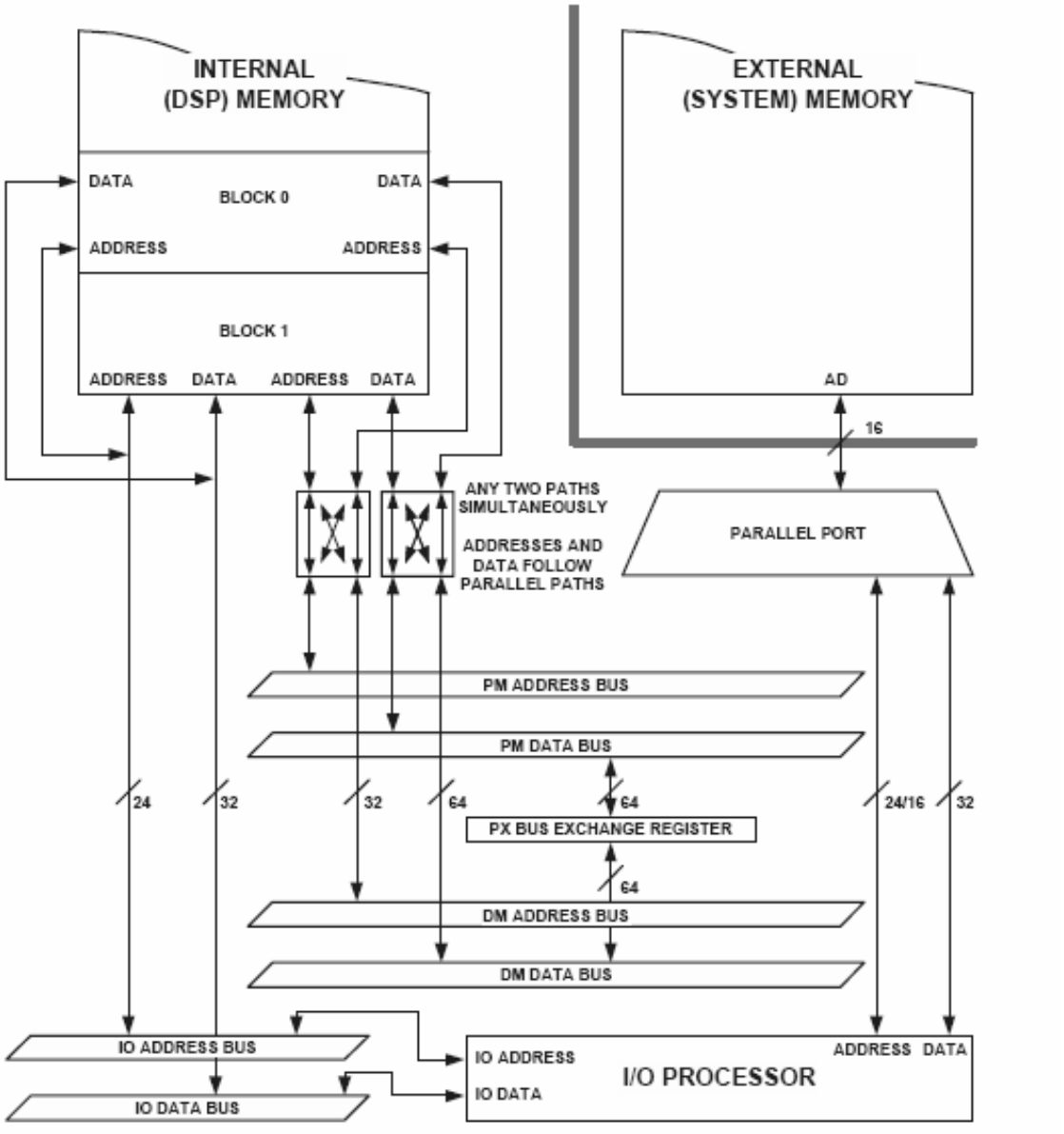

Fig. 2.6 – Struttura della memoria interna e esterna

2.6 La board di sviluppo EZKIT LITE

La Analog Devices fornisce il processore SHARC in opportuni packages per essere direttamente saldato su board apposite già programmate, oppure bundled su una board proprietaria, chiamata EZKIT. Tale board possiede diversi ingressi ed uscite, sia analogiche che digitali, in modo tale da poter interfacciare il DSP con qualsiasi tipo di sorgente d'informazione.

Il collegamento tra la EZKIT e il Personal Computer avviene tramite porta USB, o tramite un hardware dedicato chiamato JTAG. Utilizzando l'ambiente VisualDSP++, è possibile monitorare ogni aspetto del processore, dal contenuto della memoria, sia interna che esterna, a quello dei registri del DSP e della sezione di I/O. I connettori disponibili sulla board sono:

- quattro uscite analogiche RCA audio stereo;
- una uscita analogica audio stereo su jack cuffia da 3.5 mm;
- un ingresso analogico RCA audio stereo;
- un ingresso digitale SPDIF CS8416;
- 20 pin DAI (Digital Analog Interface);

La EZKIT prevede, alla base, un collegamento con un'altra board della Analog Devices, denominata "Extender", la quale, oltre a fornire una porta seriale con connettore RJ-45, monta un'interfaccia dedicata per il collegamento con un convertitore A/D 9244 della Analog Devices, con una risoluzione di 14 bit e frequenza di conversione massima pari a 65MS/s.

Per agevolare la programmazione e poter interagire con l'utente, sono disponibili 8 LED e 4 pulsanti, tutti programmabili. In figura 2.7 è riportata una foto del sistema EZKIT.

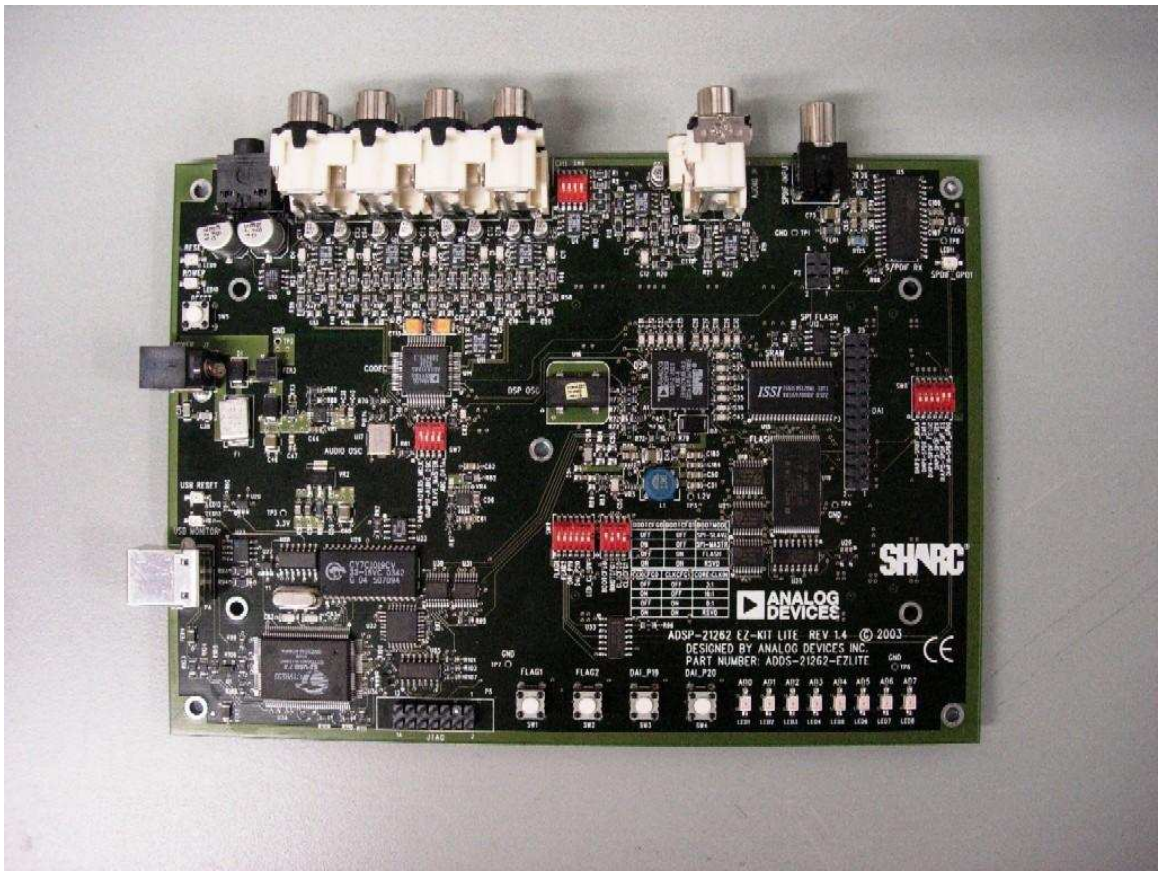

Fig. 2.7 - La board di sviluppo EZKIT LITE

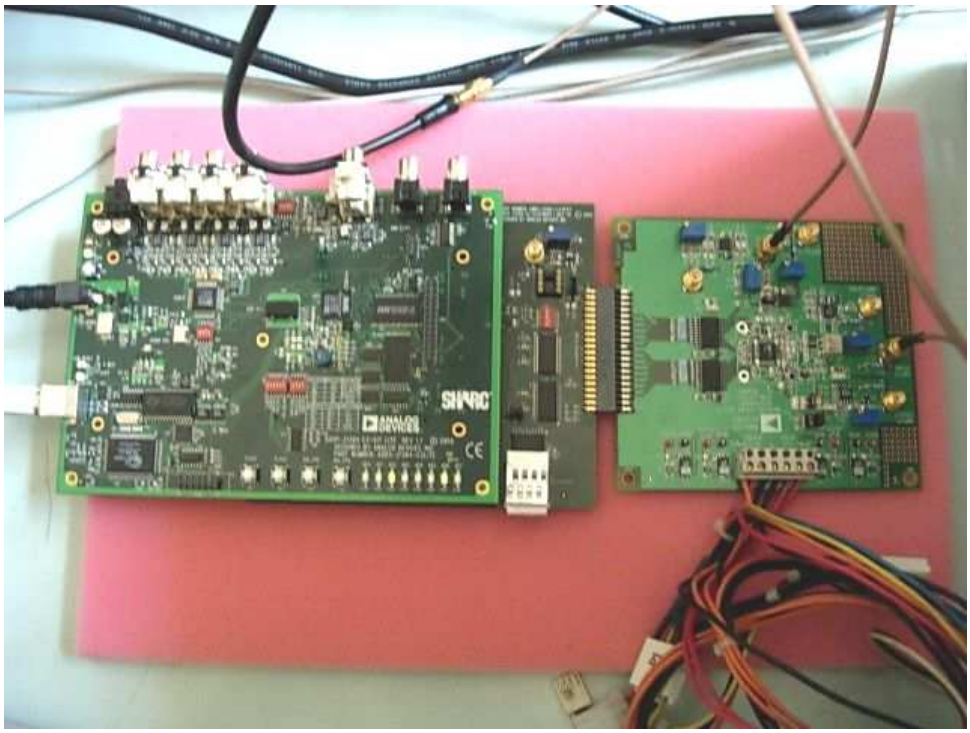

Fig. 2.8 – EZKIT, EZ-Extender e AD9244